

Erweiterte Vorgehensmodelle für die Entwicklung echtzeitfähiger, hoch-integrierter, multifunktionaler Steuergeräte-Plattformen

Andreas Baudisch, AUDI AG, Ingolstadt

Dr. Kai Richter, Syntavision GmbH, Braunschweig

Stefan Sollmann, Elektronische Fahrwerksysteme GmbH, Gaimersheim

Kurzfassung

Die Umsetzung innovativer Funktionen in den Bereichen Elektrischer-Antrieb, Fahrwerk und Fahrerassistenz, die Einführung neuer Vernetzungsarchitekturen sowie die zunehmende Software-Eigenentwicklung durch Automobilhersteller führen zu neuen hoch-integrierten Steuergeräten mit flexiblen Integrations-, Skalierungs- und Sicherheits-Konzepten. Die Hochintegration ist der technologische Motor für viele neue Trends. So erfordert die einhergehende System-Komplexität z.B. den Einsatz von leistungsfähigen Produktentwicklungsprozessen.

Komplexität ergibt sich u.a. aus der Verwendung von vernetzten Funktionen, Mehrkern-Systemen und heterogenen Sicherheitsanforderungen innerhalb eines Steuergerätes. Dabei steigen die Risiken, die während einer Funktionsintegration speziell im Bereich der nicht-funktionalen Echtzeitanforderungen auftreten, stark an. Bei der Planung der Steuergeräte-Architektur werden neben den Kosten auch die verfügbare Rechenleistung und Kommunikationsbandbreite festgelegt, die inhärent die Verfügbarkeit, Sicherheit, Erweiterbarkeit und auch den Energieverbrauch bestimmen.

Es werden Möglichkeiten zur frühen Bewertung von Steuergeräte-Architekturen und Integrationskonzepten sowie zur kontinuierlichen Überprüfung der Echtzeitfähigkeit benötigt. Fehler bei der Umsetzung der Echtzeitanforderungen werden gewöhnlich in späten Entwicklungsphasen entdeckt und die Behebung der Fehler erfordert nicht selten eine aufwändige Anpassung der Architektur. Vorgehensmodelle, die eine Echtzeitmethodik während der gesamten Entwicklungsphase einsetzen, reduzieren dieses Risiko deutlich.

Im Beitrag zeigen wir zum einen, an welchen Stellen bestehende Vorgehensmodelle in der Architektur-Entwicklung angereichert werden können und welche Techniken, Methoden und Werkzeuge sich zur Überprüfung und Optimierung der Echtzeiteigenschaften in den verschiedenen Entwicklungsphasen eignen. Zum anderen berichten wir über unsere praktischen Erfahrungen anhand konkreter Projekterfahrungen.

Abstract

Realizing new, innovative functions in e-drive, chassis, and assistance domains, the introduction of new network architectures, and the increasing software development activities by OEMs lead to novel high-integration ECU platforms. With flexible integration, scalability and safety concepts, these platforms are becoming the enablers for many new technology trends. At the same time, the system complexity is increasing, demanding new, powerful development processes.

The complexity results from (re-)using networked functions, multi-core systems and heterogeneous safety requirements in a single system. The design risk increases, in particular in the area of non-functional real-time requirements because a large part of the computation and communication resource performance is determined in early planning of the ECU architecture with an inherent impact on system availability, safety and extensibility.

We are seeking for ways to evaluate and assess ECU architectures and integration concepts early, and for ways to continuously verify the real-time requirements over the entire system development cycle. Failures in defining a reasonable architecture to fulfill all real-time requirements are typically detected very late in the development cycle, often requiring a costly re-design of the ECU architecture. Appropriate design processes that are enriched by a real-time methodology, reduce this risk significantly.

In our contribution, we analyze the existing design processes and identify which steps can be enriched by which techniques, methods, and tools for real-time design, verification, and optimization. Further, we summarize our experience gained in the introduction of a real-time methodology in concrete projects.

1. Einleitung

Die Umsetzung innovativer Funktionen in den Bereichen Elektrischer-Antrieb, Fahrwerk und Fahrerassistenz, die Einführung neuer Vernetzungsarchitekturen sowie die zunehmende Software-Eigenentwicklung durch Automobilhersteller führen zu neuen hoch-integrierten Steuergeräten mit flexiblen Integrations-, Skalierungs- und Sicherheits-Konzepten. Die Hochintegration ist somit der technologische Motor für viele neue Trends. Die damit einhergehende System-Komplexität erfordert jedoch den Einsatz von leistungsfähigen Produktentwicklungsprozessen.

Dies wird am aktuellen Audi A8 deutlich. Neben der um ca. 40% gewachsenen Steuergeräteanzahl (Vergleich zum Vorgängermodell) ist auch die Softwaremenge um ca. 400% gestiegen. Trotz der wohl auch zukünftig wachsenden Softwaremenge muss die

Anzahl der Steuergeräte möglichst gering bleiben. Nur so können Energie-, Kosten- und Bauraum-Ziele erreicht und optimiert werden. Darüber hinaus wollen Automobilhersteller auch die Vorteile, die durch Softwareeigenentwicklung entstehen, nutzen.

Dies führt automatisch dazu, dass die Integrationsdichte in den Steuergeräten steigt. Höhere Integrationsdichte bedeutet aber auch höhere Komplexität in der Steuergeräte-Software. Es wird also tendenziell deutlich schwieriger die gegebenen Echtzeitanforderungen an das System durchgängig umzusetzen.

2. Echtzeitfähigkeit von Steuergeräte-Software

Eine wesentliche Herausforderung besteht darin, dass essentielle Entwurfsentscheidungen (CPU-Auswahl, Software-Partitionierung, Sicherheitskonzept) in der frühen Architektur-Design-Phase getroffen werden, jedoch mit den herkömmlichen Mitteln des Testens erst in späten Phasen verifizierbar sind. Daher brauchen wir Möglichkeiten

- zur **frühen Bewertung** von Architektur-Konzepten während der Design-Phase sowie
- zur **kontinuierlichen Überprüfung** der Echtzeitfähigkeit während der Implementierung.

Ohne diese Möglichkeiten werden Fehler bei der Umsetzung der Echtzeitanforderungen erst spät entdeckt. Die Behebung der Fehler erfordert dann nicht selten eine aufwendige und risikoreiche Anpassung der Architektur.

Ein erweitertes Vorgehensmodell für die Produktentwicklung reduziert dieses Risiko und liefert eine zuverlässige Vorhersage der benötigten Steuergeräte-Ressourcen. Dabei werden die folgenden Bereiche der Echtzeiteigenschaften berücksichtigt:

- Last- bzw. Laufzeitanforderungen der Software, bestehend aus Zykluszeit und Ausführungszeit,
- statische Software-Architektur, bestehend aus Software-Komponenten und Runnables bzw. Funktionsblöcken,
- dynamische Software-Architektur, bestehend aus Schedule, Runnable-to-Task-Mapping und Task-Prioritäten,
- Echtzeitvorgaben, meist bestehend aus Obergrenzen für Signal-Latenzen, Reaktionszeiten (Deadlines, auch als Ende-zu-Ende-Wirkketten) und CPU-Lasten. Hinzu kommen evtl. noch Forderungen nach „Interferenzfreiheit“ (ISO 26262) sowie implizite Vorgaben, z.B. dass jede Funktion ihre Zykluszeit einhalten muss.

Die wesentliche Aufgabenstellung lautet also: Die statische und dynamische Software-Architektur muss so gewählt werden, dass die Echtzeitvorgaben in der sich anschließenden Implementierung zuverlässig und kostengünstig eingehalten werden.

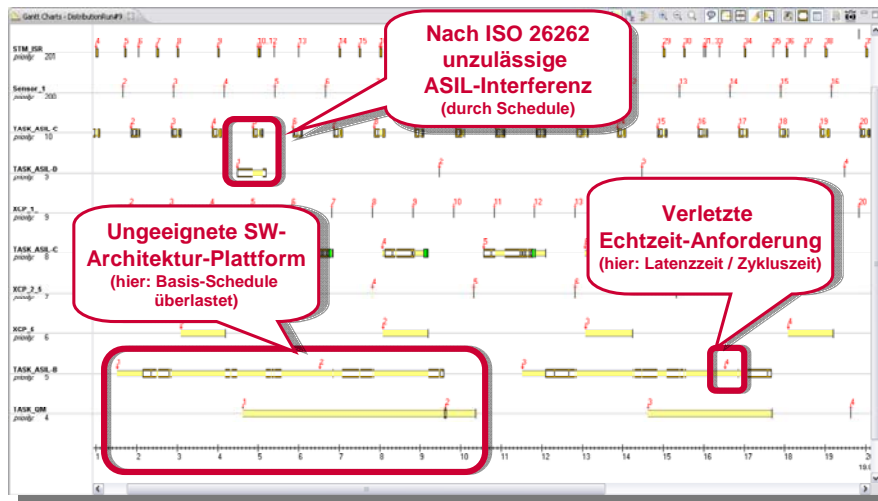


Abbildung 1 Typische Echtzeitfehler "in" einer ECU

Dies ist eine komplexe Aufgabe, denn die auf einem Steuergerät integrierten Funktionen beeinflussen sich u.U. auf vielfältige Art und Weise gegenseitig in ihrem Echtzeitverhalten. Abbildung 1 zeigt exemplarisch typische Echtzeitfehler anhand eines Blickes „in“ den zeitlichen Ablauf eines Steuergerätes hinein. Unvorhergesehene Ausführungszeiten können zu Überlastsituationen (unten links) und verpassten Zykluszeiten (rechts) führen. Wenn man die Schedule-Prioritäten nicht sorgfältig vergibt, kann die geforderte Interferenzfreiheit nicht garantiert werden. Ungünstige RTE-Konfigurationen führen zu unnötig langen Wirkketten-Verzögerungszeiten. Diesen und weiteren Echtzeitfehlern gilt es mit geeigneten Analyse-Techniken effektiv vorzubeugen.

3. Techniken zur Echtzeit-Analyse

Wir unterscheiden die folgenden Klassen von Echtzeit-Analyse-Techniken: Die Code- und Trace-basierten Techniken der Implementierungs- und Integrations-Phasen sowie die Modell- oder Anforderungsbasierten Techniken in den Entwurfsphasen (Design).

3.1 Trace-basierte Echtzeit-Verifikation

Ist das integrierte Steuergerät oder ein Prototyp verfügbar, können die Laufzeiten einzelner Funktionen, Tasks, Runnables, sowie die Laufzeiten der RTE und anderer Basissoftware durch Tracing ermittelt werden. Dazu werden zu den relevanten Zeitpunkten der Ausführung Zeitstempel aufgezeichnet (z.B. Start und Ende von Funktionen, ISRs, BSW-Routinen), nach

der Messung offline ausgewertet und mit den Echtzeitvorgaben verglichen. Diese Technik ist in der Spätphase anwendbar und mittlerweile weit verbreitet. Variante: alternativ zum „echten“ Target können für manche CPU-Architekturen zyklengenaue Hardware-Simulatoren verwendet werden.

3.2 Code-basierte Echtzeit-Verifikation

Ist der Target-Code verfügbar, kann dieser mittels WCET-Analysen (Worst-Case Execution-Time) auf das Laufzeitverhalten einzelner Funktionsaufrufe hin untersucht werden. Der Target-Code ist wichtig, da nur so die spezifischen Merkmale der genutzten Hardware-Architektur (Pipeline, Speicher, etc.) realistisch berücksichtigt werden können.

Alternativ kann eine Simulation auf Source-Code-Ebene durchgeführt werden. Dazu müssen zuvor die Laufzeiten der Code-Blöcke bestimmt und an den Source-Code annotiert werden.

3.3 Modell-basierte (oder Anforderungs-basierte) Echtzeit-Verifikation

Im Gegensatz zu den Code- und Trace-basierten Techniken kommt die Modell-basierte Timing-Analyse grundsätzlich ohne Implementierungsartefakte aus. An die Stelle der Code- bzw. Target-Details tritt ein parametrisiertes Timing-Modell mit den zu diesem Zeitpunkt bekannten Echtzeitanforderungen: Last- bzw. Laufzeitanforderungen, statische und dynamische Software-Architektur sowie Echtzeitvorgaben.

Aus diesem Modell generiert die Scheduling-Analyse dann automatisch eine Vielzahl von möglichen Ausführungssequenzen der Software, inklusive der Integrationseffekte. Die Darstellung entspricht dabei derjenigen aus dem Tracing. Es findet somit eine Vorhersage des zu erwartenden Echtzeitverhaltens statt. Daraus werden dann die zu erwartenden Echtzeiteigenschaften CPU-Last, Latenzen, Wirkketten erfasst und mit den spezifizierten Echtzeitvorgaben verglichen. Es gibt sowohl Worst-Case- als auch statistische Scheduling-Analysen (Distribution-Analysen).

3.4 Angereichertes Vorgehensmodell

Abbildung 2 zeigt nun ein V-Modell mit den relevanten Software-Entwicklungsphasen, angereichert um die Echtzeit-Analyse-Techniken aus den vorangegangenen Abschnitten. Es wird deutlich, dass in frühen Phasen (linke Seite des V-Modells) ausschließlich die Modell-basierten Techniken eingesetzt werden können, während die Code- und Trace-basierten Techniken den späteren Phasen vorbehalten sind. Im Folgenden beschreiben wir die für die

Steuergeräte-Integration essentiellen Software-Architektur-Phasen und gehen nachfolgend auf die benachbarten Phasen ein.

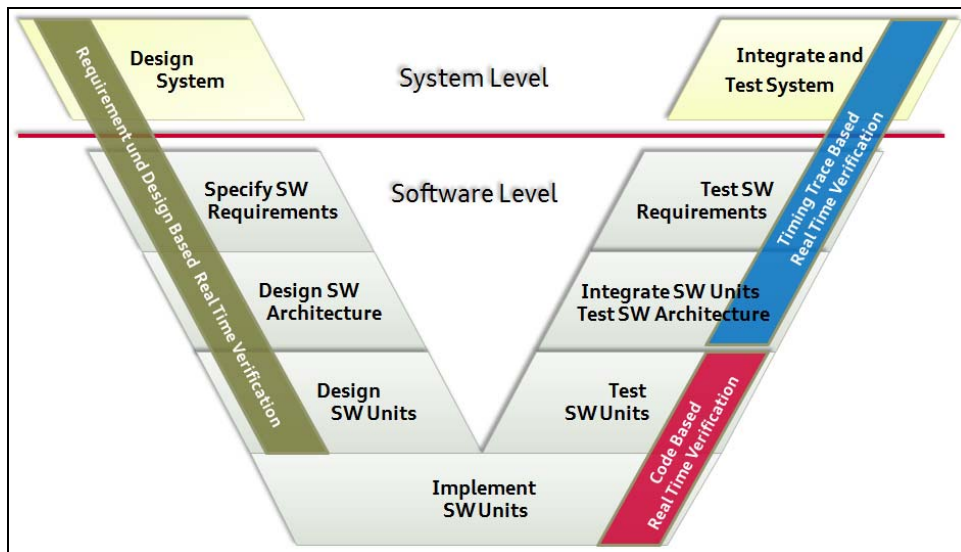


Abbildung 2 Angereichertes V-Modell

4. SW-Architektur-Design

In frühen Entwicklungsphasen steht i.d.R. noch kein Quellcode zur Verfügung. Trotzdem besteht der Wunsch Anforderungen und Entwurfsentscheidungen in den frühen Phasen überprüfen zu können. Von den in Kapitel 3 genannten Techniken ist die Scheduling-Analyse anwendbar, da sie sich auf die elementaren Architektur-Eigenschaften konzentriert und grundsätzlich ohne Code-Informationen und Trace-Möglichkeiten auskommt. Das konkrete Vorgehen ist auf der linken Seite von Abbildung 3 skizziert.

Als erster Schritt der Echtzeitbewertung wird aus den zum jeweiligen Projektstatus bekannten Informationen ein Timing-Modell erstellt. Die **Timing-Vorgaben** ergeben sich aus der Funktionsentwicklung, z.B. maximal erlaubten Reaktionszeiten oder Jitter. Bezüglich dieser Informationen kann auf bestehendes Know-How aus der Phase „Specify SW-Requirements“ zurückgegriffen werden. Die **statische und dynamische Software-Architektur** ist teilweise bereits bekannt, denn sie wird in dieser Phase grundlegend entworfen. Die für das Timing-Modell benötigten **Laufzeiten** werden in dieser frühen Phase zunächst budgetiert. Hier kann auf Erfahrungen der Funktions- und Software-Entwicklung in ähnlichen bzw. vergleichbaren Projekten zurückgegriffen werden. Sobald diese Daten erfasst sind (die Software-Architektur kann dabei leicht per AUTOSAR.XML importiert werden), erhalten wir mit Hilfe der Scheduling-Analyse wichtige Kennzahlen der zukünftigen Echtzeitfähigkeit wie CPU-Last, Latenzen und ASIL-Interferenzen.

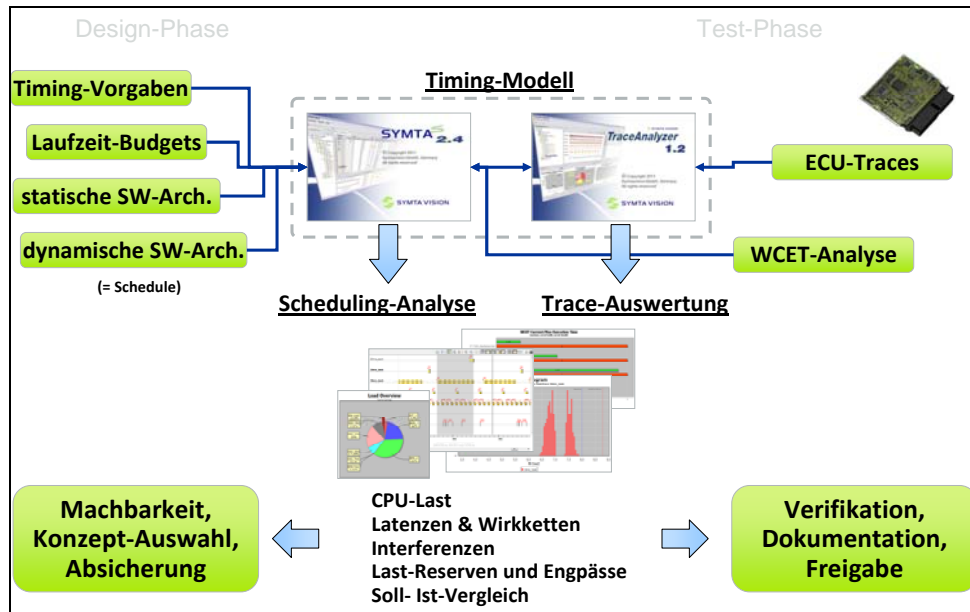


Abbildung 3 Vorgehen für Design&Test SW-Architektur

Diese geben Aufschluss über die generelle Machbarkeit des Projekts und liefern eine Absicherung des ausgewählten Architektur-Konzepts. Durch wiederholte Bewertung können alternative Architektur-Konzepte zudem miteinander verglichen werden, so dass die beste Option systematisch ausgewählt werden kann [3].

4.1 Wiederverwendung

Auch wenn die Budgets geschätzt und die Software-Architektur evtl. noch nicht 100% festgelegt sind, so ist das Ergebnis der Scheduling-Analyse das Beste, das wir zu diesem frühen Projektzeitpunkt bekommen können. Dabei liegen in der Praxis oft deutlich mehr Informationen vor, als es der reine Budgetierungs-Ansatz zunächst suggerieren mag. Für wiederverwendete Software-Teile liegen in der Frühphase des einen Projektes bereits Implementierungen aus den Spätphasen von Vorgängerprojekten vor, nebst Traces und eventuell auch WCET-Daten, die wiederbenutzt werden können. Somit beschränkt sich die Budgetierung auf die neu hinzukommenden bzw. modifizierten Software-Teile. Dasselbe gilt für bereits erfolgreich erprobte Software-Architekturen, in die „lediglich“ die neuen Software-Komponenten hinzu integriert werden müssen.

Dieser Modell-basierte Ansatz zur Echtzeitbewertung wurde bei der AUDI AG bereits in verschiedenen Projekten angewendet, um getroffene Architekturentscheidungen zu begründen und wirksam abzusichern und im Einzelfall auch zu optimieren. Über standardisierte Austauschformate zwischen den genutzten Trace-, Modellierungs- und

Analyse-Werkzeugen wurde das Vorgehen zudem teilweise automatisiert und damit der Gesamtprozess signifikant beschleunigt.

4.2 SW-Architektur-Integration und –Test

Haben wir uns bislang auf die Architektur-Design-Phase konzentriert, bietet sich in der Integrations- und Test-Phase eine effektive Kombinationsmöglichkeit aus Tracing (Standardmethode in der Test-Phase) und Scheduling-Analyse an. Basierend auf Laufzeitinformationen aus den Traces des nun vorhandenen Target werden die Scheduling-Analysen wiederholt (siehe rechte Seite von Abbildung 3), um eventuell vorhandene Lücken in den Testläufen aufzudecken. Dieses Vorgehen der zusätzlichen Absicherung der Echtzeitfähigkeit ist bereits erprobt. Dies gilt speziell für die Bewertung sicherheitskritischer Anwendungen, in denen die Testabdeckung nachweislich erhöht und somit Haftungsrisiken minimiert und die Verfügbarkeit optimiert wurden [1,2]. Analog kann das Timing-Modell auch um WCET-Informationen angereichert werden.

5. SW-Implementierung und Unit Test

Ab der Implementierungsphase kann durch eine Code-Analyse-Technik die Worst Case Execution Time (WCET) für Funktionen und Blöcke bestimmt werden. Diese Information kann auf zwei Arten genutzt werden, um frühzeitig (auch vor der Integration) Engpässe und Gefahren aufzudecken und zu eliminieren.

Erstens können die WCET-Werte jeder in der Implementierung befindlichen Funktion direkt mit den dafür festgelegten Laufzeit-Budgets aus der Design-Phase verglichen werden. Liegen die tatsächlichen Laufzeiten innerhalb der Budgets, ist die Implementierung konsistent mit dem Timing-Modell und die zugesicherten Eigenschaften bleiben weiterhin gültig.

Zweitens liefert eine neuerliche Scheduling-Analyse schnell Auskunft darüber, ob aus einem verletzten Budget ein tatsächliches Problem bei der Integration entstehen kann, oder ob die Echtzeitvorgaben trotzdem noch eingehalten werden können. Im erstgenannten Fall müssen die Implementierung und/oder die Software-Architektur angepasst werden. Im zuletzt genannten Fall reicht eine Aktualisierung des aufgestellten Timing-Modells aus.

Tatsächlich ist eine neuerliche Durchführung der Scheduling-Analyse auch dann empfehlenswert, wenn Laufzeit-Budgets nicht vollständig ausgenutzt werden, denn die so „frei“ werdenden Ressourcen können auf andere Software-Teile umgeschichtet werden. In jedem Fall wird durch die kontinuierliche Soll-Ist-Überprüfung anhand eines Timing-Modells

das Risiko später Integrationsfehler systematisch eingegrenzt. Etwaige Gegenmaßnahmen können also frühzeitig ergriffen und begründet werden.

6. Netzwerkdesign und -Test

Die Absicherung der Echtzeiteigenschaften in der System- und insbesondere der Netzwerk-Design-Phase wird durch eine Vorgehensweise erreicht, die dem Vorgehen in den SW-Architektur-Phase grundsätzlich ähnlich ist (siehe Kapitel4), denn auch hier sind die Scheduling-Entscheidungen maßgeblich für die Echtzeitfähigkeit. In einem Punkt ist diese System- bzw. Netzwerk-Ebene etwas einfacher, da nämlich keine Laufzeit-Budgets abgeschätzt werden müssen. An ihre Stelle treten die Signal-Größen, da diese maßgeblich für die „Belegungsdauer“ von Kommunikation auf den Bussen sind, zusammen mit dem Signal-to-Frame-Mapping sowie der Bus-Konfigurationsparameter (CAN Id, FlexRay Slot, etc.).

Neben der Betrachtung der reinen Bus-Kommunikation treten in Bezug auf Latenzen und Verzögerungen auch Wechselwirkungen zwischen dem System-Design und dem SW-Architektur-Design auf. Annahmen über Durchlaufzeiten auf der Systemebene werden zu Vorgaben für das SW-Architektur-Design im Steuergerät. Umgekehrt können wir mit abgesicherten Latenzeigenschaften die Modelle zur Verifikation des System-Timings anreichern.

Durch den Einsatz der genannten Techniken haben wir die Auswirkungen eines Steuergeräte-Umstiegs von CAN nach FlexRay vorhergesagt und konnten die Software-Architektur des betroffenen Steuergerätes schon in der Design-Phase systematisch an die neuen Anforderungen anpassen, insbesondere bezüglich Synchronisation und Basis-Schedule.

Die notwendige Erstellung des Timing-Modells wird in der Praxis auch hier durch Imports der gängigen Netzwerk-Formate (DBC, Fibex, AUTOSAR System Template) vereinfacht. In der Testphase können ebenfalls Bus-Traces über den TraceAnalyzer in das Timing-Modell aufgenommen und mit den Vorgaben verglichen werden.

7. Zusammenfassung

Hoch-integrierte Steuergeräte bilden den technologischen Motor für viele neue Trends, erfordern aber auch besonders leistungsfähige Entwicklungsprozesse und Vorgehensmodelle. Insbesondere im Bereich der Software-Architekturen sind Möglichkeiten zur frühen Bewertung sowie zur kontinuierlichen Überprüfung der Echtzeitfähigkeit essentiell für den

Projekterfolg. Eine späte Entdeckung von Fehlern bei der Umsetzung der Echtzeitanforderungen erfordert nicht selten eine aufwendige und risikoreiche Anpassung der Software-Architektur.

Wir haben im Beitrag ein um eine Echtzeitmethodik erweitertes Vorgehensmodell gezeigt, das dieses Risiko nachvollziehbar reduziert. Das dargestellte Vorgehensmodell und die damit verbundenen Modell-basierten Techniken erlauben uns eine frühzeitige Verifikation

- der statischen und dynamischen Software-Architektur,
- einer ASIL-gerechten Partitionierung und
- einer geeigneten HW-Plattform (CPU).

Über die verschiedenen Phasen der Produktentwicklung hinweg ist es zudem jederzeit möglich, einzelne Echtzeiteigenschaften zu überprüfen, mit den Anforderungen abzugleichen und notwendigenfalls geeignete Korrekturmaßnahmen zu ergreifen. Die eingesetzten Timing- und Scheduling-Modelle eignen sich zudem auch zum Austausch entlang der Zulieferketten, so dass auch hier die Risiken frühzeitig minimiert werden.

In mehreren Projekten haben wir nach dem dargestellten Vorgehen das dynamische Verhalten des Systems frühzeitig abgesichert und Architekturentscheidungen dokumentiert. Wir haben Lastreserven aufgezeigt, mögliche Architekturrisiken identifiziert und Optimierungen durchgeführt. Damit haben wir jeweils einen nachweislich signifikanten Beitrag zur kostenoptimierten Erreichung der geforderten Anforderungen erbracht.

Derartig angereicherte Vorgehensmodelle werden in Zukunft unverzichtbar sein, denn sie erfüllen neue, kritische Prozessanforderungen, die bei hoch-integrierten Systemen entstehen, und die es früher in diesem Umfang nicht gab. Die vorgestellte Lösung ist somit eine Grundvoraussetzung für die kostenoptimierte und zuverlässige Realisierung vieler neuer Trends. Audi und Syntavision treiben Standard-Lösungen voran und suchen den Austausch mit weiteren OEMs und Lieferanten.

Literatur

- [1] Thorsten Jablonski, Carsten Busse, Dieter Brinkema, Marek Jersak, Kai Richter. Timinganalyse als SIL-3 Sicherheitsnachweis. Hanser Automotive, Ausgabe 11.2008.
- [2] Marek Jersak, Kai Richter, Hans Sarnowski, Peter Gliwa. Laufzeitanalysen zur frühzeitigen Absicherung von Software. ATZ Elektronik, Ausgabe 01.2009.
- [3] Rafik Henia, Arne Hamann, Rolf Ernst, et.al. System-Level Performance Analysis - the SymTA/S Approach. IEE Proceedings Computers and Digital Techniques, 2005.