Das FORMUS³IC Forschungsvorhaben im Verbund

Vom Stand der Technik zu neuen Entwicklungen

Lukas Osinski, Jürgen Mottok
Laboratory for Safe and Secure Systems (LaS³),
Zentrum Digitalisierung.Bayern (ZD.B)

Im Forschungsvorhaben Multi-Core Safe and Software-intensive Systems Improvement Community wurden durch einen ganzheitlichen Ansatz die durch heterogene Multi-/Many-Core Architekturen entstehenden Herausforderungen für Automotive und Avionics gelöst. Das zu entwickelnde ganzheitliche Lösungskonzept spiegelt sich in der Berücksichtigung der verschiedenen Ebenen des Hardware-Software Co-Designs wider. Damit wurden neben Lösungen für aktuelle Probleme in erster Linie Beiträge für die effiziente Nutzung heterogener Multi- und Many-Core-Systeme geleistet.

1 Einführung

Der Forschungsverbund Multi-Core Safe and Software-intensive Systems Improvement Community (FORMUS³IC) aus dem Bereich der Informationstechnologie leistete in den Jahren 2015-2018 einen wichtigen Beitrag um sichere heterogene parallele Hardwareplattformen zu nutzen.

Am Forschungsverbund FORMUS³IC sind sechs Hochschulen beteiligt (Ostbayerische Technische Hochschule Regensburg, Friedrich-Alexander-Universität Erlangen-Nürnberg, Hochschule für Angewandte Wissenschaften München, Ostbayerische Technische Hochschule Amberg-Weiden, Technische Hochschule Ingolstadt, Technische Hochschule Nürnberg Georg Simon Ohm) und acht Unternehmen (Airbus Defence & Space GmbH, AUDI AG, Continental Automotive GmbH, Elektrobit Automotive GmbH, Infineon Technologies AG, iNTENCE automotive electronics GmbH, Timing-Architects Embedded Systems GmbH, XKrug GmbH) belegen die relevanz entlang der Wertschöpfungskette. Das Vorhaben wird von der Bayerischen Forschungsstiftung gefördert und hat ein Gesamtvolumen von ca. 4 Mio.€.

Mit sechs technischen Arbeitspaketen (TP2, …, TP6) werden unterschiedliche Fragestellungen aufgegriffen, Lösungen entwickelt und in einem Demonstrator die Machbarkeit (TP7) nachgewiesen. Die im ersten Projektjahr identifizierten und verfolgten Forschungsfragen wurden im zweiten Projektjahr vertieft und die Entwicklung von Lösungen wurde im dritten Projektjahr verfolgt. Folgende Leistungen der einzelnen Teilprojekte von FORMUS³IC sind kurz aufgezählt:

2. Architekturbeschreibung und Time Simulation: Architekturbeschreibung in ADLs erweitert und Gang Scheduling spezifiziert
3. Funktionale Sicherheit und Verifikation: Performantes Fehlertoleranz-Konzept ist erstellt, Untersuchungen zu leichtgewichtigen kryptographischen Primitiven angefertigt
4. Model-Verfeinerung / Hardware-nahe Simulation und Rekonstruktion: Prozessormodelle für HW-feine Analyse erstellt
5. Parallelisierungs-Techniken und –Pattern: Katalog an Parallelisierungs-Pattern erstellt und Auswirkungen auf Scheduling beschrieben

6. Kommunikation: Kommunikationsprotokoll für das Redundanznetzwerk ermittelt und Prototyp implementiert
7. Referenzarchitektur (Demonstrator): Drei Hardwareplattformen und deren HW-/SW-Architektur festgelegt

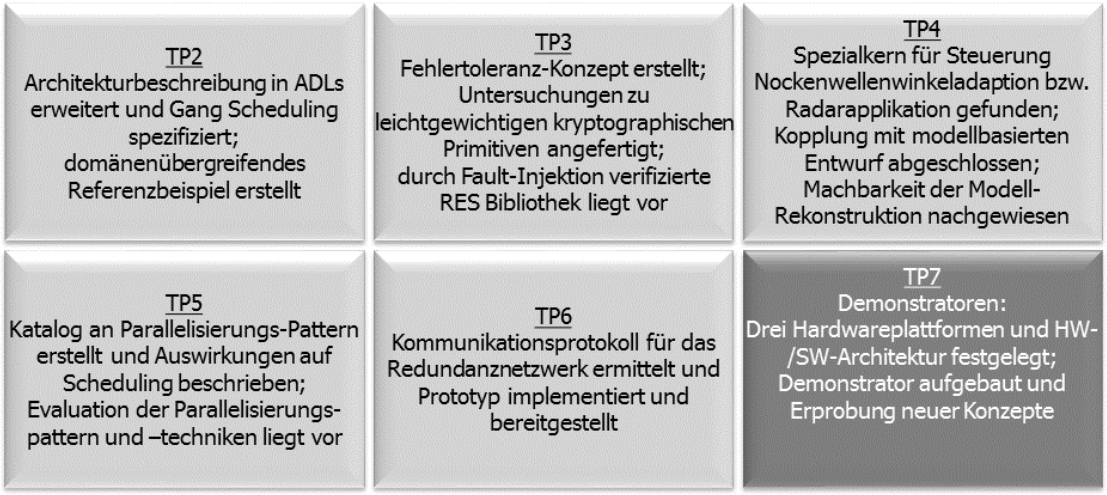

Abbildung 1: Technische Arbeitspakete des FORMUS³IC-Forschungsverbundes

Nach einem Diskurs der Ausganglage im Jahr 2015 bei Projektstart in Kapitel 2 folgt in Kapitel 3 die exemplarische Diskussion des Projektergebnisses der Funktionalen Sicherheit aus Teilprojekt 3 (TP3). Kapitel 4 wird im Anschluss eine Zusammenfassung und einen Ausblick geben.

2 Stand der Technik

Die Diskussion des Stands der Technik erfolgt einzeln für die Themenfelder.

2-1 Architekturbeschreibungssprache EAST ADL

Die EAST-ADL, kurz für Electronics Architecture and Software Technology – Architecture Description Language [1], ist eine domänenspezifische Architekturbeschreibungssprache für die modellbasierte Entwicklung und Beschreibung eingebetteter E/E-System- und Softwarearchitekturen in einem standardisierten Format mit besonderem Fokus auf dem Bedarfsfeld der Domäne Automotive [2]. Die Sprache wurde in enger Anlehnung an den etablierten Automotive Standard entworfen und nutzt dessen sprachliche Mittel zur Umsetzung einer der vier EAST-ADL Abstraktionsebenen. In diesem Sinne kann die EAST-ADL auch als Erweiterung der implementierungsnahen AUTOSAR Sicht angesehen werden, welche diese um zusätzliche, höhere Abstraktionsebenen ergänzt. Die EAST-ADL wurde ursprünglich im Rahmen des europäischer Forschungsprojektes ITEA EAST-EEA [3] entwickelt und anschließend in verschiedenen europäischen Forschungsprojekten weiterentwickelt, zuletzt insbesondere im Kontext der FP7 Projekte ATESST und ATESST2 [4] sowie MAENAD [5], welches insbesondere auf die Anforderungen moderner Elektrofahrzeuge abzielte.

2-2 Scheduling

Im Zentrum eines jeden Betriebssystems mit Multitasking-Unterstützung steht das implementierte Schedulingverfahren. Dieses legt fest welcher Programmteil zu welchem Zeitpunkt welche Priorität hat. Anhand der Priorisierung werden die Programmteile dann im Systemverlauf ausgeführt. Abhängig von der Zielsetzung des implementierten Schedulingverfahrens haben sich hier verschiedene Strategien etabliert. Das erste Prioritätsprotokoll für Multi-Prozessor-Umgebungen wurde von Rajkumar et al. in [6] präsentiert. Moderne Synchronisationsprotokolle erlauben geschachtelte Resourcenan-fragen. Dies wurde erstmals im "Flexible Multiprocessor Locking Protocol" (FMLP) von Block et al. implementiert [7]. Dieses ist sowohl für statisches, als auch für dynamisches Scheduling geeignet, erzeugt allerdings immer noch viele Prioritätsinversionen. Sowohl die Synchronisationsprotokolle von Brandenburg und Anderson [8], als auch das von Ward und Anderson [9] erlauben die geschachtelte Verwendung von Semaphoren, dies allerdings nur für statische, bzw. während der Abarbeitung einer Taskinstanz fester Prioritätsvergabe. Beide Protokolle basieren auf passivem Warten. Das von Brandenburg et al. vorgeschlagene FMLP wurde durch Alfranseder et al. In [10] erweitert. Dabei werden die kritischen Abschnitte des Synchronisationsprotokolls in einen ununterbrechbaren und in einen unterbrechbaren Teil unterteilt. Dadurch verringert sich die Zahl der Prioritätsinversionen für bestimmte Tasksets. Basierend auf den Short-Resource-Requests des FMLP Protokolls erweitern Alfranseder et al. in [11] das Prioritätsprotokoll des OSEK-Betriebssystems und ermöglichen damit den Einsatz von OSEK auf Mehrkernplattformen. In [12] präsentieren Wieder und Brandenburg eine umfassende Übersicht und Analyse verschiedener Arten von, auf aktivem Warten basierenden, Synchronisationsprotokollen. Der in [13] vorgestellte Ansatz baut auf dem Stack Resource Protocol auf und verwendet Software Transactional Memory-Mechanismen um Deadlocks und Prioritätsinversionen zu vermeiden.

2-3 Informationssicherheit

Im Kalenderjahr 2015 zeigte sich jedoch deutlich, dass das Internet der Dinge, welches die Hersteller für sich erschließen möchten, vielfältige Herausforderungen mit sich bringt. Die größte Aufmerksamkeit erzielten Charlie Miller und Chris Valasek mit ihrem Hack des Jeep Cherokee. Sie demonstrierten, wie sie über das Internet die Kontrolle über ein Fahrzeug dieses Typs erlangen konnten und anschließend praktisch sämtliche Funktionen, die der Fahrer im Auto selbst auslösen kann, beispielsweise die Scheibenwaschanlage, die Lüftung oder die Stereoanlage, über das Internet fernsteuerten. Es war ihnen sogar möglich, das Fahrzeug anzuhalten (vgl. [14]). Aber auch andere Hersteller, beispielsweise Nissan oder BMW, offenbarten und offenbaren, dass ihre Fahrzeuge aktuell nicht ausreichend gegen Hackerangriffe geschützt sind. Die Angriffe reichten vom unautorisierten Zugriff auf vertrauliche Fahrzeugdaten (bei Nissan, vgl. [15]) bis zum unautorisierten Öffnen des Fahrzeugs über das Internet (bei BMW, vgl. [16]). Die Folge derartiger, insbesondere medienwirksam inszenierter Hacks sind in der Regel teure Rückrufaktionen oder die Außerbetriebnahme von Konnektivitätsdiensten - ein Nebeneffekt ist außerdem ein Imageverlust für den jeweiligen Hersteller. Dass weder die genannten Angriffe noch die zuvor genannten Automobilhersteller Einzelfälle sind, zeigt ein Bericht von US-Senator Edward J. Markey (Massachusetts) vom Februar 2015: Die "Connected Cars" betreffend fehlen offensichtlich geeignete Sicherheitsfunktionen, um den Fahrer gegen Hackerangriffe zu schützen, die entweder die Kontrolle über das Fahrzeug übernehmen oder fahrerspezifische und damit personenbezogenen Daten sammeln, um aus diesen Kapital zu schlagen (vgl. [17]).

2-4 Funktionale Sicherheit

Sicherheitsnormen (z.B. Automotive: ISO 26262, Avionik: DO-178B) schlagen bereits verschiedene Diagnosetechniken und Überwachungstechniken zur funktionalen Absicherung von eingebetteten Systemen vor. Die Herausforderung hierbei ist, für die unterschiedlichen Anwendungsfälle bzw. Fehlerbilder eines Automotive-Systems geeignete Verfahren und Methoden der Fehlererkennung sowie Fehlerbehandlung zu identifizieren, bei Bedarf neu zu entwickeln und anzuwenden. Die Einführung von Redundanz durch unterschiedliche Software-Kanäle oder eines Kanals von speziell kodierter Software (spezifische Kodierung von Daten und Anweisungen) wird als Safely-Embedded-Software (SES) [18] bezeichnet und ist inspiriert von dem Vital Coded Processor Ansatz [19]. Der einfachste Ansatz einen redundanten Kanal zu erhalten ist die Verdoppelung der ursprünglichen Befehle und Daten. Durch die entstehende einfache Redundanz können jedoch Common Cause Failures nicht erkannt werden, da sie in beiden Kanälen auftreten. Raab et al. [18] haben gezeigt, dass die Verwendung von SES zu einer Erhöhung der Laufzeit um das 44-fache – im Gegensatz zur originalen Version mit Standartmethoden – führt. Die Autoren merken an, dass ihre Methode ein Proof-of-Concept ist und noch Potentiale zur Verbesserung der Laufzeit bestehen. In der Arbeit von Braun [20] wurde die kodierte Verarbeitung einem parallel ausgelegten System gegenübergestellt. Hierbei konnte ein besseres Langzeitverhalten des Coded Processing Ansatzes aufgezeigt werden. Durch die reduzierte Fehlerwahrscheinlichkeit mit geeignet gewähltem Coded Processing kann es ermöglicht werden COTS (components-off-the-shelf) zu verwenden, um Kosten zu sparen, wobei die Zuverlässigkeit des Systems unverändert bleibt.

Braun schlägt zudem ein Software Rejuvenation Model vor, in welchem SES durch das Partial Rejuvenation ergänzt wird [21]. Durch die Anwendung von Markov Modellen könnte eine Verbesserung der Mean-Time-To-Failure (MTTF) um mehr als den Faktor 1000 gezeigt werden. Ein wichtiger Vorteil von SES ist, dass es in der problemorientierten Programmiersprache C realisiert werden kann und dass außerdem bestimmte Fehler erkannt werden, die durch den Compiler entstehen können [22]. Mit dem Thema der kodierten Verarbeitung beschäftigt sich neben Braun et al. auch eine Forschergruppe um die TU Dresden Ausgründung SIListra. Die Forschergruppe entwickelt einen Compiler, der automatisiert C basierte Anwendungen nach den Prinzipien des Coded Processing transformiert und Hardware Fehler aufdecken kann [23]. Zu Verifikation von z.B. Sicherheitsmaßnamen schlägt die automobile Sicherheitsnorm ISO 26262 die Methode der Fault-Injection vor [19]. Dies kann auch auf Diagnose und Überwachungstechniken wie dem SES-Ansatz angewendet werden, bspw. durch eine auf dem Monte-Carlo-Prinzip beruhende modellbasierte Simulationsumgebung.

2-5 Multi- und Many-Core Simulationsumgebungen für Universalprozessoren

In diesem Themenfeld liegt der Fokus in der Auffindung von Entwurfswerkzeugen und geeigneten eigebetteten Architekturen, welche ein hohes Maß an Heterogenität aufweisen und speziell an die Bedürfnisse der Automobil-Industrie angepasst sind. Die Simulation solcher heterogenen Architekturen ist hierfür ein wichtiges Instrument, da oft noch kein synthetisiertes Hardware-Design existiert. Dieses Kapitel gibt einen Überblick über vorhandene Multi- und Many-Core Simulatoren und bewertet deren Eignung für das Vorhaben. Drei Kriterien sind für die Bewertung von Bedeutung. Diese Kriterien sind (i) die *Simulationsperformanz*, (ii) die *Verfügbarkeit von Prozessor-Modellen* für eingebettete und Low-Power-Prozessoren sowie (iii) die Möglichkeit, *Energiebedarf und Rechenleistung* zu *evaluieren*. Tabelle 1 zeigt einen

zusammenfassenden Vergleich der im Folgenden bewerteten Simulationsumgebungen.

Tabelle 1: Vergleich der Simulationsumgebungen

Simulations-umgebung	Multi- und Many-Core Simulationen sind möglich	Simulations-performanz	Verfügbarkeit von Modellen für Energie Messungen	Verfügbarkeit von eingebetteten Prozessor-Modellen
Graphite	Ja	Mittel bis hoch	Ja	Keine
Sniper	Ja	Hoch	Ja	Keine
SoCLib	Ja	Langsam bis mittel	Ja	Mittel
HORNET	Ja	Langsam bis mittel	Ja	Ein MIPS Modell
gem5	Ja	Langsam bis hoch	Ja	Hoch
QEMU	Ja	Sehr hoch	Nein	Niedrig
OVP	Ja	Sehr hoch	Ja	Sehr hoch

2-6 Parallelisierungstechniken und –pattern in C++

Trotz einer langfristig sinkenden Verbreitung ist C++ aktuell nach C und Java die Programiersprache mit der dritthäufigsten Verwendung (gemäß dem Tiobe Index [24]). Seit C++11 bietet die Sprache eine Reihe von Elementen zur Entwicklung von parallelem Code an, u.a. Threads, Speichermodelle, asynchrone Ausführung (async) und Thread-lokaler Speicher. In der kommenden Version C++17 sind neben Verfeinerungen auch parallele Algorithmen geplant. Bis 2020 sollen alle Formen von Intra-Node Parallelität mit nativen C++ abgebildet werden können, wie SIMD, Multicore CPU, GPU. OpenMP soll dann vollständig in C++ aufgehen, so dass keine OpenMP Anweisungen mehr nötig sind [25]. In C++11 wurde erstmalig die nebenläufige Programmierung im C++ Standard ermöglicht. Die wesentlichen Konstrukte dazu werden im Folgenden kurz dargestellt [26]. Der Standard C++14 unterscheidet sich von C++11 hauptsächlich durch Fehlerbereinigungen und kleineren Verbesserungen.

Mit der std::thread Bibliothek wird eine Schnittstelle für Threads bereitgestellt. Wie üblich können Threads erzeugt und verwaltet werden. Die Threads werden dabei durch die Laufzeitbibliothek auf das verwendete Betriebssystem abgebildet. Ebenso werden Synchronisierungs-mechanismen bereitgestellt, wie z.B. join().

Zur gemeinsamen Nutzung von Ressourcen werden entsprechende Objekte bereitgestellt, wie mutex, sowie entsprechende Funktionen, wie lock().

Eine Vereinfachung der parallelen Ausführung in C++11 wird durch die Verwendung von promises / futures erreicht, mit denen die Ergebnisse von nebenläufigen Ausführungen wieder zusammengeführt werden können.

Eine weitere Vereinfachung wird durch die Verwendung von async erreicht. Damit können asynchrone Funktionsaufrufe gestartet werden, welche nebenläufig ausgeführt werden. Die wesentlichen Parameter sowie die geteilten Daten werden dem Aufruf mitgegeben. Das Ergebnis wird dann zu einem späteren Zeitpunkt als future abgerufen. Die Laufzeitumgebung kann dadurch den Zeitpunkt sowie den Ort der Funktionsausführung frei bestimmen.

Der Programmierer kann durch die Verwendung von futures und async Berechnungen starten, welche dann von der C++ Laufzeitumgebung parallel ausgeführt werden können. Erst wenn das Ergebnis benötigt wird, werden die Ausführungsstränge wieder zusammengeführt. C++ wurde um Mechanismen erweitert, die die Erstellung von nicht-blockierenden Datenstrukturen ermöglichen. Dazu sind z.B. im atomic Package der Standardbibliothek verschiedene atomare Operationen enthalten, wie z.B. std::atomic::fetch_add

Die C++ Version C++17 ist als ein „major release“ mit relevanten Neuerungen umgesetzt: Ein wichtiger Aspekt wird die Bereitstellung von parallelen Versionen einiger Algorithmen der Standardbibliothek sein, wie z.B. sort oder for_each. Der Grad der Parallelisierung kann durch einen Parameter eingestellt werden. Zukünftige Erweiterungen umfassen SIMD basierte Parallelisierung sowie eine Beeinflussung der Vector-Order-of-Evaluation.

2-7 Redundanzkonzepte der Avionik

Im zivilen Sektor zählten die Architektur der Airbus-Serien A320, A330 und A340 [27], sowie bei Boeing die der 777 [29] lange Zeit als Quasi-Standard. Verfolgt wurde hierbei der Ansatz der Federated Architecture. Typisch dafür ist, dass das Gesamtsystem in Einzelsysteme – oft auch Line Replacable Units (LRU)s genannt – unterteilt ist [28]. Jede dieser LRU ist dabei für einen bestimmten Aufgabenbereich zuständig und besteht aus einem eigenen Prozessor, sowie eigenem Speicher. Miteinander verbunden sind diese über spezielle, auf hochsicherheitskritische Anwendungen ausgelegte Datenbusse wie dem MIL-STD-1553B oder dem ARINC 429. Diese Busse sind jedoch spezifisch für die Avionik-Branche und werden nicht von Mikrocontrollern selbst bereitgestellt, sondern müssen über separate Komponenten realisiert werden – was sich wiederum negativ auf Größe, Gewicht und Stromverbrauch auswirkt. Dies können entweder eigenständige I/O-Boards sein, welche in den Rechner gesteckt werden, oder aber auch FPGAs die sich auf dem Rechnerboard selbst befinden.

3 Funktionale Sicherheit (aus TP3) als exemplarisches Projektergebnis

Als Ausgangspunkt für die Weiterentwicklung des Überwachungskonzepts wurde der softwarebasierte Combined Redundancy (CoRed) [30] Ansatz für Singlecore Systeme verwendet. CoRed vereint die redundante Ausführung von Anwendungen (Prozess) in Form von Triple-Modular-Redundancy (TMR) mit arithmetischer Codierung, um den Zuverlässigkeitsengpass Mehrheitsentscheider zu beheben.

TMR ist ein weit verbreitetes Muster um Fehlertoleranz insbesondere im Hardware-Bereich zu realisieren. Dabei wird durch drei identische Elemente dieselbe Operation ausgeführt und anschließend das korrekte Ergebnis durch einen Mehrheitsentscheid ermittelt. Dieser Ansatz bietet neben der Möglichkeit Fehler zu detektieren, den Vorteil, dass auch das fehlerhafte Element identifiziert werden kann. Zudem wird ein Beitrag zur Verfügbarkeit des Systems geleistet, da durch Maskierung (Mehrheitsentscheid) die Systemausführung trotz eines fehlerhaften Ergebnisses fortgeführt werden kann. Eine kritische Fehlerstelle von TMR stellt jedoch der Mehrheitsentscheid selbst dar. Aus diesem Grund müssen an diese Komponente hohe Anforderungen in Bezug auf die Zuverlässigkeit gestellt werden. Diese Zuverlässigkeitsanforderungen können bei einer Realisierung in Hardware durch bspw. gehärtete Schaltungen erreicht werden. Bei einer rein softwarebasierten Lösung – wie im Falle von

CoRed – kann die arithmetische Codierung zur Erhöhung der Zuverlässigkeit angewendet werden.

Beim CoRed Ansatz werden Anwendungsprozesse repliziert und sequentiell, gefolgt von einem codierten Mehrheitsentscheid, auf einem einzelnen Kern ausgeführt. Beim Eintritt in den Prozess werden die zuvor arithmetisch codierten Daten decodiert, um die durch codierte Operationen hervorgerufenen Einbußen bei der Ausführungszeit auf ein vernachlässigbares Minimum zu reduzieren. Nach Fertigstellung der Tasks werden die Daten wiederum arithmetisch codiert und dem nachfolgenden codierten Mehrheitsentscheid (Prozess) zur Verfügung gestellt. Während der Zeitspanne in den Daten innerhalb der Anwendungstasks uncodiert verarbeitet werden, wird die Erkennung von Fehlern durch die redundante Ausführung der Anwendungsprozesse sichergestellt. Der Mehrheitsentscheid selbst wird auf arithmetisch codierten Daten ausgeführt, um eine Erkennung von Fehlern innerhalb des Entscheidungsprozesses zu ermöglichen und durch Rückwärtsbehebung zu korrigieren. Die Wirksamkeit des CoRed Ansatzes – insbesondere des codierten Mehrheitsentscheids – wurde durch simulative Fault Injection Experimente gezeigt. Nachteile ergeben sich jedoch durch die Notwendigkeit von partieller Virtualisierung, Run-To-Completion Semantik, Schwierigkeiten bei der Einhaltung von Echtzeitgarantien aufgrund von Rückwärtsbehebung und sequentieller Prozessausführung, sowie der fehlenden Abdeckung von permanenten Fehlern aufgrund des Single-Core Ansatzes. Zudem besteht durch die Verwendung der AN-BD Codierung und der damit verbundenen komplexen Verwaltung der B und D-Signaturen eine erhöhte Fehleranfälligkeit in der Verwaltungslogik außerhalb des Mehrheitsprozesses.

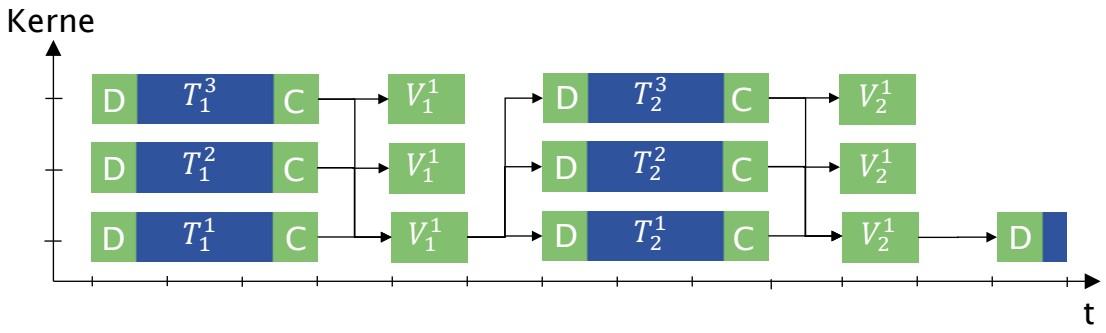

Abbildung 2 - Verfahren zur Online-Überwachung [31]

3-1 Fehlertoleranzarchitektur
Die Weiterentwicklung des ursprünglichen Ansatzes umfasste u.a. folgende Punkte:

- Parallele Ausführung der Replikate (Prozess und Mehrheitsentscheid) auf separaten Kernen zur Reduzierung der Antwortzeit
- Einführung von multiplen Mehrheitsentscheidern mit singulärer Ausgabe (Hierarchie) zur Elimination der Rückwärtsbehebung und Reduzierung der Antwortzeit
- Optimierung der arithmetischen Codierung durch
    - Eliminierung der notwendigen Redundanz bei der statischen B Signatur
    - Eliminierung der D-Signatur und der damit verbundenen komplexen und fehleranfälligen Verwaltungslogik
- Integration von Gleitkommazahlen
- Einführung von Rekonfigurationsstrategien auf Applikationsebene zur Erhöhung der Verfügbarkeit

Der Einsatz von arithmetischer Codierung auf Quellcodeebene führt zur Erhöhung der Ausführungszeiten, da alle Operationen in der codierten Domäne ausgeführt und komplexe Signaturen zur Laufzeit berechnet werden müssen. Im Kontrast dazu wird im aktuellen Überwachungskonzept die arithmetische Codierung lediglich zur Absicherung der kritischen Fehlerstellen wie z.B. dem Mehrheitsentscheid oder Speicherablage eingesetzt. Durch die Decodierung der Daten zu Beginn der Prozessausführung können die Kosten für den Einsatz der arithmetischen Codierung auf ein vernachlässigbares Maß reduziert und die Antwortzeit erheblich verbessert werden. Zudem erlaubt der selektive Einsatz der Codierung die Integration von Gleitkommazahlen innerhalb der Prozesse. Durch die Elimination von Redundanzen der statischen B-Signatur und der Verwaltungslogik der D-Signatur besteht nun zur Umsetzung des Konzepts im Industrieumfeld zudem keine Notwendigkeit der (zertifizierten) Toolunterstützung.

Eine Vielzahl von Konzepten berücksichtigt lediglich transiente Fehler. Obwohl die Fehlerrate für permanente Fehler deutlich geringer, als für transiente Fehler ist, beziehen wir diese in unsere Betrachtung mit ein. Durch die Ausführung der replizierten Prozesse auf unterschiedlichen Kernen, kann auf homogenen Systemen bereits ein kleiner Beitrag hin zur Erkennung von permanenten Fehlern geleistet werden, welcher sich jedoch erst durch den Einsatz von heterogenen Systemen verstärkt. Zudem wird durch die parallele Ausführung der Prozessreplikate die Antwortzeit des Systems verbessert und somit die Echtzeitfähigkeit gefördert. Durch die Replikation der Mehrheitsentscheider und die Ausführung auf unterschiedlichen Kernen wird die vorher notwendige Rückwärtsbehebung eliminiert. Sollte es zum Ausfall eines Mehrheitsentscheids kommen, so übernimmt einer der korrekten Entscheider die Ausgabe des korrekten Ergebnisses. Um eine singuläre Ausgabe der Mehrheitsentscheide sicherzustellen wird die Ausgabereihenfolge durch eine Hierarchie geregelt. Nach dem Ausfall eines Mehrheitsentscheids werden je nach Systemdesign (statisch/dynamisch) unterschiedliche Rekonfigurationsstrategien ausgeführt, welche sicherstellen, dass defekte Komponenten ausgegliedert und die Replikate auf gesunde Komponenten verlagert werden.

Zur Verifikation des Überwachungskonzepts wurden Fault Injection Experimente mit Daten- und Kontrollflussfehlern auf einem Infineon AURIX TriCore TC27x unter ERIKAOS2.7 durchgeführt. Zum Einsatz kamen dabei ein uncodierter und ein optimierter arithmetisch codierter Mehrheitsentscheid. Die Experimente wurden mit unterschiedlichen Eingabewerten durchgeführt um eine vollständige Zweigüberdeckung (engl. Branch Coverage) zu erreichen. Datenfehler (DF) wurden durch Injektion von transienten 1-Bitfehlern während dem Lesezugriff der genutzten Register abgebildet. Kontrollflussfehler (KFF) wurden durch die Manipulation des Programmzählers injiziert. Die Ergebnisse der Experimente wurden dabei in folgende Kategorien eingeordnet:

- Maskiert: Fehler hatte keinen Einfluss auf die Programmausführung
- Erkannt:
  - Codierung: Fehler wurde durch die Codierung erkannt
  - Trap: Fehler löste eine Hardwareausnahme aus
  - OS: Fehler wurde durch das Betriebssystem erkannt
  - Kein Ergebnis (KE): Fehler wurde erkannt, jedoch konnte kein eindeutiges Ergebnis ermittelt werden
- Unerkannter Datenfehler (UD): Fehler wurde nicht erkannt

Wie aus den Ergebnissen in Tabelle 2 zu sehen ist führen die injizierten Fehler beim uncodierten Mehrheitsentscheid zu einer nicht unerheblichen Anzahl von unerkannten Datenfehler. Bei der Injektion von Datenfehler werden 12% der ursprünglich injizierten Fehler nicht erkannt. Annähernd ähnliches Verhalten ergibt sich bei der Injektion von Kontrollflussfehlern. Hier werden ca. 13% der Fehler nicht erkannt. Im Gegensatz dazu treten bei der codierten Variante keine unerkannten Datenfehler auf.

		Uncodiert # Instr.: 17		**Codiert # Instr.: 41**	
		DF	KFF	DF	KFF
Maskiert		1664	187	2432	199
Erkannt	Codierung	-	-	832	970
	Trap	52	0	0	0
	OS	0	0	0	0
	Kein Ergebnis (KE)	292	276	0	0
Unerkannter Datenfehler (UD)		**264**	**73**	**0**	**0**
Summe		2272	563	3264	1169

Tabelle 2 - Ergebnisse der Fault Injection (Daten- und Kontrollflussfehler) - Vergleich uncodierter und codierter Mehrheitsentscheid

Der aktuell untersuchte Prototyp verfügt über umfassende Fehlererkennungsmechanismen, die selbst den Ausfall eines kompletten Kerns erkennen sollten. Es herrschen

jedoch Einschränkungen vor, die einer universell einsetzbaren Softwarelösung zur Erkennung von Fehlzuständen noch nicht vollumfassend gerecht werden.

Bisher ist der Prototyp auf die Verarbeitung von ganzzahligen Werten im Mehrheitsentscheid beschränkt, da sich die Einbindung von Gleitkommazahlen in die aktuelle Absicherung des Mehrheitsentscheiders nicht trivial gestaltet. Zwar werden während der Ausführung der Tasks keine codierten Zahlen eingesetzt, so basiert jedoch die Absicherung des Mehrheitsentscheiders auf kombinatorischen Vergleichen codierter Variablen und Parametern sowie Signaturen. Folglich bedarf es einer Erweiterung zur Einbindung von Gleitkommazahlen in die bestehenden Absicherungsmechanismen.

Neben der Erweiterung um Gleitkommazahlen soll ebenso der Maskierungsprozess an Variabilität gewinnen. Derzeit ist nur eine vergleichsweise starre Mehrheitsentscheidung möglich. Dies wiederum setzt Anforderungen an die Eingabedaten, so müssen diese bei Korrektheit identische Werte aufweisen. Diese Annahme dürfte in der Praxis, insbesondere unter Verwendung von diversitär redundanten Sensoren, schwer zu bewerkstelligen sein, da beispielsweise eine Varianz bei der Sensorcharakteristik anzunehmen ist.

Um diese Variabilität zu berücksichtigen wird der Maskierungsprozess aktuell weiterentwickelt. So werden verschiedene Maskierungsalgorithmen eingesetzt, die nicht nur auf Basis absoluter Werte eine valide Entscheidung treffen und unterschiedliche Toleranzen bezüglich der Eingabedaten berücksichtigen. Die Konfigurationsoptionen sollen die Möglichkeit bieten den Mehrheitsentscheid an die zugehörige Applikation und deren Umgebungsbedingungen anzupassen, um so ein breites Spektrum von theoretischen Einsatzorten abzudecken.

3-2 Fault-Injection Plattform für Multi-Core Systeme

Ein wichtiger Schritt im Entwicklungszyklus von zuverlässigen Systemen ist die Validierung ihrer Eigenschaften in Gegenwart von zufälligen permanenten oder transienten Hardwarefehlern. Die Sicherheitsrichtlinien für Entwurf, Entwicklung, Verifikation und Validierung von sicherheitskritischen Automobilsystemen (z.B. ISO 26262) nennen Fault Injection als geeignetes Mittel zur Validierung der korrekten und effektiven Umsetzung von funktionalen und technischen Sicherheitsmechanismen.

Aktuell öffentlich zugängliche Fault Injection Plattformen konnten unter anderem aufgrund folgender Einschränkungen nicht eingesetzt werden:

- Fehlende Anbindung für Infineon TriCore oder ARM Systeme
- Fehlende Mechanismen zur Reduktion des Fehlerraums und somit der Experimentdauer
- Hohe Experimentlaufzeiten aufgrund der Nutzung von JTAG Debuggern mit begrenztem Funktionsumfang und Performanz
- Notwendigkeit der Anpassung von Entwicklungswerkzeugen der Industriepartner aufgrund der Nutzung von z.B. LLVM
- Fehlende Möglichkeit zur Injektion auf realer Hardware (simulative Ansätze)
- Notwendigkeit der Quellcodemanipulation (sog. Fault Seeding)

Aufgrund dieser Einschränkungen wurde der Entschluss gefasst eine eigenständige Plattform für die Injektion von permanenten und transienten Hardwarefehlern auf Multicore Systemen zu entwickeln. Das entwickelte PyFI (Python-based Fault Injection) Backend erlaubt es - durch Nutzung der API des iSystem iC5000 On-Chip Analyzers – zur Laufzeit permanente und transiente Fehler auf Befehlssatzebene (Daten/Instruktionen) zu injizieren und so die gewünschten Fehlersymptome (Daten-/Kontrollflussfehler) auf Anwendungsebene zu erzeugen.

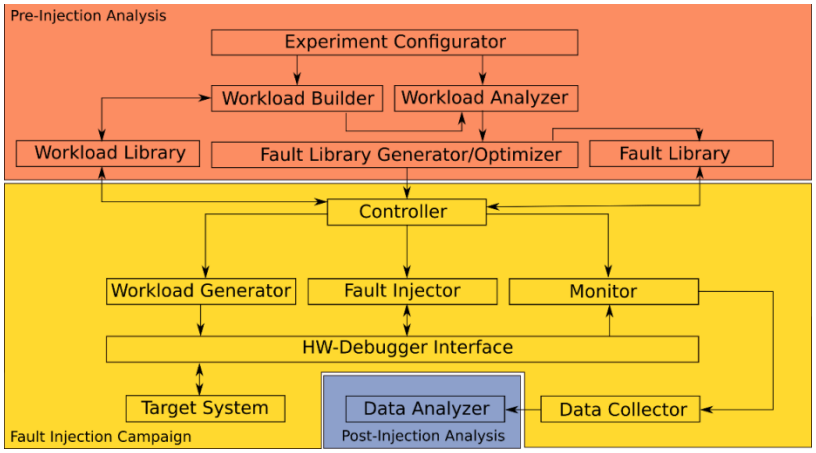

Abbildung 3 - Phasen und Module der Fault Injection [32]

PyFI erweitert die von Hsueh et al. [33] vorgeschlagene Fault Injection Architektur und gliedert die Experimentausführung in drei Phasen: Pre-Injection Analysis, Fault Injection Campaign und Post-Injection Analysis.
Während der Pre-Injection Phase wird die Anwendung sowohl einer statischen (disassembliertes ELF File) als auch dynamischen Analyse (Laufzeitverhalten) unterzogen und in Abhängigkeit der Experimentkonfiguration eine Datenbank mit zu injizierenden Fehlern erstellt. Durch die anschließende Reduktion des Fehlerraums kann bereits vorab die Experimentlaufzeit erheblich verkürzt werden indem z.B. alle Fehlerstellen entfernt werden, bei denen es sich um Register handelt deren Inhalt bei jeder Instruktionsausführung überschrieben werden (Inject-on-Read). Während der Kampagne (Fault Injection Campaign) werden die zuvor generierten Fehlerstellen abgearbeitet und das Systemverhalten (Traps, Timeout) und der Systemzustand durch Nutzung des On-Chip Analyzers aufgezeichnet. In der Post-Injection Analyse werden abschließend die Experimentaufzeichnungen auswertet und die gewünschten Metriken errechnet.

4 Zusammenfassung und Ausblick

Die Wirkung der technischen Teilprojekte (TP2, …, TP6) in der Gesamtperspektive FORMUS³IC lässt sich durch folgende Ergebnisse beschreiben:

Angepasste Austauschformate (TP2) – Beitrag zu Entwicklungsprozess

Angepasste Austauschformate (AUTOSAR, EAST-ADL) für heterogene Multi- und Many-Core-Systeme−wie im Rahmen von FORMUS³IC Teilprojekt 2 angestrebt−ermöglichen effiziente toolbasierte Geschäftsprozesse bei der Entwicklung von Mehrkernarchitekturen für OEMs und Zulieferer; dies führt zu kostengünstigen Systemen. Durch die Unterstützung von Beschreibung von Parallelität entstehen zu-

dem Guidelines für die Praxis, welche eine Effizienzsteigerung in der Entwicklung zur Folge haben und gleichzeitig bessere Wartbarkeit und Wiederverwendbarkeit ermöglichen.

Ergänzungen zu AUTOSAR Standard (TP2) – Beitrag zu Entwicklungsprozess und Architektur

Verschiedene Ergänzungen zum AUTOSAR Standard wurden entwickelt, da AUTOSAR im klassischen AUTOSAR statische Prioritätenvergabe erfolgt. Exemplarisch stellt Gang-Scheduling eine Erweiterung dar.

Gang Scheduling mit effizientem Taskmanagement (TP2) – Beitrag zur Echtzeitarchitektur

Das Schedulingkonzept wurde daher auf Basis einer, für Gangscheduling adaptierten, Implementierung für Global EDF erweitert. Diese Erweiterung ermöglicht eine gute Skalierbarkeit für globales Scheduling. Die Wiederverwendung dieses Konzepts ist möglich.

Erhöhung der Informationssicherheit (TP2) – Beitrag zur Verlässlichkeit

Die RES Bibliothek wurde entwickelt. Eine Schwachstellenanalyse und Härtung erfolgte. Die Wiederverwendung des entwickelten Konzepts der Informationssicherheit ist möglich.

Hoher Automotive Security Integrity Level (ASIL) mit Sicherheitskonzept (TP3) – Beitrag zur Architektur

Fehlerredundanzkonzept für Anwendungen im Bereich Antriebsstrang, Verbrennungsmotor-steuerung, Getriebesteuerung und Domänencontroller für höchste Echtzeitanforderungen an die Applikationssoftware (maximale Reaktionszeit) erstellt. Dabei werden eine effiziente Fehlererkennung sowie Fehlerbehebung realisiert.

Neue Fault-Injection Konzepte für Mikrocontroller (TP3) – Beitrag zur Hardwareprodukt

Beitrag zu neuer Fault Injection Plattform bei Infineon.

Weiterentwicklung des ADAS Frameworks XKLAF (TP4)

Für die Entwicklung von ADAS Software bietet der industrielle Projektpartner XKRUG die XKLAF Rapid-Prototyping Plattform als virtuelles Steuergerät auf Basis von x86 oder ARM für den Fahrzeugeinsatz unter AUTOSAR an, auf dem die Software Komponenten (SWC) für das spätere Seriensteuergerät entwickelt werden.

Die Erkenntnisse des Projekts werden direkt in die Produktentwicklung von XKLAF für künftige universelle Steuergeräte einfließen und ermöglichen den OEMs und TIER1s, die Funktionsentwicklung bereits in Versuchsfahrzeugen einzubauen und in Echtzeit zu evaluieren.

Neue Entwicklungswerkzeuge für Halbleiterhersteller (TP4) – Beitrag zur Toolkette

Die Ergebnisse des Teilprojektes haben auch entscheidenden Einfluss für den Partner Infineon für die zukünftige Bereitstellung von Werkzeugen für die SW-Entwicklung mit dem Mikrokontroller AURIX. So wurde hinsichtlich der Einsatzfähigkeit der Modelle für den AURIX und den Beschleunigerkern SPU unter Platform Architect

eine detaillierte Untersuchung in Form von Stresstests mit Signal-intensiven Algorithmen vorgenommen.

Neue Sensorfusionsalgorithmen (TP4) – Beitrag zu verlässlichen Algorithmen

Neu entwickelte Sensor-Fusions-Algorithmen für Personendetektion aus Kameradaten und Signalvorverarbeitung haben wichtige Erkenntnisse für zukünftige Einsatzgebiete des AURIX in ADAS-Applikationen aufgezeigt. Dies betrifft u.a. die effiziente Umsetzung eines Kalman-Filters und einer speicheroptimierten Variante des DBSCAN-Verfahrens auf einem TriCore-Kern.

Verfahren der Timinganalyse von ARM- und GPU-Architekturen (TP4) – Beitrag zur Laufzeitanalyse

Mit Hilfe eines von der FAU für TA bereit gestellten Beispielcodes zur Radarsignalverarbeitung und zur Verarbeitung optischer Kameradaten wurden belastbare Aussagen über die Laufzeit für ARM- und GPU-Architekturen mit Hilfe von Messungen auf dem Nvidia Drive PX 2 Entwicklungsboard ermittelt.

Supercore-Pattern und Task-Parallelität (TP5) – Beitrag zur Architektur paralleler heteorgener Plattformen

Die gefundenen Architekturkonzepte für Multicore-Prozessoren, GPU und FPGA sind für Audi und Continental von großer Bedeutung und helfen bei der Weiterentwicklung der Software-Architekturen für zukünftige Plattformen.

Redundanzarchitektur mit hoher Zuverlässigkeit für autonome Lufttaxis (TP6) – Beitrag zur Architektur für kleine, autonom agierende Flugzeuge

Bereitstellung einer zuverlässigen Redundanzarchitektur, die ausschließlich mit verfügbarer onboard-Peripherie realisiert wird. Hierbei entfällt die Zusatzhardware in Form von ASICs oder FPGAs, welche üblicherweise für die Implementierung der Redundanzfunktionalität verwendet wird. Dies bringt neben den verringerten Produktkosten für die zu entwickelnde Hardware auch Platz-, Leistungs- und Gewichtsersparnisse.

Quadruplex-Architektur als Referenz (TP6) – Beitrag zu Fail-Operational Architekturen

Die Wiederverwendung einer kostengünstigen Realisierung von Triplex- bzw. Quadruplex-Systemen für Automotive und Transportation Systemen wird möglich.

Website des Forschungsprojektes FORMUS³IC
http://formus3ic.de/

Danksagung
Diese Publikation hat die Bayerische Forschungsstiftung unterstützt, Verbundvorhaben FORMUS3IC ”Multi-Core safe and softwareintensive Systems Improvement Community”, Förderkennzeichen AZ-1165-15.

Referenzen

[1] Electronic Architecture and Software Technology – Architecture Description Language, EAST-ADL Association, http://www.east-adl.info, 2015.

[2] P. Cuenot, P. Frey, R. Johansson, H. Lönn, Yiannis Papadopoulos, M.-O. Reiser, A. Sandberg, D. Servat, R. Tavakoli Kolagari, M. Törngren, M. Weber. The EAST-ADL Architecture Description Language for Automotive Embedded Software, Model-Based Engineering of Embedded Real-Time Systems, Springer Berlin Heidelberg, 2010.

[3] EAST-EAA partners, EAST-EEA – Electronic Architecture and Software Technology – Embedded Electronic Architecture, https://itea3.org/project/east-eea.html, 2015.

[4] ATESST consortium, Advanced Traffic Efficiency and Safety through Software Technology, http://www.atesst.org, 2015.

[5] MAENAD consortium, Model-based Analysis & Engineering of Novel Architectures for Dependable Electric Vehicles, http://www.maenad.eu, 2015.

[6] R. Rajkumar, Real-Time Synchronization Protocols for Shared Memory Multiproces-sors, IEEE, 1990.

[7] A. Block, H. Leontyev, B. B. Brandenburg, J. H. Anderson, A Flexible Real-Time Locking Protocol for Multiprocessors, IEEE, 2007.

[8] B. B. Brandenburg, J. H. Anderson, The OMLP Family of Optimal Multiprocessor Re-al-Time Locking Protocols, Springer, 2013.

[9] B. C. Ward, J. H. Anderson, Supporting Nested Locking in Multiprocessor Real-Time Systems, IEEE, 2012.

[10] M. Alfranseder, M. Deubzer, B. Justus, J. Mottok, C. Siemers, An Efficient Spin-Lock Based Multi-Core Resource Sharing Protocol, IEEE, 2014.

[11] M. Alfranseder, M. Mucha, S. Schmidhuber, A. Sailer, M. Niemetz, J. Mottok, A Modified Synchronization Model for Dead-Lock Free Concurrent Execution of Strongly In-teracting Task Sets in Embedded Systems, IEEE, 2013.

[12] A. Wieder, B. B. Brandenburg, On Spin Locks in AUTOSAR: Blocking Analysis of FIFO, Unordered, and Priority-Ordered Spin Locks, IEEE, 2013.

[13] A. Barros, L. M. Pinho, P. M. Yomsi, Non-preemptive and SRP-based fully-preemptive scheduling of real-time Software Transactional Memory, Elsevier, 2015.

[14] R. Eikenberg. Hacker steuern Jeep Cherokee fern, Heise Security, http://heise.de/-2756331, 2015.

[15] T. Hunt. Controlling vehicle features of Nissan LEAFs across the globe via vulnerable APIs, http://www.troyhunt.com/2016/02/controlling-vehicle-features-of-nissan.html, 2016.

[16] D. Spaar. Auto, öffne dich! Sicherheitslücken bei BMWs ConnectedDrive, Heise c't 05/2015, S. 86, http://heise.de/-2536384, 2015.

[17] E. J. Markey. Tracking & Hacking: Security & Privacy Gaps Put American Drivers at Risk, https://www.markey.senate.gov/imo/media/doc/2015-02-06_MarkeyReport-Tracking_Hacking_CarSecurity%202.pdf, 2015.

[18] P. Raab, S. Kraemer, J. Mottok, H. Meier, S. Racek. Safe software processing by concurrent execution in a real-time operating system. In Proceedings of 16th International Conference on Applied Electronics, pages 315 - 319, September 2011.

[19] ISO26262: Road Vehicles – Functional safety, International Organization for Standardization, 2011

[20] J. Braun, D. Geyer, and J. Mottok. Alternative measure for safety related software. ATZelektronik, 04/2012:40-43, August 2012. ISSN 1862-1791.

[21] J. Braun, J. Mottok, C. Miedl, D. Geyer, and M. Minas. Increasing the reliability of single and multicore systems with software rejuvenation and coded processing. In Proceedings of the Automotive Safety & Security 2012 Sicherheit und Zuverlässigkeit für automobile Informationstechnik in Karlsruhe, ISBN 978-3-88579-604-6, pages 163-178, November 2012.

[22] J. Braun, J. Mottok, C. Miedl, D. Geyer, and M. Minas. Capability of single hardware channel for automotive safety applications according to ISO 26262. In Proceedings of the 17th IEEE International Conference on Applied Electronics 2012 (AE2012) in Pilsen, ISBN 978-80-261-0038-6, pages 41-45, September 2012.

[23] M. Suesskraut, U. Schiffel, A. Schmitt, C. Fetzer. White paper: Encoding Compiler and Encoded Processing. 2011.

[24] „Tiobe,“ [Online]. Available: http://www.tiobe.com/tiobe_index. [Zugriff am 09 05 2016].

[25] Codeplay: Wong, Michael, „Towards Massive Parallelism for C++ and OpenMP (aka Heterogeneous device/Accelerator/GPGPU/FPGA): the future of Parallel Programming Models,“ in ARCS 2016, Nürnberg, 2016.

[26] B. Stroustrup und Langenau, Frank, Eine Tour durch C++, Carl Hanser Verlag, 2015.

[27] BRITXE, Dominique ; TRAVERSE, Pascal: Airbus A320/A330/A340 Electrical Flight Controls - A Family Of Fault-tolerant Systems (1993).

[28] MOIR ; IAN ; SEABRIDGE ; ALLAN ; JUKES ; MALCOLM: CIVIL AVIONICS SYSTEMS. Second Edition, 2013.

[29] Y. C. (BOB) YEH: Triple-Triple Redundant 777 Primary Fight Computer. In: Aerospace Applications Conference, 1996. Proceedings., 1996 IEEE 02/1996 (1996).

[30] U. Schiffel, „Hardware error detection using AN-codes”, 2010.

[31] T. L. R. M. J. M. Lukas Osinski, „Challenges and Opportunities with Embedded Multicore Platforms,“ ERTS, 2018.

[32] T. L. M. S. J. M. Lukas Osinski, „PyFI - Fault Injection Platform for Real Hardware,“ ARCS Workshop 2018, 2018.

[33] M. T. T. I. R. Hsueh, Fault Injection Techniques and Tools, Computer, Vol. 40, 1997.

Autoren

	**Lukas Osinski** M.Sc. (lukas.osinski@oth-regensburg.de) promoviert im Bereich Funktionale Sicherheit für Multi- und Manycore-Systeme im Rahmen des FORMUS³IC-Projektes am Laboratory for Safe and Secure Systems (LaS³) der OTH Regensburg. Er beschäftigt sich dabei insbesondere mit Software-Verfahren für fehlertolerante Systeme.
	**ZD.B[1]-Forschungs-Professor Dr. Jürgen Mottok** (juergen.mottok@oth-regensburg.de), Projektleiter, lehrt Informatik an der OTH Regensburg. Seine Lehrgebiete sind Software Engineering, Programmiersprachen, Echtzeitsysteme, Functional Safety und IT-Security. Er leitet wissenschaftlich das Laboratory for Safe and Secure Systems (LaS³, http://www.las3.de) in Regensburg, ist zweiter stellvertretender Vorsitzender des Bavarian Cluster of IT-Security and Safety, Beirat des Automotive Forum Sicherheit Software Systeme, Beirat des ASQF Safety, Mitglied des Leitungsgremiums der Regionalgruppe Ostbayern der Gesellschaft für Informatik, Organisator des Fachdidaktik-Arbeitskreises Software Engineering der Bayerischen Hochschulen, Vorsitzender des Lehren von Software Engineering e.V. (LeSE e.V.) und Projektleiter der mit kooperativen Promotionsverfahren ausgestatteten Forschungsprojekte VitaS³, PetS³, PeCall, S³OP, S³EMO, AMALTHEA, AMALTHEA4public, ES³M, FORMUS³IC, ZeloS³, FraLa, S³CORE und EVELIN. Prof. Dr. Jürgen Mottok ist in Programmkomitees zahlreicher wissenschaftlicher Konferenzen vertreten. Er ist Träger des **Preises für herausragende Lehre**, der vom Bayerischen Staatsministerium für Wissenschaft, Forschung und Kunst im Jahr 2010 vergeben wurde. Am 04.12.2015 wurde Prof. Dr. Jürgen Mottok der „**Preis für besondere Leistungen bei der Zusammenarbeit zwischen Wirtschaft und Wissenschaft**“ verliehen. [1]Zentrum Digitalisierung.Bayern (ZD.B) Seybothstrasse 2, 93053 Regensburg