Konzept einer bibliotheksbasiert konfigurierbaren Hardware-Testeinrichtung für eingebettete elektronische Systeme

Zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEURS

von der Fakultät Elektrotechnik und Informationstechnik
der Universität Fridericiana Karlsruhe
genehmigte

DISSERTATION

von Dipl.-Ing. Carsten Bieser
aus Alzey

Tag der mündlichen Prüfung	:	05. Juli 2007
Hauptreferent	:	Prof. Dr.-Ing. Klaus D. Müller-Glaser
Korreferent	:	Prof. Dr.-Ing. Sorin A. Huss

ISBN 978-3-937295-78-7

E&B Engelhardt und Bauer – Druck und Verlags GmbH
Käppelestraße 10 ▪ 76131 Karlsruhe ▪ Germany
Telefon: 0721 / 9 62 26-100 ▪ Fax: 0721 / 9 62 26-101
Internet: www.ebdruck.de ▪ eMail: center@ebdruck.de

Meinen Großeltern

Kurzfassung

Die vorliegende Dissertation befasst sich mit dem schnellen Erzeugen von Prototypen (Rapid Prototyping, RP) und dem Test fertiger Steuergeräte (Hardware-in-the-Loop, HiL) unter Verwendung programmierbarer Logikbausteine (FPGAs). Letztere erlauben eine flexible und zeitsparende Implementierung von Funktionen in Hardware – auch zur Laufzeit des Systems. Außerdem wird somit ein hoher Wiederverwendungsgrad, sowohl der Hardware-Komponenten, als auch der darin integrierbaren Funktionen (Intellectual Property, IP) erreicht.

Wesentlicher Bestandteil der Arbeit ist die Entwicklung einer mobilen Hardware-Plattform, die sowohl für Rapid-Prototyping als auch für Hardware-in-the-Loop eingesetzt werden kann. Im Gegensatz zu Standardprodukten werden hier anstelle von dedizierten I/O-Schnittstellen ausschließlich programmierbare Interface-Karten auf Basis von FPGAs verwendet, die eine nahezu uneingeschränkte Programmierbarkeit aufweisen und damit helfen, die Kartenvielfalt stark einzuschränken. Weiterhin wird die konservative „Ein-Bus-Architektur“ aufgeweicht und hinsichtlich mehrerer Bussysteme für verschiedene Aufgaben und Zwecke erweitert. Durch dieses Konzept werden neben der Unterstützung von konzept-, architektur- und realisierungs-orientiertem RP auch Berechnungen von mehreren HiL-Testmodellen parallel auf einer Plattform durch Verteilung auf mehrere Interface-Karten möglich. Dies macht die Plattform für den Einsatz im ganzen Entwicklungsprozess innerhalb des V-Modells interessant.

Der zweite, wichtige Aspekt der Arbeit betrifft die Erstellung der Konfiguration für die auf den Interface-Karten implementierten FPGAs. Dazu wird eine neue slot-basierte Methodik vorgestellt, die es erlaubt, eine Belegung mit verschiedenen Schnittstellenmodulen für einen FPGA mit Hilfe einer grafischen Benutzerschnittstelle (Graphical User Interface, GUI) zu arrangieren, ohne dafür jegliche Kenntnisse einer Hardware-Beschreibungssprache (VHDL oder Verilog) oder spezifisches Wissen hinsichtlich der FPGA-Interna zu besitzen. Diese Methodik gewährleistet zum einen die Performanz der einzelnen, getesteten und in einer Bibliothek abgelegten Module – auch in der Gesamtkonfiguration. Zum anderen stellt sie die Verträglichkeit der Module untereinander sowie die Korrektheit des Gesamtarrangements (Bitstream) sicher.

Das gesamte System ist von einem umfassenden Sicherheitskonzept umgeben, das den Benutzer vor Fehlkonfigurationen und damit auch die Plattform vor Zerstörungen bewahrt. Das Konzept ist dazu auf verschiedene Ebenen von Hard- und Software umgesetzt, um eine optimale Abdeckung möglicher Fehlerquellen zu erreichen.

Durch diverse Experimente anhand relevanter und dem industriellen Umfeld angepasster Beispielimplementierungen aus dem Automobilbereich wird die Einsetzbarkeit und Effektivität des RP-HiL-Systems sowie der FPGA-Design-Methodik nachgewiesen.

Danksagung

Die vorliegende Arbeit ist während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Technik der Informationsverarbeitung (ITIV) der Universität Karlsruhe (TH) entstanden. An dieser Stelle möchte ich mich bei all denjenigen bedanken, die zum Gelingen derselben beigetragen haben.

Besonders bedanke ich mich bei den Leitern des Instituts, Herrn Prof. Dr.-Ing. K.-D. Müller-Glaser und Herrn Prof. Dr.-Ing. Jürgen Becker für die Möglichkeit, an ihrem Institut zu arbeiten und währenddessen meine wissenschaftliche Arbeit vorantreiben zu können. Herrn Prof. Dr.-Ing. Klaus D. Müller-Glaser gilt mein besonderer Dank für die – nicht selbstverständliche – Zuteilung des nötigen Freiraums zur Verfolgung meines eigenen wissenschaftlichen Forschungsthemas. Herrn Prof. Dr.-Ing. Sorin A. Huss von der TU Darmstadt danke ich für die freundliche Übernahme des Zweitgutachtens.

Weiterhin richtet sich mein Dank an die Mitarbeiter des ITIV für deren bereitwillige und kollegiale Unterstützung meiner Aktivitäten in allen wissenschaftlichen, technischen und organisatorischen Belangen. Namentlich zu erwähnen sind hierbei insbesondere in alphabetischer Reihenfolge Herr Dipl.-Ing. J. Becker, Herr Dipl.-Ing. R. König, Herr Dipl.-Ing. T. Köster und Herr Dipl.-Ing. A. Thomas, die stets eine fundierte Plattform für fachliche wie private Diskussionen boten und mit denen ich auch über die universitären Grenzen hinweg ein überaus freundschaftliches Verhältnis pflegte.

Mein Dank gilt ebenfalls den weit mehr als 35 Studenten, die im Laufe meiner Tätigkeit im Rahmen von Diplom-, Studien- und Seminararbeiten mit mir zusammengearbeitet haben, mich durch ihre Begeisterungsfähigkeit und ihr unermüdliches Engagement unterstützt, sowie durch ihre Ergebnisse wesentlich zum Gelingen dieser Arbeit beigetragen haben. Hierbei sind Herr Dipl.-Ing. M. Bahlinger, Herr Dipl.-Ing. M. Heinz, Herr Dipl.-Ing. M. Schwarz und Herr Dipl.-Ing. C. Stops besonders hervorzuheben.

Nicht zuletzt möchte ich meiner Frau Sandra für ihre stete Unterstützung und ihre Nachsicht und Toleranz während der Entstehungszeit dieser Arbeit und insbesondere meinen Eltern, Renate und Wilhelm, die es mir letztlich ermöglicht haben, nach meiner Berufsausbildung ein Hochschulstudium zu absolvieren und so meinen Berufswunsch vollends umzusetzen, meinen herzlichen Dank aussprechen. Einschließen darf ich an dieser Stelle auch meine Schwiegereltern Waltraud und Richard Daniel für ihren Beitrag zu meinem Fortkommen.

Abschließend danke ich meinen Studienkollegen Dipl.-Ing. H.-P. Weyand und Dipl.-Ing. M. Zöller, die mich während des Studiums begleitet haben und mit denen mich nach wie vor eine überaus gute Freundschaft verbindet, sowie meinem Berufschullehrer Herrn Dipl.-Ing. Gramberg, der mich in meinem Bestreben, die universitäre Ingenieurslaufbahn einzuschlagen, stets bestärkt hat und letztlich auch mit den Ausschlag für die Wahl des Studienortes gab.

Karlsruhe, Juli 2007 Carsten Bieser

Inhaltsverzeichnis

1 Einleitung 1

1.1 Motivation – Entwicklung und Test von Steuergeräten 2
1.2 Ziele der Arbeit – Der eigene Beitrag 4
1.3 Gliederung der Arbeit 5

2 Elektronik im Kraftfahrzeug 7

2.1 Bussysteme im Automobil 8
2.1.1 Unterscheidung von Bustypen 8
2.1.2 Controller Area Network (CAN-Bus) 11
2.1.3 Local Interconnect Network (LIN)-Bus 13
2.1.4 Media Oriented System Transport (MOST)-Bus 14
2.1.5 FlexRay-Bus 15
2.1.6 Time Triggered Protocol (TTP/C) 16
2.2 Eingebettete Systeme 17
2.2.1 Einsatzgebiete 17
2.2.2 Vernetzung 18
2.3 Anforderungen an Eingebettete Systeme im Kfz 18
2.3.1 Umgebungseinflüsse 18
2.3.2 Zeitbedingungen 19
2.4 Aktuelle und zukünftige Trends im Automobil 19
2.4.1 Reduktion von Steuergeräten – Verteilte Funktionen 19
2.4.2 Fahrerassistenzsysteme 19

3 Stand der Technik 21

3.1 Programmierbare Bausteine 21
3.1.1 Field Programmable Gate Arrays (FPGAs) 22
3.1.2 Virtex-II-FPGAs von Xilinx 23
3.1.3 Bitstream und Konfigurationslogik 24
3.1.4 Konfiguration von Xilinx-FPGAs 28
3.1.5 FPAA (Free Programmable Analog Array) 33
3.2 Sensoren und Aktoren 35

3.2.1 Klassifizierung von Sensorschnittstellen 35
3.2.2 Klassifizierung von Aktorschnittstellen 36
3.3 Digitale Schnittstellen 37
3.3.1 Parallele I/O-Ports von Mikrocontrollern und FPGAs 37
3.3.2 Serielle Schnittstellen und Protokolle 37
3.4 Signalkonditionierung 38
3.5 Takterzeugung und Verteilung 39
3.5.1 Takterzeugung 39
3.5.2 Taktverteilung 39
3.5.3 LVDS-Übertragung 39
3.6 Funktionsprinzipien von Spannungswandlern 40
3.6.1 Linearregler 40
3.6.2 Schaltregler 41
3.7 Vierquadrantenverstärker 42
3.8 Code-Erzeugung für programmierbare Schaltungen 42
3.8.1 VHDL und Verilog 42
3.8.2 IP-Cores 43
3.8.3 Programmiersprache Java und der Editor Eclipse 44
3.8.4 Kombinationswerkzeuge (Xilinx ISE/EDK/PlanAhead) 45
3.8.5 Klassenbibliothek Jbits von Xilinx 50
3.9 Vorgehensmodelle und Werkzeugeinsatz 50
3.9.1 Lebenszyklusmodelle zur Systementwicklung 50
3.9.2 Graphische Modellierung mit Case-Tools 53
3.10 Rapid-Prototyping 56
3.10.1 Arten des Rapid Prototyping 56
3.10.2 Phasen des Rapid Prototyping 57
3.10.3 Schnittstellen für Rapid Prototyping 59
3.10.4 Stand der Technik und der Forschung 60
3.10.5 Bewertung Rapid-Prototyping-Systeme 63
3.11 Hardware-in-the-Loop 64
3.11.1 Schnittstellen für Hardware-in-the-Loop 65
3.11.2 Stand der Technik und der Forschung 65
3.11.3 Bewertung der Hardware-in-the-Loop-Systeme 69
3.12 Vergleich von Rapid Prototyping und Hardware-in-the-Loop 70
3.12.1 Bewertung 70
3.13 Anforderungen 71

4 Die RP-HiL-Plattform 75

4.1 Systemkonzept des Basissystems 75
4.1.1 Trennung von Logik und Prozesskopplung bei I/O-Schnittstellen 75
4.1.2 Bedarfsgerechte Konfiguration des Systems 76
4.1.3 Einsatz von FPGAs 77
4.1.4 Ressourcenbelegung auf dem FPGA 82
4.1.5 Modularer Aufbau und Aufteilung der Systemfunktionalitäten 84

4.1.6 Identifikation der Steckkarten 89
4.1.7 Konfiguration der FPGAs 90
4.1.8 Zentrale Takterzeugung und -verteilung 90
4.1.9 Plattformkontrollinstanz 91
4.1.10 Bussystemarchitektur 93
4.1.11 Speicherung von Konfigurations- oder Messdaten 97
4.1.12 Temperaturbereich 98
4.1.13 Energieversorgung des Systems 98
4.1.14 Sicherheitskonzept 99
4.2 Konzept der Stromversorgung (Power Management Unit) 100
4.2.1 Abschätzung der Leistungsaufnahme der Plattform 100
4.2.2 Anforderungen zur Versorgung der FPGAs 103
4.2.3 Anforderungen an das Einschalten der FPGA-Spannungen 103
4.2.4 Blockschaltbild der PMU 105
4.2.5 Spannungs-Monitoring, An- und Abschaltezyklen 105
4.3 Ergebnis des Konzepts 107
4.4 Realisierung der Plattform 108
4.4.1 Überblick über die Basisplatine und die Stromversorgung 108
4.4.2 Hauptprozessoreinheit 108
4.4.3 CAN-Bus 111
4.4.4 Ethernet-Anschluss 112
4.4.5 Serielle Schnittstellen 112
4.4.6 USB-Schnittstelle 113
4.4.7 Interface-Karte 113
4.4.8 I/O-Karte 113
4.4.9 Automotive-Karte 114
4.5 Realisierung der PMU 115
4.5.1 Überblick über die Stromversorgung (PMU) 115
4.5.2 Ausgangsleistung und Wirkungsgrad der PMU-Schaltregler 115

5 Slot-basierte FPGA-Konfiguration 117

5.1 Einsatz von FPGAs in RP- oder HiL-Geräten 117
5.1.1 Problematik bei inkrementeller Änderung 117
5.1.2 Anforderungen 118
5.2 Grundidee 119
5.2.1 Aufteilung der FPGA-Logik in Bereiche 120
5.3 Konzept des „Bitstream-Merging“ 123
5.4 Hardware-Framework 124
5.4.1 Integration verschiedener Module 125
5.4.2 Kommunikation der Module mit der Umgebung 125
5.4.3 Systembus 128
5.4.4 I/O-Schnittstellen und Peripheral Communication Bus 129
5.4.5 Verteilung des Taktsignals auf dem FPGA 129
5.5 Modulerstellung 129
5.5.1 Design-Ablauf 130

5.5.2 Netzliste des Toplevel-Designs 131
5.5.3 Netzlisten der funktionalen Module 131
5.5.4 Implementierung mit PlanAhead 131
5.5.5 Anlegen eines Projekts 131
5.5.6 Exportieren des Floorplans 133
5.5.7 ISE Place-&-Route und Bitgen 133
5.5.8 Erstellen aller Bitstreams 133
5.5.9 Abweichungen vom Standard-Design-Ablauf 134
5.5.10 Vergleich der Bitstreams 135
5.5.11 Aktivierung des Taktsignals auf dem FPGA 136
5.5.12 XML-Konfigurationsdateien der funktionalen Module 136
5.6 Modulbibliothek 137
5.6.1 Ansteuerung D/A-Wandler (Modul dac) 138
5.6.2 CAN-Controller (Modul can) 139
5.6.3 LIN-Controller (Modul lin) 140
5.7 Abschließende Bewertung der Designmethode 143
5.8 Sicherheitskonzept 144

6 Software-Umgebung 147

6.1 Software für RP-HiL-Plattform 147
6.1.1 Betriebssystem Embedded Linux 147
6.1.2 Plattform-Konfigurations-Tool 150
6.1.3 Linux-Gerätetreiber 150
6.1.4 CompassLink-Applikation 151
6.1.5 Starten und Herunterfahren des Netzwerkservers 155
6.2 BistreamComposer - Modulaustausch per „Drop-Down-Menü“ 155
6.2.1 Funktion des Programms 157
6.2.2 XML-Konfigurationsdatei des Gesamtsystems 159
6.2.3 Verzeichnisstruktur der Applikation 162
6.2.4 Ergebnisse der Methodik anhand einer Beispielimplementierung 162
6.3 CompassControl - Bedienung und Steuerung des Systems 163
6.3.1 Grafische Benutzeroberfläche (GUI) 163
6.3.2 Bedienung des Programms 164
6.3.3 Klassendiagramm CompassControl 165
6.3.4 Kommunikation zwischen GUI und Plattform 167
6.3.5 Bitstream-Beschreibung im XML-Format 168
6.3.6 Kompatibilitätsprüfung zwischen Bitstream und Plattform 170
6.3.7 Dokumentation der Schnittstellenkonfiguration 173
6.3.8 Remote-RP und Remote-HiL 173
6.4 Modellbasierte Applikationserstellung 174
6.4.1 Modellierung kontinuierlicher Systeme 175
6.4.2 Modellausführung auf einem MicroBlaze 175
6.4.3 Integration als reine Hardware-Lösung 181
6.4.4 Modellierung zeitdiskreter Systeme 185
6.4.5 Modellimplementierung in Hardware 190

7 Anwendungsbeispiele und Ergebnisse 191

7.1 RP-Anwendungen 191
7.1.1 CAN2CAN-Gateway 191
7.1.2 NoC-Testszenario 194
7.2 HiL-Anwendungen 196
7.2.1 LIN-Bus-Tester 196
7.2.2 Doppel-MicroBlaze-Subsystem 202
7.2.3 PWM-Analysator für Automotive-Steuergeräte 206
7.2.4 Remote-HiL-Demonstrationsszenario 209

8 Zusammenfassung, Ergebnisse und Ausblick 213

8.1 Zusammenfassung und Ergebnisse 213
8.2 Ausblick 215

9 Verzeichnisse 219

9.1 Abbildungsverzeichnis 219
9.2 Tabellenverzeichnis 223
9.3 Formelverzeichnis 224
9.4 Abkürzungsverzeichnis 225
9.4.1 Allgemeine Abkürzungen 225
9.4.2 Eigene Abkürzungen 227
9.5 Formelzeichen 228
9.6 Literaturverzeichnis 229
9.7 Betreute Diplom-, Studien- und Seminararbeiten 249
9.7.1 Diplomarbeiten 249
9.7.2 Studienarbeiten 250
9.7.3 Seminararbeiten 251
9.8 Eigene Veröffentlichungen 252

1 Einleitung

Eingebettete Systeme haben mittlerweile in alle Lebensbereiche Einzug gehalten. Bekannte Felder aus dem täglichen Umgang sind dabei die Unterhaltungselektronik im Allgemeinen, die Medizintechnik, die Automobilbranche, aber auch die Luft- und Raumfahrt-, sowie die Automatisierungstechnik im industriellen Umfeld (Abbildung 1.1). In allen aufgeführten Bereichen ist die Komplexität der Systeme in den letzten Jahre enorm gestiegen, so dass die Entwicklungs- und Testverfahren, sowie die entsprechenden Entwicklungs- und Testsysteme angepasst bzw. verbessert werden müssen. Durch die wachsende Interaktion der bisher autonom agierenden eingebetteten Systeme ergibt sich eine weitere Dimension der Systemüberprüfung.

Abbildung 1.1: Einsatzbereiche von Eingebetteten Systemen

Wie nachhaltig sich die Verhältnisse teilweise darstellen können, lässt sich am anschaulichsten am Beispiel der Steuergeräteentwicklung im Automobilsektor demonstrieren. Dort ist

neben dem großen Bekanntheitsgrad auch die Innovationsdichte am höchsten. Daher bietet sich eine genaue Betrachtung des Automotive-Umfeldes an, das im Folgenden die Basis dieser Arbeit bildet.

1.1 Motivation – Entwicklung und Test von Steuergeräten

Im Bereich der Kraftfahrzeuge ist in den letzten eine enorme Komplexitätssteigerung der elektrischen und elektronischen Systeme zu verzeichnen. Dies gilt zum einen für die Steuergeräte selbst, als auch für deren Vernetzung untereinander. Damit diese komplexen Systeme über mehrere Jahre zuverlässig funktionieren können, sind neben einer guten Planung umfangreiche Tests notwendig. Dabei muss nicht nur die Korrektheit der Funktion überprüft werden, sondern insbesondere auch die Reaktion einer Baugruppe auf Fehlfunktionen der angeschlossenen Komponenten.

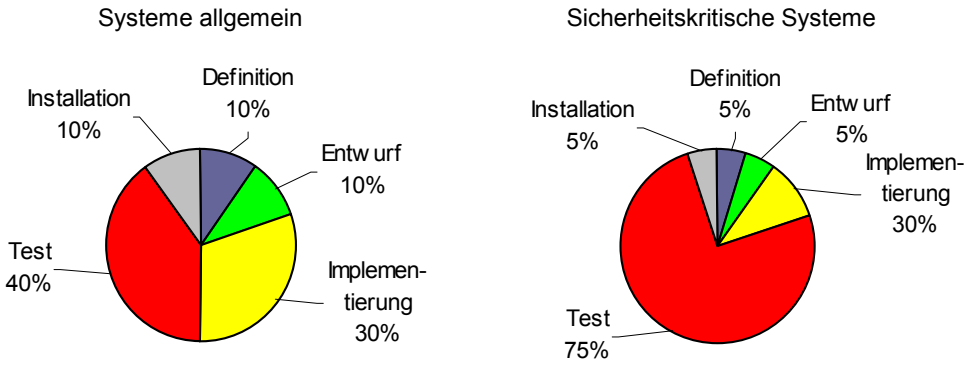

Abbildung 1.2: Aufwände (Kosten) bei der Systementwicklung [Fraun03]

In Abbildung 1.2 wird deutlich, wie groß der Testaufwand bei der Entwicklung von elektronischen Systemen ist. Handelt es sich dabei um sicherheitskritische Systeme, erhöht sich dieser Anteil im Entwicklungsprozess auf fast das Doppelte. Dieser Aufwand kann nur durch den konsequenten Einsatz von Entwicklung- und Testmethoden, sowie den entsprechenden Werkzeugen bewältigt werden.

Daher werden heute Entwicklung und Test von elektronischen Systemen im Allgemeinen nach dem V-Modell durchgeführt ([IABG06], bzw. Kapitel 3.9.1.3). Zu Beginn erfolgt zunächst eine Systemanalyse, in welcher der Auftraggeber alleine oder in Zusammenarbeit mit einem Entwicklungsteam des Auftragnehmers die Anforderungen in einem iterativen Prozess ermittelt. Im Idealfall geschieht dies bereits mit Hilfe einer formalisierten Beschreibung, damit die Konsistenz dieser Phase zu den folgenden gewährleistet bleibt. In der anschließenden Spezifikationsphase werden Lasten- und Pflichtenhefte formuliert, die das Verhalten, den Aufbau sowie weitere technische aber auch finanzielle Randbedingungen festlegen. Die vom Auftraggeber geforderten Basismerkmale hinsichtlich Funktion und Ausführung, die das Endprodukt erfüllen muss, werden im Lastenheft zusammengefasst. Nach Abschluss dieser ersten Schritte wird mit dem Systementwurf begonnen, der sich weiter in den Subsystem- und den Modulentwurf gliedert. An den Übergängen der einzelnen Phasen werden Verifikationsschritte zwischengeschaltet, um die Entwicklung auf Konsistenz zu prüfen. Der Abschluss des Entwurfs mündet dann in der Systemrealisierung, in der bereits ein Vorläufer des Produkts verfügbar ist. Bis zur Systemauslieferung und der Einsatzanalyse, werden nun alle entwickelten Systemteile (Module, Subsysteme), sowie das Gesamtprodukt getestet.

Zur Unterstützung der im V-Modell festgelegten Vorgänge kommen Methoden und Testverfahren zum Einsatz, die durch das Erreichen einer frühen Konzeptsicherheit die Fehlerquote senken helfen, zum anderen die entstandenen Teilprodukte gegen die vorher festgelegte Spezifikation testen. Die im Folgenden aufgeführten Methoden sind daher auf beiden Ästen des V-Modells (siehe Kapitel 3.9.1.3) angesiedelt.

		Umwelt	
		Simuliert	Real
System	Modell	Model in the loop	Rapid Prototyping
	Code	Software in the loop	
	Testboard	Processor in the loop	
	Steuergerät	Hardware in the loop	

Abbildung 1.3: Testverfahren zur Unterstützung der Produktentwicklung

Mit Rapid Prototyping (RP) (Kapitel 3.10) können durch eine Validierung der Kundenvorgaben bereits frühzeitig konzeptionelle Fehler eliminiert werden. Es wird daher verstärkt zwischen der Anforderungs- und Spezifikationsphase und zwischen der Spezifikationsphase und dem Systementwurf eingesetzt.

Model-in-the-Loop (MiL) ist der erste Vertreter der X-in-the-Loop-Tests. Hierbei werden die durch CASE-Tools (MatLab/Simulink oder Stateflow [Math04][Math13], ...) in Modellen beschriebenen Zusammenhänge in einer virtuellen Umgebung auf einem Rechner simuliert. Für dieses Verfahren – sowie für alle folgenden X-in-the-Loop Verfahren – wird ein Modell der Umgebung des zu Entwickelnden Steuergeräts benötigt. Das Umgebungsmodell besitzt allerdings einen recht hohen Abstraktionsgrad. Gleiches gilt für Software-in-the-Loop (SiL) Tests, bei denen anstelle eines Modells, bereits der aus dem Modell erzeugte Programm-Code (meist in C) in einer Simulationsumgebung verifiziert wird. Eine zu diesem Zeitpunkt unter Umständen noch nicht verfügbare Zielhardware wird bis zu diesem Zeitpunkt nicht benötigt. Im Gegensatz dazu hat das Processor-in-the-Loop (PiL) Verfahren das Ziel, den in der Simulationsumgebung überprüften Code auf dem tatsächlich eingesetzten Prozessor ablaufen zu lassen. Dieser wird jedoch isoliert betrachtet, d.h. er ist noch nicht im letztlichen Produkt (Steuergerät) eingebaut.

Den Abschluss der X-in-the-Loop-Testverfahren bildet der Hardware-in-the-Loop (HiL) Test (Kapitel 3.11), der ein fertiges Produkt hinsichtlich Hardware und Software verlangt. Seine Aufgabe ist es, den letzten Implementierungsschritt zu verifizieren und darüber hinaus die Überprüfung der Korrektheit des Produkts an der gegebenen Spezifikation vorzunehmen.

Diese Techniken, insbesondere für Rapid Prototyping und Hardware-in-the-Loop, sind in vielen Jahren gewachsen und finden bei der Systementwicklung im Automobilbereich Anwendung. Die dazu existierenden Testplattformen haben einen Stand erreicht, der sehr auf die Erfordernisse der bisherigen Technologien ausgerichtet (Kapitel 3.10.4 bzw. 3.11.2) und getrennte Hardware-Plattformen für RP und HiL vorsieht. Um dem schnellen technologischen Fortschritt standhalten zu können, müssen zum einen die Tests und Testmethoden intensiv weiterentwickelt, zum anderen aber auch die dazu nötigen Hardware-Plattformen an die geänderten Bedingungen angepasst werden. Dazu gehören eine höhere Flexibilität hinsichtlich der Adaptierbarkeit der Schnittstellen und der Systemarchitektur unter Beibehaltung eines modularen Aufbaus. Besonders der steigende Einsatz von schnellen Kommunikationssystemen im Fahrzeug und der damit verbundene Test verlangt nach höheren Abtastraten zur Erreichung einer höheren zeitlichen Auflösung der Signale. Aber auch rechenintensive

Prozesse benötigen entsprechende Hardware-Voraussetzungen, z.B. durch die Möglichkeit einer parallelen Ausführung. Feld programmierbare Bausteine (FPGAs, Field Programmable Gate Arrays – Kapitel 3.1.1) bieten dazu eine sehr gute Alternative zu den bisherigen Konzepten, da mit ihnen eine flexible Implementierung von Hardware, eine schnelle Anpassung an sich ändernde Umgebungen und ein hoher Wiederverwendungsgrad möglich ist.

1.2 Ziele der Arbeit – Der eigene Beitrag

Die Grundidee der Arbeit war es, ein flexibles und leistungsfähiges, aber dennoch kleines und mobiles System auf Basis feld-programmierbarer Logik (FPGA) zu entwickeln, das gleichermaßen für Rapid-Prototyping und Hardware-in-the-Loop eingesetzt werden kann und damit eine Kombinationslösung für beide Zwecke darstellt. Das Hauptaugenmerk lag dabei auf einer möglichst gut parametrierbaren Lösung, die außerdem einfach zu handhaben und zu bedienen ist.

Die Basis der Plattform bildet ein modularer Aufbau, bei dem frei programmierbaren Interface-Karten, Kommunikationsmodule für Bussysteme und Ein-/Ausgabe-Schnittstellen nach Bedarf eingesteckt werden können. Gegenüber den derzeitigen kommerziellen Geräten ergibt sich durch den Einsatz von FPGAs anstelle dedizierter I/O-Komponenten eine Erhöhung der Flexibilität hinsichtlich der Funktionsimplementierung. Die I/O-Funktionen in Form von Schnittstellen-Controller werden als Logikblöcke zur Systemlaufzeit in FPGAs geladen, wobei eine Karte mehrere verschiedene Funktionen (verschiedene I/O-Controller, z.B. digitale I/O, PWM [in, out], Nachbildung serieller Protokolle, …) enthalten kann. Dadurch lässt sich die Anzahl verschiedenartiger Karten erheblich reduzieren. Die Trennung der logischen Ansteuerung und der I/O-Treiber (Signalkonditionierung) unterstützt außerdem eine flexible Anpassung an sich ändernde Prozessumgebungen.

Die Plattform-interne Bussystemkonfiguration besteht aus verschiedenen Kommunikationswegen bzw. -kanälen, die so angelegt sind, dass auch hier eine möglichst große Flexibilität hinsichtlich des Einsatzes der Plattform und der darauf implementierten Systeme erreicht wird. Der Prozessor ist dazu mit allen Interface-Karten verbunden, wobei letztere auch untereinander, unabhängig vom Prozessor, kommunizieren können (Sub-System-Unterstützung, Modul-Synchronisierung, etc.). Die Auswahl eines externen Kommunikationsbussystems (z.B. CAN, LIN) ist nicht an eine Interface-Karte gebunden, sondern kann von allen Steckplätzen aus erreicht und auch zur Laufzeit adaptiert werden.

Zur Konfigurierung der FPGAs auf der Plattform wurde eine Methodik entwickelt, die durch Einführung einer weiteren Abstraktionsebene den kompletten Vorgang der Funktionsimplementierung verbirgt. Dadurch werden keinerlei Kenntnisse an Hardware-Beschreibungsprachen (wie VHDL, Verilog, SystemC, o.ä.), von Synthese-Tools oder FPGA-Interna benötigt, wodurch auch im Umgang oder der Programmierung von FPGAs unerfahrene Benutzer effizient mit der Plattform arbeiten können. Die gewünschten Funktionsmodule werden in einer grafischen Benutzeroberfläche (BitstreamComposer, Kapitel 6.2) aus einer Liste ausgewählt und einzelnen Bereichen innerhalb des FPGA zugewiesen. Eine Bibliothek bietet dazu vorgefertigte, funktions- und performanzgetestete Module. Die Abbildung auf die Strukturen des FPGA wird per Klick automatisch ausgeführt, wobei die ausgewählten Module in ein Framework eingebettet werden, das eine Basisfunktionalität (externe Busankopplung an den Prozessor und interne Bussysteme) bereitstellt. Das Ergebnis ist eine komplette Instrumentierungsdatei für den FPGA. Durch diesen Ansatz kann eine FPGA-Konfiguration binnen weniger Minuten zusammengestellt werden. Zeitraubende Iterationen durch Timing- oder Ressourcenprobleme wie bei herkömmlichen CASE-Tool-Ansätzen fallen nicht an.

Als Alternative zu der obigen Methodik wurde ein durchgängiger Design-Flow für Funktionsimplementierungen auf den FPGAs der Interface-Karten mit Hilfe grafischer Modellbildung durch die Case-Tools MATLAB Simulink/StateFlow und Xilinx SystemGenerator realisiert. Hierbei wird aus der grafischen Repräsentierung ein Modell-Code (VHDL oder C-Code) erzeugt, der eingebettet in Hardware-/Software-Frameworks und auf IP-Prozessoren (z.B.

PPC, MicroBlaze [XMB03] und LEON [LEON01], [Gais02]) lauffähig ist. Die jeweiligen Frameworks adaptieren die verfügbaren Prozessschnittstellen und vereinfachen so die Integration auf dem FPGA erheblich.

1.3 Gliederung der Arbeit

Die hier vorliegende Arbeit ist in folgende Abschnitte gegliedert:

In diesem ersten Kapitel werden das Umfeld der Arbeit, sowie die daraus entstandene Motivation zur Aufgabenstellung dargestellt. Es werden außerdem die abgeleiteten Ziele und der daraus resultierende eigene Beitrag erläutert.

Kapitel 2 gibt einen Überblick über die im heutigen Kraftfahrzeug verwendete Bussysteme und Elektronikkomponenten. Außerdem wird auf den Begriff der „Eingebetteten Systeme“ näher eingegangen. Aktuelle und zukünftige Trends im Bereich der Fahrerassistenzsysteme werden kurz anhand von Beispielen aufgeführt.

Kapitel 3 widmet sich dem Stand der Technik. Hier werden die für das Verständnis der Arbeit wichtigsten elektronischen Bauteile, insbesondere die programmierbaren Logikbausteine (FPGAs), sowie die verwendeten Programmiersprachen vorgestellt. Einen weiteren Schwerpunkt bilden die für die Entwicklung und den Test von Steuergeräten in der Automobilindustrie angewandten Methoden. Hier geht es im Speziellen um die Vorgehensmodelle bei der Entwicklung solcher Systeme und die Verflechtung von Rapid Prototyping (RP) und Hardware-in-the-Loop (HiL) mit den verschiedenen Lebenszyklusmodellen in diesem Zusammenhang. Die Begriffe werden dabei ausgiebig erläutert und die Einsatzgebiete und Wirkungsweisen der Methoden dargestellt.

Kapitel 4 präsentiert das aus den Anforderungen abgeleitete Konzept und dessen Umsetzung in eine kombinierte Rapid Prototyping- und Hardware-in-the-Loop-Plattform als funktionsfähige Hardware. Dabei werden sowohl das Konzept ausführlich erläutert, als auch die Realisierungsalternativen dargelegt und diskutiert. Die daraus entstandenen einzelnen Komponenten, die zum Betrieb der Plattform nötig sind, werden anschließend vorgestellt und deren Aufbau und Funktion detailliert erklärt. Außerdem wird ein für die Plattform entworfenes Sicherheitskonzept zur Fehlervermeidung thematisiert.

Die Designmethodik, mit Hilfe derer auf einfachstem Wege komplette Konfigurationsdatenströme für FPGAs durch Auswählen aus einem Menü erstellt werden können, ist Thema in Kapitel 5. Hier wird ausgehend von der eigentlichen Problematik bei der Konfiguration von FPGAs der Ansatz der Slot-basierten Konfiguration dargelegt und mit der Umsetzung der Methodik in eine funktionierende Umgebung bestehend aus VHDL-Code und einer Java-Applikation namens BitstreamComposer komplettiert.

In Kapitel 6 wird die Software-Umgebung dieser Arbeit beschrieben. Diese gliedert sich in die Steuerungs-Software zur Bedienung des Systems, die Applikationen zur Erstellung der FPGA-Konfigurationsdaten und letztlich das Betriebssystem und dessen Erweiterungen auf der Plattform selbst.

Die Anwendung und Erprobung der in dieser Arbeit entstandenen Hard- und Software-Teile wird in Kapitel 7 dargestellt. Getrennt nach Rapid Prototyping und Hardware-in-the-Loop sind verschiedene Applikationen aufgeführt, die zur Lösung dedizierter Szenarien erstellt und eingesetzt wurden.

Kapitel 8 fasst die Ergebnisse der Arbeit zusammen, zeigt die erzielten Fortschritte, aber auch die Grenzen der Arbeit auf und gibt einen Überblick über die in diesem Zusammenhang entstandenen Hardware- und Softwareteile, sowie deren Funktionen im Gesamtsystem. In einem Ausblick werden die noch nicht umgesetzten Punkte besprochen und bestehende Probleme diskutiert.

Abschließend finden sich in Kapitel 9 noch die Verzeichnisse für Abbildungen, Tabellen, Formeln, verwendete Abkürzungen und eine umfassende Literaturangabe. Die Listen mit be-

treuten Diplom-, Studien und Seminararbeiten, sowie eigenen Veröffentlichungen runden die Aufzählung ab.

2 Elektronik im Kraftfahrzeug

Neben der klassischen Elektrik zur Versorgung der Zünd- oder Lichtanlage mit Energie, sind im Kraftfahrzeug im Laufe der Zeit eine große Anzahl ehemals mechanischer Funktionen elektrifiziert worden. Diese Funktionen müssen umgesetzt, sowie deren einwandfreie Funktion während des Betriebs überprüft werden, um die Sicherheit der Insassen und anderen Personen zu gewährleisten. Dazu werden im Auto verschiedenste elektronische Komponenten in Form von eingebetteten Systemen eingesetzt, die diese Aufgaben übernehmen. Solche Komponenten lassen sich mit dem Oberbegriff der Kfz-Steuergeräte zusammenfassen.

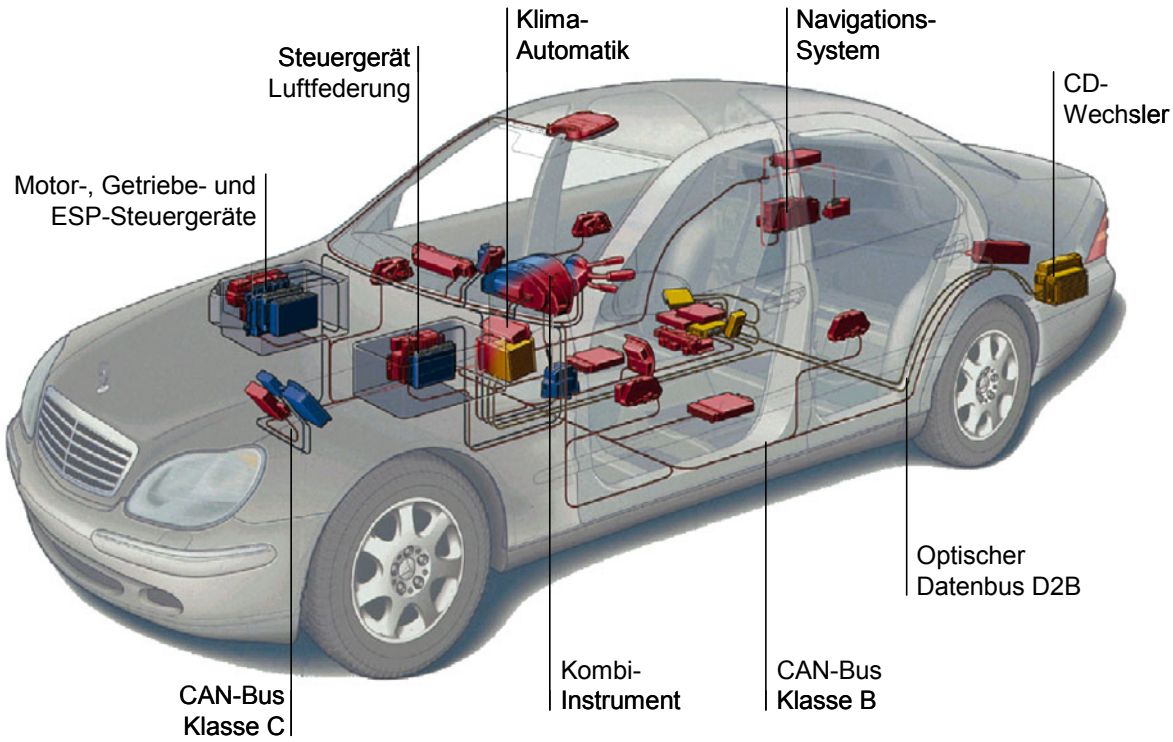

Abbildung 2.1: Steuergeräte und Busse im W220 (S-Klasse) von DaimlerChrysler

Abbildung 2.1 zeigt eine Auswahl an elektronischen Komponenten und Datenkommunikationssystemen, wie sie in heutigen Fahrzeugen zu finden sind. Der Integrationsgrad an Informationselektronik hängt dabei in erster Linie von der Fahrzeugklasse, vom Hersteller und von der vom Kunden gewählten (Sonder-) Ausstattung ab. Während in der Unterklasse Autos noch gänzlich ohne einen CAN-Bus auskommen, sind in der Oberklasse mindestens zwei verbaut und werden zusätzlich noch durch andere Kommunikationssysteme ergänzt. Auch die Verfügbarkeit von Sicherheits- und Komfortkomponenten ist in der gehobenen Kategorie eher gegeben.

2.1 Bussysteme im Automobil

In den heutigen Fahrzeugen werden zur Vernetzung von Steuergeräten so genannte Bussysteme eingesetzt, um Daten zu transportieren. Der noch immer zunehmende Anteil an Elektrik und Elektronik im Kraftfahrzeug und die damit einhergehende Notwendigkeit des Austausches von Informationen macht den Einsatz von hoch-performanten Kommunikationssystemen nötig. Allerdings gibt es dahingegen auch Entwicklungen zur Kostenreduktion in Bereichen, in denen Performanz eine untergeordnete Rolle spielt und es nur auf die Vernetzung ankommt (Body-Elektronik im Komfortbereich). Die einzelnen elektronischen Kontroll-/Steuersysteme (Steuergeräte) sind im gesamten Fahrzeug verteilt (siehe auch Abbildung 2.1). Die einzelnen Steuergeräte müssen in der Lage sein, Daten untereinander austauschen (z.B. benötigt die Getriebesteuerung vom Motorsteuergerät die aktuelle Motordrehzahl).

2.1.1 Unterscheidung von Bustypen

Bei der Kommunikation in modernen Fahrzeugen beschränkt man sich nicht auf ein Medium, sondern verwendet eine Vielzahl verschiedener Busse. Die Vielfalt der verschiedenen Systeme ist allerdings erst mit der Zeit entstanden, wie man in Abbildung 2.2 gut erkennen kann. Einige sind dabei mittlerweile schon an die Grenze ihrer Leistungsfähigkeit gestoßen, so dass die Erforschung und die Entwicklung neuer Technologien zum Datentransport intensiv vorangetrieben wird.

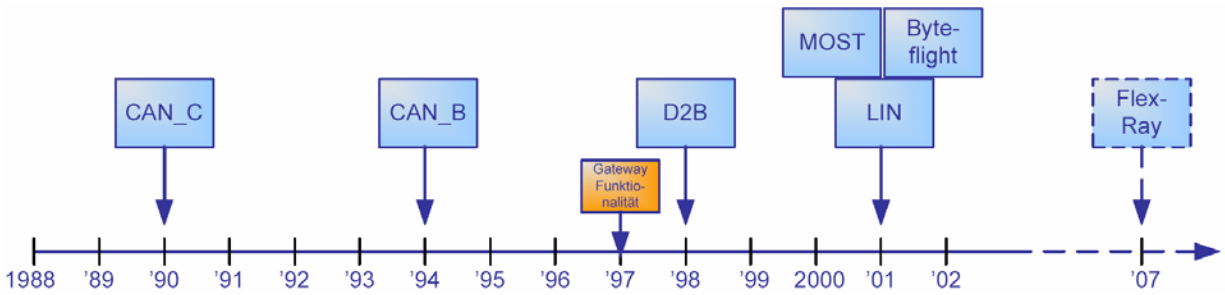

Abbildung 2.2: Etablierung von Kommunikationssystemen im Automobil

Die Bussysteme bedienen entsprechend ihrer Leistungen und Merkmale unterschiedliche Einsatzgebiete und variieren dadurch aber auch stark in Komplexität, Aufbau und Protokoll. Je nach Anforderung und Leistungsbedarf der jeweiligen Applikation (meist gemessen in der Übertragungsgeschwindigkeit) und den Kosten zur Realisierung einer Funktion werden die derzeit gängigsten Typen unterschieden. Ein weiteres Merkmal stellt auch das Übertragungsmedium dar, d.h. ob die Signalisierung über Draht, einen optischen Leiter – wie z.B. POF (Plastic Optical Fiber) – oder durch einen Funkkanal erfolgt.

Abbildung 2.3 zeigt eine Übersicht und einen Vergleich der aktuellen Bussysteme und -protokolle im Kraftfahrzeug. Es wird dabei zwischen Übertragungsgeschwindigkeit und Kosten je Netzknoten unterschieden.

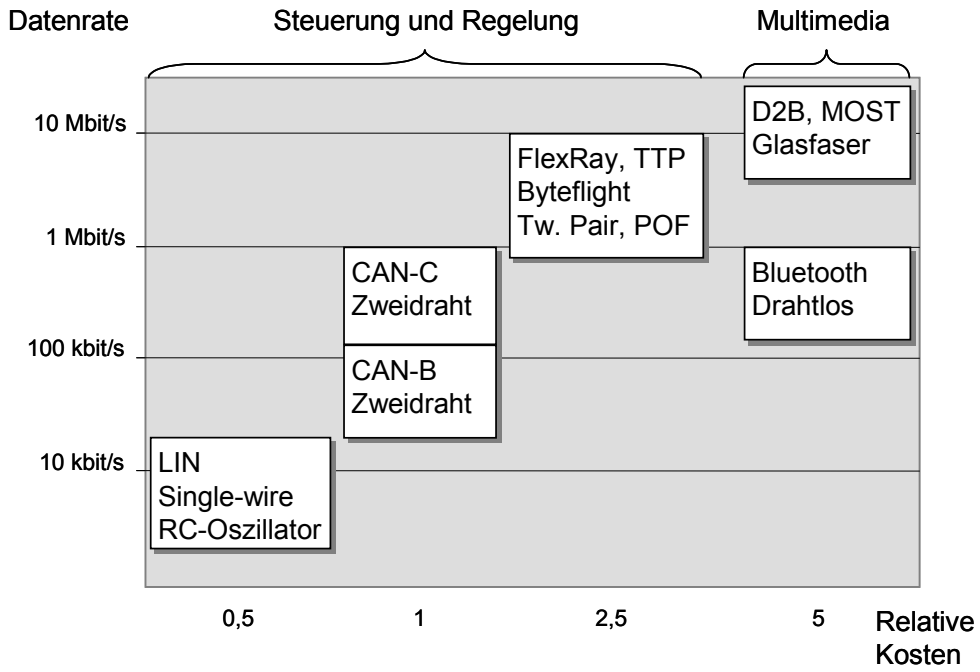

Abbildung 2.3: Aktuell im Automobil eingesetzte Bussysteme

Der nun folgende Abschnitt soll einen Überblick über die gängigen, heute im Kraftfahrzeug verwendeten Bussysteme geben und deren wichtigste Grundlagen zusammenfassen, die für das Verständnis der folgenden Kapitel notwendig sind. Für weitere Details sei an dieser Stelle auf sekundäre Literatur verwiesen, zum Beispiel [Schü97].

2.1.1.1 Topologien

Unter dem Term Topologie wird die Verbindungsstruktur eines Netzwerks verstanden. Es beschreibt die räumliche Anordnung des Verbindungsnetzwerks zum Transport von Nachrichten und Daten zwischen den einzelnen Teilnehmern.

Bei der Stern-Topologie sind alle Knoten eines Clusters über einen so genannten Sternknoten miteinander verbunden (Abbildung 2.4). Zwischen Sternknoten und den einzelnen Busknoten besteht eine Punkt-zu-Punkt-Verbindung. Die Stern-Topologie ist einfach zu verwalten und problemlos erweiterbar. Der einzige Nachteil dieser Realisierung besteht darin, dass beim Ausfall des Sternknotens das gesamte Netz lahm gelegt ist.

Für Netze, die als Ringe ausgeführt sind, gibt es zwei Arten der Realisierung, den Einfachring und den Doppelring (Abbildung 2.5). Beim Einfachring läuft die Kommunikation über einen Zyklus von Punkt-zu-Punkt-Verbindungen. Der Anschluss erfolgt über einen Ringkoppler (nicht im Bild zu sehen). Bei Doppelringen hat jeder Knoten die Möglichkeit in beide Richtungen über jeweils separate Leitungen im Ring zu senden. Die dadurch entstehende Redundanz ermöglicht den Busbetrieb bei Ausfall einer Ringleitung.

Bei Netzen, die als Bustopologie realisiert sind, werden die Knoten über eine gemeinsame Leitung miteinander verbunden (Abbildung 2.6). Der Anschluss jedes Teilnehmers erfolgt über eine Stichleitung. Die Kommunikation über nur eine Leitung impliziert eine aufwendige Kollisionsverwaltung, um einen reibungslosen Datenaustausch zu realisieren.

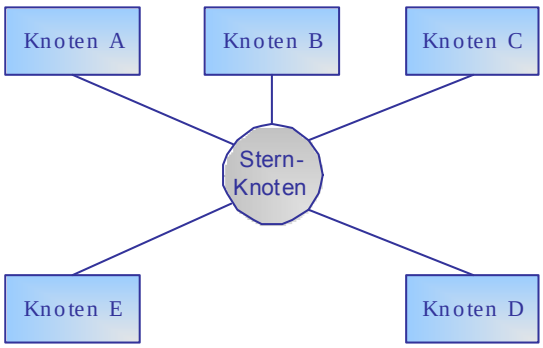

Abbildung 2.4: Stern-Topologie

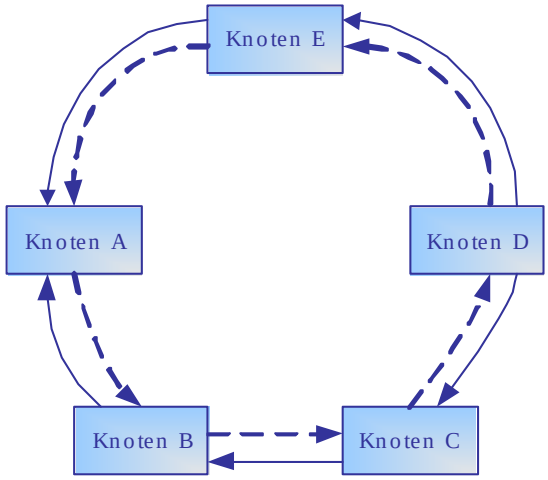

Abbildung 2.5: Ring-Topologie

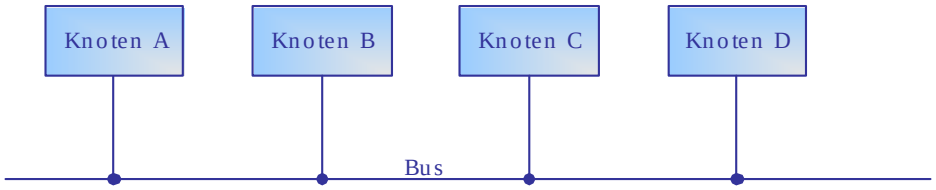

Abbildung 2.6: Bus-Topologie

2.1.1.2 Übertragungsverfahren

Bei der Datenübertragung auf einem Kanal kann hinsichtlich Datenrichtung und Grad der Parallelität unterschieden werden. Bei der Übertragungsrichtung gibt es das Simplex-Verfahren (in eine Richtung) und Halb-/Duplex-Verfahren (in beide Richtungen).

Beim Raummultiplex-Verfahren werden die zu übertragenden Signale auf mehrere Leitungen verteilt. Daraus resultiert die direkte Abhängigkeit der Datenübertragungsrate von der Anzahl der Leitungen – d.h. je mehr parallele Leitungen zur Verfügung stehen, desto mehr Daten können gleichzeitig übertragen werden.

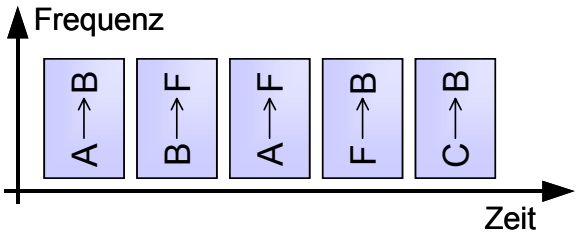

Abbildung 2.7: TDMA-Verfahren

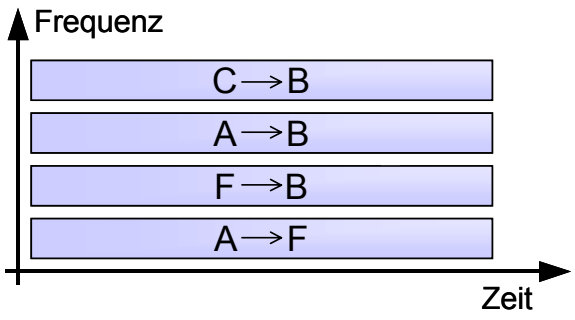

Abbildung 2.8: FDMA-Verfahren

Das Zeitmultiplex-Verfahren (Time Division Multiple Access = TDMA) teilt jedem Sendeteilnehmer (Knoten) einen genau definierten Zeitabschnitt eines Sendezyklus zu (Abbildung 2.7). Innerhalb dieses Slots darf der entsprechende Knoten Daten auf dem Bus übertragen.

Beim Frequenzmultiplex-Verfahren (Frequency Division Multiple Access = FDMA) wird das Frequenzband in Bereiche aufgeteilt, die von den Busteilnehmern zur Datenübertragung genutzt werden können (Abbildung 2.8). Zwischen den einzelnen Bändern ist ein Abstand einzuhalten, um Interferenzen zwischen den Bändern zu vermeiden.

Die zu übertragenden Signale breiten sich innerhalb der Leiter wellenförmig aus. Es existiert immer eine hinlaufende Welle und ggf. auch eine rücklaufende (reflektierte) Welle. Um die Reflektion am Leitungsende möglichst gering zu halten, ist ein Abschlusswiderstand Z_0 (Terminierung) anzubringen.

2.1.1.3 Cyclic Redundancy Check

Der Cyclic Redundancy Check CRC dient zur Fehlererkennung der nach einer Übertragung empfangenen Daten. Dieses Verfahren ist wesentlich effektiver als das Hinzufügen eines Paritätsbits an das Ende einer Sendesequenz. Das CRC-Verfahren basiert auf der Modulo-Zwei-Arithmetik. Dabei werden aber keine Überträge verwendet. Es entspricht also einer XOR-Operation.

$$0100010 + 1010111 \equiv 0001010$$

Formel 2.1: Beispiel einer CRC-Code-Berechnung (XOR)

Je nach Busprotokoll werden entweder nur das Nachrichtenfeld oder Teilbereiche einer Botschaft von einem oder mehreren CRC-Feldern abgesichert. Die Darstellung der Bitstrings erfolgt durch Polynome. Jede Stelle eines Bitstrings, welche eine Eins enthält wird durch das entsprechende n^i repräsentiert (i steht für eine Stelle innerhalb des Bitstrings).

$$01101101 \rightarrow p(n) = n^6+n^5+n^3+n^2+1$$

Formel 2.2: Beispiel für ein Polynom zur CRC-Berechnung

Durch ein so genanntes Generatorpolynom G(n) wird beim Sender eine Prüfsumme erzeugt und mit übertragen. Der Empfänger überprüft anhand der Prüfsumme und des gleichen Generatorpolynoms G(n), ob die Daten korrekt übertragen wurden. Folgende Fehler können mit der Hilfe des CRC-Verfahrens erkannt werden:

- Einzelbitfehler
- Zwei Einzelbitfehler, wenn (n^k+1) nicht durch das Generatorpolynom teilbar ist ($k \leq$ Rahmenlänge)
- Ungerade Anzahl von Bitfehlern, wenn G(n) den Faktor (n+1) enthält
- Alle Fehler-Bursts = Grand des Generatorpolynoms G(n)

2.1.2 Controller Area Network (CAN-Bus)

Das Controller Area Network (CAN) wurde 1981 von den Firmen BOSCH und Intel entwickelt und in der ISO 11898 als ein asynchrones serielles Bus-System mit einer logischen Busleitung definiert [Bosc]. Als Kommunikationsnetzwerk zum Datenaustausch einfacher Prozesse wurde es ursprünglich im Komfortbereich und im Motormanagement bei Kraftfahr-

zeugen eingesetzt, findet heute aber auch Anwendung in der Gebäude- und Produktionsautomatisierung.

Abhängig vom Datenaufkommen und zeitlichen Anforderungen unterscheidet man drei Arten von CAN-Systemen, wie sie in der folgenden Tabelle 2.1 zusammengefasst sind:

Klasse	Bezeichnung	Baudrate
CAN-C	High-Speed CAN	bis zu 1 MBit/s
CAN-B	Low-Speed CAN	bis zu 125 kbit/s
CAN-A	Super Low-Speed CAN	bis zu 10 kbit/s

Tabelle 2.1: Klassifizierung von CAN-Bussen

2.1.2.1 Eigenschaften

Der CAN-Bus basiert auf einer Bustopologie und einer Multimaster-Architektur mit bitserieller Datenübertragung. Die Busarbitrierung erfolgt durch das CSMA/CD-Verfahren (Carrier Sense Multiple Access/Collision Detect). Über den Identifier wird die Nachrichtenpriorisierung vorgenommen – dabei hat der niedrigste Wert die höchste Priorität. Die Daten werden NRZ-codiert inklusive Bitstuffing. Die Datenkonsistenz wird über ein 15Bit großes CRC-Feld und einer Hamming-Distanz von 6 sichergestellt.

Als Übertragungsmedium dient eine abgeschirmte, verdrillte Zweidrahtleitung. Die Ansteuerung erfolgt über einen differentiellen Spannungspegel, der mit Hilfe eines Transceiver-Bausteins (Buskoppler) erzeugt wird. Die Transceiver basieren auf dem Prinzip der symmetrische RS-485-Schnittstellen, von denen je einer an jedem einzelnen Busteilnehmer benötigt wird.

Beim CAN-Bus (Class C) kann nominell eine Datenübertragungsrate von bis zu 1Mbit/s erzielt werden. Durch den Protokoll-Overhead sind reell jedoch nur bis maximal 500kbit/s möglich. Eine weitere Einschränkung, d.h. Reduktion der Übertragungsrate, ergibt sich bei steigender Leitungslänge und einer Erhöhung der Buslast.

Unterstützt werden verschiedene Nachrichtentypen: Daten-, Remote-, Error- und Overload-Nachrichten. Jede Nachricht enthält mindestens ein SOF-Bit (Start-of-Frame) und einen Identifier. Das Maximum der in einem Rahmen (Frame) zu übertragenden Daten sind 8 Byte. In Abbildung 2.9 ist der Aufbau einer CAN-Botschaft dargestellt.

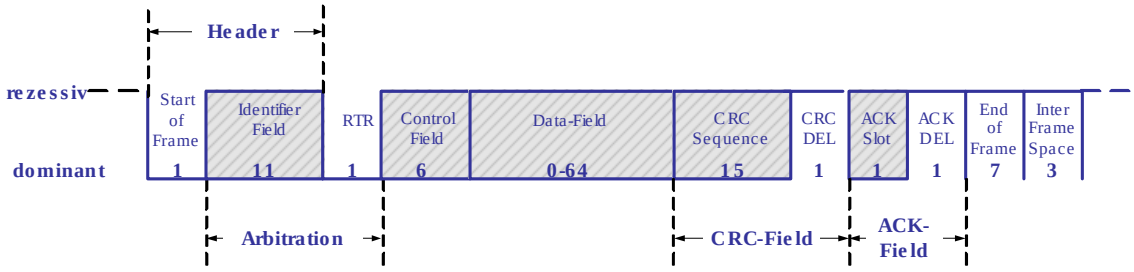

Abbildung 2.9: Aufbau einer CAN-Botschaft

Für genauere Informationen zum CAN-Bus sei an dieser Stelle auf die CAN-Spezifikation 2.0 der Firma Bosch verwiesen [Bosc91]. Diese kann kostenlos von deren Internet-Seite heruntergeladen werden. Außerdem finden sich weitere Details zum CAN-Bus und dessen Einsatz in [Law97], [Enge00], [Etsc01], [Lawr02], [Etsc02], sowie auf der Webseite der Organisation CAN-in-Automation [CiA06]. Letztere bietet auch Dienstleistungen und Literatur zu

auf CAN basierenden Systemen und Protokollen. Die Steckerbelegung nach Tabelle 2.2 wurde ebenfalls von der CiA standardisiert.

Pin	Signal	Beschreibung
1	-	reserviert
2	CAN-L	CAN-L
3	GND	Masse
4	-	reserviert
5	-	reserviert
6	(GND)	Masse (optional)
7	CAN-H	CAN-H
8	-	reserviert (error line)
9	(V +)	externe Versorgung 7V - 13V (optional)

Tabelle 2.2: CAN-Steckerbelegung nach DIN 41652 (CiA/DS 102-1)

2.1.3 Local Interconnect Network (LIN)-Bus

Das Local Interconnect Network (LIN) wurde 1989 als einfaches Bussystem von den Automobilherstellern Audi, BMW, DaimlerChrysler sowie dem Chiphersteller Motorola und dem schwedischen Softwarehersteller Volcano spezifiziert. Mittlerweile gehören dem Konsortium noch weitere Firmen an. Nach der Einführung [LINS13] erfolgte eine kontinuierliche Weiterentwicklung und Verbesserung bis zum heutigen Stand. Mit der Revision 2.0 [LINS20] steht ein ausgereiftes und voll funktionsfähiges Busprotokoll zur Verfügung.

2.1.3.1 Eigenschaften

Das Bussystem LIN basiert ebenfalls auf der Bustopologie mit bitserieller Übertragung der Daten. Ein LIN-Bussystem besteht aus einem Masterknoten und einem bis 16 Slave-Knoten. Die gesamte Kommunikation wird vom Master-Knoten aus gesteuert, so dass die Slave-Knoten sehr einfach zu realisieren sind. Die Übertragungssequenz wird in so genannten Schedule Tables organisiert. Darin erfolgt die Festlegung, in welcher Reihenfolge der Master Nachrichten an die Slaves sendet oder entsprechende Sendeanforderungen schickt. Ein Slave-Knoten darf nicht von sich aus Nachrichten auf den Bus zu legen. Dieser kann erst eine Nachricht an den Master und/oder an ein oder mehrere Slaves senden, wenn dieser die entsprechende Anforderung vom Master-Knoten empfangen hat.

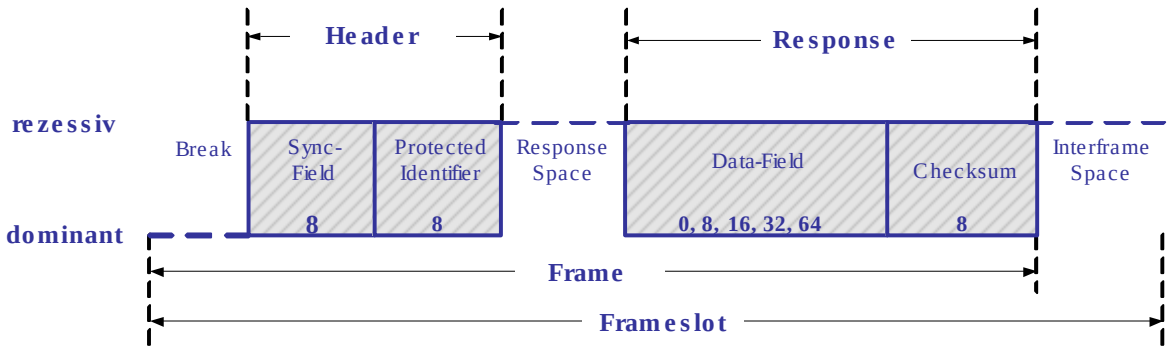

Abbildung 2.10: Aufbau einer LIN-Botschaft

Die Übertragung der Daten erfolgt zur Einsparung von Kosten NRZ-codiert über eine Eindrahtleitung (Bezugspunkt ist Masse). Zur Wahrung der Datenkonsistenz kommen verschiedene Sicherheitsmechanismen zum Einsatz. Das Identifier-Feld enthält ein zwei Bit großes Parity-Feld im Anschluss an den 6Bit langen Identifier. Im Anschluss an das Datenfeld wird eine 8Bit große Checksumme angehängt. Die einzelnen Busknoten werden über eine erweiterte UART/SCI-Schnittstelle an den Bus angekoppelt. Zur Übertragung steht eine maximale Geschwindigkeit von 20kbit/s zur Verfügung. Empfohlen werden die Geschwindigkeiten 2400, 9600 und 19200kbit/s.

Insgesamt kommen fünf verschiedene Nachrichtentypen zum Einsatz: Sporadische, ereignisgesteuerte, zyklische, diagnose- und benutzerdefinierte Nachrichten. Eine Nachricht setzt sich, wie in Abbildung 2.10 dargestellt, zusammen.

Für genauere Informationen zum LIN-Bus sei an dieser Stelle auf die LIN-Spezifikation 2.0 verwiesen, die kostenlos von der Internet-Seite des LIN-Konsortiums heruntergeladen werden kann (URL: www.lin-subbus.org). Außerdem finden sich weitere Details in [Atme05], [Grze03a] - [Grze04c], [LINS13] und [LINS20].

2.1.4 Media Oriented System Transport (MOST)-Bus

Das Kommunikationssystem MOST (Media Oriented System Transport) wurde ursprünglich von BMW entwickelt. Parallel dazu führte DaimlerChrysler das Multimedia-Bussystem D2B ein. Im Jahr 1998 gründeten die beiden Automobilhersteller gemeinsam mit den Zulieferern Harmann/Becker und OASIS Systems das MOST-Konsortium, um einen gemeinsamen Standard für Multimedia-Anwendungen im Automobil zu schaffen.

2.1.4.1 Eigenschaften

Für MOST-Netzwerke stehen verschiedene Realisierungsvarianten zur Verfügung – Punkt-zu-Punkt-Verbindung, Stern- und Ring-Topologie. Die Datenübertragung auf der physikalischen Schicht erfolgt auf einem optischen Medium, z.B. Glasfaser oder POF (Plastic Optical Fiber). Das Einkoppeln der Daten geschieht über einen Electrical Optical Converter (EOC) mit einer Lichtquelle der Wellenlänge von 650nm. Dies hat zur Folge, dass die Übertragungsstrecke bei einer maximalen Datenrate von bis zu 24Mbit/s auf 250m begrenzt ist. MOST unterstützt sowohl synchrone (bis zu 24Mbit/s) als auch asynchrone (bis zu 14Mbit/s) Datenübertragungen.

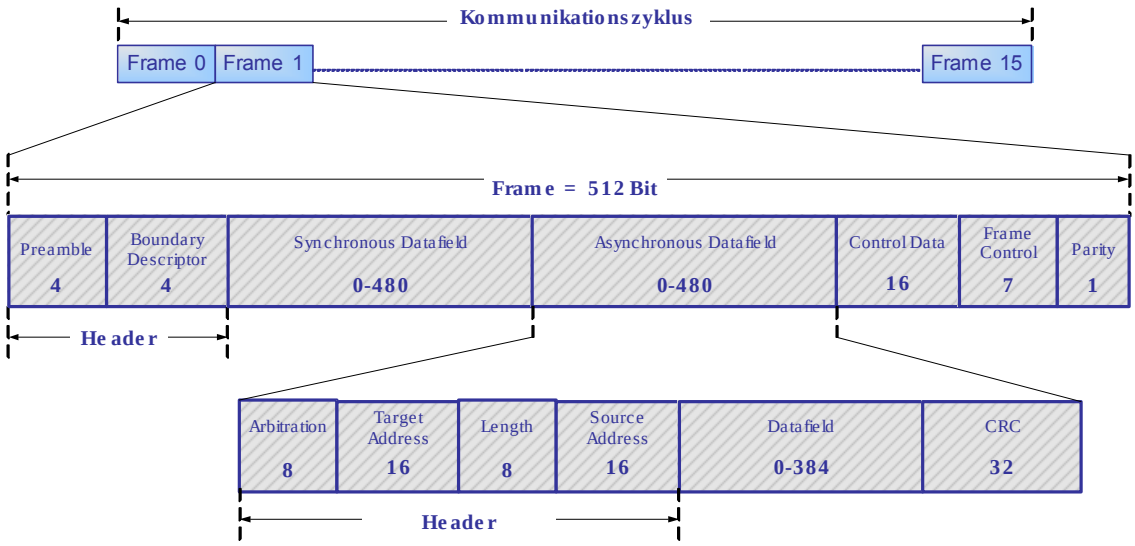

Abbildung 2.11: Aufbau eines MOST-Frames

Eine MOST-Botschaft wird als Frame bezeichnet. Jeder Frame enthält einen Header, ein synchrones sowie ein asynchrones Datenfeld und ein Kontrollfeld. Die Absicherung der Daten wird über ein CRC-Feld, sowie ein Paritätsfeld erledigt. In Abbildung 2.11 ist der Aufbau einer MOST-Botschaft dargestellt.

Für weitere Informationen sei an dieser Stelle auf den Internetauftritt des Konsortiums verwiesen (www.mostconsortium.com).

2.1.5 FlexRay-Bus

Im Jahre 1999 haben die BMW AG und die DaimlerChrysler AG zusammen mit den Halbleiterherstellern Motorola und Philips das FlexRay-Konsortium zur Entwicklung einer neuartigen Kommunikationstechnologie gegründet. Als weitere Mitglieder traten Bosch, General Motors, Volkswagen und Siemens VDO dem Konsortium bei. Das Ziel war und ist die Entwicklung eines echtzeitfähigen, deterministischen und fehlertoleranten Kommunikationssystems. Wie in Abbildung 2.2 dargestellt, soll(te) die Einführung von FlexRay im Jahre 2007 erfolgen und das System danach verstärkt eingesetzt werden. Neueste Entwicklungen zeigen, dass die Etablierung von FlexRay noch einige Jahre benötigen wird und erst nach einer intensiven Evaluierungsphase der Hersteller erfolgt.

2.1.5.1 Eigenschaften

Der eigentliche Grund zur Entwicklung von FlexRay war der geplante Einsatz im Bereich der X-by-Wire-Systeme. Hier werden schnelle und vor allem aber zeitlich deterministische Datenübertragungen verlangt. Doch inzwischen sind viele weitere Anwendungsbereiche hinzugekommen. FlexRay kann in Teilbereichen CAN ablösen und als Grundlage für eine zukünftige Backbone Vernetzungstopologie im Automobil eingesetzt werden. FlexRay bietet eine Reihe von Eigenschaften, die von den bisher im Automobil eingesetzten Kommunikationssystemen nicht abgedeckt werden. Die wesentlichen Eigenschaften von FlexRay sind [DeEl05]:

- Datenrate (10MBit/s)
- Synchronisierte Zeitbasis
- Bekannte Nachrichtenlaufzeit mit garantierter Varianz
- Redundante und nicht redundante Kommunikation (Ein- oder Zweikanalübertragung)
- Flexibilität in Redundanz, Verfügbarkeit, Durchsatz und Vernetzungstopologien

Eine FlexRay-Botschaft setzt sich aus drei Teilen zusammen: Einem Header-Segment, einem Payload-Segment und einem Trailer-Segment. Ein Netzknoten sendet eine Botschaft immer genau mit diesem Aufbau. In Abbildung 2.12 ist der Aufbau einer FlexRay-Botschaft detailliert aufgezeigt.

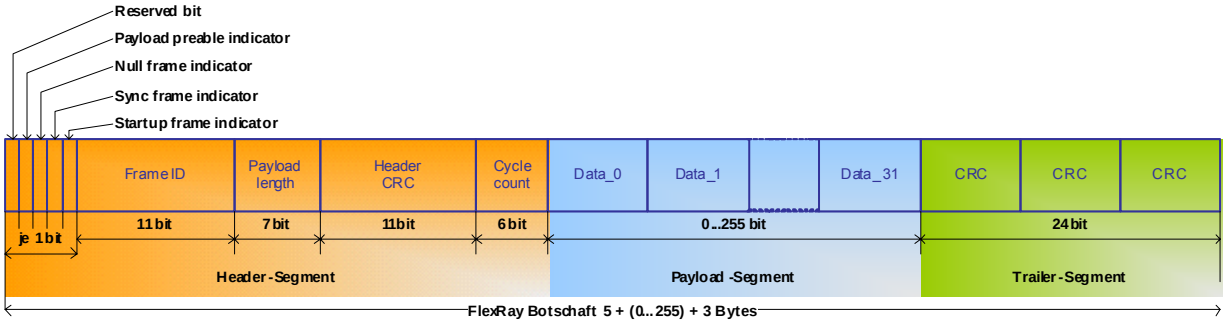

Abbildung 2.12: Datenformat einer FlexRay-Botschaft

Das Header-Segment besteht aus 5 Bytes und dient der Qualifizierung der Daten des Palyload-Segments, zur richtigen Erkennung und Behandlung der Botschaft im Empfänger. Das Payload-Segment enthält die eigentlichen Nutzdaten und kann Längen zwischen 0 und 254 Bytes besitzen. Die einzelnen Bytes des Datenfeldes werden immer von links nach rechts durchnummeriert, beginnend mit dem Data_0 Byte bis zum Data_n Byte. Das Trailer-Segment hat eine Länge von drei Bytes (24Bit) und enthält nur einen Datensatz, den CRC-Frame. Dieser dient zur Prüfung auf Korrektheit der Daten des vorangegangenen Payload-Segments. Die Prüfsumme wird über folgendes Polynom untersucht:

$$x^{24} + x^{22} + x^{20} + x^{19} + x^{18} + x^{16} + x^{14} + x^{13} + x^{11} + x^{10} + x^8 + x^7 + x^6 + x^3 + x + 1$$
$$= (x+1)^2 \cdot (x^{11} + x^9 + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1) \cdot (x^{11} + x^9 + x^8 + x^7 + x^6 + x^3 + 1)$$

Formel 2.3: Berechnung der CRC-Prüfsumme für das Payload-Segment

Für weitere Informationen sein an dieser Stelle auf die FlexRay Protocol Specification 2.1 verwiesen [Flex05].

2.1.6 Time Triggered Protocol (TTP/C)

Das Time Triggered Protocol (TTP) wurde von der Firma TTTech 1998 auf den Markt gebracht. Es wurde für den Einsatz in sicherheitskritischen Bereichen entwickelt, hier hauptsächlich für die Automobilindustrie. Das Protokoll setzt auf der Time Triggered Architecture (TTA) auf und ist ein fehlertolerantes Echtzeitprotokoll.

2.1.6.1 Eigenschaften

Bei TTA stehen verschiedene Netztopologien zur Verfügung. Es können Netzsegmente in Bustopologie oder als Sterne realisiert werden. Weiterhin sind auch Mischformen beider Topologien zulässig. Die Knoten werden immer über zwei replizierte Kommunikationskanäle verbunden. Die Übertragungsgeschwindigkeit hängt von den verwendeten Bustreibern und der Vernetzungstopologie ab. Im besten Fall kann diese bis zu 10MBit/s betragen.

Bei TTP kommen zwei Botschaftsformate zum Einsatz: Der N-Frame (Abbildung 2.14) und der I-Frame. Der I-Frame dient zur Initialisierung. Innerhalb des C-State-Feldes wird der Zustand des Controllers übertragen. Der N-Frame (Abbildung 2.13) wird zur Datenübertragung verwendet. Die Datenkonsistenz einer Botschaft wird durch ein zwei Byte großes CRC-Feld gesichert.

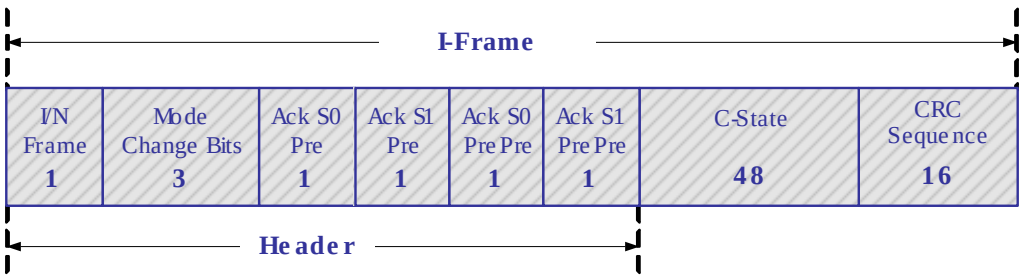

Abbildung 2.13: Botschaftsformat eines I-Frames

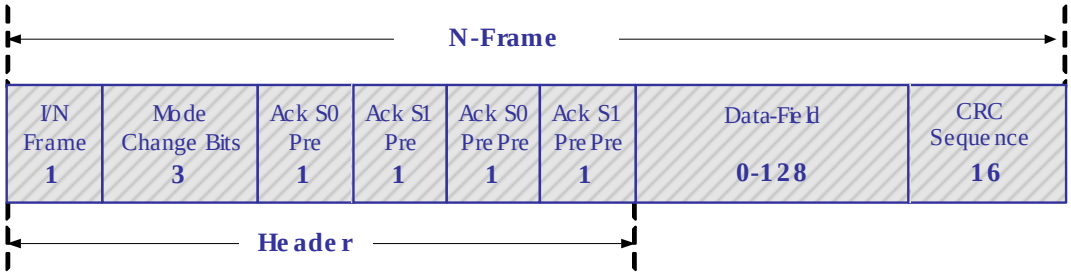

Abbildung 2.14: Botschaftsformat eines N-Frames

Für genauere Informationen zu TTP/C und der TTP-Spezifikation finden sich auf der Internet-Seite der Firma TTTech (URL: www.tttech.com/technology/specification.htm).

2.2 Eingebettete Systeme

Der Begriff Eingebettetes System (englisch: Embedded System) bezeichnet im Allgemeinen ein Rechnersystem, das in ein technisches Gerät zu Lösung komplexer Steuerungs-, Regelungs- oder Datenverarbeitungsaufgaben eingebaut (eingebettet) ist. Das System ist dabei meist nach außen nicht zu erkennen und wird daher vom Benutzer selten als solches wahrgenommen. Die Komponenten, meist Mikrocontroller oder Mikroprozessoren mit entsprechenden weiteren peripheren elektronischen Bauelementen eingesetzt, werden auf einer Platine integriert und mit Sensoren und Aktoren mit dem technischen Gerät, in das sie eingebaut sind, verbunden. Die zur Steuerung oder Regelung des Gerätes nötigen Informationen (Temperatur, Druck, Füllstand, (Winkel-) Geschwindigkeit, Drehzahl, usw.) werden zyklisch über Sensoren eingelesen und teilweise durch Eingaben des Benutzers ergänzt. Über eine in dem Prozessor oder Controller ablaufende Software werden die berechneten Daten über die Aktoren nach außen weitergegeben, d.h. wieder in physikalische Größen gewandelt.

2.2.1 Einsatzgebiete

Da eingebettete Systeme in den meisten Fällen weitestgehend unsichtbar in Geräten, Maschinen und Apparaturen verbaut sind, pflegt der Benutzer einen selbstverständlichen Umgang mit diesen Systemen, ohne davon Kenntnis zu haben. Der Einsatzzweck kann dabei verschiedene Gründe haben: Vereinfachung der Handhabung, Erledigung stupider Routinearbeiten, Bearbeitung präziser und lange anhaltender Tätigkeiten oder die Verknüpfung komplexer Aufgaben in kurzer Zeit. Mindestens einer dieser Gründe trifft bei allen technischen Geräten des täglichen Gebrauchs zu, wie z. B. in Kaffeemaschinen, Wasch- und Spülmaschinen, Mobiltelefonen, Fernsehgeräten, MP3-Playern, elektronischem Kinderspielzeug, usw. – um nur eine kleine Auswahl zu nennen. Aber auch bei komplexen Gesamtsystemen, z.B. im industriellen Umfeld, werden eingebettete Systeme eingesetzt. Ein weiteres und wichtiges Einsatzgebiet eingebetteter Systeme stellt natürlich der Kraftfahrzeugbereich dar, wobei diese dort als Steuergeräte bezeichnet werden. Die Namensgebung ist etwas unglücklich gewählt, da die meisten Rechnersysteme im Kfz nicht nur steuern, sondern Prozesse regeln.

Definition einer Steuerung (DIN 19226):

Die Steuerung, ist der Vorgang in einem System, bei dem eine oder mehrere Größen als Eingangsgrößen andere Größen als Ausgangsgrößen aufgrund der dem System eigentümlichen Gesetzmäßigkeiten beeinflussen. Kennzeichen für das Steuern ist der offene Wirkungsweg oder ein geschlossener Wirkungsweg, bei dem die durch die Eingangsgrößen beeinflussten Ausgangsgrößen nicht fortlaufend und nicht wieder über die selben Eingangsgrößen auf sich selbst wirken.

Definition einer Regelung (DIN 19226):

Die Regelung ist ein Vorgang, bei dem fortlaufend eine Größe, die Regelgröße (die zu regelnde Größe) erfasst, mit einer anderen Größe, der Führungsgröße, verglichen und im Sinne einer Angleichung an die Führungsgröße beeinflusst wird. Kennzeichen für das Regeln ist der geschlossene Wirkungsablauf, bei dem die Regelgröße im Wirkungskreis des Regelkreises fortlaufend sich selbst beeinflusst.

2.2.2 Vernetzung

Eingebettete Systeme können autonom arbeiten oder auch im Verbund mit anderen Systemen gleicher oder ähnlicher Bauart zur Lösung einer Gesamtaufgabe ([Bosc02], [CiA06]). Dazu müssen die einzelnen Systeme miteinander verbunden sein, um untereinander kommunizieren, also Daten austauschen zu können. Dies geschieht über definierte Schnittstellen und Protokolle unter Verwendung drahtgebundener Medien (zum Beispiel CAN-Bus oder TCP/IP über Ethernet) oder über Funk (WLan, Bluetooth oder ZigBee). Mitunter kommen auch Glasfaserverbindungen als Kommunikationsmedium zum Einsatz, wenn es sich um EMV-kritische Applikationen (Flugzeug, Auto) handelt oder hohe Übertragungsraten gefordert sind (MOST-Bus im Auto).

2.3 Anforderungen an Eingebettete Systeme im Kfz

Je nach Einsatzbereich werden unterschiedliche Anforderungen an das oder die eingebetteten Systeme gestellt. Das Automobil bietet dafür eine der anspruchsvollsten Umgebungen, die es zu beherrschen gilt. Es gehört zu der Gruppe der sicherheitskritischen Systeme, da Menschen darin befördert werden, hat nur eingeschränkte Energiereserven zur Verfügung und ist Umwelteinflüssen ausgesetzt. Die eingesetzten elektronischen Komponenten müssen daher auch unter ungünstigsten äußeren Bedingungen zuverlässig arbeiten.

2.3.1 Umgebungseinflüsse

Elektronische Systeme im Auto sind teilweise Umwelteinflüssen ausgesetzt, die bei der Entwicklung mit eingeplant werden müssen. Eine der wichtigsten und einflussreichsten Größen ist die Temperatur. Je nach Einsatzort, Jahreszeit und Wetter können in extremen Situationen zwischen –40°C und +200°C auftreten [VDE06]. Tabelle 2.3 zeigt einige Beispiele von Extremtemperaturen, klassifiziert nach Einbauorten im Fahrzeug.

Bauraum		**Temperatur**	
		Min.	**Max.**
Motorraum	Drosselklappe	-40°C	204°C
	Ansaugrohr	-40°C	121°C
	Spritzwand	-40°C	141°C
Fahrwerk	Isolierte Bereiche	-40°C	84°C
	An Wärmequellen	-40°C	121°C
	Antriebsstrang	-40°C	177°C
Außenbereich	Beliebig	-40°C	113°C
Innenraum	Armaturenbrett	-40°C	84°C
	Dachhimmel	-40°C	113°C
Kofferraum	Beliebig	-40°C	84°C

Tabelle 2.3: Extremtemperaturen nach Einbauräumen gemäß SAE-Standard J1211

Hinzu kommen weitere Faktoren wie chemische Substanzen (heißes, tropfendes Öl, aufgewirbeltes Salzwasser im Winter oder Treibstoffgase), die das Gehäuse nicht beschädigen und vor allem nicht in dasselbe eindringen dürfen, mechanische Einwirkungen durch starke Vibration und Beschleunigungen sowie Störungen des Spannungsversorgungsnetzes im Fahrzeug. Im Zuge der Fahrzeugvernetzung durch Funkverbindungen sind die Steuergeräte auch elektromagnetischen Störungen ausgesetzt.

2.3.2 Zeitbedingungen

Eingebettete Systeme müssen insbesondere im Auto (z.B. beim Motormanagement) Echtzeitanforderungen genügen [Bosc01], [Sens01], [Bosc02], [Just04]. Dies bedeutet, dass innerhalb einer bestimmten Zeitspanne die Neuberechnung eines Ausgabewertes erfolgt sein muss. Das schließt z.B. das Einlesen eines Sensorwertes, die Berechnung und die Ausgabe ein. Bei solchen Anwendungen werden Betriebssysteme eingesetzt, mit deren Hilfe eine zeitgesteuerte Abarbeitung der Funktionen eingehalten und so die Echtzeitfähigkeit garantiert werden kann. In der Regel arbeitet man mit Echtzeitbetriebssystemen wie VxWorks, Nucleus, OSEK, OS-9 usw., die für den Einsatz in Rechnersystemen mit eingeschränkten Ressourcen optimiert sind. Allerdings finden auch spezielle eingebettete Versionen von Standardbetriebssystemen wie Linux (Embedded Linux), NetBSD oder Windows (CE, XP Embedded, Automotive oder Embedded for PoS) inzwischen weitere Verbreitung. Sind die Systeme sehr klein, kann im Zweifel auch auf ein Betriebssystem völlig verzichtet werden, da die Existenz eines solchen, unnötigen Speicherplatz belegen und Rechenzeit beanspruchen würde, ohne wesentliche Vorteile durch ein strukturiertes Task-Scheduling, d.h. das kontrollierte Abarbeiten von Programmteilen, zu erbringen.

2.4 Aktuelle und zukünftige Trends im Automobil

2.4.1 Reduktion von Steuergeräten – Verteilte Funktionen

Mit dem verstärkten Einsatz von eingebetteten Systemen (Steuergeräten) in Automobilen tritt derzeit ein weiteres Problem zu Tage. Die Anzahl der Systeme ist inzwischen so hoch, dass neben den geringen Platzressourcen und des erhöhten Verdrahtungsaufwands, vor allem der Kommunikationsbedarf immens ansteigt. Daher geht man in neuen Steuerungen dazu über, mehrere Funktionen in einem Steuergerät zu vereinen. Dabei wird nicht wie bisher, lediglich eine dedizierte Funktion auf ein Steuergerät abgebildet, sondern versucht, diese auf mehrere innerhalb des Verbunds zu verteilen.

Dies hat zum einen den Vorteil, dass die Ressourcenanforderungen besser an den Bedarf angepasst werden können und zum anderen bei möglichen Ausfällen eines Steuergeräts, andere Systemteile diese Aufgabe wahrnehmen können. Eine Teilung in hochperformante Funktionen des Motormanagements und der Getriebesteuerung und weniger zeitkritischen Innenraumfunktionen wird dabei aber weiterhin erhalten bleiben. So wäre es beispielsweise denkbar, dass Teile der Lichtsteuerung und der Klimaregelung auf einem Steuergerät implementiert werden.

Definition: Funktionen im Auto

Eine Funktion im Auto ist eine von einer elektrischen oder elektronischen Komponente ausgeführte Aktion, die der Sicherheit, der Unterhaltung, der Information oder der Bequemlichkeit dient. Sie kann entweder vom Fahrer selbst ausgelöst werden (Blinker, Fensterheber, Sitzverstellung, Lüftung, …) oder autonom, d.h. ohne Zutun eines Benutzers, selbständig agieren (Motorsteuerung, Getriebesteuerung, etc.).

2.4.2 Fahrerassistenzsysteme

Fahrerassistenzsysteme wurden und werden entwickelt, um den Fahrer während der Fahrt zu unterstützen und dadurch zu entlasten. Hinsichtlich ihrer Wirkung unterscheidet

man zwei Kategorien. Passive Systeme, die nur zur Information dienen und aktive, die in das Fahrgeschehen eingreifen können.

Zu der Gruppe der passiven Systeme zählen z.B. das Human Machine Interface (Mensch-Maschine-Schnittstelle) und die Verkehrsschildererkennung (Traffic Sign Recognition). Beispiele für aktive Systeme sind die Abstandsregelung (Adaptive Cruise Control), der darauf aufbauende Stauassistent (Traffic Jam Assist) und der Einparkassistent (Park Assist). Letztere sind außerdem sicherheitskritische Systeme, da sie entscheidend für die Sicherheit des Gesamtsystems – hier des Fahrzeugs – und der Insassen Verantwortung tragen.

Definition: Sicherheitskritische Systeme

Sicherheitskritische oder auch sicherheitsrelevante Systeme bezeichnen Systeme, welche die Sicherheit einer Anlage garantieren bzw. sie unterstützen, d.h. die Unversehrtheit von Personen in deren Aktionsbereich gewährleisten. Im Auto sind das Systeme, die – wie oben bereits genannt – aktiv in das Fahrgeschehen eingreifen, wie X-by-Wire, ABS, ESP, ACC, etc. und bei Fehlfunktionen die Sicherheit der Insassen gefährden können.

3 Stand der Technik

Nach der Einführung in die Materie und der Vorstellung des Umfeldes, soll hier nun der Stand der Technik dargelegt werden. Ein besonderer Schwerpunkt liegt dabei auf den feldprogrammierbaren Logikbausteinen (Field Programmable Gate Array, FPGA) und deren Programmierung bzw. Konfigurierung. Alle weiteren Kapitel decken die Grundlagen, die zur Konzeptionierung, Entwicklung und dem Aufbau der Hardware-Plattformen und Software-Applikationen des RP-HiL-Testsystems benötigt wurden, ab. Außerdem wird auf die Methoden und Werkzeuge, sowie die kommerziellen Systeme, die bei der Entwicklung und dem Test eingebetteter elektronischer Systeme im Bereich der Automobilindustrie eingesetzt werden, eingegangen.

Einschränkend muss hier aber bereits erklärt werden, dass nicht alle Details, die während der Entwicklung einer solchen kombinierten Hardware- und Software-Lösung berücksichtigt werden müssen, behandelt werden können. Die erschöpfende Darstellung der Themen und Überlegungen würde durch Umfang und Komplexität den Rahmen der Arbeit sprengen. Der interessierte Leser findet die Grundlagen der Digitaltechnik in [Beck05], Grundlagen zum Entwurf von Schaltwerken in [Teic97] und das nötige Basiswissen zum VHDL-Design in [LeWS94] und [MüGl06]. Die Entwicklung von Systemen oder Teilsystemen analoger oder digitaler Art wird in [TiSC02] und [MüGl05] behandelt. Weiterhin existieren Informationsschriften und Tutorials zu den verwendeten Tools im Internet, die kostenlos heruntergeladen werden können und somit der Ergänzung des hier Dargelegten dienen.

3.1 Programmierbare Bausteine

Bei der Herstellung von ICs gilt es einen Kompromiss zwischen Preis, Stückzahl und Anwenderflexibilität einzugehen. Dies führt, wegen der unterschiedlichen Randbedingungen, zu diversen Lösungsangeboten der Hersteller.

Integrierte Schaltungen lassen sich grob in „Anwenderspezifische Integrierte Schaltkreise (ASIC)“ und „Standard-ICs“ unterteilen. Für eine genauere Unterscheidung der ASICs ist kennzeichnend welche Seite, ob Hersteller oder Anwender, die Konfiguration vornimmt. So hat der Hersteller die Möglichkeit mit Custom-ICs, die auf „Standard-Zellen“, „Allgemeinen Zellen“ oder „Full Custom“ beruhen, bereits konfigurierte ICs anzubieten, welche sehr kompakt und bei hohen Stückzahlen auch wirtschaftlich sind. Vollkundenspezifische ICs werden auf Transistorebene entwickelt. Da dedizierte Masken hergestellt werden müssen, resultiert eine lange Lieferzeit und ein hoher Preis. Bei den Zellentechniken kann auf die Verschaltung bewährter Zellstrukturen aus einer, vom Hersteller als Betriebsgeheimnis gehütete, Zellenbibliothek zurückgegriffen werden.

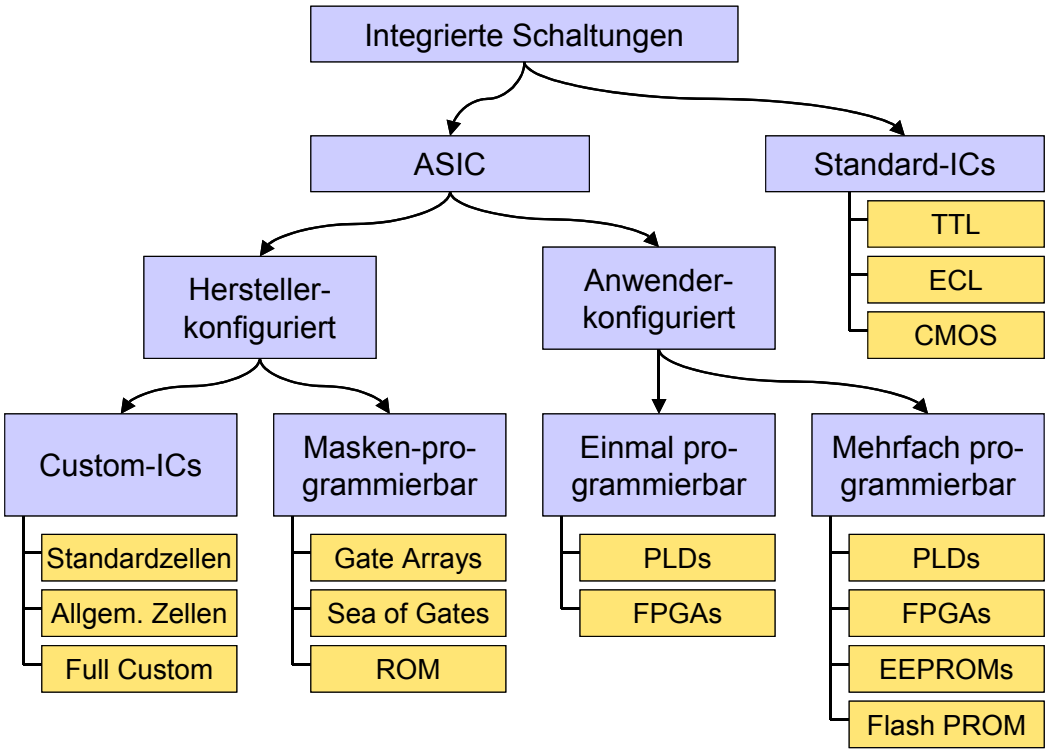

Abbildung 3.1: Integrierte Schaltungen

Um auch in Anwendungssektoren mit geringeren Stückzahlen herstellerkonfigurierte Logikbausteine kostengünstig bereitzuhalten, werden so genannte „Maskenkonfigurierbare ASICs“ angeboten. Wie der Name bereits andeutet, legt der Hersteller die Funktion dieser ICs durch die Erzeugung von Masken für das Abarbeiten der letzten Prozessschritte fest. Vertreter dieser Klasse sind „Gate Arrays“, „Sea of Gates“ und ROMs. Es sei hier kurz angemerkt, dass die Beherrschung der Mehrlagen-Metallisierung zu einer Weiterentwicklung der Gate-Arrays zu den Sea-of-Gates führte, da die Verdrahtungskanäle zwischen den regelmäßig angeordneten Zellen (bestehend aus aneinander gereihten Transistoren) sich nun über den Zellen befinden und der Aufbau damit kompakter ist.

Wichtige Vertreter des Gebiets der anwenderprogrammierbaren Schaltkreise sind „Programmable Logic Devices“ (PLDs), „Field Programmable Gate Arrays“ (FPGAs) und Anwender programmierbare Speicherbausteine.

3.1.1 Field Programmable Gate Arrays (FPGAs)

FPGAs sind auf eine möglichst hohe Nutzung der Logikressourcen ausgelegt. Eine Vielzahl kleiner Logikblöcke ermöglicht eine hohe Komplexität und Anpassungsfähigkeit. Diese Vorteile werden aber durch eine aufwändige Verdrahtung erkauft, weswegen es Schwierigkeiten mit knappen Zeitbedingungen geben kann. Der hohe Verdrahtungsaufwand bringt es auch mit sich, dass beispielsweise ein Virtex-II-1000-FPGA (z.B. XC2V1000BGA256) der Firma Xilinx theoretisch eine Million Gatter besitzt, diese jedoch praktisch nicht alle verdrahtbar sind.

Eine grobe Unterscheidung der verschiedenen FPGA-Typen einzelner Hersteller am Markt kann anhand der folgenden Liste geschehen:

- ROM-, Flash- oder RAM-basiert
- dynamisch bzw. nicht dynamisch re-konfigurierbar

- partiell bzw. nicht-partiell re-konfigurierbar
- innerer Aufbau (Blockorientierung, Streifenorientierung, ...)

Je nach Anwendung und Anforderungen der Applikation und Umgebung, haben die unterschiedlichen FPGA-Typen ihre Vorzüge und bieten damit die passende Ressource für das jeweilige Design.

Während die Firma Actel bei ihren FPGAs auf Multiplexer und eine Anti-Fuse-Technologie setzt, benutzen Altera und Xilinx eine re-konfigurierbare SRAM-Technik. Ein FPGA mit Anti-Fuse-Technologie ist zwar nur einmal programmierbar (OTP, One Time Programmable), andererseits geht sein Inhalt nach Abschalten der Versorgungsspannung aber auch nicht verloren. Das ist jedoch dann nachteilig, wenn man verschiedene Konfigurationen über der Zeit (z.B. auch nur testweise) auf dem Baustein aufbringen will. Daher eignen sich diese FPGAs nicht zum Einsatz in einem RP- oder HiL-System.

3.1.2 Virtex-II-FPGAs von Xilinx

Die Virtex-II -FPGAs der Firma Xilinx basieren auf einer SRAM-Technologie und kombinieren verschiedene Elemente wie Logikfunktionsblöcke, RAM-Speicherblöcke, Multiplizierer und Prozessoren auf einem Chip (Plattform) – daher rührt auch der Name „Plattform-FPGA“. Dies erlaubt eine effektive Entwicklung von komplexen Systemen bei gleichzeitig niedrigen Systemkosten und kurzen Entwicklungszeiten. Die FPGAs sind in Komplexitäten von 40000 bis acht Millionen Systemgattern verfügbar, arbeiten mit einer maximalen Taktfrequenz von 420 MHz und sind optimiert für einen hohen Datendurchsatz und große Packungsdichte. Um allgemeine Regelungs- und Steuerungsaufgaben zu unterstützen, stehen On-Chip-Prozessoren zur Verfügung. Durch die verwendete SRAM-Technologie besteht die Möglichkeit, die Konfigurationsdaten aus dem FPGA zurück zu lesen oder den FPGA partiell zu re-konfigurieren.

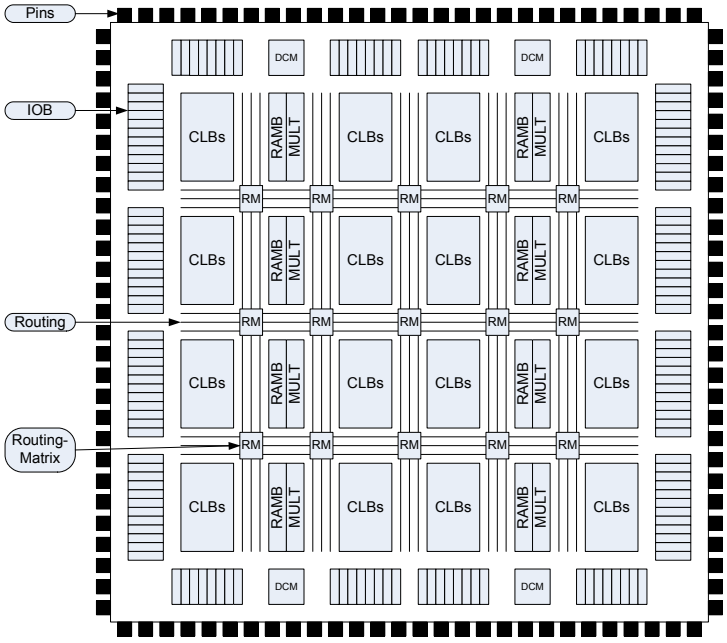

Abbildung 3.2: Architekturschema eines Xilinx-Virtex-II-FPGAs

Die wichtigsten Elemente eines FPGAs sind in Abbildung 3.2 dargestellt. Ein Configurable Logic Block (CLB) ist das Basiselement programmierbarer Logik und enthält Look-Up-Tables (LUTs) zur Realisierung logischer Funktionen, Speicherelemente, Multiplexer und zusätzliche Logikgatter. Tristate-Buffer ermöglichen den Aufbau von On-Chip-Bussen. Die Kommu-

nikation des FPGA mit der Umgebung findet mit Hilfe der bidirektionalen Input- und Output-Blöcke (IOB) statt. Dedizierte Blöcke mit einer erhöhten Packungsdichte und optimiertem und nicht änderbarer Funktion sind in Form der BlockRAMs (je 18kBit), der 18Bit-Multiplizierer und von DCMs (Digital Clock Manager) gegeben. Letztere dienen der Restaurierung und Modifizierung des extern eingespeisten Taktes. Für die Verbindung der verschiedenen Elemente auf dem FPGA steht ein hierarchisches System von Routing-Leitungen zur Verfügung, die untereinander über Switch-Matrizen verbunden sind.

3.1.3 Bitstream und Konfigurationslogik

Der Zustand der programmierbaren Elemente (z.B. LUTs, Leitungen, IOB-Standards), die das Verhalten des FPGA festlegen, werden durch Speicherelemente kontrolliert. Diese müssen beim Einschalten des FPGA konfiguriert werden. Die Daten zur Steuerung der Kontroll-Logik und der Inhalt des Konfigurationsspeichers werden dem FPGA in einem so genannten Bitstream übergeben.

3.1.3.1 Konfigurationsspeicher

Der Konfigurationsspeicher des FPGA wird unterteilt in vertikal angeordnete Einheiten, den so genannten Frames. Diese stellen die kleinste Einheit dar, die in einem Schritt programmiert werden kann. Eine bestimmte Anzahl von Frames wird zu einer Spalte zusammengefasst, die abhängig von ihrer Funktion in sechs verschiedene Gruppen eingeteilt werden kann:

- IOB-Spalten: Die IOB-Spalten legen den Spannungstyp für die Ein- und Ausgangspins am linken und rechten Rand des FPGA fest. Die Konfiguration der Ein- und Ausgangspins an der Ober- und Unterseite des FPGA wird in den CLB-Spalten festgelegt. In jedem FPGA existieren zwei IOB-Spalten.
- IOI-Spalten: Die IOI-Spalten konfigurieren IOB-Register, Multiplexer und Tristate-Buffer der Ein- und Ausgänge am linken und rechten Rand des FPGA. Für die Konfiguration der Ein- und Ausgänge an der Ober- und Unterseite des FPGA sind wieder die CLB-Spalten zuständig. Jeder FPGA enthält zwei IOI-Spalten.
- CLB-Spalten: Die CLB-Spalten haben die Aufgabe, die Funktion der CLBs und die Verbindungsleitungen zu programmieren. Sie enthalten weiterhin die Konfiguration der Ein- und Ausgangspins an der Ober- und Unterseite des Bausteins. Die Anzahl der CLB-Konfigurationsspalten ist identisch mit der Anzahl der im FPGA vorhandenen CLB-Spalten.
- BlockRAM-Spalten: Der Inhalt und die Konfiguration der BlockRAM-Speicherblöcke wird festgelegt durch die BlockRAM-Spalten. Die Anzahl der Spalten ist identisch mit der im FPGA vorhandenen Anzahl an BRAM-Spalten.
- BlockRAM-Interconnect-Spalten: Die BlockRAM-Interconnect-Spalten legen alle weiteren Einstellungen für die BlockRAMs und Multiplizierer fest. Ihre Anzahl entspricht der im FPGA vorhandenen BlockRAM-Spalten.
- GCLK-Spalten: Jeder Virtex-II besitzt eine GCLK-Spalte. Diese enthält die Kon- figuration der Taktnetze, Clock-Buffer und Digital Clock Manager.

Der für diese Arbeit verwendete FPGA XC2V1000 von Xilinx besitzt insgesamt 1104 Frames und eine Frame-Länge von 106 Wörtern mit einer Breite von je 32 Bit. Daraus ergibt sich für die Konfiguration eine Gesamtzahl von 3744768 Bits. Tabelle 3.1 stellt die Anzahl der verschiedenen Spalten dar. Die Anzahl der Frames in einer Spalte von einem bestimmten Typ kann der Tabelle 3.2 entnommen werden.

Anzahl der Spalten geordnet nach Typ					
GCLK	IOB	IOI	CLB	BRM	BRAM Interconnect
1	2	2	32	4	4

Tabelle 3.1: Anzahl der vorhandenen Konfigurationsspalten (XC2V1000)

Frames pro Spalte					
GCLK	IOB	IOI	CLB	BRM	BRAM Interconnect
4	4	22	22	64	22

Tabelle 3.2: Anzahl der Frames in einer Konfigurationsspalte (XC2V1000)

Die Anzahl der Spalten vom Typ GCLK, IOB und IOI ist in allen Virtex-II-FPGAs identisch. Dies gilt jedoch nicht für die Spalten vom Typ CLB, BRAM und BRAM-Interconnect, bei denen die Anzahl der Spalten von der Größe des Bausteins abhängt. Abbildung 3.3 stellt die Anordnung der im Virtex-II XC2V1000 vorhandenen Konfigurationsspalten dar.

	Block-Adresse	Major-Adresse
GCLK	0	0
IOB	0	1
IOI	0	2
CLB	0	3
CLB	0	4
CLB	0	5
CLB	0	6
CLB	0	7
CLB	0	8
CLB	0	9
CLB	0	10
CLB	0	11
CLB	0	12
CLB	0	13
CLB	0	14
CLB	0	15
CLB	0	16
CLB	0	17
CLB	0	18
CLB	0	19
CLB	0	20
CLB	0	21
CLB	0	22
CLB	0	23
CLB	0	24
CLB	0	25
CLB	0	26
CLB	0	27
CLB	0	28
CLB	0	29
CLB	0	30
CLB	0	31
CLB	0	32
CLB	0	33
CLB	0	34
IOI	0	35
IOB	0	36
BRAM	1	0
BRAM	1	1
BRAM	1	2
BRAM	1	3
BRAM INT	2	0
BRAM INT	2	1
BRAM INT	2	2
BRAM INT	2	3

Abbildung 3.3: Anordnung der Konfigurationsspalten für den Virtex-II XC2V1000

3.1.3.2 Adressierung des Konfigurationsspeichers

Zur Adressierung eines bestimmten Frames im Konfigurationsspeicher wird eine Block-Adresse, eine Major-Adresse und eine Minor-Adresse benötigt. Die Major-Adresse wählt eine bestimmte Spalte aus dem von der Block-Adresse bestimmten Block der Konfigurationsspalten aus. Die Minor-Adresse selektiert einen Frame in der entsprechenden Konfigurationsspalte. Beim Programmieren des kompletten Konfigurationsspeichers wird ein Frame nach dem anderen in den Konfigurationsspeicher geschrieben, beginnend bei der niedrigsten Adresse (Block-Adresse = 0, Major-Adresse = 0, Minor-Adresse = 0). Die Adresse wird im Frame Address Register übergeben (siehe Abbildung 3.4). Die obersten fünf Bit besitzen immer den Wert 0. Danach folgt die Angabe der Adresse. Diese setzt sich aus der Block-Adresse (2 Bit), der Major-Adresse (8 Bit) und der Minor-Adresse (8 Bit) zusammen (Tabelle 3.3). Die untersten neun Bit beschreiben die Auswahl eines einzelnen Bytes in einem Frame. Da beim Virtex-II keine einzelnen Bytes in einem Frame adressiert werden können, müssen diese neun Bit immer auf den Wert 0 gesetzt werden.

					BA		Major-Adresse								Minor-Adresse								Daten								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0

Abbildung 3.4: Aufbau des Adressregisters (Frame Address Register)

Block-Adresse	Adressierte Spalten-Typen
00	GCLK, IOB, IOI und CL
01	BlockRAM-Spalten
10	BlockRAM Interconnect-Spalten

Tabelle 3.3: Blockadressen

3.1.3.3 Zerlegung eines Bitstreams in Packets

Der Bitstream eines Virtex-II-FPGA wird in so genannte Packets zerlegt. Alle Daten in einem Packet werden in 32-Bit-Worte aufgeteilt. Das erste Packet im Bitstream ist ein Synchronisationswort mit dem hexadezimalen Wert 0xAA995566, das die Konfigurationslogik auf den Empfang des Bitstreams vorbereitet. Danach folgt eine bestimmt Anzahl Daten-Packets. Ein Packet besteht aus einem vorangestellten Header und einem darauf folgenden Datenteil mit variabler Länge. Dabei unterscheidet man zwei Header-Typen:

Header-Typ 1

Ein Header mit dem Typ 1 wird für kleinere Datenpakete benutzt, die Anzahl der auf den Header folgenden Datenworte ist auf $2^{11}-1$ Worte begrenzt. Abbildung 3.5 enthält den Aufbau eines Headers vom Typ 1.

Header-Typ 2

Dieser Header-Typ findet Verwendung, wenn viele Datenworte in einem Packet übertragen werden müssen, es können maximal $2^{27}-1$ Worte an den Header angehängt werden. Den Aufbau eines Headers vom Typ 2 zeigt Abbildung 3.6. Ein Header mit dem Header-Typ 2 folgt immer auf einen Header mit dem Header-Typ 1, dessen Anzahl folgender Datenworte null beträgt. Es tritt also nur die Kombination ”Header-Typ 1, Header-Typ2“ auf.

Typ			WR	RD	Registeradresse														RSVD		Wortanzahl										
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0	x	x	x	x	x	x	x	x	x	x	x

Abbildung 3.5: Aufbau eines Headers vom Typ 1

Typ			WR	RD	Wortanzahl																										
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0	x	x	x	x	x	x	x	x	x	x	x

Abbildung 3.6: Aufbau eines Headers vom Typ 2

3.1.3.4 Verarbeitung der Packets

Die Konfigurationslogik zur Steuerung des Konfigurationsvorgangs besteht aus einer Einheit zur Verarbeitung der Packets und verschiedenen Registern und Signalen, welche die Konfigurationslogik steuern. Nachdem die Verarbeitungseinheit das Synchronisationswort empfangen hat, werden die ankommenden Packets verarbeitet. Dabei wird ein Packet an seinem Header erkannt, die entsprechenden Operationen ausgeführt und nach dem Empfangen aller im Header angegebenen Datenworte auf den nächsten Header gewartet. Alle

Packet-Header laufen durch einen 64-Bit breiten Puffer, bevor sie die Verarbeitungseinheit erreichen.

3.1.3.5 Konfigurationsregister

Die Konfiguration des FPGAs wird durchgeführt, indem die verschiedenen Konfigurationsregister beschrieben werden. Die Funktionen der wichtigsten Konfigurationsregister werden in den folgenden Unterkapiteln vorgestellt.

Frame Address Register (FAR)

Die Frames zur Konfiguration des FPGA werden mit Hilfe des Frame Address Register ausgewählt. Vor einem Schreibvorgang in den Konfigurationsspeicher muss die entsprechende Adresse in das Frame Address Register geladen werden. Nach jedem geschriebenen Frame wird die Adresse im Frame Address Register automatisch um eins erhöht, damit die Adresse auf den nächsten Frame zeigt. Daraus folgt, dass bei einem Schreibvorgang nur die Startadresse des ersten Frames in das Frame Address Register geladen werden muss. Bei jeder Änderung des Frame Address Register wird erneut der im Command Register stehende Befehl ausgeführt.

Frame Data Input Register (FDRI)

Die Frames zur Konfiguration des FPGA werden mit Hilfe des Frame Data Input Register in den Konfigurationsspeicher geschrieben. Damit ein Schreibvorgang durchgeführt werden kann, muss das Command Register mit dem Befehl WCFG (Write Configuration Data) und das ID-Code-Register mit der passenden Device-ID des entsprechenden Bausteins geladen sein. Die Konfigurationsdaten werden durch einen Pufferspeicher geschoben. Während ein Frame in den Konfigurationsspeicher geschrieben wird, befindet sich bereits der nächste Frame im Pufferspeicher. Aus diesem Grund muss nach dem letzten Frame mit Konfigurationsdaten ein weiterer Frame mit beliebigen Daten folgen, um den letzten Konfigurations-Frame aus dem Pufferspeicher zu schieben. Die Konfigurationsdaten eines Frames werden parallel an die im Frame Address Register angegebene Adresse in den Konfigurationsspeicher geschrieben, nachdem die passende Anzahl Bits in das Frame Data Input Register geladen worden sind. Die Anzahl der Bits in einem Frame kann mit Hilfe des Wertes im Frame Length Register bestimmt werden.

Command Register (CMD)

Das Command Register steuert die Konfigurationslogik und führt bestimmte Operationen aus. Die Ausführung eines Befehls erfolgt direkt nach dem Beschreiben des Command Register und wird bei jedem Schreibzugriff auf das Frame Address Register erneut ausgef ührt. Eine Übersicht über die wichtigsten Befehle zeigt Tabelle 2.5.

Frame Length Register (FLR)

Die Anzahl der in einem Konfigurations-Frame enthaltenen Worte variiert für verschiedene FPGA-Größen und wird im Frame Length Register gespeichert. Zu beachten ist hierbei, dass der Wert im Frame Length Register immer um eins kleiner ist als die tatsächliche Anzahl. Für den Virtex-II mit der Bezeichnung XC2V1000, in welchem jeweils 106 Worte zu einem Frame zusammengefasst werden, muss also der Wert 105 in das Frame Length Register eingetragen werden. Dieser Vorgang muss abgeschlossen sein, bevor ein Schreibvorgang in Zusammenhang mit dem FDRI-Register stattfinden kann.

3.1.3.6 Standardformat des Bitstreams des Xilinx Virtex-II 1000

Der Bitstream zur Konfiguration eines Xilinx-Virtex-II-1000-FPGAs mit der Typenbezeichnung XC2V1000 besteht aus 224 Packets. Hierbei kann unterschieden werden zwischen Packets, die den Konfigurationsvorgang steuern und solchen, die Konfigurationsdaten zur Programmierung des FPGA enthalten. Letztere sind hier besonders interessant, da die

Kenntnisse über die Lage der Konfigurationsdaten im Bitstream zur Programmierung des FPGA wichtig sind, um später diese Konfigurationsdaten an der richtigen Stelle ausschneiden und wieder zusammenfügen zu können. Der Bitstream beginnt mit einigen Packets, die verschiedene Kontrollregister mit den passenden Werten für den Konfigurationsvorgang laden. Dazu gehören:

- Packet mit einem Dummy-Wort
- Packet mit dem Synchronisationswort
- Rücksetzen des Checksummen-Registers
- Laden des Frame Length Registers mit dem Wert 105
- Setzen des Configuration Options Registers
- Laden des ID-Code-Registers mit dem zum FPGA-Typ gehörenden Wert
- Setzen des MASK-Registers und des CTL-Registers

Nach diesen Packets folgen die Konfigurationsdaten zur Programmierung des Konfigurationsspeichers des kompletten FPGA. Dazu wird zuerst das Frame Address Register durch Schreiben des Wertes 0 auf den Anfang des Konfigurationsspeichers gesetzt (Block- Adresse = 0, Major-Adresse = 0, Minor-Adresse = 0). Das Command-Register wird mit dem Befehl WCFG (Write Configuration Data) geladen. Im nächsten Packet werden dann 117130 Worte an das Frame Data Input Register gesendet. Diese Zahl entspricht den Daten von 1104 Frames des kompletten Konfigurationsspeichers des Xilinx XC2V1000 und einem zusätzlichen Frame, um das Frame Data Input Register zu leeren. Das CRC-Wort wird in einem eigenen Packet automatisch angehängt. Danach folgen ungefähr 100 Packets mit Leerdaten, so genannte NOOP-Packets. Im Anschluss werden die Frames zwei bis vier jeder CLB-Spalte mit den endgültigen Daten geladen. Für jede vorhandene CLB-Spalte wird ein eigenes Packet definiert, das die drei Frames endgültig konfiguriert. Dazu wird erneut der Befehl WCFG in das Command-Register geschrieben. Das Frame Address Register wird auf die Adresse des zweiten Frames der ersten CLB-Spalte im Konfigurationsspeicher gesetzt (Block-Adresse = 0, Major-Adresse = 3, Minor-Adresse = 1). In das Frame Data Input Register werden nun 318 Worte geladen, was genau den Konfigurationsdaten für drei komplette Frames entspricht. Das CRC-Wort wird in einem eigenen Packet wieder automatisch angehängt. Diese Prozedur wird für alle CLB-Spalten wiederholt, indem die Major-Adresse jeweils um eins erhöht wird, bis alle CLB-Spalten abgearbeitet sind. Damit ist der Konfigurationsspeicher des FPGA vollständig geladen. Zum Abschluss des Bitstreams folgen noch einige Befehle für die Konfigurationslogik. Dazu gehören:

- Laden des Command-Registers mit dem Befehl „GRESTORE“
- Laden des Command-Registers mit dem Befehl „START“
- Laden des Checksummen-Registers mit dem Auto-CRC-Wort
- Laden des Command-Register mit dem Befehl „DESYNCH“ Danach wird der Bitstream mit vier NOOP-Packets beendet.

3.1.4 Konfiguration von Xilinx-FPGAs

Unter der Konfiguration eines FPGAs wird das Laden eines Bitstreams in den internen Konfigurations-Speicher des Bausteins verstanden. Das Beschreiben dieses Speichers legt die Logikfunktionen des FPGAs fest. Intern werden die Informationen in einem SRAM abgelegt. Da es sich hierbei um einen flüchtigen Speicher handelt, muss der FPGA nach dem

Einschalten zuerst programmiert werden, bevor er in einer Schaltung eingesetzt werden kann. Abhängig von einigen Einstellungen und der Konfigurationsmethode, lässt sich der FPGA nur einmal direkt nach dem Einschalten der Stromversorgung konfigurieren oder jederzeit auch während des Betriebs. Die FPGAs der Spartan-II Familie von Xilinx können auf vier verschiedene Arten programmiert werden: Master Serial, Slave Serial, Slave Parallel und Boundary-Scan [CaGo02]. Zusätzlich kann ausgewählt werden, wie sich die I/O-Pins während der Programmierung verhalten sollen. Diese können entweder durch interne Pull-Up Widerstände auf ein logisches Eins festgelegt werden oder hochohmig sein. Der gewünschte Konfigurationsmodus wird durch das Setzen der Mode-Pins des FPGAs festgelegt. Welche Einstellungen an den Mode-Pins vorgenommen werden müssen, um den gewünschten Konfigurationsmodus auszuwählen ist in Tabelle 3.4 dargestellt.

Konfigurations-Modus	M2	M1	M0
Master Serial	0	0	0
Slave Serial	1	1	1
Slave Parallel	1	1	0
Boundary-Scan	1	0	1
Master Serial (mit Pull-Ups)	1	0	0
Slave Serial (mit Pull-Ups)	0	1	1
Slave Parallel (mit Pull-Ups)	0	1	0
Boundary-Scan (mit Pull-Ups)	0	0	1

Tabelle 3.4: Spartan-II Konfigurationsmodi

Die folgenden Kapitel beschreiben die verschiedenen Methoden, die zum Programmieren von Xilinx FPGAs eingesetzt werden können.

3.1.4.1 Boundary-Scan

Die (IEEE 1149.1) ”Test Access Port and Boundary-Scan Architecture“, bekannt unter dem Namen JTAG, ist die am häufigsten eingesetzte Methode zum Testen von Halbleiterbauelementen auf einer Platine. JTAG steht für Joint Test Action Group, dem Namen des Komitees, das ursprünglich für die Entwicklung des Standards zuständig war. Durch den vermehrten Einsatz von SMD-Bauteilen entwickelte sich Boundary-Scan zu einem wichtigen Standard bei der Fehlersuche auf modernen Multilayer-Platinen. Mit Hilfe von Boundary-Scan lassen sich problemlos einzelne oder mehrere miteinander verkettete ICs auf einer Platine testen. Eine Erweiterung des Standards und die Möglichkeit der Ergänzung um firmenspezifische Befehle verbesserte die Möglichkeiten zum Testen, Konfigurieren und Verifizieren nochmals. Dies ermöglichte den Einsatz von Boundary-Scan zur Konfiguration von FPGAs ([Xil02] und [Xil05a]). Die FPGAs der Xilinx Spartan-II- und Virtex-II-Serie sind vollständig kompatibel zum IEEE-1149.1-Standard. Sie sind ausgerüstet mit Test-Access-Port (TAP), TAP-Controller, Instruction-Register, Instruction-Decoder, Boundary-Scan Register und einem Bypass-Register. Der TAP ist die genormte Schnittstelle auf der Seite des zu testenden Bausteins. Das Verhalten des TAPs wird durch den Zustandsautomaten im TAP-Controller festgelegt.

Pin	Name	Beschreibung
TDI	Test Data In	serieller Eingang
TDO	Test Data Out	serieller Ausgang
TMS	Test Mode Select	steuert den Zustands-Automat im TAP-Controller
TCK	Test Clock	Takteingang für den Scan-Port

Tabelle 3.5: Xilinx TAP-Controller Pins

Die Verschaltung mehrerer FPGAs zur Programmierung im Boundary-Scan-Mode sind in Abbildung 3.7 zu sehen. Alle FPGAs werden in Reihe hintereinander angeschlossen. Bei der Programmierung wird der gewünschte Baustein ausgewählt und die Register der vorgeschalteten Bausteine leiten die Daten direkt an den nachfolgenden Baustein weiter.

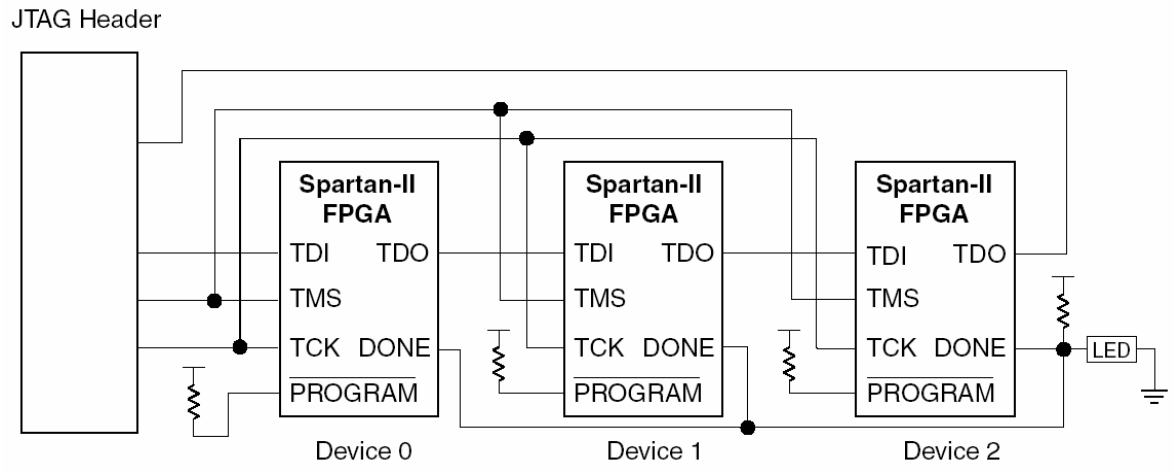

Abbildung 3.7: Schaltplan Boundary-Scan Mode

Der Zustand des TAP-Controllers wird über den Eingang TMS festgelegt. Dazu wird der Wert des Eingangs bei der steigenden Flanke des TCK-Signals ausgewertet und durch den Controller verarbeitet. Die Eingabe von Daten erfolgt über den Eingang TDI, dessen Wert ebenfalls bei der Vorderflanke des Taktsignals übernommen wird. In Abhängigkeit des Zustands des TAP-Controllers und des internen Instruction-Registers können die eingelesenen Daten in verschiedenen Registern abgelegt werden.

Registername	Größe	Beschreibung
Instruction Register	5 Bit	Speichert den OPCODE des momentanen Befehls und liest den aktuellen Zustand des Bausteins ein
Boundary-Scan Register	3 Bit	Pro I/O liest oder schreibt den Eingang, Ausgang oder das Freigabesignal des Ausgangs
Identification Register	32 Bit	Enthält die Bauteile ID sowie den Xilinx Manufacturing Code.
JTAG Configuration Register	32 Bit	Dieses Register ermöglicht den Zugang zum Xilinx Configuration Bus, wenn sich die Befehle CFG IN oder CFG OUT im Instruction Register befinden
Usercode Register	32 Bit	Ermöglicht das Abspeichern eines Benutzerprogrammierbaren Codes

Tabelle 3.6: Xilinx Spartan-II JTAG-Register

Die verfügbaren Register in der Spartan-II Serie sind in Tabelle 3.6 aufgelistet. Der TDO-Anschluss ist der serielle Ausgang des Bauteils und gibt Daten ebenfalls in Abhängigkeit des Instruction-Registers und des TAP-Controllers aus einem der folgenden Register an das JTAG-Programmiergerät zurück.

Um mittels JTAG-Schnittstelle einen bestimmten Befehl auf einem Xilinx-FPGA auszuführen und ein Datenregister auszuwählen, kann das Instruction-Register über den seriellen Eingang beschrieben werden. Der Befehlssatz enthält nicht nur die (IEEE 1149.1) Standard-

Befehle, sondern auch den Xilinx spezifischen erweiterten Befehlssatz. Die Bedeutung der Befehle des 5 Bit breiten Registers ist der folgenden Tabelle 3.7 zu entnehmen.

Boundary-Scan Befehle	Binär Code (4:0)	Beschreibung
EXTEST	00000	Boundary-Scan EXTEST Befehl
SAMPLE	00001	Boundary-Scan SAMPLE Befehl
USER1	00010	Zugriff auf das User Register 1
USER2	00011	Zugriff auf das User Register 2
CFG OUT	00100	Zugriff auf den Configuration-Bus zum Auslesen des FPGAs
CFG IN	00101	Zugriff auf den Configuration-Bus zum Programmieren des FPGAs
INTEST	00111	Boundary-Scan INTEST Befehl
USERCODE	01000	Aktiviert serielle Ausgabe des User Codes
IDCODE	01001	Aktiviert serielle Ausgabe des ID Codes
HIGHZ	01010	Aktiviert das Bypass Register und hält alle I/Os in hoch-ohmigem Zustand
JSTART	01100	Verwendet TCK als Taktsignal bei der Programmierung des FPGAs
BYPASS	11111	Aktiviert das Bypass Register
RESERVED	alle anderen	reservierte Xilinx Befehle

Tabelle 3.7: Xilinx Spartan-II Befehlssatz

Die genaue Funktion der IEEE-1149.1-Standardbefehle können in [MaTu93] nachgelesen werden.

3.1.4.2 Parallel Modes

Die Programmierung des FPGAs im Slave-Parallel-Mode erfolgt über einen bidirektionalen Datenbus. Im Slave-Mode wird das Taktsignal CCLK von einer externen Quelle, beispielsweise einem Mikrocontroller erzeugt und der FPGA damit gespeist ([Xil02c] und [Xil04f]). Die Daten werden über den 8 Bit breiten Datenbus übertragen. Eine Programmierung im Master-Parallel-Mode, der die Erzeugung des Taktes durch den FPGA vorsieht, wird z.B. von der Spartan-II-Familie nicht unterstützt.

In Abbildung 3.8 ist die Beschaltung für mehrere FPGAs im Parallel-Mode zu sehen. Die Anschlüsse (D0..D7) stellen den 8 Bit breiten Datenbus dar, dabei ist D0 das LSB. Das WRITE Signal legt fest, ob gelesen oder geschrieben wird. Über den parallelen Datenbus lassen sich mehrere FPGAs anbinden. Die Auswahl des zu programmierenden FPGAs erfolgt über die Leitung CS. Wenn mehrere FPGAs den selben Bitstream erhalten sollen, können die #CS-Leitungen der entsprechenden Bausteine verbunden werden. Die BUSY Leitung signalisiert dem schreibenden Gerät die Übernahme der Daten durch den FPGA. Sobald BUSY auf Masse liegt, werden die anliegenden Daten bei der nächsten Forderflanke des Taktes übernommen, falls das schreibende Gerät #CS und #WRITE auf Masse zieht. Ist der FPGA nicht bereit die Daten aufzunehmen, wird dies durch einen High-Pegel an der BUSY-Leitung angezeigt. In diesem Falle müssen die Daten beim nächsten Taktzyklus wiederholt werden. Diese Leitung ist allerdings nur bei CCLK-Frequenzen über 50MHz notwendig. Bei niedrigeren Frequenzen muss die Leitung nicht verbunden werden. Bei der parallelen Programmierung mehrerer FPGAs dürfen nur Taktfrequenzen unterhalb von 50MHz verwendet werden. Das CCLK-Taktsignal kann entweder von einem externen Oszillator erzeugt oder z.B. von einem Mikrocontroller generiert werden. Dies ist dann sinnvoll, wenn das WRITE-Signal und der Datenbus nicht nur für das Programmieren des FPGAs, sondern auch für

den Zugriff auf andere Bauteile, wie beispielsweise externe Speicher, verwendet wird. In diesem Falle wird der Takt einfach unterbrochen und somit die Programmierung des FPGAs so lange ausgesetzt, bis der Bus wieder zur Verfügung steht und der FPGA weiter programmiert werden soll.

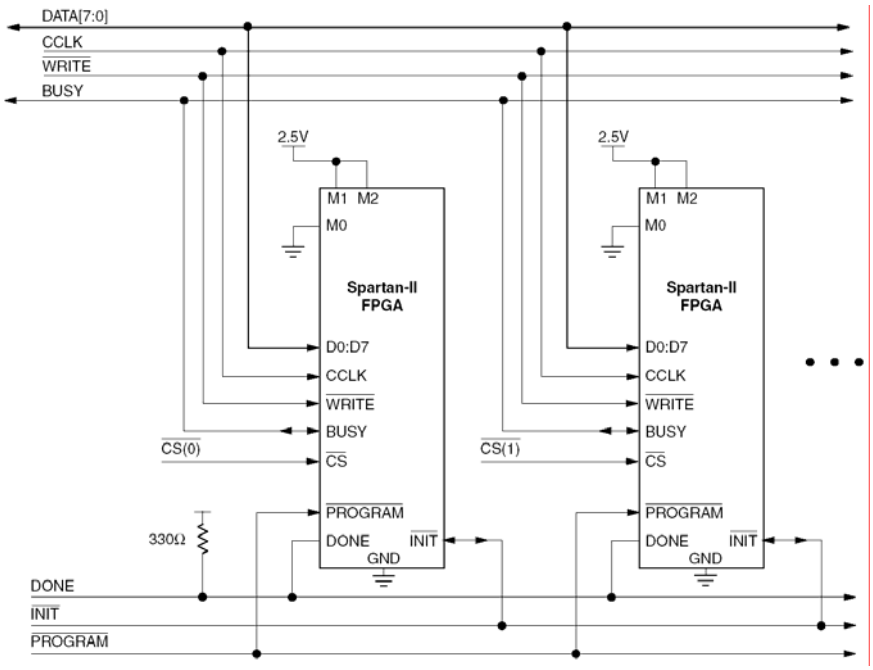

Abbildung 3.8: Schaltplan Slave Parallel Mode

3.1.4.3 Serial-Modes

Die beiden Modi ”Master-Serial“ und ”Slave-Serial“ unterscheiden sich nur in der Art ihrer Taktung. Im Master Serial-Mode wird der CCLK-Pin vom FPGA getrieben, im Slave-Serial-Mode wird der Pin von einem externen Taktgeber angesteuert. Über den Anschluss DIN werden die Daten auf den FPGA übertragen. Pro Taktflanke wird jeweils 1 Bit übertragen. Hierbei wird das vorliegende Datenbyte bitweise aufgeteilt und vom MSB zum LSB seriell in den Konfigurationseingang DIN des zu programmierenden FPGAs übertragen

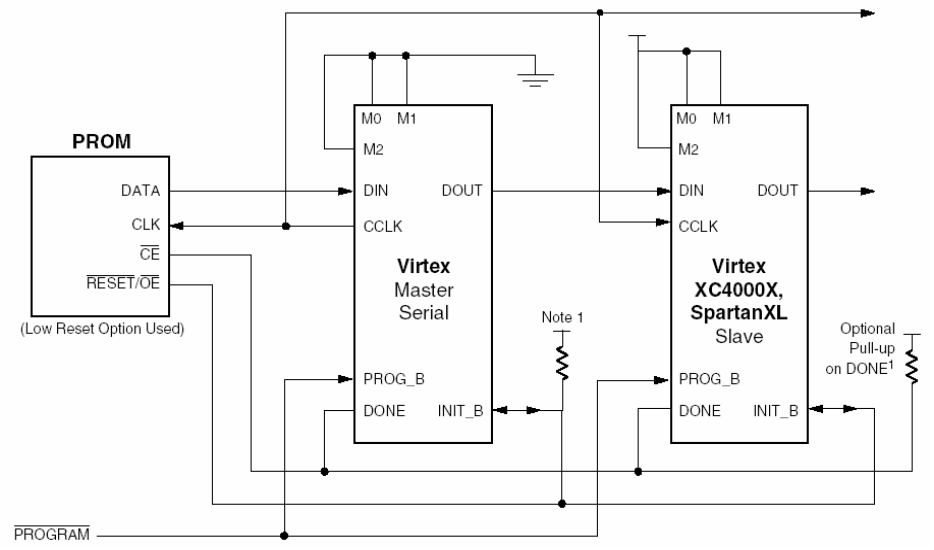

Abbildung 3.9: Master/Slave Serial Mode

Abbildung 3.9 zeigt die Programmierung mehrerer FPGAs über ein serielles EEPROM. Die verketteten FPGAs werden alle aus einem seriellen Speicher programmiert. Der abgebildete Spartan-II läuft hier als serieller Master und programmiert zuerst sich selbst und danach den angeschlossenen seriellen Slave (siehe auch [Xil02b]).

3.1.5 FPAA (Free Programmable Analog Array)

Ein FPAA (Field Programmable Analog Array) ist ein frei programmierbarer analoger Schaltkreis, konstruiert nach dem Vorbild der FPGAs. Ein FPAA besteht aus einer Matrix von konfigurierbaren, analogen Blöcken. Ein solcher Block wird CAB (Configurable Analog Block) genannt. Mit einem CAB lassen sich verschiedene analoge Grundschaltungen wie Filter oder Verstärker realisieren. Eine logische Schaltung dient zum Herstellen der notwendigen Verbindungen innerhalb und zwischen den CABs. Analoge Ein- und Ausgänge stellen die Verbindung zur Außenwelt her. Die Konfiguration erfolgt über digitale Ein- und Ausgänge, die an Speicherbausteine oder Mikroprozessoren angeschlossen werden können.

3.1.5.1 Aufbau und Funktion

Das Kernstück eines FPAA bildet die Matrix aus programmierbaren analogen Schaltungen. Hierin befinden sich zwischen 2 und 20 CABs. Diese CABs bestehen aus einem oder mehreren Operationsverstärkern und passiven Bauelementen, die über elektronische Schalter miteinander verknüpft werden können. Somit ist es möglich, eine Vielzahl einfacher analoger Schaltungen zu realisieren. Bei der Implementierung der zuschaltbaren Bauteile kommen zwei Prinzipien zur Anwendung: Die Switched-Capacitor-Technik und die Verwendung herkömmlicher Bauteile (Widerstände und Kondensatoren).

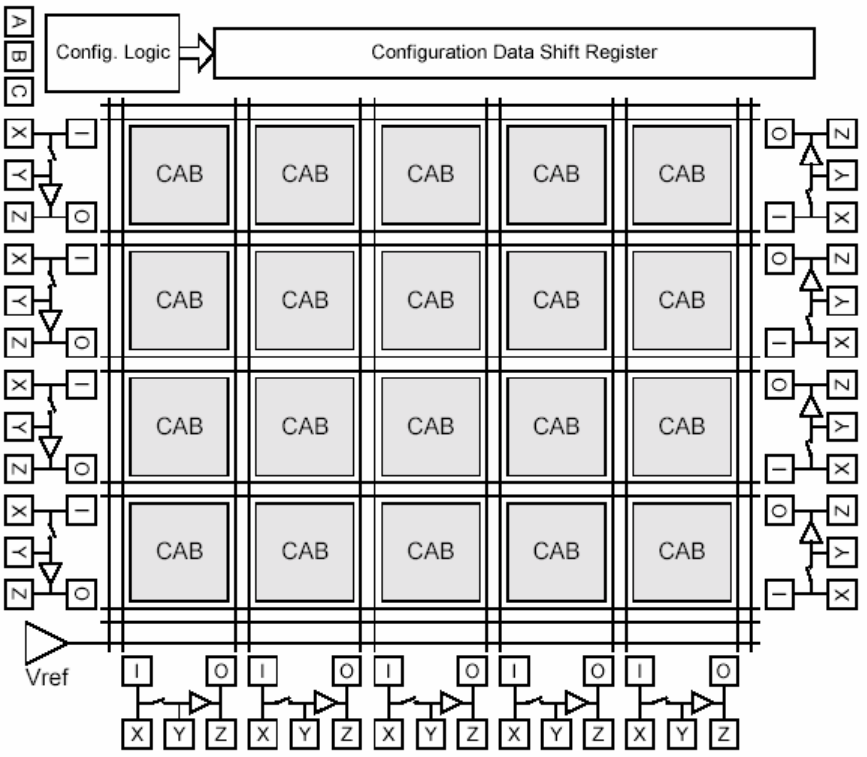

Abbildung 3.10: Innerer Aufbau eines FPAAs von Anadigm

Mit dem Switched Capacitor (SC) lassen sich sowohl Kapazitäten als auch Widerstände erzeugen. Ein Widerstand wird erzeugt, indem ein Kondensator mit hoher Frequenz zunächst zu dem negativen und zum positiven Pol des Signalwegs dazugeschaltet wird, er sich abhängig von der aktuellen Signalspannung, der Schaltfrequenz und seiner Kapazität abwechselnd teilweise lädt und entlädt. Der Vorteil dieser Technik liegt in ihrer hohen Flexibili-

tät und Stabilität. Nachteile sind die geringe Bandbreite in Abhängigkeit von der Schaltfrequenz (s. Nyquist-Theorem) und die parasitären Effekte, die ebenfalls durch die Schaltfrequenz bedingt sind.

Eine Alternative stellt die Verwendung herkömmlicher Bauelemente in Form von Einheitswiderständen und -kapazitäten dar. Je nach beabsichtigtem Verwendungszweck des FPAA sind diese teilweise fest verdrahtet. Dies ermöglicht eine höhere Bandbreite und vermeidet die Artefakte, wie sie bei abgetasteten Systemen auftreten. Ein Nachteil ist die geringere Flexibilität dieser Schaltungen.

Zusätzlich enthalten die konfigurierbaren analogen Blöcke einen digitalen Arbeitsspeicher (RAM), in den die erwünschte Konfiguration geladen wird. Einige FPAAs von Anadigm besitzen zusätzliche "Shadow-RAMs", die es ermöglichen, neue Konfigurationsdaten während des Betriebs ohne Unterbrechung des analogen Signals zu laden.

Die Analoge Matrix wird von einer Reihe an Hilfsschaltungen umgeben. Notwendig ist eine logische Schaltung, welche die Konfigurationsdaten verarbeitet und für die Speicherung an der richtigen Stelle sorgt. Ein interner digitaler Zeitgeber koordiniert das Laden der Konfiguration. Häufig stehen konfigurierbare Ein- und Ausgabezellen zur Verfügung, die z. B. mit zuschaltbaren Anti-Aliasing Filtern oder Verstärkern ausgestattet sind. Werden Switched-Capacitors genutzt, so werden analoge Zeitgeber benötigt.

3.1.5.2 Programmierung

Die Programmierung der FPAAs geschieht über ein herstellerspezifisches Protokoll, abhängig von der Architektur des Chips. Die Hersteller stellen für ihre Produkte eigene Entwicklungssoftware zur Verfügung, mit Hilfe derer die gewünschte Schaltung entwickelt und meist auch simuliert werden kann. Die resultierende Konfiguration wird dann in Form eines Bitstreams in den Chip hochgeladen.

3.1.5.3 Zukunft der FPAAs

Die Zukunft der FPAAs ist ähnlich gut, wie die der FPGAs, allerdings mit einem wesentlichen Unterschied: die Technologie ist mindestens 5 Jahre hinter der des FPGAs. Somit muss man noch mit fehlenden Standardisierungen und „Kinderkrankheiten“ rechnen. Die Hersteller geben sich naturgemäß sehr überzeugend hinsichtlich ihrer Bausteine und deren Reifegrad. Jedoch zeigen diverse Versuche und Befragungen von Firmen, die diese Technologie getestet haben, dass die Funktionsgestaltung und die Qualität der Signale bei den derzeitigen Devices leider nur sehr unzureichend sind.

Für den Einsatz in ein Produkt – und gerade in das in dieser Arbeit entstandene RP-HiL-System, das auf re-programmierbarer Technologie basiert – ist der FPAA hochinteressant. Dies gilt nicht nur für den Übergang Analog-Digital, sondern auch für mögliche Stufen im Bereich der Signalkonditionierung. Als Signalkonditionierer oder für kleine analoge Netze sind die aktuellen Produkte bereits gut verwendbar. Sobald es jedoch um komplexere elektronische Schaltungen geht, stößt die aktuelle FPAA-Technologie schnell an ihre Grenzen.

Im Folgenden sind ein paar Links zu einzelnen Herstellern angegeben. Ein gesamtes Herstellerverzeichnis ist bei dem aktuell sehr schwer überschaubaren Markt und den vielen Produktan- und -abkündigungen nicht sinnvoll.

- Zetex TRAC (Totally Reconfigurable Analog Circuit), http://www.zetex.com
- Lattice ispPAC, http://www.latticesemi.com
- Anadigm (früher Anadyne), http://www.anadigm.com
- Cypress MicroSystems PsoC, http://www.cypressmicro.com
- ALD (Advanced Linear Devices), http://www.aldinc.com

3.2 Sensoren und Aktoren

Mit zunehmendem Komfort der heutigen Kraftfahrzeuge wachsen auch Anzahl und Komplexität an Sensoren und Aktoren. Derzeit sind in einem modernen Kraftfahrzeug über 70 Sensoren und mehr als 100 Aktoren [Lins98] enthalten. Die Sensoren dienen der Erfassung des aktuellen Systemzustandes bzw. des relevanten Teils der Zustandsgrößen, die für eine aktive Beeinflussung der Fahrzeugeigenschaften, wie z.B. der Berechnung der optimalen Einspritzmenge im Motorsteuergerät oder zum Auslösen eines Warntons bei aktueller Sitzbelegung ohne eingeklickten Sicherheitsgurt, benötigt werden. Sensoren sind die primären Elemente in Mess- und Regelkreisen, die Änderungen verschiedener physikalischer Größen in der Umwelt in elektrische Signale umwandeln [Baue01]. Aktoren hingegen setzen elektrische Signale in Kräfte, Wege oder thermische Energie um, mit denen eine Zustandsänderung der Umwelt erfolgt. Hierzu zählen beispielsweise der elektromagnetische Steller, der die berechnete Einspritzmenge bereitstellt oder ein Elektromotor, der beim Betätigen der Taste die Fensterscheibe bewegt.

Für beide Gruppen, Sensoren wie Aktoren, gibt es nun ein weites Spektrum an physikalischen oder chemischen Größen, aus denen oder für die elektrische Signale generiert werden müssen. Zur Einordnung oder Unterscheidung der einzelnen Komponenten ist es sinnvoll, eine geeignete Klassifizierung vorzunehmen. Dazu lassen sich verschiedene Kriterien ableiten, nach denen eine solche Aufstellung erfolgen kann:

- Zu messende physikalische Größe
- Wirkprinzip/Art der Signalwandlung
- Basismaterial
- Verwendungszweck/Einsatzgebiet
- Elektrische Schnittstelle

Letzteres ist das Entscheidende für diese Arbeit und soll damit als Grundlage der weiteren Betrachtung dienen.

3.2.1 Klassifizierung von Sensorschnittstellen

Aufgrund der Ausrichtung der hier vorgestellten Arbeit, wird die Unterteilung der Sensoren anhand ihrer elektrischen Schnittstellen vorgenommen. Gemessen werden typischer Weise Temperatur, Druck, Feuchte, Drehzahl, Gasgemisch, Füllstand und Beschleunigung – um nur einige der wichtigsten Größen zu nennen. Zwischen der Messung der physikalischen Größe bzw. der Erfassung des Analogsignals und der Verarbeitung des Messwertes in einem Steuergerät sind mehrere Transformations- bzw. Verarbeitungsschritte nötig, die an verschiedenen Stellen im Fahrzeug oder einer vernetzten Steuerungsanlage vorgenommen werden können.

Abbildung 3.11 zeigt, dass die Schnittstellen der Sensoren je nach Komplexität der integrierten Verarbeitungsschritte variieren. Soll ein Sensor nun an ein Steuergerät angeschlossen oder derselbe auf korrekte Funktion getestet werden, besteht mit steigender „Intelligenz“ des Sensors eher die Notwenigkeit der Anpassung der Testeinrichtung an eine digitale Schnittstelle, ein Bussystem oder Bus-Protokoll als an einen analogen Ausgang.

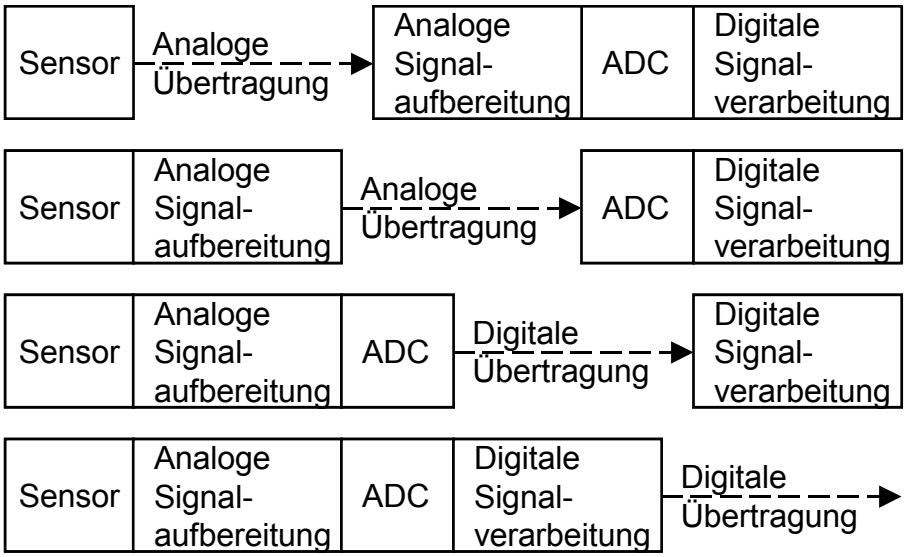

Abbildung 3.11: Klassifizierung von Sensorschnittstellen

3.2.2 Klassifizierung von Aktorschnittstellen

Ähnlich wie bei den Sensoren sieht die Unterteilung bei den Aktoren aus, jedoch werden hier keine Werte eingelesen, sondern elektrische Ströme und Spannungen in andere physikalische Größen umgewandelt. Die Art der Wandlung kann aber auch hier in verschiedene Kategorien gegliedert werden, je nach dem wie hoch der Anteil der Vorverarbeitung und Signalanpassung innerhalb des Gesamtaktors ist. Einsatzgebiete finden sich in [Iser99], [Bosc01] und [Bosc02].

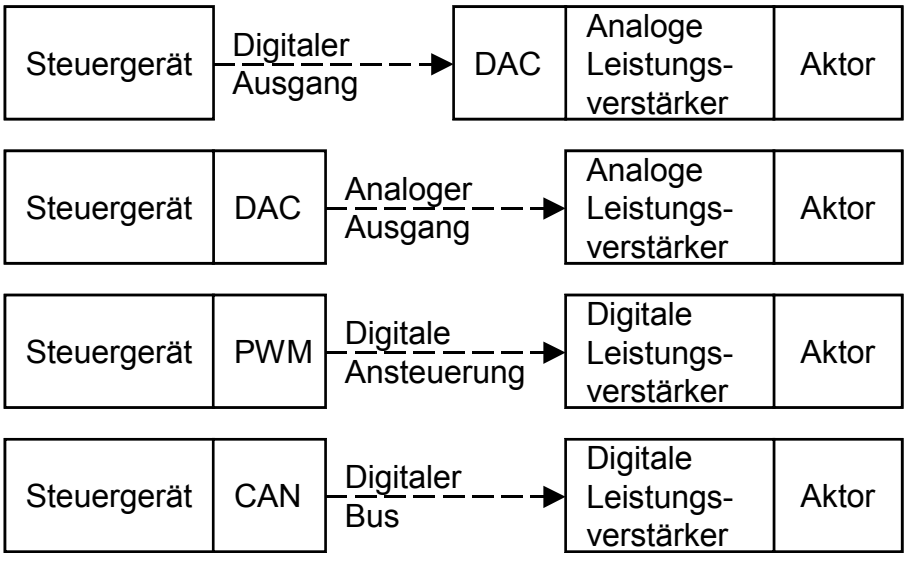

Abbildung 3.12: Klassifizierung von Aktorschnittstellen

3.3 Digitale Schnittstellen

3.3.1 Parallele I/O-Ports von Mikrocontrollern und FPGAs

Die heutigen Mikrocontroller verfügen über eine recht große Anzahl von parallelen Schnittstellen, die durch Programmierung – auch zur Laufzeit – als Ein- oder Ausgabe-Ports verwendet werden können. Dabei löst sich die Parallelität dahingehend auf, da diese oft einzeln umgeschaltet werden können und darüber hinaus mehrere alternative Funktionen auf einem einzigen Pin hinterlegt sind, die nur jeweils einzeln aktiv sein können. So ist es möglich, dass ein Pin neben der Funktion als Standard-Digital-I/O auch einen PWM-Ausgang (PWM = Pulsweitenmodulation), einen seriellen Schnittstelleneingang oder alternativ einen A/D-Wandlereingang bietet. Die für alle Funktionen gültigen Spannungslevel hängen von der allgemeinen Betriebsspannung des Mikrocontrollers ab.

Anders ist das bei den I/O-Ports eines FPGAs. Hier können die Port-Pins – zumindest bankweise – unterschiedliche Spannungslevel besitzen. Diese sind auch grundsätzlich unabhängig von der Spannungsversorgung der internen Logik, die meist um einiges niedriger gehalten ist (1,2V, 1,5V, 2,5V oder maximal 3,3V – je nach Hersteller und Baustein). In den Datenblättern finden sich neben den möglichen Bankversorgungsspannungen auch die pro Pin einstellbaren Treiberstärken und I/O-Standards, wie z.B. PCI, LVDS_25 oder LVDS_33 [Xil01], [Xil04] oder [Xil05c]. Bei den genannten FPGAs sind außerdem keine vordefinierten Funktionen (SPI, USART, PWM, CaptureCompare, ...) pro Pin hinterlegt, außer solchen zur Programmierung des Bausteins als serielle oder parallele Schnittstellen. Andere Funktionen sind vom Benutzer selbst zu implementieren.

3.3.2 Serielle Schnittstellen und Protokolle

Zur Ansteuerung peripherer Bausteine wie A/D- oder D/A-Wandler, externer CAN-Controller, EEPROM-Speicher werden im Bereich von wenigen Zentimetern auf der Platine vorrangig serielle Busse eingesetzt. Sie bieten den Vorteil einer standardisierten Schnittstelle (SPI [Phil03], I^2C [I2CB00], OneWire [1Wir02], usw.) und kommen mit wenigen Leitungen aus, meist im Bereich von drei oder vier. Über diese wird dann die gesamte Kommunikation abgewickelt, wobei je nach Medium auch mehrere Bausteine an einen Bus angeschlossen werden können, ohne zusätzliche Leitungen notwenig zu machen. Ein weiterer wichtiger Aspekt bei der Verwendung serieller Busse ist die maximale Frequenz, mit der dieser betrieben wird. Zum einen ist in diesen Fällen die Takterzeugung nicht ganz einfach, zum anderen können, je nach Steilheit der Flanken, erhebliche EMV-Probleme auftreten. Die gängigsten seriellen Protokolle sind hier kurz zusammengestellt.

3.3.2.1 I^2C (auch IIC oder I2C)

Die Abkürzung I^2C-Bus steht für Inter-IC-Bus. Spezifikation des Busses und erste verfügbare Bauteile stammen von der Firma Philips [I2CB00]. Mittlerweile gibt es aber von fast allen Herstellern Bauteile mit diesem Bus-Interface. Die Signalleitungen beschränken sich auf ein Taktsignal SCL und eine Datenleitung SCA. Zusätzlich müssen für das Businterface noch Masse und eine Versorgungsspannung anliegen, zum Betrieb des eigentlichen Bausteins sind diese aber ohnehin vorhanden. Der Bus ist als Multi-Master-Bus ausgeführt und kann somit über mehrere Bauteile verfügen, die den Bus aktiv kontrollieren. Die Arbitrierung des Busses geschieht dabei über die SCA-Leitung. Die Adressierung des Slaves erfolgt über eine 7 Bit breite Adresse, zuzüglich eines Bits zur Festlegung der Datenrichtung „read“ oder „write“. Eingeschlossen werden die Übertragungen immer von einem Start- und einem Stop-Signal, das vom Master generiert wird. Der Bus erreicht Geschwindigkeiten bis zu 100 kbit/s im Standard-Modus, 400 kbit/s im Fast-Modus oder bis zu 3.4 Mbit/s im High-Speed Modus.

3.3.2.2 SPI

Das Serial-Peripherial-Interface (SPI) wurde von Motorola eingeführt, um einen Prozessor mit Peripheriebausteinen zu verbinden. Die Verbindung benötigt zwei Steuer- und zwei Da-

tenleitungen. Der SPI-Bus ist als Single-Master Multi-Slave-System aufgebaut. Die Auswahl der Teilnehmer sowie die Erzeugung des Taktes übernimmt der Master. In der Standard-Busstruktur werden die Leitungen SCKL (Serial Clock), SDI (Serial Data In) und SDO (Serial Data Out) an allen Bausteinen angeschlossen und jeweils eine CS-Leitung vom Master zu jedem Slave verlegt. Der Master wählt immer den Baustein aus, mit dem er kommunizieren möchte. Die Daten werden seriell über die Leitung SDI in den Baustein übertragen. Dort werden sie in ein Schieberegister übertragen, das intern beliebig weiter verbunden werden kann. Die Länge des Schieberegisters ist dabei nicht festgelegt: Sie orientiert sich an der entsprechenden Datenbreite der verwendeten Bauteile, beispielsweise an der Auflösung eines Wandlers oder der Breite eines Speichers. Zur Aktivierung des SPI-Bausteins wird zunächst die low-aktive #CS-Leitung gegen Masse gezogen, danach erst erfolgt die serielle Übertragung. Die auf der Eingangsleitung angelegten Datenbits werden immer zur steigenden Taktflanke des SCLK übernommen.

3.3.2.3 MicroWire

Microwire [Micr92] wurde von National Semiconductor entwickelt. Es handelt sich um ein 4-Draht-Interface wie bei SPI. Die Datenübertragung erfolgt immer nur zwischen zwei Busteilnehmern. Eine Adressvergabe, wie bei I^2C findet nicht statt, stattdessen wird der Teilnehmer per Chip-Select Leitung ausgewählt. Übertragungen zwischen Microwire- und SPI-Geräten sind prinzipiell möglich, wenn die maximalen Datenraten eingehalten werden.

3.3.2.4 OneWire (auch 1-Wire)

Das OneWire-Interface [1Wir02] von Dallas-Semiconductor ist ein System zur Daten- und Energieübertragung über eine einzige Leitung und eine gemeinsame Masseverbindung. Die Übertragung findet zwischen einen Master und mehreren möglichen Slaves statt. Auf der Datenleitung findet eine bidirektionale Halb-Duplex-Kommunikation statt. Die bei der Datenübertragung verwendete Energie lädt im Slave einen internen Kondensator auf, der als Stromversorgung für die interne Gerätefunktion dient. Die Adressierung erfolgt über eine einmalig vergebene 64 Bit breite Adresse. Eine Änderung der Adresse ist nicht möglich, wodurch jedes OneWire-Gerät eindeutig identifizierbar wird.

3.4 Signalkonditionierung

Der Betrieb einer elektronischen Steuerung – wie z.B. eines eingebetteten Systems – in einer Automatisierungsanlage bringt den Umstand mit sich, dass die Ein- und Ausgänge der Steuerung mit den Aus- und Eingängen des Prozesses verbunden werden müssen. In den wenigsten Fällen können dazu beide Domänen direkt gekoppelt werden, da z.B. die Spannungslevel auf beiden Seiten unterschiedlich sind. Daher muss eine entsprechende Anpassung, sprich Signalkonditionierung, vorgenommen werden, um eine einwandfreie Funktion sicherzustellen und Zerstörungen auf der einen oder anderen Seite zu vermeiden. Dies kann im einfachsten Fall eine Spannungsanpassung durch einen Spannungsteiler sein, der den Ausgangspegel eines Prozessgliedes auf den Eingangslevel der Steuerung reduziert. Da die gesteuerten Prozessteile meist Maschinen sind, bedarf es entsprechender Leistung zur Ansteuerung. Die nötige Konditionierung findet daher durch Dazwischenschalten eines entsprechenden Treibers statt. Allerdings kann der Bereich der Anpassung auch eventuelle Wandlungen zwischen verschiedenen physikalischen Größen untereinander oder in äquivalente digitale Werte unter Verwendung von Analog-Digital- bzw. Digital-Analog-Wandlern, beinhalten.

Nicht selten ist neben der Signalkonditionierung auch eine galvanische Trennung vorzunehmen. Dabei werden Steuerung und Prozess aus Sicherheitsgründen aus unterschiedlichen Quellen mit Energie versorgt. Um zu verhindern, dass eine Kopplung über die Signalwege stattfindet, werden diese durch entsprechende Glieder getrennt. Dies kann bei der Kopplung von Signalen mit einem hohen Gleichspannungs-/Gleichstromanteil durch Opto-

Koppler realisiert werden. Zur galvanischen Trennung von Wechselspannungs-/Wechselstromsignalen bieten sich Koppelkondensatoren oder induktive Übertrager an.

3.5 Takterzeugung und Verteilung

3.5.1 Takterzeugung

Die Bereitstellung eines Taktes zum Betrieb digitaler Schaltungen kann auf vielen verschiedenen Wegen geschehen. Eine Unterteilung soll hier anhand ihrer Veränderbarkeit während der Laufzeit geschehen. Somit bilden sich zwei Gruppen aus, die entweder einen rein statischen Takt über die gesamte Laufzeit generieren oder eine dynamische Anpassung an die Erfordernisse erlauben. Für den ersten Fall werden im Allgemeinen Quarze eingesetzt, die aber noch einen Inverter oder Schmitt-Trigger benötigen, oder bereits in einem Gehäuse integrierte Oszillatoren. Hier muss nur eine Spannung angelegt werden, wonach das Ausgangssignal als fest eingestellter Takt abgegriffen werden kann.

Die Verwendung verschiedener zur Laufzeit veränderbarer Frequenzen erfordert dahingegen einen speziellen, programmierbaren Taktgenerator (z.B. [PCK12429], siehe Abbildung 3.13). Dieser kann durch externe Ansteuerung die Ausgangsfrequenz in einem bestimmten Bereich verändern. Zur Anregung ist ein externes Quarz vorhanden. Diese Referenzfrequenz wird über einen fest eingestellten Teiler auf die Grundfrequenz verringert. Die eigentliche Frequenzsynthese erfolgt in einem PLL (Phase-Locked-Loop).

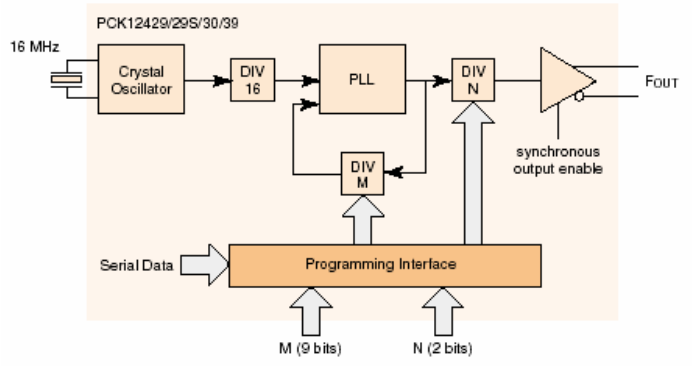

Abbildung 3.13: Programmierbarer Taktgenerator [PCK12429]

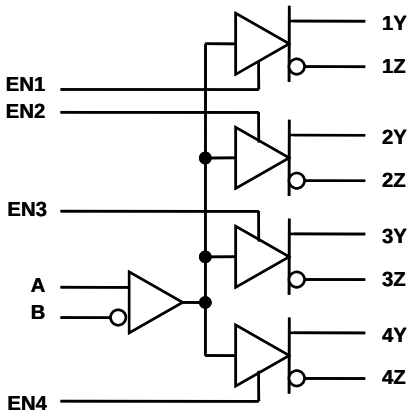

Abbildung 3.14: Taktverteiler (Repeater) für 4 Netzabschnitte

3.5.2 Taktverteilung

Die Verteilung eines Taktsignals, das für mehrere Bauteile oder Platinen verwendet werden soll, benötigt einen so genannten Repeater. Dieser ermöglicht neben der Verstärkung auch die Distribution des Taktsignals auf mehrere Empfänger unter Verwendung einer differentiellen Übertragung. In Abbildung 3.14 ist ein solcher Baustein mit vier Ausgängen dargestellt, der außerdem eine Funktion zum Ein- und Ausschalten der einzelnen Signale zur Verfügung stellt. An den differentiellen Ausgangspaaren ist jeweils ein Verstärker vorhanden, der den Pegel auf den spezifizierten Spannungsbereich anhebt.

3.5.3 LVDS-Übertragung

Low-Voltage Differential Signaling (LVDS) wurde im IEEE-Standard für “Low-Voltage Differential Signals (LVDS) for Scalable Coherent Interface (SCI)” standardisiert (IEEE Stan-

dard 1596.3). LVDS hat sich inzwischen zum führenden Standard in der Hochgeschwindigkeits-Vernetzung von ICs, Platinen und Geräten entwickelt. Aufgrund seiner Eigenschaften bietet sich LVDS zur Übertragung der Taktsignale innerhalb der Plattform an. Abbildung 3.15 zeigt den schematischen Aufbau einer LVDS-Übertragungsstrecke. Entspricht der Line-Driver nicht dem LVDS-Standard (z.B. PECL, Positive/Pseudo Emitter Coupled Logic), muss eine Anpassung vorgenommen werden, um die entsprechenden Spannungslevel zu erreichen.

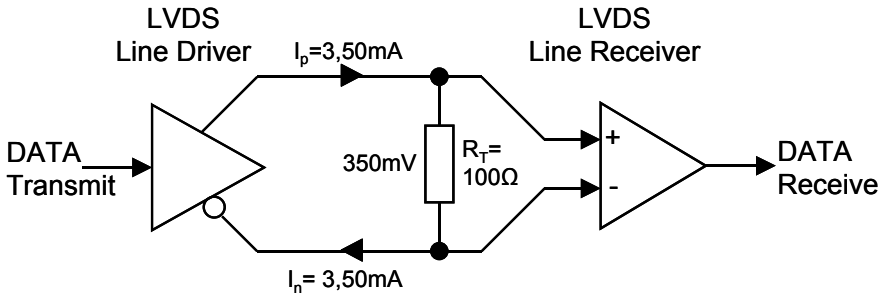

Abbildung 3.15: Schema einer LVDS-Übertragungsstrecke

Viele aktuelle Hardwarekomponenten werden jedoch bereits mit LVDS-Schnittstellen ausgeliefert, so dass hier keine zusätzlichen Anpassungen nötig sind (z.B. LVDS-Treiber auf aktuellen Xilinx- oder Altera-FPGAs).

3.6 Funktionsprinzipien von Spannungswandlern

3.6.1 Linearregler

Bei Linearreglern befindet sich zwischen Ein- und Ausgang ein Bipolar- oder Feldeffekttransistor, der als veränderbarer Widerstand betrieben wird. Durch die Ansteuerung über einen Regelverstärker mit zugehöriger Referenzspannungsquelle, fällt an diesem Transistor immer gerade die Differenzspannung ΔU zwischen Ein- und Ausgang ab („Series Pass Element“ in Abbildung 3.16).

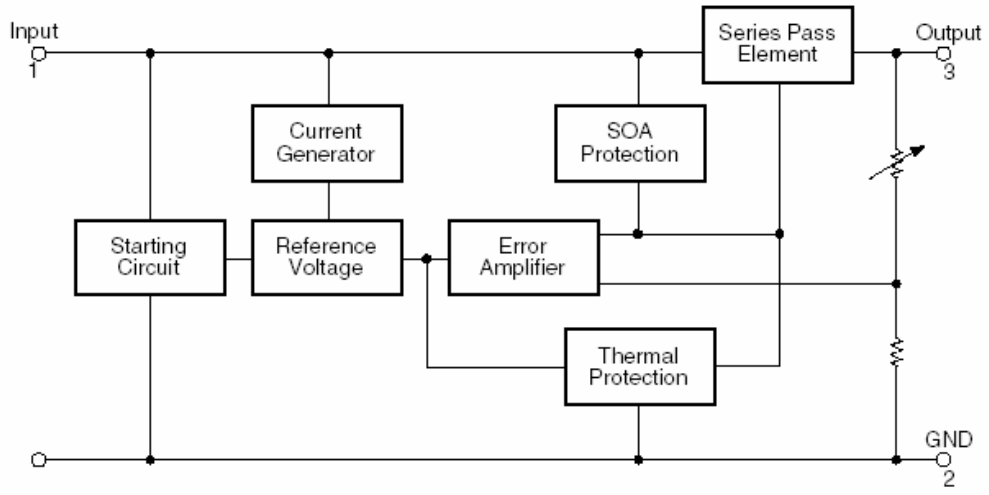

Abbildung 3.16: Schematischer Aufbau eines integrierten Spannungsreglers

Dadurch kann die Ausgangsspannung auf einen konstanten Wert geregelt werden. In integrierten Spannungsreglern findet man zusätzlich noch weitere Baugruppen, die dem Schutz des Transistors dienen. Üblich sind Schaltungen zur Strombegrenzung, zur Überwachung des sicheren Arbeitsbereichs (Save Operating Area, SOA) und der Temperatur. Der Nachteil von linearen Spannungsreglern ist die hohe Verlustleistung, die sich näherungsweise zu

$$P = I \cdot U = I \cdot (U_{ein} - U_{aus})$$

Formel 3.1: Verlustleistung eines Linearreglers (Näherung)

berechnet und der damit verbundene geringe Wirkungsgrad. Dadurch entsteht nicht nur ein großer Energieverlust, sondern auch ein entsprechendes Kühlungsproblem. Daher werden Linearregler vorzugsweise bei geringen Strömen bzw. bei geringen Spannungsdifferenzen verwendet. Wenn man unter Verwendung von SMD-Technik nur die Platine bzw. eine entsprechende Kupferfläche zur Wärmeabführung nutzt, sind abhängig vom Gehäusetyp maximal nur 1 bis 2W Verlustleistung zulässig ([IRFR5305], [IRF4905S], [LM117]).

3.6.2 Schaltregler

Die Verluste der Linearregler lassen sich stark reduzieren, indem man den kontinuierlich geregelten Transistor durch einen Schalter wie in Abbildung 3.17 ersetzt. Um die gewünschte Ausgangsgleichspannung zu erhalten, benötigt man zusätzlich einen Tiefpassfilter, der den zeitlichen Mittelwert bildet.

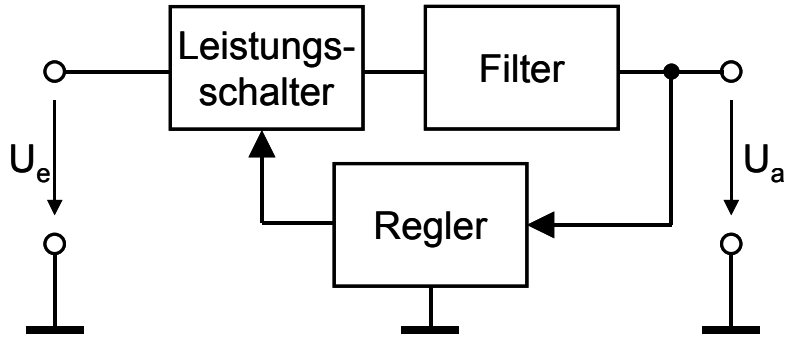

Abbildung 3.17: Schema eines Schaltreglers

Die Größe der Ausgangsspannung lässt sich in diesem Fall durch das Tastverhältnis bestimmen, mit dem der Schalter geschlossen wird. Wenn man einen LC-Tiefpassfilter verwendet, gibt es im Regler keine systematische Verlustquelle mehr. Der Wirkungsgrad kann daher über 90% betragen. Die drei Grundformen solcher Gleichspannungswandler sind der Abwärtswandler ($0 \le U_a \le U_e$), der Aufwärtswandler ($U_a \ge U_e$) und der Invertierende Wandler ($U_a < 0$). Nähere Betrachtungen hierzu findet man in [TiSc02].

3.6.2.1 Asynchrone und Synchrone Abwärtswandler

Für die Versorgung von Digitalschaltungen werden meist Spannungen kleiner/gleich 5V benötigt. Bei einer Speisung mit einer Eingangsspannung von $U_e > 5V$ kommen daher nur Abwärtswandler in Frage. Bei der technischen Realisierung eines Tiefsetzstellers gibt es zwei verschiedene Bauformen. Den synchronen und den asynchronen Abwärtswandler (Abbildung 3.18).

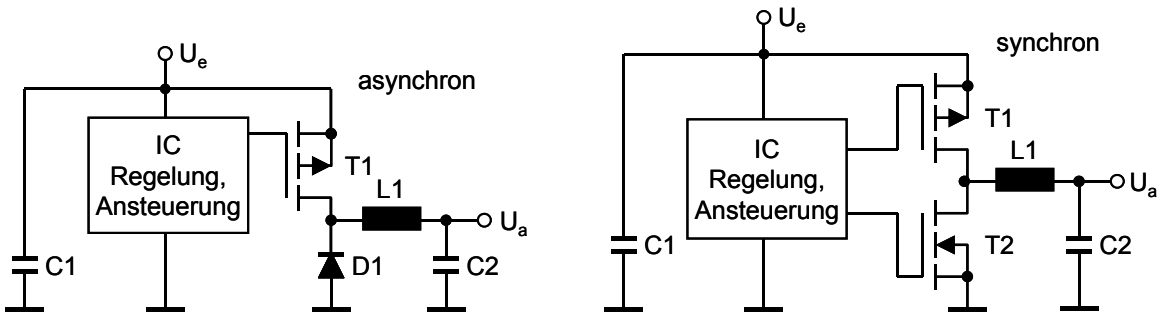

Abbildung 3.18: Asynchroner und Synchroner Abwärtswandler

Solange der FET T1 leitend ist, liegt beim asynchronen Abwärtswandler an der Diode D1 die Spannung U_e an und es fließt ein Strom durch die Spule zum Ausgang. Im gesperrten Zustand behält der Drosselstrom seine Richtung bei und die Diode D1 wird leitend. Somit trägt die Diode während der Ausschaltzeit den Spulenstrom. Da an der Diode im Vergleich zum FET ein hoher Spannungsabfall von 0,4 bis 0,5V entsteht, begrenzt die Verlustleistung an der Diode die Einsatzfähigkeit des asynchronen Abwärtswandlers. Bei Verwendung von SMD-Dioden ohne zusätzliche Kühlkörper kann er bis zu Ausgangsströmen von etwa 5A eingesetzt werden.

Für Ströme über 5A eignet sich der synchrone Wandler, bei dem ein zweiter FET (T2) den Spulenstrom während der Ausschaltzeit von T1 übernimmt. Dadurch wird die Verlustleistung verringert, da der Spannungsabfall an T2 wesentlich geringer als der an der Diode ist. Bei ähnlichem Aufbau in SMD sind dadurch ohne Kühlkörper Ströme über 15A möglich [Nati05].

3.7 Vierquadrantenverstärker

Die sicherlich universellste Art, Ausgangssignale für einen Prozess zu erzeugen – sei es nun innerhalb einer Rapid-Prototyping- oder in einer Hardware-in-the-Loop-Umgebung, ist die Verwendung eines Vierquadrantenverstärkers. Dieses Bauteil ist in der Lage, wie bereits am Namen ersichtlich, alle vier Quadranten des U/I-Koordinatensystems anzufahren. Dadurch entsteht die Möglichkeit, mit einem einzigen Treiber, Ströme und Spannungen zu erzeugen, sowie als Senke derselben zu fungieren. Implementiert man dann noch wie in [Spit01] eine Möglichkeit zur dynamischen Messung und Adaptierung bzw. Einstellung der Betriebsparameter der Baugruppe, können praktisch beliebige Szenarien in Form von Kennlinien durchfahren werden. Eine solche Baugruppe stellt damit eine ideale Ergänzung für das in dieser Arbeit entwickelte System dar.

3.8 Code-Erzeugung für programmierbare Schaltungen

3.8.1 VHDL und Verilog

VHDL und Verilog sind beide Hardware-Beschreibungssprachen (engl. Hardware Description Language), mit denen elektronische Systeme textuell spezifiziert werden können. Durch entsprechende Erweiterungen lassen sich auch nicht elektrische/elektronische Systeme damit simulieren und verifizieren.

VHDL steht für Very High Speed Integrated Circuit Hardware Description Language und wurde in den frühen 80ern entwickelt. Es ist durch den IEEE 1076 Standard von 1993 genormt. Gegenüber dem ersten Standard von 1987 (IEEE 1076-1987) wurde die Syntax vereinheitlicht und ergänzt, aber auch einige Konstrukte der alten Syntax entfernt, sowie sogar die Semantik einzelner Konstrukte verändert. Verilog HDL ist neben VHDL die weltweit

meistgenutzte Hardwarebeschreibungssprache. Während sich VHDL zum "Quasi-Standard" innerhalb Europas entwickelt hat, ist Verilog dagegen die meist verwendete Sprache innerhalb der USA. Sie wurde 1983/84 ursprünglich als Simulationssprache entworfen und ist heute als IEEE-Standard 1364-2001 genormt.

Hardware-Beschreibungssprachen unterscheiden sich in zwei Punkten signifikant von klassischen Programmiersprachen. Zum einen wird der Aufbau einer physikalischen Schaltung modellhaft beschrieben, zum anderen werden die so entstandenen Konstrukte parallel ausgeführt. Programmiersprachen dahingegen legen einen Ablauf einzelner Anweisungen fest und können von einem einzigen Prozessor nur sequentiell bzw. quasi-parallel abgearbeitet werden.

3.8.2 IP-Cores

Vorgefertigte Schaltungsteile oder Module, sofern es sich um programmierte, elektronische Versionen und nicht um bereits in Hardware gegossene Chips, wie z.B. ASICs, handelt, werden gemeinhin als Intellectual Property Cores (IP-Cores) bezeichnet. Sie wurden von einem oder mehreren Entwicklern erstellt und meist auch funktional getestet. In seltenen Fällen wurden bereits Implementierungen auf FPGAs zur Verifikation des Designs nach dem Post-Place-and-Route vorgenommen und die Ergebnisse abgedruckt.

Das größte Problem an diesen IP-Cores für den Urheber oder Erfinder desselben ist die leichte Weiterverwendung und –verbreitung dieser Designs. Sind sie einmal im Umlauf, ist dem möglicherweise rechtswidrigen Kopieren nur schwer ein Riegel vorzuschieben. Trotzdem oder gerade deswegen, gibt es Bestrebungen, solche Cores durch im Code enthaltene Mechanismen entsprechend zu schützen. Die Effektivität dieser Maßnahmen ist aber nicht immer sehr Erfolg versprechend.

Es gibt verschiedene Basen, auf denen solche IP-Cores verfügbar gemacht werden können. Die zwei der gängigsten sind hier im Anschluss aufgeführt.

3.8.2.1 Quell-Code-basierte IP-Cores

Wie aus dem Namen schon zu erkennen ist, werden hier die kompletten Source-Codes als VHDL- oder Verilog-Beschreibung weitergereicht. Damit kann der Anwender im Zweifel am Meisten anfangen, sofern noch Änderungen zur Anpassung an eigene Konstrukte nötig sind. Auch das Herauskopieren von Passagen oder Strukturen ist bei dieser Art der Verbreitung problemlos möglich. Die Portierbarkeit auf verschiedene Plattformen ist ohne Probleme möglich, sofern nicht schon in diesem Quelltextformat auf Target-spezifische Ressourcen Bezug genommen wurde. Meistens handelt es sich aber um reine Chip-unabhängige Hardware-Beschreibungen in VHDL oder Verilog.

3.8.2.2 Netzlisten-basierte IP-Cores

Bei den netzlisten-basierten IP-Cores ist der erste Schritt der Implementierung schon erledigt. Der Core hat bereits beim Hersteller einen Syntax-Check und die Synthese durchlaufen und liegt nunmehr als Netzliste – zum Beispiele im EDIF-Format (Electronic Design Interchange Format, [Burs00]) – vor. Damit ist die strukturale oder Verhaltensbeschreibung in eine gatterbezogene Notation übergegangen, die nicht ohne Weiteres zurückgewandelt werden kann – sprich zu re-engineeren ist. Änderungen sind daher in diesem Stadium bereits nicht mehr möglich. Diese Netzliste ist allerdings immer noch technologieunabhängig und kann daher nach wie vor auf verschiedenen Targets eingesetzt werden.

3.8.2.3 Soft-Prozessoren und Schnittstellen

Zu der Gattung der IP-Cores gehören auch die meisten Soft-Prozessoren oder -Controller. Sie werden entweder von den FPGA-Herstellern selbst als Erweiterung ihrer Produktpalette zum Kauf angeboten oder von herstellernahen Drittanbietern und liegen in diesen Fällen als netzlistenbasierte IP-Cores vor. Der 32-bit-Soft-Microprozessor MicroBlaze von Xi-

linx und der vergleichbare Nios von Altera gehören zu den bekanntesten. Zudem gibt es eine Menge von industriellen oder auch privaten Projekten, in denen solche Controller entstehen. Diese sind im Internet zur freien Verwendung verfügbar und dann meist auch im Source-Code herunterzuladen. Mit der Einschränkung noch unzureichender Überprüfung ihrer Funktion seitens des Designers, können sie in eigenen Projekten verwendet und auch dafür modifiziert werden. Ein bekannter Vertreter dieser Klasse ist der LEON-Prozessor [LEON01], der ursprünglich für ESA-Raumfahrtmissionen von Jiri Gaisler [Gais01] entwickelt wurde und seit einigen Jahren in der zweiten und dritten Generation frei im Internet zugänglich ist. Darüber hinaus gibt es eine Vielzahl an weiteren Prozessoren und Controllern, die auf Basis älterer, in Chip gegossener Bausteine, wie den 8085, diverse PIC-Derivate, 8051, Atmel- und auch Motorola-Controller.

Das Gleiche gilt für die Schnittstellen, die an solche Prozessoren angeschlossen werden können, um sich selbst ein optimales System zu schaffen. Hier ist mittlerweile jede Art von Ein- oder Ausgabe verfügbar, solange es sich um eine digital nachbildbare handelt. So finden sich digitale Standard-I/Os, serielle Schnittstellen, Controller für CAN-, I2C- und SPI-Busse, PWM-Generatoren, USB, Ethernet, usw. sowohl in käuflicher als auch in freier Form (www.opencores.org) im Internet.

3.8.3 Programmiersprache Java und der Editor Eclipse

Die Java-Technologie wurde ursprünglich von Sun Microsystems entwickelt und umfasst die Definition der Programmiersprache Java und die Java-Plattform, welche die Ablaufumgebung und die Programmierschnittstellen (Java Application Programming Interface, API) enthält. Zur Ausführung von Java-Programmen ist eine Java-Laufzeitumgebung notwendig, das so genannte Java Runtime Environment. Das Java Runtime Environment umfasst die Java-Klassenbibliotheken und die Java Virtual Machine. Sie bildet die Schnittstelle zur Maschine bzw. dem Betriebssystem und führt die kompilierten Java-Programme aus, die in einem speziellen maschinenunabhängigen Bytecode vorliegen.

Eclipse ist eine Open-Source-Entwicklungsumgebung, die für die Entwicklung von Java-Programmen verwendet wird. Sie wurde 2001 von IBM freigegeben und stellt eine Basis-Oberfläche zur Entwicklung von Software zur Verfügung, die mit unterschiedlichen PlugIns erweitert werden kann. Dadurch ist es z.B. auch möglich, C++-Programme zu erstellen. Die Benutzeroberfläche von Eclipse ist in Abbildung 3.19 dargestellt. Sie ist in der gezeigten Ansicht in vier Felder unterteilt, wobei die Zusammenstellung von angezeigten Informationen vom Benutzer geändert werden kann. Links befindet sich der Projekt-Navigator, indem alle derzeit geöffneten Projekte und deren verwendete Dateien aufgelistet sind. Der mittlere, obere Teil zeigt einen im Editor geöffneten Java-Quell-Code, der darin erstellt und modifiziert werden kann. Es können dabei mehrere Dateien parallel bearbeitet werden, die Auswahl erfolgt über Reiter. Rechts daneben im „Outline“ werden die zu dem Quell-Code gehörigen Funktionen, Variablen usw. angezeigt. Das Konsolenfenster am unteren Rand dient der Ausgabe von Meldungen und Statusinformationen des Compilers und eigenen, im Quell-Code verwendeten Hinweisen.

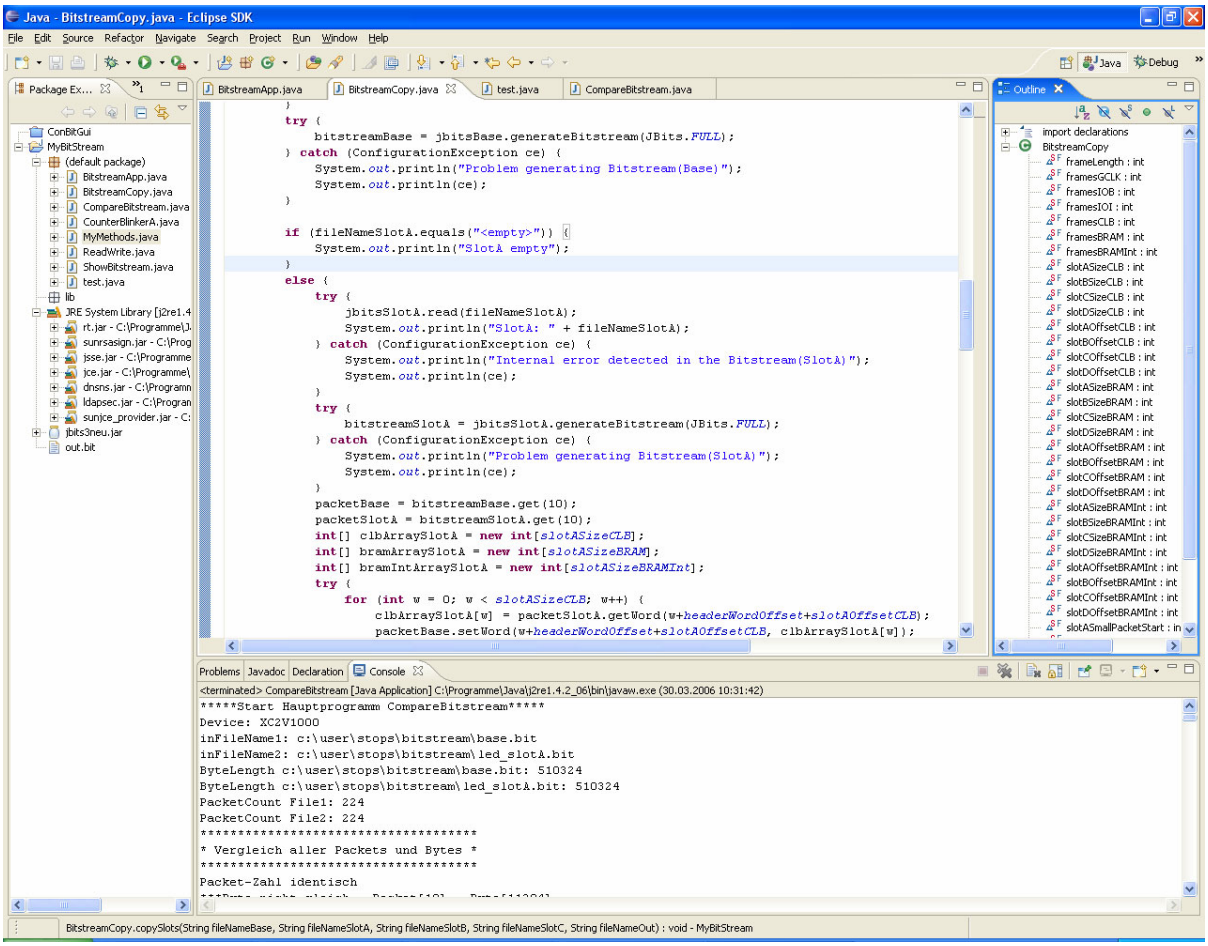

Abbildung 3.19: Screenshot der Benutzeroberfläche Eclipse

3.8.4 Kombinationswerkzeuge (Xilinx ISE/EDK/PlanAhead)

3.8.4.1 Integrated Software Environment (ISE)

Die ISE (Integrated Software Environment) [XISE05] ist eine integrierte Entwicklungsumgebung für den Entwurf von FPGA-Designs für Xilinx-FPGAs. Zu ihrem Umfang gehören der Project Navigator (siehe Abbildung 3.20) zum Verwalten von Projekten und deren Projekteinstellungen und diverse andere Tools, wie z.B. dem Floorplanner, dem FPGA-Editor oder einem Schematic Editor, die den Entwickler beim Design von FPGAs unterstützen.

Im Project Navigator können alle Schritte durchgeführt werden, die für das Design notwendig sind. Dazu gehört das Erstellen von VHDL-Quelldateien in einem Text-Editor, die Synthese mit dem integrierten Synthese-Tool XST (Xilinx Synthesis Technology) von Xilinx [Xil03], die Implementierung mit den Schritten Translate, Map und Place-&-Route sowie dem Generieren eines Bitstreams zur Programmierung des FPGA. In der aktuellen Ansicht in Abbildung 3.20 gliedert sich die Darstellung in vier Fenster. Auf der linken Seite sind im oberen Teil des Tools das geöffnete Projekt und die darin enthaltenen Dateien zu sehen. Durch einen Doppelklick lassen sich diese öffnen und werden im Falle einer textuellen VHDL- oder Verilog-Beschreibung im Editor (rechts im Bild) angezeigt. Bei grafischen Implementierungen öffnet sich automatisch der Grafik-Editor.

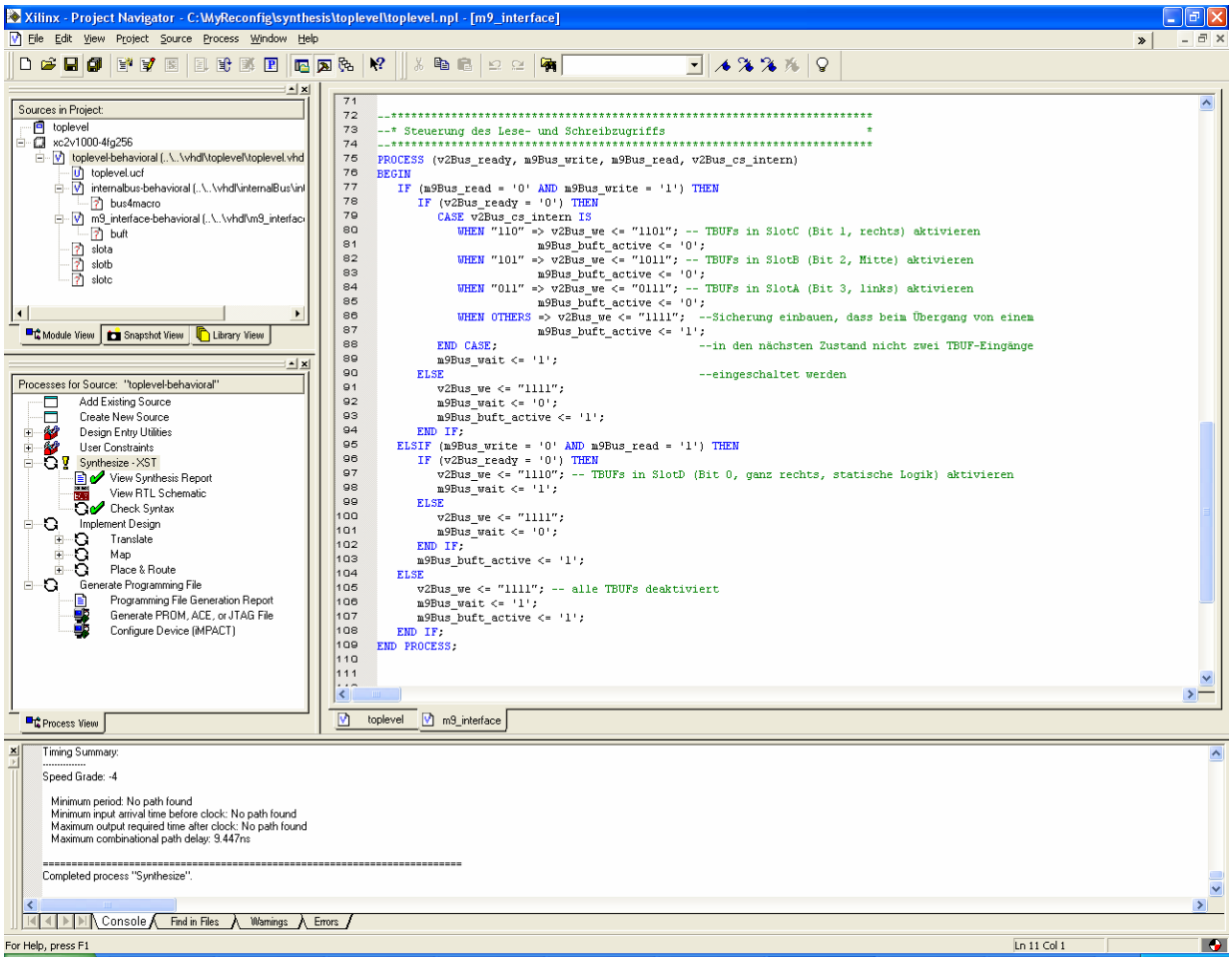

Abbildung 3.20: ScreenShot der ISE [XISE05] von Xilinx

Der Schematic-Editor kann dazu benutzt werden, Symbole verschiedener VHDL-Module in einer grafischen Oberfläche mit Leitungen zu verknüpfen und aus diesem Schematic automatisch eine VHDL-Entity des Gesamtsystems zu erstellen. Mit Hilfe des FPGA-Editor kann das implementierte Design betrachtet werden. Außerdem ist es mit dem FPGA-Editor möglich, die Schritte des Platzierens und Verdrahtens manuell durchzuführen sowie Makro-Dateien zu erstellen. Um Aktionen, wie Synthese, Mapping, Place&Route, usw. auszuführen, befindet sich auf der linken Seite eine Baumstruktur, in der die einzelnen Funktionen (und evtl. Unterfunktionen) aufgelistet sind, die bei Doppelklick auf einen Eintrag den entsprechenden Design-Schritt ausführen. Status- und Fehlermeldungen der einzelnen Schritte werden im Konsolenfenster ganz unten ausgegeben.

Für alle Designschritte können außerdem spezielle Einstellungen vorgenommen werden, um die Vorgehensweise der ISE bei den einzelnen Schritten den gewünschten Anforderungen anzupassen. Die standardmäßige Vorgehensweise beim Entwurf eines FPGA-Designs gliedert sich in folgende Schritte:

- **Erstellen von Quelldateien:** Im Text-Editor des Project-Navigator werden VHDL-Dateien erstellt, die das Verhalten eines Gesamtsystems oder einzelner Komponenten beschreiben.
- **Synthese:** Bei der Synthese wird aus einem Design-Projekt bestehend aus einer oder mehreren VHDL-Dateien eine Netzliste aus Logikgattern erstellt und in einer Ngc-Datei gespeichert.

- **Translate:** In diesem Design-Schritt wird eine synthetisierte Netzliste (Ngc-Datei) eingelesen. Anschließend werden alle Komponenten in der Netzliste auf Xilinx-NGD-Primitive (Native Generic Database) reduziert. Dabei werden Komponenten aus der Systembibliothek, Makros (Nmc-Dateien) und Verhaltensmodelle zusammengefügt. Als Ausgabe wird eine Ngd-Datei erzeugt.
- **Map:** Beim Mapping wird das logische Design auf die im FPGA zur Verfügung stehenden Komponenten abgebildet. Dazu wird die im vorigen Schritt erzeugte Ngd-Datei und alle benötigten Makro-Dateien (Nmc-Datei) eingelesen. Als Ausgabe erhält man eine Ncd-Datei (Native Circuit Description).
- **Place-&-Route:** Die beim Mapping entstandene Ncd-Datei wird als Grundlage für den Place-&-Route-Vorgang benutzt. Beim Platzieren werden die gemappten Komponenten auf passende Elemente im FPGA platziert. Im nächsten Schritt, dem Verdrahten (Routing), werden die platzierten Komponenten mit Leitungen verbunden. Sind alle Komponenten verbunden, wird das geroutete Design in einer neuen Ncd-Datei gespeichert.
- **Bitgen:** Die Ncd-Datei des vollständig verdrahteten Designs wird als Basis für die Generierung eines Bitstreams zur Programmierung des FPGA verwendet. Als Ausgabe erhält man eine Bit-Datei, welche sowohl verschiedene Informationen über das Design in einem Header als auch alle Konfigurationsdaten des FPGA enthält.

3.8.4.2 Embedded Development Kit (EDK)

Die EDK (Embedded Development Kit) [XEDK04] ist eine Sammlung von Software-Tools zur Unterstützung des Designs von Embedded Processor Systems auf einem programmierbaren Logikbaustein. So besteht die Möglichkeit, ein komplexes System, bestehend aus Hardware und Software auf einem FPGA, zu realisieren. Um ein Embedded Processor System zu entwerfen, muss als erster Schritt eine Hardware-Plattform ausgewählt werden. Eine Hardware-Plattform besteht aus einem oder mehreren Prozessoren, Bussen und Peripheriebausteinen, die über die Busse mit dem Prozessor verbunden sind. Nach der Definition des Hardwaresystems kann für dieses System Software geschrieben werden. Die EDK unterstützt sowohl Plattformen mit dem IBM-PowerPC-Core (Hard-IP-Core) als auch mit dem Xilinx-MicroBlaze (Soft-IP-Core).

Das Programm Xilinx Platform Studio [Xil04h] (Abbildung 3.21) stellt die grafische Benutzeroberfläche zur Verfügung und deckt alle Schritte vom Systementwurf über das Debugging bis hin zur Verifikation ab. Xilinx Platform Studio unterstützt den Anwender unter anderem beim Entwerfen der MHS-Datei (Microprocessor Hardware Specification) und der MSS-Datei (Microprocessor Software Specification), die benötigt werden, um das Gesamtsystem aus Hardware und Software zu beschreiben. In der gezeigten Abbildung findet sich der Projektbaum auf der linken Seite der Oberfläche. Über Reiter im oberen Teil des Fensters können verschiedene Projektansichten (Hardware, Software, ...) ausgewählt werden. Das Hardwaresystem kann mit Hilfe des Platform Block Diagram Editor, der das Hardwaresystem in einem Blockschaltbild mit seinen Komponenten (Prozessoren und deren Peripherie) und Bussen darstellt, angezeigt und editiert werden (rechtes Fenster). Darüber hinaus erlaubt Xilinx Platform Studio die Anpassung der Software-Bibliotheken (Libraries), die Definition von Treibern und das Kompilieren der vom benutzergeschriebenen C-Programme. Entgegen der ISE von Xilinx (Kap. 3.8.4.1) erfolgt hier die Steuerung der Software über die Menüleiste oben. Die ausgeführten Schritte, sowie Warnungen und aufgetretene Fehler, werden im unteren Fenster eingeblendet.

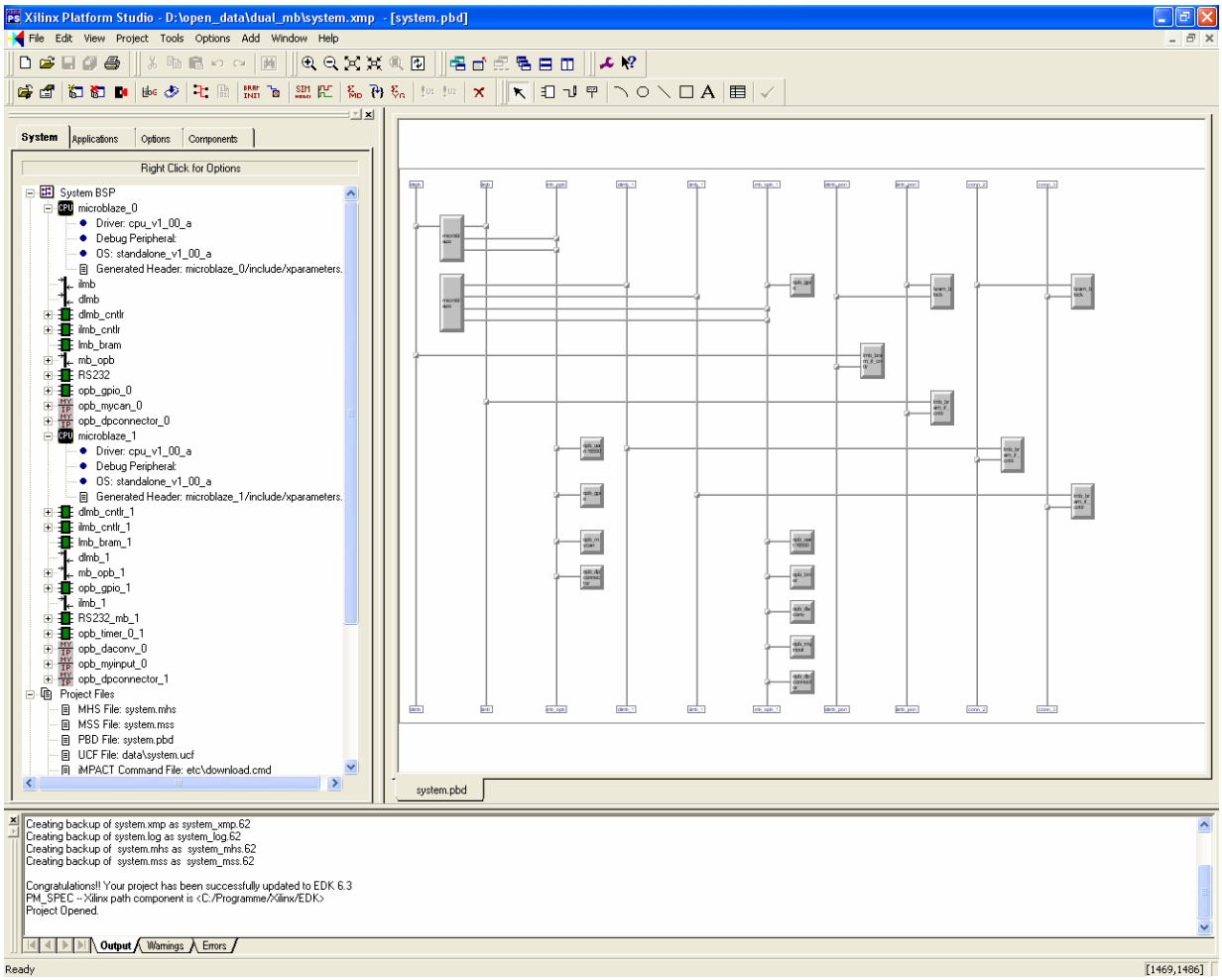

Abbildung 3.21: ScreenShot der EDK [XEDK04] von Xilinx

3.8.4.3 PlanAhead

PlanAhead [Plan05] ist ein neues Tool von Xilinx zur Entwicklung und Analyse von FPGA-Designs. Es wird zwischen der Synthese und der Implementierung eingesetzt, um das Design zu analysieren, zu verifizieren und zu verbessern. Bei der Erstellung eines neuen Projekts wird die bei der Synthese entstandene Netzliste importiert, ein gewünschter Zielbaustein ausgewählt und optional eine zum Design gehörende Constraint-Datei (ucf-Datei) hinzugefügt. Gleichzeitig wird ein Floorplan für das Design erzeugt. Im Projekt können beliebig viele Floorplans mit unterschiedlichen Constraint-Dateien angelegt und verglichen werden. Die in der Constraint-Datei enthaltenen Constraints werden dem Design hinzugefügt, sofern diese von PlanAhead verarbeitet werden können. Dabei werden z.B. in der Constraint-Datei definierte Area-Groups in einen physikalischen Block (so genannter „Pblock“) umgewandelt und Portzuweisungen übernommen. Timing-Constraints bleiben ebenfalls erhalten.

PlanAhead stellt verschiedene Funktionen zur Verfügung, um das Floorplanning zu optimieren, dem Design zusätzliche Constraints hinzuzufügen und das Design zu analysieren. Abbildung 3.22 zeigt die Ansicht der grafischen Oberfläche des Tools. Im Fenster Package View werden die im FPGA vorhandenen Pins angezeigt und können dort direkt den in der Netzliste vorhandenen Eingangs- und Ausgangsports zugeordnet werden. Das Fenster Device View (oben rechts) zeigt die Fläche des FPGA mit den vorhandenen Ressourcen an. Hier wird das Floorplanning durchgeführt.

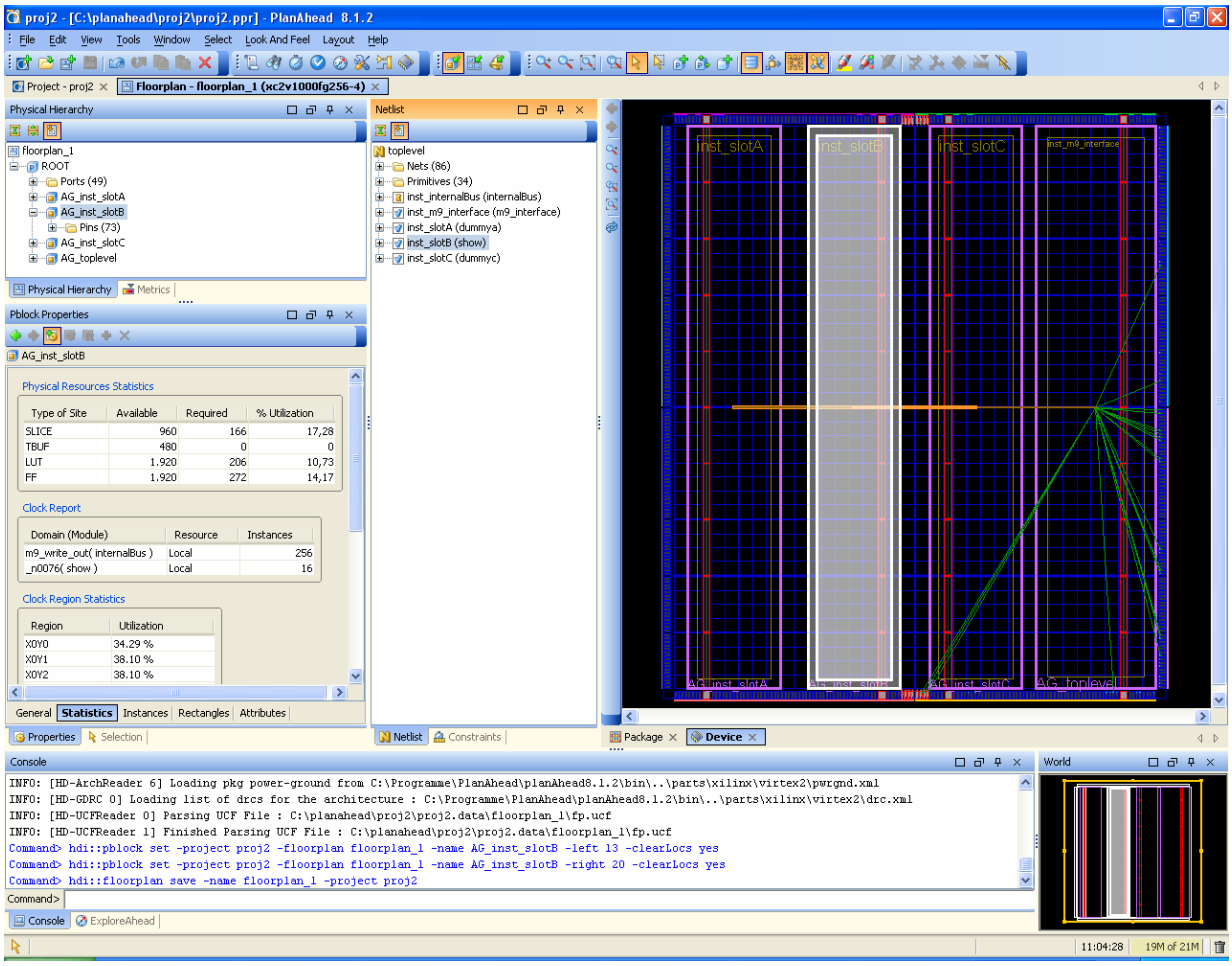

Abbildung 3.22: ScreenShot des Tools PlanAhead [Plan05] von Xilinx

Das Aufteilen des Gesamtdesigns in eine hierarchische Struktur aus kleineren Modulen erleichtert die Analyse und Verbesserung des Designs. Hierzu werden verschiedene physikalische Blöcke erstellt (helle Rahmen innerhalb des Device-Views) und einzelne Module den jeweiligen Blöcken hinzugefügt. Die Module liegen als Netzlisten vor und können im mittleren Fenster selektiert werden. Die physikalischen Blöcke können in ihrer Größe verändert und auf dem FPGA verschoben werden. Die Verbindungsleitungen zwischen den verschiedenen Modulen in den unterschiedlichen physikalischen Blöcken werden je nach Anzahl unterschiedlich dargestellt. Dadurch kann schnell erkannt werden, wie stark die Verknüpfung verschiedener Module untereinander ausgeprägt ist. Die in einem physikalischen Block belegten Ressourcen werden, ebenso wie Informationen zur Anzahl der Netze, der benutzten Taktregionen oder der Größe des Blocks im Fenster Pblock Properties (links, Mitte) angezeigt. Eine weitere wichtige Funktion stellt das Austauschen von Netzlisten des Toplevel-Designs oder auch einzelner Module dar. Auf diese Weise kann eine bestimmte Netzliste mit einer neuen Version aktualisiert werden. Die Funktion TimeAhead analysiert das Timing des Designs. Mit Hilfe der Funktion Export Floorplan kann der fertig gestellte Floorplan in einen beliebigen Ordner exportiert werden. Dem Ordner werden dabei die für das Design benötigten Netzlisten und eine Constraint-Datei hinzugefügt, welche die veränderten und hinzugefügten Constraints des Designs beschreibt. Dies bildet den Ausgangspunkt für die Durchführung der Implementierung. Die Schritte zur Implementierung eines FPGA-Designs sind ebenso wie die Funktion zum Generieren eines Bitstreams in die PlanAhead-Umgebung integriert. Die möglichen Optionen für die einzelnen Schritte Translate, Map, Place-&-Route und Bitgen werden dem Benutzer in einer grafischen Benutzeroberfläche angezeigt und können dadurch sehr einfach an die vorgegebenen Anforderungen angepasst werden. Mit Hilfe der

Funktion ExploreAhead können verschiedene Implementierungsstrategien automatisch hintereinander durchgeführt und die Ergebnisse verglichen werden. Meldungen wie Status, Warnungen und Fehler werden in der Konsole ganz unten links ausgegeben. Das rechte untere Fenster zeigt immer die gesamte FPGA-Fläche und den im Device-View gezoomten Bereich an.

3.8.5 Klassenbibliothek Jbits von Xilinx

JBits enthält eine Sammlung von Java-Klassen, die von Xilinx zur Verfügung gestellt wird, um Bitstreams auf Softwareebene zu erstellen oder zu verändern. Die JBits-Klassen erlauben den direkten Zugriff auf die Ressourcen des FPGA wie z.B. den CLBs, welche die LUTs enthalten, und die Routing-Ressourcen. Das Programm zur Veränderung des Bitstreams wird in Java geschrieben, wozu eine beliebige Java-Entwicklungsumgebung verwendet werden kann. Ausgangspunkt hierfür bildet ein vom Xilinx-Entwicklungs-Tool generierter oder ein von einem FPGA zurückgelesener Bitstream. Ursprünglich wurde JBits entwickelt, um die partielle dynamische Re-Konfiguration von FPGAs zu unterstützen, da auf diese Weise kleine Änderungen am FPGA-Design auf CLB-Ebene schnell und effektiv vorgenommen werden können. Bei der Manipulation von Bitstreams mit JBits muss sehr genau und vorsichtig vorgegangen werden, da durch die Veränderung des Bitstreams auf Bitebene ein neuer Bitstream entstehen kann, der den FPGA zerstört. Hier existieren keine Sicherheitsmechanismen, die das Programmieren eines FPGA mit einem solchen Bitstream verhindern. Die aktuellste Version von JBits trägt die Bezeichnung JBits SDK 3.0 und unterstützt ausschließlich Virtex-II-FPGAs.

3.9 Vorgehensmodelle und Werkzeugeinsatz

3.9.1 Lebenszyklusmodelle zur Systementwicklung

Um die Komplexität der Steuergeräte, die in immer kürzeren Zeiträumen und unter erheblichem Kostendruck entwickelt werden müssen, handhaben zu können, bedarf es des Einsatzes von modernen Methoden zur Systementwicklung. Von der Erstellung der Anforderungen an ein neues Produkt bis zur Auslieferung an den Kunden werden dem Entwickler damit klar definierte Vorgänge und Verfahren an die Hand gegeben, die helfen sollen, die Entwicklungsfortschritte zu bewerten und zu überwachen.

Zur Abstrahierung dieser Aktivitäten und Abläufe während des Entwurfs elektronischer Systeme wurden mehrere Lebenszyklusmodelle entwickelt, die durch verschiedene Standards normiert wurden (z.B. IEEE 830–1993, MIL–STD–498). Im Folgenden sind die bekanntesten und wichtigsten Verfahren aufgezeigt und nähere erläutert. Zum Teil sind diese sehr starr an ein bereits vorhandenes Verfahren angelehnt oder setzen sich aus einzelnen Teilen von anderen zusammen. In der Literatur existieren noch eine Vielzahl von weiteren Lebenszyklusmodellen, auf die aber an dieser Stelle nicht weiter eingegangen werden soll.

3.9.1.1 Wasserfall-Modell

Das Wasserfallmodell besteht aus den Phasen Systemanalyse, Spezifikationsphase, Systementwurfsphase und Implementierungsphase und ist in seiner ersten Version durch einen linearen Ablauf gekennzeichnet. Am Ende einer jeden Phase wird ein abgeschlossenes Teilprodukt an die nächstfolgende Phase übergeben. Der praktische Einsatz hat allerdings gezeigt, dass Iterationsschleifen an den Übergängen der Projektphasen nötig sind, die durch ungenaue Spezifikationen und Entwurfsfehler entstehen können. Abbildung 3.23 enthält bereits den von [Boeh76] angepassten Entwicklungsverlauf. Wegen des sehr späten Systemtests ist das Wasserfall-Modell kritisch zu betrachten, da erst zu diesem Zeitpunkt Spezifikationsfehler entdeckt werden. Bei Änderungen müssen nochmals alle Phasen durchlaufen werden.

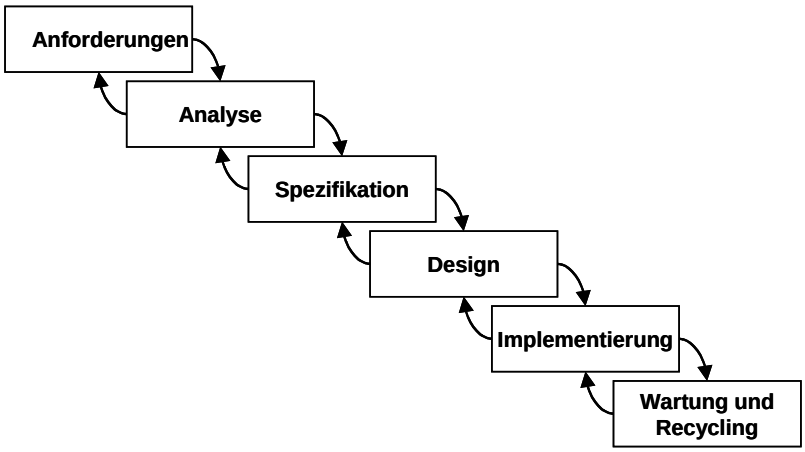

Abbildung 3.23: Wasserfall-Modell

3.9.1.2 Spiral-Modell

Bei einer Entwicklung gemäß dem Spiralmodell wird in jeder Iteration die gleiche Menge an Schritten durchgeführt. Die entscheidende Verbesserung gegenüber dem Wasserfall-Modell besteht in der Erstellung von Prototypen in jedem Entwicklungsschritt. Dadurch wird das Entwicklungsrisiko erheblich reduziert und der zeitliche Aufwand beim Auffinden von Fehlern minimiert.

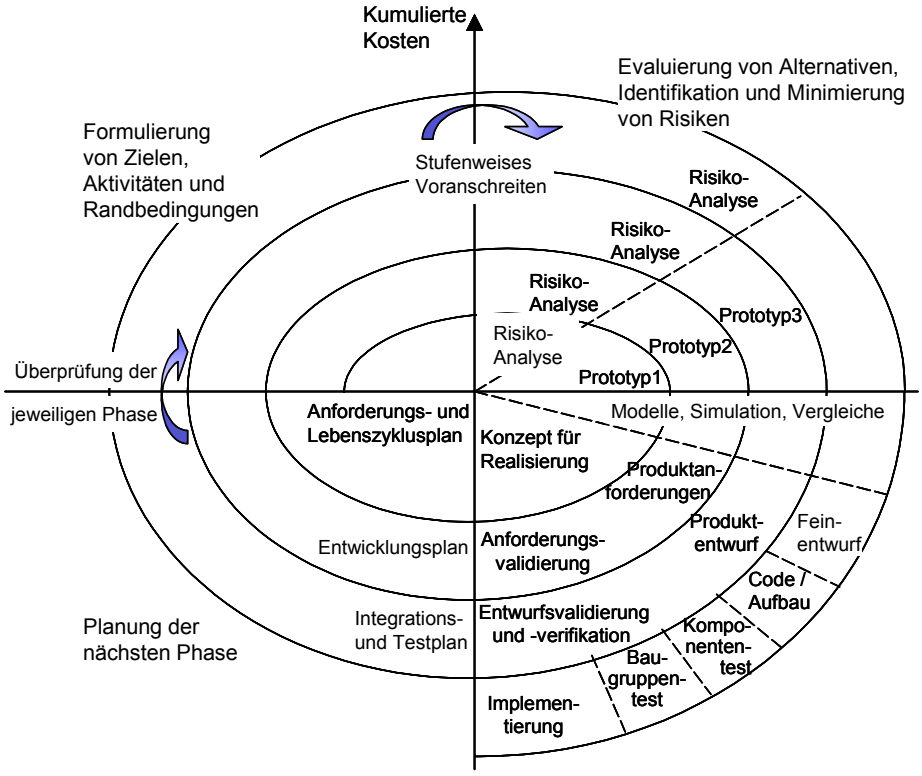

Abbildung 3.24: Spiral-Modell

3.9.1.3 V-Modell

Das V-Modell [IABG06] ist eine Weiterentwicklung des Wasserfall-Modells und enthält neben den eigentlichen Entwurfsschritten auch die den jeweiligen Phasen entsprechenden Validierungs- und Verifikationsphasen.

Definition Validierung:

Bestätigen aufgrund einer Untersuchung und durch Bereitstellung eines Nachweises, dass die besonderen Forderungen für einen speziellen beabsichtigten Gebrauch erfüllt worden sind. In Design und Entwicklung betrifft Validierung den Prozess der Untersuchung eines Produkts, um Konformität mit Erfordernissen des Anwenders festzustellen. (DIN EN ISO 8402, 1995-08 , Ziffer 2.18)

Definition Verifikation:

Bestätigen aufgrund einer Untersuchung und durch Bereitstellung eines Nachweises, dass festgelegte Forderungen erfüllt worden sind. In Design und Entwicklung umfasst Verifizierung die Untersuchung des Ergebnisses einer betrachteten Tätigkeit, mit dem Ziel, Übereinstimmung mit den an diese Tätigkeit gestellten Forderungen festzustellen. (DIN EN ISO 8402, 1995-08 , Ziffer 2.17)

Jede Phase auf der Spezifikations- und Entwurfsseite hat somit ihre Entsprechung auf der gegenüberliegenden Seite der Implementierung und Integration. Aus dieser Gegenüberstellung ergibt sich das dem Modell den Namen verleihende charakteristische V, aus dem sich auch symbolisch die stufenweise Zerlegung und Verfeinerung des Systems innerhalb der Implementierungsphase (Top-Down-Ansatz) und das Zusammensetzen der Sub-Module in der Verifikationsphase (Bottom-Up-Ansatz) ableiten lässt. Durch die den gesamten Entwicklungsprozess begleitende Verifikation und Validierung kann eine Konsistenz und Vollständigkeit der Entwurfsbeschreibung auf jeder Ebene gewährleistet werden.

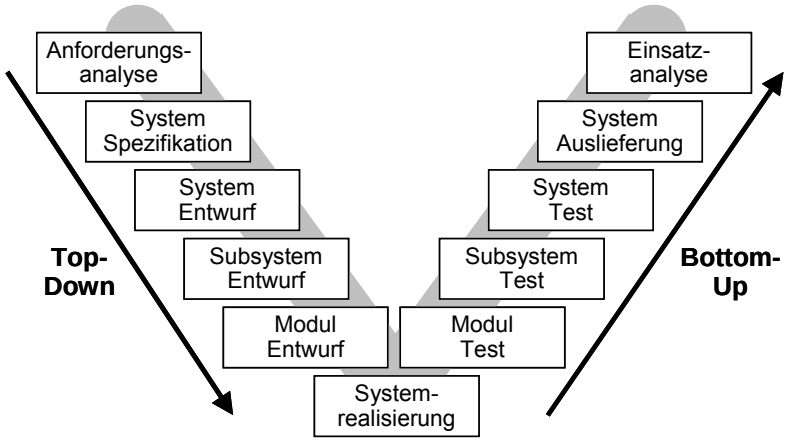

Abbildung 3.25: Das V-Modell

3.9.1.4 VP-Modell

Ergänzt man das V-Modell um die verschiedenen Phasen des Rapid Prototyping (siehe dazu auch Kapitel 3.10.2 Phasen des Rapid Prototyping), so gelangt man zu dem so genannten VP-Modell des Entwurfsvorgangs unter Einsatz von Rapid Prototyping. Die Besonderheit hierbei ist, dass die einzelnen horizontalen Phasen, d.h. die Abstraktionsebenen des

V-Modells, den Phasen des Rapid Prototyping zugeordnet wurden. Dadurch erhält man eine Dreiteilung, wie sie in Abbildung 3.26 zu sehen ist [Burs00].

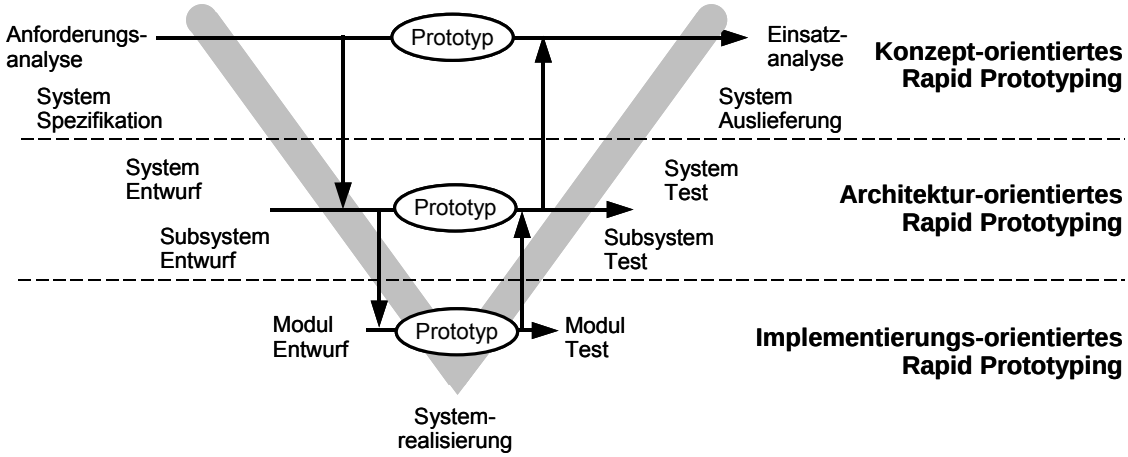

Abbildung 3.26: Das VP-Modell

3.9.2 Graphische Modellierung mit Case-Tools

Bei der Entwicklung elektronischer Systeme werden immer häufiger neben den klassischen Ansätzen der händischen Programmierung CASE-Tools eingesetzt, mit deren Hilfe die Erzeugung einer grafischen Repräsentierung der angestrebten Funktionalität möglich ist. Die bekanntesten CASE-Tools dieser Art sind MATLAB/Simulink und -/StateFlow [StFl97], MATRIXx, Statemate und ASCET-SD. Die Verlagerung des Designs auf eine höhere und abstraktere Ebene bringt verschiedene Vorteile mit sich. Zum einen kann der Benutzer – aber auch ein anderer Team-Kollege – die Zusammenhänge in einer Grafik besser verstehen und nachvollziehen als bei reinem Programmcode. Zum anderen bieten die Werkzeuge auch die Möglichkeit der Simulation des Designs, der automatischen Code-Erzeugung in mehreren Ausgabeformaten für Prototypen- und Serien-Code und nicht zuletzt eine automatische Dokumentation an.

Die Beschreibungsarten lassen sich generell in zwei Kategorien unterteilen. Man verwendet im Allgemeinen StateCharts zur Modellierung von ereignisorientierten, zeitdiskreten Systemen und Blockdiagramme zur Beschreibung zeitkontinuierlicher Systeme. Allerdings sind neben den Einzellösungen auch Mischformen möglich, die sich gegenseitig beeinflussen und über definierte Schnittstellen Daten austauschen. Im Folgenden sind beide Modellierungsarten näher erläutert.

3.9.2.1 Modellierung ereignis-diskreter Systeme

Statecharts stellen eine Erweiterung der klassischen Zustandsautomaten von Mealy und Moore oder der Petrie-Netze dar und integrieren Konstrukte wie Hierarchie und Nebenläufigkeit (parallele Ausführung von mehreren Teilen eines Automaten), um bessere Strukturierungs- und Modellierungsmöglichkeiten, insbesondere bei hohen Zustandszahlen, zu erreichen. Ferner werden durch Einführung bedingter Zustandsübergänge, die Verwendung hybrider Zustandsautomaten (Kombination von Mealy– und Moore-Automaten) und Zustände mit Rücksprungadresse die bisher nur rein sequentiellen Abläufe erweitert.

Die Basiselemente einer Beschreibung mit Statecharts sind Zustände (States), Startzustandsübergänge (Default Transitions) und allgemeine Zustandsübergänge (Transitions). Sie ergeben miteinander kombiniert die zeitdiskrete Funktion des Modells. Sollen dabei hierarchische Konstrukte dargestellt werden, so lässt sich dies durch Schachtelung von Zuständen in andere Zustände realisieren. Die Visualisierung von Nebenläufigkeiten hingegen geschieht

durch Trennung der einzelnen Zustandsfolgen innerhalb eines „Oberzustands“ mittels gestrichelter Linien.

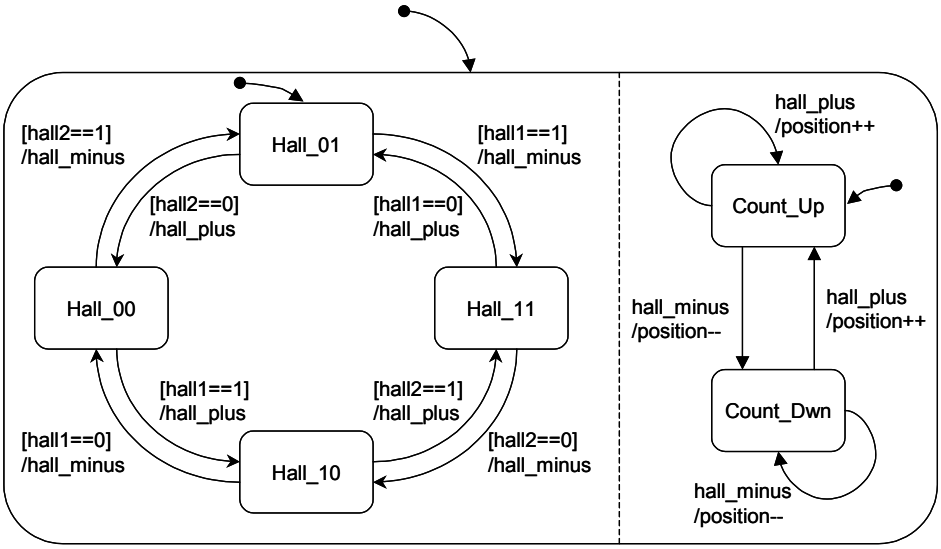

Abbildung 3.27: Beispiel eines StateCharts (MATLAB/StateFlow)

Im Beispiel in Abbildung 3.27 ist ein Positionszähler modelliert, der seinen Wert in Abhängigkeit von Hall-Signalen ändert. Dazu werden beim Start zunächst der übergeordnete Zustand und danach sofort die parallelen Zustände `Hall_01` und `Count_Up` aktiviert. Ändert sich die Variable `hall1` auf 1, ist die Bedingung (Condition) `[hall==1]` erfüllt und das Chart wechselt in den Zustand `Hall_11`. Dabei wird die Action `hall_minus` getriggert. Im parallelen Chart ändert sich durch den ausgelösten Event der Zustand auf `Count_Dwn` und die Variable `position` wird inkrementiert. Die Übergänge für andere Kombinationen der Hall-Signale laufen ähnlich dem beschriebenen ab.

3.9.2.2 Modellierung regelungstechnischer Systeme

Bei der grafischen Modellierung regelungstechnischer Systeme werden anstelle von Zuständen mathematische Basisblöcke aneinander gereiht und der Signalfluss zwischen den Blöcken mit Hilfe von Pfeilen dargestellt. Die genannten Basisblöcke umfassen dabei die aus den Operationsverstärkerbasisschaltungen der Elektronik bekannten Funktionen wie verstärken, addieren, subtrahieren, multiplizieren, integrieren, usw., sowie weitere Elemente, um Signale zu generieren, zu filtern, zu begrenzen oder gegebenenfalls auch zu visualisieren. Aus diesem Satz an Funktionen lassen sich beliebige dynamische, kontinuierliche Systeme aufbauen und simulieren. Auch hier ist es möglich, mit Hilfe der Gruppierung von einzelnen Modellteilen in so genannten Superblöcken, eine hierarchische Gliederung des Gesamtmodells zu erhalten. Durch gezieltes Abspeichern getesteter und bewährter Systemteile lassen sich außerdem eigene qualitativ gesicherte Bibliothekselemente erzeugen und damit der Grundbestand des Funktionsumfangs beliebig erweitern.

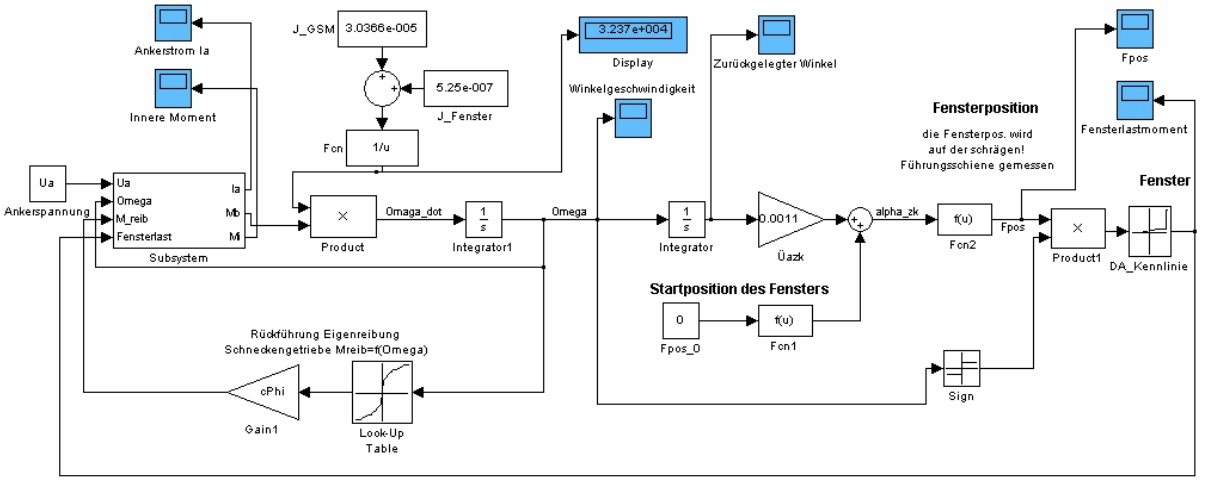

Abbildung 3.28: Beispiel eines Blockdiagramms (MATLAB/Simulink)

Im obigen Beispiel (Abbildung 3.28) ist das Modell eines Fensterhebers im Automobil dargestellt, der in Abhängigkeit der Eingangspannung U_a simuliert werden kann. Das Modell besteht aus zwei Sub-Modellen, einer Gleichstrommaschine (Antriebsmotor) auf der linken Seite und dem Fenster, dem mechanischen Teil, auf der rechten Seite. Die Kopplung zwischen beiden Sub-Modellen geschieht über die Winkelgeschwindigkeit (ω: Omega). Die Bewegung beeinflussende Größen, wie z.B. die Reibung des Schneckengetriebes, werden über ein Kennfeld und einen Verstärker nur innerhalb der Gleichstrommaschine zurückgekoppelt. Da sich die Position des Fensters ebenfalls auf den Motor auswirkt (Erhöhung des Ankerstroms bei Widerstand), ist hier ebenfalls eine Rückkopplung auf den Motor nötig. Zur Anzeige von verschiedenen Variablen (Ankerstrom, Omega, Position, usw.) sind im oberen Teil des Bildes Ausgabeinstrumente mit dem Modell verknüpft.

Die meisten der CASE-Tools zur Erstellung und Simulation kontinuierlicher Vorgänge sind nicht auf elektrische Modelle beschränkt. Es können damit auch mechanische, pneumatische, usw. Systeme modelliert, simuliert und analysiert werden, deren Ausgänge sich mit der Zeit ändern. Um dies auf einem Rechner berechnen zu können, muss das Signalflussbild in eine Zustandsraumdarstellung transformiert und unter Verwendung von Methoden zur Numerischen Integration aufgelöst werden.

3.9.2.3 Code-Erzeugung

Steht eine funktionsfähige und getestete Umsetzung der Spezifikation als grafisches Modell zur Verfügung, kann der in den CASE-Tools eingebaute Coder daraus automatisch einen äquivalenten ANSI-C-Code erzeugen. Bei der Verwendung des Real-Time-Workshops (RTW) von Mathworks [Math04] werden Dateien für den Einsatz in Echtzeit-Anwendungen generiert, die neben der Beschreibung des Zustandsdiagramms weitere Funktionen zur Initialisierung und Überwachung des Modells enthalten. Diese können dann in die Entwicklungsumgebung des Targets, d.h. des für die Ausführung des Codes verwendeten Prozessors, eingebunden und compiliert werden. Nach der Definition der Ziel-Plattform wird innerhalb des StateFlow-Editors das Zustandsmodell in einen C-Quellcode umgewandelt. Die einzelnen Elemente der Statecharts werden dazu in Programmstrukturen übersetzt, die sowohl die Zustände als auch deren Übergänge enthalten.

3.10 Rapid-Prototyping

Unter einem Prototypen wird in der Technik im Allgemeinen ein Produkt verstanden, dessen Entwicklungsstand zwischen Entwurf und Massenherstellung liegt. Er stellt eine erste funktionsfähige Einheit dar, die getestet werden kann. Um Aussagen über das fertige Produkt ableiten zu können, muss sichergestellt sein, dass der Prototyp ein möglichst genaues Abbild des fertigen Produktes ist. Während eine weitest gehende Verhaltensübereinstimmung zwischen Prototyp und fertigem Produkt erzielt werden soll, gibt es bei der Erstellung Unterschiede. Prototyp und fertiges Produkt können sich in der Beschreibungsart, Programmcodegröße, Ein-/Ausgabehardware und äußerem Design unterscheiden. Prototypen werden zielgerichtet auf Basis festgelegter Eigenschaften in Form eines Anforderungskatalogs für die Ermöglichung bestimmter Untersuchungen entwickelt [Fisc88].

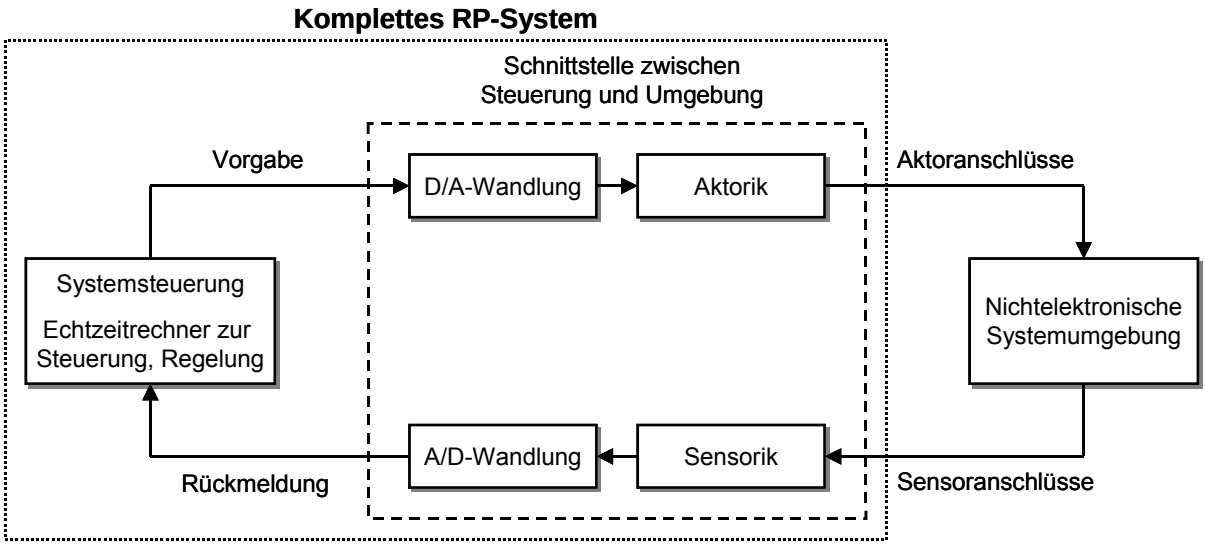

Abbildung 3.29: Aufbau eines Rapid-Prototyping Systems

3.10.1 Arten des Rapid Prototyping

Im Bereich des Echtzeitsystementwurfs wird unter dem Begriff Rapid Prototyping eine Art der Prototypenerzeugung verstanden, bei der bereits in den ersten Entwurfsphasen ein Prototyp zur konzeptionellen Überprüfung elektronischer Systeme hergestellt wird. Durch Rapid Prototyping wird ein Einsatz der erstellten Modellierung in der realen Umgebung ermöglicht. Die frühzeitige konzeptionelle Überprüfung führt im Allgemeinen zu einer Verkürzung der Entwicklungszeiten bei wachsender Konzeptsicherheit. Die Prototypen können hinsichtlich ihrer Funktionalität und Leistungsfähigkeit untersucht werden und erlauben somit schnelle Rückschlüsse auf die Richtigkeit und Vollständigkeit der Spezifikation. Zusätzlich wird die Entwicklung verschiedener Varianten und der Vergleich unterschiedlicher Ansätze stark vereinfacht und ein schnelles, iteratives Vorgehen beim Entwicklungsprozess ermöglicht. Mit Rapid Prototyping findet ein Übergang von der abstrakten Simulation der Systemfunktionalität zu einem Test unter Einbeziehung der realen Umgebung und Zeitanforderungen statt. Rapid Prototyping unterstützt dabei die folgenden Möglichkeiten:

- Automatische Abbildung der Verhaltensspezifikation in ein ausführbares Programm (Generierung eines Software-Prototypen).
- Bereitstellung einer frei konfigurierbaren Hardwareplattform zur Ausführung des Prototypen.

- Bereitstellung von Analyse- und Messverfahren zur Untersuchung des zeitlichen Verhaltens des generierten Prototypen.
- Bestimmung der Grenze zwischen Soft- und Hardware, d.h. nach Feststellung z.B. des zeitlichen oder energetischen Verhaltens können Systemteile, die in Software implementiert waren, in Hardware implementiert werden und umgekehrt (Hardware/Software-Co-Design).

Im Gegensatz zur Simulation der Systemfunktionalität, die in der Regel nicht in der realen Zeit abgearbeitet wird, muss bei einer Einbettung des Prototypen in der realen Umgebung außer der Systemfunktionalität auch die Echtzeitfähigkeit gewährleistet sein. Durch entsprechende Analysemethoden kann das zeitliche Verhalten des Prototyps überprüft werden. Stimmen sowohl Funktionalität als auch zeitliches Verhalten mit der geforderten Spezifikation überein, kann der Entwickler in den nächsten Schritten eine weitere Verfeinerung des Prototypen vornehmen. Können Funktionalität oder zeitliches Verhalten nicht gewährleistet werden, muss ein Rücksprung zur Spezifikationsphase vollzogen werden. Bei Rapid Prototyping entfällt im Gegensatz zur Simulation die Notwendigkeit, die Systemumgebung für vollständige Betrachtungen zu modellieren. Diese Modellierung der Systemumgebung kann unter Umständen (beispielsweise bei der Modellierung des Kaltlaufverhaltens seines Verbrennungsmotors) sehr umfangreich werden und deswegen zu einer vereinfachten Modellierung führen oder in Extremfällen überhaupt nicht erstellbar sein. Insbesondere unvorhergesehene Störeinflüsse der realen Umgebung und Aspekte des Echtzeitverhaltens bleiben in einer Simulation unberücksichtigt. Durch die iterative Vervollkommnung eines Prototypen, die durch Akzeptanz von Entwickler und Auftraggeber abgeschlossen werden kann, können auch mehrere Arten eines Prototypenaufbaus klassifiziert werden. In der Literatur existieren mehrere Varianten, von denen sich jedoch nur zwei Ansätze durchgesetzt haben. Der Ansatz nach Watkins untergliedert den Prototypenaufbau in fünf Arten [CoHu93]:

- **Exploratory Prototyping:** Diese Form beschreibt einen Prototyp, der in der Spezifikationsphase zu einem Zeitpunkt entwickelt wird, an dem Entwickler und Auftraggeber die Systemziele festlegen können.
- **Solution Prototyping:** Hierbei wird eine Evaluierung von Systemanforderungen vorgenommen, die sowohl für Funktions- als auch für Performanzuntersuchungen verwendet werden können. Dabei werden Animations- und Simulationstechniken angewendet.
- **Investigative Prototyping:** Hierbei wird eine Evaluierung von alternativen Lösungswegen vorgenommen, die sowohl Hardware- als auch Softwareaspekte berücksichtigt. Dabei werden oftmals Simulationstechniken eingesetzt.
- **Verification Prototyping:** Diese Form verwendet formale, mathematische Methoden, um die Korrektheit von Software- und Hardwareentwürfen nachzuweisen.
- **Evaluation Prototyping:** Hierbei werden in der tatsächlichen Systemumgebung funktionierende Prototypen eingesetzt, um Performanz, Benutzerakzeptanz und die Auswirkung von Systemänderungen testen zu können.

3.10.2 Phasen des Rapid Prototyping

Beim VP-Modell werden drei Phasen des Rapid Prototyping unterschieden: das konzeptorientierte, das architektur-orientierte, sowie das implementierungs- oder realisierungsorientierte Rapid Prototyping. Da Zielsetzung und zeitliches Auftreten der drei Ansätze unterschiedlich sind, wird jede Form des Rapid Prototypings in bestimmten Phasen des Systementwurfs eingesetzt. Die drei Formen stehen somit nicht in Konkurrenz zueinander, sondern ergänzen sich.

3.10.2.1 Konzept-orientiertes RP

Auf der höchsten Ebene des VP-Modells wird das konzept-orientierte Rapid Prototyping eingesetzt und dient hauptsächlich der Klärung der Systemziele. Es repräsentiert dabei die schnelle Umwandlung einer ausführbaren Spezifikation mittels CASE-Werkzeuge wie z.B. MATRIX$_X$ oder STATEMATE in einen funktionellen Prototyp. Die CASE-Werkzeuge können nach einer Prüfung auf Konsistenz und Vollständigkeit aus der Verhaltensbeschreibung des Modells Programmcode (z.B. C, C++, Ada, VHDL-AMS, etc.) generieren. Der konfigurierte und kompilierte Software-Prototyp wird danach auf einer geeigneten Zielplattform zur Ausführung gebracht. Die Hardware des Rapid Prototyping-Systems besteht aus einem leistungsfähigen Rechner (Zielplattform), der mit geeigneten Schnittstellen zur realen Umgebung ausgestattet ist. Die Schnittstellen müssen dabei einen weiten Signalbereich abdecken können und modular erweiterbar sein. Die Kosten eines konzept-orientierten Rapid Prototyping Systems sind nicht kritisch, da die Hardware für den Aufbau von weiteren Prototypen verwendet werden kann und nicht auf einen Prototyp begrenzt ist. Beim konzept-orientierten Rapid Prototyping steht ein schneller Systemaufbau im Vordergrund. Stellt man bei der Ausführung des Systemmodells ein Fehlverhalten fest, können in den CASE-Werkzeugen Änderungen der Spezifikation vorgenommen und innerhalb weniger Minuten das geänderte Modell erneut getestet werden. Auch die hardwareseitige Anpassung an die Systemumgebung muss unterstützt werden. Deswegen sollten die Ein-/Ausgabeschnittstellen vom Bildschirm aus konfigurierbar sein und Ein-/Ausgabehardware für die benötigten Signalbereiche vorliegen. Die Antwortzeiten heutiger Rapid Prototyping Systeme liegen im Bereich weniger Millisekunden und darunter. Damit eignet sich konzept-orientiertes Rapid Prototyping bereits für eine Vielzahl von Anwendungen, beispielsweise für Entwicklung und Test von Innenraumelektronik oder Getriebesteuerungen bei Automobilen.

3.10.2.2 Architektur-orientiertes RP

Im Gegensatz zum konzept-orientierten Rapid Prototyping wird beim architektur-orientierten Rapid Prototyping bereits die Zielarchitektur des Systems berücksichtigt, um genauere Aussagen über die endgültige Leistungsfähigkeit eines Systems vornehmen zu können. Einige Komponenten des Systems wurden bereits entwickelt oder mittels Standardkomponenten realisiert, während andere Teile sich noch im Test befinden. Dabei können für die Evaluation Prozessor- oder Mikrocontrollerboards eingesetzt werden, die über Schnittstellen mit der realen Umgebung kommunizieren und mit zusätzlicher meist programmierbarer Hardware ausgestattet sind. Die Softwarekomponenten, die auf den Prozessor- oder Mikrocontroller-Boards eingesetzt werden, werden bereits auf schnelle Reaktionszeiten und geringen Speicherbedarf hin optimiert, um den Anforderungen des Zielsystems zu entsprechen. Beim Rapid Prototyping von Prozessoren oder Prozessorkomponenten werden auch mehrfach programmierbare Hardwareplattformen als Basis für Prototypen eingesetzt, wobei meist eine Anordnung von programmierbaren Bausteinen benutzt wird, beispielsweise ein Array von FPGAs (Field Programmable Gate Arrays). Gemäß den Leistungsanforderungen des Systems findet eine erste Partitionierung und Abbildung des Prototyps auf die programmierbaren Bausteine statt. Ein Problem stellt dabei die bislang sehr geringe Ausnutzung der programmierbaren Bausteine dar, die bei etwa 10% liegt [Turn99]. Das Systemmodell wird bei diesen Systemen meistens in Form einer Hardware-Beschreibungssprache erstellt, z.B. in VHDL oder Verilog. Ebenso ist die Verwendung von Programmiersprachen in Verbindung mit einem Mikrocontroller oder digitalen Signalprozessor möglich. Die Verhaltensbeschreibung kann entweder per Hand oder mittels einer grafischen Spezifikation von CASE-Werkzeugen erzeugt werden. Die Ein-/Ausgabeschnittstellen, die beim konzept-orientierten Rapid Prototyping eingesetzt wurden, können unter Umständen zur Anbindung des Prototypen an die Systemumgebung verwendet werden. Das architektur-orientierte Rapid Prototyping erreicht nicht die kurzen Änderungszyklen des konzept-orientierten Rapid Prototypings, da die Partitionierung und Synthese des Modells wesentlich mehr Zeit in Anspruch nehmen. Somit ist dieser Ansatz weniger für eine Klärung der Systemziele geeignet, sondern dient der Überprüfung, ob mit der beabsichtigten Zielplattform die Leistungsanforderungen erfüllt werden können. Insbesondere bei den programmierbaren Bausteinen kann diese Überprüfung nur

eine erste Abschätzung darstellen. Durch die implementierungs-näheren Reaktionszeiten und die größere Nähe zur endgültigen Realisierung besitzen die architektur-orientierten Prototypen eine höhere Aussagekraft über das zeitliche Verhalten des zu entwickelnden Systems im Vergleich zum konzept-orientierten Rapid Prototyping. Dies wird um den Preis von längeren Änderungszyklen und einer stärkeren Bindung an den zu erstellenden Prototypen erreicht.

3.10.2.3 Realisierungs-/Implementierungs-orientiertes RP

Beim implementierungs-orientierten Rapid Prototyping werden leistungsfähige, hochspezialisierte Prototypen erzeugt, die auf den expliziten Anwendungsfall zugeschnitten sind und meist keine Wiederverwendung erlauben. Prototypen dieser Art dienen der genauen Analyse der Leistungsfähigkeit eines Systems und bestehen oft aus nahe an der endgültigen Systemrealisierung stehenden Prototypen oder bereits vorhandenen Systemen, die um neue Teile erweitert wurden [WeRG95]. Ein Prototypenaufbau auf dieser Ebene ist bereits seit vielen Jahren ein fester Bestandteil des Systementwurfs. Dabei müssen die Systemziele und die Systemstruktur bereits festliegen und Fragen der Realisierung und des Tests im Vordergrund stehen. Der zeitliche Aufwand zur Generierung von implementierungs-orientierten Prototypen ist vergleichsweise hoch. Ein Spezifikationsfehler in dieser Phase verursacht bereits häufig einen erheblichen Änderungsaufwand des Prototypen.

3.10.3 Schnittstellen für Rapid Prototyping

Rapid Prototyping Systeme müssen in der Lage sein, einerseits alle Signale, die in der Zielumgebung auftreten können, zu verarbeiten und andererseits die Aktoren der Zielumgebung anzusteuern. Damit dies möglich ist, werden verschiedene Schnittstellen benötigt. Die folgende Tabelle gibt einen Überblick über gängige, im heutigen Automobilumfeld benötigten und eingesetzten Schnittstellen, Treiber und Signalwandler.

Schnittstelle	Art	Eigenschaften und Anforderungen
Analog	In	12bit, 0-5V, Sample-Rate 250kHz (800kHz max.)
		16bit, ±5V oder ±10V, 15kHz max., 0-70°C
	Out	12bit, 0-4.5V, Refresh-Rate 250kHz (800kHz max.), 5 mA
		16bit, ±5V; ±10V oder 0V – 10V, 5 mA, 250kHz, 0-70°C
Digital	In	TTL-Pegel
	Out	TTL-Pegel, 5 mA
	PWM In	Frequenz oder PWM, Pulsweitenmessung, Frequenz 0.5Hz – 100kHz, Duty Cycle 0 – 100%, 16bit
	PWM Out	Frequenz 0.5Hz – 100kHz, Duty Cycle 0 – 100%, 16bit
Interface	CAN	2-4 Kanäle, 125kBaud – 1MBaud
	LIN	1-2 Kanäle, 20kBaud
	FlexRay	1-2 Kanäle
	RS232	1 Kanal
	LAN	100Mb/s

Tabelle 3.8: Schnittstellen für Rapid Prototyping

Um automobilspezifische Spannungen und Ströme an den Schnittstellen zwischen dem Rapid Prototyping System und dem Prozess bereitstellen zu können, ist teilweise Anpassung an die verwendeten Sensoren, insbesondere aber an die Aktoren erforderlich. Die Anforderungen an die Schnittstellensignale sind sehr vielfältig: Signale mit Spannungen bis 42 V und digitale Hochleistungssignale mit Strömen bis zu 40 A, digitale Eingangssignale mit veränderlichen Schwellspannungen, analoge Eingänge zur Messung von Spannungs-, Strom- und

Widerstandsänderungen sowie pulsweitenmodulierte Ausgangssignale. Tabelle 3.9 enthält eine Übersicht über übliche Leistungs- und Signalkonditionierungsstufen.

Schnittstelle	Art	Eigenschaften und Parameter
Leistungs-stufen	Vollbrücke	5A @ 40V (kontinuierlich)
	LowSide Driver	2A/5A @ 37V, Schaltzeit < 10µs, Strommessung optional
	HighSide Driver	1A/5A @ 36V, Schaltzeit < 30µs, Strommessung optional
Signalkondi-tionierung	Sensor	2.5V – 20V (einstellbar), Leistungsabgabe 1.2W, 1% Genauigkeit
	Analog In	Spannungsbereiche: ±100mV - ±50V, differentiell, Tiefpassfilter (10Hz, 50Hz, 100Hz, 1kHz), Pull-Up, Pull-Down, ...
	Digital In	progr. Schwelle, einstellbarer Bereich ±39.5V, ±60V max.

Tabelle 3.9: Leistungsstufen und Signalkonditionierung für Rapid Prototyping

3.10.4 Stand der Technik und der Forschung

Auf dem Markt finden sich mehrere kommerzielle Rapid Prototyping-Systeme, teilweise von großen Firmen, die sich in bestimmten Industriezweigen etabliert haben, aber auch von kleinen und mittelständischen Unternehmen. Letztere bedienen mit ihren Produkten meistens nur Marktnischen und arbeiten mitunter als Zulieferer und Sub-System-Entwickler der großen Anbieter und sollen daher nicht näher betrachtet werden.

In der Automobil-Branche sind es vor allem die Firmen dSPACE und ETAS, die Systeme für Rapid Prototyping anbieten. Diese Systeme bestehen dabei aus einer Software, die in Kombination mit der entsprechend darauf zugeschnittenen Hardware, anwenderprogrammierbare Funktionen ausführen kann. Die Software-Entwicklung in diesem Bereich wird weitgehend durch CASE-Tools unterstützt und mit Hilfe Tool-seitiger oder – im Falle von dSPACE – eigener Code-Generatoren und Download-Tools automatisiert. Auch die erwähnten CASE-Tools sind vorrangig Eigenprodukte der jeweiligen Firmen und werden fast ausschließlich durch MATLAB/Simulink und /StateFlow vom Mathworks ergänzt. Im Folgenden wird auf einige relevante Produkte beider Anbieter detailliert eingegangen.

3.10.4.1 ETAS GmbH

Die Firma ETAS GmbH [ETAS06] mit Sitz in Stuttgart bietet mit der ASCET-Software-Familie eine universelle und flexible Design-Umgebung für eingebettete Systeme. ASCET wird dabei für die modellbasierte Entwicklung von Applikations-Software für Steuergeräte in der Automobilindustrie eingesetzt. Mitglieder der Familie sind ASCET-MD (Modeling & Design), ASCET-RP (Rapid Prototyping) und ASCET-SE (Software Engineering), die wie oben bereits erwähnt, durch MATLAB ergänzt werden. Die genannten Software-Produkte unterstützen eine Vielfalt an Rapid Prototyping-Hardware, die speziell für den Einsatz im Fahrzeug ausgelegt sind. Dazu zählen das ES1000 RP-System, in das die verschiedenen I/O-Komponenten als Karten eingesteckt werden können. Verfügbar sind dafür neben dem System-Controller verschiedene Automotive-Busschnittstellen, digitale und analoge I/O-Interfaces, sowie Zählermodule.

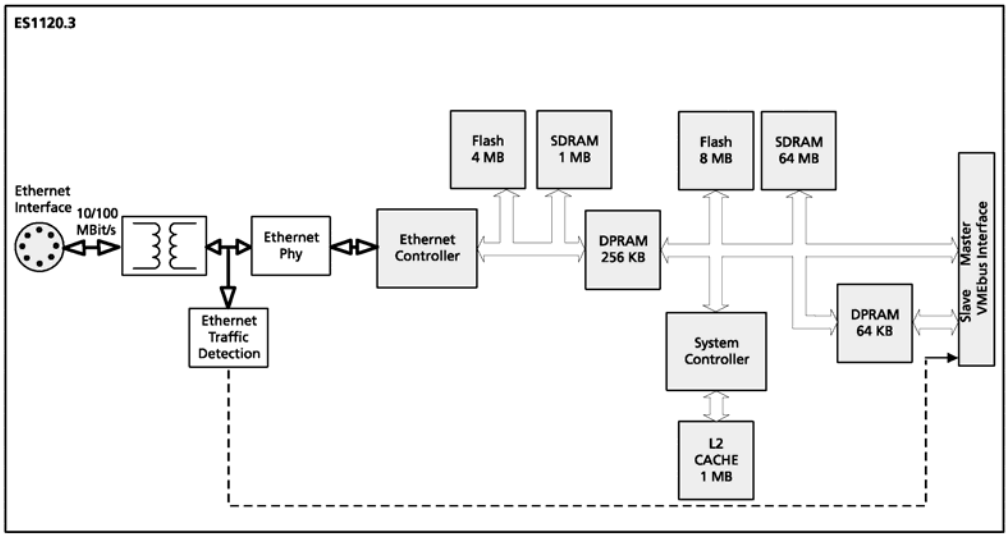

Abbildung 3.30: Blockdiagramm des SystemControllersES1120.3 von ETAS

Abbildung 3.31: Rapid Prototyping System ES1000 von ETAS

3.10.4.2 dSPACE GmbH

Ähnlich wie bei ETAS sieht auch die Produktpalette bei dSPACE [dSPA06] aus, wobei sich deren Ausrichtung in den letzten Jahren nicht mehr auf die Automobilindustrie alleine beschränkt, sondern auch in Bereiche wie Luftfahrt, Schienenverkehrstechnik und Robotik vorgedrungen ist. dSPACE vertreibt mit dem ControlDesk eine Automatisierungs-Software, die den gesamten Prozess der Funktionsentwicklung als Simulation oder als Implementierung auf dem Steuergerät kontrollieren und beobachten kann. Auch hier werden MATLAB/Simulink und MATLAB/Stateflow integriert und sowohl durch eine erweiterte Funktionsbibliothek als auch eine Unterstützung für Hardware-Zugriffe ergänzt. Die dazu passende Rapid-Prototyping-Hardware nennt sich AutoBox oder in noch etwas geschrumpfter Form, MicroAutoBox [dSPA05]. In die AutoBox können, ähnlich der ETAS-Variante, verschiedene Module in Form von Karten eingesteckt werden, die über einen internen Bus kommunizieren (Abbildung 3.33). Die Typen umfassen auch hier vornehmlich digitale und analoge Ein-/Ausgabe, sowie weitere Encoder, Decoder und die gängigen Automotive-Bussysteme.

Abbildung 3.32: Rapid Prototyping-System MicroAutoBox von dSPACE

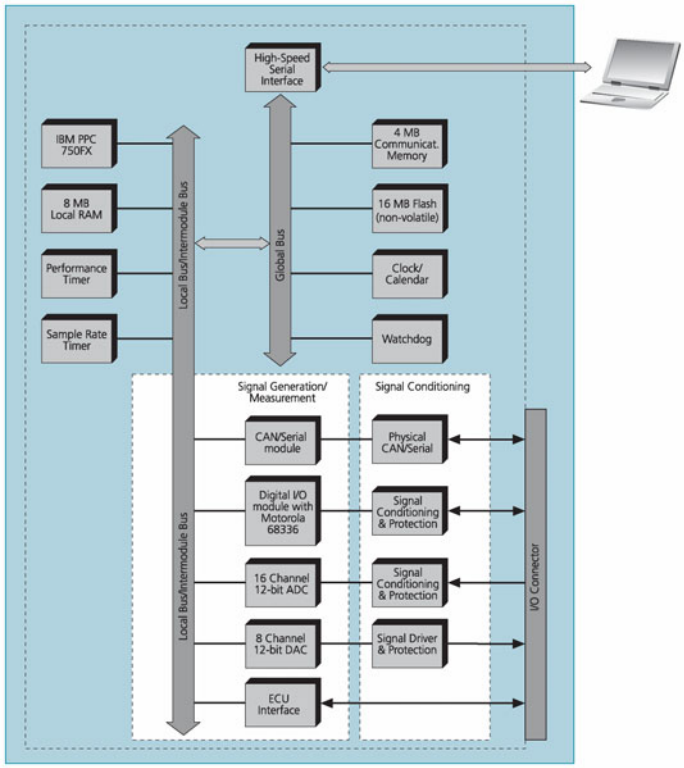

Abbildung 3.33: Blockdiagramm der MicroAutoBox von dSPACE

Neuerdings ist neben den aufgeführten Produkten auch eine RapidPro Hardware verfügbar, die als Signalkonditionierung, als intelligentes I/O-Subsystem oder auch als Stand-Alone-System eingesetzt werden kann. RapidPro ist als System stapelbarer Karten aufgebaut und kann ebenfalls nur mit dedizierten I/O-Karten erweitert werden.

3.10.4.3 Forschung

Rapid Prototyping (RP) wird in verschiedenen Bereichen der Elektronik und der Elektrotechnik betrieben und ist daher auch Gegenstand unterschiedlicher Forschungsthemen. Eine Auswahl an Arbeiten und Themen ist hier zusammengestellt, mit der die Abgrenzung gegenüber den in dieser Dissertation beschriebenen Inhalten erfolgt. Die Arbeiten sind nach Themen sortiert im Anschluss aufgelistet.

Im relativ stark vertretenen Bereich der Leistungselektronik werden Plattformen zur Entwicklung von elektronischen Ansteuerungen für Hochleistungs-Converter-Stufen [SmGi05], [KBBD05] und Gleichrichter [RSJB06] oder deren Test [MCPD06] untersucht. Hierbei kommt neben speziellen Hardware-Produkten MATLAB/Simulink zur Funktionsimplementierung zum Einsatz.

Auf dem Gebiet der Robotik wird Rapid Prototyping von [BoIS06] zur modellbasierten Steuerung für einen Roboterantrieb verwendet.

System-on-Chip-Systeme lassen sich recht einfach durch den Einsatz von FPGAs evaluieren und testen, bevor sie in die endgültige Fertigung gehen. Während sich [LiYM06] mit der Implementierung eines eigenen Controllers beschäftigt, werden in [SJSW05] und [MaCo04] bekannte Standard-Prozessoren als IPs in entsprechenden Umgebungen auf dem FPGA getestet.

Fast untrennbar damit verbunden sind Themen aus der digitalen Signalverarbeitung, wie z.B. [BaKA05] oder [BeSv06], die sowohl die Bereitstellung von Methoden zum schnellen Testen, als auch die Implementierungen auf dem Target selbst betrachten. [SMCT05] hebt dabei speziell auf die Video- und Multimedia-Applikationsentwicklung ab.

Im recht schnell wachsenden Bereich des Ubiquitious-Computing wird RP zur Entwicklung kontextbasierter Systeme[BCCG06] und für Informationssysteme mit LCDs [FCK+05] eingesetzt. [DLHS05] geht auf die Vorteile des RP in diesem Umfeld allgemein ein.

[Rupp05], [KSSK06] und [JaMI05] beschäftigen sich mit Themen aus der Kommunikationstechnik und Nachrichtenübertragung, speziell im Hinblick auf die Simulation von Antennen und Funkkanälen (z.B. WLAN und DAB).

Die Entwicklung von Hardware-Plattformen für die Ausführung von RP-Szenarien sind Gegenstand der Betrachtung in [LiYa06], [SiRB06] und [FBCH04]. Letzteres ist speziell auf den Test von ICs (Integrierten Schaltungen) ausgelegt. Alle verwenden einen Prozessor und einen daran angeschlossenen FPGA für dedizierte Anpassungen. Die Programmierung des FPGA wird dabei händisch umgesetzt, d.h. ohne Verwendung von CASE-Tools vorgenommen.

Arbeiten im Automotive-Sektor, der auch in dieser Dissertation die Basis der Betrachtung bildet, werden unter anderem von [AlBl06], [KJD+06] und [LeKJ06] verfolgt. Deren Inhalte beziehen sich hauptsächlich auf die Simulation oder Emulation des Kfz-Motormanagements. In [KSJN04] wird das Rapid Prototyping von elektronischen Antrieben untersucht. Zum Einsatz kommen hier meist kommerzielle Standard-Tools, z.B. von dSPACE, und Software-Werkzeuge wie SABER und MATLAB/Simulink von Mathworks zur Modellierung.

3.10.5 Bewertung Rapid-Prototyping-Systeme

Alle Rapid Prototyping Systeme haben inzwischen einen guten Reifegrad erreicht und werden in der Industrie umfassend eingesetzt.

Die Hersteller verwenden in ihrer derzeitigen Form einen klassischen Aufbau ihrer Hardware-Systeme. Dabei wurde auf die traditionelle Busarchitektur innerhalb der Systeme gesetzt, in der nur ein Bussystem zur Verfügung steht, über das Daten transportiert werden können. Der Datenverkehr muss außerdem vom Prozessor geregelt werden, da er Master des Systems ist. Eine breite Flexibilität der Plattformen hinsichtlich Struktur der Hardware, der Systemarchitektur oder dem Zusammenwirken von Systemteilen, die eine Ausbildung von Subsystemen oder eine prozessorunabhängige Kommunikation der einzelnen Komponenten oder Karten untereinander unterstützt, ist somit nicht möglich. Intelligente Subsysteme – wie z.B. bei dSPACE – können nur durch externe, zusätzliche Systemteile abgebildet werden, die über serielle Links angebunden sind.

Letzteres ist insbesondere bei der schnellen Funktionsentwicklung für Steuergeräte interessant, bei denen ein feldprogrammierbarer Logikbaustein (FPGA) anstelle eines Mikrocontrollers als Hauptbestanteil des Systems zum Einsatz kommt [Voll06]. Dieser Bereich ist in den derzeitigen RP-System völlig unterrepräsentiert, so dass die von den Plattformen dafür bereitgestellten Ressourcen nur sehr unzureichend, wenn überhaupt, vorhanden sind. Rapid Prototyping Systeme, die solche Nischen adressieren sind zwar am Markt erhältlich (z.B. [Deco05], [XPCT05], [Just04], [MiXi06]), allerdings eher als Insellösungen zu bezeichnen. Sie umfassen meist genau eine Platine und bieten neben einem FPGA mit fester Größe (fester Gatteranzahl) einige Schnittstellen, jedoch sind keinerlei Vorkehrungen zum Anschluss von Treibern und Signalkonditionierungsstufen zur Prozesskopplung gegeben. Darüber hinaus beschränkt sich die Breite der verfügbaren Tools zur Instrumentierung der Plattform auf die Werkzeugkette des FPGA-Herstellers. Die Einbindung in den nach dem V-Modell gegebenen Entwicklungsprozess ist damit nur sehr schwer möglich und erfordert darüber hinaus spezielle Kenntnisse des Anwenders.

Hinsichtlich der Flexibilität der I/Os ist weiterer Nachholbedarf seitens der gängigen Plattformen zu erkennen. So werden lediglich dedizierten Hardware-Schnittstellen (Digital-I/O,

PWM I/O, Timer, Counter, Analog-I/O, ...) als Steckkarten angeboten, die dem Benutzer zur schnellen Implementierung von Ein-/Ausgabefunktionen Unterstützung leisten. Jedoch wird dem Wunsch nach vom Anwender frei konfigurierbaren Schnittstellen nur unzureichend Rechnung getragen. Sonderwünsche hinsichtlich einer kunden- bzw. anwendungsspezifischen Funktionsimplementierung oder -änderung müssen wie bei allen anderen Plattformen als zusätzliche und meist externe Platine speziell gefertigt werden.

3.11 Hardware-in-the-Loop

Unter dem Begriff Hardware-in-the-Loop (HiL) versteht man die Verbindung von realem Steuergerät und einer simulierten Umgebung. Das Ziel von Hardware-in-the-Loop besteht in der Überprüfung realer Steuergeräte auf Konformität mit ihrer Spezifikation. Die Überprüfung kann dabei beliebig oft wiederholt werden oder es können bestimmte Aspekte des Tests, z.B. eine Grenzsituation, herausgegriffen werden, ohne dass Zeitaufwand und Kosten eines realen Tests erreicht werden.

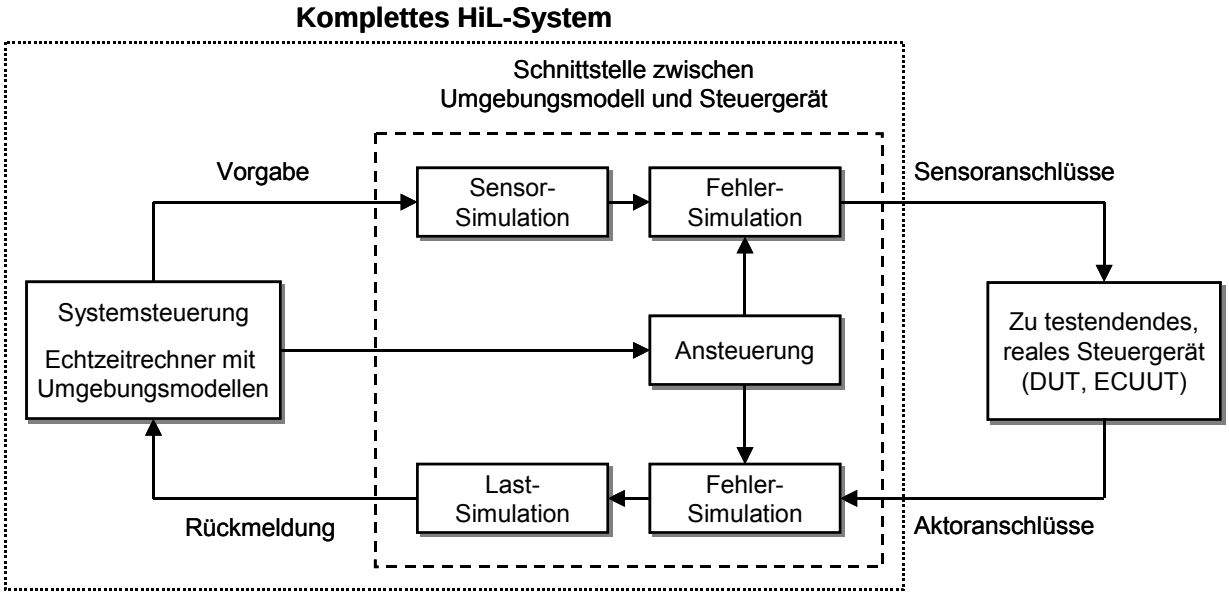

Abbildung 3.34: Hardware-in-the-Loop-Aufbau für elektronische Systeme

Abbildung 3.34 zeigt die allgemeine Struktur eines Hardware-in-the-Loop Aufbaus. Die Umgebung wird auf einem Echtzeitrechner mittels eines Modells nachgebildet, das zu testende Gerät (DUT = Device Under Test bzw. ECUUT = Electronic Control Unit Under Test) ist als reale Hardware vorhanden. Die Schnittstelle zwischen dem Umgebungsmodell und dem Steuergerät besteht dabei aus einer Simulation von Sensorik oder Last und einer anschließenden Fehlersimulation. Im Rahmen der Fehlersimulation werden gezielt verschiedene Fehlerarten wie Kurzschlüsse, Masseschlüsse, etc. zwischen Umgebungsmodell und Steuergerät simuliert, um die Reaktion des Steuergeräts im Fehlerfall testen zu können. Die Ansteuerung geschieht dabei über ein Ansteuerungsmodul, das eine Synchronisation von Echtzeit-Rechner und der Fehlersimulation zu Auswertungszwecken herstellt. Das Ansteuerungsmodul, sowie Sensor- und Fehler-Simulation können entweder als reine Software-Lösung oder mit Hilfe einer eigenen Hardware ausgeführt werden. In der Automobiltechnik werden Steuergeräte häufig parallel zu den mechanischen Teilen eines Automobils entwickelt. Damit kann es vorkommen, dass Steuergeräte zu einem Zeitpunkt getestet werden müssen, zu dem das Fahrzeug, also die Umgebung des Steuergeräts, noch nicht existiert. Um aussagekräftige Tests bei großen Systemen wie der Gesamtelektronik eines Automobils

vornehmen zu können, können dabei mehrere Steuergeräte gemeinsam getestet werden. Somit kann die Interaktion der Steuergeräte untereinander und ihr Verhalten in einer bestimmten Umgebung getestet werden. Diese erweiterte Form des Steuergerätetests wird auch als Integrationstest bezeichnet. Der Entwurfsablauf gestaltet sich bezüglich der Werkzeugverwendung ähnlich zu Rapid Prototyping, allerdings wird hier versucht, die Umgebung mittels eines mathematischen Modells nachzubilden. Im allgemeinen werden nur die für das Steuergerät entscheidenden Aspekte der Umgebung modelliert. Oftmals muss auch diese Modellierung noch vereinfacht werden, um die Echtzeitbedingungen für Rapid Prototyping einhalten zu können.

3.11.1 Schnittstellen für Hardware-in-the-Loop

Die oben beschriebenen Eigenschaften der Schnittstellen für Rapid Prototyping (Kapitel 3.10.3) stellen eine Obermenge der Ausgänge von Steuergeräten dar, so dass deren Leistungsfähigkeit als Maßstab für die Anforderungen den Sensor- und Aktorschnittstellen der Hardware-in-the-Loop-Systeme gelten kann. Zusätzlich zu den bereits erwähnten Typen und deren Eigenschaften sind weitere Schnittstellen nötig, um die Umgebung des zu testenden Systems (DUT, Device Under Test) mit einem HiL-Testsystems nachbilden zu können. In Tabelle 3.10 sind spezifische, für Hardware-in-the-Loop-Tests erforderlichen Schnittstellen zusammengestellt.

Schnittstelle	Art	Eigenschaften und Anforderungen
Digital	Kurbelwelle	0.011° Auflösung, ±29.000 rpm, 0.112 rpm Auflösung
	Widerstands-Simulator	10Ohm – 500kOhm, max. 15kHz, 12bit, max. ±12bit, max. ±20 mA Laststrom, 0°C – 70°C
	Inkremental-Geber	max. 1.25MHz, 24bit,
	Timer	30bit, 200ns Zeitbasis, 833kHz max. In/Out-Frequenz
	Waveform In	±10V, Sample-Rate: max. 40MHz, 25ns – 53.68s, 0°C – 70°C
	Waveform Out	±10V, Ausgabefrequenz: max. 2MHz, 250ns – 26s Pulsweiten, 0°C – 70°C

Tabelle 3.10: Spezifische Schnittstellen für Hardware-in-the-Loop

3.11.2 Stand der Technik und der Forschung

Im Umfeld der Hardware-in-the-Loop-Tests ist der Stand der Technik auch hier geprägt durch den Einsatz von graphischen Modellierungswerkzeugen und automatischer Codegenerierung. An der Schnittstelle zwischen dem Steuergerät und der Last werden identisch zu den Lösungen bei Rapid Prototyping dedizierte Karten eingesetzt, um den Signalaustausch sicherzustellen. Die am Markt verfügbaren Systeme der Firmen dSPACE, IPG und ETAS sind zwar traditionell auf die Anwendung im Automobilbereich ausgerichtet, werden aber auch in anderen Bereichen verwandt. Diverse weitere Systeme für den allgemeinen Einsatz sind neben den oben genannten erhältlich, allerdings würde eine Auflistung derer den hiesigen Rahmen sprengen.

Die Systeme von dSPACE und IPG verwenden für die Modellierung und Codegenerierung MATLAB Simulink und bieten Toolboxen als Erweiterungen zur Konfiguration der Hardware. ETAS setzt beim LabCar auf die eigene Software LabCar Developer, die auf den Modellierungswerkzeugen von ASCET basiert. Eine Schnittstelle zu MATLAB Simulink ist jedoch auch hier vorhanden.

3.11.2.1 dSPACE GmbH

Die Firma dSPACE [dSPA06] vertreibt mit dem dSPACE-Simulator ein skalierbares System zum Test von Automotive-Steuergeräten oder eingebetteten elektronischen Systemen im Allgemeinen. Eine Automatisierung der Abläufe kann mit dem Tool AutomationDesk erreicht werden. Der Simulator Mid-Size, eingebaut in einem 19“-Rack mit 6-12 Höheneinheiten für Hardware, eignet sich für kleinere Aufbauten, d.h. zur Simulation und dem Test von einzelnen Steuergeräten oder virtuellen Fahrzeugen. Die Auswahl an I/O-Interfaces in Form von Karten ist hier stark limitiert. Die größere Ausbaustufe des Simulators dahingegen bietet in ihrem 19“-Schrank 17-34 Höheneinheiten Platz. Sie kann für größere Projekte verwendet werden, in denen mehrere Steuergeräte bis hin zu einem gesamten Netzverbund getestet werden und entsprechend mehr I/O-Schnittstellen verfügbar sein müssen. Letzterer kann mit einer breiten Palette an Hardware-Karten aus dem dSPACE-Sortiment bestückt werden. Eine Multiprozessorarchitektur ist möglich, allerdings nur auf Basis zusteckbarer Prozessorkarten.

Abbildung 3.35: dSPACE HiL-Simulatoren Mid-Size (links) und Full-Size (rechts)

3.11.2.2 ETAS GmbH

Unter der Bezeichnung LabCar fasst ETAS [ETAS06] ihre Software-Familie inklusive der Hardware-Produkte für HiL-Tests zusammen. LabCar unterstützt die modellbasierte Simulation von Komponenten am PC ebenso wie den Test bereits als reale Hardware verfügbarer

Steuergeräte. Der automatisierte Test von Steuergeräten im LabCar wird mit dem Zusatzprodukt LabCar-Automation unterstützt. Hiermit lassen sich Abläufe automatisieren, strukturieren und zu Testprojekten zusammenfassen. Das Produkt LabCar Models enthält echtzeitfähige Fahrzeugmodelle für Closed-Loop-Anwendungen. Die LabCar-Hardware stellt die physikalische Verbindung zum Steuergerät her.

Die Hardware besteht im Wesentlichen aus dem LabCar System Base, das neben dem Gehäuse mit Spannungsversorgung und dem Hauptcontroller auch noch zwei CAN-Schnittstellen bietet. Das System Base ist erweiterbar durch eine zusätzliche Stromversorgung, Motorsteuergeräte, I/O-Schnittstellen, BreakOut-Box und eine Fehlersimulationseinheit. Außerdem stehen Module für die gängigen Ein-/Ausgabe-Signale als Einschubkartenlösungen zur Verfügung.

Abbildung 3.36: ETAS HiL-System LabCar (Serie 4100)

3.11.2.3 IPG Automotive GmbH

Die Firma IPG [IPG06] Automotive GmbH mit Sitz in Karlsruhe ist ebenfalls sehr stark im HiL-Testumfeld vertreten und hat sich dabei – wie im Namen auch erkennbar – auf den Automotive-Bereich spezialisiert.

Mit der CarMaker-Produkt-Familie bietet IPG ein Software-Environment an, das für die Dynamik-Simulation von Fahrzeugen am Rechner, aber auch für Hardware-, Software- und Model-in-the-Loop verwendet werden kann. Die reine Simulationsumgebung kann mit und ohne MATLAB/Simulink betrieben werden. Für den Hardware-in-the-Loop-Test mit realen Steuergeräten wird CarMaker/HiL eingesetzt. Es handelt sich dabei um ein HiL-Testsystem, das für die Überprüfung von ABS, ESP, ACC und CDC ausgerichtet ist. Auch hier ist eine Kopplung mit MATLAB/Simulink möglich.

Abbildung 3.37: HiL-System CarMaker von IPG

Zu CarMaker stehen weitere Software-Werkzeuge, wie z.B. IPG-Driver, IPG, Road und IPG-Kinematics, zur Verfügung, die verlinkt mit CarMaker oder als Stand-Alone Applikationen betrieben werden können. Failsafetester ist ein zusätzliches Hardware-Tool, mit dessen Hilfe software-programmierbar physikalische Fehler erzeugt werden können. Weitere Ergänzungen oder Lösungen in Form von Einschüben stehen nicht zur Verfügung.

3.11.2.4 Forschung

Hardware-in-the-Loop (HiL) wird ebenso wie Rapid Prototyping in verschiedenen Bereichen der Elektronik und der Elektrotechnik betrieben, ist aber als Gegenstand der Forschung vom Umfang insgesamt weniger stark vertreten. Eine Auswahl an Arbeiten und Themen ist hier zusammengestellt, mit der die Abgrenzung gegenüber den in dieser Dissertation beschriebenen Inhalten erfolgt. Die Arbeiten sind nach Themen sortiert im Folgenden aufgelistet.

Auf dem Gebiet der Leistungselektronik werden von [WLD+05], [WuMo05] und [WuLM04] HiL-Szenarien eingesetzt, um Netz- oder Motorstromrichter in Echtzeit zu simulieren oder wie im Falle von [Steu06], [ZDL+06] und [MFL+05] an realen Systemen zu emulieren.

Parallelitäten dazu gibt es auch bei der Leistungselektronik in der Flugzeugtechnik. [KaSe06] und [ZhJi06] untersuchen die Geräte zur Steuerung von Flugzeugen, während sich [KRM+06] mit Extremsituationen bei dem Betrieb von Weltraumflugkörpern beschäftigt.

Die vielfältigsten HiL-Tests kommen aus dem Automotive-Umfeld und decken sowohl den Bereich der reinen Simulation ([RBRC04] und [GVLO04]), als auch experimentelle Testszenarien ab. Die Anwendungen von [DJZ+05] und [ReVi06] sind sehr spezifisch und daher nur bedingt für allgemeines HiL einsetzbar. [ShPo05] dagegen beschreibt einen neuen Prüfstand, der durch den Test an einem realen Tempomat verifiziert wird. In [Brin05] wird eine neue Methodik für Komponententests präsentiert.

Sehr eng mit diesem Thema verknüpft sind Tests zur Verifikation der Verhaltensweisen elektrischer und elektronischer Fahrzeuge, wie z.B. [Sung05], [CaAl04] und [KiCo04].

Allgemeine Themen der Hardware-in-the-Loop-Simulation werden in [Baci05] betrachtet.

[LPQC05], [CGD+05] und [LuWM05] stellen jeweils kostengünstige Hardware-Plattformen zur Durchführung von HiL-Tests vor. Sie basieren auf Standardkomponenten und lassen wenig Spielraum für Modifikationen. [CGD+05] verwendet dazu eine PCMCIA-Karte von National Instruments, die in einem Laptop betrieben wird. Die verfügbaren Ein- und Ausgabeschnittstellen beschränken sich dabei allerdings auf wenige digitale und analoge I/Os, die aufgrund der Bauart der PCMCIA-Karte nicht erweiterbar sind. Der Einsatz des Systems ist außerdem auf die Erzeugung von Kurbelwellensignalen im Auto ausgerichtet. Als Basis für die HiL-Hardware in [LuWM05] und [LPQC05] wird jeweils ein Standard-PC verwendet, der im Falle [LuWM05] mit einer FPGA-Karte zur Umsetzung spezifischer Funktionen und zur Ankopplung an den Prozess ausgestattet ist. Die in [LPQC05] vorgestellte HiL-Umgebung setzt dagegen auf verschiedene Standard-PCI-I/O-Karten, die als Ein- und Ausgabe analoger und digitaler Signale dienen. Allen Systemen ist gemein, dass sie zur Funktion immer einen PC oder Laptop benötigen und nicht alleine (Stand-Alone) betrieben werden können, was sich nachteilig auf die Baugröße auswirkt. Die Verfügbarkeit an Schnittstellen ist sehr stark eingeschränkt und lässt sich – wenn überhaupt – nur in einem kleinen Rahmen erhöhen. Die zur Prozessankopplung benötigten Treiber für Strom und Spannung bzw. Signalkonditionierungskomponenten müssen ebenfalls extern angeschlossen werden, sind also nicht Bestandteil der vorgestellten HiL-Lösungen.

3.11.3 Bewertung der Hardware-in-the-Loop-Systeme

Grundsätzlich gelten hier die gleichen Einschränkungen der gängigen Plattformen, wie sie auch bereits in Kapitel 3.10.5 für die derzeitigen Rapid Prototyping Systeme ausführlich diskutiert wurden. Allerdings kommen im Bereich der Hardware-in-the-Loop-Testsysteme weitere Merkmale hinzu, da hier höhere Anforderungen an die zeitlichen Auflösungen gestellt werden. Dies betrifft zum einen die Bereitstellung von hochfrequenten Signalen als auch die Geschwindigkeit der digitalen Datenverarbeitung innerhalb der Plattform.

Da HiL-Tests auf der Nachbildung der Umgebung basieren, müssen meist mehrere Umgebungsmodelle (z.B. von Motoren, Druck- oder Temperaturgebern) parallel gerechnet werden, um eine möglichst gute Annäherung an die Realität zu erreichen. Für diese Parallelrechnung werden im Allgemeinen mehrere Prozessoren verwendet, die auf unterschiedlichen Steckkarten untergebracht sind. Daraus resultiert ein recht hoher Raumbedarf für die Hardware, sowie die Einschränkung der Implementierung der Signalverarbeitung als reine Software-Lösung. Eine günstigere Variante wäre jedoch eine Möglichkeit zur parallelen Modellrechnung vorzusehen, die möglichst ohne zusätzliche Ressourcen auskommt und außerdem die Wahl bietet, die Signalverarbeitung in Software oder in Hardware zu realisieren.

Im Zuge der Einführung neuer Kommunikationssysteme, wie z.B. FlexRay, die den Anforderungen an höhere Bandbreite und deterministisches Verhalten genügen, spielt auch die zeitliche Auflösung bzw. Auflösbarkeit von Signalen eine entscheidende Rolle. Die dazu nötigen Taktraten zur Generierung oder zur Abtastung von seriellen Bussignalen erreichen in der Regel mehrere 10MHz. Diese können zwar von speziellen Bustestern verarbeitet werden, jedoch nicht, oder nur mit sehr hohem Aufwand, von den üblichen HiL-Systemen. Um auch hier eine Reduktion der verschiedenen Komponenten zugunsten einer Integration in einer umfassenden Lösung zu erzielen, wäre eine Adaptierung wünschenswert.

3.12 Vergleich von Rapid Prototyping und Hardware-in-the-Loop

Grundsätzlich verfolgen Rapid Prototyping und Hardware-in-the-Loop – wie oben bereits ausführlich beschrieben – unterschiedliche Ziele und werden daher meist entsprechend den Phasen des V-Modells getrennt eingesetzt. Während sich Rapid Prototyping mehr für den linken Flügel des V-Modells, d.h. für die Verfeinerung der Entwurfsschritte eignet, wird Hardware-in-the-Loop eher auf der rechten Seite angewendet – zur Verifikation eines vollendeten Integrationsschritts.

Für beide Zielrichtungen sind mittlerweile graphische Modellierungs-Tools zum Standard der Anwendungs- bzw. Simulations-Software-Entwicklung geworden. Neben der Möglichkeit zur umfassenden Simulation, bieten die meisten Tools auch eine automatische Code-Generierung für verschiedene Targets an. Leider ist die Kopplung zwischen den einzelnen Modellierungs-Tools und Code-Generatoren noch nicht optimal, wodurch auf allgemeine Standardprodukte zurückgegriffen wird. Eines der in diesem Zusammenhang am meisten verwendeten oder in firmenspezifische Testumgebungen integrierten Werkzeuge ist MATLAB von Mathworks. Mit seinen beiden Varianten Simulink für die Modellierung kontinuierlicher und Stateflow für diskrete Systeme deckt es die Erfordernisse in der Automobilindustrie und auch in anderen Bereichen ab. Eine wichtige Unterscheidung zwischen RP und HiL bleibt trotz der Gemeinsamkeiten hinsichtlich des Tool-Einsatzes erhalten: Während bei RP die Funktion, also die eigentliche Firmware des Steuergerätes entwickelt wird, modelliert man bei HiL die Umgebung, in der das Steuergerät betrieben werden soll. Damit sind Modelle grundsätzlich zwar wieder verwertbar, aber immer nur in der eigenen Domäne.

Hinsichtlich der Rechnerplattform und der Ein-/Ausgabe-Hardware, die für Hardware-in-the-Loop verwendet werden, ergeben sich ebenfalls Parallelen zu den Bestandteilen eines Rapid Prototyping Systems, so dass die Komponenten im Grunde universell einsetzbar sind. Konzeptionell verwenden sowohl Hardware-in-the-Loop als auch Rapid Prototyping die gleiche leistungsfähige Rechenhardware zur Modellberechnung, bzw. eignen sich beide für eine verteilte Berechnung auf mehreren Prozessoren innerhalb eines Systems.

Die Modelle der Systemumgebung und des Prototypen haben typischer Weise die gleichen Anforderungen an die Geschwindigkeit der Signalverarbeitung. Da Rapid Prototyping bereits in sehr frühen Phasen des Systementwurfs stattfindet, kann häufig auf eine genaue Modellierung des Verhaltens zugunsten der Änderungshäufigkeit verzichtet werden. Eine zeitlich genaue Ansteuerung eines Aktors bzw. die Abtastung eines Sensors ist für das Endprodukt hingegen sehr wichtig, so dass beim Rapid Prototyping die Zeitanforderungen nicht ganz so hoch sind wie für den Hardware-in-the-Loop-Test.

3.12.1 Bewertung

Eine Kombination von Rapid Prototyping und Hardware-in-the-Loop macht gemäß der bereits angestellten Betrachtungen zunächst nur auf Basis einer gemeinsamen Hardware-Plattform Sinn. Die Ein-/Ausgabe-Hardware die für Hardware-in-the-Loop eingesetzt wird, ist entsprechend der Betrachtungen der Schnittstellen für RP und HiL (Kapitel 3.11.1 und Kapitel 3.11.1) auch für Rapid Prototyping verwendbar. Außerdem benötigen sowohl Hardware-in-the-Loop als auch Rapid Prototyping leistungsfähige Rechen-Hardware zur Modellberechnung. Jedoch muss bei der Implementierung darauf geachtet werden, dass für Charakterisierungen bei Hardware-in-the-Loop-Tests höhere Anforderungen an die Auflösung – insbesondere der zeitlichen – vorherrschen.

Außerdem kann hinsichtlich der Modellierungs- und Simulationswerkzeuge, sowie der verwendeten Software für Betrieb und Bedienung ein zusätzlicher Synergieeffekt durch Identifizierung weiterer Gemeinsamkeiten erzielt werden. Jedoch sind diese um so schwieriger zu finden, je weiter man sich in dem Abstraktionslevel von der Hardware in Richtung der beiden methodischen Ansätze entfernt. Die Modelle auf oberster Modellierungsebene sind

ob der unterschiedlichen Zielausrichtungen nicht ohne Änderung für RP und HiL gleichermaßen verwendbar.

Damit ergibt sich Hardware-seitig ein universell einsetzbares System, das konzeptionell zumindest auf den selben Modellierungs- und Simulationswerkzeugen basiert. Die einheitliche Plattformsteuerung und -bedienung reduziert damit außerdem die nötigen Tools und vermindert den Prozess der Einarbeitung.

3.13 Anforderungen

Aus den obigen Bewertungen der bestehenden Rapid-Prototyping- (Kapitel 3.10.5) und Hardware-in-the-Loop-Systeme (Kapitel 3.11.3), sowie aus dem Vergleich der beiden Domänen (Kapitel 3.12.1) lassen sich Eigenschaften als Basis der in dieser Arbeit beschriebenen konfigurierbaren Hardware-Testeinrichtung ableiten. Der weit überwiegende Teil bezieht sich dabei auf beide Domänen, d.h. sowohl auf RP, als auch auf HiL. Daraus ergeben sich Anforderungen an die Hardware, die Software, die Bedienung und die Systemsicherheit, die hier im Folgenden aufgelistet sind.

Hardware

- Verwendungsbereich: Das Testsystem soll als einheitliche Plattform auf beiden Ästen des V-Modells verwendet werden können, d.h. sowohl als Rapid Prototyping als auch als Hardware-in-the-Loop-System.
- Einsatzgebiet des Testsystems: Das Gesamtsystem soll sowohl im Labor, als auch direkt im Fahrzeug eingesetzt werden können.
- Auslegung des Temperaturbereichs: Da die Einsatzgebiete des Systems nicht auf das Labor beschränkt bleiben sollen, sind bei der Auswahl der Komponenten der strengere „Automotive“ oder „Industrial“ Temperaturbereich zu wählen (-40°C bis +125°C).
- Größe und Gewicht: Um eine portable Lösung zu erhalten, sollten die Ausmaße des Systems unterhalb von 400 × 300 × 250 mm^3 (L × B × H) liegen. Das Gewicht sollte dabei die Größenordnung von 15kg nicht übersteigen.
- Kosten des Systems: Das Testsystem ist als preiswerte Alternative zu den kommerziellen Geräten gedacht. Die Auswahl der Komponenten soll daher nach dem besten Preis/Leistungsverhältnis und der breiten Verfügbarkeit auf dem Markt geschehen. Die Gesamtkosten sollten einen Betrag von 5.000,- € nicht wesentlich übersteigen.
- Spannungsversorgung und Prozessadaption: Die Energieversorgung der Plattform sollte aus der oder den verfügbaren Spannungen (z.B. 12V – 14V im Automobil) ableitbar sein und als Referenzspannung für die Adaptierung der Signale vom oder zum Prozess dienen. Um eine räumliche Entkopplung zwischen Testsystem und Energieversorgung zu erreichen, ist eine Aufteilung in zwei Baugruppen sinnvoll. Der Energieaustausch geschieht über eine Kabelbrücke. Die Leistungsaufnahme sollte für ein portables System 100W nicht übersteigen, um keine besondere Kühlung vorsehen zu müssen.
- Steigerung der Flexibilität gegenüber derzeitigen, kommerziellen Lösungen:
    - Modularer Aufbau des Systems zur Skalierung des Systems hinsichtlich benötigter Ein- und Ausgabeschnittstellen
    - Einsatz von feldprogrammierbaren Schaltungen
    - Wiederverwendbarkeit der Implementierungen und der Art bzw. des Typs der Wiederverwendung (Soft-IP, Firm-IP oder Hard-IP)
    - Trennung von Logik und Prozesskopplung zur bedarfsgerechten Systemerstellung durch freie Zuordnung der I/O-Treiber zur digitalen Schnittstelle

  - Skalierbarkeit der Plattform hinsichtlich der benötigten Ein-/Ausgabeschnittstellen und der Rechenleistung
  - Möglichkeit zur Minimierung der Baugröße und der Leistungsaufnahme der Plattform durch bedarfsgerechte Verwendung von I/O-Schnittstellen
  - Aufgabe der klassischen Hardware-Architektur eines Single-Bus-Systems und Implementierung von prozessorunabhängigen Datenpfaden (z.B. Erstellung von Subsystemen)

- Speicherbedarf zur nicht-flüchtigen Ablage von Daten: Für die Abspeicherung von Messdaten gibt es keine Begrenzung nach oben, d.h. je größer der Speicher, desto besser und desto länger kann eine Messung ohne Unterbrechung ausgeführt werden. In kommerziellen Systemen werden nicht-flüchtige Messdatenspeicher nicht explizit zur Verfügung gestellt. Anstelle dessen sind Werte von 64 – 128 MB SDRAM für Applikation und Daten typisch [dSPA06]. Geht man davon aus, dass das Testsystem mobil sein soll, sind allerdings noch weitere Parameter, wie Gewicht, Energieaufnahme, Größe, usw. mit in die Betrachtung einzubeziehen.
- Erreichbare Zeitauflösungen: In Automobilumfeld sind Abtastraten von 10 ms bei Systemen, die für Komfort- und Innenraumfunktionen bestimmt sind. Bei der Motorsteuerung und bei Sicherheitssystemen werden geringere Zykluszeiten mit einer Dauer von 1 ms benötigt. Hinsichtlich der Signalabtastraten werden höhere Anforderungen an die Baugruppen gestellt. Diese bewegen sich derzeit im Bereich von 50ns – 25ns.
- Anzahl und Skalierbarkeit der I/O-Pins: Eine hohe Anzahl an I/O-Pins wird besonders für den Hardware-in-the-Loop-Test benötigt, da dort einzelne Steuergeräte oder gar ein ganzer Verbund getestet wird. Bei einem Steuergerät alleine sind dazu bis zu 50 Ein- und Ausgänge zu bedienen. Im Verbundtest, bei dem mehrere mit einander gekoppelte Steuergeräte geprüft werden, können so leicht Pinanzahlen von mehreren hundert entstehen. Im Falle des Rapid-Prototyping sind geringere Anzahlen ausreichend, etwa im Bereich von 20 – 30. Für eine tragbare Lösung muss die Anzahl jedoch konsequent soweit eingeschränkt werden, damit das System realisierbar bleibt. Mindestens 20 I/Os sollten aber bereitgestellt werden können.
- Funktionen von I/O-Schnittstellen zum Prozess: In Kapitel 3.10.3 und 3.11.1 sind die am häufigsten für RP und HiL benötigten Schnittstellen und deren Eigenschaften aufgeführt, die auch hier als Basis dienen sollen. Hinsichtlich der Treiberstärke sind auch hier Einschränkungen nötig, so dass der maximal lieferbare Strom pro Schnittstelle auf 1A begrenzt wird. Anforderungen an höhere Ströme bzw. Leistungen müssen mit externen Treibern realisiert werden, wie das auch bei kommerziellen Systemen gelöst wird.
- Digitale Signalverarbeitung: Für die Entwicklung von Systemen im Infotainment-Bereich sollten Ressourcen zur Umsetzung von DSP-Funktionen bereitgestellt werden, die alle auf der Plattform vorhandenen I/O- und Automotive-Schnittstellen mitbenutzen können.
- Verwendung von FPGAs in Steuergeräten: Gemäß [Voll06] sind bereits heute bis zu 18 feldprogrammierbare Logikschaltungen in einem modernen Oberklassefahrzeug verbaut, die anstelle herkömmlicher Mikrocontroller, ASICs bzw. Standard-ICs Verwendung finden. Um solche Steuergeräte zu entwickeln, müssen somit auch die dazu verwendeten Rapid-Prototyping-Plattformen mit FPGAs ausgestattet sein. Da zu Beginn eines Designs der Gatter und Flächenbedarf auf einem FPGA noch nicht feststeht und meist auch die Funktionalität, die darauf integriert werden soll noch nicht vollständig spezifiziert ist, ist eine skalierbare Lösung sinnvoll.
- Partielle dynamische Re-Konfiguration: Im Zusammenhang mit der Verwendung von FPGAs soll die partielle dynamische Konfiguration als Option offen gehalten werden. Dadurch lassen sich auch Steuergeräte, die einen FPGAs als Hauptkomponente enthalten und diese Anforderungen stellen, mit dem System entwickeln und testen.

Speziell für den Bereich Hardware-in-the-Loop können noch weitere Punkte hinzugefügt werden, die insbesondere für diese Aufgabenstellung hinsichtlich eines flexiblen Einsatzes und der höheren zeitlichen Anforderungen eine wichtige Rolle spielen:

- Bereitstellung von Ressourcen zur parallelen Berechnung mehrere Modelle außerhalb des Hauptprozessors – möglichst ohne zusätzliche, spezielle Hardware
- Möglichkeit zur sicheren Abspeicherung von Daten auf der Plattform (z.B. bei nicht stationären Einsätzen im Auto, wo keine anderen Auslagerungsmöglichkeiten vorhanden sind)

Software und Bedienung

- Wiederverwendung von Implementierungen: Die Wiederverwendung von vorhandenen Implementierungen ist eine effiziente Methode, um die Entwicklungszeit von neuen Systemen zu verkürzen. Dies kann man sich auch bei solchen Systemen zu Nutze machen, deren Aufgabe es ist, sich durch Variation der Schnittstellen an wechselnde Umgebungen anzupassen.
- Kommunikation mit dem System: Die Kommunikation mit dem System soll zumindest über eine Standardschnittstelle (z.B. Ethernet) möglich sein, um es darüber zu steuern, mit Daten zu versorgen (z.B. Parametrierung der Schnittstellen), zu Debuggen (z.B. die Firmware bzw. die Modell-Software) und auch Daten (z.B. Messwerte) wieder abzurufen.
- Einfache Funktionsimplementierung (bei Verwendung von FPGAs): Der Benutzer muss das System verwenden können, ohne sich mit den Details einer Funktionsimplementierung auf dem FPGA auseinandersetzen zu müssen. Es dürfen keine Spezialkenntnisse, wie die Beherrschung einer Hardware-Beschreibungssprache (z.B. VHDL, Verilog) oder die Erfahrung in Umgang mit FPGAs vorausgesetzt werden.
- Bedarfsgerechte Konfiguration des Systems: Möglichkeit zur Implementierung von benötigten Schnittstellen nach der Maßgabe „nur so viel wie nötig“, so dass eine optimale Systemauslastung entsteht – hinsichtlich Platzbedarf und Energieaufnahme.

Systemsicherheit

- Vorkehrung zur einfachen Zusammenstellung und Überprüfung der Schnittstellenkonfigurationen für den oder die FPGA(s) zum Ausschluss inkorrekter Implementierungen, die zur Zerstörung des/der FPGA(s) führen können
- Automatische Ressourcenprüfung der aktuellen Plattform-Hardware vor Inbetriebnahme, so dass nur gültige Schnittstellenkonfigurationen verwendet werden können, bei denen die I/O-Treiber zum Modell auf dem FPGA passen und umgekehrt
- Überprüfung des FPGA-Programmiervorgangs mit entsprechender Rückmeldung, damit Fehler erkannt werden und an den Benutzer zurückgemeldet werden können
- Definiertes Zuschalten der I/O-Treiber nach erfolgreicher Konfigurierung der FPGAs zur Vermeidung von Fehlverhalten oder Zerstörung durch offene Eingänge
- Schutz vor Vertauschen der Einsteckkarten durch entsprechende Steckerauswahl oder Kodierung derselben
- Überwachung der Energieversorgung der Plattform und Abschaltung im Fehlerfall (z.B. bei Über- oder Unterspannung) oder Abschaltung durch die Plattform selbst (z.B. bei Messungsende)

4 Die RP-HiL-Plattform

Dieses Kapitel widmet sich der Erstellung des Hardware-Konzepts des Basissystems und dessen Spannungs- bzw. Stromversorgung. Beide Systeme sind aus Gründen der Praktikabilität und der getrennten Testbarkeit als getrennte Baugruppen ausgeführt und werden daher auch getrennt betrachtet. Zunächst wird auf das Systemkonzept des Basissystems unter Betrachtung der Realisierungsalternativen eingegangen. Im Anschluss daran werden die Erfordernisse hinsichtlich der Leistungsaufnahme des Basissystems für die Konzeptionierung herangezogen.

4.1 Systemkonzept des Basissystems

4.1.1 Trennung von Logik und Prozesskopplung bei I/O-Schnittstellen

Ein entscheidender Punkt in der Umsetzung einer neuen, flexibleren Architektur ist die Aufgabe der festen Kopplung zwischen der digitalen Steuerlogik und der Elektronik zur Prozessadaptierung (siehe Abbildung 4.1). Bei einer Ausgabeschnittstelle entfällt damit der bei den gängigen Systemen mit auf dem selben Board integrierte Ausgangstreiber, der eine bestimmte Ausgangsleistung zur Verfügung stellen kann.

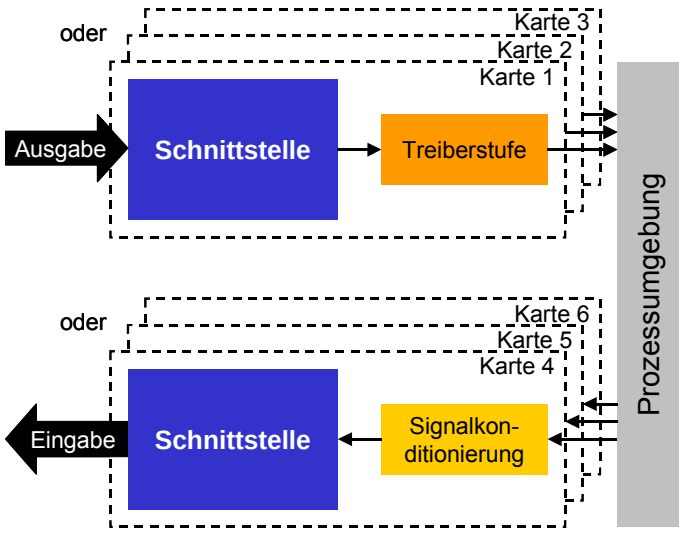

Abbildung 4.1: Struktur aktueller Ein-/Ausgabe-Schnittstellenkarten

Letzterer wird, wie in Abbildung 4.2 zu sehen, als gesondertes Modul bereitgestellt. Für Eingabeschnittstellen gilt das Gleiche, lediglich in anderer Richtung. Beispiele für solche Ein- und Ausgangstreiber bzw. Signalkonditionierungsstufen finden sich in Kapitel 3.10.3 und Kapitel 3.11.1.

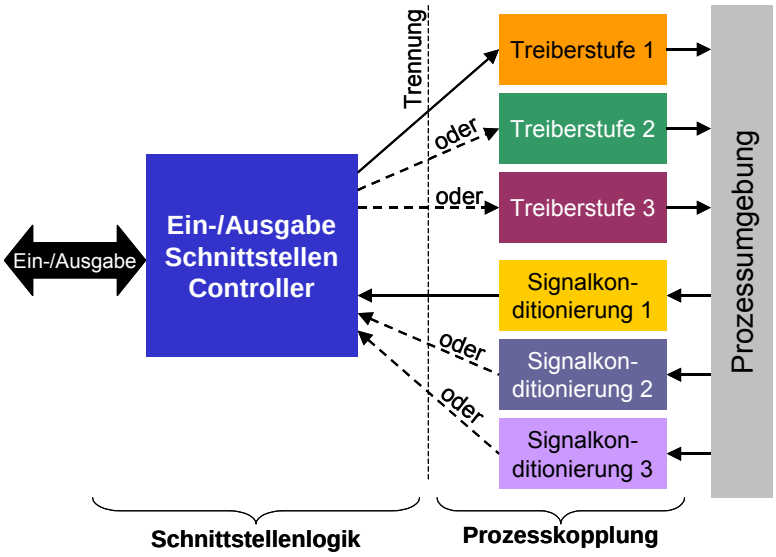

Abbildung 4.2: Struktur bei Trennung von Schnittstellenlogik und Prozesskopplung

Mit diesem Schritt lassen sich verschiedene Szenarien generieren, in denen jeweils gleich bleibende Logikimplementierungen mit unterschiedlichen Ein- oder Ausgangstreibern kombiniert, bzw. hinter einer unveränderten Signalkonditionierung, verschiedene Hardware-Schnittstellen zur Auswertung der eingehenden Signale verwendet werden. Beispielsweise kann ein digitaler PWM-Ausgang einmal an ein Ausgangstreibermodul für 5V, eines für 12V oder ±12V gekoppelt werden (Abbildung 4.2 Treiberstufe1-3), ohne dazu jeweils eine komplett andere PWM-Karte einsetzen zu müssen. Für einfache Schaltausgänge können Treiber mit verschiedenen Leistungen (z.B. 5W, 10W oder 50W) bereitgestellt werden, ohne dazu die Ansteuerungen zu ändern. Das Gleiche gilt auch für Signaleingänge (wie z.B. Zähler/Timer oder PWM-In), bei denen die digitale Auswertelogik gleich bleibt, sich aber die Spannungslevel der Prozesssignale unterschiedlich sein können (Abbildung 4.2 Signalkonditionierung1-3). Dabei reicht es aus, lediglich die Signalkonditionierung anstelle der gesamten Karte zu ersetzen. Durch dieses Vorgehen ist es auch möglich, jede Schnittstelle einzeln an die Spannungspegel oder Leistungsanforderungen des Prozesses anzupassen.

4.1.2 Bedarfsgerechte Konfiguration des Systems

Bei herkömmlichen RP- oder HiL-Systemen wird das Basissystem mit Hilfe von Karten um verschiedene Schnittstellenarten (siehe Kapitel 3.10.3 für RP- und Kapitel 3.11.1 für HiL-Schnittstellen) ergänzt, die zur Adaption an die Umgebung nötigt sind. Die dazu verwendeten Karten bieten meist nur eine Funktion (z.B. ausschließlich PWM-Erzeugung), dazu aber eine Vielzahl an identischen Kanälen an. Werden jedoch viele unterschiedliche Schnittstellen benötigt, ist pro Schnittstellentyp eine Karte erforderlich, von denen im Extremfall jeweils nur ein Kanal benutzt wird. Dieses Vorgehen ist sehr ineffektiv hinsichtlich Energieaufnahme und Platz- bzw. Raumbedarf, da die verfügbaren Ressourcen nur zu einem geringen Teil ausgelastet sind, aber dennoch komplett mit Energie versorgt werden müssen und Steckplätze im System belegen.

Die freie Programmierbarkeit der Gatter und deren Anordnung im FPGA ermöglichen es, verschiedene Funktionsblöcke parallel zu implementieren und unabhängig voneinander zu

betreiben. Da die meisten Funktionsimplementierungen innerhalb des FPGA nur eine begrenzte Fläche beanspruchen, kann der verbleibende Rest durch eine oder verschiedene andere belegt werden. Damit entsteht die Möglichkeit, die im System vorhandenen Ressourcen, hier die des FPGA, effektiv zu nutzen und somit eine bedarfsgerechte Konfiguration des Systems zu erreichen. Es können genau so viele Kanäle einer Schnittstelle implementiert werden, wie tatsächlich benötigt werden. Die parallele Verwendung verschiedener Schnittstellenarten auf einem Device reduziert außerdem noch den Platzbedarf.

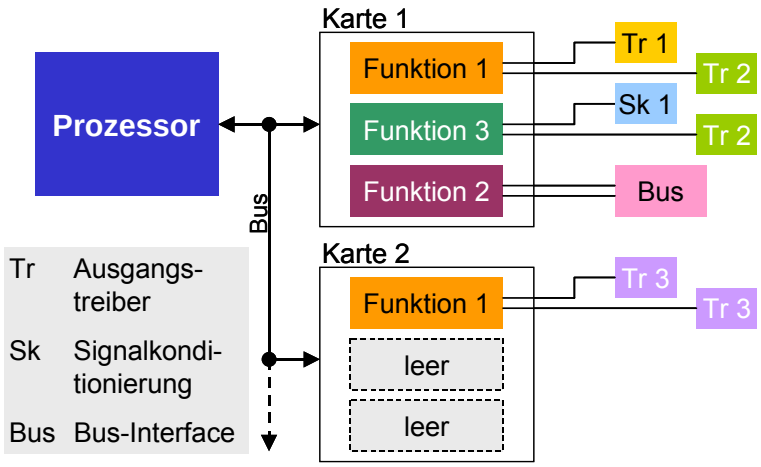

Abbildung 4.3: Bedarfsgerechte Implementierung von Funktionen

In Abbildung 4.3 ist eine schematische Beispielkonfiguration eines RP- oder HiL-Systems zu sehen, das der Einfachheit halber auf den Prozessor, I/O-Karten und die Elemente zur Prozesskopplung reduziert ist. Die Reduktion des Platzbedarfs im Hinblick auf verschiedene I/Os wird bereits bei der oberen Karte 1 deutlich, auf der drei verschiedene Funktionen parallel untergebracht sind. In den derzeit verfügbaren, kommerziellen RP- oder HiL-Umgebung müssten dazu schon drei unterschiedliche I/O-Karten verwendet werden, die jeweils eine der Funktionen 1 bis 3 umsetzen.

In dieser Konstellation wird außerdem deutlich, welche Vorteile die Auftrennung zwischen Logik und Prozesskopplung, wie sie in Kapitel 4.1.1 beschrieben wurde, mit sich bringt. Die auf der Karte 1 und Karte 2 implementierte Funktion 1 steht stellvertretend für einen Ausgabeport einer Schnittstelle (z.B. digitale Ausgabe zum Schalten einer Last). Durch die Verwendung unterschiedlicher Ausgangstreiber (Tr 1, Tr 2 und Tr 3) kann sie an die Anforderungen des Prozesses angepasst werden, wobei das eigentliche Schnittstellenverhalten (Funktion 1) identisch ist.

4.1.3 Einsatz von FPGAs

FPGAs eignen sich grundsätzlich zur Integration jeglicher Art von digitaler Logik und ermöglichen die Entwicklung anwendungsoptimierter, komplexer und hierarchischer Systeme mit geringem Aufwand. Je nach Baugröße des FPGAs können einige ICs oder auch einige CPUs inklusive Peripherie darin Platz finden. Die Systemgeschwindigkeit (max. Taktrate) ist allerdings abhängig von der Implementierung der Logik und der anschließenden Verdrahtung. Bei entsprechender Auslegung der Baugruppe, kann diese durch die Möglichkeit der Re-Programmierbarkeit der FPGAs für verschiedenste Applikationen eingesetzt werden. Im Gegensatz zu Standard-ICs oder ASICs mit einer definierten Funktion und fester Hardware-Struktur, ergeben sich so Vorteile durch leichte Abänderbarkeit, Anpassungsfähigkeit, Optimierbarkeit und Modularisierbarkeit. Voraussetzung hierfür ist jedoch ein auf den Einsatz von FPGAs optimierter Lösungsansatz, der den Benutzer bei der Implementierung der Funktionalität auf dem FPGA unterstützt (Kapitel 5).

Die oben aufgeführten Merkmale von FPGAs sollen nun hier auf die Anforderungen an ein neues Konzept (Kapitel 3.13) projiziert werden, um die Vorzüge der Bausteine zur Umsetzung des Systems so weit als möglich nutzbar zu machen. Auf diese wird im Folgenden detailliert eingegangen.

4.1.3.1 Treiberleistungen von FPGA-Portpins

Da FPGAs per se nur digitale Ports mit geringer Leistungsabgabe bzw. -aufnahme besitzen, ist die oben geforderte Auftrennung von Logik und Prozessankopplung ohnehin vorzunehmen und stellt daher keine Einschränkung hinsichtlich der Verwendung der FPGAs dar. In der folgenden Tabelle 4.1 sind die maximalen Ströme der gängigsten Modi aufgeführt. Sie gelten jeweils für den Fall, dass der Portpin als Quelle oder Senke fungiert. Die Treiberstärke kann durch Programmierung modifiziert werden. Die relevanten Spannungslevel der angegebenen Ströme unterscheiden sich je nach Modus. So sind für LVTTL und LVCMOS33 jeweils 3,3V als Referenzspannung einzuhalten, während bei LVCMOS18 maximal 1,8V angelegt werden dürfen. Daraus ergibt sich zum Beispiel im LVTTL eine maximale Treiberleistung von 79,2 mW und im LVCMOS sogar nur 28,8 mW.

Select-I/O	**Programmierbare Ströme (Garantierte Minima)**						
LVTTL	2 mA	4 mA	6 mA	8 mA	12 mA	16 mA	24 mA
LVCMOS33	2 mA	4 mA	6 mA	8 mA	12 mA	16 mA	24 mA
LVCMOS25	2 mA	4 mA	6 mA	8 mA	12 mA	16 mA	24 mA
LVCMOS18	2 mA	4 mA	6 mA	8 mA	12 mA	16 mA	n/a
LVCMOS15	2 mA	4 mA	6 mA	8 mA	12 mA	16 mA	n/a

Tabelle 4.1: Programmierbare Ströme in den Modi LVTTL und LVCMOS (je Source/Sink)

4.1.3.2 Skalierbarkeit der I/O-Schnittstellen

FPGAs bieten hinsichtlich ihrer verfügbaren I/O-Ports einen enormen Spielraum für die Skalierung von Schnittstellen für RP- und HiL-Systeme. Je nach Serie und Gehäusetyp sind Anzahlen von 150 bis weit über 1100 für frei verwendbare Pins typische Werte. Tabelle 4.2 gibt einen Überblick über die Virtex-II-Serie von Xilinx und deren Merkmale, insbesondere der verfügbaren I/O-Schnittstellen des Chips (untere Zeile). Jeder dieser I/O-Pins kann frei in Funktion programmiert werden. Die verschiedenen Spannungslevel (LVTTL, LVCMOS33, LVCMOS25, LVCMOS18, PCI66_3, usw.) können dagegen nur bankweise festgelegt werden.

	Xilinx Virtex-II FPGA XC2V...										
Merkmal	40	80	250	500	1000	1500	2000	3000	4000	6000	8000
Logikzellen	576	1,152	3,456	6,912	11,520	17,280	24,192	32,256	51,840	76,882	104,882
Max. I/O	88	120	200	264	432	528	624	720	912	1,108	1,108

Tabelle 4.2: Anzahl an freien I/Os der Virtex-II-FPGA-Serie von Xilinx

Eine hohe Anzahl an Pins wird besonders für den Hardware-in-the-Loop-Test benötigt, da dort einzelne Steuergeräte oder gar ein ganzer Verbund getestet wird. Bei einem Steuergerät alleine sind dazu im Schnitt 60 Ein- und Ausgänge zu bedienen. Im Verbundtest, bei dem mehrere mit einander gekoppelte Steuergeräte geprüft werden, können so leicht Pinanzahlen von mehreren hundert entstehen.

4.1.3.3 Erreichbare Taktraten

Die maximalen Taktraten innerhalb der heute gängigen FPGAs können mehrere 100MHz erreichen, abhängig von der FPGA-Serie und der Geschwindigkeitsklasse (bei Xilinx „Speed Grade“, siehe Tabelle 4.3). Allerdings ist dieser Wert zunächst nur von statistischem Wert, da dies stark von der Logikauslastung und dem Routing auf dem Chip abhängt.

Beschreibung	Geschwindigkeitsklasse (Speed Grade)		
	6	5	4
Toggle-Frequenz	820 MHz	750 MHz	650 MHz

Tabelle 4.3: Toggle-Frequenzen der FlipFlops in Virtex-II-FPGAs

Bei der Verwendung der FPGAs im Bereich RP und HiL sind die maximal möglichen Taktraten zum und vom Prozess interessant. Diese Werte liegen um einiges niedriger, da die Ein- und Ausgangstreiber an den Pins der FPGAs eine Reduktion des Taktes nötig machen, um die Qualität der Signale sicherzustellen. Standardwerte sind hier 133MHz im PCI-X-Modus oder 66MHz im PCI-Modus. Doch auch hier sind Frequenzen im Bereich bis 200MHz möglich, was um einiges höher liegt als bei den derzeit schnellsten I/O-Karten. Nimmt man diesen Wert auch als Grenze für eine maximale Signalabtastung, können so Auflösungen bis 5ns erreicht werden, so dass beispielsweise hochfrequente PWM-Signale oder solche mit großer Auflösung des Tastverhältnisses erzeugt bzw. aufgenommen werden können. Das Gleiche gilt für Zähler oder Timer (z.B. im Bereich der Messung oder Erzeugung von Kurbelwellensignalen), die so feinere Auflösungen erreichen können. Damit ist auch für zukünftige Anforderungen an die Signale vorgesorgt. Im Hinblick auf die Verwendung von aktuellen oder zukünftigen Bussystemen FlexRay (10MHz) oder Most (25MHz), die mit die höchsten Anforderungen an die Taktraten stellen, ist ein auf alle Fälle genügend großer Spielraum gegeben.

4.1.3.4 Re-Programmierbarkeit

RAM-basierte FPGAs lassen sich beliebig oft neu programmieren, was sie alleine schon deswegen von Standard-ICs oder ASICs abhebt, die eine fest vorgegebene, dedizierte Aufgabe erfüllen und die nicht geändert werden kann. Dies hat zum einen den Vorteil, dass man damit schnell zu ersten funktionsfähigen Prototypen gelangt und ein schnelles Time-to-Market erreicht, zum anderen aber auch die Art und Funktion der Implementierungen über der Zeit ändern kann und damit nur eine einzige Ressource für sich ändernde Aufgaben benötigt. Beides spielt im Rahmen des Rapid Prototyping eine herausragende Rolle. Bisherige Systeme lassen sich zwar auch in ihren Eigenschaften verändern, jedoch basiert diese Änderung auf Modifikation der Software oder durch Austausch von steckbarer Hardware. Durch den Einsatz von FPGAs kann so auch die Hardware verändert werden, ohne einen Eingriff in das System vornehmen zu müssen.

Dahingehend kann man sich die Re-Programmierbarkeit auch zu Nutze machen, indem der FPGA nicht als Plattform der zu erstellenden Funktion verwendet wird, sondern als flexible und konfigurierbare Schnittstelle, die mit vordefinierten Funktionen versehen werden kann. Solche Funktionen können z.B. digitale Ein- und Ausgänge, PWM-Ein- oder Augänge, Timer, Counter oder digitale Filter sein (um nur einige zu nennen). Dazu werden je nach den Schnittstellenanforderungen der Applikation die Module mit den benötigten Funktionen auf den oder die FPGA(s) geladen. Bei Änderungen in den Anforderungen können die Module ausgetauscht werden, indem eine neue Modulzusammenstellung erstellt und dem FPGA zugewiesen wird. Die dazu verwendete Methodik ist in Kapitel 5 detailliert beschrieben. Um die genannten Zusammenstellungen (auch Konfiguration) für einen FPGA zu erstellen, wurde ein Werkzeug entwickelt, das in Kapitel 6.2 vorgestellt wird.

4.1.3.5 Wiederverwendung von Implementierungen

Die Wiederverwendung von vorhandenen Implementierungen ist eine effiziente Methode, um die Entwicklungszeit von neuen Systemen zu verkürzen. Dies kann man sich auch bei solchen Systemen zu Nutze machen, deren Aufgabe es ist, sich durch Variation der Schnittstellen an wechselnde Umgebungen anzupassen.

Dafür ergeben sich bei einem FPGA mehrere Möglichkeiten, die sich in der Abstraktionsebene und damit in der Art, wie die Funktion zur Verfügung steht, unterscheiden. Wie bereits in Kapitel 3.8.2 ausführlich beschrieben, bieten sich dazu Soft-IP-Cores in verschiedenen Designzuständen, sowie auf dem Chip fest verdrahtete System-On-Chip-Lösungen (SoC), wie z.B. µController oder Prozessoren (PowerPC in Xilinx Virtex-II PRO), an. Letztere werden hier jedoch nur am Rande betrachtet, da sich die eigentliche Wiederverwendung auf die Software bezieht, nicht jedoch auf die unveränderbare, da fest implementierte, Hardware.

Im Rahmen dieser Arbeit entstand neben den schon vorgestellten Gruppen von IP-Cores noch eine weitere, nämlich die der wieder verwendbaren Teil-Bitstreams. Ein Bitstream enthält normalerweise die gesamte Konfiguration eines FPGAs. Jedoch lassen sich daraus auch Teile herausschneiden und als einzelne Einheiten – sog. Module – ablegen. Durch die regelmäßige Struktur der programmierbaren Blöcke (CLB, Configurable Logic Block) im FPGA können diese Teile dann einzeln oder neu zusammen gesetzt wieder verwendet werden. Der Vorteil bei dieser Lösung ist der, dass diese Teil-Bitstreams bereits als fertig platzierte und geroutete Funktionseinheiten vorliegen und im Falle einer neuen Zusammenstellung mit anderen, gleichartigen Modulen zusammengestellt werden können, ohne dass ein Synthesedurchlauf gestartet werden muss. Die Grundlagen dazu und die Vorgehensweise des Ausschneidens und Zusammenfügens sind in Kapitel 5 dargelegt.

4.1.3.6 Digitale Signalverarbeitung

Moderne FPGAs eignen sich hervorragend als Basis digitaler Signalverarbeitungssysteme, da sie einerseits ein enormes Angebot an programmierbarer Hardware zur Verfügung stellen, andererseits bereits verschiedene Ressourcen (wie Multiplizierer, usw.) bieten, die für Berechnungen benötigt werden. Außerdem stehen mit auf dem Chip enthaltenen Prozessorkernen (z.B. PowerPC bei Xilinx und ARM bei Altera) weitere Hilfsmittel zur Verfügung, um auch Software-gestützte bzw. kombinierte Lösungen zu integrieren.

Im Hinblick auf den Einsatz der Plattform als RP-System sind solche Eigenschaften zur Umsetzung von Anwendungen für die Automobilelektronik wie Fahrer-Informationssysteme, Infotainment (Software Defined Radio, SDR), MOST-basierte Audio/Video-Übertragung, sowie Gateways oder Netzwerke von erheblichem Vorteil bzw. unverzichtbar. Alle hier aufgeführten Anwendungen benötigen eine hohe Rechenleistung (z.B. 300 – 400 MIPS für SDR), die wegen der sequentiellen Verarbeitung eines Mikrocontrollers nur sehr schwer auf einem solchen durchgeführt werden können. Besser eignet sich dazu ein FPGA, der zum einen durch seine frei programmierbare Logik eine Parallelisierung erlaubt, zum anderen durch die schnellen Prozessorkerne eine effiziente Steuerung erzielt werden kann.

Aber auch im Bereich von Hardware-in-the-Loop-Tests sind solche Strukturen sehr gut einzusetzen, wenn es um die parallele Berechnung von Sensor- oder Aktormodellen zur Simulation der Umgebung geht. Insbesondere bei der Verwendung mehrerer FPGAs, kann beispielsweise ein dem Hauptprozessor der Plattform untergeordnetes Co-Prozessor-Cluster aufgebaut werden. Dadurch entstehen bei der Modellrechnung keine zusätzlichen Latenzzeiten durch den Datentransport vom Sensor zum Prozessor und wieder zurück zum Aktor, da sowohl die Adaptierung der Signaleingabe und –ausgabe direkt über die Logik des FPGAs vorgenommen wird.

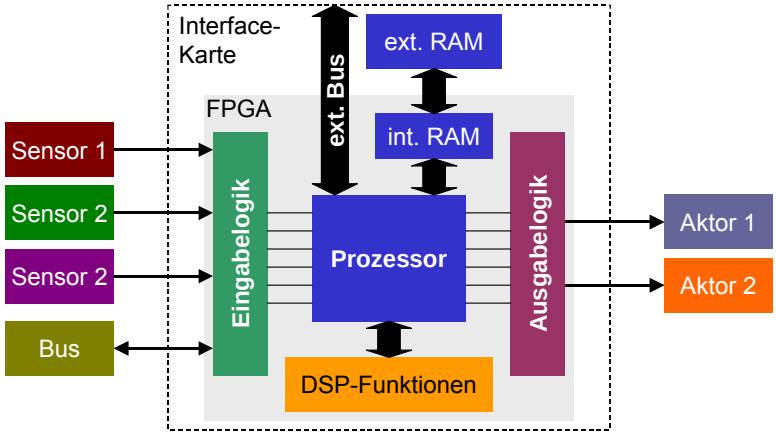

Abbildung 4.4: DSP-Subsystem mit direkter Sensor-/Aktoranbindung

In Abbildung 4.4 ist eine mögliche Implementierung eines untergeordneten Co-Prozessors dargestellt, die auf einem FPGA untergebracht ist und über einen externen Bus mit dem Hauptprozessor kommunizieren kann. Der FPGA selbst enthält den Co-Prozessor, der als fester Kern vorhanden oder als Soft-IP-Core implementiert sein kann, sowie die Ein- und –ausgabelogik, an die Sensoren (links) und Aktoren (rechts) angeschlossen sind. Im internen bzw. externen RAM können sowohl der auszuführende Code (z.B. das Rechenmodell) als auch Daten abgelegt werden. Der Block DSP-Funktionen steht hierbei stellvertretend für solche Funktionen, die für eine digitale Signalverarbeitung benötigt werden und sich effektiver in Hardware als in Software umsetzen lassen und dadurch zur Leistungssteigerung des Systems beitragen. Beispiele dafür sind Multiplizierer, MAC-Einheiten, FIR-Filter, Fest- und Fließkommaarithmetische Recheneinheiten (FPUs) usw.. Über den externen Busanschluss kann die Modellrechnung von außen beeinflusst werden.

4.1.3.7 Auswahl der passenden FPGAs

Grundsätzlich kann das vorgestellte Konzept mit FPGAs verschiedener Hersteller, wie z.B. Altera, Atmel, Lattice, Xilinx, etc. umgesetzt werden. Allerdings sind einige FPGAs in ihrer Leistungsfähigkeit und zur Verfügung stehender Gattergröße sehr stark beschränkt (Atmel bietet maximal 40.000 Gatter, Lattice maximal 1,1 Mio. Gatter), so dass letztlich nur Bausteine der beiden Marktführer Altera und Xilinx in Frage kommen. Sie bieten verschiedene Serien und innerhalb derer eine große Auswahl hinsichtlich der integrierten Gatteräquivalente. Im Hinblick darauf, die Möglichkeit die partielle dynamische Re-Konfigurierung der Bausteine für spätere Implementierungen offen zu halten, bleiben im Grunde nur FPGAs von Xilinx übrig. Altera Devices sind per se nicht partiell re-konfigurierbar [DoHa03].

Für eine erste Implementierung, d.h. eine prototypische Realisierung des RP-Hil-Systems, wurden die derzeit gängigsten, ab Lager verfügbaren und auch einigermaßen erschwinglichen Serien aus dem Xilinx-Sortiment ausgewählt. Es handelt sich dabei um die Xilinx Virtex-II-Serie, die Gatteräquivalente zwischen 40.000 und 8 Millionen in verschiedenen Gehäusetypen bietet (siehe Tabelle 4.4). Eine optimale Größe hinsichtlich Preis/Leistung, verfügbaren Gatterelementen und Pinanzahl stellte der Virtex-II-1000 mit einer Million Gatter in einem 256er BGA-Gehäuse dar. Er ist damit auch pinkompatibel zu dem Virtex-II-500 und Virtex-II-250, beide ebenfalls im BGA256-Package.

	Xilinx Virtex-II FPGA XC2V...										
Merkmal	40	80	250	500	1000	1500	2000	3000	4000	6000	8000
Logikzellen	576	1.152	3.456	6.912	11.520	17.280	24.192	32.256	51.840	76.882	104.882
BRAM (Kbits)	72	144	432	576	720	864	1.008	1.728	2.160	3.024	3.024
18x18 Multi-plizierer	4	8	24	32	40	48	56	96	120	168	168
DCM-Blöcke	4	4	8	8	8	8	8	12	12	12	12
Max. I/O	88	120	200	264	432	528	624	720	912	1.108	1.108

Tabelle 4.4: Übersicht über die Merkmale der Virtex-II-Serie von Xilinx

Im Hinblick auf die Weiterentwicklung des hier vorgestellten Ansatzes und zur Steigerung der Performance könnten auch FPGAs mit bereits integrierten Prozessorkernen eingesetzt werden, ohne das Systemkonzept ändern zu müssen. Die Bausteine der Xilinx Virtex-II-Pro-Serie („Pro“ steht für Prozessor) enthalten zuzüglich zu den frei programmierbaren Logikressourcen je nach Größe des FPGA bis zu zwei PowerPC (PPC) Prozessoren. Sie sind ebenfalls im BGA-Gehäuse erhältlich und stellen damit eine gute Alternative zu den bisher verwendeten FPGAs dar, da sie die Integration von Rechner-Subsystemen im eigentlichen System unterstützen, ohne zusätzliche Hardware zu erfordern. Tabelle 4.5 zeigt eine Übersicht über die FPGAs der Virtex-II-Pro-Serie.

	Xilinx Virtex-II-PRO FPGA XC2V...								
Merkmal	P2	P4	P7	P20	P30	P40	P50	P70	P100
Logikzellen	3.168	6.768	11.088	20.880	30.816	46.632	53.136	74.448	99.216
BRAM (Kbits)	216	504	792	1.584	2.448	3.456	4.176	5.904	7.992
18x18 Multiplizierer	12	28	44	88	136	192	232	328	444
DCM-Blöcke	4	4	4	8	8	8	8	8	12
PowerPC® Prozessoren	0	1	1	2	2	2	2	2	2
3.125 Gbps RocketIOs	4	4	8	8	8	12	16	20	20
Max. I/O	204	348	396	564	644	804	852	996	1.164

Tabelle 4.5: Übersicht über die Merkmale der Virtex-II-Pro-Serie von Xilinx

4.1.4 Ressourcenbelegung auf dem FPGA

Die Ressourcenbelegung auf dem FPGA ist bereits jetzt, also noch unabhängig von der automatisierten Lösung zur Funktionsimplementierung, wie sie in Kapitel 5 beschrieben wird, ein wichtiger Aspekt bei der Planung der Hardware-Plattform. Die Überlegungen hinsichtlich der Logikplatzierung müssen daher sehr früh angestellt werden, um ein evtl. irreversibel falsches Hardware-Konzept zu vermeiden, was letztlich auch zum Scheitern der Gesamtlösung führt.

In Abbildung 4.5 ist ein schematischer Basisaufbau des FPGA (hier Virtex-II von Xilinx) gezeigt, in der nur die Elemente enthalten sind, die für die Auswahl der am besten geeigneten Belegungsvariante ausschlaggebend sind. Eine etwas detailliertere, allerdings auch schematische, Ansicht findet sich in den Grundlagen in Abbildung 3.2. Der interne Aufbau des FPGA, bzw. dessen regelmäßige Struktur, erlaubt eine freie Belegung der Gatterelemente durch programmierbare Logik. Die folgenden Abbildungen (Abbildung 4.6 bis Abbildung 4.9) zeigen vier Möglichkeiten der strukturellen Implementierung von Funktionsblöcken (Modulen) auf dem Chip. Allerdings ergeben nicht alle dieser Konfigurationen Sinn,

wenn man an eine einheitliche und gut zu verwaltende Lösung einerseits und an eine leichte Handhabbarkeit andererseits denkt. Im Folgenden sind die Randbedingungen aufgezeigt, die es zu erfüllen gilt.

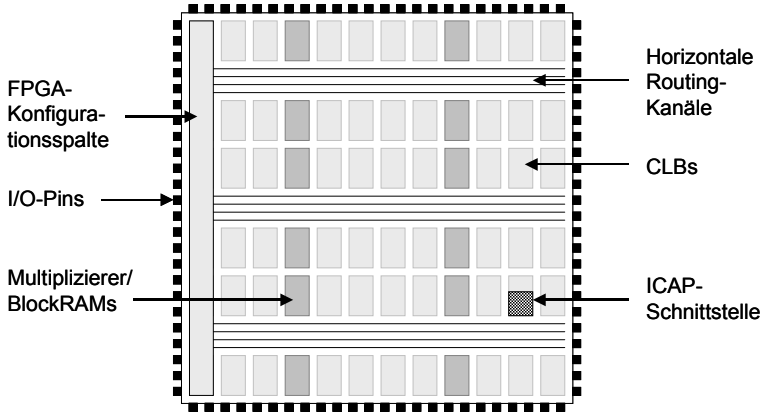

Abbildung 4.5: Grundstruktur FPGA

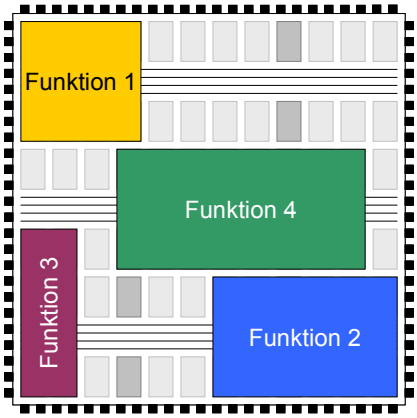

Abbildung 4.6: Funktionsbelegung 1

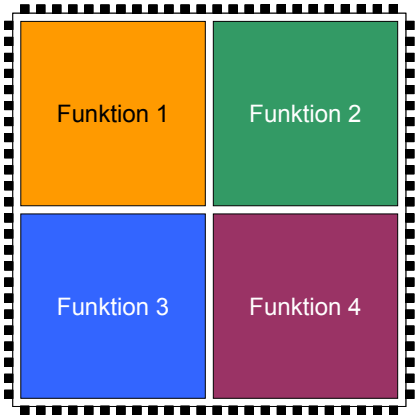

Abbildung 4.7: Funktionsbelegung 2

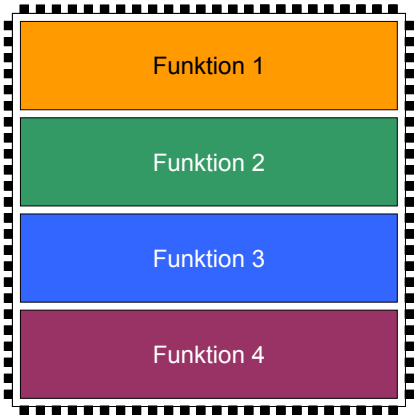

Abbildung 4.8: Funktionsbelegung 3

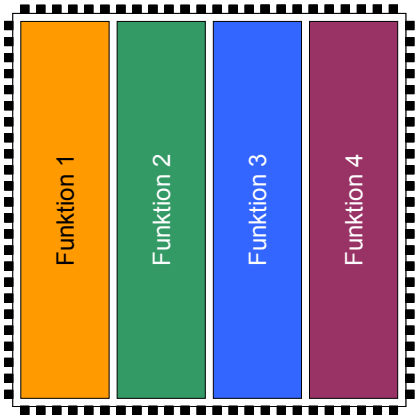

Abbildung 4.9: Funktionsbelegung 4

Basis der Überlegungen (siehe dazu auch Kapitel 5.2.1) ist eine regelmäßige Aufteilung der einzelnen Einheiten, wodurch die internen Elemente, d.h. CLBs, RAM-Blöcke und Multip-

liziereinheiten, insbesondere aber die I/O-Pins gleichmäßig den Einheiten zugeteilt werden können. Außerdem sollten zur Vereinfachung des späteren Platinen-Designs sich alle Modul-I/O-Ports zum Prozess auf einer Seite des FPGAs befinden. Dadurch ist die in Abbildung 4.6 dargestellte Lösung nicht realisierbar, da die Größe der Module nicht einheitlich gestaltet ist und zum Erreichen der I/O-Pins auf einer vorgegebenen Seite, die Verbindung zwischen Modul und zugehörigem I/O-Port teilweise durch bereits belegte Flächen oder solche, die noch belegt werden können, erfolgen muss. Letzteres ist auch der Ausschlussgrund für die in Abbildung 4.7 gezeigte Anordnung, obgleich hier eine Aufteilung des FPGAs in gleichgroße Einheiten gelungen ist.

Im Gegensatz zu den ersten beiden Varianten werden in Abbildung 4.7 und Abbildung 4.8 beide Kriterien erfüllt. Jedoch ist bei der Implementierung noch ein weiteres FPGA-spezifisches Kriterium zu beachten. Die Xilinx-FPGAs sind RAM-basiert und müssen zu Beginn konfiguriert werden. Dies geschieht – mit Blick auf die Darstellung in Abbildung 4.5 – spaltenweise, d.h. die einzelnen Konfigurationseinheiten innerhalb des FPGAs verlaufen von oben nach unten, wobei zu beachten ist, dass solche Konfigurationsspalten nur als komplette Einheit ausgetauscht werden können. Im Hinblick darauf ist es sinnvoll, die Ausrichtung der Module an der der Konfigurationsspalten zu orientieren und eine Modulbreite als Vielfaches von CLB-Spalten zu definieren. Damit scheidet auch die Aufteilung in Abbildung 4.8 aus, da hier die Module senkrecht zu den Konfigurationsspalten des FPGAs verlaufen und eine einfache Unterteilung (z.B. Belegung der CLB-Spalten 0-7 durch Modul 1) nicht möglich ist.

Der einzig verwend- und verwertbarer Ansatz ist die strukturelle Aufteilung der FPGA-Ressourcen nach Abbildung 4.9, der alle bisher genannten Forderungen erfüllt. Die Einheiten sind alle gleich groß und in einer regelmäßigen Struktur auf der FPGA-Fläche angeordnet. Die Modul-I/Os können über die im Bild oben liegende Seite aus dem Chip herausgeführt werden. Durch ihre vertikale Ausrichtung und die Ausdehnung über die gesamte Chip-Höhe, ist jedes der Module über alle horizontalen Routing-Kanäle erreichbar. Diese können somit sehr gut als Busleitungen verwendet werden, um die Module über eine einheitliche Schnittstelle, d.h. beispielsweise ein internes Bussystem, untereinander bzw. auch mit einem externen Bus zu verbinden.

Der Bus zum Datentransport von und zu den Funktionsmodulen braucht neben einer geeigneten Ansteuerung auch ein Interface nach außen, d.h. über die Chipgrenzen hinweg, um Zugriffe des Prozessors zu bedienen. Dieser Bedarf an Logik geht zu Lasten eines Bereichs, der die Ansteuerung der internen Bussysteme und das Businterface nach außen enthält und so als fester Bestandteil einer jeden Implementierung immer vorhanden sein muss. Dazu wurde explizit der ganz rechts gelegene Bereich ausgewählt, da dort auch die ICAP-Schnittstelle (Internal Configuration Access Port) der Virtex-II-FPGAs liegt (gepunktetes Feld rechts unten in Abbildung 4.5). Mit Hilfe dieser Schnittstelle kann eine interne gesteuerte Re-Konfiguration des FPGAs erfolgen. Sie wird zunächst nicht integriert, kann jedoch jederzeit eingebunden werden, wodurch ein Zugriff auf die aktuelle FPGA-Konfiguration gegeben ist. Dies kann während der Laufzeit selbsttätig durch die interne Logik oder von extern über das implementierte Bus-Interface geschehen.

4.1.5 Modularer Aufbau und Aufteilung der Systemfunktionalitäten

Um die Freiheitsgrade bei der Zusammenstellung der Ein- und Ausgabeschnittstellen zu erhöhen und damit eine höhere Flexibilität zu erreichen, wird der Aufbau der Plattform neu gegliedert. Dazu wird ein modularer Aufbau gewählt, durch den drei Kartentypen entstehen, die als Erweiterungen auf die Backplane (Basisplatine) gesteckt werden können. Ihre Aufgaben werden hier im Folgenden erklärt.

4.1.5.1 Aufgaben der Interface-Karte

Die Interface-Karte ist neben der Basisplatine des Systems das wesentliche Element innerhalb des Konzepts und bildet damit den zentralen Baustein für HiL- und RP-

Applikationen. Auf dem FPGA der Karte werden die Modelle der entsprechenden Anwendung zur Ausführung gebracht. Dazu müssen die passenden Bitstreams auf dem Host-PC ausgewählt, und über den Mikrocontroller des Basis-Boards in den FPGA geladen werden. Zur Prozessankopplung der Karte dienen die auf dem Basisboard eingesetzten Schnittstellenkarten. Die FPGA-Karte kann so über I/O- oder Automotive-Karten mit dem Prozess verbunden werden, um digitale und analoge Signale einlesen und ausgeben zu können. Eine Ansteuerung der Schnittstellenkarten kann dafür beispielsweise aus einer einfachen Logik, einem Zustandsautomaten oder einem Softcore-Prozessor erfolgen.

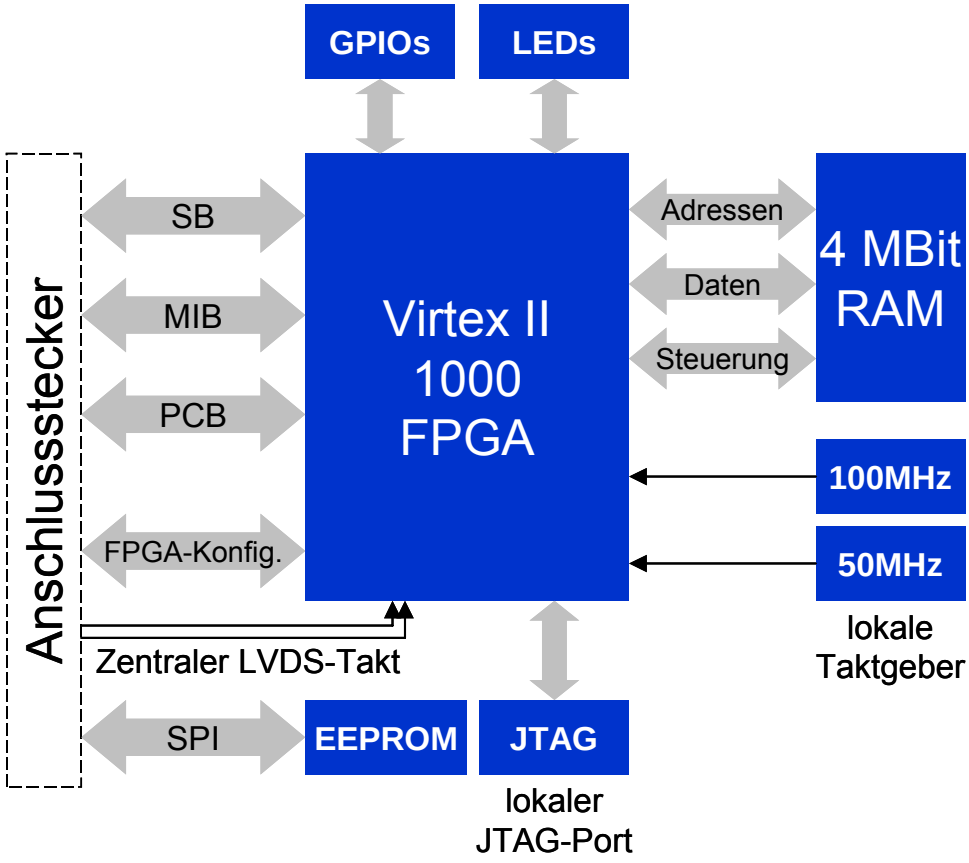

Abbildung 4.10: Schematischer Aufbau der Interface-Karte

Der Aufbau der Karte ist in Abbildung 4.10 dargestellt. Er sieht neben dem Virtex-II-FPGA als frei konfigurierbarem Schnittstellenbaustein ein RAM, zwei Oszillatoren zur Taktgenerierung, Status-LEDs, ein EEPROM und Anschlüsse zur Erweiterung der Karte vor. Das RAM hat die Aufgabe, je nach Art der Implementierung, anfallende Daten oder Zwischenergebnisse zu speichern. Es kann allerdings auch bei der Verwendung eines Prozessors innerhalb des FPGAs als dessen Programm- und Datenspeicher verwendet werden. Die lokalen Taktgeber sollen neben dem globalen, von der Basisplatine aus getriebenen, Takt die Basisfrequenzen für den FPGA liefern. Es wurden daher gängige Frequenzen von 50MHz und 100MHz ausgewählt, die bei Bedarf innerhalb des FPGA noch angepasst werden können. Der von außen zugeführte LVDS-Takt ist in 1MHz-Schritten veränderbar und wird parallel auf alle Interface-Karten geführt. Durch das eingesetzte EEPROM ist die Identifikation der Karte und deren Funktion auslesbar. Das EEPROM muss vor dem ersten Betrieb initialisiert werden.

4.1.5.2 Aufgabe der I/O-Karten

Die I/O-Module auf der Plattform bilden die Verbindung zwischen der eigentlichen Logik auf den FPGAs und dem zu steuernden oder zu regelnden Prozess. Diese Auftrennung ist aus mehreren Gründen wichtig. Einerseits wird damit eine Signalkonditionierung an den Prozess vorgenommen, da dort gemeinhin höhere Spannungen und Ströme benötigt werden, als sie die steuernde Logik auf der Plattform zur Verfügung stellen kann. Die Umsetzung geschieht dann über entsprechende Treiberbausteine, die als Verstärker der Niederspannungssignale von den Interface-Karten fungieren. In manchen Fällen ist es aber auch unumgänglich, die Logik galvanisch vom Prozess zu entkoppeln, damit Störungen oder Überspannungen keinen Schaden an der Plattform anrichten können. Auch dies kann in Form von Opto-Kopplern, oder mit Hilfe von Kondensatoren oder Spulen als kapazitive oder induktive Trennung auf den I/O-Modulen realisiert werden. Andererseits ist über die I/O-Module die Möglichkeit gegeben, die rein digitalen Schnittstellen der FPGAs durch den Einsatz von A/D- oder D/A-Wandlern an analoge Signale anzuschließen. Allerdings gilt auch hier, dass eine Signalkonditionierung durchzuführen ist, die eine entsprechende Pegelanpassung und gegebenenfalls eine galvanische Trennung vornimmt.

Modulvarianten

Der Vielfalt an Möglichkeiten zur Implementierung der Ein-/Ausgabefunktionalität vom bzw. zum Prozess sind im Grunde fast keine Grenzen gesetzt. Es können dazu Karten mit digitalen oder analogen Funktionen, eingesetzt werden, aber auch jede mögliche Kombination der genannten Arten. Diese beiden sind aus dem Grund zu unterscheiden, da bei der analogen Ein- oder Ausgabe je ein Umsetzer – d.h. ein A/D- oder ein D/A-Wandler – benötigt wird. Alle anderen Funktionen, die auf reiner digitaler Hardware basieren, benötigen nur einen passenden Treiber, da die Logik im FPGA abgelegt ist. Darüber hinaus steht es dem Entwickler frei, Treiberstufen zu integrieren, um damit eine bessere Prozessadaption und Signalkonditionierung zu erreichen. Darin enthalten ist auch eine möglicher Weise notwendige galvanische Trennung der Steuerungs- von der Prozessseite.

Die Begrenzungen in der Vielfalt der implementierbaren Schnittstellen liegen in den elektrischen Parametern, die zum einen durch die Plattform, als auch durch die I/O-Platinen selbst vorgegeben werden. In der derzeitigen prototypischen Realisierung enthält der I/O-Bus zwischen jeder Interface- und I/O-Karte nur maximal 20 Leitungen (2 x10), die als Ein-/Ausgabesignale vom oder zum Prozess verwendet werden können. Dies gilt für die ersten beiden Interface-Karten, wohingegen die zwei weiteren nur jeweils 10 Leitungen bereitstellen. Der maximale Strom, der pro Kanal auf die I/O-Karte fließen darf, ist durch den Steckverbinder begrenzt, der pro Pin nur 1A zulässt. Von Seiten der Plattform können nur 10A pro Kanal für alle I/O-Karten bereitgestellt werden. Da nur sechs Steckplätze auf der Plattform für I/O-Karten zur Verfügung stehen und pro Karte nur 1A gezogen werden kann, ist diese Einschränkung sekundär. Für die Spannungen der beiden Versorgungskanäle ist die maximal zulässige Source-Drain-Spannung des FETs entscheidend, mit dessen Hilfe die Kanäle zu- und abgeschaltet werden können. Laut Spezifikation dürfen über dem FET max. 55V abfallen, sodass ein Betrieb selbst für 42V-Systeme kein Probleme bereitet. Die maximale Ausgangsleistung der I/O-Karten ist also abhängig von der verwendeten Prozessspannung und kann so theoretisch bis zu zwei mal 55W betragen. Aufgrund des Aufbaus ist die Platinengröße auf etwa 40 x 50 mm^2 beschränkt. Aus mechanischen Gründen sollte sie möglichst klein gehalten werden, um die Hebelwirkung auf die Steckverbinder zu minimieren. In Tabelle 4.6 sind die Grenzen noch einmal zusammengefasst.

Bei Anforderungen, die über diese elektrischen Charakteristika hinausgehen, muss eine externe Treiberkarte verwendet werden, wie dies auch bei kommerziellen Systemen gehandhabt wird.

Physikalische Größe	Wert	Einheit
Strom (Kanal 1)	1	A
Strom (Kanal 2)	1	A
Spannung (Kanal 1)	55	V
Spannung (Kanal 1)	55	V
Leistung (ges., theor.)	2 x 55	W
Pins	10 bzw. 20	-
Platinengröße	40 x 50	mm^2

Tabelle 4.6: Limitierungen der I/O-Platine

4.1.5.3 Aufgabe der Automotive-Bus-Karten

Die Aufgabe der Automotive-Bus-Karten ist es, die Kopplung zwischen der Logik des RP/HiL-Systems und dem physikalischen Bussystem (z.B. CAN, LIN, FlexRay, MOST, usw.) herzustellen. Da die Controller der jeweiligen Schnittstellen für die Prozessankopplung der Plattform grundsätzlich als VHDL-Modell innerhalb des FPGA implementiert werden, ist auf den einsteckbaren Automotive-Karten lediglich noch die entsprechende Pegelanpassung nötig. Diese wird mit so genannten Transceivern (Transmitter and Receiver-ICs) vorgenommen. Im Falle des CAN-Busses werden dazu die beiden logischen Ein- und Ausgangssignale der Kanäle 1 und 2 in differentielle Bussignale umgesetzt ([SN65HVD230]). Im LIN-Bus-Treiber [Atme05] findet dagegen nur eine Pegelanpassung der Signale statt.

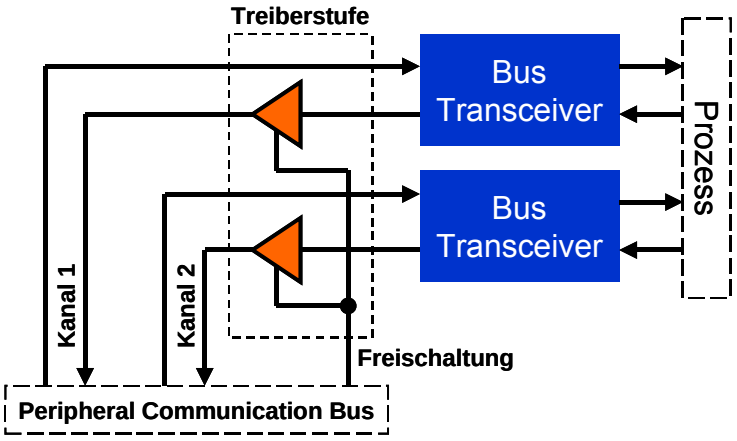

Abbildung 4.11: Schematische Darstellung einer Automotive-Bus-Karte

Alle Automotive-Buskarten werden an einem parallelen Bus, dem Pripheral Communication Bus (PCB) betrieben. Dieser ist außerdem auf alle Interface-Karten geführt und dort mit Pins des FPGAs verbunden. Da die Transceiver-Bausteine nach dem Einschalten der Versorgungsspannung bereits einen logischen Pegel an ihren Ausgängen führen, können Probleme auftreten, weil

- zwei gleiche Karten in benachbarten Slots stecken und die selben Busleitungen auf dem PCB belegen oder
- die FPGAs auf den Interface-Karten noch nicht konfiguriert sind.

Im ersten Fall würden die Ausgänge über den Bus einen Kurzschluss produzieren, da zwei Treiber gleichzeitig auf eine Verbindungsleitung zugreifen. Im zweiten Fall liegen schon vorab Signale auf den zu diesem Zeitpunkt noch nicht konfigurierten Ports eines FPGAs, was im ungünstigsten Fall zur Zerstörung des Ports führen kann. Um dies zu verhindern,

muss eine Abschaltung der Treiberbausteine vorgesehen werden. Sollte nach Auslesen der Karten-IDs keine Überschneidung der Ressourcen vorliegen, können die Treiber von der Software nach Programmierung der FPGAs frei geschaltet werden.

4.1.5.4 Steckplätze

Der modularer Aufbau erfordert, Systemteile in Form von Steckkarten hinzufügen, austauschen oder entfernen zu können, um die Adaptierung an die Erfordernisse der Applikation und der Umgebung zu erreichen. Die Bereitstellung dieser Möglichkeiten wird durch die entsprechenden Vorkehrungen, d.h. durch den Einsatz von Steckplätzen (Stecker-Buchsen-Kombinationen), innerhalb des Systems sichergestellt. Die Steckplätze werden von der Größe her unterschiedlich ausgelegt, um eine Unterscheidung zu erreichen und den Anforderungen hinsichtlich der benötigten Anschlüsse gerecht zu werden. Die benötigten Pinzahlen für jede Karte sind in Tabelle 4.7 zusammengefasst.

Kartentyp	Signale/Busse	Pinzahlen	Summen
Interface-Karte	System Bus (SB)	43	**180**
	Module Interconnect Bus (MIB)	32	
	Peripheral Comm. Bus (PCB)	16	
	I/O-Bus (IOB)	20	
	Configuration Channel (CC)	6	
	Info-Bus (IB)	5	
	LVDS-Takt	2	
	Spannungsversorgung	20	
	Masse	36	
I/O-Karte	I/O-Bus (IOB)	10	**24**
	Info-Bus (IB)	4	
	Spannungsversorgung	5	
	Masse	4	
Automotive-Karte	Peripheral Comm. Bus (PCB)	16	**30**
	Info-Bus (IB)	5	
	Spannungsversorgung	5	
	Masse	4	

Tabelle 4.7: Benötigte Pinzahlen der einzelnen Steckkarten

Stecker und Buchsen sind in verschiedenen Größen, Pinzahlen/Pinreihen und Rastermaßen (die gängigsten sind derzeit 2,54mm, 2,0mm und 1,27mm) erhältlich, allerdings ergeben sich dabei diverse Einschränkungen wie

- die Baugröße: bei 180 Pins, 4 Pinreihen und einem Rastermaß von 2,54mm ist der Steckverbinder bereits mindestens 114mm lang – für ein kleines System schon zu groß
- die Verfügbarkeit: Steckverbinder mit Pinzahlen über 100 sind recht selten zu finden, in den Abmessungen/den Pinreihen zu groß oder zu teuer und daher für die angestrebte Lösung nicht geeignet.
- die Routbarkeit auf der Platine: Bei mehr als 4 Reihen muss mindestens eine 6-Lagen-Platinen verwendet werden, die neben den Kosten für die Stecker auch den Herstellungspreis der Platine in die Höhe treibt
- die Stabilität: Anstelle von SMD-Steckverbindern kamen nur solche zur Durchkontaktierung in Frage, um eine besser mechanische Stabilität zu gewährleisten
- der Preis: Steckverbinder mit Pinzahlen über 150 sind sehr teuer

Der Kompromiss aus Größe, Preis, Verfügbarkeit usw. waren letztlich einheitliche Stecker und Buchsen mit einem Rastermaß von 2,0mm von Samtek [Samt01], angepasst auf die benötigten Pinzahlen. Für die Interface-Karte wurde ein 4-reihiger Stecker mit 200-Pins verwendet und für die I/O- und Automotive-Karten jeweils eine 2-reihige 30-Pin-Variante des gleichen Typs.

4.1.6 Identifikation der Steckkarten

Das modulare Konzept der Plattform erlaubt das Einstecken von verschiedenen Karten zur Erweiterung und Anpassung. Da nicht alle Karten die selben Eigenschaften besitzen, muss eine Möglichkeit der Unterscheidung geschaffen werden, um Fehler durch falsch eingesteckte oder nicht verwendbare Karten zu vermeiden. Eine Kodierung kann durch entsprechende Stecker vorgenommen werden (z.B. durch fehlende Pins am Stecker, duale Codierung mit Spannungspegeln auf mehreren Pins), jedoch reicht diese Maßnahme nicht aus, wenn mehrere Karten zwar den gleichen Stecker benutzen, aber andere Funktionen umsetzen. Dies gilt für alle steckbaren Karten im System – Interface-Karten können verschiedenen FPGAs, Automotive-Karten können verschiedene Bustreiber enthalten und bei den I/O-Karten besteht neben den diversen Funktionen auch noch die Variation der Anzahl Ein- und Ausgabekanäle.

Um dennoch eine Unterscheidbarkeit zu erreichen, müssen die Steckkarten elektronisch kodiert werden, d.h. jede Karte muss eine Identifikation in Form einer Nummer oder einer mehrstufigen Nummernfolge (Kartenklasse, ID innerhalb der Klasse, usw.) bekommen. Diese sollten auf der Karte abgelegt sein und von Systemseite her ausgelesen werden können. Dabei scheiden so genannte Seriennummern-Chips aus, da deren Code fix und daher nicht änderbar ist, sodass keine eigenen Daten hinterlegt werden können. Die beste Wahl zur Speicherung von Identifikationsnummern sind serielle EEPROMs. Sie können anwenderspezifische beschrieben werden, nehmen nur wenig Raum (Platinenfläche) ein und können über ein definiertes Protokoll ausgelesen werden. EEPROMs dieser Art gibt es mit verschiedenen Protokollen, die sich jedoch nicht alle für einen solchen Einsatz eignen. In Tabelle 4.8 sind die gängigsten Protokolltypen aufgeführt und bewertet.

Protokoll	Vorteil	Nachteil	Wertung	Auswahl
I^2C	zwei Leitungen	Adressvergabe im Chip	-	
OneWire	eine Leitung, bidirektional betrieben	Protokoll umständlich, schwer realisierbar, Adressvergabe im Chip	-	
SPI	Getrennter Freigabeeingang (#CS)	vier Leitungen	++	✓
MicroWire	ähnlich SPI	nicht so weit verbreitet	+	

Tabelle 4.8: Serielle EEPROMs zur Kartenidentifikation

Die Auswahl des passenden Typs hängt unmittelbar von der Busfähigkeit des Protokolls, bzw. der Bausteine und der Anzahl an benötigten Leitungen ab. Da alle Protokolle diese Eigenschaft unterstützen, muss die Unterscheidung anhand der Adressierung innerhalb des Busverbunds erfolgen. Bei den ersten beiden Arten (I^2C und OneWire), können die Speicher zwar einzelnen adressiert werden, jedoch wird dies hinfällig, wenn mehrere Speicher des gleichen Typs im selben Bussegment verwendet werden. Dabei schaffen auch die teilweise vorhandenen Adresspins am EEPROM nur bedingt Abhilfe, da damit der mögliche Adressraum eingeschränkt ist. Damit bleiben nur noch EEPROMs mit SPI-Interface übrig. Diese werden alle an einen Bus geschaltet und über getrennte Freigabeleitungen aktiviert, wodurch eine Adressierung mittels Protokoll entfällt. Die damit verbundene erhöhte Anzahl an Leitungen muss zur Realisierung in Kauf genommen werden.

4.1.7 Konfiguration der FPGAs

Die Konfiguration der eingesetzten Xilinx-Virtex-II-FPGAs kann auf verschiedene Arten geschehen (siehe Kapitel 3.1.4). Die FPGAs bieten dazu diverse Schnittstellen an, die je nach Applikationsumfeld eingesetzt werden können. Sie unterscheiden sich grob in serielle und parallele Modi, sowie in der Art des Protokolls, das zur Programmierung – teilweise auch zu Zurücklesen der Informationen aus dem FPGA – verwendet wird. Dadurch ergeben sich entsprechende Anforderungen an die Programmierstelle, welche die Konfigurationsdaten in den FPGA schreibt. In Tabelle 4.9 sind die Schnittstellen gegenübergestellt und bewertet.

Interface	Vorteil	Nachteil	Wertung	Auswahl
Master-Serial	wenige Leitungen	keine Kontrollmöglichkeit von außen	-	
Slave-Serial	voll kontrollierbar, einfaches Protokoll, wenige Leitungen	Geschwindigkeit gering	++	✓
Boundary-Scan (JTAG)	schnell, Standard-Schnittstelle, wenige Leitungen	umfangreiches Protokoll, schwer realisierbar	+	
Slave-Parallel	Geschwindigkeit hoch	Leitungsanzahl zu hoch	-	

Tabelle 4.9: Konfigurationsmodi für Xilinx-FPGAs

Verfahren, bei denen der FPGA die Kontrolle über den Programmiervorgang hat, können nicht verwendet werden, da

- der Zeitpunkt und die Reihenfolge (bei mehreren FPGAs auf der Plattform) der Programmierung nicht vorgegeben werden kann,
- die Daten in einem separaten Speicher (je einem pro FPGA oder einem gesamten für alle FPGAs) zur Verfügung gestellt werden müssten, damit sie die FPGAs beim Hochfahren selbsttätig holen können und
- vorab kein Ressourcen-Check und keine Fehlerprüfung auf der Plattform ausgeführt werden können, die eine mögliche Fehlkonfiguration (d.h. die aktuelle Konfiguration des/der FPGAs passt nicht zu den eingesteckten I/O-Karten) verhindern.

Dadurch kommt eine Konfiguration nach „Master-Serial“ nicht in Frage. Wegen der begrenzten Ressourcen auf der Plattform, sind auch parallele Modi, wie „Slave-Parallel“, nicht umsetzbar, da sie einerseits viele Steckerpins an den Karten belegen und andererseits durch die parallele Verlegung von Busleitungen enorme Platinenfläche okkupieren. Die beiden seriellen Verfahren sind im Grunde beide gleich gut geeignet und werden daher auch für die Programmierung vorgesehen. Durch entsprechende Lötbrücken auf den Platinen kann eine Umschaltung vorgenommen werden. Umgesetzt wurde aber zunächst nur „Slave-Serial“, da hier das einfachere Protokoll und die damit verbundene kürzere Implementierungsphase den Ausschlag gab. Für „JTAG“ spricht die Möglichkeit, auch dynamisch partielle Re-Konfigurationen auszuführen, was mit „Slave-Serial“ nicht möglich ist. Durch die Verfügbarkeit der ICAP-Schnittstelle im FPGA (statischer Slot) sind nunmehr zwei verschiedene Verfahren zur partiell, dynamischen Re-Konfiguration möglich: Zum einen über die extern anzuschaltenden JTAG-Schnittstelle als Ersatz der seriellen, oder aber über den FPGA-internen ICAP-Port, der über das parallele Businterface mit Daten versorgt werden kann.

4.1.8 Zentrale Takterzeugung und -verteilung

Um zwischen den einzelnen FPGA-Steckkarten einen synchronen Busbetrieb zu ermöglichen, wird eine zentrale Takterzeugung implementiert. Die Frequenz lässt sich, je nach Anforderung des RP- oder HiL-Modells, variieren, um auch während der Laufzeit des Systems

eine Adaptierung vornehmen zu können. Zur Erzeugung der gewünschten Frequenz wird ein einstellbarer Taktgenerator [PCK12429] eingesetzt. Er kann Frequenzen zwischen 25 und 400MHz synthetisieren und in Schritten von einem Megahertz über ein differentielles PECL-Interface ausgeben. Zur Erzeugung eines Referenztaktes wird ein externes 16 MHz-Quarz verwendet. Die Frequenzerzeugung des Ausgangstaktes erfolgt über einen VCO (Voltage Controlled Oscillator). Durch einen internen PLL und einen Phasendetektor wird die Frequenz auf das m-Fache der internen Referenzfrequenz eingestellt. Zusätzlich besitzt der Baustein einen einstellbaren Ausgangsteiler, der zusammen mit dem Teiler für die interne PLL die Frequenz in 1MHz-Schritten einstellbar macht. Die notwendigen Einstellungen für die beiden Teiler können über einen parallelen Port voreingestellt werden oder über eine serielle Datenverbindung während der Laufzeit erfolgen. Bei Systemstart wird eine über DIP-Schalter vorgegebene Frequenz als Basistakt verwendet, die während des Betriebs der Plattform die Frequenz über das serielle Protokoll verändert werden.

4.1.9 Plattformkontrollinstanz

Die oben beschriebenen peripheren Funktionen zur Kartenidentifikation, des Taktgenerators und der Programmierung der FPGAs auf den Interface-Karten benötigen eine Vielzahl an Steuerleitungen, die bedient werden müssen. Der zur Kartenidentifikation eingesetzte SPI-Bus verwendet vier Leitungen zur Kommunikation, zuzüglich je einer zur Aktivierung des entsprechenden EEPROMs auf jeder Karte. Für die Ansteuerung des programmierbaren Taktgenerators sind lediglich drei Anschlüsse vorzusehen, wohingegen bei den Konfigurations-Interfaces der FPGAs je sechs Anschlüsse zu bedienen sind. Geht man zur Abschätzung der nötigen Ressourcen von einer kleinen Plattform mit vier Interface-Karten mit je einem FPGA, sechs I/O-Karten und vier Automotive-Karten aus, erhält man in der Summe folgende Pinanzahl:

Anzahl	Kartentyp	Komponenten	Steuerleitungen	Summe
4	Interface-Karten	FPGA + EEPROM	6+5	44
6	I/O-Schnittstellen	EEPROM	5	30
4	Automotive-Schnittstellen	EEPROM	5	20
1	Basis-Board	Taktgenerator	3	3
Summe				**97**

Tabelle 4.10: Abschätzung der Steuerleitungen

Eventuell an einem Prozessor verfügbare, freie Ein- oder Ausgabe-Kanäle reichen dazu bei weitem nicht aus, um den gesamten Bedarf zu decken. Zur Erweiterung der vorhandenen Portpins, stehen verschiedene Möglichkeiten zur Auswahl, die in Tabelle 4.11 zusammengestellt und bewertet sind.

Mit Hilfe von Multiplexern bzw. DeMultiplexer könnte eine Umschaltung zwischen den Leitungen erzielt werden, wobei jedoch der Aufwand in keinem Verhältnis zum tatsächlichen Nutzen stünde. Sie belegen, je nach Anzahl an Kanälen, viel Fläche und sind in ihrer Funktion stark beschränkt. Das Gleiche gilt für Porterweiterungen (auch Portexpander), mit denen die Anzahl verfügbarer I/O-Ports erhöht werden kann. Diese haben zudem noch den Nachteil, dass sie seriell angesteuert werden müssen. CPLDs stellen ihrer Programmierbarkeit wegen eine weit aus bessere Alternative dar. Allerdings sind deren Logikressourcen sehr stark begrenzt.

Art	Vorteil	Nachteil	Bewertung	Auswahl
(De)Multiplexer	Einfache Zuordnung einer Leitung auf verschiedene Baugruppen	hoher Platzbedarf auf der Platine, feste Funktionalität	-	
Porterweiterung	einfache und kostengünstige Erweiterung verfügbarer Ports	serielle Ansteuerung, langsam, hoher Ansteueraufwand, feste Funktionalität	-	
CPLD	programmierbar	Logikressourcen sehr stark eingeschränkt	+	
FPGA	Funktionalität frei programmierbar, ausreichende Logikressourcen		++	✓

Tabelle 4.11: Möglichkeiten zur Porterweiterung

Im Hinblick auf eine auch später noch modifizierbare Lösung, bietet sich auch in diesem Fall der Einsatz eines FPGAs – allerdings mit weit geringerem Ressourcenangebot als die auf den Interface-Karten – an. Der entscheidende Vorteil liegt hier in der Anzahl an verfügbaren I/O-Pins, die außerdem noch in ihrer Funktion frei gestaltet und flexibel den externen Komponenten zugeordnet werden können. Da der FPGA, bzw. die interne Logik, mit Daten versorgt werden muss, ist zudem noch ein Bus-Interface zu implementieren, über das der Prozessor schreibend und lesend zugreifen kann. Mit einem Spartan-II-FPGA (z.B. als XC2S200-PQ208 mit 140 freien I/O-Pins) als so genanntem „Platform-Control-FPGA“ stehen genügend Anschlüsse zur Verfügung, um die Anforderungen abzudecken. Darüber hinaus ist er mit 200.000 Gattern gut ausgestattet, um die Logik zum Auslesen der Kartenidentifikationsdaten, zur Ansteuerung der PLL-Schaltung für die globale Taktversorgung der Plattform und zur Programmierung der FPGAs auf den einzelnen Interface-Karten aufzunehmen.

FPGA	Gehäusetyp und maximal verfügbare I/Os					
	VQ100	TQ144	CS144	PQ208	FG256	FG456
XC2S15	60	86	86	-	-	-
XC2S30	60	92	92	132	-	-
XC2S50	-	92	-	140	176	-
XC2S100	-	92	-	140	176	196
XC2S150	-	-	-	140	176	260
XC2S200	-	-	-	140	176	284

Tabelle 4.12: Freie I/Os der Spartan-II-FPGA-Serie von Xilinx

Da der Hauptprozessor auf die internen Funktionen zugreifen können muss, ist der Spartan-II an den System Bus angeschlossen. Zur Anbindung des Board-Control-FPGAs an den Systembus des MICRO-9 ist eine zu dem Local-Bus des verwendeten ARM-Prozessors kompatible Schnittstelle auf dem FPGA implementiert. Alle Funktionsblöcke des Board-Control-FPGAs sind über Register oder Dual-Port-BlockRAMs mit dem Systembus verbunden. Eine separate Busansteuerung decodiert die Adresse bei externen Zugriffen und schaltet das gewünschte Modul auf den Bus. Darin enthalten ist außerdem die Kopplung der synchronen BlockRAM-Bausteine an den asynchronen Local-Bus des Prozessors. Die im FPGA umgesetzten Module sind in Abbildung 4.12 dargestellt.

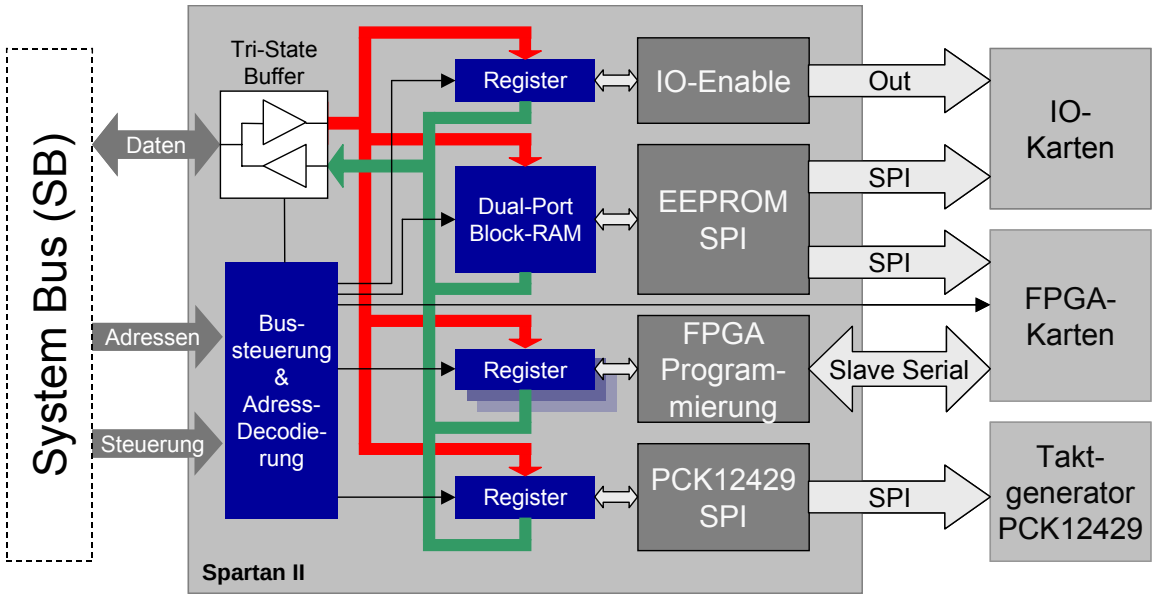

Abbildung 4.12: Aufbau und Funktionsüberblick des Board-Control-FPGAs

4.1.10 Bussystemarchitektur

Im Gegensatz zu den kommerziellen Systemen, in denen gemeinhin nur ein Bussystem vorhanden ist, um mit allen peripheren Baugruppen zu kommunizieren, sollen in dem hier vorgestellten Ansatz mehrere zum Einsatz kommen. Die Verwendung erfolgt dabei getrennt nach Aufgaben, wodurch eine Optimierung der einzelnen Kommunikationswege hinsichtlich Bandbreite, Durchsatz, Busbreite, Protokoll, usw. vorgenommen werden kann.

4.1.10.1 System Bus (SB)

Der wichtigste Bus innerhalb eines jeden eingebetteten Systems ist der Systembus (SB), der den Prozessor mit allen Busteilnehmern verbindet und über den der allgemeine Datenaustausch stattfindet (Abbildung 4.13).

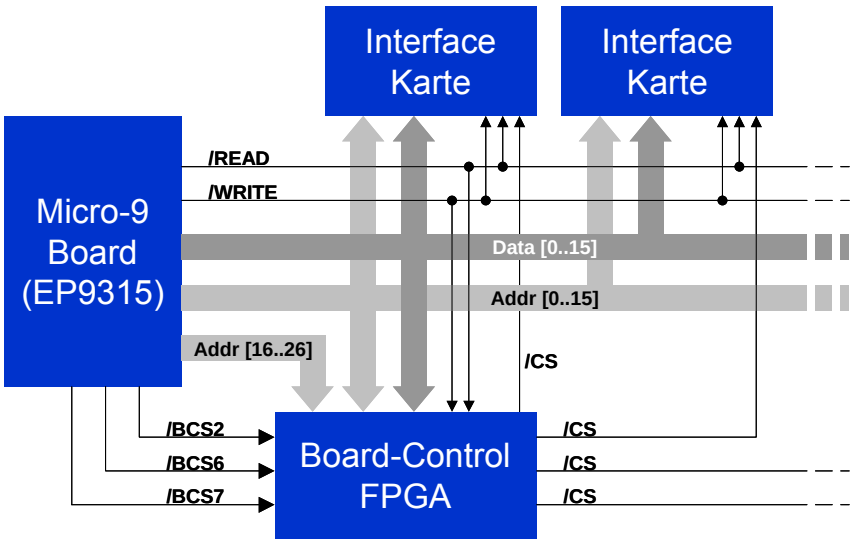

Abbildung 4.13: Aufbau des System Bus (SB)

Dieser wird auch in diesem Konzept als paralleler Bus beibehalten, wobei anstelle der Standardbusteilnehmer die FPGA-Karten und der Board-Control-FPGA (Kapitel 4.1.9) rücken, d.h. die Busschnittstellen im FPGA nachgebildet werden müssen. Die Datenwortbreite wird für den hier vorgestellten prototypischen Aufbau auf 16bit begrenzt. Den Ausschlag dazu gaben die verwendeten FPGAs (Virtex-II-1000BG 256 mit 172 freien I/O-Ports), die nicht genügend Pins für eine 32bit-Anbindung bereitstellen können, ohne dabei auf andere Komponenten, wie z.B. ein externes RAMs am FPGA auf der Interface-Karte, verzichten zu müssen. Diese Einschränkungen gelten auch für den Adressbus, von dem nur 16 Bits auf die Interface-Karten geführt sind. Sie können durch die Verwendung eines anderen Gehäusetyps, der entsprechend mehr freie I/Os bereitstellt, eliminiert werden. Allerdings ergeben sich dadurch Änderungen im Platinenlayout, da mindestens zwei weitere Lagen zur Leitungsführung vom und zum FPGA benötigt werden.

Der Adressbus ist in seiner vollen Breite von 26 Bit auf den Board-Control-FPGA geführt. Die einzelnen FPGA-Steckkarten werden jeweils mit 16-Bit-Adressen und einem Chip-Select-Signal angebunden. Zur Datenübertragung steht an allen Busteilnehmern die untere Hälfte des Datenbusses mit einer Breite von 16 Bit zur Verfügung. Theoretisch können damit 128 KByte an Speicher angesprochen werden. Dies reicht zur Adressierung der internen Speichers der Virtex-II-FPGAs bis hin zum V2C2000 aus. Bei der Verwendung größerer FPGAs kann nicht mehr direkt auf den kompletten Speicher von außen zugegriffen werden. Allerdings kann dies mit einem Bank-Switch (Umschalten zwischen Speicher-Bänken) gelöst werden. Außerdem ist zu beachten, dass die RAM-Blöcke über den gesamten Chip verteilt sind und die Ankopplung an das Businterface sehr viele Routing-Ressourcen belegt und damit nicht unbedingt sinnvoll ist, da dies die Implementierung und Verdrahtung anderer Logik nur noch eingeschränkt erlaubt.

Der Systembus (SB) wird mit 25 MHz betrieben und erreicht so eine maximale (Peak-) Datenrate von 400 Mbit/s, bzw. 50 MByte/s. Diese liegt über den Datentransferraten der kommerziellen Systeme, die mit 20 bis 30 MByte/s (siehe [dSPA06]) arbeiten. Alternativ dazu wäre auch eine serielle anstelle einer parallelen Verbindung zwischen dem Prozessor und den Busteilnehmern denkbar. Sollte diese den gleichen Datendurchsatz bieten, wäre im Falle einer 1-Bit-Datenleitung ein Bustakt von 400 MHz nötig. Für solche Taktraten ist allerdings ein erheblicher Mehraufwand in Bezug auf das Platinen-Layout zu verwenden und stünde in keinem Verhältnis zu dem Nutzen, nämlich der Einsparung von Pins. Außerdem bestehen enorme Schwierigkeiten, die ausgewählten FPGAs mit Frequenzen von über 200MHz an ihren I/O-Portpins zu beaufschlagen. Eine serielle Datenübertragung wurde daher nicht weiter verfolgt.

4.1.10.2 Module Interconnect Bus (MIB)

Um eine Kommunikation oder lediglich einen Signalaustausch zwischen zwei oder mehreren Interface-Karten zu ermöglichen, müssen diese miteinander verbunden sein. Dies ist zwar grundsätzlich schon über den Systembus gegeben, jedoch ist dort der Prozessor der Master, ohne dessen Eingreifen kein Bustransfer möglich ist (Single-Master-Betrieb). Daher muss auf ein Verbindungsnetz ausgewichen werden, das zunächst keinen dedizierten Master und damit auch kein definiertes Protokoll aufweist. Es soll dazu alle Interface-Karten in gleichem Maße verbinden und eine freie Belegung – auch durch ein später zu implementierendes Protokoll – ermöglichen. Die Anzahl der verfügbaren Leitung sollte sich an der des parallelen Systembus' orientieren, um einen zweiten unabhängigen Kommunikationsweg (Modul Interconnect Bus, MIB) für etwaige äquivalente Sub-Systeme erstellen zu können.

Die maximale Anzahl an Signalleitungen für den MIB ist das Ergebnis einer Schätzung. Für den Adressbus (AB) sind 8 Bit, für den Datenbus (DB) eine Breite von 16 Bit vorgesehen. Zur Steuerung des Buszugriffs stehen weitere 8 Leitungen (SB) zur Verfügung. Daraus ergibt sich eine Busbreite, die an dessen Auslegung angelehnt ist und somit dem Systembus entspricht. Damit entsteht die Möglichkeit ein dem Systembus ähnliches Subsystem aufzubauen.

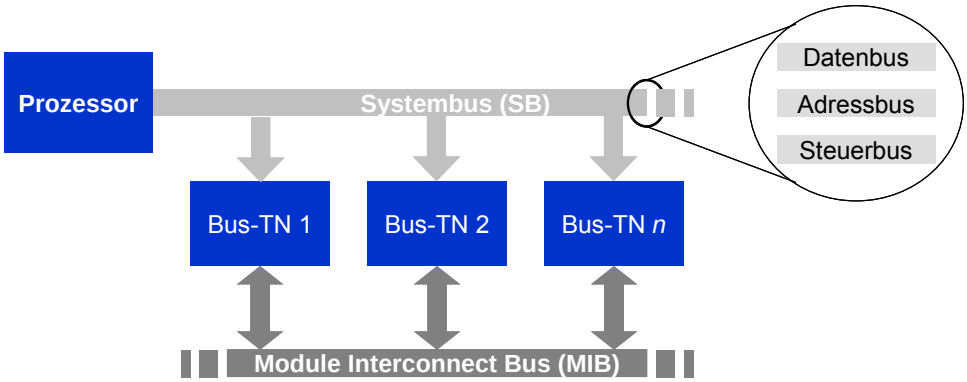

Abbildung 4.14: Kopplung von Interface-Karten über den Module Interconnect Bus (MIB)

Da der Bus zunächst nur aus einem Bündel Leitungen besteht, können aber auch serielle Links oder einzelne Signalleitungen transportiert werden. Die Verwendung ist nicht festgelegt und dem Benutzer frei überlassen. Grundsätzlich nur serielle Verbindungen (Links), Signalrichtungen durch Treiberbausteine oder feste Übertragungsprotokolle zu etablieren, schränkt den Freiraum der möglichen Implementierung von vorne herein zu weit ein, was einer maximal flexiblen Lösung widerspricht.

4.1.10.3 Peripheral Communication Bus (PCB)

Die Kommunikation mit anderen eingebetteten Systemen oder mit intelligenten Sensoren und Aktoren stellt eine weitere wichtige Schnittstelle eines Testgerätes dar. Über entsprechende Bussysteme, wie z.B. CAN oder LIN im Automobilbereich, sind solche Verbindungen möglich. Da in dieser Arbeit eine strikte Trennung von Logik und Prozesskopplung eingeführt wird, muss eine Kopplung zwischen dem eigentlichen Protokoll-Controller (für CAN oder LIN) und dem Bus-Transceiver hergestellt werden. Für diesen Zweck wird ein Peripheral Communication Bus (PCB) eingeführt, der parallel alle Interface-Karten konnektiert und diese Leitungen dann auf die entsprechenden Steckplätze der Transceiver-Bausteine führt. Die freie und flexible Zuordnung von Karte und Transceiver erhöht die Freiheitsgrade der Plattform, da sie das Umstecken von Karten vermindert und die Umschaltung zur Laufzeit unterstützt.

4.1.10.4 I/O-Kanäle

Die Signalisierung zwischen Interface- und I/O-Karten findet über die I/O-Kanäle statt. Die I/O-Kanäle stellen das Bindeglied zwischen dem auf dem FPGA implementierten Modell und der physikalischen Prozessankopplung auf der Schnittstellenkarte dar. Die Übertragung erfolgt dabei ausschließlich auf digitaler Ebene. Die Umsetzung in analoge Größen wird – wenn nötig – auf der I/O-Karte vorgenommen. Dazu können auf den Karten beliebige Treiberbausteine eingesetzt werden.

Von jeder Interface-Karte kann mindestens eine I/O-Karte angesprochen werden. Die ersten beiden FPGA-Karten (Slot 1 und Slot 2 in Abbildung 4.22) bieten eine Verbindung für je zwei, die letzten beiden jeweils für nur eine I/O-Karte. Die Steckplätze sind, ebenso wie die bereitgestellten Leitungen, den Interface-Karten fest zugeordnet. Pro I/O-Karte sind jeweils zehn Leitungen vorgesehen. Dies ermöglicht den alleinigen Zugriff des FPGAs auf die vorhandenen Ressourcen der Schnittstellenkarte. Bei der Leitungsführung wurde besonders auf kurze Wege geachtet, um eine schnelle Datenübertragung zu begünstigen. Die Verbindungen beginnen im unteren Bereich des 200-poligen Interface-Karten-Steckers und enden im oberen Teil der I/O-Slots.

4.1.10.5 Info Bus (IB) und Konfigurationskanäle (Config. Channels, CC)

Die bereits beschriebene Identifikation der Steckkarten und die Programmierung der FPGAs wird über getrennte Busse vorgenommen. Im ersten der beiden Fälle handelt es sich um serielles Protokoll (SPI), das nicht direkt über den Systembus abgefragt werden kann. Eine Seriell-Parallel-Umwandlung, nur zur Nutzung der bereits vorhandenen Ressource Systembus müsste so auf jeder Karte durchgeführt werden (Abbildung 4.15) und ist zu aufwändig und zu teuer. Somit kommt keine andere Alternative als die Implementierung eines getrennten, aber seriellen Busses in Frage. Dabei wird, wie in Abbildung 4.16 dargestellt, die Seriell-Parallel-Wandlung zentral erledigt und die EEPROMS aller Karten über einen 5-Bit breiten Bus angeschlossen. Die so entstandene zentrale Stelle erhält einen Buszugang, so dass die Daten aller EEPROMs darüber von außen gelesen oder auch geschrieben werden können.

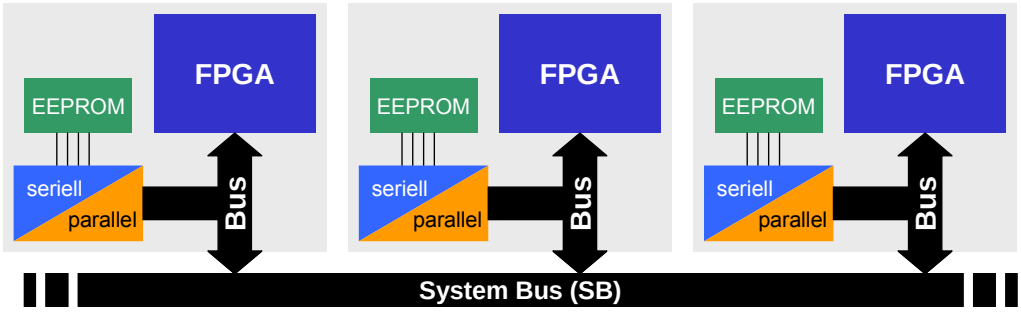

Abbildung 4.15: Ankopplung der EEPROMs (Variante 1)

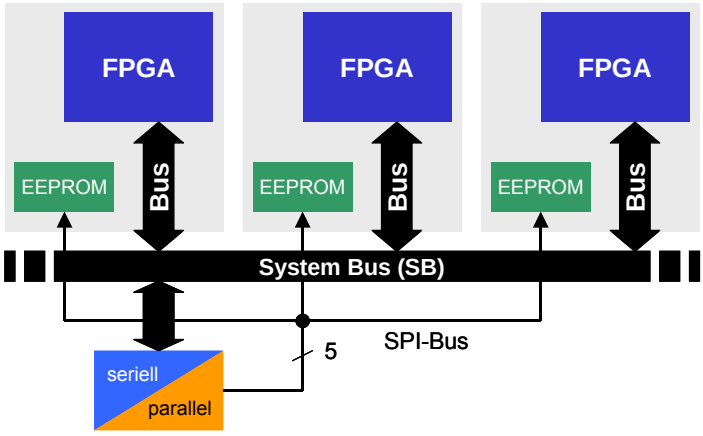

Abbildung 4.16: Ankopplung der EEPROMs (Variante 2)

Im zweiten Fall ist das Problem ähnlich. Da der FPGA zu Systemstart nicht konfiguriert ist, muss er zunächst konfiguriert werden, bevor dessen Anschluss an den Systembus verwendet werden kann. Dazu wäre auf jeder Interface-Karte eine Programmierinstanz nötig, die den Datenstrom entgegennimmt in das serielle Konfigurationsprotokoll umwandelt (z.B. SlaveSerial oder JTAG) und damit den FPGA initialisiert, will man dazu den Systembus verwenden (Abbildung 4.17). Auch in diesem Fall gilt, dass die dezentrale Lösung zu teuer ist, da pro Karte eine Komponente nötig ist, die das Programmieren durchführt. Damit wird hier auf die Verwendung des System Busses verzichtet und jeder FPGA mit Hilfe eines separaten Kanals (Config-Channel, CC) an eine zentrale Programmierinstanz angeschlossen. Letztere ist an den System Bus gekoppelt und ermöglicht den Zugriff auf jeden Kanal.

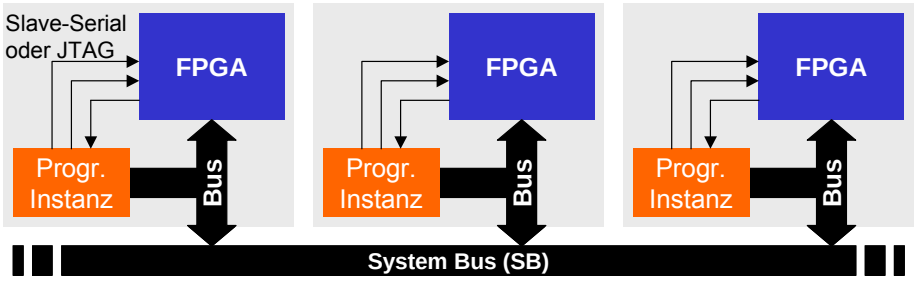

Abbildung 4.17: Dezentrale Programmier-Interfaces auf den Interface-Karten

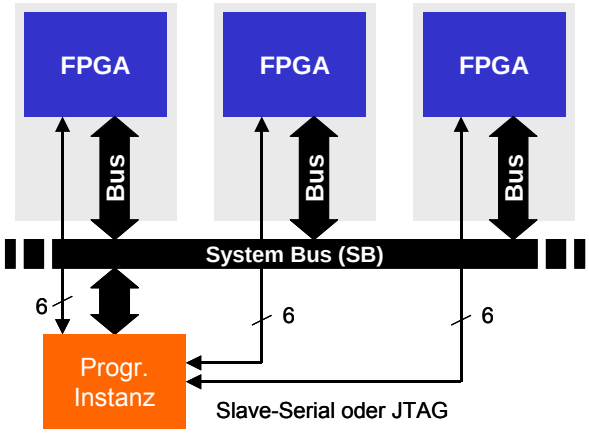

Abbildung 4.18: Zentrales Programmier-Interface für Interface-Karten-FPGAs

Da eine zentrale Lösung anstelle einer dezentralen für beide Probleme ausgewählt wurde, ist es sinnvoll, die Funktionen in einer einzigen Komponente (z.B. einen FPGA) zu bündeln. Dies wurde mit der Einführung der Plattformkontrollinstanz (Board-Control-FPGA) erreicht, der unter Kapitel 4.1.9 beschrieben ist.

4.1.11 Speicherung von Konfigurations- oder Messdaten

Die Datenspeicherung in Testsystemen ist eine der wichtigsten Aufgaben. Jeder Testlauf – insbesondere bei HiL-Tests – ist praktisch wertlos, wenn die während der Testphase aufgenommenen Messwerte nachher nicht mehr verfügbar sind. Daher muss bei der Implementierung eines Speichers entsprechender Wert auf die Art, dessen Einsatzmöglichkeiten und die geltenden Randbedingungen desselben gelegt werden. Grundsätzlich kommen verschiedene Varianten in Frage, die hier in Tabelle 4.13 aufgelistet und bewertet sind.

Die Gegenüberstellung der Merkmale der einzelnen Speichertypen zeigt, dass nicht alle für den Einsatz in einem RP-/HiL-System geeignet sind. Ein RAM ist grundsätzlich in einem solchen System verbaut, jedoch hat es den Nachteil, dass bei Spannungsverlust die Daten nicht gehalten werden. Ein Flash-Speicher bietet dagegen einen besseren Schutz. Auch er ist normalerweise bereits im System vorhanden (und beinhaltet z.B. die Firmware), allerdings müsste dann Größe anders ausgelegt werden, wobei nicht vorhersagbar ist, wie viel an Speicherplatz einmal benötigt werden wird. Außerdem können bei Flash-Speichern nur eine begrenzte Anzahl an Schreib-Zyklen ausgeführt werden, ohne diesen irreparabel zu zerstören. Eine Festplatte scheidet im Hinblick auf ein kleines und mobiles System schon alleine wegen ihrer Baugröße und den mechanischen Teilen aus. Letztere können bei hohen Beschleunigungen (z.B. bei Vibrationen bis zu 20G oder Stößen bis zu 100G, [VDE06]) Schaden erleiden und den Speicher unbrauchbar machen, wenn das System im Fahrzeug eingesetzt wird. Memory-Cards stellen in der obigen Auswahl die beste Alternative dar. Sie sind klein, bieten aber dennoch große Speichervolumina (derzeit 1MB bis zu 16GB), die dadurch leicht an einen sich ändernden Speicherbedarf angepasst werden können und um den Fak-

tor 125 höher liegen als die Speicherressourcen (typ. 128MB SDRAM) der gängigen HiL-Systeme. Sie gehören zwar ebenfalls zu den Flash-Speichern und sind daher begrenzt wiederbeschreibbar, haben aber den Vorteil, dass sie austauschbar und somit auch mobil einsetzbar sind. Die eingeschränkte Schreibgeschwindigkeit von ca. 10 Mbit/s (je nach Karte) stellt für die meisten Applikationen keine Einschränkung dar.

Typ	Vorteil	Nachteil	Wertung	Auswahl
RAM	preiswert, im System bereits vorhanden	flüchtig, zu geringe Ressourcen verfügbar	-	
Flash	vorhanden, nicht flüchtig	Speichervolumen und Schreibzyklen begrenzt, fest integriert	+	
Festplatte	großes Speichervolumen, nicht flüchtig	mechanische, rotierende Teile, Größe, Gewicht, fest eingebaut	+	
Memory-Card	steckbar, großes Speichervolumen, nicht flüchtig	eingeschränkte Schreibgeschwindigkeit	++	✓

Tabelle 4.13: Medien zur Konfigurations- und Messdatenspeicherung

Beispielrechnung:

Würde das RP-HiL-System als CAN-Daten-Logger betrieben, das an einen CAN-Bus mit einer Baud-Rate von 500kbit/s angeschlossen ist und ununterbrochen CAN-Botschaften mit einer durchschnittlichen Länge von 140 Bit (Standard-ID und 8Bytes Daten) aufnimmt, so könnte bei der Verwendung einer Memory-Card mit 16GB Speichervolumen, eine Zeitspanne von 4,4 Tagen überbrückt werden.

4.1.12 Temperaturbereich

Einer der wichtigsten Aspekte im Automobilbereich ist der weite Temperaturbereich, dem die dort eingesetzten Geräte genügen müssen. Dieser reicht von –40°C bis 125°C und ist damit ein Maß für die in den Geräten verwendeten Bauteile, die in der kompletten Spanne einwandfrei und ausfallsicher funktionieren müssen. Daraus leiten sich zwei gängige Temperaturbereiche für den Einsatz im Automobil ab, nämlich –40°C bis 85°C für den Fahrzeuginnenraum und von –40°C bis 125°C für den Motorraum.

Dies gilt auch für die hier eingesetzten FPGA-Serien Virtex-II und Spartan-II, bei denen jedoch für den prototypischen Aufbau des Systems der Standard-Temperaturbereich (Commercial) von 0°C bis 85°C aus rein finanziellen Gesichtspunkten ausgewählt wurde. Beide Typen sind aber auch im erweiterten Temperaturbereich (Industrial) von –40°C bis 100°C erhältlich. Ein Austausch ist jederzeit ohne Probleme möglich.

Die derzeit kommerziell erhältlichen Testsysteme weisen dagegen unterschiedliche Temperaturbereiche für ihre Baugruppen aus. So finden sich diverse Module für Hardware-in-the-Loop-Tests, die nur zwischen 0°C und 70°C betrieben werden dürfen, während andere sogar nur in einer Spanne von 0°C bis 55°C oder maximal 40°C arbeiten. Im Bereich Rapid Prototyping liegen die Werte zwischen 5°C bis 35°C und –40°C und 85°C (siehe [ETAS06] und [dSPA06]).

4.1.13 Energieversorgung des Systems

Insgesamt werden sieben verschiedene Spannungen auf der Plattform benötigt, von denen jedoch zwei optional angeschlossen werden können, da diese lediglich für die Anpassung an den Prozess zuständig sind. Je nach Einsatzbereich können dort Spannungen zwi-

schen 0V und max. 55V angelegt werden, für Einsätze in der Automobilindustrie sind beispielsweise Werte von 12V bzw. 24V gefordert. Diese beiden Kanäle sind so konzipiert, dass deren Massen untereinander getrennt und außerdem abgekoppelt von der Plattformversorgung sind. Dadurch lässt sich eine galvanische Trennung zwischen der Logik auf der Plattform und dem Prozess erreichen, sowie ein Betrieb mit zwei unterschiedlich hohen Prozessspannungen. Für den Fall eines höheren Strombedarfs der I/O-Treiber, lassen sich die Kanäle zusammen auf einem Spannungsniveau und mit einer gemeinsamen Masse betreiben.

Durch die Verwendung zweier verschiedener FPGA-Serien auf der Plattform, sind unterschiedliche Spannungen nötig, die für deren Betrieb nötig sind. Während die Spartan-II-FPGAs 2,5V für den Core und 3,3V für die I/O-Treiber fordern, liegt die Core-Spannung bei der Virtex-II-Serie bei lediglich 1,5V. Die in den Grenzen von 1,5V bis 3,3V frei wählbare I/O-Treiberspannung wurde hier auch auf den maximalen Wert gelegt, um eine bestmögliche Kompatibilität mit den gängigen CMOS-Komponenten zu erhalten.

Für den Betrieb der Plattform selbst müssen demnach fünf Spannungen, nämlich 1,5V, 2,5V, 3,3V, 3,3V-P und 5V bereitgestellt werden. Letztere stellt die Verwendung von Standardbausteinen auf der Plattform sicher, die bisher nicht als 3,3V-Typen erhältlich sind oder gar nicht als solche auf den Markt kommen. Diese fünf Spannungen versorgen die gesamte Logik des Systems (Prozessor, Peripherie und FPGAs), wobei die 3,3V-Versorgung in zwei Stränge aufgeteilt ist – einer (3,3V-P) versorgt den Hauptprozessor, der andere den Rest der Plattform.

Aus diesen Betrachtungen ergibt sich eine Aufstellung aller auf der Plattform benötigten Spannungen, wie sie in Tabelle 4.14 zu finden ist. Da die Spannungen von außen zugeführt werden, ist die Tabelle gleichzeitig als Steckerbelegung der Plattform zu verstehen. Als Steckverbinder wurde für diesen Zweck ein 20-poliger Stromversorgungsstecker [Samt01] vorgesehen.

Pin	Name	Kurzbeschreibung
1	24V	24V für Automotive-Karten
2	12V	12V für Automotive-Karten
3	-	nicht verwendet
4	-	derzeit nicht verwendet
5	5V	5V für Steckkarten und USB
6	3V3	3,3V für Spartan-II und FPGA-Steckkarten
7	3V3-M9	3,3V für Micro-9 und Schnittstellen-ICs auf Grundplatine
8	2V5	2,5V für Core-Spannung des Spartan-II
9	1V5	1,5V für Core-Spannung der Virtex-II-Karten
10	1V2	derzeit nicht verwendet
11-12	Masse	für Automotive-Karten
13	-	derzeit nicht verwendet
14-16	Masse	
17-18	-	derzeit nicht verwendet
19-20	Masse	

Tabelle 4.14: Steckerbelegung der Plattformstromversorgung

4.1.14 Sicherheitskonzept

Zu den Überlegungen einer einfachen Handhabung und Bedienung kommen auch Sicherheitsaspekte ins Spiel, die einen weiteren Beitrag zum problemlosen Umgang mit dem RP-HiL-System leisten sollen. Die Konfiguration der Plattform und der Betrieb derselben ist ein vielschichtiger Prozess, wodurch das Sicherheitskonzept auf verschiedenen Ebenen umgesetzt wird, um eine größtmögliche Automatisierung zur Entlastung des Anwenders zu er-

zielen. Die verschiedenen Aspekte sind hier im Folgenden nach ihren Implementierungsebenen gelistet.

FPGA-Konfigurations-Software:

- Grafische Oberfläche mit Menüführung für eine einfache Zusammenstellung von Schnittstellenkonfigurationen auf der Plattform
- Überprüfung der Schnittstellenkonfigurationen auf dem Host-PC zum Ausschluss inkorrekter Implementierungen, die zur Zerstörung des/der FPGAs führen können
- Zusammenstellung von kompletten I/O-Konfigurationen für die jeweiligen FPGAs auf der Plattform, bestehend aus der FPGA-Konfiguration (enthält die Logik der Schnittstellen), der dazu benötigten modularen Hardware (I/O-Treibermodule) und entsprechender Dokumentation in Form einer XML-Datei

Bedienungs-Software:

- Automatische Ressourcenprüfung und Anzeige der aktuell vorhandenen Hardware vor Inbetriebnahme der Plattform
- Ausschluss des Downloads bzw. der Inbetriebnahme von nicht zur Hardware-Konfiguration passenden Schnittstellenkonfigurationen auf den/die FPGA(s) → I/O-Treiber passt nicht zum Modell auf dem FPGA
- Fehlerüberprüfung bei der Programmierung der FPGAs mit entsprechender Rückmeldung

Plattform-Hardware:

- Definiertes Einschalten der Treiber für Automotive-Karten zum Schutz nicht programmierter FPGAs (Verhinderung von Kurzschlüssen)
- Schutz vor Vertauschen der Einsteckkarten durch Steckerauswahl und Sicherung
- Schutz gegen Einstecken/Verwendung falscher Karten durch EEPROMs auf den einzelnen Karte zur Identifikation

Energieversorgung:

- Überwachung der auf der Plattform benötigten Spannungen und Abschaltung im Fehlerfall (bei Über- oder Unterspannung)
- Möglichkeit zur eigenständigen Spannungsan-/-abschaltung der Plattform (Verbindung über CAN oder RS232)

4.2 Konzept der Stromversorgung (Power Management Unit)

4.2.1 Abschätzung der Leistungsaufnahme der Plattform

Da die Plattform, wie in Kapitel 4.1.13 aufgezeigt, verschiedene Spannungen benötigt, die aus einer einzigen Quelle abgeleitet werden sollen, müssen pro Spannungsbereich je ein Regler verwendet werden. Um eine Auswahl der passenden Regler vornehmen zu können, ist es zunächst wichtig, die Leistungsaufnahme der Plattform abzuschätzen.

Zu diesem Zweck müssen alle Bauteile und deren Stromaufnahme in Bezug auf ihre jeweilige Versorgungsspannung betrachtet werden. Die Summe der Ströme pro Spannungsbereich und die Summe aller Ströme auf der Plattform ergibt dann einen Hinweis auf die Gesamtstromaufnahme. Folgende Formeln werden für die Abschätzung verwendet:

$$I_{3.3V} = \sum_{i=1}^{4}\left(I_{IK}(i)\right) + I_{VCCO\_S2\_max} + \sum_{k=1}^{6}\left(I_{IO-Karte}(k)\right) + I_{Auto-Karte} + I_{CLKGen}$$
$$I_{IK}(i) = I_{VCCO\_VII\_max}(i) + I_{EEPROM}(i) + I_{SRAM}(i) \quad \forall i \in (1..4)$$

Formel 4.1: Abschätzung der Stromaufnahme der 3,3V-Spannungsschine

$$I_{3.3V\_M9} = I_{M9\_max} + I_{CF} + I_{RS232} + I_{CAN}$$

Formel 4.2: Abschätzung der Stromaufnahme der 3,3V-Spannungsschine für µP

$$I_{1.5V} = \sum_{i=1}^{4} I_{VCCINT\_VII\_max}(i)$$

Formel 4.3: Abschätzung der Stromaufnahme der 1,5V-Spannungsschine

$$I_{2.5V} = I_{VCCINT\_S2\_max}$$

Formel 4.4: Abschätzung der Stromaufnahme der 2,5V-Spannungsschine

$$I_{5.0V} = I_{USB\_max}$$

Formel 4.5: Abschätzung der Stromaufnahme der 5,0V-Spannungsschine

Die Spannungen 12V und 24V sind im Automotive-Bereich gefordert. Als maximale Stromaufnahme wurden dazu jeweils 10A festgelegt. Für den Betrieb der beiden USB-Schnittstellen werden 5V, bei mindestens 2 x 500mA für die angeschlossenen Geräte, benötigt. Durch einen den Schnittstellen vorgeschalteten IC [SP2526] werden die Ströme überwacht und im Kurzschlussfall auf je 1,25A begrenzt. Um im Worst-Case-Fall undefinierte Zustände zu vermeiden, muss also sicher gestellt werden, dass die 5V-Spannung bei 2,5A noch nicht einbricht. Weiterhin sind die 5V auf die Steckplätze für die I/O- und Automotive-Karten geführt, so dass ein Strom von insgesamt 3A als untere Grenze für die Auswahl eines entsprechenden Spannungsreglers herangezogen wurde. Die Versorgungsspannung von 3,3V für das Micro9-Mikrocontrollersystem wurde als 3V3-M9 bezeichnet (Tabelle 4.17). Weitere an 3V3-M9 angeschlossene Komponenten sind die Pegelwandler [Max3232] für die RS232-Schnittstellen, ein CAN-Transceiver [SN65HVD230], eine Compact Flash Speicherkarte, verschiedene Anzeige-LEDs, sowie mehrere ICs für die Takterzeugung und -verteilung ([PCK12429], [SN65LVDS104]). Die Leistungsaufnahme des Micro9 unter Volllast ist im zugehörigen Handbuch [Cont05] mit 1,8W bzw. 545mA angegeben, der CAN-Treiber kann im Kurzschlussfall bis zu 250mA aufnehmen. Zusammen mit dem Taktgenerator (150mA), den Pegelwandlern (2mA), der Speicherkarte (<100mA) [King] und den LEDs ergibt sich somit als Mindestanforderung ein Strom von etwa 1,1A. Die Spannungsanschlüsse V_{CCO} des Spartan-II-FPGAs (XC2S200) sind auf den Pin 6 (3V3) des Steckers in Tabelle 4.14 geführt. Hieran sind auch die Versorgungsspannungen V_{CCO} und V_{CCAUX} der Virtex-II-FPGAs

(XC2V1000) sowie alle weiteren Bauteile auf den Interface-, I/O- und Automotive-Karten, die 3,3V benötigen, angeschlossen.

Die jeweiligen, während des Betriebs auftretenden Ströme der FPGAs sind in den Datenblättern ([Xil01], [Xil04]) nicht angegeben. Es existieren hierzu auf den Internetseiten des Herstellers Xilinx (www.xilinx.com) lediglich entsprechende Programme zur Abschätzung der Stromaufnahme, wie beispielsweise XPower. Allerdings erfordern diese zahlreiche detaillierte Angaben über das jeweils implementierte Design, die zum Zeitpunkt der Konzeption gemeinhin noch nicht zur Verfügung stehen. Da die Interface-Karten mit den Virtex-II-FPGAs universell und auch für zukünftige Anwendungen einsetzbar sein sollen, ist eine genaue Abschätzung auf diese Weise nicht möglich. Daher liegen den Abschätzungen die Angaben verschiedener Halbleiter-Hersteller über die maximale Stromaufnahme zu Grunde. In Auswahltabellen für Spannungsregler der Firmen National Semiconductor [Nati05] und Texas Instruments [Texa05] werden übereinstimmend für die Spartan-II-Familie maximale Ströme von 0,2-2A für V_{CCINT} und 0,05-0,5A für V_{CCO} angegeben. Da es sich hierbei noch um relativ niedrige Ströme handelt, wurden jeweils die Maximalwerte 2A und 0,5A als Grundlage für die Abschätzung verwendet. Die Angaben für Virtex-II-FPGAs sind typabhängig. Die Werte für den verwendeten FPGA XC2V1000 betragen 0,2-10A für V_{CCINT} , 0,3A für V_{CCAUX} und 0,05-3A für V_{CCO}. Da bis zu vier Interface-Karten und damit bis zu vier Virtex-II-FPGAs an die COMPASS-Plattform angeschlossen werden können, ergibt sich bei voller Auslastung somit ein maximaler Gesamtstrom von 40A für die 1,5V-Spannung V_{CCINT}. Ein solch hoher Strom lässt sich aber nicht realisieren, da hierfür weder die Leiterbahnen auf dem Basisboard, noch der Stromversorgungsstecker, der für 10A pro Pin zugelassen ist [Samt01], ausgelegt sind.

Messungen an bereits implementierten Applikationen auf Virtex-II-FPGAs ergeben üblicherweise Ströme von unter 1A zur Versorgung der Core-Spannung. Ausgehend von diesen empirischen Werten, sind Spannungs- bzw. Schaltregler verwendet worden, die zwischen mindestens 5A und maximal 10A liefern können, um genügend Reserven für aufwändigere RP- und HiL-Anwendungen zur Verfügung zu stellen. Über die V_{CCO}- und V_{CCAUX}-Spannung fließen, wie auch aus [Nati05] ersichtlich, üblicherweise geringere Ströme. Da die 3V3-Spannung aber zusätzlich noch den Spartan-II-FPGA und weitere Bauteile mitversorgt, wird auch hier ein Strombereich von 5-10A zugrunde gelegt. Alle erforderlichen Spannungen und Ströme für die COMPASS-Plattform sind zusammenfassend in Tabelle 4.15 dargestellt.

Spannung	Maximalstrom	Toleranz
12V	10A	±5%
24V	10A	±5%
5V	3A	±5%
3V3	5-10A	±5%
3V3-M9	1,1A	±5%
2V5	2A	±5%
1V5	5-10A	±5%

Tabelle 4.15: Spannungs- und Stromanforderungen

Die zulässigen Versorgungsspannungsbereiche für die FPGAs werden in den Datenblättern ([Xil01], [Xil04]) über absolute Werte angegeben und entsprechen jeweils genau einer Toleranz von ±5%. Diese Toleranz kann mit heutigen Spannungsreglern ohne erhöhten Aufwand eingehalten werden und ist auch für die Versorgung des Micro9 angegeben [Cont05]. Daher wird diese Vorgabe auch für alle Spannungen übernommen.

Die Gesamtleistungsaufnahme der Plattform ergibt sich daraus nach der folgenden Formel zu

$$P_{ges} = \sum_{i=1}^{5} \left[U(i) \times I(i) \right]$$
$$\forall \; U \; \in \left\{ 1{,}5V ; 2{,}5V ; 3{,}3V ; 3{,}3V_{M9} ; 5{,}0V \right\}$$
$$\forall \; I \; \in \left\{ I_{1,5V} ; I_{2,5V} ; I_{3,3V} ; I_{3,3V-M9} ; I_{5,0V} \right\}$$

P_{ges} = [1, 5V · 10A + 2, 5V · 2A + 3, 3V · (10A + 1,1A) + 5V · (3A + 0, 4A)] = 73,6W

Formel 4.6: Abschätzung der maximalen Gesamtleistungsaufnahme

Die in Tabelle 4.14 aufgeführte Spannung 1V2 wurde vorgesehen, um die COMPASS-Plattform bzw. die Interface-Steckkarten auch mit FPGAs ausrüsten zu können, die eine Core-Spannung von 1,2V benötigen. Diese Option wurde allerdings in der aktuellen Version der Stromversorgung nicht realisiert. Um ein kontrolliertes Deaktivieren nicht benötigter Spannungen umsetzen zu können, sind alle in Tabelle 4.15 aufgeführten Spannungen über A/D-Wandlereingänge überwachbar und lassen sich über Logikpegel eines Mikrocontrollers an- und abschalten.

4.2.2 Anforderungen zur Versorgung der FPGAs

Um FPGAs der Firma Xilinx zu versorgen, sind mindestens zwei unterschiedliche Spannungen erforderlich - eine Core-Spannung V_{CCINT} und eine I/O-Spannung V_{CCO}. Die Core-Spannung versorgt die interne Logik, wie etwa die konfigurierbaren Logikblöcke (Configurable Logic Blocks, CLBs) und die programmierbaren Schaltmatrizen. Typische Werte für die Core-Spannung liegen zwischen 1,0V und 2,5V. Die Ausgangstreiber der I/O-Bänke beziehen ihre Spannung – meist zwischen 1,5V und 3,3V – über die zugehörigen V_{CCO}-Pins. Da die Spannung V_{CCO} die Höhe der Amplituden der Ausgangssignale bestimmt (sofern nicht der GTP oder GTLP-Standard verwendet wird), ist ein Wert von 3,3V für V_{CCO} üblich, um den FPGA mit der übrigen externen Logik zu koppeln. Viele FPGAs benötigen für die Konfigurationspins und die DCMs (Digital Clock Managers) noch eine Hilfsspannung V_{CCAUX}. An diese Spannung, die zwischen 2,5V und 3,3V liegen kann, werden besondere Anforderungen bezüglich einer geringen Restwelligkeit und geringem Rauschen gestellt.

Die Stromaufnahme von FPGAs ist nicht konstant und hängt von vielen anwendungsbezogenen Faktoren ab, beispielsweise von der Taktfrequenz und der Anzahl der verwendeten Logikblöcke. Während des Betriebs sind je nach FPGA-Typ Werte zwischen 0,1A und 10A bis hin zu 20A möglich [Nati05]. Während des Einschaltens eines FPGAs muss die Spannungsquelle mindestens einen im Datenblatt vorgegebenen Strom für V_{CCINT} liefern können. Dieser Strom kann bei einem schnellen Spannungsanstieg sehr hohe, nicht spezifizierte Werte annehmen. Die Spannung muss in jedem Fall monoton ansteigen und bei Xilinx-FPGAs innerhalb von 50ms ihren vorgegebenen Wert erreichen. Teilweise wird auch eine untere Grenze für die Anstiegszeit angegeben. Steigt die Spannung relativ langsam an, ist es außerdem erforderlich, die Core-Spannung V_{CCINT} vor der I/O-Spannung V_{CCO} anzulegen.

4.2.3 Anforderungen an das Einschalten der FPGA-Spannungen

Während des Einschaltens eines Spartan-II-FPGAs muss die Spannungsquelle für die Core-Spannung V_{CCINT} einen bestimmten Mindeststrom I_{CCPO} liefern können. Dieser ist unabhängig von den Betriebsströmen und ist in [Xil01a] näher beschrieben. Im Temperaturbereich zwischen 0°C und 85°C beträgt I_{CCPO} bei neueren FPGAs ab einem Datumscode von

„0321“ 0,25A, und 0,5A bei älteren Typen [Xil01b]. Die Core-Spannung muss monoton ansteigen und nach spätestens 50ms ihren Endwert von 2,5V erreichen (Abbildung 4.19).

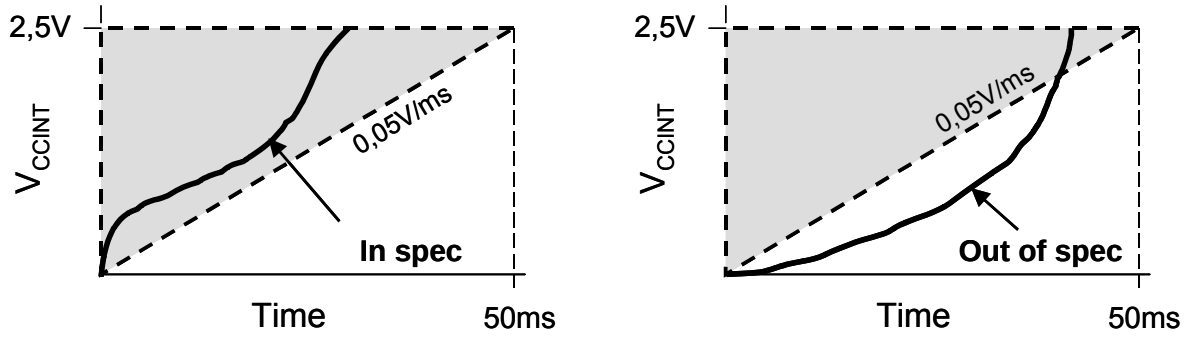

Abbildung 4.19: Spezifikation der maximalen V_{CCINT} -Anstiegszeit eines Spartan-II

Ein unterer Grenzwert für die Anstiegszeit T_{CCPO} existiert bei Spartan-II-FPGAs nicht. Aufgrund der internen Kapazitäten und der externen Stützkondensatoren nimmt der fließende Strom bei kleinem T_{CCPO} aber sehr hohe Werte im Ampere-Bereich an. Dies kann bei linearen Spannungsreglern als auch bei Schaltreglern zu Problemen führen, da unter Umständen die interne Strombegrenzung anspricht, was zu Spannungseinbrüchen und damit zur Verletzung der in Abbildung 4.19 dargestellten Spezifikation führen kann. Die Spannungen V_{CCO} und V_{CCINT} des Spartan-II können laut Datenblatt [Xil04] in beliebiger Reihenfolge angeschaltet werden. Zur Stromaufnahme der I/O-Spannung V_{CCO} während des Einschaltvorgangs werden weder im Datenblatt noch in der Application Note [Xil01a] Angaben gemacht. In anderen Quellen [Texa04] wird aber beschrieben, dass es sinnvoll ist, die Core-Spannung vor der I/O-Spannung anzulegen, da dies den Gesamtstrombedarf beim Einschalten reduzieren kann. Beim FPGA Virtex-II [Xil05e] werden für alle drei Betriebsspannungen Mindestströme vorgegeben. Bei den auf den Interface-Karten eingesetzten Typen XC2V1000 betragen diese 250mA für V_{CCINT}, 100mA für V_{CCAUX} und 50mA für V_{CCO} (siehe Tabelle 4.16).

	FPGA XC2Vxxxx							
Ströme [mA]	**40-500**	**1000**	**1500**	**2000**	**3000**	**4000**	**6000**	**8000**
$I_{CCINTMIN}$	200	250	350	400	500	650	800	1100
$I_{CCAUXMIN}$	100	100	100	100	100	100	100	100
I_{CCOMIN}	50	50	100	100	100	100	100	100

Tabelle 4.16: Erforderliche Einschaltströme für Virtex-II-FPGAs

Jede der drei Spannungen muss innerhalb von 200µs bis 50ms die jeweilige untere Betriebsspannungsgrenze erreicht haben. In dem Fall, dass die Anstiegszeit von V_{CCINT} mehr als 10ms beträgt, müssen V_{CCO} und V_{CCAUX} vor V_{CCINT} angelegt werden, andernfalls wird der FPGA u.U. nicht korrekt initialisiert. Weiterhin gibt es in [Xil05e] noch eine Empfehlung, V_{CCO} nicht vor V_{CCAUX} anzuschalten, um eine erhöhte Stromaufnahme zu vermeiden. Dies wird aber bereits dadurch erreicht, dass beide Spannungen auf den Interface-Karten miteinander verbunden sind. Da V_{CCAUX} kritische Schaltungsteile innerhalb des FPGAs versorgt, muss diese Spannung bzw. die Spannung 3V3 besonders rauscharm ausgelegt werden. Spannungsänderungen größer 200mV dürfen nur mit Raten kleiner 10mV/ms auftreten.

4.2.4 Blockschaltbild der PMU

Aus den bisherigen Überlegungen und Anforderungen der FPGAs ist das in Abbildung 4.20 dargestellte Blockschaltbild der PMU abgeleitet. Es gliedert sich in vier Bereiche – eine Eingangstufe zur Energieeinspeisung, einem Low- und einem High-Level-Bereich zur Energieweitergabe an die Plattform und einen Steuerungsteil zur Überwachung und Kontrolle der PMU.

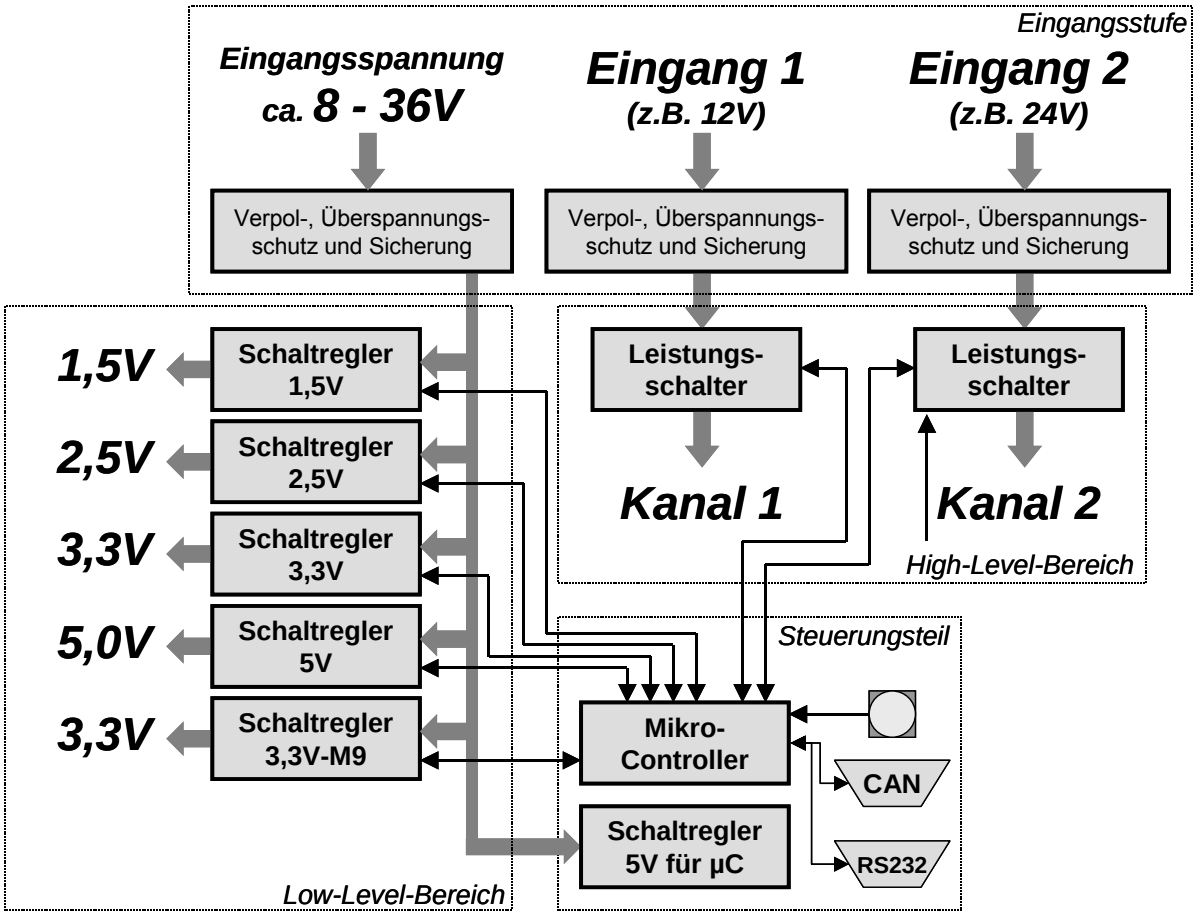

Abbildung 4.20: Spannungsversorgungseinheit für die RP-HiL-Plattform

Während die Eingänge 1 und 2 (rechts in Abbildung 4.20) der direkten Einspeisung der Referenzspannungen für den Prozess dienen, wird die Eingangsspannung links oben zur Versorgung der Regler verwendet, die daraus die verschiedenen Spannungsbereiche für die Plattform und für den Controller auf der PMU erzeugen. Alle Eingänge sind getrennt gegen Verpolung und Überspannung geschützt. Der µController ist mit allen Reglern, Leistungsschaltern und den Ausgangsleitungen der jeweiligen Spannungen verbunden, sodass er die Ausgänge der Spannungsbereiche an- und abschalten, sowie die Höhe der einzelnen Spannungen messen kann.

4.2.5 Spannungs-Monitoring, An- und Abschaltezyklen

Nach dem Zuschalten der Spannung an der PMU sind zunächst alle Regler aus- und die Leistungsschalter abgeschaltet. Nach der Initialisierung des Mikrocontrollers werden danach im Hauptprogramm zunächst die Spannung von 3,3V für das Micro9-Mikrocontrollersystem angeschaltet, damit die Plattform hochfahren und die Hardware testen kann. Anschließend

wartet das Programm auf weitere Befehle und überprüft dabei in einer Schleife ununterbrochen die 3,3V auf Über- und Unterspannung (Abbildung 4.21).

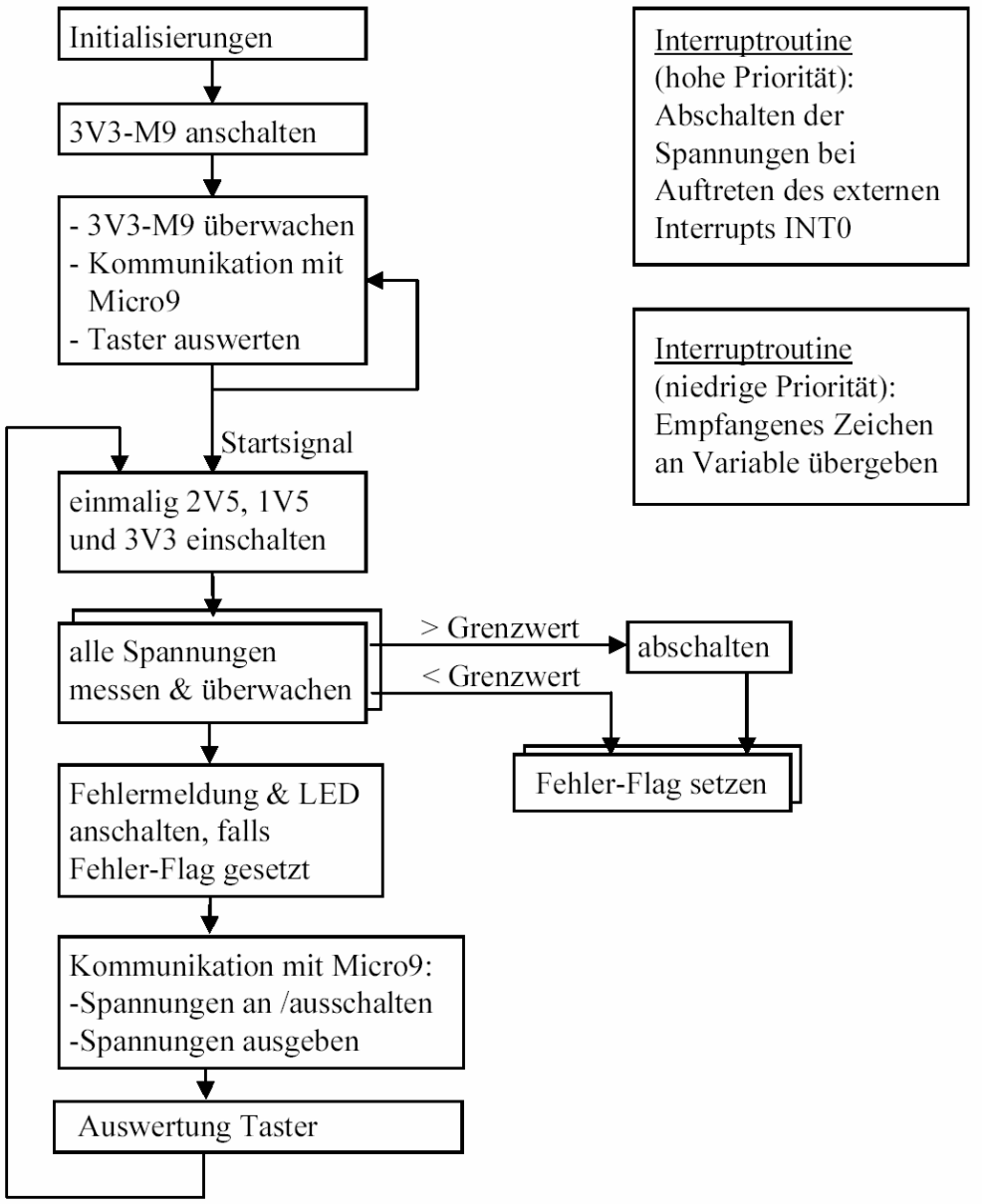

Abbildung 4.21: Vereinfachtes Flussdiagramm

Wird der Taster auf der Platine betätigt, ein ’s’ als Startzeichen über die serielle Schnittstelle oder über den CAN-Bus empfangen, schaltet der µController die für den Betrieb der FPGAs auf der COMPASS-Plattform notwendigen Spannungen von 1,5V, 2,5V und 3,3V in der richtigen Reihenfolge (2V5→1V5→3V3) zu. Die Spannungen werden dabei schon während des Einschaltvorgangs auf Überspannung kontrolliert. Anschließend folgt eine Endlosschleife, in der zunächst alle Spannungen nacheinander gemessen und kontrolliert werden.

Über die beiden Interrupt-Routinen werden externe Ereignisse abgefangen und interpretiert, wie z.B. das Auslösen des Watchdogs des Plattform-Hauptprozessors und der Eingang von Steuerungsbefehlen über CAN oder RS232. Im ersten Fall ist ein Absturz der Software auf der Plattform zu vermuten, so dass alle Spannungen zunächst abgeschaltet werden. Nach einer kurzen Zeit beginnt das Zuschalten entsprechend eines Neustarts. Bei eingehen-

den Nachrichten werden die Befehle (z.B. das Zu- oder Abschalten von Kanälen) angenommen, ausgewertet und umgesetzt. Dies ermöglicht das Auslesen des Zustands der PMU, sowie eine Steuerung von außen.

4.3 Ergebnis des Konzepts

Aus der vorausgegangenen Konzeptphase ergibt sich die Basisplatine, wie sie in Abbildung 4.22 zu sehen ist. Dort sind die auf der Plattform vorhandenen Komponenten und das vielschichtige Buskonzept, das diese verbindet, grafisch dargestellt. Hauptsteuerungselement ist der ARM-Prozessor, der über den Systembus mit den Interface-Karten und dem Board-Control-FPGA verbunden ist. Die Interface-Karten sind mit mindestens einem FPGA bestückt, auf dem in Hardware abgebildete Funktionen implementiert werden und tragen somit hauptsächlich zur Flexibilität des Gesamtkonzepts bei. Untereinander können diese Karten über einen getrennten Bus vernetzt werden, den Module-Interconnect-Bus. Zur funktionalen Entlastung des Hauptprozessors übernimmt der Board-Control-FPGA die Programmierung der FPGAs auf den Interface-Karten und das Auslesen der Kennungen auf allen Steckkarten, sowie die Ansteuerung des Taktgenerators für einen plattformweiten Referenztakt.

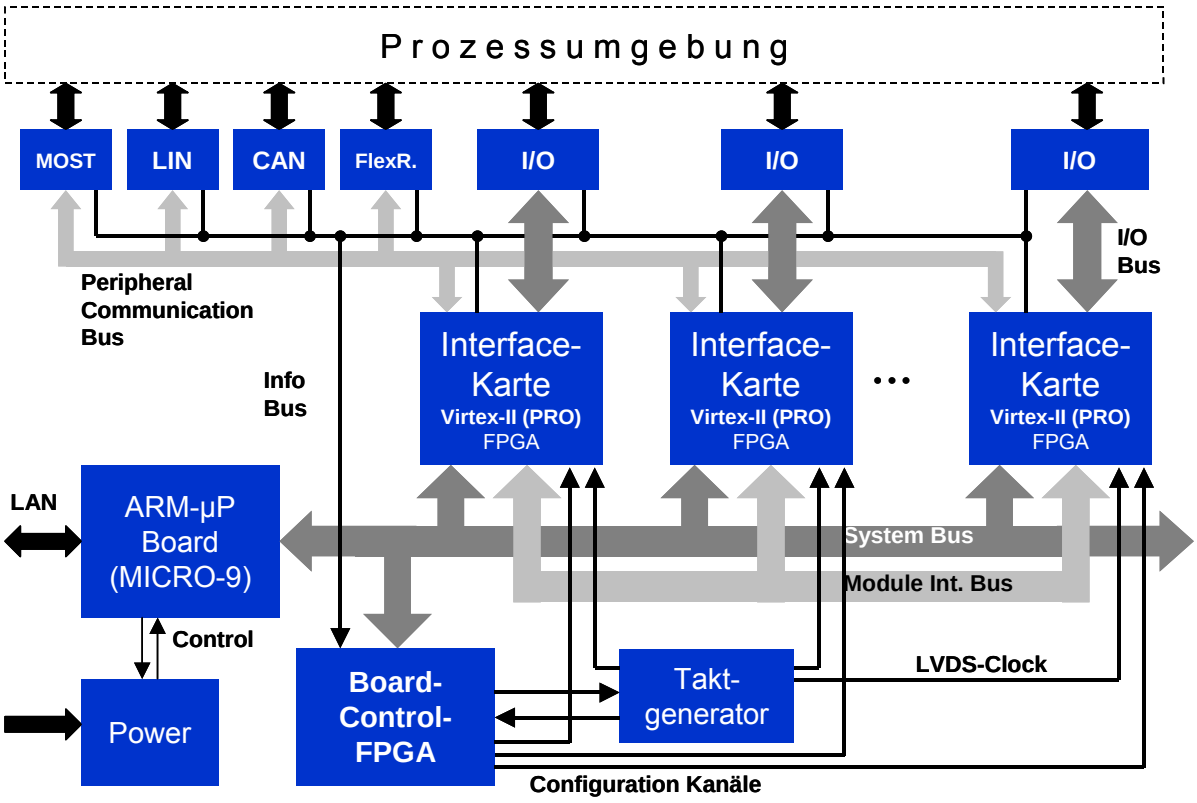

Abbildung 4.22: Schematische Übersicht über die Plattform

Im oberen Teil des Bildes finden sich die Komponenten zur Ankopplung der Plattform an den Prozess. Dies sind zum einen die I/O-Karten, mit denen digitale und analoge Ein-/Ausgabe realisiert wird. Sie sind über dedizierte Verbindungen an die ihnen zugeordneten Interface-Karten angeschlossen. Zum anderen kann über weitere Steckplätze die Kommunikation nach außen zum Prozess über (Automotive) Bussysteme etabliert werden. Diese sind über den Peripheral Communication Bus mit allen Interface-Karten verbunden, so dass eine freie Zuordnung möglich ist.

Über die LAN- bzw. Ethernet-Schnittstelle wird die Verbindung zu einem Host-PC hergestellt. Hierüber werden sowohl die Steuerung der Plattform, als auch die zur Funktion nötigen Downloads abgewickelt.

Zur Energieversorgung dient eine speziell auf diese Plattform zugeschnittene Power-Managment-Unit, die sowohl die niedrigeren Spannungen zur Versorgung der Plattform, als auch die höheren für die hier beschriebenen Automotive-Anwendungen bereitstellt. Eine getrennte und detaillierte Übersicht zu dieser Einheit ist in Kapitel 4.2.4 gegeben.

4.4 Realisierung der Plattform

4.4.1 Überblick über die Basisplatine und die Stromversorgung

Abbildung 4.23 gibt einen Überblick über die Basisplatine des RP-HiL-Systems (links im Bild) und die dazugehörige Stromversorgung (rechts).

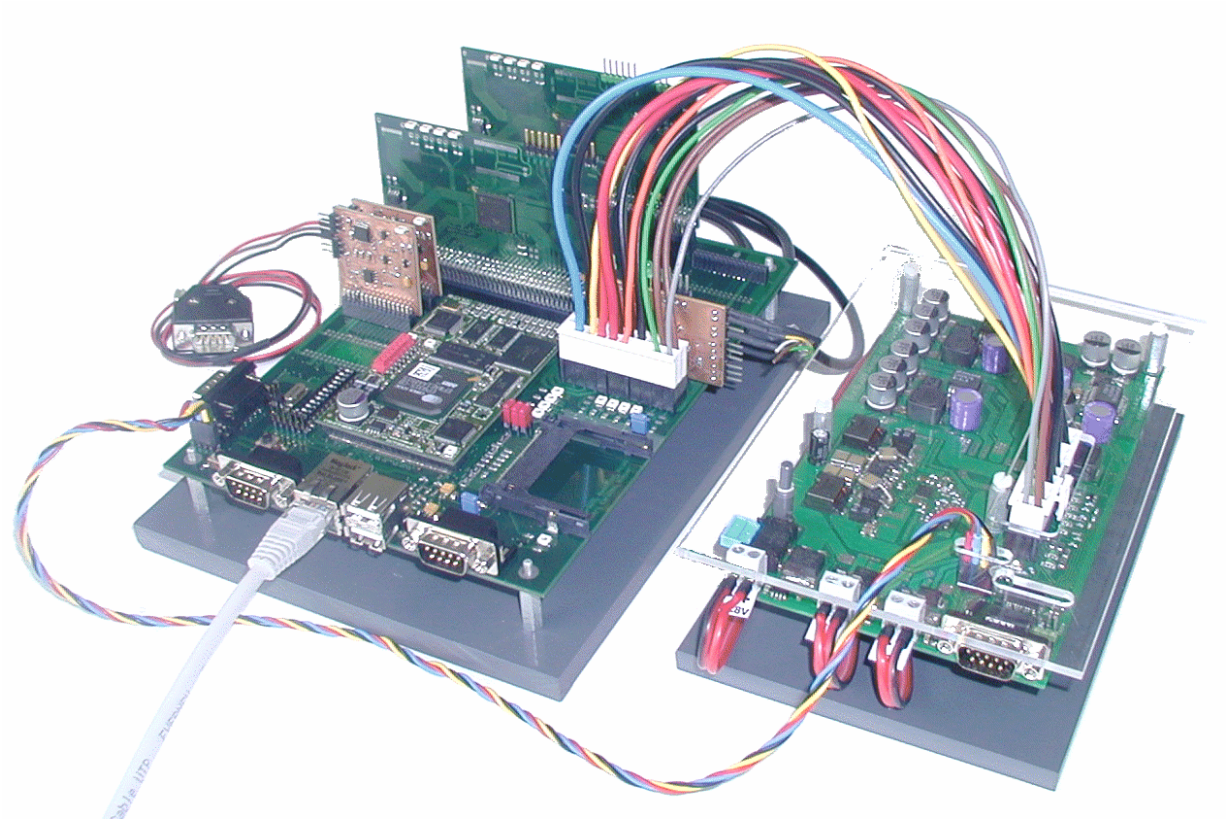

Abbildung 4.23: Basisplatine des RP-HiL-Systems

4.4.2 Hauptprozessoreinheit

Die Hauptprozessoreinheit der COMPASS-Platform bildet das kommerzielles Mikrocontroller-Board „Micro-9“ der Firma Contec mbH in Österreich (Abbildung 4.25). Es stellt den Prozessor, verschiedene Speicher sowie die Schnittstellen für die Peripherie bereit. Der auf dem Micro-9 System eingesetzte [EP9315] Prozessor von Cirrus Logic [Cirr04] beinhaltet einen schnellen ARM9-Core sowie einen MaverickCrunch Coprozessor zum Ausführen von arithmetischen Operationen. Der ARM9-Kern basiert auf einer Harvard-Architektur mit zwei getrennten 16 kByte großen Befehls- und Daten-Caches. Zur parallelen Ausführung verwendet er eine 5-stufige Pipeline, die aus den Operationen fetch, decode, execute, data memory access und write besteht. Zur Beschleunigung von Floating-Point- und 32 Bit sowie 64 Bit Fi-

xed-Point-Operationen dient der integrierte Co-Prozessor. Für die schnelle Verarbeitung von Audio- und Video-Dateien ist zusätzlich eine Multiply/Accumulate-Unit (nicht im Bild) vorhanden.

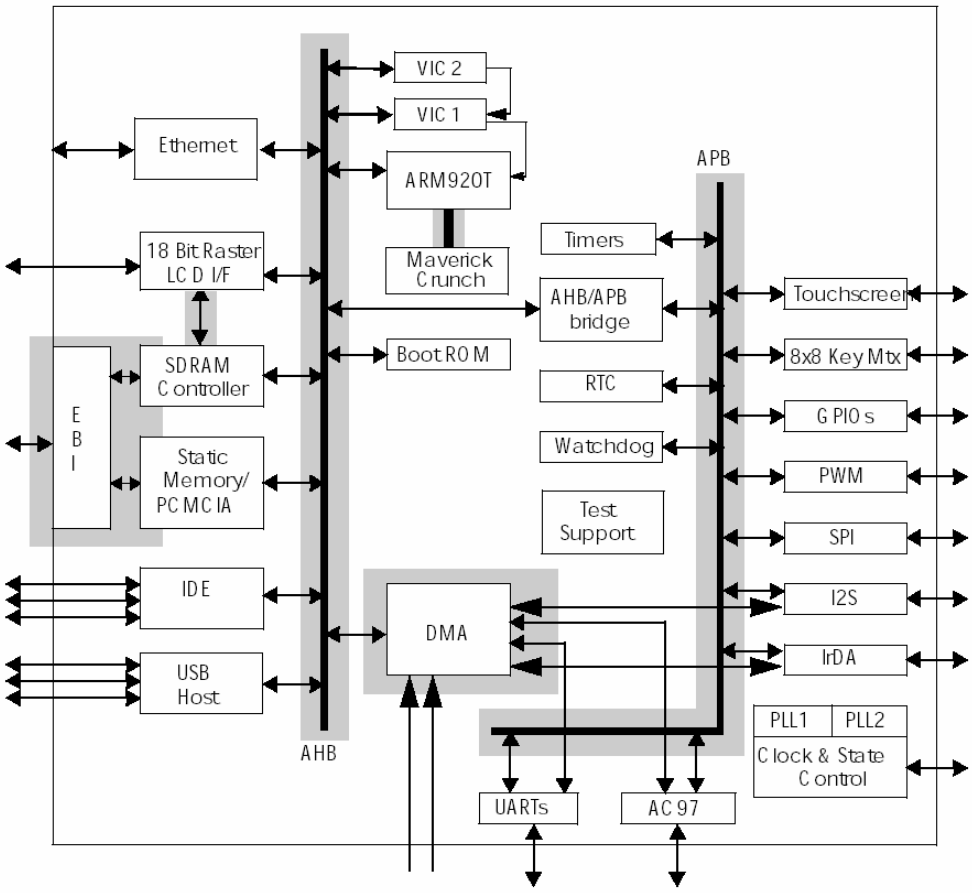

Abbildung 4.24: Übersichtsblockschaltbild des EP9315 von CirrusLogic [Cirr04]

Die interne Taktfrequenz des Prozessors beträgt 200 MHz, die des Businterfaces liegt bei 100 MHz. Der direkt angebundene 100 MHz AMBA High-Speed Bus (AHB) verbindet den Prozessorkern mit allen schnellen Peripheriebausteinen. Zum Anschluss der langsameren Komponenten, wie beispielsweise UART und SPI, ist der AMBA Peripheral Bus (APB) vorhanden, der über eine Bridge mit dem schnelleren AHB verbunden wird (siehe Abbildung 4.24). Die eingebaute Memory Management Unit (MMU) ermöglicht den Einsatz von Linux, Windows CE und anderen Embedded-Betriebssystemen. Sie ermöglicht die Adressübersetzung auf Basis von Translation-Tables sowie die Vergabe von Zugriffsrechten auf bestimmte Speicherbereiche. Ein im Prozessor eingebauter Ethernet LAN Controller beinhaltet sämtliche Logik, die zur direkten Anbindung an den AHB und an ein Media Independent Interface (MII) notwendig ist. Das MII wird direkt mit dem auf dem Micro-9-Board vorhandenen Fast Ethernet Transceiver von STMicroelectronics gekoppelt. Der eingesetzte Baustein STE100P kann eine 100BASE-TX oder 10BASE-T Twisted-Pair Verbindung treiben. Eine Auswahl der Verbindungsgeschwindigkeit, des Full- oder Half-Duplex-Betriebs sowie die Erkennung von Crossover-Verkabelungen erfolgt automatisch durch den Baustein.

Zur Verbindung zwischen dem internen AHB und externen Speicherbausteinen dient der als AHB-Slave eingesetzte Static-Memory-Controller. Über acht verschiedene Speicherbänke können verschiedene Speichertypen angeschlossen werden. Dazu zählen beispielsweise SRAM, ROM oder FLASH EPROM. Zu den Hauptaufgaben des SMC zählen unter anderem die Kontrolle der Zugriffssequenzen, die Generierung von Wait-States und der Anschluss von externen Businterfaces. Für die Einstellung der Datenbreite von 8, 16 oder 32 Bit und die

Anzahl der Wait-States kann für jeden der acht Speicherbereiche eine getrennte Auswahl getroffen werden. Die auf dem Micro-9 eingesetzten SDRAM und Flash-Speicher, sowie der Anschluss des System-Bus und der Compact-Flash-Karte werden über den SMC verbunden. In [Cont05c] sind alle wichtigen Daten des Boards zusammengestellt.

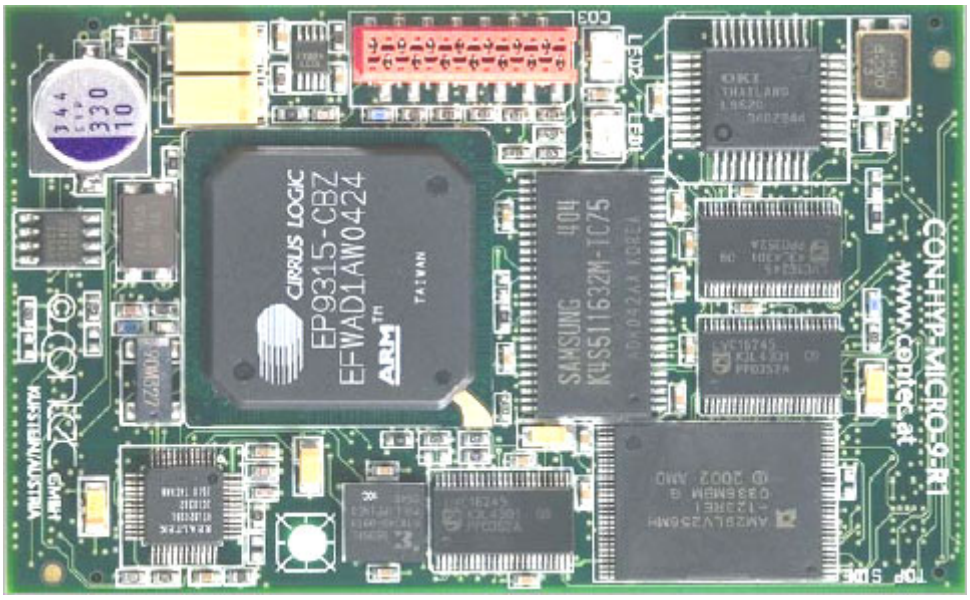

Abbildung 4.25: Oberseite des Micro-9 Board von Contec mbH, Österreich

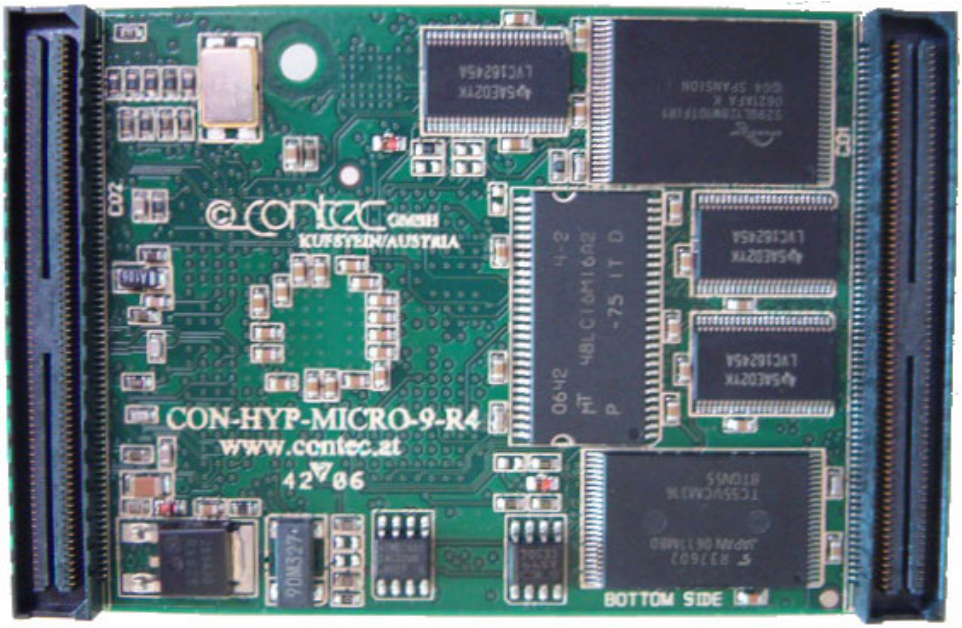

Abbildung 4.26: Unterseite des Micro-9 Board

Über die beiden 150-poligen Steckverbinder auf der Unterseite des Micro-9-Boards werden alle wichtigen Verbindungen zwischen dem Mikrocontroller-Board und den peripheren Geräten hergestellt. Dazu zählt der schnelle AHB, der über einen Buffer direkt an den Prozessor angebunden ist. Er umfasst einen 32 Bit breiten Adress-, einen 32 Bit Datenbus und weitere Leitungen zur Steuerung der Übertragung. Davon sind aber nur 26 Bit des Adressbusses über den Steckverbinder erreichbar. Um ohne das Vorhandensein der oberen Ad-

ressleitungen trotzdem eine Auswahl des angesprochenen Busteilnehmers zu ermöglichen, werden mehrere Chip-Select Leitungen vom Micro-9-System zur Verfügung gestellt. Zur Steuerung der Plattform über die CompassControl-Software wird die vorhandene Netzwerkschnittstelle genutzt. Auf dem Micro-9 ist bereits ein Ethernet-PHY von STMicroelectronics vorhanden, der einen direkten Anschluss eines Transformers zur galvanischen Trennung und Ankopplung an ein Netzwerk erlaubt. Zur Anbindung größerer Speichermedien ist eine genormte IDE-Schnittstelle verfügbar, die auf dem Basisboard als Compact-Flash-Steckplatz realisiert wurde. Für die serielle Kommunikation, sind insgesamt drei UARTs und drei USB-Anschlüsse auf dem Micro-9 verfügbar, von denen die UARTs als RS232-Schnittstellen ausgeführt sind und die USB-Schnittstellen auf passende Steckbuchsen gelegt wurden. Zur Ansteuerung der externen Power-Management-Unit wurde eine Verbindung über den auf dem Micro-9 vorhandenen CAN-Controller hergestellt. Des Weiteren sind 16 I/O-Leitungen vorhanden, die direkt vom Prozessor angesteuert werden können. Um größtmögliche Flexibilität bei der Nutzung zu erreichen, wurden diese an den Board-Control-FPGA angeschlossen. Sie könnten beispielsweise zur direkten Programmierung der Interface-Karten eingesetzt werden. Ein auf dem Micro-9 vorhandenes NV-RAM kann zur Speichererhaltung über einen speziellen Anschluss mit Energie versorgt werden, wenn die Betriebsspannung ausgeschaltet wurde. Der Anschluss einer passenden Pufferbatterie erfolgt auf dem Basisboard. Ein Reset des Mirco-9 Boards lässt sich über einen angeschlossenen Taster ausführen.

4.4.2.1 ARM-Architektur

Die ARM-Prozessorarchitektur wurde speziell auf den Einsatz in mobilen und Embedded-Systemen zugeschnitten. Dazu zählt der geringe Stromverbrauch und somit auch geringe Verlustleistung, eine gute Skalierbarkeit zur Anpassung an die jeweiligen Anforderungen sowie integrierte I/O-Komponenten. Die Architektur wurde von der englischen Firma Acorn Computers Ltd. 1983 entworfen, von der sich auch der Name ARM (Acorn RISC Machine) ableitet. Mittlerweile gibt es sehr viele Firmen, die eine Lizenz zum Einsatz des Cores erworben haben, darunter auch Intel, IBM und TI. ARM-Prozessoren verfügen über ein RISC (Reduced Instruction Set Computer) Design mit 32 Bit Daten- und Adressbus. Der Speicherzugriff erfolgt über eine Load/Store-Architektur, die den Speicherzugriff nur über spezielle Befehle erlaubt. Alle Instruktionen sind 32 Bit lang und ermöglichen eine bedingte Ausführung. Durch die bedingte Ausführung ist keine Branch-Prediction notwendig und der kompaktere Code ermöglicht eine bessere Performance. Neben dem ARM-Standard-Modus unterstützen verschiedene Modelle auch den Thumb-Modus. Dieser bietet 16 Bit Instruktionen und ermöglicht somit eine höhere Code-Dichte. Nicht alle Operationen können den vollen Registerraum ansprechen. Im Falle von Datenzugriffen mit weniger als 32 Bit kann sich die Ausführung lohnen, da diese Operationen schneller ausgeführt werden. Thumb-Modus Modelle werden durch ein ”T“ im Namen gekennzeichnet. Ein weiterer Ausführungsmodus ist der Jazelle-Modus und wird durch ein ” J“ in der Bezeichnung des Prozessors gekennzeichnet. Er ermöglicht die direkte Ausführung von Java-Bytecode in Hardware. Zum schnellen Ausführen von speziellen Instruktionen können an einen ARM-Core bis zu 16 Co-Prozessoren angebunden werden. Die Flusssteuerung wird vom ARM übernommen, der die Daten an die Erweiterung weitergibt. Die interne Verarbeitungsbreite des Coprozessors ist dabei beliebig.

4.4.3 CAN-Bus

Der als Peripheriegerät direkt auf dem Micro9-Modul an den Prozessorbus angekoppelte CAN-Controller ML9620 von Oki erfüllt alle Anforderungen, die für eine High-Speed CAN-Übertragung notwendig sind. Er hält alle Protokollspezifikationen von Bosch ein, die für den Einsatz im Automobil gefordert werden und unterstützt Datenraten bis zu 1 Mbps (≡MBaud). Für den physikalischen Busanschluss wurde auf dem Basisboard ein zum Controller passender CAN-Transceiver eingesetzt. Bei dem verwendeten Baustein handelt es sich um den Standard 3.3V Typ [SN65HVD230] von Texas Instruments. Er dient als Anpassung des Controllers an den ”Physical Layer“ und übernimmt die differentielle Übertragung auf dem Bus. Als Steckverbinder wurde ein nach DIN 41652 belegter, 9-poliger SUB-D Stecker ver-

baut (Belegung: siehe Tabelle 2.2). Auf der COMPASS-Plattform wird der CAN-Bus zur Kommunikation mit einer externen Power-Management-Unit eingesetzt. Die verwendete Steckerbelegung ist gemäß der CIA-Empfehlung ausgeführt (siehe Kapitel 2.1.2).

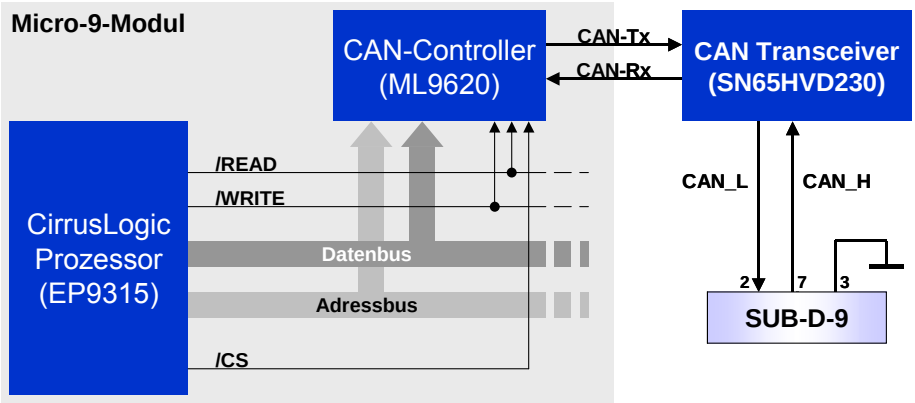

Abbildung 4.27: Realisierung des On-Board CAN-Busanschlusses

4.4.4 Ethernet-Anschluss

Die Netzwerkschnittstelle des Micro-9-Systems verfügt bereits über den Fast-Ethernet-Transceiver STE100P von STMicroelectronics. Er ist über ein Media-Independent-Interface (MII) an den Prozessor angebunden und besitzt einen Treiber zum direkten Anschluss einer Twisted-Pair-Verbindung. Für den Anschluss eines RJ-45 Steckers wurde eine Buchse von Bel Stewart Connector verbaut. Diese enthält bereits einen eingebauten Transformer, der Platz sparend im Gehäuse integriert wurde. Ein externer Transformator ist somit nicht mehr notwendig. Zur Anzeige des Netzwerkstatus befinden sich zwei LEDs in dem Gehäuse der RJ-45 Buchse. Sie dienen der Anzeige des Link- und des Kollisions-Signals der Verbindung. Die Buchse ist mit einem Übertragungsverhältnis von 1:1 optimal auf den Transceiver abgestimmt. Zur äußeren Beschaltung der Buchse werden nur Pull-Up-Widerstände und Stützkondensatoren benötigt.

4.4.5 Serielle Schnittstellen

Der eingesetzte Cirrus Logic-Prozessor EP9315 [Cirr04] verfügt über drei UARTs (Universal Asynchronous Receiver Transmitter), die zur seriellen Kommunikation eingesetzt werden können. Alle drei sind als RS232-Schnittstellen ausgeführt. Die erste serielle Verbindung bietet einen voll belegten SUB-D-9 Stecker, der von einem True-RS232-Baustein [MAX3243] getrieben wird. Er verfügt über alle Leitungen, die zur Herstellung einer Hardware-Handshake-Verbindung notwendig sind. Zum Anschluss externer Geräte wird die Sende-Spannung auf ±5.4V angehoben und die Eingangsspannung von maximal 15V auf einen Pegel von 3.3V abgesenkt. Die beiden anderen seriellen Schnittstellen werden von einem [MAX3232] bedient. Dieser besitzt zwei Leitungen für Senden und zwei für Empfangen. Um nach dem Starten des Boards auf die Schnittstellen zugreifen zu können, werden die beiden Transceiver-Bausteine mit der selben 3.3V Versorgungsspannung des Micro-9-Systems gespeist.

Zur Absicherung der Funktion des Micro-9-Boards und der gesamte COMPASS-Plattform, wird die Watchdog-Leitung des ARM-Prozessors auf den Steckverbinder der seriellen Schnittstelle 2 geführt. Diese Leitung kann durch eine externe Power-Management-Unit (PMU) abgefragt und so der Status des Boards überprüft werden. Sollte der Watchdog aufgrund eines Programmabsturzes auslösen (Watchdog-Zähler läuft über, da er vom Betriebssystem nicht gelöscht wird), kann die PMU dann beispielsweise zunächst alle Plattformspannungen ausschalten und danach einen definierten Start des COMPASS-Systems initiieren.

4.4.6 USB-Schnittstelle

Der USB-Anschluss erforderte neben dem Einbau einer geeigneten Steckbuchse noch einige Kondensatoren und Widerstände um Störungen zu unterdrücken. Die Überwachung und Absicherung der 5V USB-Spannung erfolgt über einen USB-Power-Control-Switch der Firma Sipex [SP2526]. Er erkennt beispielsweise eine zu große Stromaufnahme des USB-Geräts oder eine zu niedrige Spannung am Eingang. Zum Schutz des Bausteins ist zusätzlich ein thermischer Überlastungsschutz integriert. Die Versorgung der internen Logik erfolgt mit 3.3V, der geschaltete USB-Spannungseingang muss getrennt mit 5V versorgt werden.

4.4.7 Interface-Karte

Eine realisierte Interface-Karte ist in Abbildung 4.28 zu sehen. In der Mitte befindet sich ein Virtex-II1000-FPGA, der über den Steckverbinder unten mit dem Adress- und Datenbus des Micro-9-Systems, dem Modul-Interconnect-Bus (MIB) und der Peripheral-Communication-Bus (PCB) verbunden ist. Direkt oberhalb ist ein SRAM zu sehen. Hierfür wurde ein 4MBit großes, statisches RAM der Firma Samsung ausgewählt. Das K6R4016V1D bietet eine Datenbreite von 16 Bit bei einem Adressbereich von 18 Bit. Der maximale Datendurchsatz beim Lesen oder Schreiben liegt bei ca. 100Mbyte pro Sekunde. Außerdem finden sich noch ein serielles EEPROM (links mittig), Quarzoszillatoren (rechts oben) sowie Debug-Stecker und LEDs zur Signalisierung am oberen Rand auf der Platine.

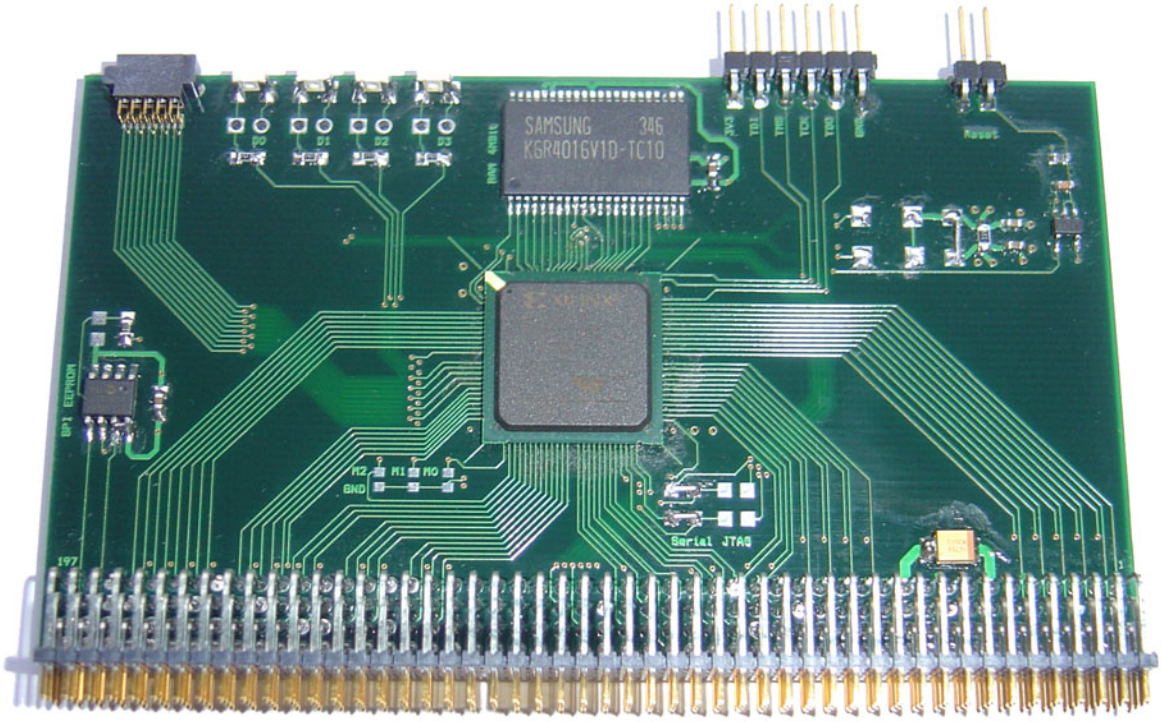

Abbildung 4.28: Interface-Karte (Oberseite)

4.4.8 I/O-Karte

Um die Funktionsfähigkeit der gesamten Plattform testen zu können, wurde exemplarisch eine I/O-Karte entworfen (Abbildung 4.29), die zur Ausgabe von digitalen und analogen Signalen genutzt werden kann. Für die Ausgabe der analogen Signale kommt der D/A-Wandler [MAX523] zum Einsatz. Er wird über eine SPI-Schnittstelle an die Ausgänge der Interface-Karte angebunden und verfügt über zwei getrennte Ausgangskanäle. Die Ansteuerung erfolgt mit 16 Bit Datenworten bei einem maximalen SPI-Bustakt von 13.5 MHz.

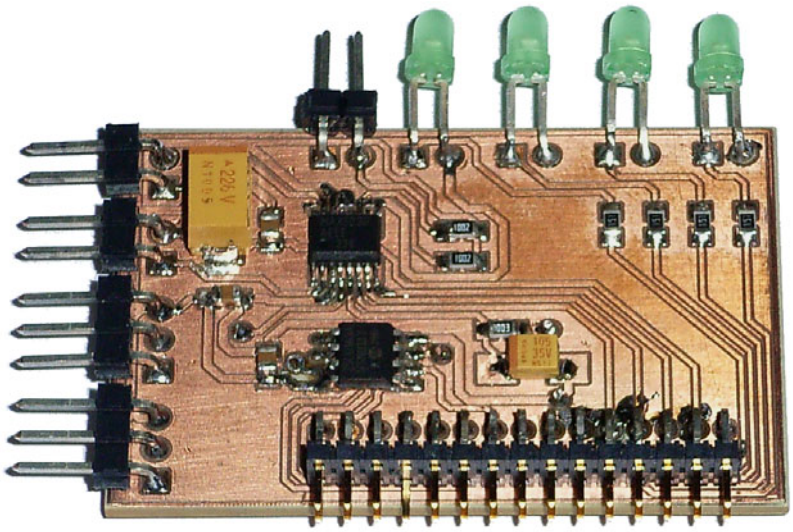

Abbildung 4.29: Foto der Beipiel-I/O-Karte

Die digitalen und analogen Signale werden über die Steckverbinder auf der linken Seite der abgegriffen. Zusätzlich existieren noch vier LEDs am oberen Rand zur Ausgabe von digitalen Informationen. Mit Hilfe des Steckverbinders am unteren Platinenrand kann das Modul einen I/O-Steckplatz eingesteckt werden. Links oberhalb dieses Steckers befindet sich ein serielles EEPROM, das die Modulinformationen enthält.

4.4.9 Automotive-Karte

Diese Beispiel-Automotive-Karte enthält zwei CAN-Transceiver vom Typ Texas Instruments [SN65HVD230]. Sie benötigen je zwei Signalleitungen zu den FPGA-Karten, RX und TX. Für die Anbindung an das CAN-Netzwerk ist pro Kanal eine 3-polige Steckleiste nach außen geführt, an der die Signale CAN-H und CAN-L sowie eine Masseverbindung anliegen. Zur Speicherung der Karten-Identifikation kommt auch hier wieder das serielle EEPROM [25LC160A] von Microchip zum Einsatz.

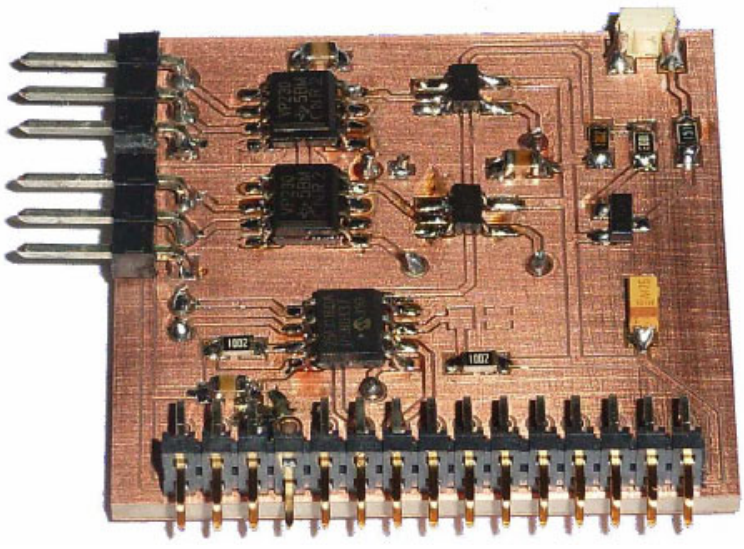

Abbildung 4.30: Automotive-Karte mit 2 CAN-Transceivern

Zwischen der Rx-Leitung des CAN-Transceivers und der zum FPGA führenden Busleitung ist je ein Tri-State-Bustreiber [SN74LVC1G125] zwischengeschaltet, der erst nach erfolgreicher Überprüfung auf Ressourcenkonflikte aktiviert wird. Um sicherzustellen, dass nicht mehrere Ausgangstreiber auf die gleiche Busleitung treiben, ist die Leitungsbelegung jeder Karte im entsprechenden EEPROM gespeichert.

Die fertig aufgebaute Automotive-Karte ist in Abbildung 4.30 dargestellt. Unten links ist das serielle EEPROM zu sehen; oben links die beiden CAN-Transceiver, deren Ausgänge für die Prozessankopplung nach links über Steckerleisten herausgeführt sind. Oben in der Mitte sieht man die Tri-State-Bustreiber und oben rechts eine LED, die den aktuellen Status der beiden Bustreiber anzeigt.

4.5 Realisierung der PMU

4.5.1 Überblick über die Stromversorgung (PMU)

Die folgende Abbildung 4.31 zeigt die aufgebaute Stromversorgungseinheit (Power Management Unit, PMU) zur Versorgung des Basissystems.

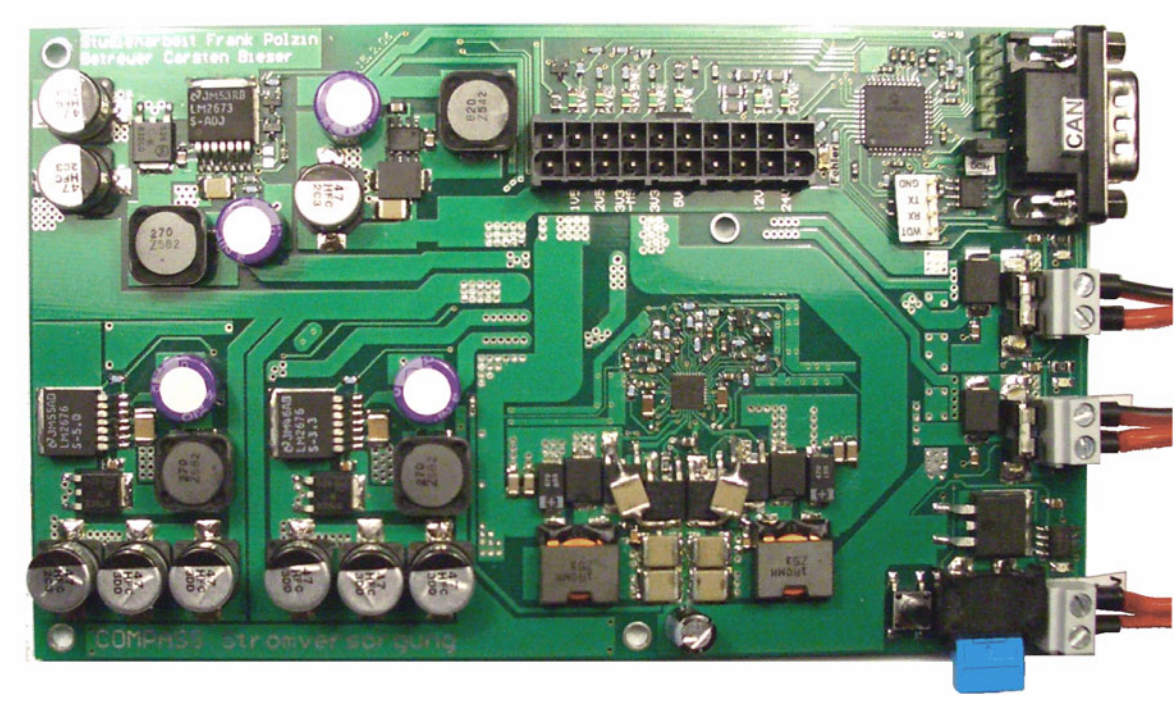

Abbildung 4.31: Stromversorgungseinheit (Power Management Unit, PMU)

4.5.2 Ausgangsleistung und Wirkungsgrad der PMU-Schaltregler

Im Hinblick auf die hohe Stromaufnahme der FPGAs wurden die Schaltregler so ausgewählt und ausgelegt, dass sie auch bei Maximalausbau der COMPASS-Plattform noch einen Sicherheitspuffer bei der Versorgung aller Komponenten bieten. Mit verschiedenen ohmschen Hochstromtestlasten wurden die einzelnen Spannungsdomänen nahe ihres Maximalstroms belastet. Die daraus resultierenden Ströme dienen deshalb nur als Anhaltspunkt und sind in Tabelle 4.17 zusammengefasst.

Spannungsdomäne	Strom (max.)
1V5	8,8A
2V5	2A
3V3-M9	2,75A
3V3	9,35A
5V	1,85A

Tabelle 4.17: Maximale Ausgangsströme der jeweiligen Spannungen

Um eine Aussage über die Effektivität der eingesetzten Schaltregler treffen zu können, wurden außerdem die Wirkungsgrade für die einzelnen Spannungsdomänen für die Transformierung der Eingangsspannung von 12V auf die jeweilige Ausgangsspannung ermittelt. Dabei wurden folgende Werte bestimmt:

Spannungen	1V5 & 3V3	2V5	3V3-M9	5V	Gesamt
P_{ein}	43W	5,7W	10W	10W	68,7W
P_{aus}	39W	4,4W	8,5W	8,9W	60,8W
Wirkungsgrad	91%	78%	85%	89%	88%

Tabelle 4.18: Wirkungsgrade der Stromversorgungsplatine

Unter Berücksichtigung der sehr geringen Energiereserven bei mobilen Messungen, z.B. im Auto, ist eine ressourcenschonende Spannungsversorgung unabdingbar. Die im Rahmen der Arbeit entwickelte Power Management Unit (PMU) zeigt dazu gute Wirkungsgrade bei der Bereitstellung verschiedener Spannungen für die Plattform. Ein weiterer wichtiger Punkt ist die Tatsache, dass die Eingangsspannung aus lediglich einer Quelle besteht und im Bereich von 9V – 36V schwanken kann.

5 Slot-basierte FPGA-Konfiguration

5.1 Einsatz von FPGAs in RP- oder HiL-Geräten

Die Verwendung von FPGAs im Allgemeinen oder wie hier vorgeschlagen in Rapid-Prototyping- und Hardware-in-the-Loop-Geräten bringt Vorteile hinsichtlich Flexibilität der Funktionsimplementierung und der dadurch erreichbaren Leistungsfähigkeit mit sich. Durch die variable Größe der Bausteine, d.h. die zur Verfügung stehenden vom Anwender programmierbaren Logikgatter auf dem Device, kann ein Gerät an die Erfordernisse und die Komplexität der zu bearbeitenden Aufgabe angepasst werden.

Um die FPGAs zu programmieren, d.h. die enthaltenen Logikgatter mit einer Funktion zu versehen, können verschiedene Ansätze gewählt werden. Diese unterscheiden sich durch ihre Abstraktionslevel und der damit verbundenen Art der Umsetzung innerhalb des Design Flows. Hier sind die wichtigsten Design-Alternativen nach steigendem Aufwand aufgelistet, wobei zwischen rein textuellen und grafischen Methoden unterschieden wird:

- Programmierung in VHDL oder Verilog (textuell)
- Zusammenstellen von Netzliste (textuell)
- CASE-Tool (grafisch)

Zwischen allen Methoden sind natürlich auch unzählige Mischformen möglich, auf die hier aber nicht weiter eingegangen werden soll. In solchen Fällen erhöht sich der Aufwand außerdem durch die Anzahl der unterschiedlichen Methoden.

5.1.1 Problematik bei inkrementeller Änderung

Allen Verfahren ist gemein, dass der Benutzer vor der Implementierung auf dem FPGA den kompletten Design Flow zur Erzeugung einer Konfigurationsdatei (Bitstream) zu durchlaufen hat (siehe Kapitel 3.8.4.1). Dies wird zwar grundsätzlich von den dafür vorgesehenen Software-Werkzeugen der jeweiligen FPGA-Hersteller erledigt, doch ist das Ergebnis nicht in allen Fällen zufrieden stellend. Vor allem aber ist der Design Flow dahingehend nicht deterministisch, dass selbst bei kleinen inkrementellen Änderungen im Design die Syntheseergebnisse ganz unterschiedlich ausfallen können. Abbildung 5.1 und Abbildung 5.2 zeigen die Post-Place-and-Route-Ergebnisse zweier Synthesen für einen Virtex-II-1000 FPGA. Dazu wurde ein Modell, bestehend aus drei Prozessen, in VHDL geschrieben und zweimal nacheinander mit der Xilinx ISE 6.3i synthetisiert, wobei für den zweiten Lauf die Originaldatei leicht verändert wurde. Die Änderung bestand darin, einen Prozess von dreien lediglich um-

zukopieren und dadurch die Reihenfolge zu modifizieren. Die Verteilung und Belegung der Ressourcen sowie das Routing ist deutlich unterschiedlich. Bei der Bestimmung der Maximalfrequenz des Designs durch das Tool ergab sich eine Änderung von ca. 50MHz vom ersten zum zweiten Lauf.

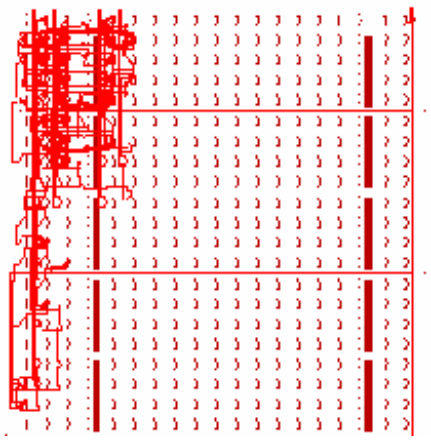

Abbildung 5.1: Syntheselauf 1 (Original)

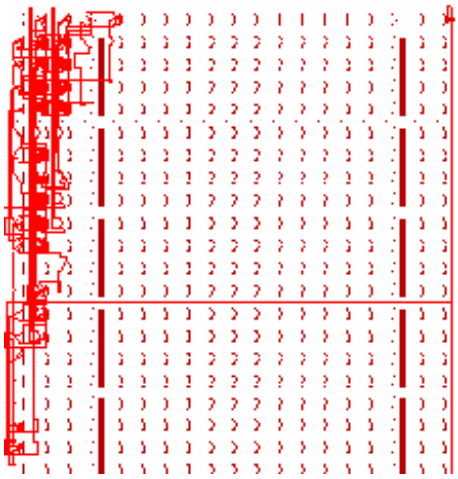

Abbildung 5.2: Syntheselauf 2 (mit inkrementeller Änderung)

5.1.2 Anforderungen

Diese Gegenüberstellung zeigt deutlich, wie stark die Syntheseergebnisse bei inkrementellen Änderungen an einem Design variieren können. Das bedeutet auch, dass sich nach jedem Syntheselauf die Implementierung in Timing und/oder Funktion ändern kann. Die Unterscheidung kann sich dabei durchaus von „voll funktionsfähig“ bis „nicht funktionsfähig“ erstrecken. In der Praxis führt dies zu einer hohen Design-Iterationsrate, die insbesondere für FPGA-unerfahrene Anwender keine geeignete Verfahrensweise darstellt, um die Vorteile, die durch die Verwendung von FPGA entstehen, für sich nutzbar zu machen. Dies kann zwar durch Setzen von Randbedingungen (engl. Constraints) hinsichtlich Timing und Flächenbelegung stark eingegrenzt werden, jedoch setzt dies eine umfassende Kenntnis der Bausteine und einige Erfahrung im Umgang mit den Werkzeugen voraus. Der Benutzer muss auf die korrekte Funktion einer Implementierung vertrauen können, ohne sich mit den Details der Erzeugung und der Implementierung auf dem Device auseinandersetzen zu müssen.

Um dem Anwender der Plattform dennoch die Ausnutzung der Flexibilität der FPGAs in Verbindung mit einer einfachen und reproduzierbaren Funktionsimplementierung zu ermöglichen, erfordert es einer Design-Methodik auf einem höheren Abstraktionslevel. Die dazu nötige Integration der vom Benutzer gewünschten Logik kann dahingegen weiterhin auf einer niedrigeren Ebene, wie z.B. dem Bitstream-Level, stattfinden. Jedoch sollte diese dem Benutzer dabei möglichst völlig verborgen bleiben. Der zur Instrumentierung der Schnittstellen nötige Kenntnisstand bezüglich der FPGA-Interna und Hardware-Programmiersprachen wie VHDL oder Verilog muss dabei ebenso niedrig gehalten werden, wie die Notwendigkeit zur Durchführung mehrerer Iterationsschritte zur Erreichung einer optimalen und funktionsfähigen Lösung.

Diese Vorgehensweise setzt außerdem voraus, dass die Funktionen, die der Anwender auf seiner flexiblen Plattform einsetzen können soll, bereits vorab verfügbar sind. Üblicher Weise werden dazu Bibliotheken verwendet, um verschiedene Funktionsmodule vorzuhalten und damit einfach zugänglich zu machen. Diese müssen zunächst erstellt und getestet werden, damit die oben genannten Bedingungen einer einfachen Handhabung und Implementie-

rung erfüllt werden können. Dieser Arbeitsschritt muss zwangsläufig von im Umgang mit FPGAs und den Design-Tools erfahrenem Personal ausgeführt werden. Nichts desto trotz besteht gleichermaßen die Möglichkeit, dass Anwender mit dem nötigen Background selbst Module erstellen, testen und in entsprechenden Bibliotheken ablegen und/oder anderen Benutzern zur Verfügung stellen. Dieses Vorgehen ist auch aus anderen Tool-Suites (z.B. MatLab/Simulink oder -/StateFlow) bekannt und sollte daher nicht ausgeschlossen sein.

5.2 Grundidee

Die Idee zur Konfiguration eines in einzelne Bereiche aufgeteilten FPGAs leitet sich von dem Prinzip eines Baukastens ab, bei dem eine Reihe von fertigen Bausteinen zur Verfügung steht, die in vorgegebene Lücken (Slots) eines Rahmens eingefügt werden können. Die Abstrahierung ist dabei durch die Bausteine selbst gegeben, da deren Inhalte, d.h. die Funktion, nur als Beschreibung bekannt ist, aber für den Anwender nicht sichtbar wird. Durch eine einheitliche Größe der Aussparungen innerhalb des Rahmens wird keine bestimmte Anordnung oder Kombination von Bausteinen vorgegeben, sondern eine wahlfreie Zuordnung ermöglicht.

Im Hinblick auf eine Automatisierung des Zusammenfügens – z.B. auf Knopfdruck und ohne weiteres Zutun des Benutzers – ist diese Vorgehensweise ebenfalls sinnvoll, um die Integration verschiedener Bausteine zu vereinheitlichen und Sonderbehandlungen möglichst auszuschließen. Daraus ergeben sich nun einige Anforderungen an die einzusetzenden Bausteine und den Rahmen, der diese aufnehmen soll. Im Hinblick auf elektronische Module, die verschiedenste Anschlüsse für die Energieversorgung und den Austausch von Daten besitzen, müssen auch in dem Rahmen (Framework), in dem sie eingebettet werden sollen, entsprechende Gegenstücke existieren. Letztere stellen somit die Grundversorgung der Module sicher und regeln weiterhin die Kommunikation nach außen über die Rahmengrenzen hinweg.

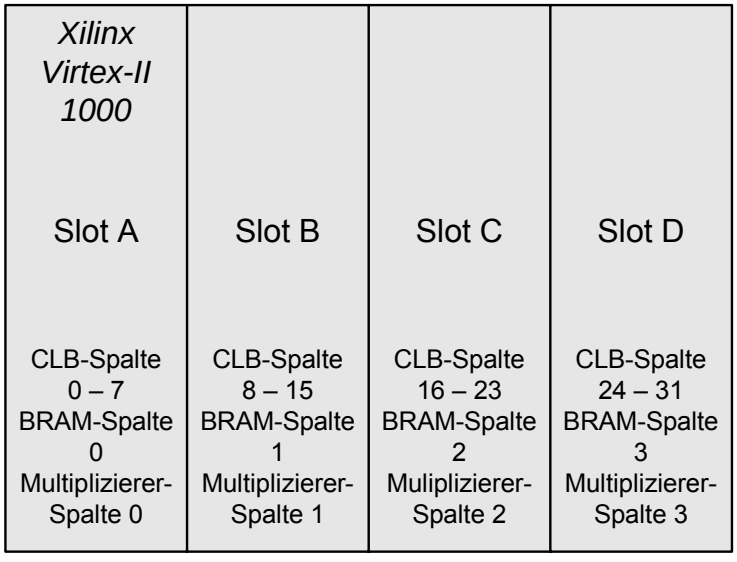

Abbildung 5.3: Aufteilung des Virtex-II 1000 in Bereiche (Slots)

Diese Verfahrensweise ist angelehnt an Arbeiten zur dynamischen Re-Konfiguration von FPGA-Teilen ([HüBB04], [KJTR05], [Bobd05]). Dabei werden eine bestimmte Anzahl von Spalten für die Re-Konfiguration freigehalten und der übrige Teil an FPGA-Fläche für statische Logik verwendet. Die in [KJTR05] verwendete Anordnung sieht statische Flächen zwischen den dynamischen vor, wodurch eine effiziente Aufteilung der FPGA-Ressourcen nicht

möglich ist. Allen Ansätzen ist gemein, dass die Kommunikationsstrukturen ausschließlich auf seriellen Verbindungen (Links) und entsprechenden Protokollen beruhen. Diese sind jedoch für eine hochperformante Datenübertragung eher ungeeignet. Darüber hinaus beschränkt sich die Kommunikation über die Bussysteme nur auf das Innere des FPGA.

Die bestehenden Ansätze wurden dahingehend erweitert, dass die internen Bussysteme nicht nur parallel ausgeführt sind, sondern auch bidirektional arbeiten können. Dadurch wird die Anzahl der benötigten Routing-Leitungen (z.B. beim Datenbus mit einer Breite von 16bit) erheblich verringert. Im Gegenzug muss jedoch dafür eine FPGA-interne Busarbitrierung vorgesehen werden, die den Zugriff auf die einzelnen Module bzw. das Belegen der Busleitungen durch die Module regelt. Zugriffskonflikte auf dem internen Bus durch zwei unterschiedliche Treiber können zu einer Zerstörung der internen Signalleitungen führen wodurch der FPGA unbrauchbar wird. Um die Kommunikation mit der Außenwelt zu etablieren, ist zusätzlich eine Kopplung der internen System- und I/O-Busse nötig. Über diese Verbindungen kann der Prozessor auf die Module innerhalb des FPGA zugreifen.

Eine serielle Ausführung der internen Busse wurde explizit nicht verfolgt, da dies zum einen die Parallel-Seriell-Wandlung der von außen kommenden, parallelen Daten erfordert hätte, die außerdem in jedem Modul hätten wieder zurückgewandelt werden müssen. Dieser Aufwand stand in keinem Verhältnis zum Nutzen, da für eine parallele Verteilung der Signale innerhalb des FPGAs genügend Leitungen zur Verfügung standen. Außerdem würden die Datenwandlungen am Businterface und in den Modulen Logikressourcen belegen, die so für die eigentliche Implementierung bereitstehen.

5.2.1 Aufteilung der FPGA-Logik in Bereiche

Ausgangspunkt der weiteren Betrachtung ist die Virtex-II-FPGA-Serie von Xilinx, die in elf verschiedenen Größen, d.h. Anzahlen an verfügbaren Gattern, Multiplizierer (MULT) und RAM-Blöcken (BRAM), erhältlich ist (Tabelle 5.1). Die Anzahl der Gatter ist hier in CLBs angegeben, die auf dem Chip in Form einer Matrix angeordnet sind. Je nach Chipgröße ändern sich die Ausmaße der Matrix, wie in der zweiten Spalte der Tabelle erkennbar ist.

FPGA-Typ (Virtex-II)	CLB Array: Reihe x Spalte	BRAM-Spalten	MULT-Spalten
XC2V40	8 x 8	2	2
XC2V80	16 x 8	2	2
XC2V250	24 x 16	4	4
XC2V500	32 x 24	4	4
XC2V1000	40 x 32	4	4
XC2V1500	48 x 40	4	4
XC2V2000	56 x 48	4	4
XC2V3000	64 x 56	6	6
XC2V4000	80 x 72	6	6
XC2V6000	96 x 88	6	6
XC2V8000	112 x 104	6	6

Tabelle 5.1: Verfügbare Ressourcen der Virtex-II-FPGAs

Zwischen den CLB-Spalten sind außerdem die Multiplizierer- und BRAM-Blöcke angeordnet, wobei für jeden FPGA-Typ die Anzahl an Multiplizierern und BRAM-Blöcken immer gleich groß ist (Spalten 3 und 4 in Tabelle 5.1). Durch diese Anordnung wird die homogene Struktur einer reinen CLB-Matrix unterbrochen und es entsteht eine Aufteilung der CLB-, MULT- und BRAM-Spalten nach folgenden Prinzipien:

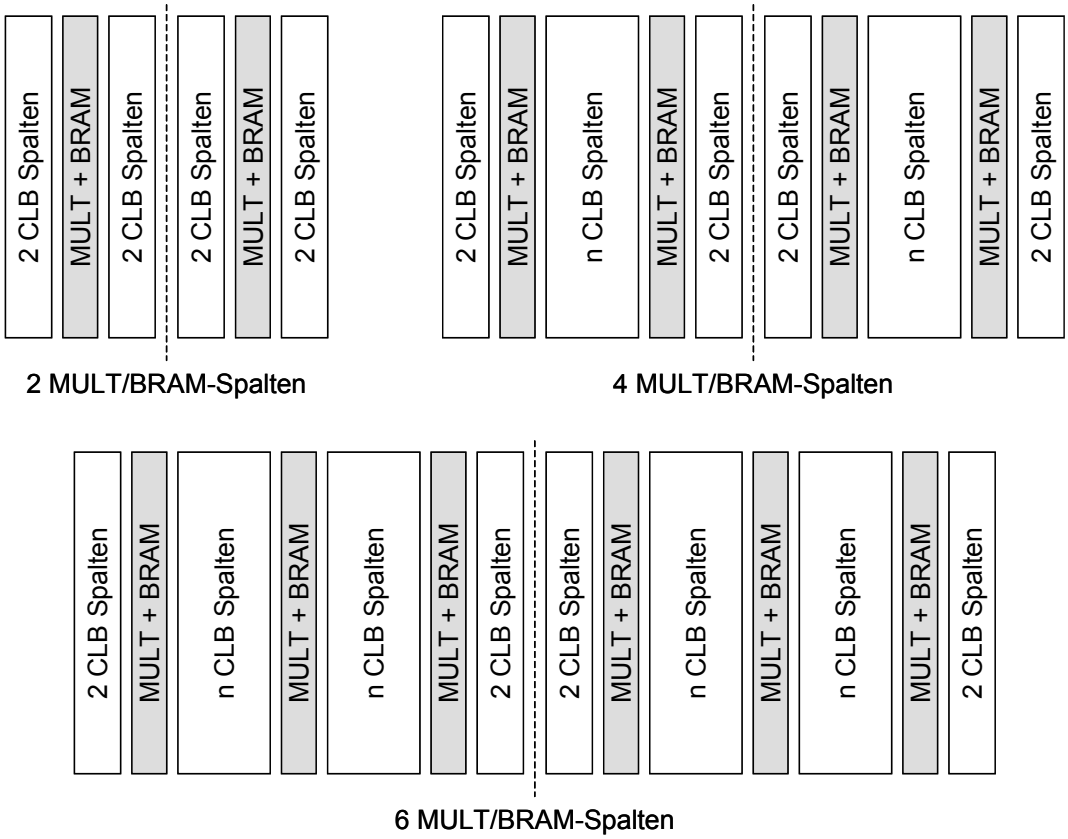

Abbildung 5.4: Verteilung der CLB-, MULT- und BRAM-Spalten in Virtex-II-FPGAs

Die Anordnung ist abhängig von der Anzahl der MULT/BRAM-Spalten im FPGA, so dass gemäß den Angaben in Tabelle 5.1 drei Typen zu unterscheiden sind, wie sie Abbildung 5.4 zeigt. Die noch unbekannten Werte für „n“ berechnen sich aus den Spaltenangaben aus der Tabelle und den bereits fest vorgegebenen CLB-Spalten nach folgenden Formeln:

$$CLBSp_{Ges} = 4 \cdot 2CLBSp + 2 \cdot nCLBSp \rightarrow n = \frac{CLBSp_{Ges} - 4 \cdot 2CLBSp}{2CLBSp}$$

Formel 5.1: Berechnung der CLB-Spalten bei 4-MULT/BRAM-Spalten

$$CLBSp_{Ges} = 4 \cdot 2CLBSp + 4 \cdot nCLBSp \rightarrow n = \frac{CLBSp_{Ges} - 4 \cdot 2CLBSp}{4CLBSp}$$

Formel 5.2: Berechnung der CLB-Spalten bei 6-MULT/BRAM-Spalten

Zur Umsetzung dieser Idee wird dieser Ansatz auf einen FPGA vom Typ Xilinx Virtex-II-1000 angewandt, der in Tabelle 5.1 grau hinterlegt ist. Da dieser FPGA-Typ vier MULT/BRAM-Spalten aufweist, erfolgt die Berechnung der „n“ CLB-Spalten gemäß Formel 5.1:

$$CLBSp_{Ges} = 32CLBSp \rightarrow n = \frac{CLBSp_{Ges} - 4 \cdot 2CLBSp}{2CLBSp} = \frac{32-8}{2} = 16$$

Formel 5.3: Berechnung der CLB-Spalten für Virtex-II-1000

Daraus ergibt sich eine Aufteilung der gesamten CLBs, wie sie in Abbildung 5.5 zu sehen ist.

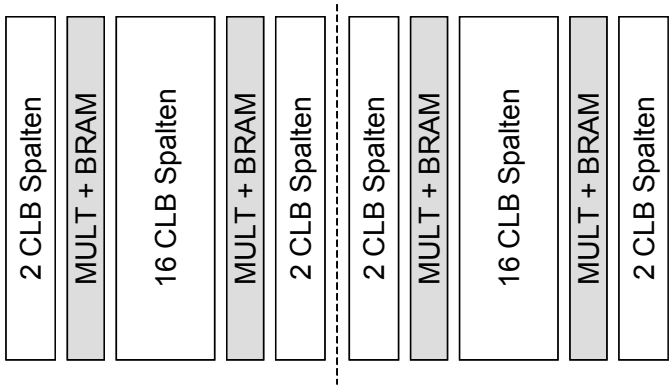

Abbildung 5.5: Aufteilung der CLB-, MULT- und BRAM-Spalten im Virtex-II-1000

Die Aufteilung der zur Verfügung stehenden Ressourcen auf dem FPGA wird nach folgenden Maßstäben durchgeführt:

- gleichmäßige Verteilung der Ressourcen (CLBs, BRAMs und Multiplizierer)
- möglichst einheitliche Größe der Bereiche
- für große IP-Cores (z.B. CAN- und LIN-Controller) ausreichend

Nach diesen Vorgaben wirde der FPGA in vier rechteckige Bereiche gleicher Größe aufgeteilt. Die Anzahl der Bereiche (Slots) wird hier so gewählt, dass eine homogene Struktur entsteht und die CLBs, sowie die BRAM- und Multiplizierer-Blöcke gleichmäßig auf alle Bereiche verteilt sind. Diese vertikal angeordneten Slots besitzen so eine Breite von acht CLB-Spalten und erstrecken sich über die gesamte Höhe des FPGA (siehe Abbildung 5.3). In jedem Slot ist je eine BRAM-Spalte und eine Multiplizierer-Spalte enthalten.

Die Slots stellen definierte Bereiche zur Verfügung, in denen die unterschiedlichen Komponenten des FPGA-Designs implementiert werden. Der rechte Slot (Slot D) nimmt die gesamte statische Logik auf, die permanent im Design vorhanden ist und sich nicht ändert. Die drei linken Slots A, B und C sind in der Lage, jeweils ein funktionales Modul aufzunehmen. Die dabei zur Verfügung stehenden Ressourcen, sind in Tabelle 5.2 dargestellt.

Ressource	Anzahl
CLB	320
Tri-State-Buffer	640
Slices	1280
Multiplizierer-Blöcke	10
BRAM-Blöcke	10

Tabelle 5.2: Verfügbare Ressourcen in einem Slot

Hinsichtlich der ausreichenden Größe der Bereiche wurde das Ressourcenangebot eines Slots mit den Ergebnissen von Syntheseläufen der CAN- und Lin-Controller verglichen. Die Logikanforderungen der beiden Cores sind in Tabelle 5.3 aufgeführt.

IP-Core	CLBs, bezogen auf FPGA		CLBs, bezogen auf einen Slot	
	belegt	prozentual	belegt	prozentual
CAN-Controller	218 von 1280	17%	218 von 320	68%
LIN-Controller	125 von 1280	9%	125 von 320	39%

Tabelle 5.3: Logikanforderungen CAN- und LIN-Controller

Damit sind die Bereiche genügend groß gewählt, um auch die genannten Cores aufzunehmen. Allerdings muss auch immer beachtet werden, dass nicht alleine die Anzahl der von einem Design belegten CLBs entscheidend für die letztlich benötigte Fläche auf dem FPGA ist, sondern auch die zur internen Verdrahtung nötigen Routing-Kanäle. Nicht alle IP-Cores, die in die CLBs eines Bereich zahlenmäßig hineinpassen, können dann auch geroutet werden. Daher sollte hier immer ein Spielraum gegeben sein.

5.3 Konzept des „Bitstream-Merging“

Ausgangspunkt für das Zusammensetzen eines Bitstreams bildet ein Basis-Bitstream, der nur das Framework ohne ein funktionales Modul enthält. Soll in dieses Framework nun ein funktionales Modul eingefügt werden, so benötigt man neben dem Basis-Bitstream einen weiteren, der das Framework und das funktionale Modul in einem der drei Slots enthält. Aus diesem werden die zu dem entsprechenden Slot gehörenden Konfigurationsspalten ausgeschnitten und an der gleichen Stelle im Basis-Bitstream eingesetzt (Abbildung 5.6).

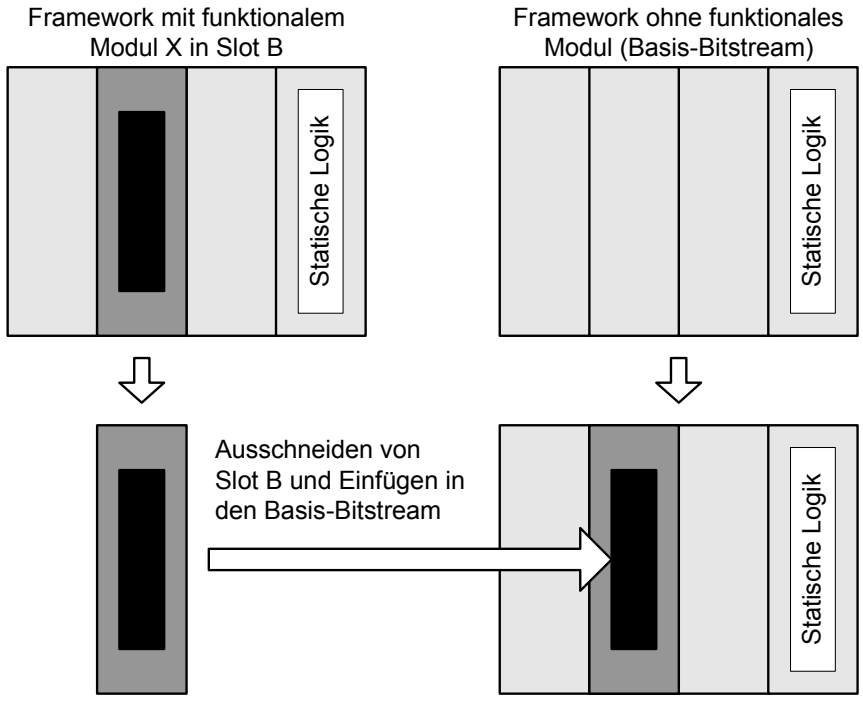

Abbildung 5.6: Vorgehensweise beim Zusammensetzen eines Bitstreams

Abbildung 5.7 stellt die Anordnung der verschiedenen Konfigurationsspalten des FPGA dar. Die obere Anordnung zeigt die Zuordnung der Konfigurationsspalten des Bitstreams zu den Bereichen der definierten Slot-Bereiche des FPGA. Die Konfigurationsspalten eines Slots umfassen acht CLB-Konfigurationsspalten, eine BRAM-Konfigurationsspalte und die dazugehörige BRAMINT-Konfigurationsspalte. Zusätzlich wird den Konfigurationsspalten von Slot A noch die linke IOB- und IOI-Konfigurationsspalte zugeordnet. Zu den übrigen Konfigurationsspalten, die weder in Slot A, B oder C enthalten sind, gehören die Konfigurationsspalten zur Konfiguration der statischen Logik in Slot D und die GCLK-Spalte. Die untere Anordnung zeigt, welche Konfigurationsspalten ersetzt werden müssen, um ein funktionales Modul in Slot B einzufügen. Dabei werden die zu Slot B gehörenden Konfigurationsspalten des Basis-Bitstreams mit den passenden Konfigurationsspalten aus einem anderen Bitstream mit dem entsprechenden Modul in Slot B ersetzt. Die ersetzten Konfigurationsspalten enthalten selbstverständlich auch Konfigurationsdaten, die zu dem implementierten Framework gehören. Deshalb ist es unbedingt notwendig, dass die Implementierung des Framework immer identisch bleibt.

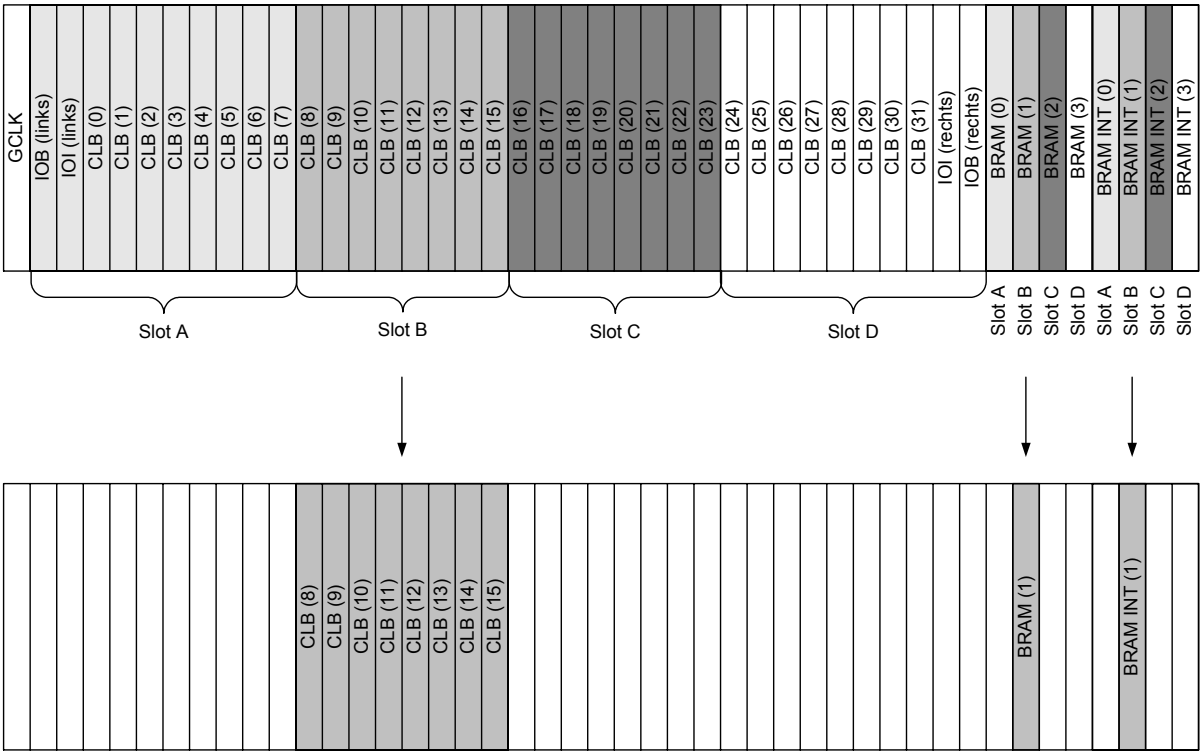

Abbildung 5.7: Aufteilung der Konfigurationsspalten in Slots

5.4 Hardware-Framework

Das Hardware-Framework stellt ein Rahmendesign dar, im dem alle Komponenten enthalten sind, die im Design permanent vorhanden und immer identisch sind. Abbildung 5.8 gibt eine Übersicht über die grundlegende Struktur des Toplevel-FPGA-Designs. Zu den genannten Basiskomponenten gehören:

- Globale Clock-Buffer zur Verteilung von Taktsignalen mit geringer Verzögerung und Taktversatz auf dem gesamten FPGA
- Digital Clock Manager (DCM) zur Generierung und Stabilisierung interner Taktsignale

- M9-Bus-Interface-Modul zur Steuerung der Kommunikation des FPGA mit dem externen Systembus der COMPASS-Plattform
- IO-Switch-Matrix-Modul zur Konfiguration der zusätzlichen Bussysteme und I/O-Schnittstellen
- Bus-Makros zum Aufbau interner Bussysteme im FPGA
- Drei Modul-Komponenten ohne Funktion für die zur Verfügung stehenden austauschbaren Slots
- Signale zur Verknüpfung der unterschiedlichen Komponenten

Die drei definierten Modul-Komponenten des Frameworks sind in der Lage, Module mit verschiedenen Funktionalitäten aufzunehmen. Sie werden im Folgenden als funktionale Module bezeichnet. Die Modul-Komponenten werden in Slot A, B und C platziert. Im Framework selbst enthalten diese Modul-Komponenten keine Funktion und werden in diesem Fall deshalb auch als Dummy-Module bezeichnet. Sie sind lediglich dazu da, bestimmte Signale der Modul-Komponenten, die mit dem Framework verbunden sind, in einen definierten Zustand zu versetzen.

5.4.1 Integration verschiedener Module

Um verschiedene Funktionalitäten zur Verfügung stellen zu können, wird eine Modul-Bibliothek mit unterschiedlichen funktionalen Modulen erstellt. Die Module werden bei Bedarf in das Framework eingebunden und ersetzen dabei ein Dummy-Modul ohne Funktion. Aufgrund der Aufteilung des FPGA in vier gleich große, vertikale Slots, von denen ein Slot für die statische Logik reserviert ist, stehen drei Slots zur Verfügung, die jeweils ein funktionales Modul aufnehmen können. Das Framework ist also in der Lage, drei funktionale Module mit unterschiedlichen Funktionen zu integrieren. Alle funktionalen Module besitzen dieselbe Schnittstellenbeschreibung. Dazu gehören ein Takt- und Reset-Signal und die Schnittstellen zum Systembus, dem IO-Bus 0 bzw. IO-Bus 1 sowie dem Peripheral Communication Bus (PCB).

5.4.2 Kommunikation der Module mit der Umgebung

Die Kommunikation der Modul-Komponenten mit der statischen Logik muss über fest definierte Signalleitungen und Verbindungspunkte erfolgen. Dies wird mit Hilfe eines internen Bus-Systems realisiert, das mit Bus-Makros aufgebaut wird. Aus diesem Grund werden Bus-Makros benötigt, welche im FPGA fest platziert werden können. Dadurch entstehen fest definierte Zugriffpunkte in den jeweiligen Slots. Die Module in den Slots müssen sowohl in der Lage sein, Daten zu empfangen als auch Daten zu senden. Dies erfordert einen bidirektionalen Datenbus. Andere Signale, die nur in einer Richtung transportiert werden, benötigen lediglich eine unidirektionale Leitung. Bei unidirektionalen Leitungen existiert immer nur ein Treiber auf der Busleitung, der durch Freischalten eines bestimmten Tri-State-Buffers festgelegt und danach nicht mehr verändert wird. Alle anderen Teilnehmer sind zwar in der Lage, das Signal auf der Busleitung zu lesen, können aber nicht auf den Bus schreiben. Um ein Bussystem sowohl mit Lese- als auch mit Schreibzugriff zu realisieren, müssen mehrere Teilnehmer in der Lage sein, auf den Bus zu schreiben. Die Steuerung der Tri-State-Buffer, die den Zugriff auf die internen Busleitungen erlauben, wird dabei entweder von einer Interface-Einheit im Framework oder den Modulen selbst übernommen. In jedem Fall muss verhindert werden, dass zwei Teilnehmer gleichzeitig auf dieselbe interne Busleitung schreiben, da dies zu einem Kurzschluss und damit zu einer Zerstörung des FPGAs führen kann. Abbildung 5.9 zeigt die grundlegende Struktur für das Kommunikationskonzept des Designs.

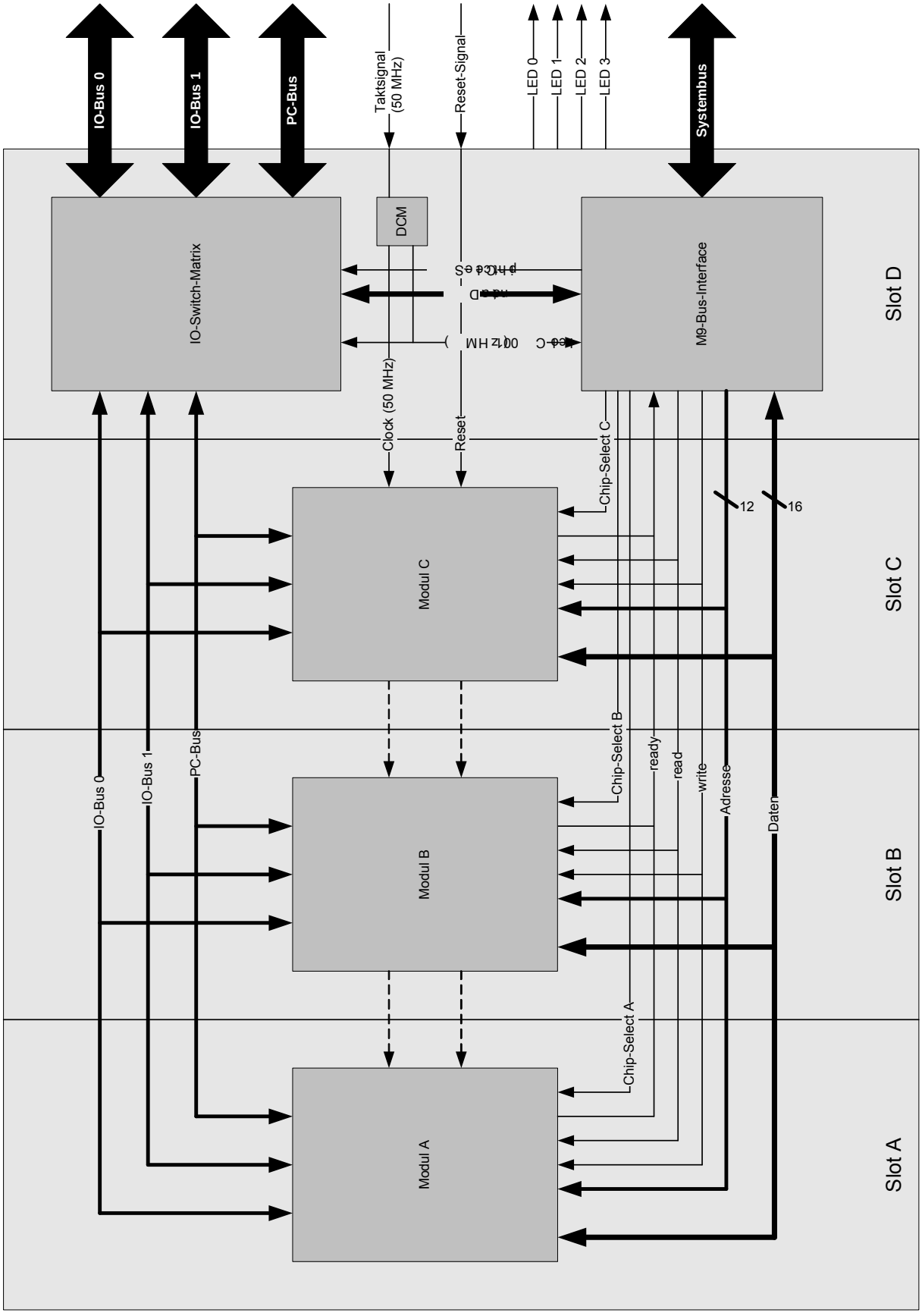

Abbildung 5.8: Struktur des Frameworks

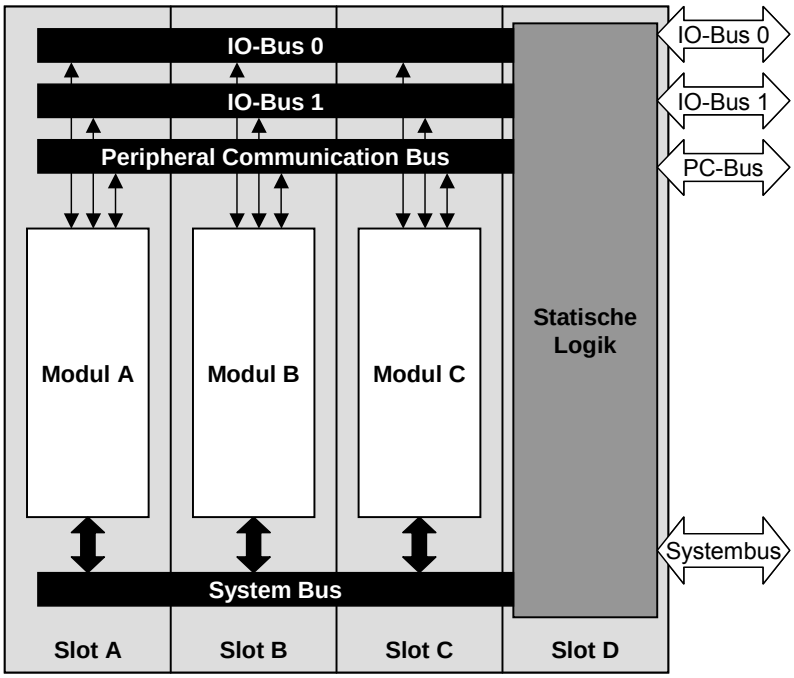

Abbildung 5.9: Kommunikationskonzept innerhalb des Frameworks

5.4.2.1 Bus-Makro

Um die interne Busstruktur auf dem FPGA zu ermöglichen, wird eine Struktur benötigt, die fest platziert werden kann und somit in jedem Slot feste Zugriffpunkte auf den internen Bus bereitstellt, die sich nicht mehr ändern. An diesen festen und definierten Positionen liegenden Punkten sollen die später eingesetzten Funktionsmodule die jeweils nötigen Bussignale abgreifen. Zur Verwirklichung einer solchen Struktur, auch Bus-Makro genannt, stehen Tri-State-Buffer und dedizierte Tri-State-Buffer-Leitungen zur Verfügung, die sich über die gesamte Breite des FPGAs ausdehnen.

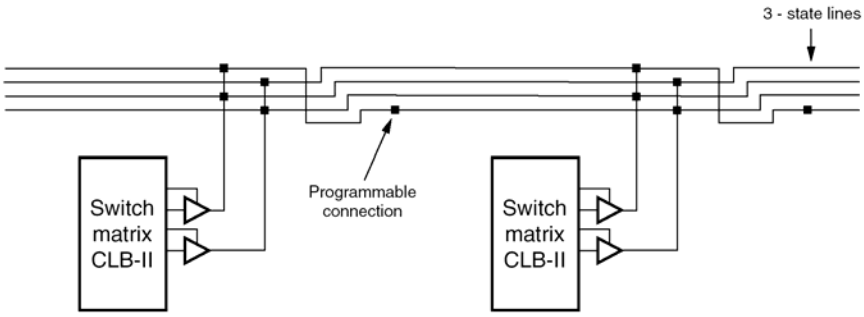

Abbildung 5.10: Tri-State-Buffer-Leitungen oberhalb einer CLB-Reihe

Oberhalb jeder CLB-Reihe verlaufen vier horizontale Tri-State-Buffer-Leitungen (Abbildung 5.10). Der Zugriff auf diese Leitungen in jedem Slot erfolgt über vier Tri-State-Buffer, wobei in jedem Slot jeweils zwei Tri-State-Buffer von zwei nebeneinander liegenden CLBs verwendet werden. Daraus ergibt sich, dass jedes Bus-Makro vier Signale transportieren kann. Der Aufbau des Bus-Makros leitet sich aus der gewählten Struktur des FPGA-

Designs ab. Abbildung 5.11 zeigt den strukturellen Aufbau des verwendeten Bus-Makros. In jedem der vier Slots befindet sich für jede Leitung des Bus-Makros ein Tri-State-Buffer, der das Schreiben auf die Busleitung ermöglicht. Die Ausgänge der Module werden auf die Eingänge der entsprechenden Tri-State-Buffer der Bus-Makros in dem jeweiligen Slot des FPGA geführt. Die Eingänge der Module greifen ein auf einer internen Busleitung anliegendes Signal direkt über eine Routing-Leitung von der Busleitung ab.

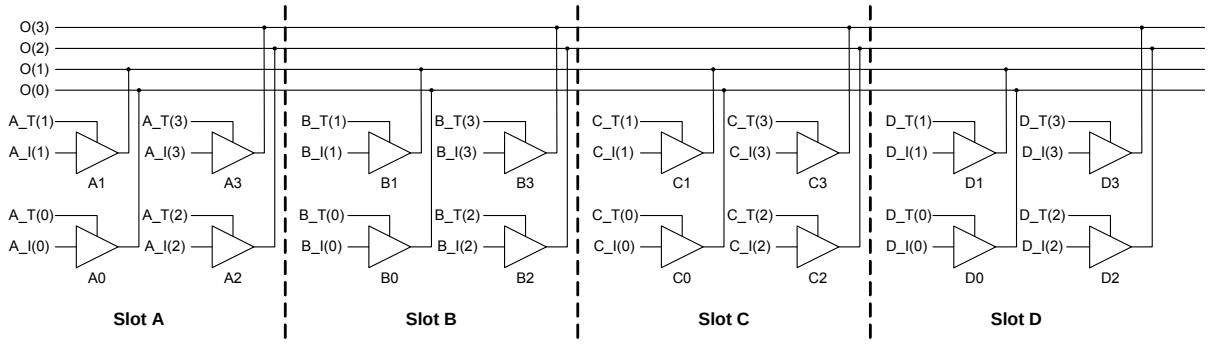

Abbildung 5.11: Struktureller Aufbau der verwendeten Bus-Makros

5.4.3 Systembus

Um einen Datenaustausch zwischen dem FPGA auf einer der Erweiterungskarten und dem zentralen Prozessor der COMPASS-Plattform zu ermöglichen, muss der FPGA an den Systembus der COMPASS-Plattform angeschlossen werden. Dies erfordert eine Umsetzung des vom zentralen Prozessor verwendeten Kommunikationsprotokolls im FPGA. Der ARM9-Prozessor der COMPASS-Plattform benutzt ein asynchrones Kommunikationsprotokoll. Die wesentlichen Steuersignale sind das Chip-Select-Signal, eine Write-Leitung und eine Read-Leitung, die alle low-aktiv sind. Sowohl der Adressbus als auch der Datenbus haben eine Breite von 16 Bit.

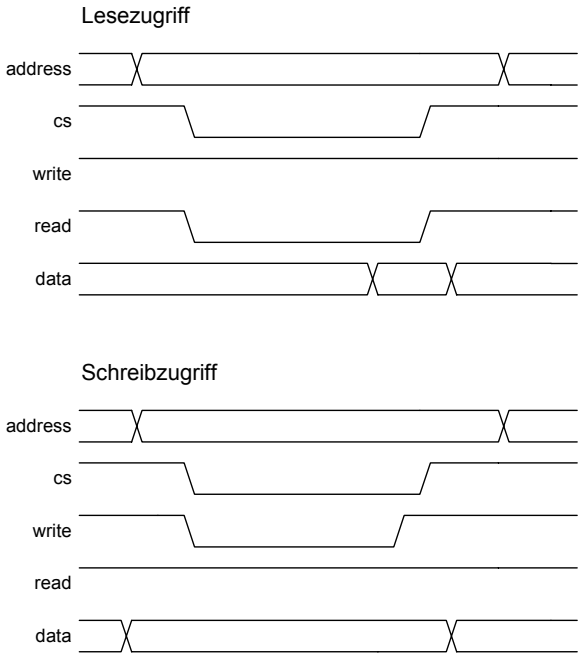

Abbildung 5.12: Prinzipieller Ablauf eines Lese- und Schreibzugriffs des ARM-Prozessors

Abbildung 5.12 zeigt den prinzipiellen Ablauf eines Lese- und Schreibzugriffs des Prozessors. Bei einem Lesezugriff wird eine Adresse auf den Adressbus gelegt. Danach wird das Chip-Select-Signal und die Read-Leitung mit einem Low-Pegel aktiviert. Bei der steigenden Flanke des Read-Signals werden die Daten, die auf dem Datenbus liegen, vom Prozessor gelesen. Gleichzeitig wird das Chip-Select-Signal zurückgenommen. Der Schreibzugriff verläuft ähnlich wie der Lesezugriff. Nach dem Anliegen einer gültigen Adresse auf dem Adressbus wird die Chip-Select-Leitung und die Write-Leitung aktiviert. Die Daten, die geschrieben werden sollen, liegen bereits auf dem Datenbus. Bei der steigenden Flanke des Write-Signals werden die Daten von dem durch das Chip-Select-Signal angesprochenen Kommunikationspartner vom Datenbus übernommen. Danach wird das Chip-Select-Signal deaktiviert. Die Dauer des Lese- und Schreibzugriffs kann durch Einfügen von Wait-States verlängert werden, wodurch sich der Buszugriff verlangsamt. Der Adressbus, der auf den FPGA geführt wird, besitzt eine Breite von 16 Bit. Mit den oberen vier Bit der Adresse werden die vier verschiedenen Slots des FPGA unterschieden. Die unteren 12 Bit adressieren in dem jeweils ausgewählten Slot die dort vorhandenen Register.

5.4.4 I/O-Schnittstellen und Peripheral Communication Bus

Neben dem Systembus müssen sowohl die zu jedem FPGA-Steckkartenplatz existierenden I/O-Busse als auch der Peripheral Communication Bus (PCB) im FPGA-Design berücksichtigt werden. Hierbei handelt es sich um einfache I/O-Schnittstellen, die der FPGA verwendet, um Daten mit Schnittstellenkarten auszutauschen. Die Leitungen dieser Bussysteme sollen in Abhängigkeit der verwendeten Funktionalität des Gesamtsystems entweder als Eingang oder Ausgang konfiguriert werden können.

5.4.5 Verteilung des Taktsignals auf dem FPGA

Die austauschbaren Module benötigen ein Taktsignal. Dieses Taktsignal ist das einzige Signal, das nicht über eine Leitung eines Bus-Makros zu den Modulen transportiert wird, da dies zu einer Verzögerung des Taktsignals und zu einem Taktversatz zwischen verschiedenen synchronen Elementen führt. Deshalb wird das Taktsignal über die dafür vorgesehenen globalen Taktnetze auf dem FPGA verteilt. Hierbei muss sichergestellt werden, dass das entsprechende Taktsignal auf dem gesamten FPGA vorhanden ist, da zu Beginn nicht vorhergesehen werden kann, an welchen Zugriffspunkten ein funktionales Modul in einem Slot das Taktsignal abgreift. Das Taktnetz muss also überall eingeschaltet sein, auch wenn es in einem Bereich nicht verwendet wird. Um eine genaue Zuordnung der Taktsignale sicherzustellen, muss ein Taktsignal immer dasselbe Taktnetz benutzen.

5.5 Modulerstellung

Die Erstellung der Module setzt spezielle Kenntnisse voraus und wird daher von einem erfahrenen Designer vorgenommen. Dessen Aufgabe ist es, Funktionen in Form von in VHDL- oder Verilog-Modellen zu programmieren, zu testen und danach dem Endanwender als Bibliothekselemente zur Benutzung mit der Plattform zur Verfügung zu stellen. Diese im folgenden auf- und durchgeführten Schritte sind für eine spätere, bloße Verwendung der Module nicht relevant.

Durch Implementierung und Generierung eines Bitstreams des Framework-Designs entsteht dabei ein erstes Basis-Design, das den Ausgangspunkt für die Integration von funktionalen Modulen in das Framework bereitstellt. Im nächsten Schritt werden verschiedene Module mit bestimmten Funktionalitäten erstellt, die anstelle der funktionslosen Modulkomponenten in das Framework eingesetzt werden können. Dabei wird so vorgegangen, dass immer nur eine Modul-Komponente durch ein funktionales Modul in einem der drei Slots des Frameworks ersetzt wird. Für dieses Design wird dann die Implementierung durchgeführt und ein Bitstream generiert. Auf diese Weise erhält man für jedes Modul drei Bitstreams, wobei das Modul nacheinander in jeweils einem der drei Slots implementiert wird, was Abbildung 5.13 zeigt. Diese Bitstreams bilden die Grundlage für die Java-Anwendung, die

einen Bitstream aus verschiedenen anderen Bitstreams zusammensetzt. Bei der Implementierung der austauschbaren Module in einem bestimmten Slot muss darauf geachtet werden, dass das jeweilige Modul die Grenzen des entsprechenden Slots nicht überschreitet. Sowohl die Platzierung als auch das Routing muss auf den zugehörigen Bereich des jeweiligen Slots beschränkt sein. Ist dies nicht der Fall, so wird beim Ausschneiden der entsprechenden Konfigurationsspalten aus dem dazugehörigen Bitstream nicht das ganze Modul erfasst und das Gesamtdesign kann nicht fehlerfrei zusammengesetzt werden. Ebenso muss sichergestellt werden, dass die Implementierung des Frameworks immer identisch bleibt. Dies ist notwendig, damit die Verbindungsleitungen, die auch durch die Bereiche der Slots A, B und C laufen, immer an derselben Stelle liegen und verbunden werden können. Nur so ist es möglich, das Gesamtdesign aus einzelnen Blöcken zusammensetzen.

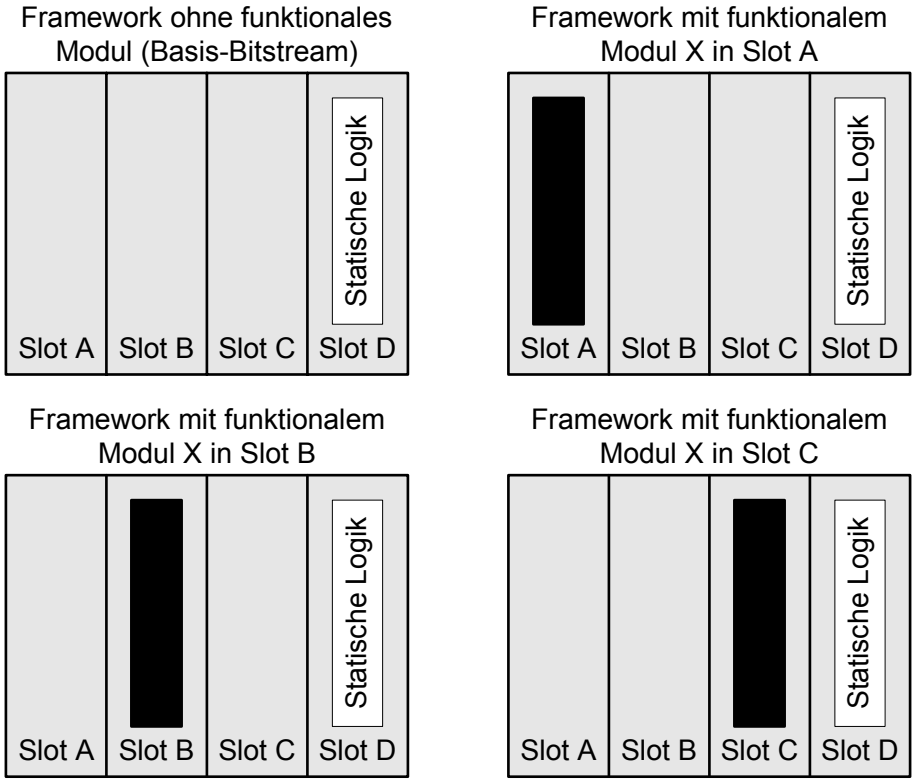

Abbildung 5.13: Schritte der Funktionsimplementierung für drei Slots

5.5.1 Design-Ablauf

Der Design-Ablauf beschreibt die gesamte Vorgehensweise von der Erstellung der VHDL-Quelldateien bis zur Generierung eines Bitstreams. Dazu werden die Xilinx ISE 6.3i [XISE05] und das Xilinx PlanAhead-Tool [Plan05] verwendet. Die Erstellung der VHDL-Quelldateien und die Synthese dieser VHDL-Dateien wird mit der Xilinx ISE durchgeführt. Für alle ISE-Projekte werden die gleichen Projekteinstellungen verwendet, die in Tabelle 5.4 zusammengefasst sind. Zur Implementierung des Designs und Generierung von Bitstreams wird das PlanAhead-Tool verwendet. Abbildung 5.14 stellt die wichtigsten Schritte des Design-Ablaufs in einem Flussdiagramm dar.

Projekteigenschaft	Einstellung
Device Family	Virtex-II
Device	xc2v1000
Package	fg256
Speed Grade	-5
Top-Level Module Type	HDL
Synthesis Tool	XST(VHDL/Verilog)
Simulator	ModelSim
Generated Simulation Language	VHDL

Tabelle 5.4: ISE-Projekteinstellungen

5.5.2 Netzliste des Toplevel-Designs

Im ersten Schritt des Design-Ablaufs wird eine Netzliste des Toplevel-Designs generiert. Für den Synthesevorgang wird in den Process Properties in der Rubrik Synthesis Options der Eintrag „Keep Hierarchy“ auf den Wert „Yes“ umgestellt, um die verschiedenen Design-Ebenen zu erhalten. Alle anderen Optionen bleiben unverändert. Das Ergebnis ist eine Netzliste mit dem Dateinamen toplevel.ngc.

5.5.3 Netzlisten der funktionalen Module

Für jedes der funktionalen Module wird ein eigenes ISE-Projekt in einem eigenen Ordner angelegt. Die VHDL-Datei mit der Beschreibung des jeweiligen Moduls wird dem Projekt hinzugefügt und synthetisiert. Dabei wird in den Process Properties des Synthesevorgangs in der Rubrik Xilinx Specific Options der Eintrag „Add I/O Buffers“ deaktiviert. Als Ergebnis erhält man für jedes der funktionalen Module eine Netzliste mit dem Dateinamen modulename „toplevel.ngc“.

5.5.4 Implementierung mit PlanAhead

Die Implementierung und Generierung der Bitstreams wird mit dem PlanAhead-Tool [Plan05] durchgeführt. Dazu wird ein Ordner Implementation angelegt. In diesem Ordner wird sowohl für das Framework, welches auch als Basis-Design bezeichnet wird, als auch für jedes vorhandene funktionale Modul ein Unterordner mit dem entsprechenden Namen des Moduls angelegt. Für die Module wird für jeden möglichen Slot, in welchem das Modul implementiert werden kann, ein weiterer Unterordner mit der Bezeichnung slotA, slotB und slotC erstellt. Für das Basis-Design wird der Ordner „base“ ausgewählt. Zu Beginn werden alle für die Implementierung benötigten Designdateien in den Unterordner mit dem Namen files kopiert, um einen schnelleren Zugriff und eine bessere Übersicht über die Dateien zu haben.

5.5.5 Anlegen eines Projekts

Im PlanAhead-Tool wird ein neues Projekt mit dem Namen „base“ angelegt und im Unterordner base gespeichert. Dabei wird die Netzliste des Toplevel-Designs „toplevel.ngc“ importiert. Nach der Wahl des gewünschten Zielbausteins und dem Hinzufügen der zum Toplevel-Design gehörenden Ucf-Datei „toplevel.ucf“ wird ein neuer Floorplan erstellt. Dieser Floorplan basiert auf den in der importierten Ucf-Datei enthaltenen Constraints. Dort sind definiert:

- Area-Groups (Bereiche zur Platzierung der Logik) für die statische Logik und die Modul-Komponenten
- Zuweisung der Ports des Toplevel-Designs an die physikalischen Pins des FPGA
- Feste Platzierung für Clock-Buffer und Digital Clock Manager
- Feste Platzierungen für die Bus-Makros

- Timing-Constraints des Taktsignals

Die Ucf-Datei „toplevel.ucf“ für das Gesamtdesign findet sich in [Stop06]. Die dort enthaltenen Area-Groups werden im Floorplan des PlanAhead-Tools als „PBlock“ dargestellt. Diese können beim Floorplanning in ihrer Größe und Form angepasst werden.

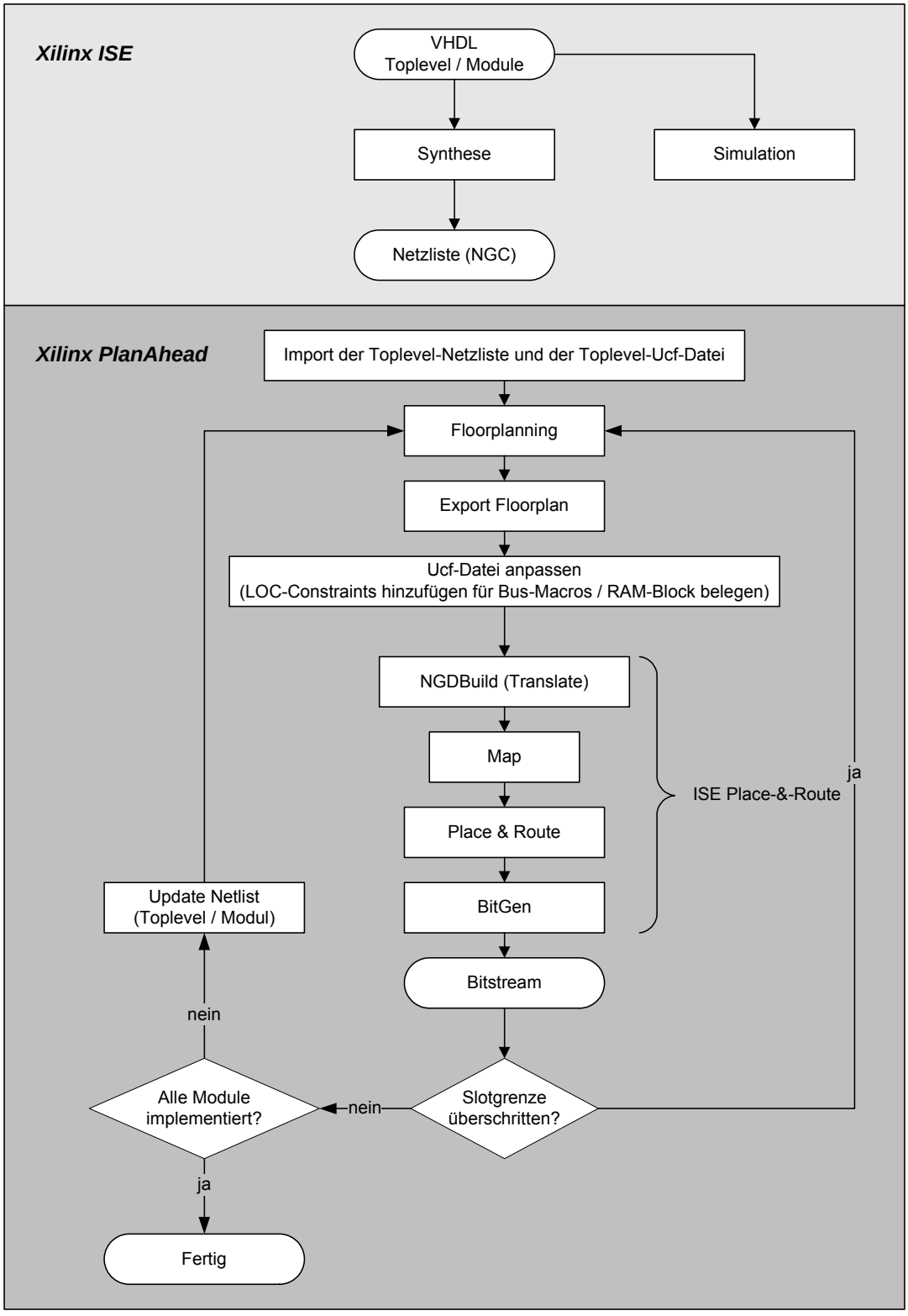

Abbildung 5.14: Design-Ablauf als Flussdiagramm

5.5.6 Exportieren des Floorplans

Um die Implementierung durchführen zu können, muss der Floorplan mit dem Befehl „File → Export Floorplan“ in das entsprechende Unterverzeichnis exportiert werden. Beim Exportieren des Floorplans werden die für das jeweilige Design benötigten Netzlisten des Toplevel-Designs und des eingesetzten funktionalen Moduls automatisch in diesen Ordner kopiert. Die Makro-Datei des Bus-Makros bus4macro.nmc muss dem Ordner manuell hinzugefügt werden. Außerdem wird von PlanAhead im entsprechenden Ordner eine Ucf-Datei angelegt, in welche die ursprünglichen Constraints aus der importierten Ucf-Datei und vom PlanAhead-Tool erzeugte Constraints übernommen werden. Die LOC-Constraints zur Platzierung der Bus-Makros werden in der neuen Ucf-Datei nicht berücksichtigt, da PlanAhead den Instanzen des Bus-Makros keine Komponentenbeschreibung zuordnen kann, da das Bus-Makro zu diesem Zeitpunkt noch als Black Box betrachtet wird. Aus diesem Grund ist es erforderlich, der exportierten Ucf-Datei in dem entsprechenden Unterordner mit einem Texteditor die LOC-Constraints für die Bus-Makros nachträglich manuell hinzuzufügen. Nach dem Abspeichern der Ucf-Datei kann mit der Implementierung des Designs begonnen werden.

5.5.7 ISE Place-&-Route und Bitgen

Die Implementierung und die Generierung eines Bitstreams kann im PlanAhead-Tool mit dem Befehl Tools → Run ISE Place-&-Route durchgeführt werden. In den folgenden Dialogfenstern können die Optionen für die aufeinander folgenden Implementierungsschritte angepasst werden. Für die Implementierung werden folgende Optionen verwendet:

- Translate: Es werden die Standardoptionen verwendet.
- Mapping: Die Option Remove Unused Logic (no -u) wird deaktiviert, damit unbenutzte Logikkomponenten nicht entfernt werden. Für die Implementierung des Frameworks mit einem funktionalen Modul in einem Slot wird im Feld Guide mode (-gm) die Option Exact ausgewählt. Als Guide-Datei wird in dem Feld Guide file (-gf) die geroutete Ncd-Datei top routed.ncd des Frameworks ohne funktionales Modul angegeben, die im Ordner base abgelegt ist. Bei der ersten Implementierung des Frameworks ohne ein funktionales Modul wird der Guide mode nicht verwendet.
- Place-&-Route: Für die Auswahl des Guide mode (-gm) gelten dieselben Vorgaben wie für das Mapping. Bei der Implementierung des Frameworks ohne funktionales Modul (Basis-Design) wird der Guide mode nicht benutzt. Bei der Implementierung des Frameworks mit einem funktionalen Modul wird im Feld Guide mode (-gm) wieder die Option Exact gewählt. Als Guide-Datei wird im Feld Guide file (-gf) hier ebenfalls die geroutete Ncd-Datei top routed.ncd des Basis-Designs im Ordner base benutzt.
- Bitgen: Es werden die Standardoptionen verwendet. Danach wird die Implementierung und die Generierung des Bitstreams gestartet.

5.5.8 Erstellen aller Bitstreams

Im ersten Schritt wird das Framework (Basis-Design) ohne ein funktionales Modul implementiert und der dazugehörige Bitstream erzeugt. Dieses Basis-Design wird dann als Grundlage für die Implementierung der funktionalen Module in den drei Slots verwendet. Im nächsten Schritt wird die Netzliste der Instanz „inst slotA“, welche die Modul- Komponente in Slot A beschreibt und die im Basis-Design nur ein Dummy-Modul enthält, durch eine Netzliste eines der funktionalen Module ersetzt. Dazu wird der Befehl „File → Update Netlist“ aufgerufen, die Option „Replace a specific module“ ausgewählt und aus dem „files“-Ordner die entsprechende Netzliste des funktionalen Moduls geladen. Die von dem eingesetzten Modul belegten Ressourcen und die prozentuale Belegung der Ressourcen in der „Area-Group“ werden im Fenster „Pblock Properties“ angezeigt. Nach dem Anpassen der Größe der Area-Group wird der Floorplan wie in Abschnitt 5.5.6 beschrieben exportiert und danach die Implementierung durchgeführt (Abschnitt 5.5.7). Diese Prozedur wird für alle Module nacheinander in Slot A durchgeführt. Danach wird die Toplevel-Netzliste wieder durch die ursprüngliche Version oh-

ne ein funktionales Modul ersetzt, um das Design wieder in den Ausgangszustand zu bringen. Dazu wird erneut der Befehl File/Update Netlist aufgerufen, die Option „Replace the entire design“ bestätigt und die Netzliste des Toplevel-Designs „toplevel.ngc“ im Ordner Unterordner „files“ ausgewählt. Nun wird die Netzliste der Instanz „inst slotB“ nacheinander mit den Netzlisten der verschiedenen funktionalen Module ersetzt. Die gleiche Prozedur wird im Anschluss mit der Instanz inst slotC durchgeführt. Sind alle funktionalen Module in den gewünschten Slots implementiert, so existiert für jedes Design ein eigener Bitstream. Die Bitstreams, die in ihren jeweiligen Unterordnern alle den gleichen Namen top.bit tragen, werden in einen eigenen Ordner kopiert und umbenannt in modulename_slotX.bit. Abbildung 5.15 zeigt die drei für jedes funktionale Modul generierten Bitstreams mit den dazugehörigen Namen.

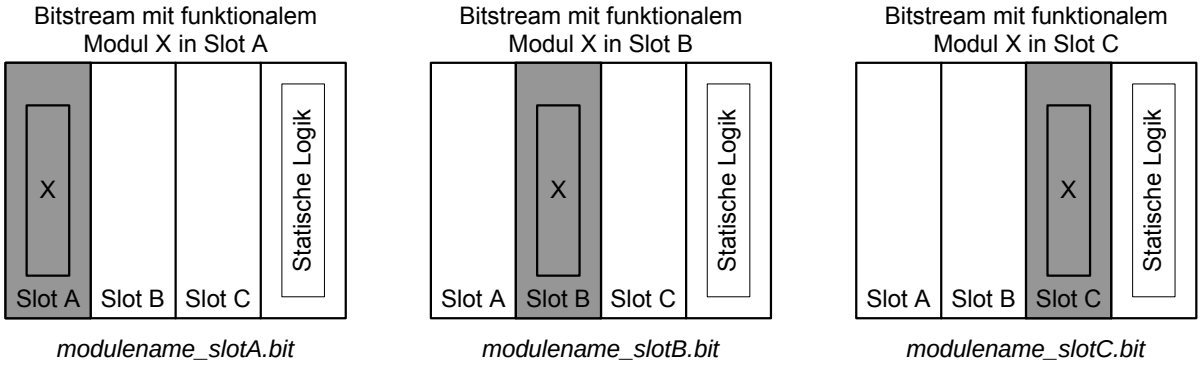

Abbildung 5.15: Generierung von drei Bitstreams für jedes funktionale Modul

5.5.9 Abweichungen vom Standard-Design-Ablauf

Das Modul „can“, welches einen CAN-Controller realisiert, nutzt aufgrund seiner Größe sehr viele Logikressourcen eines Slot-Bereiches. Aus diesem Grund ist es unmöglich, dass alle Routing-Leitungen auf den Bereich eines Slots beschränkt bleiben. Deshalb werden für das Modul „can“ zwei Slots zur Implementierung reserviert. Dazu werden die Slots A und B verwendet. Die Implementierung in Slot B und C ist nicht möglich, da hier die Slot-Grenzen sowohl auf der linken als auch der rechten Seite überschritten werden. Das Modul „lin“ kann nur in Slot A implementiert werden. Die Implementierung in Slot B oder C konnte bis jetzt nicht erfolgreich abgeschlossen werden, da zu viele Routing-Leitungen den benachbarten Bereich benutzen. Eine weitere Besonderheit dieses Moduls stellt die Tatsache dar, dass im LIN-Controller ein RAM-Block verwendet wird, der die Werte zum Einstellen der verschiedenen Bit-Breiten eines LIN-Frames speichert. Aus diesem Grund müssen manuell vor der Implementierung dieses Moduls in der Ucf-Datei des dazugehörigen exportierten Floorplans Constraints hinzugefügt werden, die den RAM-Block beim Starten mit vorher festgelegten Werten laden. Diese zusätzlichen Constraints für die Implementierung des lin-Moduls in Slot A sind in [Stop06] abgedruckt.

Einige funktionale Module, die in einem bestimmten Slot implementiert werden, benutzen Routing-Leitungen im benachbarten Slot-Bereich. Dieser Fehler kann durch einen zusätzlichen manuellen Routing-Schritt behoben werden. Dazu wird die geroutete Design-Datei „top routed.ncd“ im FPGA-Editor geöffnet. Die Leitungen, die durch den Bereich des benachbarten Slots laufen, werden mit dem Befehl „unroute“ aufgetrennt und mit dem Befehl „autoroute“ erneut automatisch verdrahtet. Dabei sucht sich der Autorouter oftmals eine andere Verbindung, welche auf den dazugehörigen Slot-Bereich beschränkt bleibt, als bei der ursprünglichen Implementierung des Designs. Das veränderte Design wird gespeichert. Im Anschluss muss nur noch ein neuer Bitstream aus dem aktualisierten Design generiert werden. Dazu

wird in der Kommandozeile in dem Ordner, der die Dateien des entsprechendes Designs enthält, folgender Befehl ausgeführt:

```
bitgen -w topffrouted.ncd top.bit top.pcf
```

Bei der Implementierung eines funktionalen Moduls in Slot C kann es passieren, dass beim Abgreifen eines Signals von einer internen Busleitung (z.B. IO-Bus 0) das Signal nicht von der Busleitung direkt abgenommen wird, sondern von der Verbindungsleitung zwischen der Busleitung und der IO-Switch-Matrix. Dies kann nur durch erneutes Floorplanning behoben werden. Dabei muss der Pblock im PlanAhead-Tool so weit wie möglich verkleinert und in der unmittelbaren Umgebung der Busleitung platziert werden.

5.5.10 Vergleich der Bitstreams

Jeder Bitstream, der ein funktionales Modul in einem der Slots enthält, wird mit einer Java-Funktion namens CompareBitstream mit dem Bitstream des Basis-Designs verglichen. Als Ergebnis zeigt die Funktion an, in welchen zu einem bestimmten Slot gehörenden Konfigurationsspalten sich die beiden Bitstreams in ihren Konfigurationsdaten unterscheiden. In dem Slot, in welchem sich ein funktionales Modul befindet, gibt es selbstverständlich eine Vielzahl von Differenzen zum Bitstream des Basis-Designs. Existieren auch in den übrigen Konfigurationsspalten, die zu einem anderen Slot gehören, Abweichungen, so kann der generierte Bitstream nicht verwendet werden. Solch ein Fehler muss durch Anpassen des Floorplannings und einem weiteren Implementierungsversuch behoben werden.

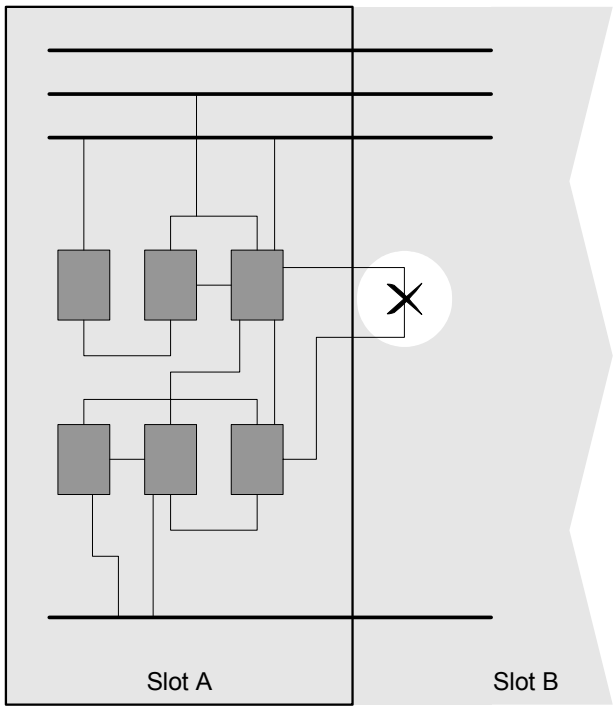

Abbildung 5.16: Fehlerhafte Implementierung eines Moduls in Slot A

Die Abweichungen des Bitstreams in dem Slot neben einem implementierten funktionalen Modul kommen dadurch zustande, dass das Routing unter der verwendeten Xilinx ISE 6.3i nicht auf den Bereich des jeweiligen Slots des funktionalen Moduls beschränkt werden kann und so Verbindungsleitungen unter Umständen den benachbarten Slot benutzen. Dieses Problem wurde zumindest bis zur Version 8.1i auch nicht gelöst. Obwohl die Abweichungen

teilweise minimal sind und nur einige wenige Bit im Bitstream betreffen, kann dieser Bitstream nicht verwendet werden, weil beim Ausschneiden des Moduls an den Slot-Grenzen Routing-Leitungen zerstört würden, was das Modul unbrauchbar macht. Abbildung 5.16 zeigt eine fehlerhafte Implementierung eines funktionalen Moduls in Slot A. Eine Routing-Leitung, die zwei zu dem funktionalen Modul in Slot A gehörende Logikelemente verbindet, läuft durch den Bereich von Slot B.

5.5.11 Aktivierung des Taktsignals auf dem FPGA

Eine Besonderheit des Design-Ablaufs stellt das Aktivieren der Taktleitungen auf dem gesamten FPGA dar. Da sich im Basis-Design kein funktionales Modul in einem der Slots A, B oder C befindet und kein Taktsignal benötigt wird, werden die Verbindungspunkte des Taktsignals zur Verzweigung in die Quadranten des FPGA nicht aktiviert. Diese Verbindungspunkte werden in der GCLK-Konfigurationsspalte konfiguriert. Da die GCLK-Spalte jedoch die Konfiguration des Basis-Designs enthält und beim Einfügen eines Moduls in einen der Slots niemals ausgetauscht oder verändert wird, müssen diese Verzweigungspunkte im Basis-Bitstream manuell aktiviert werden. Dies wird mit der Java-Funktion ConfigureGCLK durchgeführt. Die Klasse ConfigureGCLK beschreibt ein Java-Programm, welches das Taktsignal für die funktionalen Module auf dem gesamten FPGA verfügbar macht. Dazu werden im Bitstream des Basis-Designs diejenigen Bits gesetzt, welche die Treiber der Zweige des Taktnetzes auf die linke und rechte Seite des FPGA aktivieren.

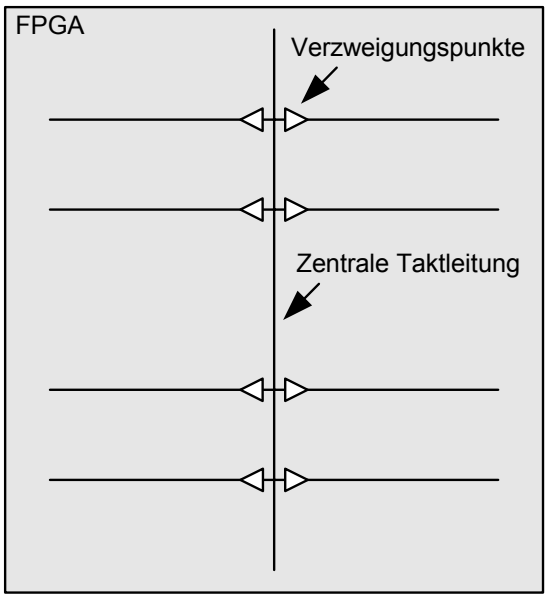

Abbildung 5.17: Aktivierung der Verzweigungspunkte des Taktnetzes

Abbildung 5.17 zeigt den schematischen Aufbau eines Taktnetzes und die Anordnung der Verzweigungspunkte. In der Mitte des FPGA verläuft in vertikaler Richtung die Hauptleitung des Taktnetzes, die über einen Clock Buffer vom Digital Clock Manager mit einem Taktsignal versorgt wird. Mit Hilfe von jeweils vier Zweigleitungen auf der linken und der rechten Seite wird das Taktsignal auf dem ganzen FPGA verteilt. Damit steht es für die funktionalen Module in allen Slots zur Verfügung und kann an einer beliebigen Stelle auf dem FPGA abgegriffen werden.

5.5.12 XML-Konfigurationsdateien der funktionalen Module

Für jedes funktionale Modul wird eine XML-Datei angelegt, die Informationen zu dem jeweiligen Modul enthält. Dazu gehören:

- Name des Moduls
- Beschreibung der Funktion des Moduls
- Anzahl der benötigten Slots
- Belegte Busleitungen der Bussysteme IO-Bus 0, IO-Bus 1 und Peripheral Communication Bus mit Angabe der verwendeten Signalrichtung
- Benötigte Schnittstellenkarten auf der COMPASS-Plattform

Im Folgenden ist ein Ausschnitt aus der XML-Konfigurationsdatei des Moduls lin als Beispiel dargestellt:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bitstream>
  <module>
    <name>lin</name> Name
    <description>LIN-Controller</description> Funktionsbeschreibung
  </module>
  <resources>
    <card class="auto" type="1" slot="pcB"/> Schnittstellenkarte
    <modulewidth number="1"/> Anzahl der belegten Slots
    <bus name="ioB0" width="10"> Ressourcenbelegung IO-Bus 0
      <signal index="0" state="unused"/> nicht benutzt
      <signal index="1" state="unused"/> nicht benutzt
      ...
    </bus>
    <bus name="ioB1" width="10"> Ressourcenbelegung IO-Bus 1
      <signal index="0" state="unused"/> nicht benutzt
      <signal index="1" state="unused"/> nicht benutzt
      ...
    </bus>
    <bus name="pcB" width="16"> Ressourcenbelegung PC-Bus
      <signal index="0" state="unused"/> nicht benutzt
      <signal index="1" state="unused"/> nicht benutzt
      ...
      <signal index="12" state="input"/> Eingang
      <signal index="13" state="output"/> Ausgang
      ...
    </bus>
  </resources>
</bitstream>
```

5.6 Modulbibliothek

Die Modul-Bibliothek besteht aus verschiedenen funktionalen Modulen, welche die im Toplevel-Design enthaltenen Dummy-Module ersetzen. Die Auswahl der Module orientierte sich an den Erfordernissen des Automotive-Bereichs, wobei zunächst allerdings nur ein Teil der dort benötigten Schnittstellen umgesetzt wurde, um die grundsätzliche Funktion der Methodik und des Gesamtsystems zu validieren. Als wichtigste Komponenten sind hierbei die beiden Kommunikationsschnittstellen CAN und LIN, sowie die rein digitale Signalein - und -

ausgabe und ein Modul zur Ansteuerung eines 12bit-A/D-Wandlers zu nennen. Diese sind auch im Folgenden näher beschreiben.

Alle Module besitzen dieselbe Schnittstellenbeschreibung, welche der Komponentenbeschreibung der im Toplevel-Design definierten und instanziierten Modul-Komponenten entspricht, was bedeutet, dass alle Port-Beschreibungen in den VHDL-Entities der funktionalen Module identisch sind. In allen VHDL-Beschreibungen der funktionalen Module wird der Ausgangsport ready mit einem Low-Pegel verknüpft. Dadurch wird dem M9-Bus-Interface bei einem Kommunikationsvorgang signalisiert, dass sich in dem angesprochenen Slot ein funktionales Modul befindet, mit dem eine Kommunikation stattfinden kann.

5.6.1 Ansteuerung D/A-Wandler (Modul dac)

Das Modul dac realisiert die Ansteuerung eines D/A-Wandlers auf einer der vorhandenen I/O-Schnittstellenkarten. Der D/A-Wandler wird über eine SPI-Schnittstelle mit drei Leitungen angesprochen. Dazu gehören das Chip-Select-Signal spi_csn, das Datensignal spi_out und die Taktleitung spi_clk. Die Ausgabe der Signale erfolgt über den IO-Bus 0, was im folgenden VHDL-Code dargestellt ist:

```
io_b0_out(1) <= spi_csn;
io_b0_out(3) <= spi_out;
io_b0_out(5) <= spi_clk;
io_b0_enable <= "1111010101";
```

Da das Taktsignal clk mit einer Frequenz von 50 MHz zu schnell ist, um als Taktsignal für die SPI-Schnittstelle zu dienen, wird das Taktsignal mit einem Taktteiler auf einen kleineren Wert heruntergeteilt. Die SPI-Schnittstelle wird mit einem Zustandsautomaten realisiert. Dieser Zustandsautomat, der in einer Endlosschleife läuft und von der steigenden Flanke des intern erzeugten Taktsignals clock intern getriggert wird, schiebt die Bits im Register spi_reg nacheinander auf die Datenleitung spi_out. Dabei wird mit dem höchstwertigen Bit begonnen. Das Register spi_reg hat eine Breite von 16 Bit und setzt sich aus den beiden im Modul definierten Registern dac_control_reg und dac_data_reg zusammen, was in Abbildung 5.18 dargestellt ist. Das Kontrollregister, das eine Breite von 3 Bit aufweist, legt fest, welcher der beiden D/A-Kanäle angesprochen wird und wann die Register des D/A-Wandlers aktualisiert werden. Der Datenteil hat eine Breite von 12 Bit. Die genaue Bedeutung der Kontrollbits kann in [Max5230] nachgeschlagen werden.

Registeradresse	Registername	Bitbreite	Funktion
0x000	dac_control_reg	3	Konfiguration des D/A-Wandlers
0x002	dac_data_reg	12	Datenausgabe an den D/A-Wandler

Tabelle 5.5: Register des Moduls dac

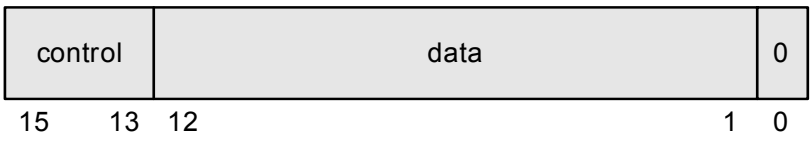

Abbildung 5.18: Aufteilung des Registers spi_reg

5.6.2 CAN-Controller (Modul can)

Das Modul „can“ realisiert einen CAN-Controller, der als synthetisierte Netzliste mit dem Namen „can_top“ zur Verfügung steht. Die Netzliste des CAN-Controllers wird von [OpCo06] zur Verfügung gestellt. Die Struktur und der Registersatz des IP-Cores basiert auf dem CAN-Controller [SJA1000] von Philips, so dass dessen Datenblatt zur Programmierung herangezogen werden kann. In der Modulbeschreibung des Moduls „can“ wird eine Instanz der Komponentenbeschreibung dieser Netzliste instanziiert und mit den entsprechenden Ports und Signalen der Modulbeschreibung verknüpft, was der folgende VHDL-Code zeigt:

```
inst_can_top : can_top
  PORT MAP (
    clk_i => clk,
    rst_i => reset,
    rx_i => pcb_in(12),
    tx_o => pcb_out(13),
    bus_off_o => OPEN,
    clkout_o => OPEN,
    addr_i(7 DOWNTO 0) => addr_shift(7 DOWNTO 0),
    rd_i => NOT read,
    wr_i => NOT write,
    data_i(7 DOWNTO 0) => data_in(7 DOWNTO 0),
    data_o(7 DOWNTO 0) => data_out(7 DOWNTO 0),
    cs_i => NOT cs,
    irq_o => OPEN );
```

Der CAN-Controller besitzt einen Adresseingang mit einer Breite von acht Bit. Dateneingang und Datenausgang sind ebenfalls acht Bit breit. Der zentrale Prozessor der COMPASS-Plattform führt jedoch immer 16-Bit-Zugriffe aus, so dass nur jede zweite Adresse verwendet werden kann. Deshalb werden die unteren acht Bit der auf das Modul geführten Adresse addr um eine Bitstelle nach rechts verschoben und auf den Adresseingang addr(i) des CAN-Controllers geführt, um ein einzelnes Byte im CAN-Controller adressieren zu können:

```
addr_shift(7) <= ’0’;
addr_shift(6 DOWNTO 0) <= addr(7 DOWNTO 1);
```

Die Empfangs- und Sendeleitung rx_i und tx_i des CAN-Controllers werden mit den Leitungen 12 und 13 des Peripheral-Communication-Bus verbunden. Die Konfiguration des Peripheral-Communication-Bus ist im folgenden VHDL-Code dargestellt:

```
pcb_out(15 DOWNTO 14) <= "11";
pcb_out(12 DOWNTO 0) <= "1111111111111";
pcb_enable <= "1101111111111111";
```

Der CAN-Controller ist kompatibel zum CAN-Controller SJA1000 von Philips. Die Funktionsweise des CAN-Controllers kann in [SJA1000] nachgelesen werden.

5.6.3 LIN-Controller (Modul lin)

Der im Rahmen dieser Arbeit entwickelte LIN-Controller hat die Aufgabe, das LIN-Protokoll sowohl spezifikationskonform [LINS20] umzusetzen, als auch mit Hilfe von benutzten Parametern die Veränderung bestimmter Eigenschaften des Protokolls gezielt vorzunehmen. Der Übertragungsrahmen ist Abbildung 5.19 dargestellt.

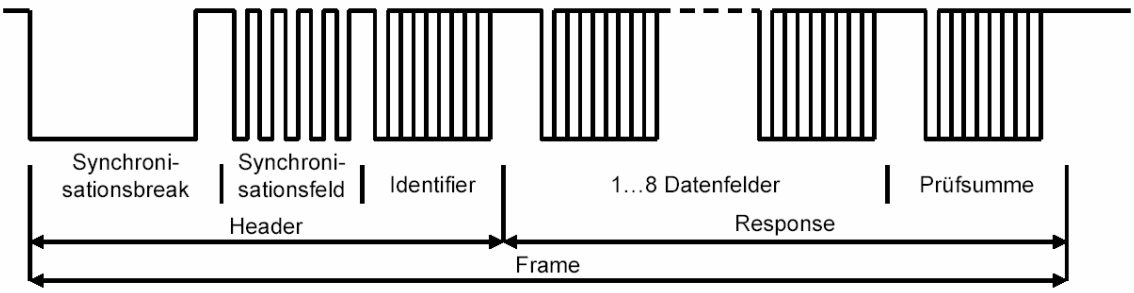

Abbildung 5.19: Ein LIN-Frame mit Header und Response

Die Manipulation des Protokolls umfasst die freie Wahl einer Basisübertragungsrate und die variable Einstellung der Bitzeiten für jedes einzelne Bit. Darüber hinaus können sowohl die Werte der Datenbits, die gesendet werden sollen, für alle Felder (Synchronisationsbreak, Synchronisationsfeld, Identifier, Datenbyte, Checksumme) einzeln und beliebig eingestellt als auch die dazugehörigen Start- und Stopbits, die Anfang und Ende eines Feldes kennzeichnen, frei gewählt werden. Dadurch lassen sich beliebige Kombinationen des Synchronisations-Breaks mit dem dazugehörigen Break-Delimiter, des Synchronisationsfeldes oder des Identifiers mit den darin enthaltenen Parity-Bits erzeugen. Weiterhin kann die Anzahl der Datenbytes zwischen null und acht variiert werden. Aufgrund dieser variablen Parameter kann das LIN-Protokoll auf unterschiedlichste Art verändert werden. Die Kombination verschiedener Manipulationsformen erlaubt so den Entwurf und die Durchführung beliebiger Testszenarien.

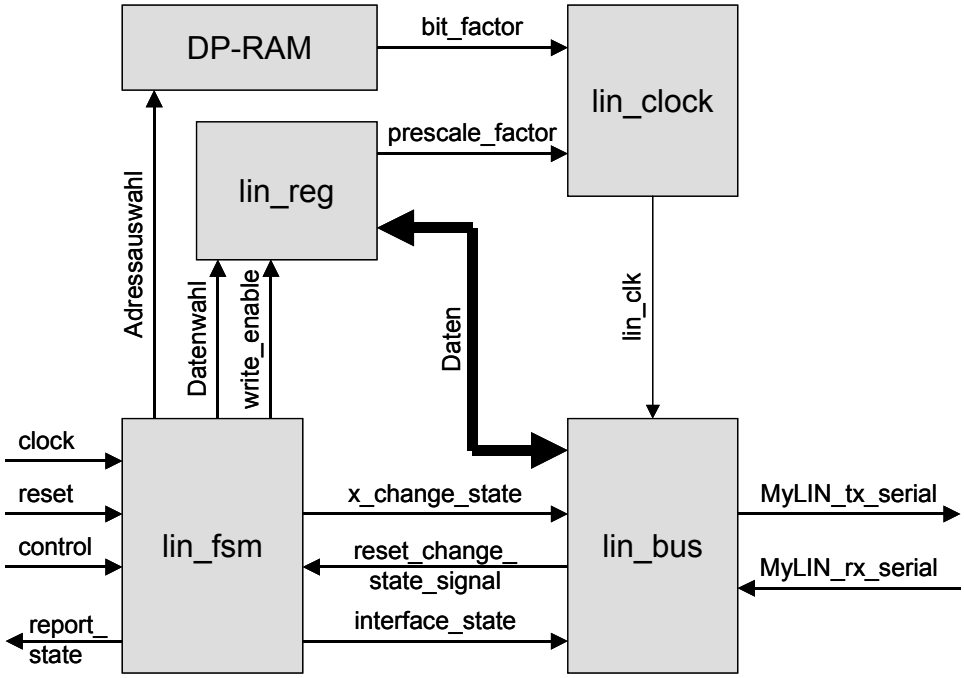

Abbildung 5.20: Aufbau des LIN-Cores mit Fehlerinjektionsmöglichkeit

Die Funktionen des LIN-Cores sind der Übersichtlichkeit wegen auf vier Module aufgeteilt, die jeweils in einer eigenen VHDL-Datei mit einer entsprechenden Entity beschrieben sind. Abbildung 5.20 zeigt die eben genannten Module in einem Blockschaltbild und ihre wichtigsten Verbindungen untereinander. Das Modul „lin_fsm“ enthält einen Zustandsautomaten, welcher für die Steuerung zuständig ist, das Modul „lin_bus“ bildet die Schnittstelle nach außen zum Anschluss des Transceivers und das Modul „lin_clock“ erzeugt ein Taktsignal mit variabler Periodendauer zur Erzeugung unterschiedlicher Bitbreiten. Daneben existieren noch ein Modul „lin_reg“ zur Speicherung und Auswahl von Daten und Parametern und ein Dual-Port-RAM-Baustein, der ebenfalls zur Speicherung von Parametern dient.

Das Modul „lin“ beinhaltet einen LIN-Controller zur Fehlerinjektion für LIN-Bus-Systeme und steht als synthetisierte Netzliste mit dem Namen „lin_top“ zur Verfügung. In der Modulbeschreibung wird eine Instanz des LIN-Controllers instanziiert und mit den entsprechenden Ports und Signalen der Modulbeschreibung verknüpft. Der LIN-Controller besitzt als Schnittstellen eine Vielzahl von Ports, die zur Konfiguration des LIN-Controllers verwendet werden. Diese Ports werden mit Registern verknüpft, die in der Modulbeschreibung angelegt werden. In Tabelle 5.6 sind die im Modul enthaltenen Register aufgelistet.

Adresse	Name	Bits	Funktion
0x800	control_reg	8	Kontrollregister
0x802	bytenumber_in_reg	4	Anzahl der Datenbytes des LIN-Frames
0x804	prescale_in_reg	8	Einstellen der Übertragungsrate
0x806	tx_syncbreak_reg	8	Synchronisationsbreak (Senden)
0x808	tx_syncfield_reg	8	Synchronisationsfeld (Senden)
0x80A	tx_identifier_reg	8	Identifier (Senden)
0x80C	tx_data1_reg	8	Datenbyte 1 (Senden)
0x80E	tx_data2_reg	8	Datenbyte 2 (Senden)
0x810	tx_data3_reg	8	Datenbyte 3 (Senden)
0x812	tx_data4_reg	8	Datenbyte 4 (Senden)
0x814	tx_data5_reg	8	Datenbyte 5 (Senden)
0x816	tx_data6_reg	8	Datenbyte 6 (Senden)
0x818	tx_data7_reg	8	Datenbyte 7 (Senden)
0x81A	tx_data8_reg	8	Datenbyte 8 (Senden)
0x81C	tx_checksum_reg	8	Checksumme (Senden)
0x81E	tx_bits_reg_low	16	Start- und Stopbits (15..0) (Senden)
0x820	tx_bits_reg_high	8	Start- und Stopbits (23..16) (Senden)
0x822	rx_syncbreak_reg	8	Synchronisationsbreak (Empfangen)
0x824	rx_syncfield_reg	8	Synchronisationsfeld (Empfangen)
0x826	rx_identifier_reg	8	Identifier (Empfangen)
0x828	rx_data1_reg	8	Datenbyte 1 (Empfangen)
0x82A	rx_data2_reg	8	Datenbyte 2 (Empfangen)
0x82C	rx_data3_reg	8	Datenbyte 3 (Empfangen)
0x82E	rx_data4_reg	8	Datenbyte 4 (Empfangen)
0x830	rx_data5_reg	8	Datenbyte 5 (Empfangen)
0x832	rx_data6_reg	8	Datenbyte 6 (Empfangen)
0x834	rx_data7_reg	8	Datenbyte 7 (Empfangen)
0x836	rx_data8_reg	8	Datenbyte 8 (Empfangen)
0x838	rx_checksum_reg	8	Checksumme (Empfangen)
0x83A	rx_bits_reg_low	16	Start- und Stopbits (15..0) (Empfangen)
0x83C	rx_bits_reg_high	8	Start- und Stopbits (23..16) (Empfangen)
0x83E	report state_reg	8	Statusinformation

Tabelle 5.6: Register des LIN-Moduls

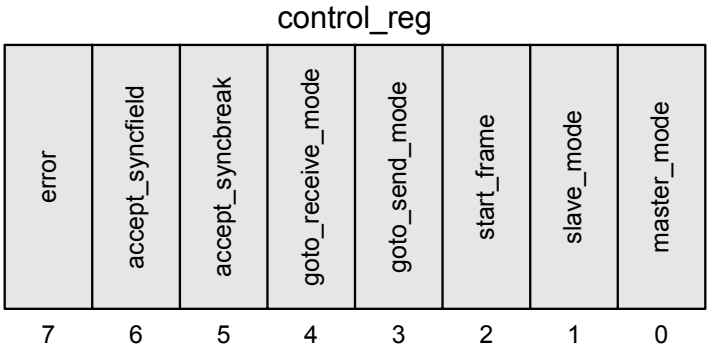

Abbildung 5.21: Belegung des Kontrollregisters

Weiterhin existiert im LIN-Controller ein RAM-Block, der ebenfalls angeschlossen wird. Der Adressbereich im LIN-Controller wird so aufgeteilt, dass die Adressen 0x000 bis 0x7FF den im LIN-Controller enthaltenen RAM-Block adressieren, während die Adressen von 0x800 bis 0xFFF für die angelegten Register zur Verfügung stehen. Als Sende- und Empfangsleitung werden identisch zum Modul „can“ die Leitungen 12 und 13 des Peripheral-Communication-Bus (PCB) verwendet. Die Belegung des Kontrollregisters „control_reg“ kann Abbildung 5.21 entnommen werden. Die detaillierte Funktionsweise des LIN-Cores ist in [Stop05] ausführlich beschrieben.

5.6.3.1 Einstellung der Bitzeiten

Die variable Einstellung einer Basisübertragungsrate und der Periodendauern des Signals lin_clk wird damit erreicht, dass mit dem Wert des Signals prescale_factor die Basisübertragungsrate gewählt wird, während mit dem Wert des Signals bit_factor die variablen Bitbreiten eingestellt werden. Der Wert des Signals prescale_factor gibt die Anzahl der Perioden des Signals clock an, die gezählt werden müssen, bis das Signal prescale_clock eine Periode vollendet hat. Die Taktfrequenz des internen Zwischentaktsignals prescale_clock berechnet sich dementsprechend nach folgender Formel:

$$f_{prescale\_clock} = \frac{f_{clock}}{prescale\_factor} \rightarrow T_{prescale\_clock} = T_{clock} \times prescale\_factor$$

Formel 5.4: Berechnung der LIN-Clock-Prescalers

Hundert Perioden des Signals prescale_clock ergeben dann die Standardbitbreite eines Bits bei der gewünschten Basisübertragungsrate. Der Wert des Signals bit_factor gibt die Anzahl der Perioden des Signals prescale_clock an, die durchlaufen werden, bis eine Periode des Signals lin_clk abgeschlossen ist. Die Taktfrequenz des Signals lin_clk wird demnach folgendermaßen berechnet:

$$f_{lin\_clk} = \frac{f_{prescale\_clock}}{bit\_factor} \rightarrow T_{lin\_clk} = T_{prescale\_clock} \times bit\_factor$$

Formel 5.5: Berechnung der Frequenz der LIN-Clock

Die exakte Periodendauer eines Bit lässt sich mit folgender Formel berechnen:

$$T_{bit} = T_{clock} \times prescale\_factor \times bit\_factor$$

Formel 5.6: Berechnung der Bitzeit einer LIN-Übertragung

Um ein Taktsignal mit der standardmäßigen Periodendauer der gewünschten Basisübertragungsrate zu erhalten, wird als Standardwert 100 für das Signal bit_factor gewählt. Damit erhält man die Formel zur Berechnung der von der Basisübertragungsrate abgeleiteten Standardbitbreite zu:

$$T_{bit(Basis)} = T_{clock} \times prescale\_factor \times 100$$

Formel 5.7: Berechnung der Standard-Bitzeit (100%)

Bei einer Breite von 8 Bit des Signals bit_factor ist die Periodendauer einstellbar in einem Bereich von 10% bis 250% der Standardbitbreite. Außerhalb dieses Bereichs kommt es zu Unregelmäßigkeiten des Taktsignals lin_clk, das Signal ist nicht mehr symmetrisch. Dieser Parameterbereich sollte daher vermieden werden.

5.7 Abschließende Bewertung der Designmethode

Mit der hier vorgestellten Design-Methodik Slot-basierter Funktionsimplementierung wird es auch einem im Umgang mit FPGAs und deren Programmierung ungeübten Benutzer keine Probleme bereiten, die in dieser Arbeit vorgestellte COMPASS-Plattform zu konfigurieren. Da die Funktionsimplementierung lediglich durch Auswahl der gewünschten Komponenten aus einer vorgegebenen, aber erweiterbaren, Bibliothek geschieht, sind keine Kenntnisse in VHDL, Verilog oder SystemC oder ein detailliertes Verständnis der FPGA-Interna nötig.

Die Bibliotheksmodule sind getestet und auf den jeweiligen Slot hin optimiert. Sie orientieren sich hinsichtlich der implementierten Funktionen an den Erfordernissen und Anforderungen für Rapid Prototyping-Szenarien und Hardware-in-the-Loop-Tests. Gegenüber anderen Push-Button-Lösungen, wie z.B. dem Xilinx SystemGenerator, kann hier die gleich bleibende Performanz der Module garantiert werden, unabhängig von der Häufigkeit der Verwendung und der Kombination mit anderen Modulen auf dem FPGA. Dies wird bedingt durch das Zusammenfügen der einzelnen Module-IPs auf Bitstream-Ebene – ohne die Durchführung heuristik-basierter Platzierungs- und Routing-Läufe, die oftmals auch nicht funktionsfähige Lösungen hervorbringen.

Nach dieser Implementierung durch Verschmelzen der einzelnen Bitstreams zu einem gesamten, ist der Zugriff auf die (Register der) einzelnen Module einer Konfiguration über eine definierte Schnittstelle möglich. Währenddessen wird automatisch eine sowohl Menschals auch Maschinen-lesbare XML-Konfigurationsdatei erzeugt. Daraus können weiterhin Daten zur automatisch Schnittstellendokumentation einer Implementierung extrahiert bzw. C/C++-Header-Dateien für spätere Software-Implementierungen generiert werden. Innerhalb des kompletten Implementierungsablaufs dient diese Datei aber auch als Basis zur Freischaltung von Ressourcen auf den FPGAs nach dem Laden der Bistreams.

Die konsequente Fehlerreduktion bis hin zum Fehlerausschluss wird durch Konfigurationsprüfungen gegen die jeweiligen Nachbarslots und die darin bereits enthaltenen Modul-IPs

erreicht. Fehlkonfigurationen werden so bereits bei der Zusammenstellung entdeckt und können durch den Tool-seitigen Ausschluss nicht zu Fehlern führen.

Eine Limitierung dieser Art der FPGA-Konfigurierung ist die Tatsache, dass eine bestimmte Anzahl von Slots mit fest definierten, identischen Breiten vorgegeben ist. Bei Modulen, die nur einen geringen Teil der innerhalb der Slot-Grenzen verfügbaren Ressourcen belegen, wird trotzdem die gesamte Fläche des Slots okkupiert.

5.8 Sicherheitskonzept

Da der Benutzer verschiedene Einstellungen in der Software vornehmen, bzw. das System in seinem Hardware-Aufbau verändern kann, z.B. durch Stecken oder Entfernen von Karten, ist es empfehlenswert, Sicherheitsmechanismen auf verschiedenen Ebenen vorzusehen, die für einen fehlerfreien Betrieb sorgen. Dabei bleibt zu beachten, dass das System nicht „Hot-Plugging-fähig ist, und sich dadurch Fehler, die durch Fahrlässigkeit entstehen, wie das Herausziehen oder Stecken von Interface-, I/O- oder Automotive-Karten während des Betriebs, nicht vermeiden lassen. Sie führen zum Absturz des Systems oder im schlimmsten Fall zu einer Zerstörung der jeweiligen Karte bzw. der gesamten Plattform.

Im Folgenden wird beschrieben, welche Maßnahmen getroffen wurden, um ein hohes Maß an Sicherheit zu erreichen. Dabei wird auf die Fehler eingegangen, die von der Hardware oder der Software erkannt werden können und wie das System, bzw. die Steuerung auf dem Host-PC, darauf reagiert. Die Zusammenstellung ist in verschiedene Ebenen gegliedert, beginnend von der Hardware bis hin zu den Steuerungsapplikationen. Einzelne Aktionen lassen sich allerdings nicht völlig getrennt beschreiben, da sie mit anderen interagieren, bzw. verschiedene Maßnahmen und Vorkehrungen für eine Aktion nötig sind.

Das Sicherheitskonzept hat außerdem den Sinn, die Handhabung und den Umgang mit der Plattform zu vereinfachen. Zum einen soll die Plattform vor Fehlkonfigurationen und damit vor partieller oder kompletter Zerstörung geschützt, zum anderen aber auch der Benutzer möglichst vor Schäden an seiner Person bewahrt werden.

Plattform-Hardware

Um die einwandfreie Funktion der Plattform zu gewährleisten und Bedienungsfehler auszuschließen, sind bereits erste Maßnahmen auf Seiten der Hardware getroffen, die eine spätere Überprüfung unterstützen.

Sämtliche Steckkarten wurden mit einem EEPROM ausgerüstet, das einen Code enthält, der die Karte eindeutig identifiziert (Tabelle 5.7). Damit lässt sich recht einfach feststellen, ob ein Slot überhaupt belegt ist und wenn ja, mit welcher Karte oder welcher Art von Karte. Interface-Karten unterscheiden sich so z.B. durch die Serie des implementierten FPGAs, während bei den I/O-Karten die Art der realisierten Schnittstellen angegeben ist. Im Falle der Automotive-Karten ist außerdem hinterlegt, auf welche Leitungsressource sie zugreifen.

Feld	Bedeutung	Wertebereich
0	Karten-Klasse	1=FPGA, 2=IO, 3=AUTOMOTIVE
1	Karten-ID	1-0xFFFF
2	Karten-Seriennummer	1-0xFFFF
3	Aktive Ausgangstreiber	Bitmaske für alle 16 Pins des I/O-Steckers

Tabelle 5.7: Bedeutung der ersten vier 16Bit-Felder der Karten-EEPROMs

Durch die gleiche Auslegung der Automotive- und der I/O-Steckplätze, können diese alternativ verwendet werden, d.h. beide Slot-Varianten können sowohl I/O- als auch Automoti-

ve-Karten aufnehmen. Es entstehen keine Defekte durch Verwechslung der Slots, da die Pinbelegung in Bezug auf die Spannungsversorgung identisch ausgelegt ist.

Eine Schutzschaltung mit Bustreibern verhindert Zerstörungen durch falsch eingesetzte Kombinationen von Automotive-Karten. Die plattformseitigen Ausgänge dieser Karten werden mit einer Freischaltung aktiviert oder deaktiviert (siehe auch Abbildung 4.11). Dadurch wird sicher gestellt, dass nicht zwei Karten gleichzeitig die selben Leitungen des Busses belegen wollen und beispielsweise zwei Ausgangsstufen gegeneinander treiben, bzw. nicht konfigurierte Portpins des FPGAs mit einem Spannungspegel beaufschlagen. Nach dem Einschalten der Plattform sind alle Automotive-Karten grundsätzlich deaktiviert. Die Freischaltung erfolgt nach einer Prüfung der Gegebenheiten durch die Software.

Power-Management-Unit (PMU)

Neben der Hauptaufgabe der PMU, die COMPASS-Plattform mit Energie zu versorgen, sind zusätzliche Vorkehrungen getroffen, die einen sicheren Betrieb gewährleisten sollen. Dazu werden schon beim Hochfahren der Plattform zunächst nur eine Versorgungsspannung, nämlich die des Prozessors, angelegt und nach dessen erfolgreichem Start die restlichen in einer definierten Reihenfolge zugeschaltet. Dies garantiert das korrekte Starten der auf der Plattform befindlichen FPGAs. Außerdem überprüft die PMU kontinuierlich die Ausgangsspannungen zur Plattform. Stellt sie eine Unter- oder Überspannung auf einzelnen Kanälen fest, wird von einem Fehler in der Versorgung oder auf der Plattform ausgegangen und der jeweilige Kanal abgeschaltet.

Zum Schutz des Micro-9-Systems prüft die Power-Management-Unit ständig den Status des Mikrocontrollers. Dies geschieht über die Alarmleitung des Prozessor-Watchdogs. Wenn der Watchdog des Controllers auslöst, werden automatisch alle Betriebsspannungen der FPGA-Steckkarten abgeschaltet. So wird beispielsweise verhindert, dass das Mikrocontroller-System durch fehlerhafte Buszugriffe der FPGA-Karten seine Software nicht mehr korrekt ausführen kann.

Aufgetretene Fehler bzw. Abschaltungen aufgrund von erkannten Fehlern werden dem Benutzer durch eine rote LED angezeigt.

BitstreamComposer

Die Zusammenstellung der FPGA-Inhalte in Form von einzelnen Modulen ist grundsätzlich eine fehlerträchtige Angelegenheit, insbesondere bei einer rein manuellen Implementierung. Allerdings wird durch das Verfahren selbst, d.h. durch die vorgegebenen und geprüften Module, deren definierte Größe und Performanz und die durchgehende Automatisierung, bereits ein hohes Maß an Sicherheit gewährleistet. Jedoch könnte es beim Zusammenfügen immer noch passieren, dass sich zwei oder mehr durch den Benutzer frei gewählte Module gegenseitig beeinflussen, in dem sie die selben Ausgangsleitungen verwenden oder dass Module eingefügt werden, die zu groß für den verleibenden Rest der Gatter sind.

Um dies zu verhindern prüft die Software die Anordnung bei jedem Einfügeschritt auf Konsistenz und Korrektheit der Ressourcenbelegung. Treten solche Konflikte auf, wird eine Fehlermeldung generiert und der Teilvorgang abgebrochen. Als Basis der Tests dienen XML-Dateien, die den Inhalt der IP-Core-Module anhand der wichtigsten Charakteristika beschreiben.

Nach der erfolgreichen Erzeugung einer Konfiguration für den FPGA, werden diese XML-Dateien dann zu einer Datei zusammengeführt, die ihrerseits wieder die Merkmale der Gesamtimplementierung enthält. Bei diesem Vorgang werden außerdem die für die Freischaltung der I/Os benötigten Registerwerte durch interne Berechnungen bestimmt. Die so entstandene XML-Datei wird später als Grundlage zum Test der Anforderungen gegen die vorhandene Hardware auf der Plattform verwendet.

CompassControl

Die bereits oben beschriebenen Hardware-seitigen Sicherheitsoptionen werden in dieser Host-Applikation geprüft. Direkt nach dem Starten und dem Verbinden mit der Plattform werden die EEPROMs aller Steckkarten abgefragt. Die Information (siehe Tabelle 5.7) über Funktionalität und Ressourcenbelegung der einzelnen Karten werden als Kopie der ersten vier Worte der EEPROMs innerhalb der Software angelegt. Da ein Hot-Plugging der Karten von der Compass-Plattform nicht unterstützt wird, ist es auch nicht notwendig, die Kompatibilität von Einsteckkarten zu einem späteren Zeitpunkt erneut zu checken. Das Ergebnis der erstellten Plattformkonfiguration wird farblich auf der Applikationsoberfläche angezeigt.

Beim Laden eines FPGA-Bitstreams wird die Verträglichkeit des Bitstreams mit den verfügbaren Einsteckkarten und den Bitstreams auf anderen eventuell vorhanden FPGA-Karten geprüft. Da diese Information nicht im Bitstream selbst abgespeichert ist, wird die bereits erwähnte XML-Datei, die während der Konfigurationserzeugung mit Hilfe des BitstreamComposers erstellt wurde, herangezogen. Sie ist im gleichen Pfad im Dateisystem abgelegt und enthält eine Referenz auf den zugehörenden Bitstream.

So ist es möglich zu prüfen, ob alle für den Betrieb des FPGA mit dem assoziierten Bitstream benötigten I/O-Karten vorhanden sind. Weiterhin ist es möglich sicherzustellen, dass die treibenden Pins des FPGA nicht mit aktiven Bustreibern auf den I/O-Karten kollidieren. Sollte mehr als ein Bustreiber dieselbe(n) Leitung(en) treiben wollen, werden die entsprechenden Karten als Konfigurationsfehler markiert und erscheinen im Hauptfenster der Applikation rot.

Als nächstes werden die bis zu zwei vorhandenen dedizierten I/O-Busse der FPGA-Karten überprüft. Für die Zuordnung zwischen FPGA-Karte und der daran angeschlossenen I/O-Slots gibt es zwei Möglichkeiten, wie leicht anhand des Screenshots in Abbildung 6.9 (Screenshot CompassControl) zu sehen ist. Die ersten beiden FPGA-Karten besitzen zwei I/O-Slots, die letzten beiden jeweils nur einen. Das Überprüfen der einzelnen Leitungen der I/O-Slots wird dann nacheinander für jeden der vier FPGAs durchgeführt, indem bei jeder I/O-Leitung die Anzahl der Ausgangstreiber aufsummiert wird und bei mehr als einem Treiber je Leitung ein Konfigurationsfehler signalisiert wird.

Da in der XML-Datei zu einem Bitstream auch Informationen bezüglich der benötigten Steckkarten abgelegt sind, erfolgt vor dem Laden des Bitstreams auch ein Vergleich von erforderlichen und vorhandenen Karten. Benötigt eine Konfiguration z.B. einen externen A/D-Wandler auf einer I/O-Karte, muss dieser vorhanden sein, damit die Software den Download des Bitstreams auf die Plattform startet und den FPGA konfiguriert. Andernfalls wird der Vorgang nicht ausgeführt. Dasselbe gilt für die Verträglichkeit der Automotive-Karten untereinander. Der Vorgang läuft daher nahezu identisch ab. Da deren Ausgänge standardmäßig über Bustreiber, die auf den Karten selbst untergebracht sind, ausgeschaltet sind, müssen sie zum Funktionieren seitens der Software frei geschaltet werden. Dies geschieht erst nach allen Prüfungen und der daraufhin erfolgten Karten- bzw. FPGA-Programmierung. Danach werden auch erst die Treiber, die sich intern an den Portpins des jeweiligen FPGAs befinden, über Register eingestellt und freigegeben. Die Einträge und deren Position sind in der globalen XML-Datei hinterlegt. Sie wurden bei der Zusammenführung der Module, d.h. bei Belegung der FPGA-internen I/O-Busse, aus deren Bedarf an Ein- und Ausgabeleitungsressourcen berechnet.

6 Software-Umgebung

6.1 Software für RP-HiL-Plattform

Für den Betrieb der COMPASS-Plattform wird ein Embedded Linux als Betriebssystem verwendet, das bereits auf dem verwendeten Mikrocontroller-Board Hypercontrol MICRO-9 vorinstalliert ist. Das ausgelieferte Linux-System baut daher für die entwickelte RP-HiL-Plattform COMPASS als Basis der Software-Applikationen auf dieser Installation auf. Hierzu werden im Folgenden zum einen das Betriebssystem selbst und zum anderen die zum Betrieb der Plattform notwendige Hauptapplikation näher betrachtet.

6.1.1 Betriebssystem Embedded Linux

Das Micro-9-System wird bereits mit einem vorinstallierten Embedded-Linux ausgeliefert. Zur Erstellung eines eigenen Kernels oder eigener Dateisysteme wird das Embedded Linux Build System ”PTXdist“ der Firma Pengutronix kostenfrei mitgeliefert. Für den Betrieb des COMPASS-Systems ist eine Anpassung des Kernels nicht nötig, da alle zum Betrieb essentiellen Module bereits in der vorinstallierten Grundkonfiguration vorhanden sind. Die Installation der benötigten Software wird auf einem x86-Rechner unter Linux vorgenommen. Das Kompilieren selbst geschriebener Software erfolgt über den bei Linux mitgelieferten gcc-Kompiler. Um eine Verbindung mit dem installierten Embedded-Linux-System herzustellen, kann entweder die Netzwerkschnittstelle oder eine RS232-Verbindung verwendet werden. Der serielle Anschluss erfordert ein Null-Modemkabel, für die Netzwerkverbindung ist ein Patch-Kabel erforderlich. Der Zugriff auf die Konsole ermöglicht das Laden neuer Programme sowie das Ausführen von Befehlen. Wenn das System im vorhandenen Netzwerk eingesetzt wird, sollte die erste Verbindung mit dem System über die serielle Schnittstelle erfolgen, da eine Anpassung der Netzwerkparameter erforderlich ist. Weitere Details zum installierten System, der entworfenen User-Space-Programme und der Anpassung diverser Prozessor-Register-Einstellungen sind im Folgenden beschrieben.

6.1.1.1 Adressbereiche

Das Hypercontrol MICRO-9 hat einen 32-Bit-Adressraum, der für den Kernel und alle angeschlossenen Geräte zu Verfügung steht. Insgesamt wären damit also 4 Gigabyte Speicher adressierbar. Da jedoch nicht alle Geräte über die volle Anzahl von 32 Adressleitungen angeschlossen sind und zur Selektion eigene Chip-Select-Signale bekommen, ist eine hierarchische Adressaufteilung notwendig. Die höchsten vier Adressbits werden dabei zur Generierung der Chip-Select-Signale benutzt. Die physikalischen Adressen der Komponenten des MICRO-9 sind in der folgenden Tabelle kurz zusammengestellt. Diese und weitere Daten zur Adressaufteilung finden sich im MICRO-9-Handbuch [Cont05c].

Gerät	Startadresse	Endadresse
SDRAM	0x00000000	0x01FFFFFF
SDRAM	0x04000000	0x05FFFFFF
FLASH	0x10000000	0x11FFFFFF
CAN 1	0x20000000	0x20FFFFFF
CAN 2	0x21000000	0x21FFFFFF
NV-SRAM	0x30000000	0x30100000
EBI	0x60000000	0x7FFFFFF

Tabelle 6.1: Physikalische Adressbereiche des HyperControl MICRO-9

6.1.1.2 Bootloader und Flash-EEPROM

Als Bootloader kommt auf dem MICRO-9 das Tool RedBoot zum Einsatz. RedBoot, die "Red Hat Embedded Debug and Bootstrap Firmware" der Firma Red Hat, Inc. ist eine vollständige Bootstrap-Umgebung für Embedded Systems, welches kostenlos und inklusive Quellcode verfügbar ist. RedBoot basiert auf dem Hardware-Abstraction-Layer des Echtzeitbetriebssystems eCos (embedded Configurable operating system) und unterliegt daher auch der eCos License [RedH02], einer etwas abgeschwächten Form der GNU General Public License [Stal91]. Redboot ermöglicht den Download und die Ausführung von Programmen über eine serielle Verbindung oder Ethernet. Die Netzwerkadresse kann dabei auch über BOOTP oder DHCP bezogen werden. Im Falle des MICRO-9 ist die RedBoot-Kommandozeile nur über die zweite serielle Schnittstelle erreichbar. Von der Kommandozeile aus können jedoch mit verschiedenen Befehlen Dateien von TFTP-Servern geladen und im RAM oder ROM abgelegt werden. Sollte kein TFTP-Server verfügbar sein, kann auch über die Terminalverbindung ein Dateitransfer stattfinden, wobei die Protokolle X- oder Y-Modem zum Einsatz kommen. Auch wenn RedBoot in erster Linie als Bootloader für eCos entwickelt wurde, lassen sich damit beliebige Programme oder Betriebssysteme ausführen. Bei Embedded Systems wird inzwischen zum Beispiel immer häufiger – wie auch im Falle des MICRO-9 – das Betriebssystem Linux eingesetzt. Der Linux-Kernel kann auch von RedBoot ausgeführt werden, da Linux von dem in der x86-Welt üblichen BIOS (Basic Input Output System) völlig unabhängig ist. Um den Startvorgang zu automatisieren, lassen sich Bootskripts anlegen, welche nacheinander mehrere RedBoot-Kommandos ausführen. RedBoot teilt den zur Verfügung stehenden Flash-Speicher in mehrere Bereiche auf. Zum einen muss der Bootloader selbst, dessen Konfigurationseinträge und die Partitionstabelle Platz finden, zum anderen ist es möglich, Bereiche für Applikationen oder Dateisysteme zu reservieren. Für das MICRO-9 ist die Applikation der Linux-Kernel und als Dateisysteme sind zwei Images im Format JFFS2 (Journalling Flash File System 2) im ROM abgelegt. Unter Linux ist das EPROM als so genanntes Memory Technology Device (MTD) sichtbar, wobei je Partition ein eigener Device-Node existiert. Die Aufteilung des Flash-EEPROMs ist dabei folgendermaßen organisiert:

- Die Partition /dev/mtdblock0 ist der RedBoot Bootloader selbst
- /dev/mtdblock1 ist der Linux-Kernel
- /dev/mtdblock2 ist das Root-Filesystem
- /dev/mtdblock3 wird von Contec als ”Save-Filesystem” bezeichnet, effektiv wird dieser Partition einfach der restliche freie ROM-Speicher zugeordnet. Gemountet ist dieses Dateisystem unter /mntcontec/part3; Hier können alle für das COMPASS-Projekt nötige Dateien abgelegt werden. Zur Verkürzung der Pfade sind Verzeichnisse dieser Partition über symbolische Links an verschiedenen Stellen ins Root-Filesystem verlinkt.
- /dev/mtdblock4 ist die Partitionstabelle von RedBoot

- /dev/mtdblock5 ist ein kleiner Speicherbereich, in dem RedBoot seine Konfigurationsdaten wie z.B. vergebene IP-Adresse, das Bootskript und Ähnliches ablegt.

Name	ROM-Adresse	RAM-Adresse	Länge	Entrypoint
RedBoot	0x10000000	0x10000000	0x00040000	0x00000000
Kernel	0x10040000	0x00218000	0x00200000	0x00218000
rootfs	0x10240000	0x10240000	0x009C0000	0x00000000
savefs	0x10C00000	0x10C00000	0x013E0000	0x00000000
FIS Directory	0x11FE0000	0x11FE0000	0x0001F000	0x00000000
RedBoot Config	0x11FFF000	0x11FFF000	0x00001000	0x00000000

Tabelle 6.2: Aufteilung des Flash-EEPROMs auf dem MICRO-9

6.1.1.3 Kernel und Gerätetreiber

Die auf dem MICRO-9 installierte Linux-Distribution besteht aus einer von Contec angepassten Version der EP93xx-Variante des Linux-Kernels und einem mit dem Build-System ptxDist der Firma Pengutronix erstellten Root-Dateisystem. Der mit dem für diese Arbeit verwendeten MICRO-9 gelieferte Kernel ist ein monolithischer Kernel der Version 2.6.12, der bereits mit Modul-Support kompiliert ist. Somit können weitere Kernel-Module während des Betriebs geladen oder auch wieder entladen werden.

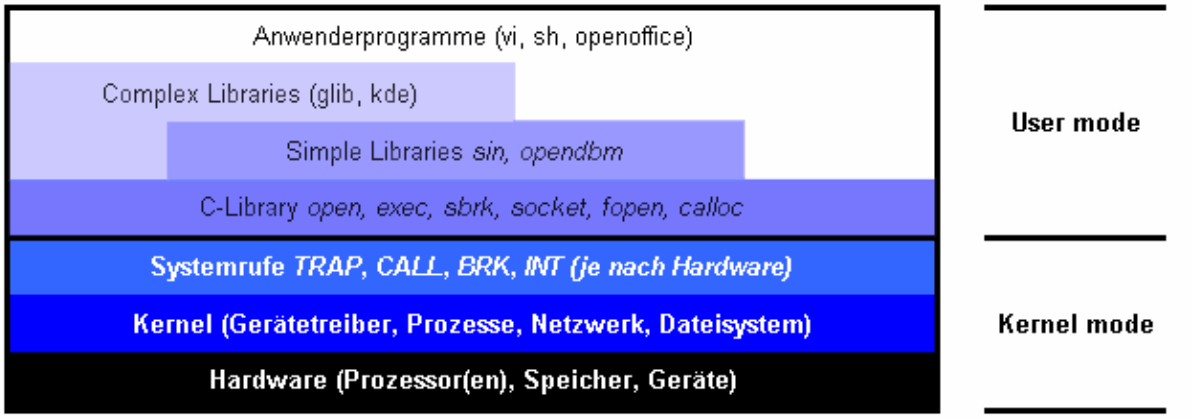

Abbildung 6.1: Abstraktionsschichten des Linux-Kernels (Grafik: Wikipedia)

Für das Compass-Projekt muss ein Kernel-Treiber erstellt werden, da direkt über den AMBA-Bus mit externen FPGAs kommuniziert werden soll. Ein Kernel-Modul ist zwar für den Zugriff auf externe Geräte per Memory-mapped-IO nicht notwendigerweise erforderlich, jedoch im Vergleich zu einem reinen Userspace-Programm die elegantere und performantere Variante. Auch da eventuell zu einem späteren Zeitpunkt Unterstützung für externe Interrupts ergänzt werden soll oder da eventuell für manche Zugriffe Echtzeit-Verhalten gewünscht ist, wird das Programmieren eines Kernel-Moduls zur Methode der ersten Wahl. Ein Kernel-Treiber muss zwingend das Makro module_init() implementieren, welches bei Übersetzung des Codes als Modul zur Initialisierungsfunktion init_module() expandiert, bei Übersetzung als Built-in-Treiber so expandiert, dass sie beim Booten des Kernels aufgerufen wird. Weiterhin muss das Makro module_exit() implementiert werden, welches beim Entladen des Kernel-Moduls mit der Funktion cleanup_module() bzw. beim Herunterfahren des Kernels De-Initialisierungen vornehmen kann. Um auf die an das MICRO-9 angeschlossenen FPGAs auch von der Userspace-Applikation aus zugreifen zu können, muss sich der Kernel-Treiber

als Gerät am System anmelden und über einen so genannten Device-Node ins Dateisystem eingebunden werden können. Einen Überblick über die hierfür nötigen Funktionen bieten [QuKu04] und [Rubi98].

6.1.1.4 Virtuelles Speicher-Management

Der virtuelle Adressraum oder auch virtuelle Speicher bezeichnet den Adressraum, der einzelnen Prozessen vom Betriebssystem zur Verfügung gestellt wird. Nur die Betriebssysteme, die eine virtuelle Speicherverwaltung verwenden, können einen virtuellen Adressraum generieren und dadurch auch nicht zusammenhängende Speicherbereiche in einen aus Sicht eines Prozesses logisch zusammenhängenden Speicherbereich abbilden. So stellt beispielsweise das Betriebssystem Linux bis zu 4 Gigabyte für Programme und Daten bereit, auch wenn deutlich weniger physikalischer Arbeitsspeicher – im Falle des MICRO-9 sogar lediglich 64 Megabyte – zur Verfügung steht. Die Umsetzung der verwendeten virtuellen Adressen auf die physikalische Adresse wird durch die Memory Management Unit erledigt, einem der wichtigsten Funktionsblöcke des Linux Kernels. Die virtuelle Speicherverwaltung ermöglicht weiterhin die Implementierung von Speicherschutzmechanismen, weshalb unter Linux selbst Kernel-Treiber nicht direkt auf die physikalischen Adressen eines per Memory-mapped-I/O angeschlossenen Geräts zugreifen dürfen. Bevor mit entsprechenden Lese- und Schreibzugriffen ein Speicherbereich genutzt werden kann, muss er in den virtuellen Adressraum gemappt werden. Dieses Speicher-Mapping erfolgt mit der Funktion ioremap (phys. address, size), die einen Zeiger auf den zugewiesenen virtuellen Adressraum zurückliefert. Auch bei Zugriffen auf Register von Peripherie-Geräten ist dieses Address-Mapping nötig.

6.1.2 Plattform-Konfigurations-Tool

Die Firma Contec liefert mit dem Mikrocontroller-Board Hypercontrol MICRO-9 alle Entwicklungswerkzeuge, mit denen sowohl der Kernel als auch das zugehörige Root-Dateisystem auf einem PC neu erstellt werden können. Hierzu kommt als Cross-Compiler der GNU-C-Compiler (GCC) zum Einsatz. Ein Cross-Compiler ist ein Compiler, der auf einer anderen Architektur ausgeführt wird als die, für welche die mit ihm erzeugten ausführbaren Binär-Dateien bestimmt sind. Diese Situation ist hier der Fall, da die Entwicklung auf einer Intel-x86-Architektur für die ARM-Architektur des MICRO-9 erfolgt. Der Vorteil dieses Compilers ist die Kompatibilität mit fast allen frei verfügbaren Open-Source-Programmen. In der Regel muss hier nach Entpacken der entsprechenden Quellcodes nur dafür gesorgt werden, dass beim Übersetzen der Programme der ARM-Crosscompiler anstelle der Compiler-Installation des Entwicklungssystems benutzt wird. Dies wurde im Rahmen dieser Arbeit mehrfach nötig, um weitere auf dem MICRO-9 nicht vorinstallierte Tools selbst zu übersetzen und nachzuinstallieren.

Als Quellcode-Editor empfiehlt sich für dieses Projekt das Programm Eclipse, da für diese ursprünglich nur als Java-IDE konzipierte Anwendung inzwischen auch Erweiterungen für die Sprachen C und C++ verfügbar sind. Somit kann für die Programmierung aller Teil-Projekte die gleiche Programmierumgebung benutzt werden. Auch der Aufruf von Makefiles zur automatisierten Programmerstellung ist dabei möglich, womit dann zum Beispiel auch das Kopieren der erstellten Binär-Dateien auf die Compass-Plattform automatisiert werden kann. Für die Entwicklung des Kernel-Treibers ist eine tiefgehende Kenntnis der Struktur des Kernel-Quellcodes und ein gutes Tool zum Durchsuchen der mittlerweile über 400 Megabyte Quellcode nötig. Hier empfiehlt sich das Tool LXR, das einen kompletten Quellcode-Baum analysieren und indizieren kann. Man erhält eine vollständig verlinkte Hypertext-Ansicht der Dateien, wobei durch Anklicken eines Bezeichners direkt zu sämtlichen anderen Referenzen desselben gesprungen werden kann. Auch eine durch gute Indizes sehr schnelle Freitext-Suche ist in das Toolset integriert.

6.1.3 Linux-Gerätetreiber

Die Ankopplung des Board-Control-FPGAs sowie der peripheren FPGA-Karten erfolgt über das externe Bus-Interface des Static-Memory-Controllers. Ein eigens dafür entwickelter

Linux-Treiber wickelt den Zugriff auf diese Geräte ab. Im Wesentlichen bedeutet dies das Handeln der Schreib- und Lesefunktionen für alle per Memory-Mapped-I/O angebundenen Busteilnehmer und die Bereitstellung einer Schnittstelle für Anwendungsprogramme mit Hilfe von Device-Nodes. Die Verwendung eines Kernel-Treibers ist dazu nicht notwendigerweise erforderlich, da die physikalischen Adressen auch direkt in den virtuellen Speicherbereich eines Userspace-Programms gemappt werden könnten. Der Vorteil eines solchen liegt jedoch in der besseren Kontrolle des Timings der entsprechenden Funktionen. Auch die Verwendung von Interrupts ist unter Linux nur im Kernel-Modus möglich. Diese werden zwar zur Realisierung der Grundfunktionalität der COMPASS-Plattform nicht benötigt, für einen späteren Ausbau des Systems sollte dieser Weg jedoch nicht bereits konzeptionell ausgeschlossen werden. Eine weitere Aufgabe des Kernel-Treibers ist die Initialisierung von Registern des Static-Memory-Controllers, der für jeden Adressbereich ein separates Bank-Configuration-Register besitzt. Darüber lassen sich verschiedene Parameter des Buszugriffs einstellen, wie z.B. die Datenbreite des im jeweiligen Adressbereich angeschlossenen Gerätes oder auch die Anzahl der Wait-States, über welche die Geschwindigkeit des Buszugriffs für langsame Peripherie-Geräte angepasst werden kann. Im Folgenden wird das Kernel-Modul als ”CompassDriver” bezeichnet.

Der hier vorgestellte Gerätetreiber registriert ein Device für eine bestimmte Major-Gerätenummer (hier: 246). Somit können verschiedene Device-Nodes angelegt werden, die den Zugriff auf die angeschlossenen FPGAs ermöglichen. Die Anzahl der FPGAs wird durch die prototypische Realisierung vorgegeben, auf der ein Plattform-Control-FPGA und vier Steckplätze für Interface-Karten vorhanden sind. Das Anlegen geschieht mit dem Kommando mknod auf der MICRO-9 Plattform:

```
mknod /dev/compass0 c 246 0      (Plattform-Control-FPGA)
mknod /dev/compass1 c 246 1      (FPGA, Steckplatz 1)
mknod /dev/compass2 c 246 2      (FPGA, Steckplatz 2)
mknod /dev/compass3 c 246 3      (FPGA, Steckplatz 3)
mknod /dev/compass4 c 246 4      (FPGA, Steckplatz 4)
```

Die fünf Compass-Device-Nodes können nun von einem Userspace-Programm geöffnet werden, um mit den peripheren FPGAs zu kommunizieren.

6.1.4 CompassLink-Applikation

Die Software CompassLink ist eine Userspace-Applikation auf dem Micro-9-Modul und wird dort nach dem Starten des Boards im Hintergrund ausgeführt, fungiert also als eine Art Netzwerkserver. Ihre Aufgabe ist es, die Verknüpfung zwischen dem Compass-Kernel-Treiber und der Ethernet-Netzwerkschnittstelle herzustellen, um Client-PCs die Möglichkeit zu bieten, direkt über eine TCP/IP Verbindung auf den Systembus des MICRO-9-Moduls zuzugreifen. Außerdem sollen weitere Dienste auf der Plattform über diese Schnittstelle ausgeführt werden können.

Zu Test- und Debug-Zwecken kann dieses Programm auch lokal bedient werden, um automatisierte Testläufe zu starten. Damit kann z.B. die Funktionalität von Lese- und Schreibzugriffen der Busschnittstelle getestet werden. Auch ein Langzeittest mit mehreren Millionen Buszugriffen wurde realisiert, um sicherzustellen, dass die in VHDL implementierte Busankopplung der FPGAs einwandfrei arbeitet und den normalen Betrieb des MICRO-9 in keiner Weise beeinträchtigt. Mit Hilfe dieser Funktion konnte ein Benchmarking hinsichtlich der maximalen Geschwindigkeit der Speicherzugriffe auf die FPGA-Register durchgeführt werden, die beim Schreiben im Burst-Mode eine max. Übertragungsrate von 50Mbyte/s ergab.

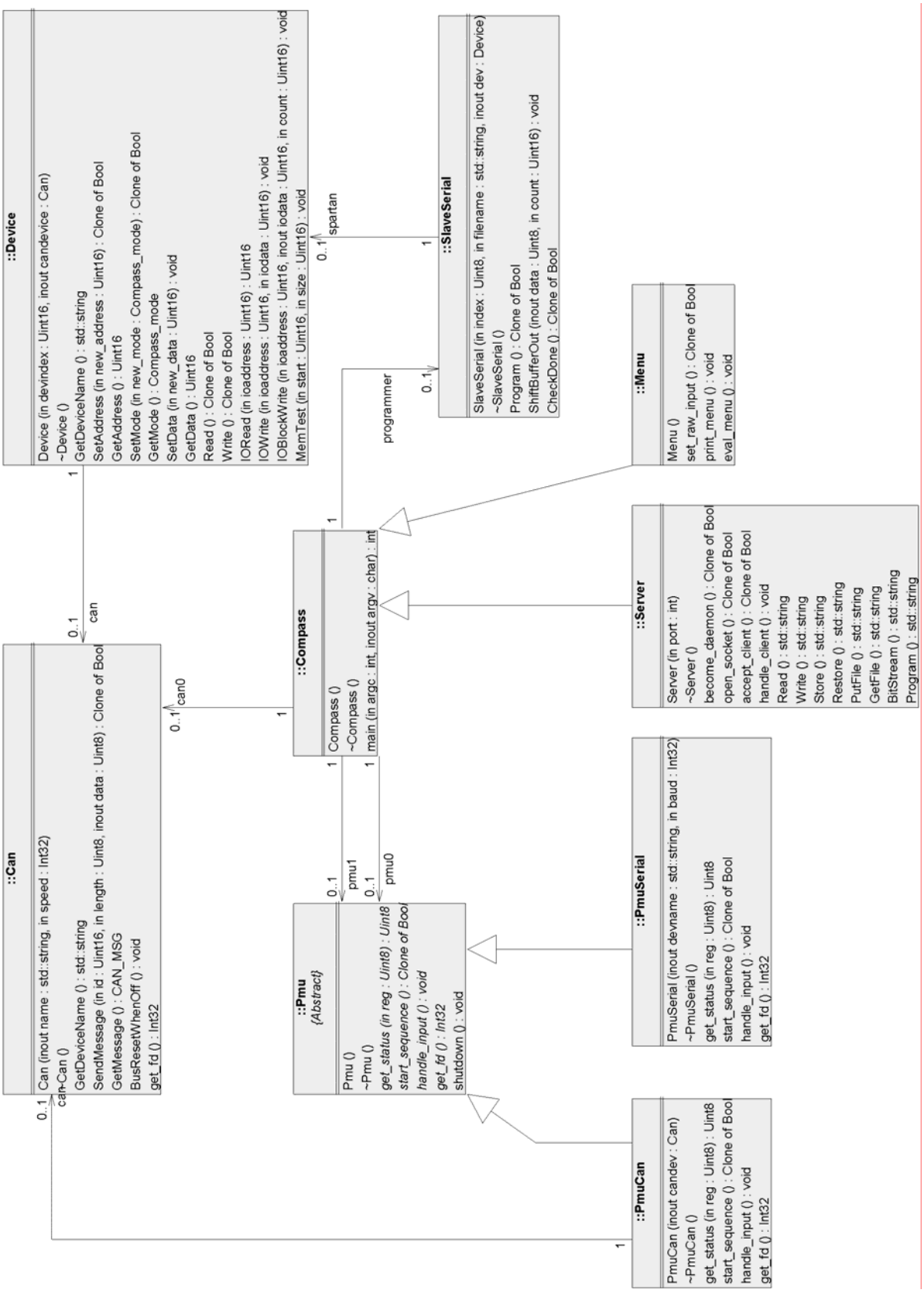

Abbildung 6.2: Klassendiagramm der CompassLink-Applikation

Alle weiteren in die CompassLink-Applikation integrierten Funktionen werden hier im Anschluss aufgeführt und deren Aufgaben näher erläutert. Weitere Details zu den einzelnen Funktionen und ihrer Umsetzung in Form der Programmierung sind unter [Bahl06] nachzulesen.

6.1.4.1 Netzwerk-Server und Datenübertragungsprotokoll

Wie bereits erwähnt, ist die Netzwerkfähigkeit eine der Haupteigenschaften der COMPASS-Link-Software, die es ermöglicht, eine grafische Benutzeroberfläche anzubinden. Hierfür wurde ein einfaches Datenübertragungsprotokoll implementiert, das es erlaubt, Lese- und Schreibzugriffe auf den Systembus durchzuführen. Ebenfalls ist es damit möglich, die Bitstreams für die FPGAs der Erweiterungskarten zu übertragen. Die Dateigröße der Bitstreams hängt dabei sehr von dem verwendeten FPGA ab, muss also variabel sein, um die Flexibilität der Plattform nicht einzuschränken.

In der Funktion handle_client() werden die vom Client empfangenen Daten gelesen und die den Befehlen entsprechenden Aktionen ausgeführt. Zur Synchronisation mit dem Client wird zuerst gewartet, bis ein NULL-Byte empfangen wird, was den Beginn eines neuen Befehls signalisiert. Sobald die Verbindung zum Client geschlossen wurde, wird wieder in die oben beschriebene Endlosschleife des Konstruktors zurückgekehrt. Nachdem die Größe des ankommenden Datenblocks bekannt ist, wird so lange die Funktion recv() aufgerufen, bis die volle Anzahl an Datenbytes angekommen ist. Dieses Vorgehen ist nötig, da verschiedenste Puffer im Client-Programm und in den Betriebssystemen dafür sorgen, dass der Datenfluss nicht kontinuierlich ist und damit auch nicht alles innerhalb eines recv()-Aufruf empfangen werden kann.

Nun muss der zu Beginn empfangene Befehl ausgewertet und die zugehörige Behandlungsroutine aufgerufen werden. Jede dieser Behandlungsroutinen liefert eine Zeichenkette zurück, welche als Antwort an den Client zurückgeschickt werden kann. Auch für diese Antwort des Servers wird zuerst die Länge der gesamten Zeichenkette übertragen, so dass der Client weiß, auf wie viele Byte noch gewartet werden muss. Bei der Übertragung der Länge als 16Bit-Worte ist darauf zu achten, dass hier die Network-Byte-Order verwendet wird, also das MSB zuerst gesendet werden muss. Nur so wird das Java-Programm auf der Gegenseite die Werte richtig interpretieren. Nach der Übertragung der FPGA-Bitstreams vom Java-Client zum Netzwerkserver wird der Bitstream als Datei im temporären Verzeichnis /tmp des MICRO-9 abgespeichert. Somit ist sowohl eine einfache Weiterverarbeitung gewährleistet, als auch die unnötige Lebensdauer-Verkürzung des Flash-ROMs vermieden, da das /tmp-Dateisystem im SDRAM des MICRO-9 angelegt ist. Eine Übersicht der Befehle des Netzwerkprotokolls ist in Tabelle 6.3 zu sehen.

Aktion	**Befehl**
Beenden	QUIT
Herunterfahren	HALT
Daten lesen	READ<dataSize>{device_id}<address><count>
Daten schreiben	WRIT<dataSize>{device_id}<address><count>[data]
Upload eines Bitstreams	BSTR<dataSize>{fpga_id}{new_bstr}<count>[data]
Start der Programmierung	PROG<dataSize>{fpga_id}

Tabelle 6.3: Netzwerkbefehle zwischen Client und COMPASS-Plattform

Ein Bezeichner in spitzen Klammern symbolisiert die Datenbreite von zwei Byte, in geschweiften Klammern die Datenbreite von einem Byte. Eckige Klammern bezeichnen ein Feld der Länge, die zuvor im Feld count angegeben wurde.

Der Befehl QUIT beendet die aktive TCP-Verbindung zwischen Netzwerk-Server und Java-Client. Mit HALT kann dagegen der Netzwerk-Server komplett beendet werden.

6.1.4.2 FPGA-Programmierung über das Slave-Serial-Interface

Die Programmierung der FPGAs auf den Erweiterungskarten erfolgt über das Protokoll Slave-Serial. Hierzu sind alle benötigten Leitungen an den Board-Control-FPGA angeschlos-

sen. Zur Implementierung kann auf den in [NgPe02] beschriebenen und auf der Xilinx Webseite verfügbaren Beispiel-Quellcode zurückgegriffen werden. Es sind lediglich kleinere Anpassungen nötig, um diesen in die Compass-Applikation zu integrieren. Da die Busschnittstelle zu diesem Zeitpunkt bereits verfügbar sein sollte, können direkt die in [NgPe02] beschriebenen drei Register je Erweiterungs-Slot angelegt und korrekt angeschlossen werden. Der C-Quellcode muss dann noch in eine C++ Klasse umgeschrieben werden, die dann der CompassLink-Software hinzugefügt wird. Einzelheiten der Implementierung sind in [Bahl06] dargelegt.

6.1.4.3 Zugriff auf die EEPROMs

Um den Hardware-Status der Plattform zu detektieren, müssen die EEPROMs programmiert und ausgelesen werden können. Hierfür ist bereits ein einfaches SPI-Protokoll im Board-Control-FPGA implementiert, das den kompletten Inhalt der EEPROMs in das Dual-Port-RAM des Board-Control-FPGAs spiegelt und umgekehrt. Somit muss von der Software nur noch der komplette RAM-Bereich ausgelesen und an die Client-Applikation (Compass-Control) geschickt werden. Der Kopiervorgang aus den bzw. in die EEPROMs wird über spezielle Befehlsworte der SPI-Kontrollregister gestartet.

6.1.4.4 Ansteuerung des programmierbaren Taktbausteins

Auch die Ansteuerung des programmierbaren Taktbausteins PCK12429 von Philips geschieht vornehmlich über die im Board-Control-FPGA integrierte State-Machine. Über ein serielles Protokoll werden die Konfigurationsdaten für die einzustellende Frequenz übertragen. Hierzu kann im Wesentlichen der für die Programmierung der EEPROMs geschriebene Software-Teil wieder verwendet werden. Lediglich kleine Anpassungen und das Entfernen nicht benötigter Teile sind hier nötig. Die von Seiten der Java-GUI des Client-PCs über ein Eingabefeld eingestellte Frequenz wird in ein FPGA-Register geschrieben und danach die Frequenzeinstellung gestartet.

6.1.4.5 Ansteuerung der Power-Management-Unit

Die Kommunikation mit der Power-Management-Unit (PMU) kann über zwei verschiedene Schnittstellen erfolgen. Zum einen gibt es eine serielle Verbindung zwischen der externen PMU und dem COMPASS-Basis-Board. Dies ist die Standard-Schnittstelle, über welche die Kommunikation normalerweise abläuft. Außerdem wurde auch eine alternative Verbindungsmöglichkeit über die CAN-Schnittstelle vorbereitet.

Für beide Kommunikationswege ist die gleiche Funktionalität implementiert, über die Statusinformationen von der PMU angefragt werden können. Dies sind insgesamt drei verschiedene Register, die über Bitmasken aufgetretene Ausnahmezustände signalisieren, zum Beispiel Überspannungen oder Unterspannungen bestimmter Versorgungsstränge oder ein Reset des MICRO-9-Watchdogs kann von der PMU detektiert werden. Im letzteren Fall würde die PMU alle durch den Netzwerkserver zugeschalteten Spannungen wieder abschalten, da man davon ausgehen muss, dass der Netzwerkserver die aktuelle Hardwarekonfiguration nicht mehr unter Kontrolle hat. Weiterhin ist auf der PMU ein Taster vorhanden, über welchen das Herunterfahren der Plattform eingeleitet werden kann. Wird von der PMU also ein entsprechender Tastendruck gemeldet, muss das Micro-9 nacheinander das Abschalten aller Versorgungsstränge anfordern und darauf das Linux herunterfahren. Das letzte Shutdown-Skript, das vom MICRO-9 ausgeführt wird, fordert schließlich das Abschalten aller MICRO-9-Versorgungsspannung an und die Plattform ist vollkommen spannungslos.

```
void Pmu::shutdown()
{ system("/sbin/shutdown -h now 'PMU shutdown request'"); }
```

Zum Hochfahren wird der PMU mit einer Nachricht signalisiert, alle Versorgungsspannungen in einer vordefinierten Reihenfolge einzuschalten. Weitere PMU-Befehle erlauben das Ein- und Ausschalten einzelner Spannungen. Auch das Messen der Spannungen über die A/D-Wandler-Eingänge des Mikrocontrollers auf der PMU ist möglich. Diese Funktionen wurden bisher jedoch nur zum Debugging benötigt und sind daher auch nicht über die graphische Benutzerschnittstelle und den Netzwerkserver steuerbar.

6.1.5 Starten und Herunterfahren des Netzwerkservers

Damit der Netzwerkserver automatisch beim Booten der Plattform startet, wurde ein Startscript angelegt. Startscripts liegen bei Linux-Systemen üblicherweise im Verzeichnis /etc/init.d, wonach das Compass-Startscript als /etc/init.d/compass abgelegt ist. Das Script lädt zuerst den im vorigen Kapitel beschriebenen Gerätetreiber, so dass die COMPASS-Geräte verfügbar sind. Danach wird mit Hilfe ioctl()-Kommandos die Anzahl der Wait-States für den Buszugriff eingestellt. Nach dem Einstellen der Baudrate der seriellen Verbindung zur PMU muss noch deren Fehlerspeicher gelöscht werden und der Netzwerkserver kann mit dem oben beschriebenen Parameter ”-d” als Hintergrundprozess geladen werden.

```
insmod /compass/micro9_compass.ko
/compass/ioctl /dev/compass0 3 6
/compass/ioctl /dev/compass1 3 31
stty -F /dev/ttyAM2 19200
echo -n p > /dev/ttyAM2
/compass/CompassLink -d
```

Da der Netzwerkserver erst ganz am Ende nach der vollständigen Abarbeitung aller anderen Startscripts ausgeführt werden soll, wurde ihm die Priorität 90 zugewiesen, wodurch das Compass-Script erst nach allen anderen angelegten Startscripts ausgeführt wird.

6.2 BistreamComposer - Modulaustausch per „Drop-Down-Menü“

Das Java-Programm BitstreamComposer stellt eine grafische Benutzeroberfläche (siehe Abbildung 6.3) zur Konfiguration eines Xilinx-Vitrex-II-1000-FPGAs zur Verfügung. Mit seiner Hilfe werden bestimme Funktionen aus der Modul-Bibliothek geladen und zu einem funktionierenden Gesamtsystem zusammengesetzt. Die drei Slots A, B und C sind von ihrer Anzahl durch den FPGA-Typ und die damit verbundene Aufteilung in drei freie (Slot A, B und C) und einen statischen Bereich vorgegeben. Bei Verwendung anderer FPGAs, muss diese Anzahl automatisch angepasst werden. Dies wurde in dieser ersten Version nicht umgesetzt, da nur der oben genannte FPGA zur Verfügung stand. Eine solche Anpassung kann aber nachträglich implementiert werden. Für jeden Slot wird aus einer Drop-Down-Liste (Abbildung 6.4) ein Modul ausgewählt. Die Beschreibung des entsprechenden Moduls wird in dem darunter liegenden Textfenster angezeigt.

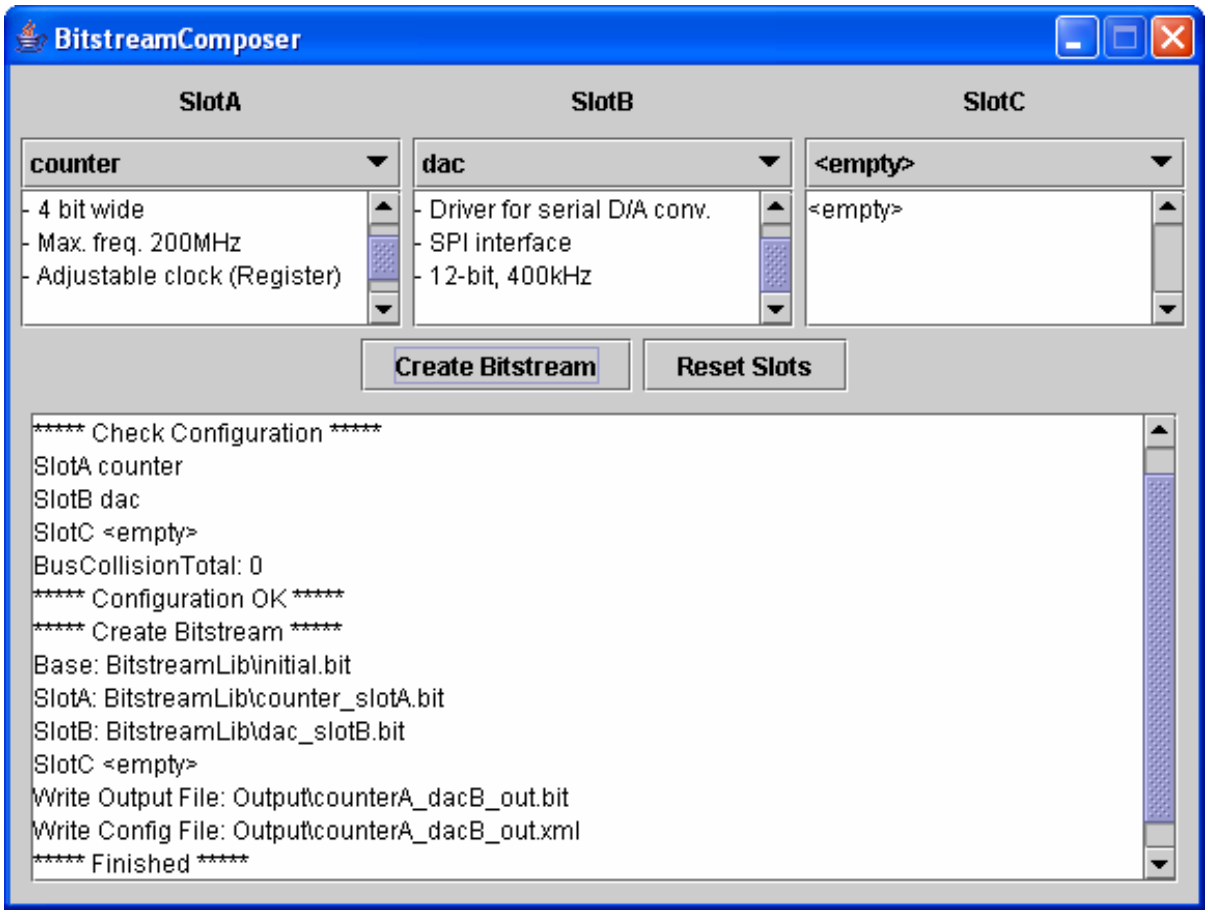

Abbildung 6.3: Screenshot des BitstreamComposer

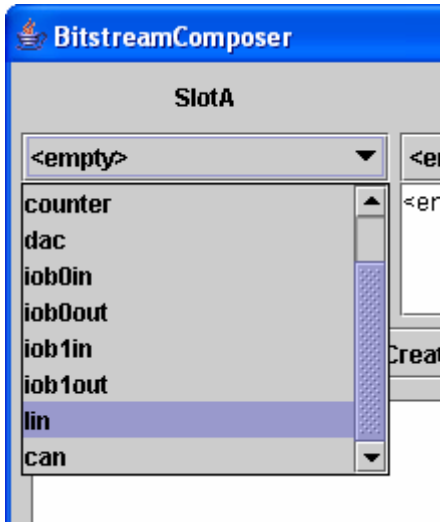

Abbildung 6.4: Modulauswahl im Drop-Down-Menü

Durch Anklicken des Buttons „Create Bitstream“, wird das Zusammensetzen des Bitstreams gestartet. Dabei überprüft die Applikation zuerst die erstellte Konfiguration auf mögliche Fehler und Ressourcen-Konflikte zwischen verschiedenen Modulen, wozu die XML-Konfigurationsdateien der einzelnen funktionalen Module herangezogen werden. Ist die Konfiguration fehlerfrei, wird ein Bitstream und eine XML-Konfigurationsdatei, die das Ge-

samtsystem beschreibt, erzeugt. Im Konsolenfenster werden Informationen zu den einzelnen Schritten angezeigt. Der Button Reset Slots setzt die Anwendung wieder in den Startzustand zurück, in dem keiner der Slots ein funktionales Modul enthält. Der Standardablauf des Zusammensetzens von Funktionen und die Erzeugung der Konfigurationsdaten ist in Abbildung 6.5 dargestellt.

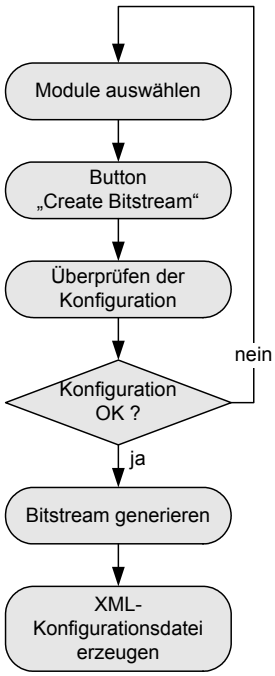

Abbildung 6.5: Programmablauf des BitstreamComposers

6.2.1 Funktion des Programms

Nach dem Start der Applikation erscheint die grafische Benutzeroberfläche aus Abbildung 6.3. Im oberen Teil befinden sich drei Drop-Down-Menüs, denen ebenfalls drei unterhalb angeordnete Textfelder direkt zugeordnet sind. Darunter folgen zwei Buttons zur Steuerung und letztlich eine Ausgabekonsole. Die drei Auswahllisten erlauben es, für jeden Slot ein funktionales Modul zu selektieren. Jedes Mal, wenn für einen Slot ein anderes funktionales Modul ausgewählt wird, werden die aktuell ausgewählten Module in einer Variablen gespeichert und die Funktion updateSlotDescription() aufgerufen, mit der die Beschreibungen der funktionalen Module aktualisiert und im dazugehörigen Textfenster angezeigt werden.

Durch Aktivieren des Buttons „Create Bitstream“ wird die entsprechende Funktion createBitstream gestartet. Nach dem Aufruf der Funktion updateModules() werden dem Benutzer die selektierten funktionalen Module im Konsolenfenster angezeigt und der entsprechende Name für die Gesamt-Konfiguration erzeugt. Die Funktion compareConfigFiles() überprüft die XML-Konfigurationsdateien der ausgewählten funktionalen Module auf mögliche Ressourcenkonflikte. Ist die Konfiguration fehlerfrei, so wird die Funktion copySlots() zum Zusammensetzen des Bitstreams aufgerufen. Abschließend wird mit Hilfe der Funktion writeFpgaConfig() eine XML-Konfigurationsdatei zur Beschreibung des Gesamtsystems generiert.

6.2.1.1 Die Funktion updateModules()

Die Funktion updateModules() verarbeitet die Auswahl der funktionalen Module und definiert, welche Bitstreams zum Zusammensetzen des Gesamtsystems geladen werden müssen. Dadurch ist das Programm in der Lage, auch funktionale Module in das Gesamtsystem

zu integrieren, die mehr als nur einen der drei Slots auf dem FPGA belegen (z.B. CAN-Modul). Überschreitet dabei die Anzahl der benötigten Slots eines Moduls die noch zur Verfügung stehende Anzahl an leeren Slots, so wird eine Fehlermeldung ausgegeben und der Slot auf den Eintrag <empty > zurückgesetzt.

6.2.1.2 Die Funktion compareConfigFiles()

Die Funktion compareConfigFiles() überprüft die XML-Dateien der ausgewählten funktionalen Module auf Ressourcenkonflikte der im FPGA-Design verwendeten internen Bussysteme. Als Argument wird der Funktion ein Array von Dokument-Objekten übergeben, die aus den XML-Dateien der verwendeten Module erstellt werden. Eine Integer-Variable BusCollisionTotal zählt die Anzahl der Busleitungen, auf denen Ressourcen-Konflikte entdeckt werden und stellt den Rückgabewert der Funktion dar. Ist das Document-Array nicht leer, so wird in die Integer-Variable busCount die Anzahl der in der XML-Datei zu überprüfenden Bussysteme eingelesen

Wird eine Signalleitung von mehr als einem Modul als Ausgang benutzt, so entsteht ein Konflikt auf dieser Busleitung, da beide Module über einen Tri-State-Treiber ein Signal auf den Bus schreiben. Dies wird durch Überprüfen der Variablen output erkannt. In diesem Fall wird eine Statusmeldung in der Konsole ausgegeben und die Variable busCollisionTotal um eins erhöht. Zum Schluss wird die Gesamtzahl der entdeckten Konflikte angezeigt und die Variable busCollisionTotal als Rückgabewert der Funktion definiert.

6.2.1.3 Die Funktion copySlots()

Die Funktion copySlots() führt das Zusammensetzen des Bitstreams durch. Im ersten Schritt wird der Bitstream des Basis-Designs eingelesen. Danach wird für jeden der drei Slots überprüft, ob für den jeweiligen Slot ein funktionales Modul ausgewählt wurde. Ist dies der Fall, wird der dazugehörige Bitstream, der das ausgewählte Modul in dem jeweiligen Slot enthält, eingelesen und verarbeitet.

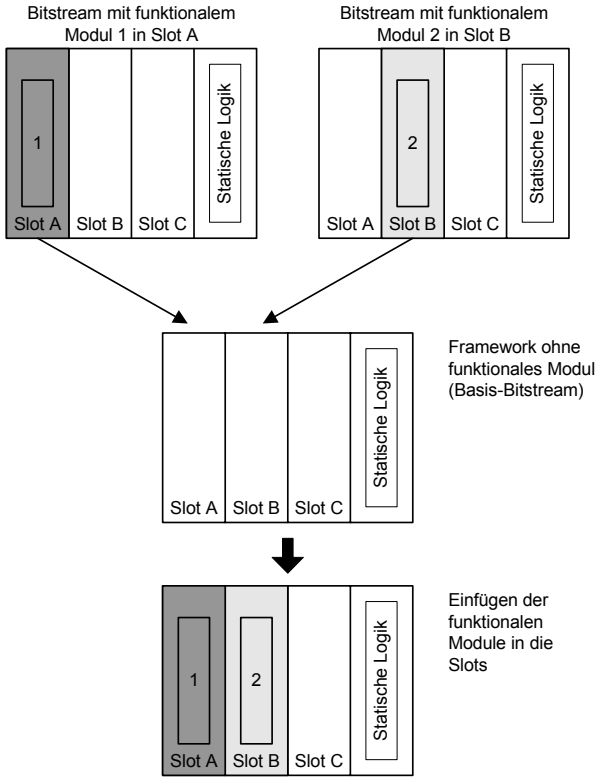

Abbildung 6.6: Funktionsweise der Funktion copySlots

Dazu werden aus dem Bitstream mit dem funktionalen Modul nacheinander in einer Schleife alle Daten aus den entsprechenden Konfigurationsspalten ausgelesen und an der gleichen Position im Basis-Bitstream wieder eingesetzt. Die Konfigurationsdaten des zu einem Slot gehörenden Bereichs im Basis-Bitstream werden so mit den Konfigurationsdaten aus dem Bitstream mit dem funktionalen Modul in diesem Slot überschrieben.

Abbildung 6.6 zeigt die Funktionsweise der Funktion copySlots() anhand einer Beispielkonfiguration. Die dort verwendeten Bitstreams enthalten jeweils ein funktionales Modul in Slot A bzw. Slot B. Diese funktionalen Module werden in den Basis-Bitstream integriert. Als Ergebnis erhält man ein Design mit den beiden funktionalen Modulen in den jeweiligen Slots.

6.2.1.4 Generierung einer XML-Datei

Die Generierung einer XML-Datei zur Beschreibung des Gesamtsystems wird in der Funktion writeFpgaConfig() realisiert. Dazu wird der Funktion ein Array mit allen zu den ausgewählten funktionalen Modulen gehörenden Dokument-Objekten übergeben, die aus den XML-Dateien der Module erzeugt werden. Weiterhin wird der Funktion der String outFileName übergeben, der den Dateinamen der zu generierenden XML-Datei festlegt. Die Struktur der erzeugten XML-Datei stimmt weitestgehend mit denen der funktionalen Module überein. Das Wurzelelement trägt die Bezeichnung bitstream. Als Unterelemente werden hier die Elemente fileName, resources und configuration definiert, wobei das Element fileName als erster Eintrag der XML-Struktur hinzugefügt wird. Danach folgen die Einträge für die Verwendung der benötigten FPGA-Karte und die Unterelemente module für jedes der funktionalen Module, die wiederum Unterelemente name und description aus der jeweiligen XML-Konfigurationsdatei des funktionalen Moduls enthalten. Alle Elemente mit der Bezeichnung card aus den XML-Dateien der funktionalen Module (siehe auch Kapitel 5.5.12) zur Spezifizierung der benötigten Schnittstellenkarten, werden ebenfalls als Unterelemente des Elements resources in die XML-Struktur übernommen.

Als nächstes werden die vorhandenen Bussysteme in die XML-Struktur übernommen. Für jedes der vorhandenen Bussysteme wird der Name und die Breite des Bussystems ermittelt und dem Element bus als Attribut hinzugefügt. Danach wird für jede Signalleitung des gerade überprüften Bussystems ein Element mit dem Namen signal angelegt, welchem als Attribut der aktuelle Index der überprüften Busleitung hinzugefügt wird. Außerdem wird ein Attribut state definiert, das je nach Verwendung der Busleitung den Wert „input“, „output“ oder „unused“ tragen kann.

Die Berechnung der Werte für die Kontrollregister zum Einstellen der I/O-Busleitungen auf Ein- oder Ausgabe schließt den gesamten Vorgang ab. Diese Register werden als Element register in der XML-Struktur angelegt und erhalten als erstes Attribut ihre Adresse, mit der sie in der IO-Switch-Matrix über den Systembus angesprochen werden können. Das zweite Attribut bekommt den berechneten Wert zugewiesen, der für das zusammengestellte Gesamtsystem in das jeweilige Register geschrieben werden muss, damit die richtigen Tri-State-Treiber aktiviert werden.

6.2.2 XML-Konfigurationsdatei des Gesamtsystems

Bei der Generierung eines Bitstreams erzeugt der BitstreamComposer automatisch eine XML-Konfigurationsdatei, die das Gesamtsystem, das sich aus den ausgewählten funktionalen Modulen zusammensetzt, beschreibt. Die dort enthaltenen Informationen umfassen:

- Name des dazugehörigen Bitstreams
- Liste aller im Bitstream enthaltenen funktionalen Module mit Name und Beschreibung
- Beschreibung der FPGA-Karte
- Benötigte Schnittstellenkarten mit Angabe der Platzierung

- Übersicht über die belegten Leitungen der Bussysteme IO-Bus 0, IO-Bus 1 und Peripheral Communication Bus und die Signalrichtungen
- Angabe der Werte für die Register der IO-Switch-Matrix zur automatischen Konfiguration der Signalrichtung der Leitungen der Bussysteme IO-Bus 0, IO-Bus 1 und Peripheral Communication Bus

Eine grafische Repräsentierung der obigen Angaben zu einer Konfiguration ist in der folgenden Abbildung 6.7 als Meta-Modell dargestellt.

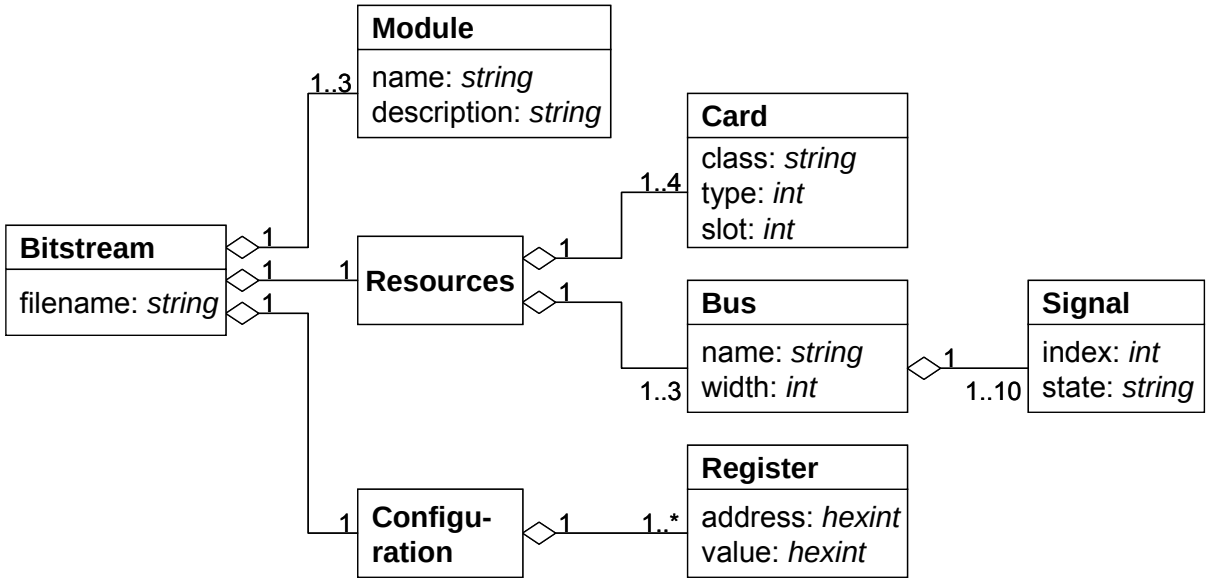

Abbildung 6.7: Meta-Modell der XML-Konfigurationsdatei

Die Umsetzung der grafischen Notation in eine XML-Datei ist hier beispielhaft gegeben. Der folgende Ausschnitt zeigt eine XML-Konfigurationsdatei für eine Konfiguration mit dem Modul „lin“ in Slot A, dem Modul „dac“ in Slot B und dem Modul counter in Slot C:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bitstream>
  <filename>linA_dacB_counterC_out.bit</filename> Zugehöriger Bitstream
  <module> Modul 1
    <name>lin</name> Name
    <description>LIN-Controller</description> Beschreibung
  </module>
  <module> Modul 2
    <name>dac</name> Name
    <description>D/A-Converter</description> Beschreibung
  </module>
  <module> Modul 3
    <name>counter</name> Name
    <description>4-bit Counter</description> Beschreibung
  </module>
<resources> Belegte Ressourcen
```

```
    <card class="fpga" type="1" slot="fpga" /> FPGA-Karte
    <card class="auto" type="1" slot="pcB" /> Automotive-Karte
    <card class="io" type="1" slot="ioB0" /> I/O-Schnittstellenkarte
    <card class="io" type="1" slot="ioB1" />
    <bus name="ioB0" width="10"> Ressourcenbelegung IO-Bus 0
      <signal index="0" state="output" /> Ausgang
      <signal index="1" state="output" /> Ausgang
      <signal index="2" state="output" /> Ausgang
      <signal index="3" state="output" /> Ausgang
      <signal index="4" state="output" /> Ausgang
      <signal index="5" state="output" /> Ausgang
      <signal index="6" state="output" /> Ausgang
      <signal index="7" state="unused" /> nicht benutzt
      <signal index="8" state="unused" />
      <signal index="9" state="unused" />
    </bus>
    <bus name="ioB1" width="10"> Ressourcenbelegung IO-Bus 1
      <signal index="0" state="unused" /> nicht benutzt
      <signal index="1" state="unused" />
      ...
    </bus>
    <bus name="pcB" width="16"> Ressourcenbelegung PC-Bus
      <signal index="0" state="unused" /> nicht benutzt
      <signal index="1" state="unused" />
      ...
      <signal index="12" state="input" /> Eingang
      <signal index="13" state="output" /> Ausgang
      ...
    </bus>
  </resources>
  <configuration> Register IO-Switch-Matrix
    <register address="0x1002" value="0x7f" />
    <register address="0x1004" value="0x0" />
    <register address="0x1006" value="0x0" />
    <register address="0x1008" value="0x0" />
    <register address="0x100A" value="0x2000" />
    <register address="0x100C" value="0x1000" />
    <register address="0x1000" value="0x31" />
  </configuration>
  </bitstream>
```

Die XML-Konfigurationsdatei des Gesamtsystems kann später dazu verwendet werden, den entsprechenden Bitstream auf den FPGA zu laden. Die Register der IO-Switch-Matrix werden dabei automatisch mit den in der XML-Konfigurationsdatei angegebenen Werten geladen, so dass ein manuelles Freischalten der Signalleitungen der Bussysteme IO-Bus 0, IO-Bus 1 und Peripheral Communication Bus nicht mehr notwendig ist. Außerdem erhält man eine automatische Dokumentation der erzeugten Konfiguration, welche in einem Browser angezeigt werden kann.

6.2.3 Verzeichnisstruktur der Applikation

Die Anwendung BitstreamComposer greift auf zwei Ordner mit den Namen BitstreamLib und Output zu, die im gleichen Verzeichnis wie die Anwendung selbst liegen. Im Ordner BitstreamLib werden alle Bitstreams, die zum Zusammensetzen des Gesamtsystems zur Verfügung stehen, und die XML-Konfigurationsdateien, welche die vorhandenen funktionalen Module beschreiben, abgelegt. Der Basis-Bitstream, der nur das Framework ohne ein funktionales Modul enthält, trägt den Namen initial.bit. Die Bitstreams, welche jeweils ein funktionales Modul in einem der drei Slots beinhalten, werden nach der Regel „modulname slotX.bit“ bezeichnet. Der Ordner Output dient als Zielordner, in welchem die von der Anwendung erzeugten Bitstreams und die dazugehörigen XML-Konfigurationsdateien gespeichert werden. Diese XML-Dateien und die Bitstreams werden verwendet, um den FPGA zu programmieren. Anhand der Dateinamen der erzeugten Bitstreams und XML-Dateien des Gesamtsystems kann die gewählte Konfiguration direkt erkannt werden. Die Namen werden nach der Regel „modul1A modul2B modul3C out“ erzeugt und erhalten die Dateiendung „*.bit“ bzw. „*.xml“.

6.2.4 Ergebnisse der Methodik anhand einer Beispielimplementierung

Zur Demonstration der Funktionsweise und der Leistungsfähigkeit der in Kapitel 5 beschriebenen Methodik, wurde eine Beispielimplementierung mit drei verschiedenen Modulen (IP-Cores) auf dem FPGA angestrebt. Als Module kamen ein LIN-Controller (Slot links), ein Treiber für einen externen seriellen A/D-Wandler (Slot in der Mitte) und ein einfacher Zähler (rechter Slot) zum Einsatz.

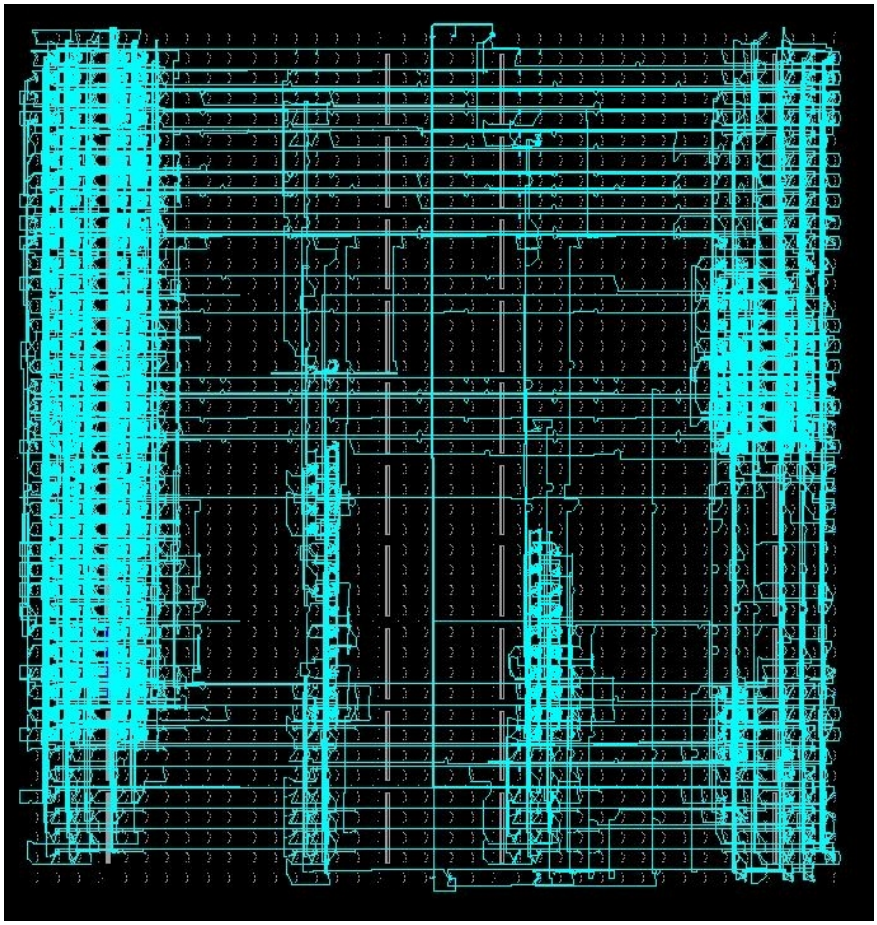

Abbildung 6.8: FPGA-Beispielimplementierung zur Darstellung der Ergebnisse

Die für die Auswahl der Komponenten (Module) aus der Bibliothek benötigte Zeit betrug unter Verwendung des BitstreamComposers weniger als eine Minute. Durch den automatisierten Ablauf wurde die Zusammenstellung der Logik für den FPGA innerhalb kürzester Zeit (<1 Sekunde) erstellt und stand dann zur sofortigen Verwendung bereit. Das Ergebnis war ein voll funktionsfähiger Bitstream für einen Virtex-II-1000. Alle Module zeigten die erwartete Funktion und deren Register konnten problemlos ausgelesen und beschrieben werden. Ein Test der Ein- und Ausgangssignale verlief ebenfalls erfolgreich.

6.3 CompassControl - Bedienung und Steuerung des Systems

6.3.1 Grafische Benutzeroberfläche (GUI)

Auf dem Windows-Host-PC, also dem Entwicklungsrechner des Benutzers der COMPASS-Plattform, läuft eine Steuerungsapplikation, die eine Verbindung zum Netzwerk-Server der Plattform herstellt.

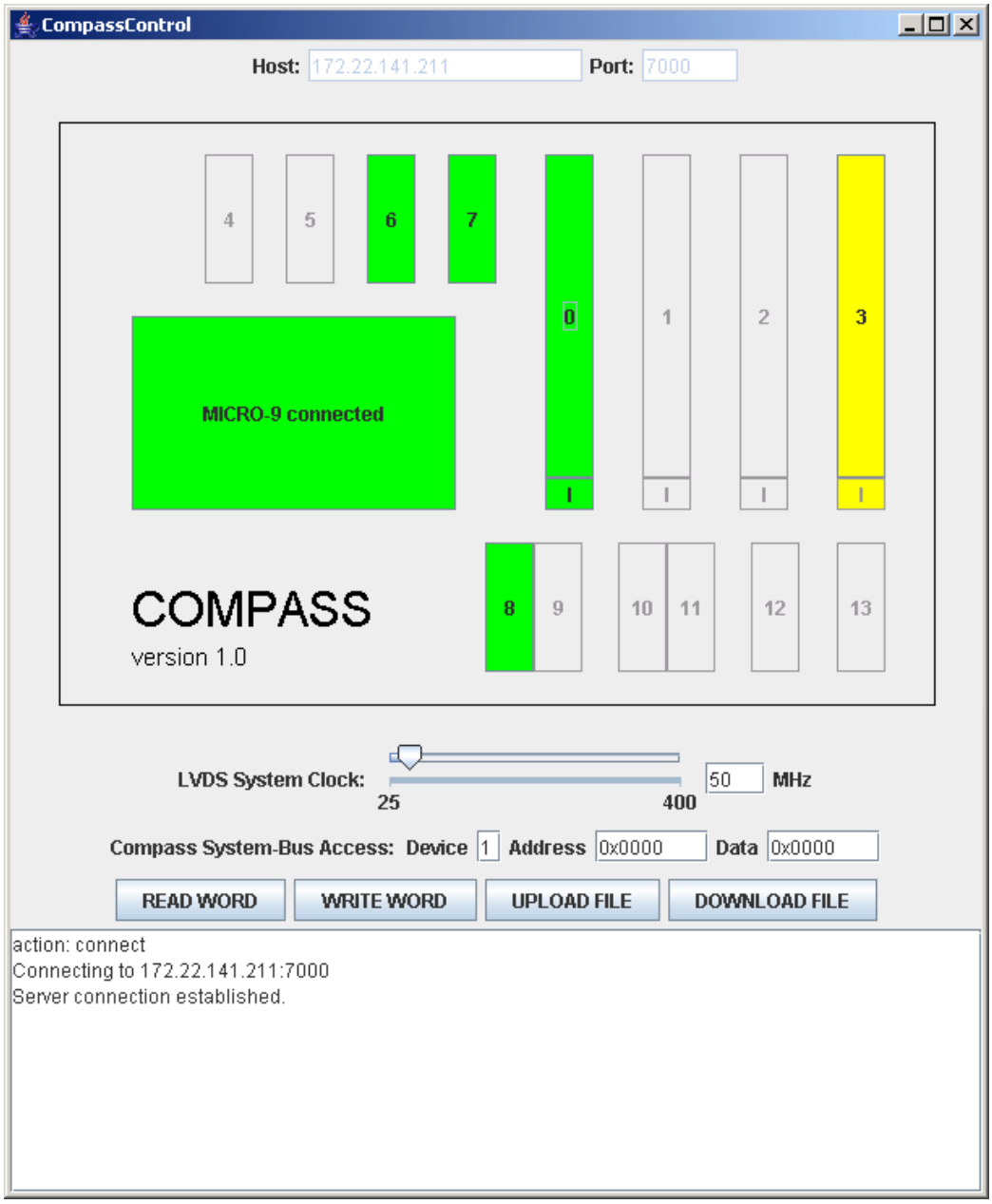

Abbildung 6.9: Screenshot der graphischen Benutzeroberfläche CompassControl

Die Applikation ist als graphische Benutzeroberfläche ausgeführt und zeigt im Hauptfenster eine Blockdarstellung der Compass-Plattform. Die Steuerung und Konfiguration der Module erfolgt durch Anklicken der als Buttons realisierten Komponenten. Der Status aller Module wird über verschiedene Farben signalisiert. In Grau dargestellte Module zeigen dabei nicht vorhandene Steckkarten an, wobei dann auch sämtliche Schaltflächen dieses Moduls deaktiviert sind. Ist eine Karte eingesteckt, wird dies nach dem Einlesen der Systemkonfiguration als gelber Block dargestellt. Nach erfolgreicher Konfiguration des Moduls ändert sich die Farbe in grün, bei Auftreten eines Ausnahmezustands wird der Block rot. Die nachfolgende Abbildung 6.9 zeigt einen Screenshot der gestarteten Applikation nach Einlesen der Systemkonfiguration.

Die Programmierung der Anwendung erfolgt in Java. Die Gründe hierfür liegen darin, dass Java zur Entwicklung plattformunabhängiger Software-Applikationen konzipiert wurde und eine einfache und schnelle Möglichkeit darstellt, graphischer Benutzeroberflächen zu gestalten. Da gerade im Bereich der Entwicklung von eingebetteten Systemen häufig auch Unix- und Linux-Systeme zum Einsatz kommen, lässt sich das Programm somit gut auch auf andere Betriebsysteme portieren.

6.3.2 Bedienung des Programms

Im geöffneten Hauptfenster der Anwendung kann die Verbindung zur Compass-Plattform durch einen Mausklick auf den großen, das MICRO-9-Board darstellenden, Button aufgebaut werden. Daraufhin wird eine Verbindung zur im oberen Bereich des Fensters einstellbaren IP-Adresse und der angegebenen TCP-Portnummer hergestellt. Die Defaultwerte sind die für den Prototypen der COMPASS-Plattform festgelegten Werte. Soll zu einem späteren Zeitpunkt die Verbindung getrennt werden, genügt ein weiterer Mausklick auf denselben Button.

Nach dem Abfragen der Systemkonfiguration werden alle eingesteckten FPGA- und I/O-Karten angezeigt. In diesem Schritt wird im selben Schritt geprüft, ob alle vorhandenen I/O-Karten miteinander kompatibel sind. Ist dies der Fall, können direkt die Ausgangstreiber aller Karten aktiviert werden. Sollte eine Inkompatibilität erkannt werden, wird dies durch eine rote Einfärbung der entsprechenden Steckplätze signalisiert. Nachdem die Verbindung hergestellt ist und keine Konflikte mehr bestehen, kann das Programmieren der FPGA-Karten beginnen. Hierzu muss der entsprechende FPGA-Slot in der Applikation mit der Maus angeklickt werden. Im sich darauf öffnenden Dateidialog kann dann der gewünschte Bitstream ausgewählt werden. Normalerweise liegen für alle speziell für die Compass-Plattform erstellten Bitstreams zugehörige XML-Dateien mit Beschreibungen und Angaben zur Ressourcen-Nutzung des Bitstreams vor. Diese XML-Datei kann dann direkt anstelle des eigentlichen Bitstreams geöffnet werden, da in ihr ein Verweis auf die zugehörige Bitstream-Datei mit der Dateiendung „*.bit“ gespeichert ist.

Ist für den zu ladenden Bitstream jedoch keine geeignete XML-Datei vorhanden, so kann auch direkt die Datei des Bitstreams verwendet werden. Dabei ist dann allerdings zu beachten, dass keine der in Kapitel 5.8 vorgestellten Sicherheitsüberprüfungen durchgeführt werden kann. Bei fehlerhafter Systemkonfiguration kann es also zu Konflikten kommen, die sogar bis zur Zerstörung der Plattform führen können. Der Vorteil bei der Nutzung dieser XML-Dateien mit Meta-Informationen liegt darin, dass die CompassControl-Applikation auch das Vorhandensein aller für diesen Bitstream benötigten Ressourcen überprüfen kann. Dies umfasst sowohl die Prüfung auf für die Funktionalität benötigte I/O-Karten als auch die Prüfung der Kompatibilität des Bitstreams mit den vorhandenen Karten.

Sind alle Überprüfungen erfolgreich, wird schließlich der Bitstream über das MICRO-9 auf die FPGA-Karte hochgeladen. Unmittelbar unterhalb eines jeden der vier FPGA-Slots befindet sich ein kleiner Info-Button, mit dem die zuvor ausgewählte XML-Datei in einem Webbrowser-Fenster geöffnet werden kann. Ist diese Datei dann mit einem geeigneten XSL-Stylesheet verknüpft, kann die XML-Datei vom Webbrowser in lesbares HTML transformiert werden (siehe Abbildung 6.13). Damit wird die Erstellung einer System-Dokumentation hinsichtlich der in einer Konfiguration verwendeten Schnittstellen optimal unterstützt.

Über den Schieberegler in der Mitte des Fensters lässt sich die Taktfrequenz des programmierbaren Taktbausteins PCK12429 von Philips einstellen. Die gewünschte Frequenz wird dabei nach den Formeln aus dem Datenblatt des Chips [PCK12429] in die benötigten Multiplikator- und Divisor-Werte umgerechnet. Nach dem Übertragen dieser Werte in das auf dem Board-Control-FPGA dafür vorgesehene Register, wird der Taktgenerator von der in VHDL programmierten FPGA-Logik neu geladen.

Unter dem Takt-Schieberegler befinden sich Eingabefelder und Schaltflächen für Schreib- und Lesezugriffe auf den Compass-Systembus. Abhängig davon, welche Schaltfläche betätigt wird, haben die Eingabefelder unterschiedliche Funktionen. In allen vier Fällen gibt jedoch der Wert im Feld Device an, auf welches Gerät am COMPASS-Systembus sich der Zugriff bezieht. Gerät 0 ist dabei der Board-Control-FPGA, auf welchen der Anwender des Systems eigentlich nie zugreifen muss. Daher ist sicherheitshalber als Default-Wert für das Device eine 1 eingetragen. Die Geräte 1 bis 4 sind dann die vier FPGA-Karten in den Slots 0 bis 3.

- Schaltfläche READ WORD:
  Hiermit kann ein einzelner Lesezugriff auf den Bus durchgeführt werden. Nach Eintragen des gewünschten Device und der zu lesenden Adresse kopiert ein Mausklick auf diesen Button den Inhalt des adressierten Registers in das Feld Data.
- Schaltfläche WRITE WORD:
  Analog zum Lesezugriff kann auch ein Schreibzugriff durchgeführt werden. Die zu schreibenden Daten sind zuvor in das Feld Data einzutragen.
- Schaltfläche UPLOAD FILE:
  Dieser Button öffnet einen Dateidialog, mit dem eine XML-Datei ausgewählt werden kann, über deren Inhalt Register oder RAM-Blöcke der FPGA-Karten beschrieben werden sollen. Für diesen Vorgang ist lediglich der in das Feld Device eingegebene Wert relevant, da die hochzuladenden Daten und die zugehörigen Adressen alle in dieser XML-Datei gespeichert sind. Diese XML-Datei braucht ein beliebiges Root-Element und kann in der nächsten Ebene dann beliebig viele Elemente des Typs register besitzen. Jedes dieser Register braucht dann ein Attribut address und ein Attribut value.
- Schaltfläche DOWNLOAD FILE:
  Eine solche Datei kann natürlich auch von der Plattform heruntergeladen werden. Dazu ist im Feld Device das richtige Gerät auszuwählen, im Feld Address die Startadresse ab der kopiert werden soll einzugeben und im Feld Data schließlich die Anzahl der zu lesenden Worte festzulegen. Nach dem Eingeben dieser Werte öffnet ein Mausklick auf die Schaltfläche Download File dann einen Dateidialog, mit dem der Pfad und der Dateiname der Zieldatei ausgewählt wird. Diese vier soeben vorgestellten Funktionen werden zum Debugging der Plattform und zum Steuern der auf den FPGA-Karten laufenden RP- oder HiL-Modellen benötigt. Die im Hauptfenster der CompassControl-Applikation ganz unten platzierte Textbox dient zur Anzeige von Statusmeldungen während des Programmlaufs. Dies wurde für die Entwicklung der Java-Anwendung selbst benötigt, kann jedoch auch bei Fehlbedienung des Benutzers hilfreiche Hinweise auf die Ursache des Problems geben.

6.3.3 Klassendiagramm CompassControl

Die Software CompassContro wurde mit Java erstellt und beinhaltet folgende Klassen (Abbildung 6.10). Der Übersichtlichkeit wegen sind die Standard-Klassen, die im Java Development Environment (JDE) bereits vorhanden sind und hier benutzt werden, nicht mit in die Darstellung mit aufgenommen. Außerdem wurden die Attribute der einzelnen Klassen in der Abbildung ausgeblendet und nur die Methoden aufgenommen.

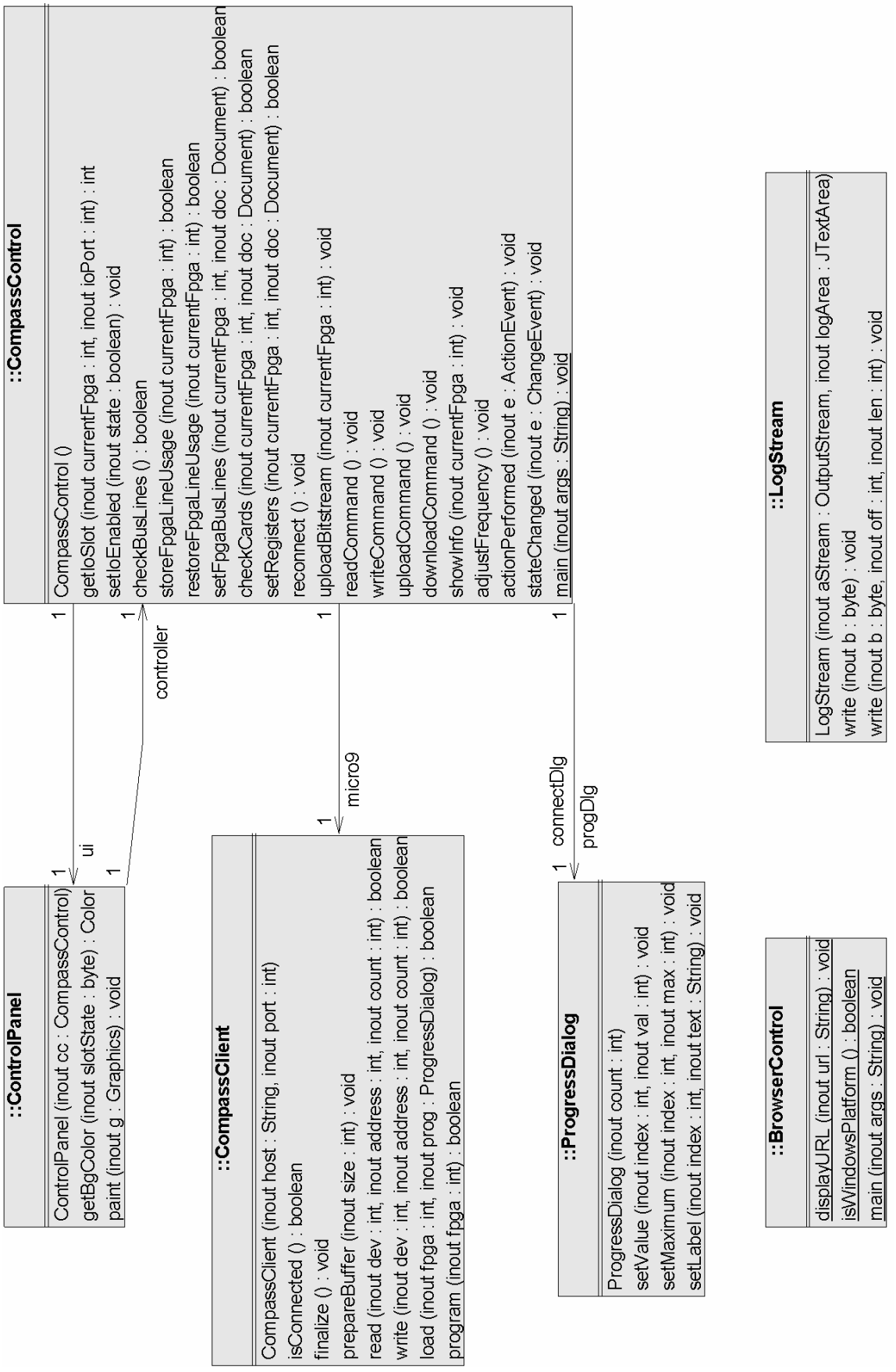

Abbildung 6.10: Klassendiagramm der CompassControl

6.3.4 Kommunikation zwischen GUI und Plattform

Die Verbindung zur Compass-Plattform erfolgt über eine beliebige IP-Verbindung mit darüber liegendem TCP-Stack. Unter Java lässt sich ein TCP-Socket über die Klasse java.net.Socket öffnen. Details zur Verwendung dieser Klasse sind zum Beispiel in [Mahm96] zu finden. Die Steuerungs-Applikation für die Compass-Plattform – im Folgenden Compass-Control genannt – erhält speziell für das Handling des TCP-Sockets eine eigene Klasse: CompassClient. Diese Klasse definiert folgende Variablen:

```
protected Socket serverConn;
private DataOutputStream dout;
private DataInputStream din;
private int allocSize;
public byte[] dataBuffer;
public int dataSize;
```

Die erste Variable vom Typ Socket hält den TCP-Socket, welcher im Konstruktor der Klasse CompassClient geöffnet wird. Jedes Mal, wenn die Anwendung CompassControl eine Verbindung zum Mikrocontroller-Board MICRO-9 herstellen will, wird also ein Objekt vom Typ CompassClient erzeugt; Soll die Verbindung beendet werden, muss diese Klasse zerstört werden. Der Verbindungsaufbau wird wie es in [Mahm96] erklärt ist, durchgeführt. Nach dem Setzen verschiedener Socket-Optionen kann mit Socket.connect() die Verbindung hergestellt werden. Danach wird der Socket mit einem DataInputStream und einem DataOutputStream verknüpft. Über diese Streams kann dann später der Datenaustausch erfolgen.

```
serverConn = new Socket();
serverConn.setTcpNoDelay(true);
serverConn.setKeepAlive(true);
serverConn.setSoTimeout(1000);
serverConn.connect(new InetSocketAddress(host,port),1000);
dout = new DataOutputStream(serverConn.getOutputStream());
din = new DataInputStream(serverConn.getInputStream());
```

Um bei einer Zerstörung der Klasse CompassClient noch rechtzeitig die Verbindung sauber abbauen zu können, ist die Methode finalize() zu überschreiben und darin die Methode Socket.close() aufzurufen:

```
if (serverConn != null)
{ serverConn.close(); }
```

Zur eigentlichen Datenkommunikation dient ein von der Klasse CompassClient verwalteter Datenpuffer. Durch den Aufruf der Methode prepareBuffer() kann der Puffer auf eine bestimmte Mindestgröße eingestellt werden. Um nicht ständig durch Vergrößern oder Verkleinern des Speicherbereichs für diesen Puffer das Programm auszubremsen, merkt sich CompassClient in der Variable allocSize, wie viel Speicher bereits belegt wurde. Sobald mehr Speicher angefordert wird, wird der Puffer vergrößert, wird weniger angefordert, muss nichts unternommen werden. In der Variable dataSize wird jeweils die zuletzt angeforderte Puffergröße gespeichert.

```
dataSize = size;
if (size > allocSize)
{ dataBuffer = new byte[size];
allocSize = size; System.gc(); }
```

Die eigentliche Datenübertragung soll hier nicht im Detail erläutert werden. Im Wesentlichen wird hier exakt das gleiche Protokoll implementiert, wie dies schon zuvor für den Netzwerkserver gemacht wurde. Der Unterschied besteht jedoch darin, dass hier jeweils zuerst ein Befehl mit zugehörigen Daten an den Netzwerkserver übertragen wird und danach die Antwort des Servers in den zuvor als Sendepuffer genutzten Speicherbereich eingelesen wird, da dieser zugleich auch Empfangspuffer ist.

Erwähnenswert ist an dieser Stelle noch die Realisierung der Übertragung von Bitstreams. Da hier Datenmengen im Bereich von 500kByte bis zu 1Mbyte übertragen werden sollen, würde dies nach dem bisher vorgestellten Übertragungsverfahren sowohl die Kapazität des Sendepuffers auf Client-Seite, als auch die des Empfangspuffers auf Server-Seite schnell überlaufen lassen. Als Lösung wurde hier das Aufteilen des Bitstreams in 10 kByte große Datenblöcke gewählt. Beim Senden des Kommandos zum Übertragen eines Bitstreams wird daher auch immer signalisiert, ob mit diesem Befehl ein neuer Bitstream übertragen wird, oder ob die anhängenden Nutzdaten einen bereits übertragenen Teil-Bitstream ergänzen. Auf diese Weise konnte eine geringe Speicherauslastung und zugleich akzeptable Datenübertragungsrate erreicht werden. Das hier implementierte Protokoll kann natürlich in der Geschwindigkeit absolut nicht mit perfekt optimierten Protokollen wie FTP oder HTTP mithalten, bietet jedoch für diesen Verwendungszweck ausreichende Performanz und einen vertretbaren Implementierungsaufwand.

6.3.5 Bitstream-Beschreibung im XML-Format

Es wurde bereits erwähnt, dass zur Speicherung der Bitstreams eine zusätzliche Datei im XML-Format benötigt wird. Diese Datei enthält neben dem Dateinamen und einer Beschreibung aller Submodule des Bitstreams auch die komplette Ressourcenbelegung (siehe Abbildung 6.13). So kann etwa angegeben werden, welche FPGA-Karte benötigt wird und welche I/O-Karten über welche Bussysteme angeschlossen sein müssen. Zur Überprüfung, dass jeweils nur ein Ausgangstreiber auf einer Busleitung aktiv ist, wird zusätzlich für jeden Bus aufgelistet, auf welchen Signalleitungen durch den Bitstream Ein- und Ausgänge definiert sind. Schließlich kann noch eine Konfiguration festgelegt werden, über die nach dem Programmieren des FPGAs automatisch bestimmte Register mit den angegebenen Werten vorbelegt werden. Die DTD (Document Type Definition) des hier verwendeten XML-Formats erlaubt folgende Elemente und Attribute:

```
<!ELEMENT bitstream (module+, resources, filename)>
<!ELEMENT filename (#PCDATA)>
<!ELEMENT module (name | description)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT resources (bus*)>
<!ELEMENT bus (signal*)>
<!ATTLIST bus
   name CDATA #REQUIRED
   width CDATA #REQUIRED
>
<!ELEMENT signal (#PCDATA)>
<!ATTLIST signal
```

```
  index CDATA #REQUIRED
  state (unused | input | output) #REQUIRED
>
<!ELEMENT configuration (register*)>
<!ELEMENT register (#PCDATA)>
<!ATTLIST register
  address CDATA #REQUIRED
  value CDATA #REQUIRED
>
```

Eine solche XML-Datei sieht zum Beispiel wie folgt aus:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bitstream>
  <filename>compass\test.bit</filename>
  <module>
    <name>Modul 1</name>
    <description>erstes Test-Modul</description>
  </module>
    <module>
    <name>Modul 2</name>
    <description>zweites Test-Modul</description>
  </module>
  <resources>
    <bus name="ioB0" width="10">
      <signal index="0" state="output" />
      <signal index="1" state="output" />
      ...
      <signal index="9" state="output" />
    </bus>
    <bus name="ioB1" width="10">
      <signal index="0" state="input" />
      ...
      <signal index="9" state="unused" />
    </bus>
    <bus name="pcB" width="15">
      <signal index="0" state="unused" />
      ...
      <signal index="15" state="unused" />
    </bus>
  </resources>
  <configuration>
    <register address="0x1002" value="0x1234" />
    <register address="0x1004" value="0x5678" />
    ...
  </configuration>
</bitstream>
```

Diese XML-Datei lässt sich auch als Klassendiagramm darstellen und hat dann folgendes Aussehen:

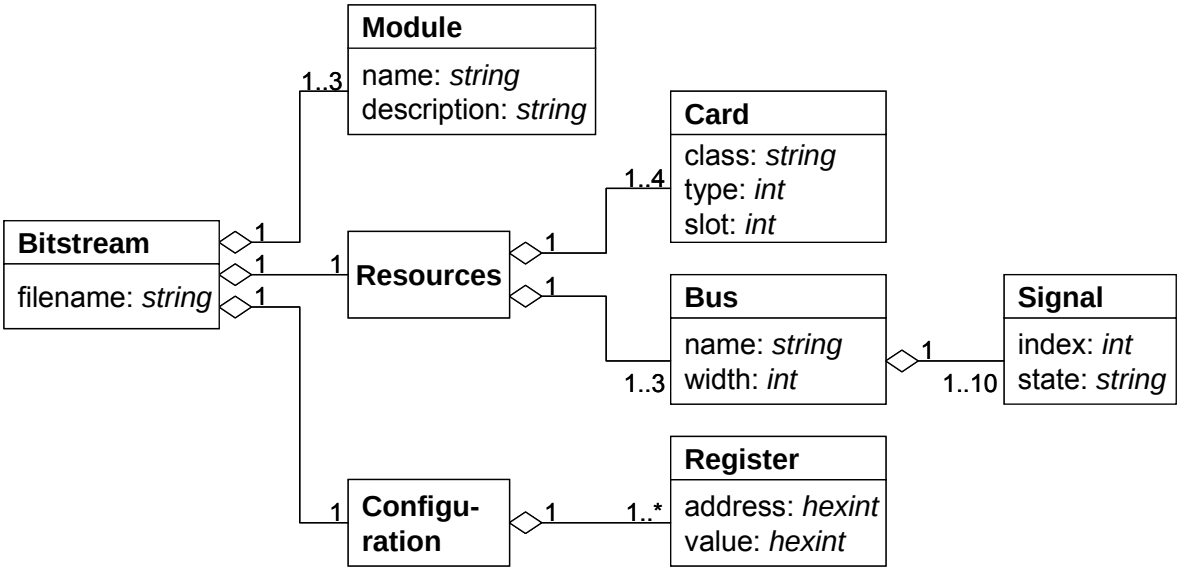

Abbildung 6.11: Klassendiagramm der XML-Datei zur Beschreibung Bitstream-Inhalte

6.3.6 Kompatibilitätsprüfung zwischen Bitstream und Plattform

Nach dem Booten der Plattform, werden wie bereits beschrieben, die EEPROMs aller Steckkarten über den Plattform-Control-FPGA eingelesen und in dessen internem RAM abgelegt. Um die vorhandene Hardware-Konfiguration zu prüfen und die Belegung von Signalleitungen validieren zu können, werden diese Inhalte nach dem Verbindungsaufbau zwischen Host-PC und Plattform vom FPGA zum Host-PC transferiert. Dieser Vorgang ist in Abbildung 6.12 visualisiert.

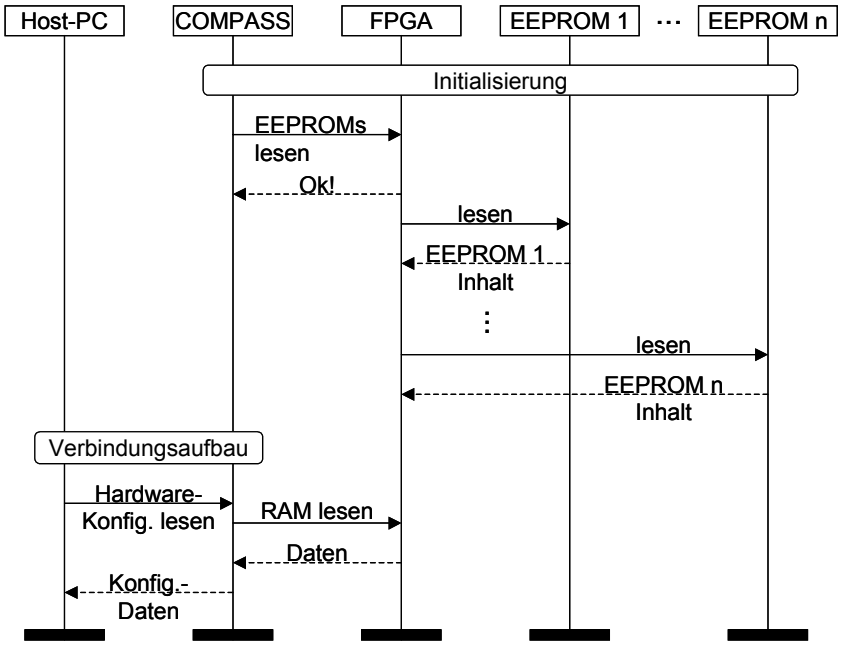

Abbildung 6.12: Auslesen der HW-Konfiguration nach Booten der Plattform

Ein weiteres Mal wird eine Überprüfung der Signalleitungen ausgeführt, bevor ein neuer Bitstream auf einen FPGA geladen wird. Dabei können dann gleich auch die benötigten Signalleitungen dieses Bitstreams mit berücksichtigt werden. Alle Informationen aus den EEPROMs und bei FPGA-Karten zusätzlich aus dem geladenen XML-File werden im zweidimensionalen Array cardInfo abgelegt. Der erste Index von cardInfo selektiert die Steckkarte, der zweite Index das gewünschte Feld der Karteninformation. Die ersten drei Felder sind dabei eine direkt Kopie der EEPROM-Daten, welche in der folgenden Tabelle 6.4 aufgelistet sind. Die nachfolgenden drei Felder werden von der CompassControl-Applikation während des Programmlaufs aus dem vierten Feld der EEPROM-Daten bzw. den Informationen der XML-Datei des entsprechenden Bitstreams erzeugt. Dabei wird die Bitmaske der Pinbelegung dann abhängig vom Steckplatz der Karte in eines der drei Felder für die Busse ioB0 (I/O-Bus 0), ioB1 (I/O-Bus 1) und PCB (Peripheral Communication Bus) kopiert. Der Bitstream des FPGAs kann dabei natürlich auf allen drei Bussen Ausgangstreiber besitzen.

Feld	Bedeutung	Wertebereich
0	Karten-Klasse	1=FPGA, 2=IO, 3=AUTOMOTIVE
1	Karten-ID	1-0xFFFF
2	Karten-Seriennummer	1-0xFFFF
3	Aktive Ausgangstreiber	Bitmaske für alle 16 Pins des I/O-Steckers

Tabelle 6.4: Bedeutung der ersten vier 16Bit-Felder der Karten-EEPROMs

Die Überprüfung des Arrays cardInfo erfolgt in der Funktion checkBusLines(), welche zuerst für jede Leitung des Peripheral-Communication-Bus die Anzahl der Bustreiber zählt. Sollte mehr als ein Bustreiber vorhanden sein, werden die entsprechenden Karten als Konfigurationsfehler markiert und erscheinen im Hauptfenster der Applikation rot.

```
for (int i=0; i<16; i++)
  {
    int numberOfDrivers = 0;
    for (int j=0; j<8; j++)
      {numberOfDrivers += (cardInfo[j][5] >> i) & 1;}
    if (numberOfDrivers > 1)
      {
        ret = false;
        for (int j=4; j<8; j++)
          {
            if (((cardInfo[j][5] >> i) & 1) == 1)
             {slotState[j] = stateBadConfig;}
}}}
```

Als nächstes werden die bis zu zwei vorhandenen dedizierten I/O-Busse der FPGA-Karten überprüft. Für die Zuordnung zwischen FPGA-Karte und daran angeschlossenen I/O-Slots steht die Funktion getIoSlot() zur Verfügung. Deren Funktion lässt sich sehr leicht anhand des Screenshots in Abbildung 6.9 nachvollziehen. Die ersten beiden FPGA-Karten besitzen zwei I/O-Slots, die letzten beiden besitzen nur den I/O-Slot ioB0. Daraus ergeben sich die folgenden Berechnungsformeln für die Slot-Nummern der I/O-Slots jeder FPGA-Karte:

```
switch (ioPort)
  {
    case 0:
      return (currentFpga<2?8+2*currentFpga:10+currentFpga);
    case 1:
      return (9+2*currentFpga);
    default:
      return -1;
  }
```

Das Überprüfen der einzelnen Leitungen der I/O-Slots kann dann nacheinander für jeden der vier FPGAs durchgeführt werden, indem bei jeder I/O-Leitung die Anzahl der Ausgangstreiber aufsummiert wird und bei mehr als einem Treiber je Leitung ein Konfigurationsfehler signalisiert wird. Der zweite I/O-Bus ist nur für die ersten beiden FPGAs zu überprüfen, da die anderen beiden nur einen I/O-Slot besitzen.

```
//check ioB0
for (int j=0; j<4; j++)
  {
    for (int i=0; i<10; i++)
      {
        int ioB0 = getIoSlot(j,0);
        if ((cardInfo[j][3] >> i&1) + (cardInfo[ioB0][3] >> i&1) > 1)
          {
            ret = false;
            slotState[j] = stateBadConfig;
            slotState[ioB0] = stateBadConfig;
  } } }

//check ioB1
for (int j=0; j<2; j++)
  {
    for (int i=0; i<10; i++)
      {
        int ioB1 = getIoSlot(j,1);
        if ((cardInfo[j][4] >> i&1) + (cardInfo[ioB1][4] >> i&1) > 1)
          {
            ret = false;
            slotState[j] = stateBadConfig;
            slotState[ioB1] = stateBadConfig;
  } } }
```

Da in der XML-Datei zu einem Bitstream auch Informationen bezüglich der benötigten Steckkarten abgelegt sind, erfolgt vor dem Laden des Bitstreams ein Vergleich von erforderlichen und vorhandenen Karten durch die Funktion checkCards(). Details hierzu können dem vollständigen Quellcode in [Bahl06] entnommen werden.

6.3.7 Dokumentation der Schnittstellenkonfiguration

Unmittelbar unter jeder der vier FPGA-Slots in der Applikation CompassControl gibt es einen kleinen Info-Button, mit dem die zuvor ausgewählte XML-Datei in einem Webbrowser-Fenster geöffnet werden kann. Ist diese Datei dann mit einem geeigneten XSL-Stylesheet verknüpft, kann die XML-Datei vom Webbrowser in lesbares HTML transformiert werden. Damit wird die Erstellung einer System-Dokumentation optimal unterstützt.

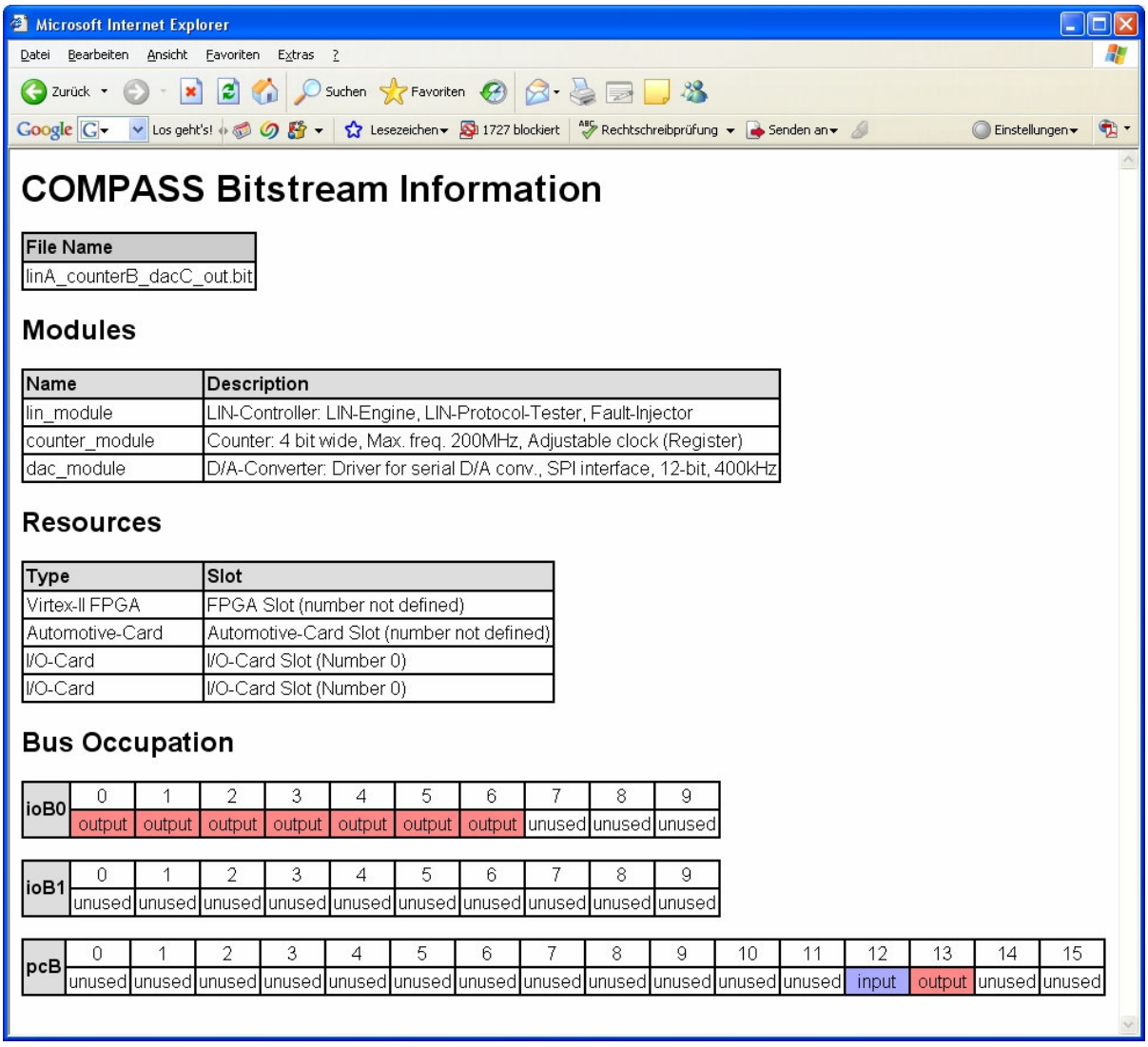

Abbildung 6.13: Automatisch erzeugte Schnittstellendokumentation in HTML

6.3.8 Remote-RP und Remote-HiL

Wie bereits in den vorhergehenden Kapiteln beschrieben, findet die Kommunikation zwischen den Entwicklung- und Steuerungstools und der COMPASS-Plattform über eine TCP/IP-Verbindung statt. Dabei ist es zunächst unerheblich, ob der Host-Rechner und das Target im selben Raum, im selben Gebäude oder weit voneinander getrennt irgendwo auf der Welt verteilt aufgestellt sind. Solange zwischen beiden eine funktionsfähige Datenverbindung besteht (zum Beispiel über das Internet), kann eine Kommunikation, in diesem Falle eine Fernsteuerung, stattfinden. Damit ist es grundsätzlich möglich, Rapid Prototyping- oder Hardware-in-the-Loop-Szenarien über große Distanzen durchzuführen, ohne als Testperson selbst vor Ort zu sein. Die Änderung der digitalen Schnittstellen ist weiterhin online möglich,

ohne einen Hardware-Austausch an der Plattform vornehmen zu müssen. Die Änderungen können vor Ort auf dem Host-PC erledigt und anschließend in Form von neuen Konfigurationsdaten über die offene Verbindung an das Target geschickt und dort hochgeladen werden.

Aus diesen grundsätzlichen Überlegungen heraus wurden die Begriffe „Remote-RP“ und „Remote-HiL“ geboren, die derzeit allerdings noch eher als Schlagworte denn als fundierte Zukunftstechnologien fungieren und somit weiteren intensiven Forschungen unterliegen. Bei solchen Tests sind weitere, bisher nicht näher betrachtete Randbedingungen, wie die Laufzeit und Konsistenz der Datenpakete, die Sicherheit der Übertragung, Abhörsicherheit im Internet durch Tunneln, die Beobachtung der ausgeführten Aktionen vor Ort (z.B. per Kamera) usw., zu beachten. Insgesamt eine Fülle von Themen, die aus den beiden Begriffen ein komplexes und wichtiges Arbeitsgebiet hinsichtlich einer weiter vorangetriebenen Politik des „Outsourcing“ machen.

In einem Experiment konnten bereits einige Erfahrungen mit extremen Distanzen bei der Datenkommunikation über Internet gesammelt und dabei diverse Schwierigkeiten identifiziert werden. Für diesen Zweck wurde in Zusammenarbeit mit der Universität in Perth, Australien, eine künstliche „Probestrecke“ für Datenpakete zwischen Steuerungsrechner und COMPASS-Plattform geschaffen. Alle von unserem Steuerungs-PC gesendeten Pakete wurden dort vor Ort über einen vorher definierten Port vom Server entgegengenommen und mit Hilfe eines Scripts auf einem anderen Port und unter einer anderen IP-Adresse wieder an unsere Plattform zurückgeschickt. Da es sich dabei um absolut unkritische Versuchsdaten handelte und auch ansonsten nur eine Hand voll Mitarbeiter auf beiden Seiten von dem Experiment wussten, wurde auf jegliche Art der Absicherung verzichtet. Die Applikation bei uns vor Ort, das Fahren eines funkferngesteuerten Modellautos, war nur sehr schwer bzw. zeitweise gar nicht möglich. Dies lag zum einen an der starken Verzögerung als auch an dem Verlust einiger Pakete.

6.4 Modellbasierte Applikationserstellung

Neben der bibliotheksbasierten Konfiguration von FPGAs, wie in Kapitel 5 bereits ausführlich beschrieben, wurden im Laufe der Arbeit auch andere Methoden untersucht, die einen größeren Freiheitsgrad für den Benutzer zur Implementierung von Funktionen bieten. Ausgehend von dem CASE-Tool MATLAB und den Erweiterungspaketen (sog. Toolboxen) Simulink und Stateflow, sowie der von Xilinx angebotenen Bibliothek namens SystemGenerator [Xil04l] lassen sich regelungstechnische und ereignisgesteuerte Systeme modellieren und simulieren.

Um diese Modelle letztlich auch als ausführbaren Code in einer realen Schaltung verwenden zu können, bietet MATLAB Code-Generatoren an. Im Falle der eigenen Toolboxen Simulink und Stateflow wird aus dem Modell ein C-Code erzeugt, der sich mit Änderungen und Anpassungen versehen auf einem Prozessor oder Controller ausführen lässt. Benutzt man den SystemGenerator von Xilinx wird eine VHDL-Datei generiert, die mit Hilfe der Xilinx Integrated System Environment (ISE, [XISE05]) zu einem Bitstream zum Download auf einen FPGA verwandelt werden kann. Allerdings können je nach Größe des Modells und Art der Modellierung mehrere Iterationsschritte oder manuelle Nacharbeit nötig werden, bevor das Modell korrekt auf der Hardware ausgeführt wird. Abbildung 6.14 gibt einen Überblick über die verschiedenen Möglichkeiten und Wege von einem Modell zu einer Hardware-Implementierung zu gelangen.

Alle genannten Varianten wurden auf ihre Tauglichkeit bezüglich einfacher Push-Button-Lösungen hin untersucht. Dabei wurde versucht, durch entsprechende Maßnahmen, eine auf die Hardware hin optimierte Umgebung zur Einbettung des Modells zu schaffen und so die bereits erwähnten händischen Eingriffe des Benutzers zu minimieren, um einen durchgängigen Design-Flow zu erreichen. Im Folgenden wird nun auf die verschiedenen alternativen Modellierungsarten eingegangen, wobei die oben genannte Umgebung als „Framework“ (Rahmen-Design) bezeichnet wird. Seine Aufgabe ist es, Ein- und Ausgänge des Systems

als definierte Schnittstellen zur Verfügung zu stellen, die es dem Anwender erleichtern auf externe Signale zuzugreifen. Der Vorteil dabei ist, dass sich der Modellentwickler vollends auf das Modell an sich konzentrieren und die übrigen Funktionen als korrekt gegeben ansehen kann.

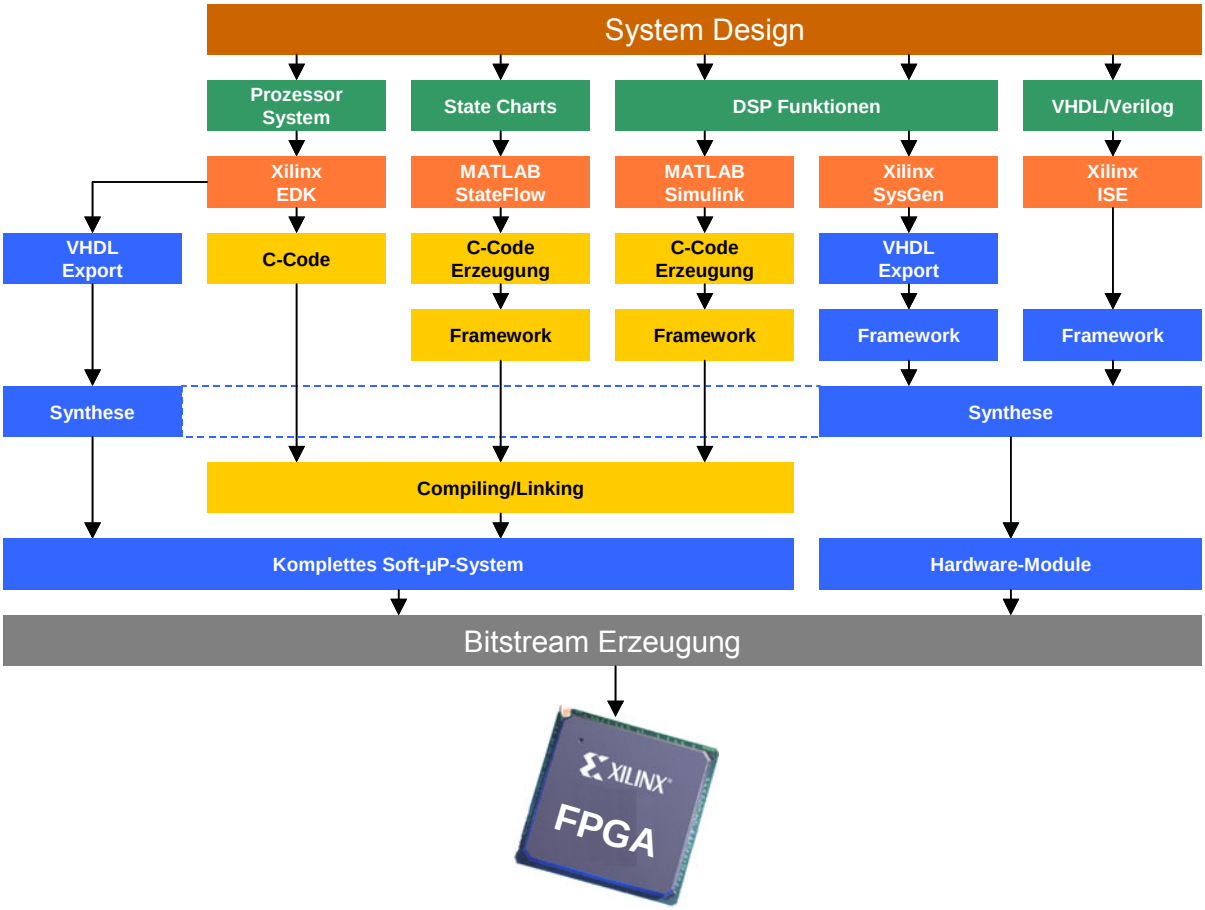

Abbildung 6.14: Möglichkeiten zur Hardware-Implementierung modellbasierter Entwürfe

6.4.1 Modellierung kontinuierlicher Systeme

Bei der Umsetzung kontinuierlicher Systeme in eine reale Hardware kommen zwei Lösungsvarianten in Frage. Zum einen besteht die Möglichkeit, die Modellierung mit MATLAB/Simulink durchzuführen und daraus C-Code zu erzeugen, der zur Ausführung einen Prozessor benötigt – hier ein MicroBlaze Soft-IP-Core von Xilinx. Die Verwendung von anderen Prozessoren, wie PowerPC (Xilinx) oder einem Nios oder ARM (Altera), die sich als feste Kerne auf dem FPGA befinden oder als IP-Core implementieren lassen, ist alternativ genauso möglich. Zum anderen kann aus einem mit dem SystemGenerator von Xilinx erstellten Modell VHDL ausgegeben werden, das als Basis für eine direkte Hardware-Implementierung auf einem FPGA dient. Beide Vorgehensweisen werden hier kurz umrissen.

6.4.2 Modellausführung auf einem MicroBlaze

Einer Modellausführung auf einem Prozessor – unabhängig davon, ob es sich dabei um eine Hardware-Komponente handelt oder einen Softcore-Prozessor – geht eine Modellierung in MATLAB/Simulink voraus. Nach eingehender Simulation des eigentlichen Modells, kann mit Hilfe eines Coders ein äquivalenter C-Code erzeugt werden. Um diesen auf dem Target ablaufen zu lassen, bedarf es einiger Änderungen oder Ergänzungen, die den Quellcode des

Modells an die Gegebenheiten des Prozessors und der Peripherie anpassen. Diese können händisch vorgenommen werden, oder durch Bereitstellung eines Frameworks automatisiert erfolgen. Im ersten Fall sind diese Eingriffe allerdings bei jedem neuen Modell und bei jeder Änderung desselben erneut zu tätigen, wodurch die Framework-Variante bei häufigen Änderungen erhebliche Vorteile mit sich bringt. Letztere wird daher auch Gegenstand der näheren Untersuchung bei der Implementierung von Modell-Code auf dem Soft-IP-Core-Prozessor sein.

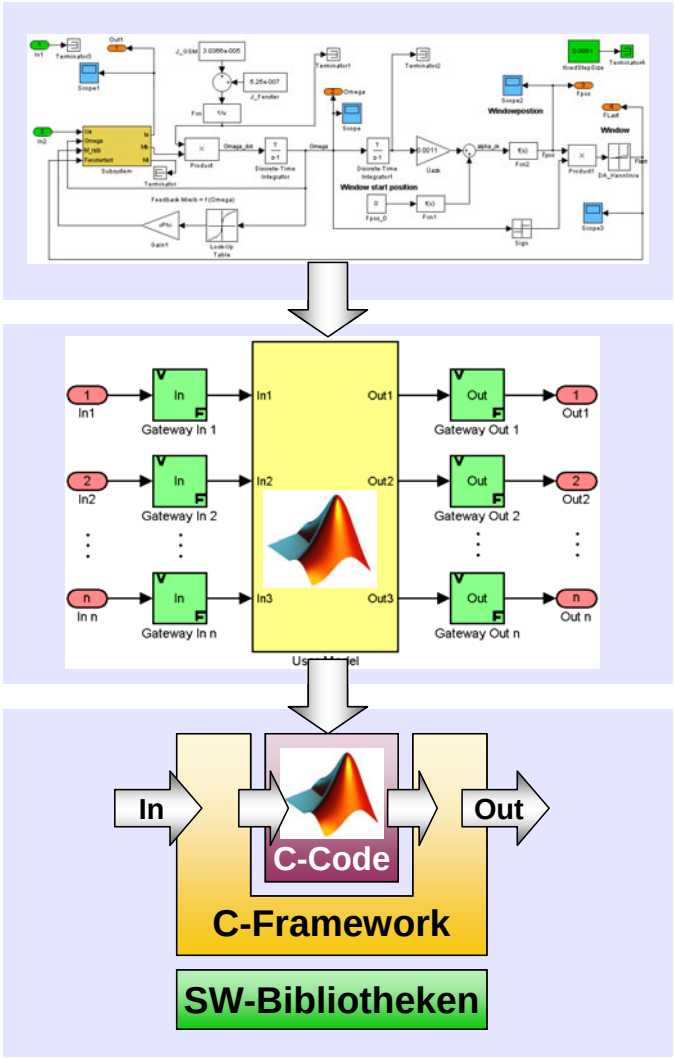

Abbildung 6.15: Ablauf der Modellierung und Einbettung in ein Framework

Im Rahmen dieser Untersuchung wird der Soft-IP-Core-Prozessor MicroBlaze von Xilinx verwendet. Mit Hilfe der Design-Umgebung EDK [XEDK04] kann dazu ein System, bestehend aus dem eigentlichen Prozessor und peripheren I/O-Komponenten, zusammengestellt werden. Diese können dabei direkt der Design-Umgebung entstammen oder vom Benutzer selbst ergänzt worden sein.

6.4.2.1 Framework für Modelle

Um einen einfachen und komfortablen Modellentwicklungsablauf zu gewährleisten, sollen die aus dem Realtime-Workshop erzeugten Quellcode-Dateien möglichst nicht mehr von Hand bearbeitet oder in irgend einer Art angepasst werden müssen. Idealerweise werden die erzeugten Modellquellcode-Dateien ohne weitere Betrachtung gleich weiterverarbeitet. Dazu

ist das Vorhandensein weiterer Code-Dateien notwendig, die z.B. eine main-Funktion zur Verfügung stellen, um ein lauffähiges Programm kompilieren zu können.

Das Framework bietet dazu das Ein- und Ausgabe-Handling, sowie neben der fest definierten Hauptfunktion den Einsprungpunkt und die Zeitsteuerung zur Modellberechnung. Die Anzahl der Ein- und Ausgabekanäle oder -variablen ist abhängig von der Applikation. Das Minimum einer Implementierung ist ein Ausgangswert. Die Summe der zusätzlichen, für jedes Modell gleichen, Quellcode-Dateien wird Framework genannt. Sie bieten den Rahmen für das Benutzermodell. Einen Eindruck, welche Dateien das Framework bilden, findet man in Abbildung 6.16. Eine modellcharakterliche Darstellung zeigt Abbildung 6.17.

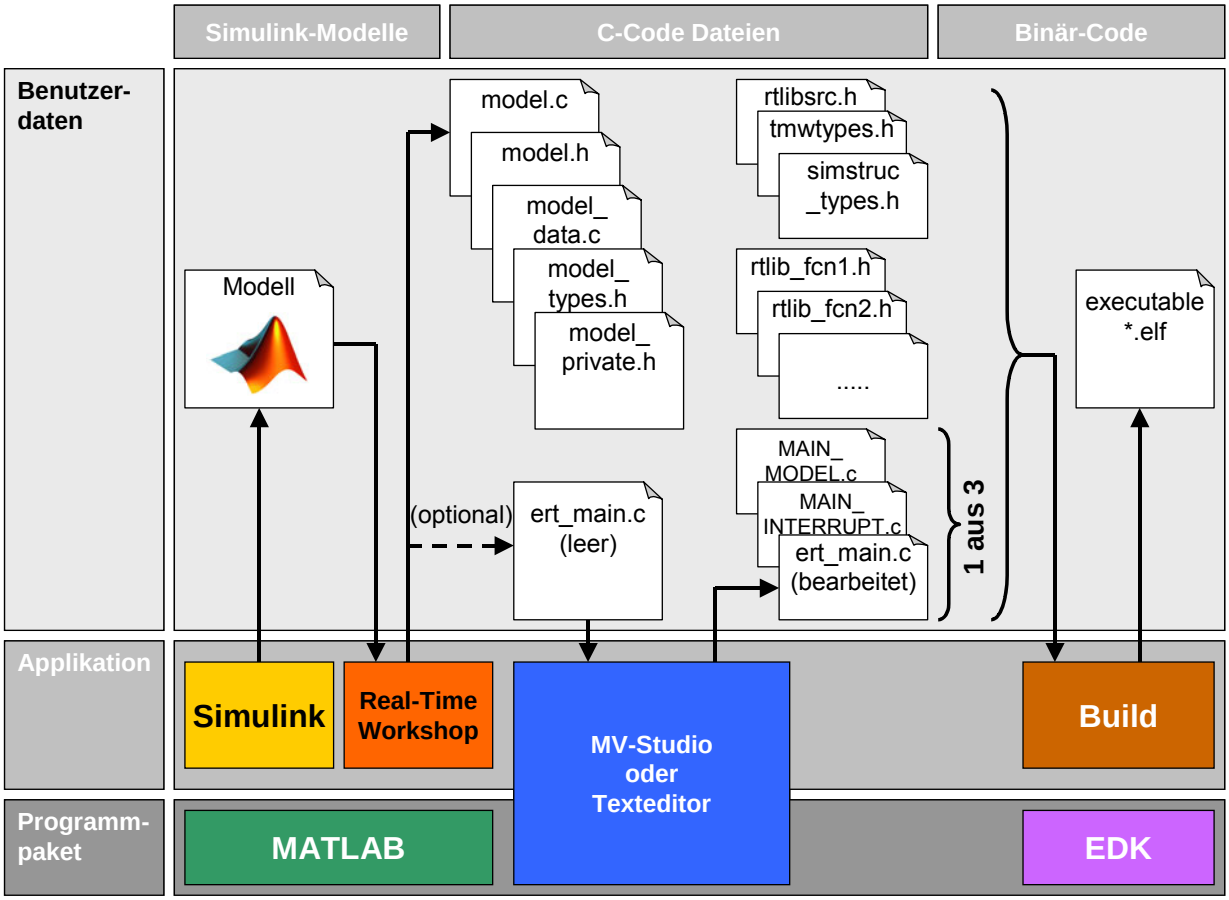

Abbildung 6.16: Tool-Kette und Datenfluss

Für die zeitliche Steuerung zum Aufruf der Berechnungsroutine, d.h. des Modells, sind zwei Möglichkeiten gegeben und daher exemplarisch in zwei Varianten implementiert. In der Datei MAIN_MODEL.c befindet sich die main-Funktion, die das Modell aufruft und sich um die Ein- und Ausgabe kümmert. Sie kann zum Debuggen verwendet werden, da sie die Modellschrittfunktion periodisch ohne Verzögerung aufruft. Die Datei MAIN_INTERRUPT.c. enthält zusätzlich eine Interrupt-Steuerung für den MicroBlaze. Voraussetzung für eine solche Steuerung ist das Vorhandsein eines Interrupt-Kanals oder eines Interrupt-Controllers in der Hardware. In der main-Funktion befindet sich eine leere Endlosschleife. Tritt ein Interrupt auf, so wird die Funktion user_interrupt_handler aufgerufen, die wiederum rt_OneStep – eine neue Berechnung – auslöst. Als Benutzer des Frameworks muss man sich für eine der beiden Varianten entscheiden.

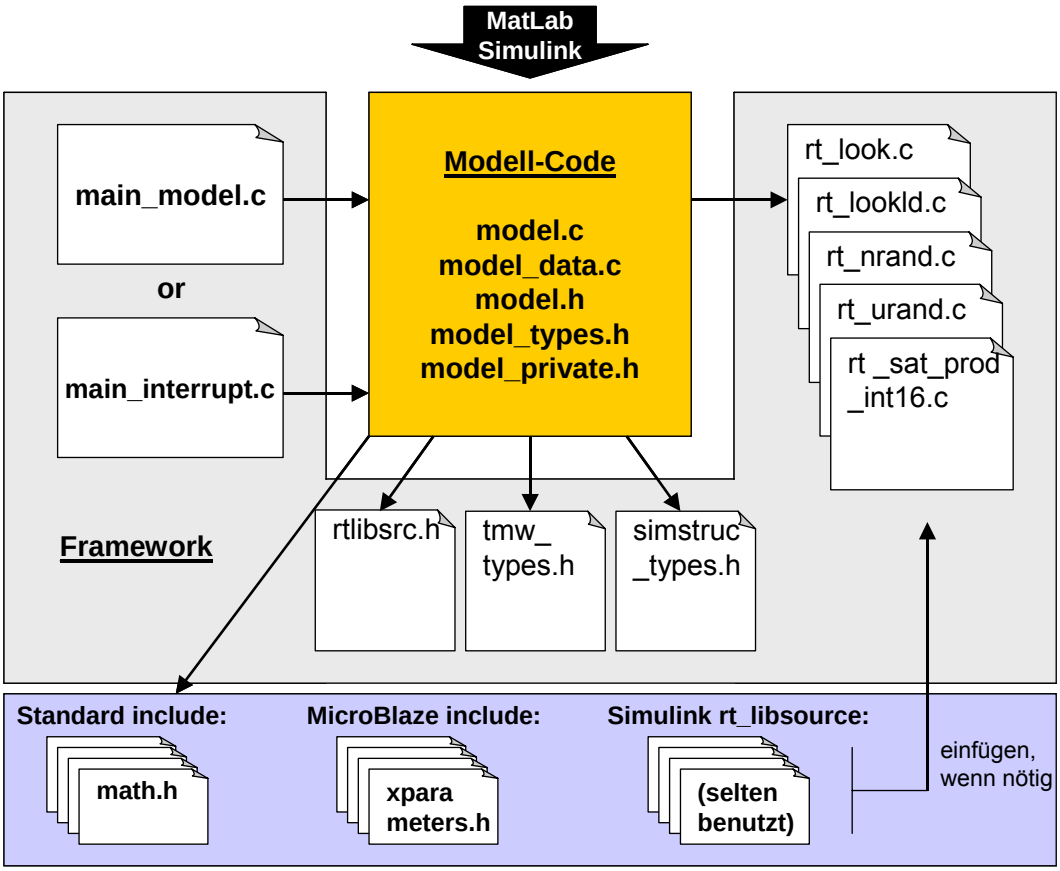

Abbildung 6.17: Modell, Framework und Bibliotheksdateien

Alle in Abbildung 6.17 dargestellten Pfeile stellen statische include-Anweisungen dar. So müssen z.B. die Dateien tmwtypes.h, rtlibsrc.h und simstructypes.h in dem gleichen Verzeichnis liegen wie die Modelldateien, damit sie bei der Kompilierung zur Verfügung stehen. Häufig verwendete Funktionen aus dem Simulink RT-Workshop, wie z.B. das Auslesen einer Tabelle, zählen auch mit zum Framework und sind somit im Framework-Verzeichnis anzutreffen. Selten benutzte Funktionen/Dateien befinden sich im Verzeichnis matlab_libsrc. Da es sich um mehr als 270 Dateien handelt, zählen sie nicht mehr zum Framework, um die Übersichtlichkeit zu gewährleisten. Die Standardbibliotheken und MicroBlaze-spezifische Dateien, z.B. für die Ein- und Ausgabe, werden von der XPS-Umgebung automatisch eingebunden.

6.4.2.2 Regeln der Modellerstellung zur Verwendung in einem Framework

Um ein eigenes Modell zusammen mit dem Framework benutzen zu können, müssen einige Vorgaben beachtet werden, damit das Zusammenspiel funktioniert. Jedes Modell hat genau definierte Eingänge und Ausgänge. Diese sind im Framework festgelegt und stellen für das Modell die Verbindung zur Außenwelt her. Bei der durch einen Interrupt getriggerten Version des Frameworks müssen die Berechnungsschritte genau an die Modellschrittweite gekoppelt sein. Die Angabe der Schrittweite wird im Modell durch eine Konstante angegeben und trägt einen bestimmten Namen – z.B. „fixedStepSize“ (fss). Erstellt man Festkommamodelle mit dem „fixed-point Blockset“, ist darauf zu achten, dass das Ausgabeformat des Konst-Blocks auf „double“ eingestellt wird. Ansonsten wird die fss nicht in dezimaler Form in der generierten Quellcodedatei model_data.c gespeichert und die Berechnung des richtigen Timer-Initialisierungswertes schlägt fehl.

Für die Namensgebung ist es wichtig, die Eingänge gemäß den Konventionen des Frameworks zu benennen (z.B. „In1“ für Eingang 1 oder „Out5“ für Ausgang 5). Das Modell trägt

üblicher Weise den Namen „model“. Die Signalleitung zwischen dem Eingang und dem ersten angeschlossenen Block darf keinen Namen tragen, da dieser Name den Eingangsname „In1“ im Quellcode ersetzen würde. Bei dem Ausgang verhält es sich gleich. Ein Prototypenmodell eines PID-Reglers sieht man in Abbildung 6.18. Er besteht aus einem Proportionalglied, einem Integrationsglied und einem Differenziationsglied, wobei alle drei parallel geschaltet sind. Deshalb lautet die Übertragungsfunktion in elementarer Form:

$$G(s) = K_p + \frac{K_I}{s} + K_D s$$

Formel 6.1: Formel für PID-Regler (1)

Häufiger wird aber die Darstellung in Formel 6.2 verwendet. Anstelle der drei Verstärkungsfaktoren erhält man die Parameter Verstärkungsfaktor K_P, die Nachstellzeit T_N und die Vorhaltezeit T_V.

$$G(s) = K_P\left(1 + \frac{1}{T_N s} + T_V s\right)$$

Formel 6.2: Formel für PID-Regler (2)

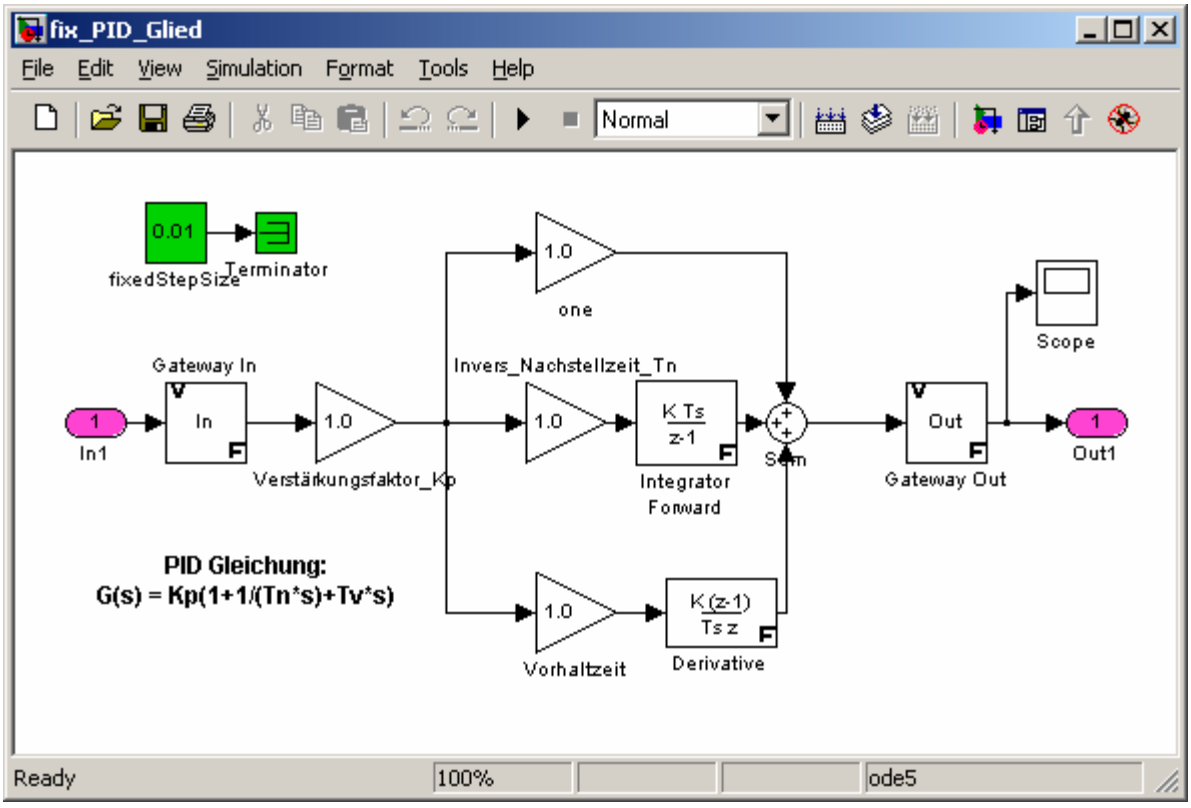

Abbildung 6.18: Prototypenmodell (PID-Regler) in SIMULINK

Entscheidet man sich für einen Modellansatz ohne Festkommaarithmetik, so können die Gateways weggelassen werden. Für Modelle ohne externen Eingang schließt man den Eingang einfach direkt mit einem Terminator kurz. Dies ist z.B. bei einer reinen Signalerzeugung der Fall.

6.4.2.3 Code-Generierung und Kompilieren

Da die Simulink-Modelle auf einem Soft-IP-Core-Prozessor in einem FPGA von Xilinx zum Ablauf kommen sollen, muss zunächst C-Code erzeugt werden. Aus MATLAB heraus kann man dies mit Hilfe des Embedded Real-Time Workshops möglichst einfach bewerkstelligen. Dieser Quellcode kann dann in XPS für den MicroBlaze Prozessor kompiliert werden. Der Quellcode eines Simulink-Modells teilt sich auf folgende fünf Dateien auf:

- model.c(enthält die Architektur des Modells)
- model_data.c (enthält alle Parameterwerte, wie z.B. Verstärkungsfaktoren)
- model_private.h
- model_types.h
- model.h

Um den Quellcode zusammen mit dem Framework das erste Mal zu kompilieren, verwendet man die Datei MAIN_MODEL.c. Das Ergebnis der Kompilierung ist eine Datei mit der Endung „*.elf“. Die Endung „elf“ steht für „executable and link format“ und stellt das kompilierte Programm in Binärform dar.

6.4.2.4 Einschränkungen für Embedded C-Coder

Bei der Verwendung des Realtime-Workshop Embedded Coder der MATLAB-Umgebung unterliegen die Modelle einigen Einschränkungen, auf die sinnvoller Weise schon bei der Modellerstellung in SIMULINK geachtet werden sollte. Realtime-Workshop Embedded Coder-Programme werden in diskreter Zeit ausgeführt, d.h. man muss ein Lösungsverfahren mit einer festen Modellschrittweite verwenden. Somit muss auf den Einsatz von Komponenten aus der kontinuierlichen Bibliothek verzichtet werden. Weitere Einschränkungen findet man in [Jung04].

6.4.2.5 Modellierung mit Festkommaarithmetik

Der Bibliotheks-Browser von MATLAB bietet nicht nur die Standard-Simulink-Bibliothek an, sondern auch ein „Fixed-Point Blockset“, mit dem man Modelle auf Basis von Festkommazahlen erstellen kann. Die Motivation sich mit Modellen zu befassen, die nicht auf Basis von Fließkommazahlen, sondern auf Basis von Festkommazahlen beruhen, ergibt sich aus dem Wunsch, höhere Ausführungsgeschwindigkeiten auf Prozessoren zu erzielen, die keine Hardware-Unterstützung für Fließkommazahlen im Format „float“ und „double“ bieten. Das ist bei dem Softcore-Prozessor MicroBlaze der Fall. Die höhere Geschwindigkeit ist vor allem für Modelle wichtig, die in Echtzeit ausgeführt werden müssen. Durch die Verwendung von Festkommaarithmetik sind teilweise enorme Geschwindigkeitssteigerungen möglich. Dies wurde anhand einiger Modelle getestet. Eine kurze Übersicht der Testmodelle zeigt Tabelle 6.5.

Modell	Variante		Verbesserung
	Fließkomma	Fixpunkt	
GSM	ca. 146us	ca. 31us	4,7
NTC	ca. 356us	ca.360us / (104us mit exp-Tabelle)	3.42 (mit Tabelle)
Fensterheber	ca. 1,45ms	ca. 133us	10.9
PID_Glied	ca. 133us	ca. 73us	1.8

Tabelle 6.5: Vergleich Berechnungsgeschwindigkeit Modellschritt (50MHz)

Die GSM modelliert die elektrischen Eigenschaften eines Gleichstrommotors bei konstanter Drehmomentbelastung, beim Fensterheber handelt es sich um die Erweiterung der GSM um den mechanischen Teil eines Autofensters. Der NTC-Temperatursensor berechnet die Ausgangsspannung eines Spannungsteilers, bei dem einer der beiden Widerstände temperaturabhängig ist. Das PID-Glied ist ein Regler, der aus Proportional-, Integral- und Differentialanteil besteht. Details zu den Implementierungen finden sich in [Jung04].

Anhand der Tabelle erkennt man, dass große Modelle mit vielen Berechnungen von der Umstellung tendenziell mehr profitieren als kleine mit wenigen Berechnungen. Bei dem Modell NTC zeigt sich, dass die fixed-point-Variante keine Verbesserung bringt, wenn für die Berechnung einer Exponentialfunktion die Zahlenwerte zwischenzeitlich wieder in das Fließkommaformat umgewandelt werden müssen. Deshalb ist die Exponentialfunktion in einer Tabelle hinterlegt. Als weiterer ausführungszeitsenkender Anteil bei dem obigen Fensterhebermodell in Fixpunktversion, ist z.B. die Startposition des Fensters als Konstante angegeben und nicht mehr über eine arcsin-Funktion berechnet, die vom Fixpunktblocksatz nicht unterstützt wird.

6.4.2.6 Vor- und Nachteile der Festkommaarithmetik

Bei der Verwendung der Festkommaarithmetik sollte man sich der Vor- und Nachteile gegenüber einer konventionellen Simulink-Modellierung im Klaren sein. Modelle im Festkommaformat können eine wesentlich schnellere Ausführungsgeschwindigkeit erzielen, wenn die Zielplattform keine Fließkommazahlen unterstützt (FPU). Dadurch ist es möglich die Modelle auf einer preiswerteren Zielplattform zur Ausführung zu bringen. Allerdings werden viele Simulink-Blöcke der Standardbibliothek nicht unterstützt, wodurch die Modelle größer werden, da sie aus mehr elementareren Blöcken bestehen. Quasi alle Funktionen außer Sinus und Kosinus müssen als Potenzreihe oder Tabelle implementiert werden, wie z.B. die Exponentialfunktion (siehe Formel 6.3).

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Formel 6.3: Exponentialfunktion als Potenzreihe

Gegenüber dem Fließkommaformat findet eine Quantifizierung der Variablen statt, die dazu führt, dass die Berechnungsergebnisse nicht mehr optimal sind und zu Fehlern im Modell führen können. Außerdem ist der aus diesen Modellen erzeugte Quellcode nicht nur umfangreicher, sondern auch komplizierter und damit schwerer zu lesen.

6.4.3 Integration als reine Hardware-Lösung

Dem Anwender soll ein Werkzeug an die Hand gegeben werden, das es ihm ermöglicht, möglichst einfach und schnell, ohne umfangreiche manuelle Anpassungen, Funktionen

und/oder Systemmodelle direkt auf einem FPGA zu implementieren. Per „Knopfdruck“ soll es möglich sein, das Modell automatisch auf reale Hardware abzubilden – mit möglichst geringer, im Idealfall gänzlich ohne manuelle Anpassung durch den Anwender. Um diese Anpassung ohne das Zutun des Bedieners durchzuführen, sind diese in vorkonfigurierten Templates umgesetzt.

6.4.3.1 SystemGenerator

Der SystemGenerator ist eine Erweiterung der MATLAB/Simulink-Umgebung im Rahmen einer entsprechenden Toolbox. Die grafische Bedienoberfläche ermöglicht es dem Anwender Funktionsmodelle per „drag & drop“ zu erzeugen und anschließend zur Weiterbearbeitung oder Implementierung auf Hardware, in HDL oder als Netzlistendatei zu exportieren. Die Bedienung erfolgt weitgehend intuitiv. Dadurch ist es möglich, ein Funktionsmodell schnell, einfach und sehr effektiv aufzubauen und zu simulieren.

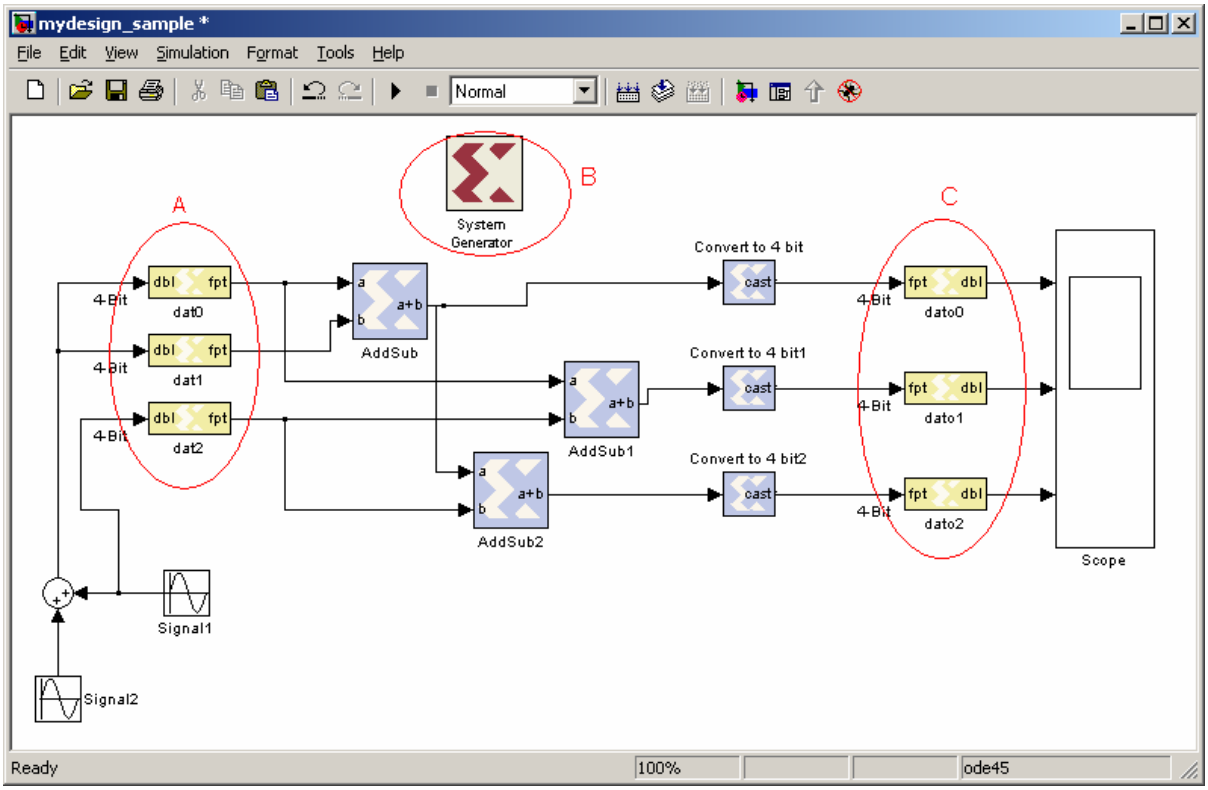

Abbildung 6.19: Beispiel eines SystemGenerator-Funktionsmodells

Ein Funktionsmodell (Abbildung 6.19) besteht grundlegend aus dem SystemGenerator Funktionsblock (Symbol wie Markierung B), MATLAB/Simulink-Komponenten (wie z.B. Scope), SystemGenerator-Designelementen (blau hinterlegt, wie z.B. AddSub) und Gateways (wie Markierungen A und C). Der SystemGenerator-Funktionsblock bildet das Interface zum SystemGenerator, um HDL-Code aus der Subsystemhierarchie zu generieren, in der sich dieser Block befindet. Ebenso ermöglicht er grundlegende, subsystemweite Einstellmöglichkeiten.

6.4.3.2 Vorgehensweise

Die Gateways innerhalb des SystemGenerators werden bei der Kompilierung in VHDL in eine Portmap umgewandelt. Die Gateway-Bezeichner werden dabei als Portnamen verwendet. Gleichzeitig bilden die Gateways die Verbindung zu MATLAB. In der MATLAB/Simulink-Umgebung wandeln sie Datenströme vom Typ Simulink integer, Floating Point oder double

in den Typ Xilinx Fixed Point um (Gateway In). Die Wandlung funktioniert natürlich auch in umgekehrter Richtung (Gateway Out). Der Bediener kann durch Doppelklick auf das entsprechende Gateway ein Fenster öffnen, in dem er detaillierte Einstellungen des Ausgabeformats vornehmen, das Gateway mit LOC Constraints belegen und die Samplerate des Gateways einstellen kann. Die LOC Constraints verbinden bei der Erzeugung des Bitfiles aus SystemGenerator heraus diese Datenleitung mit einem physikalischen I/O-Pin des FPGAs. Über ein XCF File werden eventuell gesetzte Constraints an die Downstream Tools weiter gegeben. Die Samplerate ermöglicht eine exakte Simulation der Schaltung. Mit ihr werden zum Beispiel die Sampleraten von A/D-Wandlern in die Simulation mit einbezogen.

Die Verknüpfung des SystemGenerators mit den Downstream Tools erfolgt über die Einstellungen des SystemGenerators Blocks. Dort ist es möglich, die verwendete Hardware-Beschreibungssprache (VHDL oder Verilog) zu wählen und festzulegen, welches Synthesetool verwendet werden soll (z. B. XST von Xilinx).

6.4.3.3 Das VHDL-Framework

Das VHDL-Framework ist ein Wrapper, der als Bindeglied zwischen FPGA und System-Generator-Modell fungiert. Es besteht aus den verschiedenen Komponenten, die zum Betrieb des Gesamtsystems notwendig sind. Im Fall der Testimplementierung sind dies die Ansteuerung der externen A/D- und D/A-Wandler. Die Ansteuerung ist in diesem Falle als Umsetzung der Ansteuerprotokolle zu verstehen, die innerhalb des Frameworks als jeweils eigenständige Komponenten formuliert und untereinander verbunden integriert sind. An deren Stelle können beliebige andere Bausteine eingesetzt werden, die ein dediziertes Protokoll erfordern und für das eigentliche Modell des Benutzers nur als Datenquelle oder –senke fungieren. Die Belegung der FPGA-Pins wird durch ein vordefiniertes UCF-File bestimmt. Darin sind alle Signale der Top-Level Domain des ISE-Projekts auf In- oder Out-Ports gelegt. Diese Datei ist demnach auch mit der Top-Level Entity verknüpft.

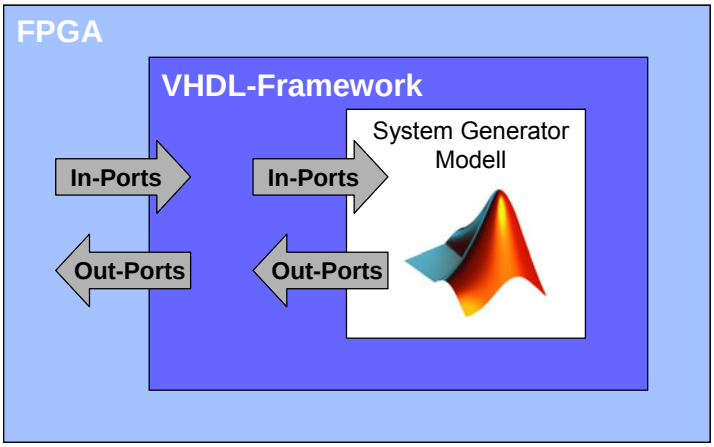

Abbildung 6.20: Schematische Darstellung des Frameworks für Modelle des SysGen

Die zu implementierende SystemGenerator Funktion wurde als „component“ bereits vordefiniert. Die darin definierten Ports entsprechen den Gateways der SystemGenerator-Templates. Die Nomenklatur der Komponente ist durch die Namensgebung der Blöcke innerhalb des SystemGenerators Designs fest vorgegeben. Bei der Generierung der Netzliste für diese Komponente verwendet der SystemGenerator den Namen des Templates und wandelt diesen um in [name_des_templates]_clk_wrapper.[datei_endung]. Die Dateiendung [datei_endung] hängt davon ab, in welche Dateiart das SystemGenerator-Modell kompiliert wird.

Alle verwendeten Komponenten sind innerhalb des Frameworks funktional fest untereinander verknüpft und über vorgenanntes UCF-File mit den I/O-Pins des FPGA verbunden. Dadurch erhält der Anwender eine feste vorgefertigte Peripherie, in die er seine Funktion einbetten kann, ohne sich um den Hardware-Aufbau und dessen Ansteuerung kümmern zu müssen. Diese wiederum ist festgelegt durch den externen Hardware-Aufbau.

Die Komponenten selbst können ebenfalls jeweils mit einem UCF-File verknüpft werden, um Area Constraints (der Bereich des FPGA, in dem die jeweiligen Komponenten platziert werden sollen) festzulegen. Dies kann mit individuellen UCF-Files für jede Komponente einzeln bestimmt werden. Es ist jedoch darauf zu achten, dass keine Constraints für die Verbindung mit I/O-Pins des FPGA in diesen UCF-Files integriert werden. Diese Festlegungen werden bereits vom UCF-File der Top-Level Entity getroffen.

Innerhalb des Frameworks wird auch die Taktung des Designs vorgenommen. Alle benötigten Takte werden erzeugt und den entsprechenden Komponenten zur Verfügung gestellt, bzw. auch über die I/O-Pins des FPGA an externe Bauteile weitergegeben. Innerhalb des SystemGenerators Designs können natürlich weiterhin Up-Sample und Down-Sample-Blocks verwendet werden. Die dadurch entstehenden Taktraten werden dann aber innerhalb des SystemGenerators Designs erzeugt und müssen den Komponenten nicht vom Framework zugeführt werden.

Das SystemGenerator-Modell liegt eingebettet im VHDL-Framework und ist über seine In-/Out-Ports mit ihm verknüpft. Im ISE-Projekt liegt das SystemGenerator-Modell als Komponente vor, ist jedoch nicht mit Inhalt gefüllt. Mittels SystemGenerator wird eine Netzliste erzeugt, die in das gleiche Verzeichnis wie das Framework kopiert wird. Danach wird diese Netzliste erkannt und ihr Inhalt in das Projekt mit eingebunden. Aus dem „Project Navigator“ heraus kann dann das Gesamtdesign in ein Bitfile synthetisiert werden.

6.4.3.4 SysGen-Template zur Modellerstellung

Das SystemGenerator-Template ist, wie in Abbildung 6.21 zu sehen, aufgebaut. Die zu implementierende Funktion ist bereits enthalten (rechteckiger Kasten in der Mitte) und kann zusammen mit dem Framework getestet werden. Um eine Netzliste erstellen zu können, die auch fehlerfrei in das vorgegebene VHDL-Framework einfügbar ist, dürfen die Gateways nicht mit LOC-Constraints belegt sein. Dies würde bewirken, dass Signale direkt mit den Pins des FPGA verbunden werden. Anders als bei dem Standard-Verfahren, bei dem ein Modell ohne Framework auf einem FPGA gebracht werden soll, ist dies hier nicht erwünscht. Die Ein- und Ausgabeports des Modells werden mit den Aus- und Eingängen des Frameworks verknüpft. Die eigentliche Pinbelegung wird erst im VHDL-Framework durchgeführt. Die Gateways des SystemGenerator-Templates werden bei der Kompilierung in die In/Out-Ports umgewandelt. Diese sind in der Portmap des Components aufgeführt, der das SystemGenerator-Design in den VHDL-Code integriert.

6.4.3.5 Ergebnisse

In [Klim05] wurden neben dem Framework auch andere Verfahren getestet. Das Framework-Verfahren ist davon das unflexibelste, da die komplette Peripherie durch das VHDL-Framework fest vorgegeben ist. Soll diese geändert werden, muss das komplette Framework überarbeitet werden. Mit VHDL-Kenntnissen des Anwenders ist dies jedoch relativ einfach zu bewerkstelligen. Der Vorteil dieses Verfahrens liegt darin, dass die einzelnen Module des Frameworks mit individuellen UCF-Files verknüpft werden können und damit das Gesamtdesign in Hinsicht auf Timing und Flächenausnutzung des FPGA optimiert werden kann. Damit ist es zum Beispiel möglich, Ansteuerfunktionen, die für verwendete externe Bauteile (z.B. AD-/DA-Wandler) nötig sind, in die Nähe der entsprechenden FPGA-Pins, die zur Ansteuerung dieser Bauteile verwendet werden, zu legen. Dadurch lassen sich Signallaufzeiten entsprechend verringern. Ein weiterer Vorteil ist die sehr einfache Verwendbarkeit dieses Verfahrens durch den Anwender.

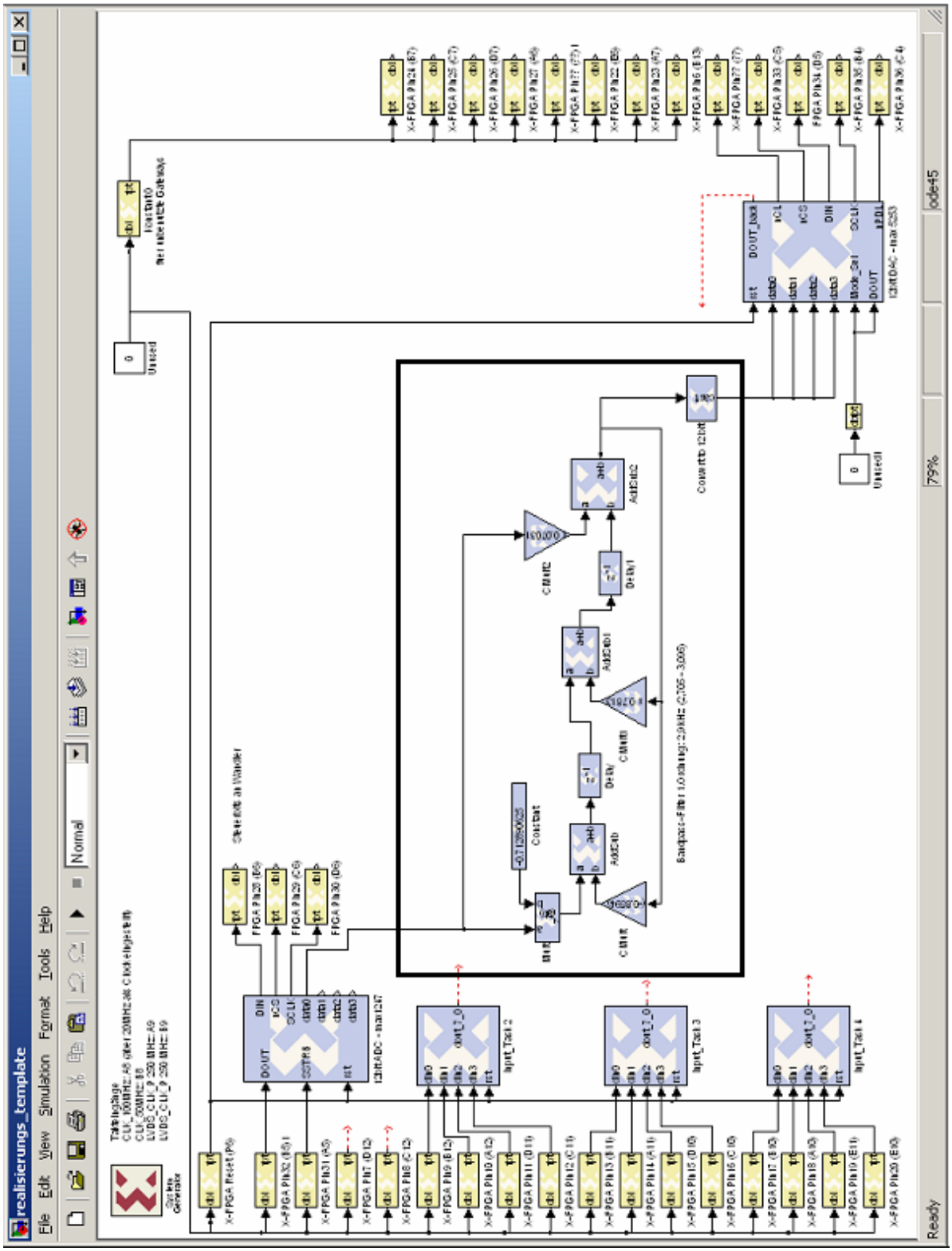

Abbildung 6.21: SysGen-Template zur Modellerstellung

6.4.4 Modellierung zeitdiskreter Systeme

Die Umsetzung ereignisgesteuerter Systeme in eine reale Hardware kann ebenso wie die bei kontinuierlichen Systemen auf zwei verschiedene Arten geschehen. Bei beiden Varianten wird zunächst das Modell in Form eines endlichen Automaten (engl.: Finite State Machine,

FSM) mit Hilfe von MATLAB/StateFlow erstellt und in dieser Umgebung simuliert und getestet. Anschließend kann die Entscheidung zugunsten einer Implementierung in Software oder in Hardware getroffen werden. Im ersten Fall wird aus dem Modell C-Code generiert, der auf einen Prozessor ausgeführt werden kann. In dem hier betrachteten Fall wird dazu der Soft-IP-Core-Prozessor LEON [LEON01] verwendet. Die Umsetzung des Modells in eine direkt auf einem FPGA zu integrierende Hardware wird unter Verwendung eines Konvertierungs-Tools (z.B. JVHDLGen [Drei06]) aus einer *.mdl Simulink-Modelldatei gewonnen und in eine Hardware-Beschreibung transformiert. Das Konvertierungs-Tool erzeugt aus der gegebenen Struktur eine State-Machine in der Hardware-Beschreibungssprache VHDL. Diese Art der Funktionsimplementierung wurde in der Arbeit allerdings nicht näher betrachtet und wird daher am Ende nur kurz angerissen.

6.4.4.1 Ausführung auf dem LEON-Prozessor

Bevor es zu einer Ausführung des aus dem Modell erzeugten C-Codes auf einem Prozessor kommt, muss derselbe und die weiterhin nötige Peripherie fertig gestellt worden sein. Es kann sich dabei um einen reinen Hardware-Aufbau handeln, bei dem die Komponenten durch reale Bauteile repräsentiert werden, oder aber um eine FPGA-Implementierung bzw. System-on-Chip-Lösung (SoC). Letztere soll auch bei der Darstellung der Modellierung der ereignisgesteuerten Systeme als Grundlage der Ausführungen dienen. Details dazu finden sich in [Schw03].

6.4.4.2 Design Flow

Die Erstellung der Steuerungs-Software beginnt mit der Modellierung der meist verbal gegebenen Spezifikation in einem CASE-Tool (Computer Aided System/Software Engineering) wie z.B. dem verwendeten MATLAB StateFLOW. Die Funktionalität wird graphisch durch Statecharts dargestellt und kann durch Simulation hinsichtlich Vollständigkeit und Korrektheit überprüft werden. In einem nächsten Schritt wird die abstrakte Verhaltensbeschreibung des lauffähigen Modells durch einen Code-Generator für die Ziel-Hardware in eine äquivalente Darstellung der Programmiersprache C oder C++ gewandelt.

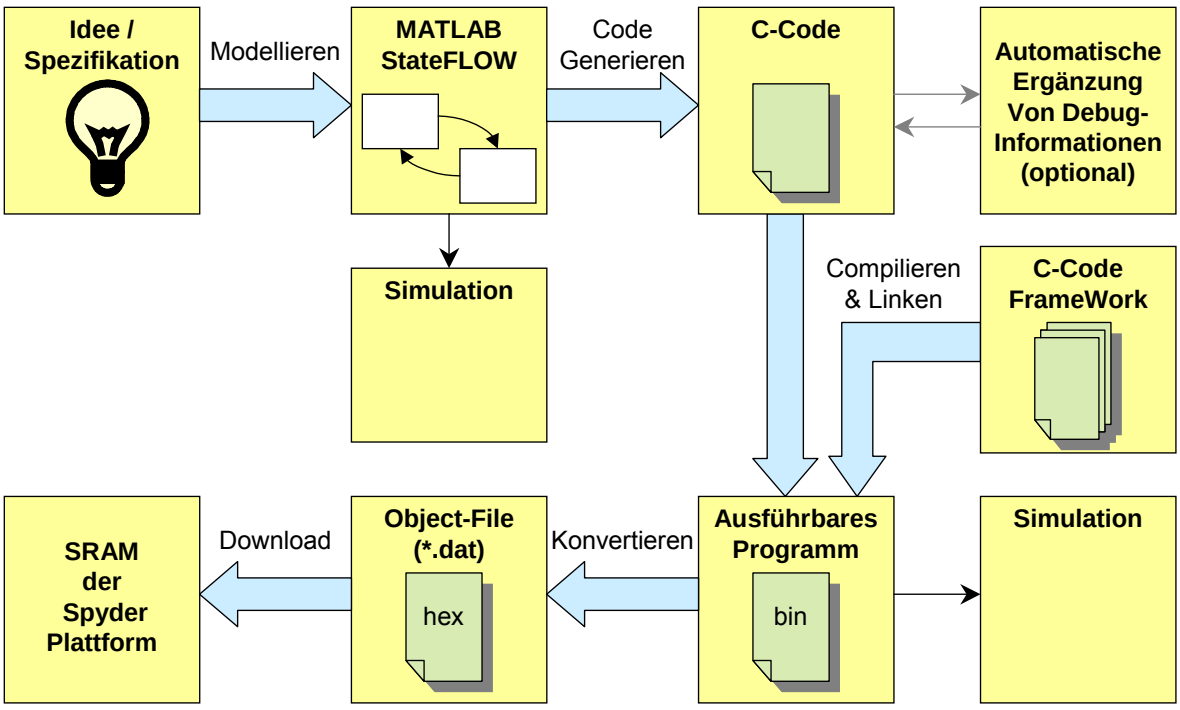

Abbildung 6.22: Design-Flow für ereignisgesteuerte Systeme

Für Test-Zwecke kann der resultierende Code optional mit Anweisungen für die Darstellung der internen Zustände versehen werden, die später im Betrieb als Debug-Informationen dienen, um Fehlfunktionen auf die Spur zu kommen. Zusammen mit einem für die Gesamtapplikation, d.h. für das Gesamtsystem, angepassten Software-Frameworks, wird aus dem Source-Code ein ausführbares Programm für den LEON [LEON01] erzeugt, das innerhalb der Entwicklungsumgebung erneut simuliert werden kann. Durch die Konvertierung in ein Object-File wird die Software in das für den Download nötige Hex-Format gewandelt und steht für die Übertragung in den Speicher des Steuergerätes bereit. Abbildung 6.22 zeigt den gesamten Design-Flow im Überblick.

6.4.4.3 Modellierung

Auch hier finden sich Parallelen zu den kontinuierlichen Systemen. Ausgehend von einer in der Spezifikation gegebenen Verhaltensbeschreibung wird der Systemablauf durch die Verwendung von Statecharts in MATLAB StateFLOW modelliert [StFl97]. Die dabei zu verwendenden Ein- bzw. Ausgangssignale sind größtenteils durch die angeschlossene Hardware vorgegeben. Aus den Eingangssignalen werden die Ereignisse abgeleitet, um den durch die Modellierung entstandenen Automaten weiterzuschalten. Die aus dem Ablauf errechneten oder resultierenden Werte und Signale werden über die Ausgänge wieder an die Hardware zurückgegeben und steuern damit deren Funktionalität.

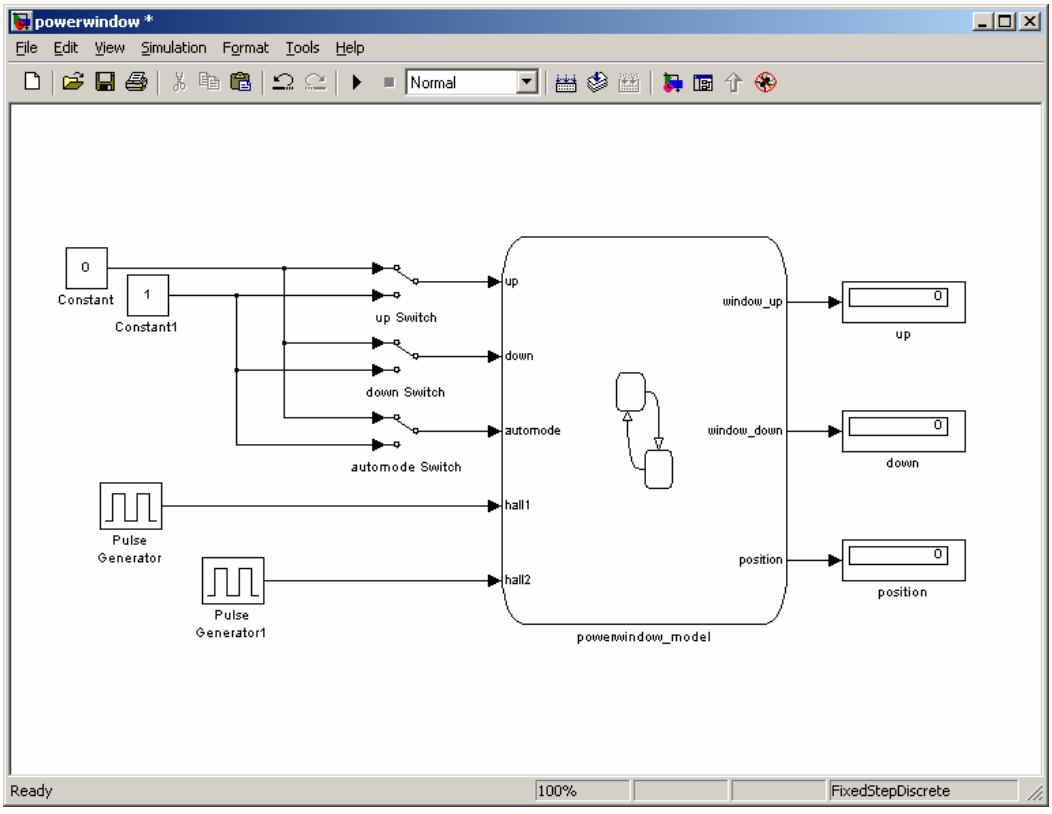

Abbildung 6.23: Ein-/Ausgabevariablen des Modells

Durch die Verwendung von Konstanten innerhalb des Modells – z.B. für Grenzen von Zählervariablen oder als Referenzwerte für Verzweigungen – kann das Verhalten des Modells leicht durch Modifizierung der Parameter an veränderte Gegebenheiten angepasst werden.

Bei der Umsetzung von Spezifikationen können auch Nebenläufigkeiten modelliert werden, die dann später als einzelne Prozesse im C-Code erscheinen. Im Modell eines Fenster-

hebers in Abbildung 6.24 sind sechs nebenläufige Prozesse enthalten. Diese werden aber der sequenziellen Arbeitsweise des Prozessors wegen nur quasi-parallel ausgeführt, was bereits bei der Modellierung beachtet werden sollte, um spätere Fehlfunktionen auszuschließen.

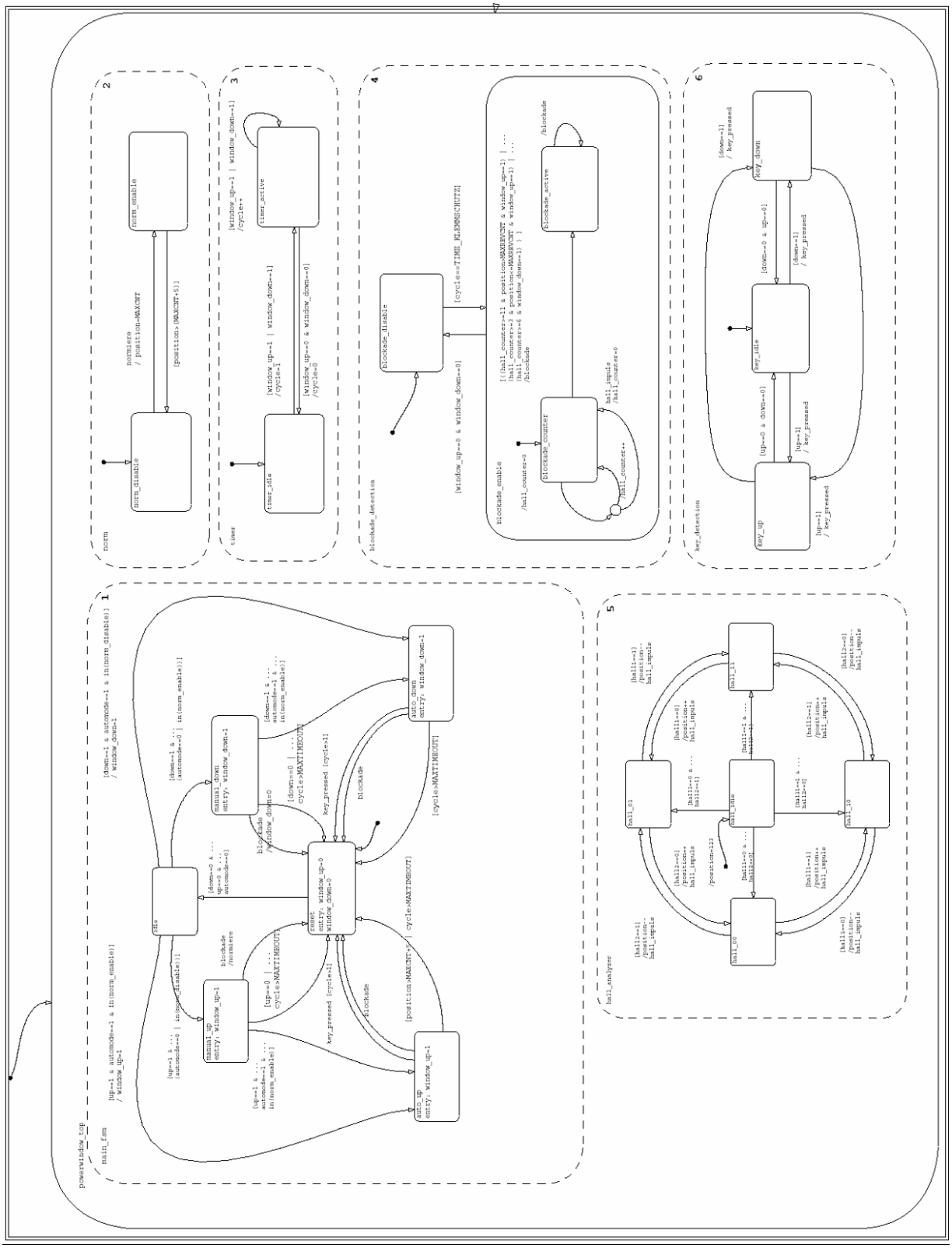

Abbildung 6.24: Beispiel für ein StateFlow-Modell mit Nebenläufigkeiten

Durch Simulation in der Simulink-Umgebung von MATLAB kann die Funktionalität überprüft und iterativ angepasst werden. Die Kontrolle auf syntaktische Richtigkeit und Konsistenz des Modells erfolgt über den im StateFlow-Editor integrierten Parser.

6.4.4.4 Code-Generierung

Steht eine funktionsfähige und getestete Umsetzung der Spezifikation zur Verfügung, kann der von MATLAB verwendete StateFlow-Coder daraus automatisch einen äquivalenten C-Code erzeugen. Bei der Verwendung des Real-Time-Workshops (RTW) werden Dateien für den Einsatz in Echtzeit-Anwendungen generiert, die neben der Beschreibung des Zustandsdiagramms weitere Funktionen zur Initialisierung und Überwachung des Modells enthalten. Sowohl die Fülle an Dateien als auch die Konvertierung der Ein- bzw. Ausgangssignale in real-Variablen erschweren die Einbindung in der Entwicklungsumgebung des LEON [Gais02].

Wesentlich einfacher und übersichtlicher gestaltet sich der Einsatz des Coders für „Stand-Alone“ Anwendungen. Sollte der LEON in der Auswahl noch nicht aufgelistet sein, so muss über die StateFlow Target Configuration eine neue Zielplattform erzeugt (Menü: Add→Target) und z.B. in LEON umbenannt werden. Damit steht das neue Target für die Generierung von ANSI-C-CODE zukünftig im Explorer zur Verfügung. Nach der Definition der Ziel-Plattform wird innerhalb des StateFlow-Editors das Zustandsmodell in einen C-Quellcode umgewandelt.

Mit einem Klick auf den Namen der Zielhardware wird das Target Configuration-Fenster geöffnet. Nach der Auswahl des Punktes Incremental Build in dem Kombinationsfeld wird die Code-Generierung mit einem Klick auf Generate Code gestartet. Die einzelnen Elemente der Statecharts werden dabei in Programmstrukturen übersetzt, die alle Zustände und deren Übergänge enthalten.

Als Resultat werden vier Code-Dateien im Projektverzeichnis erzeugt. Bei der verwendeten Vorlage sind das powerwindow_leon.c bzw. powerwindow_model.c und die gleichnamigen Header-Dateien. Letztere enthält dabei innerhalb der Funktion powerwindow_model() die Umsetzung des Modells.

6.4.4.5 Software Framework

Der aus dem Modell erzeugte Code ist allein nicht ausführbar und wird daher in ein geeignetes Software Framework integriert (Abbildung 6.25).

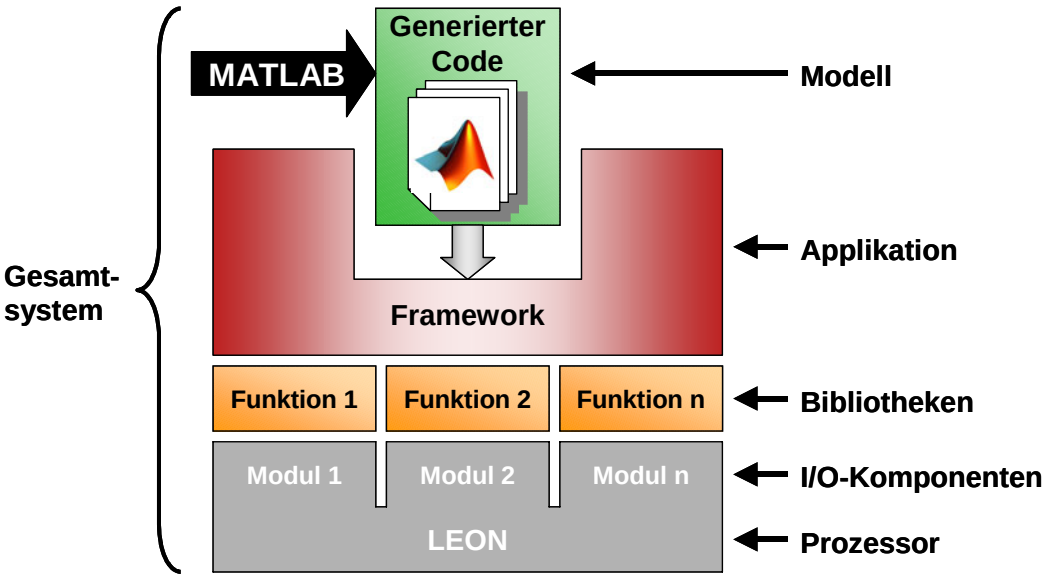

Abbildung 6.25: Aufbau der Steuer-Software

Aufgabe dieses Rahmenprogramms ist die Steuerung des Gesamtsystems. Dazu zählen die Initialisierung der im generierten Code enthaltenen Modellbeschreibung, die Ausführung bzw. Überwachung eines Modellschritts und die Anpassung der I/O-Variablen an die Signale der angeschlossenen peripheren Bauteile bzw. des Prozesses.

Die Abfrage der Eingangssignale kann auf zwei Arten realisiert werden: durch eine zyklische Abfrage oder über eine Interrupt-Auslösung. Ist im Rahmen der Steuerung der zeitliche Ablauf eher sekundär, ist eine zyklische Ausführung die beste, weil einfachere Alternative. Ist jedoch das Einlesen der Eingänge und das darüber getriggerte Weiterschalten der Zustände relevant hinsichtlich einer festgelegten zeitlichen Schrittdauer, ist die Interrupt-Variante zu empfehlen. Mit Hilfe eines Timers lassen sich in genauem zeitlichen Abstand diese Interrupts erzeugen und damit die gewünschte Zeitbasis für die Ausführung schaffen. Um während der Verarbeitung des Steuermodells eine Inkonsistenz der Daten zu vermeiden, werden die Eingänge zu Beginn eingelesen und gespeichert. Damit stehen während des gesamten Taktschrittes konstante Steuersignale fest.

Für die Kommunikation mit den internen oder externen Peripheriekomponenten greift das Framework auf Bibliotheken zurück, in denen spezifische Funktionen der entsprechenden Bausteine abgelegt sind. Damit lässt sich eine sehr gute Aufteilung nach Funktionen erreichen, die außerdem eine klare Strukturierung innerhalb der Hierarchie der Software unterstützt.

6.4.5 Modellimplementierung in Hardware

Außer der oben beschriebenen Methode, StateFlow-Modell in Software (C-Code) zu implementieren, können diese auch direkt auf eine Hardware-Struktur abgebildet werden. Dies ist allerdings nicht mit der MATLAB-Umgebung direkt möglich, da kein entsprechender Code-Generator zur Verfügung steht. Dazu bedarf es eines speziellen Tools, das in [Drei06] entwickelt wurde. Es basiert auf dem an der Universität in Berkeley entstandenen Werkzeugs SF2VHDL und nennt sich in Anlehnung an die verwendete Programmiersprache Java „JVHDLGen“. Mit ihm ist es möglich mit Simulink erzeugte Modelldateien (*.mdl) zu lesen und mit Hilfe eines Parsers deren Struktur zu scannen. Die Modellstruktur wird dann in einen äquivalenten VHDL-Code transformiert.

7 Anwendungsbeispiele und Ergebnisse

7.1 RP-Anwendungen

7.1.1 CAN2CAN-Gateway

Zur Systemintegration und dem umfassenden Test einiger während dieser Arbeit entstandenen Komponenten, wird mit der Compass-Plattform ein HiL-Szenario aufgebaut, das auf der Nutzung eines MicroBlaze-Subsystems [Xil04g] basiert. Mit Hilfe zweier CAN-Controller als Soft-IP-Cores, die an einen MicroBlaze angeschlossen werden, entsteht so ein CAN2CAN-Gateway als Subsystem auf der Plattform. Die Kopplung geschieht über den so genannten On-Chip-Peripheral-Bus (OPB) [Xil02d], der für die Anbindung auch von eigenen, selbst entwickelten Peripheriegeräten zur Verfügung steht.

Im folgenden Blockschaltbild in Abbildung 7.1 ist innerhalb des FPGAs der MicroBlaze mit seinem OPB zu sehen, an den ein UART-Core von Xilinx und zwei Can-Cores aus dem OpenCores-Projekt angeschlossen sind. Zur Parametrierung des auf dem MicroBlaze laufen Programms wird ein Dual-Port-RAM zwischen den Compass-Systembus und den OPB geschaltet. Hierzu mussten drei IP-Cores erstellt werden – ein Core mit dem Dual-Port-RAM und je einer für die Interfaces zu OPB und Systembus.

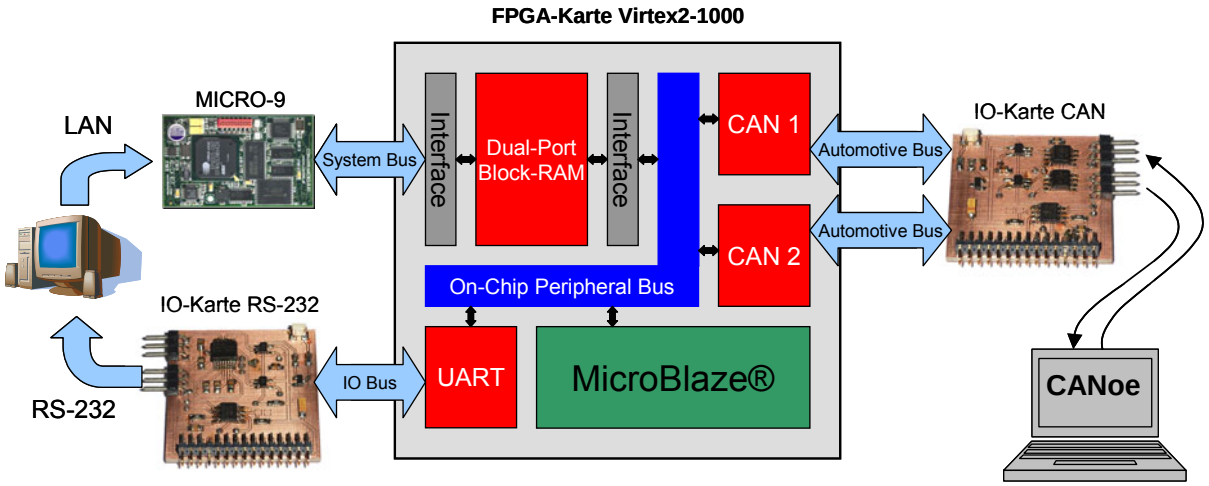

Abbildung 7.1: COMPASS-Plattform mit MicroBlaze®-Subsystem

Die Implementierung des Systems auf einem Virtex-II-1000-FPGA ist in Abbildung 7.2 als Floorplan (Belegung der CLBs auf dem Chip) zu sehen. Da zur Umsetzung nicht die Konfigurationsmethodik verwendet wurde, sieht man hier keine Slot-Struktur. Links oben und links unten befinden sich die beiden CAN-Controller, die von dem in der Mitte des Designs angeordneten Prozessor (MicroBlaze), gesteuert werden. Der UART und das DPRAM (alloziert in einer der BlockRAM-Spalten) befinden sich in der Mitte rechts.

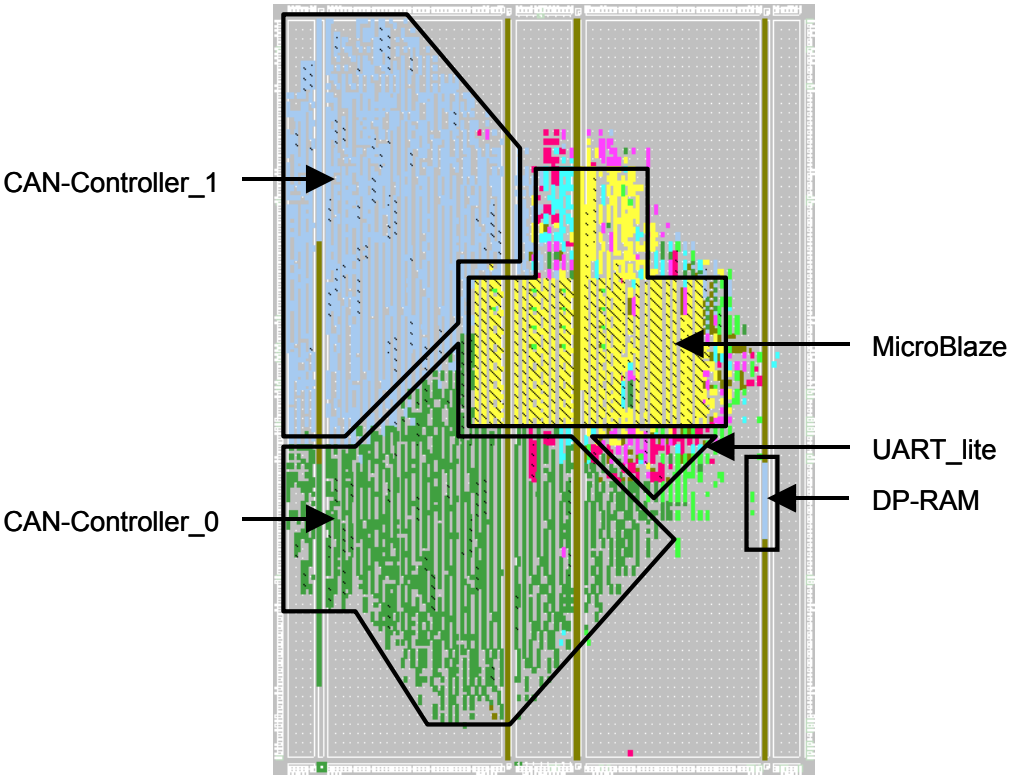

Abbildung 7.2: Floorplan des CAN2CAN-Gateways auf eine Virtex-II-1000

Durch die gegebene Anordnung der Logik ergibt sich laut Design-Tool von Xilinx eine maximale Frequenz von 57.2MHz für den Betrieb des Designs. Diese wurde aufgrund der Verfügbarkeit eines nur mit 50MHz schwingenden Oszillators auf der Karte mit dessen Takt betrieben. Die Ressourcenbelegung der Implementierung ist in Tabelle 7.1 aufgelistet.

FPGA-Ressourcen	Belegung	
	absolut	prozentual
External IOBs	46 von 172	26%
MULT18X18s	3 von 40	7%
RAMB16s	39 von 40	97%
SLICEs	2999 von 5120	58%
BUFGMUXs	2 von 16	12%
DCMs	1 von 8	12%
TBUFs	16 von 2560	1%

Tabelle 7.1: Ressourcenbelegung des CAN2CAN-Gateways

In dieser Testanwendung kommen auch die während dieser Dissertation entwickelten I/O- bzw. Bus-Karten (siehe Kapitel 4.4.8 und 4.4.9) zum Einsatz. Die RS-232-Karte dient zur Anbindung der seriellen Konsole des MicroBlaze-Prozessors an den Steuerungs-PC. Somit kann während des Betriebs die Debug-Ausgabe des ablaufenden Programms über ein Terminalprogramm auf dem PC sichtbar gemacht werden. Auch eine interaktive Programmsteuerung lässt sich hiermit realisieren, was aber nicht umgesetzt wurde. Mit der Java-Anwendung CompassControl können – wie in Kapitel 6.3 beschrieben – einzelne Schreib- und Lesezugriffe auf den Systembus ausgeführt werden. Es ist also möglich, vom Steuerungs-PC aus lesend und schreibend auf das Dual-Port-RAM des FPGAs zuzugreifen, um Einstellungen des Gateways zu ändern.

Da das hier aufgebaute HiL-Szenario die Verbindung von zwei verschiedenen CAN-Netzwerken herstellen soll, sind auch zwei CAN-Transceiver nötig, die auf einer der bereits gebauten I/O-Karten verfügbar sind. Da die in das FPGA-Design integrierten CAN-Cores völlig unabhängig voneinander sind, ist es sogar möglich, beide CAN-Netzwerke mit unterschiedlicher Baudrate zu fahren. Sofern die Datenrate des Nachrichtenaufkommens die Geschwindigkeit des langsameren der beiden CAN-Netzwerke nicht überschreitet, kann die Software auf dem MicroBlaze ihre Aufgabe erfüllen, alle Nachrichten jedes CAN-Busses auf den jeweils anderen zu spiegeln. Auch die Manipulation von Daten oder IDs einzelner Nachrichten ist dabei möglich.

Zur Verifikation der Funktionalität dieses Versuchsaufbaus diente das Tool CANoe der Firma Vector Informatik [Vect03], in Kombination mit einer PCMCIA-CAN-Karte mit zwei Kanälen. Dazu wurden parallel zwei Can-Busse angeschlossen und gesteuert, wobei die maximal auf einem CAN-Bus mögliche Baudrate von 1Mbit/s erreicht werden konnte. Mit Hilfe des Tools CANoe war es so zum Beispiel möglich, periodisch bestimmte CAN-Nachrichten auf einem Kanal zu generieren und an das CAN2CAN-Gateway zu schicken. Die Daten wurden vom CAN2CAN-Gateway entweder direkt auf den anderen Kanal gespiegelt (Variante 1) oder als leicht modifiziert Nachrichten (z.B. Änderung der ID oder der Daten) zurückgeschickt (Variante 2). Die Anzeige der Ergebnisse erfolgte in beiden Fällen in CANoe.

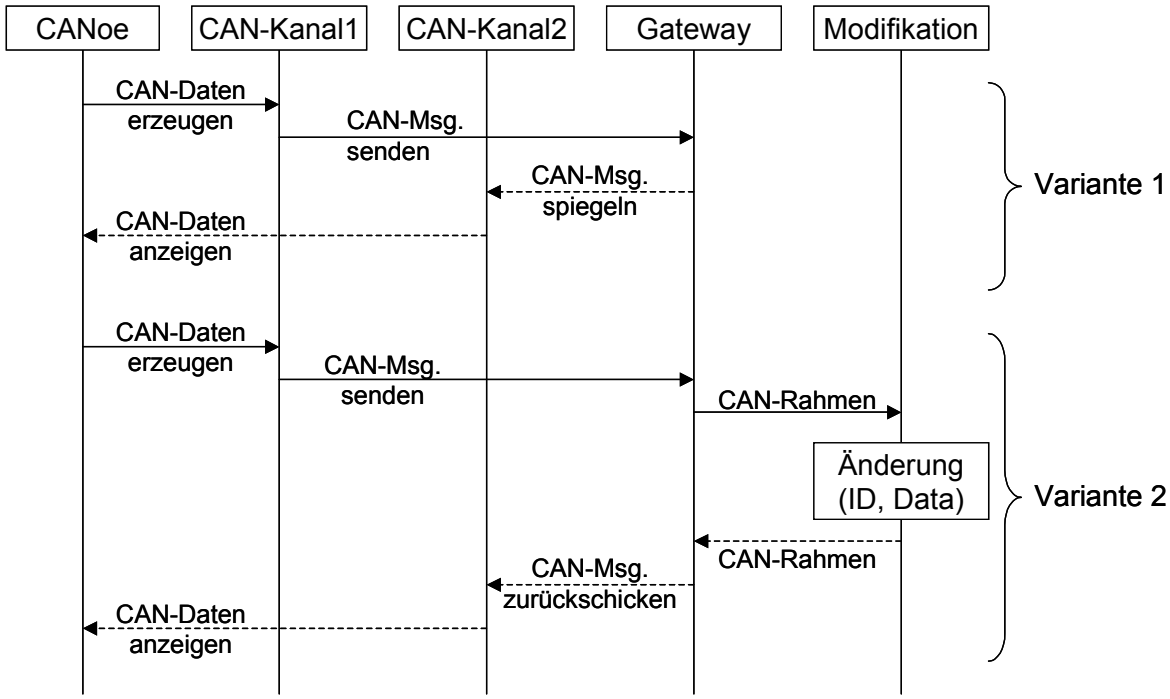

Abbildung 7.3: Testszenarien des CAN2CAN-Gateways als MSC

Da dieser Versuchsaufbau alle Teilkomponenten der COMPASS-Plattform-Software benötigt, kann dies als Systemtest verstanden werden. Die Möglichkeit, das oben vorgestellte HiL-Szenario über die einzelnen Software-Komponenten auf die Plattform zu laden, die Prozessankopplung über entwickelte I/O-Karten und schließlich die Möglichkeit der Steuerung des Versuchsaufbaus über die graphische Benutzeroberfläche, zeigt die Funktionalität und die Vollständigkeit des Gesamtsystems.

Durch den Austausch eines CAN-Interfaces in dem obigen Testszenario (siehe Abbildung 7.1), entstünde aus der CAN-CAN-Bridge ein CAN-LIN-Gateway. Da es sich bei der Vorgehensweise und der Implementierung um die gleichen Schritte handelt, wird an dieser Stelle nicht weiter auf diese Variante eingegangen. Es soll an der Stelle nur noch einmal deutlich werden, wie ohne großen Aufwand, die Interface-Struktur einer Implementierung ohne Hardware-Änderungen modifiziert werden kann.

7.1.2 NoC-Testszenario

Networks-on-Chip werden derzeit als eine zukünftige Alternative zu den heutigen Verbindungsstrukturen, die auf Bus-Architekturen basieren, auf einem Chip betrachtet. Die Entwicklung eines parametrierbaren Packet-Switched-Routers für Networks-on-Chip ist in [Wang06] beschrieben.

Um die Effizienz des Konzepts zu überprüfen, muss mehr als nur ein Router getestet werden, um Effekte durch deren Interaktion zu identifizieren und zu bewerten. Daher ist es sinnvoll ein Netzwerk aufzubauen, in dem Daten verschickt werden. Das kann grundsätzlich simulativ geschehen, verursacht aber mit steigender Komplexität auch enorme Datenmengen und einen erheblichen Zeitaufwand. Da bei einer solchen Untersuchung lediglich das logische Verhalten relevant ist und technologische Informationen wie der Leistungsbedarf zunächst nicht betrachtet werden, bietet sich eine Hardware-beschleunigte Simulation, d.h. eine Emulation an. Außerdem lässt sich der Zeitfaktor erheblich reduzieren. Zu diesem Zweck sollte ein Netzwerk auf der COMPASS-Plattform implementiert und getestet werden.

Zur Abschätzung des Flächenbedarfs eines ganzen On-Chip-Netzwerks wurde zunächst nur ein Router auf dem FPGA implementiert. Der Logikbedarf ist in Tabelle 7.2 dargestellt. Laut Synthese-Tool könnte ein solcher Router mit einer Frequenz von 246,8 MHz betrieben werden.

FPGA-Ressourcen	Belegung	
	absolut	prozentual
Slices:	536 von 5120	10%
Slice Flip Flops:	411 von 10240	4%
4 input LUTs:	916 von 10240	8%
GCLKs:	2 von 16	12%

Tabelle 7.2: Ressourcenbelegung eines Routers auf einem Virtex-II-1000

Unter Verwendung dieser Werte als Basis einer Berechnung wäre es theoretisch möglich, 10 Router auf einem FPGA unterzubringen. Jedoch ist zu beachten, dass auch genügend Routing-Ressourcen zur Verfügung stehen, um diese miteinander zu verbinden. Außerdem fehlen neben den Routern auch noch Interfaces, um Daten in das NoC einzukoppeln, so dass auch eine Abschätzung hinsichtlich des Ressourcenbedarfs eines Interfaces nötig ist (Tabelle 7.3).

FPGA-Ressourcen	Belegung	
	absolut	prozentual
Slices	144 von 5120	2%
Slice Flip Flops	137 von 10240	1%
4 input LUTs	265 von 10240	2%
TBUFs	16 von 2560	0%
BRAMs	4 von 40	10%
GCLKs	2 von 16	12%
DCMs	1 von 8	12%

Tabelle 7.3: Ressourcenbelegung eines NoC-Interfaces auf einem Virtex-II-1000

Um ein realistisches Netzwerk zu testen, brauchen alle Router kleine IP-Blöcke, die Daten annehmen und weiterleiten können. Dies wurde durch einfache Operationen auf den Datenpaketen realisiert, die keineswegs eine reale Applikation darstellen. Trotz allem war der Logikbedarf des Testnetzes unter Berücksichtigung der obigen Abschätzungen so groß, dass eine Verteilung auf drei FPGAs vorgenommen werden musste. Abbildung 7.4 zeigt das Gesamtnetzwerk, das auf der Plattform getestet wurde. Für die Kommunikation zwischen den Routern über die Chip-Grenzen hinweg wurden die Routing-Kanäle beibehalten. Als Medium der Übertragung dient dabei der Modul Interconnect Bus (MIB), der unabhängig vom Prozessor die Interface-Karten miteinander verbindet. Dieser wurde in entsprechend identische Segmente aufgeteilt, um den Durchsatz für alle Kanäle gleich gestalten zu können.

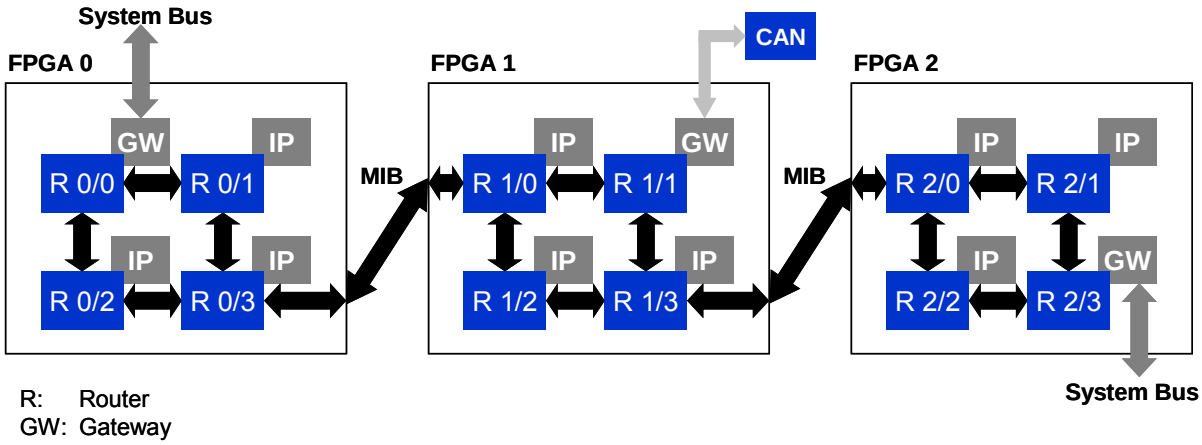

Abbildung 7.4: Network-on-Chip zum Test der Router-Funktionalität und –effektivität

Um die Daten von außen in das Netzwerk einspeisen und umgekehrt auch wieder auslesen zu können, wurde ein entsprechenden Interfaces (Abbildung 7.5) entworfen. Dieses ist auf der einen Seite mit dem externen Datenbus verbunden und auf der anderen mit einem Router, über den der Datenstrom ein- oder ausgekoppelt werden kann. Da die Interface-Karten sowohl an dem MIB als auch an dem System Bus (SB) angeschlossen sind, lässt sich die Schnittstelle recht einfach integrieren. Über diesen Weg wäre es auch möglich, das Router-Verhalten zur Laufzeit zu ändern, was aber in diese Version nicht implementiert ist.

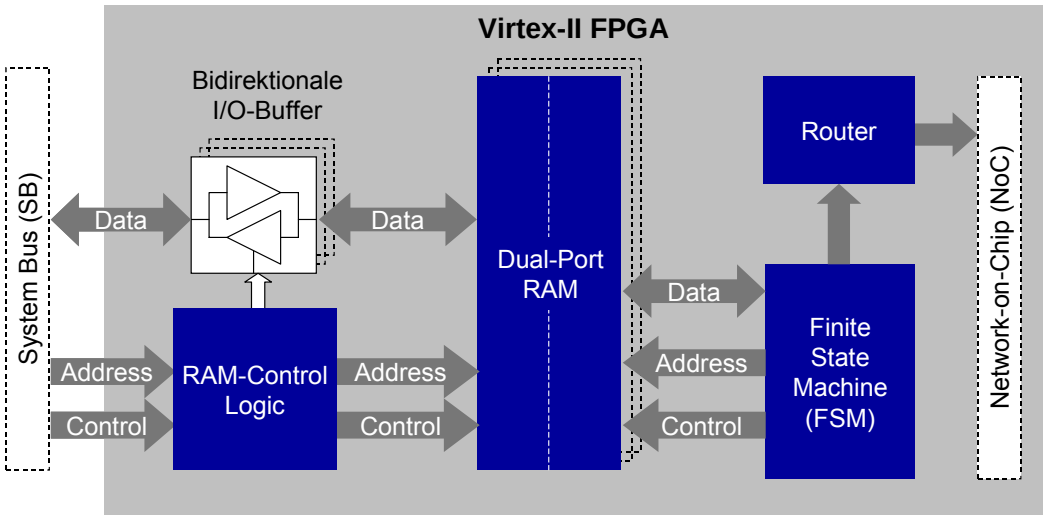

Abbildung 7.5: Interface zum Ein- und Auskoppeln von Daten in das bzw. aus dem NoC

Das Gateway, um auf das Netzwerk zugreifen zu können, ist in Abbildung 7.5 zu sehen. Als Zwischenspeicher für die Daten wird ein Dual-Port-RAM eingesetzt, das primärseitig mit Hilfe einer kombinatorischen Logik an den Systembus und sekundärseitig an einen Router angeschlossen ist. Die Weitergabe der Daten wird durch eine separate Finite State Machine kontrolliert, um einen hohen Durchsatz zu erreichen und den laufenden Netzwerkverkehr nicht zu stören, bzw. Datenverluste zu verhindern. Das Auslesen der Daten von Seiten des Prozessors geschieht auf ähnliche Weise, indem der Router über das RAM mitteilt, dass gültige Daten anliegen und abgeholt werden können.

Das Gesamt-NoC wurde letztlich wie in Abbildung 7.4 gezeigt auf drei FPGAs verteilt implementiert – jedoch ohne die Verwendung des dort dargestellten CAN-Gateways. Der durchschnittliche Maximaltakt aller Designs lag nach Angaben des Synthese-Tools bei ca. 110 MHz, so dass zentral eine Clock von 100 MHz eingestellt und zum Test verwendet wurde. Die auf einem FPGA okkupierten Ressourcen sind in Tabelle 7.4 gelistet.

FPGA-Ressourcen	Belegung	
	absolut	prozentual
Slices:	3431 von 5120	67%
Slice Flip Flops:	1843 von 10240	18%
4 input LUTs:	3585 von 10240	35%
GCLKs:	2 von 16	12%

Tabelle 7.4: Ressourcenbelegung des NoCs auf einem Virtex-II-1000

7.2 HiL-Anwendungen

7.2.1 LIN-Bus-Tester

In dem hier vorgestellten Projekt wird der LIN-Core zur Fehlerinjektion, wie er bereits in Kapitel 5.6.3 beschrieben wurde, unter Verwendung der COMPASS-Plattform eingesetzt. Mit Hilfe der Modifikationsmöglichkeiten, die der Core bietet, wurden umfassende Tests durchgeführt, wie die Manipulation der einzelnen Bitzeiten, das Verfälschen des Nachrichteninhalts und der CRC-Checksumme, das Einbauen von nicht spezifikationskonformen Protokollfehlern, Vergabe von unterschiedlichen Zykluszeiten pro Bit, usw. Das physikalische Inter-

face (Transceiver) zum eigentlichen LIN-Bus wurde als Automotive-Interface-Karte ausgeführt und über den Peripheral Communication Bus der Plattform an den FPGA angekoppelt. Als fertige LIN-Transceiver-Lösung kam ein AT6667 von Atmel zum Einsatz.

Die Ansteuerung des LIN-Cores wird über einen MicroBlaze IP-Mikrocontroller-Core von Xilinx realisiert, der über das ebenfalls auf dem FPGA implementierte parallele Interface für den System Bus (SB) mit dem Hauptprozessor der Plattform kommuniziert und damit als echtes Subsystem innerhalb der Plattform agiert. Das Subsystem nimmt Daten zum Versenden an, bzw. stellt empfangene dem Hauptprozessor zur Verfügung. Auch die Instrumentierung des LIN-Fehlerinjektors wird über diese Schnittstelle abgewickelt. Der MicroBlaze übernimmt dann die interne Datenverwaltung und Koordination der LIN-Test-Core-Ansteuerung durch Beschreiben einzelner Register im LIN-Core. Die Programmierung des IP-Mikrocontrollers erfolgte in C und wurde mit dem in der EDK von Xilinx [XEDK04] enthaltenen GNU-C-Compiler erledigt.

7.2.1.1 Testszenarien

Die Manipulation der LIN-Frames und deren Auswirkungen wurden durch Anschluss an ein kommerzielles Test-Tool (CANoe/DENoe von Vector Informatik) getestet und damit die einzelnen Fehlerfälle verifiziert.

Senden eines LIN-Frames

Zunächst wird der LIN-Core im Master-Mode getestet, indem ein kompletter LIN-Frame gesendet wird. Im Fenster Trace wird genau angezeigt, ob und welche Daten empfangen werden konnten bzw. welche Fehler im LIN-Protokoll während der Übertragung aufgetreten sind. Es wurden verschiedene Testdatensätze und Einstellungen ausprobiert, die im Folgenden näher erläutert werden. Abbildung 7.6 zeigt einen LIN-Frame mit vier Datenbytes und Checksumme, der von dem LIN-Core auf den Bus gesendet wurde.

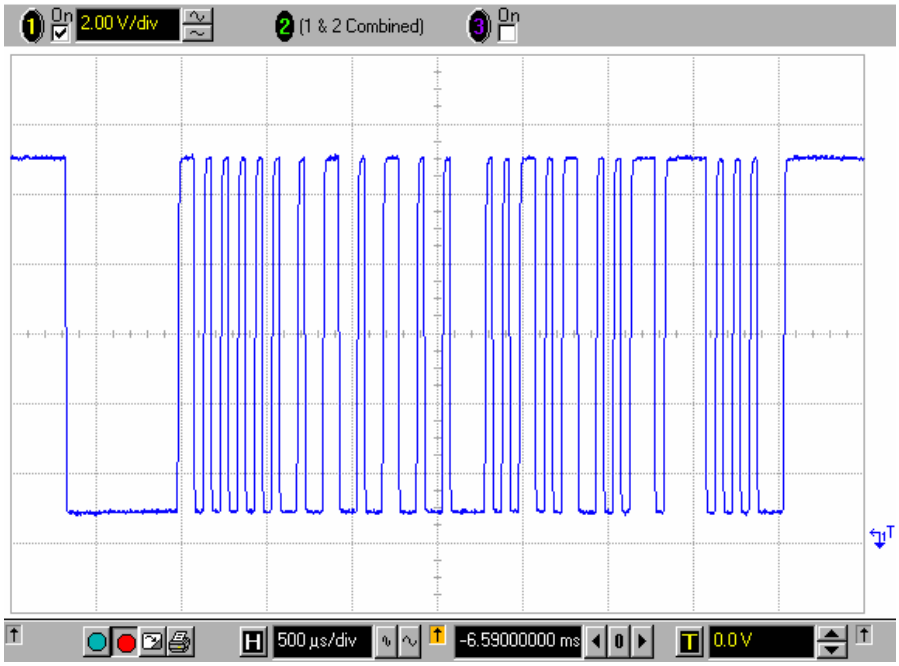

Abbildung 7.6: LIN-Frame mit vier Datenbytes und Checksumme

Spezifikationskonformer Frame

Beim Senden eines spezifikationskonformen Frames treten keine Fehler auf. Dabei werden verschiedene Frames unterschiedlicher Länge und mit verschiedenen Datenbytes ausprobiert. Als Übertragungsraten wurden 20 kbit/s und 9,6 kbit/s gewählt. Abbildung 7.7 zeigt das Trace-Fenster in CANoe.LIN nach dem Empfang eines spezifikationskonformen LIN-Frames.

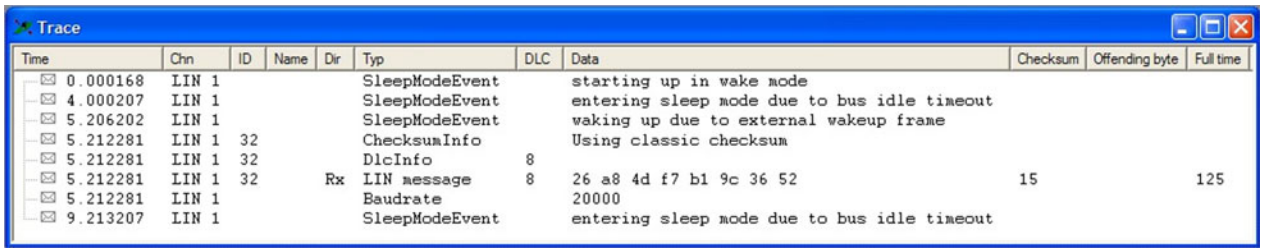

Abbildung 7.7: Trace-Fenster in CANoe.LIN nach Datenempfang

Manipulation von Synchronisationsbreak und Delimiter

Das Synchronisationsbreak wird nun mit verschiedenen Methoden manipuliert, die Übertragungsrate beträgt 20 kbit/s (Standardbitbreite von 50µs).

1. Als Synchronisationsbreak wird der Wert 0x10 übertragen, das heißt, es tritt ein high-Pegel mit der Dauer einer Bitbreite ungefähr in der Mitte des Synchronisationsbreaks auf.

   Ergebnis: Das Synchronisationsbreak wird nicht erkannt, bei dem high-Pegel wird der Synchronisationsvorgang abgebrochen, jeder darauf folgende low-Pegel wird erneut als Start eines Synchronisationsbreaks aufgefasst.

2. Die Breite des high-Pegels im Synchronisationsbreak wird sukzessive verkleinert auf 40µs, 30µs, 20µs und 10µs.

   Ergebnis: Das Break wird erst bei einem kurzen Peak von 10µs wieder richtig erkannt, allerdings hat man hier aufgrund der begrenzten Flankensteilheit auch keinen eindeutigen high-Pegel mehr.

3. Das Synchronisationsbreak wird verlängert auf den hier maximal möglichen Wert (ca. 1,13ms).

   Ergebnis: Die Übertragung ist korrekt, die Header Time verlängert sich auf 43 Bitzeiten.

4. Verkürzung des Synchronisationsbreaks auf 310µs, 400µs und 500µs.

   Ergebnis: Das Break wird nicht erkannt. Erst ab einer Dauer von 550µs (11 Bitzeiten) wird das Synchronisationsbreak richtig detektiert.

5. Die Breite des Delimiters wird verändert.

   Ergebnis: Ab einer Verkürzung des Delimiters auf 30% einer Bitzeit werden die folgenden Felder wie Identifier und Datenbytes nicht mehr korrekt zugeordnet, es treten Übertragungsfehler auf. Eine Verlängerung des Delimiters stellt kein Problem dar, sogar bei der Länge von 250% einer Bitbreite bleibt die Übertragung korrekt.

Im Folgenden wird die Übertragungsrate auf 9,6 kbit/s reduziert.

1. Senden des Wertes 0x10 für das Synchronisationsbreak.

Ergebnis: Break wird nicht erkannt, auch bei Verkleinerung des high-Pegels auf 20% der Standardbitbreite ändert sich daran nichts.

2. Der Breite des Delimiters wird in Schritten auf 80%, 60%, 40% und 20% verkürzt. **Ergebnis:** Der Frame wird korrekt detektiert.

Manipulation des Synchronisationsfeldes

Das Synchronisationsfeld wird verändert. Die Übertragungsrate wird zunächst auf 20 kbit/s eingestellt.

1. Die Breite des dritten Bits nach dem Startbit (high-Pegel) wird in Schritten von 20% variiert.

   Ergebnis: Bei 20% und 40% der Standardbitbreite funktioniert die Übertragung nicht, CANoe erkennt einen SyncError. Bei einer Breite von 60% wird der Frame richtig erkannt, die automatische Baudratenerkennung zeigt einen Wert von 20833 an, die Header Time liegt bei 36. Bei 80% verläuft die Übertragung ebenso korrekt wie bei 120%, hier wird eine Baudrate von 19607 detektiert. Bei 140% und 160% tritt ein RcvError auf, die Daten können nicht mehr richtig empfangen werden. Ab einer Bitbreite von 180% ist die Synchronisation gestört, dies wird durch einen SyncError angezeigt.

2. Für den Wert des Synchronisationsfeldes (normalerweise ein 0x55 -Datenbyte) werden nacheinander die Werte 0x95, 0x6C und 0x45 eingesetzt, die dem Wert 0x55 in ihrer Struktur ähnlich sind.

   Ergebnis: In keinem Fall funktioniert die Synchronisation.

3. Für das Synchronisationsfeld wird wieder der Standardwert 0x55 eingesetzt, alle Bitzeiten des Synchronisatonsfeldes werden um den gleichen Wert verändert.

   Ergebnis: Bei einer Verkürzung aller Bits auf 60% bzw. 80% funktioniert die Synchronisation nicht, es tritt ein „SyncError“ auf. Bei Bitbreiten von 90% kommt es zu einen „RcvError: illegal character while waiting for LIN id“, das Identifier-Feld kann nicht empfangen werden. Die Übertragung gelingt erst ab einem Wert von 96%, hier wird zusätzlich eine Baudrate von 20833 erkannt. Bei Verlängerung der Bits passiert ab einer Länge von 104% wieder ein „RcvError: framing error while waiting for checksum“, dieser macht sich durch die minimale Verschiebung aber erst beim Empfang der Checksumme bemerkbar. Bei einer Breite von 110% wird ein „RcvError: framing error while waiting for LIN id“ erkannt, hier bricht die Übertragung schon beim Empfang des Identifiers ab.

Die Übertragungsrate wird wieder auf 9,6 kbit/s umgestellt.

1. Als Synchronisationsfeld wird der Wert 0x56 und 0xA4 gesendet.

   Ergebnis: „SyncError“, die Übertragung funktioniert nicht.

2. Die Länge des dritten Datenbits des Synchronisationsfeldes wird in Schritten von 20% verändert.

   Ergebnis: Bei einer Breite von 80% verläuft die Übertragung fehlerfrei, es wird eine Baudrate von 9900 erkannt. Ebenso fehlerfrei funktioniert die Übertragung bei einer Breite von 60%, wohingegen bei 40% ein „SyncError“ auftritt.

3. Zusätzlich zu dem dritten Datenbit wird das sechste Datenbit in der Länge in entgegen gesetzter Richtung geändert.

   Ergebnis: Hat das dritte Datenbit eine Breite von 40% und das sechste Datenbit eine Breite von 160%, tritt ein „SyncError“ auf. Bei einem Verhältnis von 60% zu

140% wird der Frame korrekt erkannt, als Baudrate wird 9615 angezeigt, was exakt der eingestellten Übertragungsrate entspricht, obwohl das Synchronisationsfeld nicht symmetrisch ist.

4. Das Startbit (low-Pegel) des Synchronisationsfeldes wird in der Breite variiert.

   Ergebnis: Bei einer Breite von 80% bzw. 120% gibt es keine Fehler, es wird eine Baudrate von 9900 bzw. 9345 erkannt. Bei einer Breite von 140% kommt es zu einem „RcvError: framing error while waiting for LIN id“.

5. Das Stopbit (high-Pegel) wird in der Breite manipuliert.

   Ergebnis: Eine Verlängerung des Stopbits des Synchronisationsfeldes hat keinen Einfluss auf den Empfang des Frames, selbst bei einer Breite von 250% treten keine Fehler auf.

Abbildung 7.8 zeigt das Trace-Fenster nach dem Empfang eines LIN-Frames mit einem fehlerhaften Synchronisationsfeld. Der „SyncError“ wird in der Spalte „Typ“ angezeigt.

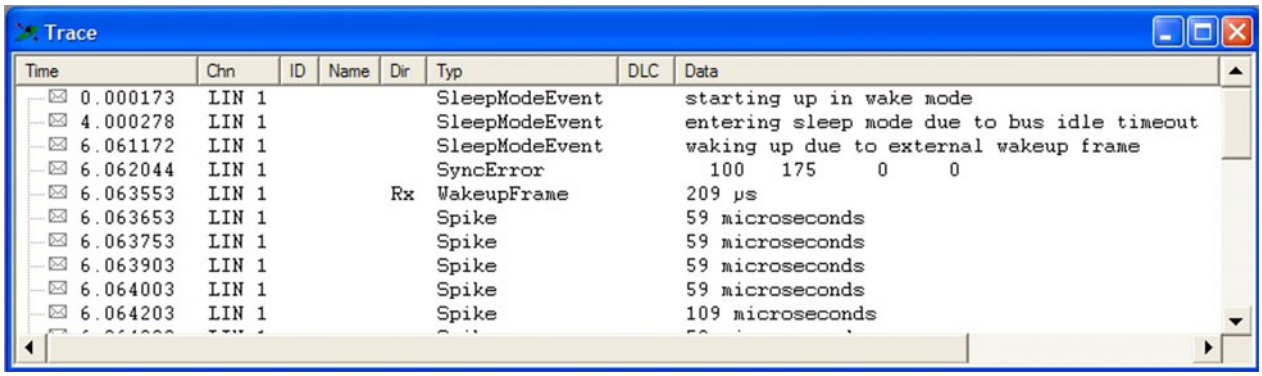

Abbildung 7.8: Empfang eines LIN-Frames mit fehlerhaftem Synchronisationsfeld

Manipulation des Identifiers und der Datenbytes

Es werden verschiedene Tests bei einer Übertragungsrate von 20 kbit/s durchgeführt.

1. Das Startbit des Datenbytes 3 wird in der Breite manipuliert.

   Ergebnis: Selbst bei einer Verkürzung auf 60% bzw. einer Verlängerung auf 140% funktioniert die Übertragung korrekt, alle Datenbytes werden richtig erkannt.

Die Übertragungsrate wird auf 9,6 kbit/s reduziert.

1. Das dritte Datenbit des Identifiers wird in der Länge variiert.

   Ergebnis: Bei einer Breite von 80% treten keine Fehler auf, bei einer weiteren Reduzierung der Breite auf 60% wird ein „RcvError: framing error while waiting for LIN id“ angezeigt. Wird die Breite des Bits auf 120% bzw. 140% erhöht, wird kein Fehler detektiert, erst ab einer Breite von 160% wird der Fehler „RcvError: framing error while waiting for LIN id“ erkannt.

2. Das Stopbit des Identifiers wird auf den maximalen Wert von 250% ausgedehnt.

   Ergebnis: Es treten keine Fehler auf.

3. Das Datenbit 5 des Datenbytes 2 und das Datenbit 2 des Datenbytes 7 werden in der Breite variiert.

Ergebnis: Wird die Breite der beiden Bits auf 80% bzw 60% reduziert, findet eine korrekte Übertragung statt. Dies gilt ebenso für die Verlängerung auf 120% und 140%. Bei einer Breite von 40% tritt ein „RcvError: framing error while waiting for data byte 2“ auf, die Abweichung der Bitbreiten führt schon bei dem Empfang des zweiten Datenbytes zu einem Fehler. Bei Bitbreiten von 160% wird der Fehler „RcvError: framing error while waiting for data byte 7“ erkannt, hier macht sich der Fehler erst beim Empfang des siebten Datenbytes bemerkbar.

4. Das Datenbit 5 des Datenbytes 2 wird auf die Standardbitbreite von 100% zurückgesetzt, das Datenbit 2 des Datenbytes 7 wird nacheinander auf 160%, 180%, 200% und 250% eingestellt.

   Ergebnis: In allen vier Fällen wird ein „CSError“ detektiert, außerdem wird statt dem Wert 0xA8 für das Datenbyte 2 der Wert 0x48 empfangen.

Manipulation der Checksumme

Wird der Wert der übertragenen Checksumme verändert, so führt dies auf jeden Fall zu einem Checksummenfehler (CSError). Dieses Feld muss also auf jeden Fall den richtigen Wert besitzen, um eine korrekte Übertragung zu ermöglichen.

Weitere Tests

- Alle Start- und Stopbits werden vertauscht, das Register tx_bits hat den Wert 0x555555.
  Ergebnis: Die Übertragung funktioniert nicht, ein „SyncError“ tritt auf, was bedeutet, dass die Übertragung bereits bei der Synchronisation fehlschlägt.
- Die Start- und Stopbits des Synchronisationsbreaks und des Synchronisationsfeldes werden wieder korrigiert, alle übrigen Felder behalten die falschen Werte bei, das Register tx_bits entspricht also dem Wert 0x55555A.
  Ergebnis: Die Übertragung schlägt fehl, ein „RcvError: illegal character while waiting for LIN id“ wird angezeigt.
- Das Register tx_bits hat nun den Wert 0x55556A, die Start- und Stopbits sind also für den Header richtig eingestellt, für die Datenbytes und die Checksumme bleiben die Werte vertauscht.
  Ergebnis: Der Header wird richtig erkannt, danach kommt es zu einem „RcvError: framing error while waiting for checksum“.
- Im Register tx_bits wird der Wert 0xAAAA8A und 0xAAAAAA eingestellt.
  Ergebnis: Es tritt ein „RcvError: framing error while waiting for LIN id“ auf.

7.2.1.2 Empfangen eines LIN-Frames

Senden eines Headers - Empfangen des Response

Der LIN-Core wird im Master-Task betrieben, er sendet einen Header mit Synchronisationsbreak, Synchronisationsfeld und Identifier auf den Bus, die Testsoftware erkennt den Header und antwortet mit einem Response, die Daten können dabei individuell konfiguriert werden. Die empfangenen Datenbytes und die Checksumme stehen danach in den entsprechenden Registern zur Verfügung und können im Hyperterminal ausgegeben werden. Das Softwareprojekt für diese Prozedur ist unter der Bezeichnung Master Task Receive in Xilinx Platform Studio zu finden.

Empfangen eines Headers - Senden des Response

Der LIN-Core arbeitet im Slave-Task. Erkennt der LIN-Core einen Header auf dem Bus, so schickt der LIN-Core ein Response-Feld zurück, die Sendedaten können dabei beliebig gewählt werden. Der empfangene Header wird in den entsprechenden Registern gespei-

chert. Der Name des entsprechenden Softwareprojekts in Xilinx Platform Studio lautet Slave Task Send.

7.2.1.3 Bewertung der Testergebnisse

Die Testergebnisse zeigen, dass es durchaus möglich ist, die Bitbreiten in einem gewissen Rahmen zu variieren, ohne dass dadurch die korrekte Übertragung eines LIN-Frames beeinträchtigt wird. Dies trifft auch auf das Synchronisationsfeld zu, das für die Synchronisation zwischen Master und Slave verantwortlich ist. Eine Änderung der Daten des Synchronisationsfeldes führt allerdings zu einem Übertragungsfehler. Das Synchronisationsbreak muss auf jeden Fall mindestens die Länge von 11 Bitzeiten aufweisen, ansonsten wird es nicht erkannt. Die Breite des Break-Delimiters darf eine bestimmte Grenze nicht unterschreiten, eine Verlängerung des Break-Delimiters hat dagegen keinen Einfluss auf die Übertragung. Werden die Daten des Identifiers geändert, so erwartet der Empfänger unter Umständen eine andere Anzahl an Datenbytes, was zu einem Übertragungsfehler führen kann. Das Verändern der Bitbreite eines beliebigen Datenbits führt zu einer Verschiebung, so dass der Empfänger nicht mehr in der Lage ist, die Daten der LIN-Botschaft richtig zu erkennen. Dies führt am Schluss des Frames meistens zu einem zusätzlichen Checksummenfehler, da die Checksumme ebenfalls falsch detektiert wird und nicht mehr zu den übertragenen Datenbytes passt. Das Vertauschen der Werte der Start- und Stopbits (high-Pegel des Startbits, low-Pegel des Stopbits) führt unweigerlich zu einem Übertragungsfehler, da die Struktur des LIN-Frames für den Empfänger nicht mehr zu erkennen ist. Diese Manipulation ist deshalb wenig sinnvoll. Die Manipulation der Checksumme resultiert in einem Checksummenfehler. Zusammenfassend kann man sagen, dass für die Manipulation des LIN-Protokolls zu Testzwecken vor allem die Änderung der einzelnen Bitbreiten sehr interessant ist. Der LIN-Core wird als Empfänger eines LIN-Frames konfiguriert, über die Testsoftware wird ein LIN-Frame auf den Bus gelegt. Der LIN-Core empfängt alle Felder korrekt, die empfangenen Werte stehen nach dem Empfang in den entsprechenden Registern.

7.2.2 Doppel-MicroBlaze-Subsystem

Wie bereits in Kapitel 6.4.1 beschrieben, lassen sich regelungstechnische oder kontinuierliche Systeme auf zwei verschiedene Arten auf einem FPGA integrieren. Eine Möglichkeit dabei ist, aus dem Modell eine Hardware-Beschreibung (VHDL oder Verilog) zu erzeugen, die dann nach einem Syntheselauf in eine Hardware-Struktur gebracht, auf einen FPGA geladen werden kann. Die in dem Modell vorhandenen Blöcke werden dabei komplett in Hardware verwandelt. Die zweite Möglichkeit (siehe Abbildung 7.9) besteht darin, aus dem erstellten Modell C-Code zu erzeugen, der mit Hilfe eines Compilers in einen Object-Code transformiert wird und so auf einem IP-Core-Prozessor ausgeführt werden kann.

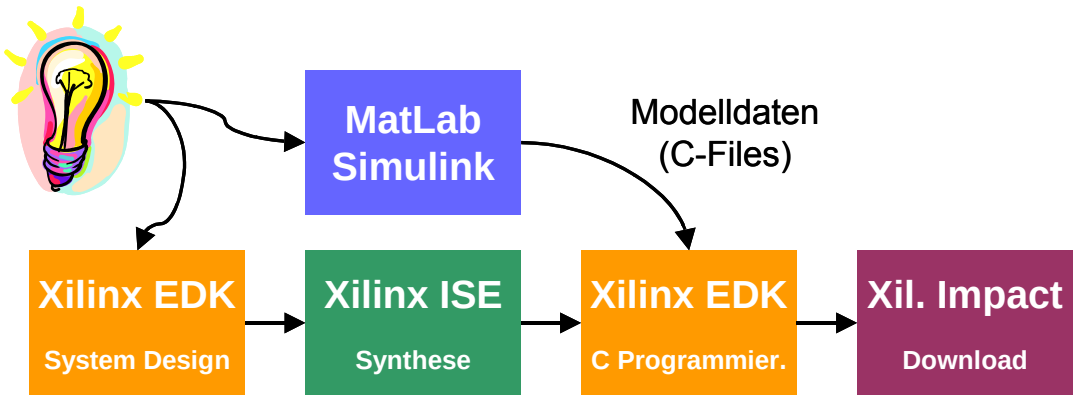

Abbildung 7.9: Designablauf für regelungstechnische Systeme auf FPGA-Basis

Zur Vereinfachung der Anbindung von Ein- und Ausgabeschnittstellen wird dazu meist ein sog. Software-Framework verwendet. Es stellt die Datenein- und -ausgabe sicher und liefert die Prozesswerte über definierte Kanäle an das Modell. Das eigentliche Modell wird bei dieser Variante als Software auf dem Prozessor ausgeführt. Dazu muss allerdings eine entsprechende Hardware-Umgebung, d.h. ein IP-Softcore-Prozessor und die nötige Peripherie bereits vorhanden sein.

Der Vorteil bei der zweiten Lösung besteht darin, dass eine Änderung des Modells weniger Aufwand zur erneuten Ausführung erfordert, als bei der ersten, da in diesem Fall nur der C-Code aus dem Modell erzeugt und dieser inklusive des Frameworks kompiliert werden muss.

Abbildung 7.10 zeigt die schematische Darstellung einer möglichen FPGA-Implementierung, die zur Ausführung von MATLAB/Simulink-Modell-Code auf Mikroprozessorbasis entwickelt wurde. Dies entspricht der oben dargelegten Lösungsalternative zwei und dem in Abbildung 7.9 gezeigten Designablauf. Sie besteht aus zwei getrennten Xilinx MicroBlaze-Prozessorsystemen, die über ein DP-RAM miteinander kommunizieren können. Durch die gegebene Struktur sind der Kommunikationsteil (links) und der Modellberechnungs- und -ausführungsteil (rechts) separiert, wodurch zum einen eine klare Trennung der Aufgaben erreicht wird.

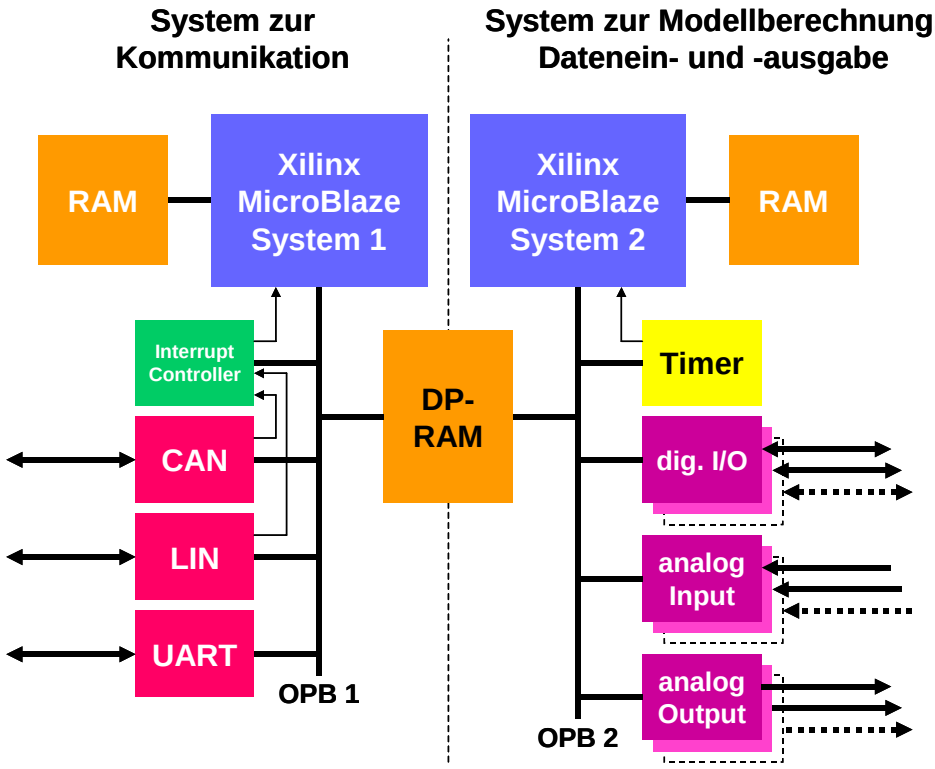

Abbildung 7.10: Doppel-MicroBlaze-Implementierung als autonomes Subsystem

Die Aufgabe des Kommunikationsteils ist es, Daten über einen der Busse anzunehmen und sie über das DP-RAM dem zweiten Prozessorsystem zur Verfügung zu stellen. Damit wird eine dynamische Parameteränderung des auf dem zweiten Prozessor ausgeführten Modells während der Laufzeit erreicht. Durch die angeschlossenen Bussysteme lassen sich aber auch Rechenwerte wieder zurückschicken. Während die Software für die Kommunikation fest implementiert ist, wird die für den Berechnungs- und -ausführungsteil mit Hilfe eines MATLAB/Simulink-Modells bestimmt. Die Funktion des Teilsystems besteht darin, digitale oder analoge Signale aus der Umgebung aufzunehmen, darauf das Modell anzuwenden und

die Ergebnisse wieder an den analogen oder digitalen Ausgängen zur Verfügung zu stellen. Durch den eingesetzten Timer wird eine programmierbare aber feste Zeitbasis bereitgestellt, die als Grundlage aller Berechnungen dient. Die Parameter zur Einstellung werden bereits im Modell vorgegeben und mit dem Code übergeben. Somit lässt sich eine gute Übereinstimmung zwischen der Simulation und dem real ausgeführten Modell erreichen.

In einem Referenzdesign wurde das oben in dem Schema gezeigte System auf einem Virtex-II-1000-FPGA umgesetzt. Um eine auf die Größe des FPGA zugeschnittene Lösung zu erhalten, musste die Vielfalt der I/O-Schnittstellen reduziert werden, da alle anderen Komponenten, wie z.B. der Timer oder der Interrupt-Controller, essentiell zur korrekten Funktion beitragen. Damit konnten neben einem CAN-Controller noch 16 digital I/Os, aufgetrennt in zwei mal 8 GPIOs, und ein DAC-Treiber zur Ansteuerung eines externen D/A-Wandlers integriert werden. Die beiden RS232-Schnittstellen dienten hier lediglich Debugging-Zwecken, um die korrekte Funktion beider Prozessoren zu verifizieren. Für den zwischen beiden Systemen als Kopplungsglied gelegene Zweiport-Speicher (DP-RAM) wurden zwei Block-RAMs verwendet, wodurch 4kByte an Speicher zur Verfügung stehen. Die Implementierung des Doppel-MicroBlaze-Prozessorsystems ist als Floorplan in Abbildung 7.11 dargestellt.

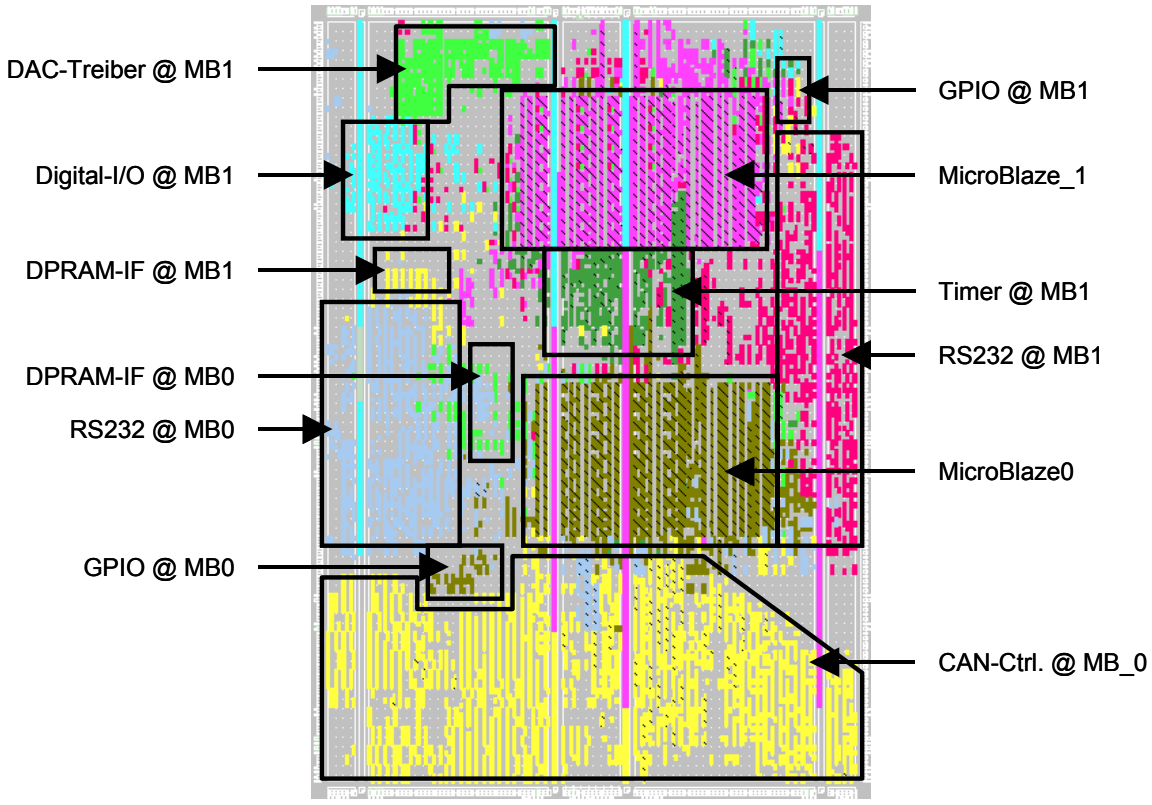

Abbildung 7.11: Floorplan des Doppel-MicroBlaze-µPsystems auf einem Virtex-II-1000

Die wichtigsten FPGA-Implementierungdaten des obigen Designs sind in der folgenden Tabelle dargestellt. Beide Prozessoren hätten gemäß des Timing-Reports mit einer Frequenz von 68.559MHz betrieben werden können. Zur Sicherheit wurde der Takt auf 50MHz reduziert. Damit konnte eine sichere Funktion des Systems erreicht werden.

FPGA-Ressourcen	Belegung	
	absolut	prozentual
External IOBs	48 von 172	27%
MULT18X18s	6 von 40	15%
RAMB16s	36 von 40	90%
SLICEs	3804 von 5120	74%
BUFGMUXs	1 von 16	6%
TBUFs	8 von 2560	1%

Tabelle 7.5: Ressourcenbelegung des Doppel-µBlaze-Systems auf dem FPGA

Software-seitig wurde ein in MatLab-Simulink erstelltes Design als Basis des Tests verwendet, das mit Hilfe eines Frameworks auf dem MicroBlaze-System2 ausgeführt wird. Es stellt einen einfachen Sinusgenerator dar, der über zwei Parameter in Amplitude und Offset modifiziert werden kann. Die Ausgabe erfolgt zunächst digital und wird außerhalb des FPGA durch einen D/A-Wandler in eine analoge Spannung gewandelt.

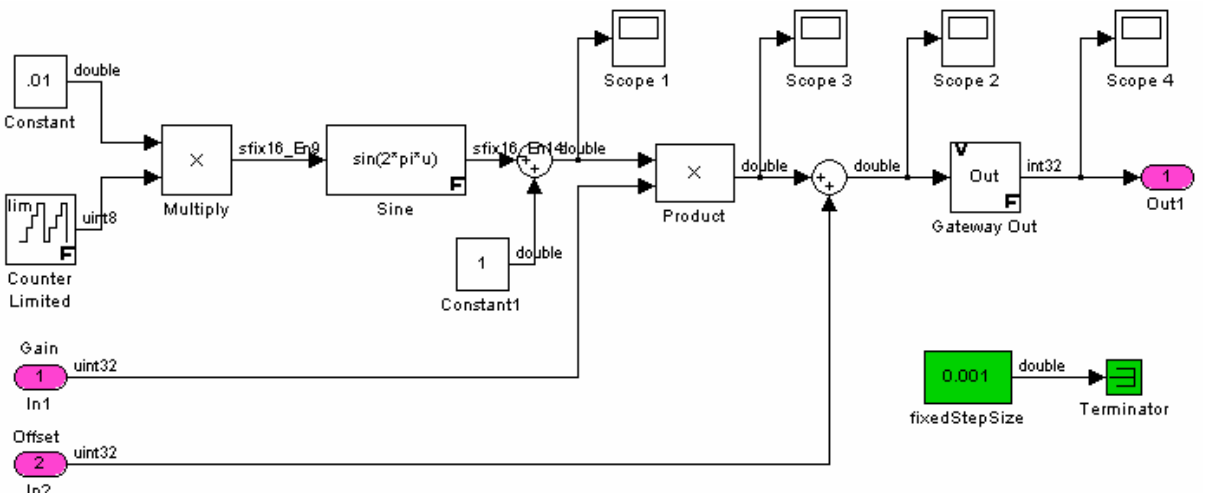

Abbildung 7.12: MatLab-Simulink-Modell eines Sinus-Generators

Die Modifikation der Parameter „Gain“ und „Offset“ in Abbildung 7.12 wurde über bestimmte Speicherstellen des DPRAMs gelöst. Dazu wurden über CAN Daten (Verstärkungsfaktor und Offset) an das MicroBlaze-System1 gesendet, das seinerseits diese Daten aus den CAN-Rahmen extrahierte und in die entsprechenden Speicherstellen im RAM ablegte. Die Auswirkungen auf das Modell konnten unmittelbar danach (z.B. mit einem Oszilloskop) beobachtet werden.

Die Signalverläufe des Ursprungssignals (Scope 1) und des generierten Ausgangssignals (Scope 4) sind in Abbildung 7.13 zu sehen. Dazu wurden die Parameter auf Gain=50 und Offset=20 eingestellt.

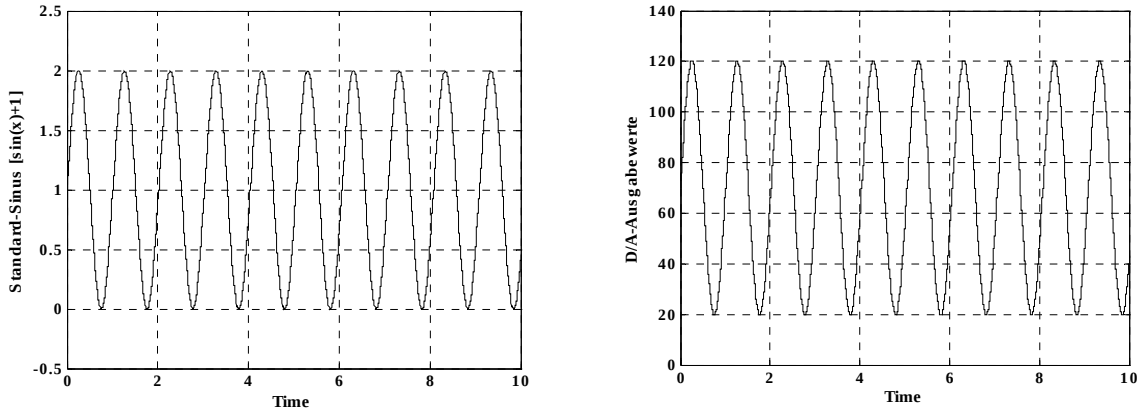

Abbildung 7.13: Basissignal (links) und Ausgangssignal (rechts) des Sinusgenerators

7.2.3 PWM-Analysator für Automotive-Steuergeräte

In Zusammenarbeit mit der Firma MB*tech* in Böblingen wurde ein Signal-Analysator entwickelt, mit dem von Steuergeräten für die Helligkeitssteuerung von LEDs erzeugte PWM-Signale vermessen und bewertet werden können. Ausgangspunkt war die Aufnahme von mindestens 20 solcher Signale, was mit einem oder auch einer Zusammenschaltung von mehreren herkömmlichen Mikrocontrollern nicht zu bewerkstelligen ist. Das Hauptproblem beim Einsatz von µControllern liegt in der geringen Anzahl verfügbarer PWM-Eingänge. Außerdem würde die Berechnungszeit für Frequenz und Duty-Cycle so groß, dass die Zeitvorgaben für die Bereitstellung der Daten (≤1ms) nicht eingehalten werden können. Aus diesem Grund wurde die Umsetzung der Applikation auf einem FPGA angestrebt, da dieser zum einen die in diesem Fall benötigte hohe Anzahl an digitalen Eingänge zur Verfügung stellt, zum anderen aber auch die Signalauswertelogik parallel implementiert werden und damit eine deutliche Performanzsteigerung erzielt werden kann. Eine schematische Übersicht über die Funktionsblöcke des Analysators ist in Abbildung 7.14 zu sehen.

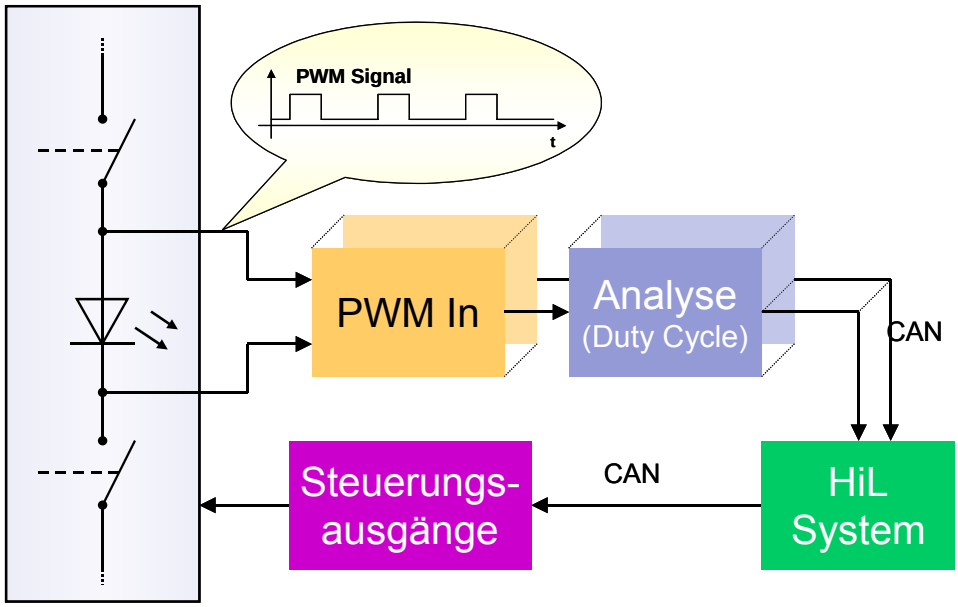

Abbildung 7.14: Schema des Gesamtaufbaus PWM-Analysator

Die Anforderungen an das System, hier im Speziellen an den PWM-Analysator-Core, wurden von den zu messenden Signalen und der gewünschten Genauigkeit vorgegeben. Diese sind hier im Folgenden aufgelistet:

- Frequenzbereich der PWM-Signale: 10Hz bis 4kHz
- Bestimmung des Tastverhältnisses (Duty Cycle) von 0 – 100%
- Auflösung des Tastverhältnisses von 0.5%
- Abtastrate für alle Kanäle ≤ 1 ms (einschließlich der Duty Cycle-Berechnung)
- Einstellbare Glitch-Filter (konfigurierbar für jeden Kanal separat)
- Steuerungsausgänge zur Emulation von Tasten und Schaltern
- Datenaustausch mit dem HiL-System via CAN-Bus

Die Realisierung des PWM-Analysators wurde eine wie in Abbildung 7.15 gezeigte Lösung angestrebt. Die zeitkritische Berechnung der Tastverhältnisse ist hierbei als reine Hardware-Lösung umgesetzt und dadurch schneller als eine in C-Code programmierte Lösung auf einem Prozessor. Die Signalaufnahme bzw. –eingabe geschieht über die bereits erwähnten PWM-Cores, inklusive vorgeschalteter Glitch-Filter, die für ein sauberes Signal sorgen. Ein endlicher Automat (FSM – Finite State Machine) übernimmt nun anstelle des Prozessors die interne Koordination der Abläufe. Die FSM liest die PWM-Werte aus den einzelnen Eingangsstufen aus und übergibt sie einer Teilerstufe zur Berechnung. Außerdem werden über sie die Glitch-Filter eingestellt. Alle fertig berechneten Werte werden in einer vorher festgelegten Reihenfolge in ein DP-RAM (Dual-Port-RAM) geschrieben. Eine weitere FSM liest das DP-RAM primärseitig aus und stellt die Daten dem Prozessor zur Verfügung, der sie seinerseits über das angeschlossene CAN-Interface an den HiL-Rechner weiterleitet.

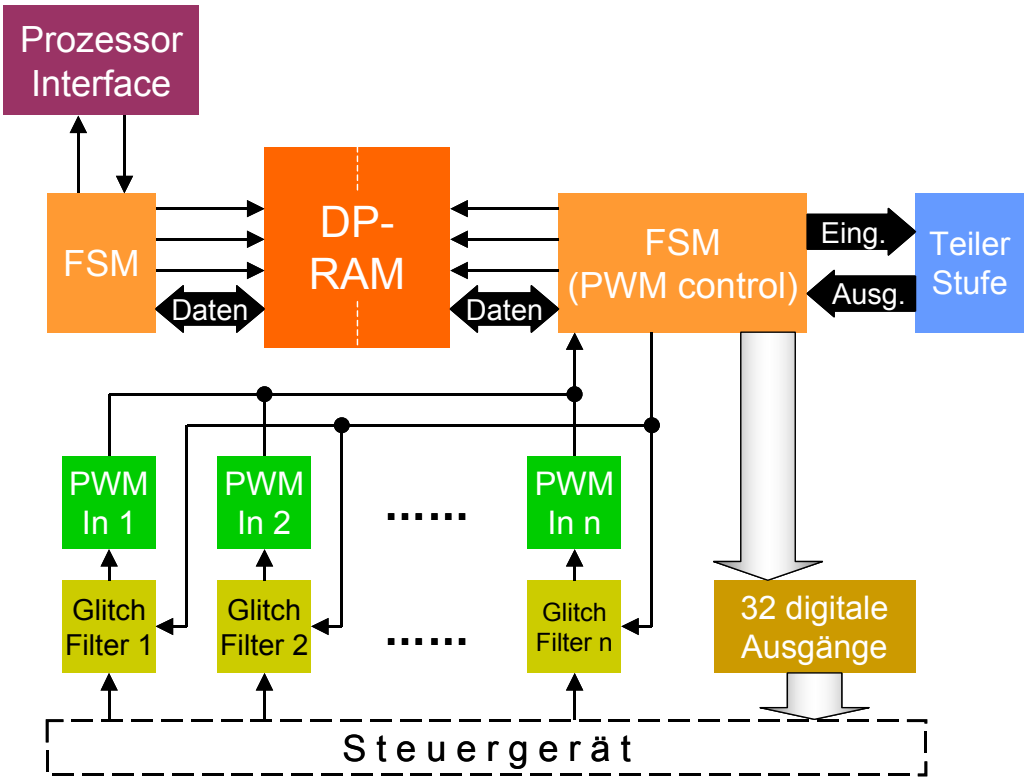

Abbildung 7.15: Realisierung des PWM-Analyzers auf dem FPGA

Das steuernde HiL-System ist auch in der Lage, Befehle an den PWM-Analysator zu schicken, um damit die digitalen Ausgänge anzusteuern. Diese werden dazu verwendet, Relais zu schalten, die entweder einen Taster oder einen Schalter nachbilden. Im Rahmen des Tests werden so Betätigungen des Fahrers emuliert. Die PWM-Signale werden direkt an den LEDs abgegriffen und mit Hilfe von differentiellen Messverstärkern auf die vom FPGA benötigten Pegel angepasst. Eine Möglichkeit zur Implementierung auf der COMPASS-Plattform ist in Abbildung 7.15 gezeigt.

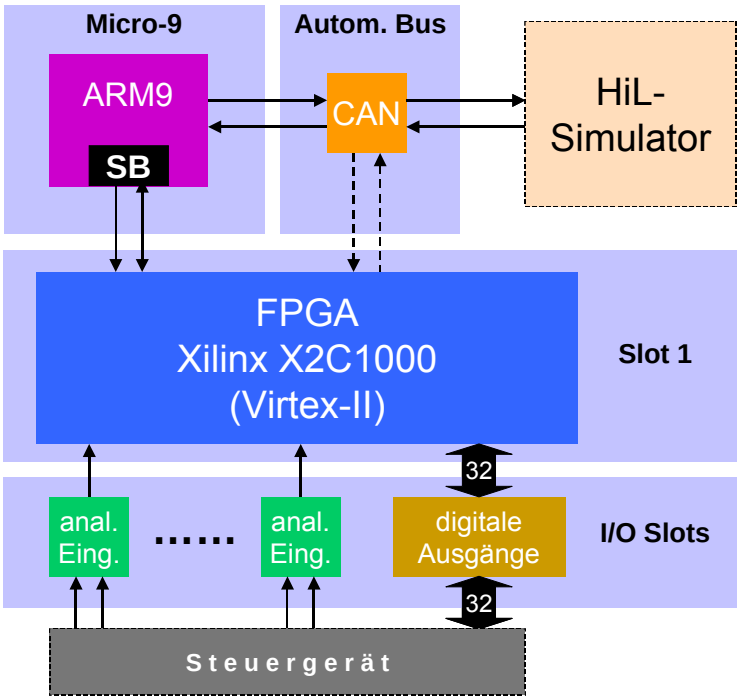

Abbildung 7.16: Partitionierung des PWM-Analysators auf der COMPASS-Plattform

Die Implementierung des Designs wurde auf einem Virtex-II-1000-FPGA durchgeführt. Tabelle 7.6 zeigt die Ressourcenbelegung des kompletten Designs. Darin enthalten sind alle in Abbildung 7.15 dargestellten Module und insgesamt 24 Eingangsstufen.

Ressourcen	Anzahl	Fläche auf Virtex-II
Slices	2052 von 5120	40%
Bonded IOBs	69 von 172	40%
BRAMs	1 von 40	2,5%
MULT18X18s	1 von 40	2,5%

Tabelle 7.6: Ressourcenbelegung des PWM-Analysators

Durch die Verwendung eines in Hardware realisierten Dividierers zur Berechnung der Duty-Cycles konnte eine Signalverarbeitungsdauer von 2400ns pro Kanal erzielt werden, was bei allen 24 Kanälen einer Gesamtberechnungsdauer von 57,6µs entspricht. Damit konnten die zeitlichen Randbedingungen zur Berechnung aller Kanäle innerhalb von 1ms mehr als erfüllt werden. Auch alle anderen genannten Anforderungen, wie eine Auflösung der Duty-Cycle-Werte von 0,5%, wurden abgedeckt. Hinsichtlich der maximalen PWM-Eingabefrequenz konnte anstelle der geforderten 4kHz sogar eine Steigerung auf das Dop-

pelte (8KHz) erzielt werden. Alle weitere Detailinformationen zu der Umsetzung in VHDL können [YeJi05] entnommen werden.

7.2.4 Remote-HiL-Demonstrationsszenario

Im Rahmen dieser Dissertation wurde eine HiL-Demoapplikation entwickelt, mit der die Funktionalität der COMPASS-Plattform im Hinblick auf die Verwendung für ferngesteuerte Testszenarien vorgeführt werden kann. Ziel war die Fernsteuerung eines 1:87 Modellautos über die COMPASS-Plattform. Bei dem zu steuernden Auto handelt es sich um das MiCK (Micro-Car-Karlsruhe), welches in Kooperation zwischen dem Institut für Produktionstechnik (wbk) und dem Institut für Technik der Informationsverarbeitung (ITIV) entstanden ist. Abbildung 7.17 zeigt den prinzipiellen Aufbau der Applikation. Hierbei ist lediglich die Verkabelung stilisiert – die Komponenten entsprechen den tatsächlich eingesetzten.

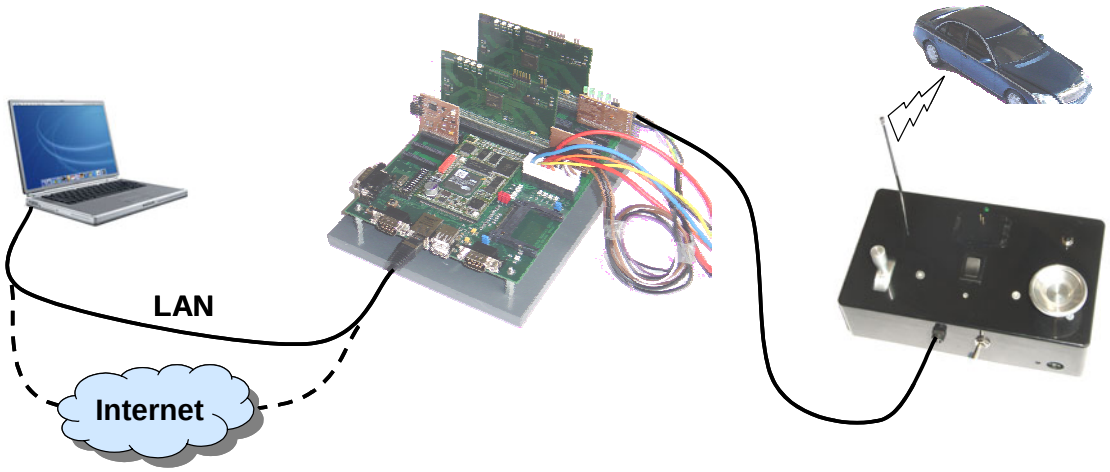

Abbildung 7.17: Aufbau der Remote-HiL-Applikation zur Fernsteuerung des MiCK

Die Ansteuerung der Fernbedienung für das Vorwärts- und Rückwärtsfahren sowie für die Lenkung erfolgte über zwei analoge Spannungen, die in deren Elektronik eingeschleift wurden. Die beiden Signale benötigen einen Spannungspegel zwischen 0 und 2V. Diese Spannungen wurden über den D/A-Wandler der I/O-Karte zur Verfügung gestellt.

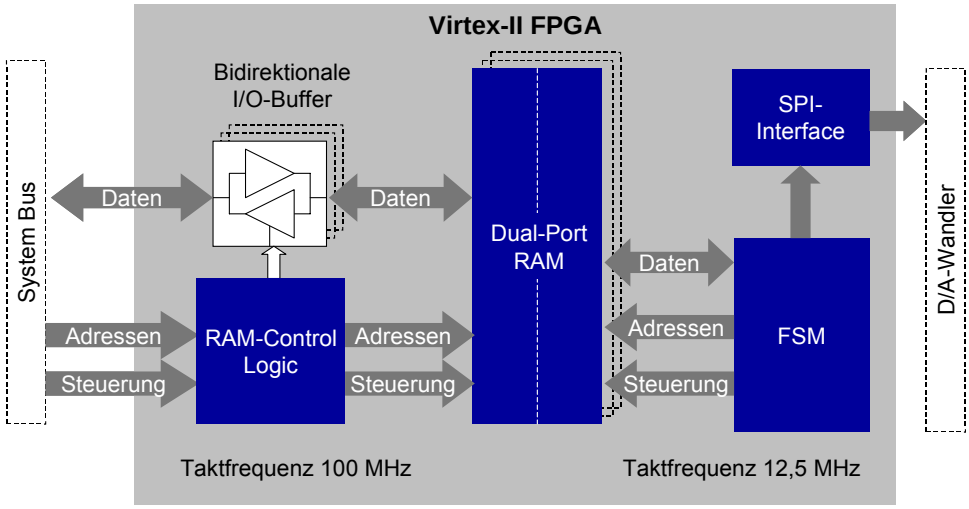

Abbildung 7.18: VHDL-Code Virtex-II-FPGA-Karte

Für diese Testapplikation entstand das in Abbildung 7.18 zu sehende VHDL-Modell, das zur Ansteuerung des D/A-Wandlers auf der I/O-Karte eingesetzt werden kann. Das Modell besteht aus einer Ansteuerlogik für den Buszugriff auf den System-Bus, sowie einem Speicher der als Dual-Port-RAM ausgeführt wurde. Die Logik ermöglicht das Lesen und Schreiben von Daten in das Dual-Port-RAM. Auf der anderen Seite des Speichers wurde ein Zustandsautomat (FSM) implementiert, der einen SPI-Core ansteuert.

Diese FSM liest nacheinander Daten aus bestimmten Registern des Block-RAMs und übergibt diese an den SPI-Core. Die aus dem Register ausgelesenen Daten müssen jetzt seriell an den D/A-Wandler ausgegeben werden. Dazu wurde im SPI-Core ebenfalls eine FSM implementiert, die den Datenstrom bitweise an den D/A-Wandler weiterleitet.

Das zum Speichern der Daten verwendete Dual-Port-RAM bietet zwei komplett unabhängige Schnittstellen. Diese beiden Schnittstellen können mit zwei unterschiedlichen Taktfrequenzen betrieben werden. Auf der Seite des System-Bus wird der Speicher mit 100MHz betrieben. Dies entspricht der Frequenz des System-Busses. Auf der SPI-Seite wird eine Frequenz von 12,5MHz verwendet. Diese wurde von der Frequenz des D/A-Wandlers abgeleitet. Dieser kann maximal mit 13,5MHz betrieben werden. Eine Erzeugung von 13,5 MHz wäre nicht ohne größeren Aufwand möglich. Aus diesem Grunde wurde ein DCM (Digital Clock Manager) zur Taktteilung herangezogen. Dieser teilt die Eingangsfrequenz von 100MHz durch acht und generiert so die 12,5MHz.

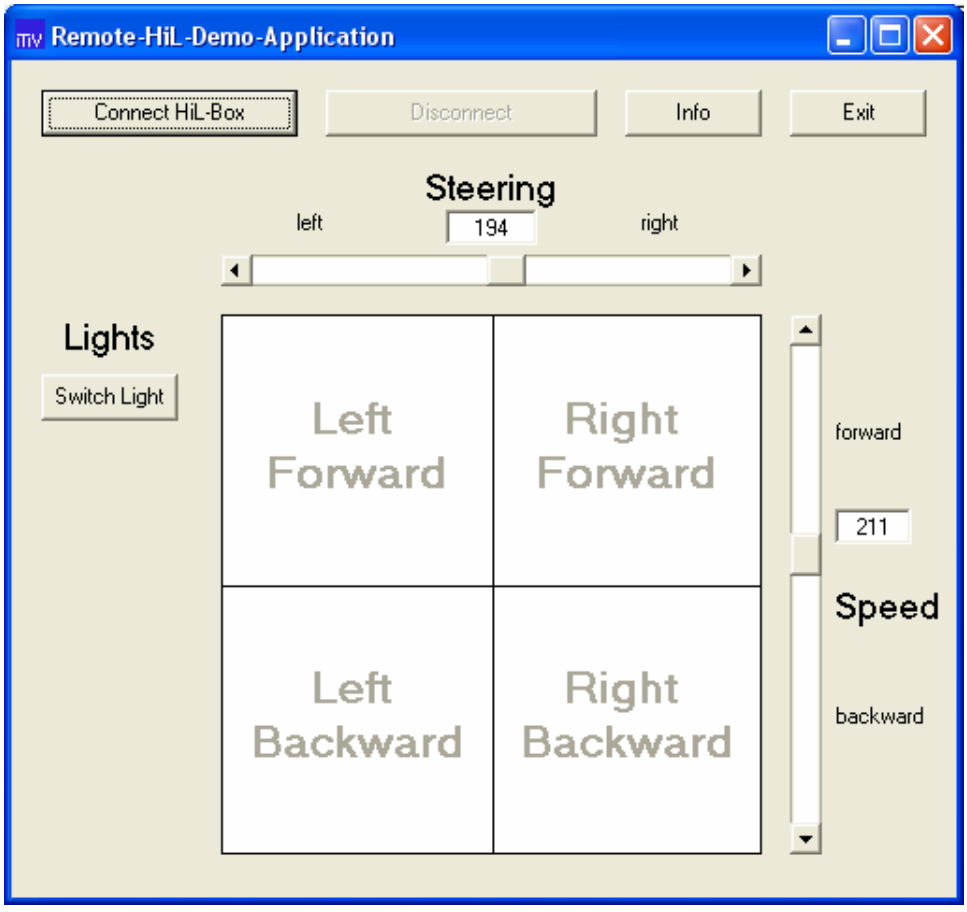

Abbildung 7.19: Remote-HiL-Steuerungsapplikation

Zunächst muss der Bitstream für die Demoapplikation geladen werden. Der Bitstream enthält die oben beschriebene Implementierung zur Ansteuerung des D/A-Wandlers. Danach kann die für diesen Zweck entwickelte ITIV-Remote-HiL-Anwendung gestartet werden. In

dieser Anwendung erfolgt die Eingabe der Steuerungsdaten für das Auto. Durch die Bewegung der Maus auf einem Eingabefeld werden die gewünschte Richtung und Geschwindigkeit eingestellt (siehe Abbildung 7.19). Die Datenübertragung erfolgt schließlich von der Software, über eine Netzwerkverbindung zum Micro-9. Von dort werden die Informationen über den System-Bus an die FPGA-Karte weitergegeben. Schließlich werden sie in einem Register abgelegt, um zur Ansteuerung des D/A-Wandlers auf der I/O-Karte von der Finite-State-Machine (FSM) ausgelesen zu werden. Diese ergänzt die Steuerungsdaten um Steuerbefehle des D/A-Wandlers, und schreibt sie anschließend über eine SPI-Schnittstelle in den Wandler. Da es sich um einen 2-Kanal-D/A-Wandler handelt, werden die Daten zuerst in beide Register des Wandlers geschrieben und danach der Befehl zum Laden der DAC-Register gegeben.

Mit dieser Anordnung wurden zwei Testszenarien gefahren, die sich in der Datenübertragungstrecke dahingehend unterschieden, dass im ersten Fall die Kopplung zwischen der Steuerung und der Plattform über das institutsinterne Netzwerk stattfand, im zweiten über eine Internetverbindung. Dazu wurde bei letzterem Szenario ein an der „School of Info. and Elec. Eng.“ der Universität von Queensland in Brisbane stehender Server konnektiert. An ihn wurden die Datenpakte von der Steuerungsapplikation geschickt und mit Hilfe eines Skripts unverändert an die COMPASS-Plattform zurückgesandt.

Die Ergebnisse der beiden Tests fielen erwartungsgemäß sehr unterschiedlich aus. Während sich die Verzögerungszeiten beim ersten Versuch noch im Bereich weniger Millisekunden befanden und das Miniaturfahrzeug noch zu einigermaßen gut gesteuert werden konnte war, waren bei der Internetverbindung und das Routing über den australischen Server bereits Verzögerungen im Bereich von Sekunden zu messen. Damit konnte eine Steuerung des Autos über die Bedien-GUI (Abbildung 7.19) nicht mehr ohne Probleme ausgeführt werden.

Aus diesen Betrachtungen wird deutlich, dass Steuerungen (Open-Loop-Control) und insbesondere Regelungen (Closed-Loop-Control) über Ethernet/Internet-Verbindungen nicht ohne weitere Vorkehrungen vernünftig durchgeführt werden können, sofern Zykluszeiten von unter einer Sekunde oder weniger gefordert sind. Außerdem ist die Übertragung nicht zeitlich deterministisch. Darüber hinaus müssen Sicherheitsaspekte bei der Übertragung der Daten via Internet betrachtet werden, die den Inhalt der Datenpakete selbst und die Durchführung eines Tests als Gesamtes betreffen, um ein Abhören bzw. eine Manipulation von außen zu unterbinden.

8 Zusammenfassung, Ergebnisse und Ausblick

8.1 Zusammenfassung und Ergebnisse

Im Zuge der steigenden Komplexität der Elektronik im Kraftfahrzeug besteht immer mehr die Notwendigkeit, Steuergeräte bzw. bereits deren Spezifikation auf Fehler zu testen. Dies kann ganz zu Anfang, aber auch im Laufe der Entwicklung recht schnell und unkompliziert durch Einsatz von Rapid Prototyping geschehen. Ist das erste Vorseriengerät verfügbar können die Validierung und Verifikation des Steuergerätes mit Hilfe eines Hardware-in-the-Loop-Testers vorgenommen werden. Dies muss zum einen in Einzelcharakterisierungen geschehen, um bereits spezielle Fehler des Steuergeräts oder dessen Software zu entdecken, wie auch in einem allübergreifenden Test, bei dem alle Komponenten in einem Kommunikationsverbund zusammengeschaltet sind.

Im Rahmen der Arbeit wurde dazu ein kleines, mobiles und preiswertes System namens COMPASS (***Co****nfigurable* ***M****odular* ***P****latform for* ***A****utomotive* ***S****ystem****s***) entwickelt (Abbildung 8.1), das gleichermaßen für Rapid-Prototyping und Hardware-in-the-Loop eingesetzt werden kann und damit eine Kombinationslösung für beide Zwecke darstellt. Die Basis der Plattform bilden die frei programmierbaren Interface-Karten, die durch den Einsatz von FPGAs anstelle dedizierter I/O-Karten eine Erhöhung der Flexibilität und eine Kostenersparnis gegenüber kommerziellen Geräten bieten. Die I/O-Funktionen werden als IP-Cores (Schnittstellen-Controller) zur Systemlaufzeit in den FPGA geladen, wobei eine Karte mehrere, verschiedene Funktionen enthalten kann (verschiedene I/O-Controller, z.B. digitale I/O, PWM [in, out], Nachbildung serieller Protokolle, …), was die Anzahl verschiedener Karten reduziert.

Die Plattform-interne Bussystemkonfiguration besteht aus verschiedenen Kommunikationswegen bzw. -kanälen, die so angelegt sind, dass auch hier eine möglichst große Flexibilität erreicht wird. Der Prozessor ist dazu mit allen Interface-Karten verbunden, wobei letztere auch untereinander, unabhängig vom Prozessor, kommunizieren können (Sub-System-Unterstützung, Modul-Synchronisierung, ...). Um bei der Auswahl eines seriellen Automotive-Bussystems nicht an einen Steckplatz gebunden zu sein, sind diese von allen Interface-Karten aus erreichbar und können zur Laufzeit adaptiert werden. Über zwei weitere völlig unabhängige Kanäle sind die FPGAs der Interface-Karten separat zu programmieren/konfigurieren und die aktuelle HW-Konfiguration der gesamten Plattform abzufragen.

Zur Konfigurierung der FPGAs wurde eine neue Methodik entwickelt, die auf der Slot-basierten Implementierung von IP-Funktionen basiert und keinerlei Kenntnisse von VHDL, Synthese-Tools oder FPGA-Interna voraussetzt. Der Benutzer kann in einer grafischen Benutzeroberfläche Funktionsmodule aus einem Baukastensystem auswählen und den einzelnen Slots zuweisen. Die Bibliothek bietet dazu vorgefertigte, funktionsgetestete Module mit konstanter Performanz, deren Eigenschaften in einer dazugehörigen XML-Datei abgelegt

sind. Die Integration auf den FPGA wird automatisch ausgeführt, wobei die ausgewählten Module in ein Famework eingebettet werden, das die Basisfunktionalität (wie die externe Busankopplung und interne Bussysteme) bereitstellt. Als Ergebnis erhält man einen kompletten Bitstream für den FPGA und einen XML-File mit allen Eigenschaften der Einzelmodule und des Gesamtarrangements. Durch diesen neuen Ansatz kann eine FPGA-Konfiguration binnen weniger Minuten zusammengestellt werden. Zeitraubende Iterationen durch Timing- oder Ressourcenprobleme wie bei herkömmlichen CASE-Tool-Ansätzen fallen nicht an.

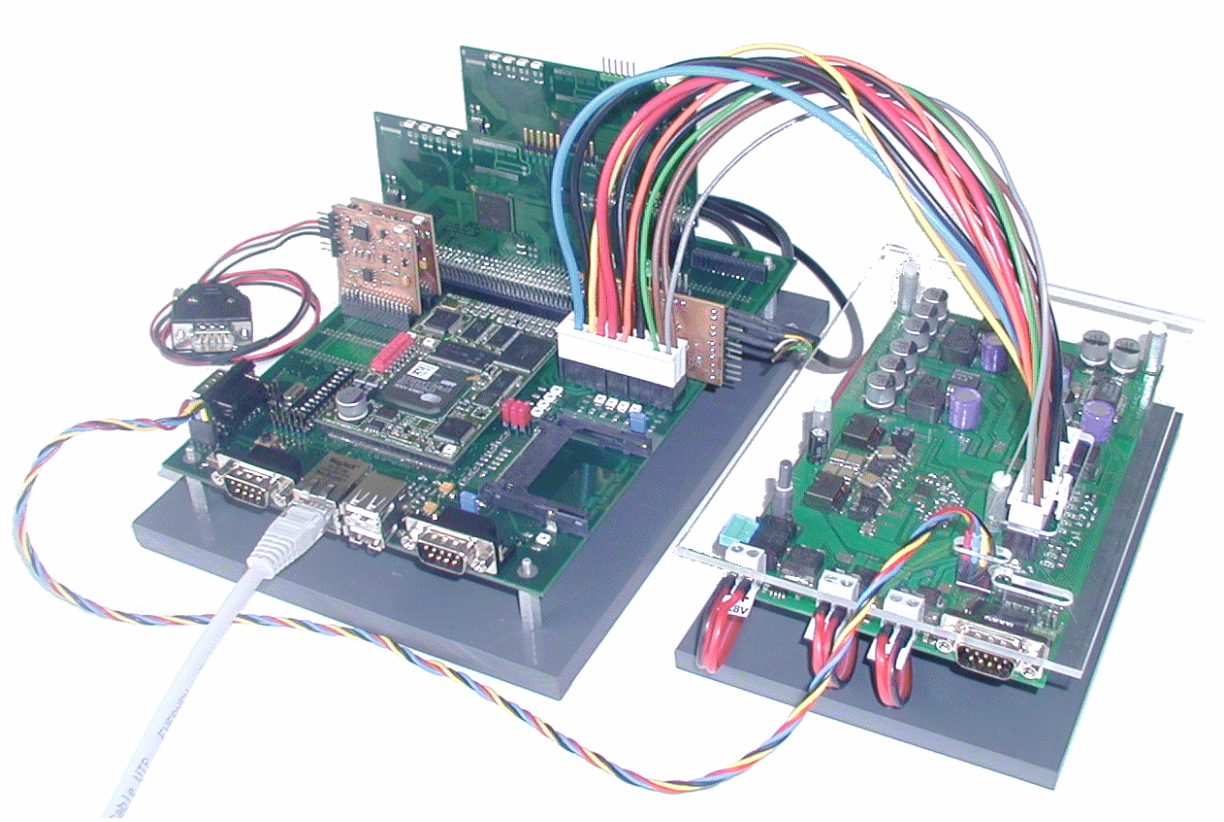

Abbildung 8.1: Foto der COMPASS-Plattform mit Power-Management-Unit (PMU)

Als Alternative zu der obigen Methodik wurde weiterhin ein durchgängiger Design-Flow für FPGA-Funktionsimplementierungen auf den Interface-Karten mit Hilfe grafischer Modellbildung durch die Case-Tools MATLAB Simulink/StateFlow und Xilinx SystemGenerator untersucht. Hierbei wird aus der grafischen Repräsentierung Modell-Code (VHDL oder C-Code) erzeugt, der eingebettet in Hardware-/Software-Frameworks und auf IP-Prozessoren (z.B. PPC, MicroBlaze und LEON) lauffähig ist. Die jeweiligen Frameworks adaptieren die verfügbaren Prozessschnittstellen und vereinfachen so die Integration auf dem FPGA erheblich.

Durch graphische Benutzeroberflächen zur Systemerstellung (Festlegung der Plug-In-Module) und zur Plattformsteuerung ist ein hoher Abstraktionslevel gegeben, der die physikalische Sicht vor dem Benutzer verbirgt und die Flexibilität der Plattform handhabbar macht. Die weitgehende Automatisierung der Abläufe entlastet den Anwender auch dahingehend, dass neben der Erzeugung der FPGA-Konfigurationsdaten, zusätzlich Schnittstellendokumentationen in XML, HTML oder ASCII-Text und passende Header-Dateien für die C-Code-Programmierung automatisch generiert werden.

Ein umfassendes Sicherheitskonzept hilft außerdem Fehlkonfigurationen der Plattform zu vermeiden, bzw. völlig auszuschließen. Bereits beim Erstellen der FPGA-Konfiguration in der GUI werden die Module in den einzelnen Slots auf eine gegenseitige Beeinflussung hin überprüft und damit fehlerhafte Kombinationen ausgeschlossen. Das anschließende automa-

tische Zusammenführen der Module durchläuft ebenso eine Korrektheitsprüfung, um einen korrekten Konfigurationsdatenstrom (Bitstream) zu garantieren. Die für diese Konfiguration benötigten HW-Ressourcen werden dabei in einer dazugehörigen XML-Datei abgelegt. Bei Inbetriebnahme der Plattform wird eine Ressourcenprüfung durchgeführt, die als Ergebnis die Positionen und Eigenschaften aller eingesteckten Karten liefert. Alle danach zum Herunterladen ausgewählten FPGA-Konfigurationsdaten werden zunächst gegen diese Anforderungen geprüft und bei fehlender Übereinstimmung abgelehnt, um Konflikte zu unterbinden. Erst nach erfolgreichem Download schaltet die Software die Treiber für die Peripherie zu. Dieser Vorgang geschieht ebenfalls automatisch. Während des Betriebs werden Spannungen und Ströme von der PowerControlUnit (PMU) überwacht, die im Fehlerfall selbsttätig Kanäle abschalten kann. Über CAN oder RS232 lässt sich die Stromversorgung auch von der COMPASS-Plattform selbst oder von extern fernsteuern.

Im Bereich des Rapid-Prototyping unterstützt die Plattform konzept-, architektur- und realisierungs-orientiertes RP, wodurch sie im kompletten Entwicklungszyklus eingesetzt werden kann. Für Hardware-in-the-Loop-Tests sind neben der Standardfunktion (Modellrechnung auf dem Hauptprozessor, FPGAs auf Interface-Karten nur für I/O) auch parallele Modellrechnungen auf den jeweiligen Karten (bis hin zu einem Modell pro I/O) möglich. Die Skalierbarkeit eines Komplett-HiL-Testsystems ist durch Kopplung einzelner Plattformen über Ethernet, CAN oder die serielle Schnittstelle gegeben.

8.2 Ausblick

Das in dieser Arbeit entstandene System ist in seiner derzeitigen Version voll funktionsfähig und hat seine Einsetzbarkeit in den in Kapitel 7 aufgeführten Anwendungsszenarien unter Beweis stellen können. Dennoch ist COMPASS selbst noch prototypisch realisiert und daher an einigen Punkten verbesserungsbedürftig, aber auch erweiterungsfähig. Dies bezieht sich sowohl auf die in dieser Arbeit vorgestellte Hardware, als auch auf die Software. Im Folgenden wird auf einige Punkte eingegangen, die modifiziert einen noch reibungsloseren Betrieb zuließen. Dabei sind die aufgeführten Themen nach Hardware und Software, sowie nach einzelnen Komponenten gegliedert.

Hardware-Plattform

Bisher wurden für die Plattform nur FPGAs der Virtex-II-Serie von Xilinx für die Interface-Karten zur Implementierung der Logik mit einer Größe von einer Million Gatteräquivalenten verwendet. In einem Re-Design der Interface-Karten könnten weitere Varianten mit verschiedenen Größen von FPGAs (d.h. mit einer anderen Anzahl an Gattern) eingesetzt werden. Dabei wäre auch denkbar, die FPGAs der Virtex-II-PRO-Serie zu integrieren, die neben der frei programmierbaren Logik auch bereits einen oder mehrere PowerPC-Kerne als feste Bestandteile beinhalten. Dadurch ließen sich rechenintensive Subsysteme ohne Verwendung von Mikrocontrollern als IP-Core, die selbst einen Großteil der Logikreserven belegen, effizienter umsetzen und deren Verarbeitungsgeschwindigkeit stark erhöhen. Ein solches Szenario ist im Bereich Hardware-in-the-Loop denkbar, in dem ein Umgebungsmodell partitioniert und zur Berechnungen direkt auf verschiedenen FPGAs ausgeführt wird. Über die zur Verfügung stehenden I/O-Pins des FPGA ließen sich die I/O-Treiber oder eine Vierquadrantenverstärker [Spit01] direkt ansteuern. Führt man dieses Szenario weiter aus, ist ein HiL-Fehlereinbau über eine modellbasierte Signalmanipulation denkbar, wobei auch hier das Rechenmodell innerhalb des FPGA ausgeführt wird.

Zur schnelleren Programmierung der Virtex-II-FPGAs könnte ein VHDL-Modul entworfen werden, das die Bitstream-Daten des Serial-Programming im Spartan-II-FPGA parallel entgegennimmt. So wäre es möglich mit einem Buszugriff beispielsweise 16 Datenbits gleichzeitig zu übertragen. Die Erzeugung des Taktsignals und die serielle Ausgabe der Daten müsste über eine State-Machine in VHDL erfolgen. Diese Implementierung würde gegenüber der aktuellen Lösung einen Geschwindigkeitsvorteil von etwa dem Faktor 16 bieten.

Für die partielle dynamische Re-Konfiguration der Virtex-FPGAs könnte ein JTAG-Interface für den Spartan-II-FPGA entwickelt werden. Die Hardware-seitigen Voraussetzungen für eine Implementierung sind bereits getroffen worden. Über ein solches Interface könnten die Steckkarten-FPGAs zur Laufzeit dynamisch re-konfiguriert werden, was mit dem momentan verwendeten Serial-Programming nicht möglich ist. Damit könnte die Plattform auch für das Rapid Prototyping von Automotiv-Applikationen eingesetzt werden, bei denen Innenraumfunktionen (Fensterheber, Sitzverstellung, Schiebedach, etc.) dynamisch gegeneinander ausgetauscht werden. Mit den von der Plattform bereitgestellten I/O-Schnittstellen wäre eine direkte Prozesskopplung möglich.

Stromversorgung

Um die Sicherheit beim Einsatz der Plattform noch weiter zu erhöhen, könnte zu dem derzeit realisierten Spannungs-Monitoring eine Überwachung der Lastströme auf den einzelnen Kanälen realisiert werden. Damit ließe sich bei Kurzschlüssen innerhalb der Plattform und auch im Bereich der Signalkonditionierung eine wirksame Schnellabschaltung zum Schutz der Baugruppen verwirklichen.

Weiterhin könnte die Software des Mikrocontrollers dahingehend erweitert werden, dass durch die Verwendung von Interrupt-Routinen für das Auslesen des A/D-Wandlers die Messgeschwindigkeit noch weiter erhöht wird, wodurch sich die Reaktionsfähigkeit auf Über- und Unterspannungen verbessert. Bei einer Weiterentwicklung der COMPASS-Plattform ist es außerdem sinnvoll, die Stromversorgung mit auf das Basis-Board zu integrieren, um Spannungsabfälle über die Anschlussstecker und Kabel zu vermeiden. In diesem Fall wäre allerdings ein Layout mit mindestens sechs Lagen empfehlenswert.

Plattformsteuerung (CompassControl)

Eine der naheliegendsten Verbesserungen hinsichtlich der Benutzerfreundlichkeit wäre das Zusammenführen der in dieser Arbeit entwickelten graphischen Benutzeroberfläche CompassControl und der Applikation zur Erzeugung von Bitstreams BitstreamComposer. Beide Programme sind in der Programmiersprache Java geschrieben und durch ihren modularen Aufbau der verschiedenen Klassen sollte dieser Integrationsschritt problemlos möglich sein. Damit entfiele das Hin- und Herschalten zwischen zwei Applikationen bei der Erstellung der FPGA-Konfiguration und dem Download auf die Plattform. Insbesondere bei Fehlkonfigurationen und einer nötigen Änderung im Design, könnte so weitere Zeit eingespart werden.

Als Ergänzung auf Seite der Benutzerschnittstelle wäre auch die Erstellung einer Funktionsbibliothek (API) zur Anbindung eigener PC-Programme des Benutzers, z.B. zur Fernsteuerung der Plattform, bzw. zur Integration in eine größere Design-Umgebung, an die Netzwerkschnittstelle der Compass-Plattform möglich. Die in der Applikation CompassControl verwendete Klasse für den Netzwerkzugriff könnte dabei als Basis für die Entwicklung einer Java-API dienen. Als Basis für Applikationserweiterung in C++ müsste die API noch angepasst werden, wobei jedoch Teile des COMPASS-Netzwerkservers wieder verwendet werden können.

Aus Sicht der Software-Implementierung wäre die Erstellung eines Linux-Kernelmoduls zur Integration des mit 200 MHz getaktete ARM-Prozessors auf dem MICRO-9 in RP- oder HiL-Simulationsmodelle sinnvoll. Die z.B. in einem Case-Tool erstellten und mit dem passenden GNU-C-Compiler für die Plattform in ein lauffähiges Programm verwandelten Simulationsmodelle könnten so schnell getestet werden. Nach der Konfiguration der Plattform würde dann automatisch die Ausführung der Software gestartet und bei Bedarf auch Rückmeldungen an den steuernden Host-PC geliefert. Eine Erweiterung dessen könnte schließlich auch in einer verteilten Simulation gipfeln, bei der Simulationsmodelle auf den einzelnen FPGA-Karten ablaufen. Die Synchronisation würde dabei über die bereits vorhandenen Interrupt-Leitungen geschehen.

Wie bereits für die Plattform als Verbesserung vorgestellt, wäre die Implementierung einer JTAG-Schnittstelle anstelle des Slave-Serial-Interfaces zur Programmierung der FPGAs auf den Interface-Karten eine nützliche, allerdings auch sehr aufwändige, Erweiterung. Da JTAG

ein sehr komplexes Protokoll ist, wurde für den Prototypen der COMPASS-Plattform die FPGA-Programmierung über das Protokoll Slave-Serial gewählt. Die Nutzung von JTAG eröffnet jedoch ganz neue Möglichkeiten wie z.B. das dynamische Re-Konfigurieren der FPGA-Karten. Die Umsetzung dessen würde dabei zum Teil in Software zur Ausführung auf dem Micro-9-Board und zum Teil im Board-Control-FPGA in VHDL geschehen. Die Infrastruktur auf dem Basis-Board und den Interface-Karten ist darauf bereits ausgelegt, so dass die Umsetzung dahingehend keinerlei Hardware-Änderungen erfordert.

Konfigurationserstellung (BitstreamComposer)

Um die Funktionalität des Gesamtsystems zu erweitern, sollten der Modul-Bibliothek weitere funktionale Module hinzugefügt werden. Die Art der zu implementierenden Module für den Automotive-Bereich kann dabei leicht aus den Prospekten der einschlägig bekannten RP- und HiL-Systemhersteller abgeleitet werden, die diese als reine Hardware-Lösungen anbieten. Dabei muss lediglich darauf geachtet werden, dass die Schnittstellenbeschreibung durch die definierten Modul-Komponenten vorgegeben ist.

9 Verzeichnisse

9.1 Abbildungsverzeichnis

Abbildung 1.1: Einsatzbereiche von Eingebetteten Systemen 1
Abbildung 1.2: Aufwände (Kosten) bei der Systementwicklung [Fraun03] 2
Abbildung 1.3: Testverfahren zur Unterstützung der Produktentwicklung 3
Abbildung 2.1: Steuergeräte und Busse im W220 (S-Klasse) von DaimlerChrysler 7
Abbildung 2.2: Etablierung von Kommunikationssystemen im Automobil 8
Abbildung 2.3: Aktuell im Automobil eingesetzte Bussysteme 9
Abbildung 2.4: Stern-Topologie 10
Abbildung 2.5: Ring-Topologie 10
Abbildung 2.6: Bus-Topologie 10
Abbildung 2.7: TDMA-Verfahren 10
Abbildung 2.8: FDMA-Verfahren 10
Abbildung 2.9: Aufbau einer CAN-Botschaft 12
Abbildung 2.10: Aufbau einer LIN-Botschaft 13
Abbildung 2.11: Aufbau eines MOST-Frames 14
Abbildung 2.12: Datenformat einer FlexRay-Botschaft 15
Abbildung 2.13: Botschaftsformat eines I-Frames 16
Abbildung 2.14: Botschaftsformat eines N-Frames 17
Abbildung 3.1: Integrierte Schaltungen 22
Abbildung 3.2: Architekturschema eines Xilinx-Virtex-II-FPGAs 23
Abbildung 3.3: Anordnung der Konfigurationsspalten für den Virtex-II XC2V1000 25
Abbildung 3.4: Aufbau des Adressregisters (Frame Address Register) 25
Abbildung 3.5: Aufbau eines Headers vom Typ 1 26
Abbildung 3.6: Aufbau eines Headers vom Typ 2 26
Abbildung 3.7: Schaltplan Boundary-Scan Mode 30
Abbildung 3.8: Schaltplan Slave Parallel Mode 32
Abbildung 3.9: Master/Slave Serial Mode 32
Abbildung 3.10: Innerer Aufbau eines FPAAs von Anadigm 33
Abbildung 3.11: Klassifizierung von Sensorschnittstellen 36
Abbildung 3.12: Klassifizierung von Aktorschnittstellen 36
Abbildung 3.13: Programmierbarer Taktgenerator [PCK12429] 39
Abbildung 3.14: Taktverteiler (Repeater) für 4 Netzabschnitte 39
Abbildung 3.15: Schema einer LVDS-Übertragungsstrecke 40
Abbildung 3.16: Schematischer Aufbau eines integrierten Spannungsreglers 40
Abbildung 3.17: Schema eines Schaltreglers 41
Abbildung 3.18: Asynchroner und Synchroner Abwärtswandler 42

Abbildung 3.19: Screenshot der Benutzeroberfläche Eclipse 45
Abbildung 3.20: ScreenShot der ISE [XISE05] von Xilinx 46
Abbildung 3.21: ScreenShot der EDK [XEDK04] von Xilinx 48
Abbildung 3.22: ScreenShot des Tools PlanAhead [Plan05] von Xilinx 49
Abbildung 3.23: Wasserfall-Modell 51
Abbildung 3.24: Spiral-Modell 51
Abbildung 3.25: Das V-Modell 52
Abbildung 3.26: Das VP-Modell 53
Abbildung 3.27: Beispiel eines StateCharts (MATLAB/StateFlow) 54
Abbildung 3.28: Beispiel eines Blockdiagramms (MATLAB/Simulink) 55
Abbildung 3.29: Aufbau eines Rapid-Prototyping Systems 56
Abbildung 3.30: Blockdiagramm des SystemControllersES1120.3 von ETAS 61
Abbildung 3.31: Rapid Prototyping System ES1000 von ETAS 61
Abbildung 3.32: Rapid Prototyping-System MicroAutoBox von dSPACE 62
Abbildung 3.33: Blockdiagramm der MicroAutoBox von dSPACE 62
Abbildung 3.34: Hardware-in-the-Loop-Aufbau für elektronische Systeme 64
Abbildung 3.35: dSPACE HiL-Simulatoren Mid-Size (links) und Full-Size (rechts) 66
Abbildung 3.36: ETAS HiL-System LabCar (Serie 4100) 67
Abbildung 3.37: HiL-System CarMaker von IPG 68
Abbildung 4.1: Struktur aktueller Ein-/Ausgabe-Schnittstellenkarten 75
Abbildung 4.2: Struktur bei Trennung von Schnittstellenlogik und Prozesskopplung 76
Abbildung 4.3: Bedarfsgerechte Implementierung von Funktionen 77
Abbildung 4.4: DSP-Subsystem mit direkter Sensor-/Aktoranbindung 81
Abbildung 4.5: Grundstruktur FPGA 83
Abbildung 4.6: Funktionsbelegung 1 83
Abbildung 4.7: Funktionsbelegung 2 83
Abbildung 4.8: Funktionsbelegung 3 83
Abbildung 4.9: Funktionsbelegung 4 83
Abbildung 4.10: Schematischer Aufbau der Interface-Karte 85
Abbildung 4.11: Schematische Darstellung einer Automotive-Bus-Karte 87
Abbildung 4.12: Aufbau und Funktionsüberblick des Board-Control-FPGAs 93
Abbildung 4.13: Aufbau des System Bus (SB) 93
Abbildung 4.14: Kopplung von Interface-Karten über den Module Interconnect Bus (MIB) 95
Abbildung 4.15: Ankopplung der EEPROMs (Variante 1) 96
Abbildung 4.16: Ankopplung der EEPROMs (Variante 2) 96
Abbildung 4.17: Dezentrale Programmier-Interfaces auf den Interface-Karten 97
Abbildung 4.18: Zentrales Programmier-Interface für Interface-Karten-FPGAs 97
Abbildung 4.19: Spezifikation der maximalen V_{CCINT} -Anstiegszeit eines Spartan-II 104
Abbildung 4.20: Spannungsversorgungseinheit für die RP-HiL-Plattform 105
Abbildung 4.21: Vereinfachtes Flussdiagramm 106
Abbildung 4.22: Schematische Übersicht über die Plattform 107
Abbildung 4.23: Basisplatine des RP-HiL-Systems 108
Abbildung 4.24: Übersichtsblockschaltbild des EP9315 von CirrusLogic [Cirr04] 109
Abbildung 4.25: Oberseite des Micro-9 Board von Contec mbH, Österreich 110
Abbildung 4.26: Unterseite des Micro-9 Board 110
Abbildung 4.27: Realisierung des On-Board CAN-Busanschlusses 112
Abbildung 4.28: Interface-Karte (Oberseite) 113
Abbildung 4.29: Foto der Beipiel-I/O-Karte 114
Abbildung 4.30: Automotive-Karte mit 2 CAN-Transceivern 114
Abbildung 4.31: Stromversorgungseinheit (Power Management Unit, PMU) 115
Abbildung 5.1: Syntheselauf 1 (Original) 118
Abbildung 5.2: Syntheselauf 2 (mit inkrementeller Änderung) 118
Abbildung 5.3: Aufteilung des Virtex-II 1000 in Bereiche (Slots) 119
Abbildung 5.4: Verteilung der CLB-, MULT- und BRAM-Spalten in Virtex-II-FPGAs 121
Abbildung 5.5: Aufteilung der CLB-, MULT- und BRAM-Spalten im Virtex-II-1000 122
Abbildung 5.6: Vorgehensweise beim Zusammensetzen eines Bitstreams 123

Abbildung 5.7: Aufteilung der Konfigurationsspalten in Slots 124
Abbildung 5.8: Struktur des Frameworks..... 126
Abbildung 5.9: Kommunikationskonzept innerhalb des Frameworks 127
Abbildung 5.10: Tri-State-Buffer-Leitungen oberhalb einer CLB-Reihe..... 127
Abbildung 5.11: Struktureller Aufbau der verwendeten Bus-Makros 128
Abbildung 5.12: Prinzipieller Ablauf eines Lese- und Schreibzugriffs des ARM-Prozessors128
Abbildung 5.13: Schritte der Funktionsimplementierung für drei Slots 130
Abbildung 5.14: Design-Ablauf als Flussdiagramm 132
Abbildung 5.15: Generierung von drei Bitstreams für jedes funktionale Modul 134
Abbildung 5.16: Fehlerhafte Implementierung eines Moduls in Slot A 135
Abbildung 5.17: Aktivierung der Verzweigungspunkte des Taktnetzes 136
Abbildung 5.18: Aufteilung des Registers spi_reg 138
Abbildung 5.19: Ein LIN-Frame mit Header und Response..... 140
Abbildung 5.20: Aufbau des LIN-Cores mit Fehlerinjektionsmöglichkeit 140
Abbildung 5.21: Belegung des Kontrollregisters..... 142
Abbildung 6.1: Abstraktionsschichten des Linux-Kernels (Grafik: Wikipedia) 149
Abbildung 6.2: Klassendiagramm der CompassLink-Applikation 152
Abbildung 6.3: Screenshot des BitstreamComposer 156
Abbildung 6.4: Modulauswahl im Drop-Down-Menü..... 156
Abbildung 6.5: Programmablauf des BitstreamComposers..... 157
Abbildung 6.6: Funktionsweise der Funktion copySlots 158
Abbildung 6.7: Meta-Modell der XML-Konfigurationsdatei 160
Abbildung 6.8: FPGA-Beispielimplementierung zur Darstellung der Ergebnisse 162
Abbildung 6.9: Screenshot der graphischen Benutzeroberfläche CompassControl..... 163
Abbildung 6.10: Klassendiagramm der CompassControl 166
Abbildung 6.11: Klassendiagramm der XML-Datei zur Beschreibung Bitstream-Inhalte..... 170
Abbildung 6.12: Auslesen der HW-Konfiguration nach Booten der Plattform..... 170
Abbildung 6.13: Automatisch erzeugte Schnittstellendokumentation in HTML 173
Abbildung 6.14: Möglichkeiten zur Hardware-Implementierung modellbasierter Entwürfe . 175
Abbildung 6.15: Ablauf der Modellierung und Einbettung in ein Framework 176
Abbildung 6.16: Tool-Kette und Datenfluss 177
Abbildung 6.17: Modell, Framework und Bibliotheksdateien 178
Abbildung 6.18: Prototypenmodell (PID-Regler) in SIMULINK..... 179
Abbildung 6.19: Beispiel eines SystemGenerator-Funktionsmodells 182
Abbildung 6.20: Schematische Darstellung des Frameworks für Modelle des SysGen 183
Abbildung 6.21: SysGen-Template zur Modellerstellung..... 185
Abbildung 6.22: Design-Flow für ereignisgesteuerte Systeme 186
Abbildung 6.23: Ein-/Ausgabevariablen des Modells 187
Abbildung 6.24: Beispiel für ein StateFlow-Modell mit Nebenläufigkeiten..... 188
Abbildung 6.25: Aufbau der Steuer-Software 189
Abbildung 7.1: COMPASS-Plattform mit MicroBlaze®-Subsystem 191
Abbildung 7.2: Floorplan des CAN2CAN-Gateways auf eine Virtex-II-1000 192
Abbildung 7.3: Testszenarien des CAN2CAN-Gateways als MSC 193
Abbildung 7.4: Network-on-Chip zum Test der Router-Funktionalität und –effektivität 195
Abbildung 7.5: Interface zum Ein- und Auskoppeln von Daten in das bzw. aus dem NoC . 196
Abbildung 7.6: LIN-Frame mit vier Datenbytes und Checksumme..... 197
Abbildung 7.7: Trace-Fenster in CANoe.LIN nach Datenempfang..... 198
Abbildung 7.8: Empfang eines LIN-Frames mit fehlerhaftem Synchronisationsfeld..... 200
Abbildung 7.9: Designablauf für regelungstechnische Systeme auf FPGA-Basis..... 202
Abbildung 7.10: Doppel-MicroBlaze-Implementierung als autonomes Subsystem 203
Abbildung 7.11: Floorplan des Doppel-MicroBlaze-µPsystems auf einem Virtex-II-1000 ... 204
Abbildung 7.12: MatLab-Simulink-Modell eines Sinus-Generators 205
Abbildung 7.13: Basissignal (links) und Ausgangssignal (rechts) des Sinusgenerators 206
Abbildung 7.14: Schema des Gesamtaufbaus PWM-Analysator 206
Abbildung 7.15: Realisierung des PWM-Analyzers auf dem FPGA 207
Abbildung 7.16: Partitionierung des PWM-Analysators auf der COMPASS-Plattform 208

Abbildung 7.17: Aufbau der Remote-HiL-Applikation zur Fernsteuerung des MiCK..... 209
Abbildung 7.18: VHDL-Code Virtex-II-FPGA-Karte 209
Abbildung 7.19: Remote-HiL-Steuerungsapplikation 210
Abbildung 8.1: Foto der COMPASS-Plattform mit Power-Management-Unit (PMU)..... 214

9.2 Tabellenverzeichnis

Tabelle 2.1: Klassifizierung von CAN-Bussen 12
Tabelle 2.2: CAN-Steckerbelegung nach DIN 41652 (CiA/DS 102-1) 13
Tabelle 2.3: Extremtemperaturen nach Einbauräumen gemäß SAE-Standard J1211 18
Tabelle 3.1: Anzahl der vorhandenen Konfigurationsspalten (XC2V1000) 25
Tabelle 3.2: Anzahl der Frames in einer Konfigurationsspalte (XC2V1000) 25
Tabelle 3.3: Blockadressen 26
Tabelle 3.4: Spartan-II Konfigurationsmodi 29
Tabelle 3.5: Xilinx TAP-Controller Pins 29
Tabelle 3.6: Xilinx Spartan-II JTAG-Register 30
Tabelle 3.7: Xilinx Spartan-II Befehlssatz 31
Tabelle 3.8: Schnittstellen für Rapid Prototyping 59
Tabelle 3.9: Leistungsstufen und Signalkonditionierung für Rapid Prototyping 60
Tabelle 3.10: Spezifische Schnittstellen für Hardware-in-the-Loop 65
Tabelle 4.1: Programmierbare Ströme in den Modi LVTTL und LVCMOS (je Source/Sink) . 78
Tabelle 4.2: Anzahl an freien I/Os der Virtex-II-FPGA-Serie von Xilinx 78
Tabelle 4.3: Toggle-Frequenzen der FlipFlops in Virtex-II-FPGAs 79
Tabelle 4.4: Übersicht über die Merkmale der Virtex-II-Serie von Xilinx 82
Tabelle 4.5: Übersicht über die Merkmale der Virtex-II-Pro-Serie von Xilinx 82
Tabelle 4.6: Limitierungen der I/O-Platine 87
Tabelle 4.7: Benötigte Pinzahlen der einzelnen Steckkarten 88
Tabelle 4.8: Serielle EEPROMs zur Kartenidentifikation 89
Tabelle 4.9: Konfigurationsmodi für Xilinx-FPGAs 90
Tabelle 4.10: Abschätzung der Steuerleitungen 91
Tabelle 4.11: Möglichkeiten zur Porterweiterung 92
Tabelle 4.12: Freie I/Os der Spartan-II-FPGA-Serie von Xilinx 92
Tabelle 4.13: Medien zur Konfigurations- und Messdatenspeicherung 98
Tabelle 4.14: Steckerbelegung der Plattformstromversorgung 99
Tabelle 4.15: Spannungs- und Stromanforderungen 102
Tabelle 4.16: Erforderliche Einschaltströme für Virtex-II-FPGAs 104
Tabelle 4.17: Maximale Ausgangsströme der jeweiligen Spannungen 116
Tabelle 4.18: Wirkungsgrade der Stromversorgungsplatine 116
Tabelle 5.1: Verfügbare Ressourcen der Virtex-II-FPGAs 120
Tabelle 5.2: Verfügbare Ressourcen in einem Slot 122
Tabelle 5.3: Logikanforderungen CAN- und LIN-Controller 123
Tabelle 5.4: ISE-Projekteinstellungen 131
Tabelle 5.5: Register des Moduls dac 138
Tabelle 5.6: Register des LIN-Moduls 141
Tabelle 5.7: Bedeutung der ersten vier 16Bit-Felder der Karten-EEPROMs 144
Tabelle 6.1: Physikalische Adressbereiche des HyperControl MICRO-9 148
Tabelle 6.2: Aufteilung des Flash-EEPROMs auf dem MICRO-9 149
Tabelle 6.3: Netzwerkbefehle zwischen Client und COMPASS-Plattform 153
Tabelle 6.4: Bedeutung der ersten vier 16Bit-Felder der Karten-EEPROMs 171
Tabelle 6.5: Vergleich Berechnungsgeschwindigkeit Modellschritt (50MHz) 181
Tabelle 7.1: Ressourcenbelegung des CAN2CAN-Gateways 192
Tabelle 7.2: Ressourcenbelegung eines Routers auf einem Virtex-II-1000 194
Tabelle 7.3: Ressourcenbelegung eines NoC-Interfaces auf einem Virtex-II-1000 195
Tabelle 7.4: Ressourcenbelegung des NoCs auf einem Virtex-II-1000 196
Tabelle 7.5: Ressourcenbelegung des Doppel-µBlaze-Systems auf dem FPGA 205
Tabelle 7.6: Ressourcenbelegung des PWM-Analysators 208

9.3 Formelverzeichnis

Formel 2.1: Beispiel einer CRC-Code-Berechnung (XOR) 11
Formel 2.2: Beispiel für ein Polynom zur CRC-Berechnung 11
Formel 2.3: Berechnung der CRC-Prüfsumme für das Payload-Segment 16
Formel 3.1: Verlustleistung eines Linearreglers (Näherung) 41
Formel 4.1: Abschätzung der Stromaufnahme der 3,3V-Spannungsschine 101
Formel 4.2: Abschätzung der Stromaufnahme der 3,3V-Spannungsschine für µP 101
Formel 4.3: Abschätzung der Stromaufnahme der 1,5V-Spannungsschine 101
Formel 4.4: Abschätzung der Stromaufnahme der 2,5V-Spannungsschine 101
Formel 4.5: Abschätzung der Stromaufnahme der 5,0V-Spannungsschine 101
Formel 4.6: Abschätzung der maximalen Gesamtleistungsaufnahme 103
Formel 5.1: Berechnung der CLB-Spalten bei 4-MULT/BRAM-Spalten 121
Formel 5.2: Berechnung der CLB-Spalten bei 6-MULT/BRAM-Spalten 121
Formel 5.3: Berechnung der CLB-Spalten für Virtex-II-1000 122
Formel 5.4: Berechnung der LIN-Clock-Prescalers 142
Formel 5.5: Berechnung der Frequenz der LIN-Clock 142
Formel 5.6: Berechnung der Bitzeit einer LIN-Übertragung 143
Formel 5.7: Berechnung der Standard-Bitzeit (100%) 143
Formel 6.1: Formel für PID-Regler (1) 179
Formel 6.2: Formel für PID-Regler (2) 179
Formel 6.3: Exponentialfunktion als Potenzreihe 181

9.4 Abkürzungsverzeichnis

9.4.1 Allgemeine Abkürzungen

µC	Mikrocontroller
µP	Mikroprozessor
ABS	Antiblockiersystem
AHB	Advances High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
API	Applikation Programming Interface
ASC	Asynchrone Serielle Schnittstelle
ASIC	Application Specific Integrated Circuit
BGA	Ball Grid Array
BIST	Built-In Self Test
BITSRB	Bit Strobing Process
BSS	Byte Start Sequence
BTL	Bit Time Length
CAN	Controller Area Network
CASE	Computer Aided Systems/Software Engineering
CC	Communication Controller
CE	Communication Element
CGROM	Character Generator ROM
CHI	Controller Host Interface
CLB	Complex Logic Block (FPGA)
CPLD	Complex Programmable Logic Device
CRC	Cyclic Redundancy Check
CSMA / CA	Carrier Sense Multiple Access / Collision Avoidance
DAC	Digital Analog Converter
DCM	Digital Clock Manager (FPGA)
DDRAM	Display Data RAM
DIP	Dual In-Line Package
DLC	Data Lenght Code
DR	Data Register
DSP	Digitaler Signal Prozessor
DUT	Device under Test
ECU	Electric Control Unit
EDK	Embedded Development Kit (Xilinx)
EEPROM	Electrical Erasable Programmable Read Only Memory
EMB	Elektromagnetische Bremse
EMC	Electromagnetic Compatibility
EMV	Elektromagnetische Verträglichkeit
EOF	End-Of-Frame oder End-Of-File
EPROM	Erasable Programmable Read Only Memory
ERRN	Error Not signal
ESP	Elektronisches Stabilitätsprogramm
FET	Field Effect Transistor
FIFO	First In First Out
FPAA	Field Programmable Analog Array
FPGA	Field Programmable Gate Array
FPLA	Field Programmable Logic Array
FPU	Floating Point Unit
FSM	Finite-State-Machine
FTDMA	Flexible Time Division Multiple Access
GAL	Generic Array Logic

GB	Giga Byte ≡ 1.073.741.824 Bytes
GNU	„GNU's not UNIX" – rekursives Akronym
GPIF	General Purpose Interface
GPIO	General Purpose Input / Output
GSM	Gleichstrommaschine
HiL	Hardware-in-the-Loop
HTML	Hyper Text Markup Language
I²C = IIC	Inter IC Connection, I^2C-Bus
ID	Identifier
IDE	Integrated Device Elektronics
IES-Schnittstelle	Information, Energie, Stoff Schnittstelle zur Außenwelt
INH	Inhibit signal
IP	Intellectual Property
IR	Instruction Register
ISE	Integrated Software Environment (= Project Navigator von Xilinx)
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
kB	Kilo Byte ≡ 1.024 Bytes
Kfz	Kraftfahrzeug
LAN	Local Area Network
LCD	Liquid Crystal Display
LECCS	LEON/ERC32 Cross-Compiling System
LIN	Local Interconnect Network
LSB	Least Significant Bit
LUT	Look Up Table (FPGA)
LVDS	Low Voltage Differential Signal
MAC	Media Access Control process
MB	Mega Byte ≡ 1.048.576 Bytes
MFC	Microsoft Foundation Classes
MOS	Metal-Oxide Semiconductor (Metalloxid Halbleiter)
MOST	Media Oriented Signal Transport
MSB	Most Significant Bit
MUT	Module under Test
MUX	Multiplexer
N-, P-, C-MOS	n-Kanal, p-Kanal, komplementäre MOS Logik
NM	Network Management
NRZ	Non-Return To Zero
NRZI	Non Return to Zero Inverted
NV-RAM	Non-Volatile Random Access Memory
OPB	On-Chip Peripheral Bus
OS	Betriebssystem (engl.: Operating System)
OSI	Open System Interconnection
PBlock	Physical Blocks (FPGA, PlanAhead-Tool, Xilinx)
PCB	Printed Circuit Boards (engl., Platine)
PCI	Peripheral Component Interconnect
PECL	Pseudo Emitter Coupled Logic
PLD	Programmable Logic Devices
PLL	Phase Locked Loop
PMU	Power-Management-Unit
POC	Protocol Operation Control
PROM	Programmable Read Only Memory
RAM	Random Access Memory
REC	Receiver Error Counter
ROM	Read Only Memory
RP	Rapid-Prototyping
RTR	Remote Transmit Request

RxD	Receives data signal from bus driver
RxEN	Receive data enable signal from bus driver
SCI	Serial Communication Interface
SDL	Specification and Description Language
SDRAM	Synchronous Dynamic Random Access Memory
SMD	Surface Mounted Device
SOF	Start-Of-Frame
SPARC	Scalable Processor Architecture
SPI	Serial Peripheral Interface
SSC	Synchrone Serielle Schnittstelle
SSRAM	Synchronous Static Random Access Memory
STBN	Standby Not Signal
STG	Steuergerät
SuF	Startup Frame
SyF	Sync Frame
TAP	Test-Access-Port
TBUF	Tri-State-Buffer (FPGA)
TDMA	Time Devision Multiple Access
TEC	Transmit Error Counter
TQ	Time Quantum
TQFP	Thin Quad-Flat-Pack
TRP	Time Reference Point
TTL	Transistor Transistor Logic
TxD	Transmit Data singal from CC
TxEN	Transmit Data Enable Not signal from CC
UART	Universal Asynchronous Receiver and Transmitter
Ucf	User Constraints File (FPGA)
UML	Unified Modelling Language
USB	Universal Serial Bus
VCO	Voltage Controlled Oscillator
VHDL	Very-High-Speed-Integrated Circuit Hardware Description Language
WUS	Wakeup Symbol
XMD	Xilinx Microprocessor Debugger
XML	eXtensibel Markup Language
XPS	Xilinx Platform Studio

9.4.2 Eigene Abkürzungen

COMPASS	**C**onfigurable **M**odular **P**latform for **A**utomotive **S**ystem**S**
CC	Configuration Channel
IB	Information Bus
IOB	I/O Bus
MIB	Module Interconnect Bus
PCB	Peripheral Communication Bus
PMU	Power Management Unit
SB	System Bus

9.5 Formelzeichen

R	Widerstand
L	Induktivität
C	Kapazität
U	Spannung, allgemein
I	Strom, allgemein
GND	Ground, Bezugsmasse
V_{DD}	Positive Versorgungsspannung
V_{CC}	Positive Versorgungsspannung
V_{SS}	Negative Versorgungsspannung
V_{EE}	Negative Versorgungsspannung
V_{CCINT}	Interne Versorgungsspannung (FPGA)
V_{CCAUX}	Hilfsversorgungsspannung (FPGA)
V_{CCO}	Versorgungsspannung Ausgangstreiber (FPGA)
V_{REF}	Referenzspannung (FPGA)
V_{BATT}	Batteriespannung (FPGA)
p(n), p(x)	Polynom
n, x	Bitstelle im Polynom
I_{M9_max}	Maximale Stromaufnahme des Micro9-Boards
$I_{VCCO\_S2\_max}$	Stromaufnahme der Spartan2-Ausgangstreiberstufen
$I_{VCCINT\_S2\_max}$	Stromaufnahme des Spartan2-Cores
$I_{VCCO\_VII\_max}$	Stromaufnahme der Virtex-II-Ausgangstreiberstufen
$I_{VCCINT\_VII\_max}$	Stromaufnahme des Virtex_II-Cores
I_{IK}	Stromaufnahme der Interface-Karte
$I_{IO\text{-}Karte}$	Stromaufnahme einer I/O-Karte
$I_{Auto\text{-}Karte}$	Stromaufnahme einer Automotive-Karte
I_{CLKGen}	Stromaufnahme des Taktgenerators
I_{EEPROM}	Stromaufnahme eines EEPROMs
I_{SRAM}	Stromaufnahme eines SRAMs (auf der Interface-Karte)
I_{CF}	Stromaufnahme einer CompactFlash-Karte
I_{RS232}	Stromaufnahme der RS232-Schnittstelle
I_{CAN}	Stromaufnahme der CAN-Schnittstelle
$I_{1.5V}$	Stromaufnahme der 1,5V-Versorgungsschine
$I_{2.5V}$	Stromaufnahme der 2,5V-Versorgungsschine
$I_{3.3V}$	Stromaufnahme der 3,3V-Versorgungsschine
$I_{5.0V}$	Stromaufnahme der 5,0V-Versorgungsschine
P, P_{ges}	Gesamtleistungsaufnahme der Plattform
CLBSp	CLB-Spalte
$f_{prescale\_clock}$	Frequenz des Vorteilers
f_{clock}	Taktfrequenz
precale_factor	Vorteilfaktor
$T_{prescale\_clock}$	Periode des Vorteilertakts
T_{clock}	Taktperiode
f_{lin_clock}	Frequenz des LIN-Taktes
bit_factor	Bit-Breite (einstellbar)
T_{lin_clk}	Periode LIN-Takts
T_{bit}	Bitperiode der LIN-Übertragung
K_P	Verstärkungsfaktor Proportionalglied
K_I	Verstärkungsfaktor Integralglied
K_D	Verstärkungsfaktor Differenzialglied
T_N	Nachstellzeit
T_V	Vorhaltezeit

9.6 Literaturverzeichnis

[1Wir02] Maxim-Dallas: “Overview of 1-Wire Technology and Its Use”; Application Note 1796, 2002.

[25LC160A] Microchip Technology Inc.: “16K SPI Bus Serial EEPROM 25LC160A”; Okt 2003. – http://ww1.microchip.com/downloads/en/devicedoc/21807b.pdf

[Acte04] Actel „Flash in Fahrt – Automobil-FPGAs“, Design&Elektronik, Seite 17, Heft 2, März 2004

[AD8139] Analog Devices, Inc.; “Low Noise Rail-to-Rail Differential ADC Driver AD8139”; Rev. 8/2004 edition, Dec 2004, http://www.analog.com/UploadedFiles/Data_Sheets/127730884AD8139_a.pdf.

[AD9430] Analog Devices, Inc.; “AD9430BSV-210 Data Sheet”, C edition, Nov 2004.

[AlBl06] Ali, A.; Blath, J.P.; “Nonlinear torque control of a spark-ignited engine”; American Control Conference 2006, 14-16 June 2006.

[AMBA20] ARM Limited: “AMBA Specification (Rev. 2.0)”, ARM (www.arm.com), May 1999

[ARML02] ARM-Linux: Homepage, 2002, http://www.arm-linux.org

[Atme05] Datasheet LIN, Transceiver, http://www.atmel.com, May 2005.

[Axel01] Axelson, Jan: „USB, Handbuch für Entwickler“; MITP-Verlag, Bonn, 2001

[Baci05] Bacic, M.; “On hardware-in-the-loop simulation”; 44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference CDC-ECC '05, 12-15 Dec. 2005.

[Bahl06] Bahlinger, Martin; „Entwicklung einer Steuerung für eine Rapid-Prototyping- und Hardware-in-the-Loop-Plattform unter Linux und Anbindung an einen Windows-Host-PC über Ethernet“; Diplomarbeit, Institut für Technik der Informationsverarbeitung, Universität Karlsruhe, 2006.

[BaKA05] Banovic, K.; Khalid, M.A.S.; Abdel-Raheem, E.; “FPGA-based rapid prototyping of digital signal processing systems”; 48th Midwest Symposium on Circuits and Systems, 2005, 7-10 Aug. 2005.

[Baue01] Bauer, Horst und weiter Autoren, „Sensoren im Kraftfahrzeug“; Gelbe Reihe, Fachwissen Kfz-Technik, Robert Bosch GmbH, Ausgabe 2001

[BCCG06] Balasubramanian, M.; Chaturvedi, N.; Chowdhury, A.D.; Ganesh, A.; “A framework for rapid-prototyping of context based ubiquitous computing applications”; IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006, June 2006.

[Beck01] Becker, Jürgen; Vorlesungsskript, Unterlagen zur Vorlesung „Entwurf elektronischer Systeme II (Synthese, Optimierung, Layout)“; 2001

[Beck05] Becker, Jürgen; Vorlesungsskript, Unterlagen zur Vorlesung „Grundlagen Digitaltechnik“; 2005.

[BeSv06] Bengtsson, J.; Svensson, B.; “A configurable framework for stream programming exploration in baseband applications”; Parallel and Distributed Processing Symposium IPDPS 2006, 2006.

[Beye03] Beyer, Marco; „Eine FPGA/DSP-Entwicklungsplattform für eingebettete audiosignalverarbeitende Echtzeitsysteme“; Diplomarbeit, Technische Universität Berlin, 2003.

[BHN+04] Bobda, C. ; Huebner, M.; Niyonkuru, A. ; Blodget, Brandon ; Majer, Mateusz ; Ahmadinia, Ali: „Designing Partial Dynamically Reconfigurable Applications on Xilinx Virtex-II-FPGAs using Handel-C“; University of Erlangen-Nürnberg, Department of Computer Science 12, Hardware- Software-Co-Design. 2004.

[Bies01] Bieser, Carsten; „Entwicklung einer FPGA–Interfaceeinheit zur digitalen Signalvorverarbeitung für das Rapid Prototyping eingebetteter Systeme mit Embedded Linux Architektur“; Institut für Technik der Informationsverarbeitung,Universität Karlsruhe (TH), Diplomarbeit, 2001.

[Bobd05] Bobda, C.; Ahmadinia, A.; Rajesham, K.; Majer, M.; “Partial Configuration Design and Implementation Challenges on Xilinx Virtex FPGAs”, Architecture of Computing Systems (ARCS), 14-17 March, 2005.

[Boeh76] Boehm, B.: “Software Engineering”; IEEE Transactions on Computers, Vol. 25, 1976.

[Boeh88] Boehm, Barry: “A Spiral Model of Software Development and Enhancement”; IEEE Computer, Vol.21, Ausg. 5, Mai 1988, pp 61-72.

[BoIS06] Bona, B.; Indri, M.; Smaldone, N.; “Rapid Prototyping of a Model-Based Control With Friction Compensation for a Direct-Drive Robot”; IEEE/ASME Transactions on Mechatronics, Volume: 11, Oct. 2006.

[Bort05] Bortolazzi, Jürgen: “Systems Engineering for Automotive Electronics: Part 3 Target Architectures Networking CAN”, Universität Karlsruhe, ITIV, 2005

[Bosc] Bosch; “Description of the CAN bus protocol and the CAN-IP-Core”, Bosch GmbH, Germany, www.can.bosch.com

[Bosc01] Robert Bosch GmbH: „Sensoren im Kraftfahrzeug“, Ausgabe 2001, Gelbe Reihe, Juni 2001

[Bosc02] Robert Bosch GmbH: „Autoelektrik/Autoelektronik“, 4. Auflage, Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 2002

[Bosc02a] Robert Bosch GmbH: „Erzeugnisprogramm Sensoren“, 2001/2002

[Bosc91] Robert Bosch GmbH: „CAN Specification Version 2.0“; Robert Bosch GmbH, Stuttgart, 1991.

[Brei01] Breitmaier, Bernd: „Entwicklung einer Ein/Ausgabe-Einheit mit PCI Interface für das Rapid Prototyping von elektronischen Steuergeräten mit Embedded Linux PCs“; Diplomarbeit, Universität Karlsruhe (TH), ITIV, 2001

[Brin05] Brinkmeyer, H.; “A new approach to component testing [automotive electronics]”; Design, Automation and Test in Europe, 2005.

[Bron95] Bronstein, I.; Semendjajew, K. A.; Musiol, G.; Mühlig, H.; „Taschenbuch der Mathematik“; 2. Auflage,Verlag Harri Deutsch, Thun, 1995

[Bros02] Broster, I.; Burns, A.; Rodriguez-Navas, G.; “Probabilistic analysis of CAN with faults”, 23rd IEEE Real-Time Systems Symposium (RTSS 2002), 3-5 Dec. 2002 Page(s):269 - 278

[Brug02] Brugger, Daniel: „Entwicklung einer Backplane mit FPGA- und Ethernet-Interface für ein minimales Rapid-Prototyping-System mit einem Prozessor INTEL StrongArm unter Embedded-LINUX“; Diplomarbeit, Universität Karlsruhe (TH), ITIV, 2002

[BrvH99a] Brunetti, Jon; Von Herzen, Brian; “The LVDS I/O Standard”, Xilinx, Inc., 1.1 Edition, Nov 1999, http://www.xilinx.com/bvdocs/appnotes/xapp230.pdf.

[BrvH99b] Brunetti, Jon; Von Herzen, Brian; “Virtex-E LVDS Drivers & Receivers: Interface Guidelines”; Xilinx, Inc., 1.0 edition, Oct 1999, http://www.xilinx.com/bvdocs/appnotes/xapp232.pdf.

[BSWM99] Burst, A.; Spitzer, B.; Wolff, M.; Mueller-Glaser, K.D.: “Interface technologies for versatile rapid-prototyping systems”; Workshop on Rapid System Prototyping (RSP), June 1999

[Buri06] Burian, Michael: “Linux 2.6 for Cirrus EP93xx CPUs”; 2006. http://www.kernel.org/git/?p=linux/kernel/git/mburian/linux-2.6-ep93xx.git

[Burs00] Burst, Alexander; „Rapid Prototyping eingebetteter elektronischer Systeme auf Basis des CDIF-Datenaustauschformats“; Institut für Technik der Informationsverarbeitung, Universität Karlsruhe (TH), Dissertation, 2000.

[CaAl04] Carter, D.E.; Alleyne, A.G.; “Earthmoving vehicle powertrain controller design and evaluation”; Proceedings of the American Control Conference, 2004, 30 June-2 July 2004.

[CaGo02] Carmichael, Carl; Goldblatt, Kim: “Configuration and Readback of the Spartan-II and Spartan-IIE Families”; Xilinx, Inc., 1.0 edition, Mar 2002. http://www.xilinx.com/bvdocs/appnotes/xapp176.pdf.

[CAN20] Robert Bosch GmbH: “CAN Specification Version 2.0”; Robert Bosch GmbH, Stuttgart, 1991

[CGD+05] Cebi, A.; Guvenc, L.; Demirci, M.; Karadeniz, C.K.; Kanar, K.; Guraslan, E.; “A Low Cost, Portable Engine Electronic Control Unit Hardware-in-the-Loop Test System”; Proceedings of the IEEE International Symposium on Industrial Electronics ISIE 2005, June 20-23, 2005.

[Chan02] Chang, Jeff: “USB 2.0 – It Just Works…”; Cypress Semiconductor, 26. September 2002, http://www.analogzone.com/iot_0916.htm

[CiA06] CAN in Automation (CiA), Internationale Organisation, Webseite (www.can-cia.org), 2006.

[Cirr04] Cirrus Logic, Inc. EP9315 User’s Guide, 1 edition, Feb 2004.

[CKR+03] Chang, Chen; Kuusilinna, Kimmo; Richards, Brian; Chen, Allen; Chan, Nathan; Broderson, Robert W.; Nikolic, Borivoje: “Rapid design and analysis of Communication Systems Using the BEE Hardware Emulation Environment”, 14th IEEE International Workshop on Rapid Prototyping (RSP) 2003

[Clar04] Clarke, Edward; „FPGAs - Programmierbare Systeme auf einem Chip“; Elektronik, 5/2004.

[CoHu93] Cooling, J.; Hughes, T.: “Animation Prototyping of Real-Time Embedded Systems”; Microprocessors and Microsystems, No.6, 1993.

[Cont05] Contec GmbH, “Micro-9 Microprocessor Module with ARM9”; Data Sheet, User Guide and Eval-Board Description, Austria, 2005, http://www.contec.at

[Cont05a] Contec GmbH: “Handbuch Evaluation Board Hypercontrol MICRO-9”; 1.2 edition, Aug 2005.

[Cont05b] Contec GmbH: “Handbuch Linux BSP V1.0 Hypercontrol Micro-9”; 1.3 edition, Aug 2005.

[Cont05c] Contec GmbH: “Handbuch Micro-Board-Computer Hypercontrol MICRO-9”; 1.1 edition, Aug 2005.

[Croc03] Crocoll, E.; Skript elektronische Schaltungen, 2003

[DAC5675] Texas Instruments, Inc.; "Digital-to-Analog Converter DAC5675"; Rev. A Edition, 2001, http://focus.ti.com/lit/ds/symlink/dac5675a.pdf.

[Deco04] Decomsys, NODE<ARM>, www.decomsys.com, Austria, 2004

[DeEl05] Design&Elektronik: „Begleittexte zum Entwicklerforum Kfz-Elektronik"; Ludwigsburg;11.Mai 2005.

[Demb93] Dembowski, Klaus; „Computerschnittstellen und Bussysteme" Markt&Technik,Buch – und Software–Verlag GmbH & Co., 1993.

[Dieb04] Diebenbusch, Klaus; „Robust und kostengünstig"; Elektronik, 2/2004.

[DJZ+05] Song Dafeng; Li Jing; Ma Zhimin; Li Youde; Zhao Jian; Liu Wei ; "Application of CAN in vehicle traction control system"; IEEE International Conference on Vehicular Electronics and Safety, 2005, 14-16 Oct. 2005.

[DLHS05] Davies, N.; Landay, J.; Hudson, S.; Schmidt, A.; "Guest Editors' Introduction: Rapid Prototyping for Ubiquitous Computing"; Pervasive Computing, Volume: 4, Oct.-Dec. 2005.

[DoSG05] Dorairaj, Nij ; Shiflet, Eric ; Goosman, Mark: "PlanAhead Software as a Platform for Partial Reconfiguration"; In: XCell Journal (2005), Nr. Fourth Quarter

[Drei06] Dreier, Rico; „Verteilte Ausführung hybrider System-Modelle auf heterogenen Rechnerplattformen", Dissertation, Universität Karlsruhe (TH), 2006.

[dSPA05] dSPACE, "MicroAutoBox"; www.dspace.com, Germany, 2005

[dSPA06] dSPACE GmbH; Gesamtkatalog 2006, 2006.

[DyPP02] Dyer, Matthias; Plessl, Christian ; Platzner, Marco: "Partially Reconfigurable Cores for Xilinx Virtex"; Computer Engineering and Networks Lab, Swiss Federal Institute of Technology, ETH Zürich.

[DyWi02] Dyer, Matthias; Wirz, Marco: "Reconfigurable System on FPGA", Eidgenössische Technische Hochschule Zürich (ETH), Diplomarbeit, 2002

[Elme02] Elmenreich, W.; Delvai, M.; "Time-triggered communication with UARTs", 4th IEEE International Workshop on Factory Communication Systems, 2002, Page(s):97 - 104

[Enge00] Engels, Horst: "CAN-Bus", Franzis Verlag, 2000.

[EP9315] Cirrus Logic: "Data Sheet EP9315 Universal Platform System-On-Chip Processor"

[ESDL01] Embedded Systems Design Laboratory: "HD44780 LCD Starter Guide"; Handout #7, October 2001

[ETAS05] ETAS, ES-1000 Series, www.etas.de, Germany, 2005

[ETAS06] ETAS GmbH; Gesamtkatalog 2006/2007, 2006.

[Etsc01] Etschberger, Konrad: "Controller Area Network - Basics, Protocols, Chips and Applications"; IXXAT Automation, 2001

[Etsc02] Etschberger, Konrad (Hrsg.): „CAN Controller Area Network: Grundlagen, Protokolle, Bausteine, Anwendungen"; 3. Auflage, Carl Hanser Verlag, München Wien, 2002

[Fabe97] Faber, Reinhard: „Entwicklung eines dynamischen Modells für einen Kraftfahrzeug-Fensterheber"; Diplomarbeit, Institut für Technik der Informationsverarbeitung, Universität Karlsruhe, 1997.

[Fanu03] Fanucci, L.; Giambastiani, A.; Rosadini, C., “VLSI design and verification methodologies for automotive embedded systems”; International Symposium on Signals, Circuits and Systems, 2003. SCS 2003, Volume 1, 10-11 July 2003 Page(s):261 - 264 vol.1

[FBCH04] Fang Pang; Brandon, T.; Cockburn, B.; Hume, M.; “A reconfigurable digital IC tester implemented using the ARM Integrator rapid prototyping system”; Canadian Conference on Electrical and Computer Engineering, 2004, 2-5 May 2004.

[FCK+05] Fitton, D.; Cheverst, K.; Kray, C.; Dix, A.; Rouncefield, M.; Saslis-Lagoudakis, G.; “Rapid prototyping and user-centered design of interactive display-based systems”; Pervasive Computing, Volume: 4, Oct.-Dec. 2005.

[Fisc88] Fischer, J.: „Softwaretechnologie ’Rapid Prototyping’ bei der Entwicklung von Kommunikationssoftware verteilter sowie eingebetteter Systeme“; Dissertation, Humboldt-Universität Berlin, 1988.

[Flex05] FlexRay Communications System: „Protocol Specification V2.1”; 12. Mai, 2005.

[FoHA] Fong, Ryan ; Harper, Scott ; Athanas, Peter: “A Versatile Framework for FPGA Field Updates: An Application of Partial Self-Reconfiguration”; Virginia Polytechnic Institute and State University, Bradley Dept of ECE, Blacksburg.

[Folb99] Folberth, Dieter: „Entwicklung und Aufbau eines CAN-Demonetzwerks mit Automotive-Anwendungen“; Diplomarbeit, FH Regensburg, Bereich Elektrotechnik, 1999

[Fong03] Fong, R.; Harper, S.; Athanas, P.; “A Versatile Framework for FPGA Field Updates: An Application of Partial Self-Reconfiguration”; Rapid Systems Prototyping, 2003. Proceedings. 14th IEEE International Workshop on, 9-11 June 2003, Page(s): 117 - 123.

[Frau03] ViSEK-Form; „Systematische Testen von Software“; Fraunhofer, Institut für Rechnerarchitektur und Softwaretechnik; 01.07.2003

[Gabr03] Gabriel, C.; Horia, H.; “Integrating Sensor Devices in a LIN bus network“, 26th International Spring Seminar on Electronics Technology, May 2003.

[Gais01] Gaisler, Jiri: “A portable fault-tolerant microprocessor based on the SPARC V8 architecture”; European Space Research and Technology Centre (ESTEC)

[Gais01a] Gaisler, Jiri: “The LEON Processor User´s Manual”; Handbuch, Gaisler Research, Version 2.4.0, November 2001.

[Gais02] Gaisler, Jiri: “The LEON/ERC32 GNU Cross-Compiler System”; Handbuch, Gaisler Research, Version 1.1.5, July 2002

[Geer94] Prof. Dr.-Ing. Geering, Hans P.; „Regelungtechnik 3“; neubearbeitete und erweiterte Auflage, Springer Lehrbuch, 1994

[Gnab02] Gnabasik, David: “Architectural I/O Improvements in the Intel StrongARM Processors”; 2002, http://ouray.cudenver.edu/~dfgnabas/CSC5593Project.doc

[GPIF02] Cypress Semiconductor; “EZ-USB® FX2™ GPIF Primer”; Cypress Semiconductor, San Jose, 2002

[Grab01] Grabbe, Cornelia: „Synthese regelmäßiger Strukturen für FPGAs mittels der JBits-API“; Universität Paderborn, Fachbereich Elektrotechnik und Informationstechnik, Fachgebiet Datentechnik, Diplomarbeit, 2001

[Grze03a] Grzemba, Andreas; „LIN-Bus - Die Technologie; Teil 1: Netzwerkarchitektur mechatronischer Systeme und Übersicht über die Technologie sowie das Protokoll“; Elektronik Automotive, 4/2003.

[Grze03b] Grzemba, Andreas; „LIN-Bus - Die Technologie; Teil 2: Fehlererkennung und Fehlerbehandlung, Netzwerk-Management, Bitübertragungsschicht“; Elektronik Automotive, 5/2003.

[Grze03c] Grzemba, Andreas; „LIN-Bus - Die Technologie; Teil 3: Systementwurf und LIN Configuration Language (Konfigurationssprache)“; Elektronik Automotive, 6/2003.

[Grze04a] Grzemba, Andreas; „LIN-Bus - Die Technologie; Teil 4: Hardware - Transceiver und Controller“; Elektronik Automotive, 1/2004.

[Grze04b] Grzemba, Andreas;„LIN-Bus - Die Technologie; Teil 5: Entwicklungsprozess und Entwicklungswerkzeuge“; Elektronik Automotive, 2/2004.

[Grze04c] Grzemba, Andreas; „LIN-Bus - Die Technologie; Teil 6: Test von LIN-Systemen“; Elektronik Automotive, 3/2004.

[GuLe98] Guccione, Steven A.; Levi, Delon: “JBits: A Java-Based Interface to FPGA Hardware”; Xilinx.

[GuLS99] Guccione, S. A.; Levi, D.; Sundararajan, P.: JBits: “A Java-based Interface for Reconfigurable Computing”; 2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD), September 1999.

[GVLO04] Gietelink, O.J.; Verburg, D.J.; Labibes, K.; Oostendorp, A.F.; “Pre-crash system validation with PRESCAN and VEHIL”; Intelligent Vehicles Symposium, 2004, 14-17 June 2004.

[HD44780] Hitachi: HD44780 (LCD-II): “Dot Matrix Liquid Crystal Display Controller/Driver”; Hitachi

[Hech01] Hecht, Hartmut; Vorlesungsskript; Unterlagen zur Vorlesung „Management für Ingenieure“; 2001

[Hech02] Hecht, Roland: “Message 1897: LCD connected to LEON”; LEON SPARC Group, March 2002

[Hein06] Heinz, Matthias; „Entwicklung eines flexiblen µController-Systems zur Ausführung von Rapid-Prototyping- und Hardware-in-the-Loop-Szenarien“; Diplomarbeit, Institut für Technik der Informationsverarbeitung, Universität Karlsruhe, 2006.

[Himp00] Himpel, G.: ITGlossar, Stichwortindex. DAASeelow, 2000, http://www.quie.de/

[Holl00] Holleczek Peter, „PEARL 2000, Echtzeitbetriebssysteme und LINUX“; Springer Verlag, 2000.

[HoLo04] Horta, Edson L.; Lockwood, John W.: “Automated Method to Generate Bitstream Intellectual Property Cores for Virtex FPGAs / Department of Electronic Engineering, Laboratory of Integrated Systems (EPSUP)”; Department of Computer Science, Applied Research Lab, Washington University, 2004.

[Hsue97] Hsueh, M.; Tsai, T.; Iyer, R.; “Fault Injection Techniques and Tools”, IEEE Computer, pp. 75-82, April 1997

[Huba99] Hubatsch, Maximilian: „Weiterentwicklung und Aufbau des CAN-Demonetzwerks mit Automotive-Anwendungen“; Diplomarbeit, FH Regensburg, Bereich Elektrotechnik, 1999

[HüBB04] Hübner, Michael ; Becker, Tobias ; Becker, Jürgen: „Real-Time LUT-Based Network Topologies for Dynamic and Partial FPGA Self-Reconfiguration“; Universität Karlsruhe (TH), Institut für Technik der Informationsverarbeitung (ITIV). 2004.

[Hübn04] Huebner, M.; Ullmann, M.; Weissel, F.; Becker, J.: “Real-time configuration code decompression for dynamic FPGA self-reconfiguration”; Proceedings of 18th International Parallel and Distributed Processing Symposium, 2004. 26-30 April 2004 Page(s):138

[HUWB04] Huebner, Michael ; Ullmann, Michael ; Weissel, Florian ; Becker, Jürgen: “Real-time Configuration Code Decompression for Dynamic FPGA Self-Reconfiguration”; Universität Karlsruhe (TH), Institut für Technik der Informationsverarbeitung (ITIV), 2004

[I2CB00] Philips; “The I2C-Bus Specification”; Version 2.1, January 2000.

[IABG06] IABG: „Das V-Modell und das V-Modell XT“; http://www.v-modelliabg.de, 2006.

[Infi00] Infineon: “C167CR User’s Manual: Chapter 18 – The On-chip CAN Interface”; Handbuch, Infineon Technologies, Version 3.1, March 2000

[Inte01a] Intel; “Intel StrongARM Microprocessor”; Developer’s Manual, http://www.intel.com, 2001.

[Inte01b] Intel: “High Speed USB Platform Design Guidelines”; Intel Corporation, 2001, http://www.usb.org/developers/docs/hs_usb_pdg_r1_0.pdf

[Inte01c] Intel / Microsoft: “PC 2001 System Design Guide Version 1.0.”; 04. September 2001, http://www.pcdesguide.org/pc2001/default.htm

[Inte03] Intersil: “HIP4081, 80VHigh Frequency H-Bridge Driver”; Application Note, Intersil Corporation, February 2003

[Inte96] Intersil: “HIP4081: 80V/2.5A Peak, High Frequency Full Bridge FET Driver”; Intersil Corporation, November 1996

[Inte99] Intel / Microsoft: “PC 99 System Design Guide Version 1.0”; 29. September 1999, http://www.pcdesguide.org/pc99/default.htm

[IPG06] IPG Automotive GmbH; Webseite (www.ipg.de); 2006.

[IRF4905S] International Rectifier. Datenblatt IRF4905S, 1997.

[IRF4905S] International Rectifier; Datenblatt IRF4905S, 1997.

[IRFR5305] International Rectifier. Datenblatt IRFR5305, 2000.

[Iser99] Isermann, R.: „Mechatronische Systeme“; Springer-Verlag, 1999

[JaMI05] Jamieson, C.; Melvin, S.; Ilow, J.; “Rapid prototyping hardware platforms for the development and testing of OFDM based communication systems”; Proceedings of the 3rd Annual Communication Networks and Services Research Conference 2005, 16-18 May 2005.

[Jung04] Jung, Markus; „Modellierung von Kfz-Aktoren und -Sensoren in MATLAB SimuLink zur Simulation auf einem Xilinx-MicroBlaze-Prozessor in Hardware-in-the-Loop-Tests“; Studienarbeit, Institut für Technik der Informationsverarbeitung, Universität Karlsruhe, 2004.

[Just03] Justen, D., Wiench, R., Sechtenbeck, C. „Flexible Motorsteuerung für Forschung und Prototyping“, Elektronik Automotive-Sonderheft, Ausgabe 3/2003

[Just04] Justen, D., Brückner, I. „Offen für Neues – FPGAs machen Entwicklungsplattform flexibel“, Design&Elektronik, Seite 22, Heft 2, März 2004

[K6R4016V1D] SAMSUNG Electronics CO., LTD.: “256Kx16 Bit High Speed Static RAM (3.3V Operating) K6R4016V1D-TC1000”; Rev 0.3. Okt 2001. – http://www.samsung.com/Products/Semiconductor/HighSpeedSRAM/AsyncFastSRAM/4Mbit/K6R4016V1D/ds_k6r4016v1d_rev40.pdf

[KaSe06] Karpenko, M.; Sepehri, N.; “Hardware-in-the-loop simulator for research on fault tolerant control of electro-hydraulic flight control systems”; American Control Conference, 2006,14-16 June 2006.

[KBBD05] Van den Keybus, J.; Bolsens, B.; De Brabandere, K.; Driesen, J.; “Protection of digitally controlled inverter units in rapid prototyping applications”; Twentieth Annual IEEE Applied Power Electronics Conference and Exposition APEC 2005, 6-10 March 2005.

[KeilCAN] Keil: “C167CR CAN Application Note: Low-level CAN functions”; Keil Software, 1997

[Kelm00] Kelm, Hans Joachim (Hrsg.): „USB 1.1, Universal Serial Bus, Datendienste, Errorhandling, Powermanagement, USB Treiber-Struktur, USB Funktions-Bausteine, USB Applikation“; Franzis-Verlag GmbH, Poing, 2000

[Kelm01] Kelm, Hans-Joachim: “Professional Series – USB 2.0”, Franzis’ Verlag GmbH, Poing, 2001

[KeRi90] Kernighan, Brian W.; Ritchie, Dennis M.; „Programmieren in C“; Hanser Fachbuch, 1990.

[Keyb02] Van den Keybus, J.; Bolsens, B.; De Brabandere, K.;Driesen, J.; Belmans, R.; „DSP and FPGA based platform for rapid prototyping of power electronic converters and ist applications to a sampled-data three phase dual-band hysteresis current controller“, 33rd Annual Power Electronics Specialists Conference (PESC 04), 2002; Volume 4, 23-27 June 2002 Page(s): 1722 - 1727

[KiCo04] King, P.J.; Copp, D.G.; “Hardware in the loop for automotive vehicle control systems development”; Mini Symposia UKACC Control 2004, 8 Sept. 2004.

[KiDa00] Kirch, Olaf; Dawson, Terry: “Linux Network Administrator’s Guide”; O’Reilly Verlag, 2 edition, 2000. http://www.oreilly.de/catalog/linag2/book/

[Kien98] Kiencke, U.; „Signale und Systeme 1“. Auflage, Oldenbourg Verlag, 1998

[King] Kingmax; “KINGMAX CompactFlash Card Specification”.

[KJD+06] Kamat, S.S.; Javaherian, H.; Diwanji, V.V.; Smith, J.G.; Madhavan, K.P.; “Virtual air-fuel ratio sensors for engine control and diagnostics”; American Control Conference, 2006, 14-16 June 2006.

[KJTR05] Krasteva, Yana E. ; Jimeno, Ana B. ; de la Torre, Eduardo ; Riesgo, Teresa: “Straight Method for Reallocation of Complex Cores by Dynamic Reconfiguration in Virtex II FPGAs”; Electronic Engineering Division, Universidad Politécnica de Madrid, 2005

[Klau03] Klausmann, Achim; „Entwicklung einer Kaffeeautomatensteuerung unter VHDL für den Einsatz auf einem Standard-FPGA im Praktikum Entwurfsautomatisierung“; Studienarbeit, Institut für Technik der Informationsverarbeitung, Universität Karlsrhe, 2003.

[Klim05] Klimm, Alexander; „Automatisierung des Design-Flows beim Entwurf von Systemmodellen unter Verwendung des SystemGenerators von Xilinx“;

Studienarbeit, Institut für Technik der Informationsverarbeitung, Universität Karlsrhe, 2005.

[Klon04] Klonne, A.; Schmitt, G.; Jarvelainen, T.; Nipp, E., “Systematic Approach of Rapid Prototyping for EC Drives in Automotive Applications”, 35th Annual Power Electronics Specialists Conference (PESC 04), 2004, 2004 IEEE; Volume 3, 20-25 June 2004 Page(s): 2245 - 2250 Vol.3

[KLV+02] Kalte, H. ; Langen, D. ; Vonnahme, E. ; Brinkmann, A. ; Rückert, U.: “Dynamically Reconfigurable System-on-Programmable-Chip”; University of Paderborn, Heinz Nixdorf Institute, System and Circuit Technology. 2002.

[Kögl00] Kögl, Markus: „Realisierung eines Fensterhebers für ein KFZ mit CAN-Anbindung und LIN-Bus auf dem XC68HC912D60 Microcontroller“; Diplomarbeit, FH Regensburg, Bereich Elektrotechnik, 2000

[Koop98] Koopman, P.; Tran, E.; Hendrey, G.; „Toward Middleware Fault Injection for Automotive Networks“, 28th International Symposium on Fault-Tolerant Computing Systems (FTCS ’98), 1998.

[KoTe] Koch, Dirk ; Teich, Jürgen: „Platform-Independent Methodology for Partial Reconfiguration“; University of Erlangen-Nürnberg

[Kotz04] Kotzsch, Vincent: „Entwurfsautomatisierung von internen Kommunikationsschnittstellen in dynamisch re-konfigurierbaren Systems On Chips (SoCs)“, Hochschule für Technik und Wirtschaft Dresden (FH), Fachbereich Elektrotechnik, Diplomarbeit, 2004

[Kram06] Kramer, David: „Entwicklung und Evaluierung einer Hardware-Infrastruktur für dynamische FPGA-basierte Prozessorbefehlssatzerweiterung unter Berücksichtigung verschiedener Erweiterungsmodelle“; Universität Karlsruhe (TH), Fakultät für Informatik, Institut für Technische Informatik (ITEC), Diplomarbeit, 2006.

[KrFu02] Krassin, Eugen; Fuhrmann, Jürgen: „Kompletter Baukasten“, Elektronik, Juni 2002.

[KRM+06] Kitts, C.; Rasay, M.; Mas, I.; Mall, P.J.; Buskirk, T.V.; “Model-based anomaly management for spacecraft mission operations at the NASA Ames Space Technology Center”; Second IEEE International Conference on Space Mission Challenges for Information Technology SMC-IT 2006, 17-20 July 2006.

[KSJN04] Klonne, A.; Schmitt, G.; Jarvelainen, T.; Nipp, E.; “Systematic approach of rapid prototyping for EC drives in automotive applications”; IEEE 35th Annual Power Electronics Specialists Conference PESC 04, 20-25 June 2004.

[KSSK06] Kuehn, S.; Schmid, T.; Schmid, D.; Kuster, N.; “Fast dosimetric assessment system for R&D of antennas and transmitter systems”; Antennas and Propagation Society International Symposium 2006, 9-14 July 2006.

[Larc02] Larcher, P.; “Designing a Compact and Flexible LIN Controller”, Cypress Application Note AN2045.

[Lawr02] Lawrenz, Wolfhard (Hrsg.): „CAN Controller Area Network - Grundlagen und Praxis“; 4. Auflage, Hüthig-Verlag, Heidelberg, 2002.

[Lawr97] Lawrenz, Wolfhard: “CAN System Engineering: From Theory to Practical Applications”; Springer-Verlag, 1997.

[LeKJ06] Lemmer, L.; Kiss, B.; Janosi, I.; “Modeling Identification, and Control of Harmonic Drives for Automotive Applications”; International Conference on Intelligent Engineering Systems, INES '06, 26-28 June 2006.

[LEON01] Gaisler, Jiri: “The LEON Processor User´s Manual”; Handbuch, Gaisler Research, Version 2.4.0, November 2001

[LeWS94] Lehmann, Gunther; Wunder, Bernhard; Selz, Manfred: „Schaltungsdesign mit VHDL“; Franzis' Verlag, Poing, 1994

[LeZb02] Lerjen, Michael ; Zbinden, Christian: „Reconfigurable Bluetooth Ethernet Bridge“; Eidgenössische Technische Hochschule Zürich (ETH), Diplomarbeit, 2002

[LINS13] LIN-Specification Package Revision 1.3, LIN-Consortium 2003, http://www.lin.org

[LINS20] LIN-Specification Package Revision 2.0, LIN-Consortium 2003, http://www.lin.org

[Lins98] Linsmeier, K.-D.: „Sensorsysteme für das Auto“; Verlag moderne Industrie, 1998

[Lipp02] Lipp, H. M.; „Technik und Protokolle von Bussystemen Skript zur Vorlesung“, ITIV, Universität Karlsruhe (TH), 2002

[LIV203] Labor der Informationsverarbeitung (LIV): Versuch 2: „Steuerung eines Fensterhebers“; Versuchsunterlagen, Universität Karlsruhe, ITIV, 2003

[LiYa06] Dong Lin, Shiyuan Yang; “An Implementation of Rapid Prototyping Platform of Embedded Systems”; IEEE Tenth International Symposium on Consumer Electronics, ISCE '06, June 2006.

[LiYM06] Ling, K.V.; Yue, S.P.; Maciejowski, J.M.; “A FPGA implementation of model predictive control”; American Control Conference, 14-16 June 2006.

[LM117] National Semiconductor. LM117/LM317A/LM317 3-Terminal Adjustable Regulator, Jun 2005.

[LPQC05] Luo, J.; Pattipati, K.R.; Qiao, L.; Chigusa, S.; “Towards an integrated diagnostic development process for automotive systems”; IEEE International Conference on Systems, Man and Cybernetics 2005, 10-12 Oct. 2005.

[LT1004] Texas Instruments, Inc, LT1004-1.2, LT1004-2.5 Micropower Integrated Voltage References, 2003

[LuWM05] Lu, B.; Wu, X.; Monti, A.; “Implementation of a low-cost real-time virtue test bed for hardware-in-the-loop testing”; 32nd Annual Conference of Industrial Electronics Society IECON 2005, 6-10 Nov. 2005.

[MaCo04] Manjikian, N.; Cordeiro, N.; “A user-configurable framework for testing prototype system-on-chip implementations”; The 2nd Annual IEEE Northeast Workshop on Circuits and Systems NEWCAS 2004, 20-23 June 2004.

[Mahm96] Mahmoud, Qusay H.: “Sockets programming in Java: A tutorial. Java world”; Dec 1996. http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets p.html.

[Math04] The MathWorks, “Real-Time Workshop Embedded Coder User Guide”, Version 3.2.1, April 2004

[Math13] The Mathworks, “MatLab/Simulink/Stateflow”, Release 13

[Math97] MathWorks: “StateFLOW for Use with Simulink”, User´s Guide Version 1, MathWorks, May 1997

[MaTu93] Maunder, Colin M. (Hrsg.) ; Tulloss, Rodham E. (Hrsg.): “IEEE Standard Test Access Port and Boundary-Scan Architecture”; The Institute of Electrical and Electronics Engineers, Inc., Jun 1993. – ISBN 1-55937-350-4

[Max3232] Maxim Integrated Products: ±15kV ESD-Protected, Down to 10nA, 3.0V to 5.5V, Up to 1Mbps, True RS-232 Transceivers MAX3232E. Rev 9. Sep 2005. – http://pdfserv.maxim-ic.com/en/ds/MAX3222E-MAX3246E.pdf

[Max3243] Maxim Integrated Products: ±15kV ESD-Protected, 1µA, 3.0V to 5.5V, 250kbps, RS-232 Transceivers with AutoShutdown MAX3243E. Rev 6. Sep 2005. – http://pdfserv.maxim-ic.com/en/ds/MAX3221E-MAX3243E.pdf

[MAX5230] Maxim: Data Sheet MAX5230 Dual DAC

[MAX5382] Maxim: MAX5380..82: Low-Cost, Low-Power, 8-Bit DACs with 2-Wire Serial Interface in SOT23, Maxim, Rev. 1, Januar 2001

[MCPD06] Majumder, R.; Chaudhuri, B. ; Pal, B.C.; Dufour, C.; "Real time dynamic simulator for power system control applications"; Power Engineering Society General Meeting 2006, 18-22 June 2006.

[Ment04] Menthor Graphics, www.modelsim.com, Modelsim SE - User's Manual, Januar 2004.

[Merm04] Mermoud, Grégory: "A Module-Based Dynamic Partial Reconfiguration tutorial"; Ecole Polytechnique Fédérale de Lausanne, Logic Systems Laboratory. 2004.

[Mesq03] D. Mesquita, F. Moraes, J. C. Palma, L. Möller, N. Calazans, "Remote and Partial Reconfiguration of FPGAs: tools and trends", Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03), 2003.

[MFL+05] Monti, A.; Figueroa, H.; Lentijo, S.; Wu, X.; Dougal, R.; "Interface issues in hardware-in-the-loop simulation"; Electric Ship Technologies Symposium, 2005, 25-27 July 2005.

[Micr03] Microchip Technology, "Application Maestro Software User's Guide", Microchip Technology Inc., 2003

[Micr04] Microchip Technology Inc., MPLAB IDE PICSTART Plus User's Guide, Microchip Technology, 2004, http://www.microchip.com/

[Micr92] National Semiconductor; MICROWIRE Serial Interface AN-452, Application Note 452, January 1992.

[MiTe02] Micron Technology, Inc.: 8Mb SYNCBURST SRAM, Datenblatt, Micron Technology Inc., Revision 2/02

[MMP+03] Mesquita, Daniel ; Moraes, Fernando ; Palma, Jos´e ; Möller, Leandro ; Calazans, Ney: "Remote and Partial Reconfiguration of FPGAs: tools and trends"; Université Montpellier II (LIRMM), Pontificia Universidade Cat´oloca do Rio Grande do Sul (FACIN). 2003. www.opencores.com

[Most04] Mosterman, P.; Sztipanovits, J.; Engell, S.; "Computer-Automated Multiparadigm in Control Systems Technology", IEEE Transactions and Control Systems Technology, March 2004.

[MSM9225a] OKI: MSM9225B User's Manual, CAN Controller, Oki Semiconductor, 1. Juli 2002

[MSM9225b] OKI: MSM9225B User´s Manual: CAN Controller, Datenblatt, OKI Electric Industry Co. Ltd., July 2002

[MüGl05] Müller-Glaser, K. D.; Vorlesungsskript, System-Analyse und -Entwurf, 2005.

[MüGl06] Müller-Glaser, K. D.; Vorlesungsskript (engl.), Hardware Modelling and Simulation, 2006.

[Müll00] Mueller-Glaser, K. D.; Sax, E.; Stork, W.; Wagner, A.; Drescher, J.; Kuehl, M.: “Design and simulation of heterogeneous embedded systems”; Symposium on Integrated Circuits and Systems Design, Sept. 2000

[Nati05] National Semiconductor; “National Semiconductor’s Power Management Solutions for Xilinx Field Programmable Gate Arrays”, 2005.

[NgPe02] Ng, Mark; Peattie, Mike: “Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Mode”; Xilinx, Inc., 1.4 edition, Nov 2002. http://www.xilinx.com/bvdocs/appnotes/xapp502.pdf.

[Nova02] Novak, J.; Fried, A.; Vacek, M.: “CAN Generator and Error Injector”, 9th International Conference on Electronics, Circuits and Systems (IICECS’02), 2002.

[Opat05] Opatz, Felix: “Tipps zur Socket-Programmierung”; Jan 2005. http://www.zotteljedi.de/permalinks/socket-tipps.

[OpCo06] OpenCores.org; Webseite (www.opencores.org), 2006.

[Oppi00] Oppitz, Carsten: „AD-Wandleransteuerung leicht gemacht“; Elektronik Industrie, http://dbindustrie.work.svhfi.de/AI/resources/1675c474d6f.pdf, 2000

[Oppi04] Oppitz, Carsten: „So wird aus Theorie gute AD-Praxis“; Elektronik Industrie, http://dbindustrie.work.svhfi.de/AI/resources/c67c9844e6e.pdf., 2004

[OrCAD02] Frantz, Patrick: “An OrCAD Tutorial”; Rice University, 2002, Revision 1.0

[Pahl00] Pahlke, Stephan ; Herzfeld, Sven: „CAN-Bus: Fehlererzeugung und Analyse“, Diplomarbeit, FH Hannover, Bereich Elektrotechnik, 2000

[PaMM02] Palma, José C. ; de Mello, Aline V. ; Möller, Leandro: “Core Communication Interface for FPGAs”; Pontificia Universidade Caóloca do Rio Grande do Sul (FACIN-PUCRS). 2002.

[Pang04] Pang, F.; Brandon, T.; Cockburn, B.; Hume, M., „A Reconfigurable Digital IC Tester implemented Using the ARM Integrator Rapid Prototyping System”, Canadian Conference on Electrical and Computer Engineering, 2004; Volume 4, 2-5 May 2004 Page(s): 1931 - 1935 Vol.4

[Parn04] Parnell, Karen (Xilinx) „Autos in die Schrottpresse – Alternde Prozessoren“, Design&Elektronik, Seite 17, Heft 2, März 2004

[Phil00] Philips: Data Sheet SJA1000, Stand-Alone CAN controller, Philips Semiconductors, 4. Januar 2000, http://www.semiconductors.philips.com/acrobat/datasheets/SJA1000_3.pdf

[Phil02] Philips Semiconductors. “Data sheet - PCK12429 25-400 MHz differential PECL clock generator”; 1. Edition, June 2002.

[Phil03] Philips Semiconductor;. “16K SPI Bus Serial EEPROM 25AA160A/B, 25LC160A/B”; Jul 2003.

[Phil97] Philips: “Data Sheet SJA1000 Stand-alone CAN Controller”; 1997

[Phyt02] Phytec: “miniMODUL-167 Hardware-Manual”, Phytec Technologie Holding AG, Mai 2002

[PIC18F4431a] Microchip Technology Inc., “PIC18F4431 FLASH MCU Programming Spec”; Microchip Technology, 2003, http://www.microchip.com/

[PIC18F4431b] Microchip Technology Inc., “PIC18F4431 Data Sheet, Microchip Technology”, 2003, http://www.microchip.com/

[Plan05] PlanAhead-Tool, Xilinx Inc., www.xilinx.com/planahead

[PMM+02] J. C. Palma, A. V. de Mello, L. Möller, F. Moraes, N. Calazans, “Core Communication Interface for FPGAs”; Proceedings of the 15 th Symposium on Integrated Circuits and Systems Design (SBCCI'02), 2002.

[Polz06] Polzin, Frank; „Entwicklung eines intelligenten Netzteils mit Powermanagement und –monitoring für die RP- und HiL-Plattform COMPASS“; Studienarbeit, Institut für Technik der Informationsverarbeitung, Universität Karlsrhe, 2006.

[QuKu04] Quade, Jürgen; Kunst, Eva-Katharina: „Linux-Treiber entwickeln: Gerätetreiber für Kernel 2.6 systematisch eingeführt“; dpunkt.verlag, Dec 2004. http://ezs.kr.hsnr.de/TreiberBuch/html/index.html.

[RaSu02] Raghavan, Anup K. ; Sutton, Peter: “JPG - A Partial Bitstream Generation Tool to Support Partial Reconfiguration in Virtex FPGAs”; Motorola Australia Software Centre Adelaide SA Australia, School of Information Technology; Electrical Engineering University of Queensland Brisbane QLD Australia. 2002.

[Ratc88] Ratcliffe, B.: “Early and not–so early prototyping - rationale support”; Proceedings COMPSAC88. IEEE Computer Press, 1988

[RBRC04] Ridao, P.; Batlle, E.; Ribas, D.; Carreras, M.; “Neptune: a hil simulator for multiple UUVs”; OCEANS '04. MTS/IEEE TECHNO-OCEAN '04, 9-12 Nov. 2004.

[RedH02] RedHat Inc. eCos license, May 2002. - http://ecos.sourceware.org/license-overview.html.

[ReVi06] Reorda, M.S.; Violante, M.; “Hardware-in-the-loop-based dependability analysis of automotive systems”; 12th IEEE International On-Line Testing Symposium IOLTS 2006, 10-12 July 2006.

[Ricc02] Ricci, F.; Hoang Le-Huy, „An FPGA-Based Rapid Prototyping Platform For Variable-Speed Drives“, 28th Annual Conference of the Industrial Electronics Society (IECON 02), IEEE 2002; Volume 2, 5-8 Nov. 2002 Page(s): 1156 - 1161 vol.2

[Rodr03] G. Rodriguez, J. Jimenez, J. Proenza, “An architecture for Injection of Complex Fault Scenarios in CAN-Networks”, Emerging Technologies an Factory Automation, 16-19 September 2003 (EFTA’03), 2003.

[Royc70] Royce, W.; “Managing the development of large software systems”; In Proceedings WESCON, August 1970.

[RSJB06] Rao, V.M. ; Sadagopan, R.; Jain, A.K.; Behal, A.; “Digital implementation of a nonlinear controller for a unity power factor rectifier: simulation and experiments”; American Control Conference, 14-16 June 2006.

[Rubi98] Rubini, Alessandro: “Linux Device Drivers”; O’Reilly Verlag, 1998.

[RuiW04] Wang, Rui; Yang, Shiyuan: „The design of a Rapid Prototyping Platform for ARM based embedded system”, Transactions on Consumer Electronics, May 2004

[Rung01] Runge, Martin; „Modellbildung und Realisierung von komplexen Steuerungen unter dem Betriebssystem Embedded–Linux auf Embedded–PC’s“; Institut für Technik der Informationsverarbeitung, Universität Karlsruhe (TH), Diplomarbeit, 2001.

[Rupp05] Rupp, M.; “Rapid prototyping in wireless system design”; The 2nd IEE/EURASIP Conference on DSPenabledRadio, 2005, 19-20 Sept. 2005.

[Rusl99] Rusling, David A.: “The Linux Kernel”; 1999. http://www.linuxhq.com/guides/TLK/tlk.html.

[RuTe03] Rucha, Bernd; Teepe, Gerd; “LIN - Local Interconnect Network”; Elektronik Automotive, Januar 2003.

[Sain05] Milan Saini; “Embedded-Prozessoren testen FPGAs”; Elektronik System On Chip, Februar 2005

[Samt01] Samtec. Datenblatt IPBT-1XX-XX-XX-D-XX, Jul 2001.

[Sand00] Sandstrom, K.; Norstom, C.; Ahlmark, M.; “Frame packing in real-time communication”, Proceedings of Seventh International Conference on Real-Time Computing Systems and Applications, 12-14 Dec. 2000 Page(s):399 - 403

[Sang01] Sangiovanni-Vincentelli, A.; Martin, G.: “Platform-based design and software design methodology for embedded systems”; Design & Test of Computers, IEEE, Volume 18, Issue 6, Nov.-Dec. 2001

[Sang04] Sangiovanni-Vincentelli, A.; Carloni, L.; De Bernardinis, F.; Sgroi, M., “Benefits and challenges for platform-based design”, Design Automation Conference (DAC), 2004. Proceedings. 41st, 2004 Page(s):409 - 414

[Sawy02] Nick Sawyer; “High-Speed Data Serialization and Deserialization (840 Mb/s LVDS)”; Xilinx Inc., 1.3 edition, Jun 2002, http://www.xilinx.com/bvdocs/appnotes/xapp265.pdf.

[Schü97] Schürmann, Bernd: „Rechnerverbindungsstrukturen, Bussysteme und Netzwerke”; Vieweg Verlag Braunschweig/Wiesbaden, 1997, ISBN: 3-528-05562-6

[Schw03] Schwarz, Markus; „Entwicklung einer LEON-Prozessor-basierten Fensterhebersteuerung mit CAN-Interface für den Einsatz im Praktikum Entwurfsautomatisierung“; Diplomarbeit, Institut für Technik der Informationsverarbeitung, Universität Karlsruhe, 2003.

[Schw94] Schwab, Adolf J.; „Elektromagnetische Verträglichkeit“, Springer Verlag, 3. Auflage, 1994. ISBN 3-540-57658-4.

[ScSc02] Schneider, David ; Schwarz, Markus: „Hardware/Software-Synthese ausgewählter Kfz-Innenraum-Funktionen und Analyse von Performanz, Parametrisierbarkeit und dynamischer Hardware-Rekonfiguration“; Team-Studienarbeit, Universität Karlsruhe, ITIV, 2002

[SEAS02] Bortolazzi, Jürgen: “Systems Engineering for Automotive Electronics: Part 3 Target Architectures Networking CAN”; Universität Karlsruhe, ITIV, 2002

[Sens01] Robert Bosch GmbH: „Sensoren für Winkel, Drehrate usw“; Erzeugnisprogramm 2001/2002

[Sera06] Serafini, I.; Carrai, F.; Zolesi, V.; Ramacciotti, T.; „Multi-Sensor Configurable Platform for Automotive Applications“; Design, Automation and Test in Europe, 6-10 March 2006, Munich, Germany

[ShPo05] Short, M.; Pont, M.J.; “Hardware in the loop simulation of embedded automotive control systems”; Intelligent Transportation Systems, 2005, 13-15 Sept. 2005.

[Siko04] Sikora, Axel: „Der PLD-Report 2004“; Elektronik, 2/2004.

[SiRB06] Silverstein, S.B.; Rosenqvist, J.; Bohm, C.; “A simple linux-based platform for rapid prototyping of experimental control systems”; IEEE Transactions on Nuclear Science, Volume: 53, June 2006.

[SJA1000] Philips: Data Sheet SJA1000, Stand-Alone CAN controller, Philips Semiconductors, 4. Januar 2000, http://www.semiconductors.philips.com/acrobat/datasheets/SJA1000_3.pdf

[SJSW05] Sang-Heon Lee; Jae-Gon Lee; Seonpil Kim; Woong Hwangbo; Chong-Min Kyung; "SoC design environment with automated configurable bus generation for rapid prototyping"; 6th International Conference On ASIC ASICON 2005, 24-27 Oct. 2005.

[SMCT05] Schumacher, P.; Mattavelli, M.; Chirila-Rus, A.; Turney, R.; "A software/hardware platform for rapid prototyping of video and multimedia designs"; Fifth International Workshop on System-on-Chip for Real-Time Applications 2005, 20-24 July 2005.

[SmGi05] Smith, C.L.; Gilliom, M.B.; "A flexible rapid-prototyping system for digital-controlled high power converters"; Applied Power Electronics Conference and Exposition, APEC 2005, March 2005.

[SN65HVD230] Texas Instruments: "3.3-V CAN TRANSCEIVER SN65HVD230", Jun 2002.

[SN65LVDS104] Texas Instruments Inc.: "4-Port LVDS and 4-Port TTL-to-LVDS Repeaters SN65LVDS104. http://focus.ti.com/lit/ds/symlink/sn65lvds104.pdf – Jan. 2005.

[SP2526] Sipex Corporation: "+3.0V to +5.5V USB Power Control Switch SP2526"; Aug 2001. – http://www.sipex.com/Files/Datasheets/sp2526.pdf

[Spit01] Spitzer, B.; „Modellbasierter Hardware-in-the-Loop Test von eingebetteten elektronischen Systemen", Institut für Technik der Informationsverarbeitung, Universität Karlsruhe, Dissertation, 2001.

[Spit01a] Spitzer, B.; Kuehl, M.; Mueller-Glaser, K.D.: "A methodology for architecture-oriented rapid prototyping"; Rapid System Prototyping (RSP), June 2001.

[Spiv03] Spivey, G.; Bhattacharyya, S.S.; Nakajima, K.; "Logic Foundry: Rapid Prototyping of FPGA-based DSP Systems", Proceedings of the Design Automation Conference Asia and South Pacific (ASP-DAC 2003), 2003; 21-24 Jan. 2003 Page(s): 374 – 381

[SpyX2E02] Weiß, Karlheinz; Steckstore, Thorsten; Oetker, Carsten; Esser, Karl Georg: „Spyder-Virtex-X2E User's Manual"; Handbuch, Version 1.3, January 2002, Revision 1.7

[Stal91] Stallman, Richard M.: "GNU General Public License"; Jun 1991. http://www.fsf.org/licenses/gpl.html.

[Stei03] Steiner, Jochen: „Konzeption und Entwicklung eines CAN-Bus-Knotens mit USB-Schnittstelle zur Einbindung unter Windows und Linux"; Diplomarbeit, Universität Karlsruhe, ITIV, 2003

[Steu06] Steurer, M.; "PEBB based high-power hardware-in-loop simulation facility for electric power systems"; IEEE Power Engineering Society General Meeting, 2006, 18-22 June 2006.

[StFl97] MathWorks: "StateFLOW for Use with Simulink"; User´s Guide Version 1, MathWorks, May 1997.

[Stop05] Stops, Christian; „Entwicklung eines LIN-Cores zur physikalischen Fehlerinjektion"; Studienarbeit, Institut für Technik der Informationsverarbeitung, Universität Karlsruhe, 2005.

[Stop06] Stops, Christian; „Entwicklung einer Design-Methode für die effiziente Synthese von IP-Core-Modulen zur Konfiguration von Xilinx FPGAs"; Diplom-

arbeit, Institut für Technik der Informationsverarbeitung, Universität Karlsruhe, 2006.

[Stri01] Stripf, Timo: “SpyderDriver library: Preliminary Developer´s Guide”; Handbuch, X2E, Release 0.12, November 2001

[Stro00] Stroustrup, Bjarne: „Die C++ Programmiersprache“; Addison-Wesley Verlag, München, 4 edition, 2000.

[Stuh04a] Stuhlmüller, Peter: „Der ADC-Report 2004 - Teil 1“; Elektronik, 2004, http://www.elektroniknet.de/topics/bauelemente/fachthemen/2004/0012/.

[Stuh04b] Stuhlmüller, Peter: „Der ADC-Report 2004 - Teil 2“; Elektronik, 2004, http://www.elektroniknet.de/topics/bauelemente/fachthemen/2004/0013/.

[SuMG01] Sundararajan, Prasanna ; McMillan, Scott ; Guccione, Steven A.: Testing „FPGA Devices Using Jbits“; Xilinx Inc., 2001.

[Sung05] Sung Chul Oh; “Evaluation of motor characteristics for hybrid electric vehicles using the hardware-in-the-loop concept”; IEEE Transactions on Vehicular Technology, Volume: 54, May 2005.

[Teic97] Teich, J.: „Digitale Hardware/Software-Systeme-Synthese und Optimierung“, Springer-Verlag, 1997.

[Texa04] Texas Instruments. “Tips for successful power-up of today’s high-performance FPGAs”; 3Q 2004.

[Texa05] Texas Instruments; “Texas Instruments Power Management Reference Guide for Xilinx FPGAs and CPLDs”; 4Q 2005.

[Thor04] Thorvinger, Jens: “Dynamic Partial Reconfiguration of an FPGA for Computational Hardware Support”; Lund Institute of Technology, Electroscience, Diplomarbeit, 2004

[TiSc02] Tietze, U.; Schenk, Ch.: „Halbleiter-Schaltungstechnik“, 12. Auflage, Springer Verlag, Berlin/Heidelberg, 2002

[TiSc93] Tietze, U.; Schenk, Ch.: „Halbleiterschaltungstechnik“; 10. Auflage, Springer-Verlag, Berlin/Heidelberg, 1993

[Turn99] Turner, R.: “System-Level Verification - A Comparison of Approaches”; Rapid System Prototyping Conference RSP, Tampa, Florida, USA, 1999.

[UHGB04] Ullmann, Michael ; Hübner, Michael ; Grimm, Björn ; Becker, Jürgen: „An FPGA Run-Time System for Dynamical On-Demand Reconfiguration“; Universität Karlsruhe (TH), Institut für Technik der Informationsverarbeitung (ITIV). 2004.

[UHGB04] M. Ullmann, M. Hübner, B. Grimm, J. Becker, “An FPGA Run-Time System for Dynamical On-Demand Reconfiguration”, Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), 2004.

[VDE06] Die Informationstechnische Gesellschaft im VDE (ITG); „Kfz-Anforderungen an Elektronik-Bauelemente“; ITG-Positionspapier, VDE 2006.

[Vect03] Vector Informatik, GmbH: „CANalyzer“, Handbuch, Version 4.1, 2003.

[Wald99] Waldner, Walter: „Ansteuerung eines LCD-Moduls mit dem kitCON-167“; Version 1.2, September 1999

[Wang06] Wang, Bo; „Implementierung und Test eines NoC auf der RP Plattform COMPASS“; Studienarbeit, Institut für Technik der Informationsverarbeitung, Universität Karlsrhe, 2006.

[Wart00] Wartburg, Iwan von; „Wissensbasiertes Management technologischer Innovation“; Witschaftswissenschaftliche Fakultät, Universität Zürich, Dissertation, 2000

[WeRG95] Weisser, M.; Rüger, B.; Geisweis, H.: „Richtig in die Gänge kommen – Durch Rapid Prototyping schneller mit der Funktionsidee ins Auto“; Elektronik, 24, 1995.

[WLD+05] Wu, X.; Lentijo, S.; Deshmuk, A.; Monti, A.; Ponci, F.; “Design and implementation of a power-hardware-in-the-loop interface: a nonlinear load case study”; Twentieth Annual IEEE Applied Power Electronics Conference and Exposition APEC 2005, 6-10 March 2005.

[Wolf97] Lawrenz, Wolfhard „CAN System Engineering – From Theory to Practical Applications“, Springer Verlag, Berlin/Heidelberg, 1997

[WuFM04] Wu, X.; Figueroa, H.; Monti, A., “Testing of Digital Controllers Using Real-Time Hardware in the Loop Simulation”, 35th Annual Power Electronics Specialists Conference (PESC 04), 2004; Volume 5, 20-25 June 2004 Page(s): 3622 – 3627

[WuLM04] Wu, X.; Lentijo, S.; Monti, A.; “A novel interface for power-hardware-in-the-loop simulation”; IEEE Workshop on Computers in Power Electronics, 2004, 15-18 Aug. 2004.

[WuMo05] Wu, X.; Monti, A.; “Methods for partitioning the system and performance evaluation in power-hardware-in-the-loop simulations - Part II”; 32nd Annual Conference of Industrial Electronics Society IECON 2005, 6-10 Nov. 2005.

[Xdriv04] Xilinx, Inc., “Erklärung aller Xilinx-Header-Dateien Xilinx Device Drivers Documentation”; Xilinx, 28. Januar 2004

[XEDK04] Xilinx, Inc., Thema: “Basisentwurf Hardwaresystem, Schreiben Applikationen, Kompilierung Simulationsbibliotheken”, Debugging in EDK Platform Studio User Guide UG113 (v1.0), Xilinx, January 30, 2004

[XEDK04a] Xilinx, Inc., Thema: “Standard C Lib libc.a, Xilinx C Lib libxil.a”; EDK OS and Libraries Reference Guide UG114 (v1.0)”, January 30, 2004

[XEDK04b] Xilinx, Inc., Thema: „Schritt für Schritt Anleitung eines Designs“; EDK MicroBlaze Tutorial Version 2.1, Xilinx 03/2004

[Xil01] Xilinx, Inc., „Virtex–II Platform FPGA Handbook“, 2001, http://www.xilinx.com

[Xil01a] Xilinx, Inc: “Power-On Requirements for the Spartan-II and Spartan-IIE Families”; v1.0. Nov 2001. – http://www.xilinx.com/bvdocs/appnotes/xapp450.pdf

[Xil01b] Xilinx, Inc: “Powering Xilinx Spartan-II FPGAs”; v1.1. Jul 2001. – http://www.xilinx.com/bvdocs/appnotes/xapp188.pdf

[Xil02] Xilinx, Inc.; “Configuration and Readback of Virtex FPGAs Using (JTAG) Boundary Scan”; 2002, http://www.xilinx.com

[Xil02a] Xilinx, Inc: “Board Routability Guidelines with Xilinx Fine-Pitch BGA Packages”; v1.2. Nov 2002. – http://www.xilinx.com/bvdocs/appnotes/xapp157.pdf

[Xil02b] Xilinx, Inc: “Configuration and Readback of the Spartan-II and Spartan-IIE Families”; v1.0. Mar 2002. – http://www.xilinx.com/bvdocs/appnotes/xapp176.pdf

[Xil02c] Xilinx, Inc: “Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Mode”; v1.4. Nov 2002. – http://www.xilinx.com/bvdocs/appnotes/xapp501.pdf

[Xil02d] Xilinx, Inc.; “Designing Custom OPB Slave Peripherals for Micro-Blaze”, 06/2002, www.xilinx.com

[Xil03] Xilinx, Inc., “XST User Guide”,06/2003, www.xilinx.com,

[Xil03a] Xilinx, Inc.; “Virtex-II Digital Clock Manager”; 1.2 Edition, June 2003, http://www.xilinx.com/products/virtex/techtopic/vtt010.pdf.

[Xil03b] Xilinx, Inc: “Spartan-II 2.5V FPGA Family: DC and Switching Characteristics”; v2.7. Sep 2003. – http://direct.xilinx.com/bvdocs/publications/ds001_3.pdf

[Xil03c] Xilinx, Inc: “Spartan-II 2.5V FPGA Family: Functional Description”; v2.2. Sep 2003. – http://direct.xilinx.com/bvdocs/publications/ds001_2.pdf

[Xil03d] Xilinx, Inc: “Spartan-II 2.5V FPGA Family: Pinout Tables”; v2.5. Sep 2003. – http://direct.xilinx.com/bvdocs/publications/ds001_4.pdf

[Xil04] Xilinx, Inc.: “Spartan-II 2.5V FPGA Family: Complete Data Sheet”; Aug 2004.

[Xil04a] Xilinx, Inc.; “Implementing a LIN Controller on a CoolRunner-II CPLD”; 07 2004.

[Xil04b] Xilinx, Inc.; “EDK OS and Libraries Reference Guide”, EDK 6.2i Edition, Jan 2004

[Xil04c] Xilinx, Inc.; “User Core Templates Reference Guide”; 1.5 Edition, Jan 2004, http://www.xilinx.com/ise/embedded/edk6_2docs/user_core_templates_ref_guide.pdf.

[Xil04d] Xilinx, Inc: “Spartan-II 2.5V FPGA Family: Introduction and Ordering Information”; v2.5. Aug 2004. – http://direct.xilinx.com/bvdocs/publications/ds001_1.pdf

[Xil04e] Xilinx, Inc.; “ISE 6 Libraries Guide”, ISE 6.1i Edition, 2004.

[Xil04f] Xilinx, Inc: “Xilinx In-System Programming Using an Embedded Microcontroller”; v3.1. Jun 2004. – http://www.xilinx.com/bvdocs/appnotes/xapp058.pdf

[Xil04g] Xilinx, Inc., “MicroBlaze Processor Reference Guide”; 06/2004, www.xilinx.com

[Xil04h] Xilinx, Inc.; “Platform Studio User Guide”, 03/2004

[Xil04i] Xilinx, Inc.; “User Core Templates Reference Guide”; 1.5 Edition, Jan 2004.

[Xil04j] Xilinx, Inc.; “Virtex-II Platform FPGAs: Complete Data Sheet”; 3.3 Edition, 2004.

[Xil04k] Xilinx, Inc.; “Xilinx Device Drivers Documentation” ; 01/2004.

[Xil04l] Xilinx, Inc.; “Xilinx System Generator v6.2 User Guide”; 2004.

[Xil05] Xilinx, Inc.; “Embedded Systems Tool Guide”; 06/2005.

[Xil05a] Xilinx, Inc.; “Configuration and Readback of Spartan-II and Spartan-IIE FPGAs Using Boundary Scan”; v2.2. Jun 2005. – http://www.xilinx.com/bvdocs/appnotes/xapp188.pdf

[Xil05b] Xilinx, Inc.; “Platform Flash In-System Programmable Configuration PROMS XCF02S”; v2.7. Jul 2005. – http://www.xilinx.com/bvdocs/publications/ds123.pdf

[Xil05c] Xilinx, Inc.; “Spartan-3 FPGA Family: Complete Data Sheet”; v1.5. Aug 2005. – http://direct.xilinx.com/bvdocs/publications/ds099.pdf

[Xil05d] Xilinx, Inc.; „Virtex-II Platform FPGA User Guide“; v2.0. Mar 2005. – http://www.xilinx.com/bvdocs/userguides/ug002.pdf

[Xil05e] Xilinx, Inc: “Virtex-II Platform FPGAs: Complete Data Sheet”; v3.4. Mar 2005.– http://direct.xilinx.com/bvdocs/publications/ds031.pdf

[Xil05f] Xilinx, Inc.; “Virtex-II Pro and Virtex-II Pro X FPGA User Guide”; v4.0. Mar 2005. – http://www.xilinx.com/bvdocs/userguides/ug012.pdf

[Xil05g] Xilinx, Inc.; “Platform Specification Format Reference Manual”; 02/2005.

[XISE05] Xilinx, Inc. ; “ISE Guide für ISE 6.3”, Onlinehilfe, Xilinx, Inc. 2005

[XMB03] Xilinx, Inc., Thema: “Architektur MB, Register, Datentypen, Assemblerbefehle MicroBlaze Processor Reference Guide”; Xilinx EDK (v6.2) December 9, 2003.

[XOPB03] Xilinx, Inc., Thema: “Interrupt-Timer Details für den On-Chip Peripheral Bus OPB Timer/Counter Version 1.11.3”, 25. November 2003.

[XPCT05] The Mathworks; “xPC TargetBox”; www.mathworks.com, 2005.

[XPS04] Xilinx, Inc., Thema: “XPS, Base System Builder, Simulation Model Generator, GNU Compiler, XMD, MHS, MPD, MSS Embedded System Tools Guide UG111”; V1.4, January 30, 2004

[YeJi05] Ye, Jian; „Entwicklung einer adaptiven Timing-Analyse-Einheit zum Test von PWM-Steuergeräteausgängen in DaimlerChrysler Fahrzeugen und Integration in ein bestehendes FPGA-System“; Diplomarbeit, Institut für Technik der Informationsverarbeitung, Universität Karlsruhe, 2005.

[YinT02] Yin-Tsung Hwang; Cheng-Ji Chang; Bor-Liang Chen, „A Rapid Prototyping Embedded System Platform and its HW/SW Communication Interface Generation and Verification“, Asia-Pacific Conference on Circuits and Systems (APCCAS '02), 2002; Volume 1, 28-31 Oct. 2002 Page(s): 481 - 484

[ZDL+06] Zhenhua Jiang; Dougal, R.A.; Leonard, R.; Figueroa, H.; Monti, A.; “Hardware-in-the-loop testing of digital power controllers”; Twenty-First Annual IEEE Applied Power Electronics Conference and Exposition. APEC '06, 19-23 March 2006.

[ZhJi06] Zhongjian Li; Jinwen An; “Modeling and Simulation of Large Aircraft's Hybrid Control System”; The Sixth World Congress on Intelligent Control and Automation WCICA 2006, 21-23 June 2006.

[Zöll02] Zöller, Martin: „Entwicklung einer FPGA-basierten Ein-/Ausgabeschnittstelle mit PCI–Interface für Rapid-Prototyping-Anwendungen“; Institut für Technik der Informationsverarbeitung, Universität Karlsruhe (TH), Diplomarbeit, 2002.

9.7 Betreute Diplom-, Studien- und Seminararbeiten

9.7.1 Diplomarbeiten

ID-994 **Brugger, Daniel**: Entwicklung einer Backplane mit FPGA- und Ethernet-Interface für ein minimales Rapid-Prototyping-System mit einem Prozessor INTEL StrongArm unter Embedded-LINUX; ITIV, Uni Karlsruhe, 2002

ID-1006 **Steiner, Jochen**: Entwicklung eines CAN-Bus-Knotens mit USB-Schnittstelle und Einbindung unter Windows und Linux; ITIV, Uni Karlsruhe, 2003

ID-1041 **Tappe, Hans-Jürgen**: Charakterisierung eines 3-Phasen-Piezo-Motors und Entwicklung einer miniaturisierten Ansteuerschaltung für das MicroCar Karlsruhe; ITIV, Uni Karlsruhe, 2003

ID-1042 **Schwarz, Markus**: Entwicklung einer LEON-Prozessor-basierten Fensterhebersteuerung mit CAN-Interface für den Einsatz im Praktikum Entwurfsautomatisierung; ITIV, Uni Karlsruhe, 2003

ID-1067 **Wang, Ke**: Implementierung eines CAN-IP-Cores und Entwicklung eines LIN-Cores in VHDL zur Verwendung mit einem Xilinx MicroBlaze-Controller; ITIV, Uni Karlsruhe, 2004

ID-1110 **Hillenbrand, Martin**: Entwicklung eines „Local Controllers“ (LC) mit PROFIBUS- und CAN-Interface zur Positionierung von Antrieben in der Bühnentechnik; ITIV, Uni Karlsruhe, 2005

ID-1125 **Ye, Jian**: Entwicklung einer adaptiven Timing-Analyse-Einheit zum Test von PWM-Steuergeräteausgängen in DaimlerChrysler Fahrzeugen und Integration in ein bestehendes FPGA-System; ITIV, Uni Karlsruhe, 2005

ID-1146 **Heinz, Matthias**: Entwicklung eines flexiblen µController-Systems zur Ausführung von Rapid-Prototyping- und Hardware-in-the-Loop-Szenarien; ITIV, Uni Karlsruhe, 2006

ID-1158 **Bahlinger, Martin**: Entwicklung einer Steuerung für eine Rapid-Prototyping- und Hardware-in-the-Loop-Plattform unter Linux und Anbindung an einen Windows-Host-PC über Ethernet; ITIV, Uni Karlsruhe, 2006

ID-1161 **Stops, Christian**: Entwicklung einer Design-Methode für die effiziente Synthese von IP-Core-Modulen zur Konfiguration von Xilinx FPGAs; ITIV, Uni Karlsruhe, 2006

ID-1181 **Matsumoto, Akira**: Enhancing an Electric Power Steering (EPS) application by an X-by-Wire functionality using a FlexRay network; ITIV, Uni Karlsruhe, 2007

9.7.2 Studienarbeiten

IL-564 **Müller, Christian**: Konzeption und Implementierung einer Funk-Verbindung im UHF-Band zur Fernsteuerung des MicroCar Karlsruhe (MICK); ITIV, Uni Karlsruhe, 2002

IL-565 **Thoma, Florian**: Entwicklung einer PCI-Karte mit Standard-FPGA zur Evaluierung von Open-Source PCI-Controller-Cores; ITIV, Uni Karlsruhe, 2002

IL-576 **Tappe, Hans-Jürgen**: Implementierung einer Software-Schnittstelle für den LAN-Controller in einem Rapid-Prototyping-System mit Prozessor Intel StrongArm unter Embedded-LINUX; ITIV, Uni Karlsruhe, 2003

IL-587 **Klausmann, Achim**: Entwicklung einer Kaffeeautomatensteuerung unter VHDL für den Einsatz auf einem Standard-FPGA im Praktikum Entwurfsautomatisierung; ITIV, Uni Karlsruhe, 2002

IL-590 **Sander, Björn**: Konzeption eines Soft-Prozessors mit einstufiger Pipeline und Realisierung als VHDL-Modell für den Einsatz im Praktikum Entwurfsautomatisierung; ITIV, Uni Karlsruhe, 2003

IL-600 **Blümel, Marco**: Entwicklung eines Fensterheber-Fernsteuermoduls basierend auf dem CAN-Protokoll zum Einsatz im Praktikum Entwurfsautomatisierung (PEA) ; ITIV, Uni Karlsruhe, 2004

IL-618 **Jung, Markus**: Modellierung von Kfz-Aktoren und -Sensoren in MATLAB Simu-Link zur Simulation auf einem Xilinx-MicroBlaze-Prozessor in Hardware-in-the-Loop-Tests; ITIV, Uni Karlsruhe, 2004

IL-626 **Butting, Björn**: Konzeption und Implementierung einer Funk-Verbindung im 2,4 GHz-Band zur Fernsteuerung des MicroCar Karlsruhe (MICK) ; ITIV, Uni Karlsruhe, 2004

IL-632 **Teiwes-Morin, Christophe**: Entwicklung einer schnellen Kommunikationsschnittstelle auf FPGAs unter Verwendung von LVDS-Treibern; ITIV, Uni Karlsruhe, 2004

IL-644 **Eyhorn, Daniel**: Entwicklung eines LIN-Bus-Knotens zur Anbindung an die USB2.0-Schnittstelle zur Verwendung unter Windows; ITIV, Uni Karlsruhe, 2005

IL-648 **Heinz, Matthias**: Entwicklung eines schnellen D/A-Wandler-Boards mit LVDS-Schnittstelle und Integration in ein bestehendes FPGA-System; ITIV, Uni Karlsruhe, 2005

IL-649 **Bahlinger, Martin**: Entwicklung eines schnellen A/D-Wandler-Boards mit LVDS-Schnittstelle und Integration in ein bestehendes FPGA-System; ITIV, Uni Karlsruhe, 2005

IL-662 **Fuchs, Sebastian**: Entwicklung eines Bedien-Panels zur Positionierung von Antrieben in der Bühnentechnik; ITIV, Uni Karlsruhe, 2005

IL-663 **Stops, Christian**: Entwicklung eines LIN-Cores zur physikalischen Fehlerinjektion; ITIV, Uni Karlsruhe, 2005

IL-671 **Klimm, Alexander**: Automatisierung des Design-Flows beim Entwurf von Systemmodellen unter Verwendung des SystemGenerators von Xilinx; ITIV, Uni Karlsruhe, 2005

IL-701 **Polzin, Frank**: Entwicklung eines intelligenten Netzteils mit Powermanagement und –monitoring für die RP- und HiL-Plattform COMPASS; ITIV, Uni Karlsruhe, 2006

IL-720 **Kurzweil, Ronald**: Interpolation und Glättung von Bahnkurven mit MatLab/Simulink und C-Code-Export zur Verarbeitung unter einem Echtzeitbetriebssystem; ITIV, Uni Karlsruhe, 2007

IL-724 **Wang, Bo**: Implementierung und Test eines NoC auf der RP Plattform COMPASS; ITIV, Uni Karlsruhe, 2006

IL-744 **Adler, Nico**: Entwicklung eines Temperaturreglers mit PT100-Sensoren für den Stand-Alone-Betrieb und Integration in eine Steuerung mit CAN-Bus; ITIV, Uni Karlsruhe, 2007

IL-748 **Zechmeister, Steffen**: Ansteuerung eines Frequenzumrichters über das CANopen-Protokoll zur Positionierung von Antrieben in der Bühnentechnik; ITIV, Uni Karlsruhe, 2007

IL-752 **Vollmer, Jörg**: Programmierung eines Treibers unter Windows zur Ansteuerung einer PCI-FPGA-Karte; ITIV, Uni Karlsruhe, 2007

IL-771 **Höss, Verena**: Entwicklung eines Steuerungs- und Überwachungssystems von Winden in der Bühnentechnik auf Basis einer PCI-FPGA-Karte; ITIV, Uni Karlsruhe, 2007

IL-772 **Gerber, Timo**: Entwicklung einer Kommunikationsschnittstelle zum Austausch von Steuerungsdaten zwischen zwei Echtzeitrechnern in der Bühnentechnik; ITIV, Uni Karlsruhe, 2007

IL-776 **Minne, Thomas**: Entwicklung einer MySQL-Datenbankanbindung zur Kaffeeabrechnung über Ethernet und einer Web-Administrationsoberfläche; ITIV, Uni Karlsruhe, 2007

IL-777 **Müller, Matthias**: Entwicklung einer Kaffeeautomatensteuerung mit RFID-Leser zur automatischen, Datenbank-gestützten Kaffeeabrechnung über Ethernet; ITIV, Uni Karlsruhe, 2007

9.7.3 Seminararbeiten

Mamasaliyev, Ziyedulla:
Feldbusse im Automotive-Umfeld (CAN, LIN, MOST, FlexRay, TTP/C); ITIV, Uni Karlsruhe, 2004

Traub, Matthias:
Kommunikationssystem FlexRay; ITIV, Uni Karlsruhe, 2005

9.8 Eigene Veröffentlichungen

Bieser, C.; Becker, J. E.; Thomas, A.; Müller-Glaser, K.-D.; Becker, J.: *Hardware/Software Co-Training by FPGA/ASIC Synthesis and programming of a RISC Microprocessor-Core*; In: Kongressbericht "International Conference on Microelectronic Systems Education (MSE)" , Anaheim/Los Angeles, USA, June 2003

Bieser, Carsten: *MultiCAN@USB2.0 - Viele CAN Schnittstellen am USB-Port;* In: "Design&Elektronik", Heft 11, November 2004, Seite 28ff

Bieser, C.; Müller-Glaser, K.-D.; *COMPASS — A Novel Concept of a Reconfigurable Platform for Automotive System Development and Test;* In: "The 16th IEEE International Workshop on Rapid System Prototyping", 2005 (RSP 2005). 08-10 June 2005, Montreal, Canada

Bieser, C.; Müller-Glaser, K.D.; Becker, J.; *Hardware/Software Co-Training Lab: From VHDL Bit-Level Coding up to CASE-Tool Based System Modeling;* In: "2005 IEEE International Conference on Microelectronic Systems Education", 2005. (MSE '05); 12-13 June 2005, Anaheim/Los Angeles, USA

Bieser, C.; Fleischer, J.; Müller-Glaser, K.D.; Volkmann, T.; *Challenges in Design and Production of Micro-Mechatronical Systems Micro System Technologies*; In: "International Conference & Exhibition on Micro-, Electro-Mechanical, Opto & Nano Systems", München, 05. - 06.10.2005

Bieser, C.; Müller-Glaser, K.-D.; *Rapid Prototyping Design Acceleration Using a Novel Merging Methodology for Partial Configuration Streams of Xilinx Virtex-II FPGAs* In: "The 17th IEEE International Workshop on Rapid System Prototyping", 2006 (RSP 2006). 14-16 June 2006, Chania, Crete

Becker, J. E.; Bieser, C.; Becker, J.; Müller-Glaser, K.-D.; *Evaluation of a Packet Switching Algorithm for Network on Chip Topologies using a Xilinx Virtex-II FPGA based Rapid Prototyping System*; In: "International Symposium on Industrial Electronics", 2006. (ISIE 2006); 09. - 13. July 2006, Montreal, Canada

Bieser, C.; Müller-Glaser, K.-D.; *Xilinx Virtex-II FPGA Design Acceleration Using a Novel Merging Methodology for Partial Configuration Bitstreams*; In: "The 49th IEEE Midwest Symposium on Circuits and Systems", 2006 (MWSCAS '06); 06. - 09. August 2006, San Juan, Puerto Rico

Bieser, C.; Bahlinger, M.; Heinz, M.; Stops, C.; Müller-Glaser, K.-D.; *A Novel Partial Bitstream Merging Methodology Accelerating Xilinx Virtex-ii FPGA Based RP System Setup;* In: "16th International Conference on Field Programmable Logic and Applications", 2006. (FPL '06), 28. - 30. August 2006, Madrid, Spain

Bieser, Carsten; *A Novel FPGA Design Acceleration Methodology Supported by an Unique RP Platform for Fast and Easy System Development*; In: "16th International Conference on Field Programmable Logic and Applications", 2006. (FPL '06), 28. - 30. August 2006, Madrid, Spain

Lebenslauf

Persönliche Daten

Name	:	Carsten Bieser
Geburtsdatum	:	27.11.1971
Geburtsort	:	Alzey
Staatsangehörigkeit	:	deutsch
Familienstand	:	verheiratet

Schul- und Berufsausbildung

August 1978 – Juli 1982	:	Grund- und Hauptschule, Flomborn
August 1982 – Juli 1989	:	Gymnasium an der Frankenstraße, Alzey
16.08.1989 – 30.06.1992	:	Ausbildung zum Kommunikationselektroniker, SIEMENS AG, Mannheim
August 1992 – Juni 1995	:	Staatliches Aufbaugymnasium, Alzey
30.06.1995	:	Allgemeine Hochschulreife
1993	:	Teilnahme bei **jugend**forscht, Bereich Technik
01.09.1995 – 30.09.1996	:	Zivildienst im Michaelshof, Kirchheimbolanden

Studium und wissenschaftliche Tätigkeit

01.10.1996 – 04.08.2001	:	Studium der Elektrotechnik an der Universität Karlsruhe (TH)
04.08.2001	:	Abschluss: Diplom-Ingenieur (Dipl.-Ing.)
15.09.2001 – 31.08.2007	:	Beschäftigung als wissenschaftlicher Angestellter am Institut für Technik der Informationsverarbeitung (ITIV) der Universität Karlsruhe (TH)
05. Juli 2007	:	Promotion zum Dr.-Ing.

Weitere Beschäftigungsverhältnisse

11.12.1997 – 31.01.2001	:	Tätigkeit als wissenschaftliche Hilfskraft bei der s.a.x Software GmbH in Karlsruhe-Durlach
01.01.1999 – 31.01.1999 und 01.11.2000 – 30.04.2001	:	Tätigkeit als geprüfte wiss. Hilfskraft am Institut für Technik der Informationsverarbeitung der Universität Karlsruhe (TH)
01.02.2001 – 31.12.2003	:	Geschäftsführender Gesellschafter der Pump-Tec Consult GmbH, Eppelsheim