Markus Deicke

Virtuelle Absicherung von Steuergeräte-Software mit hardwareabhängigen Komponenten

Wissenschaftliche Schriftenreihe
Eingebettete, selbstorganisierende Systeme

Band 16

Prof. Dr. Wolfram Hardt (Herausgeber)

Markus Deicke

Virtuelle Absicherung von Steuergeräte-Software mit hardwareabhängigen Komponenten

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Universitätsverlag Chemnitz
2018

Impressum

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Angaben sind im Internet über http://dnb.d-nb.de abrufbar.

Titelgrafik: Markus Deicke
Satz/Layout: Markus Deicke

Technische Universität Chemnitz/Universitätsbibliothek
Universitätsverlag Chemnitz
09107 Chemnitz
http://www.tu-chemnitz.de/ub/univerlag

readbox unipress
in der readbox publishing GmbH
Am Hawerkamp 31
48155 Münster
http://unipress.readbox.net

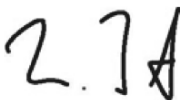
ISSN 2196-3932 print - ISSN 2196-4815 online

ISBN 978-3-96100-039-5

http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-230123

Vorwort zur Wissenschaftlichen Schriftenreihe „Eingebettete, Selbstorganisierende Systeme“

Dieser Band der wissenschaftlichen Schriftenreihe *Eingebettete, Selbstorganisierende Systeme* widmet sich der Virtualisierung der Absicherung von Steuergeräte-Software im Automobilbereich. Der stetig wachsende Funktionsumfang und die Vernetzung von Software im Fahrzeug steigern die Komplexität und erfordert daher effiziete Methoden für die Verifikation und Validierung. Dabei bietet die Virtualisierung erhebliches Potential, indem Software bereits früher im Entwicklungsprozess unabhängig von der Ziel-Hardware auf PC-Systemen abgesichert werden kann. Herr Deicke beschäftigt sich in diesem Zusammenhang mit hardwareabhängigen Anteilen der Software, sogenannte Complex-Device-Drivers, welche über direkte Hardwarezugriffe verfügen und daher nicht ohne zusätzliche Maßnahmen auf ein PC-System portierbar sind.
Dabei sind die Schwerpunkte der Arbeit die Entwicklung von Konzepten zur Berücksichtigung der hardwarespezifischen Software-Anteile inklusive der Evaluierung anhand von Projekten aus der Serien-Entwicklung. Herr Deicke beschreibt in einem seiner Konzepte die Simulation von Complex-Device-Drivers um die Verifikation der hardwareunahängigen Funktions-Software zu ermöglichen. Ein weiteres Konzept ermöglicht die Anbindung der hardwarespezifischen Software an die vorhandene Hardware des Virtualisierungssystems und erlaubt darüber die Ansteuerung realer Peripherie. Beide Lösungen werden anhand von Projekten aus der Serien-Entwicklung, wie z. B. einer Außenlichtsteuerung, einem Head-Up-Display-Treiber oder einem Elektromotortreiber evaluiert und damit erfolgreich bestätigt.
Ich freue mich, Herrn Deicke für die Veröffentlichung seiner Arbeit in dieser wissenschaftlichen Schriftenreihe gewonnen zu haben und wünsche allen Lesern viel Freude und Nutzen bei der Lektüre.



Prof. Dr. Wolfram Hardt
Professur Technische Informatik
Dekan Fakultät für Informatik

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Virtuelle Absicherung von Steuergeräte-Software mit hardwareabhängigen Komponenten

Dissertation
zur Erlangung des akademischen Grades

Dr.-Ing.

Dipl.-Ing. Markus Deicke
geboren am 17.09.1984 in München

Fakultät für Informatik
an der Technischen Universität Chemnitz

Gutachter:	Prof. Dr. rer. nat. Wolfram Hardt
	Prof. Dr.-Ing. Hans-Christian Reuss
Tag der Verteidigung:	02.10.2017

Deicke, Markus
Virtuelle Absicherung von Steuergeräte-Software mit hardwareabhängigen Komponenten
Dissertation, Fakultät für Informatik
Technische Universität Chemnitz, Dezember 2017

Danksagung

Diese Dissertation entstand während meiner Anstellung als Doktorand bei der BMW Group in Kooperation mit der TU Chemnitz. Allen, die mich während dieser Zeit unterstützt haben, möchte ich hiermit danken.
Mein erster Dank gilt meinem Doktorvater Prof. Dr. Wolfram Hardt, für die hervorragende wissenschaftliche Betreuung während der Erarbeitung dieser Dissertation und die freundliche Aufnahme an der Professur für technische Informatik. Durch seine Impulse hat er stets für die Einhaltung des richtigen Kurses der Arbeit gesorgt und mir dennoch genügend Freiraum gelassen.
Bei Prof. Dr. Hans-Christian Reuss von der Universität Stuttgart möchte ich mich herzlich für die Übernahme des Zweitgutachtens bedanken sowie für den hilfreichen Austausch bei unseren Gesprächen.
Ein besonderer Dank gilt Dr. Marcus Martinus der mir nach meiner Diplomarbeit bei der BMW Group die Möglichkeit zur Fortführung des Themas im Rahmen der Doktorarbeit gab. Während der gesamten Zeit stand er mir stets als fachlicher Betreuer mit Rat und Tat zur Seite und hatte immer ein offenes Ohr für mich.
Dr. Maximilian Fuchs möchte ich ebenso herzlich danken. Er stand mir zunächst als Abteilungsleiter, dann als Mentor zur Seite und hat durch seine Ratschläge bis zum Ende der Schreibphase immer dafür gesorgt, dass die wesentlichen Aspekte der Arbeit im Fokus blieben.
Bei den Kollegen des VAP-Teams, Stefan Jurchen, Tugba Mascolino und Fernando Espinosa Arroniz sowie den anderen internen und externen Kollegen aus meinem Umfeld bei der BMW Group und im Doktorandenkreis, möchte ich mich für die freundschaftliche und hilfsbereite Zusammenarbeit sowie die willkommene Ablenkung beim Kaffeetrinken bedanken.
Michael Scheffer von Infineon sowie Michael Folie und seinen Kollegen von IPG danke ich für die Beistellung der Software, der Hardware und die Hilfe bei der Inbetriebnahme des Elektromotortreibers.
Allen wissenschaftlichen Mitarbeitern der Professur für technische Informatik möchte ich für die gute Aufnahme in Chemnitz sowie die Anregungen und Fragen im Austausch danken.
Weiterhin danke ich meinen Korrekturlesern, insbesondere Tilman Krummeck, die dazu beigetragen haben meiner Arbeit den orthographischen Feinschliff zu verpassen.
Meinen Eltern, meiner Familie und meinen Freunden möchte ich dafür danken, dass sie mich stets motiviert und unterstützt haben.
Insbesondere danke ich meiner Frau Pauli, die mir über die gesamte Doktorandenzeit viel Liebe und Verständnis entgegengebracht hat und in allen Lebenslagen für mich da war.

Kurzfassung

Der stetig steigende Funktionsumfang im Automobil und die zunehmende Vernetzung von Steuergeräten erfordern neue Methoden zur Beherrschung der Komplexität in der Validierung und Verifikation. Die virtuelle Absicherung ermöglicht die Integration der Software in einem PC-System, unabhängig von der Ziel-Hardware, zur frühzeitigen Gewährleistung der Softwarequalität im Entwicklungsprozess. Ebenso kann die Wiederverwendbarkeit vorhandener Komponenten in zukünftigen Mikrocontrollern sichergestellt werden. Die Grundlage dafür liefert der AUTOSAR-Standard durch einheitliche Schnittstellenbeschreibungen, welche die Abstraktion von Hardware und Software ermöglichen.
Allerdings enthält der Standard hardwareabhängige Software-Komponenten, die als Complex-Device-Drivers (CDDs) bezeichnet werden. Aufgrund ihrer Hardwareabhängigkeit sind CDDs nicht direkt in eine virtuelle Absicherungsplattform integrierbar, da die spezifischen Hardware-Module nicht verfügbar sind. Die Treiber sind dennoch Teil der Steuergeräte-Software und somit bei einem ganzheitlichen Absicherungsansatz mit zu betrachten.
Diese Dissertation beschreibt sieben unterschiedliche Konzepte zur Berücksichtigung von CDDs in der virtuellen Absicherung. Aus der Evaluierung der Praxistauglichkeit aller Ansätze wird eine Auswahlmethodik für die optimale Lösung bei sämtlichen Anwendungsfällen von CDDs in der Steuergeräte-Software entwickelt. Daraus abgeleitet, eignen sich zwei der Konzepte für die häufigsten Anwendungsfälle, die im Weiteren detailliert beschrieben und realisiert werden.
Das erste Konzept erlaubt die vollständige Simulation eines CDD. Dies ist notwendig, um die Integration der Funktions-Software selbst ohne den Treiber zu ermöglichen und alle Schnittstellen abzusichern, auch wenn der CDD noch nicht verfügbar ist. Durch eine vollständige Automatisierung ist die Erstellung der Simulation nur mit geringem Arbeitsaufwand verbunden.
Das zweite Konzept ermöglicht die vollständige Integration eines CDD, wobei die Hardware-Schnittstellen über einen zusätzlichen Hardware-Abstraction-Layer an die verfügbare Hardware des Systems zur virtuellen Absicherung angebunden werden. So ist der Treiber in der Lage, reale Hardware-Komponenten anzusteuern und kann funktional abgesichert werden. Eine flexible Konfiguration der Abstraktionsschicht erlaubt den Einsatz für eine große Bandbreite von CDDs.
Im Rahmen der Arbeit werden beide Konzepte anhand von industrierelevanten Projekten aus der Serienentwicklung erprobt und detailliert evaluiert.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 2
  - 1.1.1 Software im Fahrzeug . . . 2
  - 1.1.2 Herausforderungen in der Absicherung . . . 4
- 1.2 Problemstellung . . . 5
- 1.3 Zielsetzung der Arbeit . . . 6
- 1.4 Aufbau der Arbeit . . . 6
- 1.5 Zusammenfassung . . . 7

2 Stand der Technik **9**
- 2.1 AUTOSAR . . . 9
  - 2.1.1 Konzept . . . 10
  - 2.1.2 Schichtenmodell der Softwarearchitektur . . . 10
  - 2.1.3 Aufbau einer Software-Komponente . . . 13
  - 2.1.4 Kommunikationsmechanismen . . . 13
  - 2.1.5 Fazit . . . 14
- 2.2 Complex-Device-Drivers . . . 16
  - 2.2.1 Grundlagen . . . 16
  - 2.2.2 Einsatzgebiete . . . 18
  - 2.2.3 Fazit . . . 21
- 2.3 Entwicklung und Test von Software . . . 21
  - 2.3.1 Entwicklungsprozess . . . 21
  - 2.3.2 Qualitätssicherung von Software . . . 22
  - 2.3.3 Fazit . . . 26
- 2.4 Definition der virtuellen Absicherung . . . 26
  - 2.4.1 Begriffsdefinition . . . 26
  - 2.4.2 Abgrenzung zu gängigen Absicherungsmethoden . . . 27
  - 2.4.3 Fazit . . . 28
- 2.5 Virtuelle Absicherungsplattform . . . 30
  - 2.5.1 Konzept . . . 30
  - 2.5.2 Anwendungsfälle . . . 31
  - 2.5.3 Technische Realisierung . . . 32
  - 2.5.4 Weiterentwicklung der Plattform . . . 33
  - 2.5.5 Integrationsprozess . . . 34

2.5.6 Serieneinsatz . . . 34
2.5.7 Fazit . . . 37
2.6 Verwandte Arbeiten . . . 37
2.6.1 Arbeiten zu virtueller Absicherung . . . 37
2.6.2 Arbeiten zu Hardware-Treibern . . . 38
2.6.3 Arbeiten zu Hardware-Simulation . . . 38
2.6.4 Arbeiten zu Hardware-Abstraktion . . . 39
2.7 Zusammenfassung . . . 39

3 Complex-Device-Drivers in der virtuellen Absicherung 41
3.1 Testziele für die Berücksichtigung von Complex-Device-Drivers . . . 41
3.1.1 Integration der Software-Komponenten . . . 41
3.1.2 Sicherstellung der Wiederverwendbarkeit der Funktions-Software . . . 42
3.1.3 Funktionale Absicherung der Software . . . 42
3.1.4 Fazit . . . 43
3.2 Konzepte . . . 44
3.2.1 Simulation des Ziel-Mikrocontrollers . . . 44
3.2.2 Simulation des Hardware-Verhaltens . . . 45
3.2.3 CDD-Simulation . . . 46
3.2.4 Integration der spezifischen Hardware . . . 47
3.2.5 CDD-Hardware-Abstraction-Layer . . . 48
3.2.6 Ziel-Hardware-Emulation . . . 49
3.2.7 Separate Ausführung auf dem Ziel-Mikrocontroller . . . 50
3.2.8 Fazit . . . 51
3.3 Vergleich der Konzepte . . . 52
3.3.1 Bewertungskriterien . . . 52
3.3.2 Bewertung der Konzepte . . . 54
3.3.3 Auswertung . . . 58
3.4 Konzeptauswahl und Methodik für die CDD-Integration . . . 60
3.5 Zusammenfassung . . . 64

4 Detaildarstellung der ausgewählten Konzepte 65
4.1 CDD-Simulation . . . 65
4.1.1 Automatisierte Erzeugung der Simulationskomponenten . . . 65
4.1.2 CDD-Simulation mit Benutzerinteraktion . . . 66
4.1.3 CDD-Simulation durch Funktionsmodellierung . . . 68
4.1.4 Fazit . . . 68
4.2 CDD-Hardware-Abstraction-Layer . . . 69
4.2.1 Integration des Hardware-Abstraction-Layers . . . 69
4.2.2 Struktur des Hardware-Abstraction-Layer . . . 70
4.2.3 Fazit . . . 76
4.3 Zusammenfassung . . . 76

5 Realisierung der ausgewählten Konzepte 77
5.1 CDD-Simulation . . . 77
5.1.1 Automatisierte Erzeugung der Simulationskomponenten . . . 77
5.1.2 CDD-Simulation mit Benutzerinteraktion . . . 80

5.1.3 CDD-Simulation durch Funktionsmodellierung . . . 84
5.1.4 Fazit . . . 85
5.2 CDD-Hardware-Abstraction-Layer . . . 86
5.2.1 Konfiguration und Implementierung . . . 86
5.2.2 Quellcodegenerierung und Integration . . . 91
5.2.3 Fazit . . . 92
5.3 Zusammenfassung . . . 92

6 Evaluierung der Konzepte anhand industrierelevanter Projekte 95
6.1 Außenlichtsteuerung . . . 95
6.1.1 Projektbeschreibung . . . 95
6.1.2 Konzeptauswahl . . . 96
6.1.3 Konzeptanwendung . . . 97
6.1.4 Versuchsaufbau . . . 100
6.1.5 Evaluierung . . . 100
6.2 Head-Up-Display-Treiber . . . 104
6.2.1 Projektbeschreibung . . . 104
6.2.2 Konzeptauswahl . . . 106
6.2.3 Konzeptanwendung . . . 107
6.2.4 Versuchsaufbau . . . 110
6.2.5 Evaluierung . . . 111
6.3 Elektromotortreiber . . . 113
6.3.1 Projektbeschreibung . . . 114
6.3.2 Konzeptauswahl . . . 116
6.3.3 Konzeptanwendung . . . 117
6.3.4 Versuchsaufbau . . . 122
6.3.5 Evaluierung . . . 125
6.4 Zusammenfassung . . . 126

7 Zusammenfassung und Ausblick 129
7.1 Konzepte und Umsetzung . . . 130
7.2 Ergebnisse . . . 130
7.3 Ausblick . . . 131

Glossar 135

Literaturverzeichnis 139

Abbildungsverzeichnis

1.1 Weltweite PKW-Absatzsteigerung deutscher Premium-Hersteller im Vergleich zum Gesamtmarkt [OIC15, Aud15, BMW15, Dai15]. 1
1.2 Funktionszuwachs im Fahrzeug [Fri11]. 3
1.3 Genutzter Speicher für Software im Fahrzeug am Beispiel des BMW 5er [Fri10]. 4
1.4 Complex-Device-Drivers in einem AUTOSAR-Steuergerät (links), bzw. in einer virtuellen Absicherungsplattform (rechts). 5

2.1 AUTOSAR als Abstraktion zwischen Funktions-Software und Hardware [AUT13]. 10
2.2 Der grundlegende AUTOSAR-Ansatz [AUT08]. 11
2.3 AUTOSAR-Schichtenarchitektur [AUT11b]. 11
2.4 Sender-Receiver-Kommunikation mit 1:n-Verknüpfung [AUT08]. 14
2.5 Client-Server-Kommunikation mit n:1-Verknüpfung [AUT08]. 14
2.6 Prozentuale Verteilung der Anwendungscluster von CDDs über die untersuchten Steuergeräte bei der BMW Group. 20
2.7 Prozentuale Verteilung der CDDs mit direkter Hardware-Ansteuerung über die untersuchten Steuergeräte bei der BMW Group. 21
2.8 Das allgemeine V-Modell der Systementwicklung [VMo12]. 22
2.9 Unterscheidung horizontaler und vertikaler Prototypen von Steuergeräte-Software [SZ13]. 26
2.10 Generischer Aufbau eines Systems zur virtuellen Absicherung von Steuergeräte-Software. 27
2.11 Schichtenarchitektur der VAP mit System-under-Test (weiß) und plattformspezifischen Anteilen (blau) [FJM10]. 30
2.12 Anwendungsfälle der VAP entlang des V-Modells der Komponentenentwicklung [FJM10]. 31
2.13 Weiterentwicklung des Aufbaus der VAP [DHM12b]. 33
2.14 Prozess zur Integration von Software auf der VAP im Vergleich zum Steuergerät (ECU). 35

3.1 Simulation des Ziel-Mikrocontrollers in einer VAP. 44
3.2 Simulation des Hardware-Verhaltens für die Schnittstellen des CDD. 45
3.3 Vollständige Simulation des Complex-Device-Drivers. 46
3.4 Integration der zugehörigen, spezifischen Hardware eines CDD in einer VAP. 47

3.5 Anbindung des CDD an die verfügbare VAP-Hardware über einen Hardware-Abstraction-Layer. . . . 48
3.6 Integration eines FPGA zur Emulation der Ziel-Hardware. . . . 49
3.7 Ausführung des CDD auf seiner Ziel-Plattform mit Kopplung an eine VAP. . 50
3.8 Auswahlverfahren für die Konzepte zur Berücksichtigung von CDDs in der virtuellen Absicherung. . . . 61
3.9 Methodik zur Auswahl des optimal geeigneten Konzepts für jeden Anwendungsfall zur Berücksichtigung eines CDD in der virtuellen Absicherung. . . . 63

4.1 Erzeugung der AUTOSAR-Beschreibung für eine CDD-Simulation. . . . 66
4.2 Vervollständigung der AUTOSAR-Beschreibung und Source-Code-Erzeugung für die CDD-Simulation. . . . 67
4.3 Anbindung eines UDP-Panels zur Interaktion mit der CDD-Simulation. . . . 67
4.4 Modellierung der Funktionslogik der CDD-Simulation. . . . 69
4.5 Eingliederung des CDD-Hardware-Abstraction-Layers zwischen Treiber und VAP-Hardware. . . . 70
4.6 Aufbau des CDD-Hardware-Abstraction-Layers. . . . 71
4.7 Aufbau der Emulationsschicht des CDD-HAL. . . . 72
4.8 Aufbau der Routingschicht des CDD-HAL. . . . 73
4.9 Aufbau der Hardware-Abstraktion des CDD-HAL. . . . 74
4.10 Aufbau des Monitors innerhalb des CDD-HAL. . . . 75

5.1 Grafische Oberfläche des SWC-Generators zur Erzeugung der CDD-Simulation, bestehend aus Startseite (links) und *Port*-Auswahl (rechts). . . . 78
5.2 Beispiel einer generierten CDD-Simulation in einer *Composition*. . . . 79
5.3 Beispiel eines generierten *SwcInternalBehavior* für eine CDD-Simulation. . . . 80
5.4 Beispiel eines generierten UDP-Panels für eine CDD-Simulation. . . . 83
5.5 Darstellung einer Software-Komponente zur CDD-Simulation (blau) als Subsystem in MATLAB/Simulink mit zugehöriger Testkomponente (orange). . . . 84
5.6 Darstellung der *Runnables* einer Software-Komponente als Subsysteme in MATLAB/Simulink. . . . 85
5.7 Beispielkonfiguration der Emulation. . . . 87
5.8 Beispielkonfiguration des Routings. . . . 88
5.9 Beispielkonfiguration der Hardware-Abstraktion. . . . 89
5.10 Beispielkonfiguration eines Timed-Monitors. . . . 90

6.1 Aufbau der Außenlichtsteuerung. . . . 96
6.2 Konzeptauswahl für den Anwendungsfall der Außenlichtsteuerung. . . . 97
6.3 Integration der vollständig generierten CDD-Simulation mit den Software-Komponenten der Außenlichtsteuerung. . . . 98
6.4 Integration der nachmodellierten CDD-Simulation mit den Software-Komponenten der Außenlichtsteuerung. . . . 99
6.5 Schematischer Versuchsaufbau für die Integration der Außenlichtsteuerung mit Restbussimulation. . . . 101
6.6 Schematischer Aufbau eines Head-Up-Displays. . . . 104
6.7 Software- und Hardware-Architektur von Instrumenten-Kombination und Head-Up-Display. . . . 105

6.8 Konzeptauswahl für den Anwendungsfall des HUD-Treibers. 106
6.9 Anwendung des CDD-HAL-Konzepts auf den Head-Up-Display-Treiber. . . . 108
6.10 Schreibzugriff des HUD-Treibers über den CDD-HAL auf das Head-Up-Display. 109
6.11 Lesezugriff des HUD-Treibers über den CDD-HAL auf das Head-Up-Display. 109
6.12 Schematischer Versuchsaufbau für die Integration des HUD-Treibers mit Restbussimulation. 110
6.13 Schematischer Aufbau und Ansteuerung einer permanenterregten Synchronmaschine. 113
6.14 Schematischer Aufbau des MC-ISAR eMotor-Drivers [Inf12]. 114
6.15 Verlauf der Encodersignale. 115
6.16 Konzeptauswahl für den Anwendungsfall des Elektromotortreibers. 116
6.17 Integration des MC-ISAR eMotor-Drivers auf einer VAP mit dem CDD-HAL-Konzept. 117
6.18 Konfiguration der Emulationsschicht für die Integration des Elektromotortreibers. 118
6.19 Konfiguration des Routings für die Integration des Elektromotortreibers. . . . 119
6.20 Konfiguration der Hardware-Abstraktion für die Integration des Elektromotortreibers. 120
6.21 Konfiguration des Monitors für die Integration des Elektromotortreibers. . . . 121
6.22 MATLAB/Simulink-Modell einer Synchronmaschine und der zugehörigen Elektronik. 123
6.23 Versuchsaufbau für die Ansteuerung einer simulierten Synchronmaschine mit dem Elektromotortreiber. 124
6.24 Vergleichsmessung des Tastverhältnisses des PWM-Kanals 1 auf der VAP und dem Entwicklungsboard. 125

Tabellenverzeichnis

2.1 Zugriffsmöglichkeiten von CDDs zu BSW-Modulen [AUT11b]. 16
2.2 Anwendungscluster von CDDs mit zugehörigen Beispielen. 19
2.3 Maßnahmen zur Qualitätssicherung von Software [SZ13, MIS94]. 23
2.4 Vergleich von Eigenschaften und Anwendungsfällen der XiL-Methoden und RCP mit der virtuellen Absicherung. 29

3.1 Bewertungsschema für den Integrationsaufwand. 52
3.2 Bewertungsschema für das Anwendungsspektrum. 53
3.3 Bewertungsschema für die Testzielerreichbarkeit. 53
3.4 Bewertungsschema für die Verfügbarkeit. 54
3.5 Zusammenfassung der Bewertung der Konzepte. 59
3.6 Zuordnung der Indizes für Anwendungsfälle von CDDs in der virtuellen Absicherung. 61

6.1 Beispiel eines Testfalls der Außenlichtsteuerung. 102
6.2 Durchgeführte Testfälle für die Außenlichtsteuerung mit zugehörigen Ergebnissen. 103
6.3 Beispiel eines Testfalls für den HUD-Treiber. 111
6.4 Durchgeführte Testfälle für den HUD-Treiber mit zugehörigen Ergebnissen. . 112

Quellcodebeispielverzeichnis

2.1 Direkter Zugriff auf ein Register. 17
2.2 Direkter Zugriff auf einen Speicherbereich. 17
2.3 Implementierung einer Low-Level-Funktion. 18
2.4 Aufruf einer Low-Level-Funktion in einem CDD. 18

5.1 Generierter Quellcode für ein *Runnable* einer CDD-Simulation. 82

Kapitel 1

Einleitung

Der Bedarf nach individueller Mobilität und somit die Nachfrage an Kraftfahrzeugen nimmt weltweit zu. Das Wachstum ist in den letzten Jahren zwar zurückgegangen, jedoch entwickelt sich der Gesamtabsatz weiterhin positiv. Besonders im Premiumsegment werden überdurchschnittliche Ergebnisse erreicht (siehe Abbildung 1.1). Einer der vielfältigen Gründe dafür ist die Ausbildung einer immer größeren, zahlungskräftigen Mittelschicht in den sogenannten BRICS-Staaten Brasilien, Russland, Indien, China und Südafrika [O'N01].

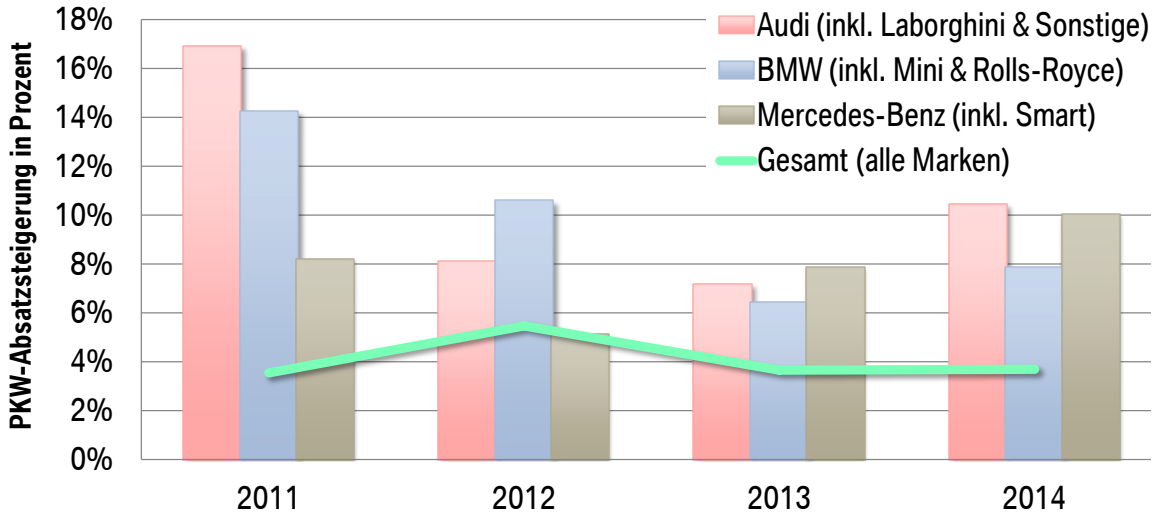


Abbildung 1.1: Weltweite PKW-Absatzsteigerung deutscher Premium-Hersteller im Vergleich zum Gesamtmarkt [OIC15, Aud15, BMW15, Dai15].

Gleichzeitig steigt der Wunsch der Kunden nach Individualisierung. Daher bieten Fahrzeughersteller eine immer größere Auswahl an Modellvarianten ihrer Fahrzeuge an, die sich durch individuelle Konfigurationsmöglichkeiten noch stärker personalisieren lassen. Laut [CAR11] wurden im Jahr 1995 von allen Automobilherstellern noch 227 verschiedene Modellreihen auf dem deutschen PKW-Markt angeboten. Bis zum Jahresende 2015 soll sich die Anzahl bereits auf 415 erhöhen. Bei einzelnen Herstellern fällt diese Entwicklung noch deutlicher aus. Die

BMW Group hat z. B. die Anzahl ihrer Modellvarianten von 17 im Jahr 2005 bereits auf 37 im Jahr 2014 gesteigert [BMW05, BMW14].
Dieses Kapitel beschäftigt sich mit den Auswirkungen der steigenden Variantenvielfalt und den wachsenden Kundenansprüchen auf den Funktionsumfang in aktuellen Fahrzeugen sowie den resultierenden Herausforderungen für die Absicherung. Daraus wird der Fokus dieser Arbeit abgeleitet und die Zielsetzung definiert.

1.1 Motivation

Im selben Maß wie die Modellvielfalt steigen auch die Ansprüche der Kunden an neue Fahrzeuge. Immer mehr Funktionen halten Einzug in die PKW aller Hersteller. Der Fortschritt zeigt sich z. B. bei der Unterstützung des Fahrers durch Assistenzsysteme, in der Unterhaltungselektronik und bei aktiven und passiven Sicherheitssystemen. Navigationssysteme sind weit verbreitet und bieten immer mehr Möglichkeiten, wie die Anzeige von Staumeldungen in Echtzeit. Head-Up-Displays zeigen dem Fahrer wichtige Informationen direkt in der Windschutzscheibe, ein Tempomat mit Abstandsregelung passt die Geschwindigkeit automatisch an das vorausfahrende Fahrzeug an, Kamerasysteme erkennen Verkehrsschilder selbständig und Parkmanöver-Assistenten übernehmen vollautomatisch den Einparkvorgang. Der Trend geht zu immer umfangreicheren Fahrerassistenzsystemen, die den Fahrer stärker entlasten sollen und es existieren bereits erste Prototypen, die vollständig autonome Fahrten ermöglichen. Abbildung 1.2 verdeutlicht den Umfang des Funktionszuwachses von den 70er Jahren bis heute und wagt eine Prognose bis 2020.
Laut [Con14] ist die grundsätzliche Akzeptanz von Fahrerassistenzsystemen in der Bevölkerung hoch. Automatisiertes Fahren wird von den Befragten der Studie überwiegend als nützlicher Fortschritt eingestuft, jedoch bestehen noch Zweifel bezüglich der Zuverlässigkeit. Dabei zeigen Erfahrungen mit aktuellen Fahrerassistenzsystemen einen positiven Einfluss auf die Bewertung des automatisierten Fahrens. Daraus lässt sich ableiten, dass die Verbreitung von Assistenzsystemen auch in Zukunft weiter zunehmen wird.
Insgesamt haben die neuen Funktionen eine Gemeinsamkeit: 90 % der künftigen Innovationen im Automobil basieren auf Elektronik, wobei der Software-Anteil in den Elektronik-Komponenten stetig zunimmt [VDA05].

1.1.1 Software im Fahrzeug

In aktuellen Fahrzeugen befinden sich, je nach Baureihe und Hersteller, bis zu 100 elektronische Steuergeräte [Bro13]. Dabei handelt es sich um eingebettete Systeme, wie sie z. B. in [Har01] anhand ihrer Haupteigenschaften erläutert werden. Steuergeräte im Speziellen bestehen aus mindestens einem Mikrocontroller und verschiedenen Ein- und Ausgängen zur seriellen Kommunikation mit anderen Steuergeräten (sog. Bussysteme) oder zur Ansteuerung von Sensoren und Aktoren (sog. digitale/analoge Ein- und Ausgänge). Insgesamt kontrollieren die Steuergeräte in einem Fahrzeug der oberen Mittelklasse mit Vollausstattung laut [RDT04] über 200 elektrische Komponenten und führen laut [Bro13] 100 Millionen Zeilen Programmcode aus. Dies macht das Fahrzeug aus technischer Sicht zu dem komplexesten Konsumgut unserer Zeit [SZ13].
Der zuvor aufgezeigte, wachsende Funktionsumfang in den Fahrzeugen führt zu einer immer höheren Komplexität der Software in den Steuergeräten. Zusätzlich besteht für neuartige

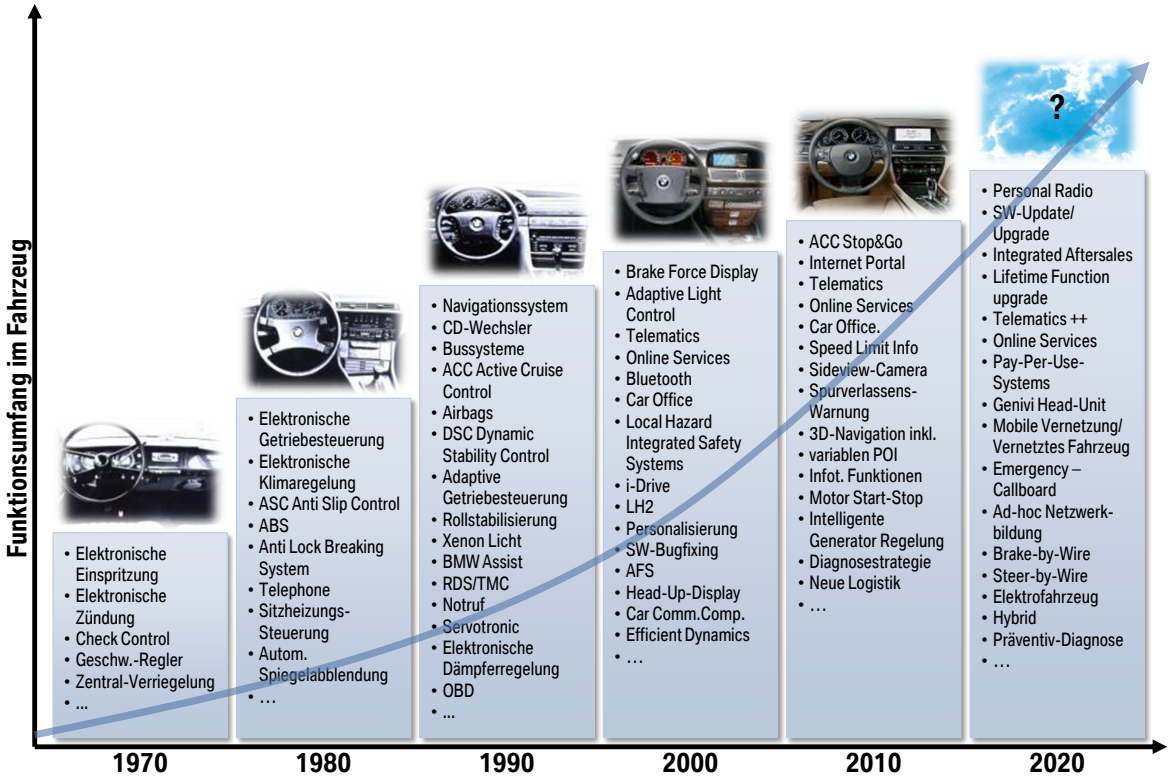


Abbildung 1.2: Funktionszuwachs im Fahrzeug [Fri11].

Fahrerassistenzsysteme der Bedarf nach vielfältigen Informationen über das Fahrzeugumfeld, wodurch die Vernetzung der Steuergeräte untereinander ebenso zunimmt und der Entwicklungsaufwand steigt. Laut [VDA05] werden 50 % bis 70 % der Entwicklungskosten von Steuergeräten durch Software verursacht, wobei sich der Software-Umfang alle zwei bis drei Jahre verdoppelt.
Dieser Verlauf steht in direktem, kausalem Zusammenhang zu dem genutzten Speicher für Software im Fahrzeug, welcher zur Veranschaulichung herangezogen werden soll. Für einen 5er BMW im Jahr 2002 hatte die Software aller Steuergeräte im Fahrzeug einen Speicherbedarf von 70 MB. Über die Produktlaufzeit hat sich dieser Wert durch zusätzliche Funktionen und Verbesserungen bereits auf 140 MB verdoppelt. Navigationsdaten wurden über separate DVDs bereitgestellt und nicht in den Steuergeräten abgelegt. In der nächsten Generation des 5er BMW aus dem Jahr 2010 belegt die Software bereits 1 GB, wobei noch ein Vielfaches für die Navigationsdaten auf einer Festplatte hinzukommt, wie Abbildung 1.3 zeigt. Bis zum Ende des Produktlebenszyklus im Jahr 2018 wird sich der Speicherbedarf schätzungsweise auf 4 bis 8 GB vervielfachen. Für die darauffolgende Produktgeneration ist erneut mit einer deutlichen Steigerung zu rechnen [Fri10].

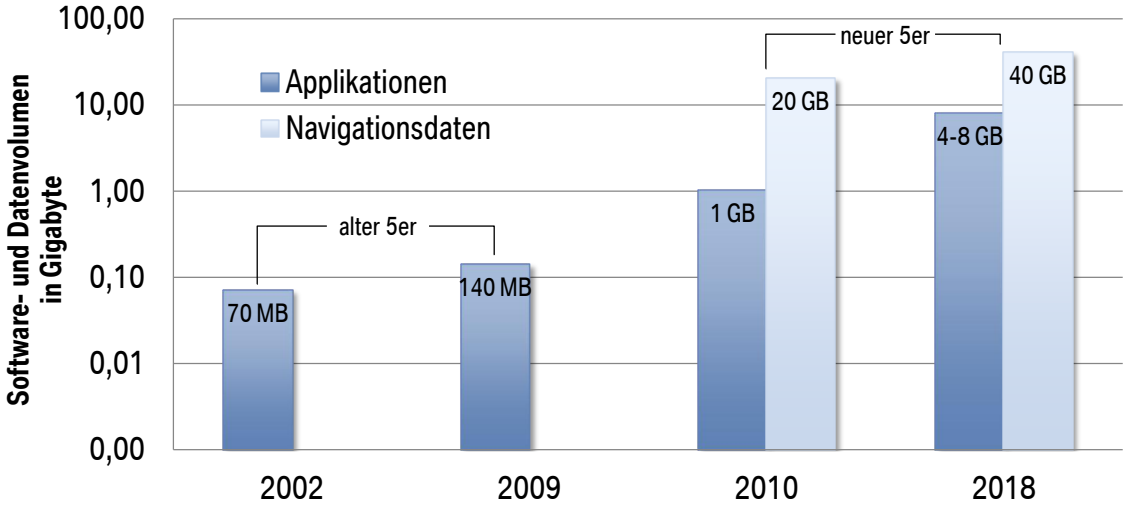


Abbildung 1.3: Genutzter Speicher für Software im Fahrzeug am Beispiel des BMW 5er [Fri10].

1.1.2 Herausforderungen in der Absicherung

Die stark ansteigende Vielfalt der Fahrzeugmodellvarianten, der Funktionszuwachs und die zunehmende Vernetzung innerhalb der Steuergeräte stellen eine besondere Herausforderung für die Absicherung dar. Viele Steuergeräte werden als Baukastenelemente entwickelt und sollen in verschiedenen Baureihen wiederverwendbar sein. Dabei ist das Zusammenspiel für jede einzelne Modellvariante sicherzustellen. Verteilte Funktionen auf mehreren Steuergeräten müssen zuverlässig abgesichert werden. Dabei sind alle angebotenen Konfigurationsvarianten zu berücksichtigen. Ebenso müssen gesetzliche Anforderungen zur funktionalen Sicherheit, z. B. sogenannte *Automotive-Safety-Integrity-Level (ASIL)* [ISO11], eingehalten werden.

Um den erhöhten Absicherungsaufwand bewältigen zu können, sind neue Methoden erforderlich, die bei geringem Kostenaufwand die hohen Qualitätsanforderungen erfüllen können. Über das Prinzip des *Frontloading* [WHR08] im Entwicklungsprozess soll so viel Entwicklungs- und Absicherungsaufwand wie möglich in die frühen Prozessstufen verlagert werden. Dies hilft bei der Kostenbeherrschung, da sich nach der 10er-Regel die Kosten zur Behebung eines Fehlers in jeder Prozessstufe verzehnfachen [SP10]. Zur Erhöhung des *Frontloading* dienen bekannte Methoden wie *Rapid-Control-Prototyping (RCP)* und *X-in-the-Loop (XiL)*, wie z. B. *Hardware-in-the-Loop (HiL)* oder *Software-in-the-Loop (SiL)* (siehe Kapitel 2.3.2).

Neues Potenzial zur Kostenreduktion und Qualitätssteigerung bietet die virtuelle Absicherung, wie sie bei der BMW Group eingesetzt wird. Die virtuelle Absicherungsplattform (VAP) bietet die Möglichkeit, Steuergeräte-Software unabhängig von ihrer Ziel-Hardware zu testen und abzusichern (siehe Kapitel 2.5). Die Plattform besteht aus einem PC-System zur Integration von Software mit definierten Standard-Schnittstellen. Ermöglicht wird dies durch den konsequenten Einsatz von Software-Standards wie AUTomotive Open System ARchitecture (AUTOSAR) [AUT13] und GENIVI [GEN12], welche die Entkopplung von Hard- und Software sicherstellen. Somit kann die Software bereits vor der Verfügbarkeit erster Hardwaremuster in Betrieb genommen werden [FJM10, MCW12].

Der Ansatz der VAP wurde bei der BMW Group bereits erfolgreich pilotiert und wird inzwischen in verschiedenen Serienprojekten angewendet. Dabei konnte seit dem Jahr 2010

eine große Anzahl signifikanter Fehler in sehr frühen Entwicklungsphasen aufgedeckt und behoben werden [DHM12b] (siehe Kapitel 2.5.6).

1.2 Problemstellung

Die virtuelle Absicherungsplattform (VAP) ist in ihrer bisherigen Ausführung ideal geeignet, um vollständig AUTOSAR-konforme, hardwareunabhängige Software-Komponenten zu integrieren, zu testen und abzusichern. Allerdings besteht die Funktions-Software in aktuellen Steuergeräten grundsätzlich nicht nur aus vollständig hardwareunabhängigen Anteilen. Der AUTOSAR-Standard (siehe Kapitel 2.1) sieht explizit die Möglichkeit vor, sehr stark auf eine spezifische Hardware-Komponente zugeschnittene und optimierte Software-Anteile in ein Steuergerät mit einzubringen. Diese Komponenten werden als Complex-Device-Driver (CDD) bezeichnet und dienen zur Erfüllung verschiedener Anforderungen (siehe Kapitel 2.2). Sie ermöglichen die Anbindung von Hardware, welche bisher nicht im Software-Standard berücksichtigt wurde. Weiterhin sind CDDs erforderlich, um besonders kritische Echtzeitanforderungen zu erfüllen oder Anforderungen aus der funktionalen Sicherheit nach [ISO11] umzusetzen. Nicht zuletzt besteht die Option, Legacy-Code gekapselt als CDD in ein Steuergerät einzubringen, welcher noch nicht nach dem AUTOSAR-Standard entwickelt wurde. Die Einordnung von Complex-Device-Drivers in die Architektur von AUTOSAR-Steuergeräten ist in Abbildung 1.4 links dargestellt.

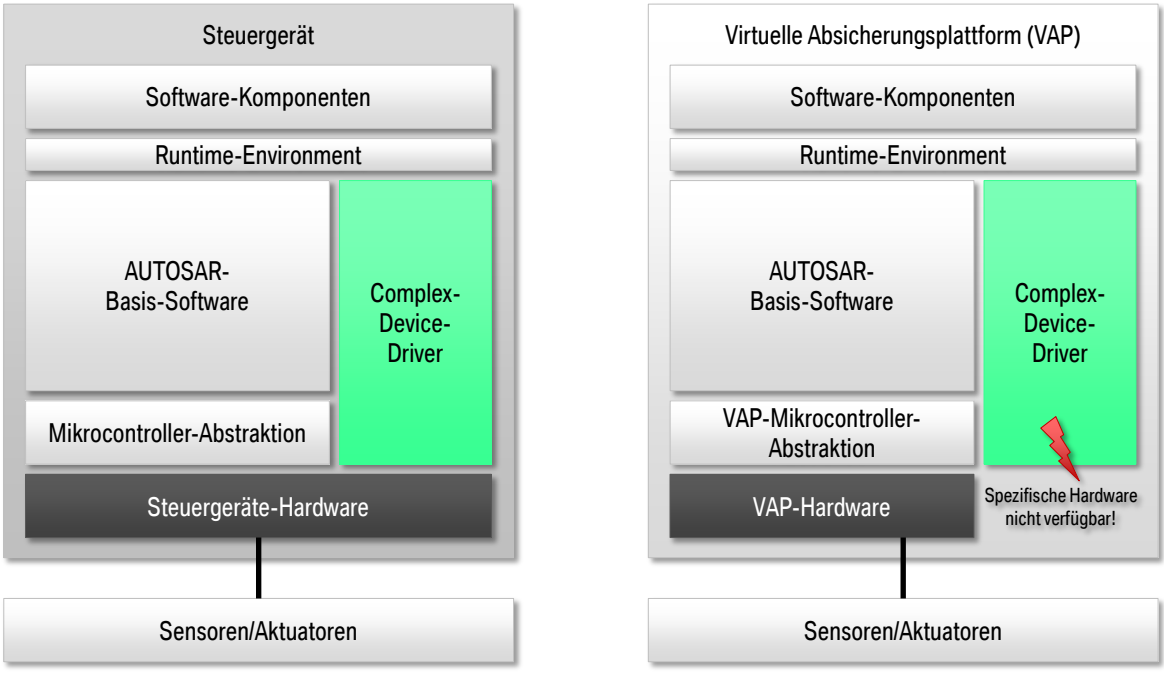


Abbildung 1.4: Complex-Device-Drivers in einem AUTOSAR-Steuergerät (links), bzw. in einer virtuellen Absicherungsplattform (rechts).

Soll ein CDD in ein hardwareunabhängiges System wie die VAP integriert werden, ergibt sich ein Problem: Die spezifische Hardware, für die der Treiber ausgelegt ist, steht normalerweise in einem generischen System nicht zur Verfügung. Somit stellt die Integration eines CDD

ein nicht triviales Problem dar. Die CDD-Schnittstellen, welche im realen Steuergerät direkt mit der Hardware verbunden sind, können nicht bedient werden, wie in Abbildung 1.4 rechts verdeutlicht. Der Treiber ist jedoch ein Bestandteil der Software und setzt relevante Funktionsumfänge des Steuergerätes um. Selbst wenn ein CDD nicht im Fokus der Absicherung steht, so besitzt er dennoch Schnittstellen zu Funktions-Software-Komponenten, welche ohne den Treiber nicht sinnvoll geprüft werden können. Daher müssen Complex-Device-Drivers in einem ganzheitlichen Ansatz, zur virtuellen Absicherung der vollständigen Steuergeräte-Software, in jedem Fall berücksichtigt werden.

1.3 Zielsetzung der Arbeit

Diese Arbeit beschäftigt sich mit dem oben erläuterten Problem der Berücksichtigung von hardwareabhängigen Software-Komponenten (genannt Complex-Device-Driver (CDD)) in einer von der Ziel-Hardware unabhängigen Plattform, wie der virtuellen Absicherungsplattform (VAP). Dabei sollen valide Konzepte für den Umgang mit CDDs erarbeitet werden, welche die Absicherung der vollständigen Steuergeräte-Software auf der VAP ermöglichen. Die Konzepte müssen die gesamte Bandbreite der Anwendungsfälle und Testziele der VAP abdecken. Über ein Auswahlverfahren ist sicherzustellen, dass für jedes Testziel das optimale Konzept selektiert werden kann. Weiterhin ist die prototypische Realisierung ausgewählter Konzepte gefordert. Dabei sollen die Ansätze an aussagekräftigen, industrierelevanten Steuergeräte-Projekten validiert und evaluiert werden.

1.4 Aufbau der Arbeit

In Kapitel 1 wurde bereits die Motivation für diese Dissertation beschrieben und auf die zu lösende Problemstellung sowie die Zielsetzung der Arbeit eingegangen.
Das folgende Kapitel stellt den Stand der Technik dar, auf welchem die weiteren Inhalte der Arbeit aufbauen. Dabei wird der Software-Standard AUTOSAR beschrieben und die sogenannten Complex-Device-Drivers (CDDs) eingeführt, welche in der weiteren Ausarbeitung im Fokus stehen. Der Begriff der virtuellen Absicherung wird definiert und die virtuelle Absicherungsplattform (VAP) im Detail beschrieben. Es folgt eine Betrachtung verwandter Arbeiten zu dem Thema.
In Kapitel 3 wird die gesamte Bandbreite der erarbeiteten Konzepte zur Lösung des Problems dargestellt. Durch definierte Bewertungskriterien erfolgt die Auswahl der am besten geeigneten Varianten für die Realisierung. Schließlich wird ein Modell zur Vorgehensweise bei der Integration von CDDs in eine VAP beschrieben, das die verschiedenen Eigenschaften der Treiber berücksichtigt und die Auswahl des jeweils besten Konzepts ermöglicht.
Die zuvor selektierten Konzepte, welche am besten für den praktischen Einsatz geeignet sind, werden in Kapitel 4 detailliert ausgearbeitet. Dies beinhaltet eine Darstellung der Struktur der jeweiligen Lösungen und der Einsatzmöglichkeiten.
Es folgt die prototypische Realisierung der Konzepte in Kapitel 5, wobei auf die Umsetzung und die Anwendung der Lösungen in der Praxis eingegangen wird.
In Kapitel 6 folgt die Validierung der ausgewählten Konzepte. Die Lösungen werden auf CDDs aus serienrelevanten Steuergeräteprojekten angewendet, um die Praxistauglichkeit zu evaluieren.

Die Arbeit schließt mit einer Zusammenfassung in Kapitel 7. Dabei werden die wichtigsten Erkenntnisse herausgestellt und ein Ausblick auf potenzielle Weiterführungen des Themas gegeben.

1.5 Zusammenfassung

Die aktuelle Situation in der Automobilindustrie bietet einige Herausforderungen, wie die Steigerung der Derivat-Vielfalt, den Funktionszuwachs und die daraus folgende Komplexitätssteigerung der Steuergeräte-Software. Diese Entwicklung ist anhand der exponentiellen Steigerung des Speicherbedarfs für Software im Fahrzeug darstellbar. Im Absicherungsprozess entstehen dadurch erhöhte Aufwände und Kosten. Eine Lösung zur Beherrschung der Komplexität bietet die virtuelle Absicherung.
Hardwareabhängige Komponenten der Steuergeräte-Software sind jedoch nicht problemlos in die virtuelle Absicherung einzubeziehen, da die zugehörige Hardware auf einer generischen Plattform nicht zur Verfügung steht. Diese Dissertation befasst sich daher mit der Berücksichtigung dieser sogenannten Complex-Device-Drivers (CDDs) in der virtuellen Absicherung auf einer Plattform unabhängig von der Ziel-Hardware. Das Ziel der Arbeit ist die Entwicklung valider Konzepte für den Einsatz von CDDs in der virtuellen Absicherung. Durch ein Auswahlverfahren für das jeweils am besten geeignete Konzept für einen spezifischen Anwendungsfall ist dabei die Absicherung der vollständigen Steuergeräte-Software zu ermöglichen.

Kapitel 2

Stand der Technik

Dieses Kapitel beschreibt den Stand der Technik, der für diese Dissertation relevant ist. Zunächst wird eine Einführung in den Software-Standard AUTOSAR und die hardwareabhängigen Komponenten (genannt Complex-Device-Drivers (CDDs)) gegeben, welche im Fokus dieser Arbeit stehen. Es folgt eine Definition der virtuellen Absicherung im Bezug auf Steuergeräte-Software und eine detaillierte Beschreibung der VAP. Verwandte Arbeiten zu der behandelten Thematik werden ebenfalls berücksichtigt.

2.1 AUTOSAR

Das AUTOSAR-Konsortium (AUTomotive Open System ARchitecture) ist ein internationaler Zusammenschluss von Automobilherstellern und Zulieferern. Die sogenannten *Core-Partner* sind die BMW Group, Robert Bosch GmbH, Continental AG, Daimler AG, Ford Motor Company, General Motors Holding LLC, Peugeot Citroën Automobiles S.A., Toyota Motor Corporation und die Volkswagen AG. Insgesamt sind, seit der Gründung im Jahr 2003, über 160 Unternehmen als *Premium-Member*, *Associated-Member* und *Development-Member* beteiligt. Die Ziele des Konsortiums bestehen darin, die zunehmende Komplexität in der Entwicklung von Steuergeräte-Software beherrschbar zu machen, Innovationen zu ermöglichen und hohe Qualitätsansprüche zu erfüllen. Dafür wurde eine Software-Architektur, ein Satz von Spezifikationen, Anwendungsschnittstellen und eine Methodik entwickelt, welche den industrieweiten Software-Standard AUTOSAR bilden. Der Grundgedanke besteht dabei darin, die Funktions-Software von der darunterliegenden Hardware zu abstrahieren und dadurch den einfachen Austausch von Software-Komponenten über verschiedene Hardware-Plattformen, Fahrzeugbaureihen und sogar Original-Equipment-Manufacturers (OEMs) zu ermöglichen. Somit sollen Zuverlässigkeit und Qualität erhöht und gleichzeitig die Kosten in Grenzen gehalten werden. Im Mai 2006 wurde die erste produktiv nutzbare Version, das Release 2.0 des Standards, veröffentlicht. Die Spezifikation wurde konsequent weiterentwickelt und steht seit Oktober 2014 im Release 4.2.1 zur Verfügung. Inzwischen ist der Standard in der Industrie weit verbreitet und wird von der Mehrheit der Mitglieder des Konsortiums produktiv eingesetzt [AUT11a, AUT13].

2.1.1 Konzept

AUTOSAR setzt auf das Prinzip der Abstraktion von Funktions-Software und Hardware. Dafür definiert der Standard eine Middleware, die einheitlich den Zugriff auf sämtliche Hardware-Komponenten bestimmt, die sogenannte AUTOSAR-Basis-Software (BSW) [AUT11c]. Dies soll die Portierung von Software-Komponenten auf andere Steuergeräte-Hardware vereinfachen und somit die Wiederverwendbarkeit der Software sicherstellen. Das Prinzip ist in Abbildung 2.1 dargestellt.

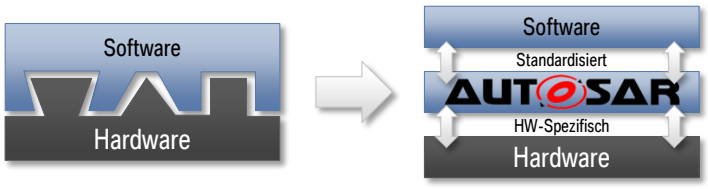


Abbildung 2.1: AUTOSAR als Abstraktion zwischen Funktions-Software und Hardware [AUT13].

Weiterhin definiert der Standard die Beschreibungsform aller Moduldefinitionen und Schnittstellen, welche im Format der Extensible-Markup-Language (XML) [BPSM+08] abgelegt werden. Die AUTOSAR-Methodology [AUT11d] beschreibt detailliert die Konzepte und Zusammenhänge hinter den einzelnen Beschreibungsdateien. Ein einfacher Überblick soll in Abbildung 2.2 gegeben werden.

Die Funktions-Software gliedert sich in einzelne *Software-Components (SWCs)* von denen jede über eine sogenannte *Software-Component-Description (SWCD)* [AUT11e] verfügen muss. Die Kommunikation aller Komponenten untereinander erfolgt über den *Virtual-Functional-Bus (VFB)* [AUT11j], welcher eine abstrakte Verbindung der Komponenten über Funktions- und Steuergerätegrenzen hinweg darstellt. Um die Komponenten auf die jeweiligen Steuergeräte zu partitionieren, sind weitere Beschreibungsdateien nötig. Im *System-Template* [AUT11i] wird die Kommunikationsmatrix der Steuergeräte untereinander festgelegt. Daraus wird für jedes Steuergerät die *ECU-Description* [AUT11g] erzeugt, welche mit der Konfiguration aller verwendeten BSW-Module (unterteilt in einzelne *BSW-Module-Descriptions* [AUT11f]) angereichert wird. Zusammen mit optionalen *System-Constraint-Descriptions* kann dann die vollständige Software für jedes Steuergerät generiert werden. Aus dem abstrakten VFB wird eine spezifische Implementierung der Kommunikation für jedes Steuergerät erzeugt, welche als *Runtime-Environment (RTE)* [AUT11h] bezeichnet wird.

2.1.2 Schichtenmodell der Softwarearchitektur

Die AUTOSAR-Architektur eines einzelnen Steuergerätes gliedert sich im Wesentlichen in vier Schichten, wobei sich die eigentliche Basis-Software (BSW) noch weiter unterteilen lässt (siehe Abbildung 2.3). Jedes Modul dieser Architektur wird in textueller Form in den Dokumenten des Standards, im Bezug auf seine Schnittstellen und Funktionalität, spezifiziert [AUT11b]. Im Folgenden werden die einzelnen Schichten zusammenfassend beschrieben.

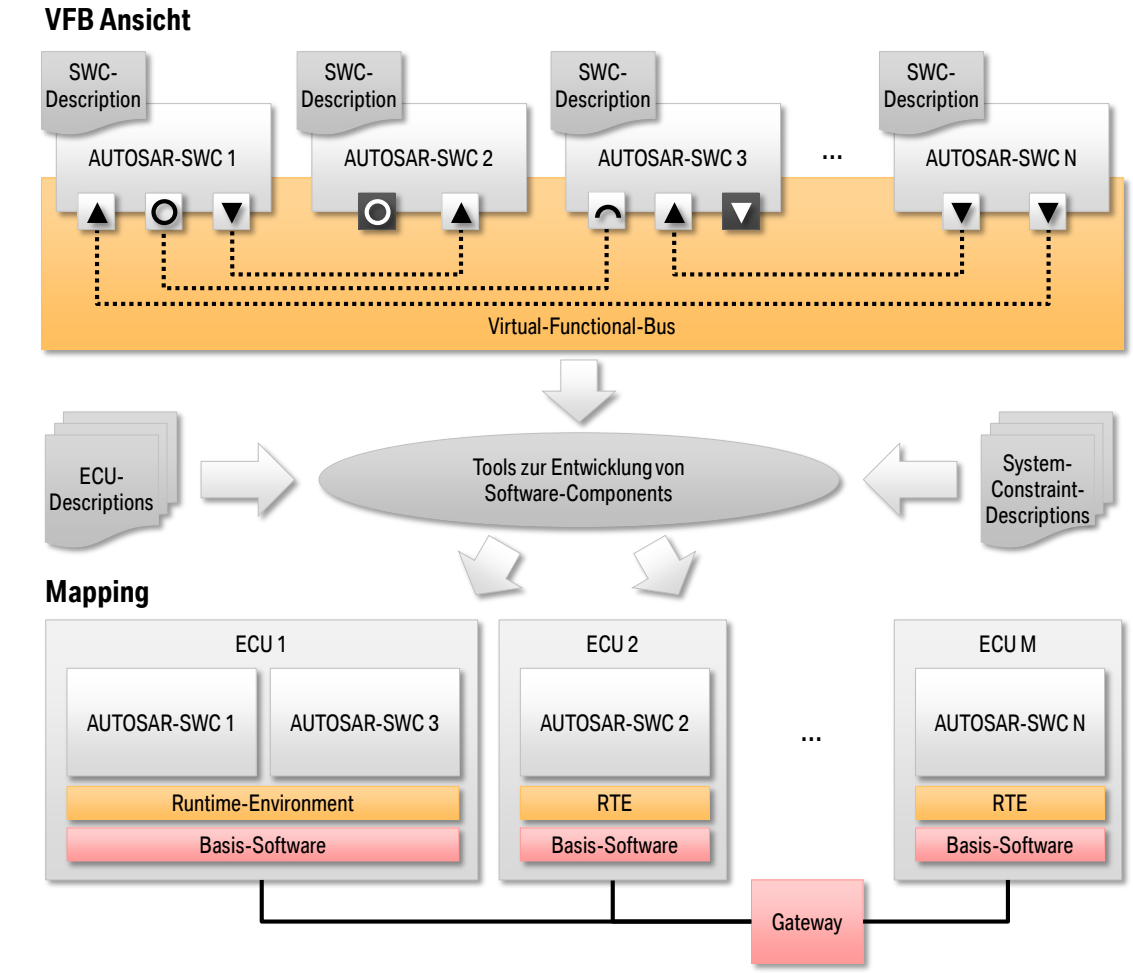


Abbildung 2.2: Der grundlegende AUTOSAR-Ansatz [AUT08].

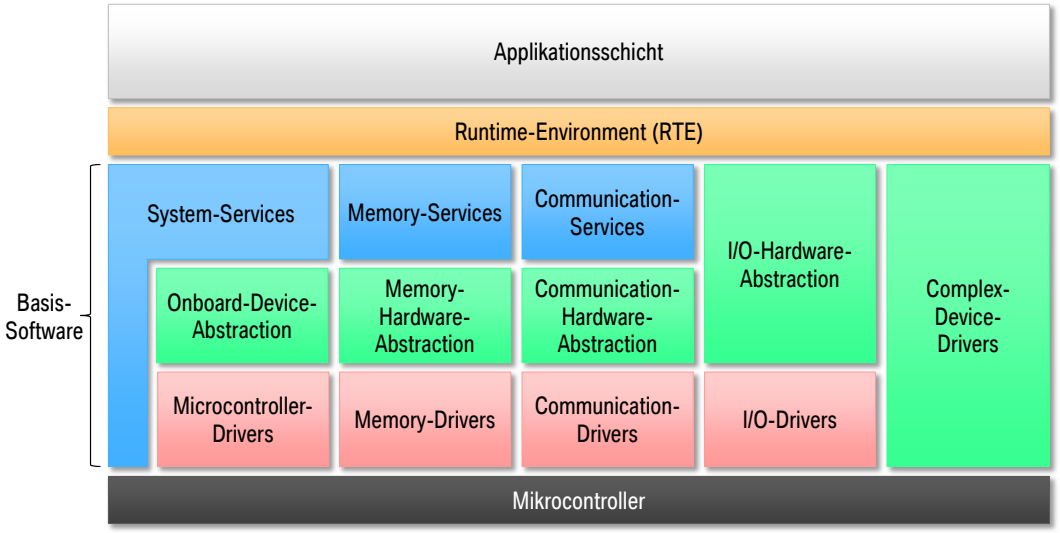


Abbildung 2.3: AUTOSAR-Schichtenarchitektur [AUT11b].

Applikationsschicht

Die oberste Ebene bildet die *Applikationsschicht*. Sie enthält alle hardwareunabhängigen Anteile der eigentlichen Funktions-Software. Diese sind in einzelne *Software-Components (SWCs)* unterteilt, welche wiederum die eigentliche Funktionsimplementierung enthalten. Zu jeder Komponente muss es eine *Software-Component-Description (SWCD)* geben, welche die einzelnen Funktionen innerhalb einer SWC als sogenannte *Runnables* definiert und alle Schnittstellen nach außen beschreibt. Durch das Abstraktionskonzept von AUTOSAR muss der Funktionsentwickler bei der Erstellung einer SWC nicht festlegen, welche Schnittstellen innerhalb des Steuergerätes eine Verbindung zu anderen Komponenten herstellen, oder welche Daten über Bussysteme zu anderen Steuergeräten übertragen sollen. Dadurch wird die Funktions-Software hardwareunabhängig und kann leicht zwischen verschiedenen Steuergeräten ausgetauscht werden.

Runtime-Environment

Jegliche Schnittstellen der SWCs interagieren über die *Runtime-Environment (RTE)*, welche die darunterliegende Schicht darstellt. Die RTE ist die Realisierung des *Virtual-Functional-Bus (VFB)* innerhalb eines Steuergerätes. Prinzipiell stellt sie einen Datenpuffer dar, der die Kommunikation aller SWCs untereinander und zu den darunterliegenden Schichten ermöglicht.

Basis-Software

Unterhalb folgen die Module der *Basis-Software (BSW)*, welche weiterhin in drei Ebenen unterteilt sind, wie in Abbildung 2.3 dargestellt. Direkt an die RTE grenzt dabei der *Service-Layer* (blau in Abbildung 2.3). Er beinhaltet das AUTOSAR-Betriebssystem, die Module zur Speicher- und Kommunikationsverwaltung und eine Vielzahl weiterer Verwaltungs- und Überwachungsfunktionen. Diese Module sind, abgesehen vom Betriebssystem, noch unabhängig von der Hardware gestaltet. Die nächste Ebene ist der *ECU-Abstraction-Layer* (grün in Abbildung 2.3). Hier sind alle Module enthalten, welche eine Abstraktion der Treiberschicht darstellen. Dies sind die *Input/Output-Hardware-Abstraction* sowie die Schnittstellenmodule aller Bussysteme und des Speichers. Hier sind Hardwareabhängigkeiten nicht mehr gänzlich zu vermeiden, da hier bereits klar sein muss, ob z. B. ein Controller-Area-Network (CAN) oder ein FlexRay als Datenübertragungskanal genutzt werden soll. Darunter folgt der *Microcontroller-Abstraction-Layer (MCAL)*, welcher aus den direkten Hardware-Treibern der gesamten Steuergeräte-Peripherie besteht. Somit liegt hier eine direkte Hardwareabhängigkeit vor.
Eine Sonderrolle nehmen die sogenannten *Complex-Device-Drivers (CDDs)* ein, die parallel zu den drei Schichten der BSW in Abbildung 2.3 angeordnet sind. Ihre Eigenschaften werden in Kapitel 2.2 detailliert beschrieben.

Mikrocontroller

Die Treiber setzen direkt auf der untersten Schicht der Architektur auf. Dabei handelt es sich um den eigentlichen Mikrocontroller, auf welchem die gesamte AUTOSAR-Software ausgeführt wird. Über die jeweiligen Schnittstellen des Mikrocontrollers erfolgt die Anbindung der Steuergeräte-Peripherie.

2.1.3 Aufbau einer Software-Komponente

Eine einzelne Software-Komponente besteht aus mindestens zwei Dateien, einer *Software-Component-Description (SWCD)* (im Format AUTOSAR-XML (ARXML)) und einer oder mehreren *Source-Code-Dateien* in den Programmiersprachen C oder C++.

Software-Component-Description

Die SWCD enthält alle Informationen, die für den AUTOSAR-Standard relevant sind. Dabei handelt es sich um die Definition der SWC selbst, mit ihren Schnittstellen und dem sogenannten *Internal-Behavior*. Die Schnittstellen sind sämtliche *Ports*, die zum Datenaustausch mit anderen Komponenten und der BSW genutzt werden. Die Kommunikation erfolgt dabei entweder nach dem *Sender-Receiver-* oder *Client-Server-Prinzip* (siehe Kapitel 2.1.4). Für jeden Port muss eine gesonderte *Interface-Definition* angelegt werden, die die übertragenen Datenelemente mit ihren Datentypen definiert.
Das *Internal-Behavior* besteht aus der Definition der einzelnen Funktionen der SWC, welche als *Runnables* bezeichnet werden. Für jedes *Runnable* werden *Events* definiert, welche die Bedingungen enthalten, die zur Ausführung der Funktion führen sollen. Dies können z. B. zyklische Zeitbedingungen (*Timing-Events*) oder der Empfang bestimmter Datenelemente (*On-Data-Receive-Events*) sein.

Source-Code

Der *Source-Code* der SWC muss für jedes *Runnable* eine C- oder C++-Funktion enthalten, deren Name und Variablen denen aus der SWCD entsprechen müssen. Ansonsten kann die C-Syntax wie gewohnt eingesetzt werden. Im Automobil-Umfeld hat sich dabei der Standard MISRA-C:2012 [MIS12] der Motor Industry Software Reliability Association (MISRA) durchgesetzt.

2.1.4 Kommunikationsmechanismen

Für die Kommunikation der einzelnen Software-Komponenten und der BSW-Module untereinander unterstützt der AUTOSAR-Standard zwei verschiedene Kommunikationsmechanismen. Beide erfordern die Definition von *Ports* innerhalb der SWCD, welche wiederum eine *Interface*-Beschreibung benötigen. Das *Interface* definiert die Datenelemente, welche über die Schnittstelle übertragen werden.

Sender-Receiver-Kommunikation

Die *Sender-Receiver-Kommunikation* ermöglicht die direkte Übertragung von Datenelementen von einem Sender (*Provider*) zu einem oder mehreren Empfängern (*Receiver*). Dabei bestimmt der Sender, zu welchem Zeitpunkt die Datenübertragung gestartet wird. Die Datenelemente werden in der RTE zwischengespeichert und können von dem Empfänger jederzeit abgerufen werden. Durch die Definition von *Data-Receive-Points* und *Data-Send-Points* im *Internal-Behavior* wird festgelegt, welches *Runnable* auf einen Port der SWC lesend oder schreibend zugreifen darf. Abbildung 2.4 zeigt die zugehörige symbolische Darstellung.

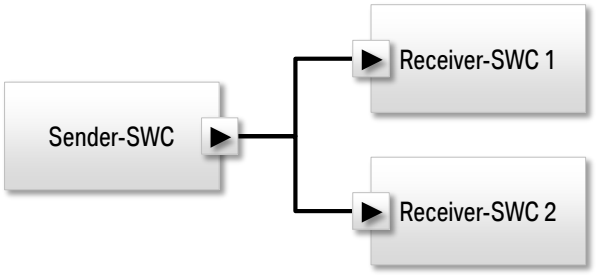


Abbildung 2.4: Sender-Receiver-Kommunikation mit 1:n-Verknüpfung [AUT08].

Client-Server-Kommunikation

Bei der *Client-Server-Kommunikation* existiert immer ein *Server*, der einen *Service* für einen oder mehrere *Clients* zur Verfügung stellt. Hier geht die Kommunikation vom *Client* aus, welcher eine Anfrage an den *Server* sendet und damit die Ausführung des *Services* initiiert. Über die Anfrage können Argumente übertragen werden, welche der *Server* verarbeitet. Nach erfolgreicher Abarbeitung der Anfrage stellt der *Server* dem *Client* seine Antwort zur Verfügung. Dabei kann zwischen synchroner und asynchroner Verarbeitung unterschieden werden. Bei synchroner Kommunikation wartet der *Client* nach der Anfrage, bis er die Antwort des *Servers* erhält. Bis dahin ist der *Client* blockiert und kann seine Ausführung nicht fortsetzen. Im asynchronen Fall wird der Task des *Client* nach der Anfrage nicht blockiert. Nach der Abarbeitung im *Server* wird der *Client* über einen *Callback* aufgerufen und erhält die Ergebnisse des *Service*. Damit ein *Runnable* die Berechtigung für den Zugriff auf einen *Client-Port* erhält, muss im *Internal-Behavior* ein *Server-Call-Point* definiert werden. Die symbolische Darstellung der *Client-Server-Kommunikation* ist in Abbildung 2.5 zu sehen [AUT08].

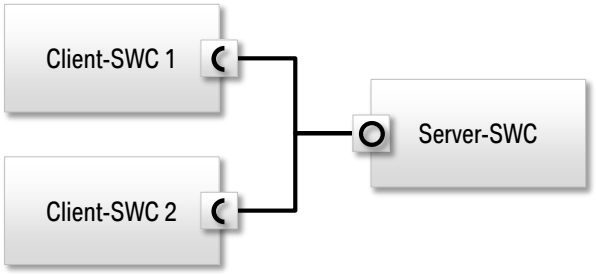


Abbildung 2.5: Client-Server-Kommunikation mit n:1-Verknüpfung [AUT08].

2.1.5 Fazit

Insgesamt bietet der AUTOSAR-Standard alle nötigen Mechanismen zur Beschreibung von aktueller Steuergeräte-Software. Die durchgängige Abstraktion von Hardware und Funktions-Software, sowie die einheitliche Beschreibung von Schnittstellen, schaffen die Voraussetzungen für eine einfache Wiederverwendbarkeit der Software-Komponenten. Durch die kontinuier-

liche Weiterentwicklung des Standards ist die Integration von zukünftigen Anforderungen sichergestellt.

2.2 Complex-Device-Drivers

Hardwareabhängige Funktionen werden im AUTOSAR-Standard als Complex-Device-Driver (CDD) bezeichnet. Der folgende Abschnitt beschreibt die zugehörigen Grundlagen und die Einsatzgebiete der CDDs bei Steuergeräten der BMW Group.

2.2.1 Grundlagen

Im AUTOSAR-Standard wurde das Konzept der sogenannten Complex-Device-Drivers definiert, um die Möglichkeit zur Integration von nicht standardisierten Modulen zu bieten. Ein CDD ist eine Software-Komponente mit erweiterten Zugriffsmöglichkeiten auf andere Software-Anteile und die Steuergeräte-Hardware. Wie in der AUTOSAR-Schichtenarchitektur in Abbildung 2.3 dargestellt, liegen die CDDs parallel zu den Schichten der Basis-Software und ermöglichen damit den direkten Zugriff auf den Mikrocontroller und die angeschlossene Peripherie. In der Implementierung eines CDD können beliebige Funktionen umgesetzt werden. Lediglich die Schnittstellen zur RTE, welche in der Architektur oberhalb des CDD liegt, müssen konform zum AUTOSAR-Standard realisiert sein. Zugriffe auf die Basis-Software sind ebenfalls möglich, allerdings nur auf bestimmte Module, über deren definierte Schnittstellen. Tabelle 2.1 zeigt eine Übersicht aller möglichen Modulzugriffe.

BSW-Modul	**Zugriffsmöglichkeit über CDD**
Serial-Peripheral-Interface (SPI)	Direkter Zugriff auf Treiber gestattet.
General-Purpose-Timer (GPT)	Direkter Zugriff auf Treiber gestattet.
I/O-Treiber	Mit Einschränkung: Paralleler Zugriff auf dieselben Gruppen/Channel/etc. normalerweise nicht erlaubt.
Non-Volatile-Read-Only-Memory-Manager (NVM)	Exklusiver Zugriffspunkt für den Speicher-Stack.
Watchdog-Manager (WDGM)	Exklusiver Zugriffspunkt für den Watchdog-Stack.
Protocol-Data-Unit-Router (PDUR)	Exklusiver Zugriffspunkt für den Kommunikations-Stack, unabhängig von Bus oder Protokoll.
Bus-Interface-Module	Exklusiver Zugriffspunkt für den Kommunikations-Stack, für busspezifische Zugriffe.
Network-Management-Interface (NM)	Exklusiver Zugriffspunkt für den Netzwerk-Management-Stack.
ECU-State-Manager (ECUM)	Exklusiver Zugriffspunkt für das Zustandsmanagement.
Development-Error-Tracer (DEM), Diagnostic-Event-Manager (DET), Diagnostic-Log-and-Trace (DLT)	Direkter Zugriff gestattet.
Operating-System (OS)	Nur soweit die OS-Objekte nicht von anderen BSW-Modulen genutzt werden.

Tabelle 2.1: Zugriffsmöglichkeiten von CDDs zu BSW-Modulen [AUT11b].

Die Schnittstellen eines CDD zum Mikrocontroller können beliebig auf die benötigte Hardware zugeschnitten werden und sind im AUTOSAR-Standard nicht weiter definiert. Grundsätzlich können jedoch vier Arten von Hardwareschnittstellen unterschieden werden:

- **Registerzugriffe:** Üblicherweise werden Register durch feste Adressen im Adressraum der Mikrocontroller-Architektur definiert. Darüber kann ein CDD direkt schreibend oder lesend auf ein Hardwareregister zugreifen. Bei Änderung des Wertes im Register werden die verknüpften Hardwareoperationen ausgelöst. So kann z. B. über einen CDD der Start einer Analog-Digital-Wandlung gesteuert werden (siehe Quellcodebeispiel 2.1). Das Ergebnis der Berechnung wird wiederum in einem anderen Register abgelegt und kann durch den Treiber ausgelesen werden.

```
1 //Zuweisung der Registeradresse per Makro (Alternativ: Pointer)
2 #define ADC1_CONV *((volatile bool *)0xADC10000);
3
4 //Start der Analog-Digital-Wandlung
5 ADC1_CONV = true;
```

Quellcodebeispiel 2.1: Direkter Zugriff auf ein Register.

- **Speicherzugriffe:** Direct-Memory-Access (DMA) ermöglicht der Steuergeräte-Peripherie einen unmittelbaren Zugriff auf den Arbeitsspeicher, ohne Beteiligung des Mikrocontrollers [SZ13]. So können eingelesene Werte effizient abgelegt werden, ohne andere Prozesse zu unterbrechen. Ein CDD kann z. B. durch zyklische Auswertung des Speicherbereichs auf die Daten zugreifen, um sie weiterzuverarbeiten. In der Umsetzung ist der Vorgang sehr ähnlich zu einem Registerzugriff. Quellcodebeispiel 2.2 stellt einen direkten Speicherzugriff dar.

```
1 //Zuweisung der Speicheradresse per Pointer (Alternativ: Makro)
2 volatile int *adcValuePtr;
3 adcValuePtr = (volatile int *)0xADCD11A1;
4
5 //Kopieren der Ergebnisse der Analog-Digital-Wandlung
6 int adcValue;
7 adcValue = *adcValuePtr;
```

Quellcodebeispiel 2.2: Direkter Zugriff auf einen Speicherbereich.

- **Funktionsaufrufe:** Ein indirekter Hardwarezugriff ist über hardwarenahe Application-Programming-Interfaces (APIs) möglich, die z. B. durch einen Low-Level-Treiber der zugehörigen Hardware bereitgestellt werden. Dabei handelt es sich um Funktionsaufrufe, welche die Registeroperationen kapseln und abstrahieren. Somit bieten sie eine einfachere Schnittstelle für den Programmierer, als den direkten Registerzugriff. Eine solche Low-Level-Funktion ist im Quellcodebeispiel 2.3 abgebildet. Der zugehörige Aufruf der Funktion innerhalb des CDD ist im Quellcodebeispiel 2.4 dargestellt.

```
1  //Deklaration der Funktion für das Auslesen einer AD-Wandlung
2  int Adc_Read (int channel){
3          int retval; //Deklaration des Rückgabewertes
4          switch(channel) { //Zuweisung der Registerwerte
5                  case 0: retval = *(0xADCE0000); break;
6                  case 1: retval = *(0xADCE0020); break;
7                  case 2: retval = *(0xADCE0040); break;
8          }
9          return retval; //Rückgabe des Wertes für einen Channel
10 }
```

Quellcodebeispiel 2.3: Implementierung einer Low-Level-Funktion.

```
1  //Deklaration der Variablen
2  int adcChannel = 1; //Einzulesender Channel
3  int adcValue; //Einzulesender ADC-Wert
4
5  //Abruf des Wertes der AD-Wandlung über die Low-Level-Funktion
6  adcValue = Adc_Read(adcChannel);
```

Quellcodebeispiel 2.4: Aufruf einer Low-Level-Funktion in einem CDD.

- **Interrupts:** Durch sogenannte Interrupts kann die Peripherie einen Mikrocontroller anweisen, den aktuellen Prozess zu unterbrechen und unmittelbar eine zugeordnete Interrupt-Service-Routine (ISR) auszuführen [SZ13]. Das Interrupt-Handling in AUTOSAR sieht vor, dass die ISR innerhalb des jeweiligen Gerätetreibers definiert sein muss [AUT09]. Ist die Peripherie über einen CDD angebunden, so muss dieser ebenso die ISR enthalten und den Interrupt abarbeiten. Die Umsetzung ist analog zu einer Low-Level-Funktion (siehe Quellcodebeispiel 2.3) mit dem Unterschied, dass der Aufruf von der Peripherie getriggert wird, anstatt von der Software.

2.2.2 Einsatzgebiete

Im Rahmen dieser Arbeit wurde eine Recherche durchgeführt, für welche Anwendungsfälle CDDs in der Praxis der Steuergeräte-Entwicklung eingesetzt werden. Dafür wurden verschiedene AUTOSAR-Steuergeräte aus den Fahrzeugen der BMW Group untersucht. Die aufgedeckten Anwendungsfälle lassen sich in vier Cluster aufteilen, welche im Folgenden beschrieben werden. Die Gliederung deckt sich dabei zu großen Teilen mit der Beschreibung aus der AUTOSAR-Spezifikation [AUT08]. In Tabelle 2.2 sind die Anwendungscluster mit einigen zugehörigen Beispielen zusammengefasst.

- **Einsatz nicht unterstützter Hardware:** Der AUTOSAR-Standard definiert eine Vielzahl von Hardware-Schnittstellen. Allerdings ist der Standardisierungsprozess grundsätzlich zeitaufwändig und würde bei neuen Technologien den Einsatz in AUTOSAR-Steuergeräten blockieren bis das Standardmodul fertiggestellt ist. Weiterhin wird spezielle Hardware oft nur für Nischenanwendungen genutzt, für die der Aufwand der Standardisierung nicht wirtschaftlich ist. Für diese Anwendungsfälle kann die spezifische Hardware über einen CDD angesteuert werden, wodurch die Integration in ein AUTOSAR-Steuergerät erst möglich wird. Ein klassisches Beispiel ist der MOST, für welchen bis AUTOSAR-Release 4.2 kein Modul definiert ist, weshalb er als CDD mit eingebunden werden muss.

Anwendungscluster	**Beispiele**
Einsatz nicht unterstützter Hardware	Media-Oriented-System-Transfer-Bus (MOST), Automotive-Pixel-Link (APIX), Inter-Process-Communication (IPC), prozessorspezifische Funktionen z. B. für Infineon TriCore (PCP, GPTA, MLI).
Zeitkritische Funktion	Regelung für Elektromotoren (<100 μs-Raster), Ventilsteuerung für Verbrennungsmotoren, AD-Wandler mit besonderen Zeitanforderungen.
Funktionale Sicherheit	Überprüfung sicherheitskritischer Signale (CRC, Alive-Timer), Umsetzung von Notlauf-Funktionen, Speicherüberwachung.
Migration	Übergangsweise für spezifische BSW-Module der BMW Group.

Tabelle 2.2: Anwendungscluster von CDDs mit zugehörigen Beispielen.

- **Zeitkritische Funktionen:** Die Module des AUTOSAR-Standards sind generisch ausgelegt, um ein breites Spektrum von Hardware zu unterstützen. Darüber sind Hardwarekomponenten nicht abdeckbar, die in einem sehr engen Zeitraster angesteuert werden oder sehr harte Echtzeitbedingungen einhalten müssen. Diese Anwendungen erfordern, dass der Treiber sehr spezifisch auf die Hardware angepasst wird, um das Optimum an Geschwindigkeit zu erreichen und die Möglichkeiten der Hardware vollständig auszunutzen. Im AUTOSAR-OS lässt sich sogar bei vielen Stacks keine Zykluszeit unter einer Millisekunde definieren, was für sehr zeitkritische Ansteuerungen nicht ausreichend ist. Beispiele für solche Funktionen sind die Regelung von Antriebselektromaschinen oder die Ventilsteuerung bei Verbrennungsmotoren.

- **Funktionale Sicherheit:** CDDs spielen ebenso eine große Rolle bei Aspekten der funktionalen Sicherheit. Sie werden genutzt, um für Signale der Bussysteme Cyclic-Redundancy-Checks (CRCs) durchzuführen und Alive-Timer zu verifizieren. Damit wird geprüft, ob die Signale korrekt zwischen den Steuergeräten übertragen wurden und die Daten fehlerfrei sind. Weiterhin ist es über CDDs möglich, Speicherbereiche zu überwachen oder eine effiziente Spannungsüberwachnung zu betreiben. Für sicherheitskritische Anwendungen lassen sich über CDDs redundante Signalpfade aufbauen oder Notlauffunktionen realisieren. Die Funktionalität der Anwendung kann dadurch sichergestellt werden, um die Anforderungen eines Automotive-Safety-Integrity-Level (ASIL) nach [ISO11] zu erfüllen.

- **Migration:** Muss Software in ein AUTOSAR-Steuergerät eingebunden werden, welche nicht nach dem Software-Standard entwickelt wurde (sog. Legacy-Software), können ebenfalls CDDs genutzt werden. So besteht die Möglichkeit, bereits vorhandene Funktionen in ein neues Steuergerät zu integrieren, ohne die Software vollständig neu entwickeln zu müssen. Dies wurde z. B. bis zum AUTOSAR-Release 3.1 genutzt, um einige der OEM-spezifischen Anteile der BSW in Steuergeräten der BMW Group einzubinden. Seit dem AUTOSAR-Release 4.0 sind die Komponenten allerdings zum größten Teil als SWCs umgesetzt.

Im Rahmen dieser Arbeit wurde ebenso die Verteilung der verschiedenen Anwendungscluster von CDDs in Steuergeräten bei der BMW Group untersucht. In zehn aktuellen AUTOSAR-Steuergeräten konnten insgesamt 44 CDDs identifiziert werden. Davon sind 13 Treiber Teil der BMW-spezifischen AUTOSAR-Beistellungen und liegen in jedem Steuergerät vor. Um diese CDDs nicht übermäßig hervorzuheben, wurden sie nur einmalig gezählt und nicht für jedes Steuergerät wiederholt berücksichtigt.

Es hat sich gezeigt, dass speziell CDDs aus dem Cluster „Einsatz nicht unterstützter Hardware“ über 50 % der Einsatzfälle ausmachen. CDDs für zeitkritische Funktionen treten am seltensten auf, da sie normalerweise nur für sehr spezielle Anwendungen benötigt werden, wie z. B. in Antriebssteuergeräten. Die Übersicht erhebt keinen Anspruch auf statistische Signifikanz. Sie zeigt dennoch, dass alle Anwendungscluster in der Steuergeräte-Software relevant sind und daher in dieser Arbeit entsprechend berücksichtigt werden müssen. Abbildung 2.6 zeigt die prozentuale Verteilung der einzelnen Cluster über die Menge der untersuchten Steuergeräte bei der BMW Group.

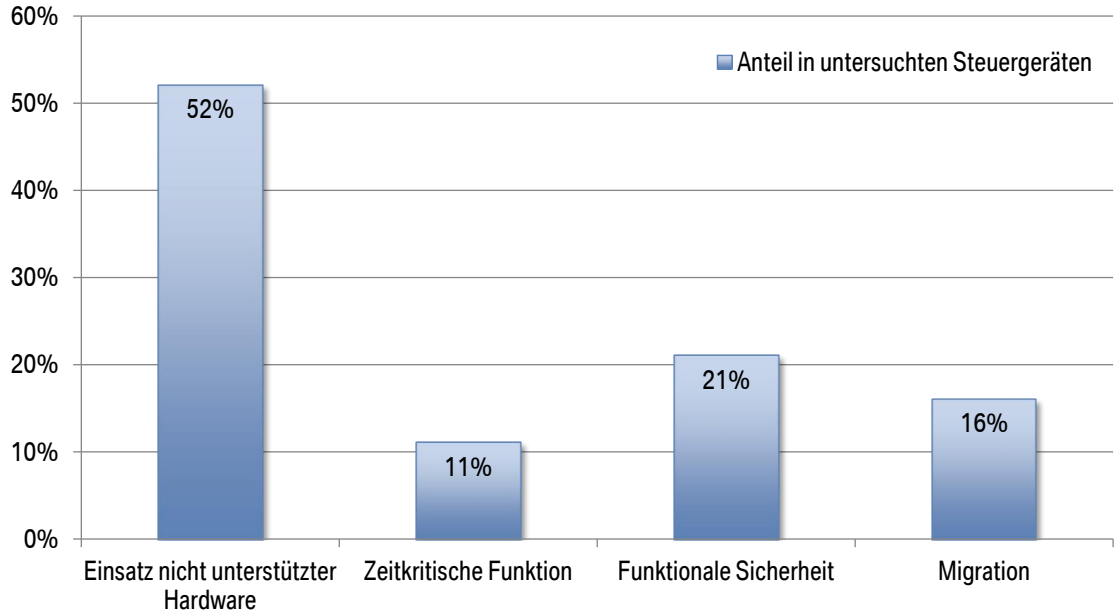


Abbildung 2.6: Prozentuale Verteilung der Anwendungscluster von CDDs über die untersuchten Steuergeräte bei der BMW Group.

Für die virtuelle Absicherung (siehe Kapitel 2.4) ist besonders relevant, ob ein CDD die Hardware eines Steuergerätes direkt ansteuert. Dies ist nicht grundsätzlich der Fall, da oftmals nur die obigen Möglichkeiten für den Zugriff auf anderen BSW-Module genutzt werden. Solche CDDs sind für die virtuelle Absicherung unproblematisch, da sie einfach auf Systemen wie einer VAP integrierbar sind. Im Rahmen dieser Arbeit wurde ermittelt, dass 59 % der CDDs in den untersuchten Steuergeräten der BMW Group über eine direkte Hardware-Ansteuerung verfügen (siehe Abbildung 2.7). Solche Complex-Device-Drivers stehen in der weiteren Ausarbeitung im Fokus, da für sie bisher keine Möglichkeiten zur Berücksichtigung in der virtuellen Absicherung verfügbar waren.

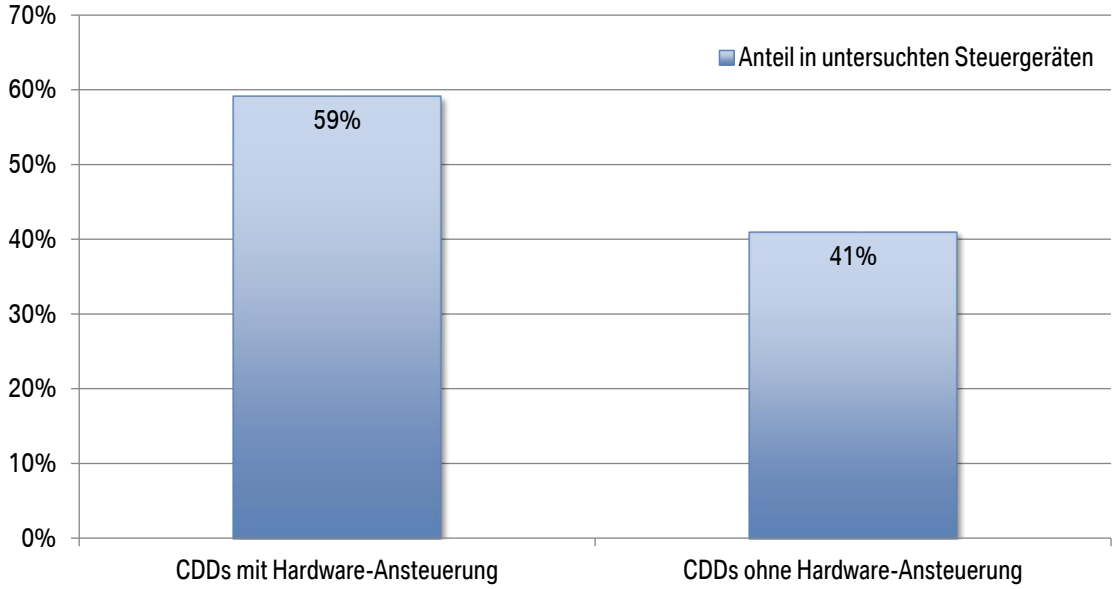


Abbildung 2.7: Prozentuale Verteilung der CDDs mit direkter Hardware-Ansteuerung über die untersuchten Steuergeräte bei der BMW Group.

2.2.3 Fazit

Complex-Device-Drivers (CDDs) bieten vielfältige Möglichkeiten zur Umsetzung von spezifischen Anforderungen in AUTOSAR-Steuergeräten. Verschiedene Varianten für einen direkten Hardware-Zugriff erlauben den Einsatz von nicht im Standard unterstützer Hardware, die Optimierung für zeitkritische Funktionen, oder die Realisierung von Aspekten der funktionalen Sicherheit. Die erweiterten Zugriffsrechte auf die AUTOSAR-Basis-Software bieten weitere Einsatzmöglichkeiten für CDDs. Diese zahlreichen Optionen haben bereits zu einer hohen Verbreitung der Treiber in der Steuergeräte-Software geführt. Trotz der stetigen Weiterentwicklung des AUTOSAR-Standards werden CDDs auch zukünftig eine wichtige Rolle spielen und müssen in der virtuellen Absicherung berücksichtigt werden.

2.3 Entwicklung und Test von Software

Die Prozesse für Entwicklung und Test von Steuergeräte-Software erfolgen in der gesamten Automobilindustrie nach ähnlichen, standardisierten Methoden. Dieses Kapitel soll einen kurzen Überblick über die Aspekte liefern, die für diese Arbeit relevant sind.

2.3.1 Entwicklungsprozess

Die Entwicklung läuft in der Automobilindustrie allgemein nach dem standardisierten V-Modell ab, welches ebenfalls für Steuergeräte und ihre Software Anwendung findet. Das Prinzip beginnt mit einem Top-Down-Ansatz, wobei erst die Anforderungen an das Gesamtsystem bestimmt und festgelegt werden. Es folgt die Spezifikation des Systems und darauf der System-Entwurf. Im Feinentwurf wird das System immer weiter unterteilt, bis die kleinsten Komponenten überschaubar genug für eine Realisierung sind. Damit ist der linke Ast des V-

Modells durchlaufen und das System und seine einzelnen Komponenten detailliert spezifiziert. Der rechte Ast befasst sich mit der Realisierung und Qualitätsprüfung. Jede Komponente wird für sich umgesetzt und über Unit-Tests abgesichert. Ist dies erfolgreich, können alle Komponenten Stück für Stück zu Teilsystemen kombiniert, und schließlich zum Gesamtsystem integriert werden. Abschließend erfolgt die Lieferung und die Abnahme des Systems beim Auftraggeber. Der vollständige Prozess wird in Abbildung 2.8 dargestellt [VMo12].

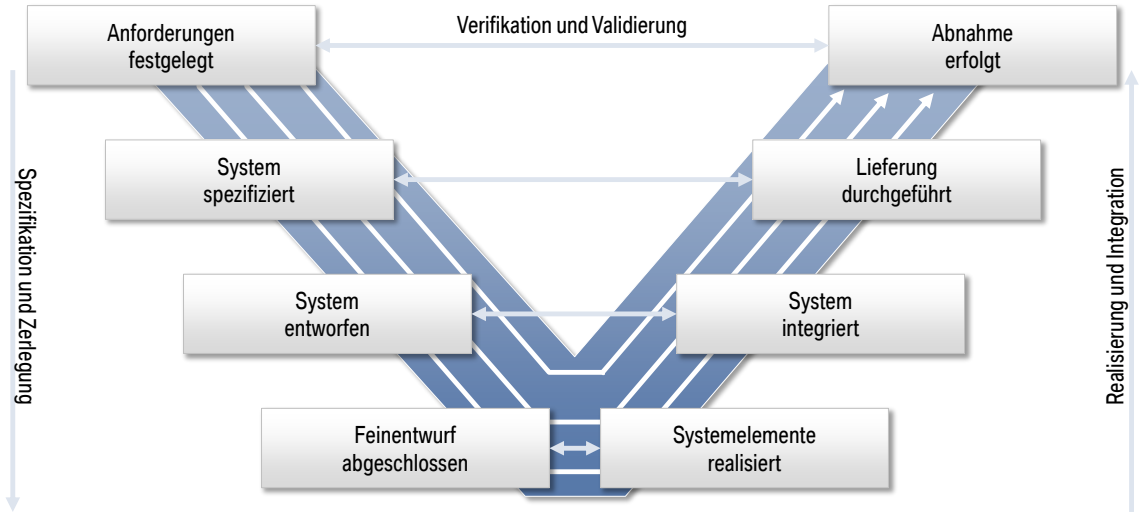


Abbildung 2.8: Das allgemeine V-Modell der Systementwicklung [VMo12].

2.3.2 Qualitätssicherung von Software

Allgemein definiert [ISO05] *Qualitätsmanagement* als „aufeinander abgestimmte Tätigkeiten zum Leiten und Lenken einer Organisation bezüglich Qualität“. Die *Qualitätssicherung* wird in [ISO05] analog definiert als „Teil des Qualitätsmanagements, der auf das Erzeugen von Vertrauen darauf gerichtet ist, dass Qualitätsanforderungen erfüllt werden“. Dies gilt genauso bei Software in der Automobilindustrie. Im rechten Ast des V-Modells (siehe Abbildung 2.8) sind verschiedene Methoden zur Qualitätssicherung verankert, die zur Identifikation und Beseitigung von Fehlern in der Software dienen.

Bei Fehlern lassen sich Spezifikations- und Implementierungsfehler unterscheiden. Für Testmethoden differenziert das V-Modell daher explizit zwischen Validierung und Verifikation. Die Validierung klärt die Frage, ob das richtige Produkt entwickelt wurde. Sie umfasst somit Methoden wie formale Spezifikation, Modellierung, Simulation und Rapid-Prototyping, die zur Aufdeckung von Spezifikationsfehlern dienen. Die Verifikation klärt die Frage, ob das Produkt richtig entwickelt wurde. Es handelt sich somit um die Aufdeckung von Implementierungsfehlern durch Methoden wie Code-Inspektion, Peer-Review, statische Analyse und Black-/White-Box-Tests. Eine Übersicht verschiedener Methoden zur Qualitätssicherung ist in Tabelle 2.3 dargestellt [SZ13, MIS94].

Verifikation	**Validierung**
Statische Techniken **Review** Walkthrough, Fagan-Inspektion, Code-Inspektion, Peer-Review etc. **Analyse** Statische Analyse, Formale Prüfung, Kontroll- und Datenfluss etc.	**Animation** Formale Spezifikation, Modellierung, Simulation, Rapid Prototyping etc.
Dynamischer Komponenten-/ Integrationstest **Black-Box-Test** Funktionale Leistungsfähigkeit, Stress, Grenzwert, Fehlererwartung etc. **White-Box-Test** Struktur, Pfad, Zweig, Bedingung, Abdeckung etc.	**Systemtest/Akzeptanztest** Funktionale Leistungsfähigkeit, Stresstests, Grenzwerttests, Fehlererwartungstests, Ursache-Wirkungs-Graph, Äquivalenzklassentests etc.

Tabelle 2.3: Maßnahmen zur Qualitätssicherung von Software [SZ13, MIS94].

Simulationsmethoden

Der Begriff der Simulation aus Tabelle 2.3 umfasst laut [SZ13] explizit die Nachbildung von Software-Funktionen als auch deren Umgebung in Form von Hardware, Sensoren, Aktuatoren und Regelstrecke. Weiterhin ist für die Durchführung von dynamischen Komponenten- und Integrationstests (Black- und White-Box-Tests, siehe Tabelle 2.3) ebenfalls die Erzeugung einer Umgebungssimulation nötig, um das Testobjekt mit plausiblen Eingangswerten zu versorgen und die resultierenden Ausgaben zu überprüfen. Für die Simulation von Testobjekt und Umgebung haben sich unterschiedliche Methoden entwickelt, die unter dem Begriff *X-in-the-Loop (XiL)* zusammengefasst werden. Alle Varianten beinhalten den Test einer spezifischen Komponente, genannt *System-under-Test (SuT)*, in einem simulierten, geschlossenen Regelkreis, der die Umgebung des SuT möglichst realitätsnah abbilden soll. Es können verschiedene Eingangswerte für die Komponente vorgegeben werden, vom Normalbereich bis zu relevanten Grenzwerten. Die jeweils resultierenden Ausgaben des SuT lassen sich dabei verifizieren. Ebenso kann das resultierende Verhalten der simulierten Umgebung untersucht werden. Die verschiedenen Varianten sind jeweils an aufeinanderfolgenden Stellen im Entwicklungsprozess angesiedelt. Sie unterscheiden sich jeweils durch die Form des SuT, welches von der Modellbeschreibung bis zum Zielsystem weiterentwickelt wird. Die Methoden sollen im Folgenden analog zu [SZ13], [WR06] und [AB06] kurz beschrieben werden.

- **Model-in-the-Loop (MiL):** Da Software heutzutage zu großen Teilen modellbasiert entwickelt wird, ist das Modell einer Software-Komponente die erste testbare Version. Als SuT dient bei der MiL-Methode somit selbiges Modell, welches in einer Simulationsumgebung ausgeführt und unter verschiedenen Bedingungen intensiv geprüft wird. Dies erfolgt üblicherweise direkt in den Modellierungswerkzeugen, wie z. B. MATLAB/Simulink [Mat10], ASCET [ETA14] oder Dymola [Das14b]. Die Werkzeuge übernehmen dabei sowohl die Simulation des SuT als auch die Umgebungssimulation.

- **Software-in-the-Loop (SiL):** Wird aus dem Modell des Systems Source-Code erzeugt oder Code von Hand geschrieben, so kann dieser über die SiL-Methode abgesichert werden. Der Code wird kompiliert und in Teilen oder vollständig ausgeführt und über eine Simulation mit plausiblen Eingangswerten versorgt. Dieser Vorgang erfolgt allerdings nicht auf der Ziel-Hardware, sondern auf einem Entwickler-PC. Viele MiL-Werkzeuge unterstützen ebenfalls SiL, wobei MATLAB/Simulink wiederum weit verbreitet ist. Im AUTOSAR-Umfeld wird z. B. auch ASIM [Das14a] eingesetzt.

- **Processor-in-the-Loop (PiL):** Bei der PiL-Methode wird ein Teil der eigentlichen Hardware des Steuergerätes eingesetzt, da der Prozessor der Ziel-Architektur als SuT betrachtet wird. Die sonstige Steuergeräte-Peripherie sowie Signalanpassungsbestandteile werden dabei normalerweise nicht mit betrachtet. Die entwickelte Software wird auf dem Prozessor ausgeführt, wobei Ressourcenverbrauch und Ausführungszeiten realitätsnah untersucht werden können.

- **Hardware-in-the-Loop (HiL):** Sobald frühe Hardware-Muster eines Steuergerätes zur Verfügung stehen, kann die HiL-Methode eingesetzt werden. Als SuT wird dabei mindestens ein Steuergerät betrachtet. Normalerweise werden Teilsysteme aus mehreren Steuergeräten untersucht, bis hin zum Gesamtsystem aller Steuergeräte. Die Simulation der Umgebung des SuT erfolgt dabei in Echtzeit auf sehr leistungsstarken Rechensystemen mit spezialisierten Modulen für z. B. besonders zeitkritische Anwendungen. Für

die Absicherung des Gesamtsystems werden z. B. der Motor inklusive Antriebsstrang sowie das Fahrwerkverhalten und die Sensorik zur Erfassung der Fahrzeugumgebung simuliert. Dadurch kann eine Testfahrt mit allen Elektrik- und Elektronik-Komponenten des Fahrzeugs im Labor realisiert werden.

Rapid-Control-Prototyping

Eine weitere Methode in der Qualitätssicherung, die hier kurz behandelt werden soll, ist das sogenannte Rapid-Control-Prototyping (RCP). Alternativ wird auch die Bezeichnung Rapid-Prototyping verwendet. Allgemein wird dabei ein Software-Prototyp einer Funktion in der Entwicklungsphase unter realen oder realitätsnahen Bedingungen erprobt. Als Ausführungsplattform werden performante Echtzeitsysteme verwendet, die in ihrer Architektur weit vom Zielsteuergerät entfernt sind. Durch entsprechende Hardware-Schnittstellen der Systeme ist die Erprobung dennoch in realen Fahrzeugen möglich, mit der direkten Anbindung von Sensoren und Aktuatoren. Weiterhin existieren speziell modifizierte Steuergeräte für die Entwicklung, in welche man über einen sogenannten *Bypass* zusätzliche Funktionen integrieren kann. Die neue Funktion wird auf einem weiteren Experimentiersystem ausgeführt und über das reale Steuergerät an das Gesamtfahrzeug angebunden. Die Software kann dafür in verschiedenen Detaillierungen vorliegen, vom groben Modell bis zur hardwarenahen Implementierung. Neue Funktionen können somit früh in der Entwicklung erprobt werden und die Zyklen, um die Funktion auf eine Hardwareplattform bis ins Fahrzeug zu bringen, werden deutlich verkürzt [SZ13]. Eine sehr ausführliche Beschreibung des RCP ist in [AB06] zu finden. Weiterhin wird in [Har96, HR97] ein umfassender Überblick über Rapid-Prototyping im Zusammenhang mit Hardware-/Software-Co-Design gegeben. Für weitere Ausführungen zu Qualitätssicherungsmethoden in der Software-Entwicklung soll auf [SZ13], [BH09] und [WR06] verwiesen werden.

Ausprägungen von Software-Prototypen

Prototypische Realisierungen von Software können in zwei Kategorien unterteilt werden. Man spricht von einem *vertikalen Prototypen*, wenn die Software für eine Teilfunktion detailliert realisiert ist und z. B. alle Ebenen der Architektur, bis zur Hardwareansteuerung, mit umgesetzt sind. Ein *horizontaler Prototyp* beinhaltet dagegen hauptsächlich grobe Realisierungen (z. B. als Modell) der Funktionslogik in der obersten Softwareebene. Eine Hardware-Anbindung kann dabei über *Bypassing* realisiert werden, wobei die Funktion in ein vorhandenes Steuergerät mit eingegliedert wird [SZ13]. Für MiL, SiL und RCP werden meist *horizontale Prototypen* eingesetzt. Bei den Methoden PiL und HiL sind *vertikale Prototypen* eher verbreitet. Abbildung 2.9 stellt beide Varianten im Vergleich zur vollständigen Software-Architektur eines Steuergerätes dar.

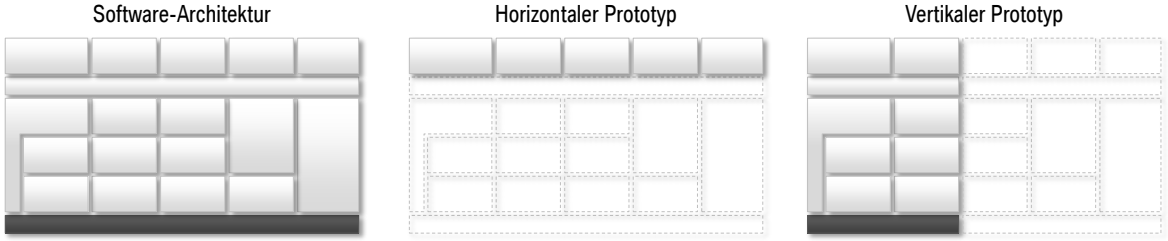


Abbildung 2.9: Unterscheidung horizontaler und vertikaler Prototypen von Steuergeräte-Software [SZ13].

2.3.3 Fazit

Im Entwicklungsprozess nach dem V-Modell ist die Qualitätssicherung von Software fest verankert. Verschiedene Simulationsmethoden, die unter dem Begriff X-in-the-Loop (XiL) zusammengefasst sind, werden bereits zur Unterstützung der Verifikation und Validierung eingesetzt. Eine weitere wichtige Rolle spielt das Rapid-Control-Prototyping. Zusätzlich etabliert sich die virtuelle Absicherung, die im folgenden Kapitel definiert und in Relation zu den obigen Methoden gesetzt wird.

2.4 Definition der virtuellen Absicherung

Der Begriff virtuelle Absicherung wird in verschiedenen Bereichen mit unterschiedlichen Bedeutungen und für stark abweichende Methoden genutzt. Daher ist es notwendig, die Methode für diese Arbeit zu definieren und die Differenzierung gegenüber angrenzenden Bereichen hervorzuheben.

2.4.1 Begriffsdefinition

Im Kontext dieser Dissertation wird der Begriff der virtuellen Absicherung, im Bezug auf Steuergeräte-Software im Automobilbereich, folgendermaßen definiert:

Virtuelle Absicherung ist die Integration und Qualitätssicherung von Steuergeräte-Software aus der Serienentwicklung auf einem generischen System, unabhängig von der Ziel-Hardware des Steuergerätes, mit dem Ziel, die Wiederverwendbarkeit der Software sicherzustellen und Fehler frühestmöglich im Entwicklungsprozess aufzudecken.

Das heißt, die Steuergeräte-Software für die Serienproduktion wird z. B. auf einem PC-System mit x86-Architektur integriert und anhand der zugehörigen Testfälle verifiziert. Dadurch wird die Ziel-Hardware in frühen Entwicklungsphasen substituiert und es entsteht ein virtuelles Steuergerät auf einer hardwareunabhängigen Plattform. Somit kann sichergestellt werden, dass die Software frei von Hardwareabhängigkeiten ist und leicht von einem Mikrocontroller zu einem anderen portiert werden kann. Es handelt sich nicht um eine Simulation der Software, sondern um eine Abstraktion der realen Serien-Software von ihrer Ziel-Hardware. Die Methode ist vergleichbar mit einer virtuellen Maschine, genauer einer *System-Virtual-Machine*, wie in [SN05] definiert. Das Basissystem stellt eine Umgebung bereit, in der ein oder mehrere Gastbetriebssysteme auf derselben Hardware parallel betrieben werden

können. Bei den Gastbetriebssystemen handelt es sich im Fall der Steuergeräte-Software um das Betriebssystem des Steuergerätes selbst (z. B. das AUTOSAR-Betriebssystem), wie in Abbildung 2.10 dargestellt.

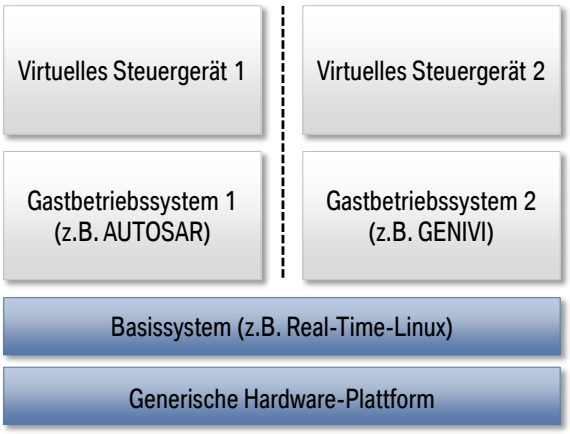


Abbildung 2.10: Generischer Aufbau eines Systems zur virtuellen Absicherung von Steuergeräte-Software.

2.4.2 Abgrenzung zu gängigen Absicherungsmethoden

Im Folgenden werden die Unterschiede der virtuellen Absicherung nach der Definition aus Kapitel 2.4.1 zu einigen der gängigsten Absicherungsmethoden dargestellt, welche in Kapitel 2.3.2 bereits beschrieben wurden. Dies soll zur Abgrenzung des Ansatzes dienen und die Differenzierung der Methoden klarstellen.

Abgrenzung zu X-in-the-Loop-Methoden

Zu manchen der verschiedenen XiL-Methoden bietet die virtuelle Absicherung Parallelen, die Abgrenzung ist jedoch jeweils deutlich zu identifizieren. In der virtuellen Absicherung wird die reale Serien-Software eingesetzt, wie sie unverändert im Steuergerät genutzt wird. Dadurch entsteht ein klarer Unterschied zum Model-in-the-Loop (MiL), da hierbei die Modelle der Software im Testfokus liegen. Üblicherweise wird aus den Modellen unmittelbar der Steuergeräte-Code erzeugt. Allerdings können bei der Codegenerierung, gerade im Bezug auf Laufzeit und Timing, deutliche Unterschiede entstehen. Weiterhin liegt oft nicht die gesamte Steuergeräte-Software im Modell vor, da wichtige Teile, wie z. B. die AUTOSAR-Basis-Software, nicht modellbasiert entwickelt werden. Im Gegensatz zur virtuellen Absicherung wird das Modell bei MiL-Tests nicht in Echtzeit ausgeführt.
Die meisten Gemeinsamkeiten bietet der SiL-Ansatz, da die Software dabei bereits im Source-Code vorliegt und auf einem PC-System ausgeführt wird. Dennoch handelt es sich bei SiL um eine Simulation, die nicht über hardwarenahe Schnittstellen verfügt, wie sie ein Steuergerät bieten kann. Ebenso spielen Echtzeitkriterien dabei keine Rolle. In der virtuellen Absicherung dagegen stehen die realen Hardwareschnittstellen zur Verfügung und die Ausführung der Software muss in Echtzeit möglich sein, um z. B. Peripherie oder Bussysteme anzusteuern.
Bei der PiL-Methode erfolgt die Ausführung der Software bereits auf dem Ziel-Mikrocontroller. Damit ist die Methode im Entwicklungsprozess nach der virtuellen Absicherung angesiedelt,

wenn die Ziel-Hardware betrachtet werden soll. Besonders der Aspekt der Wiederverwendbarkeit der Software kann dabei nicht mehr untersucht werden, da bereits der Mikrocontroller der Ziel-Plattform verwendet wird.
HiL-Systeme werden zur Umgebungssimulation für reale Steuergeräte betrieben. In der virtuellen Absicherung auf einer universellen Hardware-Plattform stehen nach außen jedoch dieselben Schnittstellen zur Verfügung wie bei der Ziel-Hardware. Daher steht das Konzept nicht im Gegensatz zur HiL-Methode. Vielmehr können HiL-Systeme parallel zur virtuellen Absicherung eingesetzt werden. Die Software-Anteile können auf einem virtuellen Steuergerät genauso mit einem HiL-Prüfstand stimuliert werden. Dabei ist der Prüfstand direkt wiederverwendbar, wenn im fortgeschrittenen Entwicklungsstand die reale Hardware und nicht mehr eine universelle Hardwareplattform eingesetzt wird.
Die Unterschiede der XiL-Methoden, im Vergleich zur virtuellen Absicherung, sind in Tabelle 2.4 aufgelistet. Die zusätzlich dargestellten Anwendungsfälle werden in Kapitel 2.5.2 definiert.

Abgrenzung zu Rapid-Control-Prototyping

Die virtuelle Absicherung und das RCP verfügen über viele Parallelen. Beide Methoden sind zur Erprobung von Software im frühen Entwicklungsstadium geeignet. Sie können unter Echtzeitbedingungen eingesetzt werden und funktionieren unter gewissen Voraussetzungen im realen Fahrzeug. Allerdings wird beim RCP nur ein Teil der Software betrachtet, meist in Form von horizontalen Prototypen (siehe Kapitel 2.3.2), die auch über eine Bypass-Schnittstelle in reale Steuergeräte integriert werden können. Die virtuelle Absicherung legt den Fokus hingegen ebenso auf die unteren Ebenen der Software eines Steuergerätes, wie die Basis-Software-Anteile. Somit handelt es sich um einen vertikalen Prototypen. Je nach Verfügbarkeit kann auch die vollständige Steuergeräte-Software abgesichert werden. Als Ausführungsplattform wird immer ein universelles Hardwaresystem eingesetzt.
Weiterhin dienen sowohl die virtuelle Absicherung als auch das RCP der frühzeitigen Validierung und Verifikation der Steuergeräte-Software, vor der Verfügbarkeit der Ziel-Hardware. Ebenso können beide Methoden frühe Mustersteuergeräte in der Integration ersetzen. Die virtuelle Absicherung legt jedoch einen zusätzlichen Schwerpunkt auf die Wiederverwendbarkeit der Software. Tabelle 2.4 berücksichtigt das RCP ebenfalls im Vergleich mit der virtuellen Absicherung.

2.4.3 Fazit

Die virtuelle Absicherung bietet die Möglichkeit, Steuergeräte-Software unabhängig von der Hardware abzusichern und damit die einfache Wiederverwendbarkeit der Funktions-Software sicherzustellen. Das wesentliche Unterscheidungsmerkmal zu den herkömmlichen XiL-Methoden ist die Abstraktion von der Ziel-Hardware bei gleichzeitiger Integration von realem Serien-Code, inkl. der Möglichkeit zur Hardware-Ansteuerung und Echtzeitausführung. Im Gegensatz zum Rapid-Control-Prototyping steht bei der virtuellen Absicherung die Wiederverwendbarkeit der Software im Vordergrund.

		Virtuelle Absicherung	Model-in-the-Loop	Software-in-the-Loop	Processor-in-the-Loop	Hardware-in-the-Loop	Rapid-Control-Prototyping
Eigenschaften	Realer Serien-Code	✓	✗	✓	✓	✓	✓
	Echtzeit	✓	✗	✗	✓	✓	✓
	Hardware-Schnittstellen	✓	✗	✗	✓	✓	✓
	Reale Steuergeräte-Hardware	✗	✗	✗	✓[1]	✓	✗
Anwendungsfälle	Entwicklungsbegleitende Tests ohne Ziel-Hardware	✓	✓	✓	✗	✗	✓
	Wiederverwendbarkeit der Funktions-Software	✓	✓	✓	✗	✗	✗
	Hardwareunabhängige Integration	✓	✗	✗	✗	✗	✓

✓ : Eigenschaft/Anwendungsfall mit Methode abdeckbar.

✗ : Eigenschaft/Anwendungsfall mit Methode nicht abdeckbar.

Tabelle 2.4: Vergleich von Eigenschaften und Anwendungsfällen der XiL-Methoden und RCP mit der virtuellen Absicherung.

[1] Der Mikrocontroller entspricht der Ziel-Plattform, die Steuergeräte-Peripherie ist in der Regel nicht vollständig vorhanden.

2.5 Virtuelle Absicherungsplattform

Zur Absicherung der Serien-Software von Steuergeräten, unabhängig von der Ziel-Hardware, setzt die BMW Group unter anderem auf die virtuelle Absicherungsplattform (VAP). In diesem Abschnitt werden das Konzept, die Anwendungsfälle und die technische Realisierung näher erläutert.

2.5.1 Konzept

Die Idee hinter der VAP besteht darin, die Funktions-Software von ihrer Ziel-Hardware zu abstrahieren und somit vor allem die Wiederverwendbarkeit der Funktion sicherzustellen. Die Voraussetzung für dieses Vorhaben liefern Software-Standards wie AUTOSAR und GENIVI. Sie schaffen eine einheitliche Abstraktionsebene zur Trennung von Hardware und Software.

Abbildung 2.11: Schichtenarchitektur der VAP mit System-under-Test (weiß) und plattformspezifischen Anteilen (blau) [FJM10].

Die Schichtenarchitektur der Plattform ist in Abbildung 2.11 dargestellt. In der obersten Ebene befindet sich dabei die eigentliche Funktions-Software, unterteilt in einzelne AUTOSAR-Software-Komponenten. Dabei handelt es sich um die reale Serien-Software, die später im Fahrzeug auf ihrer entsprechenden Ziel-Hardware ausgeführt wird. Darunter befindet sich die normale AUTOSAR-Architektur. Sie besteht aus der RTE, welche die Kommunikation der Software-Komponenten untereinander ermöglicht und die Verbindung zur darunterliegenden Basis-Software (BSW) herstellt. Bei der BSW handelt es sich um einen vollständigen AUTOSAR-Stack, wie er auch im Ziel-Steuergerät eingesetzt wird. Dadurch können nicht nur die Software-Komponenten, sondern auch die Konfiguration der BSW abgesichert werden.

Dies ist notwendig, da die große Menge an Parametern der BSW-Konfiguration ein erhebliches Fehlerpotential bietet. Erst ab den hardwareabhängigen Anteilen der BSW, genannt Microcontroller-Abstraction-Layer (MCAL), kommen plattformspezifische Module zum Einsatz. Somit können sämtliche Anteile der Software auf der VAP identisch zum Ziel-Steuergerät verwendet werden, welche in Abbildung 2.11 grau hinterlegt sind.
Die darunterliegende Hardware der VAP substituiert das eigentliche Ziel-System für die Funktions-Software. Als Plattform dient entweder ein Echtzeitrechner, welcher nach außen über alle relevanten Schnittstellen realer Steuergeräte verfügt, oder das VAP-System wird in einer virtuellen Maschine auf dem Entwickler-PC ausgeführt. Die technische Realisierung wird in Kapitel 2.5.3 näher erläutert. Das vollständige Konzept ist in [FJM10] und [MCW12] beschrieben.

2.5.2 Anwendungsfälle

Bei der BMW Group wurden für die VAP in [FJM10] verschiedene Anwendungsfälle definiert, die im Folgenden beschrieben werden. Eine Darstellung der Anwendungsfälle entlang des V-Modells ist in Abbildung 2.12 enthalten.

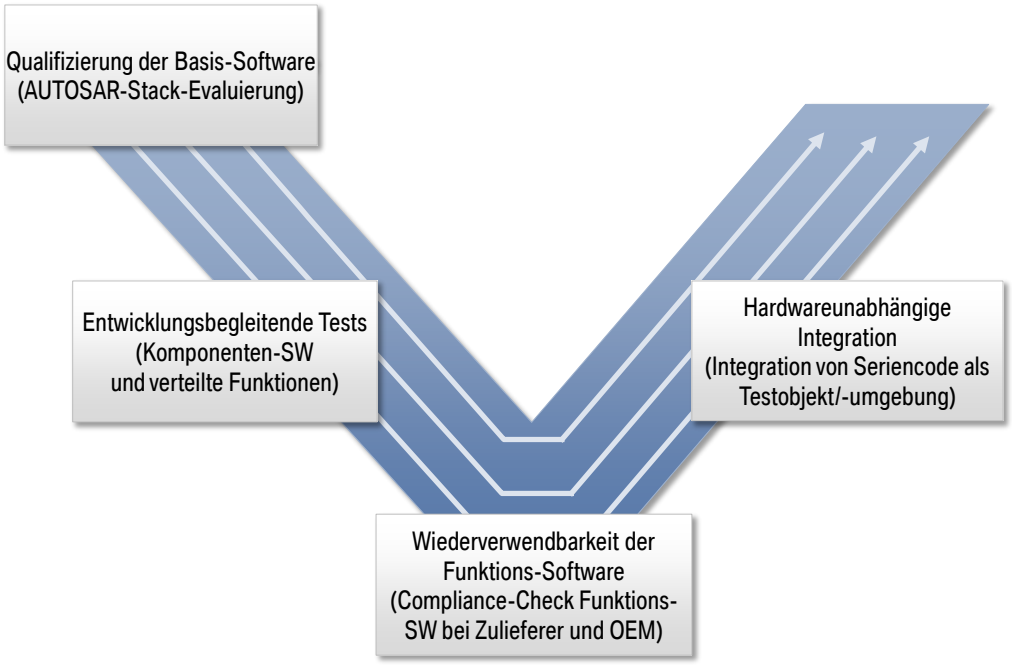


Abbildung 2.12: Anwendungsfälle der VAP entlang des V-Modells der Komponentenentwicklung [FJM10].

- **Qualifizierung der Basis-Software:** Im Rahmen der Vorentwicklung dient die VAP als Referenzplattform, um auf einem definierten, konstanten System verschiedene Basis-Software-Stacks gegeneinander vergleichen zu können. Dabei werden sowohl Stacks unterschiedlicher Hersteller als auch in unterschiedlichen Versionen betrachtet und für den Einsatz in Steuergeräten qualifiziert.

- **Entwicklungsbegleitende Tests:** Der zweite Anwendungsfall definiert den Einsatz der VAP auf der linken Seite des V-Modells, begleitend zur Funktions-Entwicklung. Dabei wird die Plattform am Entwickler-Arbeitsplatz eingesetzt und liefert die Möglichkeit, die Software bereits in der frühen Entwicklungsphase zu testen und vor allem erlebbar zu machen. Die Testphase kann somit im Entwicklungsprozess vorgezogen werden, was als *Frontloading* [WHR08] bezeichnet wird. Dadurch können Fehler früher aufgedeckt werden, was zu einer signifikanten Kostenreduktion führt, da sich die Kosten zur Fehlerbehebung nach der 10er-Regel [SP10] in jeder Prozessstufe verzehnfachen.

- **Wiederverwendbarkeit der Funktions-Software:** Am unteren Ende des V-Modells wird die VAP zur Sicherstellung der Standard-Konformität und der Wiederverwendbarkeit der Funktions-Software eingesetzt. Dies bezieht sich auf AUTOSAR-Software-Komponenten, welche auf ihre Hardwareunabhängigkeit geprüft werden. Die Serien-Software wird dazu auf dem x86-System der VAP im Bezug auf Konfigurierbarkeit, Integrierbarkeit, Kompilierbarkeit, Ausführbarkeit und Vollständigkeit untersucht und gegen die Spezifikation verifiziert.

- **Hardwareunabhängige Integration:** Die VAP kann ebenfalls auf der rechten Seite des V-Modells in der Integration und Absicherung eingesetzt werden. So ersetzt die Plattform frühe Muster von Steuergeräten im Teilsystemverbund, oder sie wird direkt in das Gesamtfahrzeug mit integriert. Dies ist möglich, da die VAP nach außen dieselben Schnittstellen zur Verfügung stellt, wie ein normales Steuergerät.

2.5.3 Technische Realisierung

Das ursprüngliche Konzept der VAP aus [FJM10] sieht zwei unterschiedliche Ausprägungen der Plattform vor, als Echtzeitrechner oder virtuelle Maschine. Beide Varianten basieren auf einem Linux-Betriebssystem, welches für Echtzeitfähigkeit erweitert wurde. Die Funktions-Software, zusammen mit dem AUTOSAR-Stack und dem AUTOSAR-Betriebssystem, wird als Applikation innerhalb des Linux-Systems ausgeführt. Bei Einsatz der virtuellen Maschine wird das Linux-System der VAP direkt auf dem Entwickler-PC virtualisiert. Es ist keine zusätzliche Hardware nötig. Dadurch kann die Funktions-Software und die Basis-Software bereits für die VAP kompiliert werden. In der virtuellen Maschine kann die Echtzeitfähigkeit des Systems jedoch nicht garantiert werden. Die Applikation ist dennoch ausführbar, wenn nur ein geringer Schnittstellenumfang genutzt wird.
Soll die Software im Teilsystem mit anderen Steuergeräten über Bussysteme verbunden oder reale Sensoren und Aktuatoren angeschlossen werden, so steht die zweite Ausführung der VAP als Echtzeitrechner zur Verfügung. Die Plattform basiert dabei auf einem PC-System mit x86-Architektur, welches je nach Hersteller z. B. als Industrie-PC mit Steckkarten im Format Compact-Peripheral-Component-Interconnect (C-PCI) ausgeführt ist. Der modulare Aufbau des Systems erlaubt die bedarfsorientierte Erweiterung um alle gängigen Schnittstellen normaler Steuergeräte-Hardware. Aktuelle Varianten der VAP unterstützen alle relevanten Bussysteme wie CAN, LIN, FlexRay und Ethernet. Weiterhin kann auf digitale und analoge Aus- und Eingänge zurückgegriffen werden sowie auf Ausgänge mit Pulsweitenmodulation (PWM). Dadurch kann eine VAP ein reales Steuergerät im Systemverbund vollständig ersetzen und ist ebenso in ein Fahrzeug integrierbar.

Bei der BMW Group als VAP bezeichnete Plattformen werden inzwischen von verschiedenen Herstellern auf dem freien Markt vertrieben [MBt14, ZELM12, DEF12, HK13] und können mit verschiedenen AUTOSAR-Stacks ausgestattet werden [Men14, Ele14, MGIL07, COM14].

2.5.4 Weiterentwicklung der Plattform

Der ursprüngliche Ansatz der VAP wurde konsequent weiterentwickelt. Im Folgenden werden die Erweiterungen dargestellt, die bereits in [DHM12b] veröffentlicht wurden. Abbildung 2.13 stellt eine schematische Übersicht aller Erweiterungsumfänge dar.

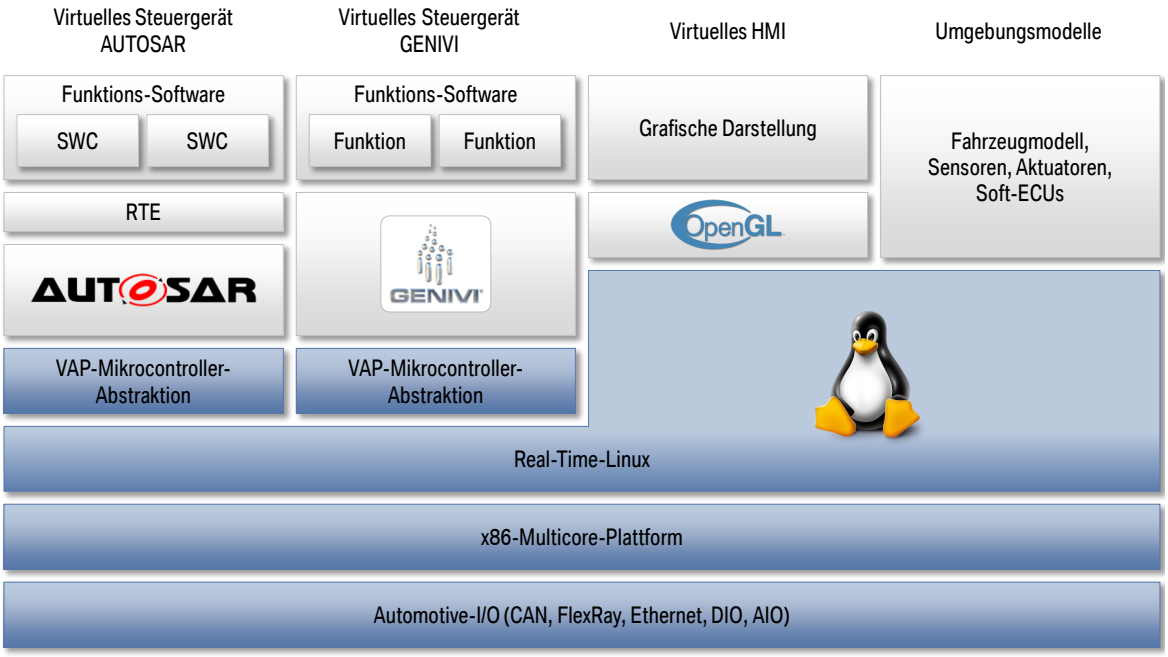


Abbildung 2.13: Weiterentwicklung des Aufbaus der VAP [DHM12b].

Eine neue Funktion erlaubt die Ausführung mehrerer virtueller Steuergeräte parallel auf einer Plattform. Dabei haben die Applikationen jeweils fest zugewiesene Hardware-Komponenten und können z. B. über die realen Busse der VAP miteinander kommunizieren. In Zukunft ist auch die Kommunikation über virtuelle Bussysteme denkbar.
Der Einsatz von Infotainment-Systemen, die auf GENIVI [GEN12] basieren, ist auf der VAP ebenfalls möglich. Der GENIVI-Anteil in den sogenannten Head-Units übernimmt z. B. Navigations- und Multimedia-Funktionen. Daneben existiert zusätzlich ein AUTOSAR-Anteil für die Buskommunikation mit anderen Steuergeräten. Diese Architektur kann auf einer VAP genauso dargestellt werden.
Weiterhin wurde eine Möglichkeit geschaffen, grafische Darstellungen von Mensch-Maschine-Schnittstellen direkt auf einem, an der VAP angeschlossenen, Display anzuzeigen. Dafür wurde eine leistungsstarke 3D-Grafikkarte mit OpenGL-Schnittstelle [Ope12] integriert. Dies ermöglicht die Abbildung der realen Steuergeräte-Architektur z. B. eines Kombi-Instruments, Head-Up-Displays oder einer Head-Unit, wobei der AUTOSAR-Anteil der Software auf einem Mikrocontroller läuft und die grafische Darstellung über einen separaten Grafikchip realisiert ist.

Ein weiteres geplantes Feature ist die direkte Integration von Umgebungsmodellen. So könnten vollständige Fahrmanöver ohne zusätzliche Hardware abgebildet werden, die plausible Werte für die Ansteuerung der virtuellen Steuergeräte liefern. Dafür bietet sich der Standard des Functional-Mockup-Interface (FMI) [Mod12] an, der sich zu einem neuen Industriestandard etablieren soll. Dies würde den Closed-Loop-Betrieb der Software mit einem HiL-System direkt am Entwicklerarbeitsplatz ermöglichen.

2.5.5 Integrationsprozess

Der Prozess zur Integration von Steuergeräte-Software auf der VAP gestaltet sich ähnlich wie bei herkömmlichen Mikrocontroller-Plattformen. Zunächst kann die Funktions-Software wie gewohnt für die geplante Ziel-Plattform entwickelt werden. Dabei kann der gewohnte AUTOSAR-konforme Architektureditor zur Erzeugung der nötigen AUTOSAR-Beschreibung der Funktions-Software genutzt werden. Genauso sind alle Standard-Werkzeuge für die Software-Entwicklung einsetzbar, wie z. B. Eclipse [Ecl14]. Wichtig ist nur, dass die Software für die Integration in einem kompatiblen Format nach dem AUTOSAR-Standard vorliegt. Je nach eingesetztem AUTOSAR-Stack handelt es sich normalerweise um das ARXML-Format für die Beschreibungsdateien und C-Sourcen, bzw. Header-Files oder Object-Files für den eigentlichen Funktions-Code.
Idealerweise wird auf der VAP derselbe AUTOSAR-Stack eingesetzt wie später auf der geplanten Ziel-Hardware. In diesem Fall kann die Konfiguration der Basis-Software von der VAP direkt für die Ziel-Plattform übernommen werden, wenn der Entwicklungsprozess weiter fortschreitet. Lediglich für die MCAL-Anteile müssen die speziell für die VAP entwickelten Module genutzt und konfiguriert werden. Bei der Nutzung eines anderen AUTOSAR-Stacks muss die BSW-Konfiguration gegebenenfalls portiert werden.
Mit den BSW-Generatoren des jeweiligen Stack-Herstellers wird aus der Konfiguration der dynamische Code-Anteil der Basis-Software erzeugt. Der vollständige Code der BSW und alle Software-Komponenten werden danach auf die VAP kopiert, dort kompiliert und zu einem Executable im ELF-Format gelinkt. Dabei wird pro Steuergerät ein Executable erzeugt, welches als virtuelles Steuergerät auf der VAP ausgeführt und getestet wird. Die parallele Ausführung mehrerer virtueller Steuergeräte auf einer Plattform ist ebenso möglich.
Der vollständige Integrationsprozess auf der VAP, im Vergleich zum realen Steuergerät, ist in Abbildung 2.14 dargestellt. Dabei wird deutlich, dass die Prozesse bis zum Kompilieren vollkommen identisch ablaufen. Es werden lediglich unterschiedliche Compiler eingesetzt, die jeweils ein eigenes Executable für die entsprechende Plattform erzeugen.

2.5.6 Serieneinsatz

Bei der BMW Group wird die VAP bereits seit dem Jahr 2010 genutzt. Nach verschiedenen Pilotprojekten wird die Plattform inzwischen über sämtliche Fahrzeugdomänen in der Serienentwicklung erfolgreich eingesetzt und fördert die Aufdeckung von Fehlern und Inkompatibilitäten in der frühen Entwicklungsphase.

Steuergeräte-Projekte

Eine Darstellung verschiedener Steuergeräte-Projekte, die bereits auf der VAP integriert wurden, ist in [DHM12b] enthalten. Es folgt eine Zusammenfassung der wichtigsten Projekte:

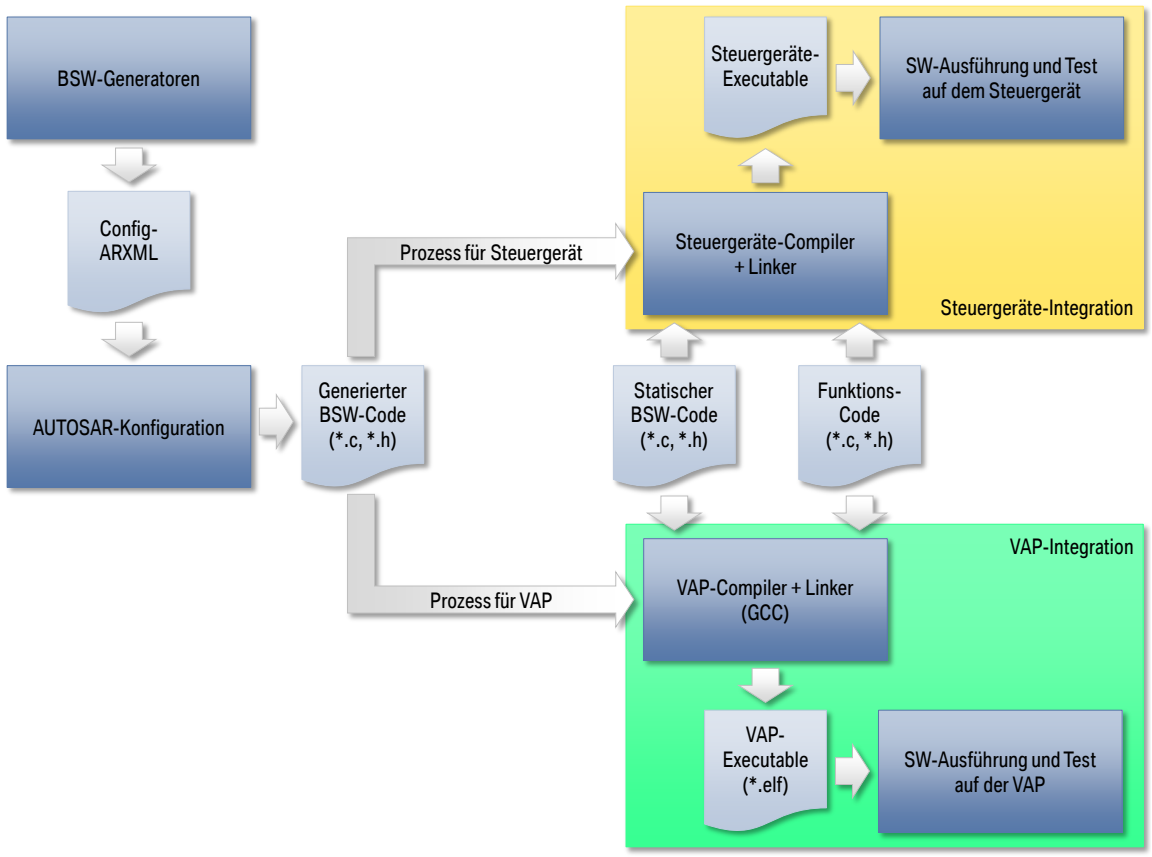


Abbildung 2.14: Prozess zur Integration von Software auf der VAP im Vergleich zum Steuergerät (ECU).

- **Instrumentenkombination:** Die Entwicklung der AUTOSAR-Anteile für die Instrumentenkombination bei der BMW Group erfolgt inzwischen ausschließlich auf der VAP. Dadurch kann die Software intensiv auf einer unabhängigen Hardwareplattform geprüft werden, bevor die Übergabe an den Zulieferer zur Integration in das Ziel-System erfolgt. Somit wird eine deutlich höhere Qualität der Lieferung erreicht, und gleichzeitig wird die Wiederverwendbarkeit der Baukasten-Software für alle Varianten der Instrumentenkombination über verschiedene Baureihen sichergestellt. Eine ausführliche Beschreibung des Projekts ist in [MDF13a, MDF13b] enthalten.
- **Motorsteuerung:** Mehrere Komponenten aus der Eigenentwicklung einer neuen elektronischen Motorsteuerung für Verbrennungsmotoren wurden auf der VAP erstmals integriert und abgesichert.
- **Fahrdynamikregelsystem:** Für ein Fahrdynamikregelsystem konnten die zuständigen Anteile für die Buskommunikation mit dem CAN-Bus und FlexRay auf der Plattform erfolgreich verifiziert werden.
- **Cabrioverdeck-Ansteuerung:** Bei der Ansteuerung eines Cabrioverdecks wurden sowohl Komponenten aus der Eigenentwicklung der BMW Group als auch Anteile eines Zulieferers in die VAP integriert und darüber die AUTOSAR-Konformität sichergestellt.
- **Fahrzeug-Zustandsmanagement:** Domänenübergreifend erfolgte die Validierung eines neuen Fahrzeug-Zustandsmanagements ebenfalls virtuell, wobei die Konzeptimplementierung erstmals auf der VAP ausgeführt wurde. Durch die Integration der Plattform in ein reales Fahrzeug konnte die Funktion in der frühen Entwicklungsphase dargestellt werden und war aus Kundenperspektive erlebbar.
- **AUTOSAR-Acceptance-Tests:** Die Referenzapplikation für die sogenannten AUTOSAR-Acceptance-Tests wurde ebenfalls auf der VAP implementiert und mehrere der Testfälle des zugehörigen Work-Packages 2.3 des AUTOSAR-Konsortiums erfolgreich erprobt.

Weitere Projekte umfassen Teile der Sicherheitselektronik, ein Domänenleitsteuergerät für die Karosserieelektronik und ein Steuergerät zur Sensordatenfusion.

Aufgedeckte Fehler

Wie ebenfalls in [DHM12b] gezeigt, konnten in allen aufgeführten Projekten vielfältige Fehler frühzeitig erkannt und behoben werden. Dies umfasst Verletzungen und Inkonsistenzen des AUTOSAR-Standards, welche jeweils an das Konsortium oder die entwickelnden Fachbereiche gemeldet wurden. Dabei sind meist unterschiedliche Interpretationen des Standards der Auslöser der Inkonsistenzen gewesen. Weiterhin konnten in der Software unterschiedlichste Implementierungsfehler sowie Fehler in den zugehörigen Testfällen aufgedeckt und behoben werden. Die Bandbreite der Implementierungsfehler reichte dabei von Unterschieden in der Groß- und Kleinschreibung der Dateien, welche durch das Linux-System der VAP auffielen, bis hin zu Programmierfehlern innerhalb der Funktionen der Software. Genauso wurden Inkonsistenzen im Scheduling der einzelnen Tasks festgestellt sowie falsche Konfigurationen in der Basis-Software identifiziert. Dabei hat sich der Einsatz einer realen Basis-Software in der VAP bereits bewährt.

In allen Projekten wurde durch die Plattform die frühzeitige Behebung von Fehlern ermöglicht und somit ein entscheidender Beitrag zu Qualität und Kostenoptimierung geleistet. Dass auch die Zulieferer davon profitieren, wird in [VDL14] dargestellt.

2.5.7 Fazit

Die VAP dient bei der BMW Group als System zur virtuellen Absicherung und ermöglicht die vollständige Integration von AUTOSAR-konformen Software-Komponenten. Die Plattform eignet sich für den entwicklungsbegleitenden Einsatz, sowie in der hardwareunabhängigen Integration und zur Verifikation der Wiederverwendbarkeit von Funktions-Software. Die Anwendung bei mehreren Steuergeräte-Projekten erlaubte die frühzeitige Identifikation einer Vielzahl von Fehlern, wodurch die Wirksamkeit der VAP bestätigt werden konnte. Die Plattform bietet jedoch bisher keine Möglichkeit zur Integration der hardwarenahen Complex-Device-Drivers, die allerdings einen wichtigen Teil der Steuergeräte-Software bilden. Für diese Problemstellung werden im weiteren Verlauf dieser Arbeit Lösungen entwickelt.

2.6 Verwandte Arbeiten

Diese Dissertation steht im Kontext zu vier verschiedenen Themengebieten. Dabei handelt es sich zum Einen um die *virtuelle Absicherung*, wie bereits in Kapitel 2.4 beschrieben. Das zweite Themengebiet sind die sogenannten Complex-Device-Drivers, welche als hardwareabhängige Komponenten in der virtuellen Absicherung berücksichtigt werden sollen und im weitesten Sinne als *Hardware-Treiber* bezeichnet werden können (siehe Kapitel 2.2). Als Lösungsansatz für die Berücksichtigung der CDDs werden in Kapitel 3 Konzepte entwickelt, die auf den Methoden der *Hardware-Abstraktion* und der *Hardware-Simulation* aufbauen, welche das dritte und vierte relevante Themengebiet darstellen.
Zu den vier Gebieten existieren bereits verschiedene Arbeiten, welche thematisch an diese Dissertation angrenzen und in der vorangegangen Recherche analysiert wurden, im weiteren Verlauf jedoch nicht mehr im Fokus stehen. Daher werden diese verwandten Arbeiten im folgenden Kapitel kurz zusammengefasst, wobei die Gliederung nach den vier Themengebieten erfolgt.

2.6.1 Arbeiten zu virtueller Absicherung

Nicht nur bei der BMW Group wird die virtuelle Absicherung von Steuergeräte-Software vorangetrieben, auch andere OEMs beschäftigen sich intensiv mit der Thematik. Bei der Daimler AG werden sehr ähnliche Ansätze und Anwendungsfälle wie bei der BMW Group verfolgt (siehe Kapitel 2.5.2), wie auch die gemeinsame Veröffentlichung [PWMS12] zeigt. Bei der Daimler AG wird die Plattform als Virtuelle Integrations- und Test-Plattform (VIT) [Hon13] bezeichnet und setzt auf eine simulierte AUTOSAR-Basis-Software, die innerhalb eines HiL-Systems betrieben wird, wie in [HK13] dargestellt.
Auch die Audi AG ist dabei, ein System zur virtuellen Software-Absicherung, genannt Virtuelle Software Entwicklungsplattform - AUTOSAR (VESP-A), aufzubauen. Auch hierbei stehen ähnliche Ziele im Fokus, wie die Absicherung vor Verfügbarkeit der Ziel-Hardware und die Einbeziehung der Basis-Software. Darüber hinaus plant die Audi AG den Einsatz von Instruction-Set-Simulation (ISS) zur Abbildung der Ziel-Plattform auf einem PC-System [SM13].

Das Problem der Integration von CDDs in die virtuelle Absicherung, womit sich diese Dissertation beschäftigt, wurde dabei bisher weder in den Veröffentlichungen der OEMs noch in der restlichen Literatur explizit betrachtet.

2.6.2 Arbeiten zu Hardware-Treibern

Das Thema der Hardware-Treiber wurde in der Literatur bereits intensiv betrachtet. Eine umfassende Quelle über hardwareabhängige Software im Allgemeinen stellt z. B. [EMD09] dar. Die darin enthaltene Veröffentlichung [EESV09] beschreibt die Modellierung und Implementierung von Schnittstellen zwischen Hardware und Software. Speziell wird dabei auf Register von Mikrocontrollern eingegangen, wobei deren Funktionen erläutert und die Zugriffsmechanismen beschrieben werden.
Ebenfalls in [EMD09] enthalten ist [SDG09], worin eine Methode zur Generierung von hardwareabhängiger Software erläutert wird. Dabei wird von einer abstrakten Spezifikation ein Modell des Systems erzeugt, welches als Eingangsparameter dient, um die hardwarespezifischen Software-Anteile automatisch zu generieren.
In Bezug auf Treiber in einem Linux-System wird auf [CRKH05] verwiesen. Darin wird die Definition, der Einsatz und die Entwicklung von Kernel-Modulen ausführlich beschrieben. Weiterhin werden Details der Implementierung von Hardware-Treibern erläutert, wie z. B. die Kommunikation mit der Hardware, die Interrupt-Behandlung oder das Memory-Mapping.
Die CDDs aus dem AUTOSAR-Umfeld werden z. B. in [DJG+10] betrachtet. Dabei liegt der Fokus auf der Erläuterung der Eigenschaften und der Funktionalität der CDDs. Weiterhin sind die Einsatzmöglichkeiten aus [AUT08] zusammengefasst, die ebenfalls in Kapitel 2.2.2 dieser Dissertation berücksichtigt werden.
Dass CDDs auch für Testzwecke einsetzbar sind, wird in [EHH11] und [EHH13] beschrieben. Dabei wird ein Testkonzept aufgestellt, mit welchem die applikationsspezifische Konfiguration der AUTOSAR-Basis-Software überprüft werden soll. Ein CDD dient dabei dazu, den direkten Zugriff auf die verschiedenen Module der Basis-Software zu ermöglichen.
Das zentrale Thema dieser Dissertation, die Betrachtung von Hardware-Treibern im Bezug auf die virtuelle Absicherung, unabhängig von der Ziel-Hardware, wurde in der Literatur bisher nicht berücksichtigt.

2.6.3 Arbeiten zu Hardware-Simulation

Die Simulation verschiedener Hardware-Komponenten wird oft für Prototyping-Lösungen und Absicherungszwecke genutzt, wenn die Ziel-Hardware noch nicht zur Verfügung steht. Verschiedene Hersteller bieten Lösungen an, mit denen vollständige Mikrocontroller-Architekturen simuliert werden können, wie z. B. der Synopsys Virtualizer [Sch10, SN13] und die Cadence Virtual-System-Platform [Avi11]. Hierbei ist jeweils die Betrachtung der Architektur und einzelner Operationen bis auf Register-Transfer-Level (RTL) möglich. Die Modelle der Mikrocontroller werden direkt vom Chiphersteller angeboten, welche ihre Architekturen mit denselben Modellen virtuell entwickeln. Als Beschreibungssprache wird für tiefgreifende Simulationen auf RTL, die Very-High-Speed-Integrated-Circuit-Hardware-Description-Language (VHDL) [IEE09] oder Verilog-HDL [IEE06] eingesetzt, wobei die Komplexität der Simulation dann keine Echtzeit zulässt. Für abstraktere Modelle auf Transaktionsebene wird meist SystemC eingesetzt (sog. Transaction-Level-Modeling (TLM) [GLMS02]). Bei hoher Abstraktion der Architektur und geringer Komplexität des Systems ist damit sogar Echtzeitfähigkeit

erreichbar. Hardware-Simulation mit SystemC im Zusammenhang mit dem AUTOSAR-Standard wird umfassend in [KBR09] behandelt, wobei CDDs zwar erwähnt, aber nicht näher berücksichtigt werden.
Für die Simulation von Architekturen wie x86, PowerPC und ARM existiert z. B. die Open-Source-Lösung QEmu [QEm14]. Dabei ist auch eine Co-Simulation mit QEmu und SystemC möglich, wie in [FQV+09] ausgeführt. Eine weitere Lösung, die zusätzlich für verschiedene Mikrocontroller nutzbar ist, stellt der OVPsim Simulator [OVP14] dar. Simulation und Modellierung mit SystemC im Zusammenhang mit Echtzeitbetriebssystemen wird in [ZMG09] behandelt.
Ein Ansatz zur Simulation der spezifischen Hardware zu Complex-Device-Drivers (CDDs) oder zur vollständigen CDD-Simulation in AUTOSAR-Steuergeräten, wie in Kapitel 4.1 dieser Dissertation vorgestellt, wurde in der Literatur bisher nicht untersucht.

2.6.4 Arbeiten zu Hardware-Abstraktion

Die Wiederverwendbarkeit von Software kann durch die Abstraktion von Hardware und Software sichergestellt werden. Dabei werden direkte Zugriffe z. B. auf Hardware-Treiber über eine zusätzliche Schicht, einen Hardware-Abstraction-Layer (HAL), realisiert. Dies ermöglicht die vollständig unabhängige Gestaltung der Funktions-Software von der darunterliegenden Hardware. Ein Beispiel dafür ist der Microcontroller-Abstraction-Layer (MCAL) im AUTOSAR-Standard [AUT11b] (siehe Kapitel 2.1.2).
In [PJ09] wird ein HAL dazu eingesetzt, einen Software-Stack auf unterschiedlichen Systems-on-Chip mit Multi-Prozessor-Architektur auszuführen. Dabei steht speziell der Aspekt der Wiederverwendbarkeit der Software im Vordergrund. An einem Beispiel wird anschaulich die Implementierung einer Funktion des HAL für verschiedene Prozessoren erläutert.
Die Definition eines HAL und sein Einsatz im Kontext zum Design von Systems-on-Chip wird in [YJ03] beschrieben. Verschiedene Methoden zur Erzeugung eines HAL werden ebenfalls dargestellt.
Der Einsatz eines HAL für die virtuelle Absicherung von CDDs im AUTOSAR-Kontext, wie in Kapitel 4.2 dieser Dissertation erläutert, wurde bisher jedoch nicht in der Literatur behandelt.

2.7 Zusammenfassung

Der Software-Standard AUTOSAR schafft die Grundlage für eine einfache Wiederverwendbarkeit von Software und soll damit die Komplexität aktueller Steuergeräte beherrschbar machen. Eine besondere Rolle spielen dabei die sogenannten Complex-Device-Drivers (CDDs). Sie bieten die Möglichkeit, dennoch hardwareabhängige Software in ein AUTOSAR-Steuergerät mit einzubringen und werden für ein breites Anwendungspektrum eingesetzt.
Die virtuelle Absicherung wird im Kontext dieser Dissertation als Methode definiert, um Steuergeräte-Software unabhängig von ihrer Hardware abzusichern und die Wiederverwendbarkeit der Software sicherzustellen. Zur Durchführung der virtuellen Absicherung wird die sogenannte virtuelle Absicherungsplattform (VAP) eingesetzt. Über das Thema der Wiederverwendbarkeit hinaus kann die Plattform zum Vergleich von Basis-Software, für die hardwareunabhängige Entwicklung und den Ersatz von frühen Mustern von Steuergeräten eingesetzt werden.

Verwandte Arbeiten zu den wesentlichen Themengebieten dieser Dissertation befassen sich ebenso mit der virtuellen Absicherung, Hardware-Treibern, Hardware-Simulation und -Abstraktion. Allerdings wurden die verschiedenen Gebiete bisher nicht im Kontext zueinander betrachtet, was im Verlauf dieser Dissertation erfolgt.

Kapitel 3

Complex-Device-Drivers in der virtuellen Absicherung

Eine virtuelle Absicherungsplattform (VAP) ist optimal für die Absicherung von vollständig AUTOSAR-konformen Software-Komponenten geeignet (siehe Kapitel 2.5). Für einen ganzheitlichen Absicherungsansatz müssen allerdings die hardwareabhängigen Anteile der Steuergeräte-Software, die Complex-Device-Drivers (CDDs), mit betrachtet werden. Dabei handelt es sich um das wesentliche Thema dieser Arbeit. Dieses Kapitel definiert zunächst die Testziele für die Berücksichtigung der Treiber in der virtuellen Absicherung. Danach folgt eine Erläuterung der verschiedenen Konzepte, die zur Berücksichtigung der CDDs in der virtuellen Absicherung entwickelt wurden. Eine Bewertung dieser Konzepte anhand ausgewählter Kriterien soll die Teilmenge mit den höchsten Erfolgsaussichten in der Praxis ermitteln.

3.1 Testziele für die Berücksichtigung von Complex-Device-Drivers

Für die virtuelle Absicherung wurden in Kapitel 2.5.2 vier verschiedene Anwendungsfälle beschrieben. Dabei ist die Berücksichtigung von CDDs in der Steuergeräte-Software in den drei Fällen nötig, in welchen die Funktions-Software betrachtet wird. Lediglich für die „Qualifizierung der Basis-Software“ sind CDDs nicht von Bedeutung. Anhand der drei relevanten Anwendungsfälle werden im Folgenden Testziele abgeleitet, die eine Berücksichtigung der CDDs erfordern.

3.1.1 Integration der Software-Komponenten

Im Anwendungsfall „entwicklungsbegleitender Tests“ liegt das erste Ziel bei Einsatz einer VAP in der Integration und Verifikation der Software-Komponenten. In vielen Fällen werden die Komponenten anteilig von Zulieferern und OEMs entwickelt und bei der Integration erstmals gemeinsam betrachtet. Dabei wird sichergestellt, dass die Schnittstellen der Software-Components (SWCs) kompatibel sind und der Datenaustausch reibungslos funktioniert. Ebenso wird die Vollständigkeit der Komponenten sowie die Anbindung an die Basis-Software

und deren Konfiguration geprüft. Im einfachsten Fall wird begutachtet, ob die Software insgesamt kompilierbar ist.
Sind Complex-Device-Drivers in der Software enthalten, so bestehen immer Schnittstellen zwischen dem Treiber und den restlichen Software-Komponenten. Um die vollständige Integrierbarkeit der Software sicherzustellen, ist die Berücksichtigung dieser Verbindungen essentiell. Software-Komponenten müssen mit korrekten Daten versorgt werden, um plausible Testergebnisse zu erzeugen. Ebenso müssen Daten verifizierbar sein, die von den Software-Komponenten an einen CDD übertragen werden. Der Treiber liegt dabei nicht im Testfokus, ist allerdings für die Absicherung der Funktions-Software notwendig.

3.1.2 Sicherstellung der Wiederverwendbarkeit der Funktions-Software

Wenn die Entwicklung der Steuergeräte-Software bereits soweit fortgeschritten ist, dass alle Komponenten verfügbar sind, kann eine VAP genutzt werden, um die „Wiederverwendbarkeit der Funktions-Software“ sicherzustellen. Dafür müssen, wie beim obigen Testziel, alle Komponenten der Software auf der Plattform integriert werden. Das Testziel ist erreicht, wenn die Integration im Bezug auf die unterschiedliche Architektur und den Wechsel des Compilers problemlos abläuft, keine Inkompatibilitäten auftreten und die Software auf der VAP korrekt ausgeführt werden kann. Dabei ist es sinnvoll, die Software mit verschiedenen AUTOSAR-Stacks zu evaluieren, da die Basis-Software anderer Hersteller Fehler und Inkonsistenzen mit dem Standard jeweils unterschiedlich tolerieren.
CDDs sind per Definition nicht auf Wiederverwendbarkeit ausgelegt. Dennoch sind sie Teil der Software und werden für die korrekte Ausführung aller Funktionen benötigt. Speziell bei der Integration der Software auf einem anderen Stack muss ein CDD mitgetestet werden, um die korrekte Funktion und die AUTOSAR-Konformität der Schnittstellen zu RTE und BSW sicherzustellen.

3.1.3 Funktionale Absicherung der Software

Bei dem Anwendungsfall der „hardwareunabhängigen Integration“ kann die VAP als Ersatz von frühen Steuergerätemustern in Teilsystemen oder im vollständigen Gesamtsystem eingesetzt werden. Die als A- und B-Muster bezeichneten Vorläufer der finalen Steuergeräte unterscheiden sich teilweise stark von der Serienversion und übersteigen deren Herstellungskosten um ein Vielfaches. Zusätzlich verhindern geringe Stückzahlen oft eine vollständige Bestückung aller Absicherungsinstanzen. Daher bietet der Ersatz der Steuergeräte-Muster durch eine VAP erhebliches Potential zur Kostenreduktion und zur Erhöhung der Verfügbarkeit.
Sowohl an Teilsystemen als auch am Gesamtsystem wird die Peripherie der Steuergeräte mit integriert. Damit die VAP einen vollwertigen Ersatz für ein Steuergerätemuster darstellen kann, muss die Ansteuerung der Sensoren und Aktuatoren genauso möglich sein. Für gängige Bussysteme oder standardisierte analoge und digitale Ein- und Ausgänge bietet die Plattform bereits alle nötigen Voraussetzungen.
Hardware-Komponenten müssen allerdings ebenso abbildbar sein, wenn sie über einen CDD angesteuert werden. Ohne den Treiber ist die Ansteuerung der zugehörigen Hardware nicht möglich und das Steuergerät kann nicht vollwertig ersetzt werden. Der CDD muss dafür idealerweise in unverändertem Zustand auf der VAP integriert werden, sonst ist die unverfälschte Bewertung des Gesamtsystems nicht sichergestellt und die Aussagekraft bei der Testdurchführung eingeschränkt.

3.1.4 Fazit

Die oben beschriebenen Testziele müssen mit einer VAP für jedes Steuergerät erreichbar sein. Dabei ist die Betrachtung der vollständigen Funktions-Software eines Steuergerätes notwendig. CDDs bilden einen wichtigen Teil der Software, weshalb die Berücksichtigung der Treiber bei der Integration auf einer VAP erforderlich ist.

3.2 Konzepte

Die VAP erlaubt bisher nur die vollständige Integration von AUTOSAR-Software-Komponenten. Die Berücksichtigung von CDDs ist, aufgrund ihrer spezifischen Hardware-Schnittstellen, nicht ohne zusätzliche Maßnahmen möglich. Im Folgenden werden unterschiedliche Ansätze dargestellt, die im Rahmen dieser Dissertation für die Lösung des Problems erarbeitet wurden. Dabei wurden Methoden zur anteiligen oder vollständigen Simulation der Treiber sowie die Integration verschieneder Hardwarekomponenten betrachtet.

3.2.1 Simulation des Ziel-Mikrocontrollers

Das erste Konzept sieht vor, die Ziel-Plattform des Complex-Device-Drivers in einer VAP zu simulieren. In Kapitel 2.6.3 wurden verschiedene Möglichkeiten zur Simulation von vollständigen Mikrocontroller-Architekturen beschrieben. Bei dem Synopsys Virtualizer [Sch10, SN13] existiert z. B. eine Version, die im Linux-System der VAP ausgeführt werden kann. Hierbei wird der vollständige Mikrocontroller in Form eines SystemC-Modells dargestellt. Ein CDD, der innerhalb der Simulationsumgebung ausgeführt wird, hat somit Zugriff auf sämtliche Schnittstellen und Register der Ziel-Hardware und kann ohne Anpassungen integriert werden. Die Ausführung der Software-Komponenten der Steuergeräte-Software läuft parallel auf derselben VAP. Abbildung 3.1 zeigt den schematischen Aufbau des Konzepts.

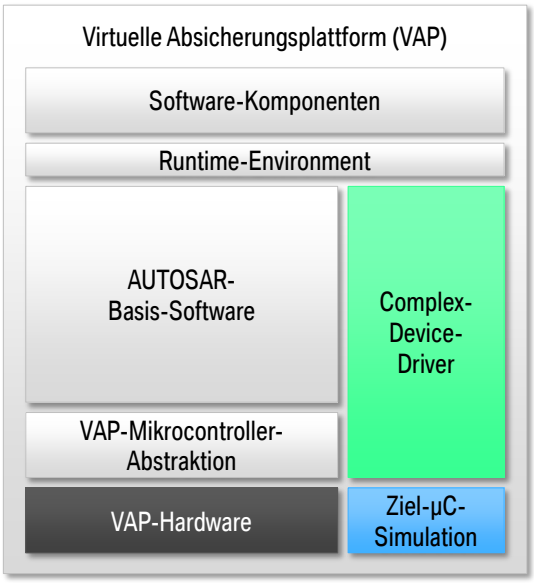


Abbildung 3.1: Simulation des Ziel-Mikrocontrollers in einer VAP.

Dadurch entsteht die Möglichkeit, den CDD auf seiner nativen Plattform auszuführen und dennoch die Wiederverwendbarkeit der restlichen Softwareanteile zu prüfen. Die Verfügbarkeit von Simulationsmodellen des jeweiligen Ziel-Mikrocontrollers ist dadurch sichergestellt, dass viele Chiphersteller die Modelle selbst für ihre Chipentwicklung erstellen. Dies erlaubt die Erprobung und Optimierung der Architektur des Mikrocontrollers bereits vor der Fertigung. Die Modelle sind grundsätzlich für die Simulation in Echtzeit nutzbar. Allerdings ist die Echtzeitfähigkeit bei komplexen Nachbildungen eines vollständigen Mikrocontrollers nicht

mehr zu garantieren. Dies schränkt den Einsatz des Konzepts auf einer VAP im Systemverbund mit realen Steuergeräten ein, da z. B. der Datenaustausch über Bussysteme ohne Echtzeitfähigkeit nicht korrekt darstellbar ist. Für die Realisierung des Konzepts ist zusätzlich eine zeitliche Synchronisation und eine zuverlässige Datenübertragung zwischen dem CDD in der Simulation und den restlichen Anteilen der Software notwendig.

3.2.2 Simulation des Hardware-Verhaltens

Alternativ zu einer vollständigen Simulation des Mikrocontrollers, wie im vorherigen Konzept dargestellt, kann eine einfache Simulation des Hardware-Verhaltens für den Complex-Device-Driver ausreichend sein. Dieses Konzept sieht vor, eine Simulation direkt an die Schnittstellen des Treibers zur Hardware anzubinden. Darüber werden die Eingänge des CDD mit Testvektoren stimuliert. Somit kann die grundsätzliche Funktionsweise des CDD und der eigentlichen Funktions-Software erprobt werden. Da die Schnittstellen zwischen Complex-Device-Driver und Hardware jedoch nicht in den zugehörigen AUTOSAR-Beschreibungsdateien dokumentiert sind, kann nicht auf ein einheitliches Beschreibungsformat zurückgegriffen werden. Dadurch ist eine automatisierte Erzeugung der Hardware-Simulation ausgeschlossen. Manuell ist dennoch eine Gegenkomponente für die CDD-Hardware-Schnittstellen erzeugbar, indem die Schnittstellen analysiert werden. Abbildung 3.2 stellt den zugehörigen Aufbau dar.

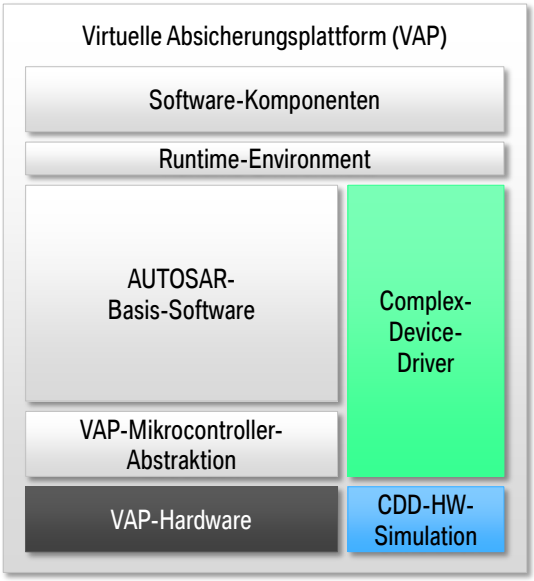


Abbildung 3.2: Simulation des Hardware-Verhaltens für die Schnittstellen des CDD.

Zugriffe auf Register sind z. B. über eine Variable in der Simulationsschicht nachzubilden, die als Ersatz für das reale Register dient. Bei Funktionsaufrufen muss die Low-Level-Funktion in der Simulationsschicht neu definiert werden, um simulierte Werte zurückzuliefern. Alternativ sind über Werkzeuge zur Testautomatisierung Testvektoren applizierbar. Da die Simulation direkt innerhalb der VAP ausgeführt wird, sind die Synchronisation und der Datenaustausch ohne zusätzliche Anpassungen möglich, im Gegensatz zu dem Konzept der Simulation des vollständigen Mikrocontrollers.

Gliedert sich der Complex-Device-Driver in einen Low-Level- und einen High-Level-Anteil, ist auch eine anteilige Integration möglich. Der High-Level-Anteil verfügt über die Schnittstellen zur RTE und kann damit den Datenaustausch mit den SWCs sicherstellen. Für die Absicherung der Schnittstellen zwischen dem Treiber und der Funktions-Software genügt somit die Integration des High-Level-Anteils des CDD. Dies erfordert eine Simulation für die Schnittstellen zwischen Low-Level- und High-Level-Anteil des Treibers. Die Erzeugung der Simulation kann sich dadurch vereinfachen, sofern der High-Level-Anteil z. B. über generische Schnittstellen verfügt.

3.2.3 CDD-Simulation

Steht der Complex-Device-Driver selbst nicht im Fokus der Absicherung, da nur die Software-Komponenten oberhalb der RTE betrachtet werden sollen, so ist dennoch eine Möglichkeit zur Datenübertragung über die Schnittstellen zwischen CDD und Funktions-Software notwendig. Ansonsten ist nicht der vollständige Funktionsumfang der Software-Komponenten testbar. Im Gegensatz zu den Zugriffen des Treibers auf die Hardware des Zielsystems sind die Schnittstellen zwischen den SWCs und dem CDD über die Software-Component-Descriptions (SWCDs) exakt definiert. Dies ermöglicht eine vollautomatische Erzeugung einer Gegenkomponente für alle Schnittstellen. Der Aufbau des Konzepts ist in Abbildung 3.3 dargestellt.

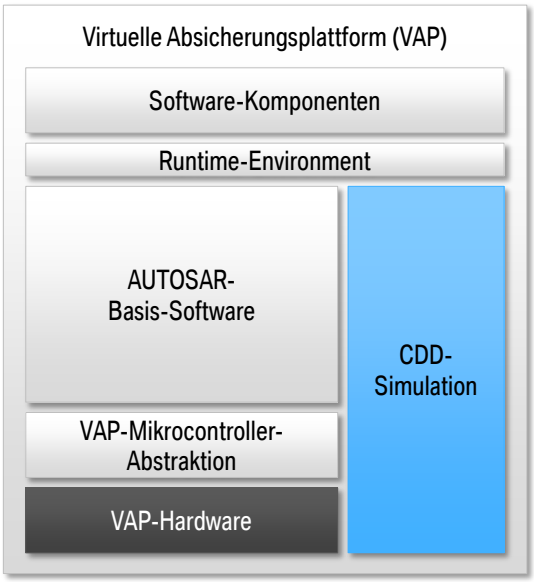


Abbildung 3.3: Vollständige Simulation des Complex-Device-Drivers.

Dafür wird auch die SWCD des CDD nicht benötigt, da alle Schnittstelleninformationen bereits in den Dateien der Software-Komponenten enthalten sind. Das Konzept sieht weiterhin eine generische Erzeugung aller nötigen AUTOSAR-Artefakte vor, inklusive des *Internal-Behavior* mit allen nötigen *Runnables*. Vollautomatisch generierter C-Code ermöglicht die reibungslose Integration der erzeugten Komponente in die reale Steuergeräte-Software. Dabei kann automatisiert eine graphische Oberfläche erzeugt werden, die einem Benutzer die Eingabe und Überwachung von Testvektoren für die CDD-Schnittstellen und damit eine Simulation des CDD-Verhaltens ermöglicht.

Für komplexe Simulationsabläufe kann die AUTOSAR-Beschreibung der Simulation in Modellierungswerkzeuge wie z. B. MATLAB/Simulink importiert werden. Dies ermöglicht die Nachbildung des Verhaltens des CDD im gewünschten Detailgrad für die Erreichung der Testziele. Hierbei ist jedoch keine Automatisierung mehr möglich. Aus dem Modell kann wiederum C-Code erzeugt werden, der sich innerhalb der CDD-Simulationskomponente in einer VAP ausführen lässt.

3.2.4 Integration der spezifischen Hardware

Grundsätzlich besteht die Möglichkeit, dass die zugehörigen Hardware-Komponenten eines Complex-Device-Drivers in einer Form vorliegen, die eine direkte Integration in eine VAP erlaubt. Dies ist z. B. der Fall, wenn ein CDD zur Ansteuerung von Komponenten genutzt wird, die auf einem Field-Programmable-Gate-Array (FPGA) modelliert sind. Abbildung 3.4 zeigt den Aufbau des Konzepts.

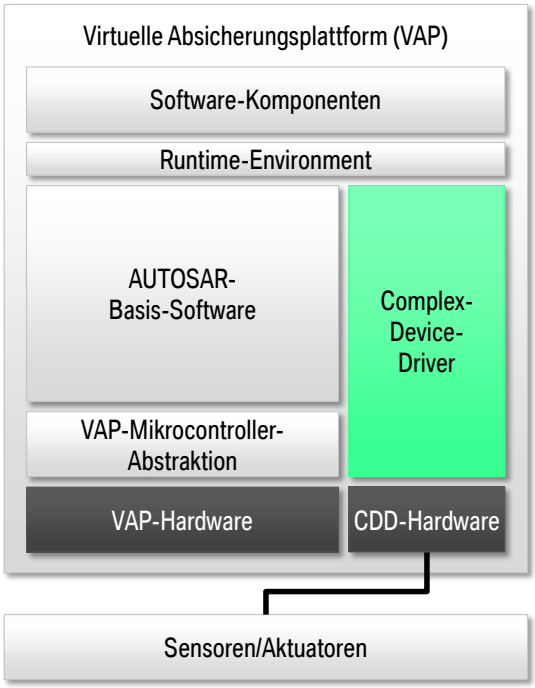


Abbildung 3.4: Integration der zugehörigen, spezifischen Hardware eines CDD in einer VAP.

Es existieren diverse FPGA-Module, die auf PCI- bzw. Compact-PCI-Karten verbaut sind, wodurch die Kompatibilität zu einer VAP sichergestellt ist. Somit sind keine zusätzlichen Software-Komponenten nötig, um den CDD mit der Hardware zu verbinden. Der Treiber kann direkt in seiner nativen Architektur betrieben werden und es entstehen keine zusätzlichen Aufwände für die Integration. Dies ermöglicht ebenfalls den Betrieb zugehöriger Sensoren und Aktuatoren, die z. B. am FPGA auf gewohnte Art und Weise anschließbar sind.
Die Hardware, welche über CDDs angesteuert wird, ist jedoch sehr vielfältig, wie in Kapitel 2.2 erläutert. Weiterhin ist die Verfügbarkeit von Hardware, die zu Standard-PC-Systemen wie der VAP kompatibel ist, aufgrund der eher geringen Verbreitung eines spezifischen CDD, stark

limitiert. Daher ist dieses Konzept der CDD-Integration nur für einen geringen Prozentsatz von Treibern anwendbar.

3.2.5 CDD-Hardware-Abstraction-Layer

Existiert keine passende Bauform der Hardware, welche kompatibel zu einer VAP ist, so besteht die Möglichkeit, auf die Standard-Hardware der Plattform zurückzugreifen. Ist z. B. ein CDD für einen spezifischen Analog-Digital-Converter (ADC) ausgelegt, kann er grundsätzlich auch mit Werten aus einem anderen ADC versorgt werden. Allerdings sind die Schnittstellen dabei üblicherweise nicht direkt kompatibel, weshalb Anpassungen in der Software nötig sind. Da ein CDD bei der Integration auf der VAP unverändert eingesetzt werden sollte, um die Vergleichbarkeit zum späteren Ziel-System zu gewährleisten, erfolgt die Anpassung der Schnittstellen bei dem hier beschriebenen Konzept über einen Hardware-Abstraction-Layer (HAL).

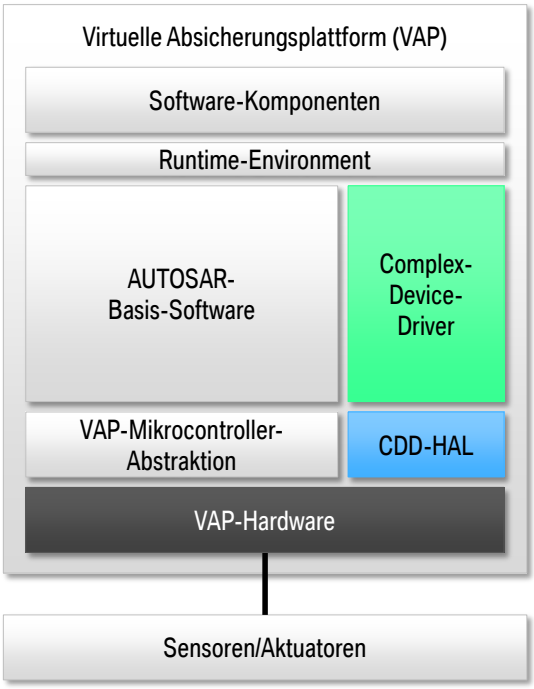


Abbildung 3.5: Anbindung des CDD an die verfügbare VAP-Hardware über einen Hardware-Abstraction-Layer.

Wie in Abbildung 3.5 dargestellt, wird dieser zusätzliche HAL zwischen dem CDD und der darunterliegenden Hardware eingezogen. Die Abstraktion der Hardware-Schnittstellen des Treibers ermöglicht so den Datenaustausch mit der Hardware der VAP. Dies schafft die Voraussetzung, um reale Sensoren und Aktuatoren über den CDD anzusteuern. Für die Anwendung dieses Konzepts ist es notwendig, dass die Daten, welche der Complex-Device-Driver über die Hardware-Schnittstellen austauschen muss, sinnvoll für die verfügbare Hardware der VAP konvertierbar sind. Gewisse Funktionen der eigentlichen Ziel-Hardware, die nicht über die Ersatzhardware auf der VAP abgebildet werden können, sind auch über

Software innerhalb des HAL simulierbar. Dies führt allerdings immer zu erhöhter Prozessorlast, was eine entsprechende Auslegung der eingesetzten VAP bedingt.

3.2.6 Ziel-Hardware-Emulation

Eine weitere Möglichkeit, mit einem Complex-Device-Driver auf einer VAP Sensoren und Aktuatoren anzusteuern, besteht in der Integration eines Field-Programmable-Gate-Array (FPGA). Wie bereits in Kapitel 3.2.4 aufgezeigt, existieren FPGAs in kompatiblen Bauformen für eine VAP. Über verschiedene Beschreibungssprachen, wie z. B. VHDL [IEE09] oder Verilog-HDL [IEE06], können vielfältige Funktionen modelliert werden. Die Abbildung auf dem FPGA erfolgt durch unterschiedliche Verschaltung der einzelnen logischen Elemente innerhalb des Arrays. Genauso lässt sich die Funktionalität der Hardware eines CDD modellieren. Die Voraussetzung dafür ist jedoch, dass ein passender FPGA verfügbar ist, der über alle logischen Elemente zur Realisierung der Funktion verfügt. Bei entsprechender Modellierung kann ein CDD auf äquivalente Schnittstellen zurückgreifen wie auf seiner Ziel-Hardware. Somit kann der Treiber ohne zusätzliche Anpassungen weiterer Software-Anteile integriert werden. Das Konzept ist in Abbildung 3.6 dargestellt.

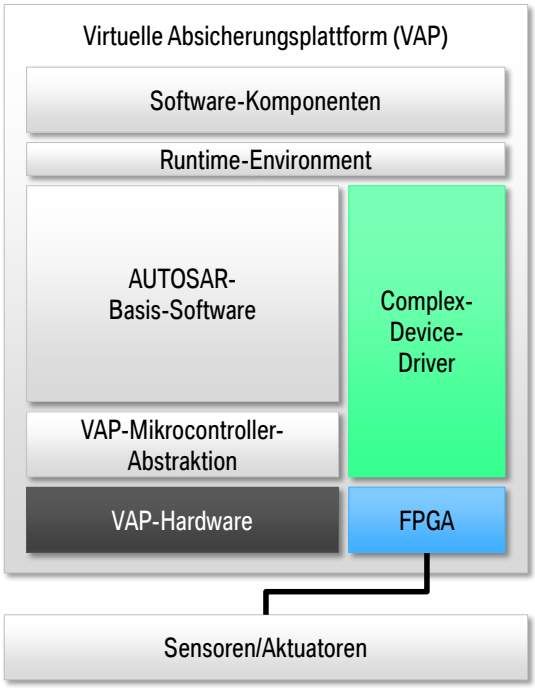


Abbildung 3.6: Integration eines FPGA zur Emulation der Ziel-Hardware.

Für die Modellierung der CDD-Hardware auf dem FPGA sind jedoch umfassende Kenntnisse über den Aufbau der Schaltung erforderlich, da jeder Bestandteil nachgebildet werden muss. Wenn die Funktion nicht zufällig auf einem FPGA entwickelt wurde, kann auch nicht auf fertige Bestandteile zurückgegriffen werden. Dadurch wird die Modellierung zu einer komplexen, zeitaufwändigen Aufgabe. Durch die vorgegebene Anzahl und Art an logischen Elementen auf einem FPGA ist die Komplexität der darstellbaren Schaltung per Definition begrenzt, weshalb sich besonders vielschichtige CDD-Hardware nicht abbilden lässt.

3.2.7 Separate Ausführung auf dem Ziel-Mikrocontroller

Complex-Device-Drivers mit Hardwareansteuerung sind Software-Anteile, die sich bei einem Wechsel der Hardware-Plattform nicht einfach wiederverwenden lassen. Daher bezieht sich das Testziel der Wiederverwendbarkeit aus Kapitel 3.1.2 hauptsächlich auf die Software-Komponenten oberhalb der RTE. Im Bezug auf die Absicherung der Kompatibilität zu anderen BSW-Stacks ist der Aspekt der Wiederverwendbarkeit jedoch auch für CDDs relevant. Ein Complex-Device-Driver muss dafür allerdings nicht zwingend auf einem hardwareunabhängigen System ausgeführt werden. Stattdessen ist, ausschließlich für den Treiber, die eigentliche Ziel-Plattform nutzbar. Die Funktions-Software des Steuergerätes wird weiterhin auf einer VAP ausgeführt, wobei die Wiederverwendbarkeit bewertbar bleibt. Abbildung 3.7 zeigt den Aufbau des Konzepts.

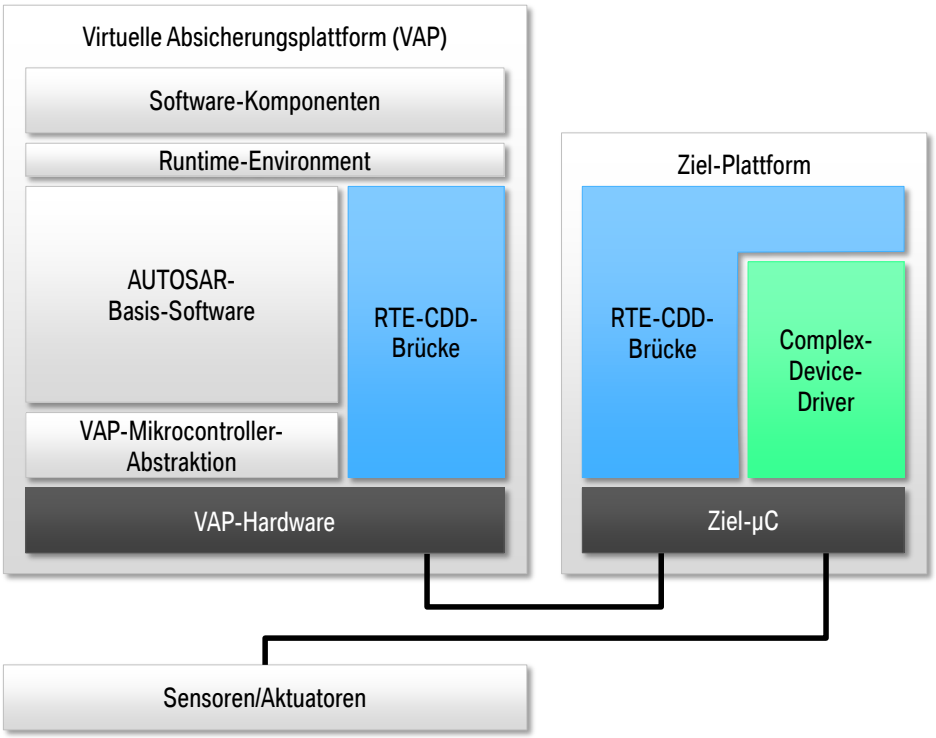


Abbildung 3.7: Ausführung des CDD auf seiner Ziel-Plattform mit Kopplung an eine VAP.

Die Herausforderung besteht dabei in der Kopplung der beiden Ausführungsplattformen. Dazu ist eine RTE-CDD-Brücke vorgesehen, welche die Synchronisation der Ausführung und die Datenübertragung zwischen dem CDD und den restlichen Software-Anteilen übernimmt. Die RTE-CDD-Brücke besteht aus zwei Anteilen, die in den jeweiligen Plattformen integriert werden. Beide Segmente bilden gegenüber der RTE bzw. dem CDD das jeweilige Gegenstück der Schnittstellen ab. Die Gegenkomponenten sind analog zu dem Konzept aus Kapitel 3.2.3 vollautomatisch erzeugbar, da die Schnittstellen in der AUTOSAR-Beschreibung vollständig definiert sein müssen. Die Übertragung der Daten zwischen beiden Anteilen der RTE-CDD-Brücke kann über eine der Kommunikationsmöglichkeiten erfolgen, die von beiden Plattformen unterstützt werden. Je nach nötiger Geschwindigkeit der Übertragung bieten sich dafür eine Ethernet-Verbindung (Kommunikation z. B. über UDP, TCP/IP) oder die Standard-

Bussysteme (CAN, LIN, FlexRay) an. Die Kopplung der beiden Plattformen über die RTE-CDD-Brücke wird allerdings nicht die Performance der Ziel-Architektur erreichen können. Die Verbindung zwischen VAP und Ziel-Plattform kann niemals mit so hoher Geschwindigkeit operieren, wie ein einfacher Zugriff auf die RTE. Für CDDs mit sehr zeitkritischem Verhalten ist dieser Ansatz somit nicht geeignet. Weiterhin muss die Ziel-Plattform bereits in einer passenden Form zur Verfügung stehen. Dies kann bei höherem *Frontloading* in den Software-Entwicklungsprozessen nicht gewährleistet werden.

3.2.8 Fazit

Die dargestellten Lösungsansätze zur Berücksichtigung von CDDs in der virtuellen Absicherung erfordern verschiedene Simulationsmethoden oder die Integration spezifischer Hardware-Komponenten. Abhängig von den Eigenschaften des Treibers und den gesetzten Testzielen ist jedes Konzept unterschiedlich gut für einen Anwendungsfall geeignet. Im folgenden Kapitel sollen durch einen detaillierten Vergleich die vielversprechendsten Lösungen ermittelt werden.

3.3 Vergleich der Konzepte

In Kapitel 3.2 wurden sieben verschiedene Konzepte vorgestellt, die grundsätzlich für die Berücksichtigung von Complex-Device-Drivers in der virtuellen Absicherung geeignet sind. Im Folgenden werden diese Ansätze miteinander verglichen, um anhand verschiedener Bewertungskriterien die besten Varianten zu identifizieren.

3.3.1 Bewertungskriterien

Um einen Vergleich der verschiedenen Konzepte zu ermöglichen, wurden gleichrangige Bewertungskriterien definiert, die eine Prognose der jeweiligen Praxistauglichkeit der Lösungen erlauben. Es folgt eine detaillierte Beschreibung der einzelnen Kriterien. Die Definition der zugehörigen Skalen für die Bewertung erfolgt in den dargestellten Tabellen.

- **Integrationsaufwand:** Die Integration von Steuergeräte-Software auf der VAP darf keinen hohen zusätzlichen Aufwand im Vergleich zur reinen Integration auf der Ziel-Hardware verursachen. Dies ist notwendig, um das Potential der virtuellen Absicherung zur Kostenreduktion optimal auszuschöpfen. Der Vorteil durch die Sicherstellung der Wiederverwendbarkeit der Funktions-Software sowie die frühere Entdeckung von Fehlern muss überwiegen. Dies gilt genauso bei der Berücksichtigung von Complex-Device-Drivers in der Plattform. Da die CDDs generell einen kleineren Anteil der Software darstellen, muss hierbei die Rentabilität besonders beachtet werden. Tabelle 3.1 stellt das Bewertungsschema dar.

Bewertung	**Symbol**	**Erläuterung**
Positiv	+	Das Konzept verfügt über Automatisierungsmechanismen zur Integration und erfordert nur geringe manuelle Eingriffe.
Neutral	0	Das Konzept erfordert manuellen Aufwand für die Integration. Dieser beschränkt sich jedoch auf die Konfiguration von generischen Anteilen.
Negativ	–	Das Konzept erfordert erheblichen manuellen Aufwand für die Integration.

Tabelle 3.1: Bewertungsschema für den Integrationsaufwand.

- **Anwendungsspektrum:** Damit ein Konzept für die Berücksichtigung von CDDs in der virtuellen Absicherung optimal einsetzbar ist, muss es für ein möglichst breites Spektrum von verschiedenen Treibern anwendbar sein. Lösungen die nur für einzelne Spezialfälle funktionieren sind weniger wirtschaftlich. Bevorzugt werden Konzepte, die generisch und flexibel auf unterschiedliche Treiber anwendbar sind. In Tabelle 3.2 wird das Bewertungsschema erläutert.

Bewertung	**Symbol**	**Erläuterung**
Positiv	+	Das Konzept kann für eine hohe Bandbreite an CDDs angewendet werden. Es ist entweder vollständig generisch ausgelegt oder erfordert keine zusätzlichen Elemente im Vergleich zur Ziel-Plattform des CDD.
Neutral	0	Das Konzept ist nicht vollständig generisch ausgelegt, kann jedoch für eine größere Menge an CDDs verwendet werden.
Negativ	–	Das Konzept ist nur für wenige, spezifische CDDs einsetzbar.

Tabelle 3.2: Bewertungsschema für das Anwendungsspektrum.

- **Testzielerreichbarkeit:** Ein weiterer Faktor bei der Konzeptauswahl ist die Erreichbarkeit der Testziele, wie sie in Kapitel 3.1 definiert wurden. Eine Lösung ist optimal geeignet, wenn alle Ziele damit abdeckbar sind. Konzepte, welche nicht mindestens eines der Testziele erreichen, werden nicht betrachtet, da sie für die Anwendung in der virtuellen Absicherung somit gänzlich ungeeignet sind. Tabelle 3.3 listet das Bewertungsschema auf.

Bewertung	**Symbol**	**Erläuterung**
Positiv	+	Alle drei definierten Testziele können mit dem Konzept erreicht werden.
Neutral	0	Zwei der drei definierten Testziele können mit dem Konzept erreicht werden.
Negativ	–	Eines der drei definierten Testziele kann mit dem Konzept erreicht werden.

Tabelle 3.3: Bewertungsschema für die Testzielerreichbarkeit.

- **Verfügbarkeit:** Die virtuelle Absicherung soll vor allem in der frühen Entwicklungsphase erfolgen, bevor die eigentliche Ziel-Hardware verfügbar ist. Somit müssen sämtliche benötigen Bestandteile für die Integration eines Complex-Device-Drivers bereits vorab zur Verfügung stehen. Konzepte die Anteile der Zielplattform benötigen erhalten daher einen Malus in der Bewertung. Das zugehörige Bewertungsschema kann aus Tabelle 3.4 entnommen werden.

Bewertung	Symbol	Erläuterung
Positiv	+	Für die Anwendung des Konzepts sind keine zusätzlichen Komponenten nötig, die nicht bereits in einer VAP enthalten sind.
Neutral	0	Für die Anwendung des Konzepts sind zusätzliche Komponenten nötig, die nicht bereits in einer VAP enthalten sind. Diese sind jedoch nicht unmittelbar an die Verfügbarkeit der Ziel-Plattform des CDD gekoppelt.
Negativ	–	Für die Anwendung des Konzepts sind zusätzliche Komponenten nötig, die nicht bereits in einer VAP enthalten sind. Diese sind unmittelbar an die Verfügbarkeit der Ziel-Plattform des CDD gekoppelt.

Tabelle 3.4: Bewertungsschema für die Verfügbarkeit.

3.3.2 Bewertung der Konzepte

Alle Konzepte aus Kapitel 3.2 werden im Folgenden anhand der Kriterien des vorherigen Kapitels bewertet. Die Einstufung der Merkmale für jedes Konzept erfolgt anhand der definierten Skala als positiv (+), neutral (0) oder negativ (–). Nach der Bewertung folgt eine Zusammenfassung und Auswertung der Ergebnisse, um die Konzepte zu identifizieren, die für den praktischen Einsatz in der virtuellen Absicherung vorzuziehen sind.

Simulation des Ziel-Mikrocontrollers

- **Integrationsaufwand (–):** Die Inbetriebnahme einer Mikrocontroller-Simulation ist, trotz diverser Assistenten in den zugehörigen Werkzeugen, eine zeitaufwändige Aufgabe. Die eigentliche CDD-Integration verhält sich wiederum identisch zur Ziel-Hardware. Da der Aufwand für die grundsätzliche Inbetriebnahme des Systems die eigentliche Integration jedoch deutlich übersteigt, muss dies in die Aufwandsbetrachtung mit einbezogen und als negativ bewertet werden.

- **Anwendungsspektrum (0):** Grundsätzlich können in der Mikrocontroller-Simulation jegliche CDDs integriert werden, die für die zugehörige Architektur ausgelegt sind. Die Treiber greifen im fertigen Steuergerät jedoch meist auf weitere Sensoren oder Aktuatoren zu, die nicht Bestandteil des Mikrocontrollers sind. Daher kann nicht lückenlos gewährleistet werden, dass die Peripherie in der Simulation unterstützt wird und somit der volle CDD-Funktionsumfang erprobt werden kann. Dies führt zu einer neutralen Bewertung.

- **Testzielerreichbarkeit (0):** Die Mikrocontroller-Simulation ermöglicht die Integration der vollständigen Software auf einer virtuellen Absicherungsplattform. Ebenso kann die Wiederverwendbarkeit sichergestellt werden, da lediglich der CDD innerhalb der Ziel-Architektur ausgeführt wird und nicht die Software-Komponenten. Durch die fehlende Echtzeitfähigkeit der Simulation und ohne die Möglichkeit reale Sensoren und Aktuatoren anzuschließen, ist die funktionale Absicherung der Software jedoch nicht

vollständig möglich. Die Erreichbarkeit von zwei der drei Testziele ergibt eine neutrale Wertung.

- **Verfügbarkeit (0):** Einige Mikrocontroller-Hersteller entwickeln ihre Produkte basierend auf Modellen, die genauso für die Simulation verwendbar sind. In diesem Fall ist die Verfügbarkeit lange vor der Chip-Produktion gewährleistet. Dies gilt allerdings nicht ausnahmslos für alle Hersteller, wodurch nicht alle Plattformen abgebildet werden können. Deshalb fällt die Bewertung neutral aus.

Simulation des Hardware-Verhaltens

- **Integrationsaufwand (−):** Die Schnittstellen jedes CDD zur Hardware sind in der AUTOSAR-Beschreibung nicht definiert und können daher nicht über generische Verfahren analysiert werden. Für die Erzeugung einer Hardware-Simulation ist somit immer manueller Aufwand nötig, um die Schnittstellen von Simulation und Treiber aufeinander anzupassen. Dies gilt auch, wenn es sich nur um eine einfache Eingangs-/Ausgangssimulation handelt. Daher entsteht ein relativ hoher Integrationsaufwand, der zu einer negativen Bewertung führt.

- **Anwendungsspektrum (+):** Bei jedem CDD mit Hardware-Ansteuerung kann über dieselben Schnittstellen genauso eine Simulation angebunden werden. Dabei ist es immer möglich, zumindest eine einfache Eingangs-/Ausgangssimulation zu erzeugen. Somit ist die Simulation des Hardware-Verhaltens grundsätzlich für jeden CDD möglich, wodurch die Wertung positiv ausfällt.

- **Testzielerreichbarkeit (0):** Die Simulation verhindert die Ansteuerung von realer Hardware. Weiterhin kann die Simulation oftmals nur ein idealisiertes Verhalten abbilden. Dies führt zu Einschränkungen in der funktionalen Absicherung des vollständigen Funktionsumfangs und damit zu einer neutralen Einstufung, da nur zwei Testziele erreichbar sind.

- **Verfügbarkeit (+):** Für die Simulation werden keine zusätzlichen Komponenten benötigt, die erst zu einem späteren Zeitpunkt in der Entwicklung verfügbar wären. Daher ist die Erzeugung der Simulation problemlos in der frühen Entwicklungsphase möglich, wodurch sich eine positive Wertung ergibt.

CDD-Simulation

- **Integrationsaufwand (+):** Aufgrund der Verfügbarkeit der Definition der Schnittstellen, zwischen den Software-Komponenten und einem CDD in der AUTOSAR-Beschreibung, ist die Erzeugung der Simulation des CDD-Verhaltens vollständig automatisierbar. Dies ermöglicht die Integration mit minimalem Aufwand und führt damit zu einer positiven Bewertung.

- **Anwendungsspektrum (+):** Die Simulation erfolgt auf Basis der RTE-Schnittstellen des CDD. Diese müssen für jeden Treiber eindeutig definiert sein und dürfen nur die vorgegebenen Kommunikationsmechanismen der AUTOSAR-Spezifikation (siehe Kapitel 2.1.4) nutzen, um eine Integration zu ermöglichen. Daher ist das Konzept

universell einsetzbar und für jegliche Arten von CDDs geeignet. Somit ergibt sich ebenfalls eine positive Einstufung.

- **Testzielerreichbarkeit (−):** Da der CDD durch eine Simulation ersetzt und somit nicht integriert wird, kann er nicht getestet werden. Lediglich das Testziel zur Sicherstellung der Integrierbarkeit der Software-Komponenten kann abgedeckt werden. Dadurch ergibt sich eine negative Wertung für die Testzielerreichbarkeit.

- **Verfügbarkeit (+):** Für die Erzeugung der Simulation sind keine zusätzlichen Komponenten nötig. Sobald die Software-Komponenten vorliegen und die Schnittstellen zu dem jeweiligen CDD definiert sind, ist das Konzept bereits einsetzbar. Daraus leitet sich eine positive Wertung ab.

Integration der spezifischen Hardware

- **Integrationsaufwand (0):** Die Integration der zugehörigen Hardware des CDD erlaubt einen identischen Integrationsprozess wie bei der Ziel-Plattform. Es existieren allerdings keine zusätzlichen Automatisierungsmechanismen, die die Integration erleichtern. Somit wird dieses Kriterium neutral gewertet.

- **Anwendungsspektrum (−):** Nur wenige spezifische Hardware-Komponenten von CDDs werden in einer Form hergestellt, die sie zu Standard-PC-Systemen kompatibel gestalten. Bereits die Tatsache, dass ein CDD zur Ansteuerung einer Komponente nötig ist, lässt bereits auf ein limitiertes Einsatzgebiet schließen, weshalb unterschiedliche Bauformen der Hardware unwahrscheinlich sind. Dadurch ist das Konzept nur auf ein limitiertes Spektrum von CDDs anwendbar, woraus sich eine negative Wertung ergibt.

- **Testzielerreichbarkeit (+):** Da die zugehörige Original-Hardware eines CDD in diesem Konzept direkt eingesetzt wird, entsteht keine Unterscheidung in der Testzielerreichbarkeit im Vergleich zur Ziel-Plattform. Der Treiber ist vollständig integrierbar und erlaubt die Ansteuerung realer Peripherie. Dies ergibt eine positive Einstufung.

- **Verfügbarkeit (−):** Für den Einsatz der CDD-Hardware muss sie in einer Bauform vorliegen, die zu einer VAP kompatibel ist. Dies ist aufgrund der meist geringen Verbreitung eines CDD und somit auch der zugehörigen Hardware nicht zuverlässig zu gewährleisten. Daraus folgt eine negative Bewertung der Verfügbarkeit.

CDD-Hardware-Abstraction-Layer

- **Integrationsaufwand (0):** Die Integration des CDD-HAL erfordert eine Konfiguration auf die Schnittstellen des Treibers zur Hardware, welche nicht automatisierbar ist. Allerdings ist der HAL vollständig generisch ausgelegt und stellt eine vordefinierte Auswahl an Parametern zur Verfügung. Anhand der Konfiguration wird der Quellcode generiert, der die Integration der Schicht in das Gesamtsystem erlaubt. Dies stellt eine Vereinfachung gegenüber der manuellen Programmierung des HAL dar. Durch die generische Auslegung ergibt sich eine neutrale Bewertung für den Integrationsaufwand.

- **Anwendungsspektrum (0):** Die generische Konzeption des HAL erlaubt die Anwendung auf ein breites Spektrum an Complex-Device-Drivers. Eine Limitierung entsteht

jedoch durch die verfügbare Hardware einer VAP. Die Schnittstellen müssen zwar nicht direkt kompatibel sein, die Umleitung der Daten des Treibers auf die VAP-Hardware ist dennoch nicht für alle Anwendungsfälle gewährleistbar. Daher ergibt sich eine neutrale Wertung für das Anwendungsspektrum.

- **Testzielerreichbarkeit (+):** Das Konzept erlaubt die vollständige Integration des CDD. Durch die Anbindung an die Hardware einer VAP entsteht ebenso die Möglichkeit zur Ansteuerung von realer Peripherie. Dies erlaubt die vollständige Abdeckung der definierten Testziele und führt zu einer positiven Einstufung.
- **Verfügbarkeit (+):** Für das Konzept werden ausschließlich Hardware-Komponenten in einer VAP verwendet, die nicht an die Verfügbarkeit der Ziel-Hardware gebunden sind. Ebenso steht der HAL bereits ohne Einschränkungen in der frühen Entwicklungsphase zur Verfügung. Daraus ergibt sich eine positive Beurteilung für dieses Kriterium.

Ziel-Hardware-Emulation

- **Integrationsaufwand (−):** Für die Emulation auf einem FPGA ist die vollständige Modellierung der Ziel-Hardware, mit allen benötigten Komponenten, eine zwingende Voraussetzung. Je nach Funktion kann sich der Prozess dabei sehr aufwändig gestalten. Die Integration des Treibers verhält sich wiederum analog zur Ziel-Plattform, wenn die Schnittstellen auf dem FPGA korrekt nachgebildet wurden. Da die Modellierung jedoch einen hohen Zusatzaufwand erfordert, ergibt sich insgesamt eine negative Bewertung.
- **Anwendungsspektrum (0):** Das Konzept erlaubt die Nutzung für eine große Bandbreite von CDDs, da aktuelle FPGAs über eine Vielzahl an Logikblöcken verfügen. Darüber können unterschiedlichste Funktionen abgebildet werden, die der Hardware eines CDD nachempfunden sind. Dennoch sind auch die Möglichkeiten von aktuellen FPGAs begrenzt und nicht jede CDD-Hardware kann optimal modelliert werden. Daraus ergibt sich eine neutrale Beurteilung des Anwendungsspektrums.
- **Testzielerreichbarkeit (+):** Das Konzept erlaubt die vollständige Integration des CDD und damit genauso der Software-Komponenten. Über den FPGA ist die Ansteuerung von Sensoren und Aktuatoren möglich, wodurch die vollständige funktionale Absicherung der Software sichergestellt werden kann. Der Wiederverwendbarkeitsaspekt lässt sich ebenso betrachten, weshalb die Bewertung der Testzielerreichbarkeit positiv ausfällt.
- **Verfügbarkeit (0):** Ein FPGA zählt nicht zum Standardumfang einer VAP, der für den Einsatz regulärer Steuergeräte-Software nötig wäre. Es handelt sich somit um eine zusätzliche Komponente. Grundsätzlich sind FPGAs unabhängig von der jeweiligen Ziel-Plattform eines CDD verfügbar. Allerdings kann nicht garantiert werden, dass eine Bauform mit der nötigen Konfiguration an Logikblöcken für den jeweiligen Einsatz zur Verfügung steht. Daraus ergibt sich eine neutrale Wertung für die Verfügbarkeit.

Separate Ausführung auf dem Ziel-Mikrocontroller

- **Integrationsaufwand (0):** Zunächst erfolgt die Integration des CDD auf dem Ziel-Mikrocontroller identisch zum Prozess für die gesamte Software im Steuergerät. Da

das AUTOSAR-Betriebssystem bei diesem Konzept auf einer VAP läuft und nicht auf dem Ziel-Mikrocontroller, muss für die Ausführung des CDD, wie z. B. das Scheduling und die Synchronisation der Systeme, zusätzlicher Aufwand investiert werden. Die Konfiguration der RTE-CDD-Brücke kann teilweise automatisiert erfolgen. Je nach Art der gewünschten Datenübertragung über Bussysteme sind jedoch weiterhin manuelle Konfigurationen nötig, um z. B. die Übertragung eines CAN-Signals vollständig zu ermöglichen. Insgesamt ergibt sich daraus eine neutrale Bewertung des Integrationsaufwandes.

- **Anwendungsspektrum (0):** Durch den Einsatz der Ziel-Hardware für die Ausführung des CDD sind der Anwendung des Konzepts zunächst keine Grenzen gesetzt. Der Datenaustausch zwischen dem Treiber und der restlichen Software über die RTE-CDD-Brücke verhindert jedoch einen Einsatz bei hohen Echtzeitanforderungen. Daher fällt die Wertung für das Anwendungsspektrum neutral aus.

- **Testzielerreichbarkeit (+):** Das Konzept erlaubt die vollständige Erreichbarkeit der Testziele. Der CDD kann integriert und funktional mit Peripherieansteuerung getestet werden. Genauso ist die Integration und Absicherung der Wiederverwendbarkeit der Software-Komponenten möglich. Daraus ergibt sich eine positive Beurteilung.

- **Verfügbarkeit (–):** Ohne eine bereits vorhandene Version der Ziel-Plattform ist das Konzepts nicht anwendbar. Somit kann in der frühen Entwicklungsphase die Einsatzbereitschaft nicht gewährleistet werden. Diese direkte Kopplung der Verfügbarkeit an die Serien-Hardware führt zu einer negativen Einstufung in diesem Kriterium.

3.3.3 Auswertung

Die Wertungen in den einzelnen Kategorien aus dem obigen Kapitel ergeben eine Gesamtbewertung für jedes Konzept. Dabei führt jede negative Bewertung (–) zum Abzug eines Punktes und jede positive (+) fügt einen Punkt hinzu. Die Ausgangsbasis ist immer eine Punktzahl von null. Neutrale Bewertungen (0) wirken sich nicht auf das Gesamtergebnis aus. Tabelle 3.5 aggregiert alle Einzelwertungen und stellt in der letzten Spalte das Gesamtergebnis jedes Konzepts dar.
Die Auswertung zeigt, dass keiner der Ansätze gleichzeitig alle definierten Testziele erreichen kann und dennoch auf ein breites Spektrum an Complex-Device-Drivers mit minimalem Aufwand und bei maximaler Verfügbarkeit anwendbar ist. Allerdings haben die Konzepte individuelle Vorzüge, die je nach Einsatz für unterschiedliche CDDs und Testziele zum gewünschten Ergebnis führen. Die höchste Wertung erreichen die Konzepte der vollständigen CDD-Simulation und des CDD-Hardware-Abstraction-Layers.
Für die CDD-Simulation wirkt sich vor allem der hohe Automatisierungsgrad positiv aus. Nur die Testzielerreichbarkeit stellt eine Einschränkung dar. Allerdings ist das erreichbare Testziel, die Integration und Absicherung der Software-Komponenten, die Grundvoraussetzung für den sinnvollen Einsatz einer VAP und üblicherweise der erste Meilenstein in der Anwendung des Systems. Steht der CDD selbst nicht zur Verfügung, ist dies zudem das einzig nutzbare Konzept.
Das Konzept des CDD-HAL leistet eine optimale Testzielerreichbarkeit aufgrund der vollständigen Integration des CDD. Es ergänzt das Konzept zur CDD-Simulation in den fortge-

	Integrationsaufwand	Anwendungsspektrum	Testzielerreichbarkeit	Verfügbarkeit	Gesamtwertung
Simulation des Ziel-Mikrocontrollers	–	0	0	0	–
Simulation des Hardware-Verhaltens	–	+	0	+	+
CDD-Simulation	+	+	–	+	++
Integration der spezifischen Hardware	0	–	+	–	–
CDD-Hardware-Abstraction-Layer	0	0	+	+	++
Ziel-Hardware-Emulation	–	0	+	0	0
Separate Ausführung auf dem Ziel-Mikrocontroller	0	0	+	–	0

Tabelle 3.5: Zusammenfassung der Bewertung der Konzepte.

schrittenen Entwicklungsphasen. Die Verfügbarkeit ist sichergestellt, da keine zusätzlichen Komponenten zu einem Standardausbau einer VAP nötig sind.

3.4 Konzeptauswahl und Methodik für die CDD-Integration

Die Bewertung der Konzepte aus dem vorherigen Kapitel ergibt einen Richtwert dafür, wie gut die Lösungen für den praktischen Einsatz geeignet sind. Bei der Auswahl der Konzepte für die Realisierung ist insgesamt eine vollständige Abdeckung aller Anwendungsfälle für CDDs in der virtuellen Absicherung nötig. Die verschiedenen Szenarien werden von zwei Faktoren bestimmt:

- **CDD-Anforderungen:** Eine wichtige Rolle für die Konzeptauswahl spielen die Hardware-Schnittstellen des Complex-Device-Drivers. Dabei ist relevant, ob ein Treiber auf einer VAP ohne zusätzliche Hardware-Komponenten integrierbar ist. Wie bereits in Kapitel 2.2.2 dargestellt, verfügen 41 % der untersuchten CDDs über keine direkten Hardware-Schnittstellen. Diese Treiber sind unproblematisch für die virtuelle Absicherung, da keine spezielle Hardwareabhängigkeit besteht. Für die anderen 59 % der Complex-Device-Drivers mit direkter Hardware-Ansteuerung wurde im Rahmen dieser Arbeit die Tauglichkeit der VAP-Hardware (siehe Kapitel 2.5.3) für die Integration analysiert. Dabei hat sich gezeigt, dass 76 % der untersuchten CDDs mit den Standard-Hardware-Modulen einer VAP abgebildet werden können.

- **Testziele:** Im Kapitel 3.1 wurden bereits drei Testziele für die Berücksichtigung von CDDs in der virtuellen Absicherung definiert. Keines der vorgestellten Konzepte kann alleine alle Testziele abdecken und gleichzeitig die weiteren Bewertungskriterien aus Kapitel 3.3.1 optimal abdecken. Daher ist zu beachten, dass die ausgewählten Konzepte zusammen alle Testziele bestmöglich erfüllen können.

Anhand der obigen Faktoren und Konzeptbewertungen wurde eine allgemeingültige Methodik für die Auswahl des jeweils besten Konzepts in einem bestimmten Anwendungsfall entwickelt. Aus den verschiedenen Kombinationsmöglichkeiten der Testziele und der CDD-Anforderungen ergeben sich insgesamt neun verschiedene Anwendungsfälle, die eine Auswahl-Matrix für die Konzepte aufspannen (siehe Abbildung 3.8). Für jeden der Anwendungsfälle wird das Konzept mit der höchsten Gesamtbewertung ausgewählt, welches sowohl mit den CDD-Anforderungen kompatibel ist, als auch das zugehörige Testziel erfüllen kann. Die Nummerierung der Anwendungsfälle erfolgt nach dem Schema $A_{\alpha\vartheta}$, wobei α den Index für die drei Varianten der CDD-Anforderungen und ϑ die Nummer des Testziels darstellt (siehe Tabelle 3.6).

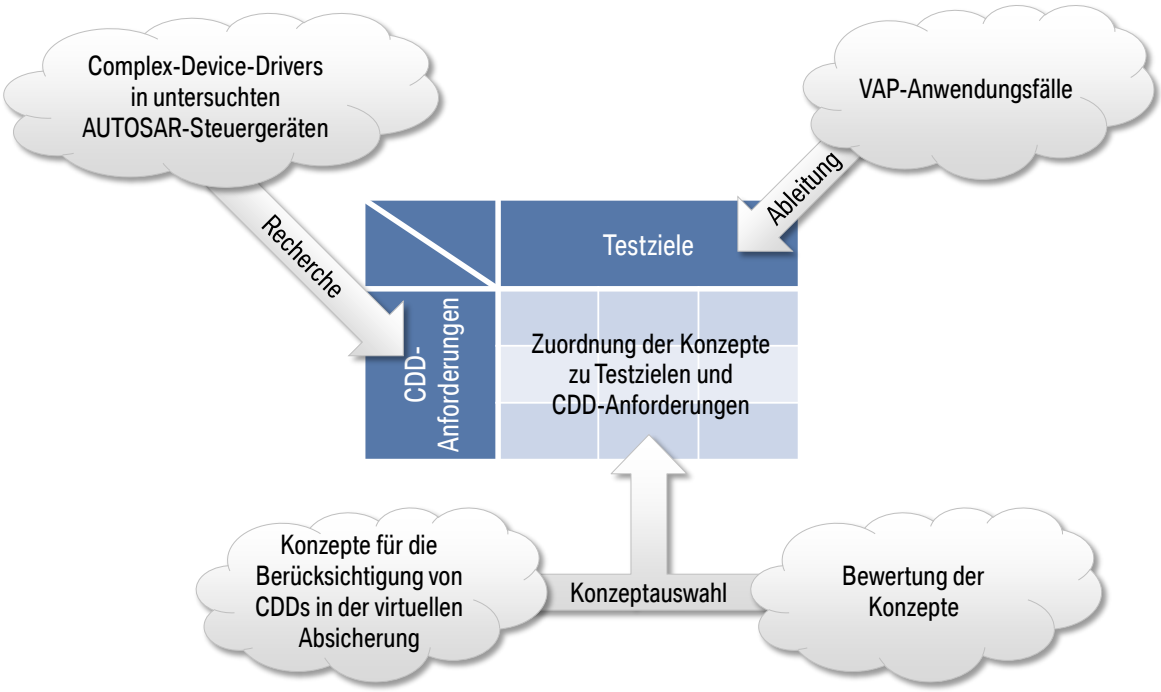


Abbildung 3.8: Auswahlverfahren für die Konzepte zur Berücksichtigung von CDDs in der virtuellen Absicherung.

CDD-Anforderungen		**Testziele**	
$\alpha = 1$:	Hardware-Schnittstellen vorhanden und VAP-Hardware geeignet	$\vartheta = 1$:	Integration der Software-Komponenten
$\alpha = 2$:	Hardware-Schnittstellen vorhanden und VAP-Hardware nicht geeignet	$\vartheta = 2$:	Sicherstellung der Wiederverwendbarkeit der Funktions-Software
$\alpha = 3$:	Keine Hardware-Schnittstellen vorhanden	$\vartheta = 3$:	Funktionale Absicherung der Software

Tabelle 3.6: Zuordnung der Indizes für Anwendungsfälle von CDDs in der virtuellen Absicherung.

Für die Anwendungsfälle A_{31}, A_{32} und A_{33}, bei denen der CDD über keine besonderen Hardwareabhängigkeiten verfügt, ist eine direkte Integration des Treibers realisierbar. Hierbei werden lediglich die Möglichkeiten des CDD für den Zugriff auf BSW-Module genutzt. Da die Integration der Original-Komponenten grundsätzlich die optimale Lösung darstellt, ist hierbei keines der entwickelten Konzepte notwendig.
Die höchste Gesamtbewertung haben die Konzepte der CDD-Simulation und des CDD-Hardware-Abstraction-Layers erhalten. Aufgrund des identischen Ergebnisses werden die Konzepte in ihren Einzelwertungen direkt verglichen. Dabei ist zu erkennen, dass die CDD-Simulation die höchste Wertung für den Integrationsaufwand, das Anwendungsspektrum und die Verfügbarkeit erreicht. Sofern nur das Testziel zur Integration der Software-Komponenten gefordert ist, stellt die CDD-Simulation somit die optimale Lösung dar. Nach der definierten Methodik wird das Konzept somit für die Anwendungsfälle A_{11} und A_{21} selektiert, welche von der Einschränkung in der Testzielerreichbarkeit nicht betroffen sind.
Der CDD-Hardware-Abstraction-Layer ermöglicht die Anwendung für alle Testziele und hat ebenso die höchste Gesamtwertung erhalten. Die Hardware der VAP muss jedoch kompatibel zum Treiber sein, wodurch das Konzept nicht für alle übrigen Anwendungsfälle einsetzbar ist. Daher wird der CDD-HAL für die geeigneten Anwendungsfälle A_{12} und A_{13} ausgewählt.
Den dritten Platz in der Rangfolge belegt das Konzept zur Simulation des Hardware-Verhaltens. Im Einklang mit der zugehörigen Testzielerreichbarkeit wird die Hardware-Simulation für den Anwendungsfall A_{22} selektiert.
Den vierten Platz in der Rangfolge belegen zwei Konzepte mit identischer Gesamtwertung. Im direkten Vergleich der Lösungen wird das Konzept zur Ziel-Hardware-Emulation bevorzugt, da es durch eine höhere Verfügbarkeit den Einsatz in der frühen Entwicklungsphase besser sicherstellen kann. Im Einklang mit der zugehörigen Testzielerreichbarkeit wird die Ziel-Hardware-Emulation für den Anwendungsfall A_{23} ausgewählt.
Aus den Konzept-Zuordnungen ergibt sich die in Abbildung 3.9 dargestellte, allgemeingültige Methodik zur Auswahl des jeweils optimal geeigneten Konzepts für jeden Anwendungsfall zur Berücksichtigung von CDDs in der virtuellen Absicherung. In den linken beiden Spalten sind zusätzlich die Häufigkeiten der jeweiligen Varianten der CDD-Anforderungen bei den untersuchten Treibern dargestellt.
Die Konzepte der CDD-Simulation und des CDD-HAL ermöglichen zusammen die Abdeckung der Testziele und können auf mehr als 76 % der untersuchten CDDs mit direkten Hardware-Schnittstellen angewendet werden. Daher konzentriert sich diese Arbeit im weiteren Verlauf auf diese beiden Konzepte als repräsentative Lösungsansätze zur Berücksichtigung von Complex-Device-Drivers in der virtuellen Absicherung.

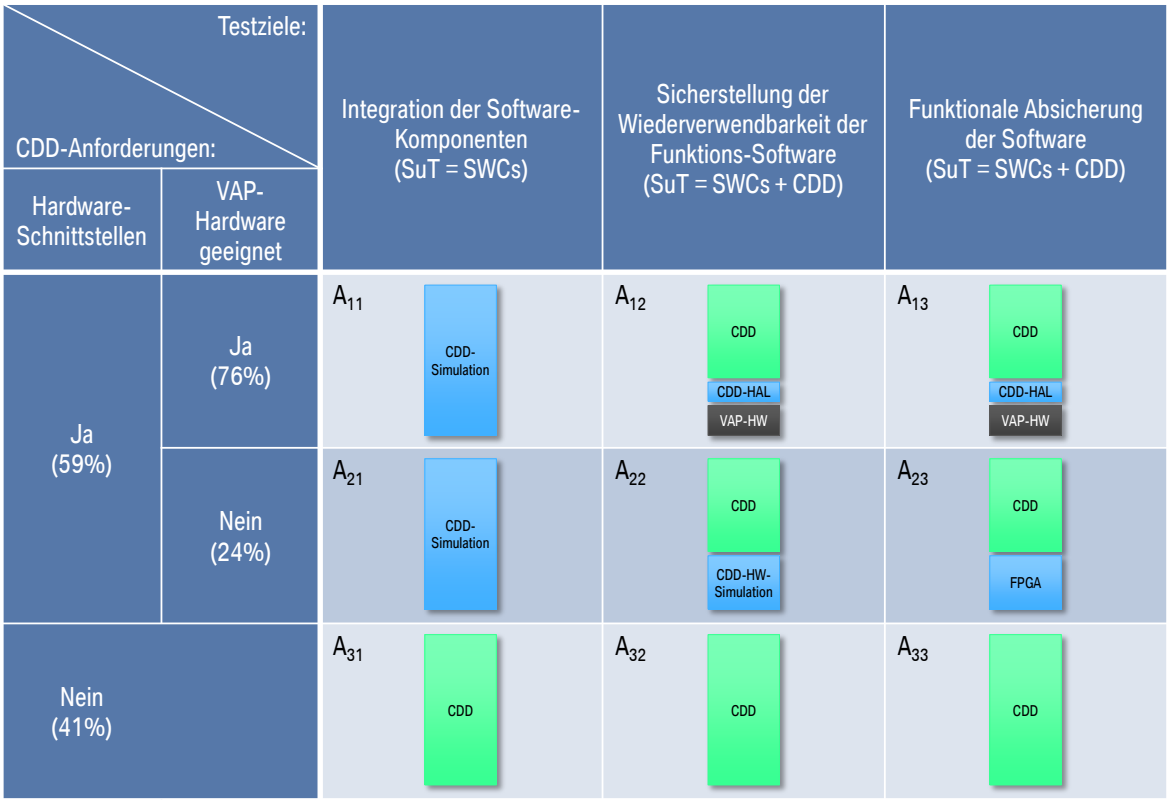


Abbildung 3.9: Methodik zur Auswahl des optimal geeigneten Konzepts für jeden Anwendungsfall zur Berücksichtigung eines CDD in der virtuellen Absicherung.

3.5 Zusammenfassung

Complex-Device-Drivers müssen in einem ganzheitlichen Ansatz zur virtuellen Absicherung der vollständigen Steuergeräte-Software berücksichtigt werden. Bei der Integration von Software-Komponenten ist die Testbarkeit der Schnittstellen zu einem CDD notwendig. Die Wiederverwendbarkeit der Software muss inklusive der Treiber vollständig analysierbar sein. Die funktionale Absicherung stellt ein weiteres wichtiges Testziel dar, welches auch mit CDDs in der Software auf einer VAP erreichbar bleiben muss.
Die im Rahmen dieser Arbeit entwickelten Konzepte ermöglichen mit unterschiedlichen Methoden die Erreichbarkeit der obigen Testziele. Dabei sind Simulationsverfahren berücksichtigt sowie Hardware-Lösungen, die eine Ansteuerung von Steuergeräte-Peripherie durch den CDD auf einer VAP erlauben.
Für die Anwendung der Konzepte in der Serienabsicherung ist nicht nur die Testzielerreichbarkeit relevant. Zusätzlich ist ein möglichst geringer Integrationsaufwand erforderlich sowie die Anwendbarkeit auf ein breites Spektrum von CDDs. Die Verfügbarkeit aller nötigen Komponenten muss in der frühen Entwicklungsphase ebenso sichergestellt sein. Anhand dieser Kriterien wurden alle Grobkonzepte bewertet.
Keines der Konzepte kann gleichzeitig alle Testziele für jegliche Eigenschaften von CDDs erfüllen. Daher wurde eine allgemeingültige Methodik definiert, welche die Auswahl des jeweils optimal geeigneten Konzepts für einen spezifischen Anwendungsfall erlaubt. Die Konzepte zur CDD-Simulation sowie zum Einsatz eines CDD-Hardware-Abstraction-Layers ermöglichen zusammen die vollständige Abdeckung der Testziele und sind für mehr als 76 % der untersuchten CDDs mit direkten Hardware-Schnittstellen einsetzbar. Daher konzentriert sich diese Arbeit im weiteren Verlauf auf diese beiden Konzepte.

Kapitel 4

Detaildarstellung der ausgewählten Konzepte

Bei den vorgestellten Konzepten aus Kapitel 3.2 handelt es sich um abstrahierte Darstellungen verschiedenster Lösungsansätze für die Problemstellung dieser Arbeit. Um eine Umsetzung anhand realer Steuergeräte-Software zu ermöglichen, ist daher eine detailliertere Ausarbeitung nötig. Diese soll für die zuvor selektierten Konzepte im folgenden Kapitel dargestellt werden.

4.1 CDD-Simulation

Wie bereits in Kapitel 3.2.3 beschrieben, besteht die Möglichkeit zur vollständigen Simulation des Verhaltens eines Complex-Device-Drivers. Dies ermöglicht die virtuelle Integration und Absicherung der Software-Komponenten eines Steuergerätes, ohne dass der CDD verfügbar sein muss. Das im Rahmen dieser Arbeit entwickelte Konzept wurde bereits in [DHM12a] sowie [DHM13] veröffentlicht und soll im Folgenden detailliert erläutert werden.

4.1.1 Automatisierte Erzeugung der Simulationskomponenten

Ist der Treiber selbst nicht vorhanden, da z. B. die Fertigstellung noch aussteht oder der Quellcode zum Schutz geistigen Eigentums unzugänglich ist, muss zunächst eine AUTOSAR-Beschreibung erzeugt werden. Dieser Rahmen für die CDD-Simulation muss das exakte Gegenstück der Schnittstellen der Funktions-Software abbilden. Die AUTOSAR-Beschreibung der Software-Komponenten muss per Definition bereits die vollständige Darstellung aller Schnittstellen enthalten. Somit kann auf diese Informationen zurückgegriffen werden, um die CDD-Simulation zu erzeugen. Eine *Software-Composition*, welche die Verbindungen aller Software-Komponenten untereinander beschreibt, kann automatisiert nach offenen Schnittstellen ohne Verbindung untersucht werden. Wenn die Software abgesehen vom CDD vollständig ist, können daraus direkt die Schnittstellen zwischen Treiber und Software-Komponenten abgeleitet werden. Für die CDD-Simulation kann somit automatisiert ein *Port* mit einer identischen *Interface-Definition* und gespiegelter Datentransferrichtung erzeugt werden. Die Kompatibilität der Schnittstellen ist direkt sichergestellt. Abbildung 4.1 zeigt das Ergebnis des Vorgangs anhand eines einfachen Beispiels mit zwei *Software-Components (SWCs)*.

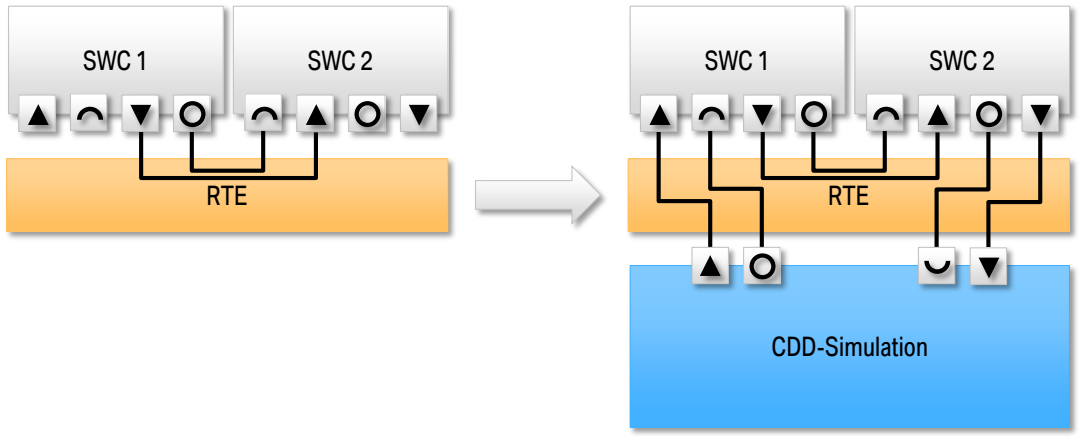


Abbildung 4.1: Erzeugung der AUTOSAR-Beschreibung für eine CDD-Simulation.

Für eine vollständige AUTOSAR-Beschreibung ist noch ein zugehöriges *Internal-Behavior* nötig. Dieses lässt sich ebenfalls automatisiert erzeugen. Zur Vereinfachung der Komplexität werden in diesem Konzept alle *Events* der definierten *Sender-Receiver-Ports* und der *Client-Ports* einem einzelnen *Runnable* zugewiesen. Für *Server-Ports* müssen per Definition jeweils eigene *Runnables* generiert werden.

Die Struktur der erzeugten *Runnables* muss sich direkt im zugehörigen C-Code jeder Software-Komponente widerspiegeln. Dies gilt genauso für die CDD-Simulation, weshalb eine Codegenerierung ebenfalls möglich ist. Anhand des *Interal-Behaviors* kann direkt das Skelett des zugehörigen Codes erzeugt werden, indem für jedes *Runnable* eine gleichnamige Funktion mit den zugehörigen Parametern angelegt wird. Darin können die *RTE-Calls* für das Senden und Empfangen von Daten über die jeweiligen *Ports* eingefügt werden. Durch eine Vorbelegung mit Standardwerten ist der C-Code bereits kompilierbar und die Integration der CDD-Simulation in die restliche Steuergeräte-Software mit minimalem Aufwand durchführbar. In Abbildung 4.2 werden die beschriebenen Vorgänge anhand des vorherigen Beispiels visualisiert.

4.1.2 CDD-Simulation mit Benutzerinteraktion

Das bisher erzeugte Grundgerüst für die CDD-Simulation enthält noch keine Funktionslogik, sondern erlaubt lediglich die Übertragung von Standardwerten. So kann zwar die Integrierbarkeit der Software-Komponenten geprüft werden, aber nicht die eigentliche Funktionalität. Um die Simulation verschiedener Eingangs- und Ausgangswerte für die Schnittstellen zu ermöglichen, kann automatisiert eine graphische Oberfläche zur Nutzerinteraktion generiert werden. Aus den bereits gesammelten Daten über alle Schnittstellen des CDD wird für jedes Datenelement eine Schaltfläche erzeugt. Die Benutzeroberfläche kann, während der Ausführung der gesamten Software auf einer VAP, dynamisch zugeschaltet werden. Die Verbindung zur CDD-Simulation erfolgt über das User-Datagram-Protocol (UDP) [Pos80], um eine performante und flexible Datenübertragung zu gewährleisten. Daher wird die graphische Oberfläche im Folgenden als UDP-Panel bezeichnet. Abbildung 4.3 zeigt die Kopplung des Panels an das vorherige Beispiel zur CDD-Simulation.

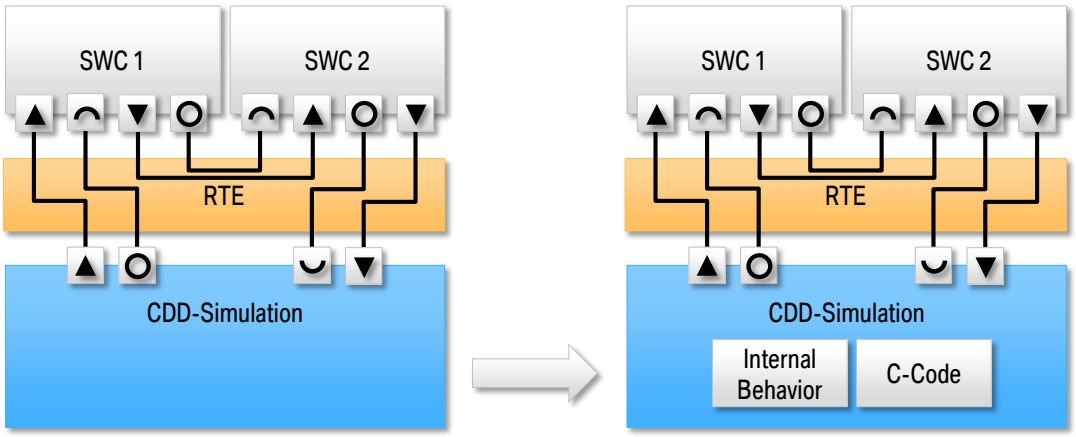


Abbildung 4.2: Vervollständigung der AUTOSAR-Beschreibung und Source-Code-Erzeugung für die CDD-Simulation.

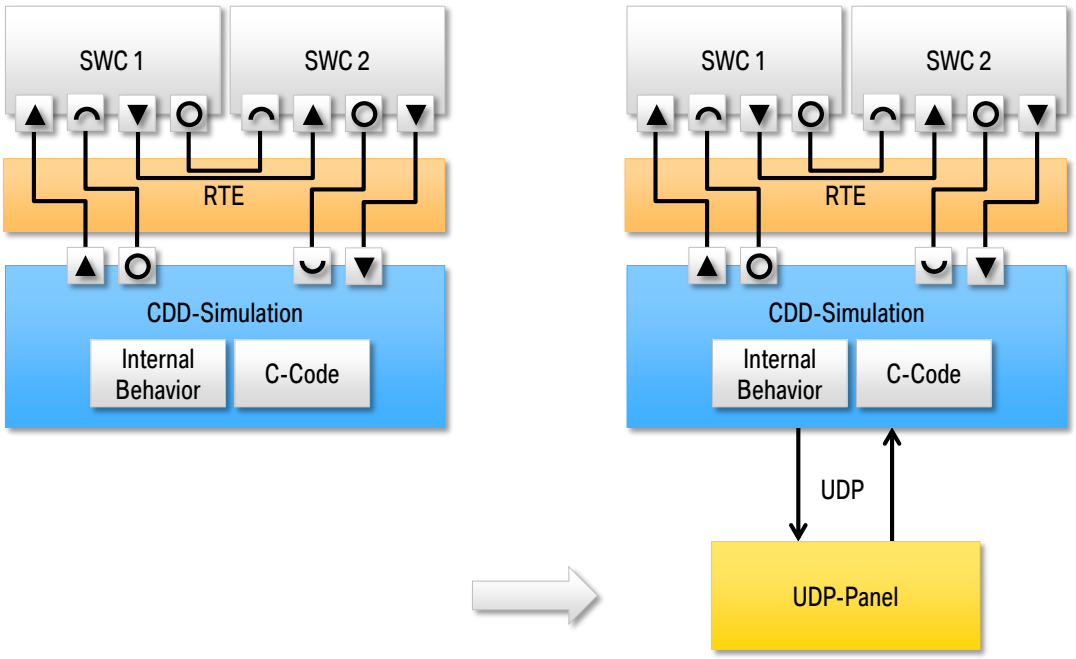


Abbildung 4.3: Anbindung eines UDP-Panels zur Interaktion mit der CDD-Simulation.

Für Werte, die von dem CDD zu den Software-Komponenten übertragen werden sollen, erlauben Eingabefelder die dynamische Vorgabe verschiedener Testvektoren während der Programmlaufzeit. Datenelemente, die von den Software-Komponenten gesendet werden, können im UDP-Panel dargestellt und somit leicht analysiert werden. Eine direkte Kopplung von Werkzeugen zur Testautomatisierung an das Panel, für die Durchführung von komplexen Testabläufen, ist ebenso möglich. Somit können sämtliche Schnittstellen zwischen einem CDD und der restlichen Steuergeräte-Software bereits abgesichert werden, ohne dass der Treiber selbst verfügbar sein muss. Dabei sind alle Schritte zur Erzeugung der CDD-Simulation nach dem obigen Verfahren vollständig automatisierbar.

4.1.3 CDD-Simulation durch Funktionsmodellierung

Wenn eine CDD-Simulation ohne Benutzerinteraktion benötigt wird, sieht das Konzept noch eine weitere Möglichkeit zur Simulationserzeugung vor. Als Ausgangsbasis dient hierfür ebenso das automatisiert erzeugte Grundgerüst, bestehend aus der AUTOSAR-Beschreibung mit sämtlichen Schnittstellen und dem *Internal-Behavior*. Modellierungswerkzeuge wie z. B. MATLAB/Simulink [Mat10] verfügen über eine Importfunktion für AUTOSAR-Beschreibungen. Darüber kann der Aufbau der CDD-Simulation mit allen Schnittstellen direkt in Funktionsblöcke in der Modellierungsumgebung umgewandelt werden. Falls kompatible Modelle für die Funktion des CDD vorliegen, sind diese direkt integrierbar. Ansonsten können die relevanten Anteile des Treibers unter Einsatz aller Möglichkeiten von MATLAB/Simulink soweit nachgebildet werden, wie es für die Absicherung der Funktions-Software nötig ist. Durch die bereits vorhandene Schnittstellendefinition ist die Kompatibilität zu den Software-Komponenten in jedem Fall sichergestellt. Aus dem Modell kann per Codegenerator wiederum direkt C-Code erzeugt werden, der die Funktion AUTOSAR-konform abbildet. Eine Anwendung dieses Prozesses ist in [Dei10] beschrieben.
Dieser Quellcode lässt sich zusammen mit der zuvor erzeugten AUTOSAR-Beschreibung nahtlos in eine VAP und die restlichen Software-Anteile integrieren, wie in Abbildung 4.4 visualisiert. Die Schnittstellen zwischen Funktions-Software und Complex-Device-Driver können somit bereits abgesichert werden, bevor der Treiber oder die Steuergeräte-Hardware verfügbar sind.

4.1.4 Fazit

Das Konzept der CDD-Simulation bietet eine effiziente und hochautomatisierte Lösung zur Absicherung der Schnittstellen zwischen Software-Komponenten und einem Complex-Device-Driver. Wie bereits in Kapitel 3.3.2 ausgeführt, ist der Ansatz auf beliebige Treiber anwendbar und kann bereits ab Verfügbarkeit der AUTOSAR-Beschreibung einzelner Software-Komponenten eingesetzt werden. Lediglich in der Erreichbarkeit der Testziele ist die Simulation nicht universell verwendbar, da der CDD selbst nicht integriert wird. Die technische Umsetzung des Konzepts folgt in Kapitel 5.1.

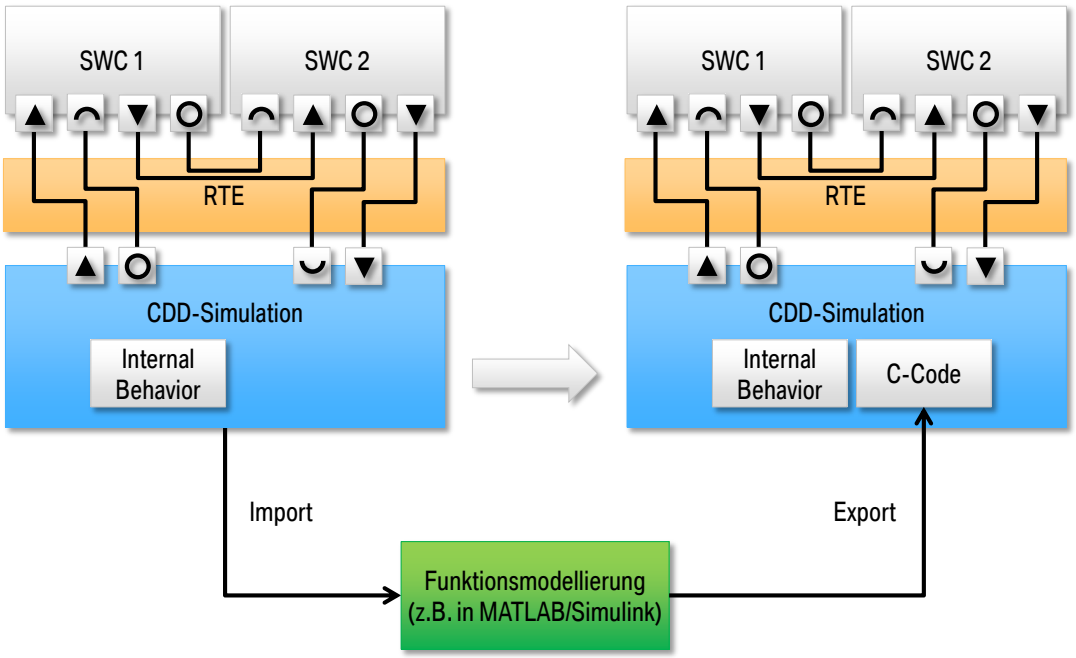


Abbildung 4.4: Modellierung der Funktionslogik der CDD-Simulation.

4.2 CDD-Hardware-Abstraction-Layer

Steht der Complex-Device-Driver ebenfalls im Testfokus, so ist die CDD-Simulation nicht zielführend. Der Einsatz eines zusätzlichen Hardware-Abstraction-Layers (HAL) ermöglicht jedoch die vollständige Integration des CDD, wie bereits in Kapitel 3.2.5 dargestellt. Dadurch kann der Treiber an die vorhandene Hardware einer VAP angebunden werden und erhält Zugriff auf reale Peripherie. Im Folgenden wird das bereits in [DHM12c] veröffentlichte Konzept detailliert erläutert.

4.2.1 Integration des Hardware-Abstraction-Layers

Im Steuergerät greift der CDD unmittelbar auf die Hardware zu. Dabei ist die Definition der Schnittstellen in keiner Weise eingeschränkt und kann vollständig auf die Gegebenheiten der Hardware angepasst werden. Allerdings ist daher auch keine Beschreibungsform der Schnittstellen innerhalb des AUTOSAR-Standards vorgeschrieben. Wie in Abbildung 4.5 dargestellt, fügt sich der CDD-HAL direkt unterhalb des Treibers ein. Dabei müssen die Schnittstellen des CDD aufgegriffen werden, um die übertragenen Daten auf die VAP-Hardware umzuleiten.
Das Konzept des CDD-HAL ist generisch aufgebaut, um ein breites Spektrum von Complex-Device-Drivers zu unterstützen. Nach dem Vorbild der AUTOSAR-BSW-Module ist der Hardware-Abstraction-Layer daher vollständig konfigurierbar ausgelegt, um eine einfache Anpassung an die unterschiedlichen Schnittstellen zu gewährleisten. Die Erzeugung des Quelltextes des CDD-HAL erfolgt dynamisch anhand der zugehörigen Konfiguration. Dabei werden dieselben Mechanismen wie bei der Basis-Software genutzt, um die Konfiguration und Generierung durchzuführen. So ist eine nahtlose Einbindung in den AUTOSAR-Workflow

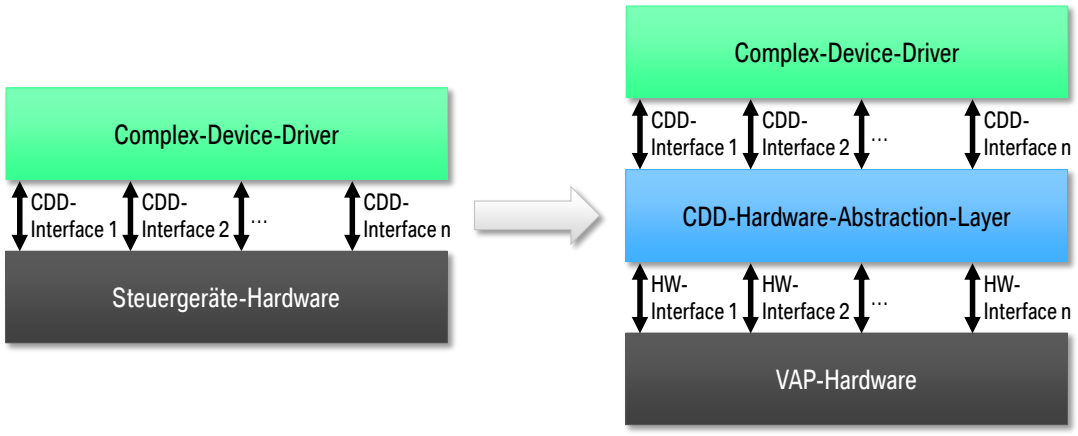


Abbildung 4.5: Eingliederung des CDD-Hardware-Abstraction-Layers zwischen Treiber und VAP-Hardware.

(siehe Kapitel 2.1.1) mit den zugehörigen Konfigurationswerkzeugen gewährleistet. Dies ermöglicht die Integration des HAL mit geringem Aufwand.

4.2.2 Struktur des Hardware-Abstraction-Layer

Der konfigurierbare Hardware-Abstraction-Layer ist in sich in weitere Schichten untergliedert, die jeweils unterschiedliche Aufgaben erfüllen. Abbildung 4.6 stellt einen Überblick über die Struktur des HAL dar, inklusive der Anbindung an den CDD und die Hardware der VAP. Die Integration des Treibers selbst muss möglichst ohne Veränderung des Quellcodes erfolgen, um eine Verfälschung der Ergebnisse auszuschließen. Daher werden die Hardware-Schnittstellen des CDD über eine Emulationsschicht nachgebildet. Der Treiber kann dadurch auf dieselbe Art und Weise mit der Emulation interagieren, wie mit der realen Hardware. Durch die Konfiguration der Schicht wird eine exakte Anpassung auf verschiedene Zugriffsformen eines CDD realisiert. Die Routing-Ebene verteilt die übertragenen Daten aus der Emulationsschicht auf die unterschiedlichen Hardware-Schnittstellen der VAP. Über die Konfiguration ist frei applizierbar, ob ein Datenelement z. B. über einen digitalen Ausgang oder den CAN-Bus ausgegeben werden soll. Die darunterliegende Ebene der Hardware-Abstraktion transferiert die Datenelemente in Sende- und Empfangsrichtung, von der Routing-Ebene zu den Hardware-Treibern. Dabei stehen sämtliche Hardware-Module zur Verfügung, die innerhalb einer VAP einsetzbar sind. Im Rahmen der zugehörigen Masterarbeit [Vog13] wurde zusätzlich die parallel zu den Schichten stehende Funktion des Monitors eingeführt. Darüber entsteht die Möglichkeit, die Eingangswerte des HAL in der Emulation und der Hardware-Abstraktion zu überwachen und dynamisch bei der Änderung von Datenelementen Operationen auszuführen. Im Folgenden werden die einzelnen Schichten des CDD-HAL detailliert dargestellt und die jeweiligen Konfigurationsmöglichkeiten erläutert.

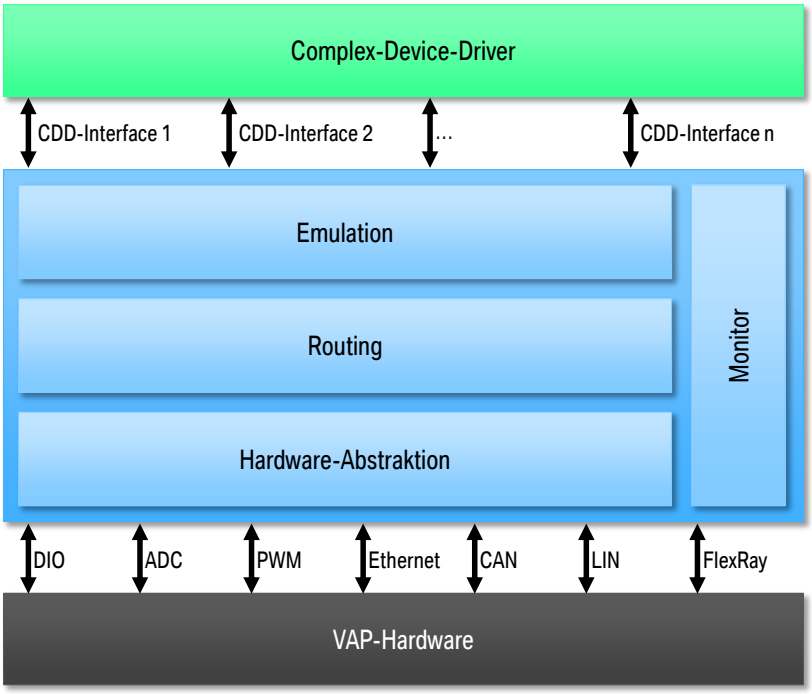


Abbildung 4.6: Aufbau des CDD-Hardware-Abstraction-Layers.

Emulation

Die oberste Schicht des CDD-HAL besteht aus einer Emulation der CDD-Hardware. Hierdurch wird der Zugriff des Treibers auf den Hardware-Abstraction-Layer auf identische Art möglich, wie auf die reale Hardware. Dabei ist eine möglichst exakte Nachbildung der Schnittstellen nötig, um funktional relevante Anpassungen am CDD zu vermeiden. Wie bereits in Kapitel 2.2.1 dargestellt, greifen CDDs entweder direkt auf Register der Hardware oder auf Speicherbereiche zu. Weiterhin sind Aufrufe von Low-Level-Funktionen und die Abarbeitung von Interrupts möglich. Alle vier Zugriffsvarianten sind durch die Emulation abdeckbar:

- **Register- und Speicherzugriffe:** Direkte Zugriffe auf Register oder Speicherbereiche eines Mikrocontrollers per DMA müssen auf einer VAP adaptiert werden. Die zugehörigen Adressen sind nicht direkt auf den Adressraum des x86-Systems übertragbar. Innerhalb der Emulationsschicht werden die Register und Speicherbereiche durch Puffer abgebildet. Da eine VAP über wesentlich mehr Arbeitsspeicher verfügt als ein Mikrocontroller, sind die Puffer direkt als Variablen im Quelltext der Emulation deklarierbar. In der Konfiguration der Emulationsschicht muss lediglich der Name des zugehörigen Registers oder Speicherbereichs im Quelltext des CDD angegeben werden. Dadurch kann z. B. die per Makro (*#define* im Quelltext) festgelegte Adresse eines Registers automatisiert durch die Variable des emulierten Registers ersetzt werden. Dies stellt zwar eine Anpassung des Quellcodes des CDD dar, allerdings bleibt die Funktion des Treibers selbst dabei unverändert. Der CDD kann auf das emulierte Register in derselben Form lesend und schreibend zugreifen wie auf ein reales Register. Selbiges gilt für Speicherbereiche, die per DMA beschrieben oder gelesen werden. Lediglich

die Adresse wird für die Ausführung auf der VAP angepasst, wodurch auch keine Geschwindigkeitseinbußen entstehen.

- **Funktionsaufrufe:** Dieser Abschnitt bezieht sich auf Funktionen, die nicht Teil des CDD-Quellcodes sind, sondern z. B. von Low-Level-Treibern als API der Hardware bereitgestellt werden. Da sich die Hardware der VAP grundsätzlich von einem Mikrocontroller unterscheidet, sind die hardwarenahen Funktionen nicht direkt verwendbar. Das Konzept sieht vor, den Funktionsaufruf durch eine neue Funktion innerhalb der Emulation zu ersetzen. Dies kann entweder durch Anpassung des Quellcodes oder der binären Dateien des CDD erfolgen. Da die Funktionen nicht Teil des Complex-Device-Drivers sind, erfolgt somit auch keine Anpassung des Treibers selbst. Durch die Konfiguration des CDD-HAL können die Argumente des Funktionsaufrufes an die entsprechenden Treiber der VAP-Hardware weitergeleitet werden. Eventuell nötige Umrechnungen der Parameter sind ebenfalls möglich. Handelt es sich bei dem Funktionsaufruf z. B. um den Abruf von Werten eines ADC, wobei der gewünschte Kanal als Argument übergeben wird, so kann der Kanal an den Funktionsaufruf des VAP-ADC übertragen werden.
- **Interrupts:** Das Konzept unterstützt ebenso die Emulation von Interrupts. Die Hardware der VAP bietet einen Interrupt-Eingang, dessen Signal über die Emulation an den CDD weitergereicht werden kann, um die zugehörige ISR zu triggern. Alternativ ist der Interrupt auch über die Emulationsschicht selbst erzeugbar, wenn die Anzahl der Interrupt-Eingänge der VAP-Hardware nicht ausreicht. Beide Varianten sind über die zugehörige Konfiguration abbildbar.

In der Konfiguration des CDD-HAL sind alle Möglichkeiten applizierbar. Abbildung 4.7 zeigt schematisch jeweils ein emuliertes Register, einen emulierten Speicherbereich, eine emulierte Low-Level-Funktion und einen emulierten Interrupt innerhalb der Emulationsschicht.

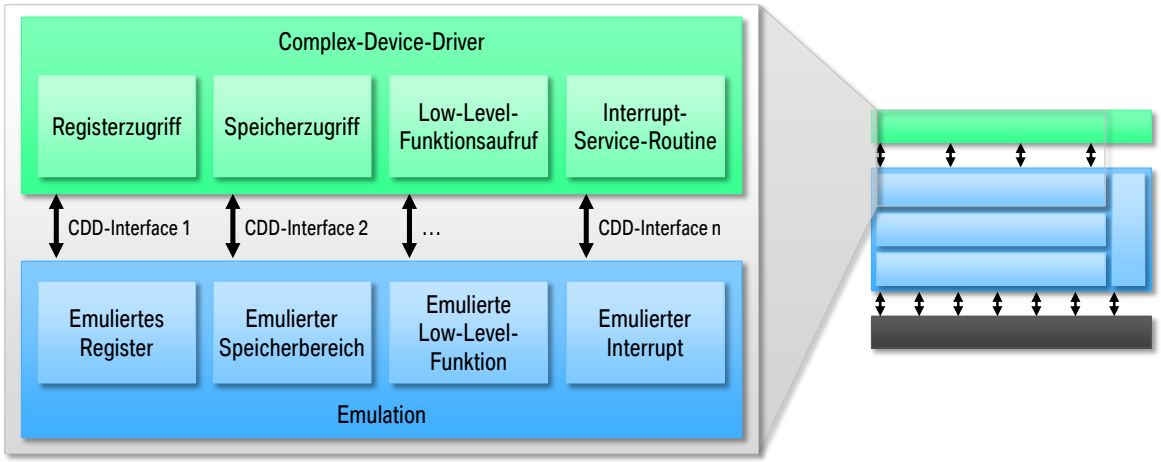


Abbildung 4.7: Aufbau der Emulationsschicht des CDD-HAL.

Routing

Die mittlere Schicht des CDD-HAL wird durch das Routing dargestellt. Diese Ebene dient zur Verknüpfung der emulierten Schnittstellen mit der Hardware der VAP. Die Konfigurationsmöglichkeit erlaubt dabei die flexible Verkettung von CDD-Schnittstellen mit allen verfügbaren Hardware-Elementen der Plattform. Die Voraussetzung ist lediglich, dass die Datenelemente mit der entsprechenden Hardware kompatibel sind oder durch geringfügige Umrechnungen angeglichen werden können. So entstehen vielfältige Verknüpfungsmöglichkeiten, welche die Anwendung auf eine große Bandbreite von Complex-Device-Drivers ermöglichen. Zu Testzwecken können geschriebene Daten des CDD ebenso auf mehreren Hardware-Modulen simultan ausgegeben werden. Alternativ ist die vollständige Datenübertragung auf einen CAN-Kanal oder einen anderen Fahrzeugbus umleitbar, um z. B. eine Testsuite anzubinden. Bei der Konfiguration müssen dafür die gewünschten Verknüpfungen angelegt werden, woraus der zugehörige Codegenerator die Repräsentation im Quellcode des CDD-HAL erzeugt. Abbildung 4.8 zeigt eine schematische Darstellung des Routings.

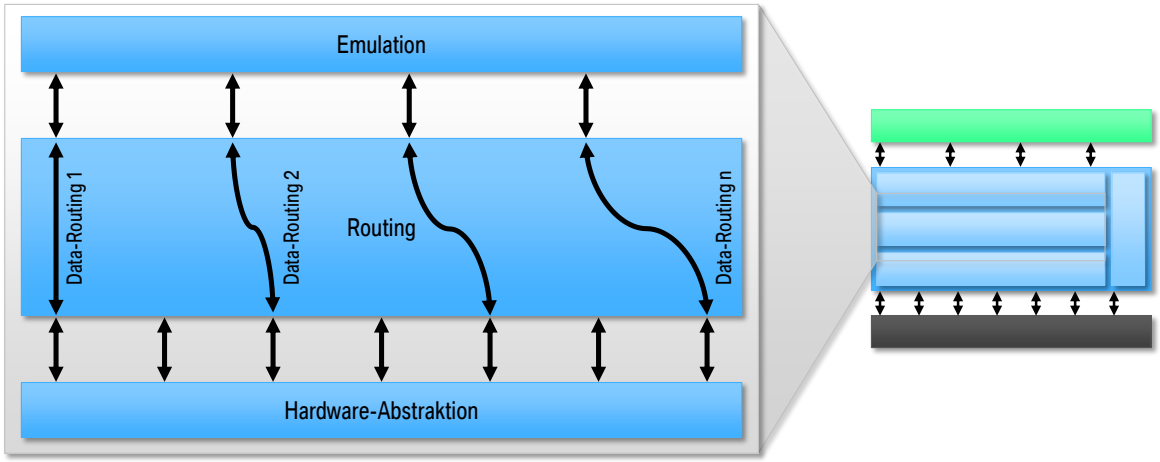


Abbildung 4.8: Aufbau der Routingschicht des CDD-HAL.

Hardware-Abstraktion

Die unterste Ebene des CDD-HAL greift auf die Treiber der VAP-Hardware zu. Dabei können sämtliche Module der Plattform eingesetzt werden. Zur Verfügung stehen die verschiedenen Fahrzeugbussysteme Controller-Area-Network (CAN), Local-Interconnect-Network (LIN), FlexRay und Ethernet. Weiterhin kann auf Analog-Digital-Wandler (ADC), digitale Ein- und Ausgänge (DIO) und Pulsweitenmodulation (PWM) zurückgegriffen werden. Die Konfiguration des CDD-HAL erlaubt eine flexible Auswahl der gewünschten Module. Für jede Hardware-Komponente existiert eine Hardware-Abstraktion, welche für den Datenaustausch mit der Routingschicht verantwortlich ist. Weiterhin erfolgt die Ansteuerung der darunterliegenden Treiber des Echtzeit-Linux-Systems der VAP über die Hardware-Abstraktion. Die Treiber werden durch die jeweiligen Hersteller der Module bereitgestellt und bieten über die zugehörigen APIs vollen Zugriff auf die Hardware. Der Aufbau der gesamten Abstraktionsschicht ist in Abbildung 4.9 detailliert dargestellt.

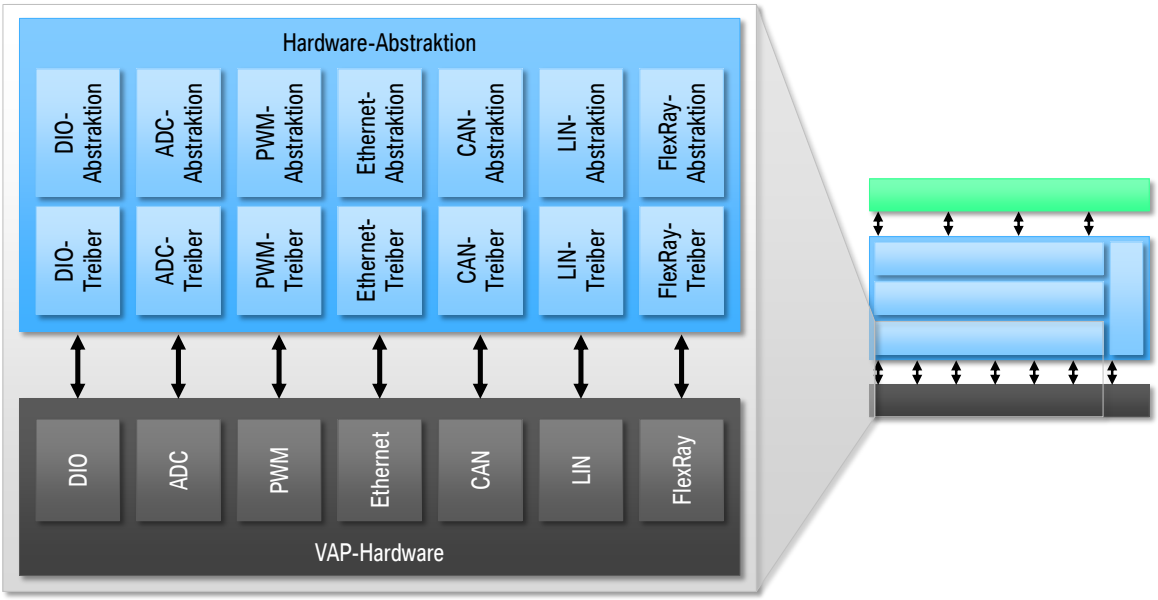


Abbildung 4.9: Aufbau der Hardware-Abstraktion des CDD-HAL.

Monitor

Im Rahmen der zugehörigen Masterarbeit [Vog13] wurde das Konzept um den sogenannten Monitor erweitert. Dieser überwacht sämtliche Ein- und Ausgänge des CDD-HAL, welche in der Emulation und der Hardware-Abstraktion liegen. So ist die Detektion von Änderungen in den Signalverläufen möglich. Wenn z. B. eine steigende Taktflanke an einem digitalen Eingang der VAP eingeht, wird diese Änderung im Monitor erkannt und die Übertragung des Wertes von der Hardware-Abstraktion über das Routing in die Emulationsschicht gestartet. Die Änderung des Eingangswertes wird dabei an den CDD weitergereicht und der Treiber kann das Signal verarbeiten. Abbildung 4.10 zeigt die Verbindungen des Monitors zu den anderen Ebenen des CDD-HAL.
Die Überwachung kann dabei sowohl zeitgesteuert im Polling-Modus als auch ereignisgesteuert erfolgen. Insgesamt wurden vier Monitoring-Möglichkeiten untersucht, die im Folgenden kurz zusammengefasst werden:

- **Timed-Monitor:** Die einfachste Monitor-Variante ist das zyklische Polling aller relevanten Ein- und Ausgangssignale. In individuell konfigurierbaren Intervallen werden so die benötigten Werte abgefragt, auf Veränderung geprüft und vom CDD zur Hardware oder entgegensetzt weitergeleitet. Nach dem nyquist-shannonschen Abtasttheorem [Sha49] muss dabei allerdings sichergestellt sein, dass das Polling des Monitors mindestens mit der doppelten Frequenz erfolgt, mit welcher sich das schnellste überwachte Signal ändert. Bei entsprechend schneller Abtastung von Eingangssignalen eignet sich der Timed-Monitor als einziger der vier Monitore zur Nachbildung von Interrupts. Dies ist notwendig, da die eingesetzte VAP-Hardware lediglich über einen Interrupt-Eingang verfügt, der in seinen Einsatzmöglichkeiten limitiert ist. Die zyklische Abtastung führt

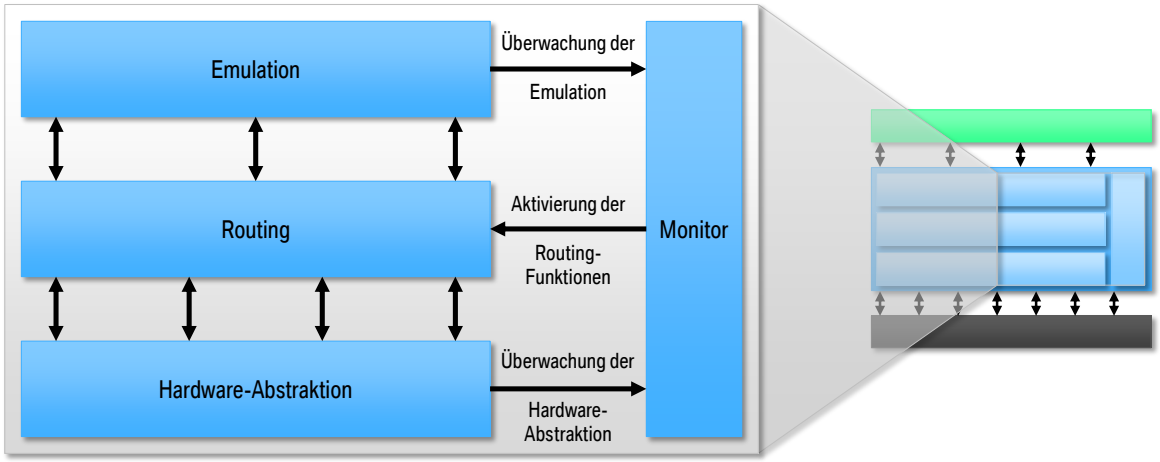


Abbildung 4.10: Aufbau des Monitors innerhalb des CDD-HAL.

zu einer Erhöhung der Rechenlast, welche jedoch durch den Einsatz performanter Prozessoren in einer VAP kompensierbar ist.

- **Breakpoint-Monitor:** Debugger nutzen auf x86-Linux-Systemen z. B. den *ptrace*-Systemaufruf [Lin10], um andere Prozesse zu überwachen und Breakpoints zu setzen. Derselbe Mechanismus wird für den Breakpoint-Monitor genutzt. Dabei ermöglicht *ptrace* die Installation von Breakpoints z. B. bei Lese- oder Schreibzugriffen auf die Speicheradresse einer Variablen innerhalb des CDD, ohne Anpassung des Quellcodes. So kann der Monitor direkt bei Zugriffen auf relevante Variablen reagieren und die Weiterleitung der geänderten Werte, zwischen Emulation und Hardware-Abstraktion, über das Routing anstoßen. Die Ausführung des überwachten Prozesses kann direkt fortgesetzt werden, um die Laufzeit nicht zu verfälschen.

- **Singlestep-Monitor:** Der *ptrace*-Systemaufruf kann ebenso dazu verwendet werden, die Ausführung des CDD und des CDD-HAL nach jeder Instruktion zu unterbrechen. So wird die Kontrolle über den Prozess nach jedem Ausführungsschritt kurz an den Monitor abgegeben. Dieser kann sämtliche Variablen auf Veränderungen prüfen und falls nötig die zugehörige Routing-Funktion anstoßen. Danach kann die Ausführung der nächsten Instruktion fortgesetzt werden. Mit einem entsprechend performanten VAP-System können die Unterbrechungszeiten in der Ausführung gering gehalten werden, um die Laufzeiten nicht zu stark zu verfälschen. Eine gewisse Laufzeitverlängerung ist jedoch in diesem Singlestep-Modus nicht zu verhindern.

- **Programminvasiver-Monitor:** Die Instrumentierung des Quelltextes eines CDD stellt eine ereignisgesteuerte Möglichkeit zur Detektion von Variablenzugriffen dar. Die Konfiguration der Emulationsschicht des CDD-HAL enthält alle relevanten Hardware-

Zugriffe des Treibers. Mit diesen Informationen kann der Quelltext des CDD, z. B. bei Lese- oder Schreibzugriffen auf die relevanten Variablen, automatisiert mit Funktionsaufrufen für den Monitor versehen werden. Bei einem entsprechenden Aufruf kann der Monitor wiederum die zugehörige Routing-Funktion auslösen. Dies stellt zwar einen Eingriff in den CDD-Quellcode dar, ist jedoch eine sehr ressourcenschonende Methode für die Überwachung von Variablen.

Die Konfiguration des Monitors erlaubt die parallele Nutzung der vier verschiedenen Lösungen. Anhand der jeweiligen Routing-Pfade ist definierbar, welche Variablen durch einen bestimmten Monitor überwacht werden. Die Erzeugung von mehreren Instanzen eines Monitor-Typs ist ebenso möglich.

4.2.3 Fazit

Die detaillierte Beschreibung des CDD-Hardware-Abstraction-Layers zeigt die Fexibilität des Konzepts im Bezug auf unterschiedliche Complex-Device-Driver und Hardware-Komponenten. Der modulare Aufbau und die Konfigurationsmöglichkeiten analog zu einem AUTOSAR-Basis-Software-Modul verringern den Integrationsaufwand. Wie bereits in Kapitel 3.3.2 dargestellt, sind alle Testziele mit dem Ansatz bei gleichzeitig hoher Verfügbarkeit erreichbar, da keine spezifische Zusatzhardware erforderlich ist. In Kapitel 5.2 folgt die technische Realisierung des Konzepts.

4.3 Zusammenfassung

Das Konzept der CDD-Simulation ist in der Lage, einen Complex-Device-Driver vollständig zu ersetzen. Dies ist sinnvoll, wenn der Treiber selbst nicht im Testfokus steht oder nicht verfügbar ist und die Schnittstellen der restlichen Funktions-Software bereits abgesichert werden müssen. Die Erzeugung einer einfachen Simulation ist vollständig automatisierbar. Es wird sowohl eine vollständige AUTOSAR-Beschreibung als Ersatz für den realen CDD erzeugt, als auch der zugehörige Quellcode für die Anbindung an eine Oberfläche zur Eingabe von Testvektoren. Komplexe Simulationsabläufe lassen sich durch die Anbindung von Testautomatisierungswerkzeugen realisieren. Alternativ kann die CDD-Funktionalität, durch den Import der generierten AUTOSAR-Beschreibung in MATLAB/Simulink, nachmodelliert und in die CDD-Simulation eingebunden werden.
Die vollständige Integration eines Complex-Device-Drivers in eine VAP ist mit dem Konzept des CDD-Hardware-Abstraction-Layers realisierbar. Die Schnittstellen des Treibers, die im Steuergerät direkt auf den Mikrocontroller zugreifen, können über den HAL auf die verfügbare Hardware einer VAP umgeleitet werden. Eine konfigurierbare Emulationsschicht innerhalb des HAL erlaubt die Anbindung an alle Schnittstellenvarianten eines CDD. Die übertragenen Datenelemente sind über eine Routingschicht frei auf die verschiedenen Hardware-Module der Plattform verteilbar. Die Ansteuerung der jeweiligen Low-Level-Treiber erfolgt innerhalb der Hardware-Abstraktion. Ein Monitor detektiert Änderungen von Werten an der oberen und unteren Grenze des CDD-HAL und stößt die Übertragung der Datenelemente über die einzelnen Ebenen an. Alle Schichten sind analog zu dem Konfigurations-Workflow eines Basis-Software-Moduls flexibel auf die Anforderungen unterschiedlicher CDDs anpassbar. Die Konzepte wurden bereits in [DHM12a, DHM12c, DHM13, DH14] veröffentlicht.

Kapitel 5

Realisierung der ausgewählten Konzepte

Aufbauend auf den theoretischen Beschreibungen der CDD-Simulation und des CDD-HAL aus Kapitel 4 folgt eine Ausarbeitung der technischen Realisierung beider Lösungen. Dabei werden die entwickelten Prototypen und Werkzeuge detailliert beschrieben.

5.1 CDD-Simulation

Für die Simulation des Verhaltens eines Complex-Device-Drivers wurde im Rahmen dieser Arbeit ein erster Prototyp umgesetzt, wie bereits in [DHM12a, DHM13] veröffentlicht. Die zugehörige Masterarbeit [Arr12] erweiterte die Lösung technisch um die Unterstützung zusätzlicher Schnittstellenvarianten und um die automatisierte Generierung von Quellcode. Alle Elemente des Konzepts (siehe Kapitel 4.1) wurden über das selbst entwickelte Eclipse-Plug-in „SWC-Generator“, basierend auf der AUTOSAR-Tool-Platform (Artop) [RVE10], realisiert und in Java [Ora15] umgesetzt. Artop dient als Grundgerüst der meisten AUTOSAR-Architektureditoren und ist für die Bearbeitung von ARXML-Dateien innerhalb des Eclipse-Modeling-Framework (EMF) [SBPM08] ausgelegt. Dies erlaubt eine komfortable Erstellung und Bearbeitung der einzelnen Elemente einer AUTOSAR-Beschreibung innerhalb der Eclipse-Plattform.

5.1.1 Automatisierte Erzeugung der Simulationskomponenten

Die CDD-Simulation erfordert als erstes die Generierung einer AUTOSAR-Beschreibung, um die Simulationskomponente in die restliche Funktions-Software integrieren zu können. Dafür erzeugt der SWC-Generator eine neue *ComplexDeviceDriverSwComponent* und eine zugehörige *SwcImplementation* innerhalb einer ARXML-Datei. Dieser Container für die Simulation benötigt zusätzlich *Ports* mit den passenden Schnittstellendefinitionen und ein eigenes *SwcInternalBehavior*.

Generierung der Ports

Für die Generierung der *Ports* der CDD-Simulation ist zunächst eine Detektion von offenen Schnittstellen an Software-Komponenten nötig. Dazu prüft der SWC-Generator, ob für die *Ports* aller *Software-Components* in einer *Composition* jeweils mindestens ein sogenannter

AssemblyConnector existiert, der die Verbindung zwischen zwei *Ports* abbildet. Innerhalb eines Wizards des SWC-Generators erhält der Anwender eine Auswahlmaske mit allen *Ports*, die der CDD-Simulation hinzugefügt werden können. Dabei erfolgt eine automatische Selektion aller unverbundenen *Ports*, die normalerweise eine Verbindung zu dem CDD hätten. Eine manuelle Selektion weiterer *Ports* ist ebenso möglich, um zusätzliche Schnittstellen in die Simulationskomponente zu integrieren. Abbildung 5.1 zeigt die zugehörige grafische Oberfläche des SWC-Generators.

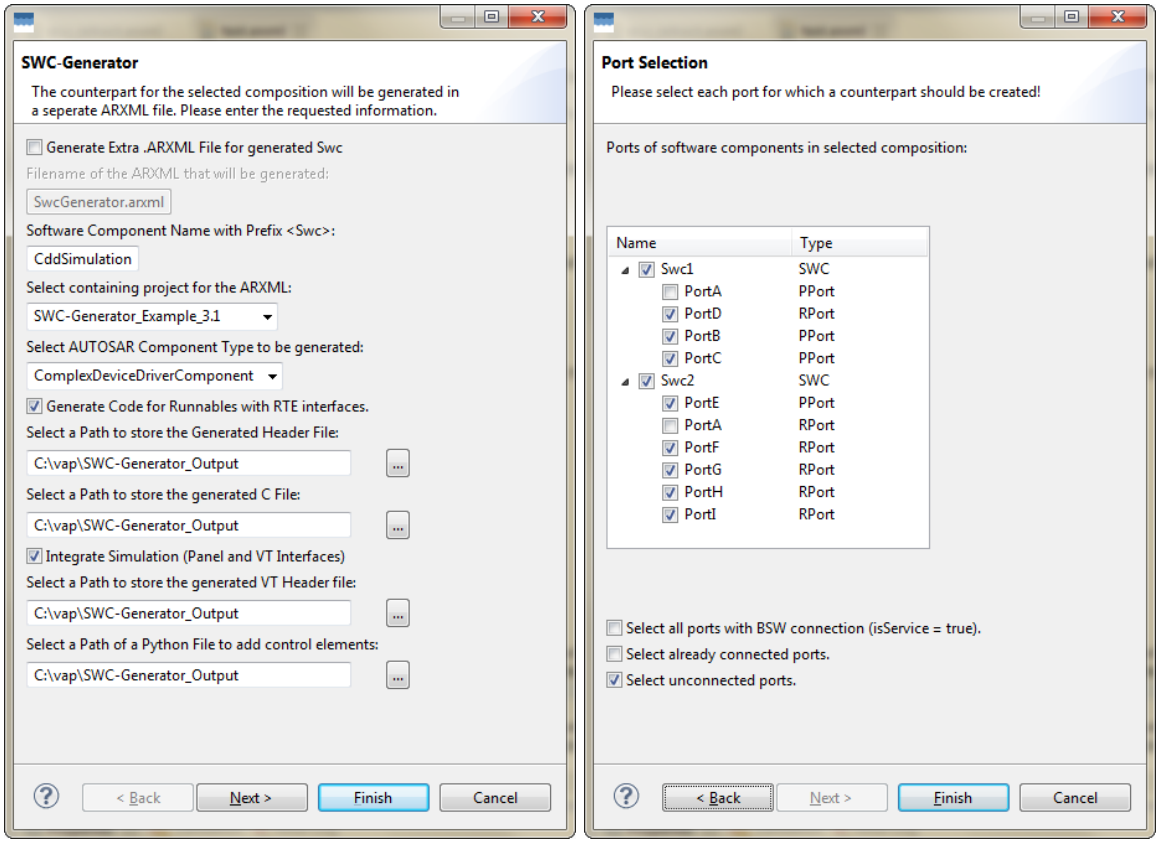


Abbildung 5.1: Grafische Oberfläche des SWC-Generators zur Erzeugung der CDD-Simulation, bestehend aus Startseite (links) und *Port*-Auswahl (rechts).

Nach Abschluss der *Port*-Auswahl erzeugt der SWC-Generator automatisch einen *Port* mit entgegengesetzter Übertragungsrichtung innerhalb der Simulationskomponente zu jeder der Schnittstellen. Für die selektierten *ProviderPorts* wird somit jeweils ein *ReceiverPort* generiert und umgekehrt. Die Schnittstellendefinition wird über ein *PortInterface* festgelegt, welches die Kommunikationsart (*Client-Server* oder *Sender-Receiver*) und die übertragenen Datenelemente bestimmt. Jeder Port muss ein solches *PortInterface* referenzieren. Der SWC-Generator kopiert automatisch die Referenz der Schnittstellendefinition aus dem ursprünglichen *Port* in den neu erzeugten *Port* mit entgegengesetzter Übertragungsrichtung. So ist die Kompatibilität der *Ports* zueinander sichergestellt. Für die Verbindung des bestehenden und des neu erzeugten *Ports* legt der SWC-Generator jeweils einen *AssemblyConnector* an. Das Ergebnis des Vorgangs ist in Abbildung 5.2 für ein einfaches Beispiel mit zwei SWCs innerhalb eines AUTOSAR-Architektureditors dargestellt.

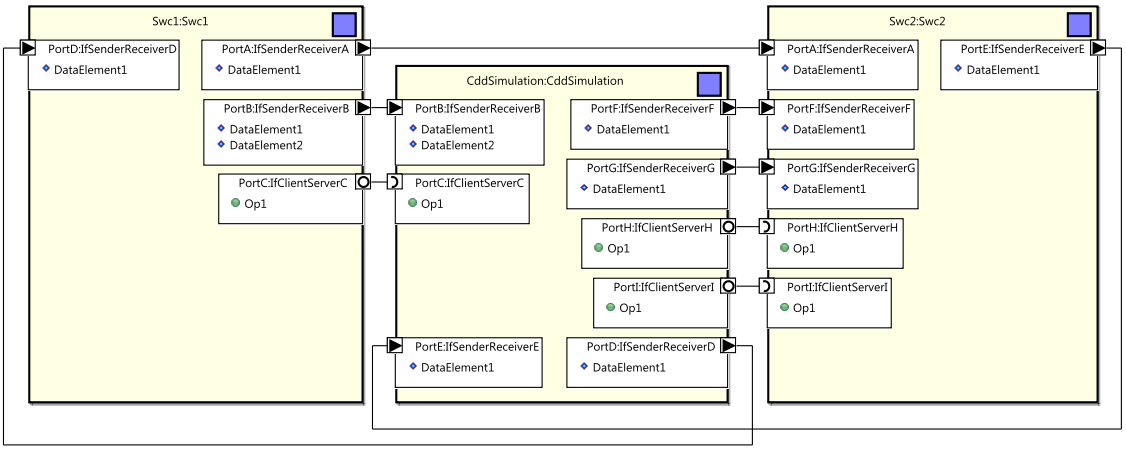


Abbildung 5.2: Beispiel einer generierten CDD-Simulation in einer *Composition*.

Generierung des Internal-Behavior

Damit die erzeugte *ComplexDeviceDriverSwComponent* für die Simulation ausführbar ist, wird ein eigenes *SwcInternalBehavior* benötigt. Dies ist ebenso durch den SWC-Generator automatisch erzeugbar. Einzelne Funktionen werden innerhalb des *SwcInternalBehavior* als *Runnables* angelegt. Die Zugriffsberechtigungen auf die *Ports* der Komponente werden über Elemente des Typs *VariableAccess* bestimmt.

Zur Reduktion der Komplexität erzeugt der SWC-Generator ein *Runnable*, welches die Zugriffsberechtigungen für alle *Sender-*, *Receiver-* und *Client-Ports* zugewiesen bekommt. Hierbei ist jedoch relevant, ob die Kommunikation nach dem Prinzip der expliziten oder impliziten Datenübertragung (für *Sender-Receiver-Kommunikation*), bzw. synchron oder asynchron (für *Client-Server-Kommunikation*) abläuft. Die Information, welches Prinzip verwendet wird, kann durch den Typ des *VariableAccess* aus den *Runnables* der Funktions-Software extrahiert werden. Der SWC-Generator setzt dafür auf die Funktion *getInverseReferences* von Artop, die das Auffinden von Referenzen zu einem AUTOSAR-Element erlaubt. Zu jedem *Port*, für den ein Gegenstück erzeugt wurde, erfolgt eine Suche nach allen Referenzen in Elementen des Typs *VariableAccess*, gefolgt von einer Analyse des Übertragungsprinzips.

Bei einem impliziten Übertragungsverhalten im *Runnable* der Funktions-Software legt der SWC-Generator, je nach Übertragungsrichtung des zugehörigen *Sender-Receiver-Ports*, ein Element vom Typ *dataReadAccess* oder *dataWriteAccess* im *Runnable* der CDD-Simulation an. Für explizite Kommunikation wird ein *dataReceivePoint* bzw. ein *dataSendPoint* generiert. Im Fall eines *Client-Ports* handelt es sich um einen *AsynchronousServerCallPoint*, bzw. einen *SynchronousServerCallPoint*.

Werden in der CDD-Simulation *Server-Ports* erzeugt, sind besondere Vorgaben des AUTOSAR-Standards zu beachten. Der SWC-Generator legt in diesem Fall für jeden *Server* ein eigenständiges *Runnable* an, da mehrere *Server* nicht über dieselbe Funktion angesprochen werden dürfen. Jeder *Server-Port* erfordert zusätzlich ein *OperationInvokedEvent*, das bei dem Start einer Operation, durch den zugehörigen *Client* in einer der Software-Komponenten, ausgelöst wird. Das *Event* dient der RTE als Trigger für die Ausführung des *Server-Runnables*.

Abbildung 5.3 zeigt ein Beispiel für ein vollständig generiertes *SwcInteralBehavior* in der grafischen Ansicht eines AUTOSAR-Architektureditors.

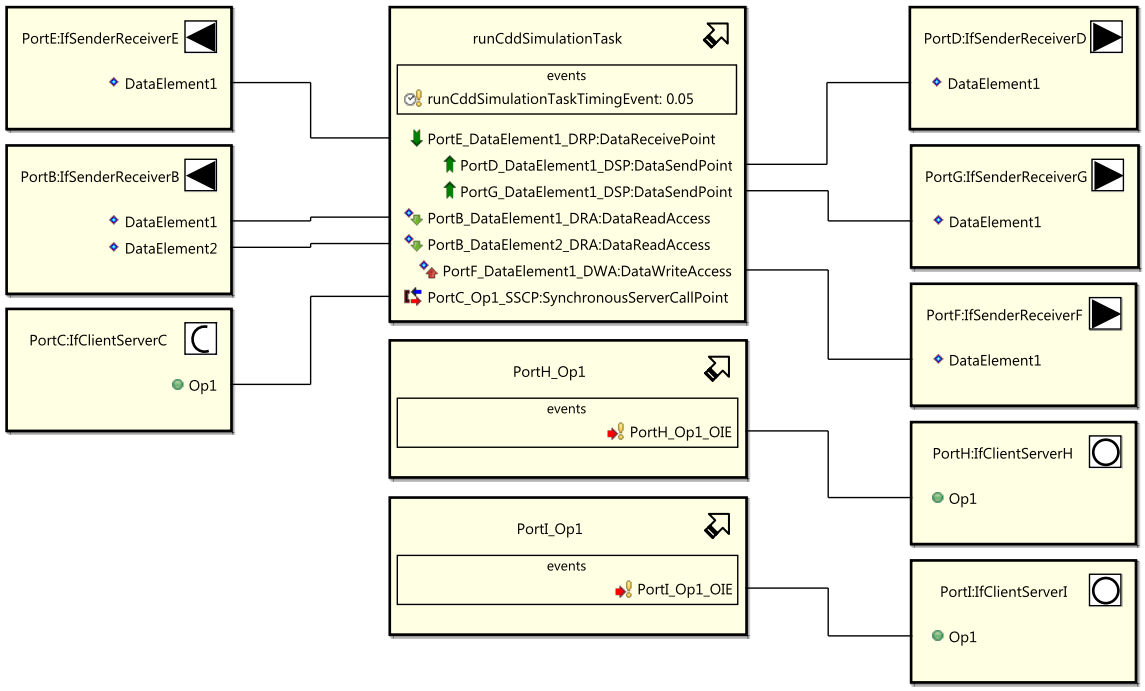


Abbildung 5.3: Beispiel eines generierten *SwcInternalBehavior* für eine CDD-Simulation.

Die erzeugte AUTOSAR-Beschreibung der CDD-Simulation wird automatisch in die restliche Funktions-Software integriert. Somit ist bereits die Ausführung der Generatoren für die Basis-Software-Module möglich, welche die Daten der Software-Komponenten als Eingangsparameter benötigen.

5.1.2 CDD-Simulation mit Benutzerinteraktion

Die bisher erzeugten Anteile der CDD-Simulation enthalten noch keine Funktionalität. Für den Kompiliervorgang und die Ausführung ist noch ein passender Quellcode für die Simulations-Komponente nötig. Damit die Schnittstellen der Software-Komponenten getestet werden können, ist weiterhin eine Möglichkeit zur Applikation von Testvektoren und zur Verifikation der Ausgabewerte erforderlich.

Quellcodegenerierung

Der SWC-Generator wurde um eine automatische Generierung des Quellcodes zur AUTOSAR-Beschreibung der CDD-Simulation erweitert. Dies ist möglich, da der AUTOSAR-Standard feste Zusammenhänge zwischen der Beschreibung einer Software-Komponente und ihrem zugehörigen C-Code definiert [AUT11h]. Jede Software-Komponente benötigt mindestens eine C-Datei und eine zugehörige Header-Datei. Beide Dateien sind über den SWC-Generator automatisch generierbar. Dabei wird für jedes *Runnable* eine eigene C-Funktion erzeugt.

Darin fügt der SWC-Generator alle Aufrufe zur Datenübertragung ein, die innerhalb des *SwcInternalBehavior* definiert wurden. Jede Kommunikation durch einen *Port* einer Software-Komponente muss über einen RTE-Aufruf erfolgen. Die Funktionsprototypen für den Zugriff auf jeden *Port* werden durch den RTE-Generator erzeugt und in einer zusätzlichen RTE-Header-Datei zu jeder SWC definiert. Bei *Sender-Receiver-Interfaces* unterscheiden sich die Aufrufe unter anderem nach Lese- oder Schreibzugriff, impliziter oder expliziter Kommunikation und ob die Datenelemente gepuffert werden oder nicht. Das Quellcodebeispiel 5.1 wurde passend zum ersten *Runnable* des obigen Beispiels eines *SwcInteralBehavior* (siehe Abbildung 5.3) generiert und enthält einige der Varianten. Zeile 9 stellt einen impliziten Lesezugriff auf *PortB* dar und Zeile 23 einen expliziten Lesezugriff auf *PortE*. Ein expliziter Schreibzugriff auf *PortD* ist in Zeile 48 definiert sowie ein impliziter Schreibzugriff auf *PortF* in Zeile 62.
Der Aufruf von *Client-Server-Interfaces* erfolgt immer über einen *Rte_Call* auf der Seite des *Clients* (siehe Zeile 36). Zu jedem der Aufrufe generiert der SWC-Generator ebenso eine passende Deklaration der nötigen Variablen. Bei einem Schreibzugriff erfolgt zusätzlich eine Initialisierung mit Standardwerten. So kann der generierte Quellcode bereits mit dem der anderen Software-Komponenten kompiliert, gelinkt und ausgeführt werden.

Generierung einer Benutzerschnittstelle zur Interaktion

Für eine dynamische Simulation von Eingangswerten wurde die AUTOSAR-Komponente der CDD-Simulation an eine grafische Oberfläche zur Interaktion und Auswertung angebunden. Aus einem internen Projekt der BMW Group stand bereits eine Python-Bibliothek zur einfachen Erstellung einer geeigneten Benutzeroberfläche zur Verfügung. Über Funktionsaufrufe im C-Code einer AUTOSAR-Applikation ist der Austausch von Datenelementen mit definierbaren Eingabefeldern möglich. Die Kommunikation erfolgt über das User-Datagram-Protocol (UDP), welches das Echtzeitverhalten der AUTOSAR-Applikation nicht beeinträchtigt. Im Folgenden wird diese Benutzeroberfläche als UDP-Panel bezeichnet.
Wie die verschiedenen RTE-Aufrufe eines *Runnable* sind die Funktionen zur Kommunikation mit dem UDP-Panel automatisiert erzeugbar. Auch der Aufbau der grafischen Oberfläche ist vollständig dynamisch generierbar. Die Erstellung des nötigen Quellcodes wurde direkt in den SWC-Generator integriert. Im Quellcodebeispiel 5.1 sind die Aufrufe für ein *Runnable* dargestellt. Für jedes gelesene oder geschriebene Datenelement wird die Funktion *VT_RX*, bzw. *VT_TX* integriert. Für Schreibzugriffe auf *Ports* der CDD-Simulation wird über die *VT_RX*-Funktion zunächst der aktuelle Wert des zugehörigen Eingabefeldes des UDP-Panels eingelesen und danach zur RTE übertragen, wie z. B. in Zeile 46 von Quellcodebeispiel 5.1. Datenelemente, welche die CDD-Simulation empfängt, werden nach dem Einlesen des Wertes per *VT_TX*-Funktion an die Benutzeroberfläche übertragen. Eine Implementierung dazu ist in Zeile 11 des Quellcodebeispiels 5.1 zu finden. Die zugehörige grafische Darstellung des UDP-Panels, welche für das obige Beispiel einer CDD-Simulation automatisch generiert wurde, ist in Abbildung 5.4 sichtbar.
Über die Benutzeroberfläche ist direkt die Eingabe von Werten möglich, welche über die *Ports* der CDD-Simulation an die Funktions-Software übertragen werden sollen. So können während der Laufzeit der vollständigen AUTOSAR-Applikation Testvektoren appliziert werden, um die Schnittstellen und die Funktionalität der Software-Komponenten zu verifizieren, ohne dass der CDD benötigt wird. Gleichzeitig visualisiert das UDP-Panel Werte, die an den Treiber übermittelt werden und ermöglicht so ebenfalls eine Verifikation der Ausgabeparameter der

```
FUNC(void, RTE_APPL_CODE) RTERunnableCddSimulationTask(){
        vtSync();

        /*---- Read from RTE rPorts ----*/

        /* PortB_DataElement1 */
        UInt8 PortB_DataElement1;
        /* Implicit read from RTE */
        PortB_DataElement1 =
            Rte_IRead_RTERunnableCddSimulationTask_PortB_DataElement1();
        /* Transmit value to UDP-Panel */
        VT_TX(PortB_DataElement1_VAL, PortB_DataElement1);

        /* PortB_DataElement2 */
        UInt8 PortB_DataElement2;
        /* Implicit read from RTE */
        PortB_DataElement2 =
            Rte_IRead_RTERunnableCddSimulationTask_PortB_DataElement2();
        /* Transmit value to UDP-Panel */
        VT_TX(PortB_DataElement2_VAL, PortB_DataElement2);

        /* PortE_DataElement1 Explicit Read */
        UInt8 PortE_DataElement1;
        /* Explicit read from RTE */
        Rte_Read_PortE_DataElement1(&PortE_DataElement1);
        /* Transmit value to UDP-Panel */
        VT_TX(PortE_DataElement1_VAL, PortE_DataElement1);


        /*---- Call clients on RTE ----*/

        /* PortC_Op1 */
        UInt8 Op1_Arg1 = 0;
        UInt8 Op1_Arg2 = 0;
        /* Read value from UDP-Panel */
        VT_RX(PortC_BTN, Op1_Arg1, Op1_Arg1_VAL);
        /* Call server on RTE */
        Rte_Call_PortC_Op1(Op1_Arg1, &Op1_Arg2);
        /* Transmit value to UDP-Panel */
        VT_TX(Op1_Arg2_VAL, Op1_Arg2);


        /*---- Write to RTE pPorts ----*/

        /* PortD_DataElement1 */
        UInt8 PortD_DataElement1 = 0;
        /* Read value from UDP-Panel */
        VT_RX(PortD_BTN, PortD_DataElement1, PortD_DataElement1_VAL);
        /* Explicit write to RTE */
        Rte_Write_PortD_DataElement1(PortD_DataElement1);

        /* PortG_DataElement1 */
        UInt8 PortG_DataElement1 = 0;
        /* Read value from UDP-Panel */
        VT_RX(PortG_BTN, PortG_DataElement1, PortG_DataElement1_VAL);
        /* Explicit write to RTE */
        Rte_Write_PortG_DataElement1(PortG_DataElement1);

        /* PortF_DataElement1 */
        UInt8 PortF_DataElement1 = 0;
        /* Read value from UDP-Panel */
        VT_RX(PortF_BTN, PortF_DataElement1, PortF_DataElement1_VAL);
        /* Implicit write to RTE */
        Rte_IWrite_RTERunnableCddSimulationTask_PortF_DataElement1(PortF_DataElement1
            );
}
```

Quellcodebeispiel 5.1: Generierter Quellcode für ein *Runnable* einer CDD-Simulation.

Abbildung 5.4: Beispiel eines generierten UDP-Panels für eine CDD-Simulation.

verbundenen Software-Komponenten. In die weiß hinterlegten Felder ist die Eingabe von Werten für die CDD-Simulation im zugelassenen Wertebereich möglich. Grau hinterlegte Felder stellen Werte dar, die über die Eingänge der CDD-Simulation empfangen wurden. Zusätzlich ist die Anbindung von Werkzeugen zur Testautomatisierung an das UDP-Panel möglich. Darüber kann sowohl die Eingabe von Parametern als auch die Auswertung der Werte der Software-Komponenten automatisiert werden. So lassen sich auch komplexe Abläufe über die CDD-Simulation darstellen.

5.1.3 CDD-Simulation durch Funktionsmodellierung

Alternativ zu der Simulation mit Benutzerinteraktion kann die Funktion des CDD auch in Grundzügen oder vollständig nachmodelliert werden. Um den Aufwand dafür so gering wie möglich zu halten, dient als Ausgangsbasis wiederum die AUTOSAR-Beschreibung der CDD-Simulation, welche durch den SWC-Generator automatisch erzeugt wurde. Viele Modellierungswerkzeuge wie z. B. MATLAB/Simulink verfügen über eine Importfunktion für ARXML-Dateien. Daher war für diesen Weg zur Erzeugung einer CDD-Simulation keine Realisierung zusätzlicher Werkzeuge notwendig. Der SWC-Generator ist wie im Fall der Simulation mit Benutzerinteraktion einsetzbar. Nach einem Import der AUTOSAR-Beschreibung in MATLAB/Simulink kann für jede enthaltene Software-Komponente ein Subsystem erzeugt werden. Dieser Funktionsblock erhält automatisch sämtliche *Ports*, die der Komponente angehören, inklusive der korrekten Schnittstellenbeschreibung. Abbildung 5.5 zeigt das Ergebnis des Imports für das Beispiel einer CDD-Simulation aus Kapitel 5.1.1.

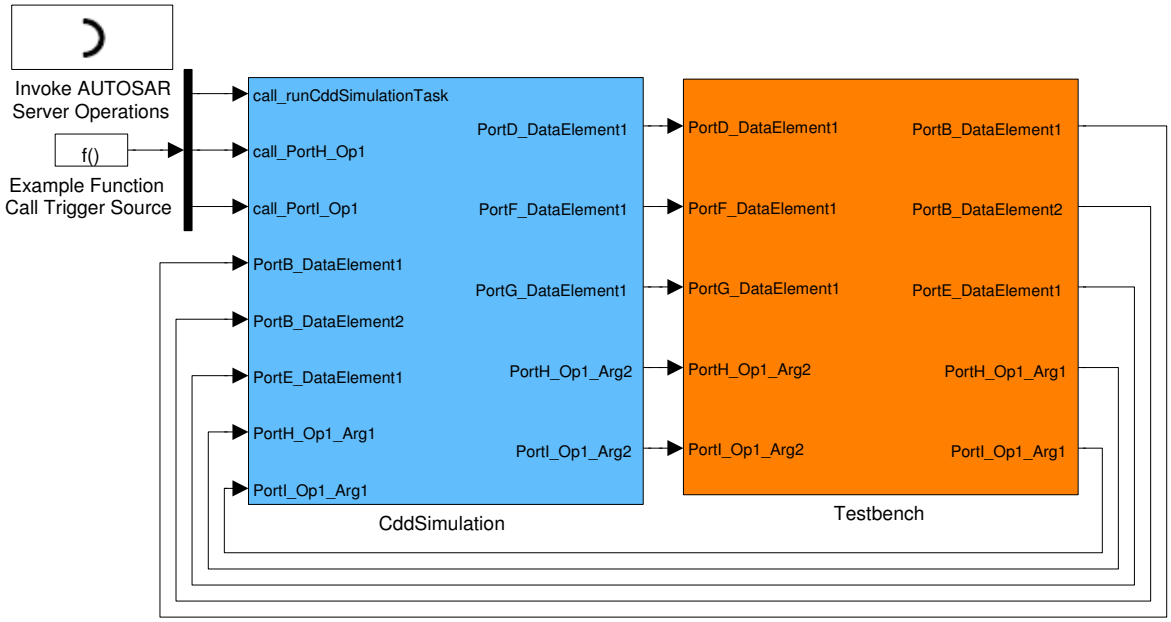


Abbildung 5.5: Darstellung einer Software-Komponente zur CDD-Simulation (blau) als Subsystem in MATLAB/Simulink mit zugehöriger Testkomponente (orange).

Alle *Runnables* werden wiederum anhand des *Internal-Behaviors* als einzelne Subsysteme innerhalb der zugehörigen SWC generiert. Nach der Definition ihrer Lese- und Schreibberechtigungen aus der AUTOSAR-Beschreibung erhält jedes *Runnable* eine Verbindung zu den

jeweiligen *Ports* der SWC. Das Ergebnis für das verwendete Beispiel der CDD-Simulation ist in Abbildung 5.6 dargestellt.

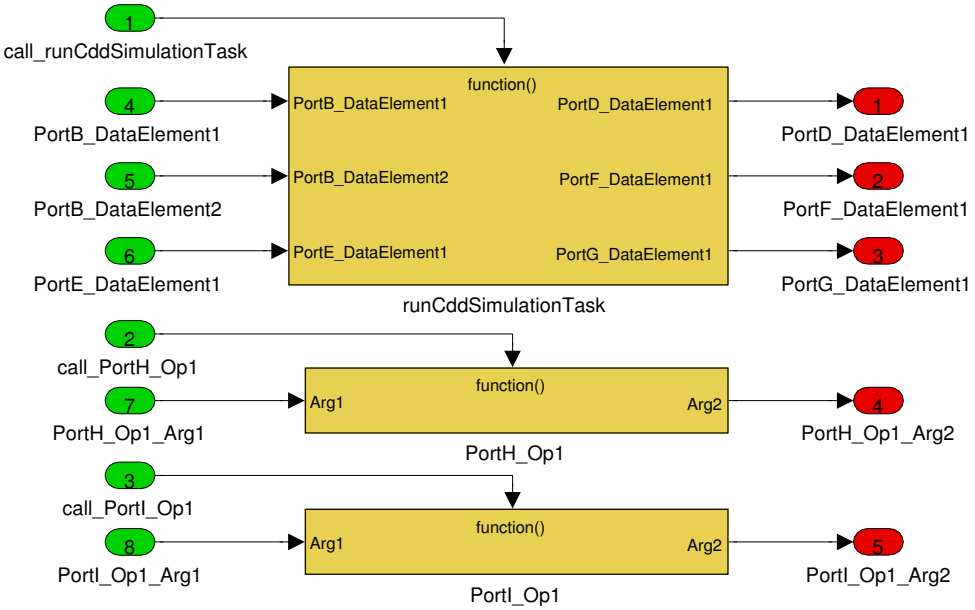


Abbildung 5.6: Darstellung der *Runnables* einer Software-Komponente als Subsysteme in MATLAB/Simulink.

Innerhalb dieses Grundgerüstes können nun sämtliche Funktionsblöcke von MATLAB/Simulink verwendet werden, um die nötige Funktionalität der CDD-Simulation zu modellieren. Dabei erlaubt MATLAB die Ausführung der Funktion direkt innerhalb des Programms, um eine isolierte Verifikation der modellierten Komponente zu ermöglichen. Sobald das gewünschte Verhalten vollständig abgebildet ist, muss aus dem Modell C-Code generiert werden. Dafür lässt sich der integrierte Embedded-Coder verwenden. Bei Änderungen an den Schnittstellen oder dem *Internal-Behavior* der Komponente kann die dadurch veränderte AUTOSAR-Beschreibung ebenfalls im ARXML-Format ausgeleitet werden. Die Beschreibungsdatei und der generierte C-Code sind, wie jede normale Software-Komponente, in die Steuergeräte-Software integrierbar. So kann der CDD nachgebildet werden, um die Schnittstellen der Software-Komponenten mit plausiblen Werten zu versorgen und ihre Funktion zu verifizieren.

5.1.4 Fazit

Mit der eigenen Entwicklung des SWC-Generators als Eclipse-Plug-in konnten alle Aspekte des Konzepts der CDD-Simulation aus Kapitel 4.1 erfolgreich umgesetzt werden. Die zugehörige Benutzeroberfläche erlaubt einen komfortablen Einsatz für Steuergeräte-Projekte mit beliebigen Complex-Device-Drivers. Durch die hochautomatisierte Erzeugung aller Artefakte für die Integration der Simulation wird der manuelle Aufwand bei dem Einsatz des Konzepts so gering wie möglich gehalten. In Kapitel 6 folgt die Evaluierung des Ansatzes anhand von industrierelevanten Beispielen, wobei die Funktionalität im praktischen Einsatz nachgewiesen wird.

5.2 CDD-Hardware-Abstraction-Layer

Das Konzept des CDD-HAL wurde im Rahmen dieser Arbeit ebenfalls prototypisch umgesetzt, wie in [DHM12c] veröffentlicht. Die Realisierung der vollständigen Konfigurierbarkeit des Hardware-Abstraction-Layers erfolgte in der zugehörigen Masterarbeit [Vog13]. Die Konfiguration des CDD-HAL wurde analog zu einem Modul der AUTOSAR-Basis-Software aufgebaut. Eine eigens entwickelte *Basic-Software-Module-Description (BSWMD)* enthält alle verfügbaren Konfigurationsoptionen, die in einem beliebigen AUTOSAR-Tool einstellbar sind. Ein eigener Generator, basierend auf Artop, erzeugt aus der Konfiguration den C-Code für den Hardware-Abstraction-Layer, der in die Steuergeräte-Software integrierbar ist. Im Folgenden wird die Implementierung der einzelnen Schichten des Konzepts und des Generators beschrieben.

5.2.1 Konfiguration und Implementierung

Wie bereits in Kapitel 4.2.2 dargestellt, untergliedert sich der CDD-HAL in vier verschiedene Schichten. Die Emulation ist für die Ankopplung in Richtung des CDD zuständig und die Hardware-Abstraktion dient als Bindeglied zu den Hardware-Komponenten der VAP. Das dazwischenliegende Routing erlaubt die flexible Zuordnung von CDD-Schnittstellen zu Hardware-Komponenten. Die Datenübertragung wird durch den Monitor ausgelöst, welcher die Signale an beiden Enden der Abstraktionsebene auf Änderungen überwacht. Jede dieser vier Schichten verfügt über eigene Möglichkeiten zur Konfiguration, welche über die BSWMD des Hardware-Abstraction-Layers definiert wurden.

Emulation

Die Konfiguration der Emulation muss exakt auf die Schnittstellen des Complex-Device-Drivers zur Hardware abgestimmt werden. Von allen Varianten von Hardwarezugriffen eines CDD (siehe Kapitel 2.2.1) trat bei den untersuchten Beispielen hauptsächlich der Registerzugriff auf. Daher wurde diese Zugriffsmöglichkeit für den Prototypen des CDD-HAL implementiert. In der Konfiguration der Emulationsschicht muss für jeden nachzubildenden Adressbereich jeweils ein Element vom Typ *Register* angelegt werden. Eine Beispielkonfiguration ist in Abbildung 5.7 dargestellt.
Jedes Register-Element verfügt über verschiedene Subelemente, welche seine Eigenschaften näher spezifizieren. Die folgenden Möglichkeiten wurden implementiert:

- **Name:** Gibt die Bezeichnung des Register-Elements an. Je nach *EmulationType* muss dieser Name der zugehörigen Repräsentation des Registers im Quellcode des CDD, als Makro oder Pointer, entsprechen. Der zugehörige Codegenerator verwendet den Parameter zur Auffindung und Manipulation der Register-Deklaration.
- **Address:** Definiert die Adresse des Registers im Adressraum der Ziel-Plattform in hexadezimaler Form. Diese Angabe wird nur benötigt, falls die *MemorySpaceEmulation* als *EmulationType* ausgewählt wird.
- **Size:** Deklariert die Größe des Registers in Bit. Dieser Wert muss exakt dem Register der Ziel-Architektur entsprechen, damit die Datenelemente korrekt gespeichert werden können.

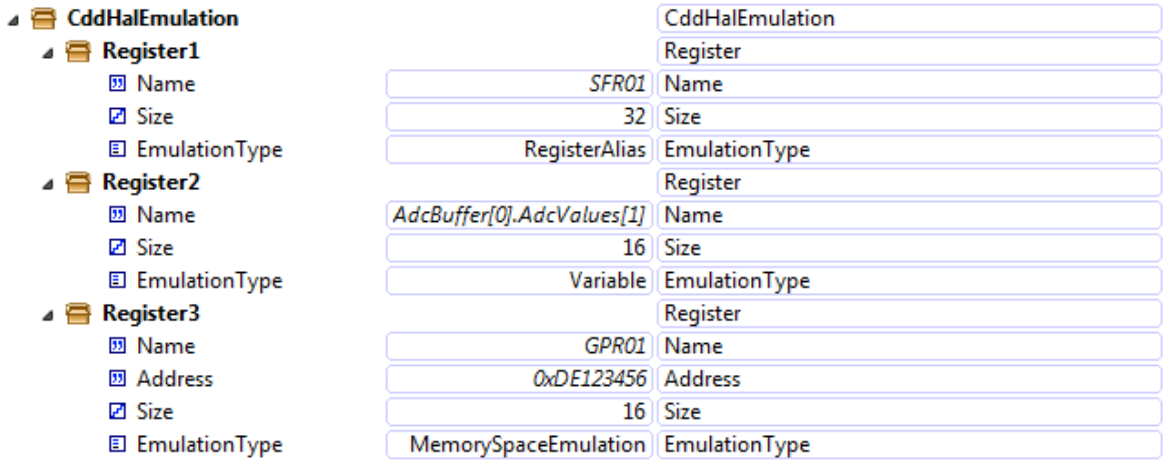


Abbildung 5.7: Beispielkonfiguration der Emulation.

- **EmulationType:** Bestimmt die Art der Emulation des Registers, die sich durch die Repräsentation im Quellcode des CDD unterscheiden. Wie bereits in Kapitel 2.2.1 beschrieben, kann ein Register per Makro, Pointer oder direkt über seine Adresse referenziert werden. In allen drei Fällen durchsucht der Codegenerator des CDD-HAL (siehe Kapitel 5.2.2) den Quellcode des Treibers nach der Deklaration des Registers anhand von regulären Ausdrücken. Je nach *EmulationType* wird dabei eine automatische Substitution der Deklaration ausgeführt, die dem zugehörigen Typ entspricht. Folgende Auswahlmöglichkeiten wurden implementiert:
  - **RegisterAlias:** Der Codegenerator manipuliert Präprozessormakros mit dem angegebenen *Name*, indem das Makro auf eine neue Variable innerhalb des CDD-HAL umgeleitet wird.
  - **Variable:** Es erfolgt eine Manipulation der Deklaration eines Pointers auf die Registeradresse mit dem angegebenen *Name*. Dabei wird die Adresse des Registers wiederum durch die Adresse einer neuen Variablen im CDD-HAL ersetzt.
  - **MemorySpaceEmulation:** Der Adressraum des Ziel-Mikrocontrollers wird vollständig in einem Array dargestellt. Dies ist möglich, da eine VAP mit deutlich mehr Arbeitsspeicher ausgestattet werden kann, als übliche Mikrocontroller. Unter Verwendung des *Address*-Elements ersetzt der Codegenerator alle direkten Aufrufe der Registeradresse durch ein Präprozessormakro, welches auf die korrespondierende Adresse des emulierten Adressraumes verweist. So wird der Zugriff auf das Array zur Adressraumemulation innerhalb des CDD-HAL umgeleitet.

Routing

Das Routing definiert die Verknüpfung aller Datenelemente aus der Emulation mit der Hardware-Abstraktion. Für jeden Parameter ist dabei zumindest eine 1:1-Verknüpfung nötig. Die Konfiguration von 1:n-Verknüpfungen ist jedoch genauso möglich, um z. B. den Wert eines Registers an mehrere Ausgabegeräte zu verteilen. Abbildung 5.8 zeigt eine Beispielkonfiguration.

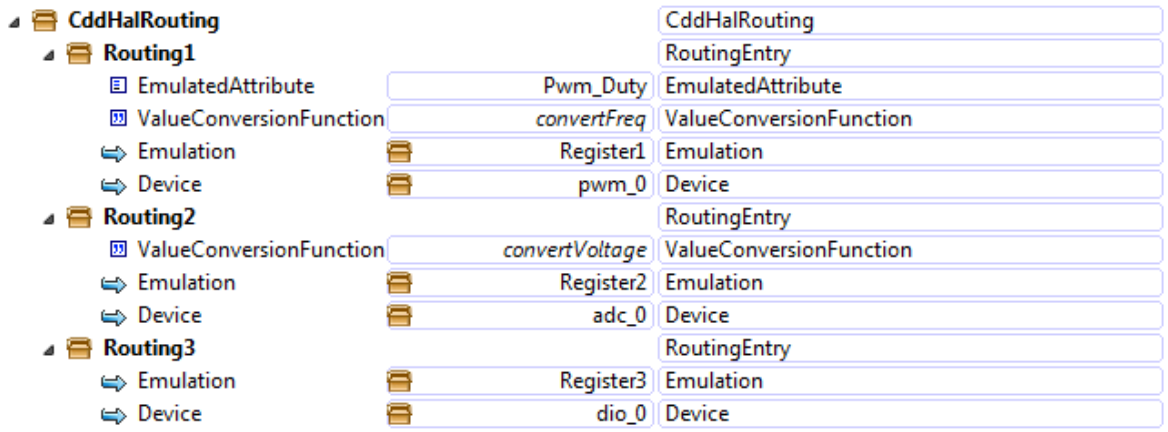


Abbildung 5.8: Beispielkonfiguration des Routings.

Für jede Verknüpfung muss ein *RoutingEntry*-Element definiert werden. Dieses verfügt über die folgenden Subelemente, welche sich je nach ausgewählter Hardware unterscheiden:

- **Emulation:** Stellt die Referenz auf ein Element der Emulation dar. Darüber wird die Verknüpfung zur Emulationsschicht definiert.

- **Device:** Gibt den Namen des Moduls der VAP-Hardware an, welches eingesetzt werden soll. Die Bezeichnung referenziert direkt ein Element aus der Konfiguration der Hardware-Abstraktion und stellt damit die Verknüpfung zu dem jeweiligen Gerät dar.

- **EmulatedAttribute:** Manche Geräte verfügen über mehrere Attribute, welche für die Datenübertragung definiert werden müssen. So sind z. B. bei einer PWM-Ausgabe das Tastverhältnis und die Periode des Signals festzulegen. Das *EmulatedAttribute* definiert, welches der verfügbaren Attribute bei einem *RoutingEntry* verwendet werden soll. Welche Auswahlmöglichkeiten zur Verfügung stehen, wird über die Konfiguration der Hardware-Abstraktion und den Gerätetyp bestimmt. Bei Geräten mit nur einem Attribut, wie z. B. einem digitalen Ausgang, ist das *EmulatedAttribute* obsolet.

- **ValueConversionFunction:** Die Wertebereiche eines bestimmten Gerätetyps können sich zwischen der Ziel-Plattform und einer VAP stark unterscheiden. Daher sind konkrete Werte meist nicht direkt übertragbar und müssen in einen anderen Wertebereich konvertiert werden. Für diesen Vorgang erlaubt die Konfiguration die Angabe einer Umrechnungsfunktion, welche vor der Datenübertragung durch das Routing ausgeführt wird. Solche Funktionen können manuell in einem definierten Bereich des CDD-HAL deklariert werden.

Der Codegenerator erzeugt für jeden *RoutingEntry* eine eigene Funktion im Quellcode, welche die Datenübertragung ausführt. Diese Funktionen werden entweder durch Interrupts von der Hardware oder durch einen Monitor gestartet. Die *RoutingEntrys* werden durch Elemente vom Typ *RoutingEntryRef* in der Konfiguration jedes Monitors referenziert.

Hardware-Abstraktion

Die Verbindung zur Peripherie der VAP erfolgt über die Hardware-Abstraktion. Diese Schicht dient der Ansteuerung der Linux-Treiber aller Module. Grundsätzlich können alle verfügbaren Schnittstellen einer VAP eingesetzt werden. Dabei erfordert jede neue Schnittstelle eine einmalige Implementierung ihrer Ansteuerung und der verfügbaren Parameter innerhalb des CDD-HAL. Danach ist die Komponente flexibel über die Konfiguration der Hardware-Abstraktion auf die Anforderungen eines CDD einstellbar. Eine Beispielkonfiguration ist in Abbildung 5.9 visualisiert.

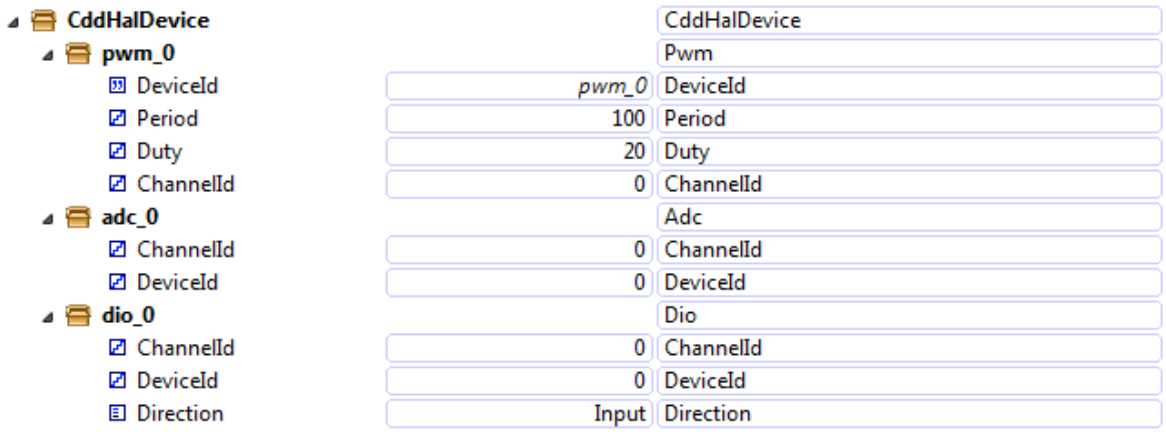


Abbildung 5.9: Beispielkonfiguration der Hardware-Abstraktion.

Hardware-Komponenten werden in der Konfiguration über Elemente eines jeweils eigenen Typs dargestellt, welcher die Art der Schnittstelle bezeichnet. Die zugehörigen Subelemente, und damit die Konfigurationsvarianten, unterscheiden sich ebenso. Häufig nutzen CDDs Analog-Digital-Converter (ADC), digitale Ein- und Ausgänge (Digital-Input/Output (DIO)) oder Pulsweitenmodulation (PWM), welche in einer VAP als C-PCI-Karten zur Verfügung stehen. Daher sollen im Folgenden die Subelemente dieser drei Schnittstellen erläutert werden:

- **DeviceId:** Bezeichnet den Index der jeweiligen Schnittstelle. Diese Information wird für die Linux-Treiber der C-PCI-Karten benötigt, da eine Karte über mehrere verschiedene Schnittstellentypen verfügen kann, die getrennt ansteuerbar sind.
- **ChannelId:** Selektiert den zu verwendenden Kanal einer Karte, z. B. den ersten oder zweiten digitalen Ausgang. Damit eine eindeutige Referenzierung eines Kanals für das Routing möglich ist, erfordert jeder genutzte Kanal ein eigenes Konfigurationselement in der Hardware-Abstraktion mit einzigartigem Namen.
- **Direction (nur für DIO):** Gibt die Übertragungsrichtung eines digitalen Ein- oder Ausgangs an. Als *Input* konfigurierte Kanäle können digitale Werte einlesen, während *Output* die Ausgabe digitaler Werte zulässt.
- **Duty (nur für PWM):** Bestimmt den Startwert für das Tastverhältnis eines PWM-Ausgangs. Es können Werte zwischen 0 % und 100 % eingegeben werden. Zur Laufzeit einer Applikation wird dieser Wert üblicherweise durch den CDD variiert und somit überschrieben.

- **Period (nur für PWM):** Legt den Startwert für die Periode eines PWM-Signals fest. Hierbei ist ebenfalls eine Variation des Wertes durch einen CDD zur Laufzeit einer Applikation möglich. Meist erfolgt bei PWM-Signalen allerdings nur eine Veränderung des Tastverhältnisses.

Monitor

Der Monitor dient zur Detektion von Signaländerungen an beiden Enden des CDD-HAL, der Emulation und der Hardware-Abstraktion. Dies ist notwendig, um auf neue Eingangsparameter der Peripherie zu reagieren und neue Daten des CDD an die Hardware weiterzuleiten. Bei Änderungen eines Wertes startet der Monitor die zugehörige Funktion des Routings und sorgt so für den Datenaustausch. Das Konzept aus Kapitel 4.2.2 beschreibt vier verschiedene Monitor-Varianten, welche jeweils in beliebiger Anzahl in der Konfiguration definierbar sind. Der Timed-Monitor hat sich als wichtigste Variante erwiesen, da er als einziger in der Lage zur Nachbildung von Interrupts ist, welche bei CDDs häufig genutzt werden. Abbildung 5.10 präsentiert eine Beispiel-Konfiguration.

CddHalMonitor		CddHalMonitor
TimedMonitor1		CddHalTimedMonitor
MonitorTiming	500000	MonitorTiming
MonitorCore	0	MonitorCore
EndHook	*Adc_TriggerFunc()*	EndHook
MonitorPriority	99	MonitorPriority
StartHook	*getPwmStatus()*	StartHook
RoutingEntryRef	Routing1	RoutingEntryRef
RoutingEntryRef	Routing2	RoutingEntryRef
RoutingEntryRef	Routing3	RoutingEntryRef

Abbildung 5.10: Beispielkonfiguration eines Timed-Monitors.

Jeder Monitor ist als Element eines eigenen Typs anzulegen, z. B. als *CddHalTimedMonitor*. Der Codegenerator erzeugt daraus jeweils einen eigenen Echtzeit-Task, der parallel zur Steuergeräte-Software und dem CDD ausgeführt wird. Es folgt eine Erläuterung der verfügbaren Subelemente des Timed-Monitors:

- **MonitorCore:** Erlaubt die optionale Zuweisung des Monitor-Tasks auf einen der Rechenkerne des VAP-Prozessors. Dies ist für die Nachbildung von Interrupts empfehlenswert, um ausreichend Rechenleistung auf einem wenig belasteten Kern zu garantieren.
- **MonitorPriority:** Bezeichnet die Priorität des Monitor-Tasks für das Scheduling des Echtzeitsystems. Dabei steht ein Wert von 0 für die niedrigste und 99 für die höchste Priorität.
- **MonitorTiming:** Bestimmt die Periodendauer in Nanosekunden, in welcher die Funktion eines Timed-Monitors zyklisch aufgerufen wird. Der Wert muss genau auf die Anforderungen des CDD abgestimmt sein und das nyquist-shannonsche Abtasttheorem (siehe Kapitel 4.2.2) berücksichtigen.
- **StartHook:** Erlaubt die Angabe einer sogenannten *User-Function*, welche durch den Monitor bei jedem Zyklus aufgerufen werden soll. Die *User-Function* ist in einem

definierten Bereich des CDD-HAL manuell implementierbar. Diese Methode erlaubt es z. B. Funktionen in Software nachzubilden, welche bei der Ziel-Plattform des Steuergerätes durch spezifische Hardware-Bestandteile realisiert sind und Eingangssignale weiterverarbeiten. Bei der Angabe der Funktion als *StartHook* erfolgt die Ausführung vor den Routing-Funktionen der *RoutingEntryRefs*.

- **EndHook:** Bezeichnet wie der *StartHook* eine *User-Function*, welche in jedem Zyklus des Monitors aufgerufen werden soll. Im Fall des *EndHook* erfolgt die Ausführung der Funktion jedoch nach den Routing-Funktionen der *RoutingEntryRefs*.
- **RoutingEntryRef:** Referenziert ein *RoutingEntry*-Element aus der Routing-Konfiguration. Für jede *RoutingEntryRef* wird die entsprechende Routing-Funktion in jedem Zyklus des Monitors aufgerufen und so die eigentliche Datenübertragung zwischen Emulation und Hardware-Abstraktion durchgeführt. Innerhalb eines Monitors sind mehrere *RoutingEntryRefs* definierbar, die nacheinander abgearbeitet werden.

5.2.2 Quellcodegenerierung und Integration

Die Module der AUTOSAR-Basis-Software verfügen jeweils über eigene Codegeneratoren. Diese erzeugen einen dynamischen Quellcodeanteil, welcher durch die gesetzten Konfigurationsparameter bestimmt wird. Nach demselben Prinzip wurde der CDD-HAL-Generator entwickelt. Die Realisierung erfolgte in Java [Ora15] und basiert auf der offenen Plattform Artop [RVE10]. Dies stellt sicher, dass der Generator nicht an einen spezifischen AUTOSAR-Stack gebunden ist. Die Implementierung des AUTOSAR-Metamodells von Artop erlaubt den einfachen Zugriff auf Elemente innerhalb von ARXML-Dateien. In dieser Form ist die Konfiguration aller BSW-Module und ebenso die des CDD-HAL gespeichert.
Wird der CDD-HAL-Generator ausgeführt, liest er zunächst alle Konfigurationsparameter von Emulation, Routing, Hardware-Abstraktion und Monitor ein. Dann erfolgt der Prozess der Codegenerierung, wobei alle Parameter bereits vor der Kompilierung direkte Auswirkungen auf den Code haben (sog. Pre-Compile-Configuration). Die einzelnen Schichten werden folgendermaßen repräsentiert:

- **Emulation:** In einer Iteration über alle Konfigurationselemente der Emulation erzeugt der Generator, z. B. für jedes definierte Register, eine passende Nachbildung mit dem angegebenen Namen und der Größe. Der *EmulationType* legt die genau Abbildung des Registers im Code fest (siehe Kapitel 5.2.1). Zusätzlich werden die Deklarationen der Register innerhalb des CDD-Quellcodes automatisch manipuliert, um die Zugriffe in die emulierten Register umzuleiten.
- **Routing:** Für jeden *RoutingEntry* in der Konfiguration des Routings erzeugt der CDD-HAL-Generator eine eigene Funktion, welche die Datenübertragung durchführt. Die angegebenen Referenzen auf die Elemente der Emulation und der Hardware-Abstraktion bestimmen die Richtung der Übertragung sowie die genutzten Variablen und Funktionsaufrufe.
- **Hardware-Abstraktion:** Anhand der Parameter der Hardware-Abstraktion werden die nötigen Aufrufe der Linux-Treiber jeder Hardware-Komponente generiert. Implementiert wurde die Anbindung für C-PCI-Karten einer VAP mit digitalen Ein- und Ausgängen, Analog-Digital-Wandlern und Pulsweitenmodulation.

- **Monitor:** Eigene Tasks im Echtzeit-Linux-System der VAP bilden die konfigurierten Monitore ab. Innerhalb jedes Tasks werden die Aufrufe aller referenzierten Routing-Funktionen angelegt, die bei jedem Ausführungszyklus ausgelöst werden. Der Typ des Monitors legt die Bedingung der Ausführung des Tasks fest, wie z. B. ein zyklischer Timer im Fall des Timed-Monitors.

Der gesamte erzeugte Quellcode des CDD-HAL-Generators wird an einem definierbaren Speicherort in Form von Quell- und Headerdateien abgelegt. Für die Integration des CDD-HAL zusammen mit dem Complex-Device-Driver und der Funktions-Software müssen alle Anteile noch kompiliert und gelinkt werden. Das so erzeugte Executable enthält damit alle relevanten Komponenten und ist auf einer VAP ausführbar. Die Hardware-Zugriffe des CDD, die eine universelle Plattform nicht unterstützt, werden über den CDD-HAL auf die verfügbare VAP-Hardware umgeleitet.

5.2.3 Fazit

Alle Elemente des CDD-Hardware-Abstraction-Layers konnten anhand des Konzepts aus Kapitel 4.2 erfolgreich in eigenen Modulen umgesetzt werden. Die Konfigurationsmöglichkeit des HAL ließ sich analog zu AUTOSAR-Basis-Software-Modulen realisieren, wodurch die hohe Flexibilität der Lösung sichergestellt ist. Durch die nahtlose Integration in die AUTOSAR-Toolkette kann der Hardware-Abstraction-Layer komfortabel eingesetzt werden. Eine Evaluierung des Konzepts anhand von industrierelevanten Beispielen, inklusive dem Nachweis der Funktionalität in der Praxis, folgt in Kapitel 6.

5.3 Zusammenfassung

Im Rahmen dieser Dissertation und der zugehörigen Masterarbeiten [Arr12, Vog13] wurden die vorgestellten Konzepte der CDD-Simulation und des CDD-Hardware-Abstraction-Layers erfolgreich realisiert.
Für die CDD-Simulation entstand der SWC-Generator, der ein neues Artop-Plugin darstellt und alle Funktionen des Konzepts umsetzt. Dieses Werkzeug analysiert die Schnittstellen von Software-Komponenten und erzeugt automatisiert die AUTOSAR-Beschreibung einer passenden Gegenkomponente, die das Grundgerüst einer CDD-Simulation ist. Zusätzlich wird passender C-Code generiert, welcher für alle Datenelemente eine Anbindung an eine grafische Oberfläche beinhaltet. Dieses automatisch erzeugte Panel erlaubt die Eingabe von Testvektoren und die Verifikation von Rückgabewerten der Software-Komponenten während der Laufzeit der AUTOSAR-Applikation. Alternativ ist die generierte AUTOSAR-Beschreibung der CDD-Simulation z. B. in MATLAB/Simulink einlesbar, um die CDD-Funktionalität nachzumodellieren. Beide Varianten erlauben die vollständige Absicherung der Schnittstellen zwischen einem Complex-Device-Driver und den Software-Komponenten.
Die Realisierung des CDD-HAL erfolgte nach dem Vorbild eines Moduls der AUTOSAR-Basis-Software. Für die vier Schichten Emulation, Routing, Hardware-Abstraktion und Monitor wurde jeweils eine Konfigurationsmöglichkeit entwickelt, die eine flexible Anpassung auf die Anforderungen eines CDD erlaubt. Eine eigene *Basic-Software-Module-Description* beinhaltet alle verfügbaren Parameter. Der entwickelte CDD-HAL-Generator erzeugt anhand der Konfigurationsparameter den nötigen C-Code der Abstraktionsschicht. Zusammen mit dem CDD und der Funktions-Software kann der generierte Quellcode des CDD-HAL auf

einer VAP kompiliert und zu einer ausführbaren Datei gelinkt werden. Die Umleitung der CDD-Schnittstellen auf die VAP-Hardware erlaubt somit die vollständige Integration und Ausführung des Treibers auf einem System zur virtuellen Absicherung.

Kapitel 6

Evaluierung der Konzepte anhand industrierelevanter Projekte

Die prototypischen Realisierungen der vorgestellten Konzepte der CDD-Simulation und des CDD-HAL wurden anhand von Complex-Device-Drivers aus industrierelevanten Steuergeräte-Projekten erprobt. Dabei sollte sowohl eine Evaluierung der Konzepte als auch der Methodik zur Selektion des geeignetsten Konzepts (siehe Kapitel 3.4) erfolgen. Die eingesetzten Steuergeräte-Projekte umfassen eine Außenlichtsteuerung, den Treiber eines Head-Up-Displays und die Regelung eines Elektromotors. Bei der Projektauswahl wurde sichergestellt, dass jedes der drei Testziele für die Berücksichtigung von CDDs in der virtuellen Absicherung aus Kapitel 3.1 abgedeckt wurde.

6.1 Außenlichtsteuerung

Die Außenbeleuchtung moderner Fahrzeuge hat sich in den letzten Jahren stark weiterentwickelt. Neue Technologien, wie leistungsstarke Leuchtdioden oder Laserlicht, ermöglichen bessere Sichtverhältnisse und neue Funktionen, wie z. B. das gezielte Anleuchten von Fußgängern auf der Fahrbahn oder einen blendfreien Fernlichtassistenten. Die Regelung der gesamten Beleuchtung erfolgt dabei elektronisch und ist funktional über mehreren Komponenten wie Bedieneinheiten, Helligkeitssensoren, Aktivierungslogik und Leistungselektronik verteilt.

6.1.1 Projektbeschreibung

Bei der BMW Group sind die wesentlichen Lichtsteuerungsumfänge in einem zentralen Karosseriesteuergerät integriert, das baureihenübergreifend in verschiedenen Fahrzeugen zum Einsatz kommt. Die Funktion gliedert sich in drei Software-Komponenten sowie einen CDD. Der Treiber entstand bei einem Zulieferer und enthält dessen geschütztes Knowhow, weshalb er für die Integration nicht im Quellcode zur Verfügung stand. Dennoch war es notwendig, die Software-Komponenten virtuell abzusichern und den CDD zu berücksichtigen, da die Funktions-Software und der Treiber über verschiedene Schnittstellen für den Datenaustausch untereinander verfügen. Abbildung 6.1 stellt die Architektur der Außenlichtsteuerung vereinfacht dar.

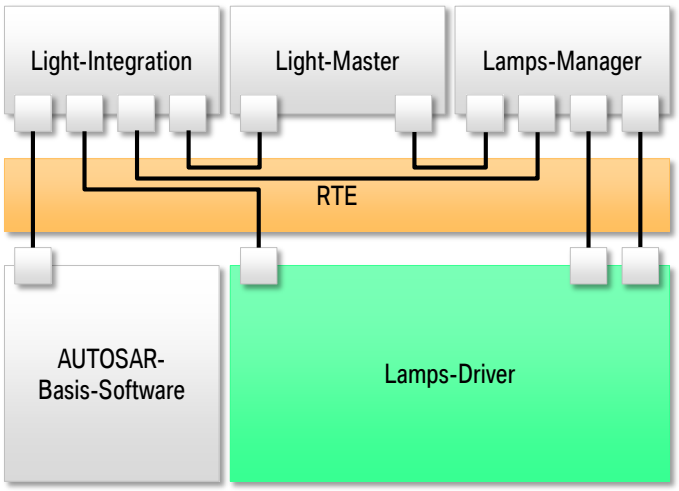


Abbildung 6.1: Aufbau der Außenlichtsteuerung.

Die Funktionen der einzelnen Komponenten werden im Folgenden beschrieben:

- **Light-Master:** Enthält die funktionale Logik der Außenlichtsteuerung. Die Komponente wurde vollständig modellbasiert in MATLAB/Simulink entwickelt. Anhand der Bedienung der verschiedenen Schalter und Taster für Lichtfunktionen im Fahrzeug sowie der Umgebungsbedingungen und Einstellungen wird der gewünschte Zustand der Außenbeleuchtung festgelegt.
- **Lamps-Manager:** Verwaltet den Status der einzelnen Leuchten. Über die Verknüpfung zum Lamps-Driver wird z. B. festgestellt, ob sämtliche Leuchtelemente noch funktionsfähig sind, oder ob ein Fehler vorliegt.
- **Light-Integration:** Stellt eine zusätzliche Abstraktionsebene dar. Der Datenaustausch der beiden anderen SWCs mit der Basis-Software läuft vollständig über die Integrations-Komponente. Dies beinhaltet z. B. Signale, welche über die Bussysteme übertragen werden.
- **Lamps-Driver:** Übernimmt als Complex-Device-Driver die direkte Ansteuerung der Elektronik der Leuchten. Der Treiber kontrolliert sogenannte SPI-Power-Controller (SPOC) [Inf14], worüber die Außenbeleuchtung geregelt wird. Die Leistungselektronik wird über SPI vom Mikrocontroller angesteuert und erzeugt individuelle PWM-Signale für jedes Leuchtelement. Zusätzlich erfolgen z. B. Überprüfungen der anliegenden Spannungen, um einen fehlerfreien Betrieb der Leuchten sicherzustellen. Die Funktionen wurden als CDD implementiert, da das Standard-AUTOSAR-SPI-Modul für den vollen Funktionsumfang nicht ausreichend war.

6.1.2 Konzeptauswahl

Für die Integration von Steuergeräte-Software mit Berücksichtigung des Complex-Device-Drivers auf einer VAP wurde die Methodik aus Kapitel 3.4 eingesetzt. Für die Außenlichtsteuerung galten die folgenden Prämissen:

- **CDD-Anforderungen:** Der Lamps-Driver verfügt über direkte Hardware-Schnittstellen für die Ansteuerung und Überprüfung der Leistungselektronik. Da die VAP in der verwendeten Form über keine SPI-Schnittstelle verfügt, ist die Hardware für die direkte Integration des CDD nicht ausreichend. Es handelt sich somit um den Fall $\alpha = 2$ aus Tabelle 3.6.

- **Testziel:** Durch die Integration der Software-Komponenten der Außenlichtsteuerung sollte die Absicherung der Schnittstellen der Funktions-Software ermöglicht werden. Nach der Nummerierung aus Tabelle 3.6 handelt es sich somit um das Testziel $\vartheta = 1$.

Aus den Werten für α und ϑ ergibt sich nach dem Schema $A_{\alpha\vartheta}$ der Anwendungsfall A_{21} der Methodik. Daher wurde für die Integration der Außenlichtsteuerung, wie für den Anwendungsfall A_{21} vorgesehen, das Konzept der CDD-Simulation eingesetzt (siehe Abbildung 6.2).

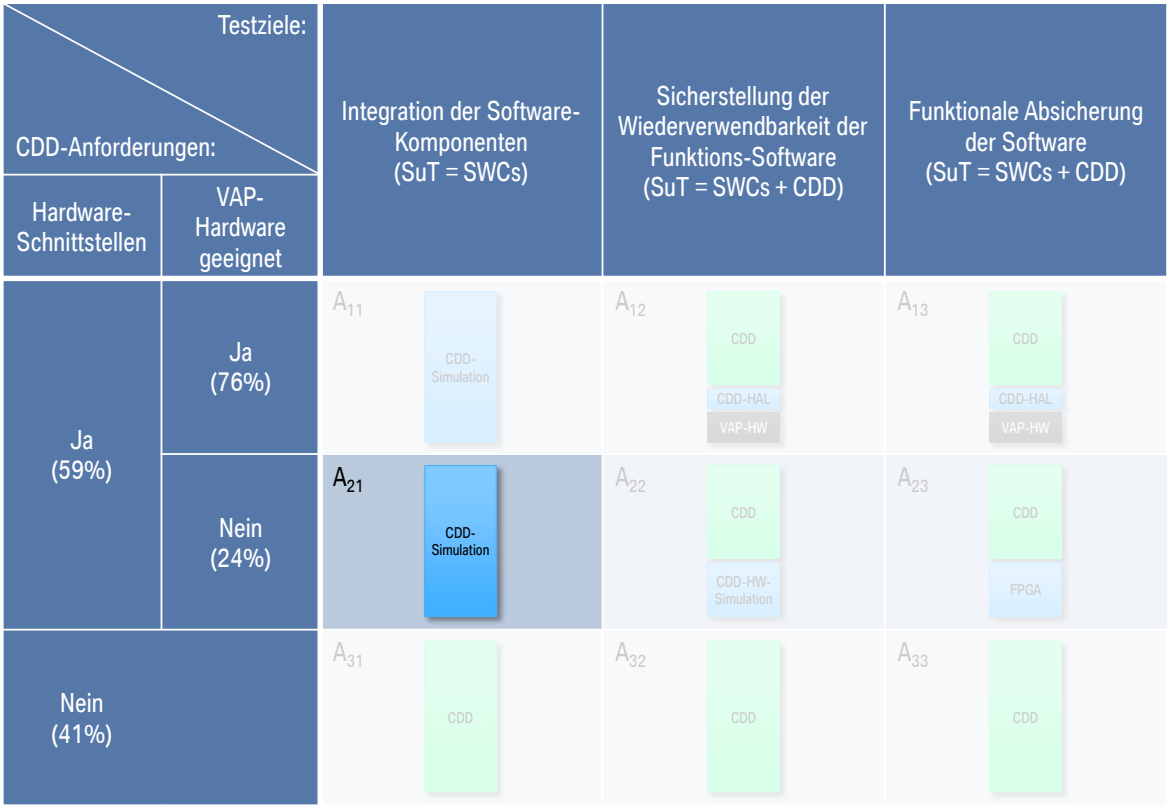


Abbildung 6.2: Konzeptauswahl für den Anwendungsfall der Außenlichtsteuerung.

6.1.3 Konzeptanwendung

Die Konzeptanwendung erfolge im Rahmen der zugehörigen Masterarbeit [Arr12]. Dabei wurden beide Varianten des Konzepts (siehe Kapitel 4.1) erprobt, die vollständige Generierung einer CDD-Simulation mit Benutzerinteraktion und die Funktionsmodellierung in MATLAB/Simulink.

CDD-Simulation mit Benutzerinteraktion

Als Ausgangspunkt stand eine *Composition* mit den drei Software-Komponenten der Außenlichtsteuerung zur Verfügung. Der CDD selbst und seine AUTOSAR-Beschreibung waren nicht erhältlich. Durch die Anwendung des SWC-Generators (siehe Kapitel 5.1.2) wurden automatisch alle offenen Schnittstellen der Komponenten identifiziert, welche in der vollständigen Software eine Verbindung zum CDD herstellen sollten. Aus den extrahierten Interface-Daten konnte das Tool erfolgreich eine AUTOSAR-Beschreibung für die CDD-Simulation erzeugen und kompatible *Ports* mit invertierter Übertragungsrichtung generieren. Ein adäquates *Internal-Behavior* mit eigenen *Runnables* für die Simulation wurde ebenso vollautomatisch produziert. So konnte mit dem SWC-Generator eine schnittstellenkompatible Nachbildung der AUTOSAR-Beschreibung des Lamps-Drivers generiert werden.

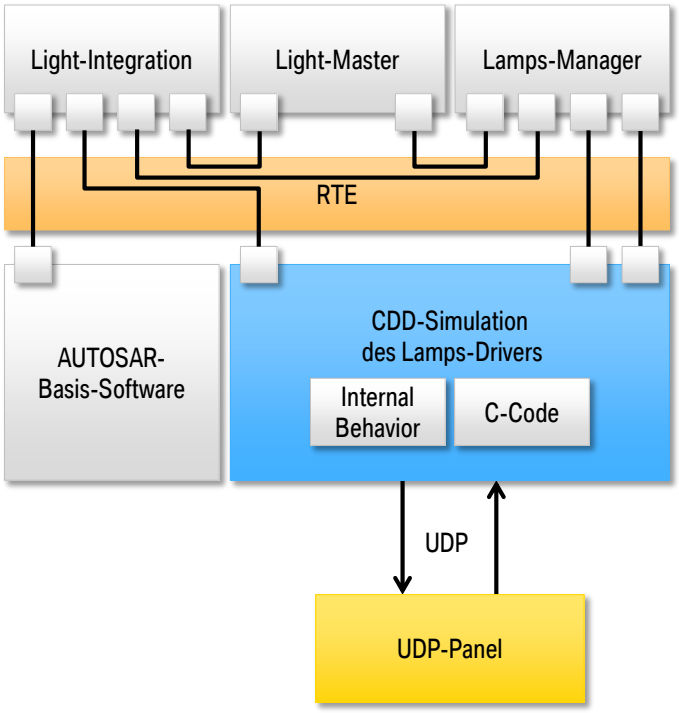


Abbildung 6.3: Integration der vollständig generierten CDD-Simulation mit den Software-Komponenten der Außenlichtsteuerung.

Zusätzlich erzeugte der Generator den vollständigen C-Code der CDD-Simulation, nach dem Verfahren aus Kapitel 5.1.2. Ein UDP-Panel zur Benutzerinteraktion ließ sich ebenfalls erfolgreich generieren, welches die vollständigen Schnittstellen des Lamps-Drivers abbildet. Die Anbindung des Panels an die Simulation erfolgte durch die vollautomatische Integration der UDP-Verbindung in den C-Code der Simulationskomponente. Abbildung 6.3 zeigt die Integration der CDD-Simulation des Lamps-Drivers mit allen Komponenten der Außenlichtsteuerung schematisch. Auf eine exakte Darstellung der Schnittstellen und somit auch des UDP-Panels muss aus Geheimhaltungsgründen verzichtet werden.

CDD-Simulation durch Funktionsmodellierung

Die zweite Variante zur Erzeugung einer CDD-Simulation wurde ebenfalls anhand der Außenlichtsteuerung getestet. Als Ausgangspunkt diente die, durch den SWC-Generator produzierte, AUTOSAR-Beschreibung der CDD-Simulation des Lamps-Drivers. Die Schnittstellen und Runnables ließen sich erfolgreich nach der Beschreibung aus Kapitel 5.1.3 in MATLAB/Simulink importieren. Daraus resultierte eine Darstellung aller *Runnables* als Subsysteme in Simulink, mit den korrekten Schnittstellenbeschreibungen.

Für den Lamps-Driver stand kein fertiges Modell zur Verfügung, da der Treiber nicht modellbasiert entwickelt wurde. Daher musste die Funktionalität über das Verhalten der Software-Komponenten und die Analyse von Lastenheften nachgebildet werden. Der Fokus lag darauf, die korrekte Ausführung der Software-Komponenten sicherzustellen, weshalb der CDD nicht vollständig nachgebildet werden musste. Abbildung 6.4 zeigt den Modellierungs- und Integrationsvorgang schematisch für die Außenlichtsteuerung.

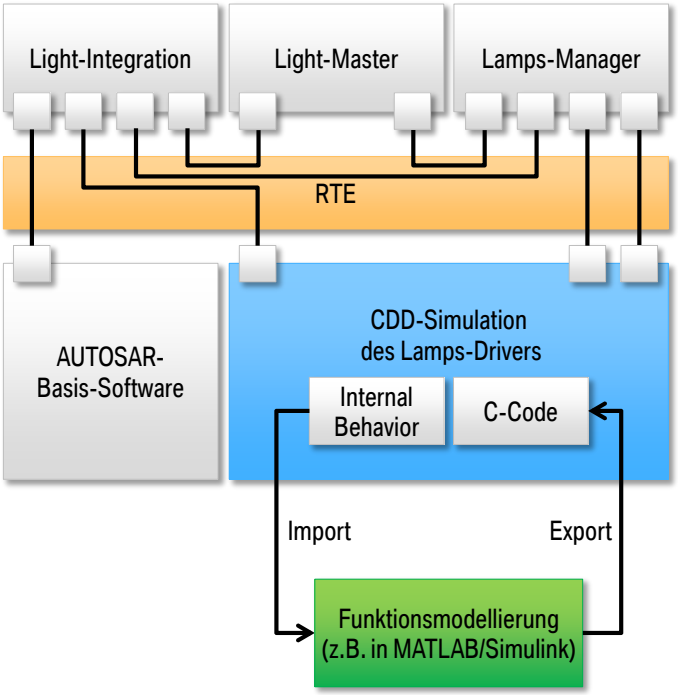


Abbildung 6.4: Integration der nachmodellierten CDD-Simulation mit den Software-Komponenten der Außenlichtsteuerung.

In Simulink konnte das Modell durch die Simulationsmöglichkeiten des Tools bereits verifiziert werden. Für die Integration in einer VAP, zusammen mit den Software-Komponenten der Außenlichtsteuerung, wurde der Embedded Coder von MATLAB eingesetzt. Durch diesen Codegenerator konnte erfolgreich der Quellcode der CDD-Simulation erzeugt werden. Der modellbasierte Ansatz mit Import der AUTOSAR-Beschreibung stellte dabei automatisch die Kompatibilität der Schnittstellen sicher und ermöglichte eine deutlich schnellere Implementierung als eine manuelle Programmierung der Simulation. Zusammen mit dem Quellcode

der Software-Komponenten der Außenlichtsteuerung ließ sich die CDD-Simulation erfolgreich kompilieren und zu einer ausführbaren Datei linken.

6.1.4 Versuchsaufbau

Um die Funktion der Software-Komponenten im Verbund zu testen, mussten plausible Testvektoren erzeugt werden. Als Eingangswerte benötigen die Außenlichtfunktionen verschiedene Parameter, wie den Klemmenstatus des Fahrzeugs oder die Position der Lichtschalter für Abblendlicht, Nebelscheinwerfer und Nebelschlussleuchte. Das Schaltzentrum an der Lenksäule beeinflusst den Status des Fernlichtes, bzw. der Lichthupe und der Blinker. Zusätzlich ermittelt ein Helligkeitssensor die aktuelle Umgebungsausleuchtung und überträgt den Wert an die Außenlichtfunktion. All diese Datenelemente werden in Form von Signalen über die Bussysteme CAN und LIN an das zentrale Karosseriesteuergerät übermittelt, welches im Fahrzeug die Außenlichtsteuerung beinhaltet.
Für den Versuchsaufbau wurde die Konfiguration der AUTOSAR-Basis-Software zur Vereinfachung so ausgelegt, dass sich alle relevanten Signale auf demselben CAN-Bus befinden. Mithilfe einer eigenen Restbussimulation waren alle benötigte Signale der Bedienelemente und der anderen Steuergeräte auf einem Messrechner erzeugbar und ließen sich über den CAN-Bus senden. Die Interaktion während der Programmausführung zur Eingabe von Testvektoren für die Signalwerte erfolgte über eine graphische Benutzeroberfläche. Die Außenlichtsteuerung, die CDD-Simulation und die AUTOSAR-Basis-Software wurden auf einer VAP ausgeführt und mit dem CAN der Restbussimulation verbunden. Der Versuchsaufbau ist in Abbildung 6.5 dargestellt und eignet sich für beide Varianten der CDD-Simulation.

6.1.5 Evaluierung

Die Software-Komponenten der Außenlichtsteuerung verfügen insgesamt über neun Schnittstellen, welche Daten mit dem Lamps-Driver austauschen. Es handelt sich sowohl um *Sender-Receiver-* als auch *Client-Server-Interfaces*, die zusammen 43 Datenelemente übertragen. Der entwickelte SWC-Generator konnte, durch die Analyse der Software-Komponenten, alle Schnittstellen des Lamps-Drivers automatisiert in einer CDD-Simulation nachbilden. Die unterschiedlichen Kommunikationsmechanismen und Datentypen ließen sich dabei vollständig berücksichtigen. Dank der lückenlosen Automatisierung des Konzepts war die Variante der CDD-Simulation mit Benutzerinteraktion in wenigen Minuten erzeugbar. Das UDP-Panel, welches alle 43 übertragbaren Datenelemente des Lamps-Drivers darstellen konnte, diente der Verifikation aller Schnittstellen zwischen den Software-Komponenten und dem Treiber. Das Panel ermöglichte die Eingabe von Testvektoren für die CDD-Simulation, deren korrekte Übertragung zu den Software-Komponenten über Debug-Ausgaben sichergestellt wurde. Ebenso ließen sich die Software-Komponenten über die Restbussimulation mit Testvektoren stimulieren, wobei die Analyse der resultierenden Übertragungswerte zu dem CDD über die Anzeige des UDP-Panels erfolgte. Mit diesem Vorgehen konnten alle Schnittstellen der Außenlichtsteuerung zwischen dem Lamps-Driver und der Funktions-Software erfolgreich verifiziert werden.
Darüber hinaus erfolgte eine funktionale Verifikation der Software-Komponenten. Dazu wurden dieselben Testfälle genutzt, die für die Absicherung der Außenlichtsteuerung am realen Fahrzeug zum Einsatz kommen. Diese beschreiben die Betätigung der verschiedenen Bedienelemente, wie den Lichtschalter für das Abblendlicht oder den Blinkerhebel, im Zusam-

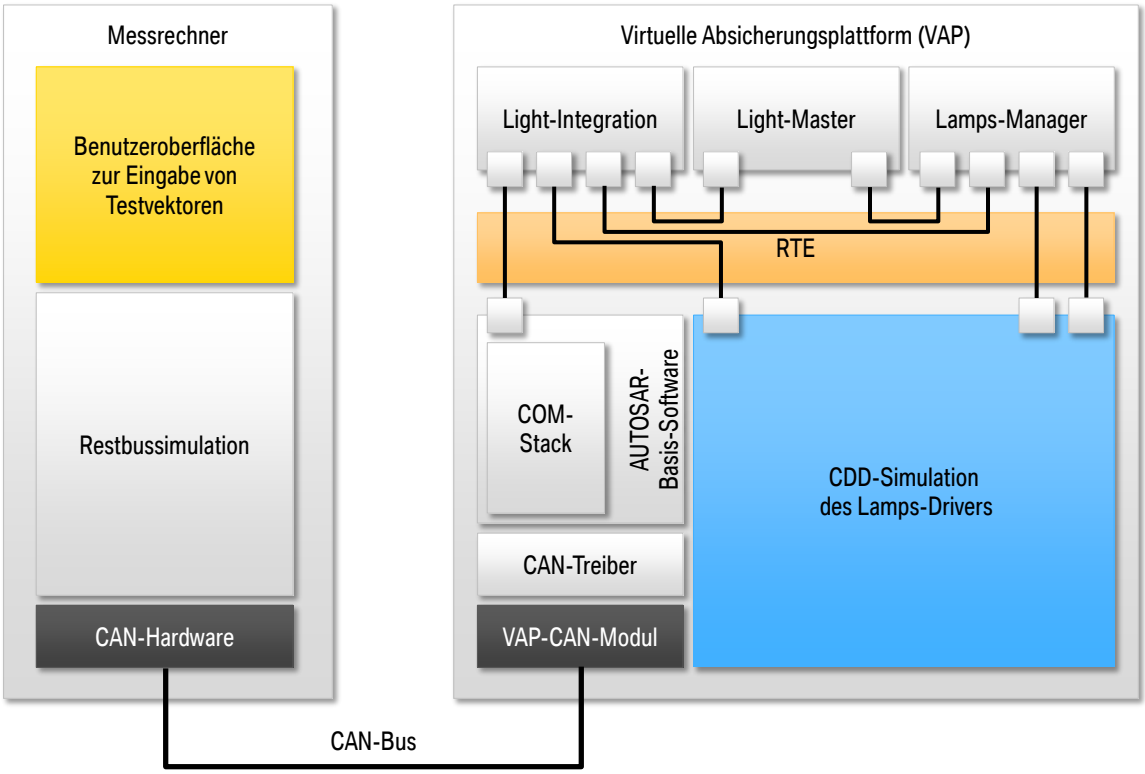


Abbildung 6.5: Schematischer Versuchsaufbau für die Integration der Außenlichtsteuerung mit Restbussimulation.

menhang mit Umgebungsbedingungen wie der Helligkeit sowie das erwartete Verhalten der Funktion. Einer der Testfälle beschreibt z. B., dass bei aktiver Zündung, Dunkelheit und dem Lichtschalter in Automatikstellung das Abblendlicht aktiviert werden muss. Ist gleichzeitig die Nebelschlussleuchte eingeschaltet, ist sie wieder zu deaktivieren, sobald der Lichtschalter in die Ausgangsposition gedreht wird (siehe Tabelle 6.1). In einem weiteren Beispiel dürfen die Rückfahrscheinwerfer nicht aktiviert werden, wenn der Rückwärtsgang bei aktiver Zündung für weniger als 300 ms eingelegt bleibt. Insgesamt existieren bei der BMW Group mehr als 100 Testfälle dieser Art für die statischen Außenlichtfunktionen sowie rund 90 Testfälle für die Blinkerfunktion, inklusive unterschiedlicher Varianten für sämtliche Baureihen und Länder.

Testschritt	**Beschreibung**
Voraussetzungen	- Zündung eingeschaltet. - Schalter Bedieneinheit Licht auf Position „Automatik“. - Lichtsensor erkennt Dunkelheit. - Abblendlicht aktiv. - Nebelschlusslicht aktiv.
Aktion	- Schalter Bedieneinheit Licht auf Position „0“ stellen.
Ergebnis	- Abblendlicht inaktiv. - Nebelschusslicht inaktiv.

Tabelle 6.1: Beispiel eines Testfalls der Außenlichtsteuerung.

Für die Testdurchführung auf der VAP wurde ein Subset von 20 Testfällen ausgewählt, welches sämtliche Bedien- und Leuchtelemente repräsentativ abdeckt. Die im Versuchsaufbau eingesetzte Restbussimulation (siehe Kapitel 6.1.4), erlaubte die virtuelle Betätigung aller Bedienelemente. Über beide Varianten der CDD-Simulation des Lamps-Drivers ließ sich der Zustand der Leuchtelemente darstellen und an die Software-Komponenten der Außenlichtsteuerung übermitteln. Die Verifikation der Testergebnisse erfolgte durch die Signale, welche von den Software-Komponenten an die CDD-Simulation rückgemeldet wurden sowie durch die übertragenen Werte auf dem angeschlossenen CAN-Bus. Die 20 Testfälle ließen sich durch beide Varianten der CDD-Simulation erfolgreich durchführen und die Funktionen der Außenlichtsteuerung konnten bestätigt werden. Tabelle 6.2 enthält eine kurze Beschreibung der durchgeführten Testfälle und dokumentiert die zugehörigen Ergebnisse.

Testfallbeschreibung	**Ergebnis**
Aktivierung/Deaktivierung Tagfahrlicht über iDrive-System.	Bestanden ✓
Aktivierung/Deaktivierung Tagfahrlicht bei Motorstart.	Bestanden ✓
Aktivierung Nebelscheinwerfer bei Bedieneinheit Licht auf Position „Automatik“.	Bestanden ✓
Aktivierung Nebelschlussleuchte bei Bedieneinheit Licht auf Position „Automatik“, unabhängig vom Helligkeitswert.	Bestanden ✓
Automatische Deaktivierung Nebelscheinwerfer bei Deaktivierung Abblendlicht.	Bestanden ✓
Automatische Deaktivierung Fernlicht bei Deaktivierung Abblendlicht.	Bestanden ✓
Rückfahrlicht bleibt bei kurzzeitigem einlegen des Rückwärtsganges deaktiviert.	Bestanden ✓
Aktivierung/Deaktivierung Fernlicht bei aktivem Abblendlicht.	Bestanden ✓
Fernlicht bei Bedieneinheit Licht auf Position „Automatik“ und Tag nicht aktivierbar.	Bestanden ✓
Fernlicht bei Bedieneinheit Licht auf Position „Automatik“ und Nacht aktivierbar.	Bestanden ✓
Automatische Deaktivierung Nebenschlussleuchte bei Deaktivierung Nebelscheinwerfer.	Bestanden ✓
Automatische Aktivierung Rückfahrlicht wenn Rückwärtsgang eingelegt.	Bestanden ✓
Aktivierung/Deaktivierung Lichthupe über Lenkstocktaster.	Bestanden ✓
Deaktivierung Tagfahrlicht beim Öffnen der Fahrertür im Stand.	Bestanden ✓
Automatische Deaktivierung von Abblendlicht und Fernlicht bei Bedieneinheit Licht auf Position „Automatik“ und Tag.	Bestanden ✓
Aktivierung Fernlicht über Lenkstocktaster nicht möglich bei Bedieneinheit Licht auf Position „0“.	Bestanden ✓
Dauerblinken bleibt bei Motorstart und -stop aktiv.	Bestanden ✓
Aktivierung Tippblinken über Lenkstocktaster.	Bestanden ✓
Aktivierung Dauerblinken über Lenkstocktaster.	Bestanden ✓
Deaktivierung Dauerblinken über Lenkstocktaster.	Bestanden ✓

Tabelle 6.2: Durchgeführte Testfälle für die Außenlichtsteuerung mit zugehörigen Ergebnissen.

6.2 Head-Up-Display-Treiber

Ein Head-Up-Display (HUD) ermöglicht die Anzeige von Informationen im Blickfeld des Fahrers. Die aktuelle Geschwindigkeit, Navigationshinweise oder Meldungen von Fahrerassistenzsystemen werden direkt in die Frontscheibe projiziert. Ein Display mit starker Hintergrundbeleuchtung übernimmt die Darstellung der Informationen und eine Spiegelvorrichtung lenkt den Strahlengang zur optimalen Anzeigeposition. Durch die optische Weglänge der Projektion erscheint die Anzeige weiter entfernt vor der Windschutzscheibe. Dies verhindert, dass die Augen bei einem Blick auf die eingeblendeten Informationen auf den Nahbereich fokussiert werden müssen. Der schematische Aufbau eines Head-Up-Displays ist in Abbildung 6.6 mit den wesentlichen Komponenten dargestellt.

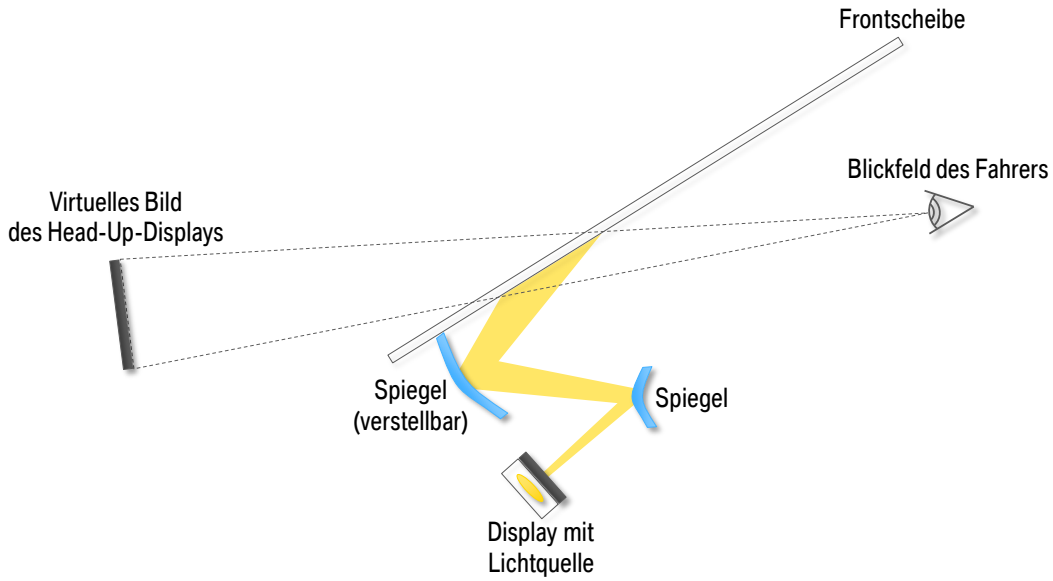


Abbildung 6.6: Schematischer Aufbau eines Head-Up-Displays.

6.2.1 Projektbeschreibung

Die Hardware des Head-Up-Displays der BMW Group besteht im Wesentlichen aus einer Lichtquelle, einem Farbdisplay, zwei Spiegeln und einem Mikrocontroller. Einer der Spiegel ist über einen Schrittmotor verstellbar, um die Höhe der Projektion in der Frontscheibe anzupassen. Die enthaltene Software des HUD ist für den Empfang von Videodaten und die Weitergabe an das Display zuständig. Weiterhin wird die Ansteuerung des Schrittmotors, die Regelung der Lichtquelle und die Kommunikation zur Instrumenten-Kombination übernommen.
Als Schnittstelle dient der Automotive-Pixel-Link (APIX), welcher die Übermittlung von Video, Audio und Kommunikationsdaten mit bis zu 3 Gbit/s über zwei abgeschirmte Twisted-Pair-Leitungen erlaubt. Ein Leitungspaar dient als *Downstream* und überträgt die Video, Audio und Kommunikationsdaten vom Sender zum Empfänger. Das zweite Leitungspaar leistet als *Upstream* die Übermittlung von Kommunikationsdaten in die Gegenrichtung. Die

Kommunikation erfolgt über ein Seitenband, auf dem ein gesicherter Austausch von Signalen durch die APIX-Automotive-Shell (AShell) erfolgt, welche eine Abstraktionsschicht für den Datenaustausch darstellt [Ino10].

Die Ansteuerung des Displays ist in der Instrumenten-Kombination integriert. Dort werden die Anzeigeelemente auf der zugehörigen Grafikeinheit des Steuergerätes berechnet und an das Head-Up-Display zur Darstellung übertragen. Die Steuerungsfunktionen sind im Head-Up-Display-Treiber gekapselt, welcher als Complex-Device-Driver realisiert ist. Der CDD ist für die Regelung der Displayfunktionen wie Helligkeit, Höhenverstellung der Anzeige sowie Aktivierung und Deaktivierung zuständig. Alle dafür relevanten Daten werden über die AShell an die Empfangseinheit im Head-Up-Display übertragen.

Die Realisierung als CDD war notwendig, da für die Kommunikation im APIX-Seitenband der direkte Zugriff auf spezielle Low-Level-Funktionen erforderlich ist. Dabei handelt es sich um den APIX-Remote-Handler [Fuj10], welcher von einem Zulieferer entwickelt wurde und ein abstrahiertes Interface sowie einen Treiber zur Datenübertragung über die AShell bereitstellt. Der Low-Level-Treiber greift schließlich auf den APIX-Transmitter zu, welcher für die Datenübertragung zum APIX-Receiver im Head-Up-Display zuständig ist. Die beschriebene Software- und Hardwarearchitektur ist schematisch in Abbildung 6.7 dargestellt.

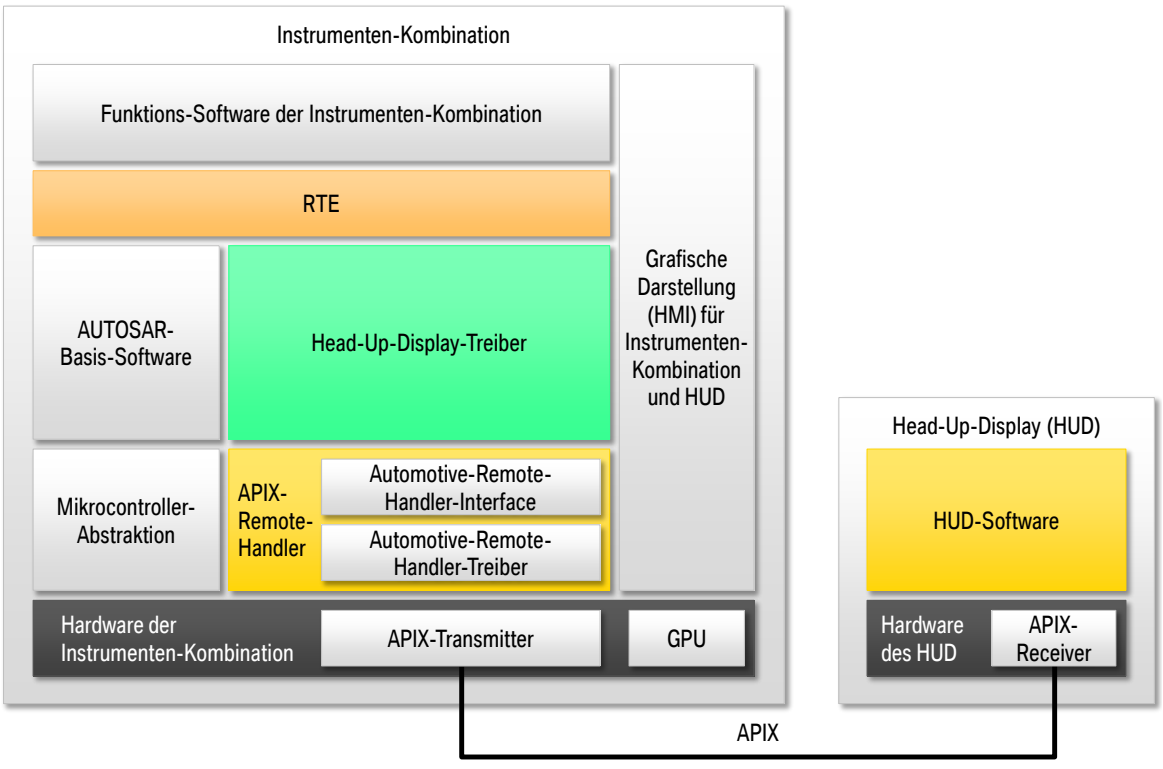


Abbildung 6.7: Software- und Hardware-Architektur von Instrumenten-Kombination und Head-Up-Display.

6.2.2 Konzeptauswahl

Für die Auswahl des Konzepts zur Integration des Head-Up-Display-Treibers auf einer VAP wurde ebenfalls die Methodik aus Kapitel 3.4 verwendet. Die folgenden Prämissen bestimmten das Auswahlverfahren:

- **CDD-Anforderungen:** Der HUD-Treiber verfügt über direkte Schnittstellen zu einer Low-Level-Funktion, welche auf einen APIX-Transmitter zugreift. Im Prinzip handelt es sich dabei um eine Aufteilung des Treibers in High-Level- und Low-Level-Anteil, wobei direkte Hardware-Schnittstellen bestehen. Für die VAP stand eine kompatible APIX-Hardware zur Verfügung, weshalb die Plattform für die Integration des Treibers geeignet war. Daraus ergibt sich der Fall $\alpha = 1$ aus Tabelle 3.6.
- **Testziel:** Das Ziel für die Integration der Software der Instrumenten-Kombination auf der VAP lag in der Sicherstellung der Wiederverwendbarkeit für andere Plattformen. Der HUD-Treiber musste dabei mit einbezogen werden, da er ebenfalls für andere Plattform verwendbar sein sollte. Nach Tabelle 3.6 entspricht dies dem Fall $\vartheta = 2$.

Insgesamt ergibt sich somit nach dem Schema $A_{\alpha\vartheta}$, mit den Werten für α und ϑ, der Anwendungsfall A_{12}. Nach der entwickelten Methodik wurde für die Integration des Treibers das Konzept des CDD-Hardware-Abstraction-Layers ausgewählt (siehe Abbildung 6.8).

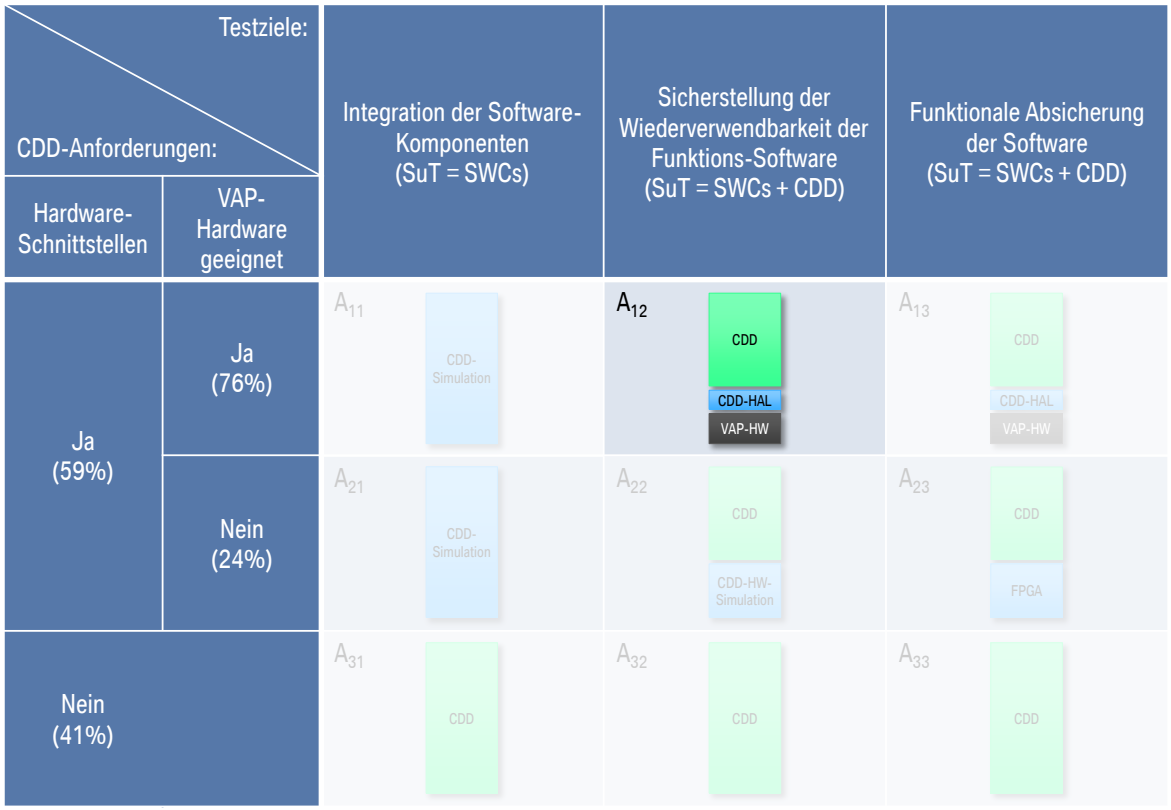


Abbildung 6.8: Konzeptauswahl für den Anwendungsfall des HUD-Treibers.

6.2.3 Konzeptanwendung

Das Konzept sollte die Ansteuerung eines realen Head-Up-Displays über eine virtuelle Absicherungsplattform ermöglichen, um die Tauglichkeit des CDD-HAL zu beweisen. Für die Software-Komponenten der Instrumenten-Kombination stand bereits eine vollständige VAP-Integration zur Verfügung. Der CDD für die Ansteuerung des HUD wurde über die RTE-Schnittstellen an die Funktions-Software angebunden, identisch zum Aufbau im realen Steuergerät.
Als APIX-Schnittstelle für die VAP kam ein Entwicklungsboard mit dem APIX-Transmitter-Chip INAP375T [Ino12] zum Einsatz. Die Anbindung des Boards erfolgte über die Schnittstelle des Universal-Asynchronous-Receiver/Transmitter (UART) [Nat95], welcher per Universal-Serial-Bus (USB) an die VAP gekoppelt wurde. Die UART-Schnittstelle des Entwicklungsboards erlaubte den direkten Zugriff auf die AShell und damit die Datenübertragung im Seitenband des APIX-Link. Für die Übertragung der grafischen Anzeigen des Head-Up-Displays von der VAP zum Board stand ein High-Definition-Multimedia-Interface (HDMI) zur Verfügung.
Die eingesetzte APIX-Schnittstelle unterscheidet sich insofern von der Variante im realen Steuergerät, dass die Bibliotheken des APIX-Remote-Handlers (siehe Abbildung 6.7) nicht nutzbar waren. Der zugehörige Treiber benötigt direkten Zugriff auf den APIX-Transmitter-Chip, ohne den Umweg über eine UART-Verbindung, wie auf der VAP. Dies war allerdings kein Ausschlusskriterium für die beschriebene Architektur, da nur der Head-Up-Display-Treiber selbst im Fokus der Absicherung stand und nicht die Bibliotheken des APIX-Remote-Handlers.
Der CDD selbst greift im Wesentlichen auf drei Funktionen des APIX-Remote-Handlers zu. Zwei dienen dem schreibenden bzw. lesenden Zugriff auf einzelne Speicheradressen des Mikrocontrollers im realen Head-Up-Display über die AShell. Sie werden im Folgenden abstrahiert als *ApixWrite()*, bzw. *ApixRead()* bezeichnet. Die dritte Funktion führt einen Lesezugriff auf mehrere aufeinanderfolgende Speicheradressen durch und soll im Folgenden *ApixReadBlock()* genannt werden. Diese Funktionen mussten über den Hardware-Abstraction-Layer nachgebildet und an das eingesetzte Entwicklungsboard für den APIX-Transmitter angeglichen werden, um die Datenübertragung des CDD zum Display zu ermöglichen.
Für den CDD-Hardware-Abstraction-Layer kam ein früher Prototyp zum Einsatz, der nach dem Konzept aus Kapitel 4.2 aufgebaut war, jedoch noch nicht über die vollständigen Konfigurationsmöglichkeiten verfügte. Die drei benötigten Funktionen wurden als Schnittstelle zum CDD innerhalb der Emulationsschicht so implementiert, dass die Aufrufparameter denen des Head-Up-Display-Treibers entsprechen. Innerhalb der Ebene der Hardware-Abstraktion wurde eine UART-Kommunikation implementiert, die Daten an das APIX-Entwicklerboard übertragen kann. Abbildung 6.9 zeigt die Integration des HUD-Treibers auf der VAP, inklusive des CDD-HAL und der Anbindung des Displays.
Für die Steuerung von Funktionen des Displays durch den HUD-Treiber, wie z. B. Helligkeit und Winkel des Spiegels, müssen mehrere Schreib- und Lesezugriffe durchgeführt werden. Mithilfe des CDD-HAL sind die Zugriffe auf einer VAP wie folgt darstellbar.
Bei einem Schreibzugriff ruft der CDD die Funktion *ApixWrite()* auf, welche im HAL nachgebildet wurde und übergibt z. B. den zu schreibenden Wert (*val*) und die Zieladresse (*adr*). Eine Funktion innerhalb des CDD-HAL erzeugt aus Wert und Zieladresse eine Bitfolge und einen zugehörigen Header, welcher für die Übertragung durch die AShell geeignet ist und legt sie in einem FIFO-Puffer (First-In-First-Out) ab. Die UART-Kommunikation mit dem APIX-Board wird zyklisch in einem eigenen Task ausgeführt. Sie liest die erzeugte Bitfolge aus

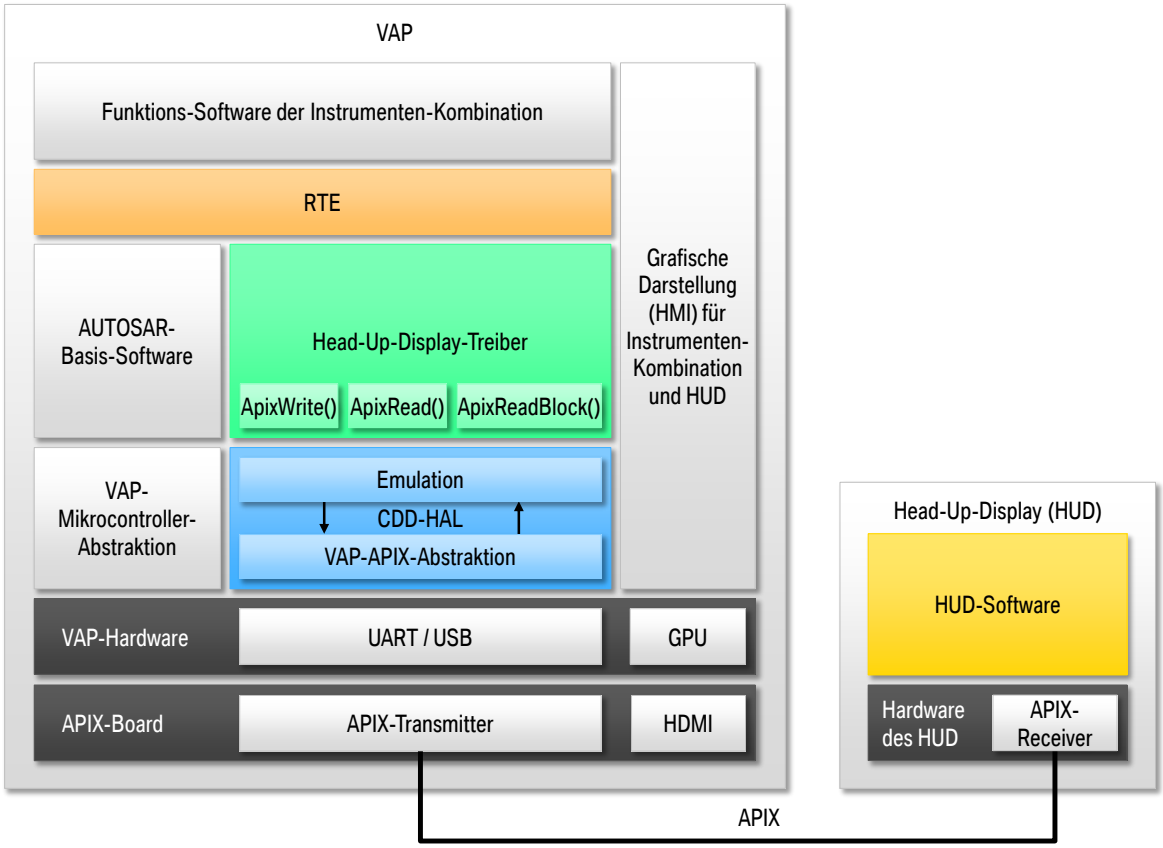


Abbildung 6.9: Anwendung des CDD-HAL-Konzepts auf den Head-Up-Display-Treiber.

dem Puffer aus und übermittelt die Daten an den APIX-Transmitter. Dort erfolgt schließlich die Übertragung des Wertes über die AShell an die Zieladresse innerhalb des HUD. Der Übertragungsvorgang ist vereinfacht in Abbildung 6.10 dargestellt.

Nach demselben Schema läuft der Lesezugriff mit der Funktion *ApixRead()* ab. Allerdings wird dabei zunächst nur eine Anfrage für das Auslesen der Zieladresse gesendet, an der sich das benötigte Datenelement befindet. Der APIX-Receiver im Head-Up-Display sendet den Rückgabewert der Anfrage über den *Upstream*-Kanal an das APIX-Board zurück. Der CDD-HAL ruft zyklisch eine Funktion auf, welche über die UART-Kommunikation das Ergebnis des Auslesevorgangs abholt und auf einem weiteren FIFO-Puffer ablegt. Schließlich wird das Ergebnis aus dem Puffer gelesen und per Callback-Funktion an den Head-Up-Display-Treiber zurückgemeldet. Abbildung 6.11 zeigt den Vorgang vereinfacht.

Der Lesevorgang für mehrere aufeinanderfolgende Speicheradressen mit der Funktion *ApixReadBlock()* funktioniert nach dem gleichen Prinzip. Der Unterschied besteht darin, dass die Anfrage die Länge des auszulesenden Speicherbereichs mit enthält und die einzelnen Datenelemente in aufeinanderfolgenden Zyklen abgerufen werden.

Durch die drei implementierten Funktionen im CDD-HAL ist der Head-Up-Display-Treiber auf einer VAP in der Lage, sämtliche Operationen zur Steuerung des Displays durchzuführen.

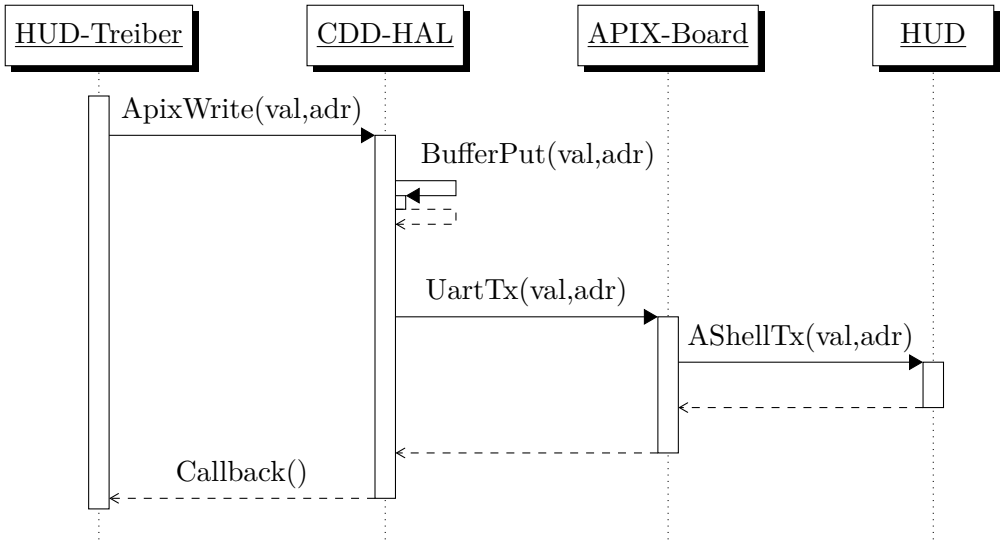


Abbildung 6.10: Schreibzugriff des HUD-Treibers über den CDD-HAL auf das Head-Up-Display.

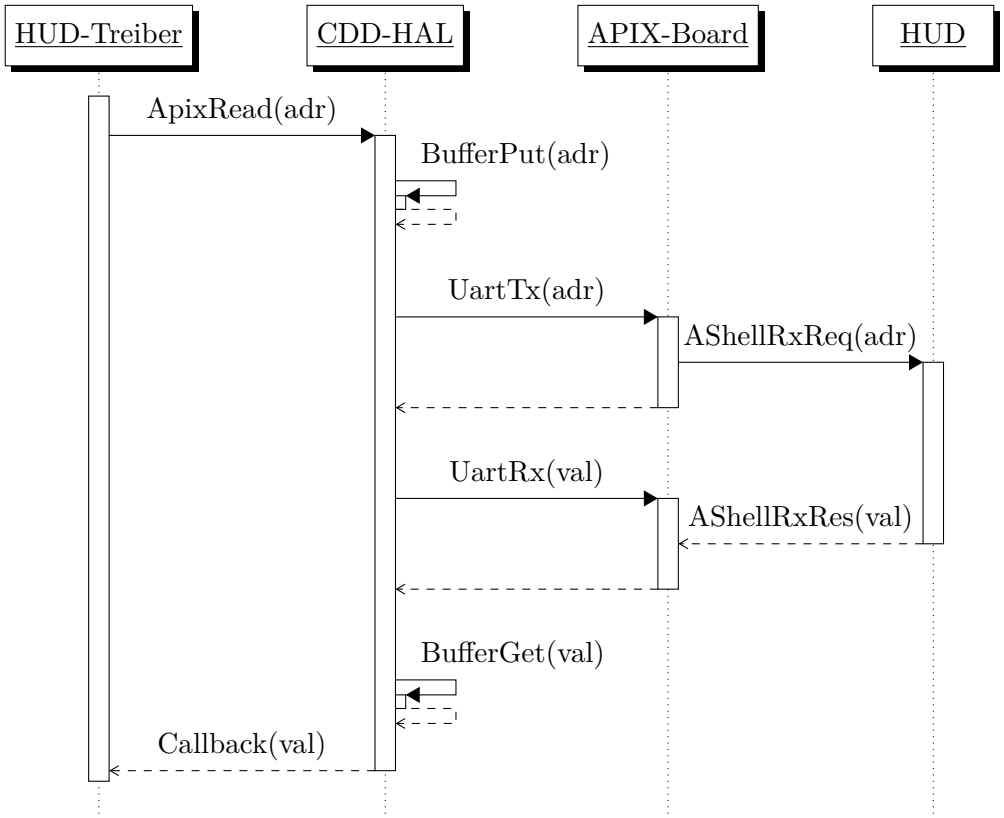


Abbildung 6.11: Lesezugriff des HUD-Treibers über den CDD-HAL auf das Head-Up-Display.

6.2.4 Versuchsaufbau

Um die Integration des Head-Up-Display-Treibers in einer VAP testen zu können, mussten die Funktionen des CDD stimuliert werden. Im realen Fahrzeug ist das HUD über das iDrive-System im zentralen Infotainment-Display steuerbar. Das zentrale Infotainment-Steuergerät übermittelt die Einstellungen über einen CAN-Bus an die Instrumenten-Kombination. Dort werden die Daten an den HUD-Treiber weitergeleitet, der wiederum das Display ansteuert. Für den CDD ist, aus Kundensicht, die Einstellungen für die Helligkeit der Anzeige relevant. Weiterhin ist die Höhenverstellung der Projektion in der Windschutzscheibe möglich, die durch den Winkel eines verstellbaren Spiegels (siehe Abbildung 6.6) geregelt wird. Darüber hinaus aktiviert bzw. deaktiviert der HUD-Treiber das Display je nach Einstellung und führt verschiedene Diagnose-Funktionen aus.

Für den Versuchsaufbau wurden sämtliche CAN-Signale, welche für den CDD relevant sind, durch eine Restbussimulation an die Software der Instrumenten-Kombination übermittelt. Die Signale enthielten alle notwendigen Parameter, z. B. für die Displayhelligkeit oder die Höhenverstellung. Über eine graphische Oberfläche der Restbussimulation ließen sich alle relevanten Einflussgrößen während der Laufzeit der Software innerhalb ihres möglichen Wertebereichs variieren. Die CAN-Signale wurden über die CAN-Karte der VAP empfangen und durch die Module des AUTOSAR-Stacks und der Funktions-Software der Instrumenten-Kombination an den HUD-Treiber weitergeleitet. Der vollständige Versuchsaufbau ist in Abbildung 6.12 dargestellt.

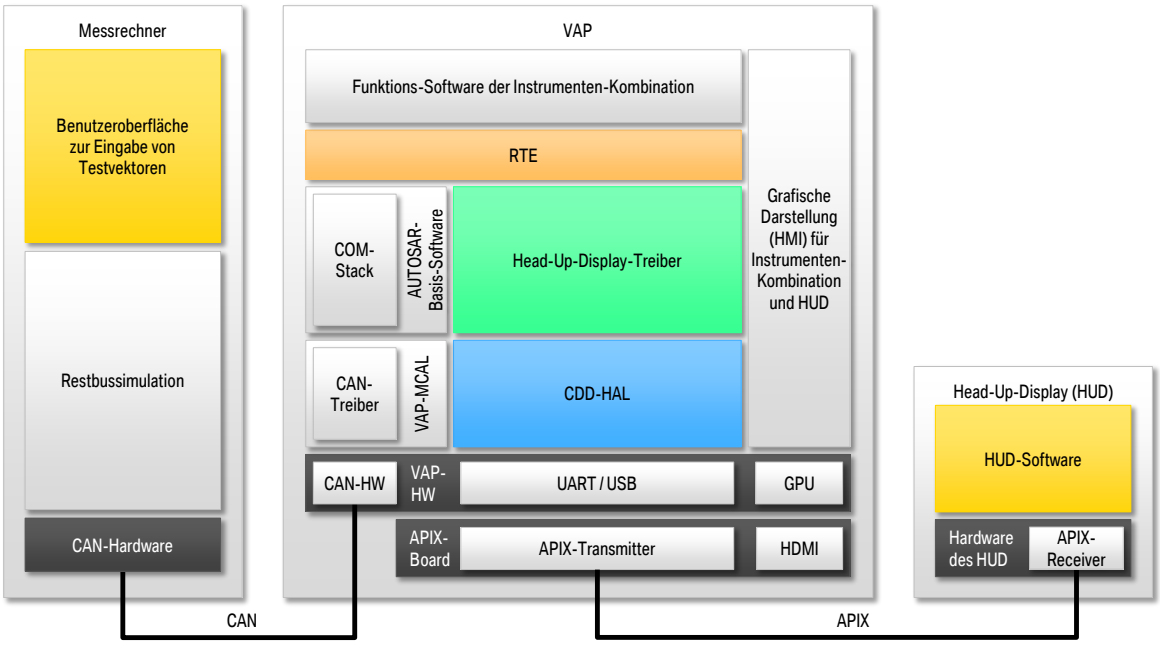


Abbildung 6.12: Schematischer Versuchsaufbau für die Integration des HUD-Treibers mit Restbussimulation.

6.2.5 Evaluierung

Mit Hilfe des CDD-Hardware-Abstraction-Layer-Konzepts ließ sich der Head-Up-Display-Treiber, zusammen mit der Funktions-Software der Instrumenten-Kombination, vollständig auf einer VAP integrieren. Im Vordergrund stand dabei die Wiederverwendbarkeit des Treibers, der in unterschiedlichen Steuergeräte-Varianten mehrerer Zulieferer zum Einsatz kommt. Der CDD konnte ohne Anpassungen vollständig für die x86-Architektur der VAP kompiliert werden und ließ sich durch die Nachbildung der Hardware-Schnittstellen über den CDD-HAL auf der Plattform ausführen. Somit war der Nachweis erbracht, dass der Treiber über keine plattformspezifischen Abhängigkeiten verfügt. Daraus lässt sich schlussfolgern, dass eine Portierung auf unterschiedliche Mikrocontroller mit hoher Wahrscheinlichkeit unproblematisch ist.

Zusätzlich erlaubte die Anbindung des CDD-HAL eine funktionale Verifikation des HUD-Treibers auf einer VAP. Von insgesamt 23 Testfällen für die Absicherung bei der BMW Group, die alle funktionalen Aspekte des Head-Up-Displays abdecken, beziehen sich sieben auf den CDD. Geprüft werden z. B. die Höhenverstellung, die manuelle und automatische Anpassung der Displayhelligkeit, sowie die Aktivierung und Deaktivierung des Head-Up-Displays bei Betätigung des zugehörigen Tasters in Abhängigkeit der Klemmenzustände des Fahrzeugs. Ein Beispiel für die manuelle Helligkeitseinstellung ist in Tabelle 6.3 dargestellt. Alle sieben Testfälle für den HUD-Treiber ließen sich durch die Integration des CDD-HAL erfolgreich und fehlerfrei durchführen, wie in Tabelle 6.4 dokumentiert. Somit konnte die Funktion des CDD auf einer VAP mit Anbindung eines realen Head-Up-Displays vollständig verifiziert werden.

Testschritt	**Beschreibung**
Voraussetzungen	- Zündung eingeschaltet. - HUD aktiviert.
Aktion 1	- Helligkeit des HUD über das iDrive-System auf den Maximalwert einstellen.
Ergebnis 1	- Helligkeit des HUD ändert sich sofort auf den Maximalwert.
Aktion 2	- Helligkeit des HUD über das iDrive-System auf den Minimalwert einstellen.
Ergebnis 2	- Helligkeit des HUD ändert sich sofort auf den Minimalwert.

Tabelle 6.3: Beispiel eines Testfalls für den HUD-Treiber.

Als limitierender Faktor bei der Integration des CDD auf einer VAP hat sich die Übertragungsgeschwindigkeit der UART-Verbindung zum APIX-Board herausgestellt. Der CDD-HAL enthält einen Task, der zyklisch Daten aus einem Puffer über UART an das APIX-Board überträgt und die Rückmeldungen ausliest (siehe Kapitel 6.2.3). Versuche haben ergeben, dass die kürzeste Zykluszeit des Tasks, bei der eine erfolgreiche Datenübertragung gewährleistet ist, 3 ms beträgt. Beim Aufstartvorgang liest der HUD-Treiber allerdings den vollständigen Speicherbereich des Head-Up-Displays in einem Lesevorgang aus. Der Watchdog des HUD erwartet, dass die Operation in wenigen Sekunden abgeschlossen ist, während sich der Vorgang durch die Limitierung der Übertragungsgeschwindigkeit auf mehrere Minuten ausdehnt. Daher musste der Watchdog innerhalb des realen HUD für die Versuchsdurchführung deaktiviert werden, wodurch jedoch keine direkten funktionalen Einschränkungen entstanden sind.

Testfallbeschreibung	**Ergebnis**
Manuelle Aktivierung HUD bei aktiver Zündung.	Bestanden ✓
Manuelle Deaktivierung HUD bei aktiver Zündung.	Bestanden ✓
Aktivierung HUD bei abgeschalteter Zündung nicht möglich.	Bestanden ✓
Deaktivierung HUD bei Abschaltung Zündung.	Bestanden ✓
Helligkeit HUD über iDrive-System auf maximalen und minimalen Wert einstellbar.	Bestanden ✓
Höhe HUD über iDrive-System auf maximalen und minimalen Wert einstellbar.	Bestanden ✓
Automatische Helligkeitsregelung HUD bei Erkennung von Tag und Nacht über Lichtsensor.	Bestanden ✓

Tabelle 6.4: Durchgeführte Testfälle für den HUD-Treiber mit zugehörigen Ergebnissen.

Die begrenzte Übertragungsgeschwindigkeit hat sich ebenso bei der Verstellung des Spiegels im HUD bemerkbar gemacht. Der HUD-Treiber sendet dem Schrittmotor pro Übertragungszyklus inkrementell Vorgaben zur Positionsverstellung und überprüft die aktuelle Position nach jedem Schritt. Dadurch entsteht eine Vielzahl an notwendigen Datenübertragungen, bis die gewünschte Zielposition erreicht ist. Die begrenzte Übertragungsgeschwindigkeit der UART-Verbindung führt somit dazu, dass eine Verstellung des Spiegels von der maximalen zur minimalen Position mehrere Minuten in Anspruch nimmt. Abgesehen von der zeitlichen Ausdehnung entstand jedoch keine weitere Einschränkung.
Die Funktionen des HUD-Treibers konnten durch den CDD-HAL auf einer VAP mit Anbindung eines realen Head-Up-Displays trotz der Limitierungen ebenso erfolgreich verifiziert werden wie die Wiederverwendbarkeit der Software.

6.3 Elektromotortreiber

Eine spezifische Form eines Elektromotors stellt die Synchronmaschine dar. Sie wird häufig in Elektrofahrzeugen als Antrieb im Motorbetrieb und zur Rekuperation im Generatorbetrieb verwendet. Erzeugen Permanentmagnete das nötige Magnetfeld, handelt es sich um eine permanenterregte Synchronmaschine (Permanent-Magnet-Synchronous-Motor (PMSM)). Die Magnete befinden sich im inneren des Motors und sind an einem drehbaren Rotor befestigt, wie rechts in Abbildung 6.13 schematisch dargestellt. Der äußere Teil des Motors wird als Stator bezeichnet und enthält mehrere Statorspulen. Die Spulen sind als sogenannte Polpaare angeordnet. Bei einem Drehstrommotor, wie der abgebildeten PMSM, existiert für jede der drei Phasen mindestens ein Polpaar. Um den Motor anzutreiben, werden die Spulen bei wechselnder Polung mit Strom versorgt. Dadurch wird in jedem Polpaar ein Magnetfeld induziert, welches in Wechselwirkung mit dem Magnetfeld des Rotors tritt. Bei korrekter Reihenfolge und Polung der Stromversorgung entsteht eine gleichförmige Drehbewegung des Rotors.

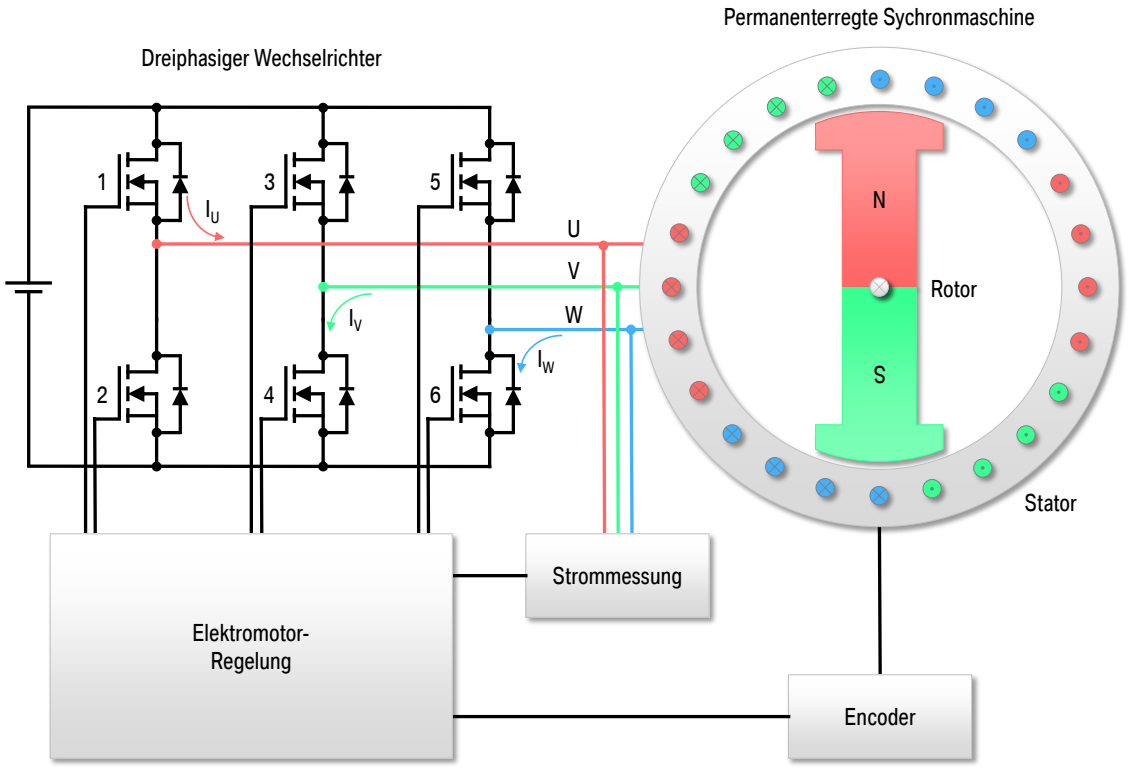


Abbildung 6.13: Schematischer Aufbau und Ansteuerung einer permanenterregten Synchronmaschine.

Die Stromversorgung der Statorspulen erfolgt über einen dreiphasigen Wechselrichter aus Leistungstransistoren. Jedes Polpaar ist mit einer der drei Halbbrücken des Wechselrichters verbunden. Die Transistoren werden so geschaltet, dass in jeder Halbbrücke immer exakt einer der zwei Transistoren den Stromkreis schließt. Die Enden der Spulen sind miteinander

verbunden, sodass sich die Ströme nach Formel 6.1 zu einander verhalten. In Abbildung 6.13 ist ein Beispiel für den Stromfluss eingezeichnet, wobei die Transistoren 1, 4 und 6 den Stromkreis schließen.

$$I_U + I_V + I_W = 0 \tag{6.1}$$

Die Ansteuerung der Transistoren erfolgt über einen Mikrocontroller, auf dem der Regelalgorithmus in Form eines CDD ausgeführt wird. Als Eingangswerte der Regelung dienen die Ströme der Statorspulen und eine inkrementelle Positionsbestimmung des Rotors durch einen Encoder.

6.3.1 Projektbeschreibung

Für die Regelung einer PMSM stand der MC-ISAR eMotor-Driver [Inf12] zur Verfügung, der für den Infineon TriCore TC1798 entwickelt wurde. Der Treiber ist als CDD implementiert, da die Regelung eines Elektromotors sehr zeitkritisch ist. Die eingesetzten Zykluszeiten einiger Funktionen betragen bis zu 50 μs, um hohe Drehzahlen zu ermöglichen.
Der Complex-Device-Driver setzt sich aus verschiedenen Modulen und Bibliotheken zusammen, die für die Regelung zum Einsatz kommen. Eine eigene Konfiguration nach dem Vorbild der AUTOSAR-Basis-Software-Module erlaubt die Einstellung verschiedener Parameter, wie z. B. die Anzahl der Polpaare des eingesetzten Motors oder die Frequenz der auszugebenden PWM-Signale. Zusätzlich ist die Auswahl des Regelungsalgorithmus zwischen feldorientierter Regelung (Field-Oriented-Control (FOC)) oder Blockkommutierung möglich. Die Konfiguration des Treibers kann auf maximal vier Motoren erweitert werden. Abbildung 6.14 stellt alle eingesetzten Module und Bibliotheken des Treibers im Fall der feldorientierten Regelung dar, welche für die Ansteuerung einer PMSM geeignet ist.

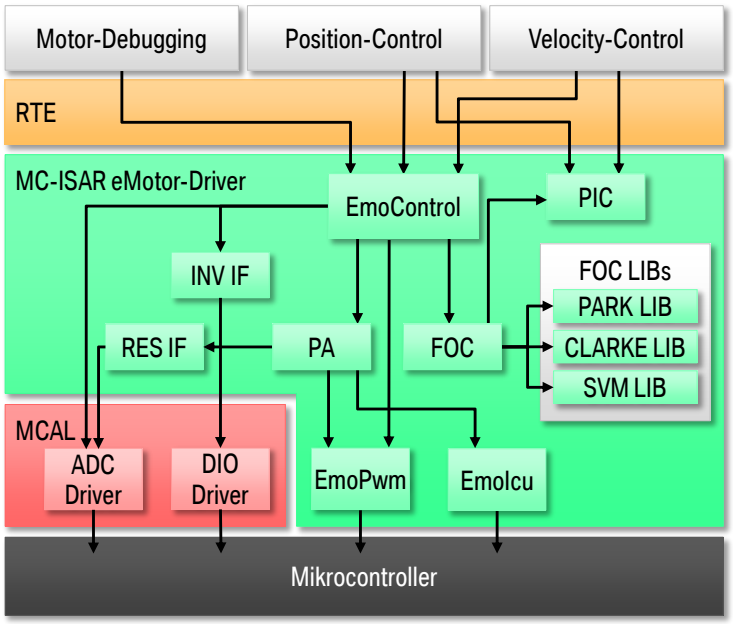


Abbildung 6.14: Schematischer Aufbau des MC-ISAR eMotor-Drivers [Inf12].

Der Treiber verfügt über Schnittstellen zu drei Software-Komponenten für die Positions- und Geschwindigkeits-Regelung sowie für einige Debugging-Parameter. Das Modul *EmoControl* verarbeitet die Geschwindigkeits-, bzw. Positionsvorgaben und ruft die restlichen Module des CDD auf. Das *FOC*-Modul ist für die feldorientierte Regelung zuständig und greift auf Bibliotheken für die nötigen Berechnungen zurück. Das *PA*-Modul (*Position-Acquisition*) übernimmt die Positionsüberwachung.
Für die Regelung des dreiphasigen Wechselrichters erzeugt der CDD sechs PWM-Signale über das Modul *EmoPwm*, welche die Transistoren im Wechsel ansteuern. Die Frequenz der PWM-Signale bleibt dabei konstant, während das Tastverhältnis je nach aktueller Rotorlage geregelt wird.
Als Eingangswerte für die Regelung werden durch ADCs die Spannungen innerhalb der Statorspulen gemessen und daraus die Werte der Stromstärken I_U, I_V und I_W berechnet. Für den ADC-Zugriff nutzt der CDD den ADC-Treiber des AUTOSAR-Microcontroller-Abstraction-Layers (MCAL).
Ein Encoder, auch als Inkrementalgeber bezeichnet, bestimmt die aktuelle Position des Rotors. Zwei Anschlüsse geben insgesamt 512 Impulse pro Umdrehung aus, wobei die zugehörigen Signale *A* und *B* um eine halbe Impulslänge versetzt sind. Der Versatz ermöglicht die Detektion der Laufrichtung. Ein zusätzliches Signal *I* wird einmal pro Umdrehung ausgegeben und bestimmt den Nullpunkt. Der Verlauf der drei Signale ist in Abbildung 6.15 verzeichnet. Für die Detektion der Impulse verfügt der CDD über eine eigene *Interrupt-Control-Unit*, die im Modul *EmoIcu* implementiert ist. Zusätzlich wird auf den DIO-MCAL zugegriffen.

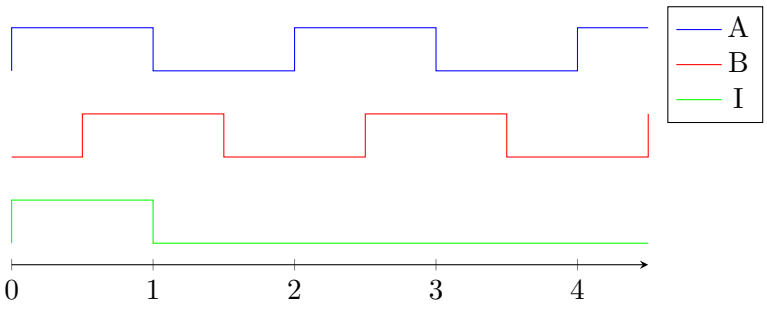


Abbildung 6.15: Verlauf der Encodersignale.

6.3.2 Konzeptauswahl

Die Integration des Elektromotortreibers auf einer VAP erfolgte nach der entwickelten Methodik aus Kapitel 3.4. Das Auswahlverfahren wurde mit den folgenden Prämissen durchgeführt:

- **CDD-Anforderungen:** Der Elektromotortreiber verfügt über direkte Schnittstellen zur Hardware, die in den Modulen *EmoPwm* und *EmoIcu* verwendet werden. Zusätzlich erfolgt ein Zugriff auf den ADC- und DIO-Treiber des MCAL. Alle benötigen Hardware-Schnittstellen sind über die Standardkonfiguration einer VAP darstellbar. Somit ergibt sich der Fall $\alpha = 1$ aus Tabelle 3.6.

- **Testziel:** Das Ziel für die Integration des Elektromotortreibers auf der VAP war die funktionale Absicherung. Es sollte sichergestellt werden, dass der Treiber erfolgreich den Betrieb einer Synchronmaschine regeln kann, auch wenn er nicht auf seiner Zielplattform ausgeführt wird. Übertragen auf das Schema aus Tabelle 3.6, entspricht das dem Fall $\vartheta = 3$.

Für den Anwendungsfall $A_{\alpha\vartheta}$ ergibt sich daher A_{13}. Nach der entwickelten Methodik wurde somit für die Integration des Treibers auf der VAP das Konzept des CDD-Hardware-Abstraction-Layers ausgewählt (siehe Abbildung 6.16).

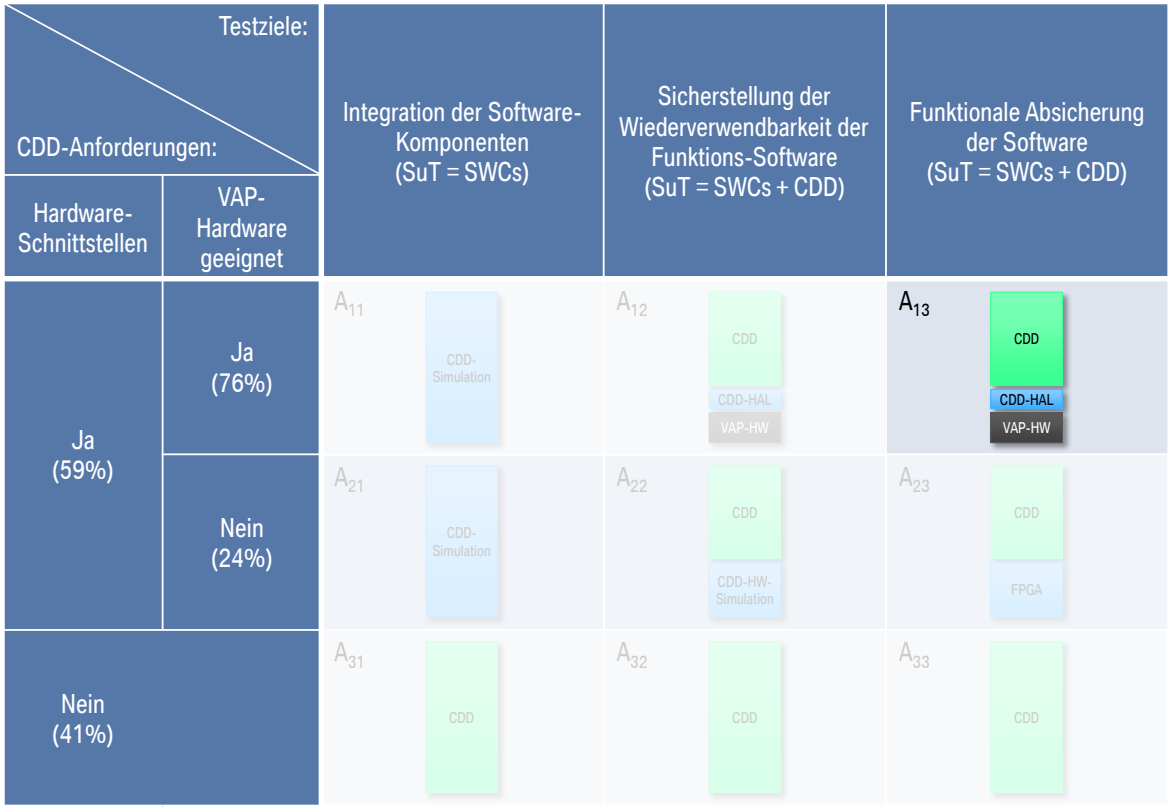


Abbildung 6.16: Konzeptauswahl für den Anwendungsfall des Elektromotortreibers.

6.3.3 Konzeptanwendung

Die Anwendung des ausgewählten Konzepts für die Integration des Elektromotortreibers erfolgte im Rahmen der zugehörigen Masterarbeit [Vog13]. Der CDD-HAL sollte die Ansteuerung einer Synchronmaschine ermöglichen, die über die realen Hardware-Schnittstellen angesteuert und innerhalb eines HiL-Systems simuliert wird. So sollte die Tauglichkeit des Konzepts auch für komplexe, zeitkritische Funktionen bewiesen werden. Dabei kam die vollständig konfigurierbare Version des HAL zum Einsatz, wie sie in Kapitel 5.2 beschrieben ist. Die Integration aller Komponenten ist in Abbildung 6.17 dargestellt.

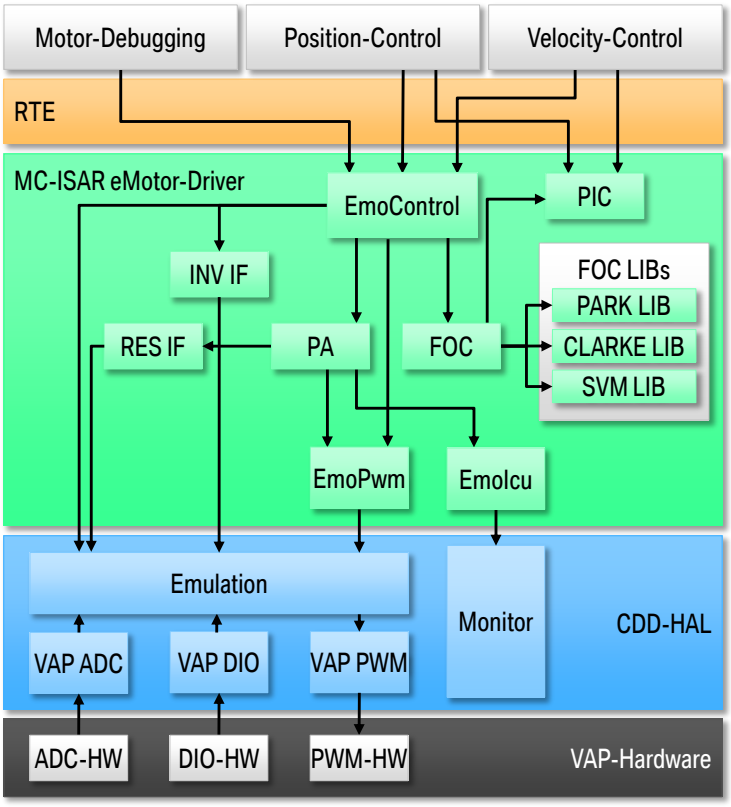


Abbildung 6.17: Integration des MC-ISAR eMotor-Drivers auf einer VAP mit dem CDD-HAL-Konzept.

Die Anbindung des CDD-HAL erfolgte direkt über die Hardware-Schnittstellen des Treibers. Dabei wurden alle ADC, DIO und PWM-Signale des CDD berücksichtigt und über die verfügbaren Hardware-Module auf der VAP mit den Ein- und Ausgängen der simulierten Synchronmaschine auf dem HiL-System verbunden.

Die Schnittstellen des Treibers zu den MCAL-Modulen der Ziel-Plattform für ADC und DIO aus Abbildung 6.14 ließen sich nicht mit den MCAL-Modulen der VAP darstellen. Daher wurde die Hardware-Anbindung ebenfalls über den CDD-HAL realisiert. Alle Schichten des Konzepts wurden im Rahmen der Integration über ihre Konfiguration auf die Anforderungen des Elektromotortreibers eingestellt.

Emulationsschicht

In der Emulationsschicht ließen sich alle nötigen Register und Variablen der Ziel-Plattform abbilden. Die Werte der drei Stromstärken in den Statorspulen, deren Messung über ADC-Kanäle erfolgt, wurden direkt von der Emulationsschicht in die korrekten Variablen des CDD übertragen. Abbildung 6.18 zeigt die Konfiguration der Emulationsschicht für die Integration des Elektromotortreibers.

CddHalEmulation		CddHalEmulation
CCU60_MCFG		Register
Name	*CCU60_MCFG*	Name
Address	*0xf0003004u*	Address
Size	32	Size
EmulationType	MemorySpaceEmulation	EmulationType
T12_Period_Reg_Period		Register
T13_Period_Reg_Trigger_Reload		Register
T12_Counter_Reg_Timer_Offset		Register
Period		Register
Name	*T12PR*	Name
Size	32	Size
EmulationType	RegisterAlias	EmulationType
U_Current		Register
Name	*&(Emo_ResultBuffer[0].AdcRawValues[1])*	Name
Size	16	Size
EmulationType	Variable	EmulationType
V_Current		Register
W_Current		Register
Duty0_Register		Register
Name	*CCU60_CC60SR*	Name
Address	*0xf0003040u*	Address
Size	16	Size
EmulationType	MemorySpaceEmulation	EmulationType
Duty1_Register		Register
Duty2_Register		Register

Abbildung 6.18: Konfiguration der Emulationsschicht für die Integration des Elektromotortreibers.

Die Abtastung der Signale des Encoders aus Abbildung 6.15 und die Berechnung der resultierenden Position des Rotors ist auf der Ziel-Plattform des Treibers direkt in der Hardware implementiert. An den CDD wird lediglich der aktuelle Winkel des Rotors in Werten von 0° bis 360°, bei einer Auflösung von 1024, über ein Register weitergegeben. Um diese Funktion auf der VAP darzustellen, wurde die Möglichkeit zur Einbindung von *User-Functions* (siehe Kapitel 5.2.1) in den CDD-HAL genutzt. Ein entsprechendes Register in der Emulationsschicht erhält den berechneten Wert der *User-Function* und stellt ihn dem Treiber zur Verfügung.
Die PWM-Signale wurden in der Konfiguration über drei Register für das Tastverhältnis und ein Register für die gemeinsam genutzte PWM-Periode dargestellt. Die Übertragung der Werte erfolgte direkt über das *EmoPwm*-Modul des Elektromotortreibers an die Emulationsschicht. Aus den drei Signalen für die Tastverhältnisse der oberen Hälfte der drei Halbbrücken (siehe

Abbildung 6.13) erfolgt auf der Ziel-Plattform automatisch die Ausgabe des jeweils invertierten Signals für die untere Seite der Halbbrücke.

Routing

Die Konfiguration des Routings bildet die Übertragung der Datenelemente zwischen den Registern der Emulationsschicht und der Hardware-Abstraktion ab. Für jede der drei Stromstärken der ADCs wurde ein Routing-Eintrag definiert. Da der ADC einer VAP und der Ziel-Plattform mit unterschiedlichen Wertebereichen arbeiten, enthalten die Routings eine Referenz auf eine Umrechnungsfunktion, welche die Daten konvertiert. Abbildung 6.19 zeigt die vollständige Konfiguration des Routings.

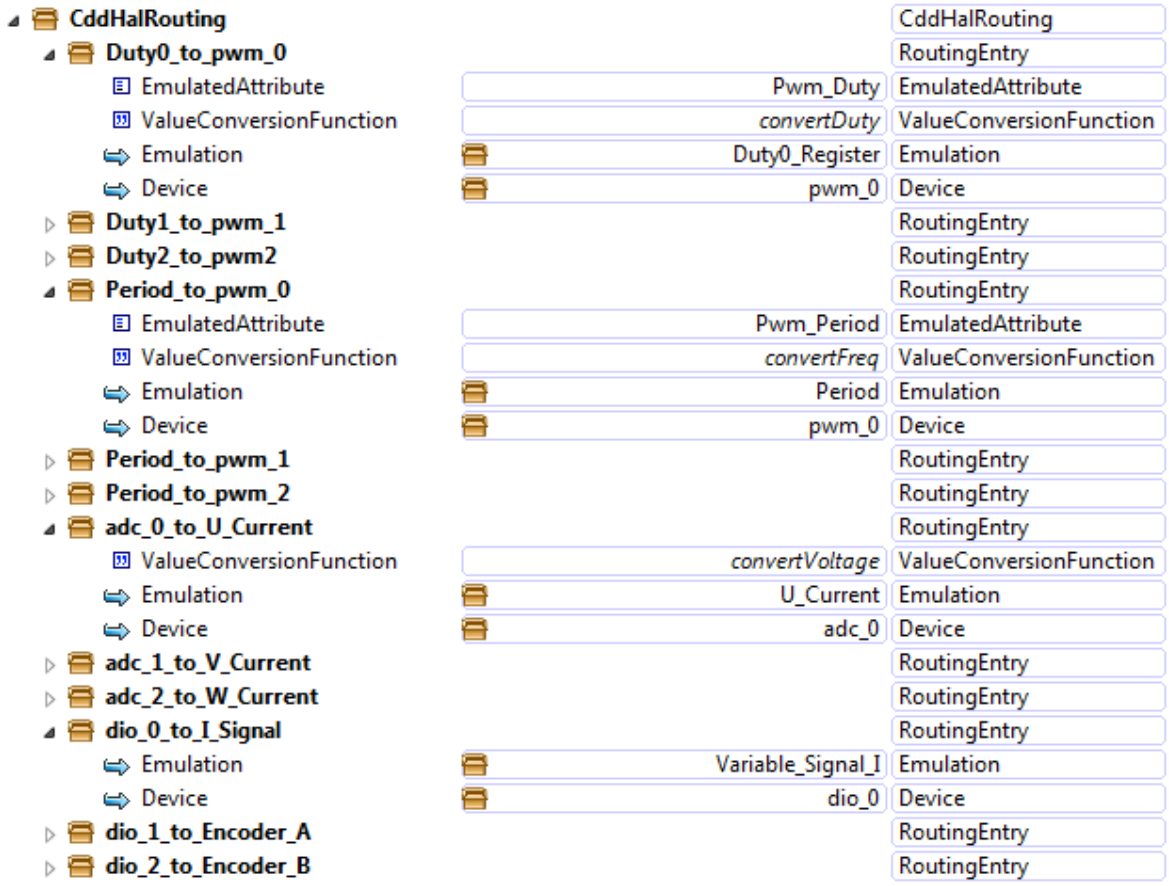


Abbildung 6.19: Konfiguration des Routings für die Integration des Elektromotortreibers.

Für die PWM-Signale waren sechs Routing-Einträge erforderlich. Jeweils drei Pfade definieren die Übermittlung der berechneten Tastverhältnisse zu den Kanälen. Drei weitere Einträge sorgen für den Transfer der Frequenz, wobei hierfür dreimal dasselbe emulierte Register aus der Emulation referenziert wird. Dies ist nötig, da auf der Ziel-Hardware alle PWM-Ausgänge die Frequenz aus einem Register auslesen, während das PWM-Modul der VAP verschiedene Frequenzen für die einzelnen Ausgänge zulässt. Sowohl für das Tastverhältnis als auch die Frequenz sind Umrechnungsfunktionen nötig, welche die Wertebereiche der Ziel-Architektur und der VAP angleichen.

Einen Sonderfall bildet das Routing für die Encoder-Signale. Drei digitale Eingänge tasten die aktuellen Werte der Signale *A*, *B* und *I* (siehe Abbildung 6.15) zyklisch ab. Die Routing-Einträge übermitteln die Werte an drei *MemoryBuffer*, die von einer *User-Function* ausgelesen werden. Diese wird durch den Monitor getriggert und führt die Berechnung der Rotorposition durch.

Hardware-Abstraktion

Die Konfiguration der Hardware-Abstraktion definiert die verwendeten Module der VAP-Hardware. Abbildung 6.20 zeigt die Parameter für die Integration des Elektromotortreibers.

CddHalDevice		CddHalDevice
pwm_0		Pwm
DeviceId	*pwm_0*	DeviceId
Period	100	Period
Duty	20	Duty
ChannelId	0	ChannelId
pwm_1		Pwm
pwm_2		Pwm
adc_0		Adc
adc_1		Adc
ChannelId	1	ChannelId
DeviceId	0	DeviceId
adc_2		Adc
dio_0		Dio
dio_1		Dio
dio_2		Dio
ChannelId	2	ChannelId
DeviceId	0	DeviceId
Direction	Input	Direction
Variable_Signal_I		MemoryBuffer
Variable_Encoder_A		MemoryBuffer
Name	*di_1*	Name
Size	4	Size
Variable_Encoder_B		MemoryBuffer

Abbildung 6.20: Konfiguration der Hardware-Abstraktion für die Integration des Elektromotortreibers.

Zur Hardware-Anbindung wurden jeweils drei ADC-, DIO- und PWM-Kanäle definiert. Die ADCs und DIOs befinden sich auf derselben C-PCI-Karte und nutzen in der Konfiguration somit dieselbe *DeviceId*. Sie verfügen jedoch über eigene, mit Null beginnende, Kanalnummerierungen. Die analogen Eingänge dienen zur Bestimmung der Stromstärke in den Statorspulen des Elektromotors. Die DIOs wurden als Eingänge definiert, welche die Signale *A*, *B* und *I* des Encoders abtasten. Die drei PWM-Ausgänge werden über eine eigene C-PCI-Karte ausgegeben. Sie verfügen jeweils über Initialparameter für das Tastverhältnis und die Periodendauer.
Zusätzlich zu den Hardware-Anschlüssen wurden in der Hardware-Abstraktion drei *Memory-Buffer* angelegt. Diese dienen als Zwischenspeicher für die *User-Function*, welche in den

abgetasteten Werten der DIOs die Taktflanken zählt und daraus die aktuelle Rotorposition bestimmt.

Monitor

Die Konfiguration des Monitors dient bei der Integration des Elektromotortreibers dazu, Funktionen des CDD auszulösen, welche in der Ziel-Hardware über Timer und Interrupts getriggert werden. Dazu kam die implementierte Monitor-Variante des Timed-Monitors (siehe Kapitel 4.2.2) zum Einsatz. Insgesamt wurden 3 Instanzen eines Timed-Monitors definiert, die unterschiedliche Aufgaben erfüllen. Die vollständige Konfiguration ist in Abbildung 6.21 dargestellt.

CddHalMonitor		CddHalMonitor
EngineCurrentTriggerFocMonitor		CddHalTimedMonitor
MonitorTiming	500000	MonitorTiming
MonitorCore	0	MonitorCore
EndHook	*Emo_FocTriggerMotorControl(0)*	EndHook
MonitorPriority	99	MonitorPriority
RoutingEntryRef	adc_0_to_U_Current	RoutingEntryRef
RoutingEntryRef	adc_1_to_V_Current	RoutingEntryRef
RoutingEntryRef	adc_2_to_W_Current	RoutingEntryRef
Speed_HandlerTask		CddHalTimedMonitor
MonitorTiming	1000000	MonitorTiming
MonitorCore	0	MonitorCore
StartHook	*SpeedPi_Handler()*	StartHook
MonitorPriority	99	MonitorPriority
EncoderSignalPollingTask		CddHalTimedMonitor
MonitorTiming	350000	MonitorTiming
MonitorCore	1	MonitorCore
EndHook	*EngineSignalPolling_task()*	EndHook
MonitorPriority	98	MonitorPriority
RoutingEntryRef	dio_0_to_I_Signal	RoutingEntryRef
RoutingEntryRef	dio_1_to_Encoder_A	RoutingEntryRef
RoutingEntryRef	dio_2_to_Encoder_B	RoutingEntryRef

Abbildung 6.21: Konfiguration des Monitors für die Integration des Elektromotortreibers.

Der *EncoderSignalPollingTask* bildet die Positionsbestimmung des Rotors, die auf der Ziel-Plattform in Hardware realisiert ist, anhand der Encodersignale nach. Die Monitor-Instanz ist als eigener Task implementiert, der zyklisch aufgerufen wird. Bei jeder Ausführung triggert der Task zunächst drei referenzierte Routing-Einträge, welche die aktuellen Werte der DIOs in die konfigurierten *MemoryBuffer* in der Hardware-Abstraktion übertragen. Danach erfolgt der Aufruf der *User-Function EngineSignalPolling_task()* über den *EndHook* des Monitors. Die Funktion zählt zyklisch die Taktflanken der Encodersignale und berechnet daraus einen Wert zwischen 0 und 1024, der die aktuelle Position des Rotors zwischen 0° und 360° widerspiegelt. Zuletzt folgt die Übermittlung des Positionswertes an den Complex-Device-Driver.
Die zweite Monitor-Instanz, der *EngineCurrentTriggerFocMonitor*, triggert alle 500 μs die Routing-Einträge zur Übermittlung der aktuellen ADC-Werte, welche die Stromstärken

der Statorspulen repräsentieren. Die Messwerte werden in den emulierten Registern in der Emulationsschicht abgelegt.
Der *EndHook* des Monitors ruft direkt die Funktion *Emo_FocTriggerMotorControl(0)* innerhalb des CDD auf, welche die Werte der Stromstärken aus den Registern ausliest und die feldorientierte Regelung für die Ansteuerung des Elektromotors durchführt. Der Parameter *0* bezeichnet bei dem Aufruf den ersten von maximal vier Elektromotoren, die von dem CDD geregelt werden können.
Über den dritten Monitor erfolgt die Emulation eines Interrupts, der auf der Ziel-Hardware jede Millisekunde die Funktion *SpeedPi_Handler()* aufruft. Darin erfolgt die Berechnung der aktuellen Geschwindigkeit des Rotors.

6.3.4 Versuchsaufbau

Für die funktionale Verifikation der Integration des Elektromotortreibers auf einer VAP war die Anbindung einer permanenterregten Synchronmaschine erforderlich. Um eine direkte Überwachung aller Parameter zu ermöglichen und kontrollierbare Umgebungsbedingungen zu schaffen, wurde kein realer Motor, sondern eine Simulationsumgebung inklusive einer simulierten Synchronmaschine eingesetzt. Dafür entstand im Rahmen dieser Arbeit ein vollständiges MATLAB/Simulink-Modell mit einer PMSM und einem dreiphasigen Wechselrichter. Der Aufbau des Modells ist in Abbildung 6.22 verzeichnet.
Für die Ausführung des Modells kam ein HiL-System mit der Software IPG CarMaker [SHWKR08] zum Einsatz, welche die Integration von MATLAB/Simulink-Modellen erlaubt. Um einen realistischen Anwendungsfall abzubilden, wurde die modellierte Synchronmaschine in ein vollständiges Fahrzeugmodell als Antrieb eingefügt. Das HiL-System stellte die entsprechenden Schnittstellen bereit, um die Ansteuerung der modellierten Synchronmaschine zu ermöglichen. So standen C-PCI-Karten zur PWM-Abtastung für die Ansteuerung des dreiphasigen Wechselrichters, analoge Ausgänge für die Ausgabe der Statorströme und eine Karte zur Erzeugung von Encoder-Signalen zur Verfügung. Über den CarMaker ließen sich die jeweiligen Signale des Motormodells mit den zugehörigen Kanälen der C-PCI-Karten verbinden, um eine Ansteuerung über reale Leitungen zu ermöglichen.
Für die Versuchsdurchführung wurden die Ein- und Ausgangskanäle der VAP-Hardware mit den zugehörigen Anschlüssen des HiL-Systems verbunden, wie in Abbildung 6.23 gezeigt. So ließen sich sämtliche notwendigen Signale für die Regelung zwischen dem Elektromotortreiber und der simulierten Synchronmaschine austauschen.
Zusätzlich erfolgte ein Vergleich der Integration des Treibers auf einer VAP mit der Integration auf der Ziel-Plattform. Dafür kam ein Entwicklungsboard mit dem Mikrocontroller TriCore TC1798 zum Einsatz, für welchen der MC-ISAR eMotor-Driver entwickelt wurde. Identisch zum Versuchsaufbau mit einer VAP ließ sich das Entwicklungsboard zur Regelung des simulierten Motors mit dem HiL-System verbinden. Somit konnte ein direkter Vergleich der Ausführung des Elektromotortreibers auf der VAP und der Ziel-Plattform durchgeführt werden. Die Vergleichsergebnisse sind im folgenden Kapitel dargestellt.

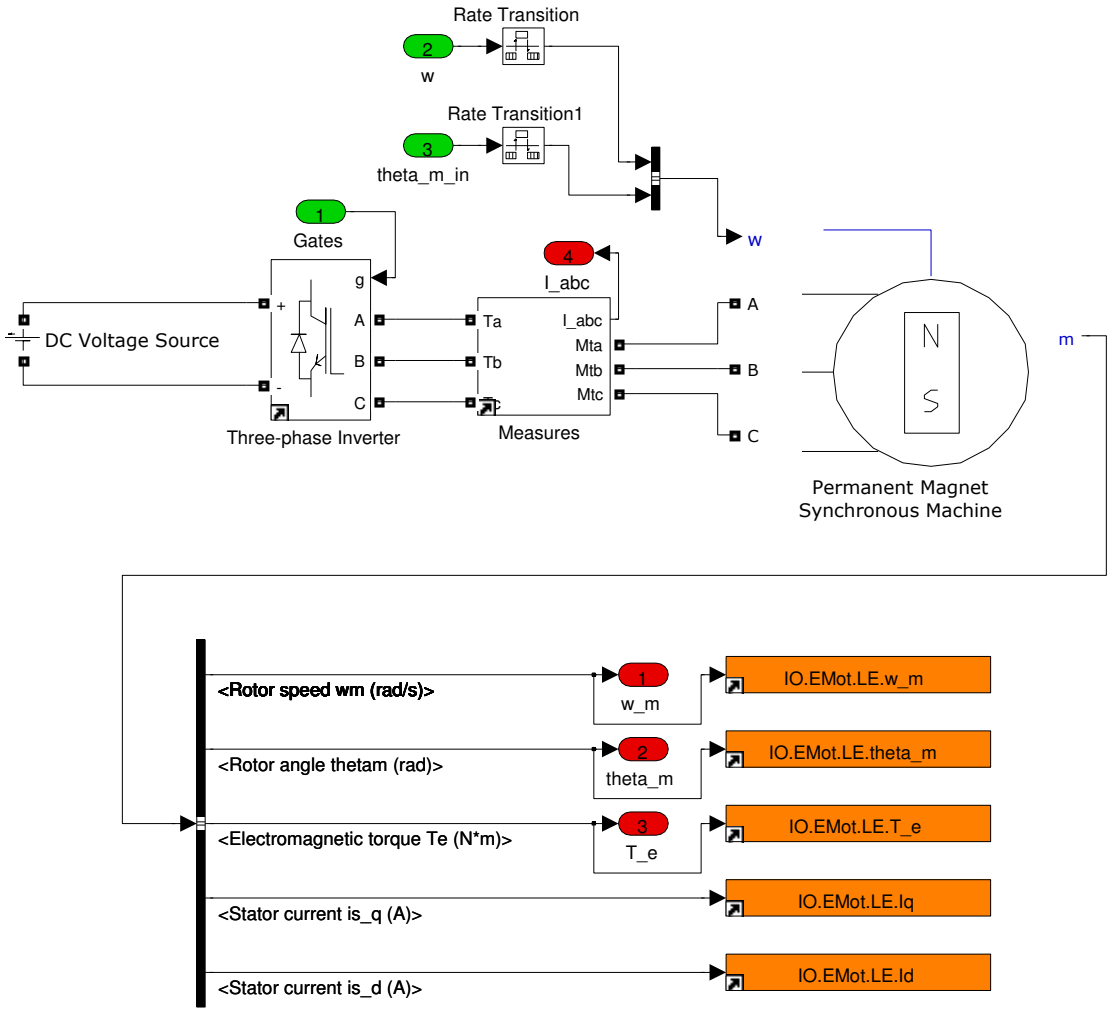


Abbildung 6.22: MATLAB/Simulink-Modell einer Synchronmaschine und der zugehörigen Elektronik.

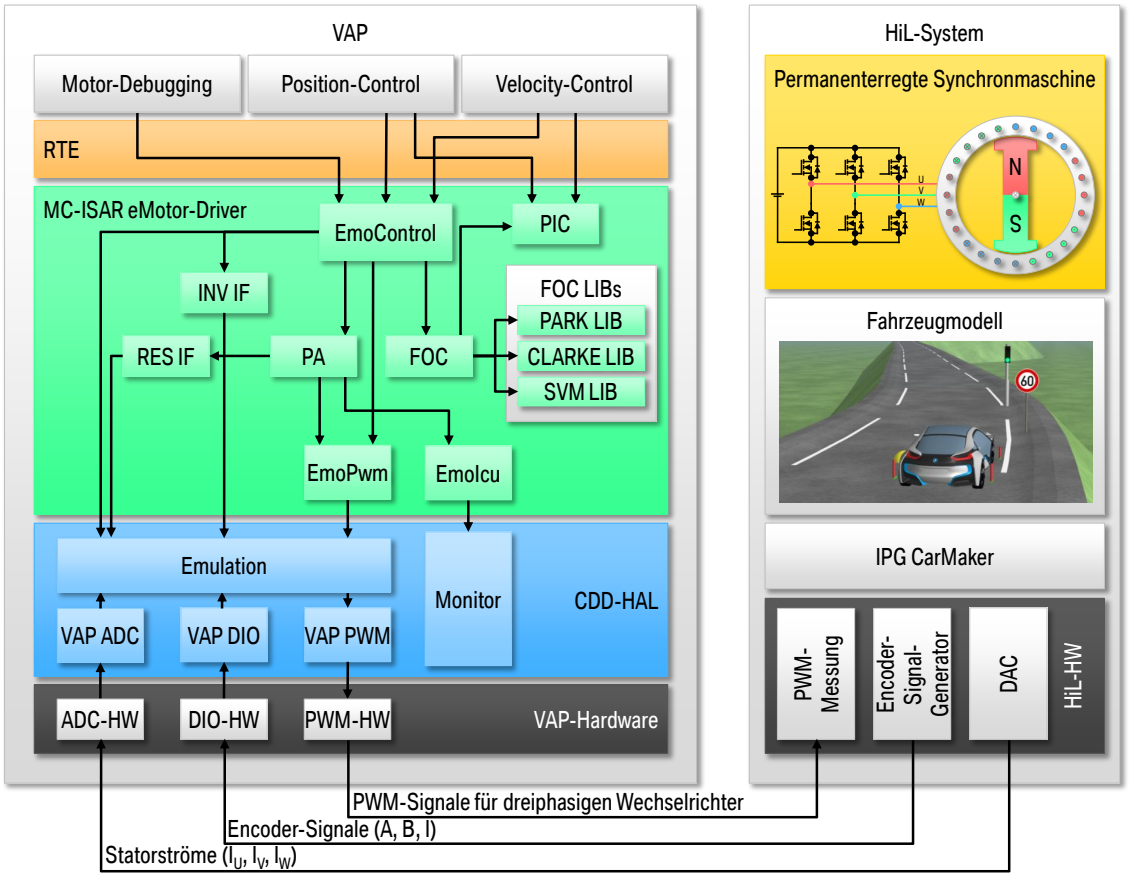


Abbildung 6.23: Versuchsaufbau für die Ansteuerung einer simulierten Synchronmaschine mit dem Elektromotortreiber.

6.3.5 Evaluierung

Das Konzept des CDD-Hardware-Abstraction-Layers hat die vollständige Integration des MC-ISAR eMotor-Drivers auf einer VAP ermöglicht. Alle direkten Hardwarezugriffe des vielschichtigen Complex-Device-Drivers, zur Durchführung der zeitkritischen Elektromotorregelung, konnten erfolgreich über den HAL an die VAP-Hardware angebunden werden. Die Konfiguration des CDD-HAL ließ sich exakt auf die Anforderungen des Elektromotortreibers ausrichten und konnte alle notwendigen Schnittstellen abdecken.
Im Fokus der Integration stand die funktionale Absicherung des Treibers auf einer VAP. Dafür kam ein simulierter Elektromotor auf einem HiL-System zum Einsatz, den der CDD ansteuern sollte. Die Versuchsdurchführung erfolgte unter identischen Bedingungen mit der Integration des Treibers auf einer VAP und der Ziel-Plattform. Somit ließ sich ein direkter Vergleich des Treiberverhaltens auf beiden Systemen vornehmen.
Für die Versuche wurde dem Treiber jeweils eine statische Zielgeschwindigkeit von 60 U/min vorgegeben. Ein simuliertes Fahrzeug mit dem Modell der Synchronmaschine als Antrieb musste auf gerader Fahrbahn aus dem Stillstand auf die Zielgeschwindigkeit beschleunigt werden und danach 60 U/min konstant einhalten. Der Vergleich erfolgte auf Basis von durchgehenden Messungen der PWM-Signale, der Encoder-Signale und der Statorströme. Die direkte Ausgabe der Regelung des CDD stellen die Tastverhältnisse der PWM-Signale für die Ansteuerung des dreiphasigen Wechselrichters dar. Abbildung 6.24 zeigt einen Ausschnitt der Messwerte des ersten PWM-Kanals auf dem Entwicklerboard und der VAP.

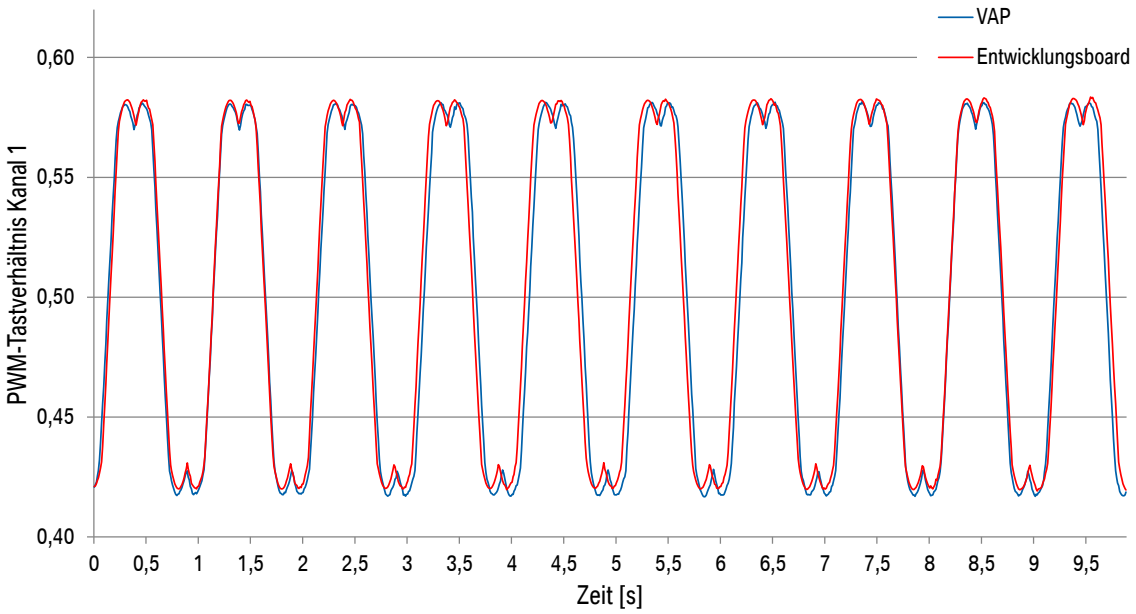


Abbildung 6.24: Vergleichsmessung des Tastverhältnisses des PWM-Kanals 1 auf der VAP und dem Entwicklungsboard.

Der Signalverlauf entspricht der typischen Ansteuerung eines dreiphasigen Wechselrichters. Es handelt sich um ein sinusförmiges Signal das zu seiner zweiten Oberschwingung addiert wurde (siehe [Mic11]). Der Vergleich der Signalausgaben der VAP und des Entwicklerboards zeigt einen sehr ähnlichen Verlauf. Zu den Zeitpunkten 1,38 s und 8,44 s überlagern sich beide Kurven nahezu vollständig. Dazwischen sind geringfügige Abweichungen der Werte

festzustellen. Dies ist auf unterschiedliche Hardware-Module zur PWM-Erzeugung auf den beiden Plattformen sowie verschiedene Signallaufzeiten und Toleranzen im Versuchsaufbau zurückzuführen. Mathematisch betrachtet ergibt sich für den Ausschnitt der Signale aus Abbildung 6.24 nach dem Pearsonschen Korrelationskoeffizient (siehe Formel 6.2, [HEK09]) eine Korrelationsgüte von $r_{xy} = 0,9805$, wobei ein Wert von $r_{xy} = 1$ der maximalen Korrelation entspricht.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x}) \cdot (y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \cdot \sum_{i=1}^{n}(y_i - \overline{y})^2}} \qquad \begin{array}{ccc} r_{xy} & : & \text{Korrelationskoeff.} \\ x & : & \text{Quellvektor} \\ y & : & \text{Prüfvektor} \\ \overline{x}, \overline{y} & : & \text{Mittelwerte} \end{array} \tag{6.2}$$

Somit lässt sich feststellen, dass das Verhalten des Treibers auf der VAP sehr ähnlich zum Verhalten auf der Ziel-Plattform ist. Beide Varianten konnten im Versuch die simulierte Synchronmaschine korrekt ansteuern und das virtuelle Fahrzeug in der Fahrsimulation antreiben.
Die einzige Einschränkung ergab sich in der maximal regelbaren Drehzahl der Synchronmaschine. Die Abtastung der Encoder-Signale auf der Ziel-Plattform ist in der Hardware realisiert und musste auf der VAP innerhalb des HAL nachgebildet werden, wie in Kapitel 6.3.3 beschrieben. Auf der verwendeten VAP ließ sich der Task zur Abtastung allerdings nicht schneller ausführen als im Zyklus von 350 μs, da dies bereits zur vollständigen Auslastung eines eigenen Prozessorkerns geführt hat. Nach dem Abtasttheorem [Sha49] in Formel 6.3 muss die Abtastung mindestens mit der doppelten Maximalfrequenz f_{max} des abgetasteten Signals erfolgen. Für die Synchronmaschine entspricht f_{max} der Rotationsfrequenz f_{Umax} mal der Anzahl der Encoder-Impulse pro Umdrehung I_U. Die Encoder-Signale A und B erzeugen jeweils 512 Impulse pro Umdrehung, womit sich bei einer Abtastfrequenz f_{abtast} von $1/350\,\mu$s nach Formel 6.4 eine maximale Drehzahl von 83,7 U/min ergibt.

$$f_{abtast} \geq 2 \cdot f_{max} = 2 \cdot I_U \cdot f_{Umax} \tag{6.3}$$

$$f_{Umax} = \frac{f_{abtast}}{2 \cdot I_U} = \frac{\frac{1}{350\mu s}}{2\,(2 \cdot 512)} = 1,395Hz = 83,7min^{-1} \tag{6.4}$$

Durch den Austausch des Prozessors in der VAP mit einem leistungsstärkeren Modell ließe sich die Abtastfrequenz weiter erhöhen. Bei niedrigeren Drehzahlen war der Betrieb des Elektromotortreibers allerdings reibungslos möglich und es traten keine weiteren Einschränkungen auf. Das Konzept des CDD-HAL mit seiner generischen Konfigurationsmöglichkeit ließ sich insgesamt optimal auf die Schnittstellen des Elektromotortreibers einstellen. Dadurch konnte der Treiber vollständig auf einer VAP ausgeführt und in seiner Funktion erfolgreich verifiziert werden.

6.4 Zusammenfassung

Die ausgearbeiteten und realisierten Konzepte der CDD-Simulation und des CDD-HAL wurden anhand von drei industrierelevanten Projekten erprobt. Die Auswahl der Anwendungsfälle stellte eine Abdeckung aller drei Testziele (siehe Kapitel 3.1) zur Berücksichtigung von CDDs auf einer VAP sicher.

Das Projekt der Außenlichtsteuerung deckte das Ziel zur Integration der Software-Komponenten ab und somit den Anwendungsfall A_{21} aus Abbildung 3.9. Dabei ließ sich das Konzept der CDD-Simulation erfolgreich anwenden. Durch den automatisierten Ansatz des Konzepts konnten die Software-Komponenten der Außenlichtsteuerung ohne den CDD mit geringem Aufwand integriert und abgesichert werden.
Die Integration des Head-Up-Display-Treibers verfolgte das Ziel zur Sicherstellung der Wiederverwendbarkeit der Software und somit den Anwendungsfall A_{12} aus Abbildung 3.9. Nach dem zugehörigen Auswahlverfahren kam das Konzept des CDD-HAL zum Einsatz. Dabei konnte die vollständige Funktions-Software, inklusive des CDD, auf der VAP integriert und ausgeführt werden. Somit ließ sich sicherstellen, dass die Software für eine Portierung auf andere Plattformen einfach wiederverwendbar ist. Der CDD-HAL ermöglichte die vollständige Anbindung des HUD-Treibers an die Hardware der VAP und darüber die Ansteuerung eines realen Head-Up-Displays.
Das dritte Testziel, die funktionale Absicherung der Software, wurde anhand eines Elektromotortreibers zur Regelung einer Antriebssynchronmaschine erprobt, was dem Anwendungsfall A_{13} aus Abbildung 3.9 entspricht. Die Integration auf der VAP erfolgte mit dem Konzept des CDD-HAL. Durch die flexible Konfiguration des Hardware-Abstraction-Layers ließ sich die vollständige Anbindung der Hardware-Schnittstellen an die VAP realisieren. Dadurch war die korrekte Ansteuerung einer simulierten Synchronmaschine und somit die vollständige funktionale Absicherung des Complex-Device-Drivers möglich.
Insgesamt konnten beide Konzepte zur Berücksichtigung von CDDs in der virtuellen Absicherung erfolgreich an Serienprojekten bestätigt werden. Sowohl die CDD-Simulation, als auch der CDD-HAL ermöglichten die Erfüllung der gestellten Anforderungen und die Erreichung der jeweiligen Testziele der Projekte. Ebenso ließ sich die entwickelte Auswahlmethodik für das jeweils geeignetste Konzept aus Kapitel 3.4 erfolgreich bestätigen, da die selektierten Konzepte in allen drei Fällen zur Erreichung der Testziele geführt haben.

Kapitel 7

Zusammenfassung und Ausblick

In der Automobilindustrie ist ein stetiger Zuwachs an Funktionen im Fahrzeug zu verzeichnen. 90 % der Innovationen basieren auf Elektronik und Software, wobei die Softwareentwicklung bis zu 70 % der gesamten Entwicklungskosten ausmacht. Zusätzlich erhöht sich die Vernetzung von Steuergeräten untereinander sowie die Anzahl der Modellvarianten. Diese Entwicklung stellt die Absicherung vor eine Herausforderung, da die Komplexität und Varianz der Systeme insgesamt zunimmt.

Die virtuelle Absicherung ist eine der Methoden, die zur Beherrschung der Komplexität beiträgt. Sie erlaubt die Abstraktion der Steuergeräte-Software von der zugehörigen Hardware und ermöglicht so eine frühere Software-Absicherung im Entwicklungsprozess. Durch die Sicherstellung der Hardwareunabhängigkeit der Software ist gewährleistet, dass alle Komponenten in neuen Baureihen mit geänderter Steuergeräte-Hardware einfach wiederverwendbar sind. Bei allen großen deutschen OEMs dienen vergleichbare Systeme als Werkzeug zur virtuellen Absicherung, die bei der BMW Group als virtuelle Absicherungsplattform (VAP) bezeichnet werden.

Die Abstraktion der Software von der Ziel-Hardware wird in einer VAP durch den AUTOSAR-Standard ermöglicht, der keine Hardwareabhängigkeiten auf Applikationsebene gestattet. Allerdings enthält der Standard sogenannte Complex-Device-Drivers (CDDs), welche dennoch einen direkten Zugriff auf die Hardware des Steuergerätes erlauben. Sie sind einsetzbar, wenn z. B. kein Standard-Modul für eine Komponente existiert oder hardwarenahe Optimierungen für zeitkritische Systeme benötigt werden. Diese Treiber sind speziell auf ihre zugehörige Hardware ausgelegt, welche in einer VAP nicht verfügbar ist. Dadurch kann ein CDD nicht auf seine üblichen Hardware-Schnittstellen zugreifen und ist nicht direkt integrierbar. Als Teil der Steuergeräte-Software müssen die Treiber jedoch bei einem ganzheitlichen Absicherungsansatz einbezogen werden.

Diese Arbeit hat sich daher mit der Berücksichtigung der hardwareabhängigen CDDs in der virtuellen Absicherung beschäftigt. Das Ziel war die Entwicklung von validen Konzepten für den Umgang mit den Treibern in einer VAP, um die Absicherung der vollständigen Steuergeräte-Software zu ermöglichen. Dabei war die gesamte Bandbreite der Anwendungsfälle und Testziele einer VAP abzudecken. Für ausgewählte Konzepte sollte eine prototypische Realisierung erfolgen, um die Lösungen anhand industrierelevanter Steuergeräte-Projekte evaluieren zu können.

7.1 Konzepte und Umsetzung

Für die Berücksichtigung von CDDs in der virtuellen Absicherung wurden im Rahmen dieser Arbeit insgesamt sieben Konzepte entwickelt. Die Ansätze reichten von der CDD-Simulation, über die Einbindung von Ersatzhardware wie FPGAs, bis zur Integration einer eigenen Hardware-Abstraktionsschicht. Über ein Bewertungssystem erfolgte die Selektion der besten Lösungen für die Erreichung der Testziele und die Abdeckung der Anforderungen unterschiedlicher CDDs. Daraus entstand eine Methodik zur Auswahl des jeweils idealen Konzepts bei der Integration eines spezifischen Treibers. Die Konzepte der CDD-Simulation und des CDD-Hardware-Abstraction-Layers erzielten die höchsten Wertungen und erlaubten die Abdeckung von mehr als 75 % der Anwendungsfälle in der Praxis.
Durch die vollständige CDD-Simulation ließ sich die Absicherung der Software-Komponenten mit Schnittstellen zu einem CDD mit geringem Aufwand sicherstellen. Der entwickelte SWC-Generator ermöglichte die vollautomatische Erzeugung einer Simulationskomponente als Ersatz für den realen Treiber. Mit Hilfe einer generierten, graphischen Oberfläche war die Übertragung von Testvektoren für alle Schnittstellen zwischen CDD und Funktions-Software möglich. Die Anbindung von Testautomatisierungswerkzeugen oder die effiziente Nachmodellierung der wesentlichen CDD-Funktionalitäten erlaubte zusätzlich die Darstellung von komplexen Testszenarien. Die generische Auslegung des Konzepts stellte die Anwendbarkeit für beliebige Treiber sicher.
Das in vier Schichten untergliederte Konzept des CDD-Hardware-Abstraction-Layers erlaubte die vollständige Integration von Complex-Device-Drivers in einer VAP. Die Hardware-Schnittstellen des Treibers, die in der Ziel-Plattform direkt auf den Mikrocontroller zugriffen, wurden in dem HAL an eine Emulationsschicht angekoppelt. Die darunterliegende Schicht der Hardware-Abstraktion ermöglichte wiederum die Ansteuerung der Treiber aller verfügbaren Ein- und Ausgänge einer VAP. Ein eigenes Routing-Konzept gewährte die freie Zuordnung einer Hardware-Schnittstelle des CDD zu einem kompatiblen VAP-Modul, während die Übertragung aller Datenelemente durch sogenannte Monitore gesichert wurde. Eine flexible Konfiguration nach dem Vorbild der AUTOSAR-Basis-Software erlaubte die Anpassung des CDD-HAL für eine Vielzahl verschiedener Complex-Device-Drivers.

7.2 Ergebnisse

Die Erprobung und Evaluierung beider Konzepte erfolgte anhand von realen Steuergeräte-Projekten aus der Serienentwicklung. Bei der Auswahl der Projekte wurde die Abdeckung aller relevanten Testziele berücksichtigt.
Für die Evaluierung der CDD-Simulation diente die Steuergeräte-Software der Außenlichtsteuerung in Fahrzeugen der BMW Group. Dabei sollte die Integration der Software-Komponenten inklusive ihrer Schnittstellen zu einem CDD sichergestellt werden. Der Treiber selbst stand aus Gründen des Informationsschutzes nicht zur Verfügung. Mit dem eigens entwickelten SWC-Generator ließ sich anhand der Schnittstellen der Software-Komponenten vollautomatisch eine passende CDD-Simulation erzeugen. Die ebenso generierte, graphische Oberfläche zur echtzeitfähigen Einspeisung von Testvektoren und Auswertung des Software-Verhaltens erlaubte eine vollständige Analyse aller CDD-Schnittstellen. Insgesamt konnte die Integration der Software-Komponenten und die Kompatibilität aller Schnittstellen anhand von realen Testfällen erfolgreich abgesichert werden.

In einem weiteren Projekt stand die Prüfung der Wiederverwendbarkeit der Software zur Ansteuerung eines Head-Up-Displays (HUD) im Fokus, inklusive dem zugehörigen CDD. Dabei kam das Konzept des CDD-Hardware-Abstraction-Layers zum Einsatz, mit welchem der Treiber vollständig an die verfügbare Hardware einer VAP angebunden werden konnte. Alle vom CDD benötigten, hardwarenahen Funktionen zur Kommunikation mit dem Display ließen sich über den HAL abbilden. Anhand der Testfälle aus der Absicherung der Serienentwicklung war die korrekte Ansteuerung des realen HUD mit einer VAP verifizierbar. Durch die hardwareunabhängige Integration ließ sich die Wiederverwendbarkeit der gesamten Software sicherstellen.
Das Testziel der funktionalen Absicherung von Steuergeräte-Software wurde im Rahmen eines dritten Projekts verfolgt. Hierzu kam ein CDD zur Regelung eines Elektromotors zum Einsatz. Die Herausforderung lag dabei in der besonders zeitkritischen Signalabtastung, Verarbeitung und Ansteuerung. Zur Integration des Treibers diente ebenfalls der CDD-HAL. Die flexible Konfiguration des Konzepts ermöglichte die Anbindung aller hardwarenahen Schnittstellen an die verfügbaren Module einer VAP. Die Funktion des Treibers wurde anhand eines Vergleichs des Verhaltens zwischen der Ziel-Plattform und der VAP sichergestellt, wobei ein funktional identisches Regelverhalten nachweisbar war.
Durch die Erprobung anhand von realer Steuergeräte-Software und den zugehörigen CDDs, konnte die Umsetzung der CDD-Simulation und des CDD-HAL evaluiert und die praktische Einsatzfähigkeit erfolgreich nachgewiesen werden.

7.3 Ausblick

Im Rahmen dieser Arbeit erfolgte eine Realisierung und Evaluierung der Konzepte, die zur Abdeckung der häufigsten Anwendungsfälle von CDDs in der virtuellen Absicherung erforderlich waren. Für eine vollständige Abdeckung der letzten beiden Kombinationsmöglichkeiten von Testzielen und CDD-Anforderungen (siehe Anwendungsfall A_{22} und A_{23} in Abbildung 3.9) wäre zusätzlich die Erprobung der Konzepte zur Simulation der CDD-Hardware und zur Ziel-Hardware-Emulation auf einem FPGA nötig. Beide Möglichkeiten wurden im Rahmen dieser Arbeit im Ansatz betrachtet (siehe Kapitel 3.2.2, bzw. 3.2.6), jedoch nicht vollständig realisiert. Mit der Umsetzung dieser weiteren Konzepte ließe sich eine Abdeckung von 100 % aller CDD-Varianten erreichen.
Die Konzepte der CDD-Simulation und des CDD-HAL wurden bisher prototypisch umgesetzt und an ausgewählten Serienprojekten erprobt. Für den produktiven Einsatz der Lösungen in den Absicherungsprozessen der Serienentwicklung, ist darüber hinaus eine großflächigere Anwendung der Konzepte an weiteren Complex-Device-Drivers notwendig, um die Realisierung zu perfektionieren.
Speziell im Fall des CDD-HAL wäre zusätzlich eine einheitliche Beschreibungsform der Hardware-Schnittstellen von CDDs hilfreich. Dafür würde sich eine Form analog zur Beschreibung der RTE-Schnittstellen anbieten, deren Standardisierung im Rahmen des AUTOSAR-Konsortiums möglich wäre. Mit einer einheitlichen, maschinenlesbaren Schnittstellendefinition ließe sich auch das Konzept des CDD-HAL weitgehend automatisieren, um die Integrationsaufwände zu minimieren und die Vorteile der virtuellen Absicherung vollständig auszunutzen.
Zusätzlich wäre eine Übertragung der Konzepte auf andere Anwendungsfälle außerhalb des AUTOSAR-Kontextes möglich. Die Lösungen eignen sich im Prinzip für alle Bereiche, sofern eine Software-Architektur hardwareabhängige Komponenten beinhaltet, die auf einem System

unabhängig von der Ziel-Hardware abgesichert werden sollen. So wäre z. B. der Einsatz in der Software-Entwicklung für die Avionik ebenfalls denkbar.

Glossar

ADC	Analog-Digital-Converter.
API	Application-Programming-Interface.
APIX	Automotive-Pixel-Link.
Artop	AUTOSAR-Tool-Platform.
ARXML	AUTOSAR-XML.
AShell	APIX-Automotive-Shell.
ASIL	Automotive-Safety-Integrity-Level.
AUTOSAR	AUTomotive Open System ARchitecture.
BSW	Basis-Software.
BSWMD	Basic-Software-Module-Description.
C-PCI	Compact-Peripheral-Component-Interconnect.
CAN	Controller-Area-Network.
CDD	Complex-Device-Driver.
CRC	Cyclic-Redundancy-Check.
DEM	Development-Error-Tracer.
DET	Diagnostic-Event-Manager.
DIO	Digital-Input/Output.
DLT	Diagnostic-Log-and-Trace.
DMA	Direct-Memory-Access.
ECU	Electronic-Control-Unit.
ECUM	ECU-State-Manager.
EMF	Eclipse-Modeling-Framework.
FIFO	First-In-First-Out.
FMI	Functional-Mockup-Interface.
FOC	Field-Oriented-Control.

FPGA	Field-Programmable-Gate-Array.
GENIVI	Offene Plattform für In-Vehicle-Infotainment.
GPT	General-Purpose-Timer.
GPTA	General-Purpose-Timer-Array.
HAL	Hardware-Abstraction-Layer.
HDMI	High-Definition-Multimedia-Interface.
HiL	Hardware-in-the-Loop.
HUD	Head-Up-Display.
IPC	Inter-Process-Communication.
ISR	Interrupt-Service-Routine.
ISS	Instruction-Set-Simulation.
LIN	Local-Interconnect-Network.
MCAL	Microcontroller-Abstraction-Layer.
MiL	Model-in-the-Loop.
MISRA	Motor Industry Software Reliability Association.
MLI	Multiprocessor-Link-Interface.
MOST	Media-Oriented-System-Transfer-Bus.
NM	Network-Management-Interface.
NVM	Non-Volatile-Read-Only-Memory-Manager.
OEM	Original-Equipment-Manufacturer.
OS	Operating-System.
PCI	Peripheral-Component-Interconnect.
PCP	Peripheral-Control-Processor.
PDUR	Protocol-Data-Unit-Router.
PiL	Processor-in-the-Loop.
PMSM	Permanent-Magnet-Synchronous-Motor.
PWM	Pulsweitenmodulation.
RCP	Rapid-Control-Prototyping.
RTE	Runtime-Environment.
RTL	Register-Transfer-Level.
SiL	Software-in-the-Loop.
SPI	Serial-Peripheral-Interface.

SPOC	SPI-Power-Controller.
SuT	System-under-Test.
SWC	Software-Component.
SWCD	Software-Component-Description.
TCP/IP	Transmission-Control-Protocol/Internet-Protocol.
TLM	Transaction-Level-Modeling.
UART	Universal-Asynchronous-Receiver/Transmitter.
UDP	User-Datagram-Protocol.
USB	Universal-Serial-Bus.
VAP	virtuelle Absicherungsplattform.
VESP-A	Virtuelle Software Entwicklungsplattform - AUTOSAR.
VFB	Virtual-Functional-Bus.
VHDL	Very-High-Speed-Integrated-Circuit-Hardware-Description-Language.
VIT	Virtuelle Integrations- und Test-Plattform.
WDGM	Watchdog-Manager.
XiL	X-in-the-Loop.
XML	Extensible-Markup-Language.

Literaturverzeichnis

[AB06] Abel, Dirk und Bollig, Alexander: *Rapid Control Prototyping - Methoden und Anwendungen*, Springer Berlin Heidelberg, 2006, ISBN 978-3-540-29525-9. (Zitiert auf Seiten 24 und 25)

[Arr12] Arroniz, Fernando Francisco Espinosa: *Simulation for Hardware Dependent Components of Electronic Control Unit Software*, Masterthesis, TU Chemnitz, Dez. 2012. (Zitiert auf Seiten 77, 92 und 97)

[Aud15] Audi AG: *Geschäftsbericht 2014*, März 2015. URL http://www.audi.com/content/dam/com/DE/investor-relations/financial_reports/annual-reports/audi_gb_2014_de.pdf (Zitiert auf Seiten v und 1)

[AUT08] AUTOSAR GbR: *Technical Overview*, R3.1 Rev 1, 2008. (Zitiert auf Seiten v, 11, 14, 18 und 38)

[AUT09] AUTOSAR GbR: *Explanation of Interrupt Handling within AUTOSAR*, R4.0 Rev 1, Nov. 2009. (Zitiert auf Seite 18)

[AUT11a] AUTOSAR GbR: *Basisinformation*, Dez. 2011. URL http://www.autosar.org/fileadmin/files/basic_information/AUTOSARBasicInformationShortVersion_DE.pdf (Zitiert auf Seite 9)

[AUT11b] AUTOSAR GbR: *Layered Software Architecture*, R4.0 Rev 3, 2011. (Zitiert auf Seiten v, ix, 10, 11, 16 und 39)

[AUT11c] AUTOSAR GbR: *List of Basic Software Modules*, R4.0 Rev 3, 2011. (Zitiert auf Seite 10)

[AUT11d] AUTOSAR GbR: *Methodology*, R4.0 Rev 3, 2011. (Zitiert auf Seite 10)

[AUT11e] AUTOSAR GbR: *Software Component Template*, R4.0 Rev 3, 2011. (Zitiert auf Seite 10)

[AUT11f] AUTOSAR GbR: *Specification of BSW Module Description Template*, R4.0 Rev 3, 2011. (Zitiert auf Seite 10)

[AUT11g] AUTOSAR GbR: *Specification of ECU Configuration*, R4.0 Rev 3, 2011. (Zitiert auf Seite 10)

[AUT11h] AUTOSAR GbR: *Specification of RTE*, R4.0 Rev 3, 2011. (Zitiert auf Seiten 10 und 80)

[AUT11i] AUTOSAR GbR: *System Template*, R4.0 Rev 3, 2011. (Zitiert auf Seite 10)

[AUT11j] AUTOSAR GbR: *Virtual Functional Bus*, R4.0 Rev 3, 2011. (Zitiert auf Seite 10)

[AUT13] AUTOSAR GbR: *Automotive Open System Architecture*, 2013. URL http://www.autosar.org (Zitiert auf Seiten v, 4, 9 und 10)

[Avi11] Avinun, Ran: *Concurrent Hardware/Software Development Platforms Speed System Integration and Bring-Up*, 2011. URL http://www.cadence.com/rl/Resources/white_papers/system_dev_wp.pdf (Zitiert auf Seite 38)

[BH09] Benra, Juliane T. und Halang, Wolfgang A.: *Software-Entwicklung für Echtzeitsysteme*, Springer-Verlag Berlin Heidelberg, 2009, ISBN 978-3-642-01596-0. (Zitiert auf Seite 25)

[BMW05] BMW Group: *Geschäftsbericht 2005*, März 2005. URL http://www.bmwgroup.com/publikationen/d/2005/pdf/geschaftsbericht_gesamt_2005.pdf (Zitiert auf Seite 2)

[BMW14] BMW Group: *Geschäftsbericht 2013*, März 2014. URL http://www.bmwgroup.com/d/0_0_www_bmwgroup_com/investor_relations/finanzberichte/geschaeftsberichte/2013/_pdf/geschaeftsbericht2013.pdf (Zitiert auf Seite 2)

[BMW15] BMW Group: *Geschäftsbericht 2014*, März 2015. URL http://www.bmwgroup.com/bmwgroup_prod/d/0_0_www_bmwgroup_com/investor_relations/finanzberichte/geschaeftsberichte/2014/_pdf/12507_GB_2014_de_Finanzbericht_Online.pdf (Zitiert auf Seiten v und 1)

[BPSM+08] Bray, Tim; Paoli, Jean; Sperberg-McQueen, C. M.; Maler, Eve und Yergeau, François: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, Nov. 2008. URL http://www.w3.org/TR/2008/REC-xml-20081126/ (Zitiert auf Seite 10)

[Bro13] Broy, Manfred: *Software-Architekturen der Zukunft*, *ATZagenda*, Bd. 2(1):S. 46–49, 2013, ISSN 2194-9174. (Zitiert auf Seite 2)

[CAR11] CAR - Center Automotive Research, Universität Duisburg-Essen: *CAR-Studie zur Typenvielfalt*, *Handelsblatt*, Aug. 2011. (Zitiert auf Seite 1)

[COM14] COMASSO e.V.: *COMASSO Basic Software Platform*, 2014. URL https://www.comasso.org/ (Zitiert auf Seite 33)

[Con14] Continental AG: *Continental Mobilitätsstudie 2013*, 2014. URL http://www.continental-corporation.com/www/presseportal_com_de/allgemein/automatisiertes_fahren_de/ov_mobilitaetsstudie_2013/ (Zitiert auf Seite 2)

[CRKH05] Corbet, Jonathan; Rubini, Alessandro und Kroah-Hartman, Greg: *Linux Device Drivers*, O'Reilly, 3. Aufl., 2005. (Zitiert auf Seite 38)

[Dai15] Daimler AG: *Geschäftsbericht 2014*, März 2015. URL http://www.daimler.com/Projects/c2c/channel/documents/2590211_Daimler_FY_2014_Geschaeftsbericht.pdf (Zitiert auf Seiten v und 1)

[Das14a] Dassault Systèmes: *AUTOSAR Builder und ASIM (AUTOSAR Simulation)*, Vélizy-Villacoublay, Frankreich, 2014. (Zitiert auf Seite 24)

[Das14b] Dassault Systèmes: *Dymola - Dynamic Modeling Laboratory*, Vélizy-Villacoublay, Frankreich, 2014. (Zitiert auf Seite 24)

[DEF12] Dietrich, Michel; Ebert, Michael und Francois, Guillaume: *Validieren und Kalibrieren mit virtuellen Steuergeräten durchgängig bis zur Serie*, *Hanser Automotive*, Bd. 10:S. 14 – 18, 2012. (Zitiert auf Seite 33)

[Dei10] Deicke, Markus: *Entwicklung einer AUTOSAR-Applikation für die virtuelle Absicherungsplattform*, Diplomarbeit, Technische Universität München, Nov. 2010. (Zitiert auf Seite 68)

[DH14] Deicke, Markus und Hardt, Wolfram: *Hardwareabhängige Software in der virtuellen Absicherung*, in *18. Euroforum-Jahrestagung: Elektronik-Systeme im Automobil*, Febr. 2014. (Zitiert auf Seite 76)

[DHM12a] Deicke, Markus; Hardt, Wolfram und Martinus, Marcus: *Simulation hardwarespezifischer Komponenten von ECU-Software in der virtuellen Absicherung*, *ATZ Elektronik*, Bd. 3:S. 226 – 230, Juni 2012. (Zitiert auf Seiten 65, 76 und 77)

[DHM12b] Deicke, Markus; Hardt, Wolfram und Martinus, Marcus: *VAP 2.0 - Die nächste Generation der virtuellen Absicherungsplattform*, in *4. AutoTest Fachkonferenz zum Thema Test von Hard- und Software in der Automobilentwicklung*, S. 133 – 140, Forschungsinstitut für Kraftfahrwesen und Fahrzeugmotoren Stuttgart (FKFS), Okt. 2012. (Zitiert auf Seiten v, 5, 33, 34 und 36)

[DHM12c] Deicke, Markus; Hardt, Wolfram und Martinus, Marcus: *Virtual Validation of ECU Software with Hardware Dependent Components Using an Abstraction Layer*, in *Simulation und Test für die Automobilelektronik, IV*, S. 181 – 188, Technische Universität Berlin & IAV GmbH, Expert Verlag, Mai 2012, ISBN 978-3-8169-3121-8. (Zitiert auf Seiten 69, 76 und 86)

[DHM13] Deicke, Markus; Hardt, Wolfram und Martinus, Marcus: *Simulation hardwarespezifischer Komponenten von ECU-Software in der virtuellen Absicherung*, in *Energieeffiziente Antriebstechnologien - Hybridisierung, Downsizing, Software und IT* (herausgegeben von Siebenpfeiffer,

Wolfgang), S. 196 – 200, Springer Vieweg, 2013, ISBN 978-3-658-00789-8. (Zitiert auf Seiten 65, 76 und 77)

[DJG+10] Dafang, Wang; Jiuyang, Zheng; Guifan, Zhao; Bo, Huang und Shiqiang, Liu: *Survey of the AUTOSAR Complex Drivers in the Field of Automotive Electronics*, in *International Conference on Intelligent Computation Technology and Automation*, 2010. (Zitiert auf Seite 38)

[Ecl14] Eclipse Foundation: *Eclipse Development Platform*, 2014. URL http://www.eclipse.org/ (Zitiert auf Seite 34)

[EESV09] Ecker, Wolfgang; Esen, Volkan; Steininger, Thomas und Velten, Michael: *HW/SW Interface*, in *Hardware-dependent Software* (herausgegeben von Ecker, Wolfgang; Müller, Wolfgang und Dömer, Rainer), S. 95–149, Springer Netherlands, 2009, ISBN 978-1-4020-9435-4. (Zitiert auf Seite 38)

[EHH11] Englisch, Norbert; Helfrich, Toni und Hardt, Wolfram: *Methode zur applikationsspezifischen Absicherung der Basissoftware von AUTOSAR Steuergeräten*, in *9. GI Workshop Automotive Software Engineering*, Berlin, 2011. (Zitiert auf Seite 38)

[EHH13] Englisch, Norbert; Helfrich, Toni und Hardt, Wolfram: *Automatisierte Testfallgenerierung für den applikationsspezifischen Test von AUTOSAR Steuergeräten*, in *Dresdner Arbeitstagung Schaltungs- und Systementwurf (DASS2013)*, S. 76–81, Frauenhofer Verlag, Apr. 2013, ISBN 978-3-8396-0545-5. (Zitiert auf Seite 38)

[Ele14] Elektrobit: *EB tresos AutoCore*, 2014. URL http://automotive.elektrobit.com/home/ecu-software/autosar/eb-tresos-product-line/eb-tresos-autocore.html (Zitiert auf Seite 33)

[EMD09] Ecker, Wolfgang; Müller, Wolfgang und Dömer, Rainer (Hg.): *Hardware-dependent Software - Principles and Practice*, Springer Netherlands, 2009, ISBN 978-1-4020-9435-4. (Zitiert auf Seite 38)

[ETA14] ETAS Group: *ASCET Software-Produkte*, 2014. URL http://www.etas.com/de/products/ascet_software_products.php (Zitiert auf Seite 24)

[FJM10] Fuchs, Maximilian; Jurchen, Stefan und Martinus, Marcus: *Virtuelle Absicherung von E/E-Umfängen*, in *3. AutoTest Fachkonferenz zum Thema Test von Hard- und Software in der Automobilentwicklung*, Forschungsinstitut für Kraftfahrwesen und Fahrzeugmotoren Stuttgart (FKFS), Stuttgart, Okt. 2010. (Zitiert auf Seiten v, 4, 30, 31 und 32)

[FQV+09] Fummi, F.; Quaglia, D.; Vinco, S.; Perbellini, G. und Saggin, S.: *Mixing Simulated and Actual Hardware Devices to Validate Device Drivers in a Complex Embedded Platform*, in *10th International Workshop on Microprocessor Test and Verification*, 2009. (Zitiert auf Seite 39)

[Fri10] Frickenstein, Elmar: *Veränderungen in der Automobilindustrie. Der neue BMW 5er - Das digitale Zeitalter hat begonnen*, in *14. Fachkongress Automobilelektronik*, Ludwigsburg, Juni 2010. (Zitiert auf Seiten v, 3 und 4)

[Fri11] Frickenstein, Elmar: *Elektronik im BMW von Morgen*, *Automobil-Elektronik*, Bd. 1(1):S. 14 – 17, Febr. 2011. (Zitiert auf Seiten v und 3)

[Fuj10] Fujitsu Technology Solutions GmbH: *Fujitsu APIX Remote Handler - Light remote access via the high-speed pixel link (APIX)*, Okt. 2010. URL http://www.fujitsu.com/downloads/MICRO/fme/documentation/a43.pdf (Zitiert auf Seite 105)

[GEN12] GENIVI Alliance: *Get your IVI products to market faster*, 2012. URL http://www.genivi.org (Zitiert auf Seiten 4 und 33)

[GLMS02] Grötker, Thorsten; Liao, Stan; Martin, Grant und Swan, Stuart: *System Design with SystemC*, Springer US, 2002, ISBN 978-1-4020-7072-3. (Zitiert auf Seite 38)

[Har96] Hardt, Wolfram: *HW-SW-Codesign auf Basis von C-Programmen unter Performanz-Gesichtspunkten*, Dissertation, Universität Paderborn, Paderborn, 1996. (Zitiert auf Seite 25)

[Har01] Hardt, Wolfram: *Integration von Verzögerungszeit-Invarianz in den Entwurf eingebetteter Systeme*, C-LAB publication, Shaker, 2001, ISBN 978-3826582516. (Zitiert auf Seite 2)

[HEK09] Hartung, Joachim; Elpelt, Bärbel und Klösener, Karl-Heinz: *Statistik Lehr- und Handbuch der angewandten Statistik*, Oldenbourg, München, 15., überarb. und wesentlich erw. aufl. Aufl., 2009, ISBN 978-3-486-71054-0. (Zitiert auf Seite 126)

[HK13] Honisch, Artur und Krügel, Karsten: *Virtuelle Integration und Test von E/E-Fahrzeugsystemen*, *ATZ Elektronik*, Bd. 5:S. 350 – 355, 2013. (Zitiert auf Seiten 33 und 37)

[Hon13] Honisch, Artur: *Virtuelle Integration und Test (VIT) Plattform - Reif für die Serie*, in *Workshop Automotive Software Engineering - Virtuelle Absicherung*, Chemnitz, Juni 2013. (Zitiert auf Seite 37)

[HR97] Hardt, Wolfram und Rosenstiel, Wolfgang: *Prototyping of Tightly Coupled Hardware/Software-Systems*, *Design Automation for Embedded Systems*, Bd. 2(3-4):S. 283–317, 1997, ISSN 0929-5585, verschiedene Formen der Emulation über FPGAs. (Zitiert auf Seite 25)

[IEE06] *IEEE Standard 1364-2005 - Verilog Hardware Description Language*, IEEE Computer Society, Apr. 2006, ISBN 0-7381-4850-4. (Zitiert auf Seiten 38 und 49)

[IEE09] *IEEE Standard 1076-2008 - VHDL Language Reference Manual*, IEEE Computer Society, New York, USA, Jan. 2009, ISBN 978-0-7381-6854-8. (Zitiert auf Seiten 38 und 49)

[Inf12] Infineon Technologies AG: *Infineon's MC-ISAR eMotor Driver - 3-phase motor control for mass production*, Juni 2012.
URL http://www.infineon.com/eMotor (Zitiert auf Seiten vii und 114)

[Inf14] Infineon Technologies AG: *SPI Power Controller / SPOC^{TM}*, 2014.
URL http://www.infineon.com/spoc (Zitiert auf Seite 96)

[Ino10] Inova Semiconductors GmbH: *APIX2 - 3GBit/s Digital Automotive Pixel Link Transmitter*, 2010.
URL http://www.inova-semiconductors.de/en/documents/INAP375TAdvanceInformation.pdf (Zitiert auf Seite 105)

[Ino12] Inova Semiconductors GmbH: *APIX2 ADK Transmitter Board (Rev2)*, Apr. 2012. (Zitiert auf Seite 107)

[ISO05] *ISO 9000:2005: Qualitätsmanagementsysteme - Grundlagen und Begriffe*, International Organization for Standardization, Dez. 2005. (Zitiert auf Seite 22)

[ISO11] *ISO 26262:2011: Road vehicles - Functional safety*, International Organization for Standardization, Genf, Schweiz, 2011. (Zitiert auf Seiten 4, 5 und 19)

[KBR09] Krause, Matthias; Bringmann, Oliver und Rosenstiel, Wolfgang: *Verification of AUTOSAR Software by SystemC-Based Virtual Prototyping*, in *Hardware-dependent Software* (herausgegeben von Ecker, Wolfgang; Müller, Wolfgang und Dömer, Rainer), S. 261–293, Springer Netherlands, 2009, ISBN 978-1-4020-9435-4. (Zitiert auf Seite 39)

[Lin10] *ISO/IEC 23360 Part 1:2010 - Linux Standard Base Core Specification 4.1*, Linux Foundation, 2010. (Zitiert auf Seite 75)

[Mat10] MathWorks, Inc., The: *MATLAB and Simulink Release 2010bSP1*, Natick, Massachusetts, United States, 2010. (Zitiert auf Seiten 24 und 68)

[MBt14] MBtech Group GmbH & Co. KGaA: *PROVEtech:RP - Virtuelle Absicherungsplattform VAP*, 2014.
URL https://www.mbtech-group.com/eu-de/electronics_solutions/tools_equipment/provetechrp_rapid_prototyping_fuer_autosar.html (Zitiert auf Seite 33)

[MCW12] Martinus, Marcus; Cutura, Zoran und Würz, Thomas: *Virtuelle Absicherungsplattform - Integration und Wiederverwendung von Software*, *ATZ Elektronik*, Bd. 1:S. 56 – 61, Febr. 2012. (Zitiert auf Seiten 4 und 31)

[MDF13a] Martinus, Marcus; Deicke, Markus und Folie, Michael: *Hardwareunabhängige Integration von Seriensoftware im virtuellen Fahrversuch*, in *16. Internationaler Kongress Elektronik im Fahrzeug*, Bd. 2188 von *VDI-Berichte*, S. 739 – 750, VDI Wissensforum GmbH, Baden-Baden, Okt. 2013, ISBN 978-3-18-092188-4. (Zitiert auf Seite 36)

[MDF13b] Martinus, Marcus; Deicke, Markus und Folie, Michael: *Virtueller Fahrversuch - Hardwareunabhängige Integration von Seriensoftware*, *ATZ Elektronik*, Bd. 5:S. 344 – 349, Okt. 2013. (Zitiert auf Seite 36)

[Men14] Mentor Graphics: *Volcano^{TM} VSTAR AUTOSAR Basic Software*, 2014. URL http://www.mentor.com/products/vnd/autosar-products/volcano-vstar (Zitiert auf Seite 33)

[MGIL07] Mössinger, Jürgen; Grote, Walter; Illg, Bernd und Lapp, Andreas: *AUTOSAR-Basis-Software CUBAS - Ressourcenschonende und zuverlässige AUTOSAR-Basis-Software von Bosch*, *Elektronik automotive*, Bd. Sonderausgabe AUTOSAR, Okt. 2007. (Zitiert auf Seite 33)

[Mic11] Michel, Manfred: *Leistungselektronik - Einführung in Schaltungen und deren Verhalten*, Springer Berlin Heidelberg, Berlin, 2011, ISBN 978-3-642-15983-1. (Zitiert auf Seite 125)

[MIS94] MISRA - The Motor Industry Software Reliability Association: *Development Guidelines for Vehicle Based Software*, Nov. 1994, ISBN 0 9524156 0 7. (Zitiert auf Seiten ix, 22 und 23)

[MIS12] MISRA - The Motor Industry Software Reliability Association: *Guidelines for the Use of the C Language in Critical Systems*, März 2012, ISBN 978-1-906400-10-1. (Zitiert auf Seite 13)

[Mod12] Modelica Association Project "FMI": *Functional Mock-up Interface for Model Exchange and Co-Simulation*, 2012. URL http://www.functional-mockup-interface.org/specifications/FMI_for_ModelExchange_and_CoSimulation_v2.0_Beta4.pdf (Zitiert auf Seite 34)

[Nat95] National Semiconductor Corporation: *PC16550D Universal Asynchronous Receiver/Transmitter with FIFOs*, Juni 1995. URL http://www.ti.com/lit/ds/symlink/pc16550d.pdf (Zitiert auf Seite 107)

[OIC15] OICA - Organisation Internationale des Constructeurs d'Automobiles: *Sales Statistics*, 2015. URL http://www.oica.net/category/sales-statistics/ (Zitiert auf Seiten v und 1)

[O'N01] O'Neill, Jim: *Building Better Global Economic BRICs*, in *Goldman Sachs Global Economics*, Nr. Paper No 66, 2001. (Zitiert auf Seite 1)

[Ope12] OpenGL: *The Industry's Foundation for High Performance Graphics*, 2012. URL http://www.opengl.org/ (Zitiert auf Seite 33)

[Ora15] Oracle Corporation: *Java*, 2015. URL http://www.java.com (Zitiert auf Seiten 77 und 91)

[OVP14] OVP: *Open Virtual Platforms - the source of Fast Processor Models & Platforms* , 2014. URL http://www.ovpworld.org/ (Zitiert auf Seite 39)

[PJ09] Popovici, Katalin und Jerraya, Ahmed: *Hardware Abstraction Layer*, in *Hardware-dependent Software* (herausgegeben von Ecker, Wolfgang; Müller, Wolfgang und Dömer, Rainer), S. 67–94, Springer Netherlands, 2009, ISBN 978-1-4020-9435-4. (Zitiert auf Seite 39)

[Pos80] Postel, Jonathan Bruce: *Internet Standard RFC 768 - User Datagram Protocol*, Aug. 1980.
URL http://tools.ietf.org/html/rfc768 (Zitiert auf Seite 66)

[PWMS12] Piasecki, Thomas; Weber, Michael; Martinus, Marcus und Schmid, Hermann: *Virtuelle Absicherung zur Beherrschung der Komplexität und Varianz von E/E Systemen*, in *16. Internationaler Fachkongress Fortschritte in der Automobil-Elektronik*, Süddeutscher Verlag Veranstaltungen GmbH, Forum am Schlosspark, Ludwigsburg, Juni 2012. (Zitiert auf Seite 37)

[QEm14] QEmu: *Open Source Processor Simulator*, 2014.
URL http://www.qemu.org (Zitiert auf Seite 39)

[RDT04] Reuss, Hans-Christian; Diesner, Stefan und Thoben, M.: *Alternative concepts of power distribution for comfort electronics in passenger vehicles*, in *AUTOREG 2004: Steuerung und Regelung von Fahrzeugen und Motoren*, S. 605 – 616, 2004, ISBN 3180918284. (Zitiert auf Seite 2)

[RVE10] Rudorfer, Michael; Voget, Stefan und Eberle, Stephan: *Artop Whitepaper*, Techn. Ber., Artop - AUTOSAR Tool Platform, 2010. (Zitiert auf Seiten 77 und 91)

[SBPM08] Steinberg, Dave; Budinsky, Frank; Paternostro, Marcelo und Merks, Ed: *EMF: Eclipse Modeling Framework*, Addison-Wesley Professional, 2. Aufl., 2008, ISBN 978-0-321-33188-5. (Zitiert auf Seite 77)

[Sch10] Schirrmeister, Frank: *Accelerating Software Driver Development Using Virtual Prototypes*, Aug. 2010.
URL http://www.synopsys.com/apps/docs/pdfs/sld/sls_usb_wp.pdf (Zitiert auf Seiten 38 und 44)

[SDG09] Schirner, Gunar; Dömer, Rainer und Gerstlauer, Andreas: *High-Level Development, Modeling and Automatic Generation of Hardware-Dependent Software*, in *Hardware-dependent Software* (herausgegeben von Ecker, Wolfgang; Müller, Wolfgang und Dömer, Rainer), S. 203–231, Springer Netherlands, 2009, ISBN 978-1-4020-9435-4. (Zitiert auf Seite 38)

[Sha49] Shannon, Claude Elwood: *Communication in the Presence of Noise*, in *Proceedings of the IRE*, Bd. 37, S. 10–21, Jan. 1949. (Zitiert auf Seiten 74 und 126)

[SHWKR08] Schick, Bernhard; Henning, Josef; Wurster, Uwe und Klein-Ridder, Birgit: *Simulationsmethoden zur Evaluierung und Verifizierung von Funktion, Güte und Sicherheit von Fahrerassistenzsystemen im durchgängigen MIL-, SIL- und HIL-Prozess*, in *3. Tagung Aktive Sicherheit durch Fahrerassistenz*, Apr. 2008. (Zitiert auf Seite 122)

[SM13] Siegl, Sebastian und Miegler, Maximilian: *Vorgelagerte Absicherung der Steuergeräte-Architektur und Funktion durch virtuelle Integration auf VESP-A*, in *Workshop Automotive Software Engineering - Virtuelle Absicherung*, Chemnitz, Juni 2013. (Zitiert auf Seite 37)

[SN05] Smith, James E. und Nair, Ravi: *The Architecture of Virtual Machines*, *Computer*, Bd. 38(5):S. 32 – 38, Mai 2005. (Zitiert auf Seite 26)

[SN13] Schutter, Tom De und Nohl, Achim: *Virtual Prototypes for Early Software Development: Requirements, Solutions and Use Cases*, Aug. 2013.
URL http://www.synopsys.com/cgi-bin/sld/pdfdla/docsdl/vp_early_sw_dev_wp.pdf?file=vp_early_sw_dev_wp.pdf (Zitiert auf Seiten 38 und 44)

[SP10] Schmitt, Robert und Pfeifer, Tilo: *Qualitätsmanagement: Strategien, Methoden, Techniken*, Carl Hanser Verlag GmbH & Co. KG, 4. vollständig überarbeitete auflage Aufl., 2010, ISBN 978-3446412774. (Zitiert auf Seiten 4 und 32)

[SZ13] Schäuffele, Jörg und Zurawka, Thomas: *Automotive Software Engineering - Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen*, Vieweg + Teubner, Wiesbaden, 5 Aufl., 2013, ISBN 978-3-8348-2470-7. (Zitiert auf Seiten v, ix, 2, 17, 18, 22, 23, 24, 25 und 26)

[VDA05] VDA - Verband der Automobilindustrie: *Jahresbericht 2005*, Juli 2005.
URL https://www.vda.de/de/downloads/477/ (Zitiert auf Seiten 2 und 3)

[VDL14] Vogel, Dirk; Deicke, Markus und Langenwalter, Joachim: *Virtuelle Absicherung im AUTOSAR-Workflow zwischen OEM und Zulieferer*, *Hanser Automotive*, Bd. 5, 2014. (Zitiert auf Seite 37)

[VMo12] *V-Modell® XT*, 2012.
URL http://www.v-modell.iabg.de/ (Zitiert auf Seiten v und 22)

[Vog13] Vogel, Dirk: *Konzeption und Umsetzung einer konfigurierbaren Hardware-Abstraktions-Schicht für Complex-Device-Driver*, Masterarbeit, TU Chemnitz, Okt. 2013. (Zitiert auf Seiten 70, 74, 86, 92 und 117)

[WHR08] Wiese, Matthias; Hetzel, Günter und Reuss, Hans-Christian: *Optimierung von E/E-Funktionstests durch Homogenisierung und Frontloading*, in *2. AutoTest Fachkonferenz zum Thema Test von Hard- und Software in der Automobilentwicklung*, Forschungsinstitut für Kraftfahrwesen und Fahrzeugmotoren Stuttgart (FKFS), Stuttgart, Okt. 2008. (Zitiert auf Seiten 4 und 32)

[WR06] Wallentowitz, Henning und Reif, Konrad (Hg.): *Handbuch Kraftfahrzeugelektronik: Grundlagen - Komponenten - Systeme - Anwendungen*, Vieweg + Teubner Verlag, 1. Aufl., 2006, ISBN 978-3-8348-9121-1. (Zitiert auf Seiten 24 und 25)

[YJ03] Yoo, Sungjoo und Jerraya, Ahmed Amine: *Introduction to Hardware Abstraction Layers for SoC*, in *Embedded Software for SoC* (herausgegeben

von JERRAYA, Ahmed Amine; YOO, Sungjoo; VERKEST, Diederik und WEHN, Norbert), S. 179–186, Springer US, 2003, ISBN 978-1-4020-7528-5. (Zitiert auf Seite 39)

[ZELM12] ZIEGENBEIN, Dirk; EBERT, Michael; LAUFF, Ulrich und MALDENER, Nico: *Multi-Level Simulation of Integrated AUTOSAR ECU Software*, in *Simulation und Test für die Automobilelektronik, IV*, S. 71 – 79, Technische Universität Berlin and IAV GmbH, Expert Verlag, Mai 2012, ISBN 978-3-8169-3121-8. (Zitiert auf Seite 33)

[ZMG09] ZABEL, Henning; MÜLLER, Wolfgang und GERSTLAUER, Andreas: *Accurate RTOS Modeling and Analysis with SystemC*, in *Hardware-dependent Software* (herausgegeben von ECKER, Wolfgang; MÜLLER, Wolfgang und DÖMER, Rainer), S. 233–260, Springer Netherlands, 2009, ISBN 978-1-4020-9435-4. (Zitiert auf Seite 39)

Eingebettete, selbstorganisierende Systeme im Universitätsverlag Chemnitz

8. Meisel, André (2010)
   Design Flow für IP basierte, dynamisch rekonfigurierbare, eingebettete Systeme
   ISBN 978-3-941003-15-6
   Volltext: http://nbn-resolving.de/urn:nbn:de:bsz:ch1-201000890

9. Vodel, Matthias (2010)
   Funkstandardübergreifende Kommunikation in Mobilen Ad Hoc Netzwerken
   ISBN 978-3-941003-18-7
   Volltext: http://nbn-resolving.de/urn:nbn:de:bsz:ch1-201001164

10. Reichelt, Toni (2012)
    A Model Driven Approach for Service Based System Design Using Interaction Templates
    ISBN 978-3-941003-60-6
    Volltext: http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-85986

11. Caspar, Mirko (2013)
    Lastgetriebene Validierung Dienstbereitstellender Systeme
    ISBN 978-3-941003-84-2
    Volltext: http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-110257

12. Heller, Ariane (2013)
    Systemeigenschaft Robustheit: ein Ansatz zur Bewertung und Maximierung von Robustheit eingebetteter Systeme
    ISBN 978-3-941003-87-3
    Volltext: http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-113369

13. Vodel, Matthias (2014)
    Energieeffiziente Kommunikation in verteilten, eingebetteten Systemen
    ISBN 978-3-944640-05-1
    Volltext: http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-131891

14. Tudevdagva, Uranchimeg (2014)
    Structure Oriented Evaluation Model for E-Learning
    ISBN 978-3-944640-20-4
    Volltext: http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-146901

15. Gruber, Thomas (2016)
    Prozessintegrierte Dokumentation und optimierte Wiederverwendung von Simulationsmodellen der automobilen Funktionsabsicherung
    ISBN 978-3-944640-89-1
    Volltext: http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-202845

16. Deicke, Markus (2018)
    Virtuelle Absicherung von Steuergeräte-Software mit hardwareabhängigen Komponenten
    ISBN 978-3-96100-039-5
    Volltext: http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-230123