Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Jens Torfstecher

Simulationsbasierte Entwicklung eines AUTOSAR-konformen Steuergerätes

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jens Torfstecher

Simulationsbasierte Entwicklung eines AUTOSAR-konformen Steuergerätes

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Technische Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Franz Korf
Zweitgutachter: Prof. Dr. rer. nat. Thomas Lehmann

Abgegeben am 10.10.2013

Jens Torfstecher

Thema der Bachelorarbeit
Simulationsbasierte Entwicklung eines AUTOSAR-konformen Steuergerätes
Stichworte
AUTOSAR, Automotive Fallstudie, Werkzeugkette, modellbasierter Entwicklungsprozess, SystemDesk
Kurzzusammenfassung
AUTOSAR (AUTomotive Open System ARchitecture) ist ein internationaler Standard der Automobilindustrie. Entwickelt und getragen wird diese offene und standardisierte Softwarearchitektur von Automobilherstellern, Automobilzulieferern und Werkzeugherstellern. Ziel dieser Arbeit ist es, ein AUTOSAR-konformes Steuergerät zu entwickeln. Die Vorgehensweise in den einzelnen Entwicklungsschritten wird anhand einer Fallstudie, bestehend aus einem vereinfachten Adaptive Frontlighting System, veranschaulicht.

Jens Torfstecher

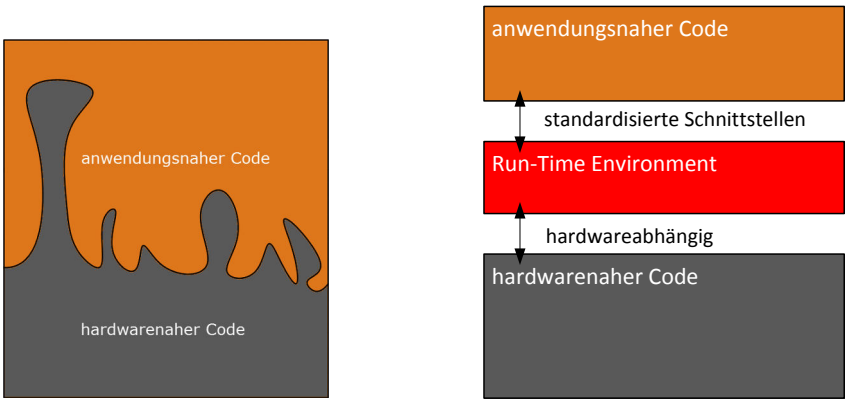
Title of the paper
Simulation-based development of an AUTOSAR-compliant control unit
Keywords
AUTOSAR, automotive case study, toolchain, model based development prozess, SystemDesk
Abstract
AUTOSAR (AUTomotive Open System ARchitecture) is an international part of automobile industry. This development partnership is open and international standardized. It's developed and used by differents parts of automobile industry. The primary target is to develop an electronic control unit that is AUTOSAR-compliant. Different stages of development are shown by a detailed case study, consisting of a simplified adaptive frontlighting system.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Entwicklung von Steuergeräten im automobilen Bereich war in der Vergangenheit sehr hersteller- und fahrzeugspezifisch. Diese steuergerätzentrierte Entwicklung bringt Nachteile mit sich. Schon eine geänderte Anforderung kann eine Neuentwicklung notwendig machen. Die Entwicklung ist mit einem hohen Aufwand und Kosten verbunden. Zusätzlich stellen höhere Varianten- und Ausstattungsvielfalt in modernen Fahrzeugen sowie die Implementierung auf heterogenen Kommunikationsplattformen mit spezialisierter Sensorik/Aktorik, Bussystemen etc. neue Herausforderungen dar [13]. Namenhafte Hersteller und Zulieferer, wie zum Beispiel BMW, Daimler, Bosch, Continental und Volkswagen, gründeten die **AUT**omotive **O**pen **S**ystem **AR**chitecture-Initiative (AUTOSAR) um eine konzeptionelle Neuorientierung herbeizuführen. Diese soll den steuergerätezentrierten Ansatz aufgeben und den funktionsbasierten Ansatz verfolgen. AUTOSAR definiert einen gemeinsamen Standard für die Spezifikation und Implementierung von Steuergeräte-Software.



(a) klassisches Steuergerät (b) Steuergerät mit AUTOSAR

Abbildung 1.1: Vergleich klassisches Steuergerät zu AUTOSAR

Durch die Einführung von einer standardisierten Basissoftware und Applikationssoftware mit einheitlichen Schnittstellen (Abb. 1.1) entstehen für alle am Entwicklungsprozess beteiligten Partner Qualitätsvorteile und Kostenoptimierungen[4].

In dieser Arbeit wird die Entwicklung eines AUTOSAR-konformen Steuergerätes beschrieben. Hierfür wurde anhand einer Fallstudie auf alle notwendigen Schritte des Entwicklungsprozesses von der Analyse bis hin zur Implementierung sowie der Integration und den Tests detailliert eingegangen. Die Fallstudie stellt ein vereinfachtes Adaptive Frontlighting System dar, das unter anderem die Funktionen des Kurvenlichtes realisiert.

Inhaltliche Gliederung

In Kapitel 2 werden Grundlagen zu AUTOSAR beschrieben. Das Konzept, die Architektur und die Methodik stehen in diesem Kapitel im Vordergrund. Das Kapitel 3 beinhaltet die Analyse der Anforderungen der durchgeführten Fallstudie. Im folgenden vierten Kapitel wird auf die Entwicklung des Steuergerätes, vom Entwurf über die Implementierung und das Testen, eingegangen. Abschließend wird im Kapitel 5 eine Zusammenfassung getätigt. Alle verwendet Abkürzungen sind im Abkürzungsverzeichnis zu finden.

Kapitel 2

Grundlagen

AUTOSAR ist ein internationaler Standard der Automobilindustrie. Diese offene und standardisierte Architektur wurde gemeinsam von namenhaften Automobilherstellern, Automobilzulieferen und Toolherstellern entwickelt, mit folgender Motiviation nach [2]:

- Verwaltung der E/E Komplexität verbunden mit Wachstum im Funktionsumfang;
- Flexibilität bei Produktänderungen, Upgrade & Update;
- Skalierbarkeit von Lösungen innerhalb und über Produktlinien hinaus;
- verbesserte Qualität und Zuverlässigkeit der E/E-Systeme

und diesen Zielen:

- Anforderungen an das Fahrzeug, wie Verfügbarkeit und Sicherheit,
  Software Upgrades / Updates und Wartbarkeit;
- erhöhte Skalierbarkeit und Flexibilität bei der Integration und der Transferierung von Funktionen;
- höhere Wiederverwendung von SW-und HW-Komponenten über Produktlinien hinaus;
- verbesserte Beherrschung von Produkt- und Prozess-Komplexität sowie Risiken;
- Kostenoptimierung von skalierbaren Systemen.

Der Standard beschreibt zum Einen die technische Infrastruktur und zum Anderen auch die Methodik und die Beschreibungsformate für die Entwicklung AUTOSAR-konformer Spezifikationen. In den folgenden Abschnitten wird AUTOSAR vom Allgemeinen bis zum Speziellen erläutert.

2.1 AUTOSAR Konzept

Das AUTOSAR Konzept lässt sich in drei Schwerpunkte aufteilen.

Den ersten Schwerpunkt bildet die **Architektur**. Durch diese wird es möglich, die Anwendungssoftware unabhängig von der benutzten Hardware zu verwenden. Für die Realisierung nutzt AUTOSAR ein Schichtenmodell (vgl. Abbildung 2.2), das die Software in Anwendungssoftware, Run-Time Environment (RTE) und Basis Software unterteilt. Auf jede Schicht wird in den folgenden Abschnitten ausführlich eingegangen.

Die **Methodik** bildet den zweiten Schwerpunkt. Sie beschreibt den Ablauf des Entwicklungsprozesses in den einzelnen Phasen. Hierfür werden Austauschformate und Beschreibungsvorlagen definiert, die der Verteilung der Software über Steuergerätgrenzen hinweg dienen.

Der dritte Schwerpunkt sind die **Anwendungsschnittstellen**. Hier werden Schnittstellen typischer Automotivsoftwareanwendungen festgelegt, um die Integration in Bezug auf Syntax und Semantik zu erleichtern.

In allen Schwerpunkten sollen mit Hilfe von Standardisierungen gewünschte Effekte wie Kostenoptimierung und Erhöhung der Qualität erreicht werden. AUTOSAR gibt dabei nur die Schnittstellen vor und lässt den Unternehmen genügend Freiraum gemäß dem AUTOSAR-Leitspruch: “Cooperate on standards, compete on implementation“.

AUTOSAR Software Component

Die Teilung in Anwendung und Infrastruktur (vgl. Kapitel 2.2) ist ein wichtiges Konzept in AUTOSAR. Eine Applikation besteht aus miteinander kommunizierenden AUTOSAR Softwarekomponenten (SW-C). Jede dieser Softwarekomponenten beinhaltet einen Teil der Gesamtfunktionalität der Anwendung. Als Atomic Software Component wird eine Instanz einer Komponente bezeichnet, wenn diese nur einer Electronic Control Unit (ECU) zugeordnet werden kann (vgl. [14]).

Virtual Functional BUS

Der Virtual Functional BUS (VFB) dient dazu, auf Systemebene Kommunikationsbeziehungen der Softwarekomponenten zu beschreiben. Dies erfolgt auf einem abstrakten hardwareunabhängigen Level. SW-C stehen somit auch Services der Basis Software, die ECU Abstraktion und komplexe Gerätetreiber (Complex Device Drivers) zur Verfügung. Fest definierte Ports dienen der Interaktion der Komponenten untereinander. Deshalb sollte darauf geachtet werden, dass keine API-Funktionen oder Variablen der einzelnen Module direkt aufgerufen werden, sondern jeweils die entsprechenden Ports mit den definierten Schnittstellen (vgl. [11]). Der Virtual Functional BUS wird im weiteren Entwicklungsprozess von der Run-Time Environment implementiert (siehe Abschnitt 2.2).

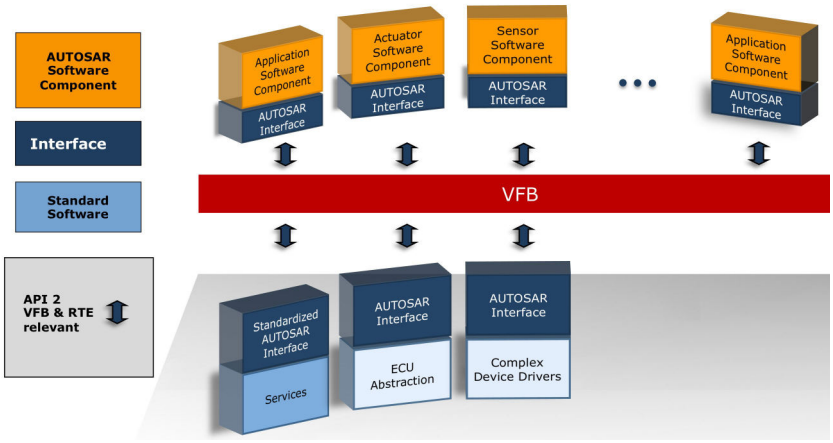


Abbildung 2.1: Virtual Functional BUS

Ein Port ist eindeutig zu einer Instanz einer Komponente gebunden, andererseits ist es aber auch möglich, dass es mehrere Instanzen einer Komponente gibt. Die Mehrfachinstanziierung dient unter anderem der Ausfallsicherheit. Für die Kommunikation stehen im Wesentlichen zwei Muster zur Verfügung:

- Client/Server-Kommunikation
- Sender/Receiver-Kommunikation

Bei der Client/Server-Kommunikation bietet der Server Dienste in Form von Operationen an, die vom Client genutzt (aufgerufen) werden können. Mithilfe der Sender/Receiver-Kommunikation werden Nachrichten von einem Sender an viele Empfänger gesendet.

Die abstrakte Modellierung der Kommunikationsbeziehungen und eine möglichst späte oder wiederholbare Extraktion der ECU-Sichten ist ein grundlegendes Mittel, um die AUTOSAR-Ziele Modularität, Austauschbarkeit, Verschiebbarkeit, Skalierbarkeit und Wiederverwendbarkeit zu erreichen.

2.2 AUTOSAR-Schichtenarchitektur

Um die Aktivitäten der Entwicklung eines Steuergerätes effizient realisieren zu können, ist eine wohlstrukturierte Architektur der Software notwendig.

AUTOSAR hat hierfür ein Schichtenmodell gewählt, welches auch als AUTOSAR Layered Software Architecture bezeichnet wird. Auf dem Steuergerät wird die Software in vier Abstraktionsschichten unterteilt, siehe Abbildung 2.2.

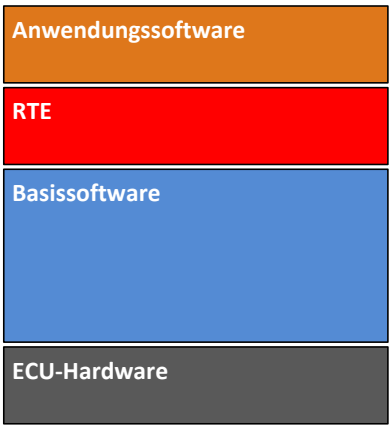


Abbildung 2.2: AUTOSAR Layered Software Architecture

Anwendungssoftware

Die oberste Schicht ist die Anwendungssoftware. Wie Abbildung 2.2 zeigt liegt diese Schicht über der RTE und beinhaltet die hardwareunabhängig Anwendungssoftwarekomponenten sowie die hardwareabhängig Sensor/Aktor-Softwarekomponenten. Die Trennung dieser Komponenten stellt später eine mögliche Portierung der Anwendungssoftwarekomponenten auf unterschiedliche Hardware sicher.

Aufgrund der Aufteilung der Software in voneinander unabhängige Komponenten ist es notwendig, dass sie untereinander und mit der Basissoftware kommunizieren. Diese Kommunikation wird in der darunterliegenden Schicht - dem RTE - realisiert.

Run-Time Environment

Das RTE implementiert die VFB Funktionalität. Das RTE ist eine Middleware, über welche die Kommunikation der Anwendungssoftware stattfindet. Für jedes Steuergerät wird das RTE spezifisch zusammen mit der jeweiligen Steuergerätebeschreibung generiert. Hierbei sind die Schnittstellen zur oberen und unteren Schicht standardisiert. Dies unterstützt die Verschiebbarkeit von Anwendungen zwischen den Steuergeräten. Werden in dem späteren Entwicklungsprozess Anwendungen zwischen ECUs verschoben, ändert sich die Systemsicht und somit auch der VFB. Dies hat eine Neugenerierung der betroffenen RTEs zur Folge.

Beim Betrachten zweier Funktionen, ist es durch das RTE möglich, dass diese ohne Kenntnis der Signalpfade Informationen miteinander austauschen, indem sie die standardisierten Kommunikationsports des RTE verwenden. Durch dieses Konzept ist es möglich, Anwendungen unabhängig von der späteren Topologie im Fahrzeug zu entwickeln.

Basissoftware

Unterhalb des RTE liegt die Basissoftware. Diese Schicht abstrahiert von der ECU Hardware. Die Basissoftware stellt das Kernstück der AUTOSAR Spezifikation dar. Aus diesem Grund wird in Kapitel 2.3 genauer auf den Aufbau und die einzelnen Module eingegangen.

Die Basissoftware lässt sich in verschiedene Layer aufteilen, wie auf Abbildung 2.3 zu sehen ist und im Folgenden beschrieben werden.

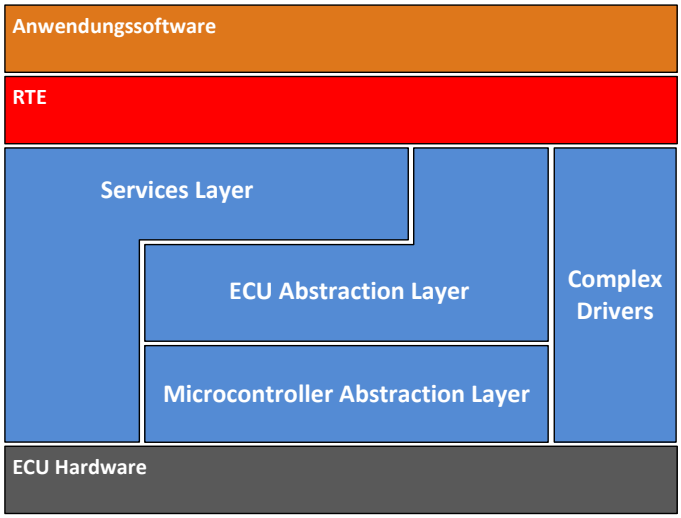


Abbildung 2.3: AUTOSAR Layered Software Archtecture 2

Serviceschicht (Services Layer)

Auf oberster Ebene unmittelbar unter dem RTE befindet sich der Services Layer.

Zu den Aufgaben des Services Layers gehört es Anwendungen und Basissoftwaremodulen grundlegende Dienste (Services) zur Verfügung zu stellen. Diese Services sind beispielsweise:

- Betriebssystem-,
- Speicherverwaltungs-,
- Diagnose- und
- Kommunikationsfunktionen.

Ein Teil der Implementierung der Module ist mikrocontroller- und steuergerätspezifisch. Nach oben sind die Schnittstellen der Module jedoch hardwareunabhängig (vgl. Abschnitt 2.2).

Steuergeräteabstraktionsschicht (ECU Abstraction Layer)

Der ECU Abstraction Layer dient der Abstraktion der steuergerätespezifischen Eigenschaften. Es wird beispielsweise von der Anzahl der verbauten CAN-Controller und deren genauem Ort (on-chip/on-board) auf dem Steuergerät abstrahiert.

Es befinden sich in dieser Schicht beispielsweise Interfacemodule für die darunterliegenden hardwareabhängigen Treiber.

Microcontrollerabstraktionsschicht (MicroController Abstraction Layer)

Der Microcontroller Abstraction Layer (MCAL) bildet die unterste Schicht der Basissoftware und liegt unmittelbar über der Hardware.

Die Aufgabe dieser Schicht besteht darin, vom Mikrocontroller zu abstrahieren. Das beinhaltet die Konfiguration und die Initialisierung des Mikrocontroller und integrierter Geräte. Die Abstraktion zur darüberliegenden Steuergeräteabstraktionsschicht erfolgt über ein Standard Interface.

Der MCAL ist somit controllerspezifisch und muss bei einem Austausch des Controllers ebenfalls ersetzt werden.

Complex Device Driver

Die Complex Device Drivers befinden sich neben der Basissoftware.

Diese Schicht hebt die geschaffene Abstraktion wieder auf, da sie das Schichtenmodell durchschneidet. Es sollte in der Entwicklung so wenig Software wie möglich als Complex Driver umgesetzt werden. Jedoch ist es in einigen Fällen erforderlich, Software mit zeitkritischen Anforderungen zu erstellen und direkt auf die Hardware zuzugreifen, wie zum Beispiel die Ansteuerung und Verwaltung von Sensoren.

Des Weiteren kann der Complex Driver bei der Migration existierender Software behilflich sein.

ECU-Hardware

Die unterste Schicht ist die ECU-Hardware. Diese umfasst die verwendete Hardware wie Prozessoren, Aktuatoren und Sensoren.

2.3 Basissoftware

Der Schwerpunkt der Standardisierungsbemühungen der AUTOSAR- Enwicklungspartnerschaft liegt auf der Basissoftware. Es hat zwar jeder OEM seine eigene standardisierte Basissoftware, einen sog. Standard-Core, jedoch kann ein einheitlicher Core aller OEMs zu

Effizienzsteigerungen und Kosteneinsparungen bei der Entwicklung führen. Der Ausgangspunkt für diese Annahme ist dabei, dass man davon ausgeht, dass die Steuergeräte und ihre Software häufig von Zulieferern entwickelt werden und diese dann nur auf einen Standard, den AUTOSAR-Core, aufsetzen und nicht eine Vielzahl von Standard-Cores unterstützen müssen.

Um einen Überblick zu geben, werden im Folgenden einige Module der Basissoftware vorgestellt.

2.3.1 Aufbau

Die Basissoftware kann in sechs vertikale Bereiche unterteilt werden:

- Systemdienste,
- Geräte-Stack,
- Speicher-Stack,
- Kommunikations-Stack,
- I/O-Stack und
- Complex Device Drivers

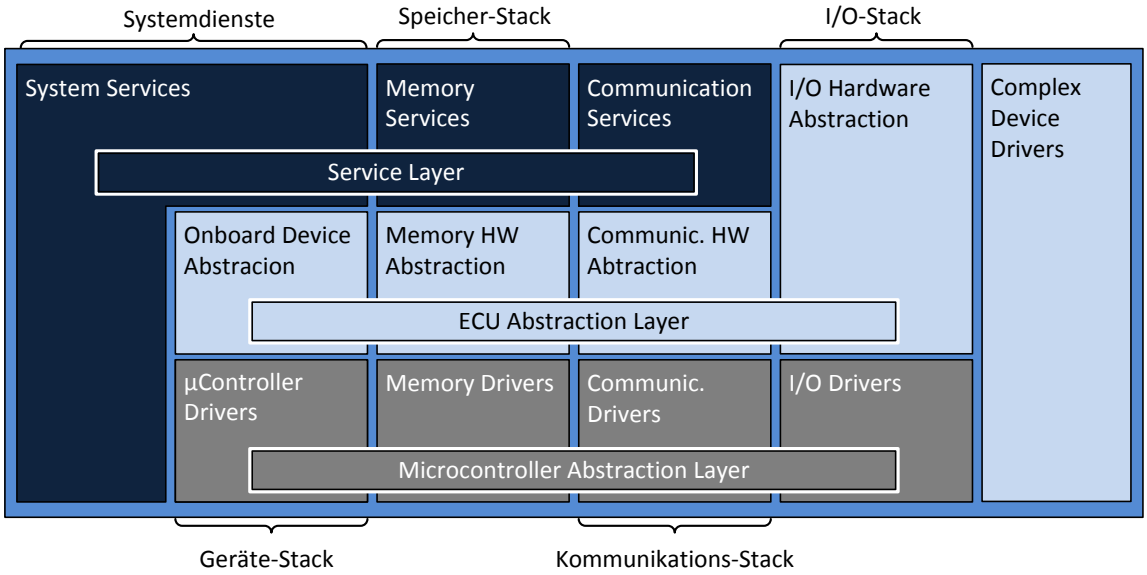


Abbildung 2.4: Horizontale und vertikale Gliederung der Basissoftware

Wie schon im oberen Abschnitt 2.2 gezeigt wird, werden diese vertikalen Bereiche von den horizontalen Abstraktionsschichten überspannt. Die Abbildung 2.4 verdeutlicht dies noch einmal. Hier werden die Layer weiter unterteilt. In einer funktionalen Gruppe wie zum Beispiel den Memory Services sind Module zusammengefasst die ähnliche Funktionen besitzen oder eng bei der Realisierung einer Aufgabe zusammenarbeiten.

2.4 AUTOSAR Methodik

Die AUTOSAR-Methodik [1] beschreibt den technischen Ablauf des AUTOSAR-Entwicklungsprozesses in mehreren Phasen. Hierzu kommen in den unterschiedlichen Phasen oftmals verschiedene Werkzeuge zum Einsatz, deren Austauschbarkeit durch festgelegte Formate ermöglicht wird. Die Abbildung 2.5 zeigt die Phasen und Abläufe der Methodik.

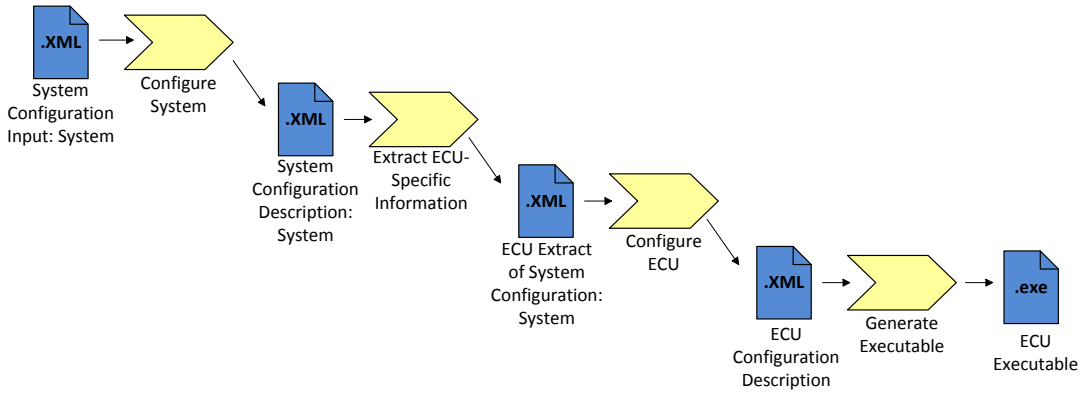


Abbildung 2.5: AUTOSAR-Methodik [1]

Die AUTOSAR-Methodik legt fest, zu welchen Zwecken und zu welchen Zeitpunkten Beschreibungen im Entwicklungsprozess verwendet werden. Die Methodik beschreibt dabei keine Zuständigkeiten und Rollen innerhalb der Entwicklung. Die Methodik teilt den Entwurf eines Systems in zwei große Bereiche auf. Im ersten Teil, der Systemebene oder Systemkonfiguration, wird der Entwurf der gesamten Software des Fahrzeugs beschrieben. Im zweiten Teil werden die Arbeitsschritte auf der Steuergerätebene beziehungsweise Steuergerätekonfiguration erläutert [11]. Diese Zweiteilung wird in der Abbildung 2.6 durch eine Trennlinie gekennzeichnet.

Systemkonfiguration

Zunächst wird die Eingabeinformation der Systemkonfiguration definiert. Hierbei werden die Software- und die Hardware-Komponenten ausgewählt und die Systemeigenschaften be-

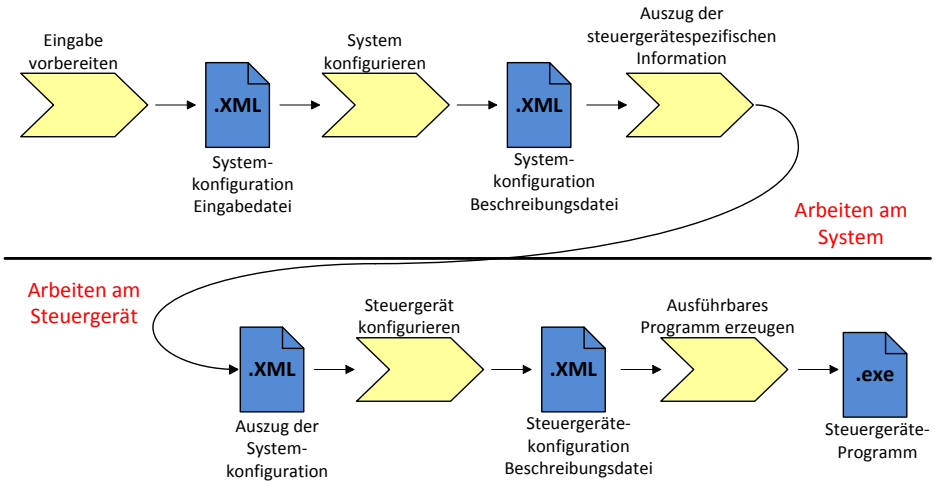


Abbildung 2.6: Zweiteilung der AUTOSAR-Methodik

stimmt. Aus praktischer Sicht bedeutet dies, dass entsprechende Vorlagen ausgefüllt beziehungsweise bearbeitet werden (AUTOSAR-Templates).

Die System-Konfiguration beschreibt die Zuordnung (Mapping) von Software-Komponenten auf Steuergeräte. Die Konfiguration erfolgt in drei Schritten:

- Beschreibung der System-Konfiguration mit allen Informationen (Mapping, Hardware-Topologie);
- Zuordnung jeder Software-Komponente zu einem Steuergerät;
- Beschreibung der Eigenschaften der verwendeten Netzwerke und Medien über die Kommunikationsmatrix.

Als letzten Schritt vor der Trennlinie in Abb. 2.6 werden steuergerätespezifische Informationen extrahiert. Dabei werden die Daten, die für die Ausführung der Software eines bestimmten Steuergeräts nötig sind, aus der Beschreibung der System-Konfiguration ermittelt. Anschließend kann mit der Konfiguration des Steuergeräts fortgefahren werden [11].

Steuergerätekonfiguration

In der Steuergerätekonfiguration werden alle notwendigen Informationen für die Implementierung (z. B. Task Scheduling, benötigte Basis-Software-Module und deren Konfiguration) hinzugefügt. Dieser Schritt liefert die Beschreibung der Steuergerätekonfiguration, welche wiederum für den letzten Schritt der AUTOSAR-Methodik benötigt wird. Hier wird der Programm-Code des jeweiligen Steuergeräts aus der gegebenen Beschreibung generiert.

Bei jeder Änderung oder Ergänzung von Software-Komponenten auf einem Steuergerät, oder der Modifikation des Kommunikationsnetzwerks zwischen den Steuergeräten muss die Steuergerätekonfiguration angepasst werden.

Kapitel 3

Analyse

Statistiken zeigen, dass Autofahrten bei Nacht das höchste Risiko bergen. Aus einem Fahranteil bei Nacht von ca. 25 Prozent gehen über 36 Prozent aller Schwerverletzten und 46,6 Prozent aller Getöteten hervor [12]. Fahrzeughersteller und Zulieferer verfolgen daher das Ziel, die Sicht bei Nacht zu verbessern. Herstellerübergreifend wurde ab 1993 im Rahmen des EUREKA-Projektes EU 1403 AFS (Advanced Frontlighting System) an der Entwicklung intelligenter Scheinwerfersysteme gearbeitet [16]. Mit Veröffentlichung der geänderten ECE-Regelungen Anfang 2003 ist es in Europa erlaubt, statisches und dynamisches Kurvenlicht einzuführen. Die zusätzlichen Lichtfunktionen verfolgen das Ziel, in bestimmten Fahrsituationen (z.B. Abbiegevorgang, Kurvenfahrt) eine Verbesserung der Sichtverhältnisse zu erreichen.

Während bisher die Lichtrichtung ausschließlich starr nach vorne ausgerichtet war, erlaubt die AFS-Gesetzgebung, Licht seitlich abzustrahlen (statisches Kurvenlicht) und in Kurven dem Straßenverlauf folgend zu schwenken (dynamisches Kurvenlicht).

In den folgenden Abschnitten werden die Funktionen des Adaptive Frontlighting System analysiert und definiert.

3.1 Adaptive Frontlighting System

In der Abbildung 3.1 ist das Adaptive Frontlighting System zusammen mit seinen peripheren Komponenten in einer schematischen Zeichnung dargestellt. Die hellgrau unterlegten Teile der Abbildung beschreiben die Funktionalitäten des AFS. Die weiß unterlegten Elemente bezeichnen diejenigen Fahrzeug-Komponenten, mit denen das AFS direkt interagiert (über Sensoren und Aktuatoren, die direkt mit dem AFS verbunden werden). Im Folgenden werden die in der Abbildung angedeuteten drei Funktionalitäten des AFS in informeller Weise aus Benutzersicht näher beschrieben.

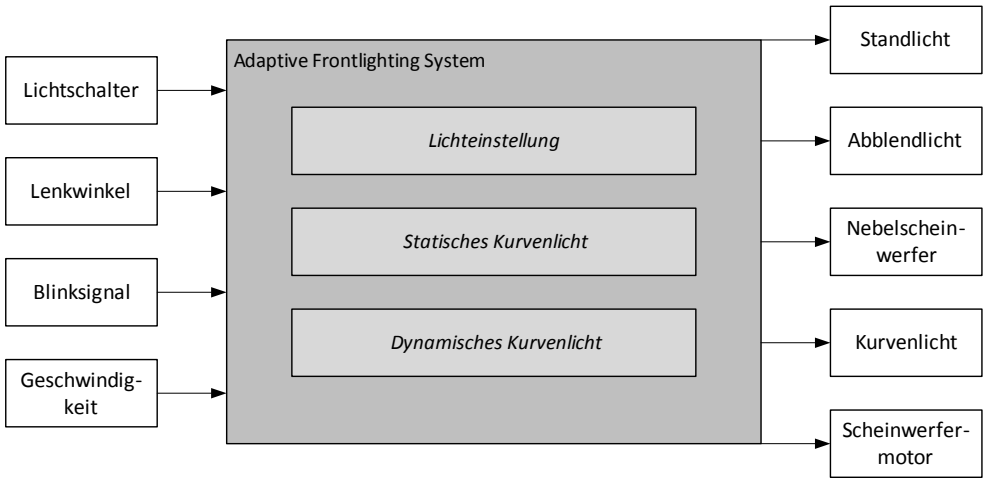


Abbildung 3.1: Adaptive Frontlighting System

3.1.1 Lichteinstellung

Der Benutzer kann mit Hilfe eines Schalters das gewünschte Fahrtlicht einstellen. Hierbei gibt es die Schalterstellungen AUS, Standlicht, Fahrtlicht und Nebellicht. Je nach Sichtverhältnissen kann die entsprechende Einstellung während der Fahrt gewählt werden.

3.1.2 Statisches Kurvenlicht

Durch das statische Kurvenlicht verbessert sich für den Benutzer die Ausleuchtung beim Abbiegen im innerstädtischen Bereich. Die Reichweite dieser Zusatzbeleuchtung ist sehr stark beschränkt. Das statische Kurvenlicht wird deshalb auch nur im Bereich niedriger Geschwindigkeiten aktiviert, wenn auf Grund des Fahrzeugzustandes (z. B. bei eingeschaltetem Richtungsblinker sowie beim entsprechenden Einschlagen des Lenkrades) auf einen Abbiegevorgang geschlossen werden kann. Mit dem Kurvenlicht wird die Sichtweite in Kurven trotz der geringen Reichweite vergrößert, weil ohne Kurvenlicht in dem Bereich nur Streulicht des Scheinwerfers verfügbar ist (Abblidung 3.2).

Funktion	Einschaltbedingungen	Ausschaltbedingungen
statisches Kurvenlicht rechts ODER links	Abblendlicht EIN UND Fahrzeuggeschwindigkeit $v \leq 40\frac{km}{h}$ UND Blinker rechts bzw. links EIN	Wegfall einer Einschaltbedingung

Tabelle 3.1: Die Ein- und Ausschaltbedingungen (statisch)

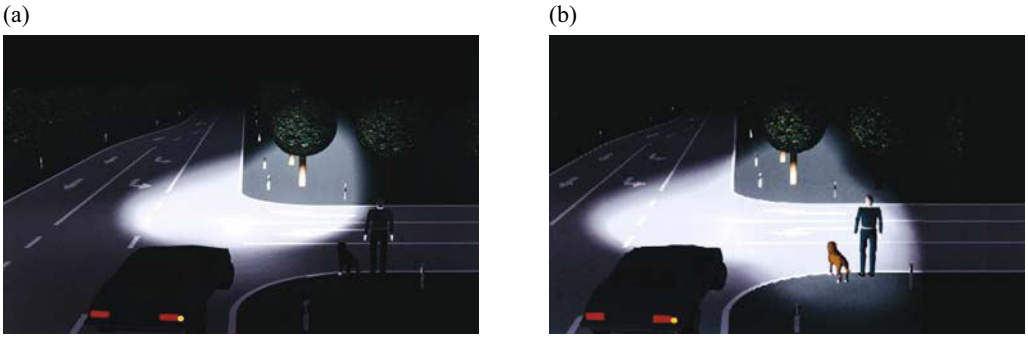


Abbildung 3.2: Sichtverhältnisse beim Abbiegen: (a) Konventionelles Abblendlicht. (b) Abblendlicht mit zusätzlich statischem Kurvenlicht [5]

3.1.3 Dynamisches Kurvenlicht

Das dynamische Kurvenlicht bietet eine Verbesserung der Fahrbahnausleuchtung bei nächtlichen Kurvenfahrten. Hierfür wird aus Messgrößen der Kurvenradius berechnet und das Abblendlicht mit Hilfe eines Motors im Schweinwerfer in die Kurve geschwenkt . Die Vorteile sind in Abbildung 3.3 sichtbar: Ein Hindernis wird frühzeitig erkannt, ein Ausweichen ist möglich. Die erzielten Reichweitenvorteile sind von erheblichem Vorteil im Bereich von Radien unter 1000 m. Bei einem Kurvenradius von 200 m beträgt der Vorteil etwa 60 Prozent (siehe Abbildung 3.4).

Funktion	Einschaltbedingungen	Ausschaltbedingungen
dynamisches Kurvenlicht	Abblendlicht EIN UND Fahrzeuggeschwindigkeit $50\frac{km}{h} \leq v < 300\frac{km}{h}$ UND Lenkwinkel $\delta > 0°$	Wegfall einer Einschaltbedingung

Tabelle 3.2: Die Ein- und Ausschaltbedingungen (dynamisch)

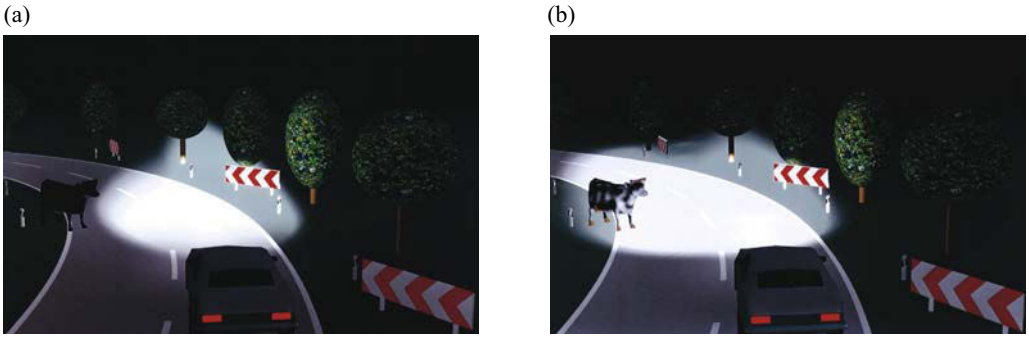


Abbildung 3.3: Sichtverhältnisse in einer Kurve: (a) Abblendlicht. (b) Dynamisches Kurvenlicht [5]

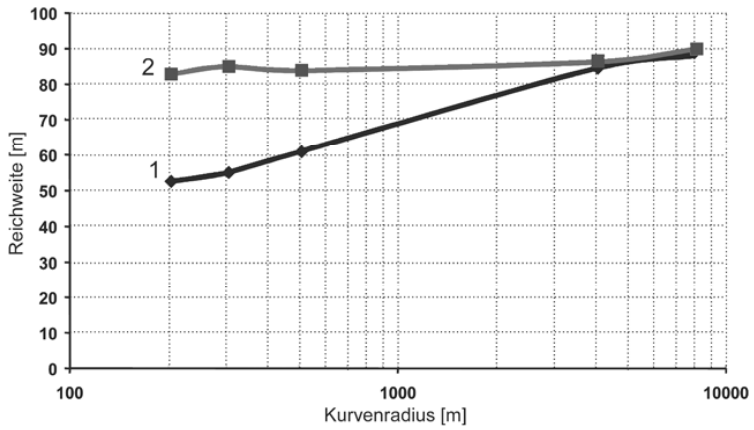


Abbildung 3.4: Reichweite des Abblendlichts in Kurven: 1 ohne, 2 mit Schwenken [16]

Berechnung Dynamisches Kurvenlicht

Bei der Berechnung des dynamischen Kurvenlichtes gibt es unterschiedliche Vorgehensweisen. Es wird in linieare und nichtlineare Berechnungen unterschieden. Die nichtlineare Berechnung verwendet beispielsweise die Vorder- und Hinterradaufhängung, den Getriebezug, die Geschwindigkeit und den Lenkwinkel. Die Parameter werden in Abhänigigkeit zueinander verarbeitet. Ein Algorithmus zu der nichtlinearen Berechnung wird in [17] ausführlich beschieben.

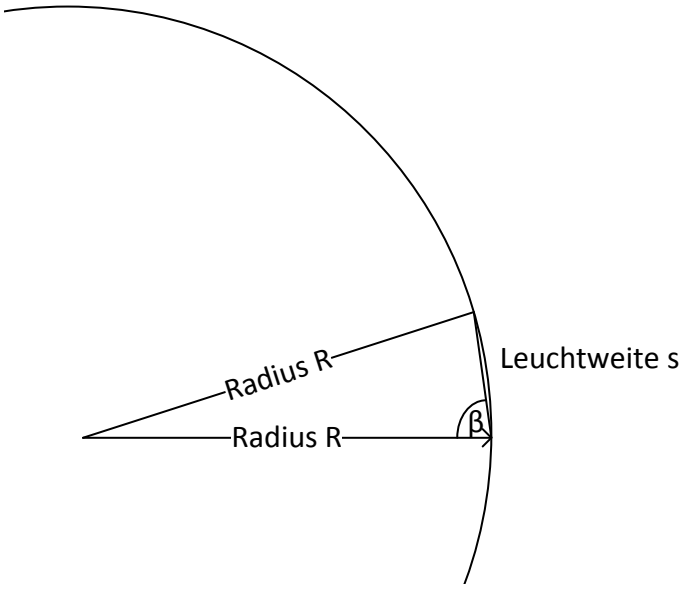


Abbildung 3.5: Kreis/Kosiunssatz

Bei der Fallstudie wurde zur Bestimmung des Scheinwerferschwenkwinkel eine lineare Berechnung verwendet. Ausgehend vom Kosinussatz und der Gleichung zur Berechnung des Kurvenradius wurde der Schwenkwinkel des Scheinwerfermotors berechnet. In Abbildung 3.5 ist ein kreisförmiger Kurvenverlauf dargestellt. Mit Hilfe des Radius R und der gewünschten Leuchtweite der Scheinwerfer s und des Kosiunssatzes ergibt sich zur der Berechnung von dem Winkel β :

$$R^2 = s^2 + R^2 - 2 \cdot R \cdot s \cdot cos\beta$$

Der Radius wird mit

$$R = \frac{I}{sin(\delta)}$$

ermittelt. δ gibt den Lenkwinkel des Fahrzeuges an, der als Eingangsparameter zur Verfügung steht. Der Radstand I ist als Konstante gegeben [7]. Die Geschwindigkeit v fließt als Faktor x in die Berechnung des Schwenkwinkels ein und ist der Tabelle 3.3 zu entnehmen. Aus den beschriebenen Funktionen ergibt sich, nachδ aufgelöst, folgende Gleichung:

$$\beta = arccos\left(\frac{s}{2 \cdot \frac{I}{sin(\delta)}}\right)$$

Aus den berechenten Größen ergibt sich der Schwenkwinkel α**:**

$$\alpha = (90° - \beta) \cdot x$$

v in $\frac{km}{h}$	50	60	70	80	90	100	120	140	160	180
x	1	1,05	1,1	1,15	1,2	2	2,05	2,1	2,15	2,2

Tabelle 3.3: Faktor in Abhänigkeit der Geschwindigkeit

Kapitel 4

Entwicklung

Das folgende Kapitel geht auf die Entwicklung der AUTOSAR Fallstudie detailliert ein. Hierbei wird auf verschiedene Schritte des Wasserfallmodells (Abbildung 4.1) eingegangen.

Als Erstes wurde zunächst die Analyse und Definition der Anforderungen in Kapitel 3 begonnen. Auf Grundlage der Anforderungen erfolgt nun der Entwurf für die Spezifikation der Software- und Systemarchitektur. Die AUTOSAR-konforme Systemarchitektur wurde mit dem Tool SystemDesk von der Firma dSpace spezifiziert.

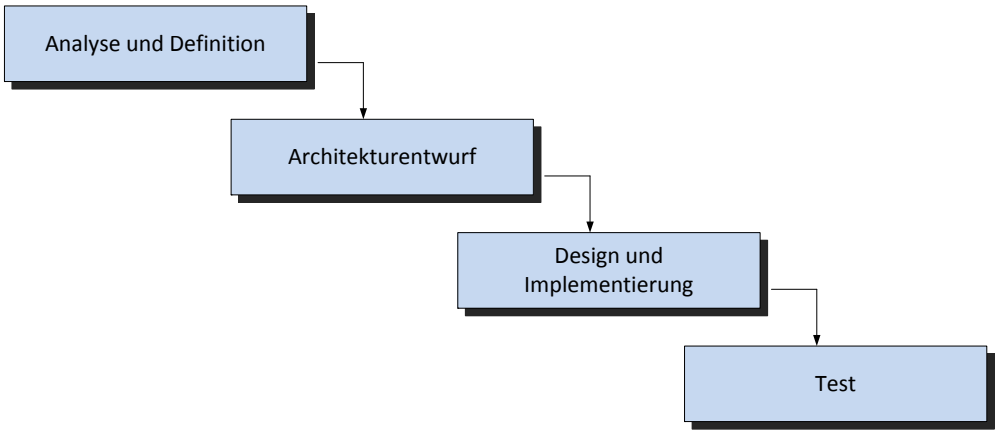


Abbildung 4.1: Wasserfallmodell

Anschließend folgt der Schritt des Designs und der Implementierung. Diese wurde im Rahmen einerseits modellbasiert mit MATLAB/Simulink durchgeführt, andererseits wurden Softwarekomponenten manuell geschrieben. Bei der modellbasierten Lösung wurde die Codegenerierung der einzelnen Softwarekomponenten letztlich mit TargetLink von dSPACE durchgeführt.

Ein entscheidender und aufwendiger Schritt ist die Konfiguration und Kompilierung des

AUTOSAR konformen Betriebssystems und die Generierung der RTE. Hierfür wurde wiederrum SystemDesk eingesetzt.

Im Anschluss kann das Verhalten der kompletten Steuergeräte-Software direkt mit VEOS, der dSPACE Plattform für Offline-Simulation, simuliert und visualisiert werden.

Die Abbildung 4.2 zeigt eine Übersicht der eingesetzten Tools in einer schematischen Darstellung. Der Datenfluss zwischen den Ebenen und Tools ist durch die jeweiligen Pfeile skizziert. Es werden Dateien erzeugt, die von den nachfolgenden Werkzeugen verarbeitet werden. Die Pfeile sind mit dem Format der übergebenen Dateien gekennzeichnet.

An dieser Stelle soll noch einmal hervorgehoben werden, dass die Toolchain auch durch Tools anderer Hersteller, wie zum Beispiel Vector oder Artop, unterstützt wird. Die Definition der Schnittstellen durch den AUTOSAR-Standard macht dies möglich.

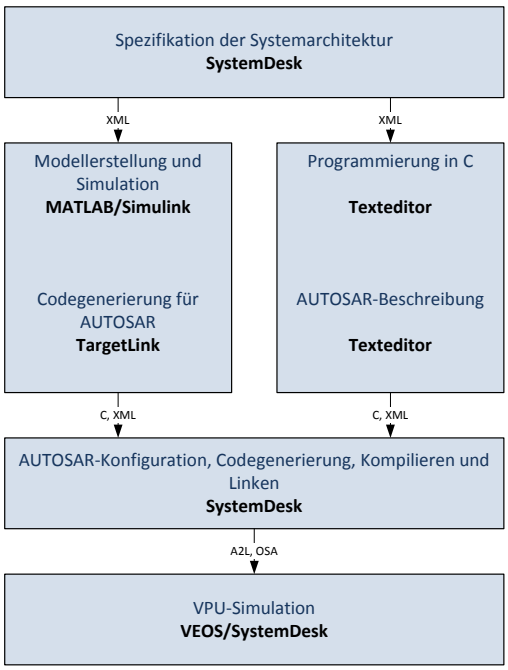


Abbildung 4.2: Toolkette

4.1 Architekturentwurf

Im Folgenden werden Schritte beschrieben, die für den Entwurf einer geeigneten Software- und Systemarchitektur notwendig sind. Ebenfalls wird auf eingesetzte Tools näher eingegangen.

4.1.1 Architekturspezifikation mit SystemDesk

Das Tool SystemDesk ist in die AUTOSAR Tool-Chain intergiert. Wie in Abbildung 4.3 zu sehen ist, interagiert SystemDesk nach oben mit den Softwareentwicklungstools und nach unten mit den ECU spezifischen Tools. Hierbei werden auch die Dateiformate und deren Abhängigkeiten dargestellt. Alle Beschreibungen sind AUTOSAR-konform.

Softwareentwicklungstool

Der obere Teil der Abbildung 4.3 stellt die Entwicklung von wiederverwendbaren Softwarekomponenten dar. Diese können mit Behavior Modeling Tools wie Simulink/TargetLink erzeugt oder manuell geschrieben werden. Jede Softwarekomponente besteht aus einer Beschreibungsdatei (SWC.xml) und dem entsprechenden C-Code (SWC.c).

SystemDesk

Die Softwarekomponentenbeschreibungen werden von SystemDesk erzeugt und können auch wieder eingelesen werden.

Die Softwarearchitektur des Systems wird vollständig in SystemDesk erstellt. Hierzu gehören folgende Aufgaben:

- Verteilung und Verbindung der SW-C,
- Angaben zur Hardwaretopologie,
- Behandlung der Netzwerkkommunikation,
- Mapping der SW-Cs auf die ECUs,
- Zuordnung der Datenelemente auf Systemsignale.

Als Ergebnis wird eine Beschreibung des gesamten Systems in Form von einer AUTOSAR Systembeschreibungsdatei (AUTOSAR System.xml) generiert.

ECU Tool

Jede ECU muss separat konfiguriert werden. Hierfür werden die Systembeschreibungen gelesen und die Aspekte extrahiert, die für die betroffene ECU relevant sind. Aus den ECU Parameter Konfigurationsdateien (ECU-C.xml) werden von den ECU Tools die enthaltenen Konfigurationsdaten für alle grundlegenden Softwarekomponenten von einer ECU gelesen und geschrieben, einschließlich der RTE-Konfiguration. Nach erfolgreicher ECU Konfiguration und Validierung, wird daraus C-Code für Basis-Software-Komponenten und der RTE

(AUTOSAR OS.c, AUTOSAR COM.c, B-SWC.c) generiert. SystemDesk deckt einige Features von den ECU Tools wie die Konfiguration des OS, COM und RTE und der Erzeugung des RTE ab, jedoch kann SystemDesk nicht die Basissoftware erstellen und konfigurieren. An dieser Stelle arbeitet die Firma dSPACE mit der Firma ElektroBit zusammen. Mit dem Tool SystemDesk und VEOS ist es möglich, ein Steuergerät zu simulieren. Hierfür kommt dann eine dSPACE spezifische Basissoftware zum Einsatz. Der Ablauf der Simulation wird im Abschnitt 4.2.4 näher beschrieben. (vgl. [8])

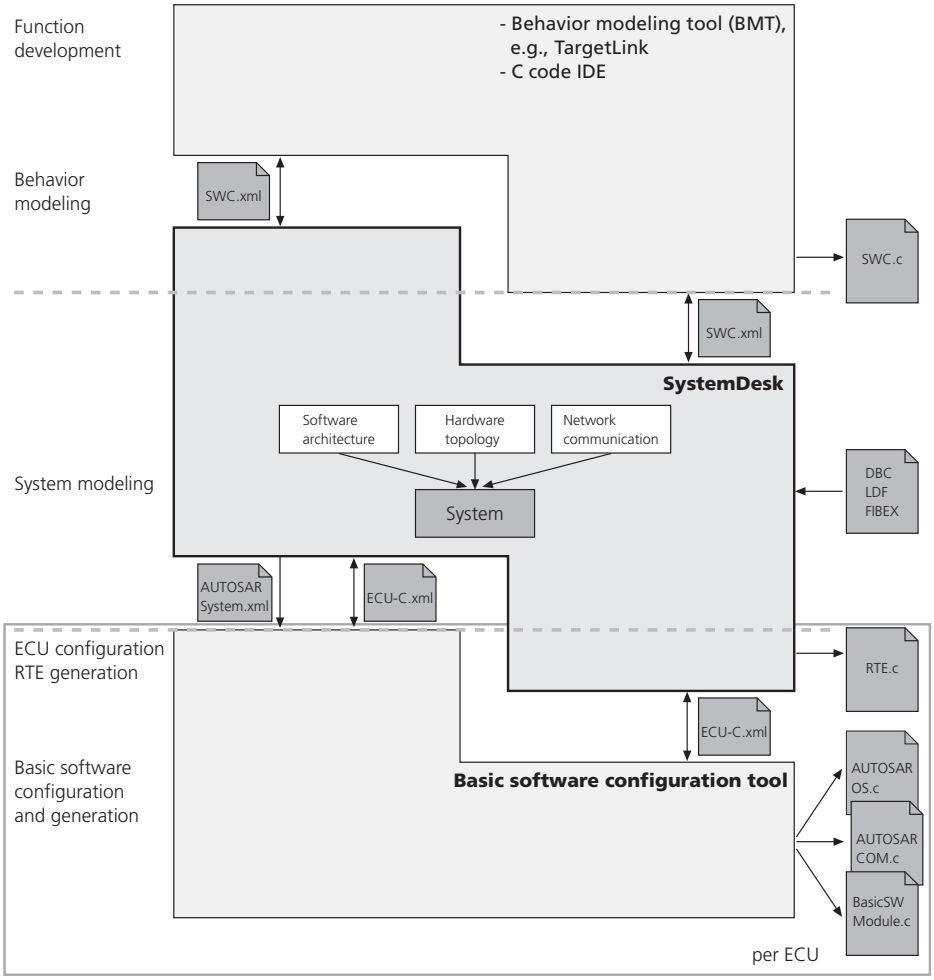


Abbildung 4.3: Tool chain - SystemDesk [8]

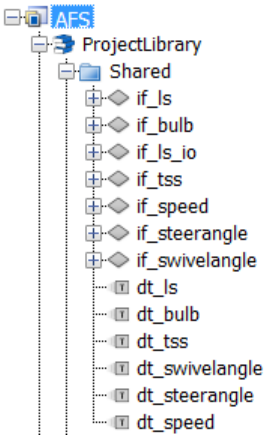


Abbildung 4.4: Interfaces und Typen - ProjectLibrary in SystemDesk

4.1.2 Modellierung der Softwarearchitektur

Vor der Modellierung des Systems können zunächst in der ProjectLibrary Typen für den Entwurf definiert werden, die später mehrfach bzw. wiederverwendet werden sollen. Hierzu zählen Komponenten, Kompositionen und vor allem Standardelemente, die häufig an verschiedenen Stellen verwendet werden. Abbildung 4.4 zeigt die Definition der Standarddatentypen und Interfaces-Typen. Diese werden später für die Spezifikation zur Typisierung von Datenelementen von Schnittstellenspezifikationen bzw. zur Typisierung von Kommunikationsports verwendet. Der Entwurf der Softwarearchitektur erfolgt durch Funktionsblöcke. Die Bestandteile der Funktionsblöcke sind entweder atomare Komponenten oder Kompositionen und werden durch Portdefinition von außen nutzbar gemacht. Die Funktionalität einer Komponente wird durch die Typisierung dieser Ports über Interfaces vorgenommen.

Der Entwurf der Softwarearchitektur beginnt mit der Definition einer Toplevel-Komposition, die alle weiteren Teilsysteme als Unterkomponenten enthält. Die in Abbildung 4.5 dargestellte Toplevel-Komposition der Fallstudie enthält die drei Untersysteme Light, StaticCorneringLights und DynamicCorneringLights. Zudem befinden sich auf dieser Ebene Sensorik-Elemente des Systems, deren Sensordaten in verschiedene Untersysteme weitergegeben werden, wie beispielsweise Geschwindigkeitsmesser (SpeedSensor). Die Aktoren empfangen die in den Untersystemen verarbeiteten Daten, wie beispielsweise das Signal für das statische Kurvenlicht (staticlight).

Ist der Entwurf abgeschlossen, ermöglicht SystemDesk auch die Spezifikation der internen Details der Komponenten, dem Verhalten (InternalBehavior). Dazu gehören unter anderem die Definition von einem oder mehreren Prozessen zur Umsetzung der Funktionalität der Komponente. Diese Runnables können entweder ereignisgesteuert oder zyklisch ausgelöst werden. Darüber hinaus können weitere interne Details wie z.B. Interprozess-Variablen

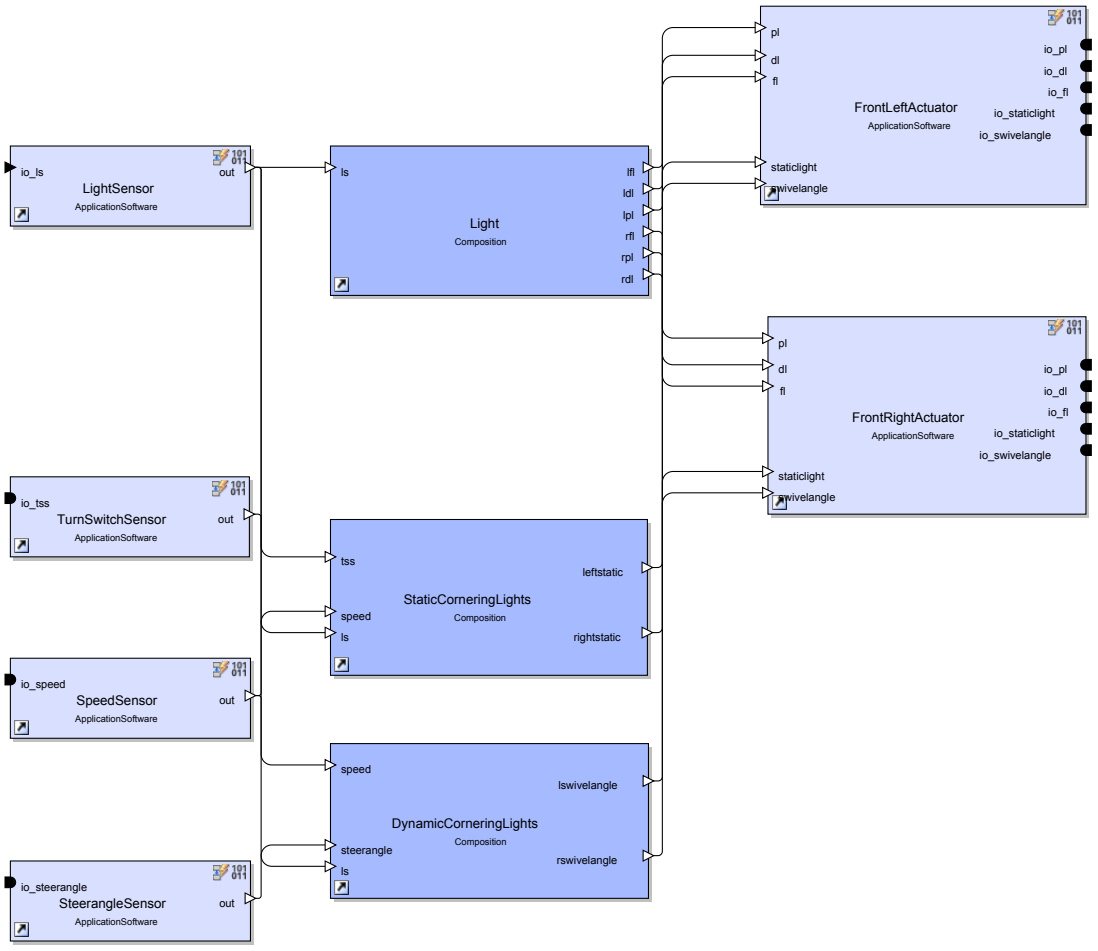


Abbildung 4.5: Toplevel-Komposition

definiert werden. Die eigentliche Umsetzung der Runnables, entweder durch entsprechende Verhaltensmodelle und anschließender Code-Generierung oder direkt in Code, erfolgt dann in der Phase des technischen Designs bzw. der Implementierung (siehe Abschnitt 4.2.1).

4.1.3 Modellierung der Hardware-Topologie

Die Modellierung der Hardware-Topologie erfolgt in SystemDesk tabellarisch durch Definition der einzelnen, im System verbauten, ECUs und deren Anbindungen an ein Bussystem. Abbildung 4.6 gibt einen Überblick über die Topologie-Spezifikation für das System aus der Fallstudie. Es wurden vier ECUs vorgesehen. Die Funktionen des Statischen und Dynamischen Lichtes werden in der CorneringLightECU realisiert und die des Frontlichtes in der FrontlightECU. Der linke bzw. rechte Frontscheinwerfer wird über die FrontLeftECU bzw.

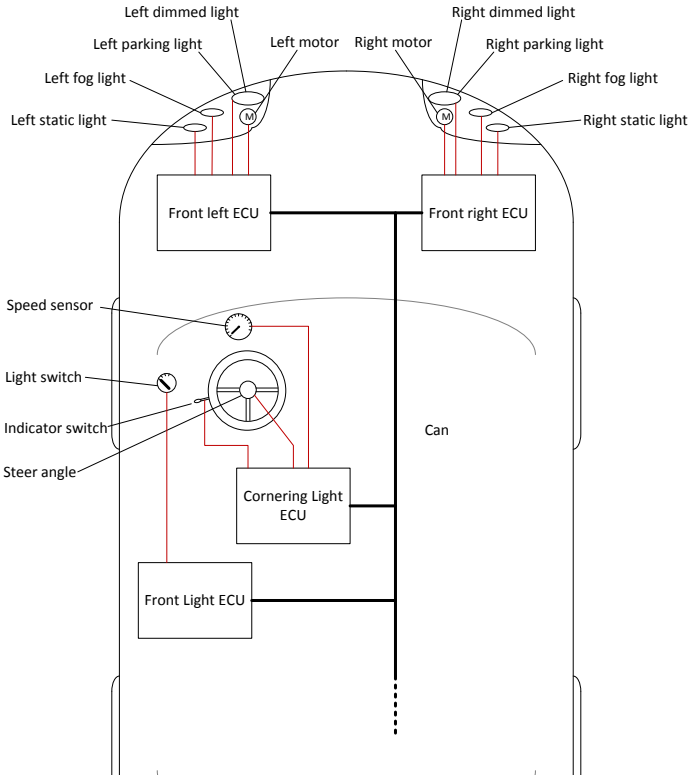


Abbildung 4.6: Topologie

FrontRightECU gesteuert. Die Integration der Steuergeräte erfolgt über einen gemeinsamen CAN-Bus.

4.1.4 Integration der Systemarchitektur

Die Verteilung der Komponenten des Systems sowie die zugehörigen Sensor/Aktor-Komponenten auf die entsprechenden Steuergeräte erfolgt durch Drag & Drop im Projektbaum. Das Ergebnis des Mappings ist in Abbildung 4.7 zu sehen. Die Definition der Kommunikationsmatrix erfolgt durch den Import einer Data Base CAN-Datei (.dbc). Diese wurde mit Hilfe von dem Tool CANdb++ von der Firma Vector erstellt. Nun erfolgt die Definition von Bussignalen zwischen den Netzwerkknoten zur Umsetzung des Austauschs von Datenelementen über die Schnittstellen von zwei über einen Konnektor miteinander kommunizierenden, verteilten Softwarekomponenten. Abbildung 4.8 zeigt exemplarisch die Signaldefinition für die CorneringLightECU. Ein Ausschnitt aus dem resultierenden Projektbaum für die vollständige Spezifikation der Systemarchitektur in SystemDesk ist in Abbildung 4.9 zu sehen. Für die nachfolgenden Phasen der AUTOSAR-Methodik, also der Verhaltensmodel-

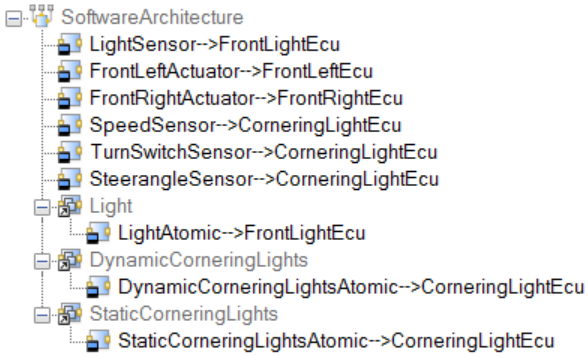


Abbildung 4.7: Software Component Mapping

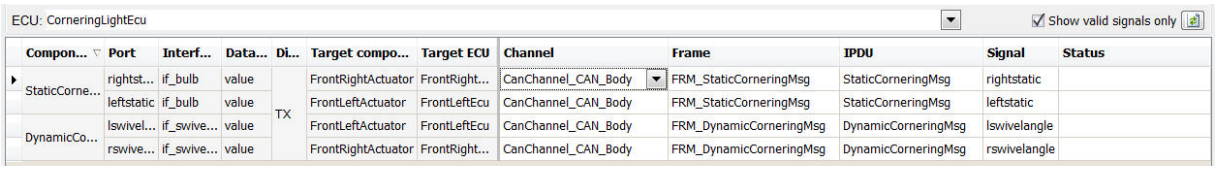


Abbildung 4.8: Signal Mapping

lierung der Softwarekomponenten sowie der System-Konfiguration, ermöglicht SystemDesk den Export der Architektur-Spezifikation im, durch den Standard vorgegebenen, AUTOSAR XML-Format.

4.2 Design und Implementierung

Ausgehend vom vorhergehend erstellten Architekturentwurf wurden in dieser Phase des Wasserfallmodells die Funktionen für die vier Steuergeräte in Programmcode umgesetzt. Die Funktionen für die FrontLightECU wurde mit dem Softwarepaket MATLAB/Simulink modelliert und mit automatischer Codegenerierung in C-Code überführt. Die Funktionen der restlichen Steuergeräte wurden direkt in C umgesetzt. Die Details hierzu sowie die Konfiguration werden in den folgenden Abschnitten beschrieben.

4.2.1 Implementierung mit C

AUTOSAR-Softwarekomponenten können grundsätzlich in der Programmiersprache C erstellt werden. Bei der Entwicklung von Steuergeräte-Software kann die klassische Programmierung in C gegenüber der modellbasierten Entwicklung Vorteile bieten oder unumgänglich sein. Bei der Anpassung von bereits entwickelten Lösungen in C auf AUTOSAR kann

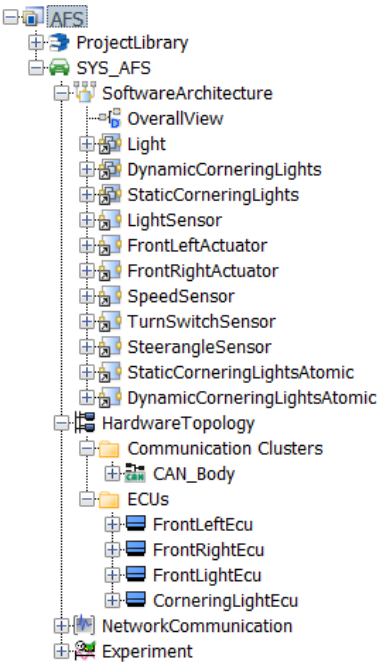


Abbildung 4.9: Projektbaum in SystemDesk

die Wiederverwendung der Module die Entwicklungszeit minimieren und das Auftreten von Fehlern reduzieren. Aus den MATLAB/Simulink-Modellen erzeugt TargetLink Quellcode in C. Im Folgenden wird beschrieben, welche grundlegenden Regeln für die Erstellung von AUTOSAR-Softwarekomponenten in C gelten. Eine umfassende Dokumentation ist unter [3] zu finden.

Die Methodologie von AUTOSAR sieht die Teilung in atomare Softwarekomponenten vor, die auch bei der Entwicklung in C gilt. Die Softwarekomponenten sind aus Funktionseinheiten bzw. Prozessen zusammengesetzt, den sogenannten Runnables. AUTOSAR OS wurde für single-threaded Hardware entwickelt [10]. Ein Runnable kann durch Events, wie zum Beispiel das Eintreffen einer Nachricht, oder durch periodische Ereignisse, wie Timing Events, von dem AUTOSAR OS gestartet werden. Die Konfiguration der Events erfolgt nicht im C-Code, sondern in der XML-Beschreibung der Softwarekomponenten (siehe Abschnitt 4.2.3). Im Folgenden wird auf ausgewählte Implementierungsdetails näher eingegangen.

Kommunikation

Die Kommunikation, der Austausch von Nachrichten, zwischen den Softwarekomponenten wird über die RTE realisiert. Im Rahmen des AUTOSAR-Zieles, dass Softwarekomponenten verschiebbar sein sollen, gibt es keinen Unterschied, ob sich die Komponenten auf dem

gleichen oder unterschiedlichen Steuergeräten befinden. Sind die Komponenten auf unterschiedlichen ECUs angeordnet, wird die Kommunikation über ein Netzwerk realisiert, was ein anderes Zeitverhalten zur Folge hat. Bei der AUTOSAR Version 3.2 wird das unterschiedliche Zeitverhalten im Entwurf nicht berücksichtigt.

Bei AUTOSAR gibt es, wie bereits in Abschnitt 2.1 dargestellt, zwei Arten von Kommunikationmechanismen zwischen den Softwarekomponenten:

- Sender/Receiver-Kommunikation oder
- Client/Server-Kommunikation.

Auf die Eigenschaften wird nun detailliert eingegangen.

Sender/Receiver-Kommunikation

Bei dieser Kommunikation können ein oder mehrere Datenelemente gesendet werden. Die Elemente können primitive und komplexe Datentypen (Strukturen) besitzen. Die Übertragung der Datenelemente erfolgt unabhängig voneinander, dabei hat jedes Datenelement einen eigenen Kanal. Sollen alle Elemente, die zu einem Port gehören, gleichzeitig übertragen werden, sind diese in einer Struktur zu kapseln.

Die Sender/Receiver-Kommunikation erfolgt asynchron. Dabei wird keine Antwort vom Receiver gesendet. Ist eine Antwort an den Sender nötig, muss eine zweite Verbindung in entgegengesetzter Richtung aufgebaut werden.

Es können verschiedene Beziehungsvarianten aufgebaut werden:

- 1:m (Multicast)
  Ein Sender sendet Daten an mehrere Empfänger.
- n:1
  Mehrere Sender senden Daten an einen Empfänger.

Eine Kombination aus beiden Beziehungen ist bisher nicht vorgesehen.

Die Multicast Beziehung bietet Effizienzvorteile, da die benötigte Bandbreite nicht mit der Anzahl der Empfänger wächst.

Diese Beziehungen werden auf VFB-Ebene modelliert. Außerdem ist es möglich durch unterschiedliche Parameter das Verhalten der Datenübertragung und die Bereitstellung zu beeinflussen. Es gibt die Möglichkeit einer expliziten und der impliziten Variante. (vgl. [6])

Bei der expliziten Variante werden die Daten direkt versendet oder empfangen. Auf der Empfängerseite gibt es eine Queue. Diese Queue kann entweder eine Länge von genau eins oder größer eins besitzen. Der erste Fall bietet sich an, wenn ein Last-the-Best Verhalten gewünscht wird, also nur der letzte Wert von Interesse ist. Der zweite Fall kommt dann zum Einsatz, wenn ein klassisches FIFO-Verhalten benötigt wird. Ein Beispiel hierfür ist die

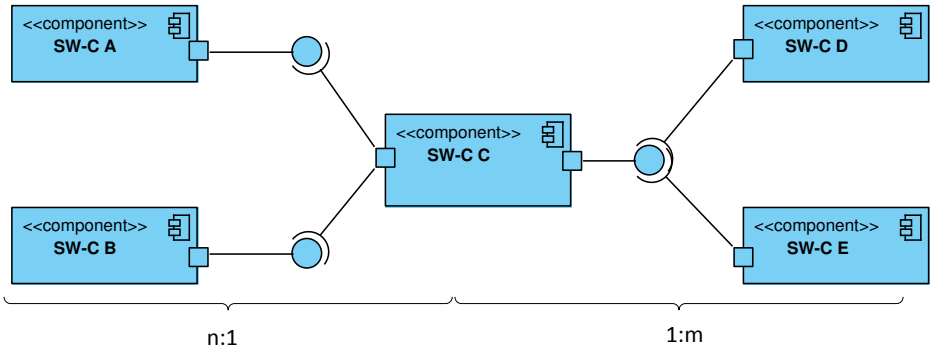


Abbildung 4.10: Beziehung Sender/Receiver

Übermittlung von Events. Aus dem Senden und Empfangen von primitiven beziehungsweise komplexen Datentypen resultieren vier API-Funktionen bei der Generierung der RTE.

Explizites Senden von Daten ($Queue = 1$)

```
Std_ReturnType
    Rte_Write_<p>_<d>(IN Rte_Instance <self>, IN <data>)
```

Explizites Empfangen von Daten ($Queue = 1$)

```
Std_ReturnType
    Rte_Read_<p>_<d>(IN Rte_Instance <self>, OUT <data>)
```

Explizites Senden von Events ($Queue > 1$)

```
Std_ReturnType
    Rte_Send_<p>_<d>(IN Rte_Instance <self>, IN <data>)
```

Explizites Senden von Events ($Queue > 1$)

```
Std_ReturnType
    Rte_Recieve_<p>_<d>(IN Rte_Instance <self>, OUT <data>)
```

`<p>`: Portname über den die Kommunikation stattfindet

`<d>`: Name des Datenelementes das empfangen oder gesendet wird

`IN Rte_Instance <self>`: damit ist es der RTE möglich, die Instance der Softwarekompontene zu identifizieren

`IN <data>`: Datenelement, welches übertragen wird. Primitive Datentypen werden „by Value" und komplexe Datentypen „by Reference" übergeben.

Wie zu sehen ist, wird bei keiner Schnittstelle die Gegenstelle identifiziert. Somit existiert beim Senden offensichtlich kein Empfänger und beim Empfang keine Quelle. Der Port, der

angegeben wird, ist der Port von der Komponente selbst. Diese indirekte Adressierung ist ein wichtiges AUTOSAR-Prinzip. Die Verbindungen werden ausschließlich in der VFB-Ebene modelliert und durch die RTE bereitgestellt. Dies erfolgt im einfachsten Fall durch ein Mapping eines Funktionsaufrufes durch ein Makro. Somit kann die jeweilige Gegenstelle bei der Implementation unbekannt sein und der Code wird unabhängig von der Umgebung. Hierdurch erreicht AUTOSAR das Ziel der Verschiebbarkeit und Austauschbarkeit der Komponenten.

Die implizite Bereitstellung ist eine Besonderheit von AUTOSAR. Es erfolgt kein direkter Aufruf, sondern es wird indirekt mittels RTE durchgeführt. Die RTE übermittelt die Daten erst nach der Terminierung der Sender-Runnable und sind dann vor Beginn der Ausführung der Runnable auf Empfängerseite verfügbar. Die implizite Kommunikation ist zum Beispiel bei der Verhaltensmodellierung in Matlab/Simulink erforderlich, da diese Tools zyklisch arbeiten. Inputdaten werden eingelesen, dann verarbeitet und letztendlich dem nächsten Funktionsblock bereitgestellt.

Beim Senden kann das Runnable beliebig oft die Daten bereitstellen, solange es arbeitet. Erst nach der Abarbeitung der Runnable werden die letzten Daten von der RTE dem Empfänger zur Verfügung gestellt. Dadurch müssen nicht an jedem Terminierungspunkt die Daten explizit übertragen werden.

Beim Empfang werden vor der Ausführung der Runnable die Daten von der RTE bereitgestellt. Die Daten sind während der gesamten Ausführungszeit verfügbar und ändern sich auch nicht in diesem Zeitraum. Dadurch muss keine Kopie angelegt werden.

Für das implizite Senden und Empfangen werden zwei API-Funktionen generiert. (vgl. [11])

Implizites Senden
`void Rte_IWrite_<re>_<p>_<d>(IN Rte_Instance <self>, IN <data>)`
Implizites Empfangen
`void Rte_IRead_<re>_<p>_<d>(IN Rte_Instance <self>, OUT <data>)`

`<re>`: Runnable, das den Aufruf ausführt

Client/Server-Kommunikation

Bei dieser Kommunikation bietet der Server einem oder mehreren Clients Dienste an. Hierbei handelt es sich um eine n:1 Kommunikationbeziehung (siehe Abbildung 4.11).

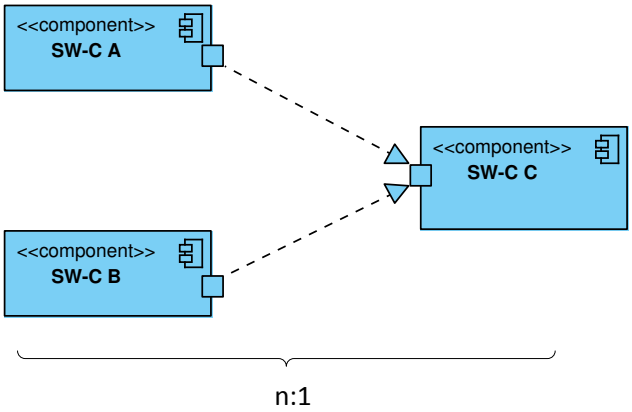


Abbildung 4.11: Beziehung Client/Server

Der Client beginnt die Kommunikation, indem er beim Server eine angebotene Operation ausführt. Über die Parameter der Operation können Daten miteinander ausgetauscht werden. Der Client kann blockierend (synchrone Kommunikation) oder nicht blockierend (asynchrone Kommunikation) sein. Wenn der Client blockiert, wartet dieser auf eine Antwort vom Server.[4]

Es wird zu jedem Parameter ein Attribut angegeben, welches die Übertragungsrichtung aus Sicht des Servers bestimmt.

IN: Bei dem Aufruf der Operation wird der Parameterwert vom Client zum Server übermittelt.

OUT: Nach Auschluss der Operation wird der Parameterwert als Ergebnis zurück zum Client übermittelt.

IN/OUT: Kombination aus IN und OUT

Bei der Übertragung werden die Daten entweder by-value oder by-reference durch die RTE dem Empfänger bereitgestellt. Im Falle eines IN-Parameters und primitiven Datentyps kommt by-value zum Einsatz. In alle anderen Fällen werden die Daten durch eine Referenz bereitgestellt.

Bei der Generierung der RTE wird dann aus einer Client/Server-Kommunikation diese Operation entstehen.

```
Std_ReturnType Rte_Call_<p>_<o>( IN Rte_Instance <self>,
                                 [IN|IN/OUT|OUT] <data_1>,
                                 [IN|IN/OUT|OUT] <data_n>)
```

<p>: Portname über den die Kommunikation stattfindet

<o>: Name der Operation, die auszuführen ist

<data_1>...<data_n>: Parameter, die bei der Kommunikation ausgetauscht werden

Auf der Seite des Servers wird eine Runnable implementiert, die den Server-Stub realisiert.

Aufruf auf Clientseite

```
Std_ReturnType <name> ( IN Rte_Instance <self>,
                        [IN|IN/OUT|OUT] <data_1>,
                        [IN|IN/OUT|OUT] <data_n>)
```

<name>: frei wählbarer Funktionsname

Runnables

Die RTE ist für die Auslösung und die Kommunikation der Runnables verantwortlich. Es wird zwischen den Kategorien 1a, 1b und 2 unterschieden. Runnables der Kategorie 1a und 1b enden nach dem Aufruf und haben somit eine endliche Ausführungsdauer. Außerdem besitzen diese implizite Lese- und Schreibzugriffe. Die Kategorie 1b verfügt zusätzlich über explizite Zugriffe.

Bei der Kategorie 2 gibt es die Möglichkeit, Wartepunkte einzuführen, um beispielsweise auf weitere Triggerereignisse zu warten. Es kann in der Regel keine feste Ausführungszeit bestimmt werden. (vgl. [3])

In C wird eine Runnable als eine Funktion ohne Rückgabewert und Argumente definiert. Der Name kann hierbei beliebig gewählt werden, z.B. `void ExampleRunnable(){...}` . Der Name wird in die XML-Beschreibung eingetragen und bei der RTE-Konfiguration einem Task zugewiesen.

4.2.2 Implementierung mit MATLAB/Simulink

Das Software-Paket mit den Komponenten MATLAB/Simulink und der Erweiterung TargetLink zur Codegenerierung für AUTOSAR dient zur modellbasierten Entwicklung von Steuergeräte-Software. Die Software MATLAB wird von der Firma The MathWorks entwickelt und vertrieben. Die Stärken von MATLAB liegen in der numerischen Berechnung von Matrizen, sowie der Datenanalyse und Visualisierung. MATLAB dient als Basiskomponente für das Zusatzpaket Simulink, das die Modellierung und Simulation von Systemen erlaubt. Simulink bietet eine grafische Bedienungsoberfläche und stellt für verschiedene Anwendungsgebiete Bibliotheken bereit, die sich individuell erweitern lassen. Zu der Software TargetLink liefert die Firma dSPACE einen solchen Bibliotheken-Block, wie in Abbildung 4.12 zu sehen ist, mit. Ein Vorteil der modellbasierten Entwicklung ist die Hardware-Unabhängigkeit,

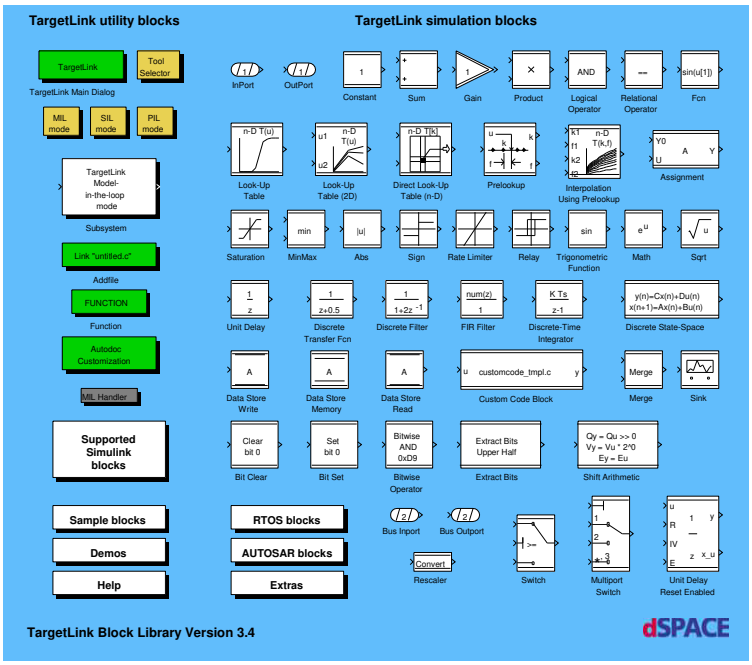


Abbildung 4.12: TargetLink Block Library

sowie die daraus resultierende Wiederverwendbarkeit der Modelle. Darüber hinaus erzeugt TargetLink aus Simulinkmodellen automatisch C-Code für AUTOSAR-Systeme.

Bei der modellbasierten Software-Entwicklung mit Simulink gibt es eine Vielzahl von Anwendungsgebieten. Neben der Funktionsmodellierung können auch mechanische Systeme entworfen werden. Daraus ergeben sich verschiedene Simulationsmöglichkeiten, durch die es möglich ist, die zu entwickelnden Modelle bereits vor der Verfügbarkeit der Hardware in der Simulation zu erproben und anzupassen. Die modellbasierte Entwicklung von Steuergerätfunktionalitäten findet bereits seit einigen Jahren bei den Automobilherstellern statt. Simulink unterstützt von vorherein die hierarchische Strukturierung des gesamten Systems und macht komplexe Projekte beherrschbar. Die Lesbarkeit und Verständlichkeit von Projekten wird durch diese hierarchisch geordneten Modelle gefördert.

Bevor mit der Umsetzung der FrontLightECU begonnen wurde, war die Konfiguration des Modells im so genannten Data Dictionary notwendig. Hierbei handelt es sich um einen zentralen Datencontainer, in dem z.B. Parameter- und Variablendefinitionen für das Modell und die Code-Generierung abgelegt werden. In diesem Data Dictionary wurden zum einen die verwendeten Softwarekomponenten, zum anderen die Interfaces, die für den Austausch von Daten zwischen Softwarekomponenten und der Außenwelt notwendig sind, sowie die Runnables spezifiziert.

Für eine AUTOSAR-konforme Modellimplementierung in MATLAB/Simulink ist es notwendig zwei Blöcke, TargetLink Main Dialog und das Subsystem TargetLink Model-in-the-

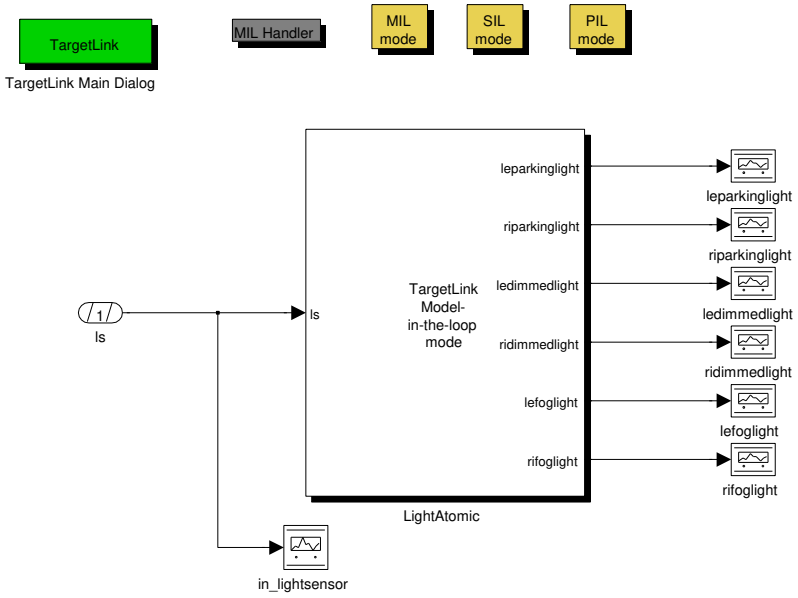


Abbildung 4.13: TargetLink

loop mode, aus der TargetLink-Bibliothek einzusetzen. In Abbildung 4.13 ist für das Steuergerät FrontLightECU eine entsprechende Implementierung zu sehen. Nach dem Abschluss der Spezifikation im Data Dictionary und der Erstellung aller Modelle kann über den Main Dialog die Generierung der C- und XML-Dateien gestartet werden. Hierbei beschreiben die C-Dateien den funktionalen Aufbau und die XML-Dateien den strukturellen Aufbau mit den äußeren Schnittstellen. In dem Subsystem LightAtomic befindet sich die eigentliche Softwarekomponente. Die oberste Modellebene ist in Abbildung 4.14 dargestellt. Auf der Abbildung sind die jeweiligen Ein- und Ausgänge, sowie zwei Subsysteme zu erkennen. Bei den Subsystemen LsPreprocessing und Trigger handelt es sich um Systeme, die den sogenannten Runnable-Block beinhalten. Auf diese Weise ist es TargetLink möglich den Code für die entsprechenden Runnables zu generieren.

4.2.3 Konfiguration

Die Konfiguration des Steuergerätes enthält alle Informationen, die für die Generierung der RTE und der Basissoftware notwendig sind. Damit ein entsprechender RTE-Code für das Steuergerät erzeugt werden kann, ist die Konfiguration folgender Elemente notwendig:

- Betriebssystem(OS)
- Kommunikation(COM)

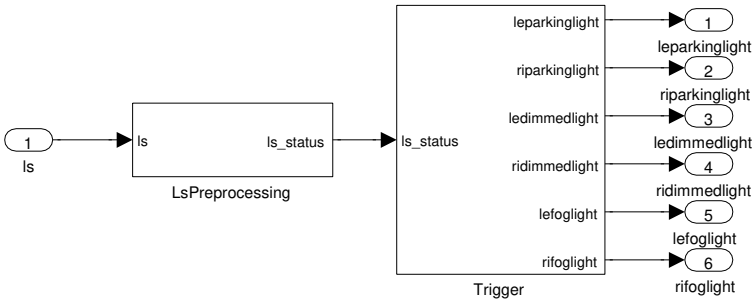


Abbildung 4.14: Umsetzung der FrontLightECU

- RTE
- sowie alle Module der Basissoftware die ein AUTOSAR-Interface besitzen, beispielsweise die I/O Hardwareabstraktionskomponente

Ein Basissoftwaremodul besteht aus einer Modulkonfiguration, die alle ECU Parameter enthält, und der Basissoftwarekomponente, die der RTE Codegenerator benötigt um passenden Code zu generieren. Hinzu kommen die Code-Files, die mit Hilfe der Modulkonfiguration und dem Basissoftwaremodul-Codegenerator erzeugt werden.

Die Abbildung 4.15 zeigt wie die Konfigurationsdateien und die Basissoftwaremodule (BSW) generiert werden. Bei der V-ECU Implementation handelt es sich um den plattformunabhängigen Teil einer virtuellen ECU, die bei der Simulation zu Einsatz kommt. Die V-ECU sowie der RTE Code werden von SystemDesk vollständig generiert. Außerdem ist es möglich die Konfigurationen in einem standardisierten AUTOSAR XML-Format zu exportieren und über Third-Party Tools passenden Code für die Basissoftwaremodule zu generieren (vgl. [8]).

4.2.4 Simulation

Nach Abschluss der Konfiguration folgt die Simulation des Systems auf dem Host-PC. SystemDesk realisiert die Simulation in Form eines Experimentes. Dieses enthält alle relevanten Elemente, wie zum Beispiel die Simulationskonfiguration und die Simulationsergebnisse (siehe Abbildung 4.16). Das zentrale Element ist das SimulationsSystem, das für die Generierung einer „Offline Simulation Application" (OSA file) benötigt wird. Die „Offline Simulation Application" wird bei einem Ablauf einer Simulation mit dem Tool VEOS ausgeführt. Bei der Generierung werden alle Codefiles der Softwarekomponenten und der RTE von dem C-Compiler kompiliert.

SystemDesk und VEOS verwenden für die Offline-Simulation virtueller Steuergeräte ein emuliertes AUTOSAR Betriebssystem. Hauptaufgabe des Betriebssystems ist es, die kon-

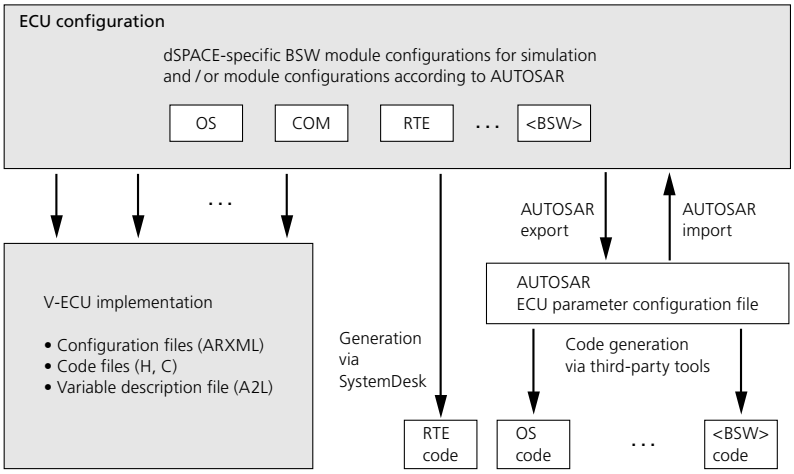


Abbildung 4.15: ECU Konfiguration[8]

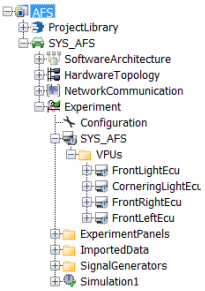


Abbildung 4.16: SystemDesk Projektbaum - Experiment

figurierten OS Tasks korrekt aufzurufen. Die Offline-Simulation besitzt eine eigene Uhr, die unabhängig von der Echtzeit ist. Dadurch ist es möglich die Simulation anzuhalten und zu starten, jedoch kann kein Echtzeitverhalten erreicht werden.

SystemDesk bietet für die Simulation der Eingangssignale einen Stimulus Generator an. Hiermit lassen sich unterschiedliche Signale, wie Sinus, Konstanten oder aufgezeichnete Werte, an den Ports erzeugen. Die Ergebnisse einer Simulation werden in dem Abschnitt 4.3 aufgezeigt.

4.3 Testen

In diesem Abschnitt folgt die letzte Phase des Wasserfallmodells - das Testen. Im Rahmen der Fallstudie wurde nach der erfolgreichen Implementierung der Funktionalitäten eine Überprüfung der Anforderungen mit Hilfe der Simulation durchgeführt.

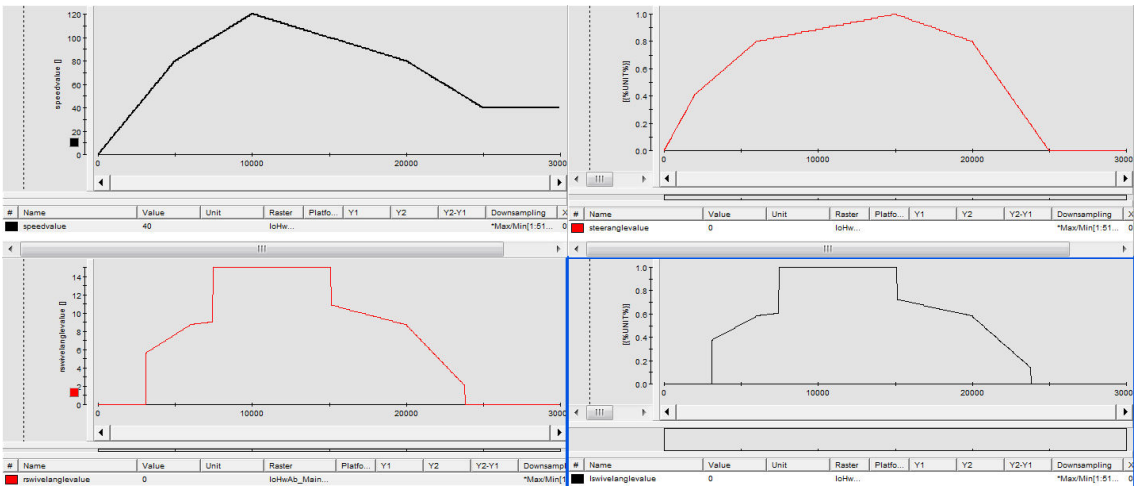


Abbildung 4.17: SystemDesk VEOS - Simulationslauf

Durch die Simulation lässt sich ein funktionsorientierter Test durchführen. Im Vorfeld wurden Testfälle anhand der Spezifikation beschrieben und sollen mögliche Fehler aufdecken. Der Stimulus Generator wurde für die entsprechenden Testfälle jeweils statisch angepasst. Eine Dokumentation der durchgeführten Testfälle ist in Anhang B vorzufinden. Des Weiteren stellt die Abbildung 4.17 exemplarisch einen Simulationslauf dar. Hierbei handelt es um einen Test des dynamischen Kurvenlichtes. Die beiden oberen Graphen zeigen die Eingänge Geschwindigkeit (links) und Lenkwinkel (rechts). Die Ausgänge des Schwenkwinkels (links und rechts) sind auf den beiden unteren Graphen zuerkennen. Der Test zeigt eine Kurvenfahrt nach rechts und in der findet eine Beschleunigung statt. Dieser Simulationslauf hat eine Dauer von 30 Sekunden wie auf der x-Achse zu beobachten ist. Auf der y-Achse sind die jeweiligen Werte der Ein- bzw. Ausgänge abgebildet. Wie erkennbar ist, wird das Kurvenlicht ab einer Geschwindigkeit größer 50 $\frac{Km}{h}$ aktiv. Der sprunghafte Anstieg des Schwenkwinkels bei 7 Sekunden ist auf den Faktor x in der Berechnung zurückzuführen.

Kapitel 5

Zusammenfassung

Es wurde eine Fallstudie eines modellbasierten Entwicklungsprozesses von der Anforderungsbeschreibung bis hin zur Implementierung, Integration und dem Test von Steuergeräten durchgeführt. Mit der Vorstellung der Grundlagen des AUTOSAR-Standards wurde begonnen. Ein weiterer Teil bildet die Analyse des vereinfachten Adaptive Frontlighting Systems. Außerdem wurden die einzelnen Entwicklungsschritte zusammen mit den verwendeten Tools (SystemDesk, Matlab/Simulink, TargetLink, VEOS) beschrieben und ausgewählte Implementierungsdetails diskutiert. Abschließend wurde die Simulation eines Steuergerätes beschrieben sowie Möglichkeiten des Testens entwickelter Softwarekomponenten aufgezeigt.

Ausblick

Dem AUTOSAR-Standard kommt und wird in Zukunft eine große Bedeutung bei der Entwicklung von Steuergeräte-Software zugeschrieben. Bei BMW beispielsweise wird AUTOSAR seit dem Jahr 2008 für die Entwicklung einiger Steuergeräte in der 7er-Serie eingesetzt[15]. Ein weiterer Aspekt ist, dass beteiligte Partner, wie zum Beispiel BMW, Daimler und VW, die Entwicklung vorantreiben (10 Releases seit 2005, aktuell Version 4.1) und AUTOSAR in ihren Fahrzeugen einsetzen. Ein Beweis hierfür liefert die Abbildung 5.1, die den rapiden Wachstum der Steuergeräte mit AUTOSAR in den nächsten Jahren dargestellt. Diese Zahlen basieren auf den AUTOSAR OEM Core Partnern [9]. Zusätzlich spricht die Aufnahme neuer Techniken im Automobilbereich wie zum Beispiel Kommunikationsprotokolle wie XCP und TCP/IP (Ethernet) dafür. Auf dem Gebiet der Automobilsoftware zählt die AUTOSAR Spezifikation weltweit zu den besten. Dieser Erfolg ist allen Partnern, trotz der Konkurrenz untereinander, zuzuschreiben [18].

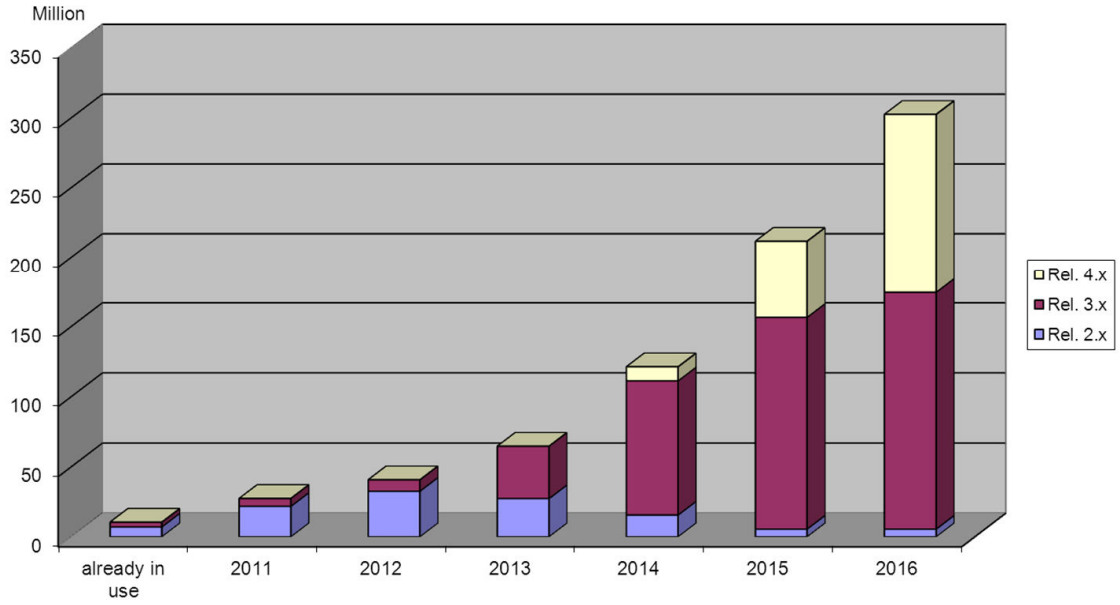


Abbildung 5.1: Wachstum der Steuergerät mit AUTOSAR [9]

Anhang A

Inhalt der CD-Rom

- Bachelorarbeit “Simulationsbasierte Entwicklung eines AUTOSAR-konformen Steuergerätes“ (PDF)
- Projektdateien der Fallstudie
- Dokumentierte Testbilder
- Alle in dieser Bachelorarbeit verwendeten Grafikdateien in Originalqualität

Anhang B

Testprotokoll

Softwareversion: 1.0

Datum: 17.07.2013

Testperson: Jens Torfstecher

Test: Lichtschalterfunktionalität

Nr.	Testfall	Erwartetes Ergebnis	Ergebnis eingetroffen
01	Initialisierung	Licht ist deaktiviert	ja
02	Alle Schalterstellungen 0->3 3->0	0=AUS; 1=Standlicht; 2=Abblendlicht; 3=Nebelscheinwerfer entsprechende Lichter werden aktiviert bzw. deaktiviert	ja

Test: Statisches Kurvenlicht

Nr.	Testfall	Erwartetes Ergebnis	Ergebnis eingetroffen
03	Initialisierung	Beide Statischen Kurvenlichter sind deaktiviert	ja
04	Kreuzung Rechtskurve	Signaländerung des Blinkers, aktiviert rechtes statische Kurvenlicht	ja
05	Kreuzung Linkskurve	Signaländerung des Blinkers, aktiviert linkes statische Kurvenlicht	ja
06	Überschreitung der Geschwindigkeitsgrenze	Bei 41 km/h wird das statische Kurvenlicht deaktiviert	ja

Test: Dynamisches Kurvenlicht

Nr.	Testfall	Erwartetes Ergebnis	Ergebnis eingetroffen
07	Initialisierung	Beide dynamischen Kurvenlichter sind Nullposition	ja
08	Überschreitung der Geschwindigkeit von 50 km/h	Dynamisches Kurvenlicht wird aktiv	ja
09	Rechtskurve	Der Winkel des Kurvenlicht entspricht den Berechnungen (siehe Testbild 3)	ja
10	Rechtskurve mit geringerer Geschwindigkeit	Der Winkel des Kurvenlicht entspricht den Berechnungen (siehe Testbild 4)	ja

Literaturverzeichnis

[1] Autosar: Methodology. http://www.autosar.org/download/AUTOSAR_Methodology.pdf; Version 1.2.1 Abruf: 31/07/2013.

[2] Autosar: Motivation and goals. http://www.autosar.org/index.php?p=1&up=1&uup=2&uuup=0; Abruf: 31/07/2013.

[3] Autosar: Specification of rte. http://www.autosar.org/download/AUTOSAR_SWS_RTE.pdf; Version 1.0.0 Abruf: 31/07/2013.

[4] Autosar: Technical overview. http://www.autosar.org/index.php?p=1&up=2&uup=0; Abruf: 20/09/2013.

[5] Hella kgaa hueck & co. http://www.hella.com; Abruf: 31/07/2013.

[6] Specification of communication. http://www.autosar.org/download/R2.0/AUTOSAR_SWS_COM.pdf; Version 2.0.1 Abruf: 31/07/2013.

[7] A. Heine, R. Nitschke. Ubiquitous computing in fahrzeugen, 2004.

[8] dSPACE GmbH. *SystemDesk 3.x - Guide*, 2012.

[9] F. Kirschke-Biller. Autosar a worldwide standard current developments, rollout and outlook, 09 2011. http://www.autosar.org/download/papersandpresentations/AUTOSAR_a%20worldwide%20standard.pdf; Abruf: 30.08.2013.

[10] F. Kluge, C. Yu, J. Mische, S. Uhrig, T. Ungerer. Implementing autosar scheduling and resource management on an embedded smt processor.

[11] O. Kindel and M. Friedrich. *Softwareentwicklung mit AUTOSAR: Grundlagen, Engineering, Management in der Praxis*. dpunkt-Verlag, 2009.

[12] K. Langwieder and H. Bäumler. *Charakteristik von Nachtunfällen*. Gesamtverband der Deutschen Versicherungswirtschaft e.V., Institut für Fahrzeugsicherheit. Gesamtverb. der Dt. Versicherungswirtschaft e.V., Inst. für Fahrzeugsicherheit, 1997.

[13] M. Wernicke, J. Rein. Einbindung bestehender steuergerätesoftware in die autosar-architektur. *ATZelektronik*, 01, 2007.

[14] R. Neue. Autosar - eine einführung. `http://ess.cs.tu-dortmund.de/Teaching/PGs/autolab/ausarbeitungen/Neue-AUTOSAR-Ausarbeitung.pdf`; Version 1.0 Abruf: 31/07/2013.

[15] PresseBox. Autosar erstmals in serie - im neuen bmw 7er, 11 2008.

[16] K. Reif. *Bosch Grundlagen Fahrzeug- und Motorentechnik: Konventioneller Antrieb, Hybridantriebe, Bremsen, Elektronik*. Bosch Fachinformation Automobil. Vieweg + Teubner, 2011.

[17] S. Alakkat, Snatosh Patel B.J., A.C.Methi. Development and implementation of control algorithm for adaptive front light system of a car, 2008.

[18] U. Seiffert and G. Rainer. *Virtuelle Produktentstehung Für Fahrzeug und Antrieb Im Kfz: Prozesse, Komponenten, Beispiele Aus Der Praxis*. ATZ/MTZ-Fachbuch. Vieweg Verlag, Friedr, & Sohn Verlagsgesellschaft mbH, 2008.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

α	Schwenkwinkel
β	Winkel
δ	Lenkwinkel des Fahrzeuges
A2L	ASAM MCD-2 MC language (Beschreibungsdatei)
AFS	Adaptive Frontlighting System
API	application programming interface
AUTOSAR	AUTomotive Open System ARchitecture
ECU	Electronic control unit
I	Radstand des Fahrzeuges
MCAL	Microcontroller Abstraction Layer
OS	Operating system
OSA	Offline Simulation Application
R	Radius der Kurve
RTE	Run-Time Environment
s	Leuchtweite des Schweinwerfer
SW-C	Software Component
v	Geschwindigkeit des Fahrzeuges
V-ECU	Virtual - electronic control unit
VEOS	Tool für eine PC-basierte Simulationsplattform

VFB Virtual Functional BUS

VPU Virtual Processing Unit

XML Extensible Markup Language

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 10.10.2013 Jens Torfstecher