Modellbasierte Architekturentwicklung und Simulation

Praxisbeispiel Fahrerassistenz mit AMALTHEA und chronSIM

Thomas Jäger, Robert Bosch GmbH
Ingo Houben, Dr.-Ing. Ralf Münzenberger, INCHRON GmbH

Ein modellbasierter Ansatz, beim dem aus einer Single-Source-Quelle weitere Prozessschritte abgeleitet werden, bietet viele Vorteile. Das Systemmodell von AMALTHEA dient hierbei als Ausgangspunkt, um eine robuste dynamische System- und Software-Architektur zu entwerfen sowie Source-Code und Testfälle zu erzeugen. In diesem Artikel wird ein Workflow vorgestellt, wie ausgehend von einem AMALTHEA Systemmodell eine Architektur simuliert, analysiert und im Fehlerfall optimiert wird.

Motivation

Die Entwicklung von Fahrerassistenz-Systemen, seien es Lösungen auf einzelnen Steuergeräten wie einer Kamera, oder Netzwerk-Verbünde mehrerer Sensoren und Kontrollrechner, beginnt mit der Konzeption des Systemdesigns und wird in der daraus abgeleiteten Software-Architektur verfeinert. Die Herausforderungen liegen in der Beherrschung der zugrunde liegenden parallelen Hardware, wie Multicore-Rechnern, oder der Nebenläufigkeit von ganzen Steuergeräte-Netzen. Die verschiedenen logischen Pfade der Datenverarbeitung auf solchen Systemen verlaufen demzufolge parallel, und weisen im Zuge der Informations-Verdichtung Synchronisations-Punkte, aber auch Verzweigungen auf. Rechenzeiten sind meist nicht konstant, und Aufrufzyklen teilweise physikalisch an den Sensor oder den Aktuator gebunden, so dass es keine einheitliche Verarbeitungsfrequenz geben kann.

Die Parameter für die Entwicklung eines solchen Systems sind vielfältig und haben darüber hinaus auch keine statische Verteilung über der Zeit. Daher ist eine Unterstützung durch geeignete Werkzeuge unumgänglich, um die Komplexität und Variabilität während der System- und Softwareentwicklung zu beherrschen und zu verstehen. Grade die dynamischen Aspekte lassen sich überhaupt erst durch eine geeignete Visualisierung untersuchen und mit Entwicklungsteam und dem Kunden klären.

Die Anforderungen an das Werkzeug werden durch die vielfältigen Anwendungsfälle bestimmt. Im Einzelnen sind es:

- Visualisierung statischer Aspekte, wie der logischen Datenpfade
- Visualisierung dynamischer Aspekte, wie z.B. Synchronisierung unterschiedlicher Frequenzgänge
- Untersuchung von zeitlichen Latenzen der Datenpfade (Wirkketten) sowie deren Schwankungen über die Zeit

- Untersuchungen zu Ressourcenauslastungen wie Speicherbandbreiten oder Rechnerauslastung (statisch und über die Zeit)
- Erkennen von Designfehlern früh in der Entwicklungsphase
- Optimierungen von Datenpfaden im Kontext des Gesamtsystems
- Visualisierungen zu Schulungszwecken; zum Verdeutlichen von Problemstellungen und als Diskussionsgrundlage
- Werkzeuggestützte Analysen, wie z.B. Überprüfung zeitlicher Anforderungen sowie mögliche Optimierungen
- Ableitung von Testfällen; Generierung von Testfällen
- Generierung von Reports und Sichten
- Generierung von Simulationen
- Generierung des Steuergeräte-Codes
- Austausch von Design-Modellen (mit Entwicklern, Kunden, Tools, ...)

Um Inkonsistenzen in den Anwendungsfällen zu verhindern, ist ein Single-Source-Ansatz wünschenswert, wie es durch das AMALTHEA Systemmodell möglich ist. Vor allem für die Anforderungen (Requirements), die Generierung von Code sowie die Testfälle wird eine durchgängige Nachweisbarkeit (Traceability) benötigt.

AMALTHEA

AMALTHEA [1] unterstützt als offenes, standardisiertes Format zur System- und SW-Beschreibung die genannten Anwendungsfälle (Abbildung 1). Es bringt ein Metamodell mit, dessen Instanzen in XML serialisiert werden können. Darüber hinaus stellt es über die Eclipse-Plattform eine mächtige Entwicklungsumgebung bereit, um auf Instanzen der XML-Datenmodelle arbeiten zu können. Damit lassen sich über Java- und Xtend-Programmierung beliebige Modell-Transformationen realisieren.

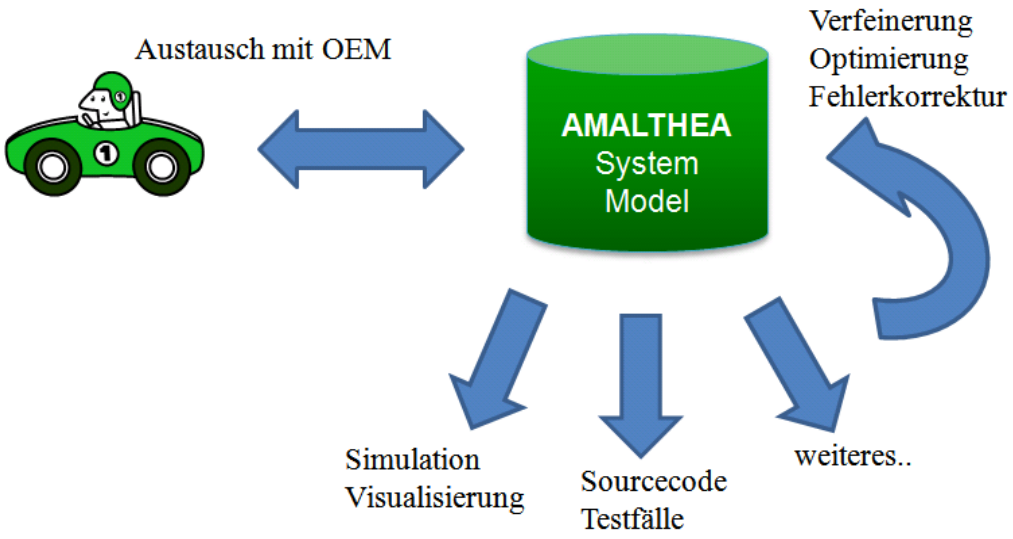

Abbildung 1: AMALTHEA und seine Anwendungen

Am Beispiel einer vereinfachten Architektur eines Kamerasystems werden im Folgenden die einzelnen Aspekte dargestellt und praktisch erläutert.

Anwendungsbeispiel

Die System-Architektur umfasst einen Imager-Chip zur Bilderfassung und einen integrierten Baustein mit mehreren Verarbeitungseinheiten. Diese sind spezialisierte Rechner für die Bildvorverarbeitung (Preprocessing, Feature-Berechnung sowie Klassifikation), und frei programmierbare Kerne. Jeder *osek-scheduler* ist genau einer Verarbeitungseinheit zugewiesen, wie beispielsweise *Sch_Osek_2* dem *Core_2* auf dem Mikrocontroller *SOC*.

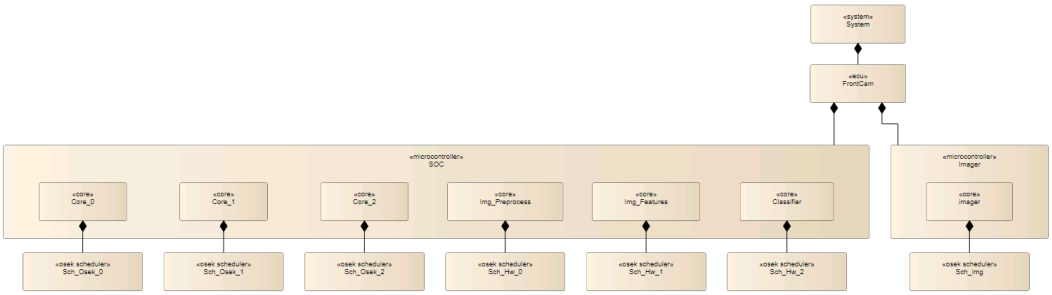

Abbildung 2: Systemübersicht des Anwendungsbeispiels

Modellierung von Tasks

Auf die Recheneinheiten lassen sich jetzt die einzelnen logischen Verarbeitungsschritte zuordnen, wie in Abbildung 3 dargestellt. Der Scheduler *Sch_Osek_2* ist hierbei mit der Task *task_2* verknüpft. Dieser Task wird periodisch getriggert durch einen 40ms Zeittrigger. Im Task wird das Runnable *runn_merge* aufgerufen.

In der gleichen Art und Weise lassen sich ISR und weitere Tasks modellieren. Alle Tasks eines Schedulers konkurrieren um Rechenzeit auf der gleichen

Verarbeitungseinheit, während die Tasks verschiedener Verarbeitungseinheiten parallel abgearbeitet werden können.

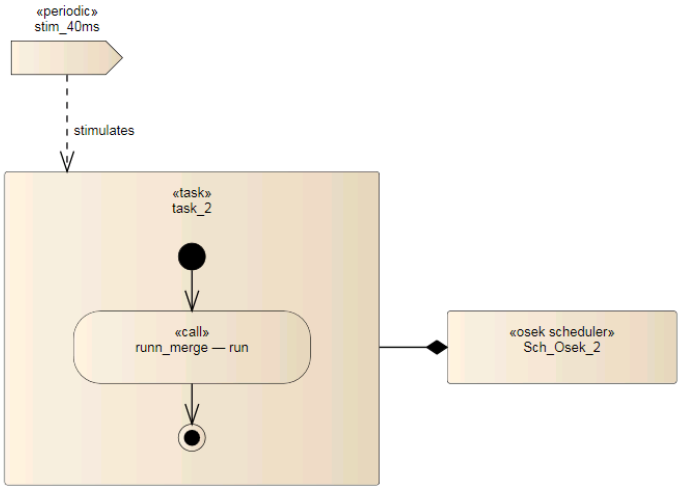

Abbildung 3: Zuordnung der logischen Verarbeitungsschritte

Datenfluss und funktionales Verhalten

Darüber hinaus ist es möglich, die logischen Datenbeziehungen zwischen Runnables zu modellieren. In Abbildung 4 erhält das Runnable *runn_merge* als Input Objekte vom Runnable *runn_algo* und Klassifikationsergebnisse vom Runnable *runn_hwclass* und liefert an die Funktionsberechnung eine fusionierte Szenenbeschreibung als Output für Runnable *runn_func*.

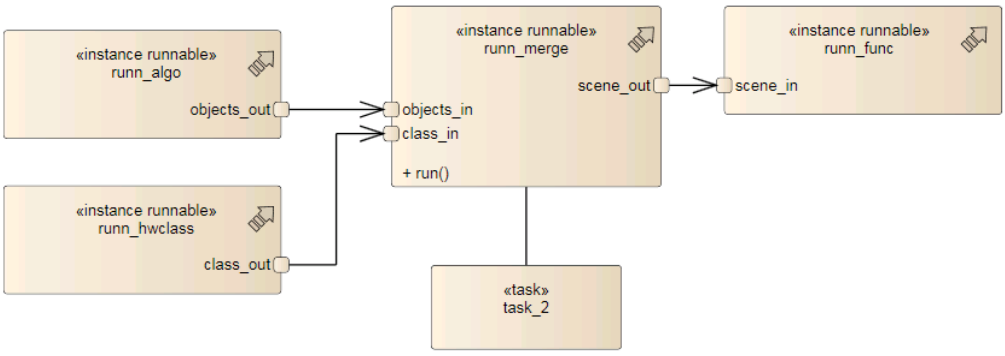

Abbildung 4: logische Datenbeziehungen

Ein Überblick über die logischen Datenbeziehungen im modellierten System lässt sich mithilfe des AMALTHEA-Frameworks als UML-Diagramm generieren (Abbildung 5).

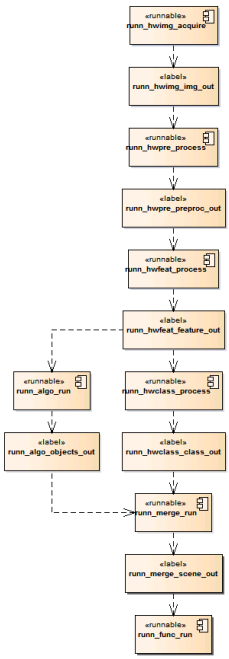

Abbildung 5: logischer Datenfluss

In der Regel sind für einen logischen Datenfluss (siehe Abbildung 5) zeitliche Anforderungen definiert, die in der Implementierung eingehalten werden müssen. Typische Echtzeit-Anforderungen sind beispielsweise maximale und minimal Latenzzeiten, Datenverlust oder Datensynchronisation, wie in [3] beschrieben.

Eine reine statische Analyse des logischen Datenflusses durch ein Summieren der einzelnen Latenzzeiten der Blöcke und Kanten ist nicht ausreichend, da weder Effekte durch Task-Scheduling oder Unterbrechungen von ISRs berücksichtig werden, noch durch Inter-Core-Kommunikation oder im Allg. durch Kommunikation zwischen den Verarbeitungseinheiten. Hierfür ist eine dynamische Betrachtung des Systemverhaltens basierend auf einem Timing-Modell notwendig.

Analyseergebnisse

In Abbildung 1 wurde als wichtige Anwendung von AMALTHEA die Simulation sowie die Visualisierung des dynamischen Systemverhaltens aufgezeigt. Das hierfür benötigte Timing-Modell wird automatisiert aus dem AMALTHEA Systemmodell generiert.

Das so erhaltene Timing-Modell wurde in die INCHRON Tool-Suite [2] geladen und mit dem Echtzeitsimulator chronSIM untersucht. Dieser Simulator zeigt das Echtzeitverhalten des untersuchten Systems aus verschiedenen Perspektiven. Neben Single-Core- und Multi-Core-Systemen können auch verteilte Systeme mit mehreren Prozessoren untersucht werden, die über simulierte CAN- und FlexRay-Busse, per Ethernet oder über andere Bussysteme kommunizieren.

Abbildung 6 zeigt den Datenfluss im untersuchten System. Dargestellt sind die Tasks auf den verschiedenen Recheneinheiten, die an den farbigen Balken links erkennbar sind. Jede Zeile zeigt den aktuellen Betriebssystemzustand eines Tasks.

Die geschwungenen Verbindungen zeigen den logischen Datenfluss zwischen den Runnables, die in den Tasks ausgeführt werden.

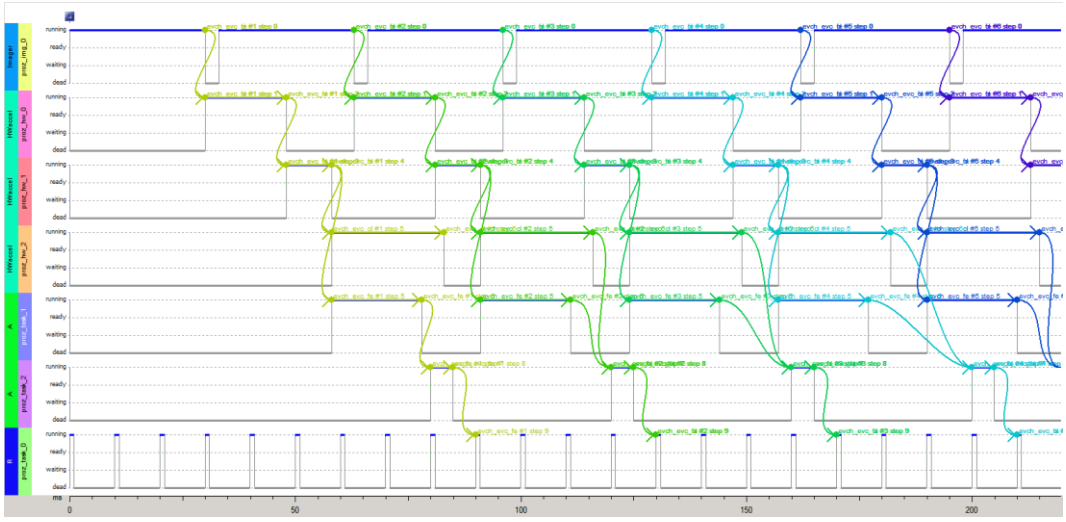

Abbildung 6: Zeitlicher Verlauf des Datenflusses

Aus Abbildung 6 ist ersichtlich, dass der zeitliche Verlauf des Datenflusses von der geradlinigen Darstellung in Abbildung 5 deutlich abweicht: Der überlappende Verlauf entsteht dadurch, dass zu einem Zeitpunkt mehr als ein Satz von Bilddaten im System bearbeitet wird.

Im Timing-Modell sind die logischen Datenflüssen als sogenannte Event-Chains modelliert. Diese werden im Modell wie Daten behandelt und zwischen den einzelnen Akteuren (Runnables, Tasks, aber auch Busnachrichten) weitergereicht.

Auf der Basis der Event-Chains können die Latenzen von Verarbeitungsketten analysiert werden (Abbildung 7). Dazu definiert man eine Echtzeit-Anforderung vom ersten bis zum letzten Schritt der Event-Chain und stellt es als Histogramm dar. Im Beispiel ist zu erkennen, dass ein Großteil der Messwerte in einem abgegrenzten Bereich schwankt, während fast 18% deutlich kürzer sind.

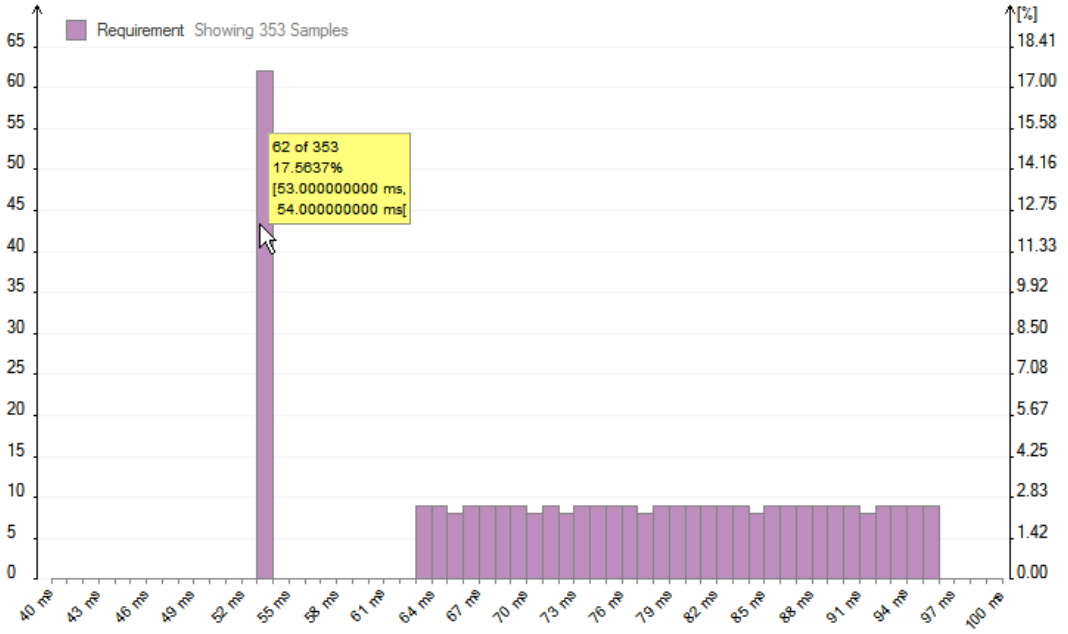

Abbildung 7: Latenzhistogramm einer Verarbeitungskette

So lässt sich schnell erkennen, wo das entworfene Design Fehler oder Engstellen aufweist (siehe Wirkkettendiagram in Abbildung 8) und etwa Verarbeitungsketten nicht bis zum Ende abgearbeitet werden, und wie sich solche Fehler dergestalt auswirken, das im Folgenden die Datenkonsistenz nicht mehr gewährleistet ist.

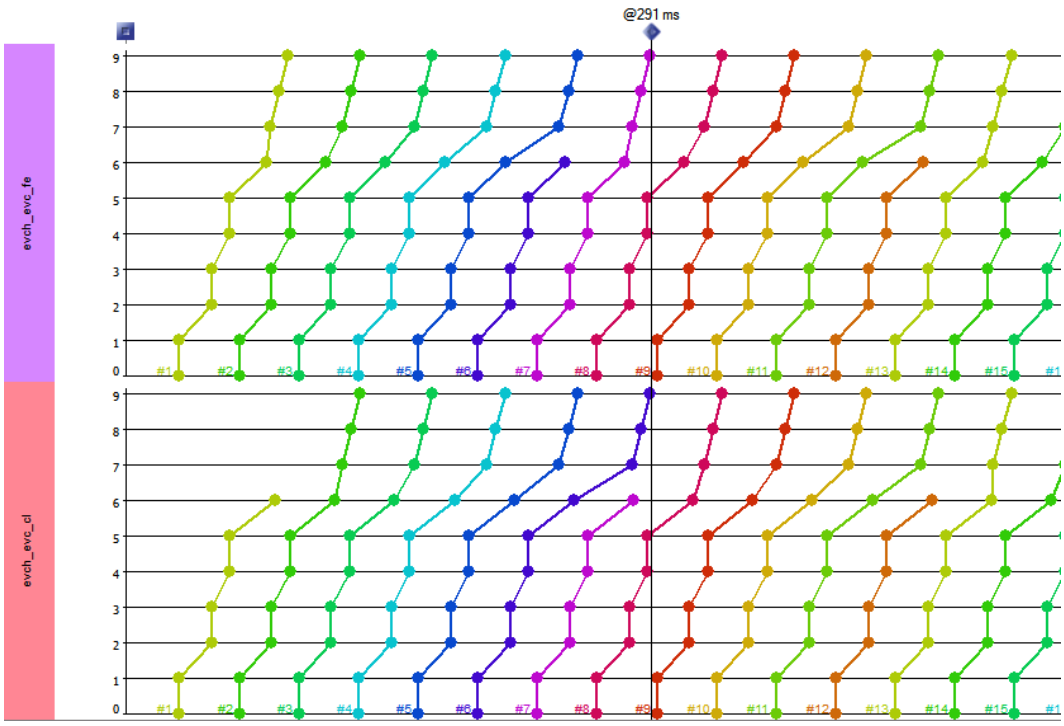

Abbildung 8: Visualisierung von Verarbeitungsketten sowie Designfehlern

Nun liefern die Untersuchungen und Auswertungen Anhaltspunkte, warum es zu einer Verletzung oder einem Engpass gekommen ist. Dazu definiert der Anwender Echtzeit-Anforderungen, die automatisiert ausgewertet werden und dem Anwender die mühevolle manuelle Suche im Wirkkettendiagram ersparen. Abbildung 9 zeigt die Datenfluss-Anforderung *Wirkketten-Abriss*. In 17,6% der Fälle läuft der Datenfluss nicht bis zum Ende. Der Anteil der Fehler deckt sich mit der Verteilung der Latenzzeiten aus Abbildung 7.

Abbildung 9: Auswertung der Datenfluss-Abrisse

Das Latenzhistogramm des Datenflusses aus Abbildung 7 hilft auch bei der Fehleranalyse weiter: Im Detail erkannt man, das die 17,6% abgerissenen Wirkketten einen Häufungspunkt bilden, und die erfolgreichen Wirkketten gleichmäßig über einen Bereich von 33 ms streuen.

Es werden also von den alle 33ms angelieferten Daten im Mittel nur ca. 82,4% verwertet. Berechnet man die mittlere Auswerteperiode als 33 / 0.824 zu 40,05ms, so findet man die Abtastrate von 40ms des Runnables *runn_merge* aus Abbildung 3, das mit einer 40ms Periode modelliert ist.

Damit ist der Grund der Datenabrisse identifiziert: Es handelt sich um eine Unterabtastung der Eingangssignale im "merge"-Schritt. Eine Möglichkeit, dem Datenabriss entgegenzuwirken, ist eine Anpassung der Abtastrate, indem z.B. der Zeittrigger für die Abtastung von 40ms auf 20ms gesetzt wird. Die automatisierte Auswertung der Datenfluss-Anforderung zeigt sofort den Effekt (siehe Abbildung 10): Alle Datenflüsse laufen bis zum Ende, es treten keine Daten-Inkonsistenzen mehr auf.

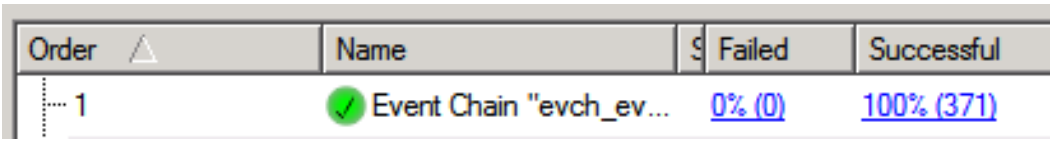

Abbildung 10: Datenfluss-Auswertung nach Erhöhung des Zeittriggers

Die dynamischen Auswirkungen der Designänderungen werden in unterschiedlichen Diagrammen visualisiert und können somit verstanden werden. Abbildung 11 zeigt die Auswirkung auf die Wirkkettenverläufe. Datenabrisse treten jetzt nicht mehr auf.

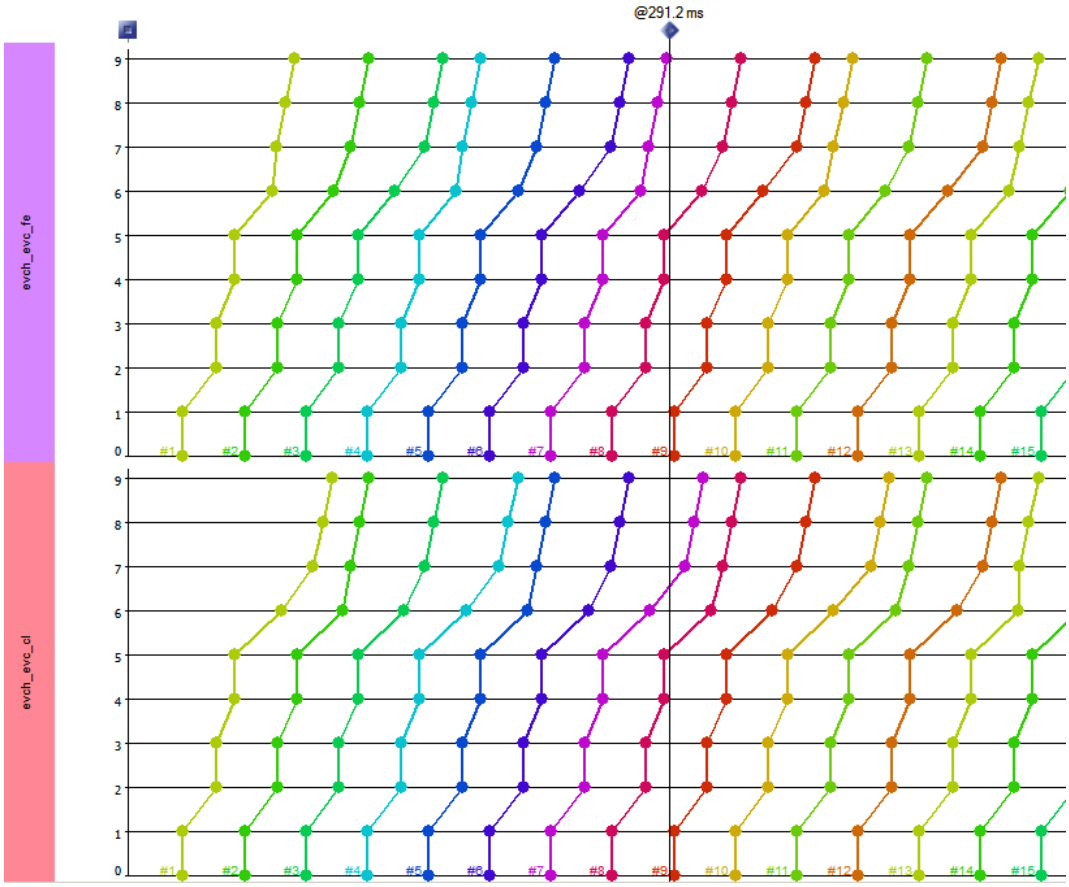

Abbildung 11: Visualisierung der Verarbeitungsketten nach Design-Korrektur

Da die Ergebnisse der Simulations-Analysen und die Designänderungen im zentralen AMALTHEA-Modell vorgenommen werden, wird eine konsistente Durchgängigkeit vom modifizierten Systemdesign, über weitere Simulation und Visualisierungen, Kunden-Reports bis hin zum Steuergeräte-Code und Test gewährleistet.

Zusammenfassung

In diesem Artikel ist ein Workflow beschrieben, um ausgehend vom AMALTHEA Systemmodell als Single-Source-Quelle wichtige Prozessschritte durchzuführen. Die Modellierung von logischen Datenflüssen (Event-Chains) sowie die Überprüfung der Echtzeit-Anforderungen sind hierbei zentrale Bausteine. Ein wesentlicher Vorteil dieses systematischen Vorgehens besteht darin, dass sich zeitliche Fehler bereits frühzeitig während der Designphase aufdecken lassen, die ansonsten erst spät im Entwicklungsprozess durch Testen und Debuggen auf der Hardware gefunden werden. Durch die Simulation wurden zeitliche Fehler bis zu 12 Monate früher entdeckt, die Ursachen durch eine geeignete Visualisierung verstanden und effizient behoben.

Literaturverzeichnis

[1] AMALTHEA: An Open Platform Project for Embedded Multicore Systems
siehe http://amalthea-project.org/

[2] INCHRON Tool-Suite: http://www.inchron.com/tool-suite/tool-suite.html

[3] T. Kramer, R. Münzenberger: Modeling and real-time analysis of complex causal loops in driver assistance systems, 3. Autotest, FKFS, October 2010.

Autoren:

Thomas Jäger ist als Fachreferent für Softwarearchitektur bei der Robert Bosch GmbH im Bereich Fahrerassistenz-Systeme tätig. Hierbei ist ein Schwerpunkt insbesondere die modellgetriebene Entwicklung der dynamischen Architektur für Multi-Core-Embedded-Systeme in Serienprojekten.

Kontakt: thomas.jaeger@de.bosch.com

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 reichhaltiges Wissen über Entwicklungs Prozesse auf die Embbeded Bereiche übertragen.

Kontakt: ingo.houben@inchron.com

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.

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