Masterarbeit

Integration und Evaluierung eines 3-Ebenen Sicherheitskonzepts auf einer Echtzeit Mehrkern-Plattform

Christoph Kreuzberger, BSc.

Institut für Technische Informatik
Technische Universität Graz
Vorstand: Univ.-Prof. Dipl.-Inform. Dr.sc.ETH Kay Römer

Beurteiler:
Dipl.-Ing. Dr.techn. Christian Kreiner

Betreuer:
Dipl.-Ing. Georg Macher
Dipl.-Ing. Dr.techn. Eric Armengaud
Dipl.-Ing. Dr.techn. Jürgen Fabian

Graz, im Mai 2015

EUREKA

Kurzfassung

Sowohl die Leistungs- als auch die Sicherheitsanforderungen sind für eingebettete Systeme in den letzten Jahren stetig gewachsen. Um künftigen Anforderungen gerecht zu werden, setzen Gerätehersteller auch im automotiven Bereich zunehmend Mehrkernarchitekturen mit erweiterten Sicherheitsfunktionen ein. Neben Kostenersparnissen durch Zusammenlegung verschiedener Applikationen in ein Steuergerät, bringt dieser Schritt auch neue Herausforderungen. Dazu zählen die Verwaltung geteilte Ressourcen, die sichere Kommunikation unter den einzelnen Kernen, sowie die Gewährleistung, dass sich Fehler einer Instanz nicht auf andere auswirken („Freedom from Interference“).

Um diese zunehmende Komplexität in den Griff zu bekommen und um den Zulieferern einen gemeinsamen Rahmen bei der Entwicklung zur Verfügung zu stellen, stehen Standards zur Gewährleistung funktionaler Sicherheit sowie Softwarestandards als gemeinsames Werkzeug zur Verfügung. Diese werden zu Beginn der Arbeit beschrieben, um einen Überblick zu den Herausforderungen und Lösungen in der automotiven Produktentwicklung zu geben.

In Anlehnung an jene Standards wird in dieser Arbeit ein Echtzeit-Betriebssystems auf einer Mehrkern Architektur integriert und evaluiert, mit dem Ziel, eine durchgängige Kette an Entwicklungswerkzeugen vom modellbasierten Design bis zum hardwareseitigen Debuggen zur Verfügung zu stellen und die Funktion des Mehrkernbetriebssystems zu zeigen.

Hierfür bietet sich die Implementierung des „Standardisierten E-Gas Überwachungskonzept für Benzin und Diesel Motorsteuerungen“ an. Das Konzept ist von einem Zusammenschluss mehrerer Automobilhersteller erstellt worden und beschreibt die Anforderungen zur Überwachung der Funktionsfähigkeit von „Drive by Wire“ Systemen.

Das Konzept wurde im Zuge dieser Arbeit für Mehrkernarchitekturen erweitert und in Form eines Demonstrators umgesetzt, an welchem die Funktion der drei Ebenen des Konzepts (Funktion, Funktionsüberwachung und Rechnerüberwachung) evaluieren wurde.

Die durch diese Arbeit entstandene Entwicklungs- und Testplattform soll als Basis für künftige Untersuchungen der Betriebssicherheit und des Zeitverhaltens von Mehrkernarchitekturen zur Verfügung zu stehen.

Abstract

Both, performance and safety requirements for embedded systems have increased in recent years. To meet these requirements, automotive suppliers introduced multi-core architectures with advanced safety functions also for embedded systems. In addition to cost savings through consolidation of various applications in a single control unit, this step also brings new challenges. These include the management of shared resources, safe communication between various cores, and ensuring that errors on one core do not propagate to other cores (freedom from interference).

To tackle the issues of increased complexity and provide a common framework to suppliers, standards to ensure functional safety as well as software standards have been developed. These standards are described at the beginning of this work in order to provide an overview of the challenges and solution strategies in the automotive product development.

Referring these standards, a real-time operating system has been integrated on and evaluated with a multi-core microcontroller architecture. The aim was, to provide a seamless toolchain from model-based design to hardware-based debugging and to show the correct function of the multi-core operating system.

Therefore, the "Standardized E-Gas Monitoring Concept for Gasoline and Diesel Engine Control Units" has been chosen. The concept has been developed by a working group of car manufacturers and describes the requirements for monitoring the functionalities of "Drive by Wire" systems.

The concept has been adapted for multi-core architectures and implemented on a Throttle-by-Wire plattform to demonstrate and evaluate the function of the three levels of the concept (function, function monitoring and controller monitoring).

The resulting testing platform serves as a basis for future evaluations of functional safety and the time behavior of multi-core architectures.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Statuary Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, am 27.05.2015

Ort, Datum
Place, Date

Unterschrift
Signature

Danksagung

Diese Diplomarbeit wurde am Institut für Technische Informatik an der Technischen Universität Graz in Zusammenarbeit mit AVL List durchgeführt.

Zunächst möchte ich meinem Betreuer am Institut für Technische Informatik, Georg Macher für die Unterstützung und hilfreichen Bemerkungen während der gesamten Arbeit bedanken. Ebenfalls bedanke ich mich bei Jürgen Fabian vom Institut für Fahrzeugtechnik für seine Unterstützung als Betreuer.

Weiters gilt mein Dank Eric Armengaud für die Betreuung von Seiten der AVL List GmbH, sowie Martin, Oswald und Ismar, welche mir in der Firma immer für Fragen zur Verfügung standen.

Meinem Beurteiler, Christian Kreiner möchte ich zusammen mit meinen Betreuern herzlichst dafür danken, dass sie bis zur letzten Minute auf meine Abgaben gewartet haben um sie dann im Akkord Korrektur zu lesen und zu beurteilen.

Eine Masterarbeit bietet die Chance das theoretisch angeeignete Wissen des Studiums auf praktische Weise anzuwenden und Lösungsansätze umzusetzen. Ich hatte in diesem Jahr die Möglichkeit und vor allem die Zeit um vieles kennenzulernen und auszuprobieren. Dabei wurden Tage und Wochen mit schier unlösbaren Hindernissen verbracht. Danken möchte ich meiner Familie und meinen Freunden, die mich durchs Jahr unterstützt haben und in vielerlei Hinsicht zur Fertigstellung dieser Arbeit beigetragen haben.

Christoph Kreuzberger
Graz, Mai 2015

Danke an

Anna, Carina, Christian, Christina, Eric, Georg, Gertraud, Ismar, Jürgen, Johann, Kristina, Lukas, Marina, Martin, Nicole, Oswald, Philipp, Rudolf, Walter

und allen Freunden im DynamoBauZeichensaal.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Sowohl die Leistungs-, als auch die Sicherheitsanforderungen sind für eingebettete Systeme im Automobilbereich in den letzten Jahren stetig gewachsen. Der elektronische Anteil am Fahrzeug steigt dramatisch, getrieben durch verschiedene Faktoren - darunter Energieeffizienz, Emissionsreduktion, erweiterte Sicherheitsfunktionen, Komfort sowie Fahrvergnügen. Es wird stetig daran gearbeitet, die Funktionalitäten der elektronischen Geräte zu erhöhen, während gleichzeitig Strategien entwickelt werden müssen, um die Fehlertoleranz sowie die Störungssicherheit aller kritischen Systeme zu erhöhen. [49]

Die Automobilindustrie erlebt momentan einen Paradigmenwechsel, in dem die meisten Originalausstatter (Original Equipment Manufacturers (OEMs)) ihre Elektronikarchitektur von einem komplexen Netzwerk einzelner Steuergeräte hin zu einer Backbone-Architektur wechseln, in welchem wenige, dafür leistungsstarke sowie betriebssichere Netzwerke für hochperformante Steuergeräte zur Verfügung stehen. Dabei besteht die Herausforderung darin, eine Kostenreduktion durch Zusammenfügen mehrerer Applikationen in einzelne, starke ECUs zu erreichen. Dafür müssen jedoch neue Architekturen und Strategien implementiert werden, um bei Erhöhung der Performanz gleichzeitig die Betriebssicherheit aller Applikationen auf dem Rechner zu gewährleisten. [49]

2008 besaß ein PKW im Schnitt bis zu 80 dedizierte und vernetzte Steuergeräte. Heutzutage wird die Anzahl auf durchschnittlich über 100 Geräte geschätzt. Während die Steuerung vieler Geräte wie Antriebsstrang, ABS (Anti-Blockier-System), Airbag etc. an und für sich selbstständig arbeitet, versucht man durch den Datenaustausch solcher Systeme einen Informationsgewinn zu erreichen, wiederum zur Erhöhung von Betriebssicherheit sowie Fehlertoleranz beitragen soll. Jedoch entsteht durch die steigende Anzahl an Verbindungen und Schnittstellen zwischen einzelnen Netzwerken ein komplexes Gesamtnetzwerk, welches weitere Entwicklungen erschwert. Auch der Aufwand für Test und Validierung, die Wahrscheinlichkeit für Systemfehler, sowie die Gesamtkosten für Geräte, Verkabelung, Einbau und Wartung steigen durch die Erhöhung der Netzwerkknoten. [49] Abbildung 1.1 zeigt die typische Bordnetzdichte in einem modernen Oberklasse-PKW.

Um diese Komplexität zu verringern, setzt man auf eine höhere Integration in den Sensoren, Aktuatoren sowie Steuergeräten. Zum Einen wird die Intelligenz von Sensoren erhöht, um bereits erste Filterungen, Selbsttests sowie eine digitale Kommunikation im Netzwerk zu ermöglichen. Zum Anderen werden Aktuatoren ebenfalls mit digitalen Schnittstellen und Diagnosemöglichkeiten versehen, um eine Verringerung der Komplexität und Kosten

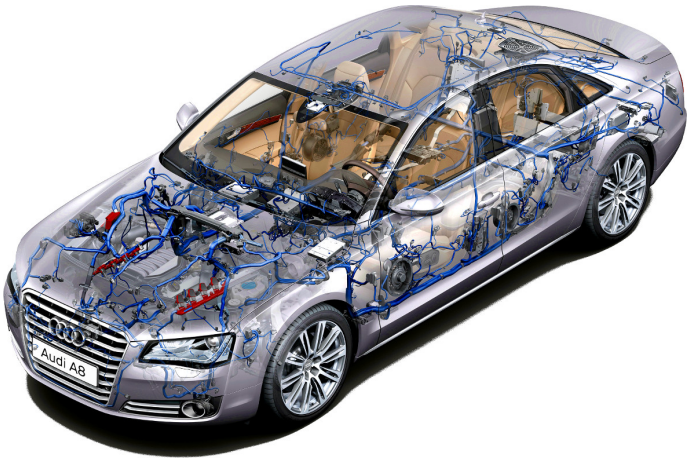


Abbildung 1.1: Bordnetz im modernen Oberklasse-PKW (Quelle: Audi AG)

sowie eine Erhöhung der Betriebssicherheit zu erreichen.

Besonders Steuergeräte sollen in Zukunft leistungsfähiger werden. Einerseits, um mehrere Funktionen zu beinhalten, was zu einer Kostenreduktion an der Hardware führt, und andererseits, um die Anforderungen des stetig steigenden Leistungsbedarfs von Regelalgorithmen zu decken, welche beispielsweise bei der Einhaltung von Abgasnormen bei Motorsteuerungen nötig sind. Hauptaugenmerk bei der Steuergeräteentwicklung liegt durch die höhere Integration an Software darin, Betriebssicherheit, Verfügbarkeit und Zuverlässigkeit für jede einzelne Funktion zu gewährleisten.

Das Hauptproblem bei hochperformanten Rechnern im automotiven Bereich ist jedoch die Wärmeentwicklung. Während im Consumer-Bereich die Taktrate stetig erhöht werden konnte, um einen Performanzgewinn zu erzielen, ist dieser Schritt in Steuergeräten für Automobile nicht beliebig möglich. Ab einer Frequenz von einigen hundert Megahertz (300MHz [12]) reicht die Kühlleistung der geschlossenen, passiv gekühlten Gehäuse nicht mehr aus, um die Verlustwärme der Prozessoren abzuführen. Erschwerend kommt hinzu, dass Bauteile im automotiven Bereich für Umgebungstemperaturen von -40°C bis +125°C (z.T. +150°C) ausgelegt sein müssen. [49]

Um künftigen Anforderungen gerecht zu werden, setzen Gerätehersteller auch im automotiven Bereich zunehmend Mehrkern Architekturen ein, welche bei gleicher Taktfrequenz eine Leistungssteigerung bringen. Neben Kostenersparnissen durch Zusammenlegung verschiedener Funktionen in ein Steuergerät, bringt dieser Schritt jedoch auch neue Herausforderungen. Dazu zählen der Zugriff auf geteilte Ressourcen, die sichere Kommunikation unter den einzelnen Kernen, oder etwa die Gewährleistung, dass Fehler auf einer Recheneinheit andere Einheiten nicht beeinflussen („Freedom from Interference“).

Die Komplexität des Gesamtsystemes verlagert sich durch das Vermindern der Steuergeräte und das Erhöhen der Funktionen pro Steuergerät von den peripheren Netzwerken auf Bordnetzebene weg, hinein in die Softwareebenen der Steuergeräte selbst. Wurde frü-

her eine Unabhängigkeit der einzelnen Applikationen durch eine physikalische Trennung erreicht, muss nun über geeignete Maßnahmen in Software, bzw. durch Hardwaremodule im Controller sichergestellt werden, dass diese sich gegenseitig nicht beeinflussen und behindern können.

Um sicherheitsrelevanten elektrischen und elektronischen Systemen im Kraftfahrzeug einen genormten Rahmen zu bieten, wurde 2011 die erste Version der ISO 26262 [37] („Road vehicles – Functional safety“) veröffentlicht. Die Norm ist eine Anpassung der generischen Norm IEC 61508 [32] und soll die funktionale Sicherheit von E/E Systemen in Personenkraftfahrzeugen bis 3500kg zulässiger Gesamtmasse gewährleisten.

Diese Norm deckt Entwicklungsprozesse sowie den Lebenszyklus eines Fahrzeugsystems ab, jedoch nicht die Implementierung selbst. Da es für die Kooperationspartner bei der Fahrzeugentwicklung wichtig ist, dass die Softwarekomponenten untereinander kompatibel sind, wurde 1993 das industrielle Standardisierungsgremium Offene Systeme für die Elektronik im Kraftfahrzeug (OSEK) gegründet und zehn Jahre später die Entwicklungspartnerschaft Automotive Open Systems Architecture (AUTOSAR). Beide entwickelten ein Konzept mit dem Ziel, eine Standardisierung wichtiger Systemfunktionen zu ermöglichen. Hauptaugenmerk dabei liegt in der Vereinheitlichung von Software Schnittstellen, um eine flexible Integration sowie den agilen Austausch von Software in Steuergerätenetzwerken zu ermöglichen und um die Anforderungen hinsichtlich Verfügbarkeit, Sicherheit (Echtzeitfähigkeit) sowie Zuverlässigkeit zu gewährleisten.

Ein verbreitetes Konzept zur Gewährleistung der Betriebssicherheit in automotiven Systemen stellt das „Standardisierte E-Gas Überwachungskonzept für Benzin und Diesel Motorsteuerungen“ dar. Dieses wurde von einem Zusammenschluss mehrerer Automobilhersteller erstellt und beschreibt die Anforderungen zur Überwachung der Betriebssicherheit von „Drive by Wire“ Systemen.

1.1 Aufbau und Ziel dieser Arbeit

Ziel dieser Arbeit ist es, das angesprochene Sicherheitskonzept unter Berücksichtigung des derzeitigen Entwicklungsprozesses im automotiven Bereich für die Verwendung von modernen Mehrkernplattformen zu adaptieren. Dazu soll eine vollständige Entwicklungsumgebung entstehen, welche das modellbasierte Design von Applikationen, die Integration des Betriebssystems sowie die Konfiguration der Basissoftware beinhaltet. Zur Evaluierung des Systems soll ein Demonstrator aufgebaut werden, auf welchem das erweiterte Sicherheitskonzept integriert und evaluiert wird.

Dazu werden zunächst die Herausforderungen und Möglichkeiten von Mehrkernsystemen beleuchtet. Es wird auf allgemeine Eigenschaften von Mehrprozessorsystemen eingegangen und im Anschluss der Fokus auf die Besonderheiten von Steuerungen im automotiven Bereich gelegt. [Siehe Kapitel 2]

Im Anschluss daran wird jene Norm beschrieben, welche für sicherheitskritische Serienentwicklungen in der Automobilbranche unumgänglich geworden ist - die ISO 26262.

Dabei wird zunächst deren Aufbau, sowie die Unterschiede zur generischen Grundnorm IEC 61508, von welcher sie sich ableitet, erklärt. Der Prozess zur Entwicklung eines sicherheitskritischen Produktes wird aus der Sicht des Entwicklungsingenieurs beschrieben, wobei auf die Grundlagen zur Gewährleistung funktionalen Sicherheit eingegangen wird. Ziel war es, den Demonstrator in Anlehnung an diese Norm zu entwickeln. [Siehe Kapitel 3]

Kapitel 4 behandelt die für diese Arbeit wichtigsten Softwarestandards, welche auf modernen Steuergeräten im automotiven Bereich Verwendung finden. Dabei wird zunächst der OSEK Standard in seinen Teilen erklärt und vor allem auf die Betriebssystemkomponente, welche auch die Basis für den Demonstrator bildet, eingegangen. Weiters wird der AUTOSAR Standard, welcher den Nachfolger von OSEK bildet und stetig an Bedeutung gewinnt, beschrieben. [Siehe Kapitel 4]

Ausgangspunkt des zu implementierenden Sicherheitskonzepts ist das E-Gas Überwachungskonzept. Dieses wird in seinen drei Ebenen (Funktion, Funktionsüberwachung und Rechnerüberwachung) beschrieben und die Anforderungen an die Soft- und Hardware sowie die geforderten Reaktionen des Systems auf Fehler erläutert. [Siehe Kapitel 5]

Den Hauptteil dieser Arbeit stellt die Entwicklung der Demonstrationsplattform dar, welche ein „Throttle-by-Wire“ System nachbildet und so zu weiteren Untersuchungen von Mehrkern-Thematiken für sicherheitskritische Echtzeitsysteme dienen soll. Dazu wurde in Anlehnung an die Entwicklungsprozesse der ISO 26262 das bereits vorhandene E-Gas Konzept für die Verwendung auf Mehrkernarchitekturen erweitert. [Siehe Kapitel 6]

Aus dem erstellten Konzept wird im Anschluss die Entwicklung auf Systemebene durchgeführt. Der Fokus wurde dabei auf die Implementierung der drei Sicherheitsebenen gerichtet, welche den sicheren Betrieb entsprechend der erstellten Sicherheitsanforderungen erlauben sollen. [Siehe Kapitel 7]

Die nächsten beiden Kapitel beschreiben die Entwicklung auf Hardware- und Softwareebene. Darin wurde sowohl die Hardware als auch die Software nach den im vorigen Kapitel erstellten Anforderungen für die Betriebssicherheit entworfen, implementiert und getestet. [Siehe Kapitel 8 und 9]

Um die Fehlertoleranz des Gesamtsystems zu testen und um seine Betriebssicherheit zu zeigen, wurden im Anschluss an die Integration selbst die Sicherheitsfunktionen der verschiedenen Ebenen getestet und evaluiert. [Siehe Kapitel 10]

Als Abschluss der Arbeit soll eine Zusammenfassung der durchgeführten Arbeit, sowie ein Ausblick für weitere Arbeiten gegeben werden. [Siehe Kapitel 11]

1.2 Kontext der Arbeit zur Industrie

Diese Arbeit ist in Zusammenarbeit mit der AVL List GmbH entstanden. Auf diesem Wege möchte ich mich für die erhaltenen Möglichkeiten bedanken und hoffe, einen nützlichen Beitrag geleistet zu haben.

Entwicklungsprozesse für Einkernsystemen sind in der automotiven Branche sehr gut etabliert und werden durch die geringere Komplexität auf Controller-Ebene, die hohe Verfügbarkeit von (günstigen) Entwicklungslösungen sowie den geringeren Zertifizierungsaufwand und die ausgereifte Entwicklung auch weiterhin bestehen.

Jedoch bietet eine Erweiterung um Mehrkernsysteme für die Industrie besonders im automotiven Bereich durch den steigenden Leistungsbedarf von Anwendungen Chancen. So können durch die höhere funktionelle Integrationsmöglichkeit Kosten auf Hardwareebene gespart und eine Erhöhung der Performanz des Gesamtsystems erreicht werden.

Im Zuge dieser Arbeit ist eine Entwicklungsumgebung zusammen mit einer Demonstrationsanwendung entstanden, welche auf eine kostengünstige Werkzeugkette zurückgreift, um prototypische Entwicklungen auf Mehrkernsystemen zu zeigen. Die Werkzeugkette soll als Alternative zu kostenintensiven Speziallösungen die Möglichkeit bieten, den Umstieg auf Mehrkernsysteme zu erleichtern und eine Abschätzung für mögliche Kostenreduktionen und Auswirkungen auf bestehende Entwicklungsprozesse zu geben.

Kapitel 2

Herausforderungen bei Mehrkern-Architekturen

Bis vor wenigen Jahren wurde eine Erhöhung der Rechenleistung durch die Erhöhung der Transistoranzahl am Siliziumchip erreicht. Dabei verhielt sich diese Steigerung dem Moore'schen Gesetz [58] ensprechend, welches besagt, dass sich die Anzahl der Transistoren in integrierten Schaltungen bei gleichbleibenden Kosten alle zwei Jahre verdoppelt. Über die Jahre wurde die Integrationsdichte so stark erhöht, dass sich auch die Taktraten der Chips exponentiell erhöhen ließen. Zusätzlich wurden interne Strukturen parallelisiert, um den Durchsatz an Instruktionen pro Taktzyklus zu erhöhen („Instruction Level Parallelism" [59]). Da jedoch diese Verfahren im Moment an physikalische Grenzen stoßen, wird auch im Bereich von eingebetteten Systemen eine weitere Erhöhung der Rechenleistung durch Vervielfachung der Rechenkerne (CPUs) realisiert („Thread Level Parallelism"). [59]

Ein Problem bei der konventionellen Erhöhung der Rechenleistung ist, dass mit steigender Transistoranzahl auch der Energieverbrauch drastisch steigt. Das Gesetz von Pollack [66] besagt, dass bei einer Verdoppelung der Transistoranzahl und somit Verdoppelung der Leistungsaufnahme in einem Einzelkernprozessor eine Leistungssteigerung von lediglich 40% möglich ist. Verglichen dazu beträgt die theoretische Leistungssteigerung bei Verdoppelung der Rechenkerne 100%.

Hierbei muss jedoch berücksichtigt werden, dass die mögliche Geschwindigkeitssteigerung durch Erhöhung der Prozessoranzahl begrenzt ist, was durch das Amdahl'sche Gesetz [2] in Gleichung 2.1 beschrieben wird:

$$S = \frac{s+p}{s+\frac{p}{n}} = \frac{1}{s+\frac{1-s}{n}} \tag{2.1}$$

Dieses drückt aus, dass die maximale Geschwindigkeitssteigerung ($Speedup \equiv S$) vom Verhältnis zwischen dem parallelisierbaren Teil eines Programmes ($parallelizable \equiv p$), dem nicht parallelisierbaren Teil ($serial \equiv s = (1-p)$) und der Anzahl paralleler Prozessoren ($number \equiv n$) abhängt. Daraus geht hervor, dass bereits ein kleiner Anteil an nicht parallelisierbaren Instruktionen eine Sättigung des theoretischen Geschwindigkeitsgewinns bewirkt. Das Gesetz geht hierbei jedoch von der Adaptierung eines Programmes von einer Einkernarchitektur zu einer Mehrkernarchitektur aus. In eingebetteten Systemen wird jedoch häufig bereits auf Systemebene ein Konzept gewählt, welches mehrere parallelisier-

bare Programme den einzelnen Kernen zuteilt, also ein hoher Grad an Parallelisierbarkeit bereits besteht. Hauptlimitierung stellt hier meist nicht die Programmaufteilung, sondern beschränkte Ressourcen dar, welche sich die Programme untereinander teilen müssen. [59]

Besonders in Bezug auf die Echtzeitfähigkeit eines Mehrkernsystems spielen geteilte Ressourcen, wie z.B. gemeinsam genutzte Daten, Busse oder Peripheriegeräte, eine wichtige Rolle. So muss zu jedem Zeitpunkt sichergestellt werden, dass die geforderten Ressourcen dem richtigen Task zur benötigten Zeit auch zur Verfügung stehen und nicht von einem anderen Task (möglicherweise auf einer anderen CPU) blockiert werden, was in weiterer Folge zur Verletzung von Echtzeitbedingungen führen kann. [61] Besonders bei Mehrkernsystemen steigt das Risiko, den Ressourcenbedarf falsch einzuschätzen, was oft erst in der späteren Integration bzw. beim Testen sichtbar wird. [69]

2.1 Mehrkern-Architekturen im Automobilbereich

Motivation für den Einsatz von Mehrkernsystemen im Automobilbereich ist das Zusammenführen von Funktionen in weniger Steuergeräte, was eine Kostenersparnis durch weniger Gehäuse, Elektronik, Kabel, Verbindungsstecker sowie Bauaufwand bedeutet. Gleichzeitig soll dadurch die Sicherheit sowie Zuverlässigkeit des Gesamtsystems erhöht werden, da über 30% elektrischer Fehler auf Verbindungsprobleme zurückgeführt werden können. [75]

Durch die Zusammenlegung mehrerer Funktionen (mit verschiedenen Kritikalitätsstufen) auf ein Steuergerät, steigt jedoch die Komplexität im Controller und damit auch der Zertifizierungsaufwand, um ein sicheres Verhalten bei Fehlern zu garantieren. Dieser ist besonders für Mehrkernsysteme um ein Vielfaches höher als bei Einkernsystemen. [70]

In harten Echtzeit-Systemen muss garantiert werden, dass sicherheitskritische Fristen auch bei sog. „Worst Case“ Szenarien eingehalten werden. Dazu muss das gesamte Design mit Fokus auf Einhaltung maximaler Laufzeiten (Worst Case Execution Time (WCET)) von allen zeitkritischen Operationen, wie Speicherzugriffe, Buskommunikationen oder Peripherieoperationen entwickelt und getestet werden. [70]

Während Mehrkernsysteme aus dem Consumer-Bereich für einen maximalen Durchsatz entwickelt werden, steht in Echtzeitsystemen die Minimierung der maximale Laufzeit im Vordergrund. Um diese Zeit bestimmen zu können, sind exakte Modelle der Hardware nötig, und auch trotz Verfügbarkeit solcher Modelle ist eine exakte Analyse durch die enorme Vielfalt an möglichen Zuständen, welche durch die Komplexität der Architektur entsteht, nur schwer möglich. [70]

Ein allgemeines Problem bei Mehrkernsystemen auf einem Chip ist die Fehlerfortpflanzung, beziehungsweise das Auftreten ungewollter Interferenzen zwischen den einzelnen Systemen. Das betrifft nicht nur sicherheitskritische Hardwarekomponenten, sondern auch Softwarekomponenten. Durch ein Zusammenlegen mehrerer Funktionen können Anwendungen unterschiedlicher Kritikalitätsebenen aufeinandertreffen. Da diese im selben System ausgeführt werden, kann sich, bei unzureichenden Schutzmechanismen, ein Fehler von

unkritischen Funktionen auf sicherheitskritische Funktionen ausbreiten. [70]

Um garantieren zu können, dass Funktionen der kritischsten Sicherheitsstufe (nach ISO 26262 - ASIL D) nicht von anderen Funktionen gestört werden können, gibt es zwei mögliche Ansätze: Ersterer fordert, dass alle Funktionen, welche am System ausgeführt werden, der höchst nötigen Sicherheitsstufe entsprechen. Die Zertifizierung von unkritischen Funktionen ist jedoch höchst unwirtschaftlich, da sich der Entwicklungs- sowie Zertifizierungsaufwand für jede Sicherheitsstufe erhöhen. Eine Zertifizierung sog. Komfortfunktionen (wie z.B. Multimediafunktionen) sind zusätzlich so komplex, dass eine Zertifizierung nach dem Sicherheitslevel, wie er z.B. für Bremssteuerungen verwendet wird, mit inadäquat hohen Kosten verbunden wäre. Die zweite Möglichkeit verwendet zuverlässige Partitionierungsmaßnahmen, welche in die Systemarchitektur eingebaut werden. Diese müssen sicherstellen, dass eine ausreichend hohe Unabhängigkeit zwischen den verschiedenen Funktionen gegeben ist. Dadurch können einzelne Applikationen auch entsprechend ihrer eigenen Sicherheitslevels zertifiziert werden. Eingebaute Schutzmaßnahmen müssen neben einem Schutz von fehlerhaften Speicherzugriffen auch sicherstellen, dass sich die Überschreitung der Ausführungszeit einer Funktion nicht auf andere sicherheitskritische Funktionen auswirkt. Ebenso muss sichergestellt sein, dass durch Zugriffe auf geteilte Ressourcen keine Verletzung am Echtzeitverhalten des System entsteht. [70]

Auswirkungen auf die Sicherheit verschiedener Anwendungen durch Zeit- und Speicherfehler müssen vermieden werden. Ein Beispiel zur Verhinderung von Speicherfehlern stellt eine Memory Management Unit (MMU) dar. Diese partitioniert Speicherbereiche und begrenzt die Zugriffe einzelner Anwendungen auf definierte Bereiche, um so eine Interferenz mit anderen Anwendungen zu verhindern.

Ebenso müssen Zeitslots partitioniert werden um ein Verhindern von Verzögerungen im Echtzeitsystem zu ermöglichen. Dafür können den einzelnen Funktionen, sowie deren Zugriffe auf bestimmte Ressourcen sogenannte Zeitbudgets zugeteilt werden, in welchen die Abarbeitung erfolgen muss. Dies kann vom Betriebssystem überprüft werden und im Fehlerfall zu einer Terminierung der zeitüberschreitenden Funktion führen, um das Echtzeitverhalten anderer Funktionen zu gewährleisten. [61]

Die Herausforderungen und Lösungen zur Migration von Echtzeitsystemen auf Mehrkernplattformen werden in [60, 77] näher behandelt. Ausgebreitet auf Gesamtsystemebene behandelt [73] die Herausforderungen und Lösungsansätze bei der Zusammenarbeit von Subsystemen unterschiedlicher Sicherheitslevels unter der Einhaltung funktionaler Sicherheit.

Kapitel 3

Die ISO 26262 - Ein Überblick

Der ISO Standard 26262 [37] mit dem Titel „Road vehicles - Functional safety“ wurde für sicherheitsrelevante E/E Systeme in Kraftfahrzeugen bis zu 3500 kg maximal zulässigem Gesamtgewicht erstellt. Die Norm wurde von der Grundnorm IEC 61508 [32] abgeleitet und an die spezifischen Gegebenheiten von automotiven Systemen angepasst. Dabei wurde sowohl auf die Eigenheiten der Serienentwicklung, als auch auf die Produktion von Systemen in Kraftfahrzeugen eingegangen, was in der IEC 61508 nur eingeschränkt enthalten ist. In [51, 78] werden die funktionale Sicherheit der ISO 26262 sowie ein Vergleich zwischen den beiden Normen näher betrachtet.

In der Grundnorm wird Betriebssicherheit als Freiheit von unvertretbaren Risiken („Freedom from unacceptable risk“ [32]) definiert und die funktionale Sicherheit hängt von der korrekten Funktion des sicherheitsrelevanten Systems ab. [11] Die ISO 26262 beschreibt zusätzlich dazu den Produktlebenszyklus angefangen von Management, Entwicklung, Produktion, Betrieb, Service hin zur Außerbetriebnahme. [26]

In diesem Kapitel werden die Hauptbestandteile der ISO 26262 beschreiben sowie auf die Herausforderungen und Herangehensweisen bei der Anwendung in der Produktentwicklung eingegangen.

3.1 Aufbau

Die ISO 26262 besteht aus den folgenden zehn Teilen:

ISO 26262-1: Vocabulary
: Teil 1 erläutert alle in der Norm verwendeten Begriffe und Abkürzungen. Die Begriffserklärung ist dahingehend wichtig, dass sich die Bedeutungen mancher Wörter je nach Normgebiet unterscheiden. Beispielhaft angeführt das Schlüsselwort „item“. [Siehe Kapitel 3.2.2]

ISO 26262-2: Management of functional safety
: In Teil 2 der Norm werden die Anforderungen des Managements während der einzelnen Phasen des Produktlebenszyklus beschrieben. Weiters wird auf notwendige organisatorische Voraussetzungen eingegangen, um ein System gemäß den geforderten automotiven Sicherheits-Integritätslevels (Automotive Safety Integration Level (ASIL)) zu entwickeln.

ISO 26262-3: Concept phase

In der Konzeptphase (Teil 3) wird zunächst das Gesamtsystem aus funktionaler Sicht definiert und funktionale Abhängigkeiten sowie nötige Komponenten eruiert. Anschließend wird eine Gefährdungsanalyse und Risikoabschätzung (Hazard Analysis and Risks Assessment (HARA)) durchgeführt. Dabei werden potentielle Gefährdungen des Systems identifiziert, indem die Fehlfunktionen des untersuchten Systems situationsspezifisch betrachtet werden. [Siehe Kapitel 3.3]

ISO 26262-4: Product development at the system level

Teil 4 beschreibt die Entwicklungsprozesse auf Systemebene. Dazu werden für das geforderte Sicherheits-Integrationslevel Anforderungen erstellt, sowie ein Konzept auf Systemebene festgelegt. [Siehe Kapitel 3.4]

ISO 26262-5: Product development at the hardware level

In Teil 5 werden die Entwicklungsprozesse auf Hardwareebene beschrieben und die verschiedenen Methodiken zur Umsetzung und Verifizierung für die jeweiligen ASILs dargelegt. Nach Erstellung technischer Sicherheitsanforderungen für die Hardware, wird diese nach den Vorgaben von Teil 5 entworfen, implementiert und getestet. [Siehe Kapitel 3.5]

ISO 26262-6: Product development at the software level

Teil 6 beschreibt die Entwicklungsprozesse auf Softwareebene. Dabei werden wiederum die Analyseverfahren zur Erreichung des jeweiligen Sicherheitsintegrationslevels beschrieben. Nach Erstellung technischer Sicherheitsanforderungen für die Software, wird diese nach den Vorgaben von Teil 6 entworfen, implementiert und getestet. [Siehe Kapitel 3.6]

ISO 26262-7: Production and operation

Der 7. Teil enthält Forderungen bezüglich Produktion, Installation, Betrieb, Wartung, Reparatur sowie die Stilllegung. Dabei wird das prinzipielle Vorgehen beim Erstellen der jeweiligen Pläne beschrieben um die geforderten Sicherheitsanforderungen gewährleisten zu können.

ISO 26262-8: Supporting processes

In Teil 8 werden unterstützende Prozesse beschrieben, welche zur Umsetzung der Anforderungen aller Sicherheitsaspekte nötig sind. Dazu gehören die Beschreibung und Zuordnung von Verantwortungen innerhalb der Entwicklungsumgebung, sowie die richtigen Spezifikationen der Anforderungen an den gesamten Sicherheitslebenszyklus. Weiters werden das Konfigurations- und Änderungsmanagement erläutert und die Vorgehensweise für Verifikation und Dokumentation beschrieben. Auch die Risikominimierung durch verwendete Werkzeuge wie Software- und Hardwarekomponenten wird angesprochen.

ISO 26262-9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analysis

Durch Steigerung an Redundanz können einzelne Sicherheitsintegrationslevel verringert werden. Die Regeln um eine Verringerung des ASILs und eine damit

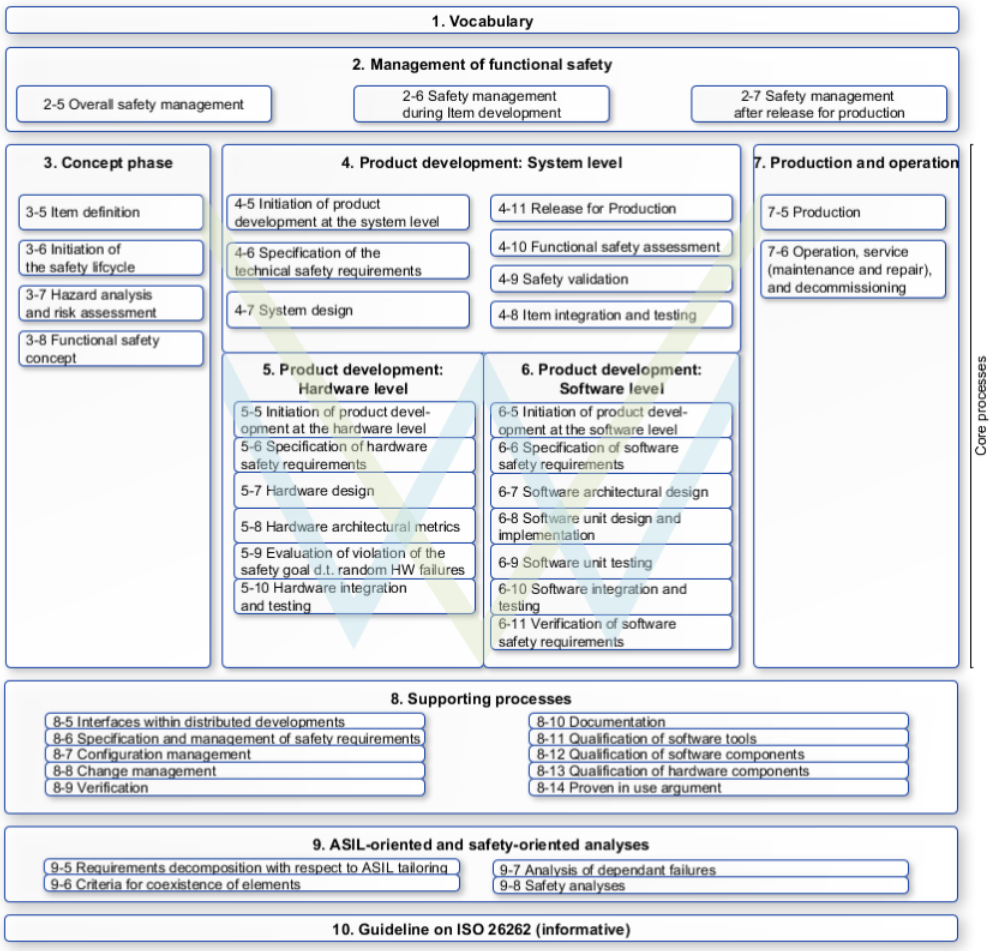


Abbildung 3.1: Übersicht der Teile in der ISO 26262 gemäß [42]

verbundene Aufwandsreduktion zu erreichen, wird in Teil 9 beschrieben. Weiters werden in diesem Teil eine Vorgabe zur Kritikalitätsanalyse sowie die Durchführung von Analysen abhängiger Ausfälle erläutert, um „Common Cause“- und kaskadierende Fehler zu identifizieren. Außerdem wird in Teil 9 auf unterschiedliche Verfahren eingegangen, um sicherheitskritische Fehler im System zu erkennen.

ISO 26262-10: Guideline on ISO 26262

Teil 10 beinhaltet Anwendungsbeispiele, Erläuterungen und weiterführende Information zu einigen Bereichen des Standards. Im Gegensatz zu den anderen Teilen ist dieser nicht normativ sondern rein informativ.

Abbildung 3.1 zeigt die Gesamtstruktur der ISO 26262. Diese basiert auf einem V-Modell als Referenz-Prozessmodell für die verschiedenen Phasen bei der Produktentwicklung. Die schattierten „V“s repräsentieren die Beziehungen während des Entwicklungsprozesses zwi-

schen den einzelnen Norm-Teilen.

3.2 Hauptkonzepte der ISO 26262

3.2.1 Unterscheidung zur IEC 61508

Die IEC 61508 [32] („Functional safety of electrical, electronic and programmable electronic (E/E/PE) safety-related systems“) wurde als generischer Standard und als Basis für funktionale Sicherheit geschaffen. Aus ihren Anforderungen leiteten sich für die jeweiligen Industriesektoren eigenen Normen mit Bezug auf funktionale Sicherheit ab. Wie oben beschrieben, leitet sich auch die ISO 26262 für die Automobilindustrie daraus ab, jedoch waren mehrere Anpassungen und Änderungen nötig, um auf die Besonderheiten dieser Sparte Rücksicht zu nehmen. [42]

Die IEC 61508 basiert auf einem „Equipment under Control“ (EUC) Modell. In einer Gefahrenanalyse werden die möglichen Gefahren in Zusammenhang mit dem EUC identifiziert, um anschließend Maßnahmen zur Gefahrenminderung zu erstellen. Das kann durch Elektrisch/Elektronisch/Programmierbar-Elektronisch (E/E/PE) Systeme, oder andere sicherheitsbezogene Systeme bzw. externe Risiko-reduzierende Maßnahmen geschehen. Risikominderung bei E/E/PE Systemen kann hier durch dedizierte Sicherheitsfunktionen gewährleistet werden, welche eigens dafür ausgelegt sind oder können in die Steuerung selbst inkludiert werden. [42]

Im Gegensatz dazu ist eine Abgrenzung in automotiven Systemen nur schwer möglich. Die Sicherheit des Fahrzeuges hängt direkt von der korrekten Funktion des Steuerungssystems ab. Deswegen verwendet die ISO 26262 anstatt des Modells separater Sicherheitsfunktionen ein Modell mit Sicherheitszielen und ein Sicherheitskonzept (Teil 3): [42]

1. In dem „Hazard Analysis and Risks Assessment (HARA)“ Verfahren werden zunächst Gefahren identifiziert, welche eine Minderung des Risikos benötigen.
2. Anschließend wird für jedes Gefahrenereignis („hazardous event“) ein Sicherheitsziel („safety goal“) formuliert.
3. Nun wird jedem Sicherheitsziel auf Grundlage von der Schwere, Häufigkeit und Kontrollierbarkeit des Gefahrenereignisses ein ASIL zugewiesen.
4. Danach werden in einem funktionalen Sicherheitskonzept („functional safety concept“) Bedingungen zur Funktion festgehalten, um das Sicherheitsziel zu erfüllen.
5. Das technische Sicherheitskonzept („technical safety concept“) beschreibt nun, wie die Funktionen aus dem funktionalen Sicherheitskonzept in Hardware und Software zu implementieren sind.
6. Zuletzt werden die Sicherheitsanforderungen für Software und Hardware abgeleitet, welche im Zuge des Hardware und Software Designs implementiert werden.

Die IEC 61508 zielt auf Einzelsysteme oder Systeme in geringen Stückzahlen ab. Dabei werden Systeme zuerst gebaut und getestet, anschließend am Bestimmungsort aufgestellt

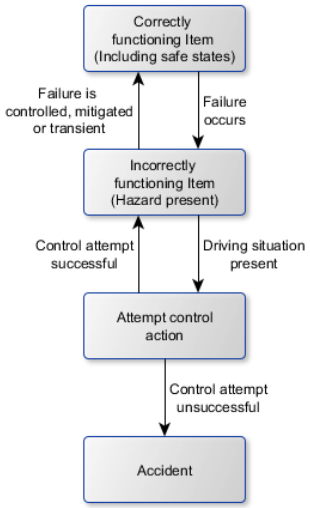


Abbildung 3.2: Zustandsmodell für das Risiko im automotiven Bereich gemäß [42]

und erst im Anschluss erfolgt eine Sicherheitsvalidierung. Bei Systemen für den Massenmarkt, wie Fahrzeuge, muss die Sicherheitsvalidierung bereits vor dem Start der Serienproduktion erfolgt sein. Somit ist auch der Ablauf der Aktivitäten im Lebenszyklus in der ISO 26262 unterschiedlich. Im Gegensatz zur IEC Norm beschreibt die ISO Norm deshalb auch Anforderungen an die Produktion selbst. [42]

In der IEC 61508 wird implizit davon ausgegangen, dass das System von einer einzelnen Organisation entworfen und implementiert wird. Automotive Systeme werden generell von mehreren Zulieferern für einen Fahrzeughersteller produziert. Die ISO 26262 beinhaltet spezifische Anforderungen für die Verwaltung einer Entwicklung, welche sich über mehrere Organisationen verteilt. [42]

Im Gegensatz zur ISO 26262 besitzt die IEC 61508 keine normativen Anforderungen zur Gefahrenklassifizierung. Hierfür besitzt die ISO Norm ein Schema, welches an die fahrzeugspezifischen Gefahren angepasst ist. Dieses berücksichtigt, dass eine Gefahr im automotiven System nicht notwendigerweise zu einem Unfall führen muss. Die Folge hängt davon ab, inwiefern gefährdete Personen der Gefahr ausgesetzt sind sowie von der Fahrsituation selbst. Weiters fließt in die Klassifizierung mit ein, ob das Resultat der Gefahr kontrollierbar ist. Abbildung 3.2 zeigt dieses Konzept in einem Zustandsmodell, welches die Entstehungskette eines Unfalls skizziert. [42]

Eine Analyse der verwendeten Vorteile und Herausforderungen der in der ISO angewendeten Strukturen zur Beurteilung der funktionalen Sicherheit eines Produktes wurde in [8] durchgeführt. Die Anforderungen bei der Hardware- und Softwareentwicklung sind in Anlehnung an den Stand der Technik in der Automobilindustrie in die ISO 26262 übernommen worden. Dazu gehört die Modell-basierte Entwicklung, welche in der IEC Norm nicht berücksichtigt worden ist. Weiters empfiehlt die ISO Norm Methoden und Maßnahmen,

welche auf Anwendungen aus dem Automobilsektor basieren. [42]

3.2.2 In der Norm verwendete Begriffe

Die Norm verwendet zur Definition der verschiedenen Teile des Produkts mehrere Begriffe, welche in einem bestimmten Bezug zueinander stehen. [42] Abbildung 3.3 zeigt den hierarchischen Aufbau dieser einzelnen Teile, zu welchen die Folgenden gehören:

Item
: Ein „Item“ besteht aus einem System bzw. einer Reihe an Systemen, welche eine Funktion auf Fahrzeugebene implementiert, auf welche die ISO 26262 angewendet wird.

System
: Ein System besteht aus mindestens einem Sensor, einer Steuerung sowie einem Aktuator, welche als Elemente in Verbindung stehen.

Component
: Eine Komponente stellt ein Element dar, welches nicht auf Systemebene besteht und welches logisch und technisch trennbar ist. Sie umfasst mehr als einen Hardwarebaustein sowie eine oder mehrere Softwareeinheiten.

Hardware Part
: Hardwarebaustein, welcher nicht weiter unterteilt werden kann.

Software Unit
: Softwareeinheit auf unterster Ebene der Software Architektur welche im Einzelnen getestet werden kann.

Element
: System oder Teil eines Systems, welches Komponenten, Hardware, Software, Hardwarebausteine und Softwareeinheiten beinhaltet.

3.2.3 Definition von Fault, Error und Failure

In der Norm unterscheidet man den Begriff „Fehler“ sinngemäß als Fehlerquelle, Fehler im Sinne von Abweichung, sowie Fehlfunktion.

Fault
: „Abnormal condition that can cause an element or an item to fail.“ - [36]
Der „Fault“ stellt einen abnormalen Zusand dar, welcher zur Fehlfunktion eines Elements bis hin zum Ausfall der Gesamtfunktion selbst führen kann. (Bsp. Kabelbruch)

Error
: „Discrepancy between a computed, observed or measured value or condition, and the true, specified, or theoretically correct value or condition.“ - [36]
Der „Error“ stellt eine Diskrepanz zwischen den berechneten, beobachteten oder ge-

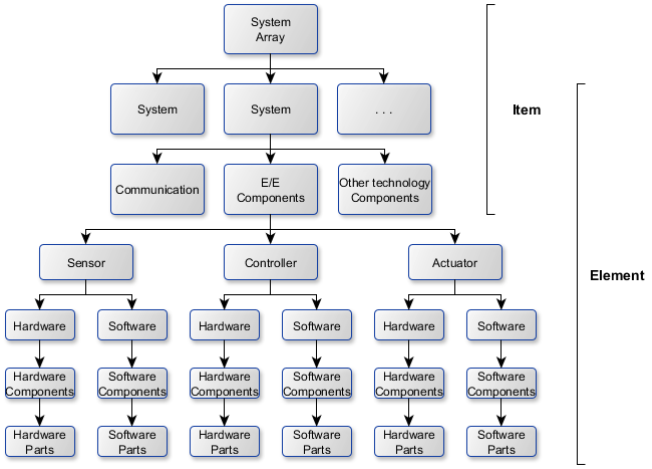


Abbildung 3.3: Bestandteile eines Items gemäß [42]

messenen Werten/Zuständen und den laut Spezifikation korrekten Werten/Zuständen dar. (Bsp. Erkennung eines Kabelbruches)

Failure

„Termination of the ability of an element, to perform a function as required.“ - [36] Der „Failure“ stellt die Fehlfunktion dar, in welcher ein Element eine Funktion nicht laut Spezifikation ausführt. (Bsp. Kabelbruch führt zu ungewollter Funktion des Systems.)

Abbildung 3.4 zeigt die Ausbreitung eines „Fault“s zu einem „Error“ hin zum „Failure“. Dabei werden drei verschiedene Ursachen genannt: Systematische Softwarefehler, zufällige Hardwarefehler sowie systematische Hardwarefehler.

Systematische Fehler entstehen typischerweise während des Entwurfs oder entstehen durch Spezifikationsfehler. Alle Softwarefehler sowie ein Teil an Hardwarefehlern sind systematisch begründet. Zufällige Hardwarefehler entstehen typischerweise durch physikalische Prozesse wie Beschädigung oder Alterung. Auf Komponenten-Ebene kann jede Fehlerursache zu verschiedenen Fehlfunktionen führen. Diese Fehlfunktion der Komponente sind wiederum Fehlerursachen in der darüber gelegenen Abstraktionsebene.

Das Beispiel in Abbildung 3.4 zeigt, dass Fehler verschiedener Ursachen zur selben Fehlfunktion auf „Item“-Ebene führen können. Ein Teil dieser Fehlfunktionen können als Gefahrenquelle bei zusätzlichen Umgebungsfaktoren zu einem Unfall beitragen, wogegen entsprechende Absicherungsmaßnahmen entsprechend der ISO 26262 getroffen werden müssen. [42]

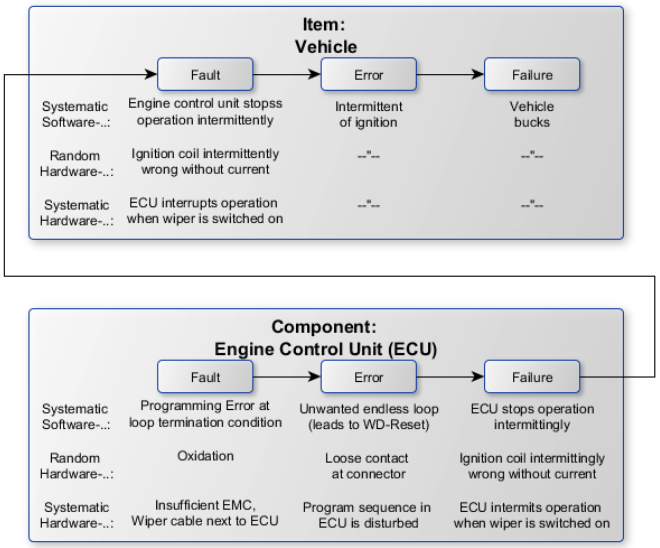


Abbildung 3.4: Beispiel einer Fehlerfortpflanzung gemäß [42]

3.2.4 Fehlerklassen

Generell werden in der ISO Norm Fehler auf zweifach-Kombinationen begrenzt, solange im funktionalen bzw. technischen Sicherheitskonzept nicht gezeigt wird, dass höhere Fehlerkombinationen relevant sind. Somit unterscheidet man zufällig auftretende Fehler in: [42]

Single Point Fault
: Dieser Fehler führt direkt zur Verletzung des Sicherheitsziels. Es ist kein Sicherheitsmechanismus implementiert, welcher den Fehler eines Bauteils mit dem Potential zur Verletzung des Sicherheitsziels kontrollieren kann.

Residual Fault
: Dieser Fehler führt ebenfalls zur Verletzung des Sicherheitszieles. Er stellt den Restfehler dar, welcher durch die implementierten Sicherheitsmechanismen nicht erkannt wird und welcher durch den Fehler eines Hardwarebauteils zur Verletzung des Sicherheitszieles führt.

Detected dual point fault
: Der Fehler trägt zur Verletzung des Sicherheitszieles bei. Diese Verletzung tritt jedoch nur in Kombination mit einem unabhängigen anderen Fehler auf. Er kann über einen Sicherheitsmechanismus in einer vorgegebenen Zeit erkannt werden, bevor dieser latent wird.

Perceived dual point fault
: Dieser Fehler trägt zur Verletzung des Sicherheitszieles bei, jedoch nur in dem Fall, dass ein zweiter, unabhängiger Fehler auftritt. Er wird vom Fahrer wahrgenommen,

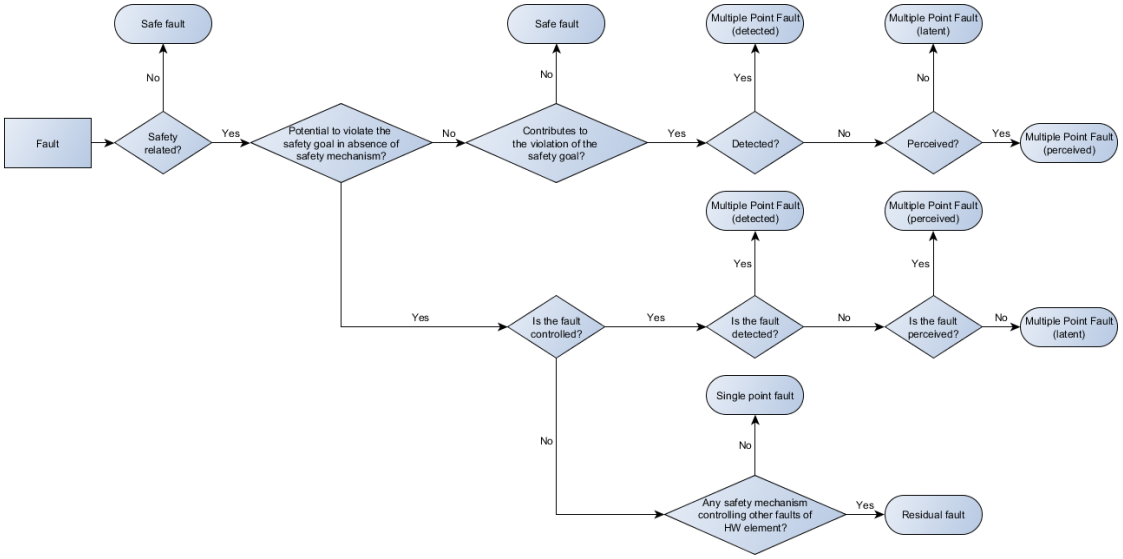


Abbildung 3.5: Flussdiagramm zur Fehlerklassifizierung gemäß [42]

bevor dieser von einem Sicherheitsmechanismus erkannt wird.

Latent dual point fault

Ähnlich der obigen Fehlerklasse, trägt dieser zur Verletzung des Sicherheitszieles bei, falls ein zweiter, unabhängiger Fehler auftritt. Jedoch wird dieser weder von einem Sicherheitsmechanismus erkannt, noch vom Fahrer wahrgenommen. Der Fehler wird toleriert, da das System noch betriebsfähig ist.

Safe fault

Dieser Fehler trägt nicht zur Verletzung des Sicherheitsziels bei und wird somit als sicher angesehen.

In Abbildung 3.5 werden die verschiedenen Fehlerklassen in einem Flussdiagramm hergeleitet. Zur Unterscheidung wird die Relevanz zur Sicherheit, Erkennbarkeit, Wahrnehmbarkeit, Kontrollierbarkeit als Parameter herangezogen. [42]

3.3 Konzeptphase (Concept Phase)

In der Konzeptphase werden die Gefährdungen betrachtet, sowie Risiken eingeschätzt, welche in Verbindung mit der funktionalen Sicherheit bei Fahrzeugsystemen stehen. Auf Basis von dieser Klassifizierung der Gefährdungen in die ASIL Stufen werden Sicherheitsziele definiert, aus denen funktionale Sicherheitsanforderungen hervorgehen, welche sowohl das Gesamtsystem, als auch ihre Teilsysteme beinhalten.

Dabei wird zunächst das „Item“ selbst definiert. Die ISO Norm wird auf jedes der definierten Items eigenständig angewendet. Dabei soll das Item so definiert sein, dass alle

verfügbaren Informationen wie nicht-funktionale, rechtliche und Sicherheitsanforderungen in Bezug auf das Item, die darin enthaltenen Elemente, sowie die Interaktion des Items mit seiner Umgebung klar verständlich sind. Weiters sollen sein Verhalten selbst, sowie mögliche Betriebsszenarien beschrieben werden.

Ist das Item definiert, wird der dazugehörige Sicherheitslebenszyklus initiiert. Je nachdem ob es sich um eine Neu- oder Weiterentwicklung handelt, wird der komplette Zyklus oder nur die Teile, welche von Änderungen betroffen sind, angewendet. Dazu zählen Änderungen am Item selbst, an der Entwicklung, Implementierung, funktionalen Erweiterungen, Optimierungen oder Veränderungen in der Umgebung des Items. Je nach Modifikation werden daraus die notwendigen Arbeitsprodukte zur Erfüllung der Anforderungen nach ISO 26262 abgeleitet. Im Falle einer Neuentwicklung sind natürlich die Arbeitsprodukte des gesamten Lebenszyklus zu betrachten. [42]

3.3.1 Gefährdungs- und Risikoanalyse (Hazard analysis and risk assessment)

Ist das Item vollständig definiert, erfolgt eine Gefährdungs- und Risikoanalyse. Dazu wird in der ISO 26262 eine Methode verwendet, welche speziell auf die Gefahren im Automobilbereich angepasst ist. Für die Analyse müssen der Anwendungsbereich, bereits bekannte Gefährdungen, Betriebsbedingungen, Einsatzbedingungen, sowie mögliche Fehlbedienungen des Items bekannt sein. Daraus werden mögliche funktionale Fehler ohne Berücksichtigung der Ursache ermittelt und deren Auswirkungen auf Fahrzeugebene beschrieben. Dabei werden die Gefährdungen sowie deren Auswirkungen je nach Einsatzsituation bewertet. Die ISO Norm berücksichtigt bei der Analyse lediglich die Funktion des Items, nicht jedoch andere Gefährdungen. [38]

Dazu wird ausgehend von der Systemdefinition eine Situationsanalyse durchgeführt, in welcher das fehlerhafte Verhalten des Systems zu einem Gefährdungsereignis („hazardous event“) führen kann. Anschließend werden die dadurch möglichen Gefahren durch geeignete Methoden wie Failure Mode and Effects Analysis (FMEA) [34] oder Fault Tree Analysis (FTA) [35] systematisch abgeleitet werden. Sind diese bekannt, werden die Gefährdungsereignisse, welche sich im Anwendungsbereich der ISO 26262 befinden, methodisch bewertet. Nun wird daraus ein ASIL bestimmt und Sicherheitsziele („Safety goals“) definiert. [Siehe Kapitel 3.3.2]

Zuletzt wird das die Gefährdungs- und Risikoanalyse, die Sicherheitsziele auf Vollständigkeit der Situationen, Übereinstimmung mit der Systemdefinition, Vollständigkeit der Gefährdungsereignisse sowie Übereinstimmung des Ereignisses mit dem zugeordneten ASIL überprüft. [38]

3.3.2 Klassifizierung der Gefährdungsereignisse und Bestimmung des ASILs

In der ISO 26262 werden zur Klassifizierung von Gefährdungen drei Kriterien verwendet. Es wird dabei nach Schwere des möglichen Schadens (Tabelle 3.1), nach Häufigkeit (Tabelle 3.2) sowie nach Kontrollierbarkeit (Tabelle 3.3) eingestuft.

Tabelle 3.1: Einteilung nach Schwere des möglichen Schadens („Severity“) gemäß [38]

Classes of severity - S			
S0	**S1**	**S2**	**S3**
No injuries	Light and moderate injuries	Severe and life-threatening injuries (survival probable)	Life-threatening injuries (survival uncertain), fatal injuries

Tabelle 3.2: Einteilung nach Häufigkeit der Situation („Probability of Exposure“) gemäß [38]

Classes of probability of exposure regarding operational situations - E				
E0	**E1**	**E2**	**E3**	**E4**
Incredible	Very low probability	Low probability	Medium probability	High probability

Tabelle 3.3: Einteilung nach Kontrollierbarkeit der Situation („Controllability“) gemäß [38]

Classes of controllability - C			
C0	**C1**	**C2**	**C3**
Controllable in general	Simply controllable	Normally controllable	Difficult to control or uncontrollable

Anhand dieser Kategorisierung wird für jede Gefährdung ein automotiver Sicherheits-Integritätslevel ASIL festgelegt. Diese bestimmen die notwendigen Sicherheitsanforderungen an das System, um ein akzeptables Restrisiko zu unterschreiten. Anhand dieser Klassen werden im Weiteren Maßnahmen und Methoden zur Risikominimierung vorgeschrieben. Dabei entspricht ASIL D der höchsten Klasse, welche dementsprechend auch die meisten Maßnahmen erfordert, um die geforderte Minimierung des Risikos zu erreichen.

Tabelle 3.4 zeigt die Einstufung des automotiver Sicherheits-Integritätslevels anhand der drei vorhin beschriebenen Kriterien.

Im Anschluss an die Klassifizierung werden für jede Gefährdung Sicherheitsziele definiert, welche der höchsten Ebene an Sicherheitsanforderungen entsprechen. Die Sicherheitsziele stellen funktionale Beschreibungen zur Vermeidung von Risiken und Gefährdungen dar,

Tabelle 3.4: ASIL Bestimmungstabelle gemäß [38]

Severity	**Exposure**	**Controllability**		
		C1	**C2**	**C3**
S1	**E1**	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	**E1**	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E1	A	B	C
S3	**E1**	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Automotive Safety Integration Level (ASIL)

die technische Realisierung bleibt jedoch noch offen. Bei mehreren Sicherheitszielen können ähnliche Ziele zusammengefasst werden, wobei der höchste ASIL übernommen wird. Zusätzlich werden zur Erreichung der Sicherheitsziele geforderte Betriebszustände festgelegt. Am Ende aller Arbeitsschritte wird eine Begutachtung der Ergebnisse gefordert. [42]

3.3.3 Das funktionale Sicherheitskonzept („Functional Safety Concept“)

Nach der Durchführung der Gefährdungs- und Risikoanalyse und der Bestimmung der ASILs, sowie der Sicherheitsziele, wird das funktionale Sicherheitskonzept erstellt. Darin werden Maßnahmen festgelegt, welche zur Erreichung des Sicherheitszieles beitragen. Dabei werden Fehlererkennung, Fehlervermeidung, Reaktionen zur Gewährleistung sicherer Betriebszustände, Mechanismen zur Fehlertoleranz und Absicherungsmethoden aus rein funktionaler Sicht festgehalten. Diese Anforderungen werden den einzelnen Elementen der vorläufigen Systemarchitektur zugeteilt, und der ASIL des Sicherheitsziels wird auf das Element übertragen. Wie im vorherigen Schritt wird das Sicherheitskonzept nach Fertigstellung auf Übereinstimmung mit den Sicherheitszielen überprüft. Dabei werden die funktionalen Sicherheitsanforderungen („Functional Safety Requirements“)durch Tests bereits hinsichtlich ihrer Wirksamkeit bewertet. [42]

3.4 Produktentwicklung auf Systemebene

Bei der Produktentwicklung auf Systemebene werden die Anforderungen an sicherheitsrelevante Systeme beschrieben. Dabei werden Maßnahmen vorgegeben, welche vor, während, und nach der Produktentwicklung auf Software- und Hardwareebene anzuwenden sind. (Siehe Abbildung 3.6)

Zunächst werden technische Sicherheitsanforderungen („Technical Safety Requirements“) erstellt, welche aus dem funktionalen Sicherheitskonzept hervorgehen. Hier erfolgt der Übergang vom generischen Konzept zu Anforderungen, welche die Hardware- und Softwarekomponenten miteinbeziehen, auf welchen sie angewendet werden. Anschließend erfolgt die Erstellung eines technischen Sicherheitskonzepts, welches die funktionalen und technischen Anforderungen aus den vorhergehenden Schritten erfüllt. Dabei muss für jedes Element eine klare Verbindung zu den jeweiligen Anforderungen bis hin zum verknüpften Sicherheitsziel bestehen.

Um systematische Fehler zu erkennen, werden je nach ASIL induktive Analysen wie eine FMEA oder deduktive Analysen wie eine FTA vorgeschrieben. Zur Erkennung zufälliger Hardwaredefekte werden in der Systementwicklung eine Diagnoseabdeckung sowie Zielvorgaben für Fehlerraten auf Elementebene beschrieben. Weiters wird die Hardware-Software Schnittstelle („Hardware-Software Interface“) spezifiziert und während des Entwicklungsprozess von Hard- und Software weiter verfeinert. Anschließend werden die Anforderungen für den restlichen Lebenszyklus erstellt. Diese betreffen die Produktion, den Betrieb, Wartung sowie die Außerbetriebnahme.

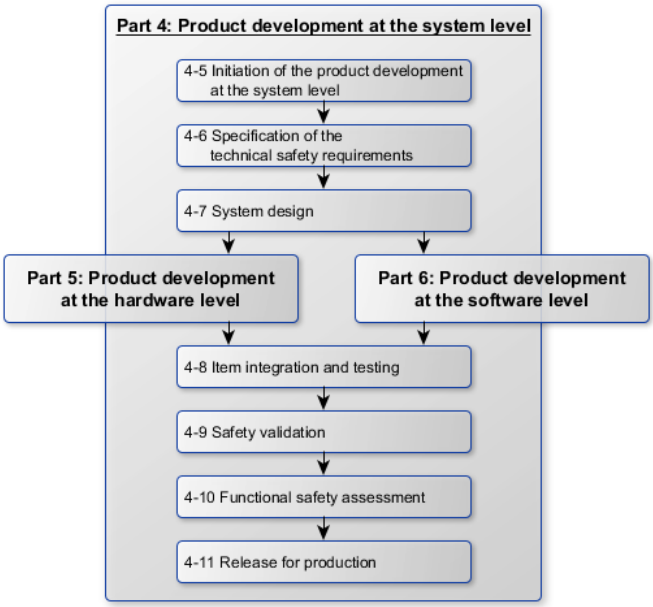


Abbildung 3.6: Flussdiagramm zur Produktentwicklung auf Systemebene gemäß [39]

Sind alle Anforderungen auf Systemebene definiert, wird das Systemdesign auf Konformität und Vollständigkeit gegenüber dem technischen Sicherheitskonzept überprüft. Nach der Produktentwicklung auf Hardware- und Softwareebene werden die Ergebnisse daraus auf Systemebene getestet. Dazu wird jedes Item gegenüber allen funktionalen und technischen Sicherheitsanforderungen auf deren Umsetzung geprüft und hinsichtlich ihrer Sicherheitsziele validiert. Nach Begutachtung der funktionalen Sicherheit kann die Freigabe für die Produktion erfolgen. [26]

Abbildung 3.6 zeigt den Ablauf der Produktentwicklung auf Systemebene. Diese umschließt die beiden Ebenen der eigentlichen Implementierung in Hard- und Software.

3.5 Produktentwicklung auf Hardwareebene

Die Anforderungen an die Hardware ergeben sich aus dem funktionalen und dem davon abgeleiteten technischen Sicherheitskonzept. Durch die Abhängigkeit der Software von der Hardware, kann auch das Softwaresicherheitskonzept Anforderungen stellen.

Zunächst werden Hardwaresicherheitsanforderungen festgelegt. Diese dienen als Vorgabe für das Hardwaredesign bei dem auch die Spezifikationen des Systemdesigns als Vorlage und zur Verifizierung dienen. Jedes Hardware-Element bekommt dabei den höchsten ASIL der jeweiligen technischen Sicherheitsanforderungen zugewiesen. Auch Überlegungen zu nicht-funktionalen Fehlerursachen sollen in das Hardwaredesign einfließen. Weiters sollen Maßnahmen bestimmt werden, mit welchen die Systemarchitektur auf deren Wirk-

samkeit bezüglich zufälliger Hardwarefehler verifiziert werden kann. Um zufällige Hardwarefehler einschätzen zu können, müssen für jedes sicherheitsrelevante Bauteil sog. Ausfallsraten (Failure in Time (FIT)) bestimmt werden, woraus sich die Single Point Fault Metric (SPFM) für Einfachfehler sowie die Latent Fault Metric (LFM) für latente Fehler ergibt. Zur Einhaltung des jeweiligen ASILs muss nachgewiesen werden, dass diese Diagnoseabdeckung eine gewisse Schwelle überschreiten. [26]

Zur Bestimmung der SPFM kann Gleichung 3.1 herangezogen werden:

$$SPFM = 1 - \frac{\sum\limits_{SR,HW}(\lambda_{SPF} + \lambda_{RF})}{\sum\limits_{SR,HW}(\lambda)} \tag{3.1}$$

Dabei stellt λ die FIT Rate (Ausfallsrate) dar. Als Ergebnis erhält man einen Wert, der das Verhältnis der erkannten Fehler zu den Gesamtfehlern darstellt und vergleichbar ist mit der Fehlerabdeckung. Die SPFM beinhaltet dabei nur Einfachfehler, also Fehler, welche ohne Absicherungsmaßnahmen direkt auf die Funktionsfähigkeit des Systems Einfluss nehmen. Tabelle 3.5 zeigt die Schwellwerte für ASIL B-D, für welche eine Mindestabdeckung an Einzelfehlern gefordert ist.

Tabelle 3.5: Geforderte Single Point Fault Metrik [40]

	ASIL A	**ASIL B**	**ASIL C**	**ASIL D**
SPFM:	-	$\geq 90\%$	$\geq 97\%$	$\geq 99\%$

Auf ähnliche Weise wird die Metrik für latente Fehler in Gleichung 3.2 berechnet:

$$LFM = 1 - \frac{\sum\limits_{SR,HW}(\lambda_{MPF,latent})}{\sum\limits_{SR,HW}(\lambda - \lambda_{SPF} - \lambda_{RF})} \tag{3.2}$$

Zur Berechnung der LFM wird die Summe der latenten Mehrfachfehlerraten durch die Summe aus der Gesamtfehlerrate abzüglich Einfachfehler und Restfehler gebildet. Tabelle 3.6 zeigt die Schwellwerte für ASIL B-D, für welche eine Mindestabdeckung an latenten Fehlern gefordert ist.

Tabelle 3.6: Geforderte Latent Fault Metrik gemäß [40]

	ASIL A	**ASIL B**	**ASIL C**	**ASIL D**
LFM:	-	$\geq 60\%$	$\geq 80\%$	$\geq 90\%$

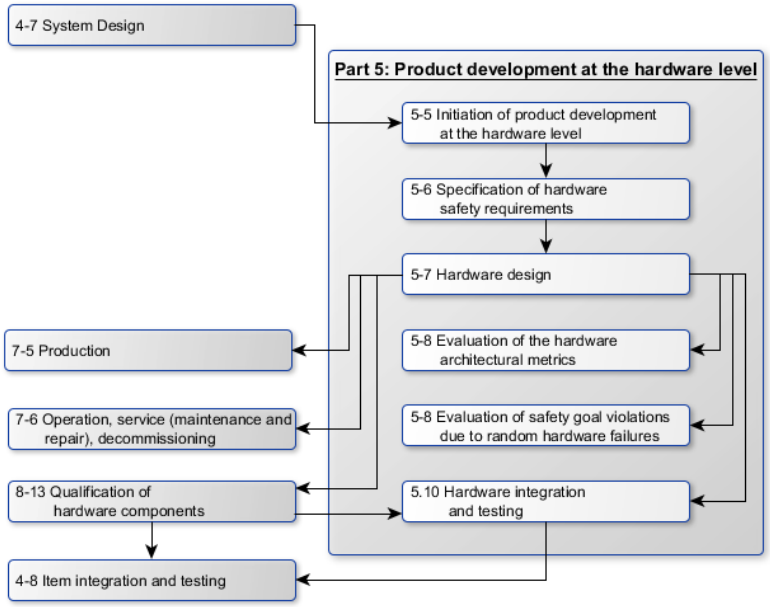


Abbildung 3.7: Flussdiagramm zur Produktentwicklung auf Hardwareebene gemäß [40]

Besonders bei Neuentwicklungen stellt die Ausfallrate (λ) ein Problem dar, da sie nur aufwendig messbar ist. Häufig verwendet man daher vermutete Ausfallsraten, welche auf anerkannte Quellen, Statistiken oder Experteneinschätzungen beruhen können.

Zur Einhaltung des ASILs ist bei der Hardwareentwicklung nicht nur die Fehlerabdeckung wichtig, sondern auch eine absolute Ausfallrate für zufällige Hardwarefehler. Diese darf je nach ASIL nicht überschritten werden. Die einzelnen Werte sind in Tabelle 3.7 angegeben.

Tabelle 3.7: Geforderte maximale Ausfallsraten des Gesamtsystems durch zufällige Hardwarefehler gemäß [40]

	ASIL A	**ASIL B**	**ASIL C**	**ASIL D**
FIT Rate:	-	$< \frac{10^{-7}}{h}$	$< \frac{10^{-7}}{h}$	$< \frac{10^{-8}}{h}$

Nachdem die Qualifizierung gegen zufällige Hardwarefehler sowie die Fehlerabdeckung abgeschlossen ist, kann die Integration der Hardware durchgeführt werden und dieselbe über geeignete Testmethoden auf Einhaltung der Anforderungen überprüft werden. Abbildung 3.7 zeigt den Ablauf der Produktentwicklung auf Hardwareebene sowie deren Abhängigkeiten im Zusammenhang mit den anderen Bereichen. [40]

3.6 Produktentwicklung auf Softwareebene

Anforderungen an die Software werden aus dem technischen Sicherheitskonzept und dem Systemdesign abgeleitet, wobei auch die Schnittstelle zwischen Hardware und Software weiter spezifiziert werden kann. Zu beachten ist, dass die Software an die Hardware gebunden ist und, obwohl sie so unabhängig wie möglich davon sein soll, um eine Wiederverwendung auf unterschiedlicher Hardware zu ermöglichen, gibt die Hardware dennoch gewisse Rahmenbedingungen für die Software vor. [40]

Wie bei der Hardwareentwicklung wird auch bei der Softwarentwicklung mit dem technischen Sicherheitskonzept begonnen, in welchem die Sicherheitsanforderungen für softwarebasierte Funktionen beschrieben werden, welche zu einer Verletzung der Sicherheitsziele führen können. Im Anschluss wird aus diesem Konzept die Softwarearchitektur entwickelt, welche die einzelnen Softwarekomponenten sowie deren Schnittstellen definiert. Dabei müssen sowohl statische Aspekte, wie Schnittstellen und Datenpfade zwischen allen Softwarekomponenten, als auch dynamische Aspekte, wie Prozessabläufe und das Zeitverhalten, berücksichtigt werden. [40]

Bei der Softwareentwicklung sicherheitsrelevanter Komponenten wird zwischen neuen Entwicklungen, Wiederverwendungen mit und ohne Modifizierungen, sowie Commercial Off-The-Shelf (COTS) Produkten unterschieden. Falls von einer Softwarekomponente mehrere Funktionen mit verschiedenen ASILs abhängen, wird für die Komponente stets der höchste ASIL der zugehörigen Funktionen angewendet. Um eine gegenseitige Beeinflussung zu verhindern, soll die Software auf geeignete Weise partitioniert und geschützt werden. Bevor mit der Integration der Software begonnen wird, muss die Softwarearchitektur sowie die einzelnen Softwarekomponenten auf Betriebssicherheit überprüft werden. [40]

Im Anschluss an die Integration wird diese mittels geeigneter Maßnahmen überprüft. Im Weiteren muss die Softwarearchitektur, sowie das Sicherheitskonzept selbst überprüft und getestet werden. Zuletzt wird auf Systemebene das Gesamtsystem integriert und getestet. [26]

Abbildung 3.8 zeigt das V-Modell welches den Entwicklungsprozess auf Softwareebene beschreibt.

Weitere Standards und Vorschriften zur Entwicklung eingebetteter Systeme auf Softwareebene werden in [13, 21] beschrieben.

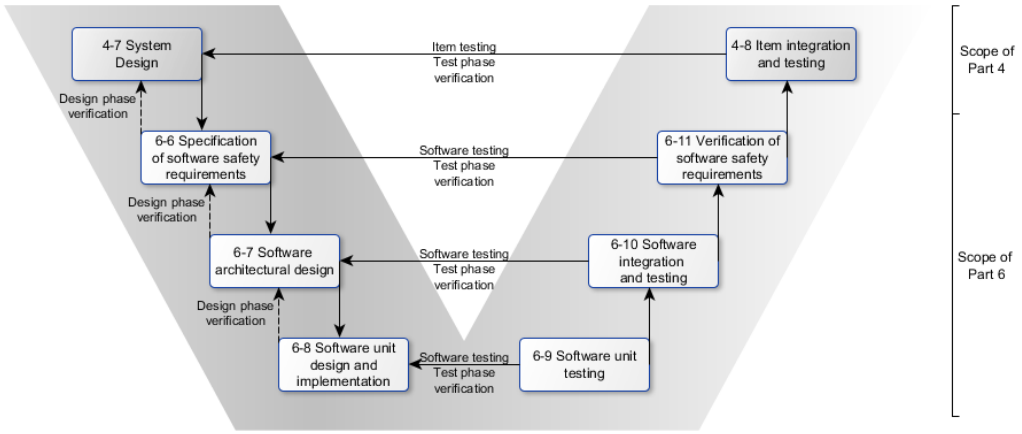


Abbildung 3.8: V-Modell zur Produktentwicklung auf Softwareebene gemäß [41]

Kapitel 4

Softwarestandards für automotive Echtzeitsysteme

Die steigende Komplexität der Fahrzeugelektronik fördert das Bestreben nach Standardisierung ebenso wie den Wunsch nach Kostensenkung. Im Softwarebereich versucht man durch Standardisierung eine Komponentenarchitektur zu ermöglichen, in welcher eine einfache Austauschbarkeit und Wiederverwendbarkeit der einzelnen Komponenten ermöglicht wird. Dazu müssen alle Schnittstellen und Kommunikationsmechanismen klar definiert werden. [71]

Eine Standardisierung hat bereits vor 1990 bei Bussystemen und Kommunikationsprotokollen vor allem hardwareseitig begonnen (Controller Area Network (CAN)[16]) und wurde allmählich auch für die Softwarearchitektur ein Thema. 1993 begann ein Gremium aus Automobilhersteller und -zulieferer unter dem Namen Offene Systeme für die Elektronik im Kraftfahrzeug (OSEK) mehrere Spezifikationen zu Betriebssystems-, Kommunikations- und Echtzeitverhalten zu erstellen. [Siehe Kapitel 4.1] 2003 wurde die darin begonnene Arbeit im Gremium für AUTOSAR fortgesetzt und an der Standardisierung der Steuergeräte-Softwarearchitektur und der Entwicklungs- und Testmethoden gearbeitet. Letzteres wurde in den letzten Jahren, getrieben durch den Architekturwandel in der Fahrzeugelektronik, stetig weiterentwickelt und enthält in der aktuellen Version (4.2) ebenfalls Spezifikationen für Mehrkern Plattformen. [82] [Siehe Kapitel 4.2]

4.1 OSEK/VDX

Das OSEK/VDX Konzept besteht aus mehreren Teilen, welche in Abbildung 4.1 dargestellt sind. Den wichtigsten davon bildet das Betriebssystemkonzept OSEK Operating System (OS) [63]. Dieses stellt ein ereignisgesteuertes Echtzeit-Multitasking-Betriebssystem dar, mit der Möglichkeit zur Task-Synchronisation sowie zur Ressourcenverwaltung. [27] Weiters definiert das Konzept die Interaktionsschicht OSEK Communication (COM) [33, Seite 4], welche sowohl den Datenaustausch zwischen Tasks im Steuergerät ermöglicht, als auch eine Kommunikation über ein Bussystem mit einem externen Steuergerät zur Verfügung stellt. Das Bussystem selbst wird in OSEK Network Management (NM) [33, Seite 5] beschrieben, welches unter anderem dafür Sorge trägt, wann sich welches Steuergerät ein- und ausschaltet. Um ein deterministisches System zu gewährleisten und um eine

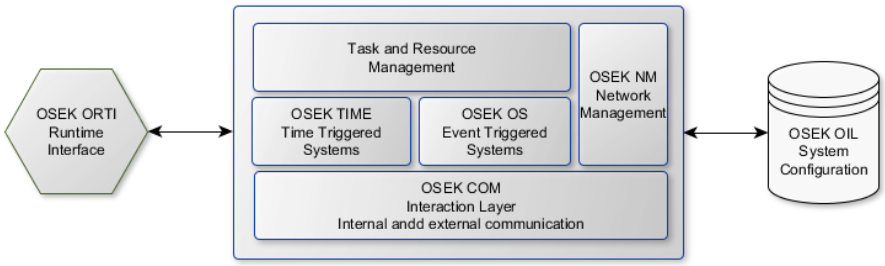


Abbildung 4.1: Bestandteile des OSEK/VDX Konzepts gemäß [82]

effiziente Implementierung mit geringem Bedarf an Rechenleistung und Speicherplatz zu ermöglichen, wird das gesamte System bereits in der Entwicklungsphase konfiguriert. Zu ebendieser Konfiguration wurde OSEK OSEK Interpration Language (OIL) [62] entwickelt. Diese Teile sowie das OSEK Glossary [33, Seite 1] und die OSEK Binding Specification [33, Seite 2] sind Bestandteil der ISO 17356 [33, Seite 6]. [82]

Die OSEK Spezifikationen geben nicht nur das Konzept selbst vor, sondern beschreiben detailliert die zugehörigen Mechanismen und Programmierschnittstellen („Application Programming Interface“ - API). Die Hauptbestandteile OS, COM und NM können unabhängig voneinander implementiert werden, was eine Anpassung an die jeweilige Zielhardware ermöglicht. Zusätzlich dazu werden in OSEK/VDX sogenannte Conformance Classes beschrieben, über welche der Implementierungsgrad der einzelnen Konzepte adaptiert werden kann.

Die ursprünglichen OSEK Spezifikationen sind in Anlehnung an das damals etablierte CAN Bussystem erfolgt, jedoch wurden auch die Nachteile solcher ereignisgesteuerten Systeme erkannt. Daher wurde die Spezifikation um ein zeitgesteuertes Betriebssystemkonzept namens OSEK Time, sowie um eine Kommunikationsschicht mit besserer Fehlertoleranz OSEK FTCOM (Fault Tolerant Communication) erweitert. Beide Konzepte fanden wenig Widerhall als OSEK Standard, wurden jedoch in den späteren AUTOSAR Konzepten weitergeführt.

Zur Erleichterung von Testwerkzeugen, wurde mit OSEK ORTI (OSEK Run Time Interface) eine Spezifikation für Schnittstellen zwischen dem Laufzeitprogramm im Steuergerät und Debuggern geschaffen. [82]

4.1.1 Der Betriebssystemkern OSEK/VDX OS

Der wichtigste Vorteil bei Verwendung eines Betriebssystems in eingebetteten Systemen ist die Möglichkeit, mehrere Aufgaben (Tasks) scheinbar parallel abarbeiten zu können. Das ist nötig, um z. B. die Regelung der Turboladerdrehzahl eines Motors bei gleichzeitiger Steuerung der Einspritzmenge und Zündungssteuerung zu ermöglichen. All diese Aufgaben haben eine unterschiedliche Periodendauer sowie verschiedene Ausführungszeiten. Hauptaufgabe des Echtzeitbetriebssystems ist es, jede dieser Aufgaben in einer Reihenfolge

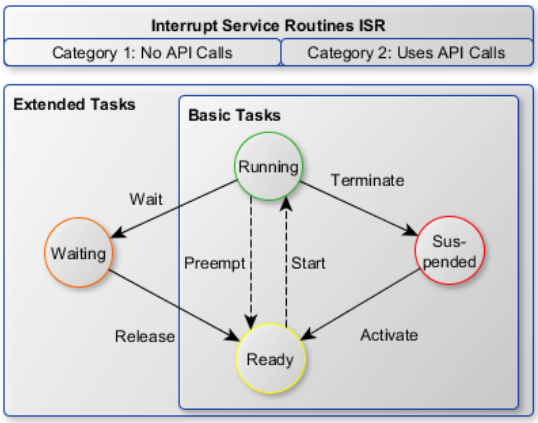


Abbildung 4.2: Task-Zustandsmodell im OSEK/VDX Konzept gemäß [82]

auszuführen und zwischen ihnen so umzuschalten, dass jeder einzelne Task in der benötigten Zeit abgearbeitet ist. (Echtzeit-Multitasking). Die dafür im Betriebssystem zuständige Komponente wird „Scheduler“ bezeichnet. [74]

Zustandsmodell

Die Tasks können sich zur Laufzeit in einem von vier Zuständen befinden. Zunächst sind alle Tasks im „Suspended“ Modus, von welchem aus sie vom Scheduler über einen Trigger in den „Ready“ Zustand wechseln und anschließend bei Verfügbarkeit aller Ressourcen im „Running“ Modus ausgeführt werden kann. Um auch eine externe Aktivierung der Tasks zu erlauben, wurde das Konzept um einen wartenden Zustand („Waiting“) erweitert, in welchem der Task seine Ressourcen freigeben kann, um von einem externen Aufruf fortgesetzt zu werden (z.B. vom Eintreffen einer Nachricht). In Abbildung 4.2 sind die einzelnen Zustände mit den Beziehungen zueinander in einem Zustandsdiagramm dargestellt.

Die Entscheidung welcher Task von mehreren bereiten Tasks ausgeführt wird, trifft in erster Linie der Scheduler. Diesem wird über verschiedene Taskprioritäten vorgegeben, in welcher Wichtigkeit die einzelnen Tasks zueinander stehen. Falls ein Task läuft und sich ein Task mit einer höheren Priorität im „Ready“ Zustand befindet, wird der aktuelle Zustand des laufenden Tasks gespeichert und dieser in den Bereit-Zustand versetzt, während der höherpriorisierte Task in den „Run“ Zustand versetzt wird.

Besitzen mehrere Tasks dieselbe Priorität, wird eine FIFO(First In First Out) Warteschlange verwendet, um zu garantieren, dass jeder Task in der Warteschlange in einer gewissen Zeit ausgeführt wird. Im allgemeinen Fall wird ein präemptives Schedulingverfahren verwendet, bei dem ein Task zu jeder Zeit vom Scheduler unterbrochen werden kann. Um dies an kritischen Stellen zu verhindern, sieht das Konzept das Blockieren des Schedulers mittels `GetResource(RES_SCHEDULER)` und anschließend die Freigabe mit

Quelltext 4.1: Verwendung der Task und ISR API

```
TASK(demoTask){
    demoReadInputs();
    demoControl();
    demoWriteOutputs();
    TerminateTask();
}

ISR(demoISR){
    demoAcknowlegdeInterrupt();
}
```

`ReleaseResource(RES_SCHEDULER)` vor. Präemptives Scheduling wird vor allem bei Tasks mit langen Laufzeiten verwendet, um höherpriorisierten, kurzen Tasks die Möglichkeit zu geben, mit geringer Verzögerung auf einen Aufruf durch den Scheduler bzw. einem externen Ereignis zu reagieren. Kurze Tasks werden häufig auch nichtpräemptiv implementiert um die Ausführungszeit durch die Unterbrechung vom Scheduler nicht unnötig zu verlängern. [82] Das Schedulingsystem nach mehreren Aspekten wie Priorität, Laufzeit, Kritikalität, WCET untersucht werden um eine entsprechend sicheres Laufzeitverhalten des Echtzeitsystems zu gewährleisten. [25]

Interrupts

Auch nichtpräemptive Tasks können durch Interrupts, welche durch Hardwaresignale am Mikrocontroller oder an einem Peripheriebaustein anliegen, unterbrochen werden. Dabei wird der Programmablauf durch sogenannte Interrupt Service Routine (ISR) unterbrochen und danach wieder fortgesetzt. Man unterscheidet im OSEK/VDX Konzept zwischen ISRs der Kategorie 1, in welchen keine Änderung des Schedulings vornehmen können, jedoch die Ausführungszeit kurz gehalten wird und nach welchen der unterbrochene Task fortgesetzt wird, und ISRs der Kategorie 2, welche auf eine beschränkte Betriebssystem API zugreifen können (z.B. um Tasks zu aktivieren) und im Zuge deren Ausführung auch der Scheduler aufgerufen wird, um zu einem höherprioren Task zu wechseln. So wie die Unterbrechung durch den Scheduler, können Tasks auch den Aufruf von Interrupt Service Routinen blockieren.

Im Quelltext werden die einzelnen Tasks und Interrupt Service Routinen mit dem Funktionskopf `TASK()` beziehungsweise `ISR()` definiert und können als normale Funktionen angesehen werden. Ein Task endet mit der Funktion `TerminateTask()`. Die Festlegung des Scheduling Verfahrens, sowie die benötigten Ressourcen, die Angabe ob es sich um einen Basic- oder Extended Task handelt und weitere Eigenschaften werden über die OIL Konfigurationsdatei beschrieben (siehe Anhang A.1) und über einen Generator in den für den Compiler verarbeitbaren Code übersetzt. In Quelltext 4.1 wird die Verwendung OSEK konformer Task und Interrupt Deklarationen gezeigt.

Ereignisse, Zähler und Alarme

Zur Synchronisation zwischen verschiedenen Tasks werden im OSEK/VDX Konzept Ereignisse verwendet. Diese werden je einem Task zugeordnet und auf solche können Tasks mittels `WaitEvent()` warten bis eine ISR oder ein anderer Task mittels `SetEvent()` ein Ereignis setzt. Dabei wird der Task wie oben beschrieben in den Wartezustand versetzt und im Falle eines Ereignisses vom Scheduler in den Ready Zustand gesetzt. Je nach Priorität wird dieser im Running Zustand fortgesetzt oder in die Warteschlange gegeben. Nachdem das Ereignis abgearbeitet wurde, wird mittels `ClearEvent()` das Ereignis zurückgesetzt und neue können gesetzt werden.

Ereignisse können zwar periodisch zur Ausführung von Tasks genutzt werden, jedoch können sich Tasks dadurch nicht selbstständig wiederholen. Dazu werden im Konzept Alarme verwendet. Diese werden über die OIL Konfigurationsdatei definiert und können periodisch Tasks, bzw. Ereignisse aufrufen. Das Auslösen der Alarme wird dabei über Zähler, welche entweder von Zeitgebern oder einer Funktion in einem Task erhöht werden, erfolgen.

Auch diese Zähler werden über die OIL Konfigurationsdatei definiert, wobei der maximale Zählerstand, die minimale Wiederholrate sowie weitere Parameter, wie ein zugrundeliegender Hardwarezähler (Timer) definiert werden.

Der Quelltext in Anhang A.1 zeigt eine exemplarische OIL Konfiguration. Dabei werden neben Betriebssystemeinstellungen alle Tasks, Zähler, Ressourcen, Ereignisse, Alarme sowie Interrupt Service Routinen inklusive deren Optionen eingestellt.

Im Konzept ist auch ein manueller Alarmaufruf mittels `SetRelAlarm()` bzw. `SetAbsAlarm()` möglich. Dabei wird der Zählerstand absolut bzw. relativ zum aktuellen Zählerstand als Parameter mitübergeben. Um den aktuellen Stand abzufragen wird `GetAlarm()` verwendet. Ein gestarteter Alarm kann mittels `CancelAlarm()` abgebrochen werden.

Ressourcen

Um Zugriffe auf ein Objekt wie Speicherbereiche, Hardwarekomponenten oder andere Module geordnet durchzuführen, werden im OSEK/VDX Konzept Ressourcen verwendet. Diese müssen zuerst vom Benutzer angefordert und exklusiv zugeteilt werden, bevor dieser auf die Komponente zugreifen darf (Mutual Exclusion). Verwenden mehrere Tasks mit unterschiedlichen Prioritäten dieselbe Ressource, kann das Problem aufkommen, dass ein niedrig priorisierter Task eine Ressource belegt und von einem höher-priorisierten Task unterbrochen wird, bevor dieser die Ressource freigeben kann. Abbildung 4.3 veranschaulicht die Problematik in welcher der hochpriorisierte Task solange unterbrochen wird, bis alle niedrigerpriorisierten, bereiten Tasks, inklusive dem Task mit der belegten Ressource, abgearbeitet sind.

Dieses Verhalten wird als Prioritätsinversion bezeichnet und behindert die zeitnahe Abarbeitung von Tasks in Echtzeitsystemen. Um dieses Problem zu lösen, verwendet das

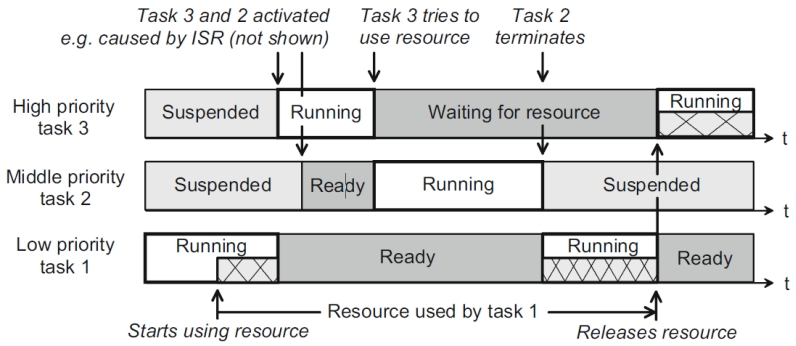


Abbildung 4.3: Ressourcenteilung mit Prioritätsinversion aus [82]

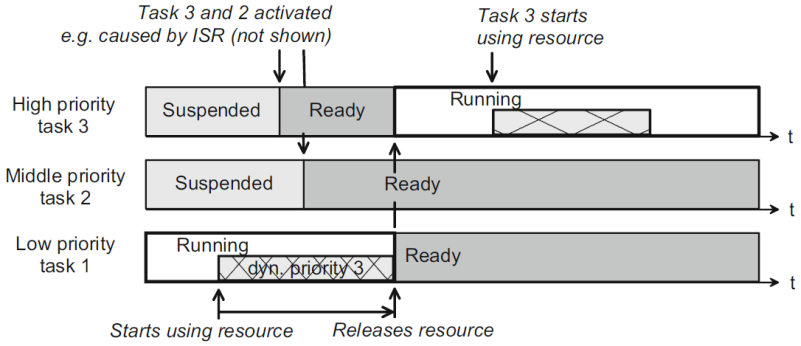


Abbildung 4.4: Ressourcenteilung mit dynamischer Prioritätsanhebung aus [82]

OSEK/VDX Konzept eine dynamische Prioritätsanhebung (Priority Ceiling). Dabei wird die Priorität eines niedrig priorisierten Tasks beim Erhalt einer Ressource soweit erhöht, dass ihn kein anderer Task welcher dieselbe Ressource benötigt, unterbrechen kann. Nach Freigabe der Ressource wird die Priorität des Tasks wieder auf den ursprünglichen Wert zurückgesetzt und ein höherpriorisierter Task kann die Ressource belegen. Ein Beispiel für eine dynamische Prioritätsanhebung ist in Abbildung 4.4 dargestellt. Dabei muss Task 3 zwar weiterhin warten, bis die Ressource von Task 1 freigegeben wird, jedoch kann Task 2 durch die erhöhte Priorität Task 1 nicht mehr verdrängen und verzögert Task 3 nicht zusätzlich.

Betriebssystemmodi

Das OSEK/VDX Konzept erlaubt es, mehrere Betriebssystemmodi zu definieren. Grundsätzlich werden alle Tasks, Alarme, Ereignisse usw. einem Default-Modus zugeordnet. Beim Start des Betriebssystems mittels `StartOS()` wird dieser Modus gestartet. Zu Entwicklungzwecken bzw. zum Debuggen oder um im Betrieb verschiedene Betriebssystemmodi zu erlauben, können weitere Modi angelegt werden, in welchen das Betriebssystem zusätzliche

Quelltext 4.2: Exemplarischer Ablauf des OS Startvorganges

```
main(){
    Init_BasisSW();
    Init_ApplicationSW();
    . . .
    while(!ExitCondition){
        switch (getAPPMode()){
            case 1:
                StartOS(OSDEBUGMODE);
                break;
            default:
                StartOS(OSDEFAULTAPPMODE);
                break;
        }
    . . .
    }
}
```

Tasks einbinden. Wurde das Betriebssystem in einem Modus nach einem Reset gestartet, muss zum Wechseln des Modus das Multitasking via `ShutdownOS()` zunächst gestoppt werden, bevor es in einem anderen Modus wieder gestartet werden kann. Quelltext 4.2 zeigt die Funktionen zum Start verschiedener Betriebsmodi.

Dieses Konzept des Anwendungsmodus erlaubt dem Betriebssystem auch, den Speicherplatz sowie die Laufzeit für die einzelnen Modi zu optimieren. [82]

Entwicklungsmodell

Um ein OSEK/VDX kompatibles Betriebssystem zu konfigurieren, werden OIL Konfigurationsdateien verwendet. Diese werden entweder manuell, oder im Idealfall über ein Konfigurationsprogramm erstellt. Daraus und aus einer OSEK Bibliothek wird über einen Generator zunächst ein für den Compiler nötiger Quelltext erstellt, in welchem alle notwendigen Betriebssystem-Objekte deklariert sind und aus welchem zusammen mit dem Quelltext der Applikations- und Basissoftware ein lauffähiges Programm erstellt wird. Um das Debuggen zu erleichtern, wird aus der OIL Datei eine OIL RealTime Interface (ORTI) Parameterdatei erstellt. Diese enthält Informationen z.B. über die einzelnen Tasks um ein übersichtliches Tracing zu ermöglichen. Der Entwicklungsprozess mit den einzelnen OSEK-Komponenten wird in Abbbildung 4.5 gezeigt.

Bei der Entwicklung mit einem OSEK/VDX Betriebssystem wird vorausgesetzt, dass der zeitrichtige Ablauf im System, die Speicherauslastung in Worst Case Situationen während der Entwicklungsphase mit Methoden und Werkzeugen ausreichend simuliert und getestet wurden. [43, 80]

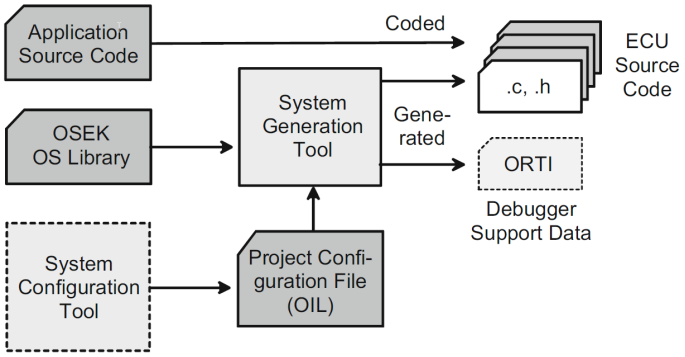


Abbildung 4.5: Bestandteile des OSEK/VDX Entwicklungsmodells aus [82]

4.2 AUTOSAR

Das OSEK Konzept sowie weitere Konzepte wurden von der Automotive Open Systems Architecture (AUTOSAR) Initiative übernommen und zu einer vollständige Softwarearchitektur für Steuergeräte integriert. Hauptziel davon ist, die Anwendungssoftware weitestgehend von der Hardware selbst zu entkoppeln und so zu ermöglichen, dass die einzelnen Komponenten durch standardisierte Schnittstellen völlig unabhängig voneinander entwickelt werden und anschließend per automatisiertem Konfigurationsprozess zu einem konkreten Projekt verbunden werden können. [44]

4.2.1 Architektur

Ausgehend von den Ebenen der OSEK Architektur werden die Basis Software, die Hardwareabstraktionsebene sowie ein sogenannter „Service Layer“, welcher das Betriebssystem, Kommunikationsprotokolle und eine Speicherverwaltung enthält, festgelegt. Über die AUTOSAR Run Time Environment werden diese darunterliegenden Ebenen mit der Anwendungssoftware selbst verbunden. Diese hohe Abstraktion soll auch ermöglichen, dass einzelne Applikationen („Applications“) auch auf unterschiedliche Hardware verschoben werden können, ohne dass sich funktionale Veränderungen ergeben. Zwar wird vom AUTOSAR Gremium nur eine Spezifikation erarbeitet, welche die eigentliche Implementierung den jeweiligen Herstellern überlässt, jedoch wird durch das komplexe System die Implementierung aber bereits stark vorgeschrieben. Abbildung 4.6 zeigt das Schichtenmodell der AUTOSAR Architektur mit den einzelnen Komponenten. [81]

4.2.2 Basissoftware

Die Basissoftware übernimmt im Groben Aufgaben zur Speicherverwaltung, Kommunikation, Hardware Ein- und Ausgabe sowie Systemdienste, wobei für jede Aufgabe mehrere Ebenen zuständig sind (siehe Abbildung 4.6). Um für echtzeitkritische Funktionen einen schnel-

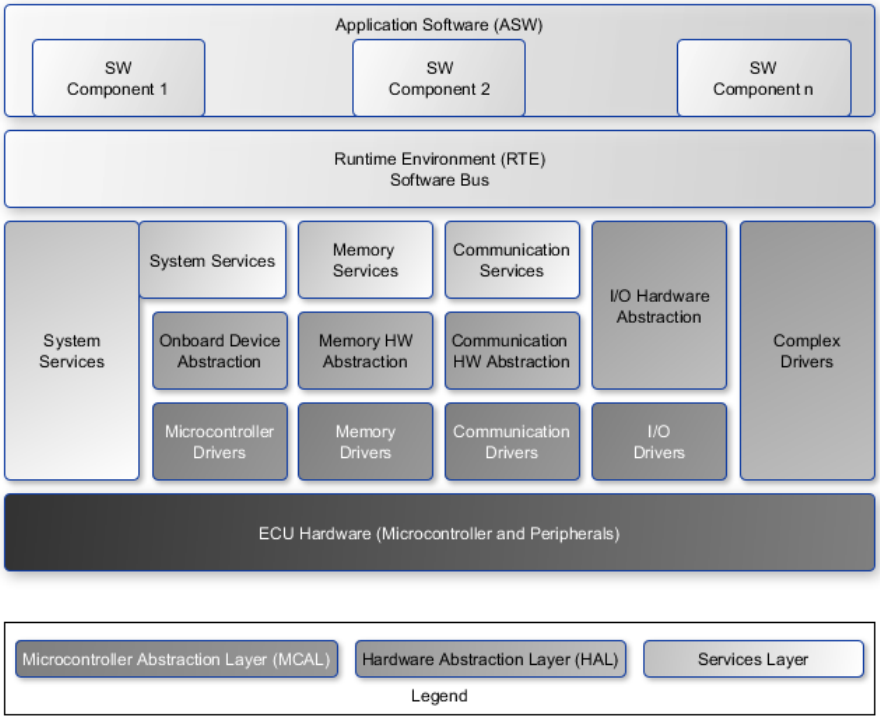


Abbildung 4.6: AUTOSAR Software Grundarchitektur in Steuergeräten gemäß [81]

len Zugriff auf die Hardware zu ermöglichen, stehen sog. Complex Drivers zur Verfügung, welche die einzelnen Schichten umgehen können. Als unterste Schicht wird die Schnittstelle zur Microcontroller-Hardware (sog. MicroController Abstraction Layer (MCAL))definiert. Diese wird vom Controller-Hersteller zur Verfügung gestellt und beinhaltet die Implementierung definierter Funktionen zur Ansteuerung der Recheneinheit selbst. Darüber befindet sich der Ebene zur Hardwareabstrahierung (Hardware Abstraction Layer (HAL)), welcher vom Steuergeräte-Hersteller geliefert wird, und eine Schnittstelle zu Peripherie-Controller liefert. AUTOSAR definiert einige Hardwaretreiber, darunter ADC-, PWM, SPI, GPT (General Purpose Timer), DIO (Digital Input/Output), Watchdog Treiber und mehr. Zur Umwandlung zwischen den hardwarenahen Registerwerten und logischen Signalen, welche der Anwendungssoftware zur Verfügung stehen, rechnet die HAL das Rohsignal in einen logischen Wert um, begrenzt es auf einen plausiblen Wertebereich und gibt ein Fehlersignal aus, falls nötig. Die Spezifikation unterscheidet dabei zwischen analogen, diskreten, pulsbreitenmodulierten sowie Fehler- Signalen.

Zusätzlich zu diesen Standardtreibern gibt es noch Treiber für Speicherbausteine für z.B. nichtflüchtige Speicher sowie Kommunikationscontroller. Auch Module für Selbsttestfunktionen sind definiert. Zur Fehlerbearbeitung wird ein sog. Diagnostic Event Manager verwendet. Dieser wird von den einzelnen Funktionen im Fehlerfall aufgerufen und stellt einen zentralen Fehlerspeicher dar. [81]

4.2.3 Funktionale Sicherheit

Zur Gewährleistung der funktionalen Sicherheit verwendet das AUTOSAR Konzept eine Ablaufüberwachung durch einen sog. „Watchdog Manager“. Dieser muss periodisch getriggert werden, um ein Auslösen eines sicheren Zustandes zu verhindern. Weiters sieht das Konzept vor, dass die Ausführungsdauer und -häufigkeit von Programmen, die Stackauslastung sowie Speicherzugriffe überwacht werden. [10] Auch Hardwarekomponenten sind durch geeignete Tests zu überprüfen. Wie bereits angesprochen, wird vor der Verarbeitung von Ein- und Ausgabesignalen eine Plausibilitätsüberprüfung durchgeführt. [5]

Um auch die Datenübertragung gegen Fehler zu schützen, können Daten über einen End-to-End Übertragungsschutz abgesichert werden. Dieser erweitert die eigentlichen Nutzdaten vor der Übertragung um einen Sequenzzählerwert, eine Prüfsumme, sowie der Inversion der Nutzdaten. [23] Auf Empfängerseite werden diese Prüfdaten vom eigentlichen Nutzsignal extrahiert und ein Abgleich zur Fehlererkennung durchgeführt. Zusätzlich kann durch Mitsenden eines Zeitwertes die Gültigkeit des Paketes festgestellt werden. [81]

Eine Häufig angewandte Testmethode zur Verifizierung der funktionalen Sicherheit stellt das Testen durch Fehlereinbringung dar. [7, 65]

4.2.4 Betriebssystem AUTOSAR OS

Das Betriebssystemkonzept von AUTOSAR ist kompatibel zu OSEK OS [siehe Kapitel 4.1.1] um eine Konvertierung von OSEK auf AUTOSAR zu erleichtern. [4] Ähnlich den

Conformance Classes bei OSEK, besitzt auch dieses Konzept ausgehend vom OSEK Konzept vier Ausbaustufen (sog. Scalability Classes), welche die Grundfunktionalität um Konzepte aus OSEK Time sowie Protected OSEK erweitert. Als Zusatzfunktionen werden darin Speicherschutzfunktionen sowie Zeitüberwachungen definiert. (Timing-Protection siehe [19])

Während bei OSEK OS ein ereignisgesteuertes Multitasking-Konzept verwendet wird, was bei vielen zeitgesteuerten Tasks zu einer Vielzahl an Alarmen führt, erweitert der AUTOSAR Standard dieses Konzept um sogenannte Schedule Tabellen. In diesen werden Taskaufrufe in Abhängigkeit eines Zählers definiert, der entweder an einen Hardware-Timer gekoppelt wird, oder als Softwarezähler implementiert wird. Schedule Tabellen können entweder periodisch, oder einmalig mit beliebigem Offset gestartet werden. Da die Tasks einer Schedule Tabelle mit anderen Tasks weiterhin konkurrieren, kommt der Prioritätsvergabe der einzelnen Tasks eine hohe Bedeutung zu.

Im Gegensatz zu OSEK OS kann aber durch die Zeitüberwachung ein Überschreiten von Ausführungszeiten erkannt werden sowie eine Synchronisation mit Buszyklen vorgenommen werden. Um eine Zeitüberschreitung zu detektieren, können für jeden Task die erlaubte Ausführungszeit (Execution Time Budget), die Wiederholrate (Inter-Arrival Rate) sowie die Ressourcensperrzeit (Locking Time) festgelegt werden. Wird eine Zeit überschritten, kann der Task terminiert, neugestartet oder auch das komplette Betriebssystem beendet werden. Als weiterer Überwachungsmechanismus ist die Stacküberwachung zu nennen, welche bei jeder Taskumschaltung eine Überschreitung des Stacks überprüft.

Ein weiteres Konzept, welches von OSEK Protected OS übernommen wurde, ist das der Anwendungsgruppen, in welchen zusammengehörige Tasks in einer „Application“ zusammengefasst werden und innerhalb dieser Objekte und Funktionen beliebig genutzt werden dürfen. Funktionen aus anderen Anwendungsgruppen dürfen nicht, beziehungsweise nur über spezielle Schnittstellen aufgerufen werden. Da die Überwachung derartiger Schutzmechanismen aufwendig ist, wird eine darüberliegende Anwendungsebene („Trusted Applications“) eingeführt. Dieser ist ein übergreifender Zugriff erlaubt, jedoch wird sie strengeren Zertifizierungsvorschriften unterworfen. [81]

Um die Auslastung des Systems durch Schutzmechanismen zu minimieren, werden diese vermehrt in Hardware ausgelagert. So können verschiedene Betriebsmodi einen Zugriff auf Funktionen wie Interruptsteuerung oder Hardwarezugriffe auf bestimmte Bereiche priviligierten Anwendungen wie dem Betriebssystem vorbehalten („Kernel Mode“), während die Anwendungssoftware im sogenannten User Mode beschränkten Zugriff erhält. Auf gleiche Weise können auch Speicherzugriffe durch „Applications“ von Kontrollmodulen wie Memory Protection Units (MPUs) auf gewisse Adressbereiche begrenzt werden. Eine Unterstützung derartiger Funktionen durch AUTOSAR ist jedoch erst zum Teil umgesetzt worden. [81]

Während die Konfiguration des Betriebssystems bis AUTOSAR R3.0 mittels OSEK konformen OIL Dateien erfolgte, setzt der Standard für spätere Versionen XML-basierte Konfigurationsdateien ein. [44]

4.2.4.1 Mehrkernunterstützung

Ab AUTOSAR 4.0 wird auch die Verwendung von mehreren Rechnerkernen in einem Steuergerät unterstützt. [6] Dabei sieht das Konzept vor, dass jede CPU eine eigene Instanz des Betriebssystems ausführt auf welcher statisch konfigurierte Tasks ausgeführt werden. Lediglich beim Start der einzelnen Instanzen erfolgt eine zeitliche Synchronisation, danach erfolgt das Scheduling unabhängig voneinander. Die Betriebssysteme können ab dieser Version auch Objekte wie Tasks, Ereignisse oder Alarme auch auf anderen Kernen steuern. Ressourcen werden jedoch weiterhin rein lokal verwaltet. Um den Zugriff auf Ressourcen zwischen den einzelnen Instanzen zu regeln, wird ein Synchronisationsmechanismus eingeführt, welcher sogenannte Spinlocks verwendet. Mit deren Hilfe werden einzelne Ressourcen entweder mittels Busy-Wait-, oder mittels Try-and-Resume-Strategie belegt und nach Abarbeitung wieder freigegeben.

Zum Datenaustausch zwischen zwei Kernen können gemeinsame Speicherbereiche über sogenannte Inter-OS-Application Communication (IOC) Funktionen genutzt werden. Diese stellen eine Sender-Empfänger Kommunikation zur Verfügung, welche unabhängig von der Zuordnung einzelner Tasks auf deren Kerne funktioniert. [81]

Um Ressourcenkonflikte zu vermeiden wird von AUTOSAR empfohlen, jeglichen Peripheriezugriff von einem Master-Kern aus zu steuern. Dieser soll auch als einziger die Basis-Software (BSW) Ebene zum Zugriff auf die Peripherie integrieren. Der Zugriff durch Kerne wird über die IOC Schnittstelle ermöglicht. Dadurch soll der gleichzeitige Zugriff auf eigentlich geteilte Hardware verhindert werden. [24]

Eine Übersicht zu AUTOSAR konformen Mehrkern-Betriebssystemen findet sich in [14].

Die Erfahrungen bei der AUTOSAR konforme Entwicklung unter Verwendung eines Mehrkernsystems findet sich in [9]. Dabei wurden mehrere Instanzen des Betriebssystems verwendet, mit dem Ziel nach maximaler Isolation und Unabhängigkeit voneinander.

Kapitel 5

Das E-Gas Sicherheitskonzept

Das standardisierte E-Gas Überwachungskonzept wurde von einem Arbeitskreis mehrerer Automobilhersteller[1] in Zusammenarbeit mit Steuergäteherstellern erstellt und stellt die Standardisierung eines „Drive-by-Wire“ Systems [67] dar. Ziel war es ein Überwachungskonzept zu entwickeln, in welchem sowohl Funktion, als auch Funktionsüberwachung sowie das Steuergerät selbst kontinuierlich überprüft werden um etwaige Fehler zu erkennen und ein sicheres System zu gewährleisten. Nach Inkrafttreten der ISO 26262 wurde die Spezifikation dahingehend analysiert und der Norm entsprechend angepasst. [3]

Die darin verwendete Methode zur Überprüfung der Betriebssicherheit und Zuverlässigkeit von Software-basierten elektronischen Systemen wird in [83] behandelt.

5.1 Entwicklungsleitlinien

Das Konzept wurde nach dem Grundsatz erstellt, dass der Personenschutz die höchste Priorität hat. Die Überwachung des Systems sollte weitestgehend unabhängig sowohl von der Funktion selbst, als auch von externen Einflüssen wie dem Fahrerwunsch sein, um eine hohe Zuverlässigkeit zu gewährleisten. Weiters soll das System so ausgelegt sein, dass Einzelfehler (auch in Verbindung mit schlafenden Fehlern) sowie Doppelfehler zu beherrschbaren Systemreaktionen führen. Um dies zu gewährleisten, muss ein Fehlersignal eindeutig erkennbar sein. Für jegliche Fehlerreaktion muss auch eine Rücksetzung dieser festgelegt sein. Falls keine andere Reaktion zur Beherrschung des Systems bleibt, darf ein Motorstopp durchgeführt werden. Da die Verfügbarkeit auch nach Ausfall einzelner redundanter Bauteile noch erhalten bleiben soll, muss dem Fahrer der Verlust dieses Sicherheitsnetzes über optischem Wege oder die Änderung des Fahrverhaltens mitgeteilt werden. Um die Wirksamkeit der redundanten Abschaltpfade überprüfen zu können, sollten diese einmal pro Fahrzyklus getestet werden. Besonders die Rechnerüberwachung als auch die Abschaltpfade sind möglichst einfach und robust zu gestalten und deren Spannungsversorgung ist auf fehlerhaftes Verhalten zu überprüfen. [3]

[1] Mitglieder des E-Gas Arbeitskreises: Audi AG, BMW AG, Daimler AG, Porsche AG, VW AG

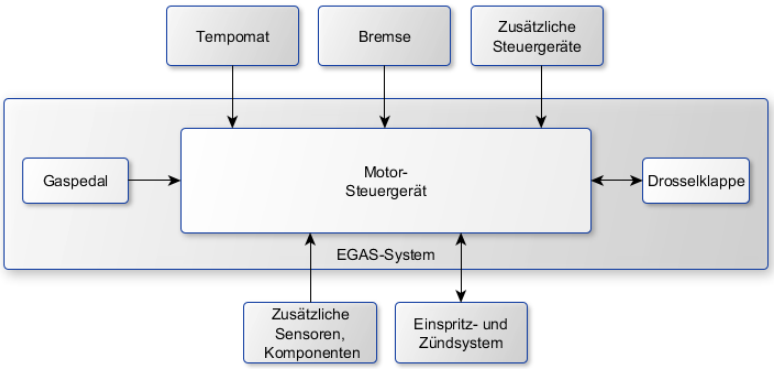


Abbildung 5.1: E-Gas Systemdefinition gemäß [3]

5.2 System Definition

Um den Bedingungen der ISO26262 zu entsprechen, wird zunächst der Systemumfang definiert: Betrachtet wird ein Verbrennungsmotor eines Fahrzeuges, welcher bei geschlossenem Antriebsstrang ein Moment auf die Räder ausübt. [3]

Dabei wurden dem Verbrennungsmotor folgende Funktionsmerkmale zugeordnet: Das Bereitstellen eines Antriebsmoments sowie das Bereitstellen eines Verzögerungsmoments. Als Anwendungsbereich wurde der PKW identifiziert. Der Aufbau ist wie folgt definiert: Als einzige Quelle für ein Antriebsmoment dient der Verbrennungsmotor, welcher über den geschlossenen Antriebsstrang direkt mit den Antriebsrädern verbunden ist. Die Ansteuerung sowie die Kontrolle des Motors erfolgt mittels Motorelektronik. [3]

Abbildung 5.1 zeigt die einzelnen Komponenten eines Systems zur Motorsteuerung, welche Geschwindigkeits- und Momentengeber, als auch die Bremse sowie Aktuatoren und Sensoren zur Motorsteuerung beinhaltet.

Für den Umfang des Konzepts wurden die folgenden Komponenten identifiziert:

- Fahrpedal
- Motorsteuergerät
- Drosselklappe

5.3 Gefahren- und Risikoanalyse

Ziel der Gefahren und Risikoanalyse ist es, durch Systemfehler potentielle Gefahren für die Person zu erkennen um im Anschluss daran Sicherheitsziele und Entwicklungsvorschriften abzuleiten. Für das E-Gas System wurden die Sicherheitsziele wie in Tabelle 5.1 definiert. [3]

Tabelle 5.1: Sicherheitsziele der Gefahren- und Risikoanalyse des E-Gas Systems aus [3]

N°	Sicherheitsziel	ASIL
SZ-01	Vermeidung ungewollter Beschleunigung	ASIL B
SZ-02	Vermeidung ausbleibender Beschleunigung	QM
SZ-03	Vermeidung ungewollter Verzögerung	QM
SZ-04	Vermeidung ausbleibender Verzögerung	QM

Als höchstes Sicherheitsziel wurde „SZ-01“ definiert, welches mit dem ASIL B versehen wurde. Die übrigen Sicherheitsziele des E-Gas Systems wurden aufgrund der Beherrschbarkeit, sowie dem geringen entstehenden Risiko nur als Qualitätsmanagement (QM) eingestuft, was bedeutet, dass in der Umsetzung ein entsprechender Entwicklungsprozess genügt. [3]

5.4 Funktionales Sicherheitskonzept

Im funktionalen Sicherheitskonzept wurde lediglich das höchste Sicherheitsziel - die ungewollte Beschleunigung - betrachtet, welche bei einer Momentenquelle nur durch eine fehlerhafte Momentenvorgabe beziehungsweise -umsetzung entstehen kann. [3]

Dieses ist in Tabelle 5.2 dargestellt:

Tabelle 5.2: Funktionales Sicherheitskonzept des E-Gas Systems gemäß [3]

N°	Komponente	Funktionales Sicherheitskonzept
FSK-01.00	Sensorik	Die Sensorsignale sind nach Signalerfassung plausibilisierbar.
FSK-02.00	Aktuatorik	Die Aktorsignale sind nach Signalerfassung plausibilisierbar.
FSK-03.00	Steuergerät	Das Motorsteuergerät erkennt Fehler an der Sensorik.
FSK-03.01	Steuergerät	Das Motorsteuergerät erkennt Fehler an der Aktuatorik.
FSK-03.02	Steuergerät	Im Motorsteuergerät ist ein Sicherheitskonzept implementiert, welches das Stellen eines unzulässigen hohen Antriebsmoments erkennt, bestätigt und als Fehlerreaktion das System in einen sicheren Zustand schaltet.
FSK-03.03	Steuergerät	Das Sicherheitskonzept benutzt die Idee einer zentralen, funktionalen Überwachung (Ebene 2).
FSK-04.00	Zentrale funktionale Überwachung	In der funktionalen Überwachungsebene (Ebene 2) wird die zu überwachende Funktion unabhängig von der Funktionsebene (Ebene 1) berechnet, überwacht und im Fehlerfall ein beherrschbarer Zustand hergestellt.
FSK-04.01	Zentrale funktionale Überwachung	Eine unabhängige Entwicklung stellt sicher, dass systematische Fehler sich nicht auf die Funktionsebene (Ebene 1) und die Überwachungsebene (Ebene 2) auf gleiche Weise auswirken.

FSK-04.02	Zentrale funktionale Überwachung	Zusätzliche Maßnahmen sind im Steuergerät zu implementieren, um die Integrität der verwendeten Steuergeräte-HW zu überprüfen und um sicherzustellen, dass Fehler in der Ebene 1 als auch Steuergeräte-HW-Fehler nicht unentdeckt auf Ebene 2 einwirken können.

Um dieses Sicherheitsziel zu erreichen, wird die Überwachung der Einhaltung einer zulässigen Fahrzeugbeschleunigung bzw. eines zulässigen Antriebsmomentes vorgesehen. Im Fehlerfall soll das Fahrzeug in einer gewissen Toleranzzeit in einen sicheren Zustand überführt werden. [3]

Anschließend werden in Tabelle 5.3 die Sicherheitsanforderungen genauer spezifiziert:

Tabelle 5.3: Sicherheitsanforderungen des E-Gas Systems gemäß [3]

N°	Komponente	Sicherheitsanforderung	SZ
SANF-01	Gaspedal	Sensoren werden plausibilisierbar ausgeführt	SZ-01
SANF-02	Drosselklappe	Sensoren werden plausibilisierbar ausgeführt	SZ-01
SANF-03	Motorsteuergerät	Das Motorsteuergerät erkennt Fehler in der Sensorik durch geeignete Plausibilisierung	SZ-01
SANF-04	Motorsteuergerät	Im Motorsteuergerät werden momentenbeeinflussende Anforderungen anderer Steuergeräte im Signalverbund abgesichert.	SZ-01
SANF-05	Motorsteuergerät	Das Motorsteuergerät erkennt Fehler in der Aktuatorik durch geeignete Plausibilisierung.	SZ-01
SANF-06	Motorsteuergerät	Im Motorsteuergerät ist ein Sicherheitskonzept implementiert, das ungewolltes Stellen eines zu hohen Antriebsmoments bzw. eine ungewollte Beschleunigung erkennt, bestätigt und als Fehlerreaktion in einen sicheren Zustand schaltet.	SZ-01
SANF-07	Motorsteuergerät	Der Funktionsrechner ist zu überwachen.	SZ-01

5.5 Technisches Sicherheitskonzept

Die Überwachung des Systems erfolgt in drei Ebenen, welche in Abbildung 5.2 dargestellt sind:

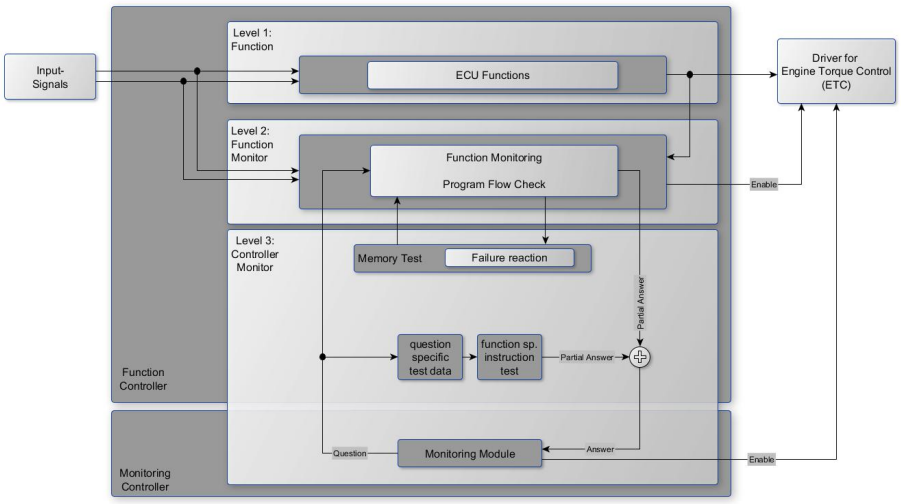


Abbildung 5.2: Übersicht des 3 Ebenen Konzepts gemäß [3]

Ebene 1

Die unterste Ebene wird als Funktionsebene bezeichnet. Diese beinhaltet Motorsteuerungsfunktionen wie das Umsetzen der angeforderten Motormomente, die Komponentenüberwachung sowie die Diagnose der Ein- und Ausgangsgrößen. Weiters übernimmt sie die Steuerung der Systemreaktionen im erkannten Fehlerfall. [3]

Ebene 2

In Ebene 2 erfolgt die Funktionsüberwachung. Hier wird der fehlerhafte Ablauf der Funktionssoftware in Ebene 1 durch Überwachung der berechneten Ausgangsgrößen erkannt und im Fehlerfall eine Systemreaktion eingeleitet. [3]

Ebene 3

Diese Ebene wird als Rechner-Überwachungsebene bezeichnet. Durch ein vom Funktionsrechner unabhängiges Überwachungsmodul wird ein Frage-Antwort-Verfahren verwendet, um den ordnungsgemäßen Ablauf der Programmbefehle des Funktionsrechners zu testen. Im Fehlerfall wird eine Systemreaktion vom Überwachungsrechner eingeleitet. [3]

5.5.1 Ebene 1: Funktion

Ebene 1 beinhaltet sämtliche Motorsteuerungsfunktionen, als auch die Diagnose sicherheitsrelevanter Ein- und Ausgangsgrößen.

Sensorkomponenten

- Pedalwertgeber
- Bremsschalter
- Drehzahlsignal
- Lastsignal
- Lambdasonde
- Motortemperaturgeber

Stellgliedkomponenten

- Drosselklappe
- Kraftstoffeinspritzabschaltung
- Raildruckregelventil
- Zumesseinheit

Signalpfade im Steuergeräte-Systemverbund

- Empfangene momenterhöhende Eingriffe (sind vom Sender-Steuergerät abzusichern)

Abgesicherte Abschaltpfade der Fahrgeschwindigkeitsregelung

- Bremsinformation

Anforderungen an die Drosselklappe

Die Sensorik der Drosselklappe soll mit einem Doppel-Sensor mit physikalisch getrennten Signalpfaden ausgestattet sein, wobei eine hohe Diagnosesensibilität im kompletten Verstellbereich gefordert wird. Weiters sollte eine hohe Auflösung sowie eine geringe Gleichlaufabweichung für gute Regelgenauigkeit und Diagnose sorgen. Um den Anforderungen im automotiven Bereich gerecht zu werden, wird ein Geringer Drift über Umgebungs- und Lebensdauerbedingungen (Einhaltung der Diagnosegrenzen) gefordert.

Es sollten durch geeignete Maßnahmen Kurz-, Nebenschlüsse und Unterbrechungen an der Drosselklappen-Sensorik (einschließlich Sensorversorgung) sowie Kurzschlüsse und Unterbrechungen am Drosselklappenantrieb als Fehler erkannt werden.

Anforderungen an den Pedalwertgeber

Auch für den Pedalwertgeber wird ein Doppel-Sensor mit physikalisch getrennten Signalpfaden gefordert, welcher entweder eine diagnostizierbare Sensor-Versorgungsspannung

oder zwei Sensor-Versorgungsspannungen besitzt. Um eine Redundanz zu ermöglichen, müssen auch die Sensor-Masseleitungen bis ins Steuergerät getrennt geführt werden. Auch eine eindeutige Plausibilisierung im gesamten Verstellbereich wird gefordert, wobei vom Konzept auf eine Ausführung mit steigenden Kennlinien unterschiedlicher Steigung verwiesen wird. Um den Anforderungen im automotiven Bereich gerecht zu werden, wird eine geringe Gleichlaufabweichung und ausreichende Auflösung für wirksame Diagnose geringe Drift über Umgebungs- und Lebensdauerbedingungen gefordert.

Eine Fehlererkennung von Kurzschlüssen, Nebenschlüssen und Unterbrechungen an der Fahrpedalwertgeber-Sensorik muss erkennbar sein. Die Sensoreingangsbeschaltungen sind so festzulegen, dass bei Leitungsunterbrechung ein Spannungspegel unter der Leerlauferkennungsschwelle entsteht.

Zur Ermittlung der Fahrpedalvorgabe wird zunächst die Sensorkennlinie von Kanal 2 auf die von Kanal 1 um-normiert. Anschließend erfolgt die Berechnung ausschließlich mit dem Minimalwert der beiden Sensorkanäle. Auf dieselbe Weise wird der Wert der Betriebsbremsen-Sensorik aufgenommen und ausgewertet. Wird im fahrenden Betrieb über das Gaspedal eine Antriebsleistung vorgegeben und zugleich ein Minimaldruck auf der Betriebsbremse festgestellt, wird die Antriebsleistung auf eine beherrschbare Maximalgrenze begrenzt. Die Betriebsbremse hat dem Gaspedal gegenüber höhere Priorität.

5.5.2 Ebene 2: Funktionsüberwachung

Ebene 2 übernimmt die Überwachung der leistungsbestimmenden Funktionen von Ebene 1. Darunter fällt der Momentenvergleich zwischen den autark gebildeten Berechnungsgrößen aus zulässigem Motormoment und Ist-Moment, bzw. der Vergleich von zulässiger Fahrzeugbeschleunigung und Ist-Beschleunigung (Abbildung 5.3). Falls Ebene 2 nicht autark eine Fehlerreaktion auslösen kann, muss die Überwachung der Fehlerreaktion von Ebene 1 ebenfalls überwacht werden, genauso wie eine zyklische Überwachung der eigenen Speicherbereiche und Rechenoperationen für die Programmablaufkontrolle.

Im Detail müssen die folgenden Komponenten überwacht werden:

5.5.2.1 Signalpfade im Steuergeräte-Systemverbund

- Inhalt der gesendeten überwachungsrelevanten Umfänge

5.5.2.2 Eingangsgrößen

- Fahrpedal
- Bremse
- Externe momentenerhöhende Eingriffe
- Luftmasse (als Hauptlastsignal)
- Saugrohr-Druck (als Hauptlastsignal)

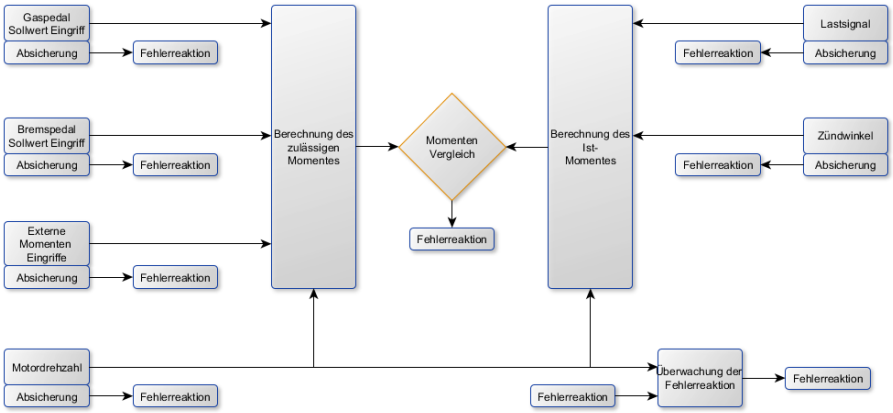


Abbildung 5.3: Funktionsüberwachung in Ebene 2 gemäß [3]

- Motordrehzahl
- Zündwinkel

5.5.2.3 Funktionsüberwachung

- Momentenvergleich (zulässiges Moment mit Istmoment)
- Abschaltpfadtest (bis Aktuatorendstufe)
- Systemreaktion (EMB) der Ebene 1 im Fehlerfall
- A/D Wandlertest
- Plausibilisierung Verlustmoment aus Ebene 1
- Plausibilisierung der Adaptions-/Korrekturwerte aus Ebene 1
- Abschaltung der Fahrgeschwindigkeitsregelung bei Bremseingriff
- Überwachung Gas/Bremse Plausibilität

5.5.2.4 Reaktionen im Fehlerfall (fehlerspezifisch)

- Reset
- Abschaltung Aktuatorendstufen
- Leistungsbegrenzender Ersatzbetrieb

Werden Größen von Ebene 1 in Ebene 2 übernommen, müssen diese auf Einhaltung von zulässigen Grenzen geprüft werden. Befinden sie sich außerhalb der Grenzen, muss eine Fehlerreaktion eingeleitet werden.

Um Ausgabefehler in Ebene 1 zu erkennen, müssen alle sicherheitskritischen Ausgabegrößen in Ebene 2 eingelesen, und mit den Sollwerten aus Ebene 1 verglichen werden um

z.B. gekippte RAM-Zellen der Ausgabe-Einheit erkennen zu können. Pro Rechenzyklus ist mindestens ein Zylinder zu prüfen. Weiters ist durch geeignete Maßnahmen sicherzustellen, dass ein unzulässiges Kopieren von Eingangsgrößen in den Ausgabespeicher erkannt wird. Die Absicherung der Momentenausgabegrößen im Steuergeräteverbund ist projektspezifisch festzulegen.

5.5.3 Ebene 3: Rechnerüberwachung

Durch die Rechnerüberwachung sollen fehlerhafte Operationen des Funktionsrechners (Rechnerkern, betroffene Bereiche im RAM/ROM) erkannt und Auswirkungen auf die Sicherheit verhindert werden. Pro Fahrzyklus müssen mindestens einmal die Speicherbausteine RAM/ROM überprüft werden und im Fehlerfall ist ein wiederholtes Initialisieren durchzuführen. Erst nach Abschluss dieser Überprüfung darf ein Motorstart eingeleitet werden. Ebene 3 besteht aus zwei Grundelementen:

Ein physikalisch unabhängiges Überwachungsmodul (E3_ÜM)

Dieser Rechner (bzw. ASIC) muss sowohl in der Spannungsversorgung, als auch in der Taktversorgung vom Funktionsrechner unabhängig sein. Weiters wird eine thermische Unabhängigkeit gefordert. Das Überwachungsmodul stellt der Überwachungssoftware im Funktionsrechner über eine geeignete Schnittstelle zyklisch eine Frage aus einer Menge von mindestens zehn diversitären Fragen, überwacht den Empfang eines zyklischen Prüfergebnisses, bewertet dieses und leitet im Fehlerfalle eine Fehlerreaktion ein. Falls das Überwachungsmodul RAM oder ROM Bausteine verwendet, müssen diese zyklisch geprüft werden.

Die Überwachungssoftware im Funktionsrechner (E3_SW)

Eine Überwachungssoftware übernimmt im Funktionsrechner die Kommunikation zum Überwachungsmodul. Diese empfängt eine Frage, woraus über mehrere Testpfade im Funktionsrechner eine Antwort abgeleitet wird. Dabei liefert jeder Testpfad ein exakt definiertes, Frage-abhängiges numerisches Teilergebnis. Das daraus berechnete Gesamtergebnis wird anschließend an das Überwachungsmodul gesandt. Somit signalisiert die Überwachungssoftware dem externen Modul den fehlerfreien Betrieb.

5.5.3.1 Überwachung durch das Überwachungsmodul

Das Überwachungsmodul (E3_ÜM) erwartet von der Überwachungssoftware (E3_SW) im Funktionsrechner innerhalb eines definierten Zeitfensters eine genau definierte Antwort. Tritt ein Fehler auf, erhöht das Überwachungsmodul einen internen Fehlerzähler und wiederholt die falsch beantwortete Frage. Überschreitet jedoch die Anzahl der Falschantworten eine definierte Grenze, schaltet das externe Modul die leistungsbestimmenden Aktuator-Endstufen ab und löst über den Funktionsrechner eine begrenzte Anzahl von SW-Resets aus. Dieselbe Fehlerreaktion wird bei einem falschen Antwortzeitpunkt ausgelöst. Weiters

ist die Fehlerzählerbehandlung im Überwachungsmodul so auszulegen, dass Zustände der Fehlererkennung schneller zum Reaktionsschwellwert führen, als dass erkannte fehlerfreie Zustände zur „Fehlerzählerheilung“ führen. Um Unabhängigkeit zum Funktionsrechner zu gewährleisten, darf das Überwachungsmodul nicht den Entwicklungs- und Änderungszyklen des Steuergerätes unterworfen werden. So sollen auch die Fragen daraus einheitlich und bereits bei der Festlegung des Motorsteuerungs-Systems bestimmt werden. Projektspezifische Eigenheiten sollen durch Bedatung von Parametern im Funktionsrechner angepasst werden. Um eine ausreichende Quantisierung für die Fehlerentprellung sicherzustellen, soll eine Wiederholrate der Frage/Antwortkommunikation einen Grenzwert von 80ms nicht überschreiten.

5.5.3.2 Überwachung durch die Überwachungssoftware im Funktionsrechner

Ebene 3 erwartet im Funktionsrechner innerhalb eines definierten Zeitfensters eine Frage und überprüft die Funktion des Überwachungsmoduls. Dies geschieht, indem die Überwachungssoftware in bestimmten Zeitintervallen gezielt eine falsche Antwort sendet. Daraufhin muss das Überwachungsmodul die Frage zusammen mit dem Fehlerzählerstand senden. Stimmen diese Werte nicht mit den erwarteten überein, wird in der Überwachungssoftware ein Fehlerzähler erhöht, der bei Überschreiten eines Grenzwertes zu einer Fehlerreaktion führt. Dabei werden die Aktuatorendstufen abgeschalten und eine begrenzte Anzahl an Resets durchgeführt.

Rechnerüberprüfung

Die Gesamtantwort der Überwachungssoftware wird aus mehreren Testpfaden gebildet. Die Programmablaufkontrolle bildet den ersten Pfad. Dabei wird geprüft, ob alle Programmmodule, die für die Überwachung relevant sind, in festen Zeitrastern und korrekter Reihenfolge abgearbeitet werden. Den zweiten Pfad bildet der funktionsspezifische Befehlssatztest. Dieser ermöglicht die Erkennung von Fehlern im Rechnerkern und in der Abarbeitung von Funktionen in Ebene 2 und muss an diese angepasst werden. Dafür werden sicherheitsrelevante Befehle aus Ebene 2 in einen eigenen Speicherbereich in Ebene 3 kopiert und diese auf Funktionsfähigkeit geprüft. Alternativ ist ein automatisch generierter Steuergeräte-Operationscode-Test zulässig, wenn er alle Befehlssätze abdeckt, die in den Überwachungsebenen 2 und 3 verwendet werden.

Überwachung selbstständig lauffähiger Hardwarebausteine

Selbstständig lauffähige Hardwarebausteine wie eine Timing Processing Unit, die einen Einfluss auf sicherheitsrelevante Signale haben können, müssen ebenfalls von der Überwachungssoftware überprüft werden. Dabei sollten die folgenden Fehler entdeckt werden können:

- Fehlerhafte Zellen des internen Parameterspeichers
- Konflikte im Datenfluss bei Systemen mit gemeinsam genutzten Speicherbereichen
- Unplausibilitäten in Berechnungsgrößen des Hardwarebausteins

Dazu bieten sich mehrere Tests an:

- Beschreibbarkeitstest des Parameterspeichers (z.B. internes Parameter-Ram)
- Speichertest des Programmspeichers (z.B. Programm-Ram zyklisch, ROM einmal pro Fahrzyklus)
- Überwachung wird in die Programmablaufkontrolle miteinbezogen.
- Plausibilisierung charakteristischer Berechnungsgrößen des Moduls im Funktionsrechner

Wird ein Fehler im Hardwarebaustein erkannt, wird vom Funktionsrechner ein Reset ausgelöst.

Abschaltpfadtest

Um die Funktion aller Abschaltpfade zu testen, wird einmal pro Fahrzyklus die Wirksamkeit der verwendeten Sicherheitselemente getestet. Bei einem negativen Prüfungsergebnis muss dieser Test zwingend wiederholt werden. Jedoch ist ein Motorbetrieb zulässig, wenn mindestens ein Abschaltpfad pro Rechner mit positivem Ergebnis getestet worden ist.

A/D-Wandlertest

Eine Überprüfung des A/D-Wandlers ist nötig, sobald sicherheitsrelevante Signale analog eingelesen werden. Dabei müssen Steigungsfehler, Offsetfehler sowie die Registerunbeweglichkeit des Wandlers wie auch ein Fehler am Multiplexer erkannt werden.

5.5.3.3 Systemverhalten bei einem Reset

Ein Reset wirkt sowohl auf das Überwachungsmodul, als auch auf den Funktionsrechner. Weiters werden alle leistungsbestimmenden Endstufen abgeschaltet. Wie lange der Reset-Status beibehalten wird, ist projektspezifisch festzulegen. Nach einem Reset müssen die gespeicherten Informationen über die Ursache eines Software-Resets bewertet werden und für nichtig erklärt werden, bevor ein erneuter Motorstart möglich ist. So ist z.B. bei einem erkannten RAM/ROM Fehler der gesamte überwachungsrelevante Speicherbereich zu prüfen, bevor eine Freigabe erfolgt. Die Überprüfung des übrigen Speicherbereichs kann während dem Fahrzyklus zu Ende geführt werden. Ist die maximale Anzahl an Software-Resets überschritten, bleiben die leistungsbestimmenden Endstufen bis zum Neustart durch den Fahrer stromlos abgeschaltet. Die Synchronisation zwischen Überwachungsmodul und Funktionsrechner erfolgt bei einem Reset durch eine definierte Sequenz in der Frage/Antwort-Kommunikation. Daran gekoppelt werden Ansteuer- und Prüfabläufe für die Abschaltpfade von beiden Rechnereinheiten. Bei erfolgreicher Prüfung werden die leistungsbestimmenden Endstufen freigegeben.

Abbildungen 5.4 und 5.5 zeigen die Reaktionen auf Fehler in Ebene 3 wobei nach Programmablaufsfehler und Speicherfehler unterschieden wird:

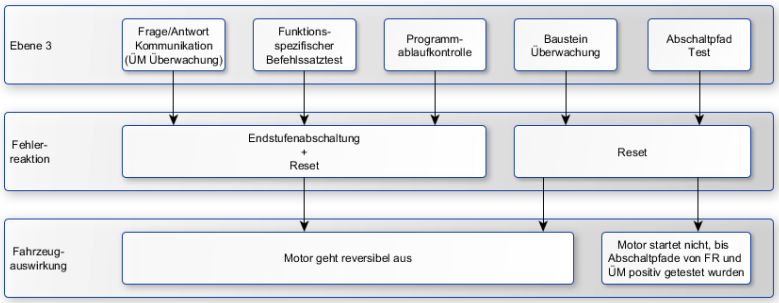


Abbildung 5.4: Fehlerreaktionen der Rechnerüberwachung in Ebene 3 gemäß [3]

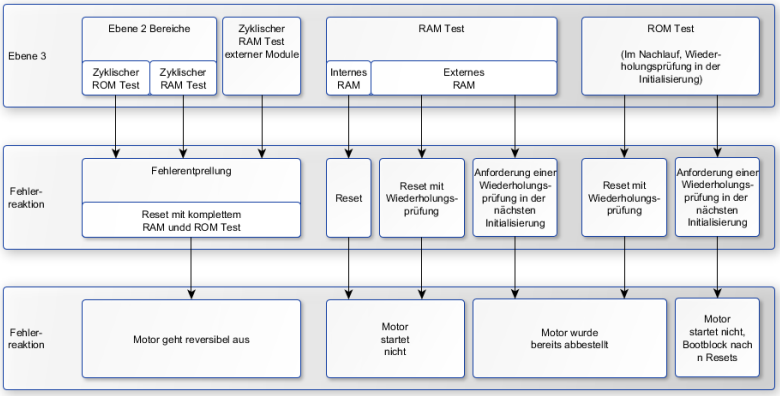


Abbildung 5.5: Fehlerreaktionen der Speichertests in Ebene 3 gemäß [3]

5.5.4 Systemreaktionen auf Fehler

Die notwendigen Plausibilisierungs-Toleranzen sind fahrzeughersteller- und projektspezifisch. Wird Notluftfahren angefordert und die Notluftposition nicht erreicht, wird eine Einspritzmengen-Begrenzung ausgelöst. Die maximale Zeitdauer von Fehlererkennung bis zum Beginn der Systemreaktion ist fehlerspezifisch zu definieren. Die Erkennung von bestimmten Fehlern in der Ebene 2 führt dazu, dass die Einspritzmengen-Begrenzung direkt oder indirekt über den Momentenvergleich ausgelöst wird.

Eine Übersicht aller Fehlerreaktionen findet sich in Anhang A.2. Es wurden für jede Ebene mögliche Fehler sowie die Reaktion darauf beschrieben.

5.5.5 Zusätzliche technische Anforderungen

Durch einen unabhängigen Abschaltpfad am Steuergerät und geeignete Maßnahmen ist sicherzustellen, dass der Verbrennungsmotor mit Erkennen von „Klemme 15 - Aus“ mit zulässiger Zeitverzögerung redundant sicher abgestellt wird.

Kapitel 6

Konzeptentwicklung

Das erstellte Konzept orientiert sich am E-Gas Sicherheitskonzept [siehe Kapitel 5]. Während das E-Gas Konzept für Einzelkernprozessoren entwickelt wurde, wird in diesem Konzept ein Steuergerät mit mehreren Kernen verwendet. Durch diese Maßnahme wird nicht nur ein Gewinn an Performanz, sondern auch eine Erhöhung der Sicherheit erwartet, da die Funktions- und Überwachungseinheit nicht mehr von ein- und demselben Kern ausgeführt werden. Dies entlastet die einzelnen Rechenkerne zum Einen, zum Anderen wirken sich lokale Störungen am Kern nicht auf die Ausführung unabhängiger Applikationen aus. Jedoch steigt dabei auch die Komplexität, welche es zu beherrschen gilt, um ein Vielfaches. [54] So muss zu jedem Zeitpunkt sichergestellt sein, dass sich die beiden Kerne nicht gegenseitig behindern, was durch das Auftreten geteilter Ressourcen durchaus möglich ist. [45]

Besonders für Ebene 3 des Überwachungskonzepts steigt die Herausforderung nun mehrere Kerne auf deren Integrität überwachen zu müssen. Weiters muss durch die konkurrierende Ausführung auch der richtige Programmablauf nicht nur am Kern selbst, sondern auch zwischen den Kernen geprüft werden. [50]

Als Ergebnis ist eine Implementierung des E-Gas Konzeptes auf einer Mehrkernarchitektur entstanden, welche auf die Anforderungen und den Stand der Technik der modernen Produktentwicklung in der Automobilindustrie eingeht und einen Entwicklungsprozess in Anlehnung an die ISO 26262 verwendet.

Systemdefinition

Der ISO Norm entsprechend, wird zunächst eine Definition des zu entwickelnden Systems selbst vorgenommen. Dabei wird aus rein funktioneller Sicht das Gesamtsystem mit den einzelnen Funktionen beschrieben. Abbildung 6.1 zeigt die Bestandteile des Demonstrators:

Über ein Gaspedal soll der Fahrerwunsch vom Steuergerät aufgenommen werden und ein Aktuator (in diesem Fall eine Drosselklappe) linear zur Pedalposition geregelt werden. Um als Benutzer eine Rückmeldung zum Zustand des Systems zu erhalten, und um das System von außen kontrollieren zu können, wird über ein Touchscreen eine Benutzerschnittstelle zur Verfügung gestellt. Über diese sollen die Positionen des Pedals und der Drosselklappe dargestellt werden, sowie eine Aktivierung und Deaktivierung des Systems

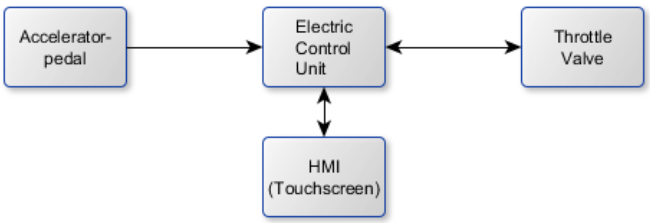


Abbildung 6.1: Systembeschreibung

mittels Tastendrucks ermöglicht werden. Um die Entwicklung zu Erleichtern und zu Demonstrationszwecken sollen auch weitere Informationen zu den Zuständen der Sensoren, Aktuatoren sowie des Steuergerätes selbst angezeigt werden.

6.1 Gefahren- und Risikoanalyse

Das höchstkritische Sicherheitsziel im E-Gas Überwachungskonzept [3] ist die Vermeidung ungewollter Beschleunigung, welches mit ASIL B bewertet wurde. Dieses Ziel wird auch für die Entwicklung des Demonstrators übernommen. Zum Zwecke der Demonstration und zur Anwendbarkeit am Demonstrator, wird dabei die Beschleunigung des Fahrzeuges in kausaler Abhängigkeit zur Beschleunigung der Drosselklappe angenommen. Somit wird das Sicherheitsziel für den Demonstrator in Tabelle 6.1 wie folgt festgelegt:

Tabelle 6.1: Sicherheitsziel der Gefahren und Risikoanalyse

N°	Sicherheitsziel	ASIL
SZ-01	Vermeidung ungewollter Beschleunigung der Drosselklappe	ASIL B

Eine Methode zur automatisierten, optimalen Zuordnung von Sicherheits-Integritätslevels auf Systemkomponenten wird in [55] behandelt.

6.2 Funktionales Sicherheitskonzept

Das funktionale Sicherheitskonzept legt rein funktionale Anforderungen an das System fest, um das Sicherheitsziel zu erreichen. Dabei wird für jede sicherheitskritische Komponente ein Konzept zur Gewährleistung des Sicherheitsziels erstellt.

Um einen Defekt am Gaspedal zu erkennen, welcher zu einer fehlerhaften Position der Drosselklappe führt, muss die Position des Gaspedals plausibilisierbar sein. Dasselbe gilt auch für die Position der Drosselklappe selbst, um ein falsches Regeln zu verhindern. Im Steuergerät muss sichergestellt werden, dass Fehler an Sensorik und Aktuatorik auch

Tabelle 6.2: Funktionales Sicherheitskonzept für den Demonstrator

N°	Komponente	Funktionales Sicherheitskonzept
FSK-01.00	Sensorik	Die Sensorsignale sind nach Signalerfassung plausibilisierbar.
FSK-02.00	Aktuatorik	Die Aktorsignale sind nach Signalerfassung plausibilisierbar.
FSK-03.00	Steuergerät	Das Steuergerät erkennt Fehler an der Sensorik.
FSK-03.01	Steuergerät	Das Steuergerät erkennt Fehler an der Aktuatorik.
FSK-03.02	Steuergerät	Die Steuerungslogik wird auf Plausibilität überwacht.
FSK-03.03	Steuergerät	Das Steuergerät wird von einem unabhängigen Überwachungsmonitor überwacht.
FSK-03.04	Steuergerät	Die maximale Zeitdauer vom Auftritt erkennbarer Fehler in der Sensorik bis zum Beginn der Fehlerreaktion darf 50ms nicht überschreiten.
FSK-04.00	Überwachungsmodul	Das Überwachungsmodul wird vom Steuergerät überwacht.
FSK-04.01	Überwachungsmodul	Das Überwachungsmodul kann einen sicheren Zustand unabhängig vom Steuergerät herbeiführen.
FSK-04.03	Überwachungsmodul	Die maximale Zeitdauer vom Auftritt erkennbarer Fehler bis zum Beginn der Fehlerreaktion darf 200ms nicht überschreiten.

als solche erkannt werden. Dazu muss auch die Funktion der Ein- und Ausgangsmodule überprüft werden.

Um zu verhindern, dass Systemfehler und Berechnungsfehler in der Steuerung zu einer Verletzung des Sicherheitsziels führen, muss das Steuerungsprogramm selbst auch auf Plausibilität überprüft werden. Zur Überprüfung der Integrität des Steuerungscontrollers, muss dieser auf von einem externen Controller überwacht werden. Dieser soll die Aktoren in einen sicheren Zustand schalten, falls dies von der Steuerung selbst nicht möglich ist bzw. ein Fehler erkannt wurde. Das Überwachungsmodul soll elektrisch unabhängig zum Steuerungscontroller arbeiten. Gleichzeitig soll das Überwachungsmodul vom Steuerungscontroller auf Funktionsfähigkeit kontrolliert werden.

Zur Erreichen des Sicherheitsziels sind die Anforderungen an die funktionale Sicherheit, wie in Tabelle 6.3 ersichtlich, festgelegt worden: Dabei ist zu gewährleisten, dass im Fehlerfall ein sicherer Zustand erreicht wird, bevor ein Sicherheitsziel verletzt wird.

Abbildung 6.2 zeigt das System welches um einen Überwachungsmonitor ergänzt wurde. Dieser stellt sicher, dass ein fehlerhaftes Verhalten des Steuergerätes erkannt wird und die Drosselklappe in einen sicheren Zustand geschaltet wird.

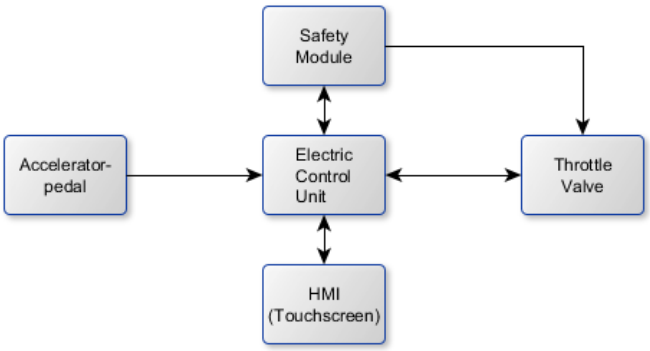


Abbildung 6.2: Systembeschreibung mit Überwachungserweiterung

Kapitel 7

Systementwicklung

Aus den vorherigen Schritten wird nun auf Systemebene ein Sicherheitskonzept beschrieben. Dieses leitet Sicherheitsmaßnahmen aus den rein funktionalen Anforderungen ab. Dabei werden diese bereits unter Berücksichtigung der technische Umsetzung erstellt. Zunächst wird das technische Sicherheitskonzept beschrieben und im Anschluss der Systementwurf durchgeführt.

7.1 Technische Sicherheitsanforderungen

Zur Erstellung technischer Sicherheitsanforderungen werden die Anforderungen an die sichere Funktion aus technischer Sicht beschrieben. Diese ist noch weitgehend unabhängig von der verwendeten Hardware beschrieben, jedoch werden Mechanismen zur Umsetzung der Funktion häufig schon vorweggenommen. Tabelle 7.1 stellt die anzuwendenden technischen Sicherheitsanforderungen dar.

Da der Fokus dieser Arbeit auf der Betriebssicherheit des Systems liegt, wurden Anforderungen zur Zuverlässigkeit und Verfügbarkeit nicht im Speziellen behandelt. Dennoch ist besonders im automotiven Bereich eine Untersuchung dahingehend notwendig. Durch den rekursiven Prozess in der Entwicklung können in den technischen Sicherheitsanforderungen auch Anforderungen enthalten sein, welche durch spätere Entwicklungsschritte hinzugefügt wurden.

Tabelle 7.1: Technische Sicherheitsanforderungen

N°	Komponente	Sicherheitsanforderung
TSANF-1.0	Fahrpedal	Zwei elektrisch unabhängige Sensoren zur Positionsmessung
TSANF-1.1	Fahrpedal	Die beiden Signale sind sowohl gegeneinander, als auch in ihrem Wertebereich plausibilisierbar
TSANF-1.2	Fahrpedal	Die Sensorsignale sollen auf Gleichtaktfehler unterschiedlich reagieren
TSANF-1.3	Fahrpedal	Die Sensorsignale ändern sich durch Umgebungsänderung und Alterung wenig
TSANF-1.4	Fahrpedal	Ein Rückstellsystem soll bei Nichtbetätigung die Nullposition selbstständig erreichen

TSANF-2.0	Steuergerät	Überwachung der eigenen Spannungs- und Taktversorgung
TSANF-2.1	Steuergerät	Überwachung der Betriebstemperatur
TSANF-2.2	Steuergerät	Diagnoseschaltung für sicherheitskritische Eingangsmodule
TSANF-2.3	Steuergerät	Diagnoseschaltung für sicherheitskritische Ausgangsmodule
TSANF-2.4	Steuergerät	Hohe Störungsfestigkeit für sicherheitskritische Signale
TSANF-3.0	Überwachungs-modul	Überwachung der eigenen Spannungs- und Taktversorgung
TSANF-3.1	Überwachungs-modul	Überwachung der Betriebstemperatur
TSANF-3.2	Überwachungs-modul	Diagnoseschaltung für sicherheitskritische Eingangsmodule
TSANF-3.3	Überwachungs-modul	Diagnoseschaltung für sicherheitskritische Ausgangsmodule
TSANF-3.4	Überwachungs-modul	Hohe Störungsfestigkeit für sicherheitskritische Signale
TSANF-4.0	Leistungs-treiber	Schutz gegen Überlastung von Treiber und Motor
TSANF-4.1	Leistungs-treiber	Zustand des Treibers ist überprüfbar
TSANF-4.2	Leistungs-treiber	Zwei elektrisch getrennte Kanäle zur Aktivierung des Treibers (für Funktion und Funktionsüberwachung)
TSANF-4.3	Leistungs-treiber	Sicherheitsschalter, welcher die Spannungsversorgung des Treibers unterbricht (für das Überwachungsmodul)
TSANF-4.4	Leistungs-treiber	Kurzschluss- und Übertemperatur Schutz
TSANF-4.5	Leistungs-treiber	Kabelbrucherkennung am Ausgang
TSANF-5.0	Drosselklappe	Zwei elektrisch unabhängige Sensoren zur Positionsmessung
TSANF-5.1	Drosselklappe	Die beiden Signale sind sowohl gegeneinander, als auch in ihrem Wertebereich plausibilisierbar
TSANF-5.2	Drosselklappe	Die Sensorsignale sollen auf Gleichtaktfehler unterschiedlich reagieren
TSANF-5.3	Drosselklappe	Geringe Gleichlaufabweichung
TSANF-5.4	Drosselklappe	Die Sensorsignale ändern sich durch Umgebungsänderung und Alterung wenig
TSANF-5.5	Drosselklappe	Ein Rückstellsystem soll die Klappe bei fehlender Ansteuerung in einen sicheren Zustand bringen.

7.2 Systementwurf

Im Systementwurf werden nun die Umsetzung der technischen Sicherheitsanforderungen in einem Konzept beschrieben und anschließend mit der Entwurfsspezifikation begonnen. Im Anschluss wird das Hardware-Software Interface erstellt, welche als Ausgangspunkt für die Hardware und Softwareentwicklung verwendet wird.

7.2.1 Technisches Sicherheitskonzept

Das technische Sicherheitskonzept lehnt sich stark an das E-Gas Überwachungskonzept (siehe Kapitel 5), wurde jedoch an die Möglichkeiten und Herausforderungen einer Mehrkernarchitektur angepasst. Wiederum besteht das Sicherheitskonzept aus drei Ebenen, wobei in diesem Fall die Ausführung der Funktion sowie die der Funktionsüberwachung über zwei getrennte Recheneinheiten (Central Processing Units (CPUs)) erfolgt. Das hat neben der Entlastung des Funktionsrechners einen Sicherheitsgewinn zur Folge, da sich lokale Fehler auf die CPU nicht direkt auf die Funktionsüberwachung auswirken können. Um die Integrität des Steuergerätes sicherzustellen, müssen nun auf Ebene 3 alle Rechenkerne, welche einen Einfluss auf sicherheitskritische Funktionen des Gesamtsystemes haben, auf Betriebssicherheit überprüft werden. Gleichzeitig muss sichergestellt sein, dass der Programmfluss über mehrere Kerne hinweg konsistent bleibt, damit keine Race Conditions[1] entstehen. [72]

Ebene 1

In Ebene 1 (Funktionsebene) werden alle Steuerungsfunktionen ausgeführt. Diese beinhaltet das Einlesen der Eingangsgrößen, Berechnungen der Regelgrößen sowie Ausgabe der Ausgangsgrößen. In Ebene 1 wird zum Zwecke der funktionalen Sicherheit eine Plausibilisierung der Eingangsgrößen durchgeführt und mit einer entsprechenden Fehlerreaktion auf Eingangsfehler reagiert. Ebene 1 wird auf dem Funktionskern ausgeführt, welcher zusammen mit dem Funktionsüberwachungskern im selben Mehrkerncontroller untergebracht ist, und sich auch die verwendeten Peripheriemodule mit anderen Kernen teilt.

Ebene 2

In Ebene 2 (Funktionsüberwachungsebene) wird die Funktion der Ebene 1 überwacht. Dabei werden unabhängig von der Steuerungsfunktion diejenigen Signale eingelesen, welche eine Plausibilisierung der richtigen Ausführung von Ebene 1 ermöglichen. Dies umfasst zumindest die Signale aller Sensoren sowie Aktuatoren, welche von Ebene 1 verwendet

[1] bezeichnet eine Konstellation, in der das Ergebnis einer Operation vom zeitlichen Verhalten der einzelnen Operationen abhängt. Das Entstehen solcher Race Conditions gilt es zu vermeiden, da die Folgen meist Berechnungsfehler sowie Datenkorruption sind.

werden. Um systematische Fehler an der Steuerungslogik zu erkennen, ist Ebene 2 unabhängig von Ebene 1 zu entwickeln. Die Ausführung von Ebene 2 erfolgt auf einem separaten Rechenkern im Mehrkerncontroller. Um die Funktion der sicherheitskritischen Ausgänge zu überprüfen, wird in Ebene 2 deren Funktionsfähigkeit periodisch getestet.

Ebene 3

Ebene 1 und Ebene 2 teilen sich systembedingt mehrere Abhängigkeiten wie die Spannungsversorgung, Taktversorgung, ggf. gemeinsamen Speicher sowie gemeinsam genutzte Ein- und Ausgabemodule. Um im Fehlerfall auf ein Versagen des Rechners reagieren zu können, überprüft eine dedizierte Steuereinheit (Monitorcontroller) über ein Frage/Antwortverfahren die Integrität des Funktionscontrollers. Dabei wird auf jedem sicherheitskritischen Kern die richtige Funktion der Instruktionen, sowie der Speicher getestet. Zusätzlich wird der richtige Programmablauf zwischen den Kernen über eine Programmflusskontrolle überprüft. Dazu sendet die Überwachungseinheit eine Frage an den Funktionsrechner. Diese wird über ein sogenanntes Daisy-Chain Verfahren von allen Kernen bearbeitet und die Antwort, welche sich durch den richtigen Ablauf der Kette sowie in Abhängigkeit vom Instruktionstest und dem Speichertest ergibt, wird zurück an das Überwachungsmodul gesendet. Wird vom Überwachungsmodul eine Abweichung von der erwarteten Antwort erkannt, wird die Frage wiederholt, bis es bei Überschreitung der Fehlerschwelle zu einer Einleitung eines sicheren Zustands vom Überwachungsmodul kommt. Auf diese Weise kann die Überwachungssoftware am Funktionsrechner die richtige Funktion des Überwachungsmoduls selbst überprüfen. Wie am Funktionsrechner muss auch die Funktionstüchtigkeit der Sicherheitsschalter periodisch getestet werden.

Abbildung 7.1 zeigt den Aufbau des Sicherheitskonzepts mit den drei Ebenen, welche auf zwei unabhängige Rechner verteilt sind, sowie deren Aufgaben und Sicherheitspfade.

7.2.2 System Entwurfsspezifikation

Das Sicherheitskonzept angewendet auf den Demonstrator ist in Abbildung 7.2 als Systemarchitektur dargestellt. Im Unterschied zum reinen Konzept sind hier die Ein- und Ausgangsgrößen spezifiziert.

Ebene 1 liest über das Eingangsmodul die Positionen des Fahrpedals, sowie der Drosselklappe ein, um daraus einen Ausgabewert für den Drosselklappen-Motor zu berechnen. Dieser wird über das Ausgabemodul zum Leistungstreiber geschickt, welcher den Motor ansteuert. Zur Erkennung von Fehlern werden beide Positionssignale über je zwei getrennte Kanäle gesendet, um eine Plausibilisierung zu ermöglichen. Weiters wird von Ebene 1 auch der Zustand des Leistungstreibers überwacht. Im Fehlerfall wird das Steuerungssignal abgeschaltet, sowie der Leistungstreiber deaktiviert. Die Drosselklappe befindet sich ohne Stromversorgung in einem sicheren Zustand.

Ebene 2 überwacht die Funktion von Ebene 1 auf Plausibilität durch einen (nach Möglichkeit) diversitär entwickelten Algorithmus. Falls ein Fehler eintritt, kann Ebene 2 selbst-

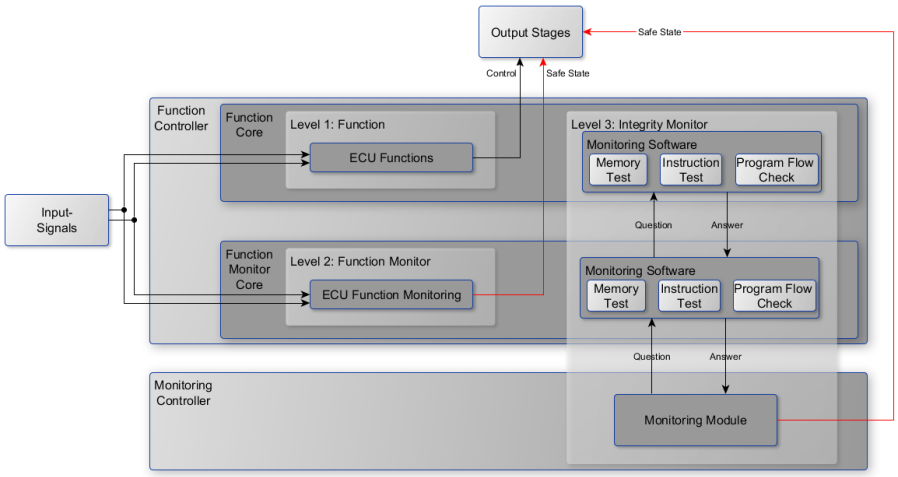


Abbildung 7.1: Beschreibung des technischen Sicherheitskonzepts

ständig die Leistungsstufe deaktivieren, um einen sicheren Zustand zu erreichen. Weiters wird von Ebene 2 sowohl das Eingangs- als auch das Ausgangsmodul auf richtige Funktion überwacht. Um ein sicheres Abschalten zu gewährleisten, muss von Ebene 2 ein Abschaltpfadtest durchgeführt werden. Da dies im laufenden Betrieb oft nicht möglich ist, muss dieser zumindest vor, bzw. nach dem Betrieb erfolgen.

Das Konzept sieht vor, dass Ebene 3 auf einem separaten Rechner ausgeführt wird und die Integrität des Steuergerätes überprüft. Falls ein Fehler eintritt, kann das Überwachungsmodul über einen Sicherheitsschalter alle sicherheitskritischen Leistungsendstufen von der Spannungsversorgung trennen, wodurch sich ein sicherer Zustand einstellt. Auch der Sicherheitsschalter muss zumindest vor oder nach dem Betrieb auf Funktionsfähigkeit getestet werden.

Wie in der Kapitel 9.3 später beschrieben wird, wurde der Überwachungsmonitor im Zuge dieser Arbeit als Virtualisierung im Mehrkerncontroller umgesetzt, zum Zwecke einer agilen Weiterentwicklung und Testbarkeit. Dennoch wurde die Implementierung so vorgenommen, dass das virtuelle Modul auch auf dedizierter Hardware übertragbar ist. Das Hauptaugenmerk liegt in diesem Falle auf dem Konzept selbst.

7.2.3 Hardware-Software Interface (HSI)

Das Hardware-Software Interface beschreibt alle Signale welche über eine entsprechende Schnittstelle von Hardware in Software übergehen bzw. von der Software in eine physikalische Größe ausgegeben werden, um für die Hardware- und Softwareentwicklung einen gemeinsamen Ausgangspunkt zu ermöglichen. Da im Systementwurf eine genaue Spezifikation dieser Schnittstelle noch nicht möglich ist, wird dieses Dokument während der Hardware- und Softwareentwicklung weiter verfeinert und angepasst.

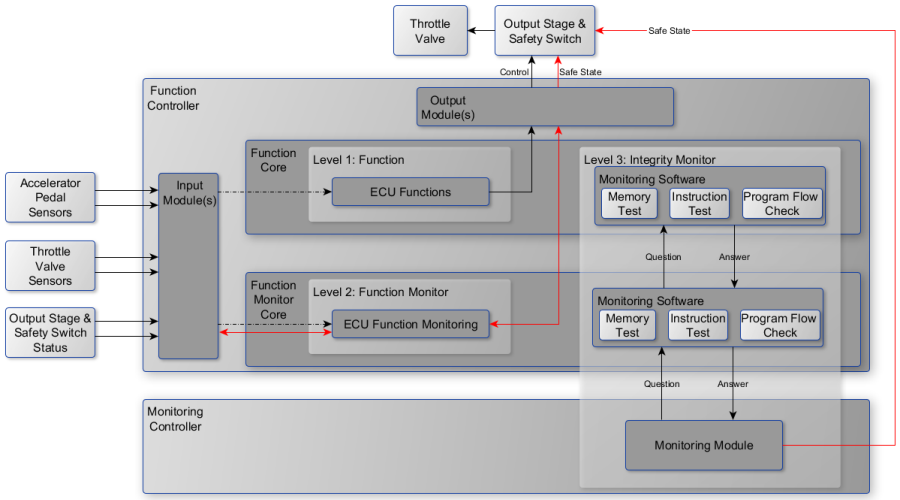


Abbildung 7.2: Systemarchitektur des Sicherheitskonzepts

Eine Tabelle zum Hardware-Software Interface des Demonstrators befindet sich in Anhang A.5.

7.2.4 Abweichungen zur ISO Norm

Um dem Entwicklungsprozess nach der ISO 26262 zu entsprechen, is es nötig, nach jedem der beschriebenen Entwicklungsschritte eine Evaluierung zur Einhaltung der Sicherheitsziele sowie in Abhängigkeit von der Kritikalität des Sicherheitszieles bestimmte Verifizierungsmaßnahmen der jeweiligen Arbeitsergebnisse durchzuführen. Diese Maßnahmen lagen jedoch nicht im Fokus der Arbeit. Eine vollstängige Auflistung der geforderten Arbeitsschritte zur Einhaltung der Norm findet sich in Anhang A.4.

Kapitel 8

Hardwareentwicklung

Ist das Sicherheitskonzept auf Systemebene abgeschlossen, wird mit der Entwicklung der Hardware begonnen. Bevor jedoch die eigentliche Implementierung beginnt, werden zunächst Sicherheitsanforderungen erstellt, welche nach dem Hardwareentwurf auf Einhaltung überprüft werden müssen.

Abbildung 8.1 zeigt den Ablauf der Hardwareentwicklung, eingebettet in den Produktlebenszyklus beginnend bei der Konzepterstellung, über die Entwicklung auf Systemebene hin zur Hardwareentwicklung selbst, inklusive der begleitenden Evaluierungsprozesse.

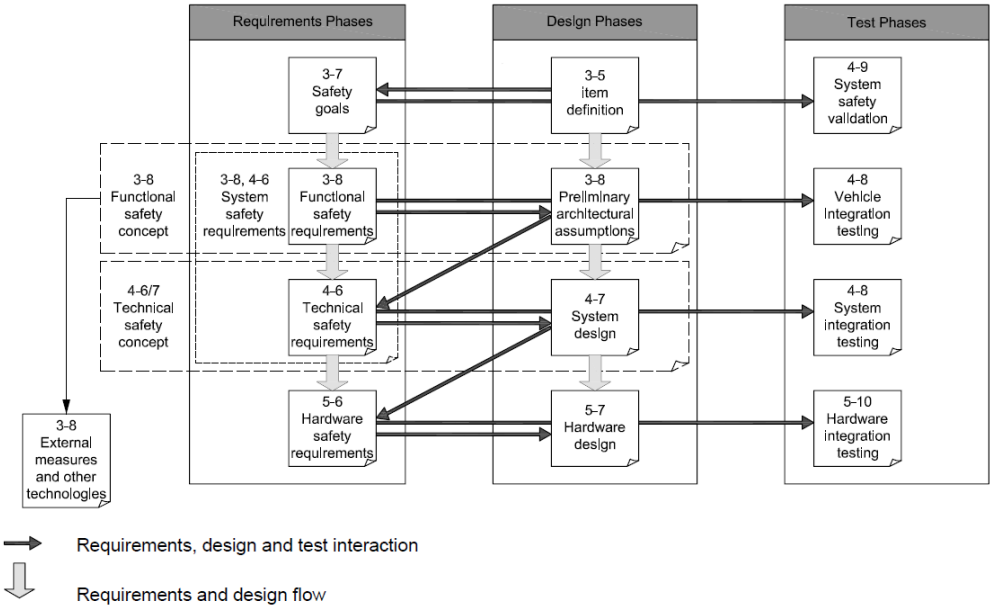


Abbildung 8.1: Hardwareentwicklung eingebettet in den Gesamtentwicklungsprozess aus [42]

8.1 Sicherheitsanforderungen an die Hardware

Für jede Hardwarekomponente im System müssen Sicherheitsanforderungen („SANFH“) aufgestellt werden, um die Einhaltung des Sicherheitsziels zu gewährleisten. Diese werden in Tabelle 8.1 aufgelistet.

Da der Fokus dieser Arbeit auf der Betriebssicherheit der Hardware liegt, wurden Anforderungen bezüglich Zuverlässigkeit und Verfügbarkeit nicht im Speziellen behandelt. Dennoch ist besonders im automotiven Bereich eine Untersuchung dahingehend wichtig.

Tabelle 8.1: Sicherheitsanforderungen an die Hardware

N°	Komponente	Sicherheitsanforderung
SANFH-01	Fahrpedal	Zwei Sensoren mit physikalisch getrennten Signalpfaden
SANFH-02	Fahrpedal	Zwei physikalisch getrennte Versorgungsleitungen
SANFH-03	Fahrpedal	Zwei physikalisch getrennte Masseleitungen
SANFH-04	Fahrpedal	Beide Sensorsignale besitzen plausibilisierbaren Wertebereich (ohne Überschneidung)
SANFH-05	Fahrpedal	Beide Sensorsignale besitzen unterschiedlichen Steigungsfaktor bei gleicher Positionsänderung
SANFH-06	Fahrpedal	Die beiden Sensorsignale steigen in entgegengesetzter Richtung bei Positionsänderung
SANFH-07	Fahrpedal	Beide Sensorsignale besitzen geringe Gleichlaufabweichung sowie hohe Auflösung zur Diagnose
SANFH-08	Fahrpedal	Beide Sensoren verhalten sich konstant gegen Umgebungs- und Lebensdauerbedingungen
SANFH-09	Steuergerät	Überwachung der Spannungsversorgung
SANFH-10	Steuergerät	Überwachung der Taktfrequenz
SANFH-11	Steuergerät	Überwachung der Temperatur
SANFH-12	Steuergerät	Diagnoseschaltung für sicherheitskritische Eingangsmodule
SANFH-13	Steuergerät	Diagnoseschaltung für sicherheitskritische Ausgangsmodule
SANFH-14	Steuergerät	Hohe Störungsfestigkeit für sicherheitskritische Signale
SANFH-15	Überwachungsmodul	Überwachung der Spannungsversorgung
SANFH-16	Überwachungsmodul	Überwachung der Taktfrequenz
SANFH-17	Überwachungsmodul	Überwachung der Temperatur
SANFH-18	Überwachungsmodul	Diagnoseschaltung für sicherheitskritische Eingangsmodule
SANFH-19	Überwachungsmodul	Diagnoseschaltung für sicherheitskritische Ausgangsmodule
SANFH-20	Überwachungsmodul	Hohe Störungsfestigkeit für sicherheitskritische Signale
SANFH-21	Leistungstreiber	Schutzschaltung gegen Überlastung von Treiber und Motor
SANFH-22	Leistungstreiber	Statussignal für Rückmeldung über Zustand des Treibers
SANFH-23	Leistungstreiber	Eingang zur Aktivierung des Treibers

SANFH-24	Leistungstreiber	Eingang zur Deaktivierung des Treibers
SANFH-25	Leistungstreiber	Sicherheitsschalter, welcher die Spannungsversorgung des Treibers unterbricht
SANFH-26	Leistungstreiber	Kurzschluss- und Übertemperatur Schutz
SANFH-27	Leistungstreiber	Kabelbrucherkennung am Ausgang
SANFH-28	Drosselklappe	Zwei Positionssensoren mit physikalisch getrennten Signalpfaden
SANFH-29	Drosselklappe	Gegengleiche Steigung des Signals bei Positionsänderung
SANFH-30	Drosselklappe	Hohe Genauigkeit im gesamten Signalbereich zur Regelung und Diagnose
SANFH-31	Drosselklappe	Geringe Gleichlaufabweichung
SANFH-32	Drosselklappe	Plausibilisierbarkeit beider Signale
SANFH-33	Drosselklappe	Beide Sensoren sind robust gegen Umgebungs- und Lebensdauerbedingungen
SANFH-34	Drosselklappe	Kurz-/Nebenschlüsse und Unterbrechungen an Sensorik und Aktuatorik werden erkannt

8.2 Hardwareentwurf

Im Hardwareentwurf werden die vorgegebenen Sicherheitsanforderungen in der Entwurfsspezifikation umgesetzt. Dazu werden nicht nur die einzelnen Anforderungen der vorangegangenen Schritte , sondern auch das sogenannte Hardware-Software Interface herangezogen, welches in Zusammenarbeit mit der Softwareentwicklung erstellt wird.

8.2.1 Hardware Entwurfsspezifikation

Hier erfolgt die Auswahl der Hardware sowie die Umsetzung des Systementwurfs in Hardware. Zunächst werden die einzelnen Bauteile beschrieben und im Anschluss daran ein Schaltplan erstellt, welcher die Bauteile unter Einhaltung aller Anforderungen zum Gesamtsystem verbindet.

8.2.1.1 Fahrpedal

Als Fahrpedal wurde ein handelsüblicher Winkelsensor mit mechanischer Rückstellfeder ausgewählt. Diese besitzt zwei gegenläufige Potentiometer mit einem unterschiedlichem Signalhub. Jeder Sensor besitzt eine eigene Versorgungs-, Signal- sowie Masseleitung und ein Kurzschluss mit einer Leitung ist durch den gewählten Wertebereich erkennbar. Um Gleichtaktfehler sowie Steigungsfehler zu erkennen, besitzen die beiden Kanäle einen gegenläufigen Verlauf mit unterschiedlichem Hub (siehe Abbildung 8.2).

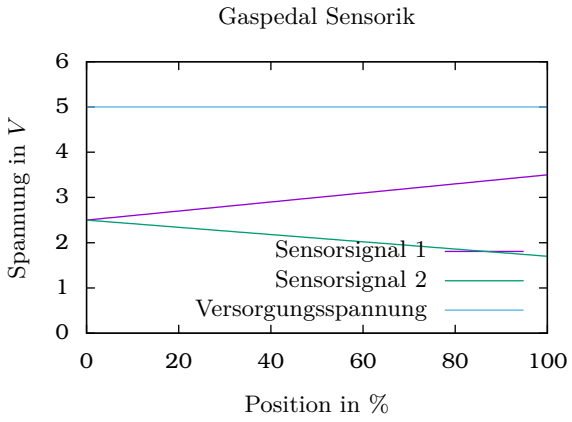


Abbildung 8.2: Sensorspannungen beim Gaspedal

8.2.1.2 Drosselklappe

Als sicherheitskritischer Aktuator wird eine handelsübliche Drosselklappe aus dem PKW Sektor verwendet. Diese beinhaltet einen Gleichstrom-Motor mit mechanischem Rückstellsystem, sowie zwei gegenläufige Potentiometer zur Positionserkennung. Die Ansteuerung des Motors erfolgt über ein Pulsweitenmodulation (PWM) Signal über zwei Kanäle, welche ein Öffnen sowie Schließen der Klappe ermöglicht. Wird der Stromkreis unterbrochen, fährt die Rückstellfeder eine Position an, welche eine minimale Luftzufuhr erlaubt. Das dient dazu, dass ein Leerlauf des Motors weiter möglich ist. Zur Stromversorgung wird ein 12V-Gleichspannungsnetz verwendet. Da die Stromaufnahme abhängig von der Geschwindigkeit der Positionsänderung bis zu 3 Ampere beträgt, wird ein Leistungstreiber dieser Größenordnung benötigt.

Zur Positionserkennung werden zwei gegenläufige Sensoren mit gleichem Hub verwendet. Da der Wertebereich der Sensorsignale nur einen Teil des gesamten Wertebereichs durch die Versorgung einnimmt, können Kurzschlüsse zu Versorgungs- und Masseleitung erkannt werden. Bei einem Kurzschluss zwischen den beiden Sensorsignalen kann lediglich eine Fehlfunktion erkannt werden, nicht jedoch das fehlerhafte Signal selbst, wie es beim Gaspedal der Fall ist. Abbildung 8.3 zeigt die Signalverläufe der Positionssensoren.

8.2.1.3 Motortreiber und Sicherheitsschalter

Als Motortreiber wird eine H-Brücke verwendet, welche eine Last bis zu 6.6 Ampere treiben kann und die Signale mit bis zu 30 kHz von den Logikeingängen an die beiden Ausgangskanäle weitergibt. Um den Treiber zu aktivieren, müssen über zwei getrennte Eingänge mit inverser Logik Freigabesignale anliegen. Zur Erkennung von Übertemperatur, Überlast, Kurzschluss, Kabelbruch sowie der Ansteuerungsaktivierung wird der Zustand (Aktiv/Inaktiv bzw. Fehler) über einen Statuspin an das Steuergerät mitgeteilt.

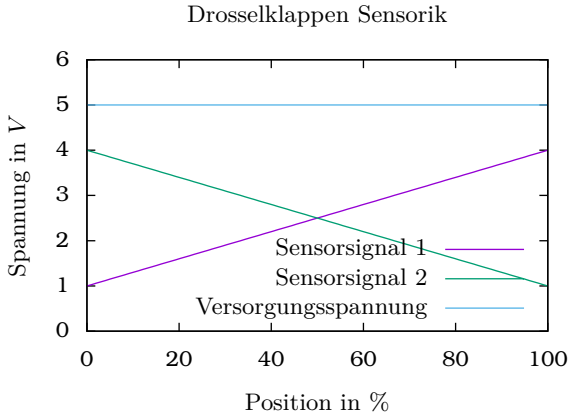


Abbildung 8.3: Sensorspannungen bei der Drosselklappe

Um im Falle eines Fehlers des Motortreibers (z.B. Kurzschluss der Leistungstransistoren) in einen sicheren Zustand schalten zu können, wird die Spannungsversorgung des Treibers über einen Sicherheitschalter abgesichert. Dieser unterbricht im Fehlerfall die komplette Spannungsversorgung des Aktuators und über das mechanische Rückstellsystem wird der sichere Zustand erreicht. Da der Sicherheitsschalter über keine Diagnosemöglichkeit verfügt, muss die Funktionsfähigkeit zu Beginn des Betriebes durch einen Abschalttest überprüft werden.

In Anhang A.3 ist der Schaltplan sowie ein Aufbauplan des Motortreibers abgebildet.

8.2.1.4 Steuergerät

Als Steuergerät wird ein Infineon Aurix Application Kit [29] verwendet. Dieses enthält einen Infineon TC275C Microcontroller [31], eine Spannungsversorgung für den Controller und die Peripherie sowie einen 320x240 Pixel Touchscreen, welcher über Serial Peripheral Interface (SPI) angesteuert wird. Weiters besitzt es noch 4 Status LEDs sowie Anschlüsse für ADC/GPIO/CAN/LIN/RJ45/SPI/I²C/USB, eine Echtzeituhr und einen Anschluss für eine MicroSD-Karte. Abbildung 8.4 zeigt das Blockdiagramm des Steuergerätes mit den verbauten Modulen und Anschlüssen.

Das Steuergerät wird über eine Adapterplatine mit den Sensoren sowie der Ausgangsstufe verbunden. Die Platine liefert auch die Spannungsversorgung des Steuergerätes. Das verwendete Steuergerät muss neben der Mehrkernfähigkeit auch besonders im automotiven Bereich den Anforderungen zur funktionalen Sicherheit gerecht werden. Die entstehenden Herausforderungen werden in [70, 54] beschrieben. Der ausgewählte Controller wurde speziell für sicherheitskritische Funktionen im Automobilbereich entwickelt und besitzt 3 Rechenkerne welche unabhängig voneinander operieren können.

Abbildung 8.5 zeigt das Blockschaltbild des Aurix. Die drei Rechenkerne mit eigenem

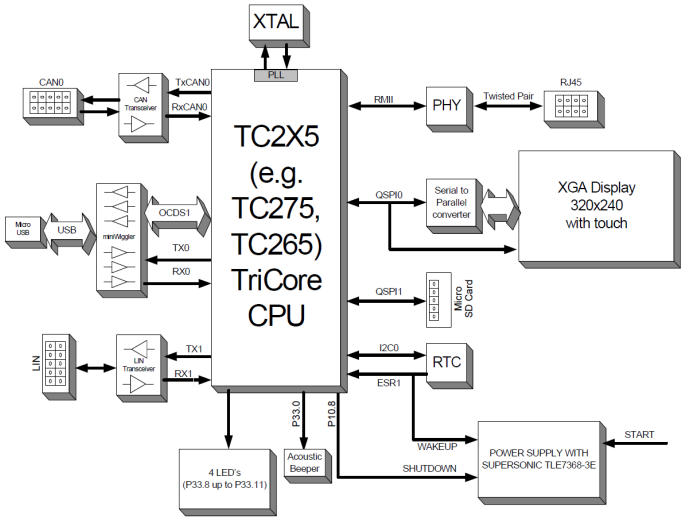


Abbildung 8.4: Blockdiagramm des Steuergerätes aus [29]

Programm- und Datenspeichermodul sind über einen sicheren Bus miteinander sowie mit einem gemeinsamen RAM und ROM Speicher verbunden. Weiters werden über diesen Bus die Brücke sowie die DMA Controller zu den Peripheriemodulen verbunden.

Sicherheitskritische Module im Controller sind mit Funktionen zur Fehlererkennung ausgerüstet, welche mit einem dedizierten Sicherheitsmodul verbunden sind. Im Falle der Fehlererkennung kann dieses Modul konfigurierbare Reaktionen wie das Senden von Interrupts, ein Abschalten einzelner Peripheriemodule oder auch das Zurücksetzen des Controllers einleiten.

Folgende Sicherheitsfunktionen werden im Aurix zur Erhöhung der Ausführungssicherheit per Hardware ermöglicht:

- Lockstep Komparator zur Erkennung von CPU Fehlern
- SRAM: Einzelbit-Fehler Korrektur
- SRAM: Erkennung unkorrigierbarer Fehler
- SRAM: Überlauf des Adresspuffers
- SRAM: Adressierungsfehler
- FLASH: Einzel-Bitfehler Korrektur
- FLASH: Zweifach-Bitfehler Korrektur
- FLASH: Erkennung unkorrigierbarer Fehler
- FLASH: Einfach- und Zweifach-Bitfehler Erkennung bei Adresspufferüberläufen
- FLASH: Adressierungsfehler
- Schutz des Prozessor Busses (Shared Resource Interconnect (SRI)) durch Sicherheitsmechanismen

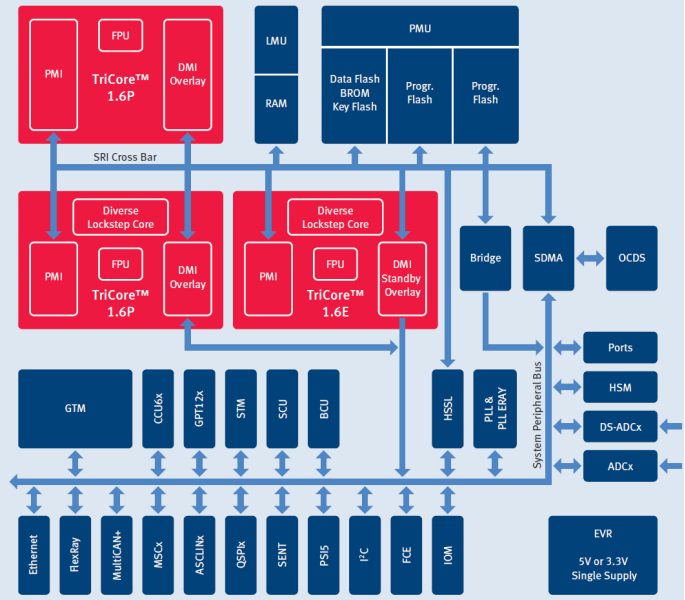


Abbildung 8.5: Systemarchitektur der Aurix Microcontroller Familie aus [31]

- Register Überwachung
- Taktüberwachung
- Erkennung von Frequenzfehlern der Phase-Locked Loops (PLLs)
- Modul zur Überwachung der Ein- und Ausgänge
- Überwachung der Hardware-Interrupts
- Spannungsüberwachung
- Fehlermeldungen vom SRI Bus und vom System Peripherie Bus
- Temperaturüberwachung des Chips

Abbildung 8.6 zeigt die Sicherheitsmechanismen zur Erhöhung der Betriebssicherheit. Zusätzlich enthält der Großteil der Peripheriemodule Funktionen zur Selbstüberprüfung. Auch Module wie die Direct Memory Access (DMA) Module zur Datenübertragung besitzen Funktionen zur Sicherstellung, sodass Übertragungsfehler erkannt werden können.

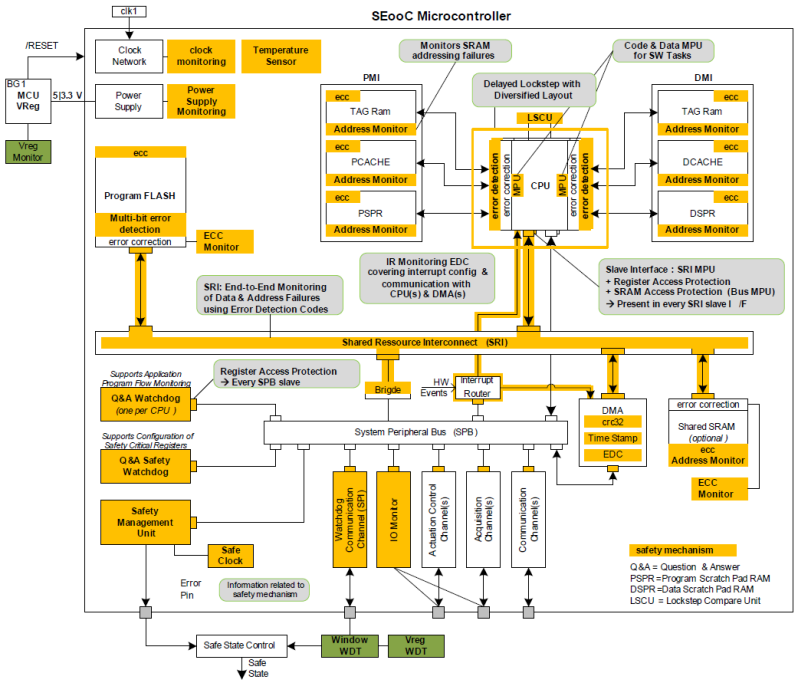


Abbildung 8.6: Sicherheitsmechanismen beim Infineon AURIX aus [30]

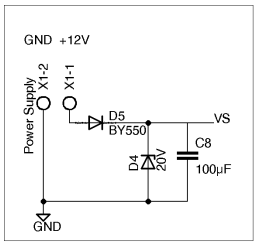
8.2.2 Schaltplan

Abbildung 8.7 zeigt den Schaltplan der Demonstratorplattform. Darin abgebildet sind die Spannungseingangsfilterung, die Spannungsversorgung für das Steuergerät und das Gaspedal sowie der Sicherheitsschalter, welcher den Ausgangstreiber im Fehlerfall vom Netz trennt. Weiters sind die Anschlussleitungen des Gaspedals abgebildet, sowie die Versorgung und Filterung der Drosselklappensensorik und der PWM Treiberbaustein Infineon TLE7209-2R [28].

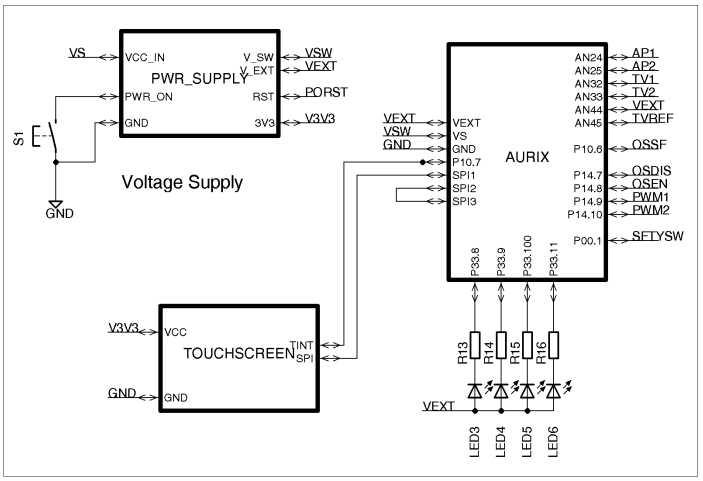
8.3 Hardware Integration

In der Hardwareintegration wurden einzelnen Komponenten aufgebaut und zu einem Modul zusammengefügt. Das Ergebnis findet sich in Abbildung 8.8. Links vorne sieht man das Gaspedal, welches mit den Eingängen des Steuergerätes links oben im Bild verbunden ist. Der Ausgangstreiber findet sich zusammen mit der Spannungsversorgung und der Filterung für die Drosselklappensensorik unter dem Steuergerät.

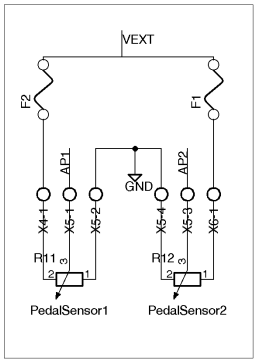
Wie man am Display erkennen kann, befindet sich zum Zeitpunkt der Aufnahme bereits Software für die Benutzerschnittstelle am Steuergerät, welches eine Überprüfung der Werte von Sensorik und Aktuatorik ermöglicht und Informationen zu den Betriebszuständen der



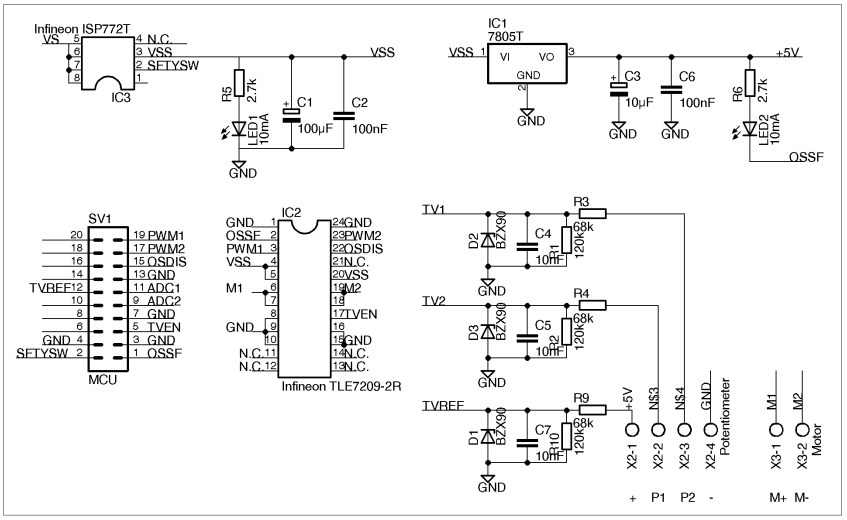
Voltage Supply



Electrical Control Unit: Infineon Aurix Application Kit



Acceleration Pedal Sensors



Output Stage with Safety Switch and Position Filter

Abbildung 8.7: Schaltplan des Throttle-by-Wire Demonstrators

drei Sicherheitsebenen und des Ausgangstreibers anzeigt.

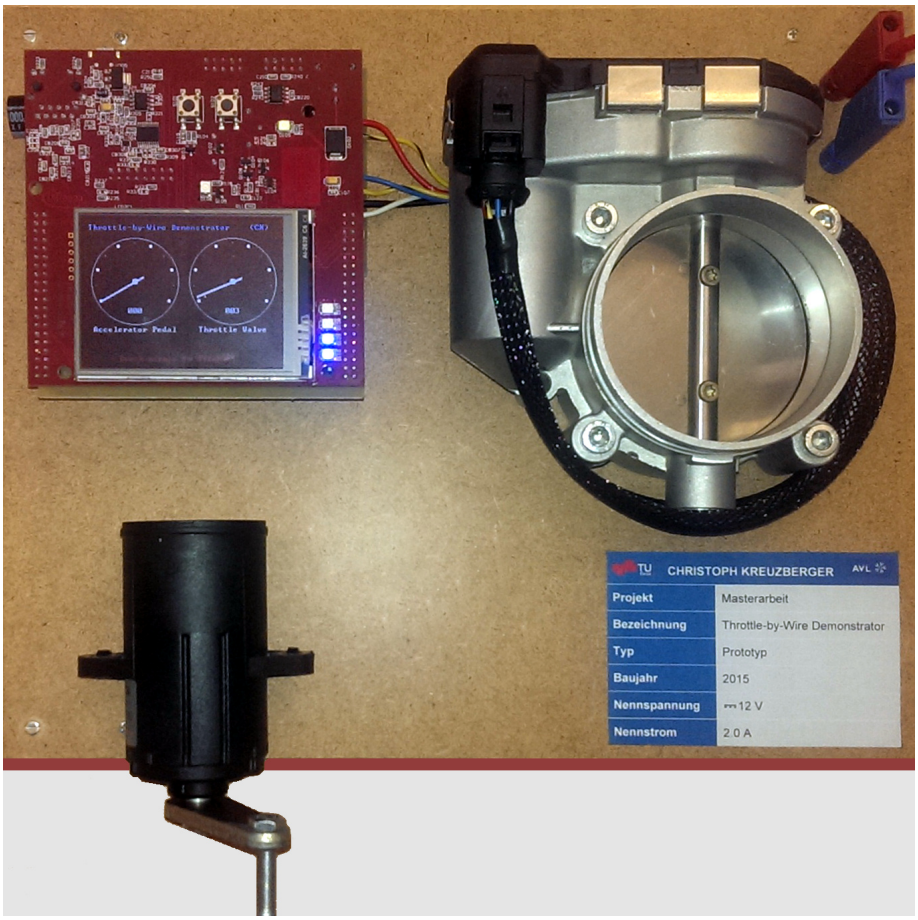


Abbildung 8.8: E-Gas Demonstrator

8.3.1 Abweichungen zur ISO Norm

Die Hardwareentwicklung wurde in Anlehnung an die ISO 26262 durchgeführt, jedoch wurden nicht alle geforderten Entwicklungsschritte im Zuge dieser Arbeit durchgeführt. So sind die Evaluierung von Hardware Architekturmetriken und die Evaluierung der Einhaltung von Sicherheitszielen in Bezug auf zufällige Hardware Fehler zur Einhaltung der Norm nicht enthalten. Der Erhalt nötiger Parameter wie FIT Raten zur Analyse zufälliger Hardware Fehler war zum gegenwärtigen Zeitpunkt nicht möglich. Eine vollstängige Auflistung der geforderten Arbeitsschritte zur Einhaltung der Norm findet sich in Anhang A.4.

Kapitel 9

Softwareentwicklung

Zur Softwareentwicklung wird zunächst ein Überblick über die verwendeten Werkzeuge („Toolchain“) gegeben. Um ein sicheres System zu gewährleisten, müssen auch die verwendeten Programme zur Erstellung der Software den ASIL Anforderungen genügen. Im Anschluss daran wird die Sicherheitsanforderung spezifiziert und der Entwurf der Softwarearchitektur durchgeführt. Dabei ist darauf zu achten, dass alle Anforderungen an die Softwarearchitektur sowie an die verwendete Werkzeugkette eingehalten werden.

Ist die Architektur definiert, wird der Entwurf der einzelnen Softwareteile durchgeführt und diese auf Einhaltung der an sie gestellten Anforderungen getestet. Im Anschluss wird die Software in das Gesamtsystem integriert und evaluiert.

Abbildung 9.1 zeigt den Ablauf der Softwareentwicklung, eingebettet in den Produktlebenszyklus von der Konzepterstellung, über die Entwicklung auf Systemebene hin zur Softwareentwicklung selbst, inklusive der begleitenden Evaluierungsprozesse.

9.1 Entwicklungsumgebung

Ausgangspunkt der Softwareentwicklung ist die integrierte Entwicklungsumgebung (Integrated Development Environment (IDE)) in welcher die einzelnen Softwarekomponenten zu einer Einheit gebündelt werden und von welcher aus der Erstellungsprozess des programmierbaren Binärcodes gestartet wird. Hierfür wurde die IDE Eclipse [15] verwendet.

Einen wichtigen Bestandteil der Softwarekomponenten stellt das Echtzeitbetriebssystem (Real Time Operating System (RTOS)) dar, welches die einzelnen Tasks ausführt. Weiters ermöglicht das es den Zugriff auf die Schnittstellen zur Kommunikation zwischen den Kernen. [Siehe Kapitel 9.1.3]

Zur Konfiguration des Betriebssystems (RTOS) dient das für Eclipse verfügbare „RT-Druid“ Plugin [18]. Darin werden alle Tasks sowie deren Eigenschaften definiert und sowie deren Ressourcen eingestellt. Der Konfigurator definiert auch Alarme, Ereignisse, gemeinsame Ressourcen, sowie Interrupt Service Routines (ISRs). [Siehe Kapitel 9.1.4]

Die Basis-Software (BSW) beinhaltet Konfigurationen welche die Schnittstellen zur eigentlichen Hardware betreffen. Dazu zählt z.B. die Initialisierung des AD-Wandlers (ADC), der digitalen Ein- und Ausgänge sowie digitaler Busschnittstellen. Die BSW leitet sich di-

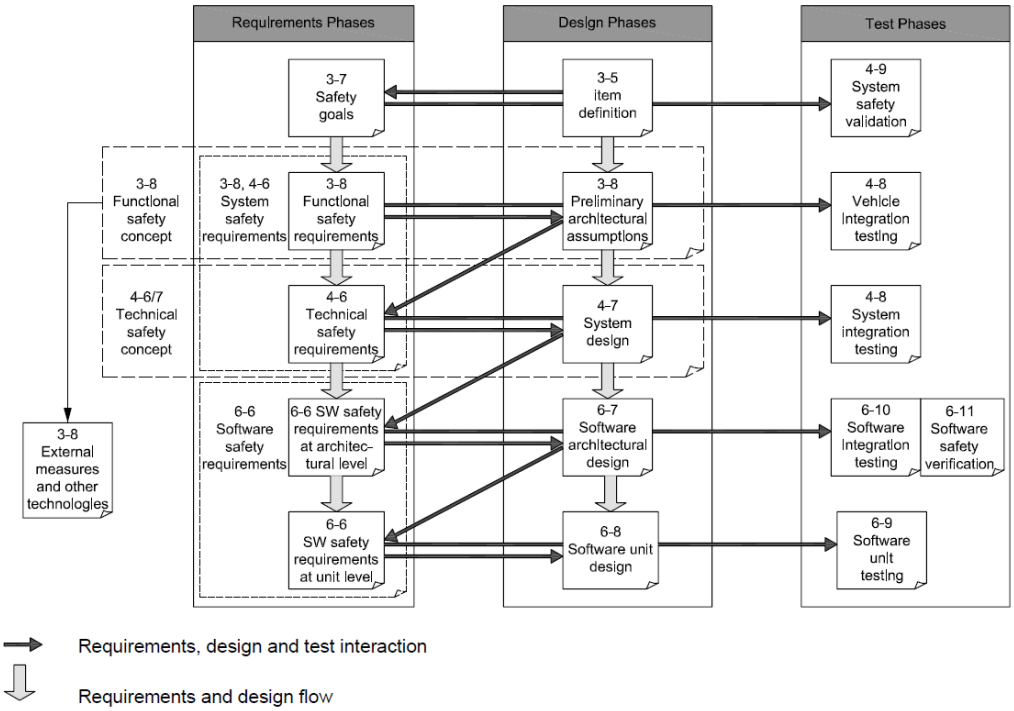


Abbildung 9.1: Softwareentwicklung eingebettet in den Gesamtentwicklungsprozess aus [42]

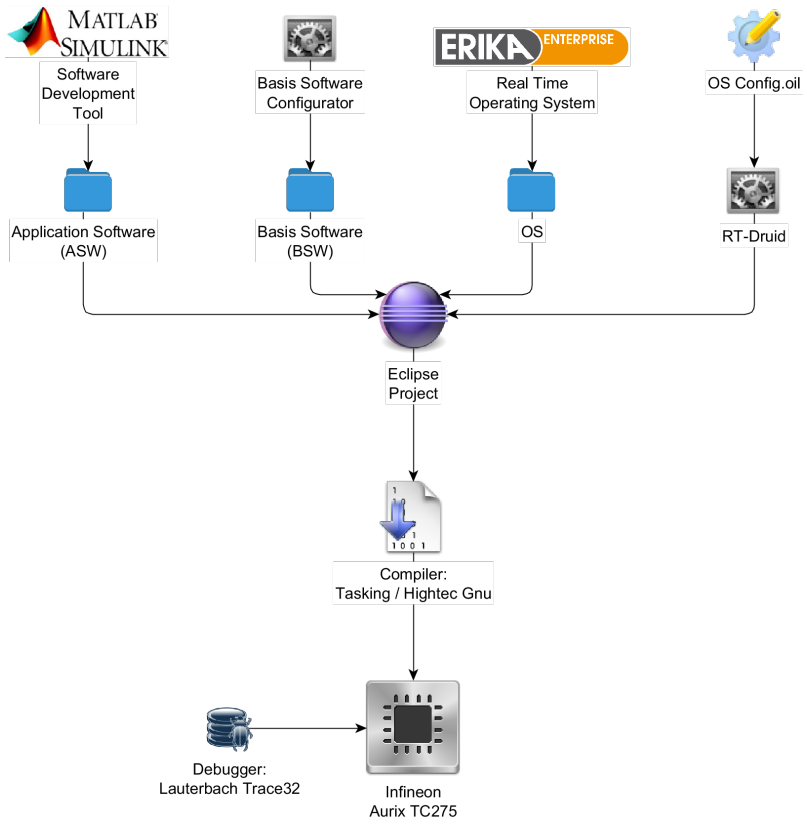


Abbildung 9.2: Verwendete Entwicklungswerkzeuge

rekt aus dem Hardware-Software-Interface (HSI) ab und verwendet meist eine „Low Level Driver“-Bibliothek, um die Entwicklung zu erleichtern. [Siehe Kapitel9.1.2]

Die Anwendungs-Software (ASW) wird über eine modellbasierte Entwicklungsumgebung erstellt und im Anschluss via Code-Generator in kompilierbaren Quelltext umgewandelt. [Siehe Kapitel9.1.1] Die Modell-basierte Entwicklung sicherheitskritischer Anwendungen von der Anforderung zur Implementierung wird in [52, 53, 46] behandelt, wobei in [79] speziell auf Verwendung von OSEK eingeht.

Zur Kompilierung wird der Altium Tasking Compiler 4.3R1 verwendet. [1] Dieser erstellt aus dem Quelltext die Binärdatei, welche über ein Flashingprogramm auf die Hardware überspielt werden kann. Zum Debuggen wird über die JTAG-Schnittstelle auf die CPU zugegriffen.

Abbildung 9.2 zeigt die verwendeten Werkzeuge, welche im Folgenden näher beschrieben werden.

9.1.1 Anwendungssoftware (ASW)

Die Anwendungssoftware wird als Matlab-Simulink Modell entwickelt. Dabei werden die Ein- und Ausgangsgrößen definiert und die Anwendung im Stil eines Flussdiagramms erstellt. Zur Konfiguration wird ein Matlab Skript verwendet, welche alle notwendigen Parameter definiert. Zur Codeerstellung wird das „Embedded Coder“ Modul [57] verwendet. Dieses erstellt funktionsfähige Initialisierungs-, Step- und Terminierungsfunktionen aus dem Modell. Ein Vorteil der modellbasierten Entwicklung ist die schnelle Testbarkeit der Anwendung durch eine sogenannte Model-in-the-Loop Technik, in welcher das Modell in einem übergeordneten Modell eingebunden und getestet wird.

9.1.2 Basissoftware (BSW)

Die Basissoftware initialisiert die verwendete Hardware und stellt die Schnittstelle zu selbiger zur Verfügung. Infineon liefert hierfür ein sogenanntes Infineon Low Level Driver (iLLD) Paket, über welches die Funktion aller Module im Mikrocontroller initialisiert und konfiguriert werden. Ein zweites Paket, welches die Sicherheitsfunktionen der Module aktivieren und konfigurieren soll, ist zum gegenwärtigen Zeitpunkt nicht erhältlich.

9.1.3 Betriebssystem (RTOS)

Das Betriebssystem stellt eine der wichtigsten Komponenten der Software dar. Es verwaltet die einzelnen Aufgaben und Ressourcen und muss die Echtzeitfähigkeit des Systems gewährleisten. Das bedeutet, dass die auftretenden Verzögerungs- und Bearbeitungszeiten ein definiertes Limit nicht überschreiten dürfen.

Als Betriebssystem wurde das freie Echtzeitbetriebssystem „ERIKA Enterprise“ [17] von Evidence verwendet. Dieses basiert auf dem OSEK Konzept und wurde 2014 auf dem Infineon AURIX portiert und für OSEK OS, OSEK COM und OSEK OIL zertifiziert. Im Zuge dieser Arbeit wurden einige Adaptionen am Betriebssystem vorgenommen, um die Unterstützung auf der gegebenen Hardware und des verwendeten Compilers unter Verwendung mehrerer Kerne zu ermöglichen. Auf die Portierung des Betriebssystems zur Verwendung für diese Arbeit wird hier nicht näher eingegangen.

9.1.4 Konfiguration der Laufzeitumgebung (RTE)

Zur Konfiguration der Laufzeitumgebung wird das Eclipse-Plugin „RT-Druid“ [18] verwendet. Dieses ist kompatibel mit ERIKA Enterprise und verwendet eine Syntax, welche aus dem OSEK/VDX Konzept stammt und für die Verwendung von ERIKA Enterprise und von Mehrkernarchitekturen erweitert wurde.

Zu Beginn des Buildprozesses wird die OIL-Konfiguration, welche die einzelnen Betriebssystemkomponenten beinhaltet, vom Konfigurator in Quelltext umgewandelt und anschlie-

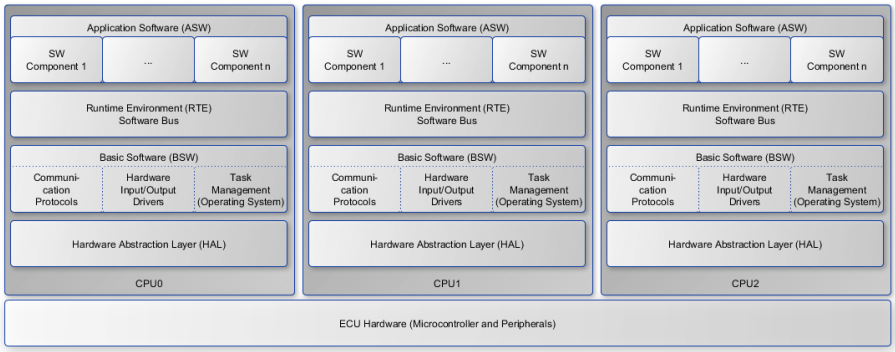


Abbildung 9.3: Konzept der Softwarearchitektur

ßend startet der eigentliche Kompilierprozess. Der von RT-Druid erzeugte Quelltext dient dem Betriebssystem zur richtigen Auswahl der benötigten Komponenten.

9.1.5 Aufbau der Softwarearchitektur

Da es sich um eine Mehrkernarchitektur handelt, muss auch die Softwarearchitektur daraufhin ausgelegt sein, um eine Mehrkernfähigkeit zu ermöglichen. In der OIL Datei wird jedem Objekt ein Rechenkern zugewiesen. Während des Kompilierprozesses werden über diese Konfiguration für jede CPU eine eigene Instanz des Betriebssystems mit eigener Basissoftware, der darunterliegenden Hardware-Abstrahierungsebene sowie der darüberliegenden Laufzeitumgebung erstellt. Darüber liegt die jeweilige Anwendungssoftware, welche auf die darunterliegenden Ebenen zugreift und die eigentliche Funktionalität der Software beinhaltet. Abbildung 9.3 zeigt die einzelnen Ebenen der Softwarearchitektur über die Rechnerkerne verteilt.

9.2 Sicherheitsanforderungen an die Software

Für jede zu entwickelnde Softwarekomponente müssen Anforderungen erstellt werden um die Einhaltung des Sicherheitsziels zu gewährleisten.

Die Sicherheitsanforderungen an die Software (hier SANFS genannt) sind in Tabelle 9.1 aufgelistet.

Da der Fokus dieser Arbeit auf der Betriebssicherheit des Software liegt, wurden Anforderungen zur Zuverlässigkeit und Verfügbarkeit nicht im Speziellen behandelt. Dennoch ist besonders im automotiven Bereich eine Untersuchung dahingehend wichtig.

Tabelle 9.1: Sicherheitsanforderungen an die Software

N°	Komp.	Sicherheitsanforderung
SANFS-01	Ebene 1	Alle Eingangsgrößen müssen auf Plausibilität geprüft und begrenzt werden.
SANFS-02	Ebene 1	Zur Ermittlung der Pedalposition wird der Mittelwert bei Einhaltung der Abweichungstoleranz verwendet.
SANFS-03	Ebene 1	Zur Ermittlung der Klappenposition wird der Mittelwert bei Einhaltung der Abweichungstoleranz verwendet.
SANFS-04	Ebene 1	Das Kippen einzelner Speicherzellen muss erkennbar sein.
SANFS-05	Ebene 1	Der Ausgangstreiber wird auf Fehler über die Statusleitung überprüft.
SANFS-06	Ebene 1	Gegen das gleichzeitige Aktivieren beider Richtungskanäle muss abgesichert werden.
SANFS-07	Ebene 1	Als Fehlerreaktion wird eine Deaktivierung des Ausgangstreibers veranlasst.
SANFS-08	Ebene 1	Alle Ausgangsgrößen müssen auf Plausibilität geprüft und begrenzt werden.
SANFS-09	Ebene 1	Konfiguration und Überprüfung der Sicherheitsfunktionen der Hardwaremodule.
SANFS-10	Ebene 2	Sicherheitskritische Eingangsgrößen müssen auf Plausibilität geprüft und begrenzt werden.
SANFS-11	Ebene 2	Sicherheitskritische Eingangsmodule müssen überwacht werden.
SANFS-12	Ebene 2	Sicherheitskritische Ausgangsmodule müssen überwacht werden.
SANFS-13	Ebene 2	Ein Abschalttest der sicherheitskritischen Abschaltpfade muss zum Systemstart durchgeführt werden.
SANFS-14	Ebene 2	Die Klappenposition muss mit der Pedalposition plausibilisiert werden.
SANFS-15	Ebene 2	Ebene 2 soll diversitär zu Ebene 1 entwickelt werden.
SANFS-16	Ebene 2	Als Fehlerreaktion wird eine Deaktivierung des Ausgangstreibers veranlasst.
SANFS-17	Ebene 2	Das Kippen einzelner Speicherzellen muss erkennbar sein.
SANFS-18	Ebene 3	Das Überwachungsmodul sendet periodisch (mind. 20Hz) eine Frage, welche von der Überwachungssoftware innerhalb der Periode zu beantworten ist.
SANFS-19	Ebene 3	Die verwendeten Fragen haben eine Wiederholrate von mind. 1000 Zyklen.
SANFS-20	Ebene 3	Die Fragen werden von der Überwachungssoftware auf die sicherheitskritischen Kerne verteilt, welche diese in richtiger Reihenfolge bearbeiten müssen.
SANFS-21	Ebene 3	Bei der Beantwortung der Fragen werden sowohl der Programmablauf als auch die Ausführung der CPU auf Richtigkeit getestet.
SANFS-22	Ebene 3	Zur Überprüfung des Überwachungsmoduls werden zyklisch Falschantworten gesendet welche eine Wiederholung der Frage zur Folge haben.

SANFS-23	Ebene 3	Als Fehlerreaktion wird eine Deaktivierung des Sicherheitsschalters veranlasst.
SANFS-24	Ebene 3	Ein Abschalttest der sicherheitskritischen Abschaltpfade muss zum Systemstart durchgeführt werden.
SANFS-25	RTOS	Zeit-überschreitende Tasks müssen terminierbar sein.
SANFS-26	RTOS	Unerlaubte Speicherzugriffe müssen verhindert werden.
SANFS-27	RTOS	Sicherheitskritische Tasks dürfen nicht durch unkritische Tasks unterbrochen werden.
SANFS-28	RTOS	Ein Blockieren durch Ressourcenbelegung muss verhindert werden.
SANFS-29	RTOS	Der Zugriff auf geteilte Ressourcen zwischen den RTOS-Instanzen muss geschützt werden.

9.3 Entwurf der Softwarearchitektur

Zum Entwurf der Softwarearchitektur wird die Systemarchitektur um die nötigen Softwarekomponenten erweitert. Die Softwarearchitektur legt die Implementierungsanforderungen an die Software fest, wobei alle Ebenen von der Hardwareabstraktionsebene über die Laufzeitumgebung bis hin zur Anwendungssoftware definiert werden.

Zum Start des Controllers wird lediglich CPU0 gestartet, die übrigen Kerne befinden sich noch im Halt-Modus. Nach der Initialisierung wird zunächst die Basissoftware konfiguriert und im Anschluss daran die Anwendungssoftware initialisiert. Danach startet der Master die beiden Slave-Kerne und die eigene Betriebssysteminstanz. Da es sich bei dem Betriebssystem um eine asynchrone Variante handelt, wird nur beim Start der CPUs eine Synchronisation durchgeführt. Nachdem die beiden Kerne ebenfalls Basissoftware und Anwendungssoftware initialisiert haben, wird das Betriebssystem gestartet und nach der Synchronisation arbeitet jede Betriebssysteminstanz separat (Abbildung 9.4).

Vom Betriebssystem werden entsprechend der Konfiguration in der OIL Datei zeitgesteuert die jeweiligen Tasks aufgerufen, welche die Funktionen des Systems sowie der einzelnen Sicherheitsebenen ausführen.

Dabei wurden für die einzelnen OS-Instanzen (bzw. Rechnerkerne) folgende Funktionen zugewiesen:

CPU0 beinhaltet als Funktionsrechner die eigentliche Steuerung der Drosselklappe in Ebene 1 (TaskCpu0_5ms). [Siehe Kapitel 9.4.1]

Da diese Steuerung von der Aktivierung durch den Benutzer abhängt, wird ein ereignisgetriggerter Task (TaskCpu0Motor Control) verwendet. Dieser empfängt über Betriebssystem-Ereignisse (Events) Befehle zur Aktivierung bzw. Deaktivierung der Steuerung und gibt diese entsprechend weiter.

Um den internen Watchdog zu triggern, wird ein niedrig priorisierter, periodischer Task (TaskCpu0_WDT) verwendet. Dieser sendet im 10ms Takt ein Signal an das

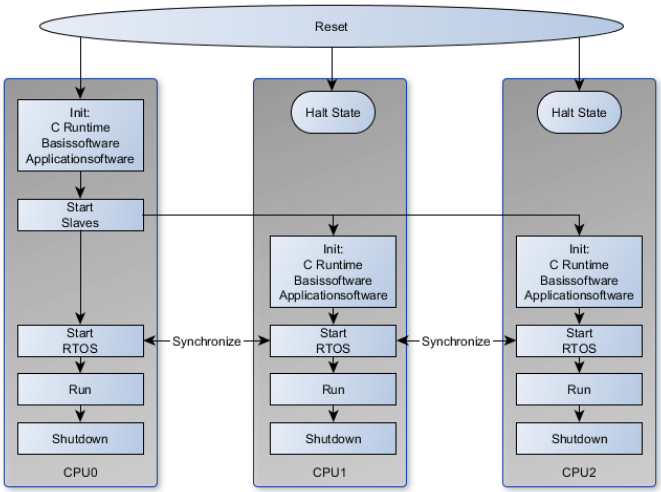


Abbildung 9.4: Start des Softwaresystems

Sicherheitsmodul, um die ordnungsgemäße Abarbeitung der CPU zu bestätigen. Im Falle einer Überschreitung der erlaubten Zeit zur Triggerung leitet das interne Sicherheitsmodul den sicheren Zustand ein, indem das Steuergerät abgeschaltet wird.

Zuletzt befindet sich noch ein Task (TaskCpu0_lvl3_ChainLink) zur Programmablaufkontrolle und zur Abarbeitung des Instruktionstests auf dieser Betriebssysteminstanz (Sicherheitsebene 3). [Siehe Kapitel 9.4.3.2]

CPU1 Auf CPU1, dem Funktionsüberwachungsrechner, findet sich der Task zur Funktionsüberwachung (TaskCpu1_10ms). In diesem wird unabhängig von CPU0 die Pedalposition sowie die Drosselklappenposition über das ADC-Modul eingelesen und auf Plausibilität zueinander geprüft. Um diese Plausibilisierung vornehmen zu können, werden ebenfalls der Status der Ausgangsstufe sowie der Aktivierungszustand der Steuerung überwacht. [Siehe Kapitel 9.4.2]

Wie auf CPU0, läuft auch auf dieser CPU ein niedrig priorisierter Watchdog Task (TaskCpu1_WDT), welcher dem internen Sicherheitsmodul die Einhaltung der Ausführungszeiten periodisch bestätigt.

Des weiteren befindet sich die Software für die Sicherheitsebene 3 auf CPU1. (TaskCpu1_lvl3_SW) Diese liest über den SPI Bus eine Fragepaket vom Überwachungsmodul und speichert dieses im gemeinsamen Speicher. Daraufhin greifen die einzelnen Level 3 „Chainlink“ Tasks (für CPU1: (TaskCpu1_lvl3_ChainLink)) in richtiger Reihenfolge zu und bearbeiten das Datenpaket, bevor über den SPI Task das entstandene Antwortpaket vorab geprüft und zurückgesendet wird. [Siehe Kapitel 9.4.3.2]

CPU2 wurde zunächst als sicherheitsunkritische Benutzerschnittstellensteuerung konzipiert. Auf diesem Kern läuft ein interruptgesteuerter Task (TaskCpu2_MotorControl), welcher die Eingabe am Touchscreen entgegennimmt und

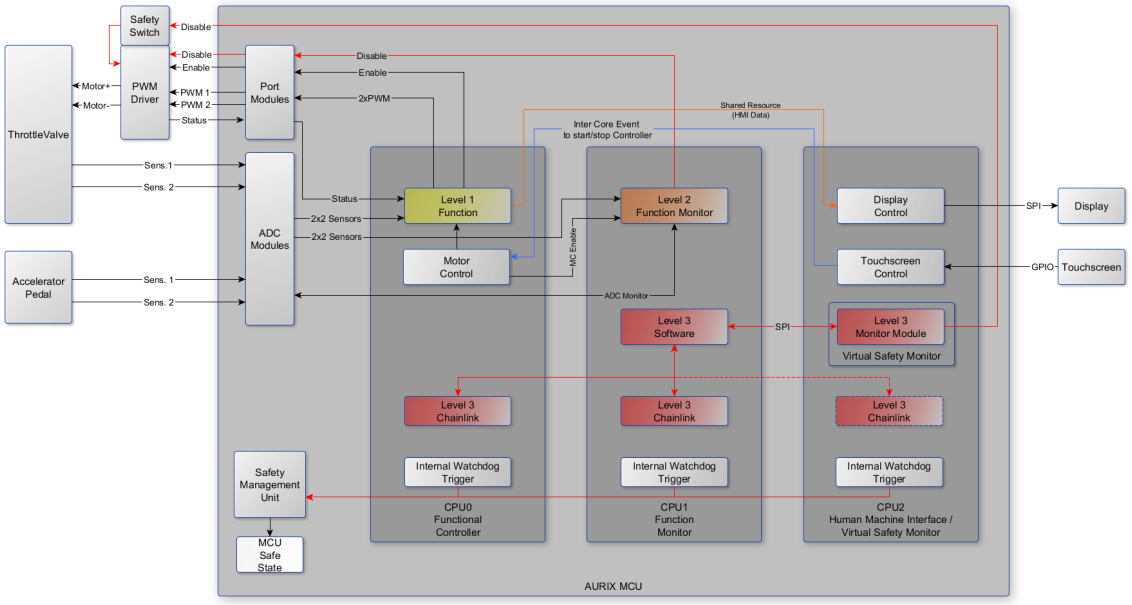


Abbildung 9.5: Verteilung der Tasks sowie Ansteuerung der Peripherie

das entsprechende Ereignis an den Motorcontrol Task auf CPU0 sendet. Weiters befindet sich noch der Display Task (TaskCpu2_50ms) auf CPU2. Dieser liest Daten vom gemeinsamen Speicher, welche vom Funktionstask auf CPU0 zur Verfügung gestellt werden, aus und berechnet daraus die Anzeige für das Display welches via SPI Bus verbunden ist.

Wie auf CPU0 läuft auch auf dieser CPU ein niedrig priorisierter Watchdog Task (TaskCpu2_WDT), welcher dem internen Sicherheitsmodul die Einhaltung der Ausführungszeiten periodisch bestätigt.

Um eine agile Entwicklungs- und Testumgebung zu schaffen, wurde auf einen dedizierten Controller als Überwachungsmodul verzichtet und diese Funktion (TaskCpu2_lvl3_HW) in CPU2 integriert. Zwar divergiert diese Entscheidung mit dem erstellten Sicherheitskonzept, da eine Unabhängigkeit von Spannungs- und Taktversorgung, sowie eine Entkoppelung thermischer Abhängigkeiten nicht mehr gegeben ist, dennoch soll es zum Zwecke der Demonstration genügen. Diese Wahl ermöglicht zudem weiterführende Tests mit Fehlerindizierungen, welche bei Einsatz der dedizierten Hardware nicht in dem Umfang und mit der Agilität möglich wären. Da die Kommunikation über den SPI Bus stattfindet, kann das virtuelle Überwachungsmodul leicht auf dedizierte Hardware übertragen werden. [Siehe Kapitel 9.4.3.1]

Abbildung 9.5 zeigt den Mikrocontroller mit den 3 CPUs auf welchen je eine Betriebssysteminstanz ausgeführt wird. Darin eingezeichnet sind die jeweiligen Tasks welche die Funktionen der drei Ebenen ausführen. Weiters sind die jeweiligen Signalpfade von den Eingangsmodulen über die Softwarekomponenten zu den Ausgangsmodulen eingezeichnet.

Das Betriebssystem stellt Ereignisse, sowie geteilte Speicherbereiche zur Verfügung, welche zur „Inter-Core-Communication“ verwendet werden, worüber Informationen zwischen den einzelnen Instanzen ausgetauscht werden.

Zusätzlich zum 3-Ebenen Sicherheitskonzept wird, wie bereits beschreiben, über dedizierte Tasks die Controller-interne Sicherheitsüberwachungseinheit angesteuert, welche im Fehlerfall den Controller in einen sicheren Zustand schaltet. Dieselbe Überwachungseinheit übernimmt auch die Reaktion auf Fehler welche durch Sicherheitsmechanismen in Hardwaremodulen implementiert sind. Die Aktivierung der jeweiligen Fehlererkennungen [siehe Kapitel 8.2.1.4] in den Hardwaremodulen und die Konfiguration dementsprechender Fehlerreaktionen (von der Abschaltung einzelner Ausgänge bis hin zum Systemreset) war zum gegenwärtigen Zeitpunkt nicht möglich, da die entsprechende Sicherheitsbibliothek vom Controllerhersteller nicht verfügbar war.

9.3.1 Scheduling der einzelnen Controllerkerne

Das Scheduling der einzelnen Tasks auf den Kernen nimmt eine wichtige Rolle ein. Bereits im Entwicklungsprozess wird festgelegt, welcher Task von welchem Kern bearbeitet wird (partitioniertes Scheduling), im Gegensatz zum globalen Scheduling, bei welchem die Abarbeitung dynamisch vom Betriebssystem auf einen Kern zugewiesen wird. Verschiedene Herangehensweisen zum Scheduling, besonders im automotiven Bereich und für sicherheitskritische Anwendungen, wird in [19, 20, 25, 45, 48, 76] näher behandelt.

Das OSEK/VDX Konzept verwendet, wie in Kapitel 4 beschrieben, ein Prioritätsschema zur Einteilung der Taskabfolge. Um einen fehlerhaften Programmablauf zu erkennen, werden der interne Watchdog-Task sowie der Task für Ebene 3 möglichst niedrig priorisiert. Dies hat zur Folge, dass an diesen Tasks eine Überschreitung der Laufzeit zuerst feststellbar ist, da höherpriorisierte Tasks deren Ausführung verhindern. Lediglich der Task zur Displayansteuerung findet sich durch die lange Ausführungszeit unter deren Prioritäten, was jedoch vertretbar ist, da es sich nicht um eine sicherheitskritische Funktion handelt.

Während die Tasks von Ebene 1 und 2 alle $5ms$ aufgerufen werden, um eine schnelle Steuerung zu ermöglichen, sowie um auf Fehler rechtzeitig reagieren zu können, wird die Rechnerüberwachung in Ebene 3 alle $50ms$ aufgerufen. Dies bringt einen akzeptablen Kompromiss zwischen Lastminimierung und Reaktionsschnelligkeit im Fehlerfall.

Die gewählte Konfiguration der einzelnen Tasks mit den zugeordneten Prioritäten, Wiederholraten, zugeordneten Ereignissen und Ressourcen sowie deren Scheduling-Einstellungen sind in der OIL-Konfigurationsdatei in Anhang A.6 zu finden.

In den folgenden Tabellen 9.2, 9.3 und 9.4 sind die geplanten Taskabläufe abgebildet. Anzumerken ist, dass die Priorität in der Tabelle (je CPU) nach oben hin steigt, und die Tasklaufzeiten geschätzt wurden. Befindet sich ein Task im Wartezustand, kennzeichnet das eine blaue Linie.

Tabelle 9.2: Schedulingentwurf auf Cpu0 (Priorität steigend)

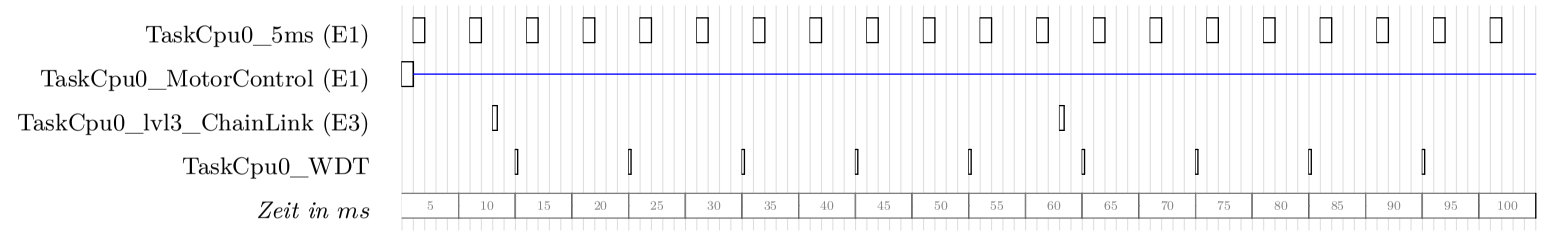


Tabelle 9.3: Schedulingentwurf auf Cpu1 (Priorität steigend)

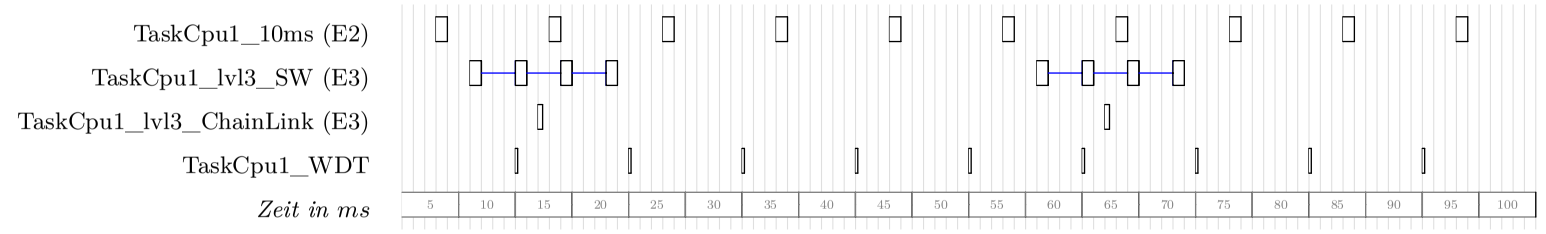
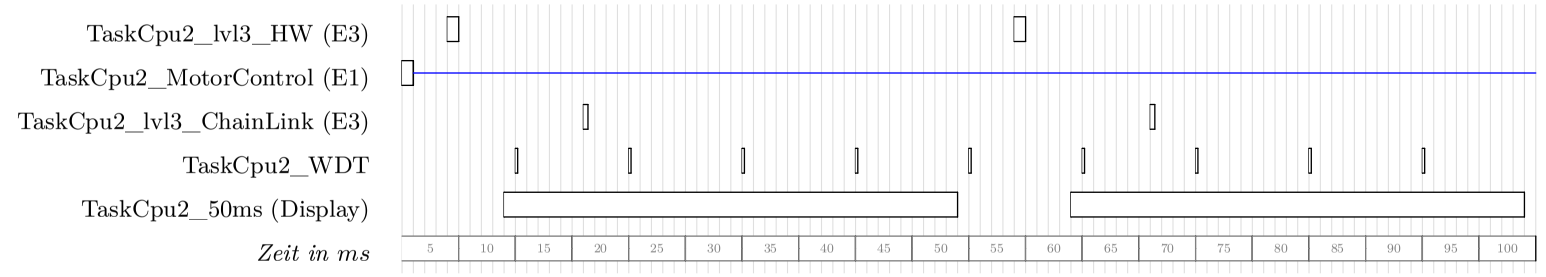


Tabelle 9.4: Schedulingentwurf auf Cpu2 (Priorität steigend)



9.4 Entwurf und Implementierung der Softwarefunktionen

Der Entwurf der Softwarefunktionen fokussiert sich auf die 3 Sicherheitsebenen. Diese wurden als Matlab Simulink Modell entwickelt und durch ein Matlab Skript parametrisiert. Dies bringt den Vorteil, dass eine schnelle Testbarkeit ermöglicht wird sowie eine erhöhte Übersichtlichkeit, woraus gleichzeitig eine Erhöhung der Sicherheit gegenüber systematischer Fehler resultiert.

Alle drei Ebenen stellen die Anwendungssoftware dar und hängen somit von der korrekten Ausführung der darunterliegenden Softwareebenen ab.

9.4.1 Funktion (Ebene 1)

Das Modell für die Funktionsebene (dargestellt in 9.6) verwendet die Rohwerte der beiden Pedalsensoren sowie deren Versorgungsspannung als Referenzgröße. Diese werden auf Plausibilität geprüft, gefiltert und als begrenzter Wert zur die Positionsvorgabe für den Regler der Klappenposition verwendet. Ebenso werden zur Ermittlung der Drosselklappenposition beide Sensorsignale sowie deren Versorgungsspannung eingelesen und eine Plausibilisierung vorgenommen. Der gefilterte und begrenzte Wert entspricht dabei dem Ist-Wert für den Positionsregler. Im Falle eines Fehlers während der Signalaufbereitung wird eine Fehlerreaktion aufgerufen, welche bei Überschreitung der internen Fehlergrenze den Regler und somit die Ansteuerung der Drosselklappe deaktiviert.

Als letzter Eingangsparameter für Ebene 1 wird das Freigabesignal vom Human-Machine-Interface (HMI) eingelesen. Nur bei Aktivierung durch den Anwender wird die Regelung der Drosselklappe aktiviert. Um eine Anzeige der Positionen am Display zu ermöglichen, werden neben der Ausgangsgröße für die Drosselklappe auch die gefilterten Positionssignale ausgegeben. Auch das Fehlersignal wird in mehreren Größen ausgegeben.

9.4.2 Funktionsüberwachung (Ebene 2)

Die Funktionsüberwachung auf CPU1 (Sicherheitsebene 2) verwendet dieselben Eingangssignale wie die erste Ebene, diese werden jedoch in einem separaten Zyklus vom AD-Wandler abgerufen. Zusätzlich wird der Status der Benutzersteuerung abgefragt, um eine fehlerhafte Plausibilisierung bei ausgeschaltetem Zustand zu unterbinden. Nach einer Bereichswertüberprüfung werden die Positionen von Pedalwertgeber und Drosselklappe verglichen und so die richtige Funktion von Ebene 1 bewertet. Durch die physikalische Messung der Drosselklappenposition wird eine Funktionsüberwachung des Ausgangsmoduls, der Leistungsstufe inklusive Sicherheitsschalters, sowie des Motors selbst mit durchgeführt, da eine Abweichung der gemessenen Ist-Position von der Soll-Position vom Gaspedal auf einen Fehler der Übertragungskette schließen lässt.

Weiterhin sicherheitskritisch und somit zu überwachen sind die AD-Wandler, welche den physikalischen Wert in einen digitalen Wert umwandeln. Um die Funktionsfähigkeit der Module zu überprüfen, werden an je einem Eingang vor dem Multiplexer abwechselnd zwei

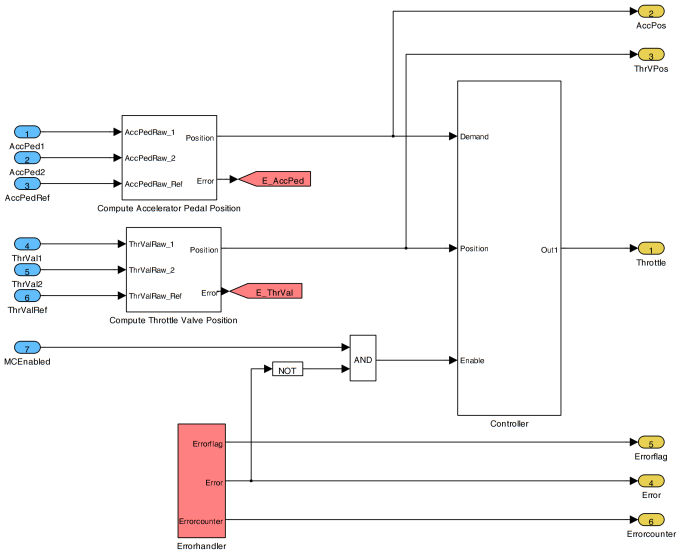


Abbildung 9.6: Softwareimplementierung der Ebene 1

verschiedene Pegel angelegt und somit ein Rechtecksignal erzeugt. In Sicherheitsebene 2 wird daraus der Mittelwert gebildet um so ein „Einfrieren“ des Multiplexers, bzw. des AD-Wandlers zu erkennen.

Bei einer Fehlererkennung wird durch den Errorhandler eine Filterung vorgenommen und bei Überschreitung der Fehlergrenze der sichere Zustand durch Abschalten der Leistungsstufe eingeleitet.

Das Modell der Funktionsüberwachung ist in Abbildung 9.7 abgebildet.

9.4.3 Rechnerüberwachung (Ebene 3)

Ebene 3 besteht aus zwei Teilen, welche sich aus dem Softwaremodul am Funktionsrechner und dem Überwachungsmodul am Überwachungsrechner zusammensetzen. Beide Module kommunizieren über den SPI Bus miteinander, wobei die Frage vom Überwachungsmodul innerhalb eines gewissen Zeitrahmens richtig beantwortet werden muss, um eine Fehlerreaktion zu verhindern.

Das an das Softwaremodul übertragene Datenpaket besteht dabei aus einem Fragedatenwort, der Startidentifikationsnummer sowie der Fehleridentifikationsnummer. Das Paket wird vom Softwaremodul am Funktionsrechner entgegengenommen und im gemeinsamen Speicher abgelegt. Nun werden die einzelnen Submodule (Chainlinks) aufgerufen, welche das Datenpaket bearbeiten und wieder in den gemeinsamen Speicher zurückschreiben.

Zur Bearbeitung wird das Datenwort mittels entsprechendem Algorithmus verändert (hier: Pseudozufallsgenerator), welcher den Instruktionstest nachstellt. Gleichzeitig wird vom Modul die Identifikationsnummer bei richtigem Eingangswert erhöht, ansonsten wird

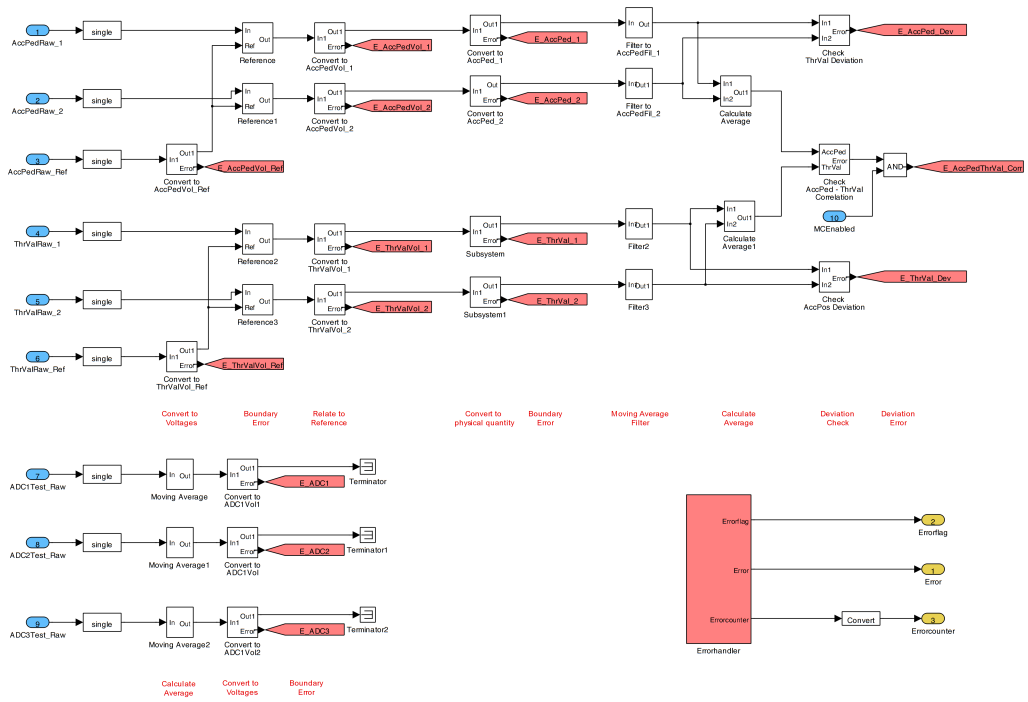


Abbildung 9.7: Softwareimplementierung der Ebene 2

die ID des Submoduls als Fehleridentifikationsnummer im Datenpaket gespeichert. Anschließend wird an das Softwaremodul die fertige Abarbeitung mitgeteilt. Sind alle sicherheitskritischen Kerne überprüft worden, schickt das Softwaremodul die Antwort via SPI zurück an das Überwachungsmodul. Dieses vergleicht alle drei Parameter auf Übereinstimmung mit den erwarteten Werten und sendet bei korrekter Antwort ein verändertes Datenpaket an das Softwaremodul zurück.

Wird vom Überwachungsmodul ein Fehler festgestellt, wird der interne Fehlerzähler erhöht und dem Softwaremodul dasselbe Datenpaket mit erhöhter Fehler-ID erneut gesendet. Wird die Fehlergrenze überschritten, öffnet das Überwachungsmodul den Sicherheitsschalter und leitet so einen sicheren Zustand des Systems ein. Abbildung 9.8 zeigt den Ablauf der Rechnerüberwachung mit Überwachungsmonitor, Überwachungssoftware sowie den Testprogrammen (Chainlinks), welche die Integrität des Prozessors prüfen.

9.4.3.1 Überwachungsmodul

Das Überwachungsmodul liest die Antwort des Softwaremoduls ein und vergleicht diese mit der selbst berechneten Antwort, bzw. mit den erwarteten Werten. Stimmt die Antwort überein, wird eine neue Frage ausgegeben, im Fehlerfall kommt es zur Wiederholung der Frage. Dadurch kann die Fehlererkennung vom Softwaremodul aktiv getestet werden. Erst bei einer Überschreitung der Fehlergrenze leitet der Errorhandler eine Fehlerreaktion ein,

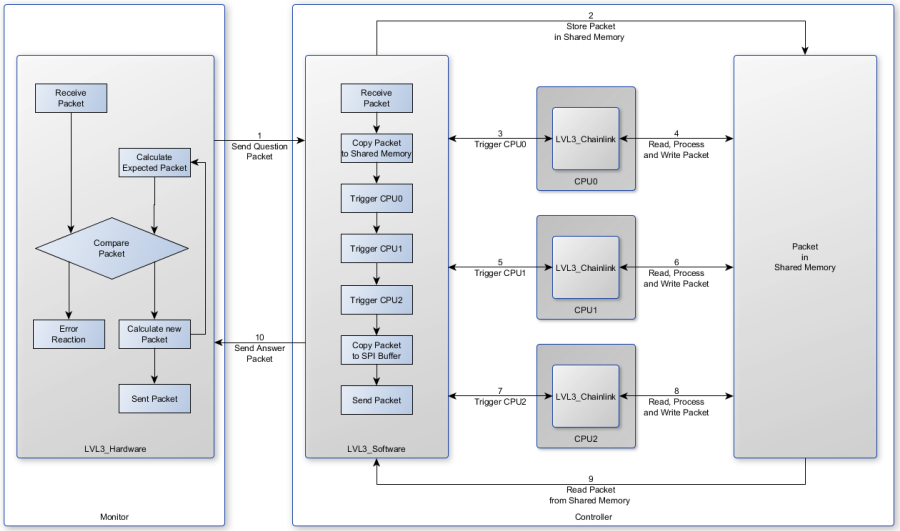


Abbildung 9.8: Programmablauf in Ebene 3

welche zum Abschalten des Sicherheitsschalters führt. Abbildung 9.9 zeigt das Modell des Überwachungsmoduls.

9.4.3.2 Softwaremodul

Das Softwaremodul besteht aus mehreren Teilen. Wird eine Frage empfangen, wird diese im geteilten Speicher abgelegt. Anschließend werden, wie oben beschrieben, die Tasks zur Überprüfung der CPU-Integrität getriggert, welche nun dieses Datenpaket entsprechend bearbeiten, wobei nicht nur die richtige Ausführung der Antwortberechnung, sondern auch die zeitgerechte Abfolge der einzelnen Tasks, welche auf den Rechenkernen verteilt sind (Level 3 Chainlinks - abgebildet in Abbildung 9.10), zur richtigen Antwort beiträgt. Jeder Task sendet ein Event nach erfolgreicher Abarbeitung an das Softwaremodul - welches wiederum bei vollständiger Bearbeitung aller Chainlinks das Antwortpaket zurück an das Überwachungsmodul sendet. Benötigen die Chainlink Tasks zu lange zum Senden des Events, wird ein Timeout am Softwaremodul ausgelöst und eine falsche Antwort an das Überwachungsmodul zurückgesendet.

Instruktionstest

Um die richtige Ausführung der einzelnen Instruktionen zu überprüfen, müssten auf jedem sicherheitskritischen Kern diejenigen Befehle ausgeführt und getestet werden, welche während des Steuerungsablaufs verwendet werden und welche eine Verletzung des Sicherheitsziels verursachen können. Um diesen Instruktionstest demonstrativ nachzubilden, wird ein Pseudozufallsgenerator verwendet, welcher mit einem begrenzten Instruktionsumfang

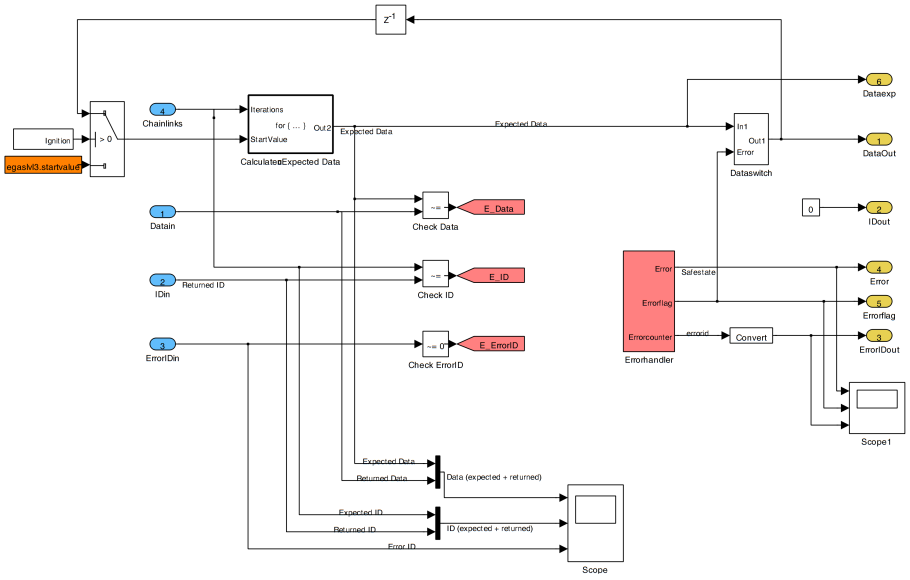


Abbildung 9.9: Softwareimplementierung der Ebene 3 auf dem Überwachungsmodul

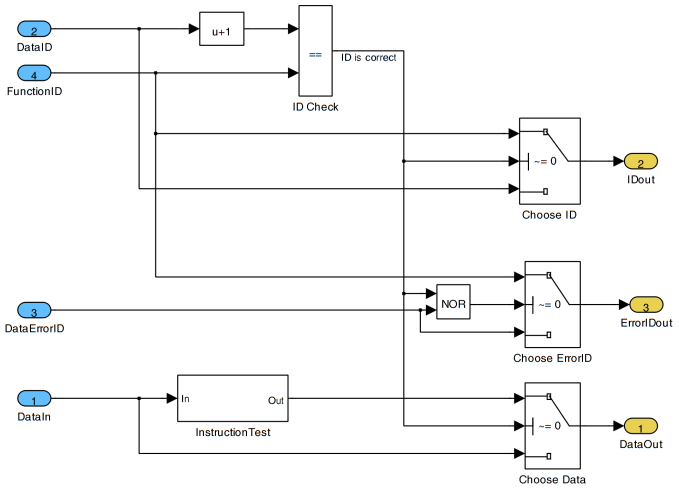


Abbildung 9.10: Softwareimplementierung der Ebene 3 auf dem Funktionsrechner

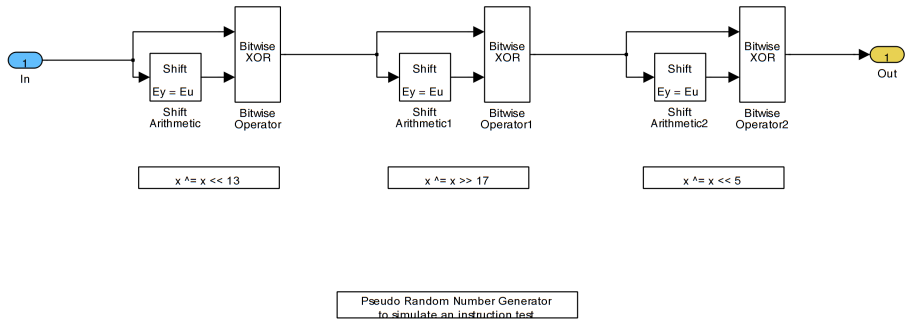


Abbildung 9.11: Softwareimplementierung des Instruktionstests

arbeitet und durch die hohe Streuung der Startwerte Registerfehler aufzeigen soll. Der verwendete Algorithmus ist in Abbildung 9.11 abgebildet. Es wurde ein Xor-Shift Verfahren [56] verwendet, welches einen Instruktionstest lediglich demonstriert, da während der Ausführung nur ein minimaler Bereich der verwendeten Befehle benötigt wird.

Mögliche Strategien zu Software-basierender Selbsttests, eine Behandlung über die Wirksamkeit der Erkennung permanenter und kurzzeitiger Fehler sowie mögliche Methoden zur periodischen Selbstüberprüfung während dem Betrieb werden in [22, 47, 64] behandelt.

9.5 Test der einzelnen Softwareteile

Um die einzelnen Modelle zu testen und zur Verifikation der Sicherheitsanforderungen, werden die beschriebenen „Model in the Loop“ Tests angewandt. Dabei wird das Modell der Sicherheitsebene in einem darüber liegenden Modell eingebettet, welches die Umgebung, in welcher das zu testende Modell später verwendet wird, nachbildet und die Reaktion auf gezielt verursachte Fehler überprüft. [7]

9.5.1 Funktions- und Funktionsüberwachungsebene

Zur Überprüfung der Funktions- und Funktionsüberwachungsebenen wurde sowohl das Gaspedal, als auch die Drosselklappe nachgebildet. Als Eingangsgrößen für die Pedalposition wurden mehrere Signalverläufe definiert, um verschiedene Fehlerszenarien darstellen zu können. Die simulierten Sensorwerte wurden als Eingangsgrößen für das zu testende Modell verwendet und dessen Reaktion auf Fehlerfälle in Sensorik und Aktuatorik wurden aufgezeichnet und auf Einhaltung der Sicherheitsanforderungen überprüft. Sicherheitsebene 2 wurde ebenfalls mit der selben Methodik auf Funktionsfähigkeit überprüft. (Abbildung 9.12)

Abbildung 9.13 zeigt eine Simulation der Ebenen 1 und 2. Dabei wird während des Betriebs der Ausfall eines Sensorsignals im Gaspedal (dargestellt in Diagramm 2) sowie darauf der Ausfall eines Drosselklappensensors (Diagramm 1) nachgestellt. In Diagramm 3

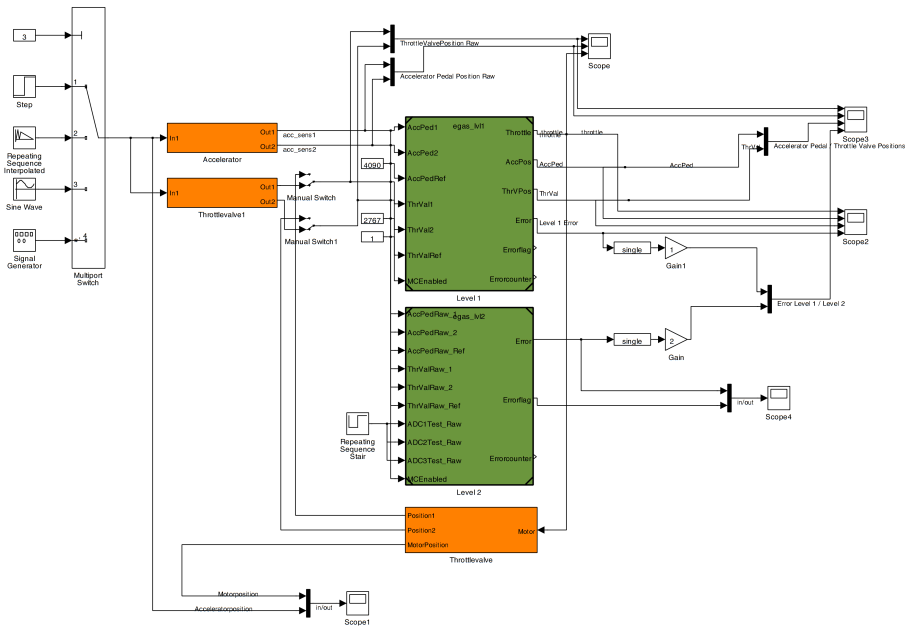


Abbildung 9.12: Testmodell für Ebene 1 und 2

finden sich die daraus berechneten Positionen der beiden Geräte, welche zur Verwendung für die Benutzerschnittstelle vorgesehen sind. In der letzten Reihe sind die Fehlersignale beider Ebenen dargestellt. In der Abbildung ist gut zu erkennen, wie sich ein Fehler bereits auf die Berechnung der Positionswerte auswirkt, bevor die Bestätigung des Fehlers erfolgt und ein definierter Wert für das jeweilige Positionssignal ausgegeben wird. Weiters lässt sich die zeitliche Verschiebung der beiden Fehlerreaktionen zwischen den beiden Ebenen erkennen.

9.5.2 Rechnerüberwachungsebene

In Ebene 3 wurde das Überwachungsmodul mit mehreren Instanzen des Softwaremoduls (bzw. der Chainlinks) verbunden und verschiedene Fehlerfälle wie der Ausfall eines Moduls, Reihenfolgenfehler sowie Berechnungsfehler nachgebildet und die Reaktion darauf überprüft (Abbildung 9.14).

Abbildung 9.15 zeigt eine Simulation auf Ebene 3. Dabei wurden Fehler mit zunehmender Anzahl injiziert und das Verhalten am Überwachungsmonitor beobachtet. Man erkennt, dass bei Einzelfehlern keine Fehlerreaktion durch das Modul ausgelöst wird sowie dass der Fehlerzähler bei korrekter Antwort durch die Überwachungssoftware wieder abnimmt. Mit steigender Fehleranzahl wird jedoch bald jene Grenze überschritten, ab welcher das Überwachungsmodul eine Fehlerreaktion auslöst.

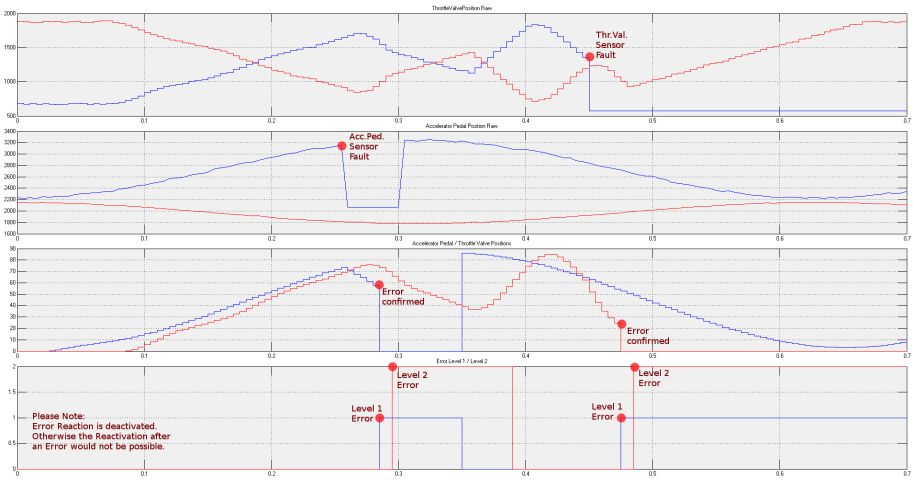


Abbildung 9.13: Diagramm einer Fehlersimulation in für Ebene 1 und 2

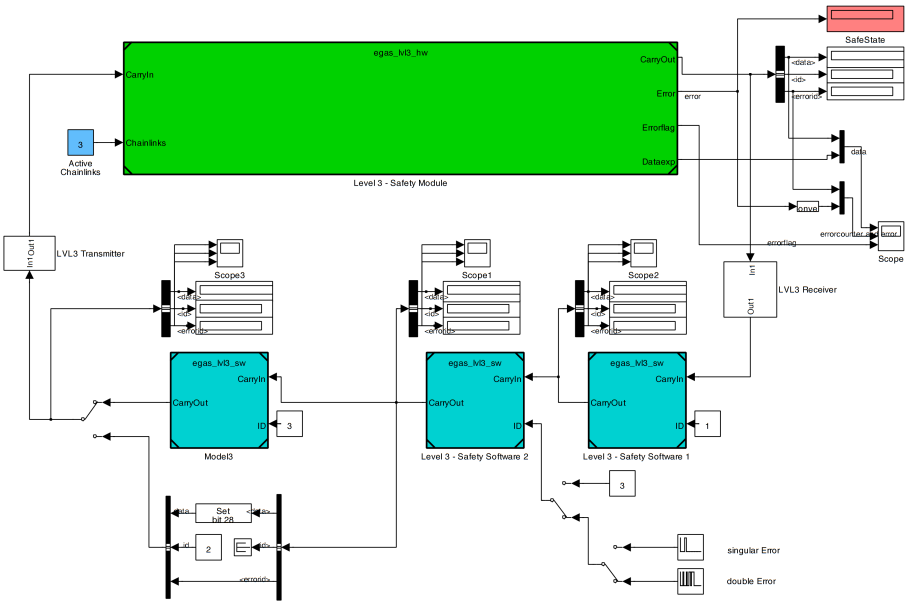


Abbildung 9.14: Testmodell von Ebene 3

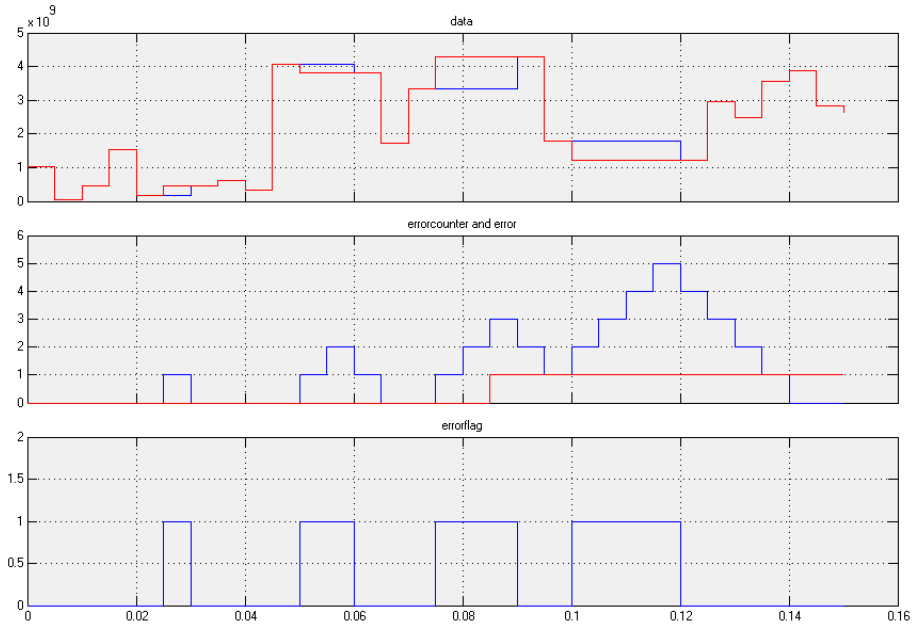


Abbildung 9.15: Diagramm einer Fehlersimulation auf Ebene 3

9.5.3 Abweichungen zur ISO Norm

Um dem Entwicklungsprozess nach der ISO 26262 zu entsprechen, is es nötig, nach jedem der beschriebenen Entwicklungsschritte eine Evaluierung zur Einhaltung der Sicherheitsziele sowie in Abhängigkeit von der Kritikalität des Sicherheitszieles bestimmte Verifizierungsmaßnahmen der jeweiligen Arbeitsergebnisse durchzuführen. Diese Maßnahmen lagen jedoch nicht im Fokus der Arbeit. Eine Auflistung der Arbeitsschritte zur Einhaltung der Norm findet sich in Anhang A.4.

Kapitel 10

Evaluierung der Implementierung

Nachdem das System in Hardware und Software implementiert und zu einem Gesamtsystem integriert wurde, wird dieses zur Einhaltung der ISO Norm hinsichtlich der Wirksamkeit der Sicherheitsanforderungen überprüft und verifiziert. Dazu wird auf System-, Hardware- und Softwareebene kontrolliert, ob die Sicherheitsanforderungen so umgesetzt wurden, dass sie im Fehlerfall den Fehler erkennen und entsprechend darauf reagieren. Die Überprüfung umfasst auch eine Kontrolle der externen sowie internen Schnittstellen und eine Analyse funktionaler Abhängigkeiten. Weiters sollen auch gemeinsame Auslöser für Fehler überprüft und verhindert werden. Durch ein gezieltes Einbringen von Fehlern wird die Effektivität von Sicherheitsmechanismen bewertet und die korrekte Implementierung der Sicherheitsanforderungen demonstriert [65].

Im Fall einer Verletzung der Sicherheitsanforderungen werden alle Entwicklungsschritte entsprechend dem V-Modell vom auslösenden Prozessschritt abwärts geändert und erneut durchlaufen, dies betrifft auch nachfolgende Prozessschritte und Prüfverfahren. Im Zuge dieser Arbeit wurde lediglich die Funktionalität des Sicherheitskonzepts sowie die Einhaltung der erstellten Sicherheitsanforderungen geprüft. Eine vollständige Verifizierung entsprechend der ISO 26262 wurde nicht weiter durchgeführt.

10.1 Systemevaluierung

Zur Evaluierung des Systems werden die einzelnen Ebenen durch gezieltes Einbringen von Fehlern getestet. Weiters wurde eine Analyse der Implementierung durchgeführt.

Abbildung 10.1 zeigt den Ablauf der einzelnen Tasks über alle drei Betriebssysteminstanzen. Zur Messung der Zeiten wurden sog. PreTaskHooks bzw. PostTaskHooks verwendet, um die Startzeit sowie die Stoppzeit der einzelnen Tasks aufzuzeichnen und in einem Array im RAM abzuspeichern. Der entstandene Zeitversatz durch diesen Mehraufwand für den Rechner wurde bei der Auswertung vernachlässigt, da dieser, im Vergleich zur Tasklaufzeit selbst, gering ist. Zum Auslesen der Daten wurde das Debugging-Tool verwendet und die Rohdaten wurden im Anschluss via Matlab-Skript grafisch aufbereitet. Dabei werden die einzelnen Tasks nach Priorität geordnet (steigende Position bedeutet höhere Priorität) und nach CPU gruppiert dargestellt.

Eine zeitliche Analyse gibt Aufschluss darüber, wie lange einzelne Tasks laufen und

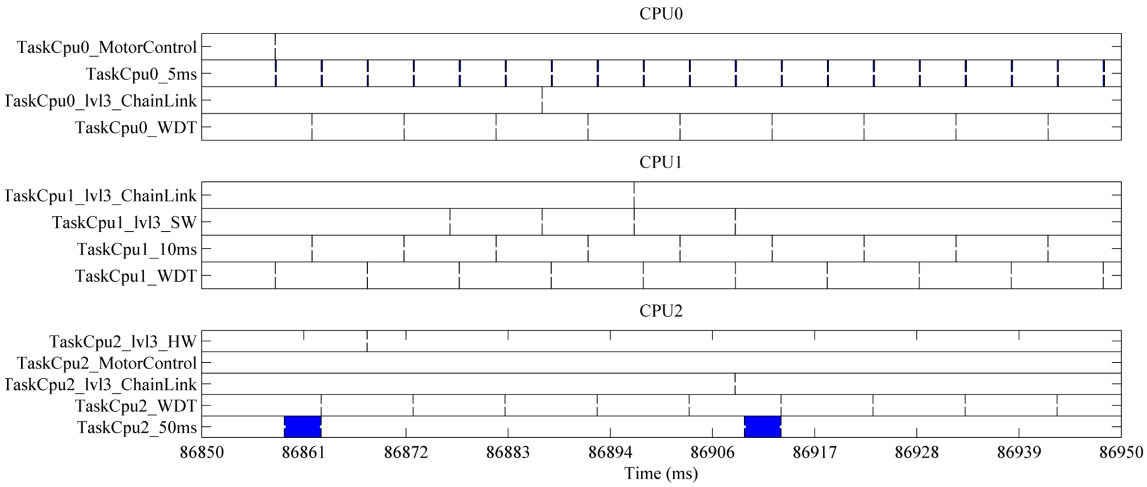


Abbildung 10.1: Messung der Taskabläufe auf den drei CPUs

gleichzeitig können über die Ausführungsreihenfolge der einzelnen Tasks Abhängigkeiten kontrolliert und auch Einzelfehler erkannt werden. Auf die Herausforderungen von Performance-Analysen, den Umgang von hierarchischen Ereignismodellen sowie die Eigenheiten von Zeitanalysen für Mehrkernplattformen wird in [72] eingegangen.

10.1.1 Funktionsebene (Ebene 1)

Aufgabe von Ebene 1 ist die Gewährleistung einer fehlertoleranten, zuverlässigen Regelung. In ihr werden die Eingangssignale plausibilisiert und zur Verarbeitung im Regelkreis verwendet. Durch die Ausführung auf einem separaten Kern wird ein Fehler gekapselt und kann sich nicht auf andere Ebenen fortpflanzen. Um eine stabile Regelung mittels PID Regler zu ermöglichen und für eine kurze Reaktionszeit sowie zur Optimierung der Signalfilterung wurde eine Periodendauer von $5ms$ gewählt. Wie Abbildung 10.1 zeigt, besitzt lediglich der Task zur Steuerung per Benutzerschnittstelle eine höhere Priorität (im Bild durch höhere Position dargestellt), um ein Abschalten durch den Bediener im Fall eines Fehlers im Funktionstask zu ermöglichen. Eine Überprüfung der Funktionsebene wurde im Zuge diese Arbeit auf Systemebene durch Simulation durchgeführt. [Siehe Kapitel 9.5.1]

Fehlererkennung

Ebene 1 erkennt lediglich Fehler der Eingangssignale, welche durch Plausibilisierung und Bereichswertbegrenzung festgestellt werden können. In der beschriebenen Implementierung wird die Funktion des Ausgangs nicht überwacht, da über die Drosselklappen-Sensoren eine Reaktion auf die Steuervorgabe erkennbar und plausibilisierbar ist.

Reaktionszeit

Die Regelung wird alle $5ms$ ausgeführt. Zur Erhöhung der Zuverlässigkeit wird eine Fehlerreaktion erst nach Bestätigung des Fehlers über einen Zeitraum von $20ms$ eingeleitet. Somit muss der Fehler über 4 Zyklen bestehen, bevor es zu einer Abschaltung des Ausganges kommt. Dadurch wird die Forderung zur Einhaltung der Reaktionszeit von $50ms$ (FSK-03.04 in Kapitel 6.2) erfüllt.

10.1.2 Funktionsüberwachung (Ebene 2)

Wie auch Ebene 1 profitiert diese Ebene von der separaten Ausführung auf einem eigenen Rechnerkern bzw. durch eine eigene Betriebssysteminstanz. Die durch Verlagerung der Überwachungsfunktion auf den separaten Kern entstandene Lastminderung fällt - wie die Zeitmessung ergab - in diesem Fall wenig ins Gewicht, da die Last bei einem Task mit der Periodendauer von $10ms$ und einer Laufzeit von wenigen μs verschwindend klein ist. Jedoch ist dieser Schritt bei Forderung nach sehr kurzen Reaktionszeiten (und demnach kurzen Periodendauern) in Bezug auf Lastverteilung sinnvoll, besonders bei einer Vielzahl von Applikationen - wie bei Motorsteuergeräten üblich. Viel wichtiger ist in diesem Fall die Möglichkeit auf einen lokalen Fehler am Rechnerkern, am Betriebssystem, oder an der Anwendungssoftware reagieren zu können, wenn dies bei Ausführung am selben Kern nicht mehr möglich ist. Auslöser davon können zufällige sowie systematische Hardwarefehler in der CPU, beziehungsweise systematische Fehler in der Software sein. Eine Überprüfung der Funktionsüberwachung wurde im Zuge dieser Arbeit auf Systemebene durch Simulation durchgeführt. [Siehe Kapitel 9.5.1]

Fehlererkennung

Ebene 2 erkennt die richtige Funktion von Ebene 1 dadurch, dass sie alle zur Funktion von Ebene 1 nötigen Komponenten überwacht und im Falle einer Fehlererkennung abschaltet. Dazu werden, wie auf Ebene 1, die Eingangssignale auf Plausibilität überprüft, wodurch sich Fehler an der Sensorik sowie Aktuatorik erkennen lassen. Um auch Fehler des AD Wandlers zu erkennen, wird dieser mit einem Rechtecksignal an einem Eingang angesteuert. Stoppt der Multiplexer bzw. der Wandler seine Funktion, wird das durch die Überprüfung des Testsignales erkannt. Die Funktion der Ausgangsstufe wird überprüft, indem der Soll-Wert des Gaspedals mit dem Ist-Wert der Drosselklappe abgeglichen wird und somit Fehler in der Funktion von Regler, Ausgangsmodul, Leistungsstufe und vom Motor selbst erkannt werden können.

Reaktionszeit

Um dem Regler die Möglichkeit zu geben, selbstständig auf Fehler zu reagieren, wird eine Fehlerreaktionszeit von $30ms$ eingestellt. Da die Überprüfung alle $10ms$ stattfindet, muss der Fehler 3 Zyklen lang bestehen, um eine Reaktion des Systems auszulösen. Dadurch

wird die Forderung zur Einhaltung der Reaktionszeit von $50ms$ (FSK-03.04 in Kapitel 6.2) erfüllt.

10.1.3 Rechnerüberwachung (Ebene 3)

In Ebene 3 sendet der Überwachungsmonitor alle $50ms$ ein Datenwort über die SPI Schnittstelle an den Funktionsrechner, welches sich aus einer numerischen Frage, einem Zähler, sowie einem Fehlerzähler zusammensetzt. Die Software am Empfänger nimmt die Daten entgegen und veranlasst die jeweiligen Kerne dazu, den Datensatz in richtiger Reihenfolge abzuarbeiten.

In einer ersten Implementierung erfolgte diese Abarbeitung komplett unabhängig voneinander durch einen zeitgetriggerten, versetzten Aufruf der Chainlink-Tasks. Da es sich bei dem verwendeten Betriebssystem jedoch um ein asymmetrisches Multiprozessorsystem handelt und die einzelnen Kerne nach der Initialisierung unabhängig voneinander arbeiten, kam es zu Zeitversätzen zwischen den einzelnen Instanzen. In Folge davon trat nach längerer Betriebszeit eine Verschiebung der Abarbeitungsfolge auf, sodass das Überwachungsmodul eine ungewollte Fehlerreaktion auslöste.

Diese Implementierung wurde deshalb von einer ereignisgesteuerten Aktivierung der einzelnen Bearbeitungsroutinen durch das Softwaremodul auf Ebene 3 ersetzt. Dabei löst die Schnittstelle zum Überwachungsmodul beim Empfang eines neuen Datenpakets den Aufruf der einzelnen Tasks auf den verschiedenen Kernen aus. Durch die niedrigste Priorisierung dieser Tasks auf den sicherheitskritischen Instanzen wird dieser Aufruf erst nach Fertigstellung aller vorrangigen Tasks aufgerufen, wodurch geprüft werden kann, dass die Bearbeitungsreihenfolge am System noch intakt ist.

Abbildung 10.2 zeigt den Ablauf der einzelnen Tasks über das gesamte System. Zunächst erzeugt der Task, welcher das Überwachungsmodul nachbildet, eine Frage (a) und übermittelt diese via SPI an den Empfängertask am Funktionsüberwachungskern (b). Dieser speichert das Datenpaket im gemeinsamen Speicher und veranlasst über Ereignisse die Abarbeitung der Daten durch die eingebundenen Kerne (c). Diese senden nach erfolgreicher Abarbeitung ein Ereignis an den Empfängertask zurück (d,e,f). Wurde vom Empfängertask kein Fehler in der Abarbeitung erkannt, werden die Antwortdaten in das SPI Modul zur Abarbeitung durch den Bus-Master (das Überwachungsmodul) abgelegt. (f)

Fehlererkennung

Durch das periodische Überprüfen der Funktionsfähigkeit der einbezogenen Kerne mittels dieser Abwandlung eines Challenge-Response-Verfahrens, kann festgestellt werden, ob die getriggerten Tasks vom jeweiligen Betriebssystem richtig verarbeitet wurden und ob eine Laufzeitüberschreitung durch höherpriorisierte Tasks aufgetreten ist.

Um die richtige Ausführung der für die Funktion des Systems verwendeten Befehle zu überwachen, müsste der verwendete Befehlssatz auch zur Erstellung der Antwort komplett miteingebunden werden. Da jedoch keine Untersuchung des umfassenden Befehls-

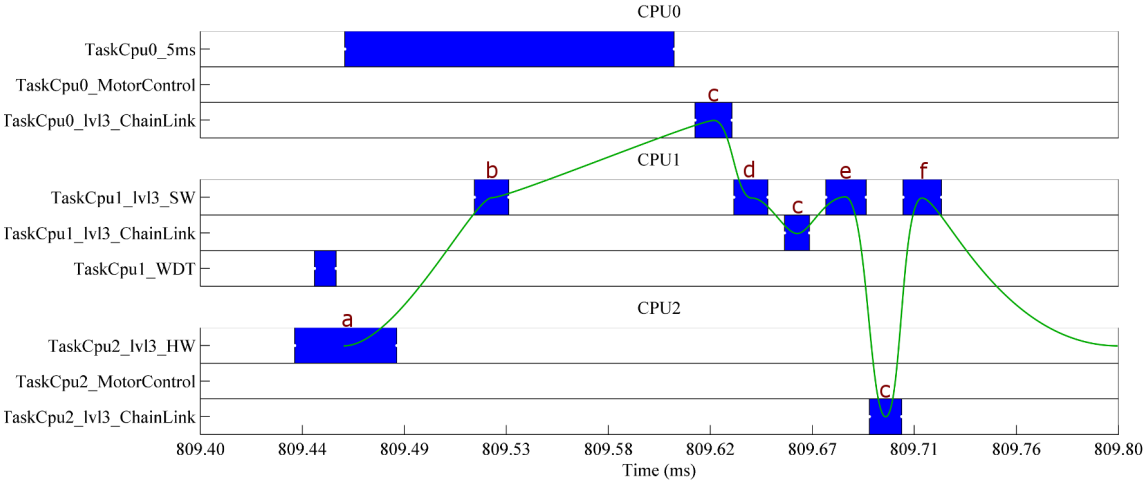


Abbildung 10.2: Messung des Ablaufs in Ebene 3

satzes durchgeführt wurde, wird die Integrität der Recheneinheit nur demonstrativ getestet. Die verwendete Implementierung demonstriert diesen Befehlssatztest mit Hilfe eines Pseudozufallszahlen-Generators auf Basis des XorShift [56]. Vorteil daran ist, dass die Register mit allen möglichen Bit-Kombinationen beschrieben werden sowie die geringe Lasterzeugung und Speichernutzung des Systems durch den Test. Durch den statischen Ablauf und die wenigen Instruktionen kann jedoch keine Garantie über die richtige Funktion der übrigen verwendeten Befehle gegeben werden.

Um eine Fehlererkennung am Überwachungsmodul zu testen, wird in $1s$ Abständen eine Falschantwort vom Softwaremodul gegeben, worauf die geforderte Wiederholung der Frage durch das Überwachungsmodul vom Softwaremodul überprüft wird.

Abbildung 10.3 zeigt den Ablauf der Tasks zur Fehlererkennung in Ebene 3 in mehreren Szenarien. Während die erste Überprüfung noch richtig verläuft (a) und die Antwortgenerierung nach wenigen Millisekunden abgeschlossen ist, führt ein Fehler auf CPU0 (b) zu einer Laufzeitverzögerung. Dieser Fehler hat zur Folge, dass die Überwachungssoftware nach $40ms$ ein Timeout generiert und eine entsprechende Antwort an das Überwachungsmodul zurücksendet (c). Dieser Fall erhöht den Fehlerzähler am ÜM und führt zu einer Wiederholung der Frage. Durch den anhaltenden Fehler auf Kern 0 wird die Antwortgenerierung wiederum verzögert, jedoch kann die Abarbeitung vor der Generation eines Timeouts abgeschlossen werden und die übrigen Kerne bearbeiten das Datenpaket bevor es rechtzeitig zurückgesendet wird (d). Das Überwachungsmodul löst keine Fehlerreaktion aus. Der letzte Zyklus wird wieder normal abgearbeitet (e).

Reaktionszeit

Das Überwachungsmodul sendet alle $50ms$ eine Anforderung an die Überwachungssoftware. Diese hat zur Bearbeitung bis zum nächsten Zyklus (ca. $50ms$) Zeit zur Generierung

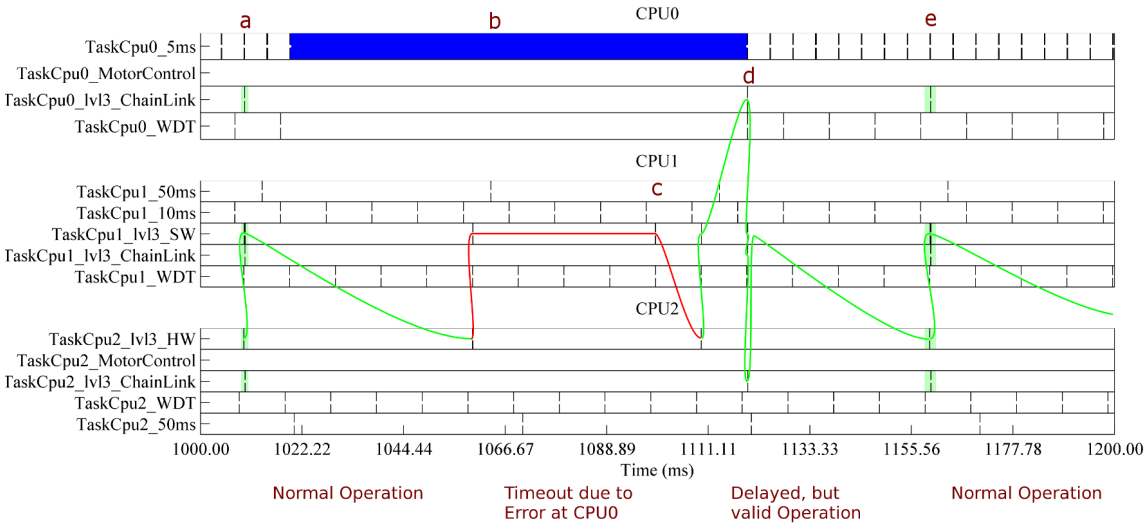


Abbildung 10.3: Diagramm einer Fehlersimulation in Ebene 3

der Antwort. Tritt ein Fehler in der Übertragung, bzw. bei der Berechnung der Antwort auf, wird die Frage zur Erhöhung der Verfügbarkeit des Systems wiederholt. Kommt es zu zweimaligem Auftreten einer falschen Antwort, leitet das Überwachungsmodul mit einer Reaktionszeit von $100ms$ den sicheren Zustand des Systems ein. Dadurch wird die Forderung zur Einhaltung der Reaktionszeit von $200ms$ (FSK-04.03 in Kapitel 6.2) erfüllt.

10.2 Hardwareevaluierung

Eine Evaluierung der Hardware wird im Folgenden beschrieben. Da diese jedoch nicht im Fokus der Arbeit lag, werden lediglich die wichtigsten Komponenten beleuchtet.

10.2.1 Evaluierung von Hardware Architekturmetriken

Zur Bewertung der Betriebssicherheit auf Hardwareebene müssen die verwendeten Bauteile und Schaltungen auf deren Wirksamkeit überprüft werden und Überlegungen angestellt werden, inwiefern sich die einzelnen Hardwarefehler (Leitungsunterbrechungen, Kurzschlüsse, Versagen von (Schutz-)Bauteilen) auf die Funktion des Gesamtsystems auswirken.

Der verwendete Mikrocontroller hat, wie in Punkt 8.2.1.4 beschrieben, eine Vielzahl an hardwareseitiger Fehlererkennung integriert. Jedoch ist zur Reaktion auf derartige Fehler die Konfiguration der Fehlererkennung nötig, welche via Software durchgeführt werden muss. Da jedoch eine Schnittstelle zur Durchführung dieser Konfiguration vom Chiphersteller zum Zeitpunkt dieser Arbeit nicht erhältlich war, kann eine Aktivierung und Evaluierung der hardwareseitigen ECU Fehlererkennung nicht durchgeführt werden.

AD Wandler (ADC)

Das Stoppen der Funktion des AD-Wandlers stellt ein großes Gefahrenpotential dar, weil die Regelung der Drosselklappenposition direkt davon abhängt. Um derartige Fehler zu erkennen, wird ein Testsignal je Wandler verwendet, um die richtige Wandlung zu überprüfen. Da das Testsignal bereits vor dem Multiplexer angelegt wird, testet man nicht nur den Wandler selbst, sondern auch den Multiplexer. Durch die Überprüfung offen ist die Erkennung von einzelnen Registerfehlern (z.B. Einfrieren eines Sensorsignals), jedoch wird durch die Redundanz beider Sensoren eine Erkennung eines Signalausfalls ermöglicht.

Internes Sicherheitsmodul (SMU)

Die vorher angesprochenen ECU-Fehler werden im Chip zu einem Sicherheitsmodul geleitet, welches je nach Konfiguration die Betriebszustände sicherheitsrelevanter Module steuern kann und so ohne Einfluss durch die Software auf Fehler reagieren kann. Dieses beinhaltet auch einen Watchdog, welcher per Software getriggert werden muss, um eine Fehlerreaktion des Moduls zu verhindern. Somit werden Laufzeitfehler bereits im selben Controller per Hardware erkannt und dieser leitet selbstständig ein Abschalten des Steuergerätes ein.

Leistungstreiber

Der verwendete Leistungstreiber hat Schutzmechanismen eingebaut, um sich selbst und den Motor zu schützen. Dazu gehören eine Kurzschlusserkennung, ein Übertemperaturschutz sowie eine Unterspannungserkennung. Diese Mechanismen garantieren jedoch lediglich den Schutz der Bauteile vor Schäden. Um die Sicherheit während des Betriebs sicherstellen zu können, muss der Treiber über diverse Kanäle abschaltbar sein. Da dies mit dem verwendeten Bauteil nicht garantiert werden konnte, wurde der Sicherheitsschalter in den Leistungspfad geschaltet, welcher im Fall eines Defekts Betriebssicherheit durch Abschaltung der Versorgung wiederherstellen kann.

Sicherheitsschalter

Da während des Betriebes ein Abschalten zur Überprüfung der Sicherheitskette nicht erlaubt ist, wird die Funktionsüberprüfung der Sicherheitsschalter beim Starten des Systems durchgeführt. Wird ein Abschaltpfad im Betrieb defekt, wird durch Redundanz sichergestellt, dass die Abschaltung dennoch zu einer hohen Wahrscheinlichkeit ermöglicht kann. Eine Untersuchung von zufälligen Hardwarefehlern der Bauteile im Abschaltpfad wurde nicht durchgeführt.

10.2.2 Evaluierung der Einhaltung von Sicherheitszielen in Bezug auf zufällige HW Fehler

Um die Einhaltung der Sicherheitsziele in Bezug auf zufällige Hardwarefehler zu beweisen, müssen für jedes Bauteil sog. FIT Raten bestimmt werden, welche als Maß zur Ausfallswahrscheinlichkeit verwendet werden. Da die Untersuchung diesbezüglich durch Nichtverfügbarkeit von Werten nicht ohne weiteres möglich war, wird nicht näher darauf eingegangen.

10.3 Softwareevaluierung

In der Softwareevaluierung werden ähnlich der Hardwareevaluierung alle Module auf mögliche Fehlerquellen untersucht, sowie eine Analyse der Fehlerfortpflanzung durchgeführt. Insbesondere bei der Software liegt das Hauptaugenmerk der Evaluierung auf der Rückwirkungsfreiheit zwischen den einzelnen Modulen, da keine physikalische Trennung bei Software möglich ist und geteilte Ressourcen unumgänglich sind. Eine etablierte Methode zur Verifizierung von Software-Schutzmechanismen ist das gezielte Einbringen von Fehlern (Fault Injection). [65]

Eine erste Evaluierung wurde bei der Entwicklung der einzelnen Softwaremodule durch Simulationen durchgeführt. [Siehe Kapitel 9.5] In diesem Abschnitt wird auf Schutzmechanismen durch das Betriebssystem eingegangen, welche sicherheitskritische Fehler erkennen, und entsprechende Reaktionen einleiten muss, um die Einhaltung der Sicherheitsanforderungen zu gewährleisten.

10.3.1 Timing Protection

Laufzeitüberschreitende Tasks sollten bereits auf Betriebssystemebene erkannt und terminiert werden, bevor diese eine sicherheitskritische Laufzeitverzögerung des Gesamtsystems auf der CPU verursachen. Dazu werden bereits zur Entwicklungsphase maximale Laufzeiten für die Ausführung, die Interrupt-Deaktivierung, die Ressourcenbindung sowie die maximale Wiederholrate des Tasks angegeben. Und die Einhaltung der Zeiten mit internen Zählern durch den Scheduler überprüft.

Quelltext 10.1 zeigt beispielhaft die Konfiguration der Timingprotection in der OIL-Konfigurationsdatei.

Zwar wurde im verwendeten Betriebssystem vom Hersteller eine Timing Protection vorgesehen, im Zuge dieser Arbeit konnte diese jedoch nicht verwendet werden, da der Compiler-abhängige Code noch nicht für die Verwendung des Tasking Compilers 4.3R1 erweitert wurde.

Quelltext 10.1: Konfiguration der Timingprotection

```
TASK TaskCpu0_5ms {
  CPU_ID      = "cpu0";
  PRIORITY    = 1;
  AUTOSTART   = FALSE;
  STACK       = SHARED;
  ACTIVATION  = 1;
  SCHEDULE    = NON;
  TIMING_PROTECTION = TRUE {
    EXECUTIONBUDGET         = 0.0025;
    MAXALLINTERRUPTLOCKTIME = 0.0001;
    RESOURCE = RESOURCELOCK {
      RESOURCELOCKTIME = 0.0002;
      RESOURCE = RES_SCHEDULER;
};};};
```

Quelltext 10.2: Konfiguration einer Applikation sowie der Memoryprotection

```
  MEMORY_PROTECTION = TRUE;
  STACKMONITORING = TRUE;

APPLICATION UserApp1 {
  TRUSTED = FALSE;
  ISR  = App1Isr1;
  ISR  = App1Isr2;
  TASK = TaskApp1Prio2;
  TASK = TaskApp1Prio4;
  TASK = TaskApp1Prio6;
  MEMORY_SIZE = 0x1000;
  SHARED_STACK_SIZE = 256;
  IRQ_STACK_SIZE = 256;
};
```

10.3.2 Memory Protection

ERIKA Enterprise unterstützt die Verwendung der MPU zum Schutz der Integrität des Speichers vor falschen Zugriffen. Um diese zu verwenden, müssen jedoch einzelne Tasks und Interrupt Service Routinen in AUTOSAR konforme Applications zusammengefügt werden, welche beim Kompilierprozess in einen gemeinsamen Speicherbereich abgelegt werden. Der MPU werden vom Scheduler die erlaubten Speicherbereiche zu jedem Task- und ISR-Wechsel mitgeteilt und von dieser auf Einhaltung der Speichergrenzen überwacht. Wird der Zugriff auf eine unerlaubte Speicherstelle versucht, verhindert das die MPU und löst einen Interrupt aus, worauf das Betriebssystem entsprechend reagieren kann. Quelltext 10.2 zeigt beispielhaft die Konfiguration der Memoryprotection in der OIL-Konfigurationsdatei.

Die Verwendung der Memoryprotection ist jedoch im Zuge dieser Arbeit nicht möglich gewesen, da der Compiler-abhängige Code noch nicht für die Verwendung des Tasking Compilers 4.3R1 erweitert wurde.

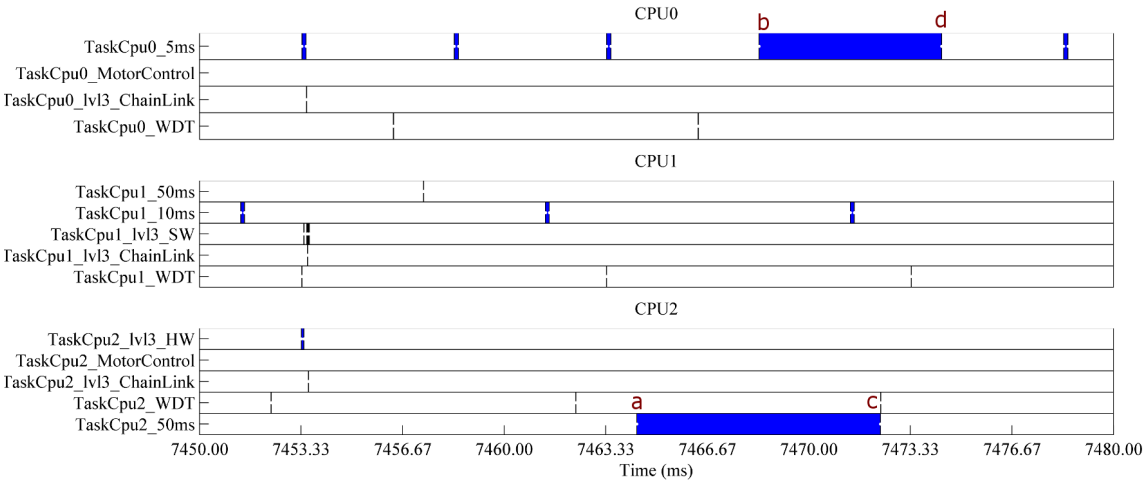
10.3.3 Geteilte Ressourcen

Problematisch hinsichtlich der Verifizierung von Echtzeitsystemen auf Mehrkernplattformen sind gemeinsam genutzte Ressourcen, da eine Blockade durch eine Instanz direkte Auswirkungen auf die anderen Systeme haben kann. Abbildung 10.4a zeigt die Auswirkung auf das Mehrkernsystem bei Blockade einer gemeinsam genutzten Ressource durch den Task einer Recheneinheit. Dabei belegt der Displaytask (TaskCpu2_50ms) das zusammen mit dem Funktionstask (TaskCpu0_5ms) genutzte Sensordatenarray und verzögert die Freigabe durch einen injizierten Fehler (a). Wird nun der Funktionstask auf CPU0 aufgerufen (b) und versucht, die Werte dieser gemeinsamen Ressource zu aktualisieren, verbleibt dieser solange im Busy-Wait Zustand, bis die Ressource vom Displaytask freigegeben wird (c). Danach erst werden die Daten vom Funktionstask aktualisiert und der Task kann beendet werden (d). Da dadurch auf CPU0 der höchstpriorisierte Task blockiert wird, ist das gesamte System durch einen Fehler auf einem anderen Kern betroffen. (Keine „Freedom of Interference“ gegeben!)

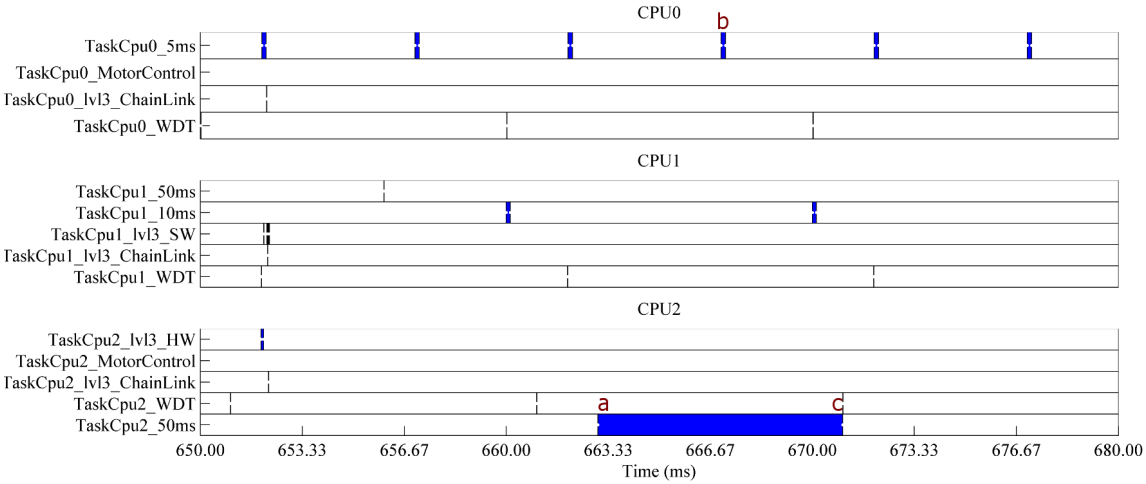
Eine mögliche Lösung bietet hier die Timingprotection, welche die Laufzeit eines Tasks auf ein gewisses Maximum begrenzt, und so die Ausführung niedrig priorisierter Tasks Verzögerung durch den hochpriorisierten Task ermöglicht. Eine effizientere Methode ist jedoch die Verhinderung von Busy-Wait Zustände selbst. So wird in der verwendeten Implementierung ein Verfahren benutzt, welches lediglich versucht, eine Ressource zu belegen. Ist diese nicht verfügbar, kann der Versuch wiederholt werden, oder mit entsprechenden Ersatzwerten fortgesetzt werden. Abbildung 10.4b zeigt wiederum das Ablaufdiagramm bei injiziertem Fehler. Erneut wird der Displaytask (TaskCpu2_50ms) aufgerufen und ein Fehler während der Ressourcenbelegung injiziert (a). Will nun der Funktionstask (TaskCpu0_5ms) auf die gemeinsame Ressource zugreifen (b), testet dieser vorab die Belegung der gemeinsamen Ressource. Trotz der Blockade durch den niedrig priorisierten Task auf CPU2 führt das jetzt nicht mehr zur Laufzeitverzögerung auf CPU0. Der Funktionstask wird ohne Aktualisierung der Werte fortgesetzt und abgeschlossen (b). Erst nachdem die Ressource vom Displaytask erfolgreich freigegeben wurde (c), kann auch der Funktionstask wieder auf das gemeinsame Sensordatenarray zugreifen und die Werte aktualisieren.

Ist es jedoch für die Funktion von sicherheitskritischer Bedeutung, dass gemeinsame Ressource verfügbar sein müssen, ist eine Evaluierung der zeitlicher Konsequenzen nötig. Mögliche Methoden zur Quantifizierung von Auswirkungen geteilter Prozessorressourcen finden sich in [68].

In der Evaluierung wurde lediglich auf die, aus Sicht dieser Arbeit, wichtigsten Komponenten eingegangen, da eine vollständige Evaluierung nach den Vorgaben der ISO 26262 [Siehe Anhang A.4] den Rahmen dieser Arbeit gesprengt hätte.



(a) Ressourcenblockade mit Auswirkung auf das System



(b) Ressourcenblockade ohne Auswirkung auf das System

Abbildung 10.4: Gezielte Ressourcenblockade und deren Auswirkungen

Kapitel 11

Zusammenfassung und Ausblick

Im letzten Kapitel werden die Ergebnisse der Arbeit zusammengefasst und ein Ausblick auf weitere Arbeiten mit den verbundenen Möglichkeiten und Herausforderungen gegeben.

11.1 Zusammenfassung

Durch den steigenden Bedarf an Rechenleistung in eingebetteten Systemen findet auch die Verwendung von Mehrkern-Architekturen in diesem Bereich immer größeren Zuspruch. Dabei stellen sich jedoch neue Herausforderungen aus Sicht der funktionalen Sicherheit, welche im Zuge dieser Arbeit angesprochen wurden. Themen wie Ressourcenabhängigkeiten, Task Synchronisierung, Einhaltung von Ausführungszeiten sowie die Rückwirkungsfreiheit zwischen den einzelnen Rechnerinstanzen im Controller wurden angesprochen.

Im Zuge dieser Arbeit wurde in Zusammenarbeit mit AVL List GmbH eine Entwicklungsumgebung zur Verwendung mit der gewählten Mehrkernplattform aufgebaut und die einzelnen Komponenten aufeinander abgestimmt, um eine durchgängige Entwicklung vom Systemmodell über die BSW- und Laufzeit-Konfiguration hin zum kongruenten Debuggen zu ermöglichen.

Weiters wurde das E-Gas Sicherheitskonzept vom ursprünglichen Einkernsystem um die Verwendung mit Mehrkernsystemen erweitert und eine Demonstrationsplattform in Hardware und Software implementiert, welche die Funktion sowohl des adaptierten Mehrkernbetriebssystems als auch der einzelnen Ebenen (Funktion, Funktionsüberwachung und Rechnerüberwachung) des erweiterten Sicherheitskonzeptes zeigen soll.

Um eine frühzeitige Evaluierung während der Entwicklung der einzelnen Ebenen bereits auf Modell-Ebene durchführen zu können, wurde die Verwendung von Model-in-the-Loop Verfahren gezeigt und Simulationsergebnisse beschrieben.

Eine der größten Herausforderungen beim Umstieg auf Mehrkernsysteme ist der parallele Ablauf der jeweiligen Applikationen. Um die richtige Funktion des Systems in Hinsicht auf den zeitlichen Ablauf zu zeigen, wurden die gegebenen Debugging-Möglichkeiten um ein Werkzeug erweitert, welches den zeitlichen Verlauf der einzelnen Tasks aller Kerne mit gleicher Zeitbasis aufnimmt und eine grafische Darstellung unter Berücksichtigung der Prioritäten sowie Taskunterbrechungen ermöglicht.

Zusammen mit den Simulationsergebnissen konnte damit die Funktion der drei Ebenen des Sicherheitskonzeptes gezeigt und evaluiert werden.

11.2 Ausblick

Das implementierte Sicherheitskonzept stellt eine fundierte Grundlage für sicherheitskritische Anwendungen dar, welche sich von der Anwendung im „Throttle-by-wire“ System auf eine Vielzahl weiterer sicherheitskritischer Systeme adaptieren lässt.

In Bezug auf die Fehlerreaktion ist jedoch anzumerken, dass das verwendete Sicherheitskonzept davon ausgeht, dass ein sicherer Zustand durch eine meist verminderte Ansteuerung bzw. Abschaltung der Aktoren herstellbar ist. Dies ist jedoch nicht immer der Fall und die Forderung nach fehlertoleranten Systemen, welche auch nach Erkennung des Fehlers noch eine zuverlässige vollständige Funktionalität gewährleisten, kann durch dieses Konzept nicht gänzlich erfüllt werden. Aus dem Bereich der Avionik sind Sicherheitskonzepte bekannt, welche durch mehrfach redundante Systeme und Verwendung eines Voters volle Funktionalität auch bei Ausfall eines oder mehrerer Teilsysteme gewährleisten können. So ist eine Verwendung von Mehrkernsystemen mit redundanten Anwendungen und einem Kontrollorgan (Stichwort: 2-von-3) ein möglicher Ansatz zur Verbesserung der Zuverlässigkeit und Sicherheit.

Nichtsdestotrotz bietet der Demonstrator durch die Verwendung eines offenen Betriebssystems sowie durch die Implementierung des Sicherheitskonzeptes, eine ideale Grundlage für künftige Prototypenentwicklung und -evaluierung und kann zu weiteren Untersuchungen an Mehrkernsystemen sowie Sicherheitskonzepten verwendet werden.

Das verwendete System eignet sich zwar für die Entwicklung von Prototypen sehr gut, jedoch sind hinsichtlich des Betriebssystems noch weitere Evaluierungen und Verbesserungen nötig, um den Ansprüchen eines Echtzeitbetriebssystems für sicherheitskritische Anwendungen zu genügen. Beispielhaft angeführt sind Erweiterungen der Timing- und Memory-Protection, welche für den verwendeten Compiler noch nicht zur Gänze umgesetzt wurden.

Auch die implementierten Sicherheitsebenen stellen in der jetzigen Form lediglich eine Grundfunktionalität bereit, welche zur Erhöhung von Zuverlässigkeit und Genauigkeit sowie in Richtung Fehlererkennung und Fehleraufzeichnung noch weiter verfeinert werden können.

Anhang A

Anhang

A.1 Beispiel einer OIL Konfigurationsdatei

Quelltext A.1: Beispiel einer OIL Konfigurationsdatei

```
CPU demoApplication{
    OS demoOS {
        STATUS = EXTENDED;
        ErrorHook = TRUE;
        StartupHook = FALSE;
        . . .
        };
    TASK demoTask {
        TYPE = EXTENDED;
        PRIORITY = 1;
        SCHEDULE = FULL;
        ACTIVATION = 1;
        AUTOSTART = TRUE;
        EVENT = demoEvent;
        RESOURCE= demoResource;
        };
    TASK demoTask2 {
        TYPE = EXTENDED;
        PRIORITY = 2;
        SCHEDULE = NONE;
        ACTIVATION = 1;
        AUTOSTART = FALSE;
        EVENT = NONE;
        RESOURCE = demoResource;
        };
    RESOURCE demoResource;
    EVENT demoEvent{
        MASK = AUTO;
        };
    COUNTER demoCounter {
        MAXALLOWEDVALUE = 65535;
        TICKSPERBASE = 1;
        MINCYCLE = 1;
        };
    ALARM demoAlarm
        COUNTER = demoCounter;
        ACTION = ACTIVATETASK{
            TASK = demoTask1;
            };
        };
    ISR demoISR {
        CATEGORY = 2;
        };
};
```

A.2 E-Gas Fehlerreaktionen

A.2.1 Ebene 1

Pedalwertgeber

Tabelle A.1: Fehlerabdeckung beim Pedalwertgeber aus [3]

N°	Fehlerbeschreibung	Fehlerreaktion
FB-01.01	Sollwert 1 > Schwelle	Ersatzbetrieb Sollwert 2, mit Begrenzung max. Wert und max. Gradient, Bremse betätigt / Bremssignalfehler = Leerlaufvorgabe
FB-01.02	Sollwert 1 < Schwelle	Ersatzbetrieb Sollwert 2, mit Begrenzung max. Wert und max. Gradient, Bremse betätigt / Bremssignalfehler = Leerlaufvorgabe
FB-01.03	Sollwert 2 > Schwelle	Ersatzbetrieb Sollwert 1, mit Begrenzung max. Wert und max. Gradient, Bremse betätigt / Bremssignalfehler = Leerlaufvorgabe
FB-01.04	Sollwert 2 < Schwelle	Ersatzbetrieb Sollwert 1, mit Begrenzung max. Wert und max. Gradient, Bremse betätigt / Bremssignalfehler = Leerlaufvorgabe
FB-01.05	|Sollwert 1 - Sollwert 2| > Schwelle	Ersatzbetrieb mit Minimum aus Sollwert 1 und Sollwert 2 mit Begrenzung max. Wert und Max. Gradient, Bremse betätigt / Bremssignalfehler = Leerlaufvorgabe
FB-01.06	Versorgungsspannung außerhalb des zulässigen Bereichs	Leerlaufvorgabe
FB-01.07	Ersatzbetrieb mit Sollwert 1 und Sollwert 1 > Schwelle	Leerlaufvorgabe
FB-01.08	Ersatzbetrieb mit Sollwert 2 und Sollwert 2 > Schwelle	Leerlaufvorgabe
FB-01.09	Ersatzbetrieb mit Sollwert 1 und Sollwert 1 < Schwelle	Leerlaufvorgabe
FB-01.10	Ersatzbetrieb mit Sollwert 2 und Sollwert 2 < Schwelle	Leerlaufvorgabe

Elektromechanisches Stellsystem

Tabelle A.2: Fehlerabdeckung beim elektromechanischen Stellsystem aus [3]

N°	Fehlerbeschreibung	Fehlerreaktion
FB-02.01	DKS 1 > Schwelle	Ersatzbetrieb mit DKS 2 und Vergleich mit DEW und Begrenzung des Maximalwertes von DKS 2 als Funktion der Motordrehzahl
FB-02.02	DKS 1 < Schwelle	Ersatzbetrieb mit DKS 2 und Vergleich mit Ersatzwert und Begrenzung des Maximalwertes von DKS 2 als Funktion der Motordrehzahl
FB-02.03	DKS 2 > Schwelle	Ersatzbetrieb mit DKS 1 und Vergleich mit Ersatzwert und Begrenzung des Maximalwertes von DKS 1 als Funktion der Motordrehzahl
FB-02.04	DKS 2 < Schwelle	Ersatzbetrieb mit DKS 1 und Vergleich mit Ersatzwert und Begrenzung des Maximalwertes von DKS 1 als Funktion der Motordrehzahl
FB-02.05	DKS 1 und 2 > Schwelle und DKS 2 plausibel zu DEW	Ersatzbetrieb mit DKS 2 und Vergleich mit DEW, dabei Begrenzung des Maximalwertes DKS 2 als Funktion der Motordrehzahl
FB-02.06	DKS 1 und 2 > Schwelle und DKS 1 und 2 unplausibel zu DEW	Irreversible Einspritzmengenbegrenzung der Ebene 1, Drosselklappe stromlos
FB-02.07	DKS 1 und 2 > Schwelle und DKS 1 plausibel zu DEW	Ersatzbetrieb mit DKS 1 und Vergleich mit DEW, dabei Begrenzung des Maximalwertes DKS 1 als Funktion der Motordrehzahl
FB-02.08	Ersatzbetrieb mit DKS 1, Plausibilisierung mit DEW und zusätzlich DKS 1 < bzw. > Schwelle	Irreversible Einspritzmengenbegrenzung der Ebene 1, Drosselklappe stromlos
FB-02.09	Ersatzbetrieb mit DKS 2, Plausibilisierung mit DEW und zusätzlich DKS 2 < bzw. > Schwelle	Irreversible Einspritzmengenbegrenzung der Ebene 1, Drosselklappe stromlos
FB-02.10	Ersatzbetrieb mit DKS 2 und Plausibilisierung mit DEW und Lastsensor-Fehler	Irreversible Einspritzmengenbegrenzung der Ebene 1, Drosselklappe stromlos
FB-02.11	Lagereglerfehler: z.B. Fehlerhafte Stellgrößenvorgabe, mechanisch klemmende DK	Irreversible Einspritzmengenbegrenzung der Ebene 1, Drosselklappe stromlos
FB-02.12	Endstufenfehler	Irreversible Einspritzmengenbegrenzung der Ebene 1, Drosselklappe stromlos

Überwachung externer Eingriffe

Tabelle A.3: Fehlerabdeckung bei externen Eingriffen aus [3]

N°	Fehlerbeschreibung	Fehlerreaktion
FB-03.01	Fehlerhafte Botschaft für externe Momentenanforderung (Erkennung in Ebene 1)	Sperrung der Anforderung kundenspezifisch reversibel oder irreversibel, Momenten-Übergangs-Funktion kundenspezifisch

Überwachung Programmierung und Versorgungsspannung

Tabelle A.5: Fehlerabdeckung bei Programmierung und Versorgungsspannung aus [3]

N°	Fehlerbeschreibung	Fehlerreaktion
FB-04.01	Flash: Programmierung nicht beendet	Verbleib in Bootlock
FB-04.02	Flash: Programmieirungsfehler	Verbleib in Bootlock
FB-04.03	Versorgungsspannung außerhalb Spezifikation	Reset

Bremsinformationen

Tabelle A.7: Fehlerabdeckung bei Bremsinformationen aus [3]

N°	Fehlerbeschreibung	Fehlerreaktion
FB-05.01	Unplausibilität der redundanten Bremssignale	Fahrgeschwindigkeitsregelung abschalten

A.2.2 Ebene 2

Tabelle A.9: Fehlerabdeckung in Ebene 2 aus [3]

N°	Fehlerbeschreibung	Fehlerreaktion
FB-06.01	Fehlerhafte Botschaft für externe momentenerhöhende Anforderungen	Sperrung der Anforderung kundenspezifisch reversibel oder irreversibel, Momenten-Übergangs-Funktion kundenspezifisch
FB-06.02	Motordrehzahl fehlerhaft: Abweichung zwischen Ebene 1 und 2	Reset
FB-06.03	Fahrerwunscherkennung fehlerhaft: Abweichung zwischen Ebene 1 und 2	Nach einer applizierbaren Anzahl von Resets und nicht erfolgter Fehlerheilung irreversible EMB aus Ebene 2 anfordern/überwachen, DK stromlos
FB-06.04	Fehler in der Abschaltung bzw. unzulässige Aktivierung der FGR	FGR Eingriff deaktiviieren, falls Sperren nicht möglich: Ansprechen Momentenvergleich, irreversible EMB aus Ebene 2 anfordern/überwachen, DK stromlos
FB-06.05	Kraftstoffmasse / Lambda / Lastsignal fehlerhaft	Plausibilisierung in Ebene 2 nur im Schichtbetrieb, irreversibles Sperren Schichtbetrieb, Übergang in Homogenbetrieb
FB-06.06	Ansteuerdauer Einspritzung fehlerhaft	Umschaltung in Homogenbetrieb
FB-06.07	Fehlerhafter Zündwinkel	Nach einer applizierbaren Anzahl von Resets und nicht erfolgter Fehlerheilung irreversible EMB aus Ebene 2 anfordern/überwachen, DK stromlos
FB-06.08	Fehler beim Plausibilisieren des Lastsignals mit dem DKS	Nach einer applizierbaren Anzahl von Resets und nicht erfolgter Fehlerheilung irreversible EMB aus Ebene 2 anfordern/überwachen, DK stromlos
FB-06.09	Unzulässige Motormomentüberschreitung durch Fehler in Ebene 1	Nach einer applizierbaren Anzahl von Resets und nicht erfolgter Fehlerheilung irreversible EMB aus Ebene 2 anfordern/überwachen, DK stromlos
FB-06.10	EMB wird nicht umgesetzt in Ebene 1	Abschaltung der leistungsbestimmenden Endstufen
FB-06.11	A/D-Wandlerfehler	Nach einer applizierbaren Anzahl von Resets und nicht erfolgter Fehlerheilung irreversible EMB aus Ebene 2 anfordern/überwachen, DK stromlos
FB-06.12	Fehler Verlustmoment aus Ebene 1	Nach einer applizierbaren Anzahl von Resets und nicht erfolgter Fehlerheilung irreversible EMB aus Ebene 2 anfordern/überwachen, DK stromlos

Tabelle A.9: Fortsetzung

N°	Fehlerbeschreibung	Fehlerreaktion
FB-06.13	Fehler in der übernahmen von Adaptionswerten/Korrekturfaktoren aus Ebene 1 in Ebene 2 (Toleranzeinengung), Fehler im Pfad Istmomentrückrechnung	Nach einer applizierbaren Anzahl von Resets und nicht erfolgter Fehlerheilung irreversible EMB aus Ebene 2 anfordern/überwachen, DK stromlos
FB-06.14	Fehler in der übernahmen von Adaptionswerten/Korrekturfaktoren aus Ebene 1 in Ebene 2 (Toleranzeinengung), Fehler im Pfad Berechnung zulässiges Moment	Nach einer applizierbaren Anzahl von Resets und nicht erfolgter Fehlerheilung irreversible EMB aus Ebene 2 anfordern/überwachen, DK stromlos

A.2.3 Ebene 3

Tabelle A.10: Fehlerabdeckung in Ebene 3 aus [3]

N°	Fehlerbeschreibung	Fehlerreaktion
FB-07.01	Falsche Zeit-/Fehlerzählerrückmeldung in der Frage/Antwortroutine (Erkennung durch FR)	Reset
FB-07.02	Falsche Zeit-/Fehlerzählerrückmeldung in der Frage/Antwortroutine (Erkennung durch ÜM)	Reset
FB-07.03	Fehler im Abschaltpfadtest	Reset bis Motorbetrieb zulässig
FB-07.04	Fehler in nichtflüchtigen Speichern	Reset
FB-07.05	Fehler in flüchtigen Speichern	Reset
FB-07.06	Fehler in Modul-Überwachung	Reset

A.3 PWM Treiber Schaltplan und Aufbau

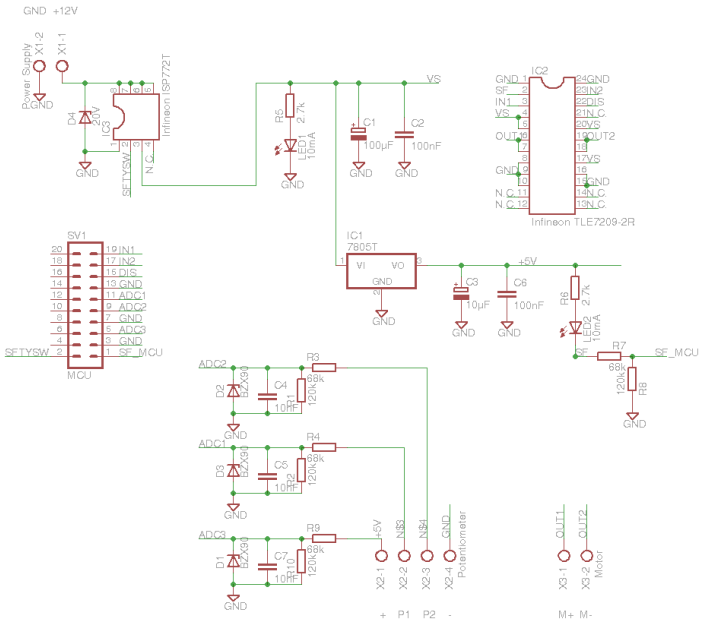


Abbildung A.1: Schaltplan des PWM Treibers mit Sicherheitsschalter

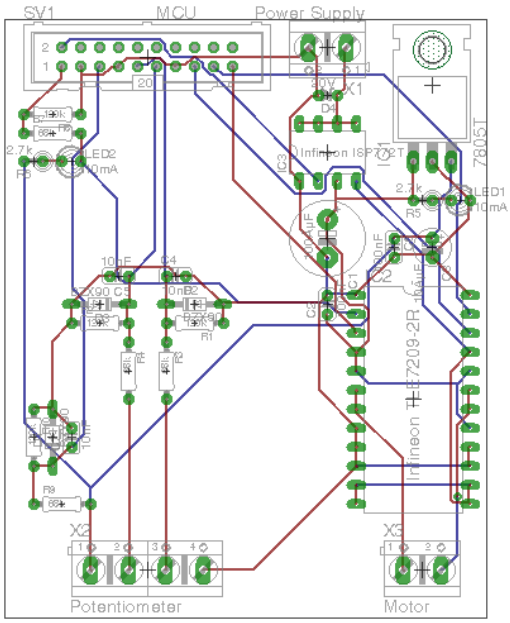


Abbildung A.2: Aufbauplan des PWM Treibers mit Sicherheitsschalter

A.4 Geforderte Arbeitspakete nach ISO 26262

Die folgende Auflistung zeigt die Arbeitspakete des Produktlebenszyklus aus der ISO 26262, aufgeteilt nach den jeweiligen Teilen aus [38] Rechts neben den einzelnen Arbeitsschritten markiert ein Symbol, dass dieser Schritt im Zuge der Arbeit berücksichtigt worden ist.

2 ISO26262-2 Management of functional safety
- 2.1 Overall safety management
  - 2.1.1 Organization-specific rules and processes for functional safety
  - 2.1.2 Evidence of competence
  - 2.1.3 Evidence of quality management
- 2.2 Safety management during the concept phase and the product development
  - 2.2.1 Safety plan
  - 2.2.2 Project plan (refined)
  - 2.2.3 Safety case
  - 2.2.4 Functional safety assessment plan
  - 2.2.5 Confirmation of measure reports
- 2.3 Safety management after the item's release for production
  - 2.3.1 Evidence of field monitoring

3 ISO26262-3 Concept Phase
- 3.1 Item definition ✓
  - 3.1.1 Item definition ✓
- 3.2 Initiation of the safety life-cycle
  - 3.2.1 Impact analysis
  - 3.2.2 Safety plan
- 3.3 Hazard analysis and risk assessment ✓
  - 3.3.1 Hazard analysis and risk assessment ✓
  - 3.3.2 Safety goals ✓
  - 3.3.3 Verification review report of the hazard analysis and risk assessment and the safety goals
- 3.4 Functional safety concept ✓
  - 3.4.1 Functional safety concept ✓
  - 3.4.2 Verification report of the functional safety concept

4 ISO26262-4 Product development at the system level
- 4.1 Initiation of product development at the system level
  - 4.1.1 Project Plan (refined)
  - 4.1.2 Safety Plan (refined)
  - 4.1.3 Item integration and testing plan(s)
  - 4.1.4 Validation plan
  - 4.1.5 Functional safety assessment plan (refined)
- 4.2 Specification of the technical safety requirements ✓
  - 4.2.1 Technical safety requirements specification ✓

- 4.2.2 System verification report
- 4.2.3 Validation plan (refined)

4.3 System design ✓
- 4.3.1 Technical safety concept ✓
- 4.3.2 System design specification ✓
- 4.3.3 Hardware-software interface specification (HSI) ✓
- 4.3.4 Specification of requirements for production, operation, service and decommissioning
- 4.3.5 System verification report (refined)
- 4.3.6 Safety analysis reports

4.4 Item integration and testing
- 4.4.1 Item integration and testing plan (refined)
- 4.4.2 Integration testing specification(s) ✓
- 4.4.3 Integration testing report(s)

4.5 Safety validation
- 4.5.1 Validation plan (refined)
- 4.5.2 Validation report

4.6 Functional safety assessment
- 4.6.1 Functional safety assessment report

4.7 Product Release
- 4.7.1 Release for production report

5 ISO26262-5 Product development at the hardware level

5.1 Initiation of product development at the hardware level
- 5.1.1 Safety Plan (refined)

5.2 Specification of hardware safety requirements ✓
- 5.2.1 Hardware safety requirements specification (including test and qualification criteria) ✓
- 5.2.2 Hardware-software specification (refined) ✓
- 5.2.3 Hardware safety requirements verification report

5.3 Hardware design ✓
- 5.3.1 Hardware design specification ✓
- 5.3.2 Hardware safety analysis report
- 5.3.3 Specification of requirements related to production, operation, service and decommissioning

5.4 Evaluation of the hardware architectural metrics ✓
- 5.4.1 Analysis of the effectiveness of the architecture of the item to cope with random hardware failures ✓
- 5.4.2 Review report of evaluation of the effectiveness of the architecture of the item to cope with the random hardware failures

5.5 Evaluation of safety goal violations due to random HW failures
- 5.5.1 Analysis of safety goal violations due to random hardware failures
- 5.5.2 Specification of dedicated measures for hardware, if needed, including the rationale regarding the effectiveness of the dedicated measures
- 5.5.3 Review report of evaluation of safety goal violations due to random hardware failures

	5.6	Hardware integration and testing	✓
		5.6.1 Hardware integration and testing report	
6	ISO26262-6 Product development at the software level		
	6.1	Initiation of product development at the software level	
		6.1.1 Safety plan (refined)	
		6.1.2 Software verification plan	
		6.1.3 Design and coding guidelines for modelling and programming languages	✓
		6.1.4 Tool application guidelines	✓
	6.2	Specification of software safety requirements	✓
		6.2.1 Software safety requirements specification	✓
		6.2.2 Hardware-software interface specification (refined)	✓
		6.2.3 Software verification plan (refined)	
		6.2.4 Software verification report	
	6.3	Software architectural design	✓
		6.3.1 Software architectural design specification	✓
		6.3.2 Safety plan (refined)	
		6.3.3 Software safety requirements specification (refined)	✓
		6.3.4 Safety analysis report	
		6.3.5 Dependent failures analysis report	
		6.3.6 Software verification report (refined)	
	6.4	Software unit design and implementation	✓
		6.4.1 Software unit design specification	✓
		6.4.2 Software unit implementation	✓
		6.4.3 Software verification report (refined)	
	6.5	Software unit testing	✓
		6.5.1 Software verification plan (refined)	✓
		6.5.2 Software verification specification	✓
		6.5.3 Software verification report (refined)	
	6.6	Software integration and testing	✓
		6.6.1 Software verification plan (refined)	
		6.6.2 Software verification specification (refined)	
		6.6.3 Embedded Software	✓
		6.6.4 Software verification report (refined)	
	6.7	Verification of software safety requirements	
		6.7.1 Software verification plan (refined)	
		6.7.2 Software verification specification (refined)	
		6.7.3 Software verification report (refined)	
	6.8	Software Configuration	✓
		6.8.1 Configuration data specification	✓
		6.8.2 Calibration data specification	✓
		6.8.3 Safety plan (refined)	
		6.8.4 Configuration data	✓

- 6.8.5 Calibration data ✓
- 6.8.6 Software verification plan (refined)
- 6.8.7 Verification specification
- 6.8.8 Verification report

7 ISO26262-7 Production and Operation

- 7.1 Production
  - 7.1.1 Safety-related content of the production plan
  - 7.1.2 Safety-related content of the production control plan including the test plan
  - 7.1.3 Control measures report
  - 7.1.4 If applicable, requirements specification on the producibility at system, hardware or software development level
  - 7.1.5 Assessment report for capability of the production process
- 7.2 Operation, service (maintenance and repair) and decommissioning
  - 7.2.1 Safety-related content of the maintenance plan
  - 7.2.2 Repair instructions
  - 7.2.3 Safety-related content of the information made available to the user
  - 7.2.4 Instructions regarding field observations
  - 7.2.5 Safety-related content of the instructions for decommissioning
  - 7.2.6 If applicable, requirements specification relating to operation, service and decommissioning at system, hardware or software development level

8 ISO26262-8 Supporting Processes

- 8.1 Interfaces within distributed developments
  - 8.1.1 Supplier selection report
  - 8.1.2 Development interface agreement (DIA)
  - 8.1.3 Supplier's project plan
  - 8.1.4 Supplier's safety plan
  - 8.1.5 Safety assessment report
  - 8.1.6 Supply agreement
- 8.2 Configuration management
  - 8.2.1 Configuration management plan
- 8.3 Change management
  - 8.3.1 Change management plan
  - 8.3.2 Change request
  - 8.3.3 Impact analysis and change request plan
  - 8.3.4 Change report
- 8.4 Verification
  - 8.4.1 Verification plan
  - 8.4.2 Verification specification
  - 8.4.3 Verification report
- 8.5 Documentation
  - 8.5.1 Document management plan
  - 8.5.2 Documentation guideline requirements
- 8.6 Confidence in the use of software tools

8.6.1 Software tool criteria evaluation report
8.6.2 Software tool qualification report
8.7 Qualification of software components
8.7.1 Software component documentation
8.7.2 Software component qualification report
8.7.3 Safety plan (refined)
8.8 Qualification of hardware components
8.8.1 Qualification plan
8.8.2 Hardware component test plan
8.8.3 Qualification report
8.9 Proven in use argument
8.9.1 Safety Plan (refined)
8.9.2 Definition of a candidate for proven in use argument
8.9.3 Proven in use analysis reports
9 ISO26262-9 Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses (informative)
9.1 Requirements decomposition with respect to ASIL tailoring
9.1.1 Update of architectural information
9.1.2 Update of ASIL as attribute of safety requirements and elements
9.2 Criteria for coexistence of elements
9.2.1 Update of ASIL as attribute of sub-elements of elements
9.3 Analysis of dependent failures
9.3.1 Analysis of dependent failures
9.4 Safety analyses
9.4.1 Safety analyses

A.5 Hardware-Software Interface

Signal	Signalname	Bits	Internal cycle time (ms)	External cycle time (ms)	Source (CAN/ADC/DIO)	Supply voltage	Tolerance of sensor signal	Type of signal	Physical signal range	Scaling	Signal range in the SW	Accuracy	ASIL	Default	Latency time (ms)	Reaction time (ms)	Type (SW)
Accelerator Pedal Pos 1	AccPed1	16	5	0	ADC3.0	5V	0.02V	V	2.5-4.0V	-	0-100\%	1\%	A(B)	0\%	5ms	10ms	float
Accelerator Pedal Pos 2	AccPed2	16	5	0	ADC3.1	5V	0.02V	V	2.4-0.5V	-	0-100\%	1\%	A(B)	0\%	5ms	10ms	float
Accelerator Pedal Reference	AccPedRef	16	5	0	ADC5.4	5V	0.02V	V	4.8-5V	-	4.8-5V	0.1V	B	5V	5ms	10ms	float
Throttle Valve Pos 1	ThrVal1	16	5	0	ADC4.0	5V	0.02V	V	0.5-3.0V	-	0-100\%	1\%	A(B)	0\%	5ms	10ms	float
Throttle Valve Pos 1	ThrVal2	16	5	0	ADC4.1	5V	0.02V	V	3.0-0.5V	-	0-100\%	1\%	A(B)	0\%	5ms	10ms	float
Throttle Valve Reference	ThrValRef	16	5	0	ADC5.5	3.3V	0.01V	V	3.0-3.3V	-	3.0-3.3V	0.1V	B	3.3V	5ms	10ms	float
Output Stage Status	OutStStat	1	5	0.1	P10.6	5V	0.3V	V	0-5V	-	0/1	-	B	0V	5ms	10ms	uint8
Output Stage Enable	OutStEn	1	5	0.1	P14.8	5V	0.3V	V	0-5V	-	0/1	-	B	0V	5ms	10ms	uint8
Output Stage Disable	OutStDis	1	5	0.1	P14.7	5V	0.3V	V	0-5V	-	0/1	-	B	3.3V	5ms	10ms	uint8
Output Stage Channel1	OutStCh1	1	5	0.1	P14.9	5V	0.3V	V	0-5V	-	0-100\%	1\%	B	0V	5ms	10ms	float
Output Stage Channel2	OutStCh2	1	5	0.1	P14.10	5V	0.3V	V	0-5V	-	0-100\%	1\%	B	0V	5ms	10ms	float
Safetyswitch Disable	SafSwDis	1	5	0.1	P10.4	5V	0.3V	V	0-5V	-	0/1	-	B	3.3V	5ms	10ms	uint8
Display SPI0 CLK	SPI0CLK	1	50	5.00E-07	P20.11	5V	0.3V	V	0-5V	-	0/1	-	QM	0V	-	50ms	-
Display SPI0 \olCS	SPI0CS	1	50	5.00E-07	P20.6	5V	0.3V	V	0-5V	-	0/1	-	QM	0V	-	50ms	-
Display SPI0 MISO	SPI0MISO	1	50	5.00E-07	P20.12	5V	0.3V	V	0-5V	-	0/1	-	QM	0V	-	50ms	-

Signal	Signalname	Bits	Internal cycle time (ms)	External cycle time (ms)	Source (CAN/ADC/DIO)	Supply voltage	Tolerance of sensor signal	Type of signal	Physical signal range	Scaling	Signal range in the SW	Accuracy	ASIL	Default	Latency time (ms)	Reaction time (ms)	Type (SW)
Display SPI0 MOSI	SPI0MOSI	1	50	5.00E-07	P20.14	5V	0.3V	V	0-5V	-	0/1	-	QM	0V	-	50ms	-
LV3_HW SPI2 CLK	SPI2CLK	1	50	5.00E-07	P15.3	5V	0.3V	V	0-5V	-	0/1	-	B	0V	-	50ms	-
LV3_HW SPI2 \olCS	SPI2CS	1	50	5.00E-07	P15.2	5V	0.3V	V	0-5V	-	0/1	-	B	0V	-	50ms	-
LV3_HW SPI2 MISO	SPI2MISO	1	50	5.00E-07	P15.4	5V	0.3V	V	0-5V	-	0/1	-	B	0V	-	50ms	-
LV3_HW SPI2 MOSI	SPI2MOSI	1	50	5.00E-07	P15.5	5V	0.3V	V	0-5V	-	0/1	-	B	0V	-	50ms	-
LV3_SW SPI3 CLK	SPI3CLK	1	50	5.00E-07	P22.3	5V	0.3V	V	0-5V	-	0/1	-	B	0V	-	50ms	-
LV3_SW SPI3 \olCS	SPI3CS	1	50	5.00E-07	P22.3	5V	0.3V	V	0-5V	-	0/1	-	B	0V	-	50ms	-
LV3_SW SPI3 MISO	SPI3MISO	1	50	5.00E-07	P22.1	5V	0.3V	V	0-5V	-	0/1	-	B	0V	-	50ms	-
LV3_SW SPI3 MOSI	SPI3MOSI	1	50	5.00E-07	P22.0	5V	0.3V	V	0-5V	-	0/1	-	B	0V	-	50ms	-
Touchscreen Button	TouchScBtn	1	0.01	0.1	P10.7	5V	0.3V	V	0-5V	-	0/1	-	B	0V	5ms	5ms	uint8
Power Supply Disable	PowSupDis	1	0.01	0.1	P10.8	5V	0.3V	V	0-5V	-	0/1	-	QM	5V	5ms	-	uint8
Status LED 1	LED0	1	0.01	0.1	P33.8	5V	0.3V	V	0-5V	-	0/1	-	QM	5V	5ms	5ms	uint8
Status LED 2	LED1	1	0.01	0.1	P33.9	5V	0.3V	V	0-5V	-	0/1	-	QM	5V	5ms	5ms	uint8
Status LED 3	LED2	1	0.01	0.1	P33.10	5V	0.3V	V	0-5V	-	0/1	-	QM	5V	5ms	5ms	uint8
Status LED 4	LED3	1	0.01	0.1	P33.11	5V	0.3V	V	0-5V	-	0/1	-	QM	5V	5ms	5ms	uint8

Tabelle A.11: Hardware-Software Interface

A.6 OIL Konfigurationsdatei

Quelltext A.2: OIL Konfigurationsdatei

```
CPU EGAS_application {
  OS EE {
    EE_OPT = "EE_DEBUG";            /* Include Debug Information          */
    EE_OPT = "EE_ICACHE_ENABLED";   /* Enable Instruction Cache           */
    EE_OPT = "EE_DCACHE_ENABLED";   /* Enable Data Cache                  */

    EE_OPT = "EE_APPKIT_TC2X5";     /* Use Applikation Kit TC2X5          */
    EE_OPT = "EE_USE_DISPLAY";      /* Use Display Driver                 */
    EE_OPT = "EE_USE_BUTTONS";      /* Use Button Driver                  */
    EE_OPT = "EE_USE_LEDS";         /* Use Led Driver                     */
    EE_OPT = "CPU0_USE_ILLD";       /* Use Infineon Low Level Drivers     */
    EE_OPT = "CPU1_USE_ILLD";       /* Use Infineon Low Level Drivers     */
    EE_OPT = "CPU2_USE_ILLD";       /* Use Infineon Low Level Drivers     */

    CFLAGS = " -Wc-w537 ";          /* Warning: Unused Variable           */
    CFLAGS = " -Wc-w549 ";          /* Warning: Condition is always true  */
    /* Include Paths for Infineon Low Level Drivers */
    CFLAGS = "-I../../BSW/Driver/_ifx/TC27xC";
        CFLAGS = "-I../../BSW/Driver/_ifx/TC27xC/_Reg";
    CFLAGS = "-I../../BSW/Driver/_ifx/SrvSw -I../../BSW/Driver/_ifx";
    CFLAGS = "-I../../BSW/Driver/_ifx/TC27xC/Cpu/Std";



    REMOTENOTIFICATION  = USE_RPC;  /* Use Remote Procedure Call System   */
    MASTER_CPU          = "cpu0";   /* Use "cpu0" ID as Master Cpu        */
    STATUS              = EXTENDED; /* Enable extended Status             */
    ERRORHOOK           = TRUE;     /* Enable Errorhooks                  */
    PRETASKHOOK         = TRUE;     /* Enable PreTaskhooks (For Tracing)  */
    POSTTASKHOOK        = TRUE;     /* Enable PostTaskhooks (For Tracing) */
    USEREMOTETASK       = ALWAYS;   /* Enable Control of Remote Tasks and Events
        */
    KERNEL_TYPE         = ECC2;     /* Set Kernel Type: BCC1/2, ECC1/2, FIFO
        */


    CPU_DATA = TRICORE {            /* CPU Type                           */
      ID              = "cpu0";     /* CPU ID                             */
      CPU_CLOCK       = 200.0;      /* CPU Frequency                      */
            /* ASW Sources (CPU specific) */
      APP_SRC         = "$(shell find ../../ASW/Cpu0/   -name '*.c')";
            /* ASW Sources (Shared)       */
      APP_SRC         = "$(shell find ../../ASW/Shared/ -name '*.c')";
            /* Common Files (unspecific)  */
      APP_SRC         = "$(shell find ../../ASW/Common/ -name '*.c')";
            /* BSW Sources (CPU specific) */
      APP_SRC         = "$(shell find ../../BSW/Cpu0/   -name '*.c')";
            /* BSW Sources (Shared)       */
      APP_SRC         = "$(shell find ../../BSW/Shared/ -name '*.c')";
            /* Infineon Low Level Drivers */
      APP_SRC         = "$(shell find ../../BSW/Driver/ -name '*.c')";
            /* Common Files (unspecific)  */
      APP_SRC         = "$(shell find ../../BSW/Common/ -name '*.c')";
      MULTI_STACK     = TRUE;      /* Allow Multiple Stacks              */
      SYS_STACK_SIZE  = 1024;      /* System Stack Size                  */
      COMPILER_TYPE   = TASKING;   /* Set Compiler in Use: GNU / TASKING */
    };

    CPU_DATA = TRICORE {
      ID              = "cpu1";
```

```
    APP_SRC         = "$(shell find ../../ASW/Cpu1/   -name '*.c')";
    APP_SRC         = "$(shell find ../../ASW/Common/ -name '*.c')";
    APP_SRC         = "$(shell find ../../BSW/Cpu1/   -name '*.c')";
    APP_SRC         = "$(shell find ../../BSW/Common/ -name '*.c')";
    APP_SRC         = "$(shell find ../../BSW/Driver/ -name '*.c')";
    MULTI_STACK     = TRUE;
    SYS_STACK_SIZE  = 512;
    COMPILER_TYPE   = TASKING;
  };

  CPU_DATA = TRICORE {
    ID              = "cpu2";
    APP_SRC         = "$(shell find ../../ASW/Cpu2/   -name '*.c')";
    APP_SRC         = "$(shell find ../../ASW/Common/ -name '*.c')";
    APP_SRC         = "$(shell find ../../BSW/Cpu2/   -name '*.c')";
    APP_SRC         = "$(shell find ../../BSW/Common/ -name '*.c')";
    APP_SRC         = "$(shell find ../../BSW/Driver/ -name '*.c')";
    MULTI_STACK     = TRUE;
    SYS_STACK_SIZE  = 512;
    COMPILER_TYPE   = TASKING;
  };

      /* Use TC27X as MCU           */
  MCU_DATA          = TRICORE { MODEL = TC27x; };
      /* Board is defined as EE_OPT */
  BOARD_DATA        = NO_BOARD;
};

/* COUNTERS ******************************************************/
COUNTER system_timer_cpu0 {       /* Define system Counter             */
  CPU_ID          = "cpu0";       /* Assign Counter to a CPU           */
  MINCYCLE        = 1;            /* Minimum allowed counter ticks     */
  MAXALLOWEDVALUE = 2147483647;   /* Maximum Counter Value before Reset */
  TICKSPERBASE    = 1;
  TYPE = HARDWARE {               /* Specify Hardware/Software Counter  */
    DEVICE        = "STM_SR0";    /* Use this Hardware Counter          */
    SYSTEM_TIMER  = TRUE;
    PRIORITY      = 2;            /* Set Priority of Counter Interrupt  */
  };
  SECONDSPERTICK  = 0.001;        /* Define Counter Period in Seconds   */
};

COUNTER system_timer_cpu1 {
  CPU_ID          = "cpu1";
  MINCYCLE        = 1;
  MAXALLOWEDVALUE = 2147483647;
  TICKSPERBASE    = 1;
  TYPE = HARDWARE {
    DEVICE        = "STM_SR0";
    SYSTEM_TIMER  = TRUE;
    PRIORITY      = 2;
  };
  SECONDSPERTICK  = 0.001;
};

COUNTER system_timer_cpu2 {
  CPU_ID          = "cpu2";
  MINCYCLE        = 1;
  MAXALLOWEDVALUE = 2147483647;
  TICKSPERBASE    = 1;
  TYPE = HARDWARE {
    DEVICE        = "STM_SR0";
    SYSTEM_TIMER  = TRUE;
    PRIORITY      = 2;
  };
  SECONDSPERTICK  = 0.001;
};
```

```
/* TASKS  CPU0 ****************************************/
TASK TaskCpu0_WDT {       /* Task Name                    */
  CPU_ID      = "cpu0";   /* Specify CPU to run on        */
  PRIORITY    = 0;        /* Priority Level               */
  AUTOSTART   = FALSE;    /* Don't start at startup       */
  STACK       = SHARED;   /* Use common stack             */
  ACTIVATION  = 1;        /* Maximum allowed activations  */
  SCHEDULE    = NON;     /* Use Scheduler for Preemtion  */
};
TASK TaskCpu0_lvl3_ChainLink {
  CPU_ID      = "cpu0";
  PRIORITY    = 1;
  AUTOSTART   = FALSE;
  STACK       = SHARED;
  ACTIVATION  = 1;
  SCHEDULE    = NON;
};
TASK TaskCpu0_MotorControl {
  CPU_ID      = "cpu0";
  PRIORITY    = 2;
  AUTOSTART   = TRUE;
  STACK       = PRIVATE { SYS_SIZE = 256; };
  ACTIVATION  = 1;
  SCHEDULE    = FULL;
  EVENT       = MCOnEvent;
  EVENT       = MCOffEvent;
};
TASK TaskCpu0_5ms {
  CPU_ID      = "cpu0";
  PRIORITY    = 3;
  AUTOSTART   = FALSE;
  STACK       = SHARED;
  ACTIVATION  = 1;
  SCHEDULE    = FULL;
  TIMING_PROTECTION = TRUE {
   EXECUTIONBUDGET         = 0.0025;
  };
};


/* TASKS  CPU1 ****************************************/
TASK TaskCpu1_WDT {
  CPU_ID      = "cpu1";
  PRIORITY    = 0;
  AUTOSTART   = FALSE;
  STACK       = SHARED;
  ACTIVATION  = 1;
  SCHEDULE    = NON;
};
TASK TaskCpu1_lvl3_ChainLink {
  CPU_ID      = "cpu1";
  PRIORITY    = 1;
  AUTOSTART   = FALSE;
  STACK       = SHARED;
  ACTIVATION  = 1;
  SCHEDULE    = NON;
};
TASK TaskCpu1_lvl3_SW {
  CPU_ID      = "cpu1";
  PRIORITY    = 2;
  AUTOSTART   = FALSE;
  STACK       = PRIVATE { SYS_SIZE = 256; };
  ACTIVATION  = 1;
  SCHEDULE    = FULL;
  EVENT       = LVL3_CPU0done;
  EVENT       = LVL3_CPU1done;
```

```
  EVENT       = LVL3_CPU2done;
  EVENT       = Cpu1_Timeout;
};
TASK TaskCpu1_10ms {
  CPU_ID      = "cpu1";
  PRIORITY    = 3;
  AUTOSTART   = FALSE;
  STACK       = SHARED;
  ACTIVATION  = 1;
  SCHEDULE    = NON;
  };

TASK TaskCpu1_50ms {
  CPU_ID      = "cpu1";
  PRIORITY    = 4;
  AUTOSTART   = FALSE;
  STACK       = SHARED;
  ACTIVATION  = 1;
  SCHEDULE    = FULL;
};



/* TASKS  CPU2 ****************************************/
TASK TaskCpu2_50ms {
  CPU_ID      = "cpu2";
  PRIORITY    = 0;
  AUTOSTART   = FALSE;
  STACK       = SHARED;
  ACTIVATION  = 1;
  SCHEDULE    = FULL;
};
  TASK TaskCpu2_WDT {
  CPU_ID      = "cpu2";
  PRIORITY    = 1;
  AUTOSTART   = FALSE;
  STACK       = SHARED;
  ACTIVATION  = 1;
  SCHEDULE    = NON;
};

TASK TaskCpu2_lvl3_ChainLink {
  CPU_ID      = "cpu2";
  PRIORITY    = 2;
  AUTOSTART   = FALSE;
  STACK       = SHARED;
  ACTIVATION  = 1;
  SCHEDULE    = NON;
};
  TASK TaskCpu2_MotorControl {
  CPU_ID      = "cpu2";
  PRIORITY    = 3;
  AUTOSTART   = TRUE;
  STACK       = PRIVATE { SYS_SIZE = 256; };
  ACTIVATION  = 1;
  SCHEDULE    = FULL;
  EVENT       = TouchscreenEvent;
};
TASK TaskCpu2_lvl3_HW {
  CPU_ID      = "cpu2";
  PRIORITY    = 4;
  AUTOSTART   = FALSE;
  STACK       = SHARED;
  ACTIVATION  = 1;
  SCHEDULE    = NON;
};
```

```
/* SPINLOCKS ************************************************/
SPINLOCK SL_HMI;
SPINLOCK SL_MCEnable;
SPINLOCK SL_Carry;

/* RESOURCES ************************************************/


/* EVENTS ***************************************************/
EVENT MCOnEvent  { MASK = AUTO; };
EVENT MCOffEvent { MASK = AUTO; };
EVENT LVL3_CPU0done { MASK = AUTO; };
EVENT LVL3_CPU1done { MASK = AUTO; };
EVENT LVL3_CPU2done { MASK = AUTO; };
EVENT TouchscreenEvent { MASK = AUTO; };
EVENT Cpu1_Timeout { MASK = AUTO; };

/****** ALARMS CPU0 ******/
ALARM AlarmCpu0_WDT {
  COUNTER   = system_timer_cpu0;
  ACTION    = ACTIVATETASK { TASK = TaskCpu0_WDT;};
  AUTOSTART = TRUE { ALARMTIME = 9; CYCLETIME = 10; };
};
ALARM AlarmCpu0_5ms {
  COUNTER   = system_timer_cpu0;
  ACTION    = ACTIVATETASK { TASK = TaskCpu0_5ms;};
  AUTOSTART = TRUE { ALARMTIME = 1; CYCLETIME = 5; };
};


/****** ALARMS CPU1 ******/
ALARM AlarmCpu1_WDT {
  COUNTER   = system_timer_cpu1;
  ACTION    = ACTIVATETASK { TASK = TaskCpu1_WDT;};
  AUTOSTART = TRUE { ALARMTIME = 1; CYCLETIME = 10; };
};
ALARM AlarmCpu1_10ms {
  COUNTER   = system_timer_cpu1;
  ACTION    = ACTIVATETASK { TASK = TaskCpu1_10ms; };
  AUTOSTART = TRUE { ALARMTIME = 9; CYCLETIME = 10; };
};
  ALARM AlarmCpu1_50ms {
  COUNTER   = system_timer_cpu1;
  ACTION    = ACTIVATETASK { TASK = TaskCpu1_50ms; };
  AUTOSTART = TRUE { ALARMTIME = 45; CYCLETIME = 50; };
};
  ALARM AlarmCpu1_lvl3_SW_Timeout {
  COUNTER   = system_timer_cpu1;
  ACTION    = SETEVENT { TASK = TaskCpu1_lvl3_SW; EVENT = Cpu1_Timeout; };
  AUTOSTART = FALSE;
};


/****** ALARMS CPU2 ******/
ALARM AlarmCpu2_WDT {
  COUNTER   = system_timer_cpu2;
  ACTION    = ACTIVATETASK { TASK = TaskCpu2_WDT;};
  AUTOSTART = TRUE { ALARMTIME = 9; CYCLETIME = 10; };
};
ALARM AlarmCpu2_50ms {
  COUNTER   = system_timer_cpu2;
  ACTION    = ACTIVATETASK { TASK = TaskCpu2_50ms; };
  AUTOSTART = TRUE { ALARMTIME = 1; CYCLETIME = 50; };
};

ALARM AlarmCpu2_lvl3 {
```

```
    COUNTER   = system_timer_cpu2;
    ACTION    = ACTIVATETASK { TASK = TaskCpu2_lvl3_HW; };
    AUTOSTART = TRUE { ALARMTIME = 40; CYCLETIME = 50; };
  };


  /****** ISRs CPU1 ******/
  ISR qspi3TxISR {
      CPU_ID    = "cpu1";
    CATEGORY  = 1;
    PRIORITY  = 10;
  };

  ISR qspi3RxISR {
      CPU_ID    = "cpu1";
    CATEGORY  = 1;
    PRIORITY  = 20;
  };

  ISR qspi3ErISR {
      CPU_ID    = "cpu1";
    CATEGORY  = 1;
    PRIORITY  = 0x30;
  };

  /****** ISRs CPU2 ******/
  ISR ISRTouchscreen {
    CPU_ID    = "cpu2";
    CATEGORY  = 2;
    PRIORITY  = 3;
  };

  ISR qspi2TxISR {
      CPU_ID    = "cpu2";
    CATEGORY  = 1;
    PRIORITY  = 30;
  };

  ISR qspi2RxISR {
      CPU_ID    = "cpu2";
    CATEGORY  = 1;
    PRIORITY  = 40;
  };

  ISR qspi2ErISR {
      CPU_ID    = "cpu2";
    CATEGORY  = 1;
    PRIORITY  = 0x31;
  };


};
```

Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Abkürzungsverzeichnis

ADC	Analog-Digital Konverter
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
ASIL	Automotive Safety Integration Level
ASW	Anwendungs-Software
AUTOSAR	Automotive Open Systems Architecture
BSW	Basis-Software
CAN	Controller Area Network
COTS	Commercial Off-The-Shelf
COM	Communication
CPU	Central Processing Unit
DEW	Drosselklappenersatzwert
DMA	Direct Memory Access
DK	Drosselklappe
DKS	Drosselklappensensor
E/E/PE	Elektrisch/Elektronisch/Programmierbar-Elektronisch
E/E	Elektrisch/Elektronisch
ECU	Electric Control Unit
EMB	Einspritzmengenbegrenzung
ERIKA	Embedded Real Time Kernel Architecture
EUC	Equipment Under Control
E3-ÜM	Überwachungsmodul für Ebene 3 des Sicherheitskonzepts
E3-SW	Überwachungssoftware für Ebene 3 des Sicherheitskonzepts
FB	Fehlerbeschreibung
FGR	Fahrgeschwindigkeitsregelung
FIFO	First In – First Out
FIT	Failure in Time
FMEA	Failure Mode and Effects Analysis

FSK	Funktionales Sicherheitskonzept
FTA	Fault Tree Analysis
HAL	Hardware Abstraction Layer
HARA	Hazard Analysis and Risks Assessment
HMI	Human-Machine-Interface
HSI	Hardware-Software-Interface
HW	Hardware
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IOC	Inter-OS-Application Communication
ISO	Internationale Organisation für Normung
ISR	Interrupt Service Routine
LFM	Latent Fault Metric
MCAL	MicroController Abstraction Layer
MIL	Model In the Loop
MMU	Memory Management Unit
MPSoC	MultiProcessor System-on-Chip
MPU	Memory Protection Unit
MUX	Multiplexer
NM	Network Management
OEM	Original Equipment Manufacturer
OIL	OSEK Interpration Language
ORTI	OIL RealTime Interface
OS	Operating System
OSEK	Offene Systeme für die Elektronik im Kraftfahrzeug
PLL	Phase-Locked Loop
PKW	Personenkraftwagen
PWM	Pulsweitenmodulation
QM	Qualitätsmanagement
RAM	Random-Access Memory
ROM	Read-only Memory
RTE	Runtime Environment
RTOS	Real Time Operating System
SANF	Sicherheitsanforderung
SPFM	Single Point Fault Metric

SPI	Serial Peripheral Interface
SRI	Shared Resource Interconnect
SW	Software
SZ	Sicherheitsziel
VDX	Vehicle Distributed Executive
WCET	Worst Case Execution Time
WDT	Watchdog Timer

Literaturverzeichnis

[1] Altium. *TASKING VX-toolset for TriCore User Guide*. v4.3. Juli 2013.

[2] G. M. Amdahl. „Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities“. In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. AFIPS '67 (Spring). Atlantic City, New Jersey: ACM, 1967, Seiten 483–485. DOI: 10.1145/1465482.1465560. URL: http://doi.acm.org/10.1145/1465482.1465560.

[3] Arbeitskreis EGAS. *Standardisiertes E-Gas Überwachungskonzept für Benzin und Diesel Motorsteuerungen*. Technischer Bericht. Arbeitskreis EGAS, Juli 2013. URL: https://www.iav.com/publikationen/technische-veroeffentlichungen/e-gas-monitoring-concepts.

[4] AUTOSAR Development Cooperation. *AUTOSAR AUTomotive Open System ARchitecture*. Englisch. AUTOSAR Development Cooperation, 2009. URL: www.autosar.org.

[5] AUTOSAR Development Cooperation. *Technical Safety Concept Status Report*. Technischer Bericht Document Version: 1.1.0, Revision 2. AUTOSAR development cooperation, Okt. 2010. URL: %7Bwww.autosar.org%7D.

[6] AUTOSAR Development Cooperation. *Guide to Multi-Core Systems*. Englisch. 2013. URL: %7Bwww.autosar.org%7D.

[7] G. Baumgarten u. a. „First results of automatic fault-injection in an AUTOSAR tool-chain“. In: *Industrial Informatics (INDIN), 2014 12th IEEE International Conference on*. Juli 2014, Seiten 170–175. DOI: 10.1109/INDIN.2014.6945503.

[8] J. Birch u. a. „Safety Cases and Their Role in ISO 26262 Functional Safety Assessment“. English. In: *Computer Safety, Reliability, and Security*. Herausgegeben von F. Bitsch, J. Guiochet und M. Kaâniche. Band 8153. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, Seiten 154–165. ISBN: 978-3-642-40792-5. DOI: 10.1007/978-3-642-40793-2_15. URL: http://dx.doi.org/10.1007/978-3-642-40793-2_15.

[9] S. Brewerton u. a. *Demonstration of Automotive Steering Column Lock using Multicore AUTOSAR Operating System*. SAE Tech Paper 2012-01-0031. SAE International, 2012. DOI: 10.4271/2012-01-0031.

[10] D. Chen u. a. „Integrated safety and architecture modeling for automotive embedded systems*“. English. In: *e & i Elektrotechnik und Informationstechnik* 128.6 (2011), Seiten 196–202. ISSN: 0932-383X. DOI: 10.1007/s00502-011-0007-7. URL: http://dx.doi.org/10.1007/s00502-011-0007-7.

[11] S. Christiaens, J. Ogrzewalla und S. Pischinger. *Functional Safety for Hybrid and Electric Vehicles*. SAE Technical Paper 2012-01-0032. Warrendale, PA: SAE International, Apr. 2012. URL: http://papers.sae.org/2012-01-0032/ (besucht am 23. 05. 2015).

[12] D. Claraz u. a. „Introducing Multi-Core at Automotive Engine Systems“. In: *ERTS 14*. ERTSS, 2014.

[13] D. Cortese. „New Model-Based Paradigm: Developing Embedded Software to the Functional Safety Standards, as ISO 26262, ISO 25119 and ISO 13849 through an efficient automation of Sw Development Life-Cycle“. In: *SAE Technical Paper*. SAE International, Sep. 2014. DOI: 10.4271/2014-01-2394. URL: http://dx.doi.org/10.4271/2014-01-2394.

[14] K. Devika und R. Syama. „An Overview of AUTOSAR Multicore Operating System Implementation“. In: *International Journal of Innovative Research in Science, Engineering and Technology*. Band Vol. 2. Issue 7. IJIRSET. 2013, 3163–3169.

[15] Eclipse Foundation. *About the Eclipse Foundation*. Website. 2015. URL: http://www.eclipse.org/org/ (besucht am 23. 05. 2015).

[16] K. Etschberger. *Controller Area Network (CAN) Grundlagen, Protokolle, Bausteine, Anwendungen. Dritte Auflage*. Hanser Verlag. Apr. 2002.

[17] Evidence Srl. *ERIKA Enterprise Manual*. 1.4.5. Real-time made easy. Dez. 2012. URL: http://erika.tuxfamily.org/download/manuals/pdf/ee_refman_1_4_5.pdf.

[18] Evidence Srl. *RT-Druid reference manual*. 1.5.0. Dez. 2012. URL: http://erika.tuxfamily.org/download/manuals/pdf/rtdruid_refman_1_5.0.pdf.

[19] C. Ficek, N. Feiertag und K. Richter. *Applying the AUTOSAR timing protection to build safe and efficient ISO 26262 mixed-criticality systems*. Technischer Bericht. ERTS, 2012.

[20] C. Ficek, K. Richter und N. Feiertag. „Schedule Design to Guarantee Freedom of Interference in Mixed Criticality Systems“. In: *SAE International Journal of Passenger Cars - Electronic and Electrical Systems* 5.1 (Jan. 2012), Seiten 46–54. ISSN: 1946-4614, 1946-4622. DOI: 10.4271/2012-01-0036. URL: http://saepcelec.saejournals.org/content/5/1/46 (besucht am 22. 05. 2015).

[21] V. Gebhardt u. a. *Funktionale Sicherheit nach ISO 262626 - Ein Praxisleitfaden zur Umsetzung*. Band 1. Auflage. dpunkt.verlag, 2013.

[22] D. Gizopoulos, A. Paschalis und Y. Zorian. „Processor-Based Testing of SoC“. English. In: *Embedded Processor-Based Self-Test*. Band 28. Frontiers in Electronic Testing. Springer US, 2004, Seiten 185–194. ISBN: 978-1-4419-5252-3. DOI: 10.1007/978-1-4020-2801-4_7. URL: http://dx.doi.org/10.1007/978-1-4020-2801-4_7.

[23] M. Graniou, H. Sivencrona und R. Svenningsson. „Advantages and Challenges of Introducing AUTOSAR for Safety-Related Systems“. In: *SAE Technical Paper*. SAE International, Apr. 2009. DOI: 10.4271/2009-01-0750. URL: http://dx.doi.org/10.4271/2009-01-0750.

[24] J. Han u. a. *Efficient Multi-Core Software Design Space Exploration for Hybrid Control Unit Integration*. SAE Technical Paper 2014-01-0260. Warrendale, PA: SAE International, 2014. URL: http://papers.sae.org/2014-01-0260/ (besucht am 22. 05. 2015).

[25] R. Hilbrich, J. van Kampenhout und H.-J. Goltz. „Modellbasierte Generierung statischer Schedules für sicherheitskritische, eingebettete Systeme mit Multicore-Prozessoren und harten Echtzeitanforderungen“. German. In: *Herausforderungen durch Echtzeitbetrieb*. Herausgegeben von W. A. Halang. Informatik aktuell. Springer Berlin Heidelberg, 2012, Seiten 29–38. ISBN: 978-3-642-24657-9. DOI: 10.1007/978-3-642-24658-6_4. URL: http://dx.doi.org/10.1007/978-3-642-24658-6_4.

[26] M. Hillenbrand. *Funktionale Sicherheit nach ISO 26262 in der Konzeptphase der Entwicklung von Elektrik/Elektronik Architekturen von Fahrzeugen*. Herausgegeben von M. Hillenbrand. KIT Scientific Publishing, 2012.

[27] M. Homann. *OSEK: Betriebssystem-Standard für Automotive und Embedded Systems*. [Konfiguration, Kommunikation, Netzwerkmanagement, Design Patterns, detaillierte Dokumenatation aller API-Funktionen, mit zeitlich unbegrenzter Demoversion (Windows) auf CD]. mitp-Verlag, 2005. ISBN: 9783826615528.

[28] Infineon Technologies AG. *7 A H-Bridge for DC-Motor Applications TLE 7209-2R*. 1.3. Jan. 2005.

[29] Infineon Technologies AG. *Application Kit TC2X5 User's Manual*. 2013-05. Infineon Technologies AG. 81726 Munich, Germany, Mai 2013.

[30] Infineon Technologies AG. *AURIX TC27x C-Step User's Manual*. V2.0. Infineon Technologies AG. 81726 Munich, Germany, Juli 2014.

[31] Infineon Technologies AG. *TriCore™ Brochure including AURIX™*. 85579 Neubiberg, Germany, Feb. 2014. URL: http://www.infineon.com/dgdl/Infineon-Tricore+Family+BR_2015-BC-v01_00-EN.pdf.

[32] ISO - International Organization for Standardization. *IEC 61508 Functional safety of electrical/ electronic / programmable electronic safety-related systems*. International Organization for Standardization.

[33] ISO - International Organization for Standardization. *ISO 17356 Road vehicles – Open interface for embedded automotive applications*. International Organization for Standardization, 2005.

[34] ISO - International Organization for Standardization. *IEC 60812 Analysis techniques for system reliability - Procedure for failure mode and effects analysis (FMEA)*. International Organization for Standardization, 2006.

[35] ISO - International Organization for Standardization. *IEC 61025 Fault tree analysis (FTA)*. International Organization for Standardization, Dez. 2006.

[36] ISO - International Organization for Standardization. *ISO 26262 Road vehicles Functional Safety Part 1 - Vocabulary*. International Organization for Standardization, 2011.

[37] ISO - International Organization for Standardization. *ISO 26262 Road vehicles Functional Safety Part 1-10*. International Organization for Standardization, 2011.

[38] ISO - International Organization for Standardization. *ISO 26262 Road vehicles Functional Safety Part 3 - Concept phase*. International Organization for Standardization, 2011.

[39] ISO - International Organization for Standardization. *ISO 26262 Road vehicles Functional Safety Part 4 - Product development at the system level*. International Organization for Standardization, 2011.

[40] ISO - International Organization for Standardization. *ISO 26262 Road vehicles Functional Safety Part 5 - Product development at the hardware level*. International Organization for Standardization, 2011.

[41] ISO - International Organization for Standardization. *ISO 26262 Road vehicles Functional Safety Part 6 - Product development at the system software level*. International Organization for Standardization, 2011.

[42] ISO - International Organization for Standardization. *ISO 26262 Road vehicles Functional Safety Part 10 - Guideline on ISO26262*. International Organization for Standardization, 2012.

[43] D. Kästner u. a. „Meeting real-time requirements with multi-core processors“. In: *Computer Safety, Reliability, and Security*. Springer, 2012, Seiten 117–131.

[44] O. Kindel und M. Friedrich. *Softwareentwicklung mit AUTOSAR: Grundlagen, Engineering, Management in der Praxis*. Herausgegeben von O. Kindel und M. Friedrich. dpunkt, 2009.

[45] F. Kluge u. a. „Implementing AUTOSAR Scheduling and Resource Management on an Embedded SMT Processor“. In: *Proceedings of th 12th International Workshop on Software and Compilers for Embedded Systems*. SCOPES '09. Nice, France: ACM, 2009, Seiten 33–42. ISBN: 978-1-60558-696-0. URL: http://dl.acm.org/citation.cfm?id=1543820.1543828.

[46] H. Kopetz u. a. „Automotive Software Development for a Multi-Core System-on-a-Chip“. In: *Fourth International Workshop on Software Engineering for Automotive Systems, 2007. ICSE Workshops SEAS '07*. Fourth International Workshop on Software Engineering for Automotive Systems, 2007. ICSE Workshops SEAS '07. Mai 2007, Seiten 2–2. DOI: 10.1109/SEAS.2007.2.

[47] N. Kranitis u. a. „Software-Based Self-Testing of Embedded Processors“. English. In: *Processor Design*. Herausgegeben von J. Nurmi. Springer Netherlands, 2007, Seiten 447–481. ISBN: 978-1-4020-5529-4. DOI: 10.1007/978-1-4020-5530-0_20. URL: http://dx.doi.org/10.1007/978-1-4020-5530-0_20.

[48] E. Lalo und M. Deubzer. „Effects of Task Priority Assignment in Embedded Multicore Real-Time Systems“. In: *Embedded Systems Engineering* (Juni 2014).

[49] P. Leteinturier, S. Brewerton und K. Scheibert. „MultiCore Benefits & Challenges for Automotive Applications“. In: *SAE Technical Paper*. SAE International, Apr. 2008. DOI: 10.4271/2008-01-0989.

[50] Y. Li, S. Makar und S. Mitra. „CASP: Concurrent Autonomous Chip Self-test Using Stored Test Patterns“. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. DATE '08. Munich, Germany: ACM, 2008, Seiten 885–890. ISBN: 978-3-9810801-3-1. DOI: 10.1145/1403375.1403590. URL: http://doi.acm.org/10.1145/1403375.1403590.

[51] P. Löw, R. Pabst und E. Petry. *Funktionale Sicherheit in der Praxis: Anwendung von DIN EN 61508 und ISO/DIS 26262 bei der Entwicklung von Serienprodukten*. Dpunkt.Verlag GmbH, 2010. ISBN: 9783898645706.

[52] G. Macher. *Seamless Model-Based Safety Engineering from Requirement to Implementation*. Englisch. Institute for Technical Informatics. Nov. 2014. URL: http://ceur-ws.org/.

[53] G. Macher, E. Armengaud und C. Kreiner. „Bridging Automotive Systems, Safety and Software Engineering by a Seamless Tool Chain“. In: *7th European Congress Embedded Real Time Software and Systems Proceedings*. 2014, 256–263.

[54] G. Macher u. a. „Automotive Embedded Software: Migration Challenges to Multi-Core Computing Platforms“. INDIN 2015 IEEE International Conference on Industrial Informatics. 2015.

[55] R. Mader u. a. „Automatic and optimal allocation of safety integrity levels“. In: *Reliability and Maintainability Symposium (RAMS), 2012 Proceedings - Annual*. Jan. 2012, Seiten 1–6. DOI: 10.1109/RAMS.2012.6175431.

[56] G. Marsaglia u. a. „Xorshift rngs“. In: *Journal of Statistical Software* 8.14 (2003), Seiten 1–6.

[57] MathWorks. *Embedded Coder*. 2014.

[58] G. E. Moore. „Cramming more components onto integrated circuits“. In: *Electronics* 38.8 (1965), Seiten 114–117.

[59] B. Moyer. *Real World Multicore Embedded Systems*. Herausgegeben von B. Moyer. 1. Auflage. Newnes, Mai 2013.

[60] F. Nemati, M. Behnam und T. Nolte. „Efficiently Migrating Real-Time Systems to Multi-Cores“. In: *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*. 2009.

[61] J. Nowotsch und M. Paulitsch. „Leveraging Multi-core Computing Architectures in Avionics“. In: *Dependable Computing Conference (EDCC), 2012 Ninth European*. Mai 2012, Seiten 132–143. DOI: 10.1109/EDCC.2012.27.

[62] OSEK/VDX Steering Committee. *OSEK/VDX System Generation OIL: OSEK Implementation Language*. http://portal.osek-vdx.org/files/pdf/specs/oil25.pdf. OSEK Group, 2004. URL: %7Bhttp://portal.osek-vdx.org/files/pdf/specs/oil25.pdf%7D.

[63] OSEK/VDX Steering Committee. *OSEK/VDX Operating Systems*. http://portal.osek-vdx.org/files/pdf/specs/os223.pdf. OSEK Group, Feb. 2005. URL: %7Bhttp://portal.osek-vdx.org/files/pdf/specs/os223.pdf%7D.

[64] A. Paschalis und D. Gizopoulos. „Effective software-based self-test strategies for on-line periodic testing of embedded processors“. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 24.1 (Jan. 2005), Seiten 88–99. ISSN: 0278-0070. DOI: 10.1109/TCAD.2004.839486.

[65] L. Pintard u. a. *Using Fault Injection to Verify an AUTOSAR Application According to the ISO 26262*. SAE Technical Paper 2015-01-0272. Warrendale, PA: SAE International, Apr. 2015. URL: http://papers.sae.org/2015-01-0272/ (besucht am 23. 04. 2015).

[66] F. J. Pollack. „New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies (Keynote Address)(Abstract Only)“. In: *Proceedings of the 32Nd Annual ACM/IEEE International Symposium on Microarchitecture*. MICRO 32. Haifa, Israel: IEEE Computer Society, 1999, Seiten 2–. ISBN: 0-7695-0437-X. URL: http://dl.acm.org/citation.cfm?id=320080.320082.

[67] A. Pruckner, R. Stroph und P. Pfeffer. „Drive-By-Wire“. English. In: *Handbook of Intelligent Vehicles*. Herausgegeben von A. Eskandarian. Springer London, 2012, Seiten 235–282. ISBN: 978-0-85729-084-7. DOI: 10.1007/978-0-85729-085-4_11. URL: http://dx.doi.org/10.1007/978-0-85729-085-4_11.

[68] P. Radojković u. a. „On the Evaluation of the Impact of Shared Resources in Multithreaded COTS Processors in Time-critical Environments“. In: *ACM Trans. Archit. Code Optim.* 8.4 (Jan. 2012), 34:1–34:25. ISSN: 1544-3566. DOI: 10.1145/2086696.2086713. URL: http://doi.acm.org/10.1145/2086696.2086713.

[69] K. Richter und S. Schliecker. „Sicher auf Multi-Core umsteigen“. In: *HANSER automotive* 10 (2013).

[70] C. Salloum u. a. „The ACROSS MPSoC – A New Generation of Multi-core Processors Designed for Safety-Critical Embedded Systems“. In: *2012 15th Euromicro Conference on Digital System Design (DSD)*. 2012 15th Euromicro Conference on Digital System Design (DSD). Sep. 2012, Seiten 105–113. DOI: 10.1109/DSD.2012.126.

[71] J. Schäuffele und T. Zurawka. *Automotive Software Engineering*. de. Wiesbaden: Springer Fachmedien Wiesbaden, 2013. ISBN: 978-3-8348-2469-1, 978-3-8348-2470-7. URL: http://link.springer.com/10.1007/978-3-8348-2470-7 (besucht am 21. 04. 2015).

[72] S. Schliecker u. a. „System Level Performance Analysis for Real-Time Automotive Multicore and Network Architectures“. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 28.7 (Juli 2009), Seiten 979–992. ISSN: 0278-0070. DOI: 10.1109/TCAD.2009.2013286.

[73] D. Schneider, E. Armengaud und E. Schoitsch. „Towards Trust Assurance and Certification in Cyber-Physical Systems“. English. In: *Computer Safety, Reliability, and Security*. Herausgegeben von A. Bondavalli, A. Ceccarelli und F. Ortmeier. Band 8696. Lecture Notes in Computer Science. Springer International Publishing, 2014, Seiten 180–191. ISBN: 978-3-319-10556-7. DOI: 10.1007/978-3-319-10557-4_21.

[74] J. Schoof. „OSEK/VDX-OS — Betriebssystemstandard für Steuergeräte in Kraftfahrzeugen“. German. In: *PEARL 2000*. Herausgegeben von P. Holleczek. Informatik aktuell. Springer Berlin Heidelberg, 2000, Seiten 43–52. ISBN: 978-3-540-41210-6. DOI: 10.1007/978-3-642-59575-2_5. URL: http://dx.doi.org/10.1007/978-3-642-59575-2_5.

[75] J. Swingler und J. McBride. „The synergistic relationship of stresses in the automotive connector“. In: *Proceedings of the 19th International Conference on Electric Contact Phenomena*. 1998, Seiten 141–145. URL: http://eprints.soton.ac.uk/21175/.

[76] A. S. Tanenbaum und H. Bos. *Modern Operating Systems*. 4th. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014. ISBN: 013359162X, 9780133591620.

[77] G. Wassen, S. Lankes und T. Bemmer. „Harte Echtzeit für Anwendungsprozesse in Standard-Betriebssystemen auf Mehrkernprozessoren“. German. In: *Herausforderungen durch Echtzeitbetrieb*. Herausgegeben von W. A. Halang. Informatik aktuell. Springer Berlin Heidelberg, 2012, Seiten 39–48. ISBN: 978-3-642-24657-9. DOI: 10.1007/978-3-642-24658-6_5. URL: http://dx.doi.org/10.1007/978-3-642-24658-6_5.

[78] U. Wilhelm, S. Ebel und A. Weitzel. „Funktionale Sicherheit und ISO 26262“. In: *Handbuch Fahrerassistenzsysteme*. Herausgegeben von H. Winner u. a. ATZ/MTZ-Fachbuch. Springer Fachmedien Wiesbaden, 2015, Seiten 85–103. ISBN: 978-3-658-05733-6, 978-3-658-05734-3. DOI: 10.1007/978-3-658-05734-3_6. (Besucht am 22. 05. 2015).

[79] G. Yang u. a. „Model-based Design and Verification of Automotive Electronics Compliant with OSEK/VDX“. In: *Proceedings of the Second International Conference on Embedded Software and Systems*. ICESS '05. Washington, DC, USA: IEEE Computer Society, 2005, Seiten 237–245. ISBN: 0-7695-2512-1. DOI: 10.1109/ICESS.2005.70. URL: http://dx.doi.org/10.1109/ICESS.2005.70.

[80] R. Zalman, A. Griessing und P. Emberson. „Timing Correctness in Safety-Related Automotive Software“. In: *SAE Technical Paper*. SAE International, Apr. 2011. DOI: 10.4271/2011-01-0449. URL: http://dx.doi.org/10.4271/2011-01-0449.

[81] W. Zimmermann und R. Schmidgall. „AUTOSAR-Softwarearchitektur für Kfz-Systeme“. German. In: *Bussysteme in der Fahrzeugtechnik*. ATZ/MTZ-Fachbuch. Springer Fachmedien Wiesbaden, 2014, Seiten 367–413. ISBN: 978-3-658-02418-5. DOI: 10.1007/978-3-658-02419-2_8. URL: http://dx.doi.org/10.1007/978-3-658-02419-2_8.

[82] W. Zimmermann und R. Schmidgall. „Software-Standards: OSEK und HIS“. German. In: *Bussysteme in der Fahrzeugtechnik*. ATZ/MTZ-Fachbuch. Springer Fachmedien Wiesbaden, 2014, Seiten 331–365. ISBN: 978-3-658-02418-5. DOI: 10.1007/978-3-658-02419-2_7. URL: http://dx.doi.org/10.1007/978-3-658-02419-2_7.

[83] T. Zurawka und J. Schaeuffele. „Method for checking the safety and reliability of a software-based electronic system“. German. DE112004001617D2, EP1639464A2, US20070016840, WO2005003972A2. 13. Jan. 2007.