Multicore-Hardware-Tracing in der Praxis
Eine industrielle Fallstudie

Felix Martin, Maximilian Hempe und Michael Deubzer,
Timing-Architects Embedded Systems GmbH

In der Automobilbranche ist ein stetig wachsender Bedarf an immer komplexeren eingebetteten Systemen getrieben von innovativen Funktionen gegeben. Auch in Zukunft wird die Anzahl neuer Funktionen, die vollständig durch Software umgesetzt werden steigen, insbesondere für die Themen ADAS, Connected Cars, automatisches Fahren und Mobility Services. Dieser Bedarf wird durch performantere eingebettete Systeme gedeckt, welche Multi- und Many-Core-Controller enthalten und zunehmend auch in den klassischen Fahrzeugdomänen eingesetzt werden. Diese Fallstudie ist in der klassischen Domäne Lenkung angesiedelt, jedoch lässt sie sich auch auf andere klassische Domänen übertragen. Sicherheitskritische Funktionen in diesen Bereichen unterliegen hohen Echtzeitanforderungen, um Menschenleben nicht durch Fehlfunktion zu gefährden. Aus diesem Grund ist ein Nachweis der Performance und des Timings der Funktionen notwendig und durch ISO 26262 gefordert. Mit der steigenden Komplexität der Funktionen und der Systeme wächst der Bedarf an Automatisierung und Abstraktion, der sich in neuen und erweiterten Tools widerspiegelt. Als Nachweismethoden des Zeitverhaltens spielt neben der Simulation und statischen Analysen das Tracing eine wichtige Rolle. Durch Tracing kann das dynamische Verhalten von zeitkritischen Applikationen detailliert aufgezeichnet und validiert werden. In dieser Arbeit werden Tracing-Methoden betrachtet und miteinander verglichen, mit dem Ziel, eine optimale Lösung mit dem Anspruch, sehr hoher Messtiefe, Messbreite und Messlänge bei minimaler Messabweichung zur Verfügung zu stellen. Bestehende Ansätze werden hierzu kombiniert und eine Auswahl empfehlenswerter Werkzeuge getroffen.

Einleitung

In Echtzeitsystemen hängt korrektes Systemverhalten nicht nur von der Richtigkeit der berechneten Daten und den daraus abgeleiteten Entscheidungen ab, sondern auch davon, wann diese Daten vorliegen und die entsprechenden Entscheidungen getroffen werden. Die Zeit, die eine Funktion einhalten muss um korrekt zu funktionieren, wird in informalen Zeitanforderungen definiert. Insbesondere in sicherheitskritischen Domänen, wie der Automobilindustrie müssen diese Anforderungen eingehalten werden, um das Gefährdungspotential für Mensch und Umwelt zu minimieren [1]. Die Verwendung von Multi-Core-Prozessoren erschwert die Thematik, aus dem Grund, dass das Zeitverhalten durch den simultanen Zugriff verschiedener Softwarekomponenten auf Speicher und Peripherie negativ beeinflusst wird [2].

Echtzeitanforderungen auf den verschiedenen Ebenen in sicherheitskritischen Systemen werden während des gesamten Entwicklungsprozesses für jeden Release geprüft [3] [4] . Mittels Tracing können Zeitpunkte der Ausführung von Funktionen und Zugriffe auf Daten aufgezeichnet werden, um das korrekte Zeitverhalten der Funktionen auswerten

zu können. In dieser Arbeit wird gezeigt, welche Schritte, unter Abwägung der Kosten und Nutzen empfehlenswert sind, um einen solchen Prozess in der Praxis aufzusetzen. Nach kurzer Einführung der Grundlagen zur Echtzeitbetrachtung und Tracing, sowie einer Gegenüberstellung der heute bestehenden Tracing-Methoden, werden im Folgenden die konkreten Schritte diskutiert, wie ein empfehlenswerter Trace-Prozess zur Prüfung von zeitkritischen Funktionen gestaltet werden sollte. In diese Bewertung und Empfehlung fließt gesammelte Erfahrung aus einem realen Projekt eines Lenksystems ein.

Grundlagen Echzeitanalyse und Tracing

Die meist informalen Zeitanforderungen in bestehenden Projekten müssen zur weiteren automatisierten Verarbeitung in eine formale Beschreibung übersetzt werden. Hierzu ist ein Datenmodell wünschenswert, welches sich an aktuelle Standards in der Automobilindustrie anlehnt. Eine Möglichkeit dafür sind die AUTOSAR Timing Extensions [3], deren Ansatz im Folgenden betrachtet wird. Für die Wahl der Tracemethodik, mit möglichst hoher Messtiefe, Messbreite und Messlänge bei minimalem Messfehler, werden zunächst Trace-Techniken betrachtet und gegenübergestellt. Als Format für die Aufzeichnung der Traces ist in diesem Fall das im AMALTHEA Standard spezifizierte offene BTF-Format gewählt worden, da es eine vollständige Unterstützung für Multi-Core und Multiprozessor Traces bietet [4]. Für die Auswertung von Traces gibt es verschiedene Lösungen. Der letzte Abschnitt dieser Sektion erläutert die grundsätzliche Funktionsweise solcher Werkzeuge. In dieser Methodik hat sich der TA Inspector von Timing-Architects als beste Wahl erwiesen, da sowohl Profiling, sowie automatische Auswertung und Wiederverwendung von formalisierten Zeitanforderungen möglich sind [5].

Zeitanforderungen

Zeitanforderungen sind auf verschiedenen Abstraktionsebenen, wie z.B. der System-, ECU-, und Softwareebene definiert. Auf allen Ebenen ist die Auswertung des Echtzeitverhaltens von Interesse. Auf der Softwareebene befinden sich Speicherzugriffe und ausgeführte Instruktionen. Erstere können Lese- oder Schreibzugriffe auf eine Variable oder auf ein Hardwareregister sein. Letztere bezeichnen die Ausführung eines CPU Befehls. Im Gegensatz dazu, befinden sich auf der ECU-Ebene Teile der Applikationssoftware wie z.B. Softwarekomponenten und Runnables, sowie Teile der RTE und Middleware wie unteranderem Tasks und Kommunikationsinterfaces.

Alle Arten von Zustandsänderungen im System werden als Event bezeichnet und müssen für die Echtzeitanalyse mit einem Zeitstempel annotiert sein. Zeitanforderungen können für die Perioden zwischen zwei oder mehreren unterschiedliche Events definiert werden. Ein Beispiel aus der gegebenen Domäne ist das Lesen und Verarbeiten des Signals *Drehmoment* des Drehmomentsensors und Berechnung des Sollmoments für den Motor einer elektromechanischen Lenkung. Hier sind beobachtbare Events das Lesen und Schreiben der Signale *Sensordrehmoment* und *Motorsollmoment* und die Ausführung, bzw. die Statuswechsel der Funktionen *Verarbeitung* und *Sollmomentberechnung*. Diese vier Events müssen in einer kausalen Kette innerhalb einer maximalen Zeit auftreten oder das System muss rechtzeitig innerhalb einer

sicherheitskritischen Fehlertoleranzzeit reagieren und damit die Gefährdung von Menschenleben verhindern.

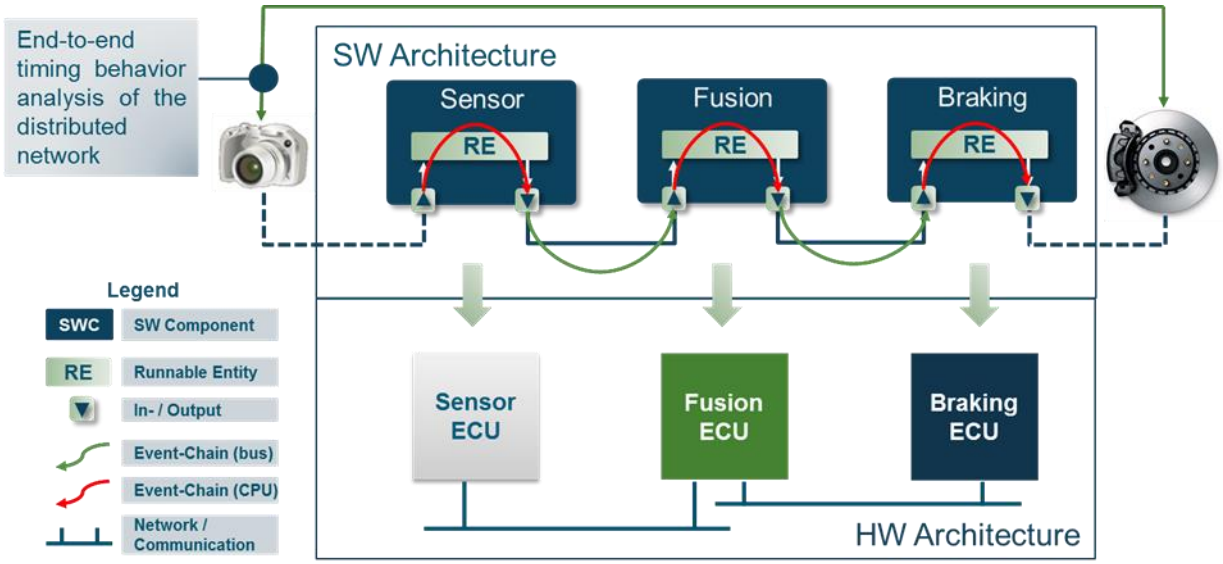

Abbildung 1: Definition einer Wirkkette über verschiedene Steuergeräte. Die Events sind in diesem Beispiel auf die Kommunikationsschnittstellen der beteiligten Softwarekomponenten definiert. Zeitanforderungen lassen sich für die Wirkkette als Ganzes oder für einzelne Segmente definieren.

Tracing

Um die definierten Zeitanforderungen auszuwerten ist es notwendig, einen Trace aufzuzeichnen. Ein Trace ist eine Liste von Events mit dem jeweiligen Zeitstempel zu dem das Event aufgetreten ist. Um die Auswertung aller Anforderungen zu ermöglichen, müssen alle Events, die mit Anforderungen in Verbindung stehen, im Trace enthalten sein. Für die Aufzeichnung von Traces gibt es nach Ferrari drei unterschiedliche Möglichkeiten: Softwarebasiertes, hybrides und hardwarebasiertes Tracing [6].

Softwarebasiertes Tracing erlaubt das Aufzeichnen von Traces ohne dedizierter Hardware. Stattdessen werden für die Analyse relevante Events instrumentiert. Für das Hinzufügen der Instrumentierung gibt es verschiedene Strategien. Eine Methode ist das Instrumentieren im Source Code an den Stellen, an denen die relevanten Events stattfinden. Die zweite Möglichkeit ist es Hooks zu verwenden, die vom Betriebssystem bereitgestellt werden. Falls der Source Code nicht verfügbar ist kann die Instrumentierung auch direkt zum Binary der Software hinzugefügt werden. Beim softwarebasiertem Tracing kann der Trace auf zwei Arten für die Auswertung bereitgestellt werden. Die erste Option ist die Daten zur Laufzeit im Speicher auf dem Steuergerät abzulegen und nachträglich auszulesen. Alternativ dazu können die Daten auch in Echtzeit über eine Schnittstelle, zum Beispiel CAN, von der ECU transportiert werden [7].

Im Gegensatz zum softwarebasiertem Tracing ist beim hardwarebasiertem Tracing a priori keine Instrumentierung notwendig. Stattdessen werden die relevanten Events über einen dedizierten Hardwarebaustein (Emulation Device) und ein entsprechendes Interface direkt ausgelesen. Mit diesem Ansatz ist es möglich, sowohl Speicherzugriffe via Data-Tracing, als auch ausgeführte Instruktionen via Program-Flow-Tracing aufzuzeichnen [6]. Dazu hat der dedizierte Hardwarebaustein Zugriff auf die Kerne und die Speicherbusse des Prozessors. Die detektierten Events werden mit Zeitstempeln versehen und über das Trace-Interface gesendet.

Hybrides Tracing bezeichnet Ansätze die software- und hardwarebasiertes Tracing kombinieren. Beispielsweise können Betriebssystem-Hooks dazu verwendet werden Events in den Speicher zu schreiben, die für die Auswertung relevant sind. Diese Nachrichten können im darauffolgenden über existierende Datenschnittstellen wie CAN ausgelesen werden.

Grundsätzlich ist softwarebasiertes Tracing einfacher aufzusetzen, da keine dedizierte Hardware notwendig ist. Im Gegensatz dazu, erfordert hardwarebasiertes Tracing ein Emulation Device und spezielle Trace-Hardware. Der Vorteil von hardwarebasiertem Tracing ist dafür eine signifikant größere Bandbreite, die es erlaubt längere Traces mit einer größeren Anzahl an Objekten und mehr Messtiefe aufzuzeichnen. Gerade die Anzahl der Objekte und die Trace-Länge sind bei softwarebasierten oder hybriden Ansätzen ohne dediziertem Trace-Interface aufgrund des verfügbaren Speichers und der limitierten Bandbreiten klassischer Datenschnittstellen eingeschränkt.

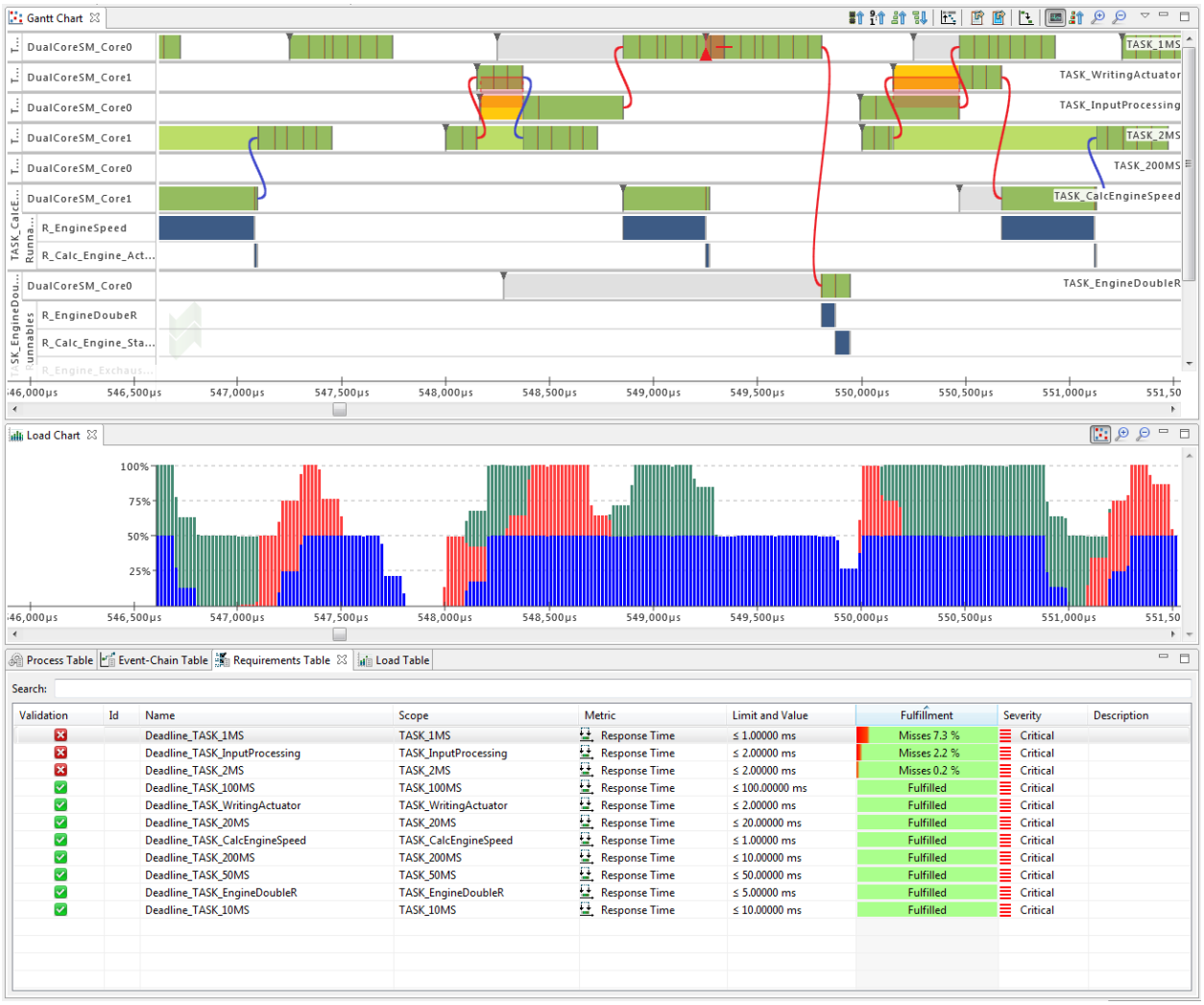

Abbildung 2: Trace-Auswertung mit Visualisierung von Blockierungs-Effekten (Oben), CPU Auslastung (Mitte) und Anforderungs-Überprüfung (Unten).

Werkzeuggestützte Traceauswertung

Unabhängig von der gewählten Trace-Technik ist das Ergebnis ein Trace, der die Auswertung der Zeitanforderungen ermöglicht. Die Auswertung erfolgt in zwei Schritten: Im ersten Schritt werden alle Metriken berechnet, die für die Validierung der Anforderungen relevant sind. Im zweiten Schritt werden die berechneten Metriken mit den Anforderungen verglichen. Das Ergebnis ist eine Liste von Anforderungen zusammen mit der Information in wie vielen Fällen diese jeweils verletzt wurde. Die unterste Zeile in Abbildung 2 zeigt die graphische Darstellung der Ergebnisse einer Anforderungsanalyse im TA Inspector. Nach dem Ampelprinzip kann der Nutzer erkennen welche Anforderungen verletzt wurden.

Tracebasierte Echtzeitanalyse in der Praxis

In dieser Sektion wird genauer beschrieben welche Schritte notwendig sind, um die Zeitanforderungen und das komplette dynamische Verhalten für ein Steuergerät in der

Automobilindustrie zu überwachen. Das Referenzsystem ist ein Lenksystem eines großen deutschen OEMs. Der erste Abschnitt geht auf die Auswahl der richtigen Trace-Technik ein und welche Fragestellungen sich beim Umsetzen von hardwarebasiertem Tracing ergeben haben. Anschließend wird beschrieben, wie der Trace in das BTF-Format konvertiert werden kann. Der letzte Abschnitt zeigt, in welcher Form die erzeugten Daten zur weiteren Analyse aufbereitet werden können.

Tracing

Bevor ein Prozess zum Tracing aufgesetzt werden kann, sollte definiert werden, welche Anforderungen an die generierten Traces gestellt werden. Für das Lenkungssteuergerät sollen neben Tasks und ISRs, auch Runnables und Zugriffe auf bestimmte Signale aufgezeichnet werden. Zusätzlich ist die Aufzeichnung von Traces über mehrere Sekunden notwendig, um verschiedene Bewegungsmuster der Lenkung abzudecken, zum Beispiel die Bewegung von einem Lenkanschlag zum anderen. Beide Anforderungen kombiniert erfordern zwingend einen hardwarebasierten Traceansatz. Das Zwischenspeichern von Tracedaten für nahezu 300 Runnables ist eine Aufgabe, die ein reiner Software-Trace nicht leisten kann.

Für hardwarebasiertes Tracing ist ein dedizierter Tracebaustein auf dem Prozessor zwingend erforderlich. Der für das Steuergerät eingesetzte Prozessor ist ein NXP Leopard (MPC5643L). Ein Blick in das Datenblatt zeigt, dass der Prozessor Data- und Program-Flow-Tracing entsprechend des NEXUS Standards zur Verfügung stellt [8]. Ersteres kann in dem Datenblatt unter data trace messaging (DTM), letzteres unter branch trace messaging (BTM) gefunden werden [9]. Damit ist die erste Voraussetzung erfüllt.

Nun müssen die aufgezeichneten Daten vom Prozessor transportiert werden. In Abhängigkeit vom verwendeten Package hat der Leopard entweder vier oder 12 message data out (mdo) pins über welche die Events vom Chip gesendet werden können. Die hier verwendete Version des Gehäuses bietet vier Ausgangspins, was die maximale Tracebandbreite einschränkt. Ein größeres Problem ist, dass die NEXUS Pins nicht auf eine Buchse des Steuergeräts geroutet sind. Dadurch gibt es keine Möglichkeit die Tracedaten abzugreifen.

Es gibt unterschiedliche Möglichkeiten dieses Problem zu lösen, zwei davon werden im Folgenden diskutiert. Der Lieferant des Boards kann dazu beauftragt werden das Platinenlayout zu ändern, sodass die vier Pins zu einer Buchse führen. Das ist aber keine Lösung, die kurzfristig umsetzbar ist. Außerdem wäre das Interface auf dem Board aufgrund der Verbauung im Gehäuse nicht zugänglich.

Die zweite Lösung ist die Verwendung eines Emulation Boards [10]. Dabei emuliert ein Leopard Prozessor auf dem Board mit BGA257 Package (also dem größeren Package, das ein 12 Pin Nexus Interface hat) den eigentlichen Prozessor des Steuergerätes. Damit können zwei Probleme gelöst werden: Erstens wird die Limitierung des vier Pin Interfaces umgangen. Zweitens ist das Nexus Interface über das Emulation Board nach außen geführt. Somit sind keine weiteren Änderungen am Steuergerät selbst notwendig.

Um das Emulation Board mit dem Steuergerät zu verbinden muss der existierende Prozessor mit QFP100 Package abgelötet werden. Anschließend wird ein Lötsockel an der gleichen Stelle angebracht. Dieser Sockel kann nun dazu verwendet werden das Emulation Board an die ECU anzustecken. Um das zu ermöglichen, wurde ein Loch in das Gehäuse der ECU geschnitten. Dieser Aufbau hat das gleiche Verhalten wie die ECU im ursprünglichen Zustand mit dem Vorteil, dass das z12 Pin Nexus Interface zum Tracing zur Verfügung steht.

Transformation

Mit dem Nexus Interface ist es nun möglich Traces aufzuzeichnen. Zur Echtzeitanalyse müssen diese in das BTF-Format transformiert werden. Die Objekte, die in diesem Projekt analysiert werden sollen sind Tasks, ISRs, Runnables und ausgewählte Variablenzugriffe. In OSEK Betriebssystemen können Tasks und ISRs über Data-Tracing aufgezeichnet werden. Variablenzugriffe werden ebenfalls über Data-Tracing abgebildet. Runnable-Events sind Funktionsaufrufe, welche mittels Program-Flow-Tracing registriert werden können.

Bei der Implementierung der Transformation auf diese Weise ergaben sich jedoch Schwierigkeiten. Erstens wird das µC/OS Betriebssystem von Micrium verwendet, welches nicht OSEK konform ist [11]. Entsprechend ist der übliche Task-Trace Ansatz über das OSEK Task State Array nicht anwendbar. Ein weiteres Problem ist, dass die ISRs der OSEK Kategorie Eins entsprechen. Das Betriebssystem bekommt also nicht mit, wann ISRs ausgeführt werden und kann diese Information folglich nicht in tracebaren Datenstrukturen ablegen. Um mit dieser Einschränkung umzugehen, kann ein Program-Flow-Trace auch für die ISRs verwendet werden. Dies ist jedoch aufgrund der dritten Limitierung nicht möglich. Zur Zeit der Durchführung dieses Projekts unterstützte iSYSTEM noch kein voll ständiges Multi-core Profiling. Damit ist die Verwendung des Program-Flow-Traces für mehrere Kerne nicht möglich. Da die simultane Analyse beider Kerne obligatorisch ist, kann das Program-Flow-Tracing nicht verwendet werden und damit können auch Runnable und ISR Events nicht direkt aufgezeichnet werden.

Letztendlich ist es daher notwendig, trotz des Emulation Adapter, einen hybriden Traceansatz zu verwenden. Dieser sieht vor Runnable und ISR Information via Instrumentierung aufzuzeichnen. Die Vorrausetzung dafür sind zum einen, dass der Source Code für die Instrumentierung zur Verfügung steht. Zum anderen darf der Einfluss der Instrumentierung die Laufzeiteigenschaften der Software nicht signifikant verfälschen. Der erste Punkt ist erfüllt und der zweite Punkt muss aufgrund fehlender Alternativen vorerst in Kauf genommen werden.

Die Instrumentierung selbst wird nach dem folgenden Schema durchgeführt: Zuerst wird für alle ISRs und Runnables die für die Analyse von Interesse sind eine ID vergeben. Dann wird der Source Code nach den Aufrufen der Runnables und den ISR Definitionen durchsucht. Wird der Aufruf eines relevanten Runnables gefunden, wird vor und nach dem Runnable die entsprechende ID in eine dedizierte Variable gelegt. Zusätzlich wird nach dem Aufruf ein zusätzliches Bit gesetzt, das dazu reserviert ist zu indizieren, ob ein Start oder eine Terminierung vorliegt. Ähnlich ist das Vorgehen für die ISRs, mit der

Ausnahme, dass die Instrumentierung im Kontext der ISR selbst hinzugefügt wird. Damit ist das Aufzeichnen von Runnable und ISR Events möglich, indem die dedizierte Variable per Data-Tracing beobachtet wird. Das Gleiche gilt für die Zugriffe auf bestimmte andere Variablen, die von Interesse sind. In diesem Fall ist das zum Beispiel die System-State-Variable, die den momentanen Zustand in dem sich das System befindet, anzeigt.

Zuletzt muss untersucht werden, wie die Taskzustände rekonstruiert werden können. Dazu ist ein genauerer Blick auf das Betriebssystem notwendig. Dabei fällt auf, dass ein Array von Taskkontextblöcken (*OSTCBTbl*) existiert. Jeder dieser Blöcke wird vom Betriebssystem dazu verwendet, verschiedene Informationen, wie Adresse des Taskstacks, Größe des Taskstacks, Priorität und auch Zustand der Task zu speichern. Außerdem gibt es eine Variable, die eine Referenz auf den Kontextblock der momentan laufenden Task enthält (*OSTCBCur*). In OSEK Betriebssystemen würde der Zustand im Taskkontextblock ausreichen um die Events für eine Task zu rekonstruieren. Im Falle von µC/OS gibt es aber keine separaten Zustände für *Ready* und *Running*. Stattdessen werden diese im Zustand *Runnable* zusammengefasst, ähnlich wie im Taskzustandsmodell von Linux. Daraus ergibt sich, dass es zusätzlich notwendig ist die Variable *OSTCBCur* zu tracen, um die Unterscheidung zwischen Ready und Running zu treffen.

Mit diesem Wissen ist es nun möglich den kompletten Trace nach BTF zu konvertieren. Dazu werden per Data-Tracing Schreibzugriffe auf die dedizierte Variable für die ISRs und Runnables, das komplette Taskkontextblockarray und die Variable OSTCBCur aufgezeichnet. Der exportierte Datentrace ist dann ein CSV, der folgende Felder enthält: Zeitstempel, Adresse der geschriebenen Variable, Wert der geschrieben wurde und Kern von dem der Zugriff erfolgt ist. Zusätzlich zum Trace selbst sind noch die Zuordnungen von IDs zu Runnables und ISRs, von Adressen zu Variablen Namen und von Taskzustandsvariablenadressen zu Tasknamen notwendig. Ersteres wird bei der Instrumentierung erstellt. Die anderen beiden Punkte können mit dem Debugger ausgelesen werden.

Auswertung

Der letzte Schritt ist die Auswertung des Traces im BTF-Format mit dem TA Inspector. Neben den bereits diskutierten Metriken, die sich auf die Zeit zwischen zwei oder mehreren Events beziehen, werden auch Metriken berechnet, welche nicht die Einheit Zeit haben, aber für die Analyse von Bedeutung sind. Beispiele dafür sind die Anzahl der Unterbrechungen einer Taskinstanz, die Last die von einer ISR auf einen bestimmten Kern verursacht wird und die Anzahl der multiplen Taskaktivierungen. Der TA Inspector ist in der Lage all diese Metriken zu berechnen.

Auf der anderen Seite sind nicht immer alle Metriken interessant, die während der Analyse des Traces berechnet werden. Aus diesem Grund unterstützt die TA Tool Suite die automatisierte Erstellung von Reports in verschiedenen Formaten. Die darin enthaltenen Metriken und Anforderungen können frei konfiguriert werden. Die Reporterstellung ist in verschiedenen Formaten wie HTML, LaTeX und XML möglich. Sowohl die Auswertung an sich, als auch die Reporterstellung ist via

Konsolenschnittstelle bedienbar. Somit lässt sich die Reporterstellung komplett automatisiert ausführen.

Vorteile Tracebasierter Echtzeitanalyse
Ist der Prozess zur tracebasierten Echtzeitanalyse implementiert, müssen die erzeugten Daten verwendet werden um die Robustheit der zeitkritischen Funktionen permanent zu überwachen und zu verbessern. Für das Lenksystem konnten die Evaluierungsergebnisse für die folgenden Use-Cases verwendet werden:

- Performance-Tests: Timing- und Schedulingnachweis auf Systemebene
- Resource Usage Tests: Robustheitstests zum Nachweis von Timing-Budgets auf Systemebene und Unit-Ebene

Die Lenkung im Fahrzeug ist ein sicherheitskritisches Bauteil mit der Einstufung ASIL D nach ISO 26262 [12]. Aus diesem Grund sind auf Systemebene nach der Norm das korrekte Echtzeitverhalten durch festgelegte Methodiken nachzuweisen, insbesondere für Safety-Umfänge des Systems. Auf Systemlevel ist es das Ziel, die korrekte funktionale Performance und die Fehlerfreiheit von Sicherheitsmechanismen festzustellen. Die Norm definiert hierzu im Abschnitt 8.4ff zum einen Performance Tests, welche zum Timing- und Schedulingnachweis auf Systemlevel anzuwenden sind und zum anderen Resource Usage Tests, um Speichergrößen und Laufzeit von Runnables auf festgelegte Budgets und damit die Robustheit und Verfügbarkeit des Systems sicherzustellen.

Aus diesem Grund ist ein wichtiges Geschäftsziel der Lenkungsentwicklung die vollständige Nachweisführung der Echtzeitfähigkeit: Der Nachweis, das Timing-Anforderungen, wie Wirkketten im System vom Sensor bis Aktor nach Spezifikation in der korrekten Zeit funktionieren und maximale Ausführungszeiten von Funktionen und maximale Datenalter von Signalen eingehalten werden. Beispielsweise muss sichergestellt werden, dass bei einer elektromechanischen Lenkung das Handmoment über einen Drehmomentsensor über die Vorverarbeitung der Basis-Software, der Berechnung von Kundenfunktionalität in Applikationen bis hin zur Momentstellung am Motor rechtzeitig innerhalb der sicherheitskritischen Anforderungen geschieht. Fehlertoleranzzeiten dienen dabei der rechtzeitigen Reaktion auf Signalfehler, inkorrekte Verarbeitung der Signale, fehlerhafte Applikation oder fehlerhaftem Momentstellen am Motor, welche durch Überwachungsfunktionen sichergestellt werden. Zum Nachweis der Einhaltung dieser Fehlertoleranzzeiten je nach Sicherheitsziel wird ebenfalls die hier gezeigte Methodik herangezogen.

Zusätzliche Budgetierung des Gesamtsystems bei verteilter SW-Entwicklung wird vorgenommen, um das Gesamtsystem zu Partitionieren, um so der Basis-Software, der Applikation und gegebenenfalls weiteren Parteien die notwendigen Ressourcen zuzuteilen. Weiterhin werden in SW-Feinspezifikationen detaillierte Ressourcenbudgets auf Unit-Ebene festgelegt. Diese Budgets zur Partitionierung von Integrationsanteilen und der Units werden geprüft durch Resource Usage Tests, welche auf Systemebene für

jeden Release anhand von Worst-Case-Timing-Szenarien – auch Stresstests genannt – durchgeführt werden.

Als zusätzliches Geschäftsziel ist die Performancebewertung und -optimierung zu nennen, bei der das System auf Ressourcenengpässe und Top-Verbraucher analysiert wird und mögliche Laufzeitoptimierungen erkannt werden. Hierzu ist die Auswertung von HW-Traces ein wichtiges Mittel, um im Gesamtsystemkontext bewerten zu können, welche Timing-Probleme vorliegen, beispielsweise lassen sich Worstcases durch Scheduling-Effekte erklären. Analysen, wie die Interference-Analyse unterstützen dabei, diese Effekte zu erkennen.

Für die Ziele des Timing-Nachweises und der Analyse werden System-, Bauteil-, und SW-Anforderungen an das Zeitverhalten in formale maschinenlesbare Timing-Requirements und Constraints übersetzt. Das Ziel in diesem Verarbeitungsschritt ist es, eine automatisierte Auswertung der Timing-Anforderungen auf den aufgezeichneten Traces durch Toolunterstützung durchführen zu können. Komfortabel hierzu ist die Funktion in der TA Tool Suite, die es dem Timing-Analysten ermöglicht, formalisierte Timing-Requirements/Constraints nicht nur tabellarisch, sondern auch graphisch zu definieren und sie in zukünftigen Tests einfach wiederzuverwenden. Eine Regressionsteststrategie ist damit möglich. In der hier vorliegenden Fallstudie wurden die in der Tabelle 1 genannten Timing-Requirements/-Constraints verwendet, um die im Projekt gegebenen Timing-Anforderungen zu formalisieren.

Zusätzlich zu diesen Timing-Requirements/-Constraints werden Metriken zur Analyse des dynamischen Verhaltens des Systems aus den aufgezeichneten BTF-Traces analysiert. Tabelle 2 zeigt die hier verwendeten Metriken.

Ebene Timing-Anforderung	Timing-Requirement/-Constraint
Gesamt-performance	Utilization Constraint -> Max. Load in %
Unit-Budgets	Upper Limit -> Netto Execution Time
Task-/ISR-Deadlines	Upper Limit -> Response Time Tasks/ISRs
Wirkketten	Min/Max-Intervall -> Delay Requirement -> Event-Chain-Requirement Events: Task/ISR/Runnable-State-Wechsel oder Signal-Zugriff (Lesen/Schreiben)

Tabelle 1: Verwendete Requirement-Typen zur Definition von Timing-Anforderungen für die Nachweisführung

Ebene	Metrik	Kurzname	Definition
System/ CPU	CPU-Load der Cores	CLC	CPU Load Cores; relative Last in % eines Kerns, Zeit in der ein Kern aktive Berechnungen durchführt.
Tasks/ISRs	CPU-Load	CLP	CPU Load Processes; relative Netto-Antwortzeit in % einer Task/eines ISRs, Zeit von Start bis Ende einer Task abzüglich der Unterbrechungen.
	Normierte	NRT	Normalized Response Time; normierte Netto-Antwortzeit einer

	Antwortzeit		Task/eines ISRs, Zeit von Start bis Ende einer Task abzüglich der Unterbrechungen.
	Ausführungszeit	NET	Netto Execution Time; Netto-Ausführungszeit einer Task/eines ISRs, Zeit in ms von Start bis Ende einer Task abzüglich der Unterbrechungen.
	Antwortzeit	RT	Response Time; Antwortzeit einer Task/eines ISRs, Zeit in ms von Aktivierung einer Task bis Ende inkl. Unterbrechungen.
	Aktivierungszeit	A2A	Activate-2-Activate Time; Distanz zwischen zwei Aktivierungen einer Task/eines ISRs
Runnables	Netto-Ausführungszeit	NET	Netto Execution Time; Netto-Ausführungszeit eines Runnables, Zeit in ms von Start bis Ende exklusive Unterbrechungen.
	Brutto-Ausführungszeit	GET	Gross Execution Time; Brutto-Ausführungszeit eines Runnables, Zeit in ms von Start bis Ende inklusive Unterbrechungen.

Tabelle 2: Verwendete Metriken für Performance-Tests

Der Timing-Nachweis wird iterativ für jeden Haupt- und Zwischen-Release einer Software durchgeführt [13]. Wie Abbildung 3 zeigt, sind zur Durchführung des Timing-Nachweises drei wesentliche Schritte notwendig:

1. Die Aktualisierung der Testspezifikation für den gegebenen Release wird durchgeführt: Geänderte und hinzugekommene Timing-Anforderungen werden formalisiert in Timing-Requirements und –Constraints festgehalten. Als Format zur Beschreibung der Timing-Requirements/-Constraints wurde hier AMALTHEA gewählt, als quasi Industriestandard mit dem größten Set an Ausdrucksmöglichkeiten für Anforderungen an das dynamische Verhalten einer Software [14].
2. Die Testdurchführung des HW-Trace über eine hybride Methode bestehend aus Funktions- und Daten-Trace, teilweise instrumentiert. Die Methode verwendet wie oben beschrieben ein Nexus-Interface und den Trace-Debugger iC5000 von iSYSTEM. Die erfassten Rohdaten werden ins BTF-Format übersetzt, welches AMALTHEA-konform ist.
3. Bei der Nachweisführung werden die in Schritt 2 aufgezeichneten BTF-Traces analysiert und evaluiert. Die in Schritt 1 aktualisierten Requirements/-Constraints werden automatisiert geprüft. Ein konfigurierbarer Report wird mit Ergebnis der evaluierten Anforderungen und gewünschten Profilingdaten (Timing-Metriken) ausgegeben. Der Report gibt direkten Hinweis auf Verletzung von Anforderungen zur Ableitung von Abstellmaßnahmen bspw. durch Design-Änderungen. Die Nachweis- und Reporterzeugung wird mit dem Werkzeug TA Inspector durchgeführt.

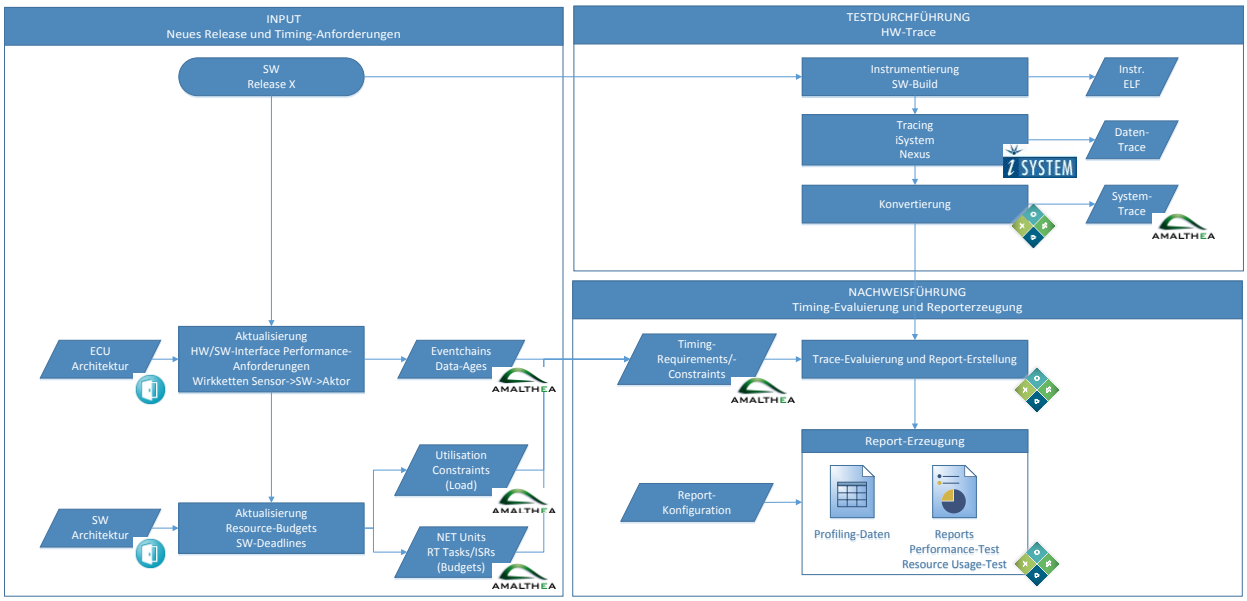

Abbildung 3: Überblick Ablauf Ressourcennachweis

Fazit und Ausblick

Diese Arbeit betrachtet wie ein kombinierter Ansatz aus software- und hardwarebasiertem Tracing zur Auswertung sicherheitskritischer Echtzeitanforderungen verwendet werden kann. Dieser Ansatz erlaubt die Aufzeichnung von Traces mit einer Breite, Tiefe und Länge die mit einem softwarebasierten Trace-Ansatz nicht möglich ist. Mit der Transformation in das BTF-Format ist die Auswertung der Anforderungen auch für ein System mit mehreren Kernen oder sogar ECUs möglich. Der TA Inspector ermöglicht die Auswertung formal definierter Anforderungen und unterstützt die Standards AUTOSAR und AMALTHEA. Damit kann die Sicherheit der Applikation nach ISO 26262 über alle Softwareebenen sichergestellt werden. In der Zukunft sollen Anforderungen für Events, die sich über mehrere ECUs verteilen, überprüfbar gemacht werden. Dazu müssen Bus- und ECU-Traces kombiniert und konsolidiert, also auf einen gemeinsamen Zeitpunkt, synchronisiert werden. An diesem Problem wird auch schon im Kontext von einem Forschungsprojekt gearbeitet.

Quellen

[1] R. Hilbrich, R. van Kampenhout und H.-J. Goltz, „Modellbasierte Generierung statischer Schedules fuer sicherheitskritische, eingebettete Systeme mit Multicore-Prozessoren und harten Echtzeitanforderungen," in Herausforderungen durch Echtzeitbetrieb, Springer, 2012, pp. 29-38.

[2] K. Schmidt, D. Marx, K. Richter, K. Reif, A. Schulze und T. Flämig, „On timing requirements and a critical gap between function development and ECU integration," in SAE Technical Paper, 2015.

[3] AUTOSAR, „Specification of Timing Extensions," 2014.

[4] Timing Architects Embedded Systems GmbH, „BTF-Specification," AMALTHEA ITEA2 Project (https://wiki.eclipse.org/images/e/e6/TA_BTF_Specification_2.1.3_Eclipse_Auto_IWG.pdf).

[5] Timing-Architects Embedded Systems GmbH, „TA Tool Suite - TA Inspector," http://www.timing-architects.com/ta-tool-suite/inspector/ (Accessed: 2016-09-10), Regensburg, 2016.

[6] D. Ferrari, Computer systems performance evaluation, Prentice Hall, 1987.

[7] J. Kraft, A. Wall und H. Kienle, „Trace recording for embedded systems: Lessons learned from five industrial projects," in Runtime Verification, Springer, 2015, pp. 315-329.

[8] J. Turley, „Nexus standard brings order to microprocessor debugging," www.nexus5001.org, 2004.

[9] NXP, MPC5643L Microcontroller Reference Manual, http://cache.nxp.com/files/32bit/doc/ref_manual/MPC5643LRM.pdf (Accessed: 2016-09-10), 2013.

[10] iSYSTEM, „Nexus Emulation Board," http://www.isystem.com/files/products/OnChip/MPC55xx/IA257BGA100TQ-564xL_V13.pdf, 2012.

[11] M. Holenderski, M. van den Heuvel, R. J. Bril und J. J. Lukkien, „Grasp: Tracing, visualizing and measuring the behavior of real-time systems," in International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), 2010, pp. 37--42.

[12 ISO, „ISO/FDIS 26262-4:2010(E) Road vehicles Functional safety - Part 4: Product development: system level," ISO, Geneva, 2012.

[13] A. Dr. Schulze, S. Richter, T. Flämig, K. Dr. Schmidt, D. Marx, H. Christlbauer, K. Richter, S. Schliecker und C. Ficek, „Multi-core-Hardware-Tracing in der Praxis," ELIF-Kongress, Baden-Baden, 2015.

[14] L. Michel, T. Flämig, D. Claraz und R. Mader, „Shared SW development in multi-core automotive context," ERTS Kongress, Toulouse, 2016.

Autoren

Felix Martin studierte Elektrotechnik an der Ostbayerischen Technischen Hochschule (OTH) Regensburg und erlangte dort seinen Master of Science in Angewandter Forschung. Seit 2014 ist Herr Martin bei Timing-Architects Embedded Systems GmbH tätig und beschäftigt sich während seines Studiums im Unternehmen unter anderem mit der Analyse von Echtzeit-Systemen mittels Hardware-Tracing. Heute entwickelt Herr Martin in seiner Rolle als Research Engineer innovativen Lösungen und Produkte.
Email: felix.martin@timing-architects.com

Dr. Michael Deubzer ist Geschäftsführer des High-Tech Unternehmens Timing-Architects, dass er nach seiner Promotion an der TU München mit seinem Forschungskollegen 2011 gründete. Im Jahr 2015 wurde er vom deutschen Magazin des MIT als einer der zehn innovativsten Köpfe Deutschlands ausgezeichnet.
Email: michael.deubzer@timing-architects.com

Maximilian Hempe studierte Elektro-/Informationstechnik und Informatik an der Hochschule München und erreichte dort seinen akademischen Grad des Master of Science. Seit 2014 unterstützt Herr Hempe Timing-Architects Embedded Systems GmbH, unter anderem in der Forschung durch die Modellierung des dynamischen Verhaltens von AUTOSAR Betriebssystemen. Heute ist Herr Hempe im Technischen Consulting der Firma tätig. Er betreut verschiedene Kundenprojekte, unter anderem im strategischen Umstieg von Single- auf Multi-core Systeme, bei der Zusammenarbeit zwischen OEMs und Tier1s, und bei der Timing Analyse von verteilten Systemen zur effizienten Weiterentwicklung von hochautomatisierten Fahrzeugen.
Email: maximilian.hempe@timing-architects.com