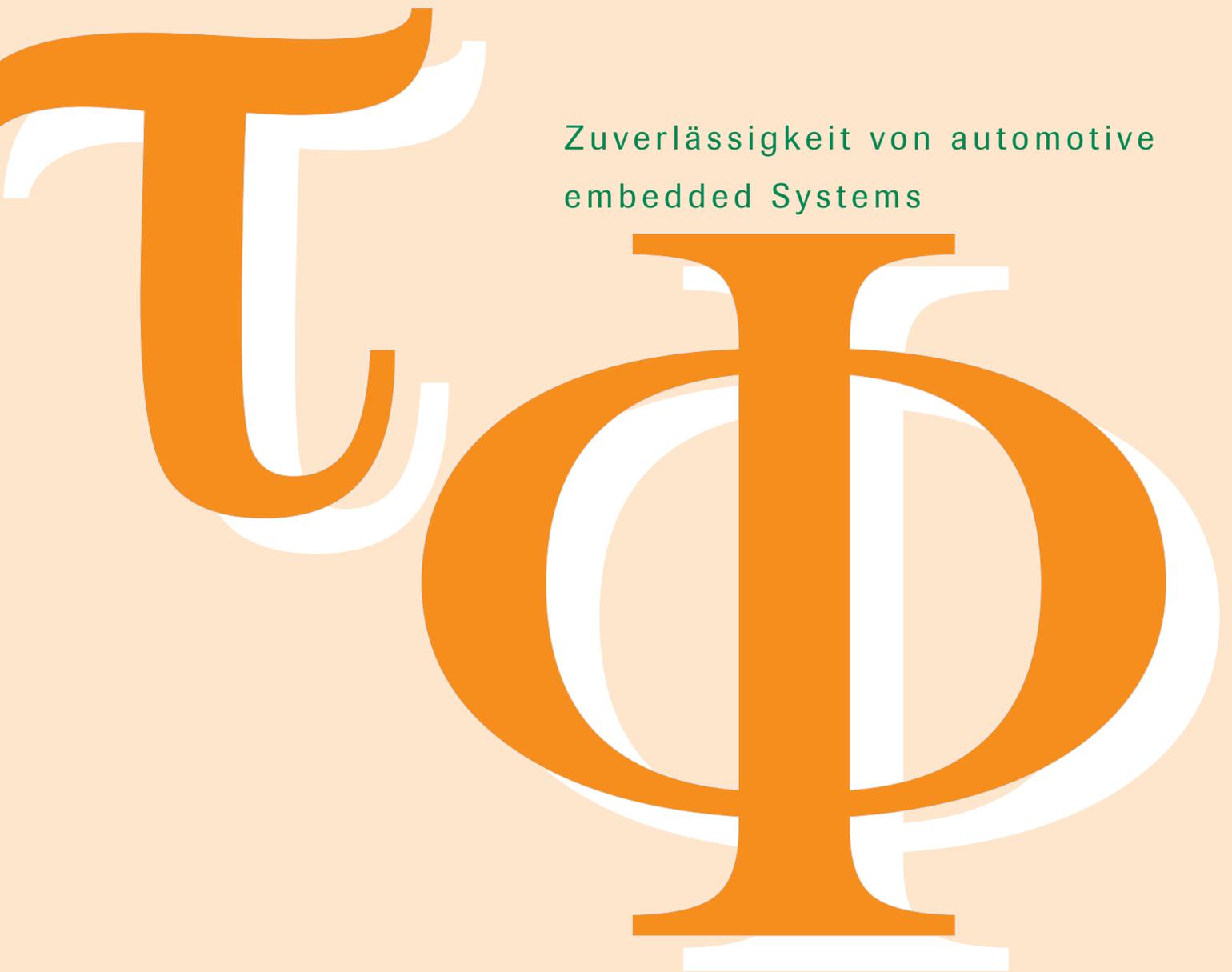


FAT 231

Zuverlässigkeit von automotive
embedded Systems



Zuverlässigkeit von automotive embedded Systems

Auftraggeber:

Forschungsvereinigung Automobiltechnik e.V. (FAT)

Auftragnehmer:

RWTH Aachen University

Lehrstuhl Informatik 11

Software für Eingebettete Systeme

Prof. Dr.-Ing. Stefan Kowalewski

Verfasser:

Eva Beckschulze

David Boymanns

Ramona Dülks

Thomas Gatterdam

Prof. Dr.-Ing. Stefan Kowalewski

Martin Lang

Dr.-Ing. Falk Salewski

Thomas Siegbert

Inhalte

1	Vorbemerkungen.....	4
1.1	ASIL-Einstufungen	4
1.2	Begrifflichkeit Zuverlässigkeit/Verfügbarkeit	4
2	Kurzdarstellung.....	5
3	Einleitung	6
4	Aufbau der Studie	6
4.1	Fragestellungen	6
4.2	Bedeutung der funktionalen Sicherheit	6
4.3	Vorgehensweise	6
4.4	Auswahl der Architekturen	8
4.5	Anwendungsbeispiel	8
4.6	Randbedingungen.....	9
5	Durchführung der Studie	10
5.1	Konzepterstellung.....	10
5.1.1	Funktionales Konzept.....	10
5.1.2	Funktionales Sicherheitskonzept.....	11
5.1.3	Technisches Sicherheitskonzept.....	19
5.1.4	Fehlermetriken und Diagnoseabdeckung.....	26
5.2	Umsetzung des Konzeptes	26
5.3	Bewertung des Entwicklungsaufwandes.....	27
5.4	Bewertung des Einflusses durch ASIL.....	27
5.5	Untersuchung der Zuverlässigkeit.....	27
5.6	Untersuchung der Testbarkeit	28
5.7	Betrachtung der Änderbarkeit	28
6	Ergebnisse der Studie.....	29
6.1	Aufwand der SW Implementierung	29
6.1.1	Dokumentation des Projektfortschritts und von Problemen	29
6.1.2	Bewertung des Aufwandes durch Entwickler	29
6.1.3	Fazit Entwicklungsaufwand	31
6.2	Zuverlässigkeit der Ansätze	31
6.2.1	Analyse der Architekturen hinsichtlich der Zuverlässigkeit	31
6.2.2	Erhöhung der Zuverlässigkeit der Dual-Core-Architektur.....	32
6.2.3	Erhöhung der Zuverlässigkeit der MCU-FPGA-Architektur.....	32
6.2.4	Allgemeine Ansätze zur Erhöhung der Zuverlässigkeit.....	34
6.2.5	Abwägung Sicherheit ⇔ Zuverlässigkeit	35
6.2.6	Fazit Zuverlässigkeit	35
6.3	ASIL Überlegungen	35
6.3.1	Auswirkungen eines ASIL-Wechsels auf die Architektur	36
6.3.2	Entwicklung ASIL B / ASIL C	36
6.3.3	Fazit ASIL Überlegungen	37
6.4	Einflüsse auf Testbarkeit.....	38
6.4.1	Durchführung Akzeptanztests	38
6.4.2	Durchführung Fehlerinjektionstests	38
6.4.3	Bewertung der Testbarkeit	39
6.4.4	Fazit Testbarkeit.....	40

6.5	Einflüsse auf Änderbarkeit.....	40
6.5.1	Fazit Änderbarkeit.....	41
6.6	Vergleich mit alternativen Sicherheitsansätzen	42
6.6.1	Funktionsüberwachung	42
6.6.2	Bauteilüberwachung.....	43
6.6.3	Funktionsüberwachung vs. Bauteilüberwachung.....	43
6.6.4	Vergleich verschiedener Architekturen.....	45
6.6.5	Fazit Vergleich	48
6.7	Validität der Ergebnisse	49
7	Zusammenfassung.....	50
8	Ausblick.....	51
	Referenzen.....	52
	Anhang A: Fehlerbäume zur Verifikation der funktionalen Sicherheitsanforderungen	53
	Anhang B: Bestimmung der Fehlermetriken und der Diagnoseabdeckung.....	57
	Bestimmung der SPFM und LFM.....	57
	SPFM und LFM für Dual-Core-Mikrocontroller Architektur.....	58
	SPFM und LFM für Mikrocontroller-FPGA Architektur	61
	SPFM für Single-Core-Mikrocontroller Architektur gemäß ASIL B	63
	Bestimmung der Diagnoseabdeckung (DC).....	64
	DC für Dual-Core-Mikrocontroller Architektur	64
	DC für Mikrocontroller-FPGA Architektur	65
	DC für Single-Core-Mikrocontroller Architektur	65
	Anhang C: Dokumentation der Arbeitsschritte	66
	Anhang D: Testfälle	72
	Auflistung der Akzeptanztests	72
	Auflistung der Fehlerinjektionstests.....	77
	Anhang E: Aufbau der Test- und Simulationsumgebung	79
	Anforderungen	79
	Aufbau.....	79
	Realisierung.....	80
	Fehlerinjektions- und Eingabemöglichkeiten.....	81
	Anhang F: Umfrage zu Trends und Herausforderungen in der Automobilelektronik... 84	84
1.	Hintergrund	84
2.	Ergebnisse	84
3.	Verwendeter Fragebogen zur Trendermittlung in der Automobilelektronik.....	86
0.	Personenbezogene Daten.....	86
1.	Allgemeine Fragen	86
2.	Bewertung von Trends	87
3.	Weitere eigene Trenderwartungen	88
4.	Probleme aus der Praxis	89
5.	Abschließende Anmerkungen	89
	Abkürzungsverzeichnis.....	90

1 Vorbemerkungen

1.1 ASIL-Einstufungen

In dieser Studie werden Annahmen über die Einordnung von Anwendungen in ASILs nach der ISO WD 26262 vorgenommen. Diese Annahmen sind notwendig, um die vorab aufgestellten Forschungsfragen untersuchen zu können, zum Beispiel welchen Einfluss die Auswahl einer bestimmten Hardwareplattform auf den Aufwand zur Entwicklung einer ASIL C unterworfenen Funktion hat. Aus diesem Grund stellen die angenommenen ASIL-Einstufungen Arbeitshypothesen zur Beantwortung spezifischer Fragen dar und sind auf keinen Fall als Empfehlungen der Autoren für die ASIL-Festlegung der hier betrachteten Anwendungen zu verstehen. In dem konkreten Fall der Cabrio-Verdeck-Steuerung ist es sogar so, dass realistische Bewertungen aus der Praxis zu einem geringeren als dem hier nur hypothetisch angesetzten ASIL C geführt haben.

1.2 Begrifflichkeit Zuverlässigkeit/Verfügbarkeit

Im Rahmen dieses Berichts wird unter *Zuverlässigkeit* allgemein die Anforderung an ein System bzw. die Eigenschaft eines Systems verstanden, die geforderte oder erwartete Funktion unter gegebenen Bedingungen zu erfüllen. In diesem allgemeinen Sinne ist Zuverlässigkeit ein Oberbegriff, der zum Beispiel nicht zwischen reparierbaren und nicht-reparierbaren Systemen unterscheidet und daher die Anforderung der *Verfügbarkeit* (als mittlerer Anteil der Zeit, in der sich das System in einem „funktionierenden“ Zustand befindet, an der gesamten Betriebszeit) subsumiert. Dies sollte zu keinen Missverständnissen führen, da wir bei den Untersuchungen genau unterscheiden, ob die Zuverlässigkeit als Anforderung an das Kernsystem, das Sicherheitssystem oder Gesamtsystem betrachtet wird. Bei der Betrachtung des Gesamtsystems erfüllt zum Beispiel der im Gefahrenfall spezifizierte Übergang in einen sicheren Zustand eine Zuverlässigkeitsanforderung. Aus der Sicht auf das Kernsystem wird dadurch aber dessen Zuverlässigkeit beeinträchtigt. Eine Unterscheidung dieser beiden Sachverhalte ist also auch ohne Verwendung des Begriffs Verfügbarkeit für das Gesamtsystem möglich.

2 Kurzdarstellung

Im Rahmen dieses Projektes wurden neuartige Hardware-Architekturen für Ihre Eignung in Automobilanwendungen untersucht. Der Fokus lag dabei auf dem Aufbau einzelner Steuergeräte und basierte auf einer Untersuchung zu Trends und Herausforderungen in der Automobil-elektronik in den nächsten 15 Jahren.

Erste Untersuchungen verdeutlichten die zunehmende Bedeutung sicherheitskritischer Funktionen im Automobil und die damit einhergehenden Sicherheitsanforderungen (vor allem durch die ISO 26262). Bei der Implementierung solcher Funktionen sind vielfältige durch die Norm vorgegebene Anforderungen zu erfüllen. Obwohl es im Arbeitskreis vorrangig um die Untersuchung der Zuverlässigkeit geht, wurden in diesem Projekt auch Anforderungen gemäß der ISO WD 26262 berücksichtigt. Dies beinhaltet auch die Berücksichtigung von transienten und permanenten Hardwarefehlern, die in sicherheitskritischen Systemen zu berücksichtigen sind.

Vor diesem Hintergrund wurden in diesem Projekt zwei neuartige Hardware-Architekturen (Dual-Core Mikrocontroller und eine Kombination aus Mikrocontroller und FPGA) untersucht. Ausgangspunkt der Wahl dieser Architekturen waren viel versprechende Möglichkeiten, die in diesen Systemen gegebenen Redundanzmechanismen zur Fehlererkennung und Fehlertoleranz zu nutzen. Die Untersuchung geschah anhand von zwei exemplarischen Implementierungen eines Anwendungsbeispiels, das vom Arbeitskreis vorgegeben wurde. Es konnte gezeigt werden, dass mit beiden Ansätzen eine Implementierung des Anwendungsbeispiels gemäß der abgeleiteten Sicherheitsanforderungen (Arbeitshypothese: ASIL C) möglich ist. Die zusätzlich untersuchten Aspekte Implementierungsaufwand, Zuverlässigkeit, Änderbarkeit und Testbarkeit ergaben, von vereinzelt Startschwierigkeiten bezüglich des Implementierungsaufwandes abgesehen, keine nennenswerten Unterschiede zwischen den beiden Architekturen.

Im Rahmen des Projektes konnten jedoch zahlreiche Unterschiede zwischen dem für beide Architekturen gewählten Sicherheitskonzept (Überwachung sicherheitskritischer Funktionen) und einem alternativen Ansatz identifiziert werden. Bei dem alternativen Ansatz handelt es sich um eine von der Anwendung weitgehend unabhängige Überwachung des Bauteils, welches die sicherheitskritische Funktion ausführt. In der Literatur werden verschiedenartige Hardware-Architekturen vorgeschlagen, die diesen Ansatz verfolgen. Neben allgemeinen Ansätzen basieren viele dieser Ansätze auf der Verwendung von Dual-Core Mikrocontrollern. Während im gewählten Ansatz der Funktionsüberwachung Standard-Bauteile Verwendung finden können, erfordert der alternative Ansatz speziell angefertigte Hardware. Weiterhin ist dieser Ansatz zunächst auf die Behandlung von Hardware-Fehlern beschränkt, während im gewählten Ansatz auch sicherheitsrelevante Software Fehler behandelt werden können. Auf der anderen Seite ist der alternative Ansatz der Bauteilüberwachung unabhängig von der Anwendung und dem gewählten Sicherheitskonzept, was zu den Stärken dieses Ansatzes führt. Für das betrachtete Anwendungsbeispiel scheint der gewählte Ansatz der Funktionsüberwachung aus Gründen der preiswerteren Hardware und der einfach zu realisierenden Sicherheitsfunktion, die sowohl Hardware als auch sicherheitsrelevante Softwarefehler tolerieren kann, am besten geeignet zu sein.

3 Einleitung

Im Rahmen dieses Projektes wurden verschiedene neuartige Hardware-Architekturen für Ihre Eignung in Automobilanwendungen untersucht. Der Fokus lag dabei auf dem Aufbau einzelner Steuergeräte, während Steuergeräte-übergreifende Funktionalitäten nur am Rande betrachtet wurden. Die im Projekt durchgeführten Untersuchungen stützen sich auf eine Studie, die auf der exemplarischen Implementierung eines Anwendungsbeispiels auf zwei verschiedenen Hardware-Architekturen beruht. Diese Studie, welche im ersten Projektjahr geplant und vorbereitet wurde, wird in den Kapiteln 3 und 4 beschrieben. Aus der Studie abgeleitete Ergebnisse und die Ergebnisse weiterführender Untersuchungen werden dann in Kapitel 5 beschrieben. Eine Zusammenfassung ist schließlich in Kapitel 6 zu finden.

4 Aufbau der Studie

4.1 Fragestellungen

Die im Rahmen dieser Studie zu beantwortenden Fragen wurden gemeinsam mit den Arbeitskreismitgliedern erarbeitet. Dazu wurde zunächst eine Umfrage zu Trends und Herausforderungen in der Automobilelektronik in den nächsten 15 Jahren durchgeführt (Ergebnisse dieser Umfrage sind im Anhang F zu finden). Basierend auf dieser Umfrage wurden im Arbeitskreis folgende Fragen erarbeitet:

1. Wie hoch ist der Aufwand der SW Implementierung neuartiger Hardware-Architekturen einzuschätzen?
2. Lohnt es sich, in Bezug auf Aufwand, Produktkosten und Zuverlässigkeit, grundsätzlich nach ASIL C zu entwickeln? Welchen Einfluss haben dabei Sicherheitsmechanismen auf die Zuverlässigkeit?
3. Wie hoch ist die Zuverlässigkeit neuartiger Hardware-Architekturen einzuschätzen?
4. Welchen Einfluss haben neuartiger Hardware-Architekturen auf folgende Aspekte?
 - a. Änderbarkeit
 - b. Testbarkeit

4.2 Bedeutung der funktionalen Sicherheit

Nach einer Abschätzung der Arbeitskreismitglieder werden einige der im Auto realisierten Funktionen nach der Sicherheitsnorm ISO WD 26262 nach ASIL C einzustufen sein. Bei der Implementierung solcher Funktionen sind vielfältige durch die Norm vorgegebenen Anforderungen zu erfüllen. Obwohl es im Arbeitskreis vorrangig um die Untersuchung der Zuverlässigkeit geht, werden daher in dieser Studie auch Anforderungen gemäß der ISO WD 26262 berücksichtigt. Für die Untersuchungen wurde im Weiteren hypothetisch davon ausgegangen, dass die zu untersuchende Anwendung nach ASIL C einzustufen ist (siehe Vorbemerkung, Seite 4).

4.3 Vorgehensweise

Um obige Fragestellungen zu beantworten, wurde eine exemplarische Implementierung eines Anwendungsbeispiels im Rahmen einer Studie durchgeführt. Da absolute Aussagen schwierig zu formulieren sind, wurden verschiedene Architekturen miteinander verglichen. Vor diesem Hintergrund wurden die obigen Fragestellungen angepasst:

1. Wie hoch ist der Aufwand der SW Implementierung der beiden Ansätze im Vergleich?

2. Lohnt es sich (Aufwand, Produktkosten, Zuverlässigkeit), grundsätzlich nach ASIL C zu entwickeln? Welchen Einfluss haben dabei Sicherheitsmechanismen auf die Zuverlässigkeit?
3. Wie hoch ist die Zuverlässigkeit (Basis ASIL C) der beiden Ansätze im Vergleich?
4. Sind Einflüsse der Architekturen auf folgende Aspekte zu beobachten?
 - a. Änderbarkeit
 - b. Testbarkeit
 - c. Produktlinienansätze

Die Implementierung des Anwendungsbeispiels erfolgte im Rahmen von zwei Diplomarbeiten. Die beiden Studenten wurden jeweils durch eine studentische Hilfskraft bei Implementierungsaufgaben unterstützt.

Zur Beantwortung der obigen Fragen wurde die Studie in fünf Arbeitsschritte aufgeteilt (Siehe Abbildung 4-1: Zeitplan der Studie (* = Implementierung nur nach positiver Aufwand / Nutzen Abschätzung). Zunächst wurde das Anwendungsbeispiel auf den verschiedenen Architekturen implementiert. Dieser Arbeitsschritt diente in erster Linie der Beantwortung der ersten Frage. Im nächsten Arbeitsschritt wurde der Einfluss eines Wechsels des ASIL Levels auf verschiedene Aspekte (Aufwand, Produktkosten, Zuverlässigkeit) der jeweiligen Implementierung untersucht. Ziel war es, zu untersuchen, ob eine grundsätzliche Entwicklung nach ASIL C gegenüber einer gemischten Entwicklung ASIL C/ASIL B Vorteile bietet, oder ob sie einen unverhältnismäßigen Mehraufwand darstellt (Frage 2). Die Zuverlässigkeit der Systeme wurde im Folgenden dritten Arbeitsschritt untersucht. In diesem Zusammenhang waren Schwachstellen der Architektur bezüglich Zuverlässigkeit zu identifizieren und mögliche Ansätze zur Verbesserung der Zuverlässigkeit zu erarbeiten und zu bewerten. Im vierten Arbeitsschritt war zunächst die Änderbarkeit der untersuchten Architekturen zu untersuchen. Im Verlauf der Studie wurde dieser Arbeitsschritt jedoch dazu genutzt, verstärkt die Testbarkeit der Systeme zu betrachten. Der letzte Arbeitsschritt beinhaltete, unter Verwendung einer parallel entwickelten Simulations- und Testumgebung, das Testen der erstellten Systeme, um weitere Informationen zur Zuverlässigkeit dieser Systeme zu erhalten. Dies umfasste die Untersuchung einer vorgegebenen Menge von Fehlern (Fehlerinjektion in HW-Komponenten) bei gegebenen Sicherheitsmechanismen.

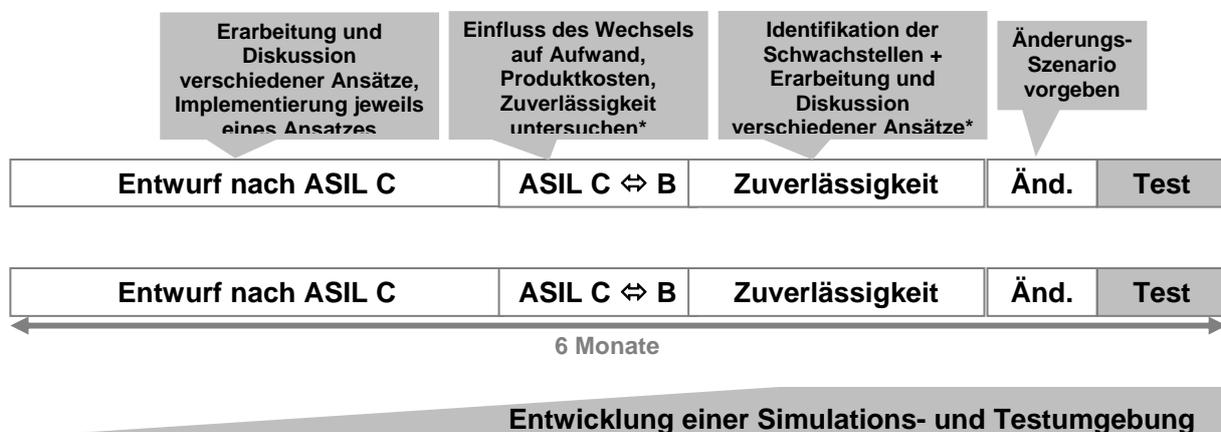


Abbildung 4-1: Zeitplan der Studie (* = Implementierung nur nach positiver Aufwand / Nutzen Abschätzung)

Weiterhin werden im Rahmen der Studie folgende Punkte dokumentiert:

1. Betreuer protokolliert wöchentlich den Stand der Entwicklung, Probleme und gefundene Fehler. Probleme und Fehler werden dahingehend bewertet, ob es sich um Initialaufwand (Aufwand, der nur bei erstmaliger Verwendung einer bestimmten Hardware und/oder Technik entsteht) oder Folgeaufwand handelt.
2. Der Aufwand für einzelne Arbeitsschritte wird von den Studenten bewertet und anschließend mit dem Betreuer diskutiert.

4.4 Auswahl der Architekturen

Folgende Architekturen wurden für die Untersuchung vorgeschlagen:

1. Single-Core MCU + Standard-Watchdog*
2. Single-Core MCU + externer FPGA
3. Single Core MCU with built in HW fault handling + Standard-Watchdog*
4. Single-Core MCU mit Co-Prozessor + Standard-Watchdog*
5. Standard Dual-Core MCU + Standard-Watchdog*
6. Dual-Core MCU in Lockstep Architektur + Standard-Watchdog*
7. FPGA + Standard-Watchdog*

Basierend auf den Ergebnissen der Umfrage und Diskussionen im Arbeitskreis wurden die Architekturen 2 und 5 für die Studie ausgewählt.

* Standard-Watchdog: externer ASIC, der einen Neustart des Systems und eine Deaktivierung sicherheitskritischer Aktuatoren durchführt sobald er ungültige Signale vom System erhält (typisches gültiges Signal: Impulse zu definierten Zeitpunkten). Im Anwendungsbeispiel wurde ein sog. time-windowed Watchdog eingesetzt, der sowohl eine obere als auch eine untere Grenze für eine gültige Reaktionszeit vorgibt.

4.5 Anwendungsbeispiel

Das Anwendungsbeispiel sollte folgenden Anforderungen genügen: Es sollte

- eine hinreichend große Komplexität haben, um belastbare Ergebnisse zu ermöglichen,
- eine Komplexität vorweisen, die es noch ermöglicht, die Anwendung im Rahmen der Studie mit Studenten zu implementieren,
- Sicherheitsanforderungen beinhalten, um der Bedeutung der funktionalen Sicherheit gerecht zu werden.

Weiterhin wäre die Verwendung einer existierenden, ggf. modifizierten Spezifikation aus Gründen der Zeitersparnis vorteilhaft.

Auf Grund von Firmengeheimnissen war eine direkte Verwendung einer existierenden Spezifikation nicht möglich. Dennoch konnte in Zusammenarbeit mit Arbeitskreismitgliedern auf Basis von einzelnen Dokumenten und einer gegebenen Funktionssimulation eine Spezifikation für eine Cabrio-Verdecksteuerung erstellt werden (siehe auch [Spezifikation]).

Die Funktion der Cabrio-Verdecksteuerung beinhaltet das Einlesen eines Bedienelements, das Einlesen zahlreicher Sensoren zur Bestimmung des Dachzustands und die Ansteuerung der Hauptaktuatorik (Hydraulikpumpe und Hydraulikventile) sowie von zwei weiteren Elektromotoren für Verriegelungsaufgaben. Weiterhin ist eine Kommunikation mit weiteren Steuergeräten über den CAN-Bus erforderlich. Eine Schnittstellenbeschreibung des Systems ist in Abbildung 4-2 Schnittstellenbeschreibung der Cabrio-Verdecksteuerung in Form eines Kontextdiagramms zu finden.

Die erstellte Spezifikation wurde den Arbeitskreismitgliedern präsentiert und Änderungs- und Ergänzungsvorschläge (Hinzufügen von Akzeptanzkriterien) umgesetzt. Im Rahmen des Hinzufügens der Akzeptanzkriterien wurden vereinzelte Unzulänglichkeiten der Spezifikation identifiziert und entsprechend korrigiert.

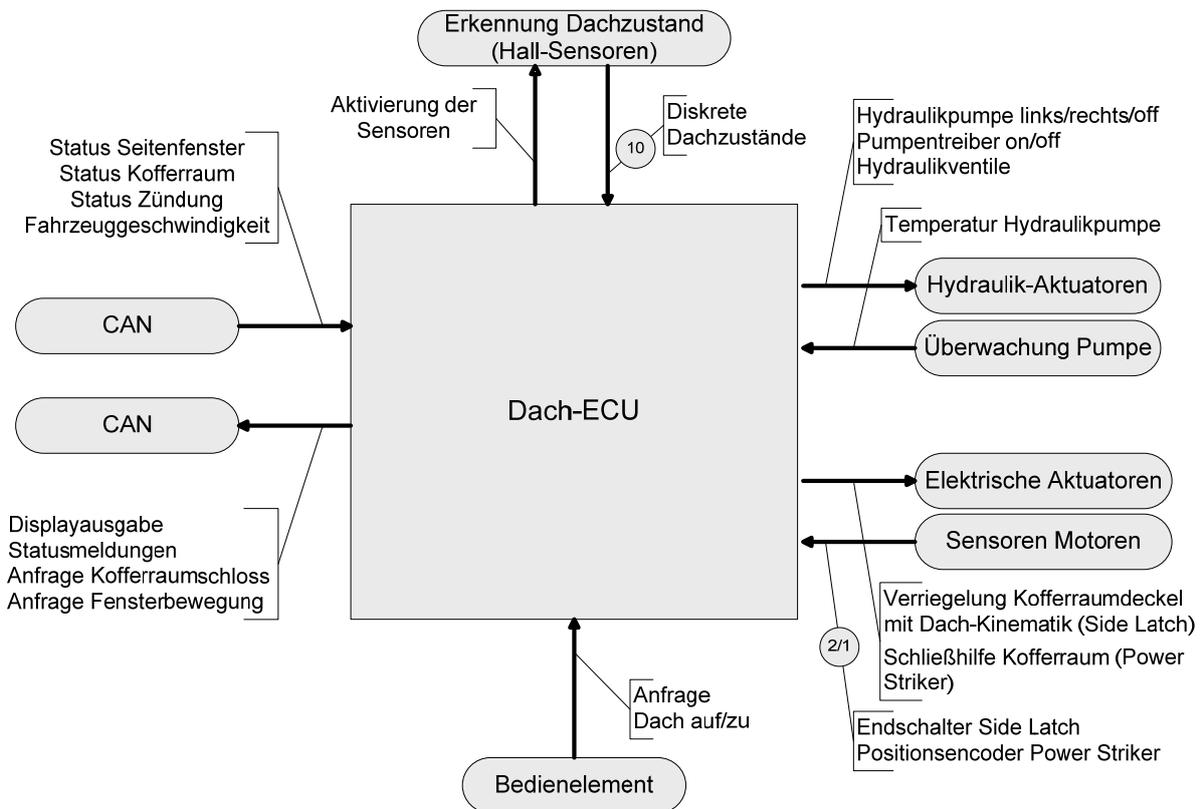


Abbildung 4-2 Schnittstellenbeschreibung der Cabrio-Verdecksteuerung

Für weitere Details zum Anwendungsbeispiel sei auf [Spezifikation] verwiesen.

4.6 Randbedingungen

Für die Umsetzung des Anwendungsbeispiels wurden einige Randbedingungen vorgegeben, die im Folgenden kurz vorgestellt werden.

1. Verwendung eines OSEK Betriebssystemes für die Umsetzung des Anwendungsbeispiels.
2. Anlehnung der Software-Architektur an die AUTOSAR-Schichten. Dies sollte durch eine Schichten-Architektur realisiert werden, in der die Anwendung nur über Tasks der Basis-Software auf die Hardware zugreift.

5 Durchführung der Studie

Im Folgenden wird die Durchführung der Studie beschrieben. Dies umfasst die Konzepterstellung (funktionales Konzept, funktionales Sicherheitskonzept, technisches Sicherheitskonzept), die Umsetzung des Konzeptes, die Bewertung des Einflusses durch ASIL, die Untersuchung der Zuverlässigkeit der erstellten Implementierungen, die Untersuchung der Testbarkeit beider Systeme und schließlich Betrachtungen zur Änderbarkeit.

Die in der Studie durchgeführten Sicherheitsbetrachtungen beziehen sich auf die zu Projektbeginn verfügbare Version der Sicherheitsnorm ISO WD 26262 (Baseline 7). Lediglich bei der Bestimmung der Fehlermetriken und Diagnoseabdeckung in Anhang B wurde die neuere Baseline 9 genutzt.

5.1 Konzepterstellung

5.1.1 Funktionales Konzept

Die funktionalen Anforderungen ergeben sich aus der Systemspezifikation zum Anwendungsbeispiel [Spezifikation]. In beiden Architekturen erfolgte die Implementierung des Anwendungsbeispiels auf einer CPU auf Basis eines OSEK Betriebssystems. Stellvertretend für beide Architekturen ist das funktionale Konzept der Dual-Core-Architektur in Abbildung 5-1: Aktivitätsdiagramm Dachsteuerung dargestellt. Herausforderungen sind in der Vielzahl möglicher Kombinationen von Eingangsgrößen und Systemzuständen zu sehen. Für weitere Details zum funktionalen Konzept sei auf die Diplomarbeiten [Siegbert] und [Boymanns] verwiesen.

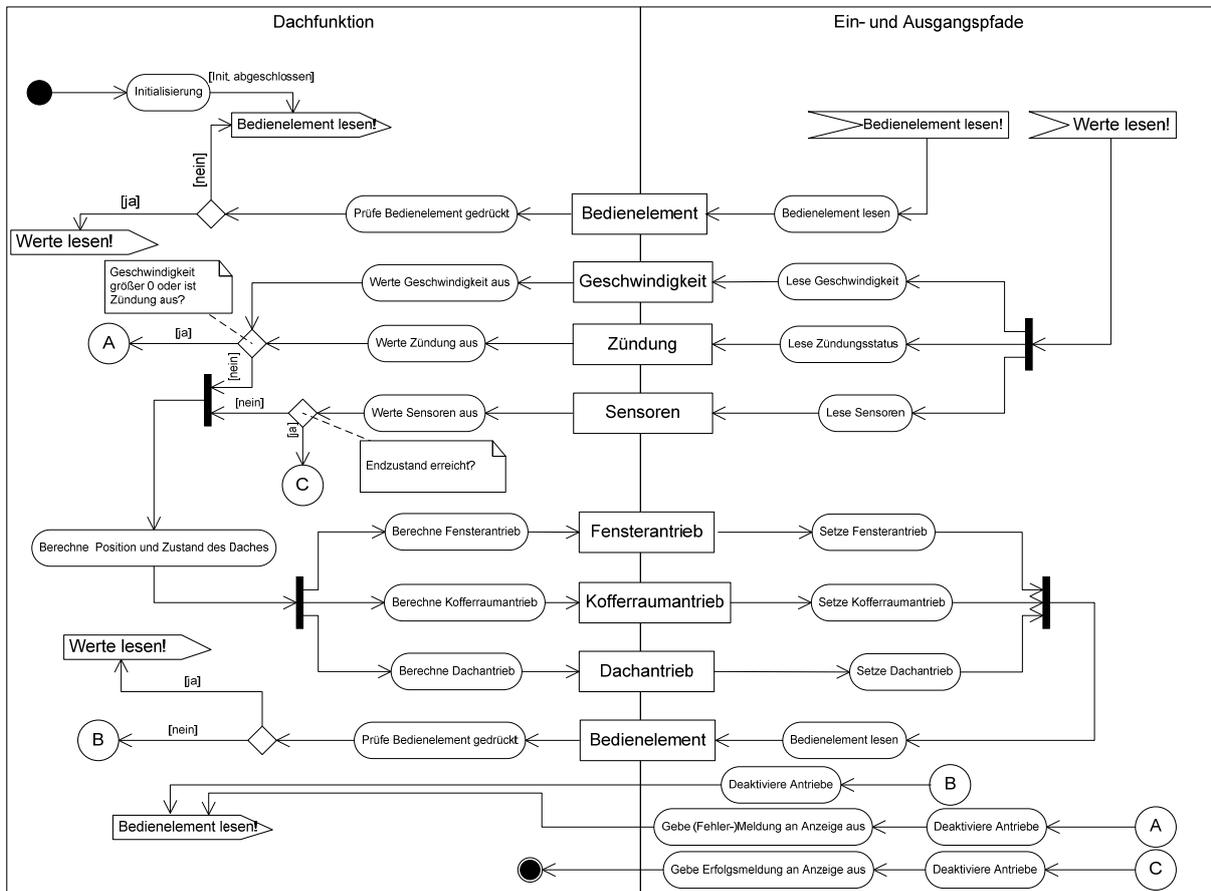


Abbildung 5-1: Aktivitätsdiagramm Dachsteuerung

5.1.2 Funktionales Sicherheitskonzept

In diesem Abschnitt wird auf Basis von Gefahrenanalysen ein funktionales Sicherheitskonzept gemäß ISO26262-3 erstellt.

5.1.2.1 Betrachtete Hazards

Folgende zwei Hazards wurden identifiziert (kein Anspruch auf Vollständigkeit):

1. Gefährdung durch ein bei der Fahrt abreißendes Dach
2. Einklemmen

Beim Hazard Einklemmen wird nur die Bewegung des eigentlichen Daches betrachtet und nicht Gefahren, die durch die Bewegung der Fensterscheiben während des Öffnen -u. Schließvorgangs des Daches entstehen können (wird durch Fenstersteuergerät behandelt).

5.1.2.2 Sicherheitsziele (*Safety Goals*)

Die Bestimmung der Sicherheitsziele (*Safety Goals*) bezieht sich auf ISO26262-3, 6.4.7. Ein Sicherheitsziel ist definiert als die Negierung des Top-Schadensereignisses der Sicherheitsanalyse (ISO26262-1: *Negation of safety goal is top event of safety analysis*). Damit lassen sich aus den oben identifizierten Hazards folgende Sicherheitsziele ableiten:

Sicherheitsziel für Hazard A:

Dach wird ausschließlich bewegt, wenn das Fahrzeug nicht fährt*.

Safe states: Dach ist vollständig auf oder vollständig zu

ASIL C (Arbeitshypothese)

Sicherheitsziel für Hazard B:

Dach wird ausschließlich bewegt, wenn dies vom Benutzer gewünscht ist (da dieser die Verantwortung trägt).

Safe state: Dach bewegt sich nicht

ASIL C (Arbeitshypothese)

*alternativ: Das Dach reißt während der Fahrt durch das Öffnen des Daches nicht ab.

→ Lässt die Möglichkeit offen, dass Dach mechanisch zu verstärken um ein Abreißen zu verhindern; würde zu einem anderen Top Event und einem weiteren Zweig im FTA führen (Dachmechanik hält Belastung stand ⇔ hält Belastung nicht stand). Es wird hier jedoch davon ausgegangen, dass das Dach den Belastungen nicht standhalten kann.

Annahmen: Ein nicht ausreichend verschlossenes Dach (Abreißen bei hoher Geschwindigkeit möglich) ist für den Fahrer erkennbar. Dieser trägt dann die Verantwortung in diesem Zustand nicht loszufahren.

5.1.2.3 Ableitung der funktionalen Sicherheitsanforderungen

In diesem Abschnitt erfolgt die Ableitung der funktionalen Sicherheitsanforderungen aus den zuvor bestimmten Sicherheitszielen (*Safety Goals*) und deren Zuordnung zu den einzelnen Systemkomponenten. (ISO26262-3, 7.1)

Das funktionale Sicherheitskonzept spezifiziert dabei in Form von funktionalen Sicherheitsanforderungen die Grundaufgaben der Mechanismen und Maßnahmen, durch welche die Sicherheitsziele erfüllt werden sollen. Dieses Konzept beschreibt, **was** von jeder Komponente des Systems getan werden muss, um die Sicherheitsziele zu erfüllen. Das später zu erstellende

technische Sicherheitskonzept beschreibt dann, wie diese funktionalen Sicherheitsanforderungen technisch implementiert werden. (ISO26262-3, 7.2)

Die funktionalen Sicherheitsanforderungen müssen in einer geeigneten Weise von den Sicherheitszielen und *Safe States* abgeleitet werden. Dies wird typischerweise durch eine Sicherheitsanalyse, wie beispielsweise einer FMEA, FTA, etc. unterstützt. (ISO26262-3, 7.4.2.1). Die folgenden Informationen müssen dabei berücksichtigt werden:

- Das Architekturkonzept
- Die Analyse der Operationsmodi bzw. Systemzustände
- Die *fault tolerant time span* und die *emergency operation time*

Für das in dieser Studie untersuchte Anwendungsbeispiel ist das Architekturkonzept in Abbildung 5-2: Architekturkonzept zu berücksichtigen:



Abbildung 5-2: Architekturkonzept

Weiterhin sind die folgenden Operationsmodi bzw. Systemzustände vorhanden:

- Öffnen, Schließen, Stand-by, Initialisierung
- geöffnet, geschlossen, unvollständig geöffnet/geschlossen

Für die fehlertoleranten Zeitspannen (*fault tolerant time span*) und die Notlaufzeit (*emergency operation time*) ergeben sich aus der Analyse der Sicherheitsziele folgende Werte:

- *Fault tolerant time span*
 - Ungewolltes Öffnen während der Fahrt: **1s** (im vollständig geöffneten bzw. vollständig geschlossenen Zustand des Daches, was eine Voraussetzung für das Aufnehmen einer Fahrzeugbewegung darstellt, kann bis zu eine Sekunde lang eine ungewollte Ansteuerung des Daches toleriert werden, da die Mechanik des Daches erst verzögert auf diese Ansteuerung reagiert.)
 - Einklemmen: **100ms** (Der Fahrer trägt durch die Betätigung des Bedienelementes die Verantwortung dafür, dass durch die Bewegung des Daches kein Schaden (z.B. Einklemmen von Personen) entsteht. In diesem Zusammenhang wird davon ausgegangen, dass eine Zeit von 100ms zwischen einer Aktion des Fahrers und dem Anhalten des Daches toleriert werden kann.)
- *Emergency operation time*
 - In beiden Fällen: **0s** (es wird keine Notlaufzeit benötigt um die Sicherheitsziele zu erreichen)

Zur Ableitung der funktionalen Sicherheitsanforderungen wurden Fehlerbaumanalysen für beide Sicherheitsziele durchgeführt. Dazu wurden die Sicherheitsziele jeweils negiert als Top-Schadensereignis im Fehlerbaum betrachtet. Die Ergebnisse dieser Analyse befinden sich in den folgenden beiden Unterkapiteln.

5.1.2.4 Funktionale Sicherheitsanforderungen für Sicherheitsziel A

Zur Herleitung der funktionalen Sicherheitsanforderungen für Sicherheitsziel A wird eine Fehlerbaumanalyse durchgeführt.

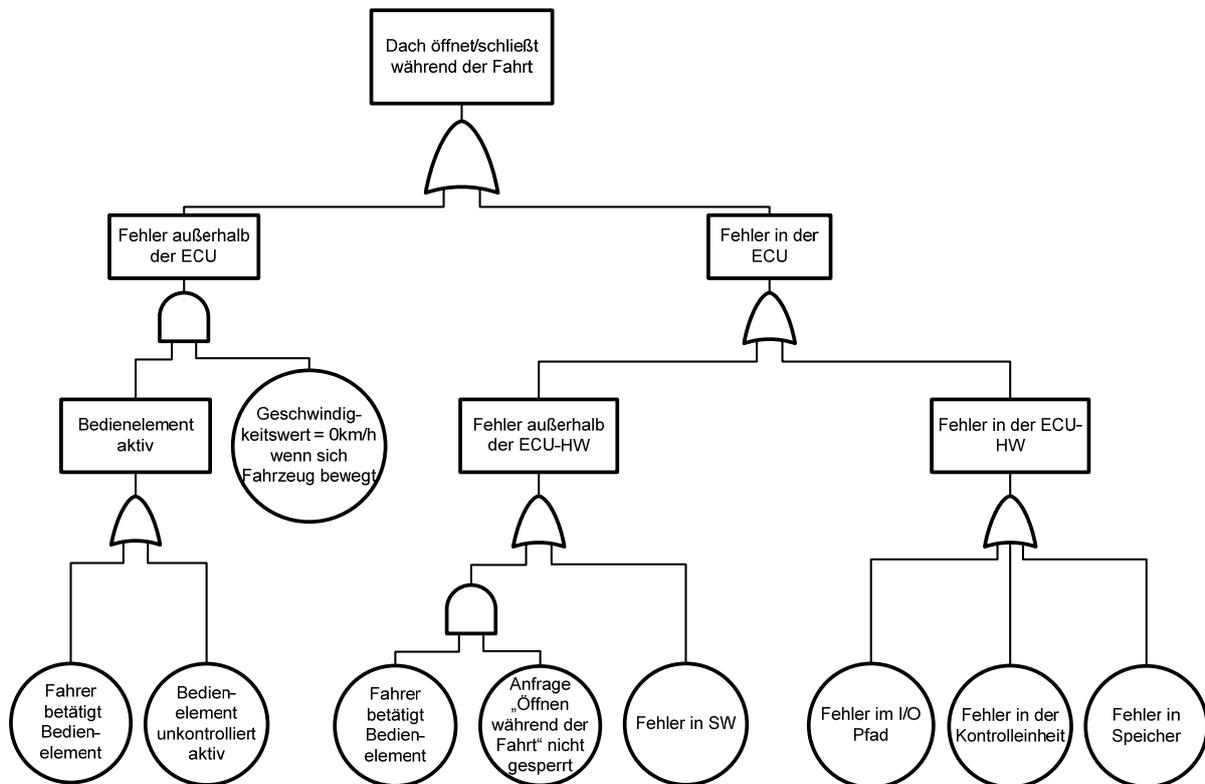


Abbildung 5-3: Fehlerbaumanalyse für Sicherheitsziel A

Aus dem für Sicherheitsziel A aufgestellten Fehlerbaum (Abbildung 5-3: Fehlerbaumanalyse für Sicherheitsziel A) lassen sich folgende funktionalen Sicherheitsanforderungen ableiten:

1. Der Fehler *Geschwindigkeitswert fälschlicherweise = 0km/h* muss erkannt und behandelt werden.
2. Der Fehler *Bedienelement unkontrolliert aktiv* und eine fehlerhafte Bedienung (Anforderung Dachbewegung durch Fahrer während der Fahrt) müssen erkannt und behandelt werden.
3. Fehler im I/O Pfad, in der Kontrolleinheit und in Speichern, die das Sicherheitsziel verletzen können, müssen erkannt und behandelt werden.
4. SW-Fehler, die das Sicherheitsziel verletzen können, müssen erkannt und behandelt werden.

Weiterhin werden gemäß ISO26262-3, 7.4.2.5 folgende Attribute für alle Anforderungen festgelegt:

- ASIL C
- *Operating mode*: Stand-by
- *Fault tolerant time span*: 1s (Trägheit der Mechanik)
- *Safe states*: komplett geöffnetes Dach, komplett geschlossenes Dach
- *Keeping the safe state*: Deaktivierung des Dachantriebs sicherstellen

5.1.2.5 Funktionale Sicherheitsanforderungen für Sicherheitsziel B

Zur Herleitung der funktionalen Sicherheitsanforderungen für Sicherheitsziel B wird eine Fehlerbaumanalyse durchgeführt.

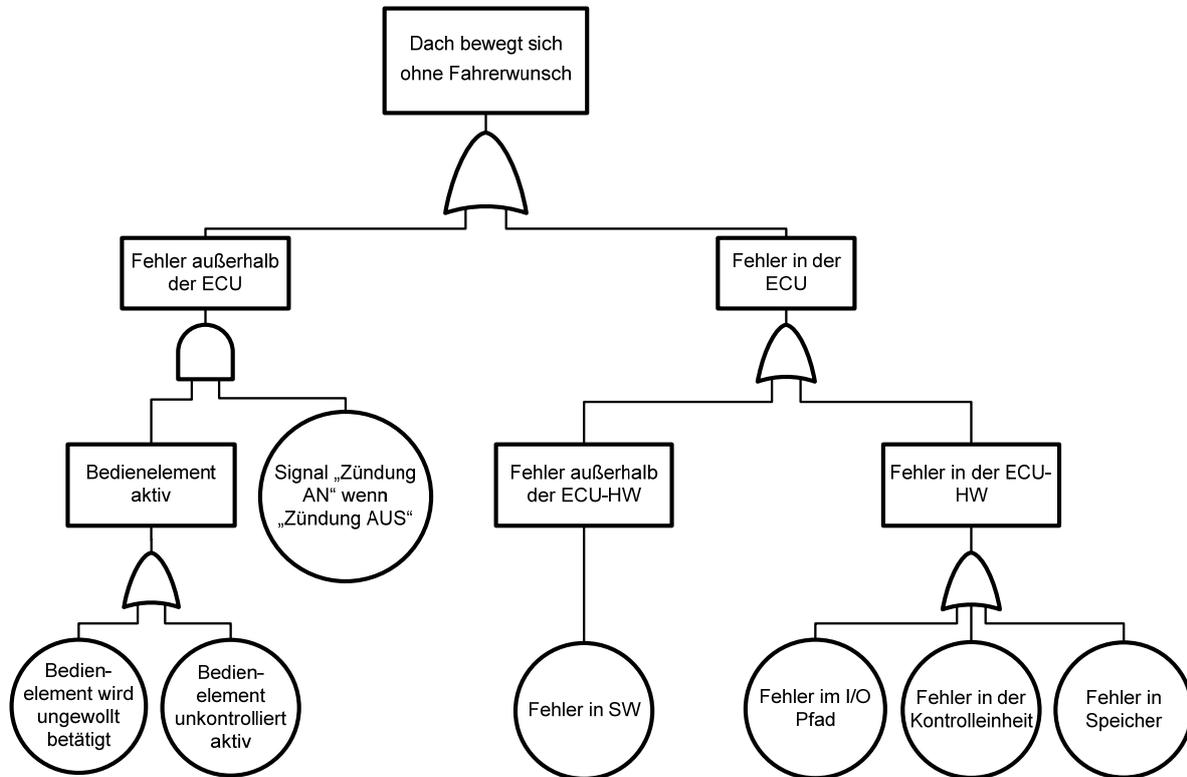


Abbildung 5-4: Fehlerbaumanalyse für Sicherheitsziel B

Aus dem für Sicherheitsziel B aufgestellten Fehlerbaum (Abbildung 5-4: Fehlerbaumanalyse für Sicherheitsziel B) lassen sich folgende funktionale Sicherheitsanforderungen ableiten:

1. Fehler im gemeldeten Zündungsstatus (Zündung AN wenn Zündung AUS) muss erkannt und behandelt werden.
2. Der Fehler *Bedienelement unkontrolliert aktiv* muss erkannt und behandelt werden.
3. Fehler im I/O Pfad, in der Kontrolleinheit und in Speichern, die das Sicherheitsziel verletzen können, müssen erkannt und behandelt werden.
4. SW-Fehler, die das Sicherheitsziel verletzen können, müssen erkannt und behandelt werden.

Weiterhin werden gemäß ISO26262-3, 7.4.2.5 folgende Attribute für alle Anforderungen festgelegt:

- ASIL C
- *Operating mode*: Initialisierung, Öffnen, Schließen, Stand-by (alle Modi)
- *Fault tolerant time span*: 100ms
- *Safe states*:
 - bevorzugt: komplett geöffnet, komplett geschlossen
 - sonst: gestopptes Dach in beliebiger Position
- *Transition to safe state*: Deaktivierung des Dachantriebs

5.1.2.6 Allgemeine weitere Anforderungen

Nach dem „*Warning and degradation concept*“ (ISO26262-3, 7.4.2.3) soll der Fahrer über auftretende Fehler informiert werden und ggf. der Funktionsumfang des Systems reduziert werden.

Eine Fehlbedienung des Systems wird dem Fahrer auf dem Display signalisiert. Hierzu zählt das Öffnen während der Fahrt, das Losfahren mit halb geöffnetem Dach oder das Öffnen ohne eingeschaltete Zündung.

Fehler, die zu einer Verletzung eines Sicherheitsziels führen, werden erkannt und behandelt. Dies geschieht durch Überführung in einen sicheren Zustand und somit zur Deaktivierung des Dachantriebs. Handelt es sich um einen permanenten Fehler, so wird der Fahrer hierüber durch einen „Repair Request“ informiert.

Eine weitere Anforderung ist, dass definiert wird, welche Handlungen des Fahrers oder einer anderen Person nötig sind, um die Sicherheitsziele zu erfüllen („*Actions of the driver...necessary to fulfil the safety goals*“ (ISO26262-3, 7.4.2.7)). Im Anwendungsbeispiel ergibt sich die Anforderung, dass bei einem ungewollten Bewegen des Daches der Fahrer das Bedienelement loslässt oder die Zündung abschaltet. Sollen alle Einfachfehler (z.B. *stuck at high* eines Tasters während der Dachbewegung) im Bedienelement entdeckt werden, ist es zudem nötig, dass bei einer ungewollten Dachbewegung die Taste für die entgegengesetzte Richtung betätigt wird.

5.1.2.7 Funktionale Systemarchitektur mit Sicherheitsmaßnahmen

Die Umsetzung der einzelnen Sicherheitsanforderungen fließen in den Entwurf der funktionalen Systemarchitektur mit Sicherheitsmaßnahmen ein, die in Abbildung 5-5 dargestellt ist. Sensoren und Aktuatoren sind in Form von Kreisen dargestellt, weitere Komponenten in Form von Rechtecken. Zwei Gruppen von Komponenten wurden zu den Funktionsgruppen Dachfunktion und Überwachungsfunktion zusammengefasst. Während die Funktionsgruppe Dachfunktion die eigentliche Steuerung des Cabriooverdecks übernimmt, ist es die Aufgabe der Überwachungsfunktion sicherzustellen, dass die aufgestellten Sicherheitsziele nicht verletzt werden. Zwischen den einzelnen Komponenten sind bereits mehr oder weniger detaillierte Kommunikationswege festgelegt worden. In den Fällen, in denen die spätere Implementierung der Kommunikation zwischen den einzelnen Elementen noch unklar ist, ist dies durch *-Symbole gekennzeichnet.

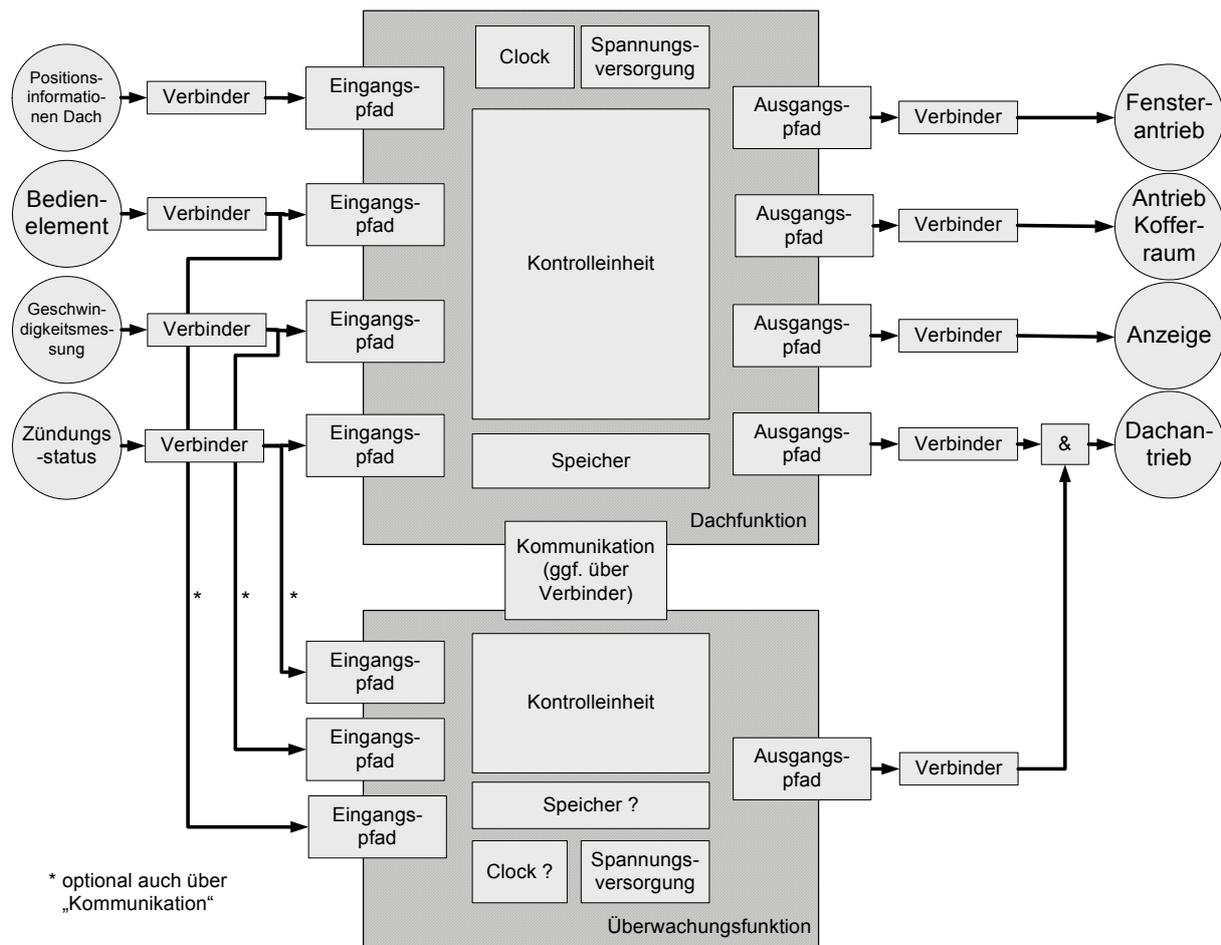


Abbildung 5-5: Funktionale Systemarchitektur

5.1.2.8 Verifikationskriterien für die Sicherheitsanforderungen

(Verification criteria for safety requirements)

Die Verifikationskriterien für die Sicherheitsanforderungen werden auf Basis der im vorherigen Kapitel aufgestellten Sicherheitsarchitektur formuliert.

Unter Fehlerbehandlung ist im Folgenden zu verstehen, dass das Dach bei erkanntem Fehler nicht bewegt werden darf und dass bei permanenten Fehlern ein „Repair request“ gemeldet wird.

Sicherheitsziel A:

1. Der Fehler *Geschwindigkeitswert fälschlicherweise = 0km/h* muss erkannt und behandelt werden.
 - a. Fehler in der Geschwindigkeitsmessung, die zum in 1 genannten Fehler führen können, müssen erkannt und behandelt werden.
 - b. Fehler im Verbindungselement und im Eingangspfad, die zum in 1 genannten Fehler führen können, müssen erkannt und behandelt werden.
2. Der Fehler *Bedienelement unkontrolliert aktiv* muss erkannt und behandelt werden.
 - a. Fehler im Bedienelement, die zum in 2 genannten Fehler führen können, müssen erkannt und behandelt werden.
 - b. Fehler im Verbindungselement, die zum in 2 genannten Fehler führen können, müssen erkannt und behandelt werden.

- c. Eine *fehlerhafte Bedienung* (Anforderung Dachbewegung durch Fahrer während der Fahrt) muss erkannt und behandelt werden.
- 3. Fehler im I/O Pfad, in der Kontrolleinheit und in Speichern, die das Sicherheitsziel verletzen können, müssen erkannt und behandelt werden.
 - a. Fehler in der Spannungsversorgung (zu niedrig, zu hoch, Ausfall), die zur Verletzung des Sicherheitsziels führen können, müssen erkannt und behandelt werden.
 - b. Fehler in der Clock (zu niedrig, zu hoch, Ausfall), die zur Verletzung des Sicherheitsziels führen können, müssen erkannt und behandelt werden.
 - c. Fehler in der Kontrolleinheit (HW: transient, permanent; SW: permanent), die zur Verletzung des Sicherheitsziels führen können, müssen erkannt und behandelt werden.
 - d. Fehler in den Speichern, die zur Verletzung des Sicherheitsziels führen können, müssen erkannt und behandelt werden.
 - e. Fehler im Eingangs- und Ausgangspfad (Dachantrieb), die zur Verletzung des Sicherheitsziels führen können, müssen erkannt und behandelt werden.

Sicherheitsziel B:

- 1. Fehler im *gemeldeten Zündungsstatus* (Zündung AN wenn Zündung AUS) müssen erkannt und behandelt werden.
 - a. Fehler in der Erkennung des Zündungsstatus, die zum in 1 genannten Fehler führen können müssen erkannt und behandelt werden.
 - b. Fehler im Verbindungselement und im Eingangspfad, die zum in 1 genannten Fehler führen können müssen erkannt und behandelt werden.
- 2. Der Fehler *Bedienelement unkontrolliert aktiv* muss erkannt und behandelt werden.
 - a. Fehler im Bedienelement, die zum in 2 genannten Fehler führen können, müssen erkannt und behandelt werden.
 - b. Fehler im Verbindungselement, die zum in 2 genannten Fehler führen können, müssen erkannt und behandelt werden.
 - c. Eine *fehlerhafte Bedienung* (Anforderung Dachbewegung durch Fahrer während der Fahrt) muss erkannt und behandelt werden.
- 3. Fehler im I/O Pfad, in der Kontrolleinheit und in Speichern, die das Sicherheitsziel verletzen können, müssen erkannt und behandelt werden.
 - a. Fehler in der Spannungsversorgung (zu niedrig, zu hoch, Ausfall), die zur Verletzung des Sicherheitsziels führen können, müssen erkannt und behandelt werden.
 - b. Fehler in der Clock (zu niedrig, zu hoch, Ausfall), die zur Verletzung des Sicherheitsziels führen können, müssen erkannt und behandelt werden.
 - c. Fehler in der Kontrolleinheit (HW: transient, permanent; SW: permanent), die zur Verletzung des Sicherheitsziels führen können, müssen erkannt und behandelt werden.
 - d. Fehler in den Speichern, die zur Verletzung des Sicherheitsziels führen können, müssen erkannt und behandelt werden.
 - e. Fehler im Eingangspfad und im Ausgangspfad (Dachantrieb), die zur Verletzung des Sicherheitsziels führen können, müssen erkannt und behandelt werden.

5.1.2.9 Zuordnung der funktionalen Sicherheitsanforderungen

(Allocation of functional safety requirements (ISO26262-3, 7.4.3))

Die funktionalen Sicherheitsanforderungen müssen in diesem Schritt Systemen, Systemelementen und Komponenten zugeordnet werden. Dies ist im Folgenden für die aufgestellte funktionale Sicherheitsarchitektur aufgeführt.

- Der Fehler *Geschwindigkeitwert fälschlicherweise = 0km/h* und Fehler im *gemeldeten Zündungsstatus (Zündung AN wenn Zündung AUS)* müssen erkannt und behandelt werden. In der Dachfunktion müssen daher die Signale auf Plausibilität geprüft werden. Falls dies nicht ausreicht, um alle Fehler zu erkennen, müssen bei der Quelle der Signale entsprechende Maßnahmen getroffen werden (Selbsttest, Prüfsumme beifügen, ...).
- Eine Bewegung des Daches ist, trotz angeforderter Dachbewegung, während der Fahrt zu unterbinden. Dazu müssen die Voraussetzungen in der Dachfunktion abgefragt und geprüft werden.
- Hardware-Fehler im Bedienelement und eine *fehlerhafte Bedienung* (Anforderung Dachbewegung durch Fahrer während der Fahrt) müssen erkannt und behandelt werden. Deshalb sind in der Dachfunktion die Signale vom Bedienelement zumindest auf Plausibilität zu prüfen (u.A. sind als Eingangssignale Flanken und nicht Pegel zu wählen um stuck-at Fehler zu erkennen).
- Fehler im I/O Pfad und in der Kontrolleinheit (inkl. Spannungsversorgung, Clock, Speicher, Software) müssen erkannt und behandelt (sicherstellen, dass Sicherheitsziel nicht verletzt wird) werden. Diese Überwachung geschieht in der Überwachungsfunktion.

5.1.2.10 Verifikation der funktionalen Sicherheitsanforderungen

(Verification of functional safety requirements (ISO26262-3, 7.4.5))

Zur Verifikation des funktionalen Sicherheitskonzeptes wird eine Fehlerbaumanalyse auf Basis der in 4.1.2.7 vorgestellten funktionalen Systemarchitektur mit Sicherheitsmaßnahmen durchgeführt. Ziel dieser Analyse ist es zu zeigen, dass die so genannten „Minimal Cut Sets“ größer werden, also die Anzahl der Basisereignisse, die gemeinsam auftreten müssen, um zur Verletzung des Sicherheitsziels zu führen, größer wird.

Die entsprechenden Fehlerbäume sind im Anhang A aufgeführt. Um die Erweiterung der „Cut Sets“ zu dokumentieren, sind alle im Rahmen des Sicherheitskonzeptes hinzugefügten Bestandteile des Fehlerbaums grau dargestellt. Alle Basisereignisse werden durch die Sicherheitsfunktionen erweitert (Verknüpfung der Basisereignisse über UND mit Sicherheitsfunktion). Lediglich im Unterbaum 5 wurden keine weiteren Sicherheitsfunktionen eingefügt. Dies ist damit zu begründen, dass die Überwachung des Bedienelementes schon im funktionalen Konzept vorgesehen war. Die Überwachung des Bedienelementes wiederum wird in den hinzugefügten Sicherheitsfunktionen berücksichtigt.

Anhand dieser Argumentation wird angenommen, dass das funktionale Sicherheitskonzept die gegebenen Sicherheitsanforderungen erfüllt.

5.1.3 Technisches Sicherheitskonzept

Aus dem funktionalen Sicherheitskonzept ist nun das technische Sicherheitskonzept abzuleiten. Da sich die technischen Konzepte der beiden untersuchten Architekturen unterscheiden, erfolgt die Betrachtung in getrennten Unterkapiteln.

5.1.3.1 Dual-Core Mikrocontroller Architektur

In diesem Abschnitt wird zunächst das technische Sicherheitskonzept für die Dual-Core Architektur abgeleitet. Hierzu müssen zunächst die Elemente des technischen Sicherheitskonzepts den konkreten Bauteilen der Hardware-Architektur zugeordnet werden. Anschließend wird festgelegt, welche Maßnahmen zur Fehlererkennung und -behandlung zu treffen sind und wann diese ausgeführt werden. Dies beeinflusst auch den Aufbau der Software-Architektur.

Konkretisierung des funktionalen Sicherheitskonzepts nach ISO26262-3, Abschnitt 7:

Kontrolleinheiten	Jeweils eine eigenständige CPU: Dachfunktion ↔ CPU1, Überwachungsfunktion ↔ CPU2
Speicher	Für jede CPU einen physikalisch getrennten RAM und ein gemeinsamer Flashspeicher
Clock und Spannungsversorgung	Für Dachfunktion und Überwachungsfunktion wird die selbe Clock und Spannungsversorgung genutzt
Eingangs- und Ausgangspfade	Analoge und digitale Ein- und Ausgänge sowie ein CAN-Controller
Verbinder	Kabel, Pins und CAN-Bus
Kommunikation	Exchange RAM (spezieller RAM für den Datenaustausch zwischen beiden Cores)

Spezifikation der technischen Sicherheitsanforderungen (ISO26262-4, 5.4.10):

Erkennung / Behandlung von Fehlern im System selbst:

Speicherfehler	Checksumme und Überprüfung dieser Checksumme im Programmspeicher; zusätzlich RAM-Tests (MarchC)
Clock	Benutzung eines externen Watchdogs
Spannungsversorgung	Verwendung eines Brown-Out-Chips ¹
Ein- u. Ausgangspfade und Kontrolleinheit	Zurücklesen sicherheitsrelevanter Aktuatoren durch den zweiten Core (dieser findet Fehler im ersten Core)
Bedienelement	Plausibilisierung

Erkennung / Behandlung von Fehlern in anderen ECUs oder Kommunikationssystemen:

- Fehler in der Übermittlung der Geschwindigkeitsmessung und des Zündungsstatus werden durch die Verwendung einer Prüfsumme und eines Nachrichtenzählers in den CAN-Nachrichten und durch einen Selbsttest des CAN-Controllers erkannt.

¹ Ein Brown-Out-Chip dient der Überwachung der Spannungsversorgung und führt bei zu geringer Spannung einen Neustart des Systems durch, um undefinierte Systemzustände zu vermeiden.

Zeitpunkte der Fehlererkennung:

Checksummentest	Während Initialisierung
Watchdog und Brown-Out Erkennung	Permanent
Überwachung CPU1 durch CPU2	Permanent
Überwachung CPU2 durch Watchdog	Zyklisch
Nachrichtenzähler und Prüfsumme für CAN-Nachrichten	Bei jeder Geschwindigkeits- und Zündungsnachricht
Selbsttest CAN-Controller	Während Initialisierung
E/A Plausibilisierung und Zurücklesen	Zyklisch

Maßnahmen zum Erreichen und Behalten des sicheren Zustandes:

- Deaktivieren der Hydraulikpumpe bzw. Pumpe deaktiviert lassen
- Deaktivieren des Enable-Eingangs der Hydraulikpumpe
- Ggf. Warnmeldung an den Benutzer

Weitere Sicherheitsanforderungen

Redundanz (ISO26262-4, 5.4.15.1 u. 5.4.15.2):

- Diverse Software-Redundanz soll die Deaktivierung der Pumpe sicherstellen
- Hardware-Redundanz bzgl. des Speichers und der CPU

Common Cause Failure (CCF) (ISO26262-4, 5.4.15.4):

- Ausfall beider CPUs oder des ganzen Mikrocontrollers durch extreme äußere Einflüsse (z.B. Hitze), durch Fehler in der Clock oder in der Spannungsversorgung
- Fehler in der Software für beide CPUs durch Fehler im gemeinsamen ROM
- Fehler in der Software durch teilweisen Zugriff beider Software-Lösungen auf die gleichen Treiber

Zum Erkennen von CCF werden ein externer Watchdog (Erkennung von Fehlern, die einen Gesamtausfall zur Folge haben) sowie ein Brown-Out-Chip (Erkennung von Fehlern in der Spannungsversorgung) eingesetzt. Weiterhin sind getrennte Speicher für beide Cores vorzusehen. Im vorliegenden Dual-Core Mikrocontroller wird der Programmcode für beide Cores aus technischen Gründen zu Beginn im gleichen ROM abgelegt und erst während der Initialisierung in getrennte Speicher verschoben. Um CCF in diesem Speicher zu behandeln, wird dieser mit einer Checksumme versehen und zumindest beim Start des Systems geprüft (Erkennung von Fehlern im ROM). CCF in Software werden dadurch vermieden, dass die Dachfunktion und die Überwachungsfunktion unterschiedlich implementiert werden (komplexe Funktion vs. einfache Überwachungsfunktion). Eine Quelle für CCF könnten jedoch gemeinsam genutzte Treiberfunktionen (*I/O-Funktionen*: einlesen bzw. setzen von Ports, *Timer-Funktionen*: Timer initialisieren und deren Interrupts aktivieren, *Interrupt-Funktionen*: einem Interrupt-Vektor eine Funktion zuweisen und diesen aktivieren) sein. Es ist daher durch geeignetes Testen sicherzustellen, dass diese nicht direkt zu Verletzung der Sicherheitsziele führen können.

Hardware/Software Architektur

Die aus den Anforderungen abgeleitete Hardware/Software Architektur für die Dual-Core Architektur ist in Abbildung 4-6 dargestellt, während die Software-Architektur in Abbildung 5-7 näher beschrieben wird. Die Architektur der erarbeiteten Funktionsüberwachung ist schließlich in Abbildung 5-8 beschrieben.

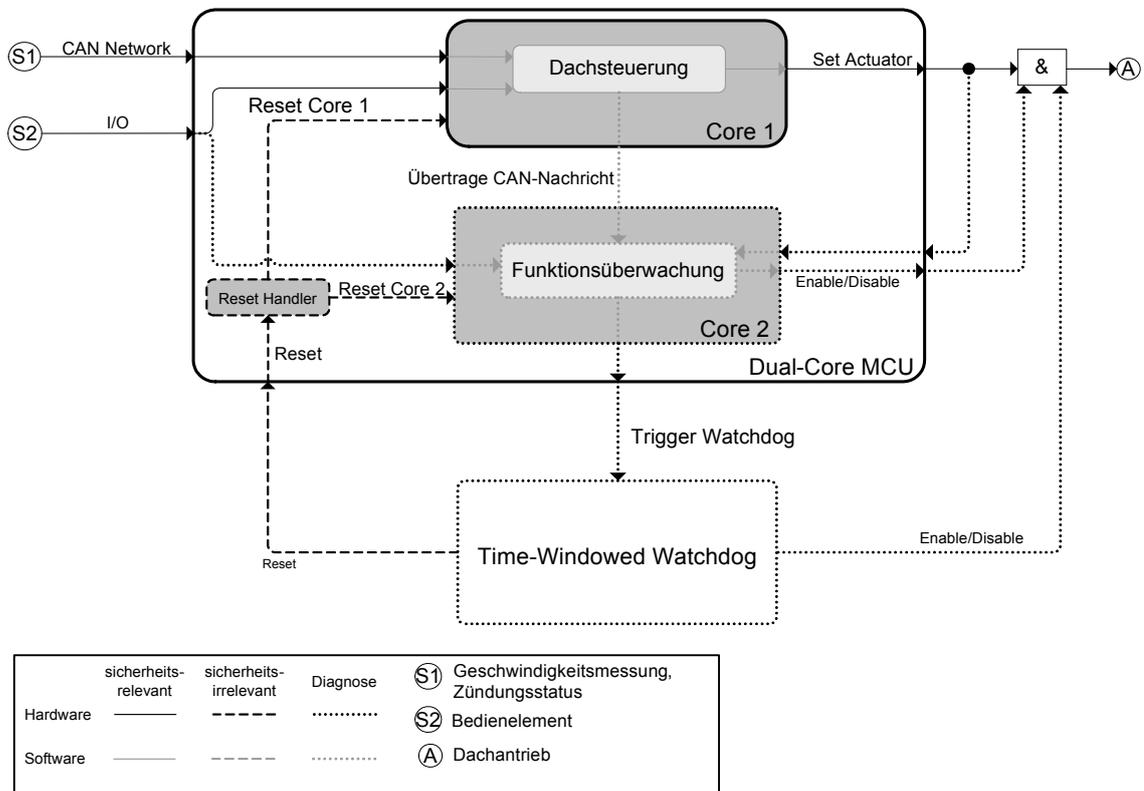


Abbildung 5-6: Dual-Core Hardware-Architektur

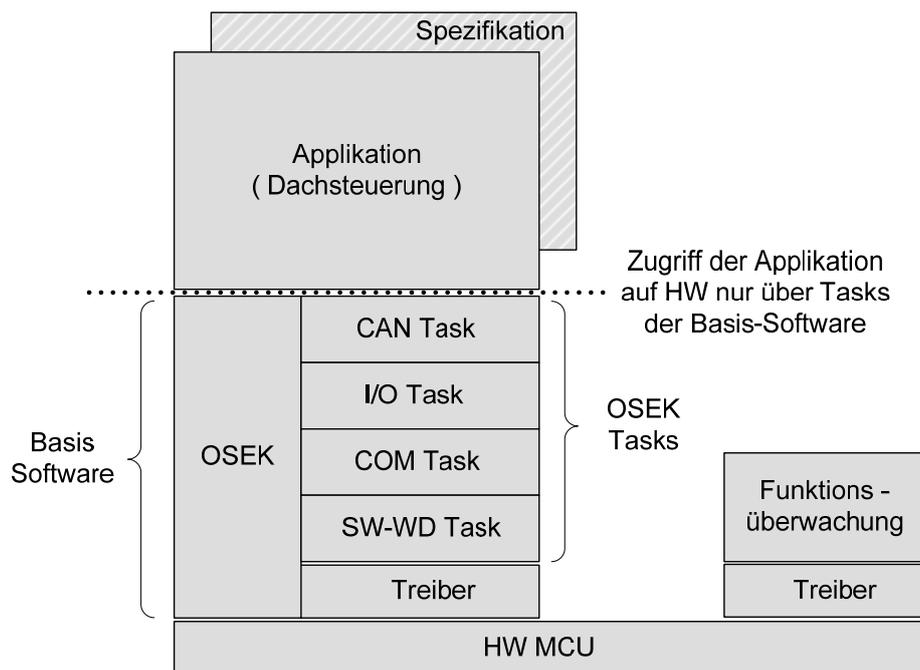


Abbildung 5-7: Dual-Core Software-Architektur

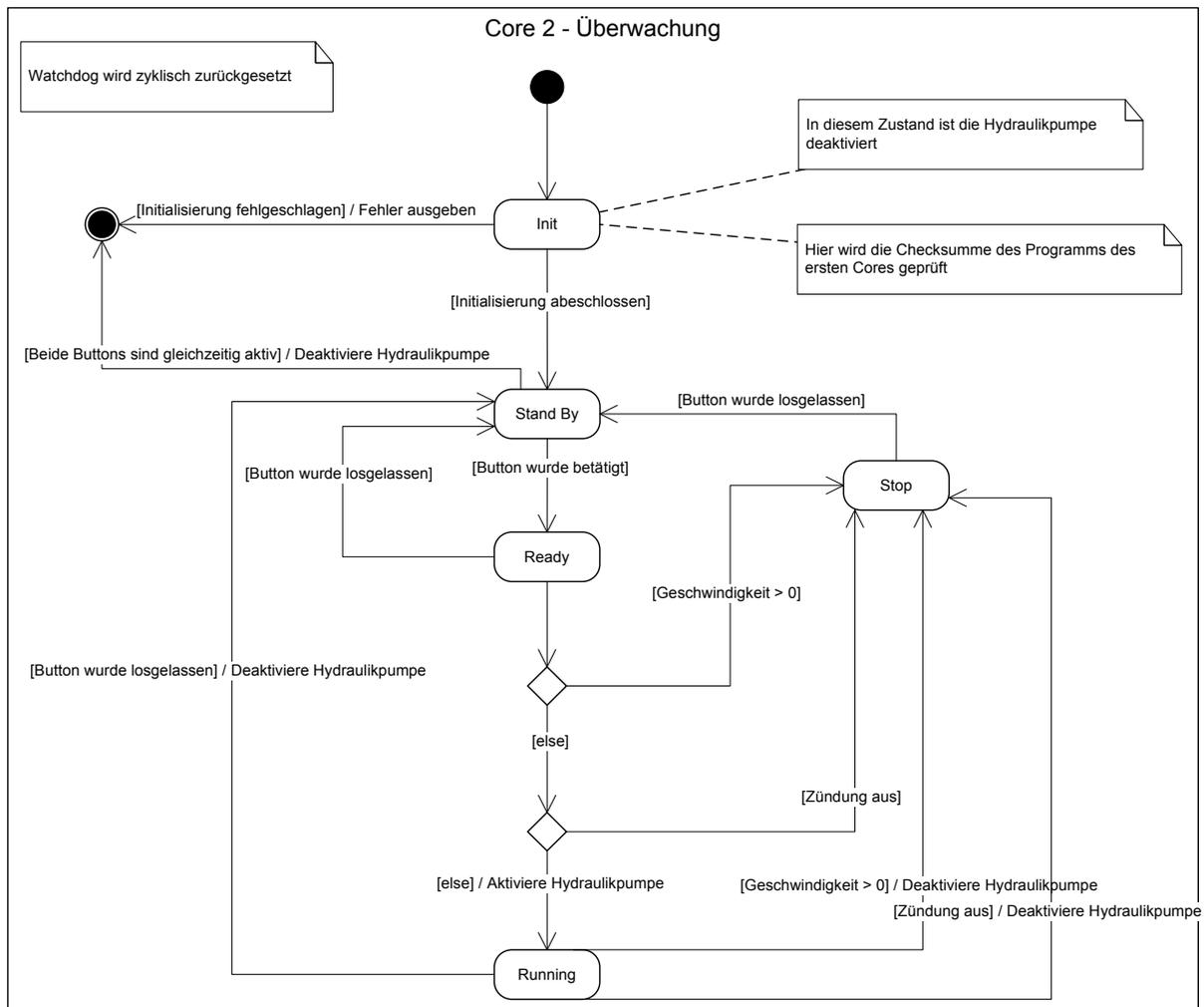


Abbildung 5-8: Dual-Core Überwachungsfunktion

5.1.3.2 Mikrocontroller + FPGA Architektur

Analog zur Dual-Core Architektur leitet sich das technische Sicherheitskonzept für die Mikrocontroller + FPGA Architektur ab.

Konkretisierung des funktionalen Sicherheitskonzepts nach ISO26262-3, Abschnitt 7:

Kontrolleinheiten	Dachfunktion ⇔ MCU, Überwachungsfunktion ⇔ FPGA
Speicher	Jeweils getrennter Speicher für die Dachfunktion und die Überwachungsfunktion
Clock und Spannungsversorgung	Jeweils getrennte Clock und Spannungsversorgung für die Dachfunktion und die Überwachungsfunktion
Eingangs- und Ausgangspfade	Digitale Ein- und Ausgänge bei MCU und FPGA, analoge Eingänge sowie CAN auf der MCU
Verbinder	Kabel, Pins und CAN-Bus
Kommunikation	SPI Schnittstelle

Spezifikation der technischen Sicherheitsanforderungen (ISO26262-4, 5.4.10):

Erkennung / Behandlung von Fehlern im System selbst:

Speicherfehler	Jeweils andere Kontrolleinheit erkennt Ausfall durch Ausbleiben zyklischer Kommunikation (Watchdog-funktionalität), außerdem Zurücklesen des Treibersignals für die Pumpe (sicherheitskritischer Aktuator)
Clock	Siehe Speicherfehler
Spannungsversorgung	Verwendung eines Brown-Out-Chips
Eingangs- und Ausgangspfade und Kontrolleinheit	Redundante Umsetzung der Aktivierung des sicherheitskritischen Aktuators und Zurücklesen des Aktuators
Bedienelement	Plausibilisierung sowie redundantes Einlesen

Erkennung / Behandlung von Fehlern in anderen ECUs oder Kommunikationssystemen:

- Fehler in der Übermittlung der Geschwindigkeitsmessung und des Zündungsstatus werden durch die Verwendung von einer Prüfsumme und eines Nachrichtenzählers in den CAN-Nachrichten und durch Selbsttest des CAN-Controllers erkannt.
- Fehler in der SPI-Kommunikation werden durch die Verwendung einer Prüfsumme und eines Nachrichtenzählers erkannt.

Zeitpunkt der Fehlererkennung:

Brown-Out-Erkennung	Permanent
Überwachung MCU durch FPGA	Permanent
Nachrichtenzähler und Prüfsumme für CAN-Nachrichten	Bei jeder Geschwindigkeits- und Zündungsnachricht
Selbsttest CAN-Controller	Während Initialisierung
Nachrichtenzähler und Prüfsumme für SPI-Kommunikation	Bei jeder gesendeten Nachricht über die SPI Schnittstelle
Plausibilisierung der Eingänge	Bei jedem Einlesen
Zurücklesen von Ausgängen	Zyklisch

Maßnahmen zum Erreichen und Behalten des sicheren Zustandes:

- Deaktivieren der Hydraulikpumpe bzw. Pumpe deaktiviert lassen
- Deaktivieren des Enable-Eingangs der Hydraulikpumpe
- Ggf. Warnmeldung an den Benutzer

Weitere Sicherheitsanforderungen

Redundanz (ISO26262-4, 5.4.15.1 u. 5.4.15.2):

- Diverse Software-Redundanz → Diverse Implementierung der Deaktivierung der Pumpe
- Hardware-Redundanz (getrennte Hardware für Dachfunktion und Überwachung)

Common Cause Failure (CCF) (ISO26262-4, 5.4.15.4):

Fast alle CCF werden durch die physikalische Trennung und diverse Implementierung von Dachfunktion und Überwachung ausgeschlossen. Zu beachten sind allerdings die folgenden CCF:

- Ausfall von MCU und FPGA durch extreme äußere Einflüsse (z.B. Hitze)
- Fehler in der Spannungsversorgung (z.B. eine zu hohe Spannung)

Diese CCF sind grundsätzlicher Natur und betreffen alle betrachteten Architekturen. Aus diesem Grund wurden Maßnahmen zur Behandlung dieser Fehler im Rahmen dieser Arbeit nicht weiter untersucht.

Hardware/Software Architektur

Die aus den Anforderungen abgeleitete Hardware/Software Architektur für die Mikrocontroller-FPGA Architektur ist in Abbildung 5-9 dargestellt, während die Software-Architektur in Abbildung 5-10 näher beschrieben wird. Die Architektur der erarbeiteten Funktionsüberwachung ist schließlich in Abbildung 4-11 beschrieben.

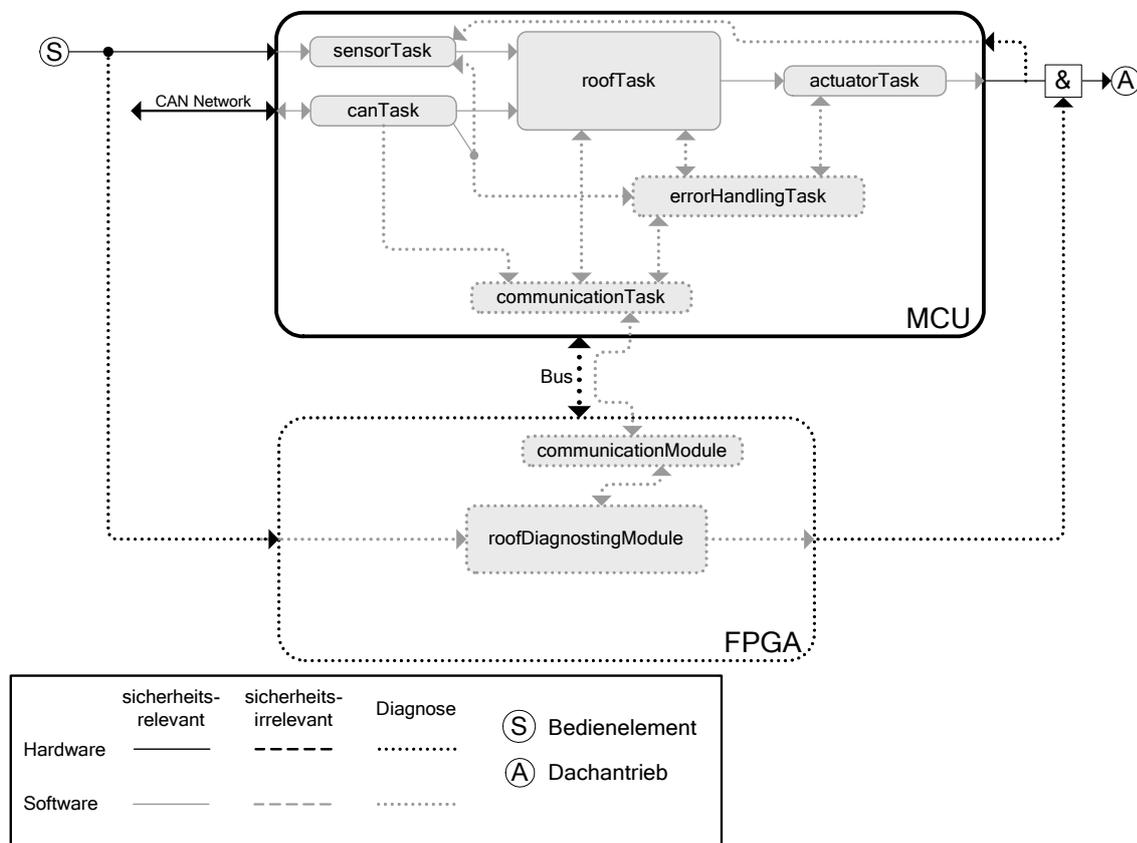


Abbildung 5-9: MCU-FPGA Hardware-Architektur

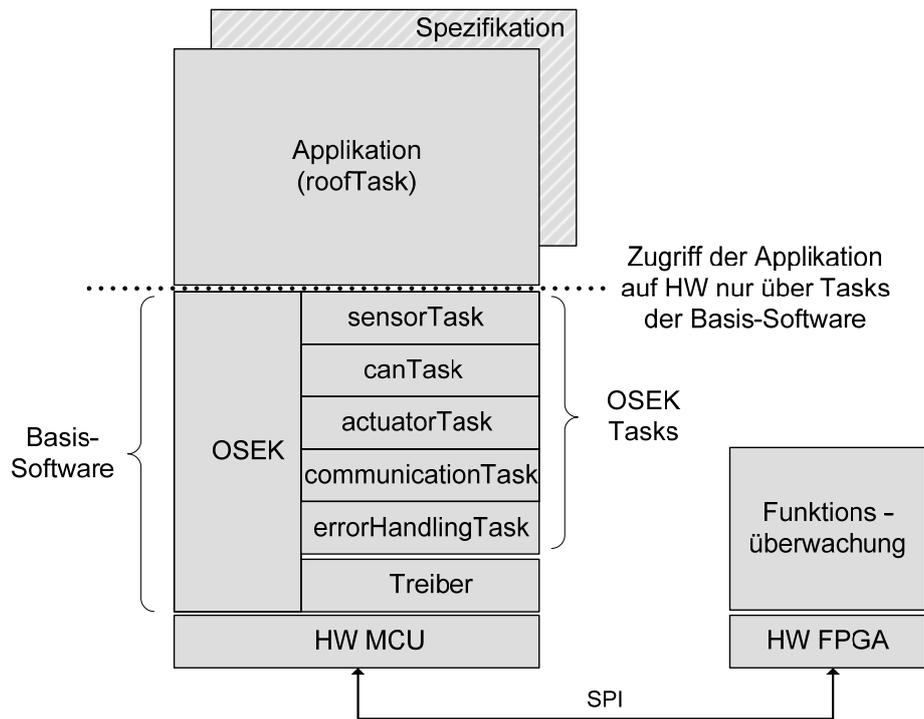


Abbildung 5-10: MCU-FPGA Software-Architektur

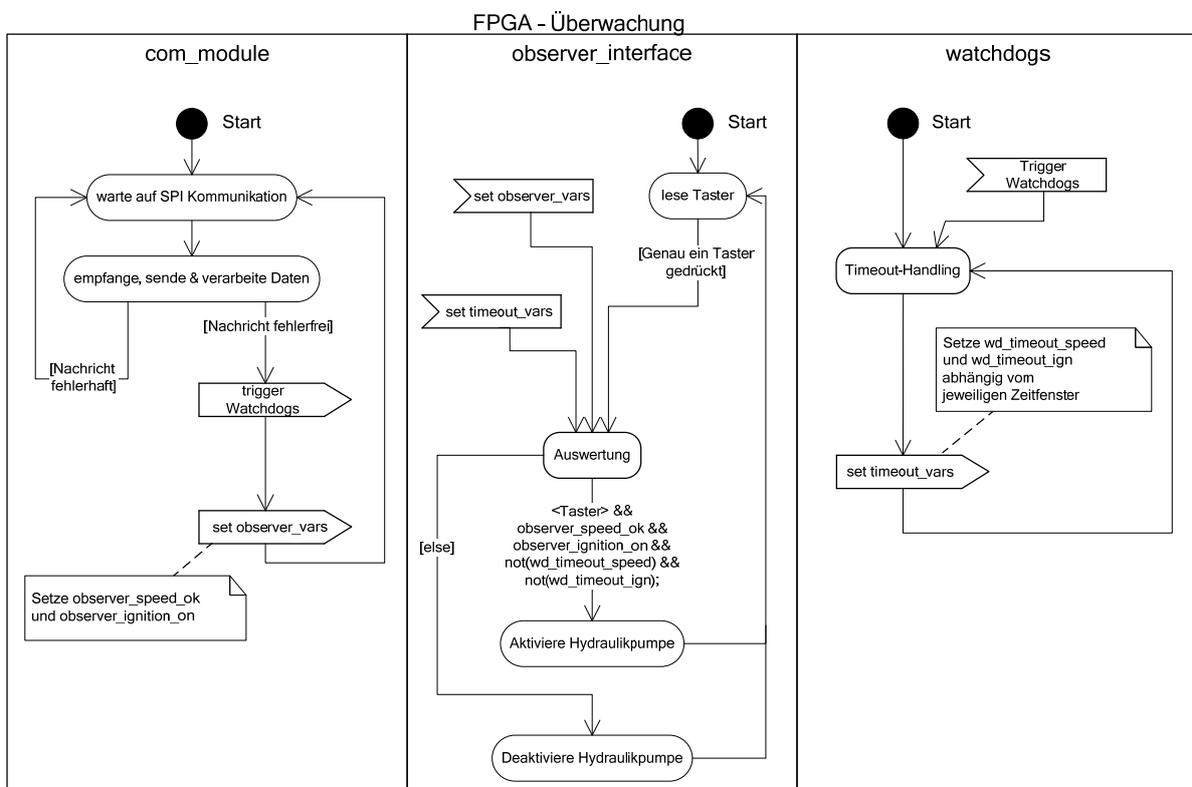


Abbildung 5-11: MCU-FPGA Überwachungsfunktion

5.1.4 Fehlermetriken und Diagnoseabdeckung

Die ISO26262 fordert die Einhaltung gewisser Metriken für Single Point Fehler, Residual Fehler und Latent Multiple Point Fehlern (*ISO26262-5, Abschnitt 7*). Weiterhin ist zu zeigen, dass die Diagnoseabdeckung sicherheitsrelevanter Komponenten ausreichend hoch ist.

Für die Untersuchungen in dieser Studie wurden die Ausfallraten von Vergleichskomponenten für die Bestimmung der Fehlermetriken verwendet, da für die verwendeten Bausteine keine geeigneten Daten vorlagen. Die aufgestellten Tabellen zur Bestimmung der Metriken sind im Anhang B zu finden, während im Folgenden die Ergebnisse für beide Architekturen kurz dargestellt sind.

5.1.4.1 Dual-Core Mikrocontroller Architektur

Gesamtfehlerrate Dual-Core Architektur: 53,6 FIT
Residual/Single-Point Fehlerrate: 1,12 FIT → SPFM: 97,90%
Latent Multiple-Point Fehlerrate: 6,48 FIT → LFM: 87,64%

SPFM: **97,90%** (*Single Point Fault Metric*, gefordert: >97% (*ISO26262-5, Table 5*))
LFM: **87,64%** (*Latent Fault Metric*, empfohlen: >80% (*ISO26262-5, Table 5*))

Folglich werden die Anforderungen beider Fehlermetriken von dieser Architektur erfüllt. Außerdem ist die Diagnoseabdeckung für die verwendeten Komponenten ausreichend hoch (siehe Anhang B).

5.1.4.2 Mikrocontroller + FPGA Architektur

Gesamtfehlerrate Mikrocontroller-Architektur: 93,2 FIT
Residual/Single-Point Fehlerrate: 0,004 FIT → SPFM: 99,99%
Latent Multiple-Point Fehlerrate: 4,19 FIT → LFM: 95,50%

SPFM: **99,99%** (*Single Point Fault Metric*, gefordert: >97% (*ISO26262-5, Table 5*))
LFM: **95,50%** (*Latent Fault Metric*, empfohlen: >80% (*ISO26262-5, Table 5*))

Folglich werden die Anforderungen beider Fehlermetriken von dieser Architektur erfüllt. Außerdem ist die Diagnoseabdeckung für die verwendeten Komponenten ausreichend hoch (siehe Anhang B).

5.2 Umsetzung des Konzeptes

Parallel zur Konzepterstellung erfolgte eine Einarbeitung der Studenten in die entsprechenden Entwicklungsumgebungen und Hardware-Architekturen. Weiterhin wurden schon zu diesem Zeitpunkt benötigte Treiber implementiert und das OSEK Betriebssystem eingerichtet.

Die eigentliche Implementierung des Anwendungsbeispiels und der entwickelten Sicherheitsfunktionen erfolgte nach Abschluss der Konzepterstellung. Bei dieser Umsetzung wurden Programmierrichtlinien (Coding guidelines) angewendet und automatisch (PCLint) und manuell überprüft. Die erstellten Treiber wurden ebenfalls hinsichtlich dieser Richtlinien überarbeitet. Weiterhin wurden im Fall der Dual-Core Architektur Komplexitätsmetriken eingesetzt um den erstellten Quellcode lesbarer zu machen.

Als Hardware-Komponenten wurden Entwicklungsboards eingesetzt, um den Entwicklungsaufwand gering zu halten. Im Fall der Dual-Core Architektur wurde lediglich eine Schaltung benötigt, um einen time-windowed Watchdog anzubinden. Für die Mikrocontroller-FPGA Architektur wurden jeweils ein Entwicklungsboard für den Mikrocontroller und den FPGA verwendet, die über eine serielle Schnittstelle (SPI) miteinander verbunden wurden. Als Interface zwischen FPGA (Betriebsspannung 3,3V) und dem Mikrocontroller (Betriebsspannung 5V) wurde eine vorhandene Schnittstellenplatine eingesetzt.

Die Dokumentation der Arbeitsschritte (inkl. aufgetretener Probleme) ist in Anhang C zu finden.

Für ein erstes Testen der Anwendung wurde eine Platine zur manuellen Umgebungssimulation entworfen und gefertigt. Diese ist mit Tastern (digitale Sensoren), einem Potentiometer (analoge Sensoren) und LEDs (digitale Ausgänge) bestückt. Weiterhin wurde ein CAN-USB Monitor zur Simulation des CAN Busses genutzt. Umfangreiches Testen und das Testen von Echtzeitbedingungen sind auf diesem Weg jedoch nur eingeschränkt möglich. Daher wurde parallel dazu in einer weiteren Diplomarbeit eine Simulations- und Testumgebung erstellt. Diese Umgebung basiert auf einer Kombination aus einem PC zur Simulation der Umgebung und als Benutzerschnittstelle (beides realisiert in Matlab Simulink) und einem FPGA als flexible und echtzeitfähige Schnittstelle zum Testobjekt. Die Kommunikation zwischen diesen beiden Komponenten ist über eine serielle Schnittstelle realisiert. Für Details zur Testumgebung sei auf den Anhang E und die Diplomarbeit [Dülks] verwiesen.

5.3 Bewertung des Entwicklungsaufwandes

Der Entwicklungsfortschritt und aufgetretene Probleme wurden wöchentlich dokumentiert (siehe Anhang C). Weiterhin wurden die Studenten gebeten, ihren eigenen Aufwand für die unterschiedlichen Teilbereiche selber zu bewerten.

5.4 Bewertung des Einflusses durch ASIL

Der bestimmte ASIL hat Auswirkungen auf den Entwurf und die Umsetzung der Anwendung. In diesem Arbeitsschritt wird daher untersucht, welche Maßnahmen nach fertig gestellter Entwicklung eingespart werden können, falls die Anwendung statt als ASIL C nun als ASIL B klassifiziert würde. Als mögliches Szenario wurde vorgegeben, dass die entwickelte Anwendung nun in einem anderen Kontext eingesetzt wird (z.B.: leicht modifiziertes Dach), der zu einem niedrigeren ASIL führt.

Weiterhin wird bewertet, welcher Aufwand bis zum durchgeführten Wechsel (z.B.: ASIL C → ASIL B) bereits angefallen ist, der bei einer Entwicklung nach ASIL B nicht nötig gewesen wäre. Ziel dieser Untersuchung ist es, die Frage zu beantworten, ob eine grundsätzliche Entwicklung nach ASIL C einer Entwicklung mit angepasstem ASIL für jedes individuelle Produkt vorzuziehen ist.

5.5 Untersuchung der Zuverlässigkeit

Die Untersuchung der Zuverlässigkeit erfolgte in mehreren Schritten. Zunächst wurden die erstellten Architekturen hinsichtlich ihrer Zuverlässigkeit analysiert. Diese Analyse beinhaltete sowohl eine theoretische Untersuchung möglicher Schwachstellen der Architekturen, als auch das Testen der Systeme mit der erstellten Simulations- und Testumgebung. Identifizierte Schwachstellen wurden hinsichtlich ihrer Risiken bewertet.

In einem zweiten Schritt erfolgten Überlegungen zur Optimierung hinsichtlich der Zuverlässigkeit der beiden Architekturen. Folgende Optimierungen sind in diesem Zusammenhang grundsätzlich möglich:

- Eine Optimierung der Sicherheitsfunktionen, um die Auftretswahrscheinlichkeit von unerwünschten Übergängen in den sicheren Zustand (Falsch Positive Fehlerreaktion) zu reduzieren.
- Eine Nutzung vorhandener Redundanzen zur Erhöhung der Zuverlässigkeit (z.B. Nutzung der zwei Cores im Dual-Core Mikrocontroller). In diesem Zusammenhang muss jedoch darauf geachtet werden, dass die Anforderungen der ISO26262 gemäß ASIL C weiterhin erfüllt werden.

Des Weiteren wurden Analysen bezüglich der Wechselwirkungen zwischen Sicherheit und Zuverlässigkeit durchgeführt.

5.6 Untersuchung der Testbarkeit

Zur Überprüfung der Funktionsfähigkeit der Implementierungen des Anwendungsbeispiels sollten im Rahmen der Studie Tests durchgeführt werden. Der Schwerpunkt dieser Tests teilte sich in 2 Teilbereiche auf. Zum einen wurde die Konformität der entwickelten Implementierungen mit der Systemspezifikation überprüft. Zum anderen sollte die Effektivität der entworfenen Sicherheitsmechanismen durch Testen bewertet werden. Da sich die Effektivität dieser Mechanismen nur im Fehlerfall zeigt, wurden zum Testen gezielt Fehler in die zu testenden Systeme eingefügt (Fehlerinjektion).

Abschließend sollte bewertet werden, wie gut sich die entwickelten Systeme testen ließen und inwiefern die ermittelten Testergebnisse einen Rückschluss auf die allgemeine Effektivität der gewählten Überwachungskonzepte zulassen.

5.7 Betrachtung der Änderbarkeit

Ursprünglich sollten gezielt ausgewählte Änderungsszenarien für beide Architekturen untersucht werden, um eventuelle Unterschiede zu identifizieren. Im Rahmen der Festlegung möglicher Änderungsszenarien im Arbeitskreis wurde deutlich, dass im Falle der beiden Architekturen vermutlich keines der diskutierten Änderungsszenarien zu unterschiedlichen Ergebnissen führen würde. Zu ähnlichen Ergebnissen kam man in Bezug auf Produktlinienfähigkeit der beiden Ansätze. Aus diesem Grund wurde der Untersuchung der Zuverlässigkeit und der Testbarkeit eine höhere Priorität zugewiesen und die Untersuchung der Änderbarkeit auf eine Diskussion möglicher Änderungsszenarien mit den am Projekt beteiligten Studenten beschränkt.

6 Ergebnisse der Studie

6.1 Aufwand der SW Implementierung

6.1.1 Dokumentation des Projektfortschritts und von Problemen

Die Dokumentation des Projektfortschritts ist in Anhang C zu finden. Es wird deutlich, dass ein großer Teil des Aufwandes in die Erarbeitung der Sicherheitskonzepte geflossen ist. Dies ist, wie im folgenden Kapitel näher erläutert wird, auf die benötigte Einarbeitung in Sicherheitsnormen zurückzuführen. Der Aufwand in der Konzepterstellung unterscheidet sich für die beiden Architekturen daher nur unwesentlich. Lediglich der Punkt der *Common Cause* Fehler (CCF) musste bei der MCU-FPGA-Architektur auf Grund der räumlichen Trennung zwischen Applikation (Mikrocontroller) und Sicherheitsfunktion (FPGA) weniger Beachtung finden als bei der Dual-Core-Architektur, in der beide Funktionen auf einem Chip realisiert wurden.

In Bezug auf die Implementierung floss der größte Teil des Aufwandes in die eigentliche Anwendung (Cabrioüberdecksteuerung). Da diese in beiden Fällen auf einem Mikrocontroller implementiert wurde, unterschieden sich die Vorgehensweisen für die beiden Architekturen nur unwesentlich. Die Sicherheitsfunktion wurde auf unterschiedliche Arten realisiert, ließen sich jedoch vergleichsweise einfach implementieren. Der Aufwand hier lag lediglich in der einmaligen Implementierung der Kommunikation zwischen dem MCU und dem FPGA (SPI-Verbindung) bzw. der Kommunikation über ein Exchange-RAM im Falle des Dual-Core Mikrocontrollers. In letzterem Fall mussten zudem einige Besonderheiten von Dual-Core Mikrocontrollern beachtet werden (Handhabung von Interrupts für jeden Core einzeln, Trennung der Programmspeicher für beide Cores). Dieser Zusatzaufwand wird in beiden Fällen als gering und als Initialaufwand eingeschätzt.

Im späteren Verlauf der Entwicklung hat sich herausgestellt, dass im Fall der Dual-Core-MCU Architektur die Benutzung eines Debuggers für beide Kerne gleichzeitig unerwartet aufwendig ist. Die Ursache hierfür wird zum einen in der starken Verzahnung des Systems (Core2 ruft Daten von Core1 ab) und zum anderen in der noch experimentellen Entwicklungsumgebung gesehen. Während letzteres als Initialaufwand einzustufen ist, handelt es sich bei ersterem um ein prinzipielles Problem der gewählten Architektur. Im Fall der FPGA-MCU Architektur traten zwar ähnliche Problematiken auf, äußerten sich jedoch nicht wie oben dargestellt, da der Fehlerbereinigungsprozess im Fall eines FPGA auf eine andere Art und Weise durchgeführt wird (Nutzung detaillierter Ausgaben, Logik-Analysator oder Simulation).

6.1.2 Bewertung des Aufwandes durch Entwickler

Es wurde zum einen der Aufwand für den Entwurf, zum anderen der Aufwand für die Implementierung bewertet (siehe Abbildung 6-1: Bewertung des Entwurfs- und Implementierungsaufwands für Anwendungsbeispiel). Um Initialaufwand (zusätzlicher Aufwand, der bei erstmaliger Anwendung der Norm und der verwendeten Hardware-Plattformen anfällt), von Folgeaufwand (Aufwand bei neuer Implementierung einer ähnlichen Funktion auf gleicher Hardware) zu unterscheiden, wurde zu dem der Aufwand für ein Folgeprojekt (Fenstersteuerung) auf gleicher Hardware bewertet (siehe Abbildung 6-2: Bewertung des Entwurfs- und Implementierungsaufwands für Folgeprojekt).

Aus den Bewertungen geht hervor, dass bei der ersten Anwendung der Entwurf der Sicherheitskonzepte 80-90% des Aufwandes ausmacht. Dies ist durch den Einarbeitungsaufwand in die Sicherheitsnorm und die nur mittlere Komplexität der zu implementierenden Funktion zu erklären. Dies wird bestätigt durch die Bewertung des Folgeaufwands, bei dem nur noch 70% des Aufwandes auf die Sicherheitskonzepte entfallen. Weiterhin sind keine wesentlichen Unterschiede zwischen den beiden Architekturen zu erkennen.

Im Falle des Implementierungsaufwandes wird in beiden Fällen nur ein verhältnismäßig kleiner Aufwand in der Implementierung der Sicherheitsfunktion gesehen (15%), während die Implementierung der eigentlichen Anwendung den größten Aufwand darstellt (55-75%).

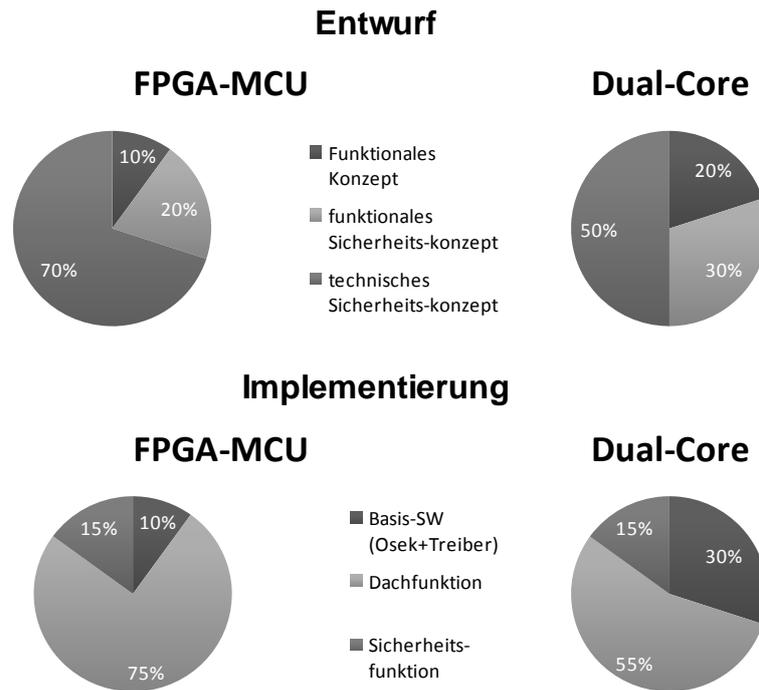


Abbildung 6-1: Bewertung des Entwurfs- und Implementierungsaufwands für Anwendungsbeispiel

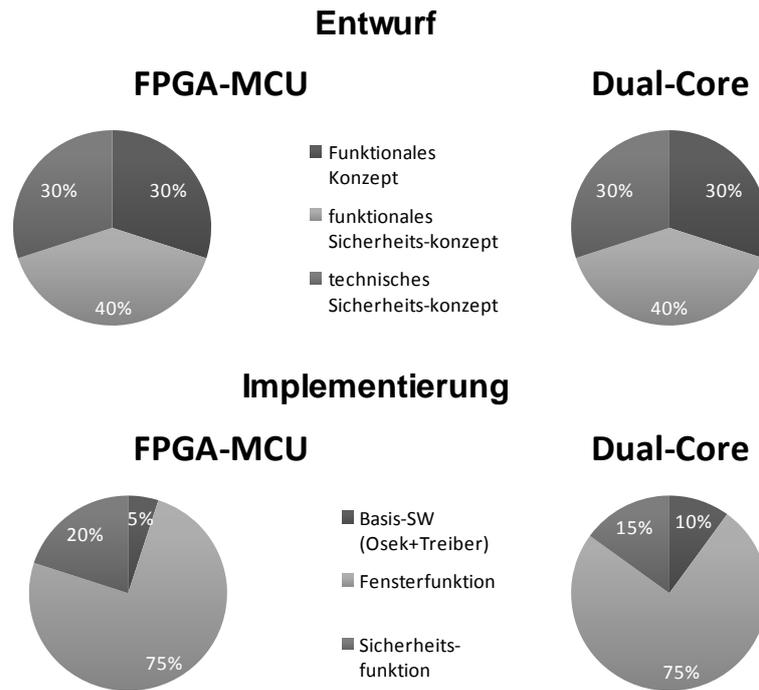


Abbildung 6-2: Bewertung des Entwurfs- und Implementierungsaufwands für Folgeprojekt

6.1.3 Fazit Entwicklungsaufwand

Zwischen den beiden Architekturen konnten keine wesentlichen Unterschiede bezüglich des SW-Entwurfs und der SW-Implementierung festgestellt werden. Der Grund ist hauptsächlich darin zu sehen, dass der Punkt der die beiden Ansätze unterscheidet (Realisierung der Sicherheitsfunktion) nur einen kleinen Anteil am Gesamtaufwand trug.

6.2 Zuverlässigkeit der Ansätze

Zu beiden Ansätzen gibt es verschiedene Überlegungen, welche Fehler möglich sind und in welcher Weise sie sich negativ auf die Zuverlässigkeit auswirken können. Weiterhin werden verschiedene Konzepte vorgestellt, wie eine geeignete Behandlung dieser Fehler aussehen könnte, um die Zuverlässigkeit des jeweiligen Systems zu erhöhen. Da eine Implementierung der Ansätze zur Erhöhung der Zuverlässigkeit aus Zeitgründen nicht erfolgte, basieren die im Folgenden vorgestellten Ergebnisse auf theoretischen Überlegungen.

6.2.1 Analyse der Architekturen hinsichtlich der Zuverlässigkeit

In beiden Architekturen können Fehler in Hardware oder Software dazu führen, dass das Programm falsch ausgeführt wird und sich beispielsweise in einer Schleife aufhängt. Eine korrekte Funktion der Anwendung ist dann nicht mehr möglich. Ein Lösungsansatz für beide Architekturen ist, neben der Überwachung sicherheitskritischer Komponenten auch die Ausführung der Dachfunktion zu überwachen. Da es sich hier nicht um eine Überwachungsfunktion zur Gewährleistung der funktionalen Sicherheit handelt, sondern um eine Maßnahme zur Erhöhung der Zuverlässigkeit, sind die Anforderungen der ISO 26262 an eine solche Überwachung gering. Eine einfache Maßnahme der Überwachung stellt beispielsweise ein Watchdog dar, der in beiden Architekturen zur Erkennung des oben beschriebenen Fehlverhaltens verwendet werden kann (interner Watchdog ist in den meisten MCUs vorhanden, FPGA kann Watchdog-Funktion in MCU-FPGA Architektur übernehmen).

Durch transiente oder permanente Hardwarefehler (z.B. Bitkipper oder Stuck-At-Fehler in Speichern, im Prozessor oder auf Busleitungen) kann es vorkommen, dass die Dachfunktion oder aber auch die Funktionsüberwachung fälschlicherweise einen Aktuator deaktiviert. Solche falsch positiven Fehler lassen sich typischerweise durch Redundanzen minimieren. Einige Konzepte zu beiden Plattformen werden in 6.2.2 und 6.2.3 vorgestellt.

Wird, wie oben diskutiert, ein Watchdog zur Erhöhung der Zuverlässigkeit eingesetzt, der das System bei einem erkannten Fehler zurücksetzt (Reset), ergibt sich ein weiterer Zuverlässigkeitsaspekt, die Verfügbarkeit. Zwar beseitigt ein Reset transiente Fehler, durch den Neustart des Systems kommt es jedoch zu einer Verzögerung des Systemablaufs. Die Dauer der Verzögerung hängt stark von der Anwendung und der gewählten Implementierung ab (Wie umfangreich sind Initialisierungen und Selbsttests beim Start etc.). Weiterhin bestimmt die Anwendung, welche Verzögerungen akzeptabel sind (Die Verdecksteuerung des gewählten Anwendungsbeispiels toleriert vermutlich höhere Verzögerungen als eine Drive-by-wire Anwendung). Eine Optimierung dieser Vorgehensweise wird in 6.2.4 erläutert.

6.2.2 Erhöhung der Zuverlässigkeit der Dual-Core-Architektur

Wie bereits im vorherigen Abschnitt erläutert, können durch die Verwendung des internen Watchdogs bestimmte Fehler im Programmablauf erkannt werden. Dazu zählen neben Implementierungsfehlern auch transiente Hardwarefehler die das Programm in eine Endlosschleife versetzen. In nahezu jedem modernen Mikrocontroller ist ein solcher Watchdog integriert. Dieser kann jedoch aus Sicherheitsüberlegungen heraus häufig nicht für sicherheitskritische Aufgaben verwendet werden, da die Möglichkeit von Common Cause Failures besteht. Dieser interne Watchdog steht somit in vielen Fällen für nicht sicherheitskritische Aufgaben zur Verfügung. Auch im Fall des verwendeten Dual-Core Mikrocontrollers kann ein vorhandener interner Watchdog genutzt werden und es fallen somit keine zusätzlichen Hardwarekosten an. Auch die Implementierung ist vergleichsweise einfach zu realisieren. Ein großer Nachteil jedoch ist, dass man die Laufzeit im fehlerfreien Zustand genau kennen muss. Denn wird eine zu geringe Trigger-Zeit gewählt, so wird möglicherweise ein ungewollter und unnötiger Reset ausgeführt. Dies wiederum wirkt sich negativ auf die Zuverlässigkeit aus. Wird die Zeitspanne zu groß gewählt, werden Fehler unter Umständen nicht erkannt.

Eine weitere Möglichkeit zur Erhöhung der Zuverlässigkeit ist die Verdopplung der Dachfunktion auf dem zweiten Core. Durch einen Austausch der Information, welche Aktuatoren angesteuert werden, können Fehler in der Ansteuerung, verursacht durch Hardwarefehler, entdeckt werden. Auch hier fallen keine zusätzlichen Hardwarekosten an, wenn man die Überprüfung in Software realisiert. Bei dieser Variante ist eine Synchronisierung der beiden Cores jedoch zwingend erforderlich, da ein Vergleich nur sinnvoll ist, wenn die Berechnungen auf den gleichen Eingangs- und Zustandswerten basieren. Es ist jedoch fraglich, ob dieser Ansatz die Zuverlässigkeit generell erhöht. Zwar kann eine verbesserte Fehlererkennung in der Dachfunktion realisiert werden, jedoch erhöht sich die Wahrscheinlichkeit zufälliger Hardwarefehler durch die Verdopplung der Dachfunktion. Weiterhin setzt eine Fehlerbehandlung bei diesem Ansatz in der Regel einen Neustart des Systems voraus, da Fehler zwar erkannt, aber nicht lokalisiert werden können.

6.2.3 Erhöhung der Zuverlässigkeit der MCU-FPGA-Architektur

Analog zur Dual-Core-Architektur kann die Benutzung des internen Watchdogs des Mikrocontrollers der MCU-FPGA-Architektur die Zuverlässigkeit des Systems erhöhen. Implementierungsfehler sowie Hardwarefehler, die zum Aufhängen des Systems führen, werden so sicher erkannt. Aufwands- und Hardwarekosten sind vergleichsweise gering.

Eine weitere Möglichkeit der Optimierung der Zuverlässigkeit ist, die Überwachungsfunktion auf dem FPGA zu duplizieren um Fehler in einer der Überwachungsfunktionen tolerieren zu können (siehe Abbildung 6-3: MCU+FPGA mit verdoppelter Funktionsüberwachung und 1oo2-Entscheider). Mit einem 1oo2-Entscheider (logisches Oder) zur Aktivierung der sicherheitskritischen Aktuatoren (hier: Hydraulikpumpe), der ebenfalls im FPGA implementiert werden kann, wird die Anzahl der falsch positiven Fehler (z.B. Bitkipper oder Stuck-At-Fehler), die zum unbeabsichtigten Deaktivieren der Aktuatoren führen, reduziert, da ein Überwachungsmodul ausfallen kann, ohne dass das System abschaltet. Dies ermöglicht eine Erhöhung der Zuverlässigkeit und ist leicht umzusetzen (Modul kann im FPGA einfach dupliziert werden). Voraussetzung für eine Erhöhung der Zuverlässigkeit ist, dass der Entscheider (intern implementiert oder extern verschaltet) fehlerfrei oder zumindest sehr robust arbeitet. Die Sicherheit bei diesem Ansatz ist geringer einzustufen als bei der ursprünglichen Implementierung (die Wahrscheinlichkeit, dass eine Funktionsüberwachung den Aktuator im Fehlerfall fälschlicher Weise aktiviert, ist bei zwei Funktionsüberwachungen höher als bei einer). Dennoch können bei den gegebenen Ausfallraten der verwendeten Bauteile und einer hinreichend niedrigen Ausfallrate des 1oo2-Entscheiders die Fehlermetriken und sonstigen Anforderungen gemäß ASIL C erfüllt werden.

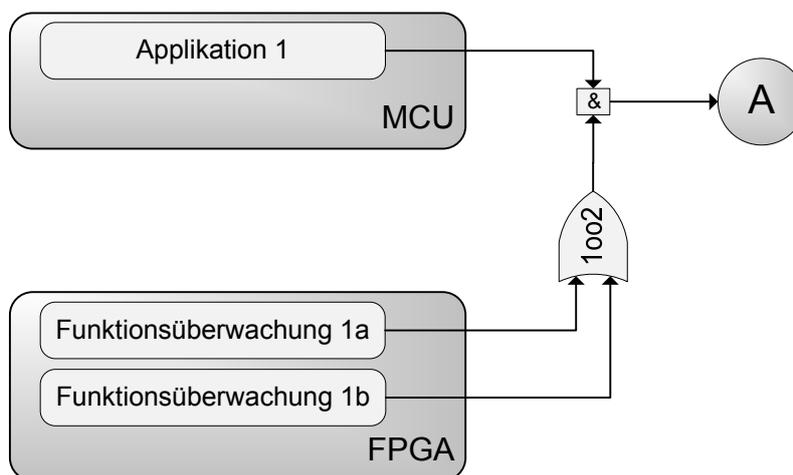


Abbildung 6-3: MCU+FPGA mit verdoppelter Funktionsüberwachung und 1oo2-Entscheider

Weiterhin lässt es diese Architektur zu, die Überwachungsfunktion auf dem FPGA auch zu verdreifachen (siehe Abbildung 6-4: MCU+FPGA mit dreifacher Funktionsüberwachung und 2oo3-Entscheider). Bei der Verwendung eines 2oo3-Voters steigen in diesem Fall sowohl die Zuverlässigkeit, als auch die Sicherheit. Falsch positive Fehler werden erkannt, aber ein einzelner falsch negativer Fehler kann nicht zur Freischaltung des Aktuators führen. Die Umsetzung ist auch hier vergleichsweise einfach. Allerdings steigen vermutlich bei beiden Ansätzen mit einer Vervielfachung der Funktionsüberwachung die Hardwarekosten, da ein größerer FPGA (etwa zweifache bzw. dreifache Chipfläche) und ein geeigneter Entscheider verwendet werden müssen.

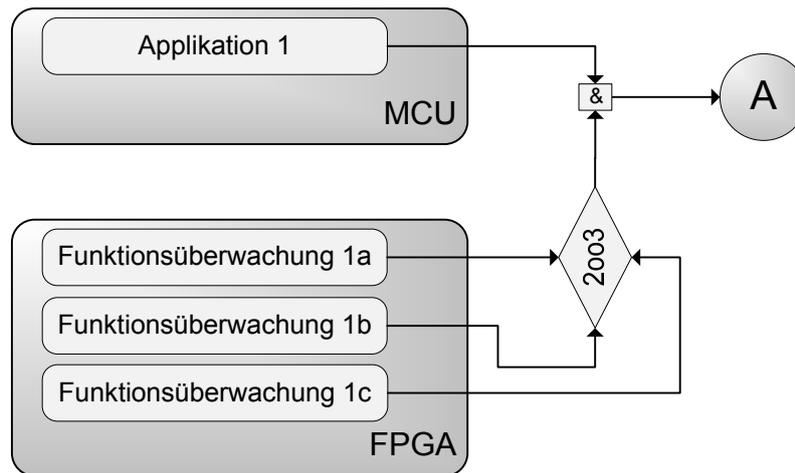


Abbildung 6-4: MCU+FPGA mit dreifacher Funktionsüberwachung und 2003-Entscheider

Eine weitere Variante ist es, die Dachfunktion zusätzlich zur MCU auch auf dem FPGA zu implementieren, um Fehler in der Dachfunktion besser erkennen zu können. Hierbei fallen jedoch erhöhte Hardwarekosten an (erfordert erheblich größeren FPGA) und die Implementierung ist sehr aufwendig. Wenn neben der Erkennung von Fehlern in der Dachfunktion (Behandlung über Neustart) auch eine Toleranz von Fehlern in der Dachfunktion gefordert wird, besteht die Möglichkeit, eine Simulation des Ablaufs der Dachfunktion in der Software der MCU zu implementieren. Diese Simulation ermöglicht dann bei widersprüchlichen Ergebnissen von MCU und FPGA die Identifizierung des korrekten Ergebnisses. Die Sicherheit bleibt bei diesem Ansatz ohne Einbußen erhalten, da die Funktionsüberwachung im FPGA von den zusätzlichen Modulen zur Erhöhung der Zuverlässigkeit getrennt ist.

6.2.4 Allgemeine Ansätze zur Erhöhung der Zuverlässigkeit

Eine alternative Methode zur Erhöhung der Zuverlässigkeit, die weitgehend unabhängig von der verwendeten Architektur ist, ist die Verwendung von Wiederherstellungspunkten (*Recovery Points*). Anstatt einen Neustart auszuführen, wird hier im Fehlerfall zu einer bestimmten Stelle im Code gesprungen und die Berechnung erneut oder auf eine andere Weise durchgeführt. Dies ist wesentlich schneller als ein kompletter Neustart und erhöht somit die Verfügbarkeit. Der Nachteil ist jedoch in den höheren Anforderungen an die Software-Architektur und an die Entwicklung zu sehen. Dies umfasst die Auswahl der Wiederherstellungspunkte und ein angepasstes Programmdesign, welches einen Wiedereinstieg an diesen Punkten ermöglicht. Diese Methode lohnt sich daher nur, wenn die Wiederherstellung eines geeigneten Zustands wesentlich schneller durchgeführt werden kann als ein Neustart des Systems. Für komplexere Berechnungen ist es zudem möglich, nach dem Sprung zu einem Wiederherstellungspunkt einen alternativen Algorithmus zu verwenden, um so systematische Fehler in der Software tolerieren zu können.

Zur Erhöhung der Zuverlässigkeit eines Systems ist es zudem verbreitet, Fehlerzähler zu nutzen. Wichtige Berechnungen werden innerhalb des erlaubten Zeitfensters mehrfach berechnet bzw. Sensoren häufiger abgetastet. Wenn die Anzahl der Fehler im Zeitfenster unter einer bestimmten Grenze bleibt, werden die Fehler ignoriert. Erst, wenn der Fehlerzähler einen bestimmten Wert erreicht, wird eine Fehlerbehandlung durchgeführt. Diese Vorgehensweise ermöglicht es, transiente Fehler, die sich nicht auf die Sicherheit des Systems auswirken, zu tolerieren.

6.2.5 Abwägung Sicherheit ↔ Zuverlässigkeit

Wie in den vorherigen Punkten schon angedeutet, kann eine Steigerung der Zuverlässigkeit auch Auswirkungen auf die Sicherheit haben. Bei einer Verwendung von Redundanzen in der Sicherheitsfunktion lassen sich folgende Änderungen feststellen:

- Ein 1oo2-System kann sich negativ auf die Sicherheit auswirken, da das System erst deaktiviert, wenn beide Sicherheitsfunktionen dieser Meinung sind. (vgl. Abb. 5.4)
- Ein 2oo3-System steigert neben der Zuverlässigkeit auch die Sicherheit, da der Ausfall einer Funktion das System nicht fälschlicherweise aktivieren kann. (vgl. Abb. 5.5)

Wenn die Ausfallraten der verwendeten Bauteile bekannt sind, lässt sich die Zuverlässigkeit quantifizieren. Diese Daten können dann für einen Vergleich verschiedener Ansätze herangezogen werden. Im Rahmen dieser Studie wurde für die Dual-Core Architektur ein solcher Vergleich zwischen dem ursprünglichen System und einem 3oo4 System durchgeführt (siehe [Siegbert]). Als Ergebnis sinkt der Vorteil der 3oo4 Architektur gegenüber der ursprünglichen Architektur mit besser werdenden Fehlerraten erheblich. Für ein System, dessen Komponenten hohe Fehlerraten aufweisen, bietet ein 3oo4 System also einen großen Vorteil für die Zuverlässigkeit. Hat das System jedoch gute, niedrige Fehlerraten, so steigert sich die Zuverlässigkeit durch diese Architektur nur wenig.

6.2.6 Fazit Zuverlässigkeit

Für beide betrachteten Architekturen lässt sich die Zuverlässigkeit der Anwendung durch die Integration eines zusätzlichen Watchdogs für die Dachfunktion erhöhen. Als Fehlerbehandlung ist entweder ein Neustart des Systems (einfach zu implementieren, aber schlecht für Verfügbarkeit) oder der Rücksprung zu einem Wiederherstellungspunkt (aufwendiger zu implementieren, aber führt zu höherer Verfügbarkeit) möglich.

Im MCU-FPGA Ansatz führt eine Verdopplung oder Verdreifachung der Sicherheitsfunktion im FPGA zu einer Steigerung der Zuverlässigkeit der Sicherheitsfunktion (toleriert Einfachfehler in Überwachungsfunktion). Ebenfalls positiv wirkt sich eine redundante Ausführung der Dachfunktion auf dem FPGA aus. Dies erfordert jedoch einen stark erhöhten Entwicklungsaufwand. Weiterhin erfordern diese Ansätze z.T. erheblich mehr Ressourcen und dieser Mehraufwand ist im Einzelfall gegen den erzielbaren Nutzen abzuwägen

Der Vergleich zwischen Dual-Core und MCU-FPGA lässt zudem vermuten, dass der Dual-Core, auf Grund der Tatsache, dass die Kommunikation zwischen Dachfunktion und Überwachung Chip-intern stattfindet, zuverlässiger ist als ein System, welches von einer SPI Verbindung zwischen diskreten Bauteilen abhängig ist. Lötstellen und Leiterbahnen stellen hier die Schwachstellen dar.

6.3 ASIL Überlegungen

In diesem Abschnitt werden zwei Aspekte bezüglich unterschiedlicher ASIL-Einstufungen betrachtet. Zum einen, wie die Hardware und Software Architektur vereinfacht werden kann (Kostenreduktion), wenn nicht mehr ASIL C, sondern lediglich ASIL B zu erfüllen ist. Zum anderen wird erörtert, welche Maßnahmen im Entwurfsprozess eingespart werden können, wenn nach ASIL B statt nach ASIL C entwickelt wird. In beiden Fällen sind Wechselwirkungen mit der Zuverlässigkeit des Systems zu beachten.

6.3.1 Auswirkungen eines ASIL-Wechsels auf die Architektur

Bei einem System mit ASIL B sind geringere Sicherheitsanforderungen zu erfüllen als bei einem ASIL C System. Wichtige Unterschiede von ASIL B im Vergleich zu ASIL C sind beispielsweise:

- Eine Behandlung von Mehrfachfehlern ist nicht notwendig (*ISO26262-5, 8.4.3.7ff.*).
- Die SPFM sollte mindestens 90% betragen (bei ASIL C: 97%), ist jedoch nur „empfohlen“. Daher ist keine Rechtfertigung bei Nichtanwendung erforderlich.
- Single-Point Fehler (SPF) sind zugelassen, falls die Fehlerrate der entsprechenden Komponente „Occurrence 1“ beträgt (Wenn „Occurrence 1“ nicht gegeben ist, dann müssen Sicherheitsmechanismen implementiert werden, um die entsprechende *Diagnostic Coverage* zu erreichen (*ISO26262-5, 8.4.3.6*))

Auf Grund dieser reduzierten Anforderungen für ASIL B ließe sich das betrachtete Anwendungsbeispiel in diesem Fall vermutlich auf einem Single-Core Mikrocontroller in Kombination mit einem externen time-windowed Watchdog realisieren. Weiterhin sind einige Überwachungsfunktionen zu implementieren, die den Arbeits- und Programmspeicher überprüfen. Eine exemplarisch aufgestellte SPFM erreichte ca. 93% und erfüllt somit die Anforderung für ASIL B (siehe Anhang B).

Im Fall der beiden betrachteten Architekturen kann eine Migration auf eine Single-Core Architektur durchgeführt werden. Im Fall der MCU-FPGA Architektur kann auf die auf dem FPGA realisierte Funktionsüberwachung verzichtet werden, um die Bauteilkosten zu reduzieren. Anstelle der Funktionsüberwachung sind ein externer Watchdog und die oben erwähnten Überwachungsfunktionen einzusetzen. Auch im Fall der Dual-Core Architektur ist für ein ASIL B System eine Migration auf eine Single-Core Architektur möglich, um die Bauteilkosten zu senken. Da der zweite Core nicht einfach entfernt werden kann, ist eine Migration auf einen Single-Core Mikrocontroller mit gleichartigem Core und ähnlichen Peripheriekomponenten erstrebenswert (erleichterte Portierbarkeit der Software). Alternativ bietet sich jedoch auch die Möglichkeit, den Dual-Core Mikrocontroller beizubehalten und den zweiten Core für anderweitige Aufgaben zu nutzen.

6.3.2 Entwicklung ASIL B / ASIL C

Werden zwei ähnliche Produkte entwickelt, von denen eines als ASIL B, das andere jedoch als ASIL C klassifiziert ist, so stellt sich die Frage, ob eine einheitliche Entwicklung beider Produkte nach ASIL C einer Entwicklung nach getrennten Anforderungen vorzuziehen ist. Bei einer getrennten Entwicklung wird das eine Produkt nach ASIL B entwickelt, das zweite nach ASIL C. Um diesen Aspekt näher zu beleuchten wurden beide Szenarien für das gegebene Anwendungsbeispiel in Bezug auf die Aspekte Zuverlässigkeit, Aufwand und Produktkosten untersucht.

In Bezug auf die Zuverlässigkeit ergeben sich in beiden Fällen Einschränkungen durch das Sicherheitskonzept. In dem gegebenen Anwendungsbeispiel ist eine sicherheitskritische Dachbewegung sowohl für eine ASIL B als auch für eine ASIL C Anwendung zu unterbinden. Wie sich dieses Sicherheitskonzept auf die Zuverlässigkeit der Anwendung auswirkt, hängt von der individuellen Implementierung ab und nicht vom gegebenen ASIL.

Der Aufwand für eine getrennte Entwicklung wird durch folgende Faktoren bestimmt. Zum einen sind zwei verschiedene Prozesse zu pflegen (ASIL B und ASIL C Prozess). Dies lässt einen erhöhten Aufwand vermuten. Auf der anderen Seite sind die beiden Prozesse genau an die Anforderungen des jeweiligen Produktes angepasst (kein Overhead im ASIL B Prozess).

Weiterhin sind die Produktkosten optimal, da sowohl für das ASIL B System als auch das ASIL C System eine kostenoptimale Architektur gewählt werden kann.

Bei einer einheitlichen Entwicklung nach ASIL C werden beide Produkte nach ASIL C entwickelt und das ASIL B System an geeigneter Stelle auf die Anforderungen an ASIL B Systeme reduziert, um die Produktkosten gering zu halten. Es bestehen für ein ASIL C System höhere Anforderungen an die Validierung als bei einem ASIL B System, die sich positiv auf die Zuverlässigkeit des ASIL B Systems auswirken könnten. Allerdings ist zu beachten, dass sich diese erhöhten Anforderungen nur auf sicherheitsrelevante Komponenten des Systems beziehen. Weiterhin ermöglicht diese Variante die Anwendung eines einheitlichen Prozesses für beide Produkte, was eine Reduzierung des Aufwands ermöglichen könnte. Andererseits ist eine Reduzierung des ASIL B Systems gegen Ende der Entwicklung nötig, um die Produktkosten dieses Systems niedrig zu halten. Diese Reduzierung erfordert zusätzlichen Aufwand, der von der Anwendung und dem gewählten Sicherheitskonzept abhängt. Wichtig sind in diesem Zusammenhang saubere Schnittstellen in HW und SW, um diejenigen Teile, die für ASIL B nicht benötigt werden, sauber abtrennen zu können.

Zusammenfassend kann gesagt werden, dass sich eine gemeinsame Entwicklung nach ASIL C nur dann lohnt, wenn die beiden Systeme sich sehr stark ähneln (wenig Overhead im ASIL B System durch ASIL C Anforderungen da die meisten Komponenten auch für das ASIL C System entwickelt werden müssen) und die für das ASIL B System nicht benötigten Komponenten mit geringem Aufwand abgetrennt werden können (wenig Overhead im ASIL B System durch Abtrennen).

Es bleibt anzumerken, dass die Empfehlung sehr stark vom Mehraufwand durch den ASIL C Prozess abhängt. Dies betrifft in großem Maße administrative Tätigkeiten wie die Dokumentation aller Arbeitsschritte, die in dieser Studie nicht berücksichtigt wurden. Bei einer Entscheidung für eine getrennte oder gemeinsame Entwicklung sind diese Aspekte jedoch zu berücksichtigen.

6.3.3 Fazit ASIL Überlegungen

Beide implementierten Systeme können ASIL C erfüllen. In diesem Abschnitt wurde zudem untersucht, welche Auswirkungen eine Reduzierung der Sicherheitsanforderungen auf ASIL B hat. Als Ergebnis kann, bei hinreichend guten Ausfallraten der verwendeten Bauteile die Überwachung vermutlich auf einen time-windowed Watchdog in Kombination mit einigen einfachen Software-Überwachungen reduziert werden. Im Fall der MCU+FPGA Architektur kann der FPGA durch einen windowed Watchdog ersetzt werden. Zusätzlich müssen die erwähnten Software-Überwachungen (Speicherüberprüfung etc.) auf der MCU realisiert werden. Im Fall des Dual-Core Mikrocontrollers gibt es zwei Möglichkeiten. Entweder ist die Anwendung auf einen Single-Core Mikrocontroller (mit gleichem Rechnerkern und Peripheriebausteinen) zu portieren oder der zweite Core des Dual-Core Mikrocontrollers kann anderweitig genutzt werden. In beiden Fällen ist weiterhin der time-windowed Watchdog zu verwenden und die benötigten Software-Überwachungen sind zu implementieren. Folglich stellt sich ein Wechsel von ASIL C nach ASIL B bei der MCU-FPGA Architektur als etwas einfacher dar. Weiterhin können bei einem ASIL B System einige Maßnahmen eingeschränkt werden (Metriken, Prozess). Dies betrifft jedoch beide Architekturen gleichermaßen.

Schließlich wurde untersucht in wie weit bei ähnlichen Produkten der Klassen ASIL B und ASIL C einheitlich nach ASIL C entwickelt werden sollte. Dieses Vorgehen erscheint nur für sehr ähnliche Produkte vorteilhaft und die Entscheidung hängt stark vom Mehraufwand für die ASIL C Entwicklung ab.

6.4 Einflüsse auf Testbarkeit

Im Rahmen der Studie wurden Tests mit zwei unterschiedlichen Zielrichtungen durchgeführt. Zum einen wurden Akzeptanztests durchgeführt, welche die im Rahmen der Spezifikation erstellten Akzeptanzkriterien überprüften. Hierbei sollte die Konformität der Beispielimplementierungen mit der erstellten Spezifikation untersucht werden. Bei der anderen Testzielrichtung handelte es sich um Tests zur Überprüfung der Wirksamkeit der implementierten Sicherheitsmechanismen. Als Grundlage dienten hierzu die Verifikationskriterien für die Sicherheitsanforderungen, wie in Abschnitt 4.1.2.8 beschrieben. Diese Tests wurden mit Hilfe von Fehlerinjektion auf der Zielhardware durchgeführt und werden deshalb im Folgenden auch als Fehlerinjektionstests bezeichnet. In den folgenden Abschnitten wird nun die Durchführung der beiden Tests beschrieben und im Anschluss auf die Testbarkeit der untersuchten Architekturen eingegangen.

6.4.1 Durchführung Akzeptanztests

Zur Durchführung der Akzeptanztests wurde die im Rahmen der Studie entwickelte Test- und Simulationsumgebung benutzt (siehe auch Anhang E). Beide Architekturen wurden dabei unter den gleichen Rahmenbedingungen getestet. Lediglich der Anschluss an die Testumgebung, sowie die Installation eines architekturenspezifischen automatischen Resets erfolgte an die einzelnen Plattformen angepasst. Die Testfälle wurden direkt aus den Akzeptanzkriterien abgeleitet und für eine Durchführung mit der Test- und Simulationsumgebung dokumentiert (Black Box Testing). Eine vollständige Beschreibung der durchgeführten Testfälle findet sich im Anhang D.

6.4.2 Durchführung Fehlerinjektionstests

Als Grundlage für die Fehlerinjektionstests dienten die in Anhang A aufgeführten Fehlerbäume. Die dort identifizierten Ereignisse wurden als Basis für die entworfenen Testfälle herangezogen. Hierbei sollten auf der MCU gezielt Zustände injiziert werden, die ein solches Ereignis zur Folge haben. Entsprechende Testfälle wurden für alle Komponenten der MCU entworfen, um so einen Rückschluss auf zufällige Hardwarefehler zu ermöglichen. Da beide Architekturen auf dasselbe funktionale Sicherheitskonzept zurückgreifen, sind die durch die Sicherheitsfunktion überprüften Rahmenbedingungen und somit die erstellten Testfälle in beiden Ansätzen gleich. Im vorliegenden Fall wird der Status der CAN-Nachrichten (Geschwindigkeit/Zündung) sowie des Bedienelements geprüft. Die Freischaltung der Hydraulikpumpe erfolgt nur, wenn die Geschwindigkeit Null, die Zündung eingeschaltet und nur eine Seite des Bedienelements betätigt ist. Softwarefehler werden durch die Überwachungsfunktion abgedeckt, wenn sie zu einer Verletzung der dargestellten Rahmenbedingungen führen. Da der Unterschied der beiden Architekturen im Vordergrund stand und davon auszugehen ist, dass ähnliche Softwarefehler gefunden werden können, wurde der Einfluss der Sicherheitsfunktion auf diese nicht näher betrachtet. Es sollte geprüft werden, ob es Unterschiede in der Fehlerbehandlung zwischen den zwei gewählten Ansätzen gibt. Die Fehler wurden durch das Einfügen von zusätzlichem Programmcode, welcher über einen externen Interrupt aufrufbar ist, in der MCU der Dachfunktion simuliert oder – falls möglich – durch die gezielte Manipulation äußerer Einflüsse erzielt. Es wurden Tests für Fehler in folgenden Komponenten der MCU entworfen: Programmspeicher (Flash), RAM, I/O Unit, CPU, CAN-Controller, Taktgeber (Clock), Spannungsversorgung. Eine Auflistung der konzipierten Testfälle findet sich im Anhang D.

Stück-At Tests für Komponenten innerhalb der MCU, die Tests der Clock und ein Überspannungstest sind nicht durchgeführt worden und werden in der Übersicht lediglich der Vollständigkeit halber aufgeführt. Eine Fehlerinjektion nach dem oben bezeichneten Konzept innerhalb der MCU (Ausführung von Fremdcode durch externe Interrupts) ist lediglich in der Lage, einmalig ein Fehlerereignis bei der Ausführung herbeizuführen, kann jedoch nicht

garantieren, dass das Fehlerszenario erhalten bleibt. Ein Stuck-At Test würde somit die Möglichkeiten des gewählten Testverfahrens überschreiten. Eine Option für fortführende Tests besteht in der Nutzung der für beide Architekturen vorhandenen On-Chip-Debug Systeme. Durch die Nutzung von Datenbreakpoints an den zu prüfenden Adressbereichen ist es so möglich, die Werte nach jeder Änderung durch das Originalprogramm erneut zu verfälschen. Hierbei ist jedoch eine Automatisierung notwendig, um den zeitlichen Overhead nicht zu groß werden zu lassen, weil sonst die Echtzeitfähigkeit des Systems verloren geht. Es wurde jedoch angenommen, dass im Fall von positiven Testergebnissen der durchgeführten Tests die aufwendige Simulation von Stuck-At Fehlern keine neuen Informationen über die Systemzuverlässigkeit bringt. Stuck-At Fehler würden lediglich zu einem andauernden Systemversagen führen, welches jedoch stichprobenartig bereits durch die ausgeführten Testfälle überprüft wird. Fehler in der Systemclock sowie der Fehlertyp *Überspannung* wurden aus Kostengründen nicht ausführt, da eine Beschädigung der verwendeten Evaluationsboards wegen der notwendigen Hardwareeingriffe recht wahrscheinlich ist. Es ist zu erwarten, dass Fehler in der Clock entweder durch einen time-windowed Watchdog gefunden werden können oder ähnliche Auswirkungen besitzen wie der Ausfall der Interrupts (Ausfall des Systemzeitgebers der Dachfunktion). Zur Vervollständigung der Ergebnisse müssten die ausgelassenen Testfälle noch ausgeführt werden, worauf jedoch im Rahmen dieser Studie wegen der schlechten Kosten/Nutzen Relation verzichtet wurde.

6.4.3 Bewertung der Testbarkeit

Der Aufwand der Testdurchführung für die Akzeptanztests war für beide Architekturen in etwa ausgeglichen. Kleinere Unterschiede werden als Initialaufwand, bedingt durch die Einarbeitung in die verwendeten Systeme eingeschätzt und lassen keinen konkreten Rückschluss auf den architekturenspezifischen Aufwand zu. Hierbei handelt es sich um das Anschließen des zu testenden Geräts an die Testumgebung, sowie die Konfiguration der automatischen Resets, um vor dem Beginn eines neuen Tests das System wieder in einen klar definierten Ausgangszustand zu bringen. Die Tests konnten ausschließlich mit der Test- und Simulationsumgebung durchgeführt werden, die ein von der Zielhardware unabhängiges Testen ermöglicht (Black Box Test).

Im Rahmen der Fehlerinjektionstests konnte die Wirksamkeit der Sicherheitsmechanismen für die in Anhang D beschriebenen Testfälle erfolgreich gezeigt werden. Hierbei ergaben sich lediglich geringe architekturbedingte Unterschiede. Darunter ist hervorzuheben, dass der FPGA systembedingt über eine schnellere Reaktionszeit auf Ereignisse verfügt, die an seinen Eingängen anliegen und unmittelbar von der Logik verarbeitet werden können. Im vorliegenden Beispiel kann dies jedoch nicht als direkter Vorteil gewertet werden, weil die festgesetzte Fault Tolerant Time Span so hoch lag, dass auch die Dual-Core MCU Architektur bedeutend schneller als notwendig reagieren konnte. In beiden Fällen konnte beobachtet werden, dass das Einfügen von transienten Fehlern in den RAM oder Register der CPU in vielen Fällen durch das zu testende System toleriert wird. Bedingt durch die Implementierung der Dachfunktion werden die Werte im RAM so oft überschrieben, dass es nur extrem selten zu wahrnehmbaren Auswirkungen kam. Eine Auswirkung von in die MCU Komponenten injizierten transienten Fehlern konnte im Rahmen der durchgeführten Tests nur dann beobachtet werden, wenn Statusregister der CPU verändert wurden. Die Überwachungsfunktion beider Architekturen reagierte erwartungsgemäß auf einen Ausfall der Dachfunktion (z.B. durch blockierte Interrupts) durch Abschalten der Hydraulikpumpe nach Ablauf der spezifizierten Zeitperioden. Aufgrund des stark begrenzten Eingabewertebereichs der Sicherheitsfunktion ist davon auszugehen, dass die Ergebnisse sich auf das Systemverhalten verallgemeinern lassen. Die Sicherheitsfunktion überprüft Fehler im Bedienelement, sowie den zwei CAN Nachrichten, wobei für die CAN Nachrichten die Fehlermodi „Zeitüber-

schreitung“, „Nachrichtenzähler inkorrekt“, und „Nachricht inkorrekt“ zu unterscheiden sind. Es wird deshalb davon ausgegangen, dass die durchgeführten Tests aller einzelnen Fehlerursachen bereits eine hohe Aussagekraft für das Gesamtsystem besitzt.

Um die Testabdeckung zu erhöhen würde es sich empfehlen, einen automatisierten Test aller möglichen Eingabewerte der Überwachungsfunktion durchzuführen, um Fehler aufzudecken, die durch Seiteneffekte verursacht werden. Im Rahmen dieser Studie wurde darauf jedoch verzichtet, da das Risiko von Seiteneffekten aufgrund der einfachen Struktur und Umsetzung als gering eingestuft wird. Die durchgeführten Tests orientierten sich deshalb an möglichen Fehlerszenarien im Steuergerät unter Beachtung der Testabdeckung für jede Einzelfehlerursache.

6.4.4 Fazit Testbarkeit

Die Testbarkeit profitiert im Falle der hier gewählten Funktionsüberwachung von deren einfachem Aufbau. So ist es möglich, mit einer überschaubaren Anzahl von Testfällen (Testen der Einhaltung von Rahmenbedingungen) bereits eine hohe Abdeckung des Eingabewertebereichs zu erzielen. Dies kann entweder getrennt von der zu überwachenden Anwendung geschehen oder ist direkt im Verbund möglich. Für letzteres ist es möglicherweise notwendig, eine Fehlerinjektion auf dem Steuergerät, sowie mechanische Eingriffe am Aufbau (z.B. Abtrennen der Systemclock) durchzuführen. Im Gegensatz zum getrennten Testen kann jedoch einerseits ein getrennter Testaufbau für die Überwachungseinheit entfallen und andererseits eine direkte Nachvollziehbarkeit der Systemreaktion auf Fehler in der zu überwachenden MCU erreicht werden.

Im Gegensatz dazu erscheint das Testen von generischen Ansätzen (z.B. Error Correction Codes für Speicher) vergleichsweise kompliziert. Sollten durch die Hardware keine speziellen Testroutinen zur Verfügung gestellt werden, ist lediglich eine indirekte Fehlerinjektion (z.B. durch Bestrahlung der MCU) möglich. Aufgrund der bereits oben dargestellten Situation, dass sich viele Fehler gar nicht auswirken, da die fehlerhaften Werte gar nicht mehr gelesen werden, ist die Injektion von sehr vielen Fehlern notwendig und eine Nachvollziehbarkeit der Fehlerursache schwierig oder gar nicht möglich. Es kann lediglich das Gesamtverhalten bei unbestimmter Fehlerinjektion getestet werden.

Abschließend ist festzuhalten, dass die Testergebnisse den Schluss zulassen, dass eine Funktionsüberwachung generell geeignet ist, um ein System wie die Cabriovertdecksteuerung angemessen zu überwachen. Die Fehlerinjektionstests zeigten dies im Fall simulierter Hardwarefehler in der ECU der Dachfunktion. Im Fall der Dual-Core MCU Architektur ist jedoch noch gesondert der Nachweis zu erbringen, dass Fehlerfälle des ersten Cores – wie Überhitzung – den zweiten Core nur derart beeinflussen können, dass dieser entweder weiter korrekt arbeitet oder sein Fehlverhalten durch einen externen Standard-Watchdog erkannt werden kann (Totalausfall).

6.5 Einflüsse auf Änderbarkeit

Um die Flexibilität der entworfenen Architekturansätze zu untersuchen, werden im Weiteren ausgewählte Änderungsszenarien betrachtet. Hierbei werden die Erweiterbarkeit der Architekturen in Bezug auf eventuelle Änderungen an der Umgebung des Anwendungsbeispiels und der Aufwand für eine Übertragung auf ein komplett anderes Anwendungsgebiet im Automobil diskutiert.

Folgende sechs Änderungsszenarien wurden diskutiert:

- Integration eines Regensensors mit CAN Ausgabe auf dem ersten Core des Dual-Core Mikrocontrollers bzw. dem Mikrocontroller in der MCU+FPGA Architektur.
- Integration eines Regensensors mit CAN Ausgabe auf dem zweiten Core des Dual-Core Mikrocontrollers bzw. dem FPGA in der MCU+FPGA Architektur.
- Eine fehlerhafte Verriegelung des Daches im geschlossenen Zustand kann nun auch das Sicherheitsziel A verletzen (Verriegelung muss sichergestellt werden).
- Es soll ein Einklemmschutz derart realisiert werden, dass das Dach angehalten wird, wenn das Drehmoment einen bestimmten Schwellwert überschreitet.
- Das Dachsteuergerät inkl. des realisierten Sicherheitskonzeptes soll modifiziert werden, um es als Türsteuergerät zu verwenden (Fensterheber, Hazard: Einklemmen).
- Das Dachsteuergerät incl. des realisierten Sicherheitskonzeptes soll modifiziert werden, um es in einem Fahrerassistenzsystem „Hindernisausweichen“ zu verwenden (Crash Avoidance).

Der Änderungsaufwand wurde jeweils abgeschätzt und die Ergebnisse in Tabelle 5.1 dargestellt:

Szenario	Aufwand
Regensensor (1st Core / MCU)	Gering
Regensensor (2nd Core / FPGA)	Gering – Mittel (CAN Kommunikation muss mit Core 1 / MCU abgestimmt werden)
Sicherstellung Verriegelung	Gering (weiteren Sensor mit ins Sicherheitskonzept aufnehmen, Warnmeldung an Fahrer)
Einklemmschutz	Gering - Mittel (weiteren Sensor für Drehmomentmessung mit ins Sicherheitskonzept aufnehmen, höherer Aufwand wenn Schaltschwelle an Alterung des Daches angepasst werden soll)
Übertragung auf Türsteuergerät	Sicherheitsfunktion: Gering (andere Sensoren und Aktuatoren, Konzept kann aber beibehalten werden)
Übertragung auf Fahrerassistenzsystem	Sicherheitsfunktion: Hoch , da komplexe Sensorauswertung und Einschätzung des Systemzustandes erforderlich ist

Tabelle 5.1: Aufwandsabschätzung für ausgewählte Änderungsszenarien

6.5.1 Fazit Änderbarkeit

Die wesentliche Aufgabe der Überwachungsfunktion in den gewählten Architekturansätzen besteht in der statischen Überprüfung einer überschaubaren Menge von einfach auszuwertenden Rahmenbedingungen. Hierbei handelt es sich um digitale Sensorwerte und eine begrenzte Anzahl von CAN Nachrichten, die durch einfache Timeout-Kriterien abgesichert sind. Somit kann eine weitestgehend gedächtnislose Entscheidung über die Korrektheit des Systemzustandes getroffen werden. Weiterhin kann das System durch eine einfache Abschaltung sicherheitskritischer Aktuatoren (hier: Hydraulikpumpe) in einen sicheren Zustand gebracht werden. Die implementierten Architekturen wurden beide entsprechend dieses Konzepts umgesetzt.

Solange Änderungen im Rahmen des oben beschriebenen Grundprinzips bleiben (das Modell für die Verletzung der Sicherheitsanforderungen ist einfach und benötigte Informationen sind leicht zugänglich/messbar und der sichere Zustand kann durch Deaktivieren eines Aktuators/mehrerer Aktuatoren erreicht werden), lassen sie sich vergleichsweise einfach umsetzen.

Änderungen, die jedoch eine starke Vergrößerung des Logikaufwands in der Überwachungsfunktion mit sich bringen würden, bedeuten wohl in den meisten Fällen eine komplette Neukonzeption der Implementierung der Überwachungsfunktion. Ein Beispiel dafür wäre ein komplexes Modell für die Verletzung von Sicherheitsanforderungen, welches auch Daten aus der Vergangenheit mit einbezieht oder versucht das Systemverhalten voraus zu berechnen (Beispiel Motorsteuerung). Dies gilt in ähnlicher Art und Weise, wenn die Komplexität der Sensordaten steigt (analoge Auswertung / mehr CAN Nachrichten) oder ein Notfallsystembetrieb unterstützt werden muss, der nicht lediglich durch das Abschalten des oder der Aktuatoren zu erreichen ist. Auch wenn in diesen Fällen die Umsetzung des Sicherheitskonzepts durch eine Funktionsüberwachung möglich ist, bedarf es wohl großer Änderungen einiger grundlegender Ansätze im Vergleich zu den hier erarbeiteten Architekturen.

6.6 Vergleich mit alternativen Sicherheitsansätzen

Wie in Kapitel 4.4 vorgestellt, gibt es neben den Architekturen, auf denen das Anwendungsbeispiel implementiert wurde, noch weitere Architekturen, mit denen sicherheitskritische Funktionen realisiert werden können. Diese Ansätze wurden in der Arbeit [Beckschulze] untersucht. Grundsätzlich können zwei verschiedene Konzepte angewendet werden. Für das Anwendungsbeispiel Cabriooverdecksteuerung wurde das Konzept der Funktionsüberwachung auf zwei verschiedenen Hardware-Architekturen realisiert. Alternativ kann funktionale Sicherheit durch eine Überwachung der Bauteile erreicht werden, die für die Implementierung der Anwendung benötigt werden und zudem sicherheitsrelevant sind. Da dieser Ansatz weitgehend unabhängig von der Anwendung ist, wird er als generisch bezeichnet. Im Folgenden werden die beiden Konzepte mit ihren Vor- und Nachteilen losgelöst vom Anwendungsbeispiel vorgestellt.

6.6.1 Funktionsüberwachung

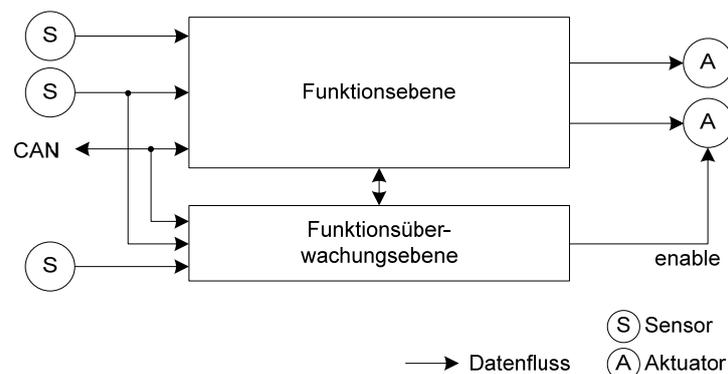


Abbildung 6-5: Konzept einer Funktionsüberwachung

Das Konzept der Funktionsüberwachung (siehe Abbildung 6-5: Konzept einer Funktionsüberwachung) ist ein Zwei-Ebenen-Konzept. Die erste Ebene bildet die Funktionsebene, welche alle Funktionen umfasst, die in der Systemspezifikation gefordert werden. Dazu verfügt die Funktionsebene über eine Menge von Eingangsgrößen (Sensorwerte, Nachrichten über Bussysteme) und eine Menge von Ausgangsgrößen (Aktuatorenwerte, Nachrichten über Bussysteme). Die zweite Ebene wird als Funktionsüberwachungsebene bezeichnet. Sie beruht auf einem vereinfachten Modell der sicherheitsrelevanten Teile des Systems. Dazu verfügt die Funktionsüberwachungsebene über redundante Eingangsgrößen und Sensorwerte von möglicherweise vorhandenen redundanten Sensoren. Darauf basierend wird ein zulässiger Systemzustand ermittelt. Abhängig von der

Anwendung kann daraufhin ein sicherheitskritischer Aktuator entweder unabhängig von der Funktionsebene freigeschaltet werden oder erst nach Abgleich mit dem von der Funktionsebene berechneten Systemzustand. Um dementsprechend reagieren zu können, ist in der Funktionsüberwachungsebene eine Ausgangsgröße vorgesehen.

6.6.2 Bauteilüberwachung

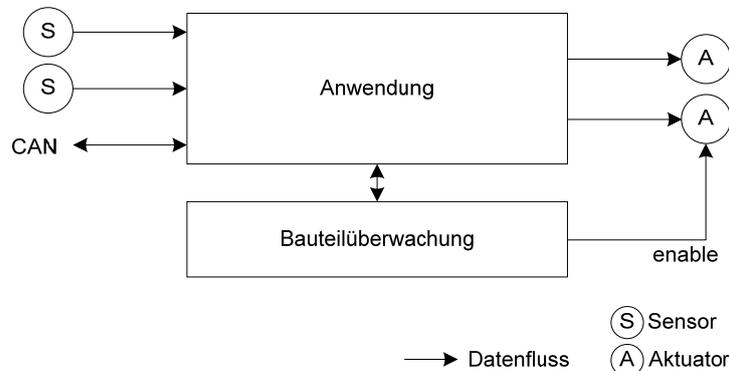


Abbildung 6-6: Konzept einer Bauteilüberwachung

Das Konzept der Bauteilüberwachung ist in Abbildung 6-6: Konzept einer Bauteilüberwachung dargestellt. Informationen über die Anwendungen werden bei diesem Konzept lediglich dafür benötigt, die sicherheitsrelevanten Komponenten festzustellen, welche überwacht werden müssen. Die Bauteilüberwachung verfügt ebenfalls über eine Ausgangsgröße mit der die Aktivierung sicherheitsrelevanter Aktuatoren verhindert werden kann. Davon wird Gebrauch gemacht, falls die Bauteilüberwachung einen Fehler entdeckt.

6.6.3 Funktionsüberwachung vs. Bauteilüberwachung

Während bei der Implementierung einer Funktionsanwendung für das Anwendungsbeispiel auf zwei verschiedenen Hardware-Architekturen keine nennenswerten Unterschiede festgestellt werden konnten, weisen die in Abbildung 6-5 und Abbildung 6-6 vorgestellten Konzepte zahlreiche Unterschiede auf. Ein Vergleich dieser Konzepte für Dual-Core Mikrocontroller wird beispielsweise in der Veröffentlichung [ISoLA08] durchgeführt.

Die Unterschiede betreffen neben der Sicherheit weitere Aspekte. In der Arbeit [Beckschulze] wurden die Auswirkungen der Sicherheitsmechanismen auf Zuverlässigkeit, Änderbarkeit und Kosten untersucht. Die Kosten wurden aufgeschlüsselt in den Speicher-Overhead, den Laufzeit-Overhead und die Kosten die durch den Logik-Overhead entstehen. Im Folgenden wird auf diese Aspekte im Einzelnen eingegangen.

6.6.3.1 Sicherheit

Die durch ISO WD 26262 gestellten Anforderungen, die das Risiko, das durch das System entsteht, auf ein akzeptables Niveau reduzieren sollen, können mit beiden Konzepten erfüllt werden. Jedoch ist der Aufwand, der dazu notwendig ist, unterschiedlich. Das Konzept der Funktionsüberwachung kann anschaulich verstanden werden kann. Falls die Sicherheitsfunktion einfach ist, ist die Funktionsüberwachung gut umzusetzen und auch der Nachweis der Sicherheit ist in diesem Fall vergleichsweise einfach. Während für eine Architektur mit Funktionsüberwachung auf Komponentenebene nur die Behandlung von Common Cause Failures notwendig ist, muss bei dem generischen Ansatz für jede Überwachungsfunktion einer Komponente gezeigt werden, dass diese einen hinreichend hohen Anteil der Ausfälle erkennt.

Im Gegensatz zu einer Funktionsüberwachung, die Fehler auf Systemebene erkennt, ist es beim generischen Ansatz auf Komponentenebene nicht möglich, zu entscheiden, ob ein Fehler sicherheitsrelevante Auswirkungen hat. Für die Komponente Speicher können zwar die Sicherheitsmechanismen auf die sicherheitsrelevanten Teile des Speichers beschränkt werden, es findet jedoch keine inhaltliche Bewertung der Fehler statt. Fehler in niederwertigen Bits von sicherheitskritischen Variablen können beispielsweise tolerierbare Änderungen hervorrufen, durch die kein sicherheitskritischer Zustand entsteht.

Durch die Fehlererkennung auf Systemebene werden bei Verwendung einer Funktionsüberwachung auch sicherheitskritische Softwarefehler erkannt. Diese Erkennung funktioniert zuverlässig, wenn die Funktionsüberwachung durch einen anderen Rechner als die Applikation ausgeführt wird, so dass sich Softwarefehler in der Funktionsebene nicht auf die Funktionsüberwachungsebene auswirken können. Eine zusätzliche Funktionsüberwachung, realisiert als periodisch aufgerufener Software-Task, kann aber auch in generischen Architekturen zusätzlich zu einer Bauteilüberwachung implementiert werden, um sicherheitskritische Softwarefehler abzudecken.

Ein weiterer Unterschied zwischen den beiden Konzepten ist die Bedeutung der Sicherheitsanalyse. Während für generische Architekturen nur die Menge der sicherheitsrelevanten Komponenten von Interesse ist, bildet die Sicherheitsanalyse die Basis einer Funktionsüberwachung. Nicht identifizierte Hazards können deswegen schon nach Auftreten eines einzelnen Fehlers eintreten. Die umfangreichere Fehlerabdeckung einer generischen Architektur dagegen erkennt und behandelt einige dieser Fehler auch ohne dass sie als sicherheitskritisch identifiziert wurden.

6.6.3.2 Zuverlässigkeit

Sicherheitsmechanismen können die Zuverlässigkeit des Systems herabsetzen, wenn unnötig häufig in einen sicheren Zustand gewechselt wird, so dass die Verfügbarkeit des Systems verringert wird. Nachteilige Auswirkungen auf die Zuverlässigkeit des Systems durch die implementierten Sicherheitsmechanismen sind bei der Implementierung beider Konzepte möglich.

Wie zuvor diskutiert können so genannte False Positives - durch die Sicherheitsfunktion fälschlicherweise festgestellte Fehler - die Zuverlässigkeit verringern. Die in Abschnitt 6.2 vorgestellten Ansätze, die Zuverlässigkeit der Sicherheitsfunktion durch deren redundante Auslegung zu erhöhen, helfen allerdings nur gegen transiente Fehler in der Funktionsüberwachung. Die Zuverlässigkeit kann aber auch durch einen systematischen Fehler beim Entwurf der Funktionsüberwachung beeinträchtigt werden. Dies kann dann der Fall sein, wenn die Funktionsüberwachung zu Ergebnissen kommt, die stärker von den Ergebnissen der Funktionsebene abweichen als die definierten Toleranzbereiche zulassen. In manchen Anwendungen kann deswegen die Herausforderung darin bestehen, einen Kompromiss zwischen Sicherheit und Zuverlässigkeit zu finden. Möglicherweise wird die Beeinträchtigung der Zuverlässigkeit aber erst festgestellt, wenn das System bereits eingesetzt wird.

Es ist anzunehmen, dass in generischen Architekturen häufiger Fehler entdeckt werden, da die umfassende Fehlererkennung von Komponentenfehlern mehr Redundanz erfordert als bei einer Funktionsüberwachung. Weiterhin kann auf Komponentenebene in der Regel nicht festgestellt werden, ob ein Fehler sicherheitsbezogen ist. Dies erfordert eine Behandlung aller erkannten Fehler. Damit im Vergleich zu einer Funktionsüberwachung die häufiger vorkommende Fehlerbehandlung nicht die Verfügbarkeit des Systems herabsetzt, ist es erstrebenswert, Techniken zu nutzen, bei denen eine schnelle Wiederherstellung (*recovery*) möglich ist. Im Gegensatz zur Funktionsüberwachung, in der nur sicherheitsbezogene Fehler erkannt und behandelt werden, besteht durch effiziente Bauteilüberwachung mit Wiederherstellungstechniken auch die Möglichkeit, die Gesamtzuverlässigkeit des Systems zu erhöhen.

Inbesondere kann die Zuverlässigkeit durch Techniken erhöht werden, die effizient Fehler korrigieren können (z.B. Error Correction Codes für Speicher).

6.6.3.3 Änderbarkeit

Wie bereits in Abschnitt 6.5.1 diskutiert, hängt die Änderbarkeit einer Architektur mit Funktionsüberwachung von deren Komplexität in der konkreten Anwendung ab. Ist die Überwachungsfunktion einfach, so ist anzunehmen, dass Änderungen mit geringem Aufwand vorgenommen werden können. Da die Abhängigkeit von der Anwendung für Architekturen mit Bauteilüberwachung nicht existiert, müssen dort nur geringe Anpassungen vorgenommen werden, falls Änderungen an der Spezifikation vorgenommen werden oder die Hardware-Architektur für eine andere Anwendung genutzt werden soll. Bei einer Wiederverwendung der Hardware-Architektur kann zudem der Umfang der Tests der Sicherheitsfunktionen eingeschränkt werden.

6.6.3.4 Speicher-Overhead

Der Speicher-Overhead der verschiedenen Hardware-Architekturen lässt sich nicht unabhängig von der Anwendung vergleichen. Bei Architekturen mit Funktionsüberwachung hängt der Speicher-Overhead von der Menge der Variablen ab, welche die Funktionsüberwachung benötigt. Im Fall der Cabrioüberdecksteuerung benötigt die Funktionsüberwachung sehr wenig Speicher. Im Allgemeinen lässt sich jedoch nicht sagen, dass die Funktionsüberwachung am wenigsten zusätzlichen Speicher benötigt, da sicherheitskritische Variablen verdoppelt werden und nicht nur wenige redundante Bits verwendet werden. Erfolgt in einer generischen Architektur eine Einschränkung der Redundanz im Speicher auf die sicherheitskritischen Teile, so kann der Speicher-Overhead minimal gehalten werden, mit dem Nachteil, dass die Unabhängigkeit von der Anwendung eingebüßt wird.

6.6.3.5 Laufzeit-Overhead

Ein Laufzeit-Overhead kann entstehen, wenn der Prozessor zusätzlich zu den Funktionen der Applikation Funktionen ausführen muss, welche der funktionalen Sicherheit dienen. Dies kann problematisch sein, falls die Einhaltung der Echtzeitanforderungen nicht mehr garantiert werden kann und ein schnellerer Prozessor notwendig ist als ursprünglich vorgesehen. Durch zusätzliche Logik (Hardwarekomponenten) kann ein Laufzeit-Overhead vermieden werden und gleichzeitig eine höhere Fehlerabdeckung erreicht werden (Vermeidung von Common Cause Failures). Jedoch steigen auch die Kosten, wenn zusätzliche Hardwarekomponenten benötigt werden.

6.6.3.6 Kosten durch Logik-Overhead

Der Logik-Overhead soll dahingehend bewertet werden, welche Kosten durch ihn verursacht werden. Hardware, die speziell auf eine Anwendung oder speziell auf sicherheitskritische Anwendungen zugeschnitten ist, verursacht höhere Kosten als Standardgeräte, die vielseitig eingesetzt werden. Es ist festzustellen, dass generische Architekturen auf Spezialhardware angewiesen sind, um eine ausreichend hohe Fehlerabdeckung zu erreichen. Dagegen stellt eine Funktionsüberwachung keine besonderen Ansprüche an die Hardware und kann deshalb auch auf Standardhardware realisiert werden.

6.6.4 Vergleich verschiedener Architekturen

In der Arbeit [Beckschulze] wurden verschiedene Hardware-Architekturen untersucht und miteinander verglichen. Für den grafischen Vergleich wurde für jede untersuchte Hardware-Architektur ein Kiviatdiagramm aufgestellt, in dem jeder der zuvor diskutierten Aspekte

klassifiziert wird. Kiviatdiagramme wurden bereits in [Koopman] zur Bewertung verschiedener sicherheitskritischer Systeme eingesetzt. Dort wurden jedoch andere Parameter ausgewertet.

Es folgt zunächst eine kurze Beschreibung der betrachteten Hardware-Architekturen.

Architekturen mit Funktionsüberwachung

1. Single-Core MCU mit ASIC (E-Gas-Überwachungskonzept)

In diesem Ansatz werden die zwei Ebenen der Funktionsüberwachung auf einer Single-Core MCU implementiert. Zur Feststellung von Common Cause Failures dient eine dritte Ebene, die getriggert durch einen ASIC mit dem Mikrocontroller eine Frage-Antwort-Kommunikation und Speichertests durchführt.

2. Dual-Core Mikrocontroller + time-windowed Watchdog

Funktionsüberwachung gemäß der Implementierung [Siegbert]

3. Single-Core Mikrocontroller + FPGA

Funktionsüberwachung gemäß der Implementierung [Boymanns]

4. Single-Core Mikrocontroller + Watchdog

In dieser Architektur soll das Drei-Ebenen-Konzept auf einem Single-Core Mikrocontroller implementiert werden. Da die dritte Ebene nicht alle Hardware-Fehler entdecken kann, wird zusätzlich ein time-windowed Watchdog getriggert.

Generische Architekturen

5. Dual-Core Lockstep Architektur

Die zentrale Idee der Lockstep Architektur besteht darin, beide Prozessoren das selbe Programm ausführen zu lassen und die Ausgangsgrößen kontinuierlich zu vergleichen, so dass sich bei Ungleichheit auf einen Fehler schließen lässt. Für die anderen sicherheitsbezogenen Komponenten müssen weitere Sicherheitsmechanismen implementiert werden. Diese sind der Delphi Secured Microcontroller Architecture [Fruehling] entnommen.

6. MCU + Watchdog-Prozessor

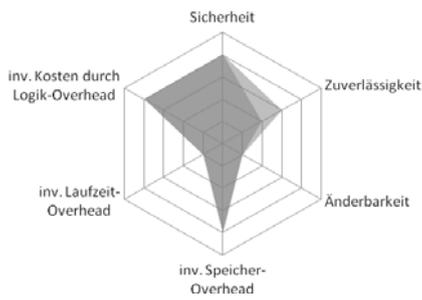
Die Grundidee einer Architektur mit Watchdog-Prozessor, angelehnt an [Benso], ist es, durch Überwachung des Busses Fehler zu entdecken. Prozessorfehler sollen durch die Überprüfung von Signaturen erkannt werden, Speicherfehler durch den Vergleich mit Schattenvariablen.

7. Single-Core Mikrocontroller + Überwacher (Yogitech)

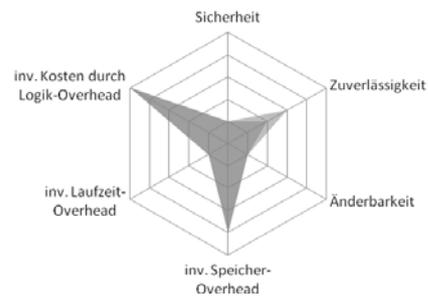
Dieser Ansatz ist ein System-on-Chip, ein System, bei dem alle Komponenten auf einem Silizium integriert sind. Die funktionale Sicherheit wird durch Konfiguration und Programmierung von Hardware- und Software-IPs gewährleistet. Das Konzept stammt von der Firma Yogitech Spa und ist [Mariani] entnommen.

8. Zwei Single-Core Mikrocontroller

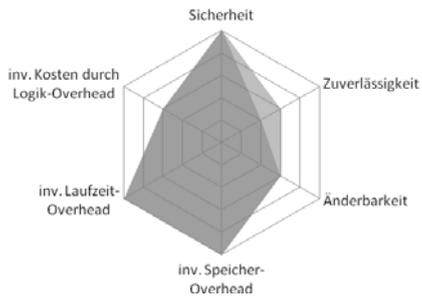
Diese Architektur beschreibt den naiven Ansatz, den ganzen Mikrocontroller zu verdoppeln und sicherheitskritische Ausgangsgrößen durch Komparatoren zu vergleichen.



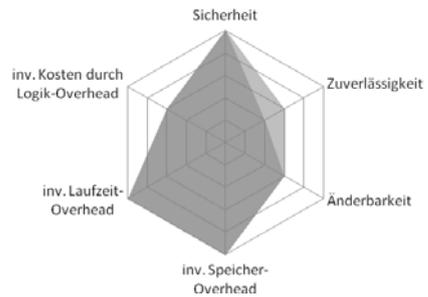
Single-Core MCU mit ASIC



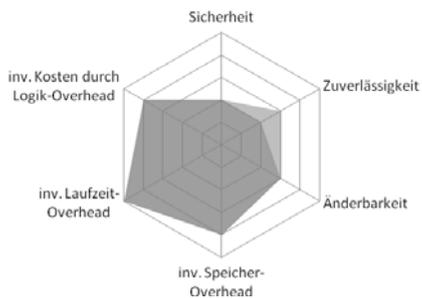
Single-Core Mikrocontroller + Watchdog



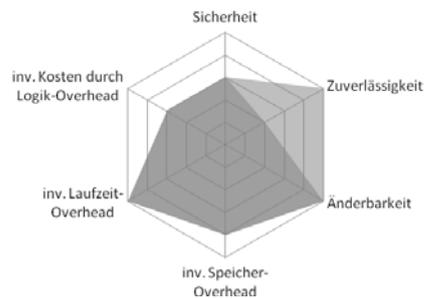
Dual-Core MCU + Watchdog



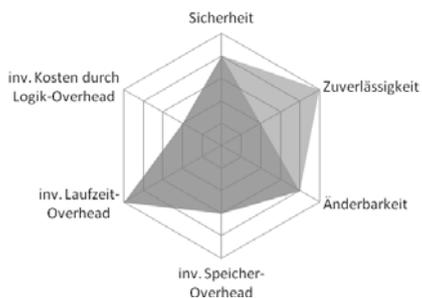
MCU + FPGA



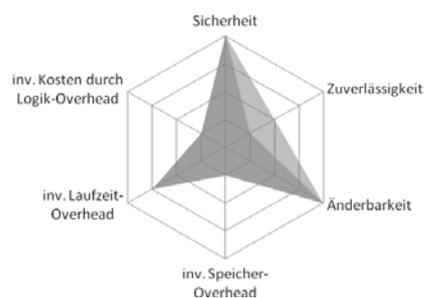
MCU + Watchdog-Prozessor



Dual-Core Lockstep



MCU + Überwacher



2 MCUs

Abbildung 6-7: Grafischer Vergleich

Grafischer Vergleich

Die in den Kiviatdiagrammen (siehe Abbildung 6-7: Grafischer Vergleich) aufgetragenen Aspekte werden jeweils in fünf Klassen von „--“ bis „++“ eingeteilt. Dabei steht „++“ für hohe Sicherheit, Zuverlässigkeit und Änderbarkeit und für geringen Overhead. Für Zuverläss-

sicherheit wird ein Intervall angegeben, das die Abhängigkeit von der Applikation bzw. von der Effizienz der Fehlerbehebung berücksichtigt.

Eine kurze Begründung zur Klassifizierung der betrachteten Architekturen in Bezug auf die untersuchten Vergleichsaspekte ist in Abbildung 5-9 zu finden.

	MCU + ASIC (E-GAS)	Single Core + time-windowed WD	Dual Core + Watchdog	MCU + FPGA
	+	--	++	++
Sicherheit	bei sorgfältigem Zeitmanagement können Anforderungen erfüllt werden	nicht ausreichend Unabhängigkeit zwischen Funktion und Überwachung	hohe Abdeckung aller sicherheitskritischen HW+SW-Fehler	hohe Abdeckung aller sicherheitskritischen HW+SW-Fehler
Zuverlässigkeit	-,0 abhängig von Güte der Funktionsüberwachung und Wiederherstellung	-,0 abhängig von Güte der Funktionsüberwachung und Wiederherstellung	-,0 abhängig von Güte der Funktionsüberwachung und Wiederherstellung	-,0 abhängig von Güte der Funktionsüberwachung und Wiederherstellung
Änderbarkeit	-- Verifikation der Echtzeitanforderung erneut nötig, Anpassung der Funktionsüberwachung und Frage-Antwort-Kommunikation	-- Verifikation der Echtzeitanforderungen erneut nötig, Anpassung der Funktionsüberwachung, Anpassung der WD-Triggerung	o Anpassung der Funktionsüberwachung	o Anpassung der Funktionsüberwachung
inv. Speicher Overhead	+ Overhead gering, Variablen der Funktionsüberwachung werden doppelt gespeichert	+ Overhead gering, Variablen der Funktionsüberwachung werden doppelt gespeichert	++ durch Funktionsüberwachung so gering wie möglich	++ durch Funktionsüberwachung so gering wie möglich
inv. Laufzeit Overhead	-- MCU führt Funktion, Funktionsüberwachung und Antwort-Kommunikation durch	-- MCU führt Funktion, Funktionsüberwachung, Selbsttests und WD-Triggerung durch	++ getrennte HW von Funktion und Funktionsüberwachung	++ getrennte HW von Funktion und Funktionsüberwachung
inv. Kosten durch Logik-Overhead	+ Kosten durch ASIC	++ Kosten so gering wie möglich	o möglicherweise bald Standardhardware	o akzeptabel, FPGA-Größe mitentscheidend
	Single Core + WD-Prozessor + time-windowed WD	Dual Core lockstep "Delphi SMA"	MCU + Überwacher "Yogitech"	2 MCUs
Sicherheit	- Kontrollflussalgorithmus kann nicht alle Prozessorfehler finden	o Parität für RAM hat zu geringe Fehlerabdeckung	+ hohe Abdeckung aller sicherheitskritischen HW-Fehler	+ hohe Abdeckung aller sicherheitskritischen HW-Fehler
Zuverlässigkeit	-,0 Wiederherstellungsmöglichkeit (statische) Recovery	-,++ kann durch Retry-Mechanismus für die Prozessoren erhöht werden	-,++ ECC + Scrubbing kann Zuverlässigkeit erhöhen	-,0 hohe Fehlerlatenzzeiten, keine effiziente Recovery möglich
Änderbarkeit	o Anpassung der Signaturen und sicherheitskritischen Variablen	++ generische Sicherheitsarchitektur, kaum Anpassungen notwendig	+ Anpassung der Sicherheitsstufen des Speichers	+ Anpassung der Zeitfenster zum Vergleichen der Ausgänge
inv. Speicher Overhead	+ doppelte Ablegung sicherheitskritischer Variablen	+ Parität für gesamten RAM	o Overhead gering, wenn Anpassung an Kritikalität der Daten vorgenommen wird	-- Speicher wird verdoppelt
inv. Laufzeit Overhead	++ WD-Prozessor liest Busverkehr (es werden keine weiteren Infos benötigt)	++ vollständig nebenläufige Fehlerentdeckung	++ vollständig nebenläufige Fehlerentdeckung	o Laufzeit-Overhead kann durch Kommunikation bzw. Synchronisation entstehen
inv. Kosten durch Logik-Overhead	+ kann als heterogener Dual-Core realisiert werden	o wenn lockstep Konfiguration optional, dann möglicherweise bald Standardhardware	- nur für sicherheitskritische Anwendungen	-- sehr hohe Kosten

Abbildung 6-8: Übersicht zur Klassifizierung

6.6.5 Fazit Vergleich

Die Unterteilung der Hardware-Architekturen in die beiden vorgestellten Konzepte ermöglicht es, die Gemeinsamkeiten und die grundlegenden Unterschiede der Hardware-Architekturen herauszustellen. In der ISO WD 26262 wird insbesondere gefordert, dass sicherheitskritische Hardwarefehler nur in begrenztem Maß zum Versagen des Systems in Bezug auf Sicherheit führen können. Grundsätzlich können diese Anforderungen mit beiden

Konzepten erfüllt werden. Wie den Kiviatdiagrammen im Abschnitt 5.6.4 entnommen werden kann, schneiden bei der Bewertung der Hardware-Architekturen mit Funktionsüberwachung die Architekturen Dual-Core Mikrocontroller + Watchdog und Mikrocontroller + FPGA am besten ab, da sie einen hohen Grad an Unabhängigkeit zwischen der Funktionsebene und der Überwachungsebene gewährleisten können. Bei den generischen Architekturen liegt die Herausforderung insbesondere in der Notwendigkeit einer hohen Abdeckung von Prozessorfehlern. In den Architekturen Dual-Core Lockstep und dem Ansatz „MCU + Überwacher“ können diese Anforderungen erfüllt werden. Der Vorteil einer Funktionsüberwachung kann je nach Anwendung in deren Kompaktheit und Anschaulichkeit liegen. Im Gegensatz zu einer generischen Architektur schließt die Funktionsüberwachung die Erkennung sicherheitskritischer Software-Fehler mit ein. In einer generischen Architektur kann dieses durch eine (partielle) Funktionsüberwachung, realisiert als Software-Task, erreicht werden. Eine rein generische Architektur (ohne zusätzliche Funktionsüberwachung) hat den Vorteil, problemlos auf andere Anwendungen übertragen werden zu können. Bei der Funktionsüberwachung bestimmt deren Komplexität den Aufwand von Änderungen. Die Auswirkungen der Sicherheitsmechanismen auf die Zuverlässigkeit des Systems müssen in beiden Architekturen beachtet werden.

6.7 Validität der Ergebnisse

Vor allem die in den Abschnitten 5.1 bis 5.5 vorgestellten Ergebnisse beruhen auf der durchgeführten Studie und unterliegen daher gewissen Einschränkungen bezüglich Ihrer Übertragbarkeit, die im Folgenden diskutiert werden.

Die Studie und darauf aufbauende Untersuchungen stützen sich auf das verwendete Anwendungsbeispiel. Dieses basiert auf einer realen Automobilanwendung, jedoch wurde der Funktionsumfang reduziert um eine Implementierung im Rahmen der Studie zu ermöglichen. Dazu wurde auf eine Implementierung der Funktionen zum Netzwerkmanagement verzichtet. Während diese Anpassung grundsätzlich einen Einfluss auf die Allgemeingültigkeit der Ergebnisse haben könnte, wird dieser Effekt hier als gering eingeschätzt, da der Fokus der Untersuchungen auf der Umsetzung der Sicherheitsfunktionen lag, die vom Netzwerkmanagement weitgehend unabhängig ist.

Weiterhin ist die Wahl des Anwendungsbeispiels insgesamt kritisch zu betrachten, da die Art der Anwendung einen Einfluss auf die Ergebnisse haben kann. Die gewählte Anwendung wird jedoch als repräsentativ für solche Anwendungen angesehen, die einen sicheren Zustand haben und vergleichsweise einfach umzusetzende Sicherheitsfunktion ermöglichen. Für Anwendungen ohne sicheren Zustand oder solche mit einer komplexen Sicherheitsfunktion ergeben sich Unterschiede (siehe Abschnitt 5.6) die gesondert zu untersuchen sind. Folglich sind Untersuchungen auf Basis alternativer Anwendungen wünschenswert.

Schließlich wurden die Entwicklungstätigkeiten in dieser Studie von Studenten durchgeführt. Diese verhalten sich vermutlich anders als professionelle Entwickler. Diesem Aspekt wurde Rechnung getragen, in dem nicht absolute Fehler und Entwicklungszeiten gemessen wurden, sondern in dem der Entwicklungsfortschritt und auftretende Probleme in wöchentlichen Treffen dokumentiert wurden. Dies ermöglichte eine Unterscheidung zwischen Problemen, die auf mangelnde Erfahrung zurückzuführen sind und solchen, die charakteristisch für die gewählten Ansätze sind. Im Fall der durchgeführten Studie wurden keine Ergebnisse beobachtet, die als Ursache eine mangelnde Erfahrung der Entwickler vermuten ließen.

7 Zusammenfassung

Der Schwerpunkt der Untersuchungen bildete eine Studie, in der zwei ausgewählte neuartige Hardware-Architekturen auf ihre Eignung bezüglich ihres Einsatzes in Automobilanwendungen untersucht wurden. Dazu wurden die Aspekte Sicherheit, Zuverlässigkeit, Entwicklungsaufwand, Testbarkeit und Änderbarkeit untersucht.

Die Bewertung der Sicherheit erfolgte nach den Vorgaben der Sicherheitsnorm ISO WD 26262. Als Ergebnis ist die Erfüllung der Sicherheitsanforderungen (ASIL C) mit beiden betrachteten Hardware-Architekturen (Dual-Core-Architektur und MCU-FPGA-Architektur) möglich. In beiden Architekturen kann genügend Unabhängigkeit zwischen den Anwendungsfunktionen und den Sicherheitsfunktionen erreicht werden. Als Sicherheitskonzept wurde eine Funktionsüberwachung verwendet, welche eine inhaltliche Überprüfung der sicherheitskritischen Funktionen durchführt. Diese war für die Cabrioüberdecksteuerung einfach und stellte keine speziellen Anforderungen an die verwendete Hardware. Im Zusammenhang mit den Sicherheitsuntersuchungen wurde weiterhin untersucht, wie sich ein Wechsel des ASIL von ASIL C nach ASIL B auswirkt. Die reduzierten Anforderungen des niedrigeren ASIL ermöglichen vermutlich einen Verzicht auf getrennte Hardware für die Ausführung der Funktionsüberwachung. Daher könnte im MCU-FPGA Ansatz auf den FPGA verzichtet werden (Ersatz durch Standard-Watchdog) während beim Dual-Core Ansatz eine Migration auf einen Single-Core Mikrocontroller oder eine anderweitige Nutzung des zweiten Cores nötig wäre. Schließlich erfolgte eine Betrachtung der Testbarkeit der Sicherheitsfunktionen für die gewählten Ansätze. Aufgrund des gleichen Sicherheitskonzeptes in beiden Ansätzen (Funktionsüberwachung) unterschieden sich die Testvorgänge nur in kleineren Details und es konnten keine entscheidenden Unterschiede bezüglich der Testbarkeit festgestellt werden.

Aufgrund der gleichartigen Sicherheitskonzepte wurden im Bezug auf die Zuverlässigkeit nur vereinzelte Unterschiede zwischen den beiden untersuchten Architekturen festgestellt. Vorteile bezüglich der Zuverlässigkeit ergeben sich für die Dual-Core Architektur dadurch, dass hier nahezu das gesamte System in einem Baustein integriert ist. Vorteile für die MCU+FPGA Architektur ergeben sich dadurch, dass im FPGA die Zuverlässigkeit kritischer Funktionen durch deren Verdopplung oder Verdreifachung erhöht werden kann.

Signifikante Unterschiede im Entwicklungsaufwand der Anwendung und der Funktionsüberwachung auf den beiden Hardware-Architekturen konnten nicht festgestellt werden. Jedoch ergibt sich ein zusätzlicher Aufwand gegenüber generischen Konzepten durch die Notwendigkeit eine Funktionsüberwachung für die Erkennung von Hardware-Fehlern in Software zu implementieren. Es ist jedoch zu beachten, dass dieser Zusatzaufwand auch bei einem generischen Ansatz nötig sein kann, um Softwarefehler zu behandeln.

Weiterhin wurde die Änderbarkeit mit Hilfe möglicher Änderungsszenarien betrachtet. Als Ergebnis lassen sich Änderungen in beiden Architekturen vergleichsweise gut einpflegen wenn das Model für die Verletzung der Sicherheitsanforderungen einfach ist, diese Informationen leicht zugänglich/messbar sind und ein sicherer Zustand durch Deaktivieren eines Aktuators/ mehrerer Aktuatoren erreicht werden kann.

Schließlich wurden die gewählten Architekturen mit alternativen Architekturkonzepten verglichen, die sowohl auf dem Prinzip der Funktionsüberwachung als auf generischen Konzepten beruhen. Für das gewählte Anwendungsbeispiel ergaben sich die meisten Vorteile für die in der Studie gewählten Architekturen. Wie bereits angesprochen kann für Anwendungen, die sich im Bezug auf die umzusetzende Sicherheitsfunktion von der untersuchten

Anwendung maßgeblich unterscheiden, die Wahl einer anderen Architektur von Vorteil sein. Ein wichtiges Ergebnis dieser Untersuchung sind die identifizierten Unterschiede zwischen den beiden alternativen Sicherheitskonzepten der Funktionsüberwachung und der Bauteilüberwachung (generischer Ansatz). Beide Ansätze haben ihre individuellen Vor- und Nachteile und ihre Eignung hängt sehr von der jeweiligen Anwendung und insbesondere der zu realisierenden Sicherheitsfunktionen ab.

8 Ausblick

Neben den vorgestellten Ergebnissen blieben vereinzelte Fragen unbeantwortet, und es ergaben sich zudem weitergehende Fragestellungen. Eine Einschränkung der gewonnenen Ergebnisse liegt in ihrer Abhängigkeit vom verwendeten Anwendungsbeispiel. Zwar wurde die Übertragbarkeit der Ergebnisse auf andere Anwendungen im Automobil diskutiert, jedoch konnte bisher aus Zeitgründen keine systematische Analyse erfolgen. Es wäre erstrebenswert, wenn eine Verallgemeinerung der Ergebnisse in der Hinsicht möglich wäre, dass bezogen auf eine beliebige Automobilanwendung (Funktion + Anforderungen an Sicherheitsfunktion) eine Empfehlung für eine bestimmte Architektur gegeben werden könnte (incl. der jeweiligen Vor- u. Nachteile). Weiterhin wurden Überlegungen zur Zuverlässigkeit der betrachteten Architekturen vorgestellt. Für eine weiterführende Betrachtung der Zuverlässigkeit wäre es jedoch wünschenswert eine geeignete Systematik zur Zuverlässigkeitsbewertung zu erarbeiten. Für diese Bewertung muss vor allem geklärt werden, wie beispielsweise Fehlertoleranz systematisch untersucht werden kann, welche Fehlerarten durch welche Mechanismen abgefangen werden, und welchen Einfluss die Bildung von „Fault Containment Regions“ auf die Bewertung hat.

Des Weiteren ist darauf hinzuweisen, dass eine geeignete Sicherheitsarchitektur auch durch eine zweckmäßige Zusammenarbeit mehrerer Steuergeräte möglich ist (Sicherheitskonzept über Systemverbund). Während bei diesem Ansatz auf einen gewissen Teil der in dem hier vorgestellten Sicherheitskonzept enthaltenen Hardware-Redundanz verzichtet werden kann, steigt der erforderliche Kommunikationsaufwand zwischen den einzelnen Steuergeräten. Weiterhin ist zu beachten, dass in diesem Fall die erforderlichen Sicherheitsnachweise sowohl alle beteiligten Steuergeräte als auch das verwendete Bussystem beinhalten. Dieser Aspekt führt vermutlich zu einem komplexeren Sicherheitsnachweis. Beispielsweise wird das Bussystem vermutlich von zahlreichen weiteren Steuergeräten genutzt. Es ist daher sicher zu stellen, dass zu keiner Zeit unerwünschte Seiteneffekte zwischen den einzelnen Nachrichten auftreten können.

Referenzen

- [ISO26262] ISO WD 26262 – Road vehicles – Functional Safety, International Organization for Standardization, working draft baseline 7
- [Beckschulze] Eva Beckschulze, Diplomarbeit, Lehrstuhl Informatik 11, RWTH Aachen, 2008
- [Benso] Alfredo Benso, Stefano Di Carlo, Giorgio Di Natale, Paolo Prinetto “A Watchdog Processor to Detect Data and Control Flow Errors” Proceedings of the 9th IEEE International On-Line Testing Symposium (IOLTS’03)
- [Boymanns] David Boymanns, Diplomarbeit, Lehrstuhl Informatik 11, RWTH Aachen, 2008
- [Dülks] Ramona Dülks, Diplomarbeit, Lehrstuhl Informatik 11, RWTH Aachen, 2008
- [Fruehling] Terry Fruehling, “Delphi Secured Microcontroller Architecture”, SAE Technical Paper, 2000-01-1052
- [ISoLA08] Eva Beckschulze, Falk Salewski, Thomas Siegbert, Stefan Kowalewski: “Fault Handling Approaches on Dual-Core Microcontrollers in Safety-Critical Automotive Applications”, 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), 2008
- [Koopman] Philip Koopman, Jennifer Morris, “Representing Design Tradeoffs in Safety-Critical Systems”, International Conference on Software Engineering, Proceedings of the 2005 workshop on Architecting dependable systems
- [Mariani] Riccardo Mariani, Boris Vittorelli, Peter Fuhrmann “Cost-effective Approach to Error Detection for an Embedded Automotive Platform“, SAE Technical Paper 2006-01-0837
- [Siegbert] Thomas Siegbert, Diplomarbeit, Lehrstuhl Informatik 11, RWTH Aachen, 2008
- [Spezifikation] Systemspezifikation Anwendungsbeispiel, Lehrstuhl Informatik 11, RWTH Aachen, 2008

Anhang A: Fehlerbäume zur Verifikation der funktionalen Sicherheitsanforderungen

Darstellung: grau = Ergänzungen durch Sicherheitsfunktion, 1,2,3,4,5 = Unterbäume

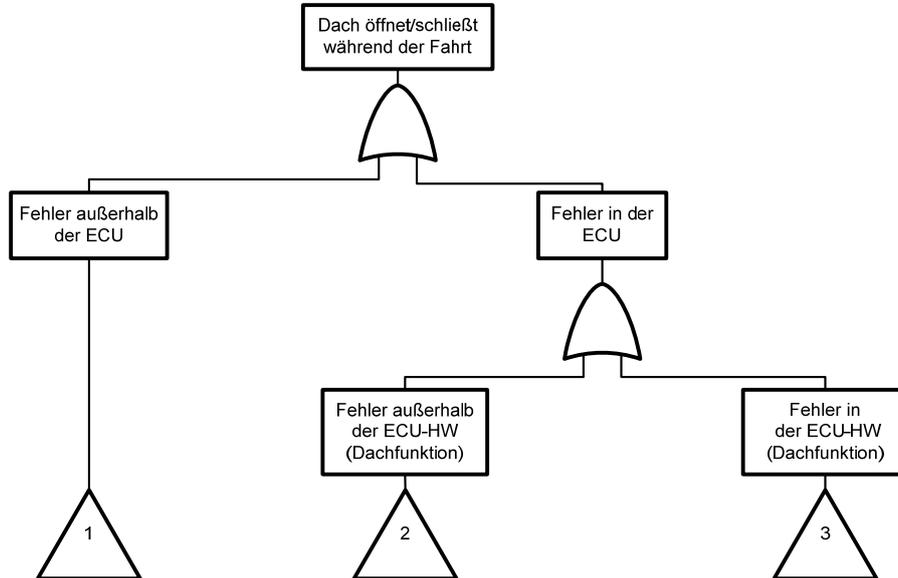


Abb.A.1: Fehlerbaum für Sicherheitsziel A

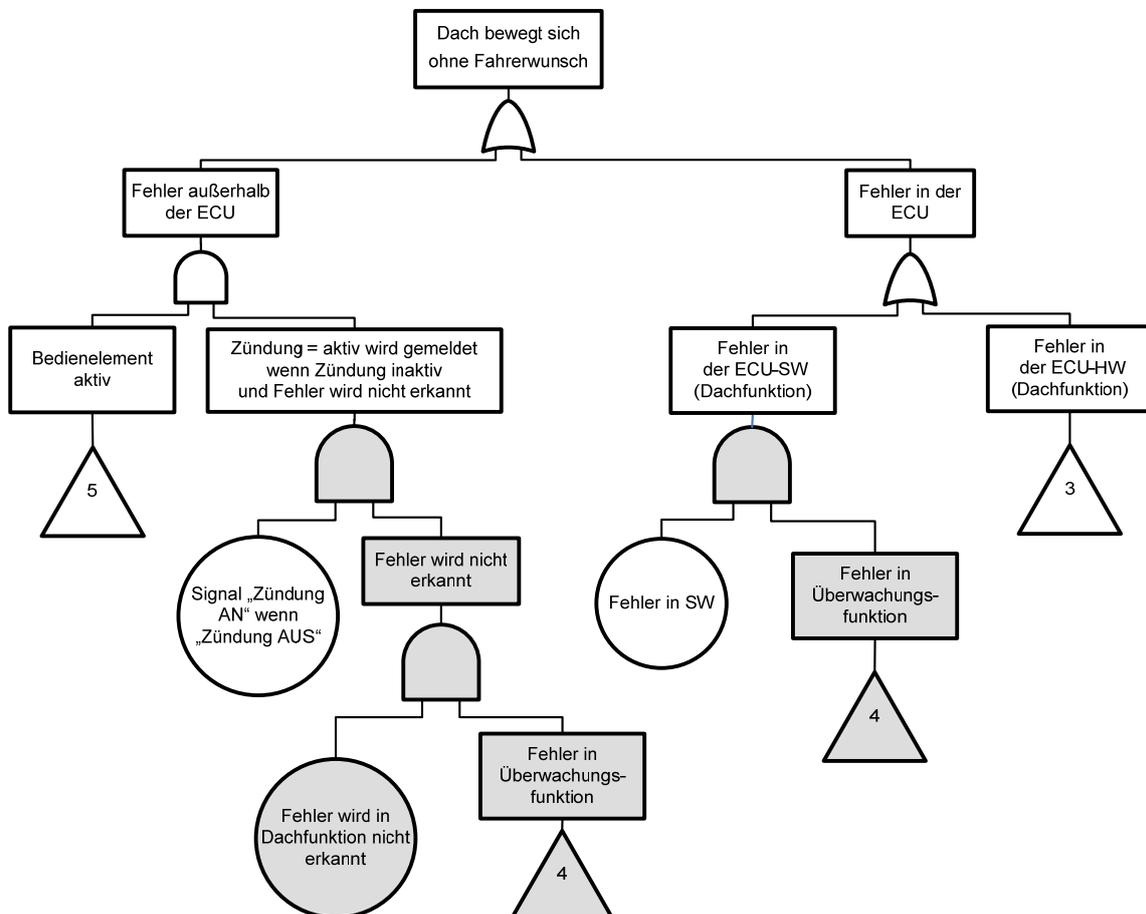


Abb.A.2: Fehlerbaum für Sicherheitsziel B

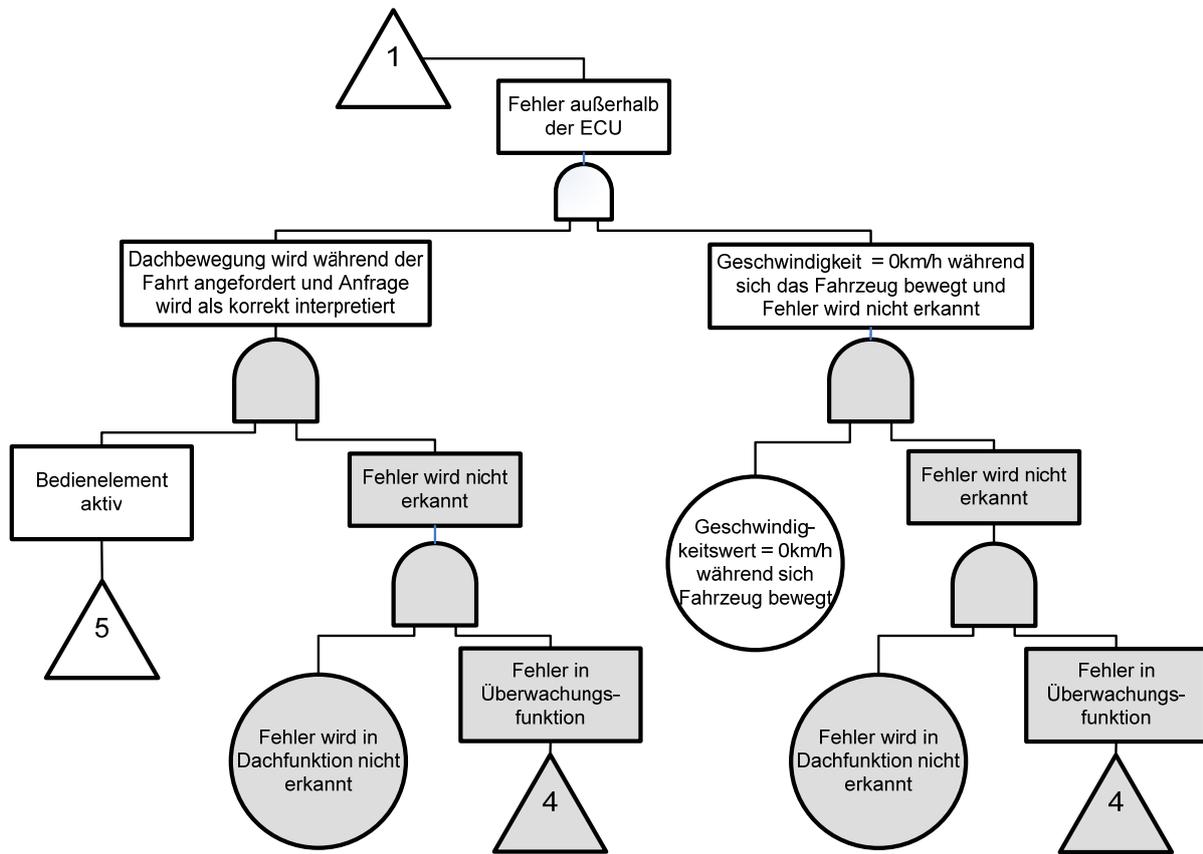


Abb.A.3: Unterbaum 1

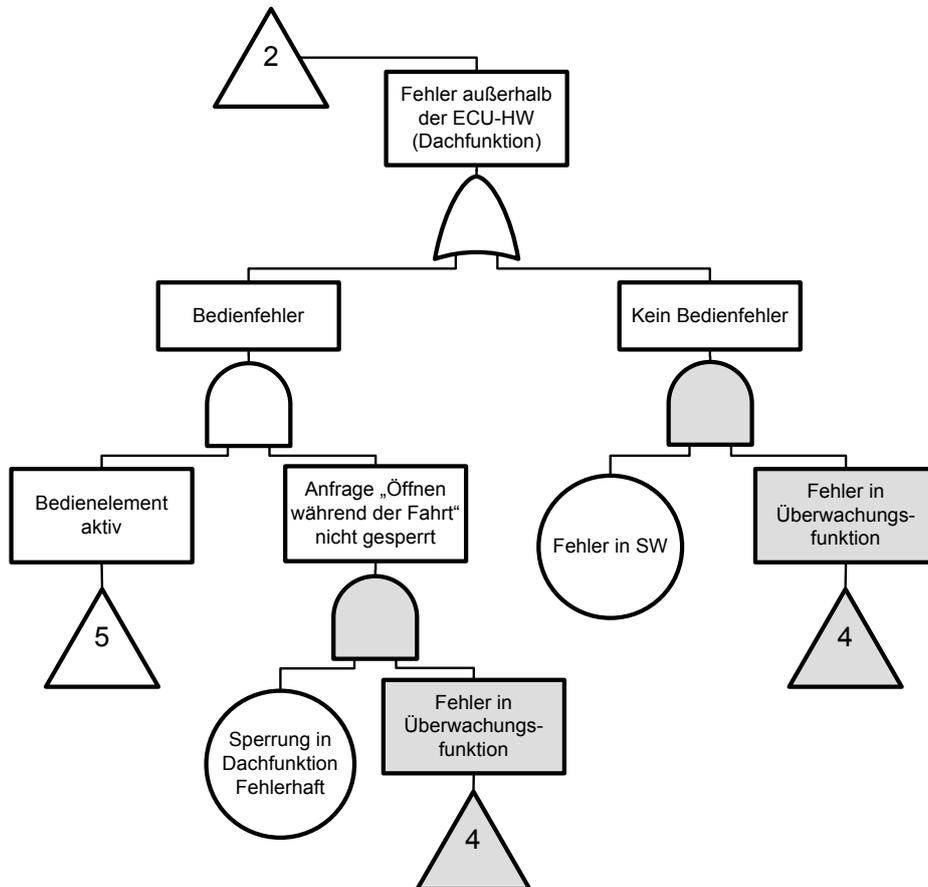
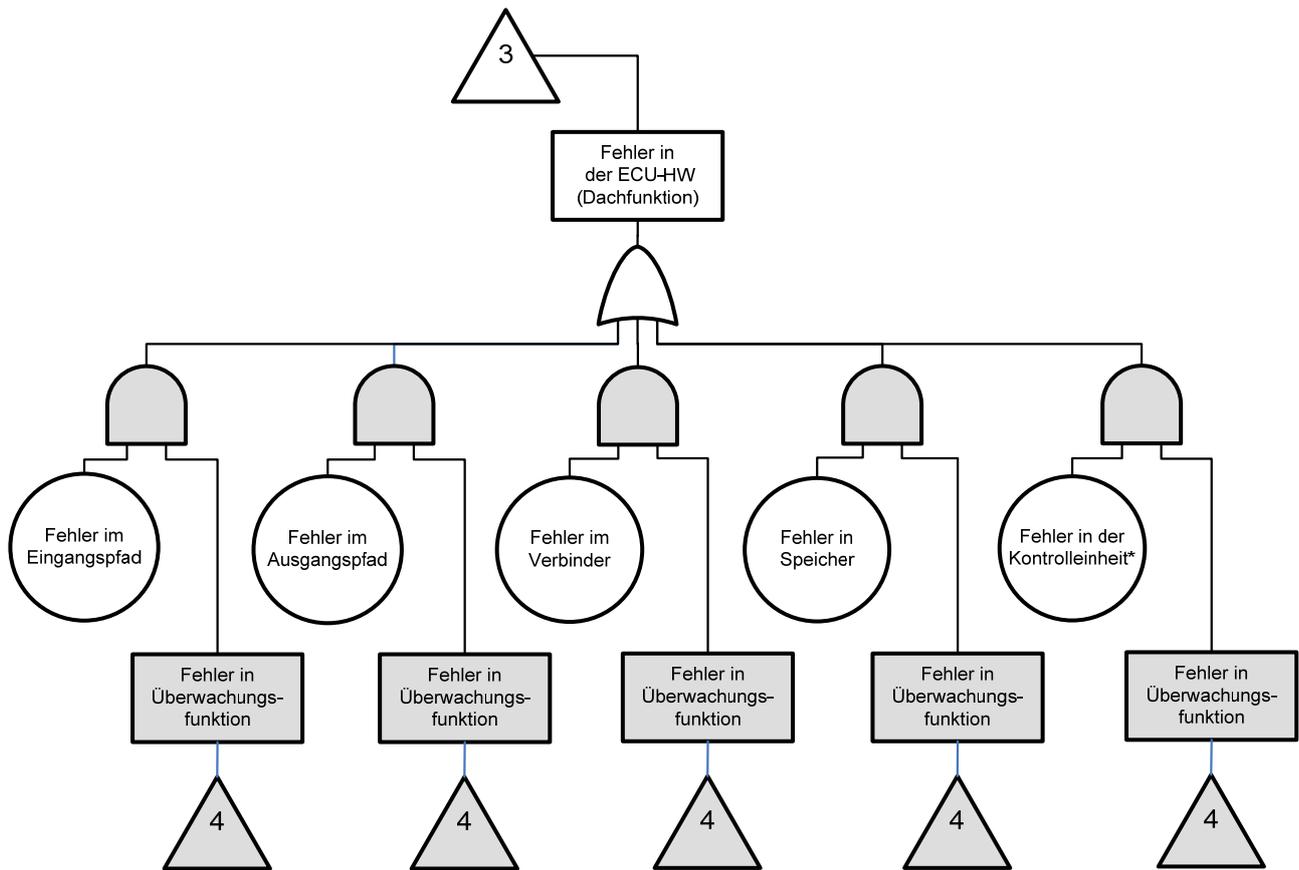
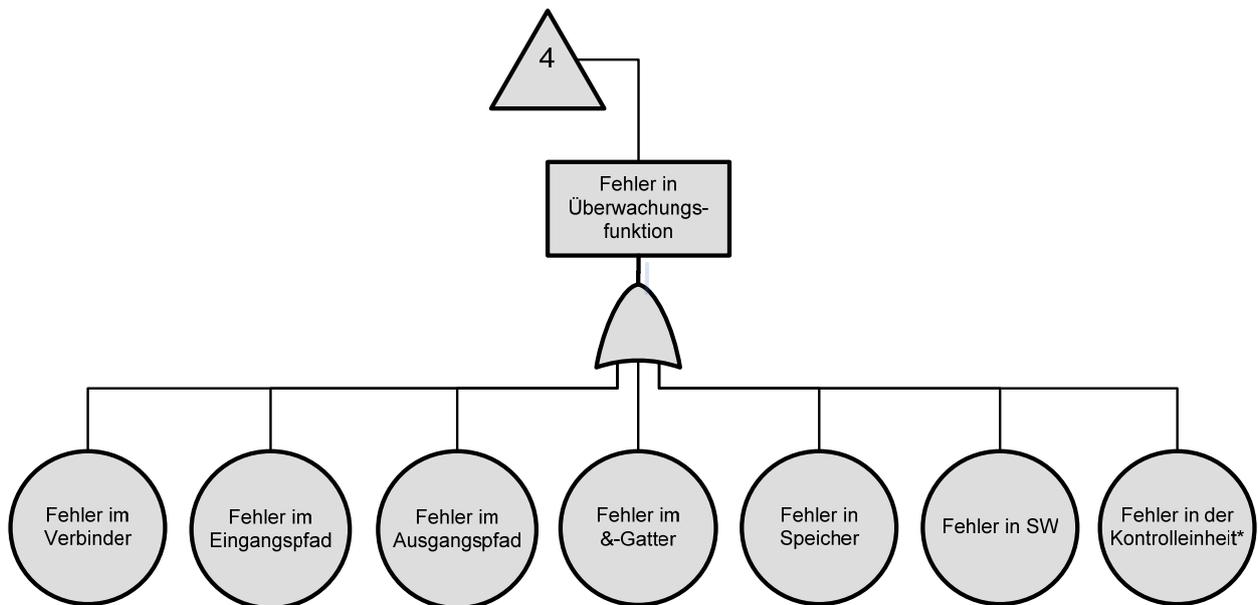


Abb.A.4: Unterbaum 2



* Kontrolleinheit beinhaltet hier Clock und Spannungsversorgung. Wenn die Überwachungsfunktion die gleiche Clock und Spannungsversorgung benutzt wie die Hauptfunktion, ist zu zeigen, dass dies keine Single Point Fehler sind!

Abb.A.5: Unterbaum 3



* Kontrolleinheit beinhaltet hier Clock und Spannungsversorgung. Wenn die Überwachungsfunktion die gleiche Clock und Spannungsversorgung benutzt wie die Hauptfunktion, ist zu zeigen, dass dies keine Single Point Fehler sind!

Abb.A.6: Unterbaum 4

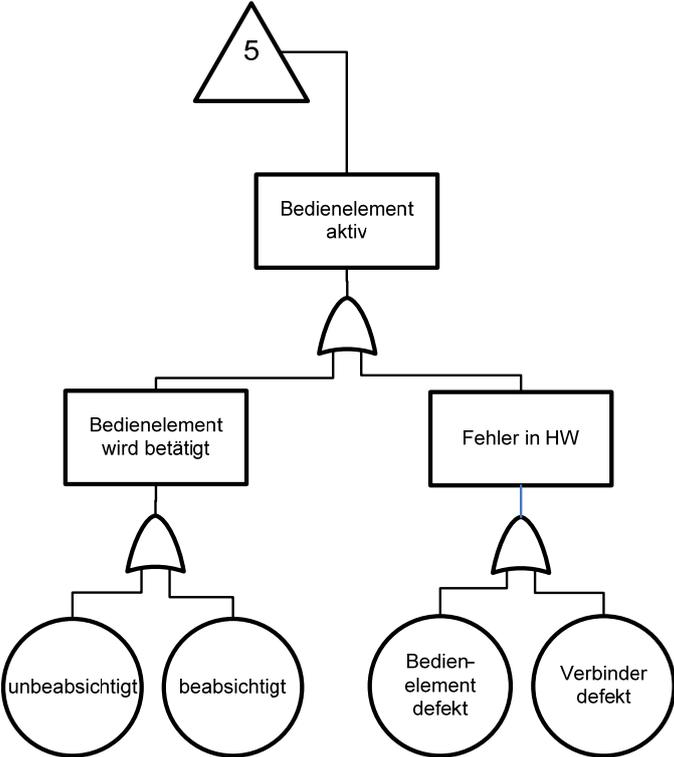


Abb.A.6: Unterbaum 5

Anhang B: Bestimmung der Fehlermetriken und der Diagnoseabdeckung

Dieser Anhang beinhaltet die Berechnung der Single Point Faults Metric (SPFM) und der Latent Faults Metric (LFM) für die betrachteten Architekturen sowie die Single-Core-Mikrocontroller Architektur. Die Berechnung richtet sich nach ISO WD 26262-5 (Baseline 9).

Weiterhin wird in diesem Anhang die Bestimmung der Diagnoseabdeckung (Diagnostic Coverage) dargestellt. Diese Untersuchungen basieren auf den Vorgaben von ISO WD 26262-5 (Baseline 9), Annex A.

Bestimmung der SPFM und LFM

Die Berechnung der SPFM und der LFM ist nach einem ähnlichen Schema gegliedert. Grundlage für die Berechnung sind jeweils die Ausfallraten der verwendeten sicherheitsrelevanten Bauteile (nach ISO WD 26262: Bauteile, die zu einer Verletzung des Sicherheitsziels oder zum Fehlschlagen eines Sicherheitsmechanismus führen können) in FIT (Ausfälle in 10^9 h \approx 114.000 Jahre). Für jede Komponente werden alle Fehlerarten identifiziert und mit ihrer Auftrittshäufigkeit relativ zur Gesamtkomponente aufgelistet. Für alle Fehlerarten ist dann zu prüfen, ob und auf welche Weise sie zu einer Verletzung des Sicherheitsziels führen können.

Zur Bestimmung der Single Point- bzw. Latent Multiple Point Failure Rate (SPFR/LMFR) für eine Fehlerart wird die durch die ISO WD 26262 vorgeschlagene Abschätzung mit Hilfe der zugehörigen Diagnostic Coverage herangezogen. Hierbei wird davon ausgegangen, dass diejenigen Fehler Residual Faults bzw. Latent Multiple Point Faults sind, welche nicht durch entsprechende Mechanismen abgedeckt werden.

So berechnet sich die Fehlerrate für eine Fehlerart als:

$$\text{SPFR}_{\text{Fehlerart}} = \text{FIT}_{\text{Bauteil}} * \text{relative Häufigkeit der Fehlerart} * (1 - \text{DC}_{\text{SPF}(\text{Fehlerart})})$$

$$\text{LMFR}_{\text{Fehlerart}} = \text{FIT}_{\text{Bauteil}} * \text{relative Häufigkeit der Fehlerart} * (1 - \text{DC}_{\text{LMF}(\text{Fehlerart})})$$

Die Fehlermetriken können mit Hilfe der Summe der Gesamtbauteilfehlerraten (FIT) und der Summe über alle SPFR bzw. alle LMFR berechnet werden.

Die Single Point Faults Metrik soll ein Maß für die Robustheit des Systems gegen Single Point Faults (ein Fehler im System, der allein zur Verletzung des Sicherheitsziels führt) darstellen. Sie repräsentiert den Anteil an allen möglichen Fehlern im System, die nicht direkt (als Single Point Fault) zur Verletzung des Sicherheitsziels führen können. Man teilt daher alle Fehler, die keine Single Point oder Residual Faults (Restfehler, die bei der Vermeidung von Single Point Faults verbleiben) sind durch die Gesamtfehlerrate aller Bauteile. Damit ergibt sich folgende Formel für die Single Point Fault Metrik:

$$\text{SPFM} = 1 - \left(\frac{\sum \text{SPFR}_{\text{Fehlerart}}}{\sum \text{FIT}_{\text{Bauteil}}} \right)$$

Die Latent Faults Metrik repräsentiert den Anteil der Multipoint Faults (gemeint sind hier die Teilfehler, die zusammen mit anderen Fehlern zur Verletzung des Sicherheitsziels führen – nicht das Auftreten der Fehlersumme, die als Ganzes zur Verletzung des Sicherheitsziels führt), welche durch entsprechende Überwachungsmechanismen erkannt werden und damit nicht latent im System bleiben. Man erhält sie durch das vorherige Abziehen der gesamten Single Point Failure Rate von der Gesamtfehlerrate aller Bauteile:

$$\text{LFM} = 1 - \left(\frac{\sum \text{LMFR}_{\text{Fehlerart}}}{\left(\sum \text{FIT}_{\text{Bauteil}} - \sum \text{SPFR}_{\text{Fehlerart}} \right)} \right)$$

Im Folgenden findet sich eine mit dieser Methode durchgeführte Aufstellung der Fehlermetriken für die im Rahmen der Studie betrachteten Architekturansätze.

Hierbei sind in allen 3 Fällen folgende Annahmen gemacht worden:

- Die Treiberstufe und das Bedienelement sind „Proven in use“. Dadurch kann ihre Betrachtung in den Fehlermetriken entfallen, obwohl es sich um sicherheitsrelevante Komponenten handelt.
- Die Ausfallraten der verwendeten Bauteile basieren auf Werten von Vergleichskomponenten von Maxim IC (www.maxim-ic.com)
- Ein Sicherheitsziel wird verletzt, wenn die Hydraulikpumpe ungewollt angetrieben wird oder nicht (rechtzeitig) abgeschaltet werden kann. Eine genauere Betrachtung der Art und Weise wie die einzelnen Failure Modes zur Verletzung des Sicherheitsziels führen können, wird in diesem Beispiel aufgrund der sehr geringen Anzahl an betrachteten Hazards nicht weiter durchgeführt. Es kann sich im Falle von komplexeren Systemen durchaus als sinnvoll erweisen, die unten aufgeführten Tabellen um eine Spalte zu erweitern, die nochmals im Detail ausführt, welches der Sicherheitsziele auf welche Art und Weise verletzt werden kann.

SPFM und LFM für Dual-Core-Mikrocontroller Architektur

Die erstellten Fehlermetriken für die Dual-Core Architektur sind in den Abbildungen B1 und B2 zu finden. Zu beachten ist bei den zugrunde liegenden Berechnungen, dass das Programm für den zweiten Core (Überwacher) nach dem Startup in den RAM des zweiten Cores geladen wird, wodurch nur beim Startup ein Fault durch den ROM entstehen kann, welcher zu diesem Zeitpunkt durch CRC16 abgefangen wird. Während des Programmablaufs auftretende Veränderungen am Programmcode sind Multiple Point Faults.

Die Systemclock kann in unserem Modell lediglich zu einem Single Point Fault oder einem Safe Fault führen, da außer dem Time-Windowed Watchdog keine Frequenzüberwachung durchgeführt wird und die Clock als hinreichend unabhängig von anderen sicherheitskritischen Komponenten angesehen wird.

Der Sicherheitsmechanismus „RB“ bezeichnet das Zurücklesen der Pumpenleitung durch den zweiten Core (CPU2).

Component name	FIT	Failure Mode	Failure Rate Distribution	Dual-Core + external WD + Brownout Chip			
				Single Point Faults Metric			
				possible SPF?	Safety Mechanism	Failure Mode Coverage	SPFR
RAM 1	9,8	Bit flip	33%	X	RB	100%	0
		Stuck-at for data and addresses	33%	X	RB	100%	0
		No, wrong or multiple addressing	34%	X	RB	100%	0
ROM	4,9	Bit flip	50%	X	CRC16	99%	0,0245
		Stuck-at for data and addresses	50%	X	CRC16	99%	0,0245
CPU 1	10	Programmcounter-Flip	10%	X	RB	100%	0
		Stackcounter-Flip	10%	X	RB	100%	0
		Internal RAM, register: stuck at fault	10%	X	RB	100%	0
		Wrong Coding or no execution	10%	X	RB	100%	0
		Stuck at address calculation	10%	X	RB	100%	0
		CPU stop working	10%	X	RB	100%	0
		overheat affects whole MCU	10%	X	RB	100%	0
		No interrupts	10%	X	RB	100%	0
		Continuous interrupts	10%	X	RB	100%	0
Cross-over of interrupts	10%	X	RB	100%	0		
RAM 2	9,8	Bit flip	33%	This component is only for diagnostic use and therefore there are no single point faults which are caused by this component.			
		Stuck-at for data and addresses	33%				
		No, wrong or multiple addressing	34%				
CPU 2	10	Programmcounter-Flip	10%				
		Stackcounter-Flip	10%				
		Internal RAM, register: stuck at fault	10%				
		Wrong Coding or no execution	10%				
		Stuck at address calculation	10%				
		CPU stop working	10%				
		overheat affects whole MCU	10%				
No interrupts	10%						
Continuous interrupts	10%						
Cross-over of interrupts	10%						
I/O (digital)	0,13	Stuck at 0 (Output)	25%	Safe Fault			
		Stuck at 1 (Output)	25%	Only LMPF because of redundancy			
		Stuck at 0 (Input)	25%				
		Short Circuit between pins	25%				
Systembus	1	some errors	100%	X		0%	1
Clock	6,8	unsteady/wrong frequency	50%	X	external watchdog	99%	0,034
		no frequency	50%	X	external watchdog	99%	0,034
serial interface (CAN)	0,09	Stuck-at (data or addresses)	13%	X	frame counter	90%	0,00113
		Time out	13%	X	SW-Timeout (Alarm)	100%	0
		Stuck-at of arbitration signals	12%	X	SW-Timeout (Alarm)	90%	0,00108
		No, continuous or wrong arbitration	12%	X	SW-Timeout (Alarm)	90%	0,00108
		Message corruption	13%	X	frame counter	90%	0,00117
		Message delay	13%	X	SW-Timeout (Alarm)	100%	0
		Message loss	12%	X	frame counter	100%	0
Unintended message repetition	12%	X	frame counter	90%	0,00108		
Exchange RAM	0,7	Some kind of error in Exchange RAM	100%	Diagnostic unit has only LMPF			
Brown-Out Chip	0,12	Some kind of error in Brown-Out Chip	100%				
Watchdog	0,09	some kind of error in WD	100%				
power supply	0,12	power supply too low	50%	X	Brownout + ext. WD	99%	0,0006
		power supply zero	50%	Safe Fault			
Sum:	53,6	SPFM = 1 - (1,12314 / 53,6)		Single Point Faults Metric:			Sum: 1,12314 SPFM: 97,90%

Abb. B 1 Single Point Fault Metric Dual-Core Architektur

Component name	FIT	Failure Mode	Failure Rate Distribution	Dual-Core + external WD + Brownout Chip			
				Latent Faults Metric			
				possible LMF?	Safety Mechanism	Failure Mode Coverage	LMFR
RAM 1	9,8	Bit flip	33%	X	RB	100%	0
		Stuck-at for data and addresses	33%	X	RB	100%	0
		No. wrong or multiple addressing	34%	X	RB	100%	0
ROM	4,9	Bit flip	50%	X	CRC16	99%	0,0245
		Stuck-at for data and addresses	50%	X	CRC16	99%	0,0245
CPU 1	10	Programmcounter-Flip	10%	X	RB	100%	0
		Stackcounter-Flip	10%	X	RB	100%	0
		Internal RAM, register: stuck at fault	10%	X	RB	100%	0
		Wrong Coding or no execution	10%	X	RB	100%	0
		Stuck at address calculation	10%	X	RB	100%	0
		CPU stop working	10%	X	RB	100%	0
		overheat affects whole MCU	10%	X	RB	100%	0
		No interrupts	10%	X	RB	100%	0
		Continous interrupts	10%	X	RB	100%	0
		Cross-over of interrupts	10%	X	RB	100%	0
RAM 2	9,8	Bit flip	33%	X	MarchC	60%	1,2936
		Stuck-at for data and addresses	33%	X	MarchC	60%	1,2936
		No. wrong or multiple addressing	34%	X	MarchC	60%	1,3328
CPU 2	10	Programmcounter-Flip	10%	X	external WD	60%	0,4
		Stackcounter-Flip	10%	X	external WD	60%	0,4
		Internal RAM, register: stuck at fault	10%	X	external WD	90%	0,1
		Wrong Coding or no execution	10%	X	external WD	100%	0
		Stuck at address calculation	10%	X	external WD	100%	0
		CPU stop working	10%	X	external WD	100%	0
		overheat affects whole MCU	10%	X	external WD	100%	0
		No interrupts	10%	X	external WD	100%	0
		Continous interrupts	10%	X	external WD	100%	0
		Cross-over of interrupts	10%	X	external WD	90%	0,1
I/O (digital)	0,13	Stuck at 0 (Output)	25%		Not safety related		
		Stuck at 1 (Output)	25%	X	RB	60%	0,013
		Stuck at 0 (Input)	25%	X	none	0%	0,0325
		Short Circuit between pins	25%	X	none	0%	0,0325
Clock	6,80	unsteady/wrong frequence	50%		Only Single Point Fault		
		no frequency	50%				
serial interface (CAN)	0,09	Stuck-at (data or addresses)	13%	X	frame counter	90%	0,001125
		Time out	13%	X	SW-Timeout (Alarm)	100%	0
		Stuck-at of arbitration signals	12%	X	SW-Timeout (Alarm)	90%	0,00108
		No. continous or wrong arbitration	12%	X	SW-Timeout (Alarm)	90%	0,00108
		Message corruption	13%	X	frame counter	90%	0,00117
		Message delay	13%	X	SW-Timeout (Alarm)	100%	0
		Message loss	12%	X	frame counter	100%	0
		Unintended message repetition	12%	X	frame counter	90%	0,00108
Systembus	1	Some errors	100%	X	None	0%	1
Brown-Out Chip	0,12	some kind of error in WD	100%	X	none	0%	0,12
Exchange RAM	0,7	Some kind of error in Exchange RAM	100%	X	MarchC	60%	0,28
Watchdog	0,09	some kind of error in WD	100%	X	none	0%	0,09
power supply	0,12	power suppy too low	50%		Only Single Point Fault		
		power suppy zero	50%		Not safety related		
Sum:	53,6	LFM = 1 - (6,4775 / (53,6 - 1,1231))		Latent Faults Metric:			Sum: 6,477535 LFM: 87,64%

Abb. B 2 Latent Fault Metric Dual-Core Architektur

SPFM und LFM für Mikrocontroller-FPGA Architektur

Component Name	FIT	Failure Mode	Failure Rate Distribution	MCU + FPGA			
				Single Point Faults Metric			
				possible SPF?	Safety Mechanism	Failure Mode Coverage	SPFR
RAM	9,8	Bit flip	33%	X	FPGA (1)	100%	0
		Stuck-at for data and addresses	33%	X	FPGA (1)	100%	0
		No, wrong or multiple addressing	34%	X	FPGA (1)	100%	0
ROM	0,8	Bit flip	50%	X	FPGA (1)	100%	0
		Stuck-at for data and addresses	50%	X	FPGA (1)	100%	0
CPU	10	Programmcounter-Flip	10%	X	FPGA (1)	100%	0
		Stackcounter-Flip	10%	X	FPGA (1)	100%	0
		Internal RAM, register: stuck at fault	10%	X	FPGA (1)	100%	0
		Wrong Coding or no execution	10%	X	FPGA (1)	100%	0
		Stuck at address calculation	10%	X	FPGA (1)	100%	0
		CPU stop working	10%	X	FPGA (1)	100%	0
		overheat affects whole MCU	10%	X	FPGA (1)	100%	0
		No interrupts	10%	X	FPGA (1)	100%	0
		Continous interrupts	10%	X	FPGA (1)	100%	0
		Cross-over of interrupts	10%	X	FPGA (1)	100%	0
Systembus	1	unspecified fauls	100%	X	FPGA (1)	100%	0
I/O (digital)	0,13	Stuck at 0 (Output)	25%		safe fault		
		Stuck at 1 (Output)	25%	X	FPGA (1)	100%	0
		Stuck at 0 (Input)	25%	X	FPGA (1)	100%	0
		Short Circuit between pins	25%	X	FPGA (1)	100%	0
Clock	29	unsteady/wrong frequency	50%	X	FPGA (1) + FPGA (2)	100%	0
		no frequency	50%	X	FPGA (1) + FPGA (2)	100%	0
serial interface (CAN)	0,09	Stuck-at (data or addresses)	13%	X	frame counter + CRC	90%	0,00113
		Time out	13%	X	software timeout	100%	0
		Stuck-at of arbitration signals	12%	X	loss of communication	90%	0,00108
		No, continous or wrong arbitration	12%	X	CRC	90%	0,00108
		Message corruption	13%	X	frame counter + CRC	90%	0,00117
		Message delay	13%	X	software timeout	100%	0
		Message loss	12%	X	software timeout + frame counter	100%	0
SPI interface	0,09	Unintended message repetition	12%	X	frame counter	100%	0
		Stuck-at (data or addresses)	17%		no SPF, only needed for the monitoring system		
		Time out	16%				
		Message corruption	17%				
		Message delay	17%				
		Message loss	16%				
Unintended message repetition	17%						
power supply	0,12	supply voltage too low	50%	X	FPGA (1)	100%	0
		supply voltage zero	50%		safe fault		
Failures in safety mechanism							
FPGA (XC3S200)	12		100%		monitoring system has no direct connection to the acutators that could lead to a violation of the safety goals so there are no SPF caused by this components		
ROM	0,8	Bit flip	50%				
		Stuck-at for data and addresses	50%				
I/O (digital)	0,13	Stuck at 0 (Output)	25%				
		Stuck at 1 (Output)	25%				
		Stuck at 0 (Input)	25%				
		Short Circuit between pins	25%				
Clock	29	unsteady/wrong frequency	50%				
		no frequency	50%				
SPI interface	0,09	Stuck-at (data or addresses)	17%				
		Time out	16%				
		Message corruption	17%				
		Message delay	17%				
		Message loss	16%				
		Unintended message repetition	17%				
power supply	0,12	supply voltage too low	50%				
		supply voltage zero	50%				
Sum:	93,2	SPFM = 1 - (0,004 / 93,2)		Single Point Faults Metric:		Sum:	0,004
						SPFM:	99,995%

Abb. B 3 Single Point Fault Metric MCU-FPGA Architektur

Component Name	FIT	Failure Mode	Failure Rate Distribution	MCU + FPGA			
				Latent Faults Metric			
				possible LF?	Safety Mechanism	Failure Mode Coverage	LMFR
RAM	9,8	Bit flip	33%	X	FPGA (2) + MCU (1)	95%	0,1617
		Stuck-at for data and addresses	33%	X	FPGA (2) + MCU (1)	95%	0,1617
		No, wrong or multiple addressing	34%	X	FPGA (2) + MCU (1)	95%	0,1666
ROM	0,8	Bit flip	50%	X	FPGA (2) + MCU (1)	95%	0,02
		Stuck-at for data and addresses	50%	X	FPGA (2) + MCU (1)	95%	0,02
CPU	10	Programmcounter-Flip	10%	X	FPGA (2) + MCU (1)	95%	0,05
		Stackcounter-Flip	10%	X	FPGA (2) + MCU (1)	95%	0,05
		Internal RAM, register: stuck at fault	10%	X	FPGA (2) + MCU (1)	95%	0,05
		Wrong Coding or no execution	10%	X	FPGA (2) + MCU (1)	95%	0,05
		Stuck at address calculation	10%	X	FPGA (2) + MCU (1)	95%	0,05
		CPU stop working	10%	X	FPGA (2) + MCU (1)	95%	0,05
		overheat affects whole MCU	10%	X	FPGA (2) + MCU (1)	95%	0,05
		No interrupts	10%	X	FPGA (2) + MCU (1)	95%	0,05
		Continous interrupts	10%	X	FPGA (2) + MCU (1)	95%	0,05
		Cross-over of interrupts	10%	X	FPGA (2) + MCU (1)	95%	0,05
Systembus	1	unspecified faults	100%	X	FPGA (2) + MCU (1)	95%	0,05
I/O (digital)	0,13	Stuck at 0 (Output)	25%		safe fault		
		Stuck at 1 (Output)	25%	X	MCU (1)	95%	0,001625
		Stuck at 0 (Input)	25%	X	FPGA (1)	90%	0,00325
		Short Circuit between pins	25%	X	MCU (1)	90%	0,003250
Clock	29	unsteady/wrong frequency	50%	X	FPGA (2)	99%	0,145
		no frequency	50%	X	FPGA (2)	100%	0
serial interface (CAN)	0,09	Stuck-at (data or addresses)	13%	X	frame counter + CRC	90%	0,001125
		Time out	13%	X	software timeout	100%	0
		Stuck-at of arbitration signals	12%	X	loss of communication	90%	0,00108
		No, continous or wrong arbitration	12%	X	CRC	90%	0,00108
		Message corruption	13%	X	frame counter + CRC	90%	0,00117
		Message delay	13%	X	software timeout	100%	0
		Message loss	12%	X	software timeout + frame counter	100%	0
		Unintended message repetition	12%	X	frame counter	100%	0
SPI interface	0,09	Stuck-at (data or addresses)	17%	X	framecounter + CRC	99%	0,000153
		Time out	16%	X	software timeout	95%	0,00072
		Message corruption	17%	X	framecounter + CRC	99%	0,000153
		Message delay	17%	X	software timeout	95%	0,000765
		Message loss	16%	X	software timeout	95%	0,00072
		Unintended message repetition	17%	X	framecounter	100%	0
power supply	0,12	supply voltage too low	50%	X	FPGA (2) + brown-out	99%	0,0006
		supply voltage zero	50%		safe fault		
Failures in safety mechanism							
FPGA (XC3S200)	12		100%	X	MCU (2)	90%	1,2
ROM	0,8	Bit flip	50%	X	MCU (2) + 32bit CRC	99%	0,004
		Stuck-at for data and addresses	50%	X	MCU (2) + 32bit CRC	99%	0,004
I/O (digital)	0,13	Stuck at 0 (Output)	25%		safe fault		
		Stuck at 1 (Output)	25%	X	none	0%	0,0325
		Stuck at 0 (Input)	25%	X	none	0%	0,0325
		Short Circuit between pins	25%	X	none	0%	0,0325
Clock	29	unsteady/wrong frequency	50%	X	MCU (2)	90%	1,45
		no frequency	50%	X	MCU (2)	99%	0,145
SPI interface	0,09	Stuck-at (data or addresses)	17%	X	framecounter + CRC	99%	0,000153
		Time out	16%	X	timeout	95%	0,00072
		Message corruption	17%	X	framecounter + CRC	99%	0,000153
		Message delay	17%	X	timeout	95%	0,000765
		Message loss	16%	X	timeout	95%	0,00072
		Unintended message repetition	17%	X	framecounter	100%	0
power supply	0,12	supply voltage too low	50%	X	none	0%	0,045
		supply voltage zero	50%		safe fault		
Sum:	93,2	LFM = 1 - (4,19 / (93,2 - 0,0045))			Latent Faults Metric:	Sum:	4,19
						LFM:	95,50%

Abb. B 4 Latent Fault Metric MCU-FPGA Architektur

- FPGA (1): FPGA aktiviert nur, wenn (speedZero & ignitionOn & button) erfüllt ist
- FPGA (2): FPGA deaktiviert Pumpe, wenn keine oder falsche SPI Nachrichten empfangen werden
- MCU (1): MCU liebt Pumpensteuerleitung zurück und benachrichtigt FPGA
- MCU (2): MCU deaktiviert Dachsteuerung, wenn keine oder falsche SPI Nachrichten empfangen werden

SPFM für Single-Core-Mikrocontroller Architektur gemäß ASIL B

In unten stehender Abbildung ist die Single Point Fault Metric für eine Single-Core Architektur dargestellt.

Component Name	FIT	Failure Mode	Failure Rate Distribution	Single-Core-MCU + externer WD + Brownout Chip			
				possible SPF?	Safety mechanism	Failure Mode Coverage	SPFR
RAM	9,8	Bit flip	33%	X	Safety Task with Software ECC	95%	0,1617
		Stuck-at for data and addresses	33%	X			0,1617
		No, wrong or multiple addressing	34%	X			0,1617
ROM	0,8	Bit flip	50%	X	CRC16 in Range of Safety Task	99%	0,00792
		Stuck-at for data and addresses	50%	X			0,00792
CPU	10	Programmcounter-Flip	10%	X	external WD / Safety Task	92%	0,08
		Stackcounter-Flip	10%	X			0,08
		Internal RAM, register: stuck at fault	10%	X			0,08
		Wrong Coding or no execution	10%	X			0,08
		Stuck at address calculation	10%	X			0,08
		CPU stop working	10%	X			0,08
		overheat affects whole MCU	10%	X			0,08
		No interrupts	10%	X			0,08
		Continuous interrupts	10%	X			0,08
		Cross-over of interrupts	10%	X			0,08
I/O (digital)	0,13	Stuck at 0 (Output)	25%		safe fault		
		Stuck at 1 (Output)	25%	X	online-monitoring	60%	0,013
		Stuck at 0 (Input)	25%	X			0,013
		Short Circuit between pins	25%	X			0,013
Clock	6,8	unsteady/wrong frequency	50%	X	external WD	99%	0,06732
		no frequency	50%	X			0,06732
serial interface (CAN)	0,09	Stuck-at (data or addresses)	13%	X	frame counter	90%	0,001125
		Time out	13%	X			0,001125
		Stuck-at of arbitration signals	12%	X			0,001125
		No, continuous or wrong arbitration	12%	X			0,001125
		Message corruption	13%	X			0,001125
		Message delay	13%	X			0,001125
		Message loss	12%	X			0,001125
Unintended message repetition	12%	X	0,001125				
Systembus	1	system bus error	100%	X	none	0%	1
Watchdog	0,09	Error in WD unit	100%	X	none	0%	0,09
power supply	0,12	supply voltage too low	50%	X	Brownout-Chip + ext. WD	99%	0,0006
		supply voltage zero	50%				
Sum:	28,83	SPFM = 1 - (2,57418 / 28,83)		Single Point Faults Metric:		Sum:	2,57418
						SPFM:	91,07%

Abb. B 5 Single Point Fault Metric für Single-Core Architektur

Der bei dieser Beispielrechnung verwendete Safety Task soll neben den anderen Applikationstasks interruptgesteuert, zyklisch ausgeführt werden, so dass die Einhaltung der Fault Tolerant Timespan gewährleistet wird. Er verwendet einen eigenen Stack sowie einen eigenen Bereich im RAM zur Speicherung seiner Daten und sichert diese durch Software ECC.

Der Task übernimmt die gleichen Aufgaben wie die Überwachungsfunktion in den anderen vorgestellten Architekturen. Hierzu schreibt die Applikation die gelesenen CAN Nachrichten möglichst direkt aus dem CAN Speicher ebenfalls zusätzlich als Kopie in den Speicherbereich des Safety Tasks. Dieser liest an einem getrennten Port redundant die Daten des Bedienelements ein und entscheidet bei der zyklischen Ausführung anhand der vorliegenden Rahmenbedingungen über die Aktivierung des Enable Output. Zur Vermeidung eines Single Point Faults in der MCU wird der Enable Output an einem von der Steuerleitung der Pumpe unabhängigen Port ausgeführt. Zur Überprüfung der digitalen I/O des Mikrocontrollers werden die gesetzten Werte nochmals an einem anderen Port zurück gelesen. Zur Sicherstellung der eigenen Funktionalität triggert der Safety Task den Watchdog, um regelmäßige Ausführung zu gewährleisten und testet den eigenen ROM Bereich mit Hilfe einer CRC16 Checksumme.

Wie man der Rechnung entnehmen kann, sind trotz der großen Menge an Software Sicherheitsmaßnahmen die für ASIL C erforderlichen 97% noch nicht erreicht. Aufgrund des immer größeren Overheads für weitere Sicherheitsmechanismen (insbesondere im Hinblick auf die Einhaltung der Fault Tolerant Timespan) wurde eine weitere Untersuchung in diese Richtung für wenig sinnvoll erachtet.

Es wird deshalb davon ausgegangen, dass eine Single Core MCU lediglich für ASIL B Systeme angemessen ist. Die Diagnostic Coverage Tabelle wird sich entsprechend auf ASIL B beziehen.

Bestimmung der Diagnoseabdeckung (DC)

Aus Abschnitt 8.4.3 der ISO WD 26262-5 ergeben sich Anforderungen an die Diagnoseabdeckung für die verwendeten Bauteile abhängig von ihrer Ausfallrate (Occurrence). Die gesamte Diagnoseabdeckung für ein Bauteil ergibt sich durch den mit den Auftretswahrscheinlichkeiten gewichteten Mittelwert der Diagnoseabdeckung für die einzelnen Fehlerarten, wie sie in den Fehlermetriken aufgeführt sind. Zur Berechnung der Diagnoseabdeckung wurden die Tabellen aus Annex A der ISO WD 26262-5 herangezogen.

Im Folgenden wird für die untersuchten Architekturen dargestellt, welche Anforderungen sich aus der Norm für ASIL C für *residual faults* (RF) und *latent faults* (LF) ergeben und gezeigt, dass die im Rahmen der Studie implementierten Systeme diese erfüllen. In dem Fall, dass sich keine speziellen Anforderungen ergeben, ist das entsprechende Tabellenfeld mit „-“ beschriftet.

DC für Dual-Core-Mikrocontroller Architektur

Anforderungen an die Diagnoseabdeckung				Erreichte DC durch Dual-Core Architektur	
Komponente	Occurence Grad	Forderung DC nach ISO für RF	Forderung DC nach ISO für LF	Erreichte DC nach ISO für RF	Erreichte DC nach ISO für LF
RAM 1	3	≥ 90%	≤ 90% ***	100%	100%
RAM 2	3	≥ 90%	≤ 90% ***	kein SPF	60%
ROM	3	≥ 90%	≤ 90% ***	99%	99%
CPU 1	3	≥ 90%	≤ 90% ***	100%	100%
CPU 2	3	≥ 90%	≤ 90% ***	kein SPF	90%
Clock	>3	≥ 99%	≥ 90%	99%	nur SPF
I/O	2**	-	-	kein SPF	< 60%
CAN	1	-	-	90%	90%
Systembus	2**	SPF	-	0%	-
Watchdog	1	-	-	-	-
Spannungsversorgung	2	≤ 90% ***	-	99%	-

DC für Mikrocontroller-FPGA Architektur

Anforderungen an die Diagnoseabdeckung				Erreichte DC durch MCU-FPGA Architektur	
Komponente	Occurrence Grad	Forderung DC nach ISO für RF	Forderung DC nach ISO für LF	Erreichte DC nach ISO für RF	Erreichte DC nach ISO für LF
RAM	3	≥ 90%	≤ 90% ***	100%	100%
ROM	3	≥ 90%	≤ 90% ***	100%	100%
CPU	>3	≥ 99%	≥ 90%	100%	100%
Systembus	2*	≤ 90% ***	-	100%	95%
I/O	2*	≤ 90% ***	-	kein SPF	100%
Clock	>3	≥ 99%	≥ 90%	100%	100%
CAN	1	≥ 90%	-	93%	93%
SPI	2*	≤ 90% ***	-	100%	100%
Spannungsversorgung	2*	≤ 90% ***	-	100%	99%
FPGA	>3	≥ 99%	≥ 90%	100%	100%
IO	2*	≤ 90% ***	-	100%	100%
Clock	>3	≥ 99%	≥ 90%	100%	100%
SPI	>3	≥ 99%	≥ 90%	100%	100%
Spannungsversorgung	2*	≤ 90% ***	-	100%	0%

DC für Single-Core-Mikrocontroller Architektur

Wie bereits im obigen Abschnitt festgestellt wurde, erscheint eine Umsetzung eines ASIL C Systems auf einer Single-Core-MCU Architektur als wenig sinnvoll. Es soll deshalb der Vollständigkeit halber hier eine Auflistung der Anforderungen an die Diagnoseabdeckung für ASIL B gegeben werden, um zu zeigen, dass die Anforderung erfüllt werden können. Die Latent Fault Metric mit zugehöriger DC Anforderung ist in diesem Fall lediglich optional und wird deshalb hier nicht betrachtet.

Empfehlungen an die Diagnoseabdeckung				Erreichte DC durch Single-Core Architektur	
Komponente	Occurrence Grad	Empfohlene DC nach ISO für RF	Empfohlene DC nach ISO für LF	Erreichte DC nach ISO für SPF	Erreichte DC nach ISO für LF
RAM	3	≥ 90%	-	95%	Nicht betrachtet
ROM	3	≥ 90%	-	99%	
CPU	3	≥ 90%	-	92%	
Clock	>3	≥ 95%	-	99%	
I/O	2	≤ 90% ***	-	90%	
CAN	1	-	-	90%	
Systembus	2	≤ 90% ***	-	0%	
Watchdog	1	-	-	-	
Spannungsversorgung	2	≤ 90% ***	-	99%	

*) Die ISO WD 26262 erlaubt Komponenten, die einen Single Point Fault besitzen, solange diese „Occurrence 2“ nicht überschreiten und die Bauteilqualität durch „Special Measures“ gesichert wird. Weiterhin muss die SPFM durch das Gesamtsystem erfüllt werden.

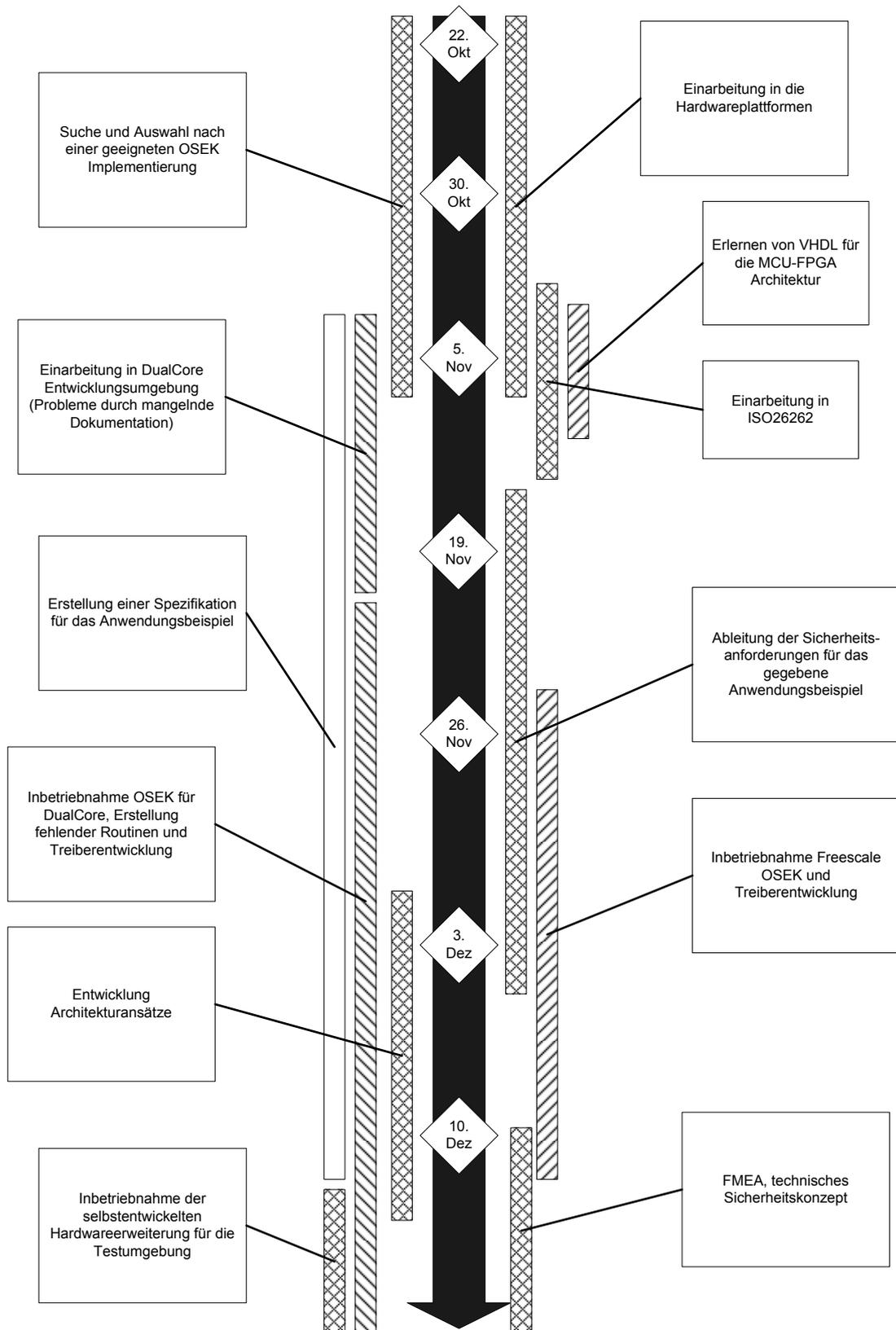
**) Mit „Special Measures“ gemäß ISO WD 26262

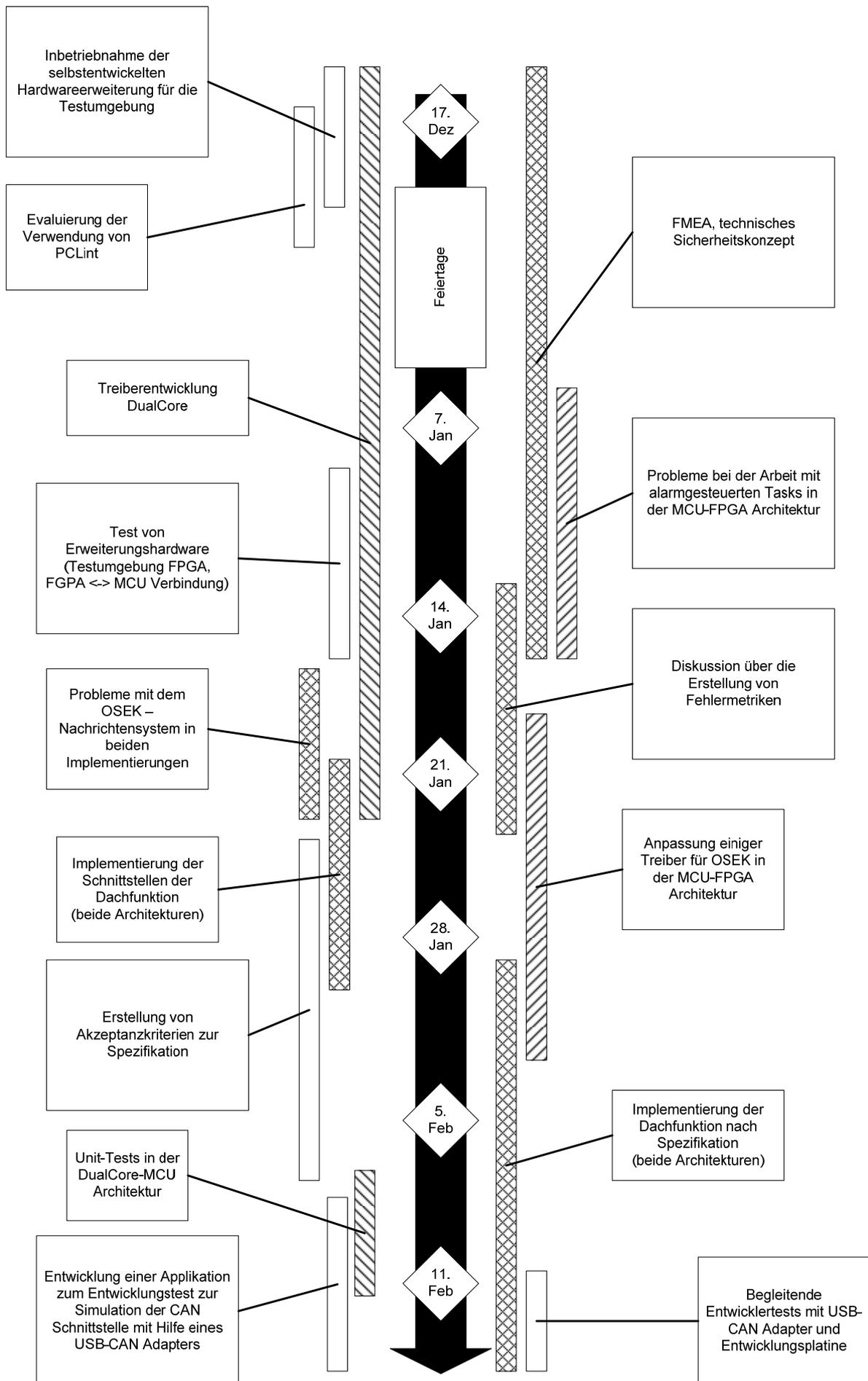
***) Die Anforderung „≤ 90%“ entstammt der Baseline 9 und ersetzt „≥ 60%“ aus Baseline 7 an den markierten Stellen. Diese werden deshalb aufgenommen, obwohl es sich nicht im eigentlichen Sinne um eine Anforderung an die Diagnoseabdeckung handelt.

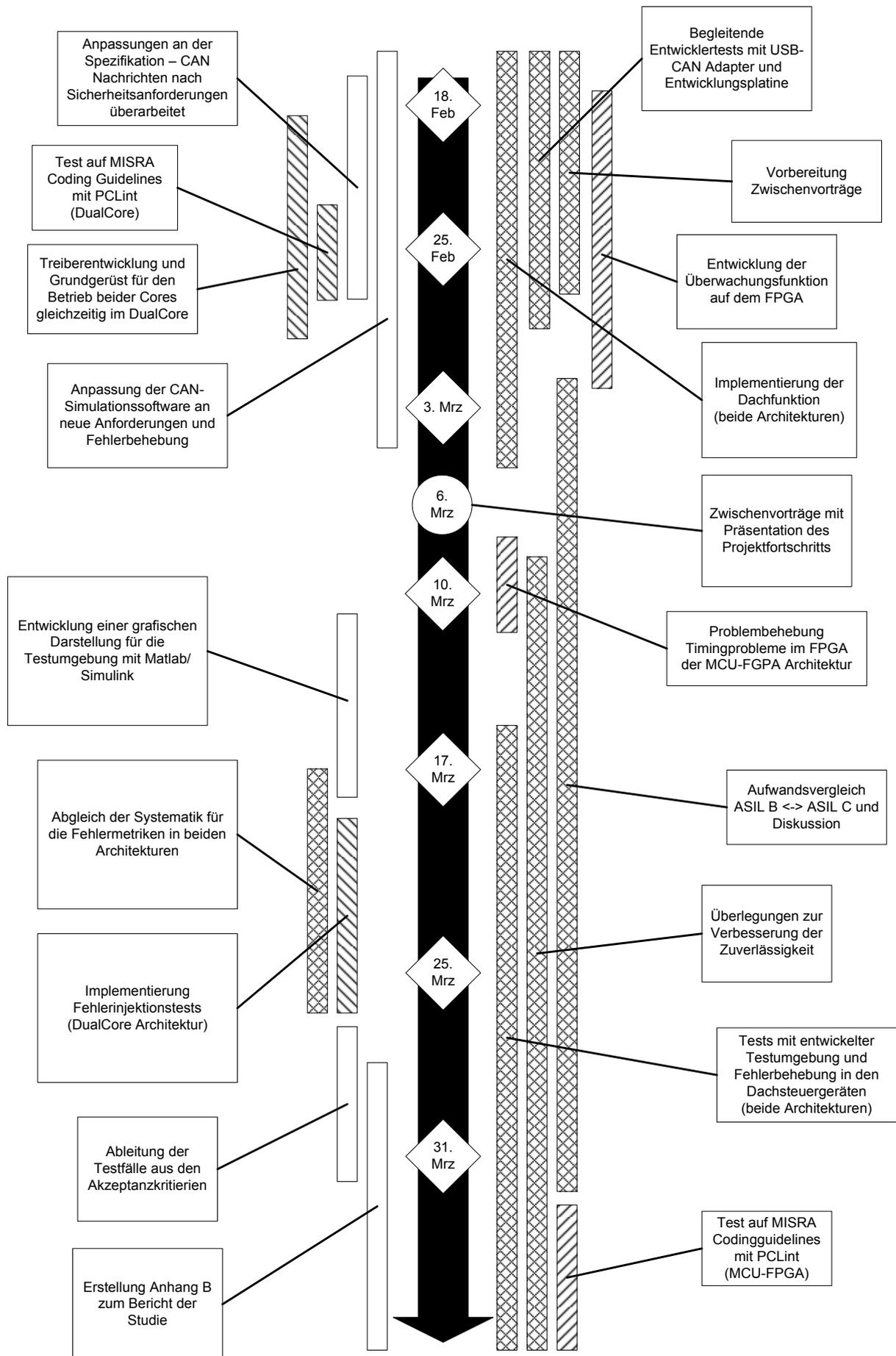
Occurrence 1 entspricht 0,1 FIT, Occurrence 2 entspricht 10*Occurrence 1, Occurrence 3 entspricht 100*Occurrence 1.

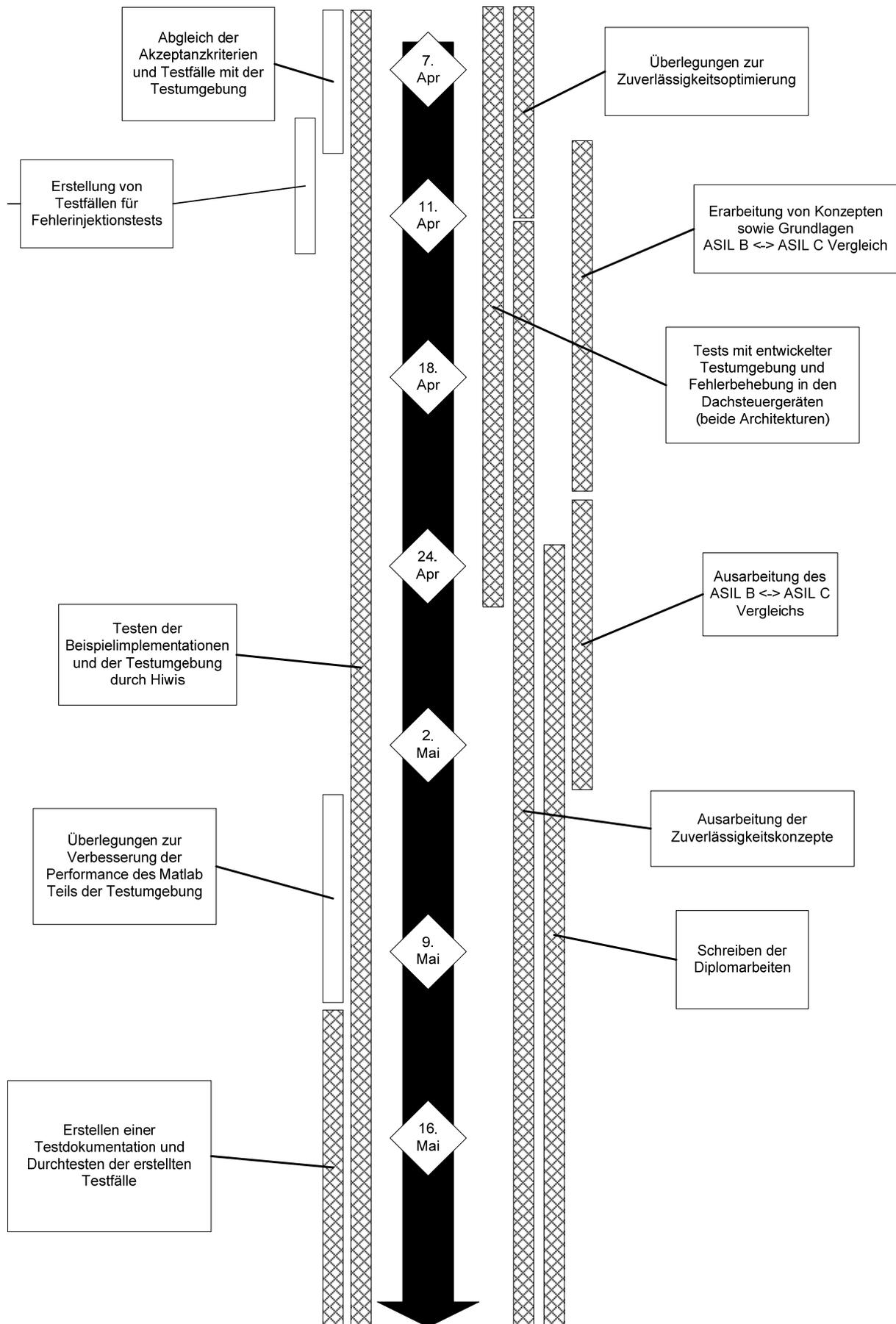
Anhang C: Dokumentation der Arbeitsschritte

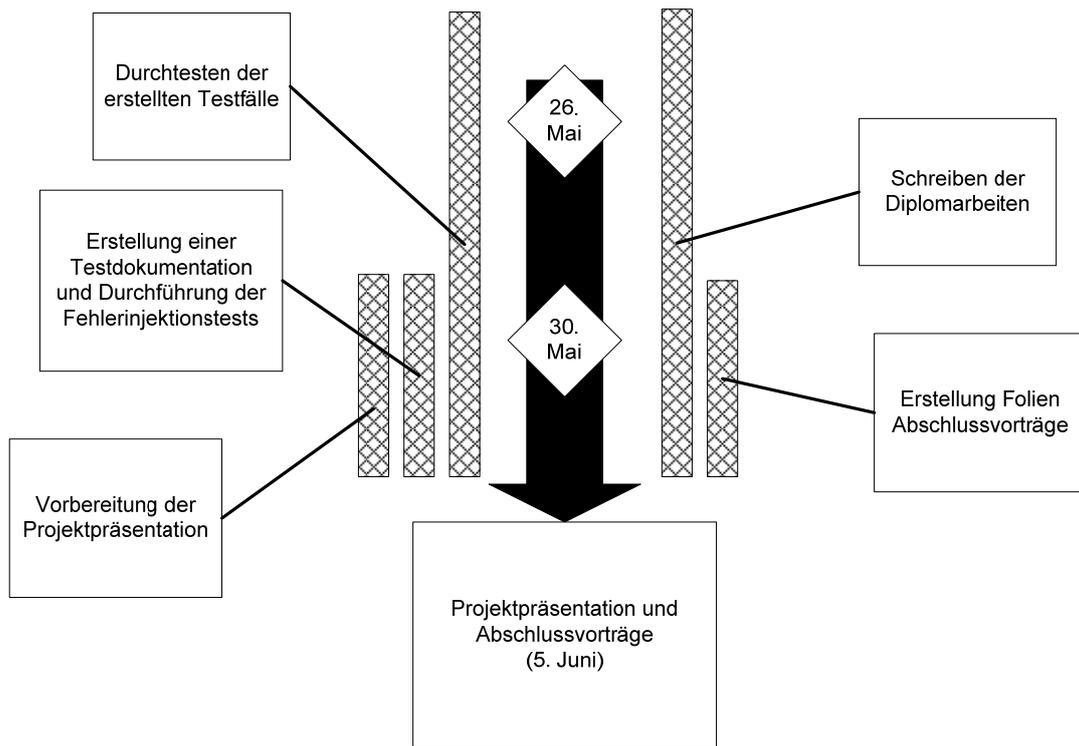
Dieser Anhang enthält eine chronologische Auflistung der durchgeführten Arbeitsschritte. Er soll einen Überblick über den Projektfortschritt und Verzögerungen durch aufgetretene Probleme bieten (eine Legende befindet sich am Ende der Darstellung).



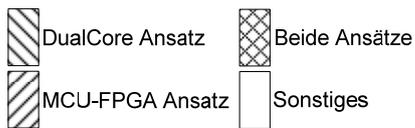








Legende:



Anhang D: Testfälle

Dieser Anhang enthält eine detaillierte Auflistung der verwendeten Testfälle.

Auflistung der Akzeptanztests

Im Rahmen der Erstellung der Systemspezifikation wurden zur späteren Überprüfung der Rahmenbedingungen des implementierten Systems Akzeptanzkriterien abgeleitet. Die Akzeptanzkriterien, sowie die daraus erstellten Testfälle sind in der folgenden Tabelle aufgelistet. Sie sind in die Typen (Q)uality, (R)eliability und (S)afety unterteilt entsprechend ihres Ursprungs in der Spezifikation.

Nr.	Funktionsgruppe	Typ	Testanforderung		erwünschtes Ergebnis	maximale Reaktionszeit	Testdurchführung
			Was?	Wann?			
1	Display	Q	Wird Displaynachricht "Kofferraum wird geöffnet" (ID1) während des Öffnens des Kofferraums angezeigt?	Während Dachbewegung öffnen/schließen	ja	1s	Dachbewegung simulieren, Display beobachten
2	Display	Q	Wird Displaynachricht "Kofferraum wird geschlossen" (ID2) während des Schließens des Kofferraums angezeigt?	Während Dachbewegung öffnen/schließen	ja	1s	Dachbewegung simulieren, Display beobachten
3	Display	Q	Wird Displaynachricht "Dach wird geöffnet" (ID3) während des Öffnens des Dachs angezeigt?	Während Dachbewegung öffnen	ja	1s	Dachbewegung simulieren, Display beobachten
4	Display	Q	Wird Displaynachricht "Dach wird geschlossen" (ID4) während des Schließens des Dachs angezeigt?	Während Dachbewegung schließen	ja	1s	Dachbewegung simulieren, Display beobachten
5	Display	Q	Wird Displaynachricht "Dachoperation komplett" (ID5) nach vollständiger Dachoperation angezeigt?	Nachdem Dachbewegung abgeschlossen	ja	1s	Dachbewegung simulieren, Display beobachten
6	Display	Q	Erlischt die Displaynachricht "Dachoperation komplett" (ID5) spätestens 10s nach Abschluss der Dachoperation wieder?	Nachdem Dachbewegung abgeschlossen	ja	1s	Dachbewegung simulieren, Display beobachten
7	Display	Q	Wird Displaynachricht "Bitte anhalten" (ID6) angezeigt, wenn eine Dachoperation aktiv ist oder angefordert wird, sich das Fahrzeug jedoch in Bewegung befindet?	Während Dachbewegung öffnen/schließen Anforderung Dachbewegung bei geöffnetem/geschlossenem Dach	ja	1s	Dachbewegung simulieren, während Simulation Geschwindigkeit erhöhen, Display beobachten Vor Simulation Geschwindigkeit erhöhen, Dachbewegung simulieren, Display beobachten
8	Display	Q	Displaynachricht "Bitte anhalten" (ID6) erlischt nach spätestens 2s nachdem das Fahrzeug wieder im Stillstand ist?	nach Situation gemäß Nr.7	ja	1s	siehe 7, anschließend Geschwindigkeit auf 0 setzen, Display beobachten
9	Display	Q	Wird Displaynachricht "Kofferraum schließen" (ID7) angezeigt, wenn eine Dachoperation angefordert wird, jedoch der Kofferraum noch offen ist?	Anforderung Dachbewegung bei geöffnetem/geschlossenem Dach	ja	1s	Vor Simulation Kofferraum öffnen, Dachbewegung simulieren, Display beobachten
10	Display	Q	Erlischt die Displaynachricht "Kofferraum schließen" (ID7) spätestens 2s nachdem der Kofferraum geschlossen wird?	nach Situation gemäß Nr.9	ja	1s	siehe 9, anschließend Kofferraum schließen, Display beobachten
11	Display	Q	Wird Displaynachricht "Hydraulikpumpe zu heiß" (ID8) angezeigt, wenn eine Dachoperation wegen zu heißer Hydraulikpumpe verweigert oder angehalten wird?	Während Dachbewegung öffnen/schließen	ja	1s	Dachbewegung simulieren, während Simulation Temperatur der Pumpe auf "zu heiß" setzen, Display beobachten Vor Simulation Temperatur der Pumpe auf "zu heiß" setzen, Dachbewegung simulieren, Display beobachten

Nr.	Funktionsgruppe	Typ	Testanforderung		erwünschtes Ergebnis	maximale Reaktionszeit	Testdurchführung
			Was?	Wann?			
12	Display	Q	Erlischt die Displaynachricht "Hydraulikpumpe zu heiß" (ID8) spätestens 2s nachdem die Hydraulikpumpe wieder in arbeitsfähigem Temperaturbereich ist?	nach Situation gemäß Nr.11	ja	1s	siehe 11, anschließend Temperatur auf "normal" setzen, Display beobachten
13	Display	Q	Wird Displaynachricht "Dachbewegung unterbrochen" (ID9) angezeigt, wenn der Benutzer durch loslassen des Bedienelements die Dachbewegung unterbrochen hat?	Bei unvollständig geöffnetem/geschlossenen Dach	ja	1s	Dachbewegung simulieren, Bedienelement loslassen, Display beobachten
14	Display	Q	Wird Displaynachricht "Interner Busfehler" (ID10) angezeigt, nachdem eine Fehlersituation bei den CAN-Nachrichten aufgetreten ist?	nach Situation gemäß Nr. 36,37,77,78	ja	1s	siehe 36,37,77,78, Display beobachten
15	Seitenfenster	R	Keine Dachbewegung solange Fenster RR über 80%?	Anforderung Dachbewegung	ja	--	Fenster RR auf Error setzen, Dachbewegung simulieren, Dachbewegung beobachten
16	Seitenfenster	R	Keine Dachbewegung solange Fenster RL über 80%?	Anforderung Dachbewegung	ja	--	siehe 15
17	Seitenfenster	R	Keine Dachbewegung solange Fenster FR über 80%?	Anforderung Dachbewegung	ja	--	siehe 15
18	Seitenfenster	R	Keine Dachbewegung solange Fenster FL über 80%?	Anforderung Dachbewegung	ja	--	siehe 15
19	Seitenfenster	R	Keine Dachbewegung, wenn die CAN Nachricht der Fensterposition älter als 200ms ist?	Anforderung Dachbewegung	ja	--	Dachbewegung simulieren, FL/RL auf "stop_sending" setzen, Dachbewegung beobachten Dachbewegung simulieren, FR/RR auf "stop_sending" setzen, Dachbewegung beobachten
20	Seitenfenster	R	Fenster RR wird durch die Steuerung während der Dachoperation gesenkt, wenn es über 80% ist?	vor Dachbewegung öffnen/schließen	ja	1s	Sicherstellen, dass Fenster geschlossen sind, Dachbewegung simulieren, Fenster RR beobachten
21	Seitenfenster	R	Fenster RL wird durch die Steuerung während der Dachoperation gesenkt, wenn es über 80% ist?	vor Dachbewegung öffnen/schließen	ja	1s	siehe 20
22	Seitenfenster	R	Fenster FR wird durch die Steuerung während der Dachoperation gesenkt, wenn es über 80% ist?	vor Dachbewegung öffnen/schließen	ja	1s	siehe 20
23	Seitenfenster	R	Fenster FL wird durch die Steuerung während der Dachoperation gesenkt, wenn es über 80% ist?	vor Dachbewegung öffnen/schließen	ja	1s	siehe 20
24	Seitenfenster	Q	Fenster RR wird durch die Steuerung nach dem Öffnen des Dachs ganz geöffnet.	nach Dachbewegung öffnen	ja	1s	öffnende Dachbewegung simulieren, nach Öffnen Fenster RR beobachten
25	Seitenfenster	Q	Fenster RL wird durch die Steuerung nach dem Öffnen des Dachs ganz geöffnet.	nach Dachbewegung öffnen	ja	1s	siehe 24
26	Seitenfenster	Q	Fenster FR wird durch die Steuerung nach dem Öffnen des Dachs ganz geöffnet.	nach Dachbewegung öffnen	ja	1s	siehe 24
27	Seitenfenster	Q	Fenster FL wird durch die Steuerung nach dem Öffnen des Dachs ganz geöffnet.	nach Dachbewegung öffnen	ja	1s	siehe 24
28	Seitenfenster	Q	Fenster RR wird durch die Steuerung nach dem Schließen des Dachs ganz geschlossen	nach Dachbewegung Schließen	ja	1s	schließende Dachbewegung simulieren, nach Schließen Fenster RR beobachten
29	Seitenfenster	Q	Fenster RL wird durch die Steuerung nach dem Schließen des Dachs ganz geschlossen	nach Dachbewegung Schließen	ja	1s	siehe 28
30	Seitenfenster	Q	Fenster FR wird durch die Steuerung nach dem Schließen des Dachs ganz geschlossen	nach Dachbewegung Schließen	ja	1s	siehe 28

Nr.	Funktionsgruppe	Typ	Testanforderung		erwünschtes Ergebnis	maximale Reaktionszeit	Testdurchführung
			Was?	Wann?			
31	Seitenfenster	Q	Fenster FL wird durch die Steuerung nach dem Schließen des Dachs ganz geschlossen	nach Dachbewegung Schließen	ja	1s	siehe 28
32	Seitenfenster	R	Es wird keine Fensterbewegung angefordert solange keine Dachbewegung angefordert wird	Solange keine Dachbewegung angefordert wird (Bedienelement aktiv)	ja	--	Testumgebung und Steuergerät starten, Fenster beobachten, Fensterposition beliebig modifizieren, Fenster weiter beobachten
33	Geschwindigkeit	S	Dachbewegung stoppt innerhalb von 1s, sobald die CAN Nachricht eine Fortbewegung des Autos anzeigt?	Während Dachbewegung öffnen/schließen	ja	--	Dachbewegung simulieren, während Simulation Geschwindigkeit erhöhen, Dachbewegung beobachten und Reaktionszeit ablesen
34	Geschwindigkeit	S	Dachbewegung wird fortgeführt sobald die Geschwindigkeit wieder 0 beträgt, ohne dass der Benutzer den Taster erneut betätigt?	nach Situation gemäß Nr.33	nein	--	siehe 33, anschließend Geschwindigkeit auf 0 setzen, Dachbewegung beobachten
35	Geschwindigkeit	S	Dachbewegung stoppt innerhalb von 1s, sobald die CAN Nachricht einen Fehler meldet	Während Dachbewegung öffnen/schließen	ja	1s	Dachbewegung simulieren, während Simulation Geschwindigkeitsfehler setzen, Dachbewegung beobachten und Reaktionszeit ablesen
36	Geschwindigkeit	S	Dachbewegung stoppt innerhalb von 1s, falls mehr als 2 aufeinanderfolgende CAN-Nachrichten ausfallen	Während Dachbewegung öffnen/schließen	ja	1s	Dachbewegung simulieren, Geschwindigkeitsnachricht blocken (Do not send Message - Speed) (siehe Wunschliste), Dachbewegung beobachten und Reaktionszeit ablesen
37	Geschwindigkeit	S	Dachbewegung stoppt innerhalb von 1s, falls ein Fehler im Nachrichtenzähler der Geschwindigkeitsnachricht auftritt	Während Dachbewegung öffnen/schließen	ja	1s	Dachbewegung simulieren, Geschwindigkeitsnachrichtenzähler verfälschen (Insert error in counter - Speed), Dachbewegung beobachten und Reaktionszeit ablesen
38	Geschwindigkeit	S	Dachbewegung darf erst wieder aufgenommen werden, nachdem Bedienelement erneut betätigt und min. 2 korrekte Geschwindigkeitsnachrichten empfangen wurden	nach Situation gemäß Nr. 36,78,77,37,38	ja	1s	siehe 36,37, anschließend Fehler beseitigen, Dachbewegung beobachten, Bedienelement erneut betätigen, Dachbewegung beobachten
39	Geschwindigkeit	R	Dachbewegung wird fortgeführt sobald die Geschwindigkeit wieder 0 beträgt und der Taster erneut betätigt wurde?	nach Situation gemäß Nr. 34	ja	1s	siehe 33, anschließend Geschwindigkeit auf 0 setzen, Dachbewegung beobachten, Bedienelement erneut betätigen, Dachbewegung beobachten
40	Hallsensoren	R	Werden die Hallsensoren aktiviert, bevor Werte eingelesen werden?	vor jeder Dachbewegung	ja	--	kein Testfall notwendig, da bei deaktivierten Hallsensoren alle Werte 0 sind und somit keine Dachbewegung möglich ist
41	Hallsensoren	R	Widersprüchliche Werte führen zum Stoppen der Dachbewegung: SP_FL_C und SP_FL_O aktiv?	Während Dachbewegung öffnen/schließen	ja	1s	Dachbewegung simulieren, SP_FL_C und SP_FL_O aktivieren, Dachbewegung beobachten und Reaktionszeit ablesen
42	Hallsensoren	R	Widersprüchliche Werte führen zum Stoppen der Dachbewegung: SP_M_C und SP_M_O aktiv?	Während Dachbewegung öffnen/schließen	ja	1s	siehe 41
43	Hallsensoren	R	Widersprüchliche Werte führen zum Stoppen der Dachbewegung: SP_TFFL_L und SP_TFFL_R länger als 2s unterschiedlich?	Während Dachbewegung öffnen/schließen	ja	1s	Dachbewegung simulieren, SP_TFFL_L und SP_TFFL_R auf unterschiedliche Werte setzen, 2s warten, Dachbewegung beobachten und Reaktionszeit ablesen
44	Hallsensoren	R	Widersprüchliche Werte führen zum Stoppen der Dachbewegung: SP_HL_L und SP_HL_R länger als 2s unterschiedlich?	Während Dachbewegung öffnen/schließen	ja	1s	siehe 43
45	Hallsensoren	R	Werden die Hallsensoren deaktiviert, nachdem die Dachbewegung abgeschlossen wurde?	nach jeder Dachbewegung	ja	10s	Dachbewegung simulieren, nach Abschluss Status der Hallsensoren beobachten
46	Hydraulikpumpe	R	Dachbewegung stoppt innerhalb 1s, wenn die Hydraulikpumpe über 120° C heiß wird?	Während Dachbewegung öffnen/schließen	ja	1s	Dachbewegung simulieren, Temperatur auf 121 setzen, Dachbewegung beobachten und Reaktionszeit ablesen

Nr.	Funktionsgruppe	Typ	Testanforderung		erwünschtes Ergebnis	maximale Reaktionszeit	Testdurchführung
			Was?	Wann?			
47	Hydraulikpumpe	R	Die Dachbewegung "Öffnen" ist in einem Temperaturbereich von 90-120°C der Hydraulikpumpe möglich?	Bei Anforderung einer Dachbewegung öffnen	nein	--	Vor Simulation Temperatur auf 100 setzen, öffnende Dachbewegung simulieren und beobachten
48	Hydraulikpumpe	R	Die Dachbewegung "Schließen" ist in einem Temperaturbereich von 90-120°C der Hydraulikpumpe möglich?	Bei Anforderung einer Dachbewegung schließen	ja	--	Vor Simulation Temperatur auf 100 setzen, schließende Dachbewegung simulieren und beobachten
49	Hydraulikpumpe	R	Eine Dachoperation ist möglich, während die Temperatur der Hydraulikpumpe über 120°C liegt?	Bei Anforderung einer Dachbewegung öffnen/schließen	nein	--	Vor Simulation Temperatur auf 121 setzen, Dachbewegung simulieren und beobachten
50	Hydraulikpumpe	R	Die Dachbewegung "Öffnen" wird auch in einem Temperaturbereich von 90-120°C der Hydraulikpumpe zu Ende geführt	Während Dachbewegung öffnen	ja	--	öffnende Dachbewegung simulieren, Temperatur auf 100 setzen, Dachbewegung beobachten
51	Kofferraum	R	Nach Abschluss der Dachoperation wird in der Applikationsnachricht keine Kofferraumaktion mehr angefordert		ja	--	Dachbewegung simulieren, nach Abschluss Trunk Lock beobachten
52	Kofferraum	R	Wird das Kofferraumschloss geöffnet bevor die Kinematik des Dachs gestartet wird?	Anforderung Dachbewegung	ja	--	Dachbewegung simulieren, vor der Dachbewegung Trunk Lock beobachten
53	Kofferraum	R	Wird die Dachbewegung auch dann begonnen, wenn die CAN Nachricht, welche die Kofferraumschlossinformation enthält älter als 200ms ist?	Anforderung Dachbewegung	nein	--	Vor Simulation Kofferraum Nachrichten unterdrücken (Do not send Message - TL und PS Status), Dachbewegung simulieren und beobachten
54	Kofferraum	R	Wird die Dachbewegung erst begonnen nachdem in der entsprechenden CAN Nachricht signalisiert wurde, dass der Kofferraum nicht mehr im Schloss eingerastet ist	Anforderung Dachbewegung	ja	--	Vor Simulation Trunk Lock auf Error setzen, Dachbewegung simulieren und beobachten, Trunk Lock manuell öffnen, Dachbewegung beobachten
55	Overall	R	Verhält sich das System im Normalfall wie auf dem Ablaufdiagramm spezifiziert?	Während Dachbewegung öffnen/schließen	ja	--	Dachbewegung simulieren, Simulation beobachten
56	Power Striker	R	Dreht sich der Powerstriker Motor zum Ausfahren des Hakens links herum?	Während Dachbewegung öffnen/schließen	ja	--	Dachbewegung simulieren, Motor und Direction beobachten
57	Power Striker	R	Dreht sich der Powerstriker Motor zum Einfahren des Hakens rechts herum?	Während Dachbewegung öffnen/schließen	ja	--	Dachbewegung simulieren, Motor und Direction beobachten
58	Power Striker	R	Nach dem vollständigen Öffnen des Kofferraums wird der Powerstriker ausgefahren	Während Dachbewegung öffnen/schließen	ja	--	Dachbewegung simulieren, warten bis Kofferraum offen, Motor und Direction beobachten
59	Power Striker	R	Wird der Powerstriker erst eingefahren nachdem das Einrasten durch die entsprechende CAN Nachricht signalisiert wurde.	Während Dachbewegung öffnen/schließen	ja	--	Dachbewegung simulieren, wenn Kofferraum geöffnet Trunk Lock auf Error setzen, nach dem Schließen manuell Trunk Lock einrasten, Powerstriker beobachten
60	Power Striker	R	Wird der Motor des Power Strikers gestoppt, wenn nach maximal 2s immer noch nicht genügend Zählimpulse erkannt wurden?	Während der Bewegung des Power Strikers	ja	200ms	Vor Simulation Power Striker auf Error setzen, Dachbewegung simulieren, Motor und Direction beobachten (ggf. Reaktionszeit ablesen)
61	Power Striker	R	Wird beim Bewegen des Power Strikers die definierte Anzahl von Zählimpulsen über den Positionencoder (250) um maximal 5 überschritten?		ja	--	Dachbewegung simulieren, Error message beobachten
62	Side Latch	R	Side Latch wird vor der Dachoperation verriegelt?	vor jeder Dachbewegung	ja	--	Dachbewegung simulieren, Error message beobachten
63	Side Latch	R	Side Latch wird nach der Dachoperation entriegelt?	nach jeder abgeschlossenen Dachbewegung (Endlage erreicht)	ja	2s	Dachbewegung simulieren, Side Latch beobachten
64	Side Latch	R	Dachbewegung ist möglich, wenn beide Endlagerschalter des Side Latch Aktiv sind?	Während Dachbewegung öffnen/schließen	nein	--	Vor Simulation SL_opened und SL_closed aktivieren, Dachbewegung simulieren und beobachten
65	Side Latch	R	Motor reagiert innerhalb von 100ms auf Endlagerschalter?	während der Verriegelung/Entriegelung des Kofferraumes	ja	100ms	Dachbewegung simulieren, Reaktionszeit ablesen

Nr.	Funktionsgruppe	Typ	Testanforderung		erwünschtes Ergebnis	maximale Reaktionszeit	Testdurchführung
			Was?	Wann?			
66	Side Latch	R	Dreht sich der Side Latch Motor zum Verriegeln des Latches links herum?	während der Verriegelung des Kofferraumes	ja	--	Dachbewegung simulieren, Motor und Direction beobachten
67	Side Latch	R	Dreht sich der Side Latch Motor zum Entriegeln des Latches rechts herum?	während der Entriegelung des Kofferraumes	ja	--	Dachbewegung simulieren, Motor und Direction beobachten
68	Stromversorgung	S	Dachbewegung stoppt innerhalb von 1s falls eine instabile Stromversorgung (ggf. quantifizieren) entdeckt wird?	Während Dachbewegung öffnen/schließen	ja	1s	Dachbewegung simulieren, während der Simulation Spannungsversorgung auf 75% absenken, Dachbewegung beobachten
69	Stromversorgung	R	Dachbewegung darf erst wieder aufgenommen werden, nachdem Bedienelement erneut betätigt wurde	Nach Situation gemäß Nr. 68	ja	--	siehe 68, anschließend Spannungsversorgung normalisieren, Dachbewegung beobachten, Bedienelement erneut betätigen, Dachbewegung beobachten
70	Bedienelement	S	Dachbewegung stoppt innerhalb von 100ms nachdem der Benutzer das Bedienelement losgelassen hat?	Während Dachbewegung öffnen/schließen	ja	100ms	Dachbewegung simulieren, Bedienelement loslassen, Dachbewegung beobachten und Reaktionszeit ablesen
71	Bedienelement	R	Eine über das Bedienelement unterbrochene Dachbewegung kann durch erneutes betätigen fortgesetzt werden.	Während Dachbewegung öffnen/schließen	ja	--	Dachbewegung simulieren, Bedienelement loslassen, nach Unterbrechung der Dachbewegung Bedienelement betätigen, Dachbewegung beobachten
72	Bedienelement	S	Sind beide Leitungen des Bedienelements (Schließenanfrage und Öffnenanfrage) aktiv, so wird die Dachbewegung innerhalb von 100ms gestoppt?	Während Dachbewegung öffnen/schließen	ja	100ms	Dachbewegung simulieren, beide Bedienelemente betätigen, Dachbewegung beobachten und Reaktionszeit ablesen
73	Bedienelement	S	Bei aktivem Bedienelement zu Beginn der Operation wird die Dachbewegung nicht gestartet bis das Bedienelement einmal deaktiviert und wieder aktiviert wurde	Beim Start des Steuergeräts	ja	1s	Vor Simulation Bedienelement betätigen, Simulation starten, Dachbewegung beobachten, Bedienelement loslassen und erneut betätigen, Dachbewegung beobachten
74	Zündung	S	Dachbewegung stoppt innerhalb von 100ms nachdem die Zündung ausgeschaltet wurde?	Während Dachbewegung öffnen/schließen	ja	100ms	Dachbewegung simulieren, Zündung ausschalten, Dachbewegung beobachten und Reaktionszeit ablesen
75	Zündung	S	Dachbewegung wird nach Wiedereinschalten der Zündung fortgeführt, ohne ein erneutes betätigen des Bedienelements	nach Situation gemäß Nr. 74	nein	--	siehe 74, anschließend Zündung wieder einschalten, Dachbewegung beobachten
76	Zündung	R	Dachbewegung wird nach Wiedereinschalten der Zündung fortgeführt, nachdem der Benutzer das Bedienelement erneut betätigt hat?	nach Situation gemäß Nr. 74	ja	2s	siehe 74, anschließend Zündung wieder einschalten, Bedienelement loslassen und erneut betätigen, Dachbewegung beobachten
77	Zündung	S	Stoppt das System eine Dachbewegung wenn bei der Nachrichtenzählung der Zündungsnachricht (CAN) eine Diskrepanz auftritt?	Während Dachbewegung öffnen/schließen	ja	100ms	Dachbewegung simulieren, Zündungsnachrichtenzähler verfälschen (Insert error in counter - Ignition status), Dachbewegung beobachten und Reaktionszeit ablesen
78	Zündung	S	Dachbewegung stoppt innerhalb von 100ms, falls keine Zündungsnachrichten (CAN) empfangen werden		ja	100ms	Dachbewegung simulieren, Zündungsnachricht blocken (Do not send Message - Ignition status), Dachbewegung beobachten und Reaktionszeit ablesen
79	Zündung	R	Dachbewegung darf erst wieder aufgenommen werden, nachdem Bedienelement erneut betätigt und min. 2 korrekte Zündungsnachrichten empfangen wurden	nach Situation gemäß Nr. 36, 78, 77, 37,38	ja	1s	siehe 77,78, anschließend Fehler beseitigen, Dachbewegung beobachten, Bedienelement erneut betätigen, Dachbewegung beobachten

Abb. D 1: Akzeptanzkriterien

Auflistung der Fehlerinjektionstests

Zur Simulation zufälliger Hardwarefehler in der MCU der Dachfunktion wurden Fehlersituationen injiziert, wie sie auch durch einen transienten Hardwarefehler auftreten könnten. Die Manipulation wurde so gewählt, dass die Verletzung eines Sicherheitsziels möglich ist. Die folgende Tabelle enthält eine Auflistung aller entworfenen Testfälle nach Komponenten sortiert. Die mit * gekennzeichneten Testfälle wurden im Rahmen der Studie nach negativer Aufwand/Nutzenanalyse nicht durchgeführt.

Die in der folgenden Tabelle aufgeführten maximalen Reaktionszeiten beziehen sich jeweils auf die beschriebene Testdurchführung. Sie bestimmen sich durch die Fault Tolerant Timespan des Safety Goals welches im konkreten Testfall verletzt würde.

Funktionsgruppe	Typ	Fehlerfall		Max. Reaktionszeit	Testdurchführung
		Was?	Wann?		
Flash	S	Bitkipper im Programm-Code	Beim Start des Steuergeräts	100ms	In den Binary-File einen Bitkipper injizieren, Reaktion des Steuergeräts beim Start beobachten
Flash	S	Stuck-At-Address Fehler	immer	100ms	<i>Nicht durchgeführt</i>
RAM	S	Bitkipper im RAM-Speicher, so dass die Enable Variable der Pumpe diese aktiviert	immer	100ms	Bitkipper in Enable Variable per externem Interrupt einfügen, Hydraulikpumpe beobachten
RAM	S	Bitkipper im RAM-Speicher, so dass die Zündungsnachricht verfälscht wird	immer	100ms	Bitkipper in gespeicherte Zündungsnachricht per externem Interrupt einfügen, Hydraulikpumpe beobachten
RAM	S	Bitkipper im RAM-Speicher, so dass die Geschwindigkeitsnachricht verfälscht wird	immer	1s	Bitkipper in gespeicherte Geschwindigkeitsnachricht per externem Interrupt einfügen, Hydraulikpumpe beobachten
RAM	S	Bitkipper im RAM-Speicher, so dass die Bedienelement-Eingabe verfälscht wird	immer	100ms	Bitkipper in gespeicherte Bedienelement-Eingabe per externem Interrupt einfügen, Hydraulikpumpe beobachten
RAM	S	Stuck-At*	Beim Start des Steuergeräts	100ms	Automatisiertes Debugging Interface, welches durch Hardware Breakpoints auf die zu testende RAM-Position die Werte bei Zugriffen verfälscht.
I/O	S	Input Stuck-At high für Bedienelement öffnen	immer	100ms	Vor Simulation den Öffnen Pin auf high legen, Dachbewegung beobachten, Fahrerreaktion durch Drücken des Schließen Bedienelements ausführen, Dachbewegung muss stoppen
I/O	S	Input Stuck-At high für Bedienelement schließen	immer	100ms	Vor Simulation den Schließen Pin auf high legen, Dachbewegung beobachten, Fahrerreaktion durch Drücken des Öffnen Bedienelements ausführen, Dachbewegung muss stoppen
I/O	S	Output Stuck-At high des Hydraulikpumpen Enable Signals	immer	100ms	Output ignorieren, vor/während Simulation an Leitung high/low anlegen (auch an vorhandene Rücklesungen), Hydraulikpumpe beobachten
CPU	S	Program Counter springt an kritische Stelle, so dass die Hydraulikpumpe aktiviert wird	immer	100ms	Program Counter per ext. Interrupt auf sicherheitskritische Stelle setzen, so dass der Enable Ausgang gesetzt wird, Hydraulikpumpe beobachten
CPU	S	Interrupts werden nicht mehr ausgeführt	immer	100ms	Dachbewegung simulieren, Timer-Interrupt per ext. Interrupt deaktivieren, Bedienelement loslassen oder Geschwindigkeit > 0 oder Zündung ausschalten, Hydraulikpumpe beobachten

Funktionsgruppe	Typ	Fehlerfall		Max. Reaktionszeit	Testdurchführung
		Was?	Wann?		
CPU	S	Interrupts werden kontinuierlich ausgeführt	immer	100ms	Timer-Interrupt Flag nicht löschen nach der Interrupt Ausführung (erfordert Eingriff in Program-Code), sicherheitskritische Akzeptanzkriterien testen
CAN	S	Nachrichten kommen zu spät	immer	100ms	Dachbewegung simulieren, Message Delay in der Testumgebung einfügen, Dachbewegung beobachten
CAN	S	Nachrichten kommen gar nicht an	immer	100ms	Dachbewegung simulieren, CAN Error in der Testumgebung einfügen oder Verbindung unterbrechen, Dachbewegung beobachten
Clock	S	Das Clock Signal fällt aus*	immer	100ms	Dachbewegung simulieren, den Oszillator vom Controller trennen, Dachbewegung beobachten
Clock	S	Das Clock Signal kommt unregelmäßig*	immer	100ms	Den Oszillator durch einen einstellbaren Taktgenerator ersetzen, Dachbewegung simulieren, die Frequenz auf verschiedene Werte (sowohl höher als auch niedriger) einstellen, Dachbewegung beobachten
Clock	S	Das Clock Signal ist dauerhaft zu langsam / zu schnell*	immer	100ms	Den Oszillator durch einen Oszillator mit niedrigerer/höherer Taktrate ersetzen, Dachbewegung simulieren, Dachbewegung beobachten
Spannung	S	Die Versorgungsspannung ist zu hoch und führt zu einem Fehler in der Ansteuerung der Pumpe und nicht-deterministischem Verhalten des Controllers*	immer	100ms	siehe hierzu Testfälle für RAM, I/O, CPU Ausfall simulieren durch Power off
Spannung	S	Die Versorgungsspannung ist zu niedrig und führt zu nicht-deterministischem Verhalten des Controllers	immer	100ms	Versorgungsspannung mit Netzteil bereitstellen, vor der Simulation Spannung erniedrigen, jeweils sicherheitskritische Akzeptanzkriterien testen Versorgungsspannung mit Netzteil bereitstellen, Dachbewegung simulieren, während der Simulation Spannung erniedrigen, Dachbewegung beobachten

Abb. D 2: Testfälle Fehlerinjektion

Anhang E: Aufbau der Test- und Simulationsumgebung

Anforderungen

Die Steuergeräte reagieren auf 10 diskrete Hall-Sensoren, zwei diskrete Sensoren für den Side-Latch, einem PWM Signal des Power Strikers, die analoge Temperaturangabe und auf die von der Test- und Simulationsumgebung gesendeten CAN-Nachrichten. Angesteuert werden von den Steuergeräten zwei Elektromotoren (Power Striker und Side Latch) und eine Hydraulikpumpe sowie zwei Hydraulikventile für die eigentliche Dachbewegung (Roof, Trunk Lock und Package Lock). Des Weiteren werden die Bewegungen der vier Seitenfenster über eine CAN-Nachricht von den Steuergeräten angefordert. Der CAN Bus wird zudem genutzt um eine Displaynachricht und eigene Statusinformationen zu senden (Siehe auch Abbildung 4-2 Schnittstellenbeschreibung der Cabrio-Verdecksteuerung). Für eine Simulation der Umgebung der Steuergeräte und die Durchführung von verschiedenen Tests ergeben sich folgende Aufgaben, die die Test- und Simulationsumgebung erfüllen muss:

- Simulation der benötigten Umgebung, um das Steuergerät auf Funktionalität und Sicherheits- und Zuverlässigkeitsanforderungen hin überprüfen zu können. Dies beinhaltet die Fähigkeit in Echtzeit auf die Signale des Steuergeräts zu reagieren (d.h. Ergebnisse der Berechnungen müssen innerhalb gegebener Zeitintervalle vorliegen)
- Überprüfung der Reaktionszeiten der Steuergeräte (nötig zur Überprüfung der Akzeptanzkriterien)
- Möglichkeit
 - manueller Tests über geeignete Benutzerschnittstelle (schnelle Testdurchführung zur Fehlersuche)
 - automatischer Testdurchläufe (hohe Testabdeckung zur Verifikation) mit geeigneter Datenspeicherung für eine automatisierte Auswertung
- Fehlerinjektion in der simulierten Umgebung um Fehlerbehandlung durch das Steuergerät zu überprüfen (z.B. Anhalten der Aktuatoren, falsche Werte für Sensoren, Manipulation der CAN-Nachrichten). Die Fehlerinjektion soll sowohl manuell als auch automatisch erfolgen können.

Zu diesem Zweck wurde ein Hardware-in-the-Loop (HiL) System entwickelt, welches im folgenden Abschnitt kurz beschrieben wird. Weitere Details zur Testumgebung sind der entsprechenden Diplomarbeit [Dülks] zu entnehmen.

Aufbau

Obwohl eine Umsetzung der Testumgebung auf einem Personal Computer (PC) aus Kosten und Flexibilitätsgründen sehr attraktiv erscheint, ist dieser Ansatz ohne zusätzliche Hardwarekomponenten auf Grund der geforderten Schnittstellen und erforderlichen Reaktionen in Echtzeit nur schwerlich umzusetzen. Aus diesem Grund wurde hier ein PC (Pentium 4, 3GHz, 1GB RAM) in Kombination mit einem FPGA (Xilinx Spartan-3 Starter Kit (XC3S200)) verwendet. In dieser Kombination werden die geforderten Schnittstellen und Echtzeitaufgaben durch den FPGA umgesetzt. Der Datenaustausch zwischen PC und FPGA findet über eine serielle Schnittstelle (RS232) statt. Programmiert wurde der FPGA in VHDL (Very High Speed Integrated Circuit Hardware Description Language), für die Realisierungen auf der PC-Seite wurde im Hinblick auf Wartung und Wiederverwendung Matlab/ Simulink/ Stateflow verwendet. Wie bereits erwähnt wurden zeitkritische Aufgaben auf dem FPGA realisiert, während die restlichen Aufgaben (Benutzerschnittstelle, Modell der Umgebung, etc.) auf dem PC umgesetzt wurden. Die resultierende Architektur ist in Abbildung E 1 dargestellt und wird im folgenden Abschnitt näher beschrieben.

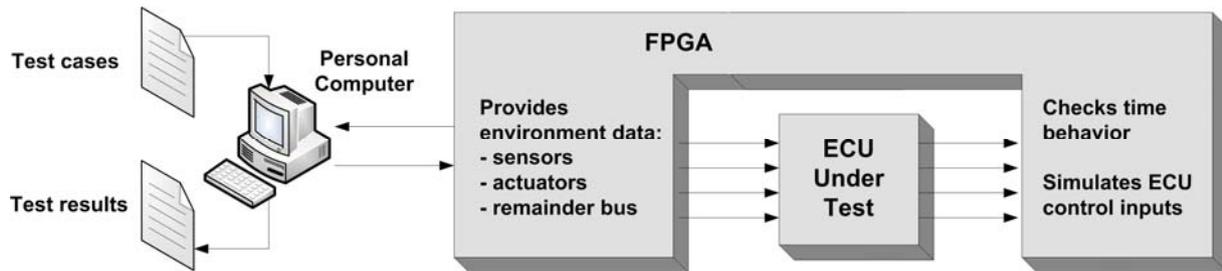


Abb. E1: Architektur der Test- und Simulationsumgebung

Realisierung

Wie in Abbildung E2 zu sehen ist, werden die graphische Benutzerschnittstelle (GUI), die Testautomatisierung und nicht zeitkritische Simulationen auf dem PC realisiert.

Die GUI wurde in Matlab erstellt. Elemente, die viele Test- und Simulationsumgebungen gemeinsam haben, werden in getrennten Text-Dateien beschrieben. Zu diesen Elementen gehören Aktuatoren, Sensoren und Zeitmessungen und deren Ausgabe. Die Text-Datei für die Zeitmessungen beinhaltet pro Zeile die Textangabe, die in der GUI neben dem Ergebnisfeld angezeigt werden soll. Basierend auf dieser Datei kann das Panel für die Zeitmessungen dann automatisch aufgebaut werden. Bei der Text-Datei für die Sensoren sind pro Zeile der Sensorname und sein Initialwert anzugeben, der Aufbau der GUI erfolgt bei Programmstart

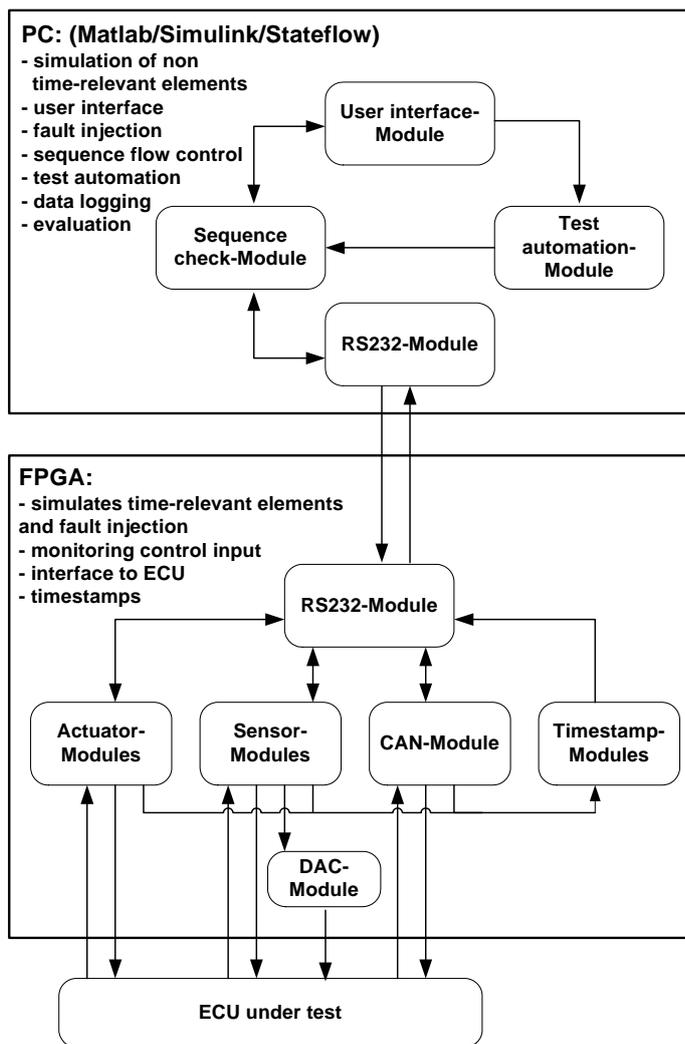


Abb. E2: Aufgabenverteilung zwischen PC und FPGA

ebenfalls automatisch. Die Aktuatoren-Text-Datei besteht aus Aktuatorname, Wertebereich (Max, Min), Initialwert und einem String für die verschiedenen Fehlermeldungen. Jede Fehlermeldung wird durch „|“ getrennt und es dürfen keine Leerzeichen enthalten sein. Möglich ist auch eine Unterteilung der Aktuatoren, Sensoren und Zeitmessungen. Für jede Unterteilung ist eine eigene Text-Datei zu erstellen und einem Panel zuzuweisen. Dieses Panel wird dann automatisch aufgebaut. So können Änderungen schnell und einfach in einer dieser Textdateien vorgenommen werden und die GUI später für andere Test- und Simulationsumgebungen wieder verwendet werden.

Der Tester kann über die GUI selbst Fehler injizieren oder eine automatische Testdurchführung aktivieren. Weiterhin lassen sich die Systemzustände in dieser GUI ablesen (z.B. die Position der Aktuatoren, siehe auch Abb. E3).

Die zeitunkritischen Simulationen auf dem PC beinhalten in erster Linie ereignisgesteuerte Abfolgen (z.B. Ansteuerung muss so lange erfolgen

bis die geforderte Position erreicht ist.) In den Fällen, in denen eine bestimmte Reaktionszeit des Steuergerätes nicht überschritten werden darf, werden diese Signale vom FPGA beim Einlesen mit einem Zeitstempel versehen. Ob die Reaktion zeitlich korrekt war kann dann vom PC anhand dieser Zeitstempelinformation ausgewertet werden. Diese Vorgehensweise ermöglicht es einen großen Teil der Simulation auf dem nicht echtzeitfähigen PC in einem Stateflow-Modul zu realisieren. Dies ermöglicht eine einfache Umsetzung der gewünschten Abläufe zur Simulation der Umgebung der Steuergeräte.

Für die geforderte Datenspeicherung wird jedes Datenpaket, das über die serielle Schnittstelle zum FPGA versendet oder von diesem empfangen wird, in einer Textdatei gespeichert. Um die gesendeten Testdaten und die zugehörigen Testergebnisse zuordnen zu können, werden die Nachrichten mit einem Ringzähler versehen, der mit jeder an den FPGA gesendeten Nachricht erhöht wird. Die Daten, die vom FPGA zum PC geschickt werden, beinhalten den zuletzt empfangenen Ringzählerwert. Somit lassen sich die durchgeführten Tests offline detailliert auswerten.

Auf dem FPGA werden die zeitkritischen Simulationen (in erster Linie Erzeugung von Echtzeitsignalen) realisiert. Da Zustände und die Ein- und Ausgänge der Testumgebung in festen Intervallen gespeichert werden, können schnelle Änderungen der Ausgänge des Prüflings so unter Umständen nicht erkannt werden. Aus diesem Grund dokumentieren die Aktuator-Module zusätzlich eventuell auftretende Signalwechsel zwischen zwei Speicherzeitpunkten und stellen diese Information späteren Auswertungen zur Verfügung. Somit ist es möglich Änderungen im Millisekundenbereich (bei verwendetem FPGA sogar bis 20ns) zu erkennen. Die Auflösung der Zeitstempel für die eingelesenen Signale beträgt je nach Eingangssignal 1ms bis 10ms und es könnten bei Bedarf ebenfalls Werte von 20ns realisiert werden. Somit lassen sich Zeitmessungen durchführen, die beispielsweise für die Validierung der Akzeptanzkriterien benötigt werden. Gestartet wird eine Zeitmessung, wenn der in den Akzeptanzkriterien beschriebene Systemzustand vorliegt und angehalten, wenn die entsprechend geforderte Reaktion des Steuergerätes erfolgt. Um den Datenaustausch zwischen PC und FPGA gering zu halten wurden zwei Arten der Zeitmessung realisiert. Zum einen gibt es Zeitmessungen, die die gemessenen Zeitspannen ausgeben während der zweite Typ die gemessene Zeitspanne direkt auf dem FPGA auswertet und lediglich das Ergebnis versendet.

Fehlerinjektions- und Eingabemöglichkeiten

Im Folgenden werden die Fehlerinjektions- und Eingabemöglichkeiten aufgezählt, die in der Test- und Simulationsumgebung realisiert wurden. Abbildung E3 zeigt die grafische Benutzerschnittstelle, über die in diesem Zusammenhang manuell folgende Fehler in das Verhalten der Umgebung eingebracht werden können:

- Manipulation der Aktuatoren:
 - Anhalten der Simulation einzelner oder mehrerer Aktuatoren
 - Manuelle Positionsveränderung (Ausnahmen: Power Striker: keine manuelle Bewegung möglich; Fenster: nur, wenn keine Dachbewegung statt findet)

- Manipulation der Sensoren
 - Sensorwerte können manuell auf permanent aktiv oder permanent inaktiv gesetzt werden
 - Fehlerinjektion kann für alle Sensoren zu jeder Zeit erfolgen und auch wieder rückgängig gemacht werden

- Manipulation der CAN-Nachrichten
 - Das Senden einzelner Nachrichten kann unabhängig von anderen Nachrichten deaktiviert werden.
 - Bei der Geschwindigkeits- und Zündungs-Nachricht können Fehler für die in diesen Nachrichten integrierten Ringzählern injiziert werden. Des Weiteren können 1, 2 oder 3 Nachrichtensendungen unterdrückt werden (diese Injektion wird nach der Unterdrückung automatisch zurück gesetzt)

- Weitere Schnittstellen
 - Trunk Lock: Zeigt an, ob der Kofferraum vollständig geschlossen ist oder offen ist. Manuell kann dieser Sensor ein- und ausgeschaltet werden; eine Fehlerinjektion hält aktuellen Zustand fest und unterdrückt die Ansteuerung des Steuergeräts
 - Zündung (Ignition): Bei Ignition kann die Zündung ein- und ausgeschaltet werden. Bei Aktivierung einer Fehlerinjektion wird die gesendete CAN-Information auf einen definierten Fehlerwert geändert.
 - Temperatur der Hydraulikpumpe: Kann beliebig eingegeben werden, so dass eine Reaktion bei einer warmen bzw. heißen Pumpe überprüft werden kann
 - Geschwindigkeit des Fahrzeugs (Fehler: Dachbewegung während der Fahrt)
 - Bedienelemente (Fehler: gleichzeitige Aktivierung der Taster für *Dach auf* und *Dach zu*).

Mit Hilfe dieser Eingabemöglichkeiten für Zustände und simulierte Fehler, wie z.B. *stecken bleiben eines Aktuators* oder *falscher Wert bei einem Sensor*, können die Akzeptanzkriterien aus [Spezifikation] überprüft werden. Die Abbildung E4 zeigt schließlich die Test- und Simulationsumgebung im Einsatz.

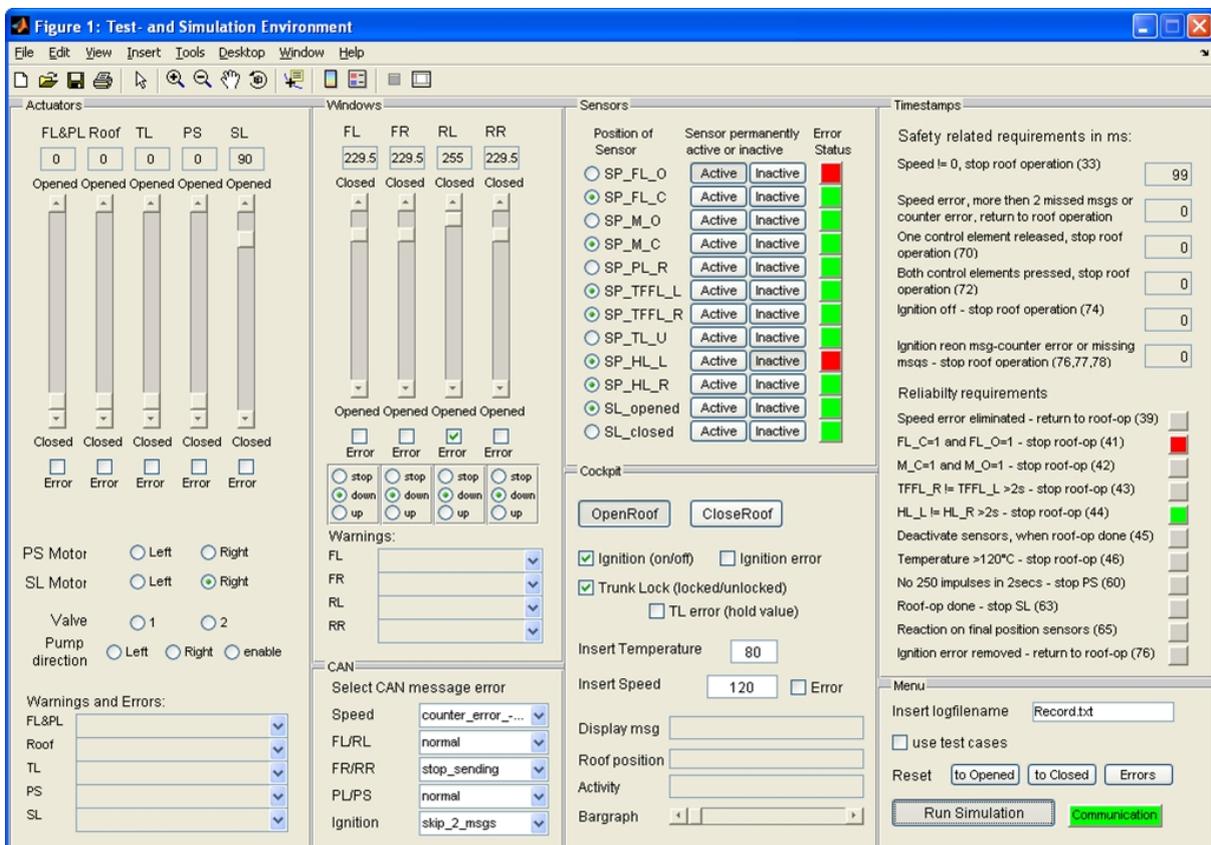


Abb. E3: grafische Benutzerschnittstelle

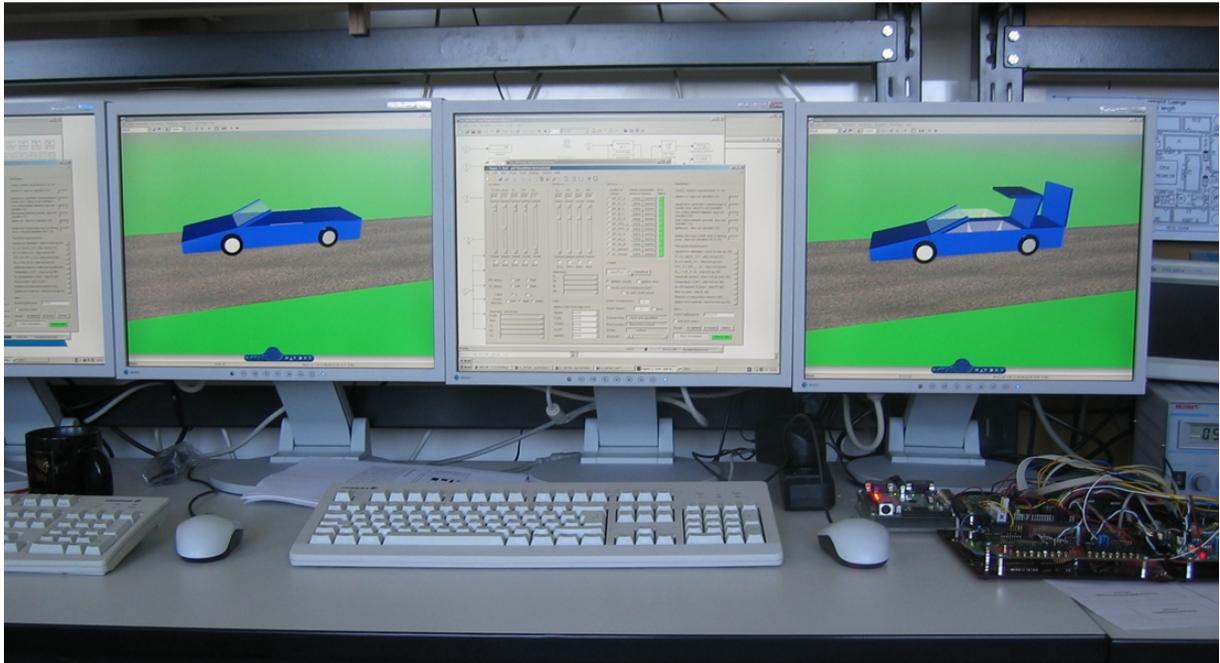


Abb. E4: Test- und Simulationsumgebung im Einsatz

Anhang F: Umfrage zu Trends und Herausforderungen in der Automobilelektronik

1. Hintergrund

Als Grundlage für die Untersuchungen im Arbeitskreis wurde eine Umfrage zu relevanten Trends in der Automobilelektronik durchgeführt. Als Zeithorizont sollten Trends aufgeführt werden, die bis zum Jahre 2015 Anwendung finden werden. Die Umfrage wurde auf Basis eines Fragebogens durchgeführt, der im Abschnitt 3 angefügt ist.

2. Ergebnisse

Eine Übersicht über die erhaltenen Umfrageergebnisse ist in Abbildung F1 grafisch dargestellt. In dieser Umfrage wurde zwischen eine Bewertung der Relevanz der Trends und einer Bewertung derer Problematik unterschieden:

- Relevanz:**
- 0: keine Relevanz (kein Trend)
 - 1: sehr geringe Relevanz (unwahrscheinlicher, aber möglicher Trend)
 - 2: geringe Relevanz (möglicher Trend, aber 2015 noch nicht relevant)
 - 3: mittlere Relevanz (Trend, aber bedeutendere Trends vorhanden)
 - 4: starke Relevanz (gehört zu den bedeutendsten Trends bis 2015)
- Problematik:**
- 0: keine Problematik (durch Trend keine bes. Problemlösungen erforderlich)
 - 1: sehr geringe Problematik
 - 2: geringe Problematik
 - 3: mittlere Problematik
 - 4: hohe Problematik (Trend erfordert umfassende Entwicklung neuer Problemlösungen)

Folgende Fragestellungen aus der Umfrage wurden dabei auf Grund hoher Relevanz und Problematik als besonders interessant erachtet:

- 3: Standardisierte ECU aus Billiglohnländern
- 7: Heterogene Strukturen (MCU, FPGA, Dual Core)
- 8: Multi-Core
- 9: HW/SW Trennung
- 12: Anzahl ECUs sinkt
- 13: AUTOSAR (niedrige Problematik)
- 17: Komfortfunktionen werden sicherheitskritisch
- 18: Wachsende Anzahl von Systemen die sowohl sicherheitskritisch als auch hoch verfügbar sein müssen
- 23: autom. Code-Generierung

Auf dieser Basis wurde im Arbeitskreis die Durchführung der im Bericht vorgestellten Studie erarbeitet.

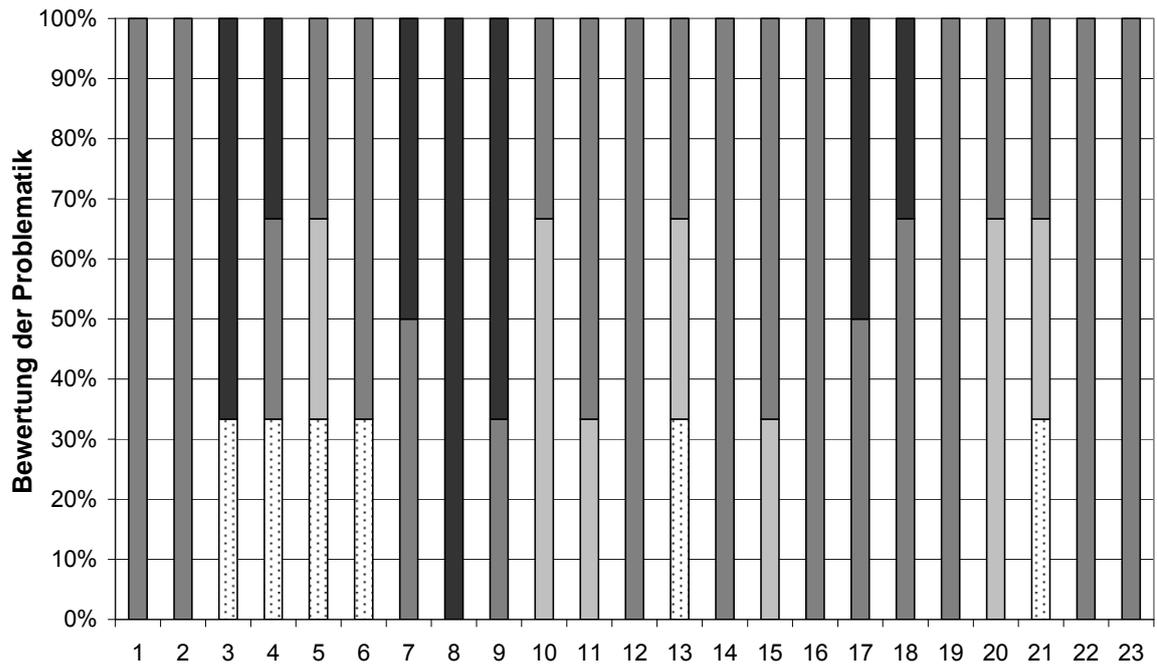
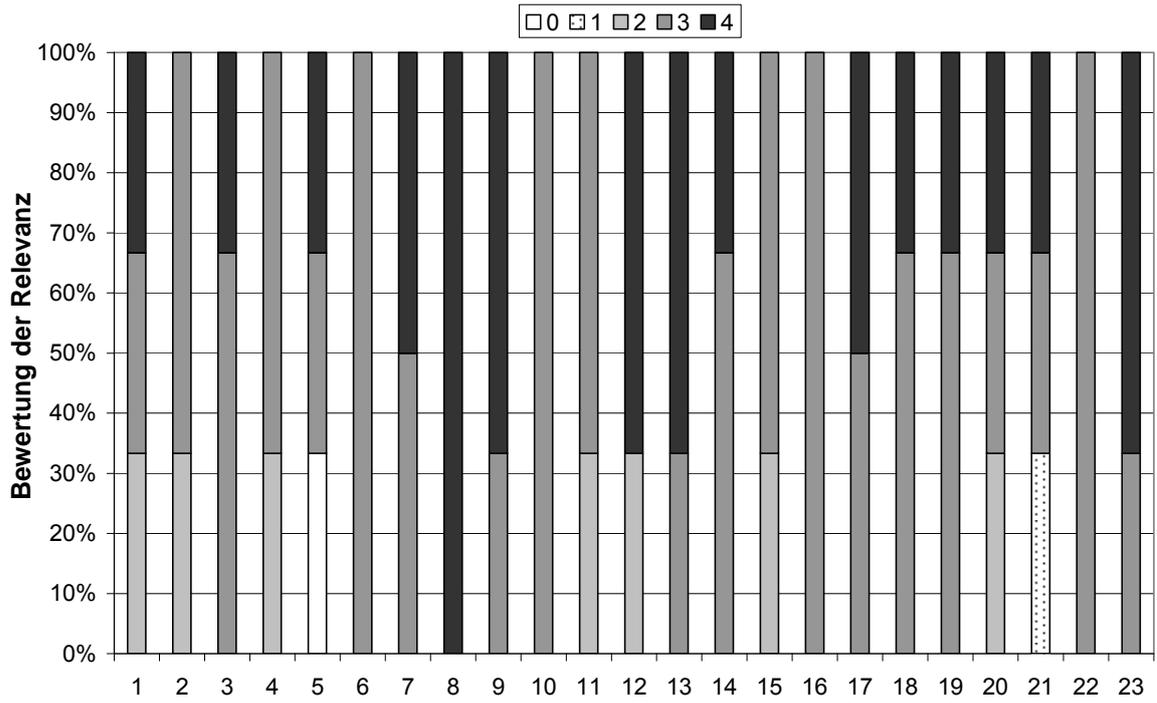


Abb. D 3: Ergebnisse der Umfrage

3. Verwendeter Fragebogen zur Trendermittlung in der Automobilelektronik

Das FAT-Projekt *Zuverlässige Automotive Embedded Systems* beschäftigt sich mit Auswirkungen aktueller Technologie-Trends auf die Zuverlässigkeit der Automobilelektronik. Betrachtet werden dabei Hardware- und Softwareaspekte sowie Wechselwirkungen zwischen Hardware- und Software.

Der hier vorliegende Fragebogen soll zur Ermittlung relevanter Trends in der Automobilelektronik bis zum Jahr 2015 dienen. Um möglichst repräsentative Ergebnisse zu erlangen ist eine aktive Teilnahme aller Projektpartner an der Umfrage wünschenswert.

Der Fragebogen kann einerseits ausgefüllt per Email (salewski@informatik.rwth-aachen.de) oder Fax (0241/80-22150) an Herrn Salewski zurückgesendet werden, andererseits können wir die Antworten auch gerne in einem persönlichen Gespräch mit Ihnen in Ihrem Unternehmen erarbeiten.

0. Personenbezogene Daten

Name:	
Funktion:	
Firma:	

1. Allgemeine Fragen

1.1 Nennen Sie den für Sie und Ihr Unternehmen wichtigsten **Technologietrend** in der **Automobilelektronik** bis zum Jahr 2015 (und ggf. daraus resultierende Probleme):

1.2 Nennen Sie den für Sie und Ihr Unternehmen wichtigsten **allgemeinen Trend** in der **Automobilelektronik** bis zum Jahr 2015 (und ggf. daraus resultierende Probleme):

2. Bewertung von Trends

In diesem Abschnitt werden Sie gebeten Trends (aus einem Brainstorming beim Kickoff Treffen) hinsichtlich Ihrer Relevanz und Ihrer Problematik zu beurteilen. Dabei sind folgende Bewertungen möglich:

- Relevanz R:**
- 0: keine Relevanz (kein Trend)
 - 1: sehr geringe Relevanz (unwahrscheinlicher, aber möglicher Trend)
 - 2: geringe Relevanz (möglicher Trend, aber 2015 noch nicht relevant)
 - 3: mittlere Relevanz (Trend, aber bedeutendere Trends vorhanden)
 - 4: starke Relevanz (gehört zu den bedeutendsten Trends bis 2015)

- Problematik P:**
- 0: keine Problematik (durch Trend keine besonderen Problemlösungen erforderlich)
 - 1: sehr geringe Problematik
 - 2: geringe Problematik
 - 3: mittlere Problematik
 - 4: hohe Problematik (Trend erfordert umfassende Entwicklung neuer Problemlösungen)

Weiterhin können Sie zu einzelnen Trends Problem und Fragestellungen ergänzen. Markieren Sie diese **Ergänzungen** bitte blau.

Trend	Problem/Fragestellung	R [0..4]	P [0..4]
Steigende Taktfrequenzen der Steuergeräte	Höhere Empfindlichkeit gegenüber Störungen (Bitkipper etc.)		
Schrumpfende Strukturgrößen der Steuergeräte	Höhere Empfindlichkeit gegenüber Störungen (Bitkipper etc.)		
Standardisierte Low Cost Steuergeräte aus Niedriglohnländern	Wie müssen diese Steuergeräte aussehen? Werden diese Steuergeräte den Zuverlässigkeitsanforderungen gerecht?		
Standardisierung von Steuergeräten in einzelnen Domänen (z.B. Body-Domain)	Wie müssen diese Steuergeräte aussehen? Welche Anforderungen gibt es an die Zuverlässigkeit dieser Steuergeräte?		
Einsatz von Mikrocontrollern mit in Hardware integrierten HW-Fehler Toleranzmechanismen	Welche Vor- u. Nachteile bieten diese Controller? Einsatz in sicherheitskritischen Anwendungen oder grundsätzlich um Verfügbarkeit zu erhöhen?		
Einsatz von Controllern mit automobil-spezifischen Prozessoren (FPGA/ASIC Design)	Welche Vor- u. Nachteile bieten diese Controller?		
Heterogene Strukturen (FPGA, MCU, Dual-Core,...)			
Multicore-Systeme	Auswirkungen auf Zuverlässigkeit: Fluch oder Segen?		
Trennung: Funktion ⇔ Steuergeräte Hardware; Software als Produkt	Geht das bei heterogenen Strukturen? Geht das bei hoch zuverlässigen Systemen? Auswirkung auf Zuverlässigkeit des Gesamtsystems?		
Smart Sensorik/Aktuatorik	Auswirkungen auf die Steuergeräte und deren Zuverlässigkeit?		

Trend	Problem/Fragestellung	R [0..4]	P [0..4]
Verbesserte Abschottung einzelner SW Module untereinander	Welche Anforderungen stellen sich dadurch an die Steuergeräte?		
Anzahl der Steuergeräte steigt nicht weiter oder sinkt sogar (Zentralisierung) → leistungsstärkere Steuergeräte	Welche Anforderungen stellen sich an diese leistungsstärkeren Steuergeräte? Sind diese Steuergeräte Bestandteil von sicherheitskritischen Anwendungen (z.B. Drive by Wire)?		
Steigende Anzahl von AUTOSAR konformen Steuergeräten	Besondere Anforderungen für Einsatz in sicherheitskritischen Anwendungen?		
Qualitativ höherwertige Spezifikationen, z.B. durch ausführbare Spezifikationen und Modellbasierte entwicklung	Auswirkung auf Steuergeräte?		
Mehr Bedarf an Security	Anforderungen an Steuergeräte?		
Anzahl sicherheitskritischer Funktionen steigt	Anforderungen an Steuergeräte, Entwurfsprozess,		
Komfortfunktionen werden zunehmend sicherheitskritisch (Fahrerassistenzsysteme)	Anforderungen an Steuergeräte, Entwurfsprozess,		
Wachsende Anzahl von Systemen die sowohl sicherheitskritisch als auch hoch verfügbar sein müssen (z.B. Drive-by-Wire)	Anforderungen an Steuergeräte, Entwurfsprozess,		
Wachsende Anzahl von Problemen auf System-Ebene	Auswirkung auf Steuergeräte?		
Steigende Wiederverwendung von SW	Auswirkungen auf Zuverlässigkeit abhängig von verwendeter Hardware Plattform?		
Funktions- Reuse ja ↔ Code- Reuse nein	Auswirkung auf Steuergeräte?		
Umfang von SW Funktionen steigt	Wie wird man der steigenden Komplexität gerecht? Wie können HW-Plattformen dazu beitragen?		
Automatische Code-Generierung in sicherheitskritischen Anwendungen	Auswirkung auf Steuergeräte? Auswirkungen auf den Entwurfsprozess? Probleme verschieben sich z.T. in Codegeneratoren		

3. Weitere eigene Trenderwartungen

In diesem Abschnitt können Sie eigene Trends (Bitte im gleichen Format wie im vorangegangenen Abschnitt) ergänzen.

Trend	Problem/Fragestellung	R [0..4]	P [0..4]
Eigener Trend (bis 2015)	Problem das daraus für Automobilelektronik folgt		

4. Probleme aus der Praxis

In diesem Abschnitt werden Sie gebeten, die letzten 10 großen Fehler, die in Automobilelektronik (HW, SW, HW/SW) in ihrem Unternehmen aufgetreten sind aufzuführen. Diese Daten werden für sämtliche Präsentationen selbstverständlich anonymisiert.

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

5. Abschließende Anmerkungen

Sie können uns gerne Anmerkungen und Anregungen zur Umfrage in untenstehendem Feld geben.

Vielen Dank für Ihre Teilnahme !

Abkürzungsverzeichnis

ASIL	Automotive Safety Integrity Level (ISO 26262)
CCF	Common Cause Failure (ISO 26262)
DC	Diagnostic Coverage (ISO 26262)
ECU	Electronic Control Unit
FPGA	Field Programmable Gate Array
LF	Latent Fault (ISO 26262)
MCU	Mikrocontroller Unit
RF	Residual Fault (ISO 26262)
SPF	Single Point Fault (ISO 26262)
VHDL	(Very High Speed Integrated Circuit) Hardware Description Language

Bisher in der FAT-Schriftenreihe erschienen (ab 2005)

Nr.	Titel	Preis / €
187	Zell- und molekularbiologische Untersuchungen zur DNS-schädigenden Wirkung des Rußkerns in einem Multi-Dose-Modell zur Erfassung von Dosis-Schwellenwert, 2005	34,-
188	Verwertung von Kunststoffbauteilen aus Altautos -Analyse der Umwelteffekte nach dem LCA-Prinzip und ökonomische Analyse, 2005	45,-
189	Darstellung des Schwingungsverhaltens von Fahrzeug-Insassen - Symbiose aus Experiment und Simulation, 2005	45,-
190	Elektromagnetische Feldverteilung und Einkopplungen bei Mobilfunkbetrieb im Kraftfahrzeug, 2005	45,-
191	Leichtbau mit Hilfe von zyklischen Werkstoffkennwerten für Strukturen aus umgeformten höherfesten Feinblech, 2005	30,-
192	Grundsatzuntersuchung zum quantitativen Einfluss von Reifenbauformen und -ausführung auf die auf die Fahrstabilität von Kfz bei extremen Fahrmanövern, 2005	45,-
193	Bewertung der Wirkung von Nutzfahrzeugkollektiven auf Spurrinnen und Ermüdung, 2005	28,-
194	Ermittlung der aktuellen Konzentration und Verteilung von Platingruppenelementen (PGE), 2005	30,-
195	Neue Wege des Effektmonitorings für partikelgebundene Schadstoffe in Dieselabgasen, Hemoglobin adducts of dinitropyrenes as a marker for Diesel emission exposure in humans (Nutzung von Dinitropyren-Hämoglobinaddukten als Biomarker für Dieselabgasexposition), 2005	39,-
196	Bewertung lokaler Berechnungskonzepte zur Ermüdungsfestigkeit von Punktschweißverbindungen , 2005	30,-
197	Berechnungsmethoden für die Lebensdauerabschätzung von MSG- bzw. lasergeschweißten Kehlnähten dünnwandiger Stahlblechstrukturen, 2005	30,-
198	Verbesserung der Prognosefähigkeit der Crashsimulation aus höherfesten Mehrphasenstählen durch Berücksichtigung von Ergebnissen vorangestellter Umformsimulation, 2005	30,-
199	Anwendungspotentiale und Prozeßgrenzen der Klebtechnik für die Umformung von Doppel-lagenblechen, 2006	76,-
200	Thermisches Fügen für die stahlintensive Hybridbauweise im Fahrzeugleichtbau, 2006	76,-
201	Lufthygienische Beurteilung von Pkw-Innenraumfiltern, 2006	96,-
202	Vergleich verschiedener Konzepte der Bodensimulation und von drehenden Rädern zur Nachbildung der Straßenfahrt im Windkanal und deren Auswirkung auf Fahrzeuge, 2006	40,-
203	Methodischer Ansatz im Stahlleichtbau am Beispiel Federbein/Dämpfer, 2006	50,-
204	Innovative Nfz-Konzepte - Gesamtwirtschaftliche Effekte durch Einführung schwerer und langer Lkw, 2006	45,-
205	Technische Kompatibilität von innovativen Nutzfahrzeugkonzepten auf den kombinierten Verkehr Straße/Schiene sowie den Containerverkehr, 2006	50,-
206	Größenaufgelöste physikalische und chemische Bestimmung von elementarem und Organischem Kohlenstoff in Nanopartikeln, 2006	55,-
207	Erstellung einer VHDL-AMS-Modellbibliothek für die Simulation von Kfz-Systemen, 2006	50,-

208	Fahrer-Fahrzeug-Wechselwirkungen bei Fahrmanövern mit Querdynamikbeanspruchungen und zusätzlichen Vertikaldynamikstörungen, 2006	50,-
209	Innovative Nfz-Konzepte - Akzeptanzuntersuchungen zur Einführung und zum Einsatz Innovativer Nutzfahrzeuge, 2007	40,-
210	Das Konzept des Situationsbewusstseins und seine Implikationen für die Fahrsicherheit, 2007	50,-
211	FAT-Richtlinie Dynamische Werkstoffkennwerte für die Crashsimulation, 2007	40,-
212	Innovative Nfz-Konzepte - Wirtschaftlichkeitsanalyse EuroCombi, 2007	35,-
213	Störfestigkeit von Fahrzeugelektronik bezüglich ESD und Impulseinkopplung, 2007	40,-
214	Betriebsfeste Bemessung von mehrachsig belasteten Laserstrahlschweißverbindungen aus Stahlfeinblechen des Karosseriebaus, 2007	40,-
215	Örtlich ertragbare Beanspruchungen bei Spannungskonzentrationen in Karosseriebauteilen aus hoch- und höherfesten Stählen, 2008	35,-
216	Auswirkung der Berücksichtigung lokaler Größen des E-Moduls im Hinblick auf die verbesserte Auslegung umgeformter Karosserieblechstrukturen aus Stahl, 2008	35,-
217	Aktueller Stand und Trends in der CFK-Berechnung im Fahrzeugbau, 2008	49,-
218	In-vitro-Untersuchungen zur Bioverfügbarkeit von an Dieselpartikel gebundenen polyaromatischen Kohlenwasserstoffen und Nitropyrenen, 2008	35,-
219	Bewertender Vergleich der aktuellen Empfehlungen zu den Luftqualitätsgrenzwerten NO ₂ + Evidenz-basierter Vergleich der epidemiologischen Studien 2002-2006 zu Gesundheitseffekten durch NO ₂ , 2008	40,-
220	Fahrdynamische Analyse innovativer Nfz-Konzepte (EuroCombi), 2008	49,-
221	Entwicklung einer Methode zur vergleichenden Bewertung von Schwingfestigkeitsversuchen mit gefügten Stahlblechen in Abhängigkeit des Versagensverhaltens, 2009	40,-
222	Untersuchung zur Wahrnehmung von Lenkmomenten bei Pkw, 2009	35,-
223	Entwicklung einer Prüfspezifikation zur Charakterisierung von Luftfedern, 2009	35,-
224	Klimatische Daten und Pkw-Nutzung - Klimadaten und Nutzungsverhalten zu Auslegung, Versuch und Simulation an Kraftfahrzeug-Kälte-/Heizanlagen in Europa, USA, China und Indien, 2009	35,-
225	CO ₂ -Einsparung durch Verflüssigung des Verkehrsablaufs - Abschätzung staubedingter CO ₂ -Emissionen und von Reduktionspotentialen durch Verbesserung des Verkehrsablaufs, 2009	35,-
226	Modellbasierte Systementwicklung, 2009	50,-
227	Schwingfestigkeitsbewertung von Nahtenden MSG-geschweißter Dünnbleche aus Stahl, 2010	40,-
228	Systemmodellierung für Komponenten von Hybridfahrzeugen unter Berücksichtigung von Funktions- und EMV-Gesichtspunkten, 2010	35,-
229	Methodische und technische Aspekte einer Naturalistic Driving Study, 2010	40,-
230	Analyse der sekundären Gewichtseinsparung, 2010	40,-
231	Zuverlässigkeit von automotive embedded Systems, 2010	kostenloser Download



VDA | Verband der
Automobilindustrie

FAT | Forschungsvereinigung
Automobiltechnik

Behrenstraße 35
10117 Berlin
www.vda.de
www.vda-fat.de