Kommunikationsoverhead bei Multi-Core Systemen früh beherrschen

Frühzeitiges Erkennung und Ableiten von Maßnahmen im Prozess

Jan Meyer, Hella KGaA Hueck & Co.
Ingo Houben, Dr. Ralf Münzenberger, INCHRON GmbH

Kostendruck und technologische Herausforderungen bedingen den Trend die Anzahl der Steuergeräte im Automobil zu reduzieren. Parallel gibt es einen erheblichen Anstieg an Funktionalität und einen steigenden Druck neue Funktionalität schneller im Auto bereit zu stellen. Daher erscheint der Einsatz von Multi-Core Prozessoren eine Lösung darzustellen, um eine große Steigerung der Performance zu erreichen. Häufig zeigt es sich erst in späten Entwicklungsphasen, dass der gewünschte Performancegewinn nicht erzielt wird. Bei genauerer Untersuchung der Ursachen stellt sich häufig heraus, dass es einen impliziten Lastzuwachs und eine Erhöhung der Latenzen von zeitkritischen Wirkketten durch die Inter-Core-Kommunikation gibt, hervorgerufen durch die Verteilung der Software auf verschiedene Rechenkerne.

In diesem Artikel wird eine Methodik vorgestellt, um bereits in frühen Designphasen – und nicht erst in der Integrationsphase – diese Aspekte einzubeziehen. Ein großer Vorteil dieses Ansatzes besteht in der Vermeidung von Zeit und kostenaufwändigen Re-Designs und führt insgesamt zu einer deutlichen Reduzierung des Projektrisikos, Fehlentscheidungen zu treffen. Diese Methodik wird in der Praxis bereits beim Automobilzulieferer Hella KGaA Hueck & Co. erfolgreich im Entwicklungsprozess eingesetzt.

Ansteigender Performance-Bedarf von Applikationen

Innovationen im Automobilbereich sind in letzter Zeit vor allem durch eine Zunahme von Software und kooperierenden Systemen geprägt. Ein Beispiel ist die Adaptive Cruise Control (ACC), welche die Funktion eines Tempomats mit einer Objekterkennung kombiniert, um das Fahrzeug mit einer sicheren Geschwindigkeit voranzubringen. Die Funktionalität wird in Abb. 1 dargestellt. Zunächst wird das Fahrzeug auf eine voreingestellte Geschwindigkeit beschleunigt. Wird mittels der Objekterkennung ein vorausfahrendes langsameres Fahrzeug erkannt, so wird die Geschwindigkeit automatisch reduziert. Wenn das vorrausfahrende Fahrzeug nicht mehr vorhanden ist oder beschleunigt, wird die Geschwindigkeit bis zur ursprünglichen

Abb. 1: Schematische Darstellung eines Adaptive Cruise Control System

Geschwindigkeit wieder erhöht.

Für einen automobilen Zulieferer ist dabei von Bedeutung, wie mit dem Softwarezuwachs umgegangen werden kann. Um den benötigten Lastzuwachs in einem Mikrocontroller zu integrieren, werden vermehrt Multi-Core Prozessoren eingesetzt. Dieser suggeriert bei der vergleichbaren Baugröße ein Vielfaches an Rechenleistung durch die verbauten zusätzlichen Kerne. Nicht selten tritt jedoch als Problem auf, dass der benötigte Leistungsgewinn sich nicht einstellt oder deutlich geringer als erwartet ausfällt. Ein häufiger Grund hierfür ist die notwendige Kommunikation zwischen den Rechenkernen. Damit bei der Entwicklung keine Überraschungen bezüglich der Leistungsfähigkeit entstehen, muss dieses Thema bereits in sehr frühen Entwicklungsphasen betrachtet werden. Unsere Erfahrungen aus vielen Entwicklungsprojekten zeigen deutlich, dass auch erfahrene Architekten die Probleme bei der Migration von Single Core zu Multi-Core unterschätzen.

Ausgangspunkt Single-Core Prozessor

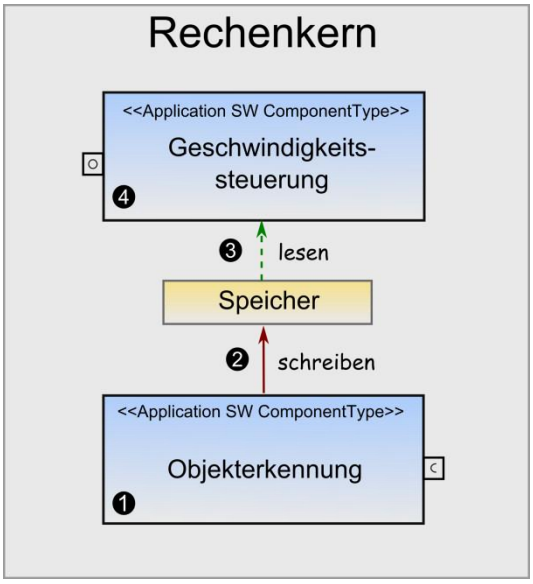

Abb. 2: Kommunikation zwischen zwei SW-Komponenten innerhalb eines Prozessorkerns

Anhand des übersichtlichen ACC-Beispiels wird die Kommunikation zwischen den beiden SW-Komponenten Objekterkennung und Geschwindigkeitskomponente sowie die Datenverarbeitung betrachtet (siehe **Fehler! Verweisquelle konnte nicht gefunden werden.**). Diese besteht aus folgenden vier grundlegenden Schritten, wobei aus Darstellungsgründen auf eine Beschreibung der konkreten AUTOSAR Kommunikationsmechanismen [1] verzichtet wird: In der *Objekterkennungskomponente* werden vorausfahrende Fahrzeuge erkannt und entsprechende Daten vorbereitet und in einem zweiten Schritt in den *Speicher* geschrieben. Die *Geschwindigkeits-komponente* liest diese Daten in einem dritten Schritt und berechnet anschließend im vierten und letzten Schritt die zu fahrende Geschwindigkeit des Fahrzeugs. Die Prozessorlast wurde unter Einsatz des Echtzeitsimulators chronSIM ermittelt und als Lastdiagramm über die Zeit für jede SW-Komponente in **Fehler! Verweisquelle konnte nicht gefunden werden.** übersichtlich dargestellt.

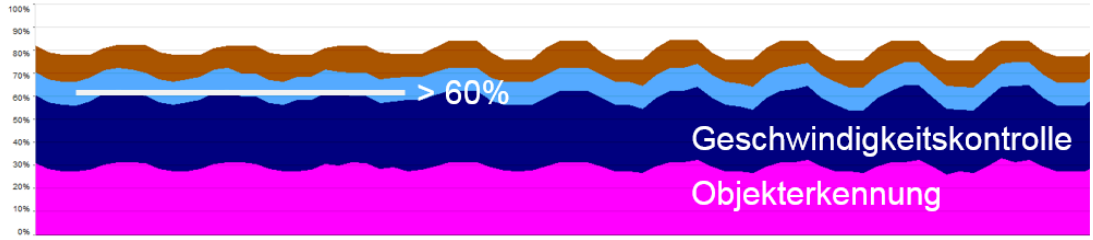

Abb. 3: Lastverlauf über die Zeit auf einem Single-Core Prozessor

Die beiden betrachteten SW-Komponenten *Objekterkennung* und *Geschwindigkeitsteuerung* werden auf einem Prozessorkern ausgeführt. Zusammen erreichen beide bereits einen Lastanteil von *über 60%*. Da keine ausreichenden Lastreserven für sogenannte Change Requests vorhanden sind und weitere Applikationen zu integrieren sind, ist der Einsatz eines Mehrkernprozessors vorgesehen.

Lastanstieg durch Inter-Core Kommunikation

Werden die beiden SW-Komponenten auf unterschiedlichen Prozessorkernen implementiert, können Effekte auftreten, die so nicht erwartet werden. Eine genauer Untersuchung und Vergleich der auftretenden Lasten zeigt auf, dass ein deutlicher Lastzuwachs vorliegt. Bei ersten Überlegungen ist dies nicht offensichtlich, da der gleiche Code ausgeführt und die gleiche Kommunikation stattfindet.

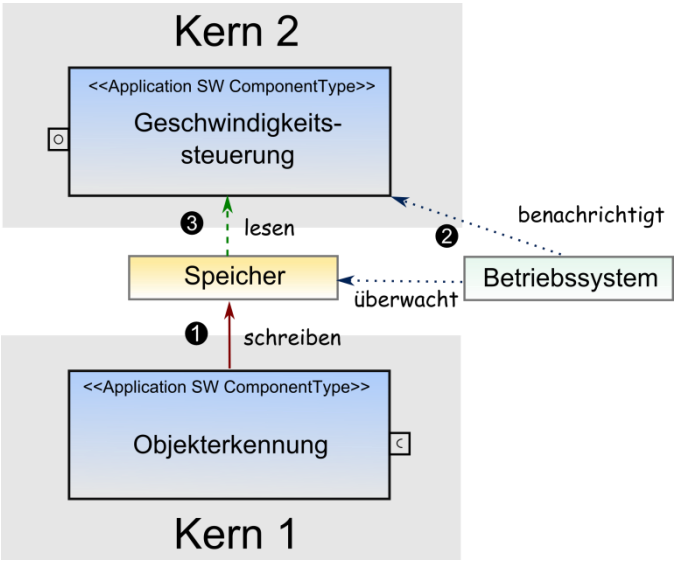

Abb. 4: Kommunikation zwischen zwei SW-Komponenten über Prozessorkerngrenzen

Bei näherer Betrachtung stellt sich heraus, dass in Abhängigkeit der gewählten Kommunikationsart zusätzliche Lasten entstehen. Im Detail verändert sich der Ablauf zu: *Objekterkennung* erzeugt Daten und schreibt diese in den *Speicher* (siehe **Fehler! Verweisquelle konnte nicht gefunden werden.**). Das Laufzeitsystem erkennt dieses und stellt der *Geschwindigkeitssteuerung* die neuen Daten zur Verfügung. In Abhängigkeit des gewählten Kommunikationsmechanismus (z.B. Sender-Receiver, Client-Server, mit oder ohne Empfangsbestätigung) ist die durch die Kommunikation erzeugte Last sehr stark unterschiedlich.

In dem aufgezeigten Bespiel wird eine Client-Server Kommunikation verwendet, bei der die Objekterkennung mehrmals in einen Bearbeitungszyklus Daten in den Speicher schreibt. Die Geschwindigkeitssteuerung wird unter Verwendung eines Interrupts über die Bereitstellung der neuen Daten informiert, wobei für jedes Datum ein Interrupt auf Prozessorkern zwei ausgelöst wird. Eine eingehende Betrachtung zeigt den durch die „Kommunikations-„ Interrupts verursachten Lastzuwachs deutlich: Das Verteilen der Applikationen auf zwei unterschiedliche Kerne hat eine zusätzliche Interrupt-Last von ungefähr 8% zur Folge (siehe Abb.5). Das heißt, jeder Kern wird mit zusätzlichen 8% belastet und das gesamte System mit 16% mehr Last, im Vergleich zur Single-Kern Implementierung.

Darüber hinaus entsteht ein nicht unerheblicher Start-to-Start Jitter sowie längere Unterbrechungen für weitere Applikationen und ggf. auch für nieder-priore Interrupt-

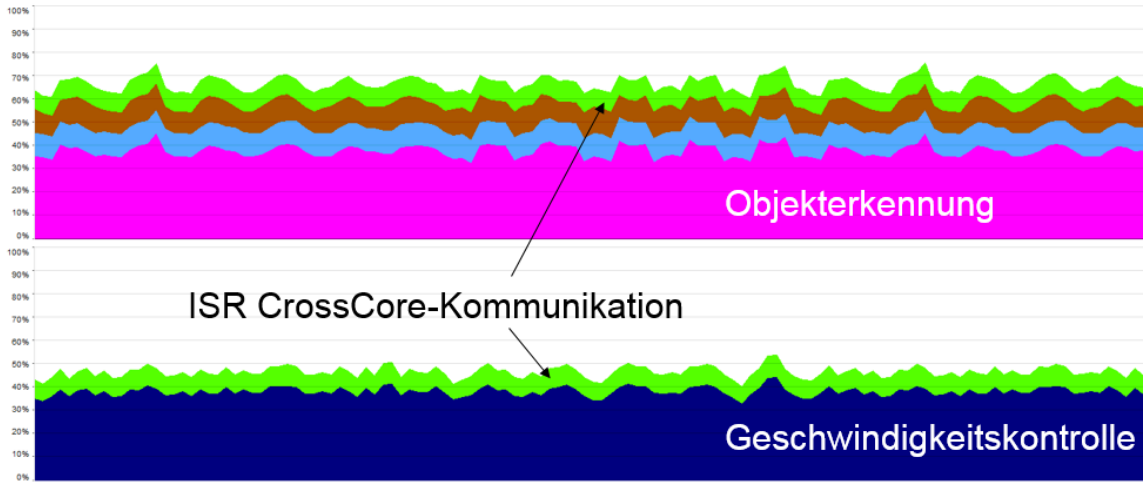

Abb. 5: Lastverlauf über die Zeit auf einem Dual-Core Prozessor

Service-Routinen. Dies ist dadurch verursacht, dass zuerst die „Kommunikations-„ Interrupt-Service-Routinen ausgeführt werden, da diese häufig eine höhere Priorität aufweisen. Hierdurch kann, zum Beispiel beim zeitgesteuerten FlexRay-Bus, das Lesen und Schreiben von Nachrichten vom im Kontext einer Interrupt-Service-Routing aus dem Tritt geraten. Insgesamt ist ein solches Verhalten ohne umfangreiche Erfahrung mit Multikernprozessoren und einer eingehenden Untersuchung der Kommunikation zwischen den Prozessorkernen, bzw. der Auswirkungen auf das System kaum vorhersagbar. Die Verwendung von Konfigurations-Tools und die automatisierte Realisierung der Inter-Prozess-Kommunikation durch ein Laufzeitsystem erschwert den Architekten einzuschätzen, welche Auswirkungen die Auswahl eines Kommunikationsmechanismus hat. Ein für Prozessoren mit einem Kern effizienter Mechanismus ist ggf. für einen Prozessor mit zwei Kernen ungeeignet. Die Vorteile der höheren Automatisierung und Vereinfachung hat leider den großen Nachteil der Entfremdung von den realen Effekten. Die Sichtbarkeit für solch technische Effekte wird in den Hintergrund gedrängt, bzw. geht verloren. Basierend auf vielfältigen Erfahrungen aus Migrationsprojekten kann dieser Lastzuwachs nicht selten bis zu 35 % betragen. In Projekten nach dem AUTOSAR 4.x Standard zeigt sich ein solch großer Lastanstieg häufig allein verursacht durch die Inter-Core Kommunikation von Diagnose-Nachrichten (DEM).

Zusätzlich ist zu beachten, dass es zu einem Lastanstieg auf einem Prozessorkern kommen kann, der durch eine Applikation auf einen anderen Prozessorkern bedingt ist: Dies wird durch die notwendige Kommunikation verursacht. Es gibt somit Lasterhöhungen und Auswirkungen auf Laufzeiten über Prozessorgrenzen hinweg und das obwohl die Applikation nicht an der Kommunikation beteiligt ist und die störenden Auswirkungen von einem anderen Prozessorkern ausgehen. Ein solcher Effekt ist weder intuitiv noch ist der induzierte Lastanstieg ohne eingehende Untersuchungen quantifizierbar.

Insgesamt bedarf es zusätzlicher Maßnahmen, um solche Probleme frühzeitig zu erkennen und entsprechend gegenzusteuern. Ein adäquates Mittel ist die virtuelle Integration [2]. Hierbei werden bereits während der System- und Software-

Architekturphase – und nicht erst bei der Integration – die Applikationen zu einem Gesamtsystem zusammengefügt (virtuell integriert). Die Bewertung und Optimierung der Architektur kann simulativ, z.B. mit dem Echtzeitsimulator chronSIM, oder durch eine Worst-Case Validierung, wie sie z.B. chronVAL ermöglicht, bereits in dieser frühen Phase erfolgen. Eine Architektur wird somit insgesamt bereits per Design robust [3]. Die Hella KGaA Hueck & Co nutzt bereits seit längerer Zeit die sich aus dieser Methodik ergebenen Vorteile.

Hierdurch wird sichergestellt, dass die unerwünschten Nebeneffekte bei der Nutzung eines Multi-core Prozessors nicht auftreten. Damit die virtuelle Integration möglichst ohne großen Aufwand durchgeführt werden kann, ist eine Einbettung in den Entwicklungsprozess zwingend notwendig. Erst hierdurch wird sichergestellt, dass die notwendigen Informationen frühzeitig bereitgestellt werden und eine Analyse möglich ist.

Einbettung in den Entwicklungsprozess

Der Entwicklungsprozess eines automobilen Zulieferers entspricht oftmals dem klassischen V-Modell und ist nach dem Prozessreferenzmodell Automotive SPICE [4] in mehrere Entwicklungsphasen (ENG.1-10) aufgeteilt. Die virtuelle Integration ist dabei den Entwicklungsphasen *Systemarchitektur* bzw. *Softwarearchitektur* zugeordnet (siehe Abb. 6). Die Systemarchitektur ist die Grundlage für die Integration. Sie liefert die Basis-Informationen für Aussagen bezüglich der Auslastung und der Performance des Systems. In der Softwarearchitektur sind zusätzliche Daten verfügbar, die weitergehende Ergebnisse aus der virtuellen Integration ermöglichen, beispielsweise die Sicherstellung der Datenintegrität oder aber die Wirkkettenanalyse.

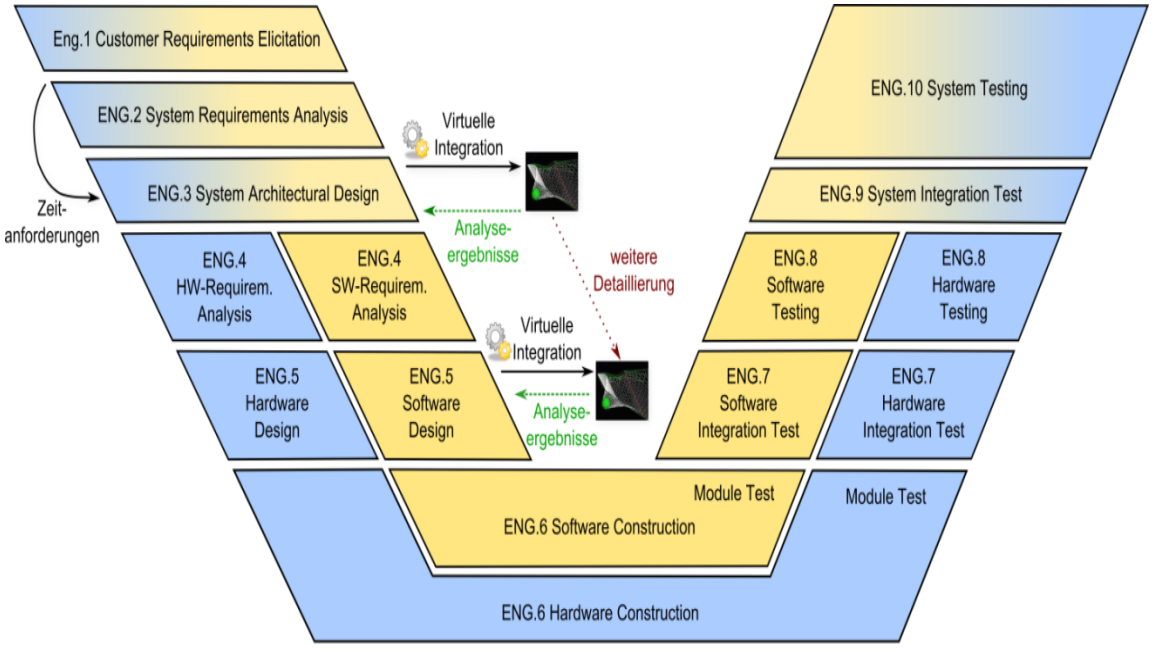

Abb. 6: Einordnung der virtuellen Integration in den Entwicklungsprozess

Das V-Modell wird im Laufe der Entwicklung in mehreren Musterphasen immer wieder durchlaufen. Dies fängt bei der Angebotsphase an und hört beim Start der Produktion (Start of production (SOP)) auf (siehe Abb. 7). Wesentliche Entscheidungen für das System werden bereits in der Angebotsphase vorgenommen. Hier werden aufbauend auf einer ersten Systemarchitektur die Baukosten für ein Steuergerät bestimmt und ein Angebot erstellt. Muss diese Architektur später überarbeitet werden, kann eine deutliche Kostenzunahme erfolgen, die nicht mehr durch das Angebot abgedeckt wird. Dies betrifft vor allem den Austausch des Prozessors. Von daher wird an dieser Stelle die virtuelle Integration genutzt, um die Last des Prozessors (besonders bei einem Multi-Core) zu ermitteln, damit es in den späteren Entwicklungsphasen keine bösen Überraschungen gibt.

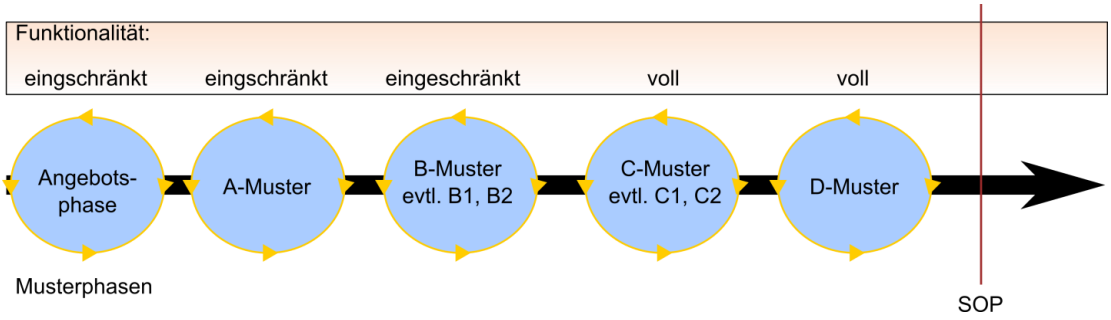

Abb. 7 Musterphasen in der automobilen Steuergeräteentwicklung

Als Voraussetzung für die virtuelle Integration sind Eingangsdaten aus der Architektur notwendig. Früher wurde bei der automobilen Steuergeräteentwicklung ein dokumentenbasierter Ansatz gewählt, der dafür sorgte, dass Daten in verschiedenen Dokumenten verteilt sind und mühsam zusammengesucht werden mussten. Zur Optimierung dieses Vorgehens und vor allem zur Konsistenzsicherstellung wurde bei der Hella KGaA Hueck & Co. ein modellbasierter Ansatz mit der Systems Modeling Language (SysML), Unified Modeling Language (UML) und AUTOSAR entwickelt und eingeführt [5]. In dem Ansatz sind automobile und echtzeitbeschreibende Erweiterungen integriert, die dafür sorgen, dass die für die virtuelle Integration notwendige Daten im Modell hinterlegt sind [7]. Auf Grund dieser Tatsache sind die notwendigen Informationen zentral an einer Stelle aufzufinden und stehen für die virtuelle Integration zur Verfügung.

In der Angebotsphase ist nicht genügend Zeit gegeben, um eine komplette Entwicklung durchzuführen. Von daher beinhaltet sie nur die angebotsrelevanten Bestandteile, wie eine grobe Architekturdarstellung und eine erste Aufteilung der Software auf Hardware. Zusätzlich wird nun die Kommunikation der Softwarekomponenten mit berücksichtigt. Infolgedessen werden in der virtuellen Integration mehrere Verteilungen analysiert, um eine möglichst optimale Performance zu erreichen. Dies ist besonders bei der Verteilung auf einem Multi-Core Prozessor notwendig. Zur Vermeidung von zusätzlicher Arbeit werden die Informationen aus der Architekturmodellierung mittels einer automatischen Transformation direkt von dem SysML/UML Modell in die INCHRON Tool-Suite übertragen [5]. In vorherigen Entwicklungsprozessen wurden im ersten Schritt die Verteilung der Software auf die Hardware und die Ausführungszeiten übertragen [7]. Es

hat sich aber heraus gestellt, dass für AUTOSAR Systeme mit einem Multi-Core Prozessor die Modelltransformation, um Kommunikationsbeziehungen ergänzt werden musste. Somit kann nun mit dem neuen Ansatz schnell überprüft werden, ob die erwartete Performance erreicht wird. Deswegen wurde das SysML/UML Modell mit den Parametern für die Kommunikationsbeziehungen erweitert, diese werden nun je nach Verteilung der Komponenten vom Simulationswerkzeug chonSIM automatisch eingefügt. Nun können basierend auf diesen Beziehungen die Abschätzungen wesentlich genauer erfolgen und fundierte Architekturentscheidungen getroffen werden.

Zusammenfassung

In diesem Beitrag wurde auf die Problemstellung Lasterhöhung beim Einsatz von Multi-Core Prozessoren eingegangen. Der Fokus lag hierbei auf Lasterhöhungen bedingt durch die Inter-Core Kommunikation. Es wurde aufgezeigt, dass bereits in frühen Entwicklungsphasen Kommunikationsbeziehungen eingehen zu untersuchen sind, damit die versprochene Leistungssteigerung sich in dem endgültigen System wiederfindet. Die vorgestellte Methodik kombiniert einen modellbasierten Ansatz mit der virtuellen Integration der Applikation / SW-Komponenten auf dem Multi-Core Prozessor, so dass noch keine reale Hardware oder eine Implementierung der Software vorhanden sein muss. Machbarkeitsstudien, Optimierungen sowie ein Treffen der richtigen Architekturentscheidungen sind durch eine toolbasierte Untersuchung und Bewertung der Vor- und Nachteile von Architekturvarianten schnell möglich. Ausgangspunkt für einen hohen Automatisierungsgrad sind die bei der Hella KGaA Hueck & Co. durchgeführten Erweiterungen bei der Architekturmodellierung (SysML/UML). Dies wird kombiniert mit einer Bewertung der Ergebnisse aus der INCHRON Tool-Suite. Bereits in der Angebotsphase lassen sich die hieraus gegebenen Vorteile nutzen: Dies sind neben einem zeit-und kosteneffizienten Vorgeben auch eine deutliche Reduzierung des Projektrisikos für Fehlentscheidungen.

Literatur- und Quellenverzeichnis

[1] www.autosar.org/specifications/.

[2] T. K. R. Münzenberger, Absicherung des Echtzeitverhaltens mittels virtueller Integration, Hanser Automotive, 11-2010.

[3] T. K. R. Münzenberger, AUTOSAR – geringes Projektrisiko mit robusten Architekturen, Hanser Automotive, 12 2012.

[4] J. Meyer und W. Horn, „Modellbasiertes Systemengineering zur Qualitätsverbesserung bei der Entwicklung eines automobilen Steuergerätes," in *Tag des Systems Engineering (TdSE)*, Carl Hanser Verlag, 2013, pp. 315 - 324.

[5] W. Horn und J. Meyer, „Modellbasiertes Systemengineering zur Qualitätsverbesserung bei der Entwicklung eines automobilen Steuergerätes," in *Tag des Systems Engineering (TdSE)*, Carl Hanser Verlag, 2013, pp. 315 - 324.

[6] T. Kramer, U. Nickel und J. Meyer, „Wie hoch ist die Performance?,“ *Automobil-Elektronik,* Bd. 03, 2010.

Autoren:

Jan Meyer arbeitete 2006 - 2011 als wissenschaftlicher Mitarbeiter am Software Quality Lab (s-lab) der Universität Paderborn. Er war dort zusammen mit industriellen Kooperationspartnern an verschiedenen Forschungsprojekten im Bereich der modellbasierten und verteilten Softwareentwicklung, insbesondere im automobilen Umfeld, beteiligt. Seit 2011 arbeitet er bei der Hella KGaA Hueck & Co. in der Abteilung Prozesse, Methoden und Tools. In dieser ist er verantwortlich für die modellbasierte Architekturentwicklung mit SysML, UML, (AUTOSAR) und für Echtzeitsimulationen.

Ingo Houben ist als Business Development Engineer bei der INCHRON GmbH tätig. Seit 2010 beschäftig er sich mit dem Zeitverhalten von Embedded Systemen, Bussen und Netzwerken. Er hat langjährige Erfahrungen in der Mikroelektronik und das detaillierte Wissen über Entwicklungs Prozesse auf die Embbeded Bereiche übertragen.

Dr.-Ing. Ralf Münzenberger ist als Mitgründer der INCHRON GmbH für den Bereich Professional Services als Geschäftsführer verantwortlich. In mehr als 150 Projekten hat er Kunden rund um das Thema Design von robusten dynamischen Architekturen und bei der Sicherstellung der Echtzeitfähigkeit insgesamt weitreichend unterstützt. Dies umfasst im einzelnen Themen wie Architekturoptimierung, Migration von Single-Core nach Multi-Core, funktionale Sicherheit oder Prozessberatung. Seine wissenschaftlichen Ergebnisse zum Thema Echtzeitmodellierung hat er in Standardisierungsorganisationen (ITU, AUTOSAR) umfangreich eingebracht.

Kontakt:
Internet: www.inchron.com
Email: info@inchron.com