ENTWICKLUNGSWERKZEUGE

Automatisierte Optimierung von eingebetteten Systemen:

Perfektes Zusammenspiel

Für komplexe Fahrzeugarchitekturen eine optimale Lösung zu finden, wird zu einer immer größeren Herausforderung. Erst das Zusammenspiel von Werkzeugen sowohl für die Simulation als auch für die Worst-Case-Analyse und die automatisierte Optimierung stellt sicher, dass sich die vorgegebenen Ziele auch erreichen lassen.

Von Dr. Ralf Münzenberger, Ingo Houben, Dr. Karsten Albers und Prof. Dr. Frank Slomka

Eine wesentliche Aufgabe in Vorentwicklungs- und Serienentwicklungsprojekten ist die Optimierung der System- und Software-Architektur. Ob Notbremssysteme, Einparkassistenten oder moderne Fahrdynamikregelungen – immer neue Funktionen im Auto bedeuten gleichzeitig auch immer mehr Software-Komponenten. Dementsprechend steigt auch die Anzahl der Tasks, Runnables und Nachrichten auf den Bussen sowie CPUs und Cores, und somit die Komplexität der Architektur stetig an. Hier stets eine optimale Lösung zu finden, erfordert die Unterstützung durch spezielle Tools für eine automatisierte Generierung von Architekturvorschlägen. In diesem Artikel geht es darum, welchen Anforderungen ein solcher Optimierer genügen muss.

Kritisch wird die Komplexität eines Systems, sobald die Einflussfaktoren der Architektur und deren Auswirkung auf das Systemverhalten nicht mehr ausreichend verstanden werden. Dann schnellt das Risiko in die Höhe, Fehler erst sehr spät im Entwicklungsprozess oder sogar gar nicht zu finden. Immerhin lässt sich mittels Simulation das Verhalten dynamischer Architekturen sehr gut verstehen, und Worst-Case-Analysen sichern deren Korrektheit ab. Somit stehen bereits heute geeignete und in der Serienentwicklung erprobte Verfahren zur Reduzierung der Komplexität bereit.

Darüber hinaus ist eine frühzeitige Optimierung der dynamischen Architektur ein wichtiger Meilenstein in jedem Entwicklungsplan. Eine automatisierte Optimierung bietet dabei einen erheblichen Effizienzgewinn, da diese von den vielen möglichen Varianten die am besten geeignete auswählt. Im Folgenden wird Schritt für Schritt aufgezeigt, welche Vorgaben und Verfahren für eine automatisierte Optimierung benötigt werden. Die Berücksichtigung aller Schritte führt zu einer zielgerichteten und somit schnellen Optimierung.

Optimierungsziele vorgeben

Im ersten Schritt werden die konkreten Ziele einer Optimierung betrachtet. Dazu gehören insbesondere das Einhalten von vorgegebenen Grenzwerten sowie die Verbesserung des Systemverhaltens und der Architektur. Die Einhaltung von Grenzwerten ist für einen stabilen und sicheren Betrieb erforderlich. Bei der Verbesserung des Systemverhaltens geht es um größere Reserven, höhere Reaktivität, geringere Kosten oder niedrigeren Energieverbrauch. Diese verschiedenen Optimierungsziele müssen quantifiziert und hierarchisiert bzw. gewichtet werden. Bei der Einhaltung von Grenzwerten stehen dabei folgende Punkte im Vordergrund: Maximal zulässige Bus-, CPU-, Core-Auslastung

- Maximal zulässige Antwortzeiten von ISRs und Tasks
- Minimal zulässiger Durchsatz
- Maximal zulässiger Start-to-Start Jitter von Tasks und Runnables
- Maximal zulässige Latenzzeit von Wirkketten
- Nicht erlaubte Wirkkettenabrisse
- Maximal zulässiges Datenalter

Darüber hinaus ist darauf zu achten, dass nicht erlaubte Wirkkettenabrisse und die ebenfalls nicht erlaubte Mehrfachverarbeitung von Daten bzw. Signalen vermieden und die Reservierung von Bandbreiten eingehalten werden. Zudem muss bei Systemen mit Sensordatenfusion oder redundanter Berechnung häufig die Synchronität von bestimmten Daten und Signalen gewährleistet sein.

Bei den Verbesserungen des Systemverhaltens und der Architektur geht es typischerweise um folgende Punkte:

- Robustheit der dynamischen System- und Software-Architektur
- Erweiterbarkeit
- Minimierung der Bus-, CPU- und Core-Auslastung und Lastbalancierung
- Minimierung von Interrupt Service Routine (ISR)- und Task-Antwortzeiten sowie der Latenzzeiten von Wirkketten

Auf der Basis von mehr als 150 Kundenprojekten hat INCHRON analysiert, welche Anforderungen am häufigsten nicht eingehalten wurden [1]. In 67 % aller Projekte gab es Antwortzeitverletzungen von Tasks und ISRs, in 60 % eine Überschreitung der maximalen Last und in 53 % Wirkkettenfehler. Wirkketten geben den exakten Verkauf des Datenflusses vom Sensor (z.B. Beschleunigung, Radar, Kamera) bis zum Aktuator (z.B. Bremse, Lenkung, Motorsteuerung, Dämpfung) oder auch von Teilschritten wieder. D.h., der Wirkkettenverlauf und die Anforderungen an die Wirkkette können sowohl nur ein einzelnes Steuergerät als auch mehrere miteinander vernetzte ECUs betreffen. Eine Nichteinhaltung der maximalen Latenzzeit einer Wirkkette ist i.d.R. weder für den Normalbetrieb noch aus der Sicht der funktionalen Sicherheit akzeptabel. Wirkkettenfehler sind somit fast immer Show Stopper, also Programmfehler, die so gravierend sind, dass sie die Weiterentwicklung oder den Einsatz eines Produktes verhindern. Daher müssen ihre Ursachen immer verstanden und beseitigt werden. Dies ist bei Antwortzeit- oder Lastüberschreitungen häufig nicht notwendig.

In **Bild 1** ist der Wirkkettenverlauf für den Datenfluss eines vernetzten Systems in einem Gantt-Diagramm zu sehen. Der Datenfluss startet in der Task T_05ms auf der CPU1. Nach weiteren

Verarbeitungsschritten auf dieser Task erfolgt eine Kommunikation mit der Aktuator-ECU über einen CAN-Bus. Im Bild hervorgehoben sind die drei am häufigsten auftretenden Wirkkettenfehler: Überschreitung der maximalen Latenzzeit, Datenverlust durch Wirkkettenabriss und Mehrfachverarbeitung von Daten.

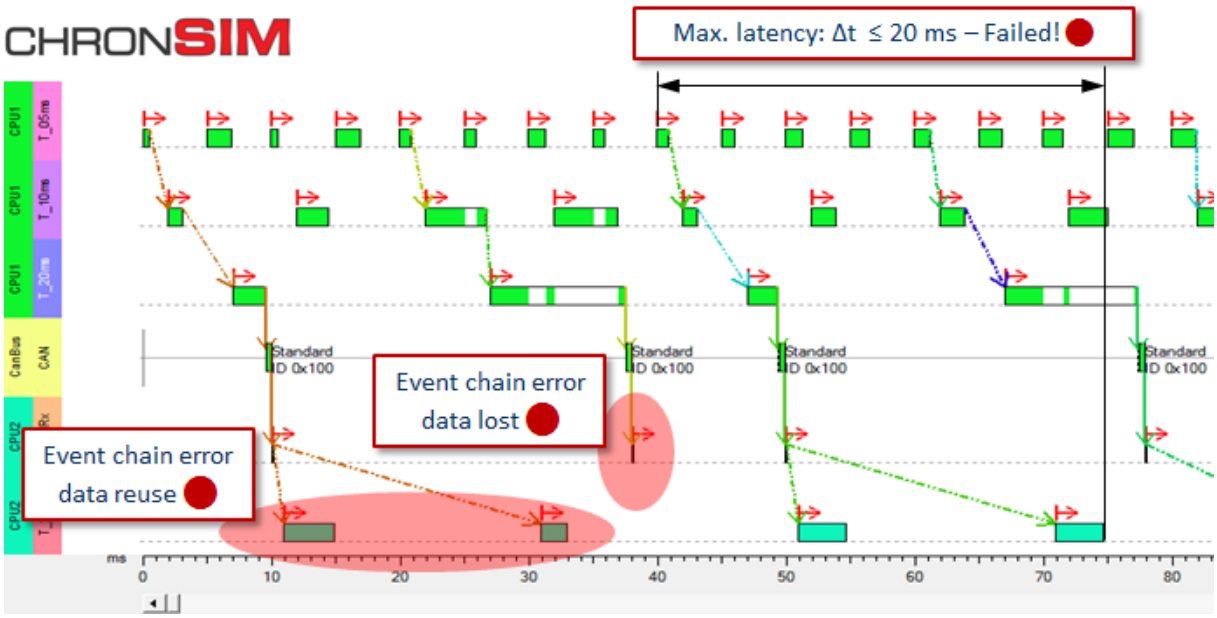

Bild 1. In der Wirkkettenanalyse sind die drei häufigsten Fehler hervorgehoben: Überschreitung der maximalen Latenzzeit, Datenverlust und Mehrfachverarbeitung von Daten. (Bild: INCHRON)

Freiheitsgrade festlegen

Nach den Optimierungszielen ist festzulegen, welche Eigenschaften der Architektur geändert werden dürfen. Diese sogenannten Freiheitsgrade geben als Stellschrauben an, welche Eigenschaften des Systems das Optimierungsverfahren ändern darf und in welchem Rahmen. Wichtige Freiheitsgrade sind:

- ISR- und Task-Prioritäten
- Task-Offset
- Zuordnung von Runnables zu Tasks
- Position der Runnables innerhalb einer Task
- Zuordnung von Tasks auf ECUs, CPUs oder Cores

Bei einer alleinigen Betrachtung von Tasks mit unterschiedlichen Prioritäten auf einer CPU ergeben sich bei 10 Tasks 3,6 Millionen Kombinationen. Bei 20 Tasks sind es bereits über zwei Trillionen (2,4 x 10^{18}) und bei 30 Tasks 2,6 x 10^{32}. Zwar können viele Kombinationen als nicht sinnvoll ausgeschlossen werden, doch ist die Task-Priorität auch nur ein Freiheitsgrad unter vielen, der zu berücksichtigen ist.

Heuristische Optimierungsverfahren

Eine Untersuchung aller möglichen Architekturvarianten (Lösungsraum) mit einem exakten Verfahren zum Auffinden des Optimums ist aufgrund der sehr großen Anzahl der Lösungen nicht sinnvoll. Hier hat es sich gezeigt, dass heuristische Optimierungsverfahren die Mittel der Wahl sind.

Dabei bezeichnet eine Heuristik ein Vorgehen, wie mit unvollständigen Informationen und wenig Zeit dennoch zu wahrscheinlichen Aussagen oder praktikablen Lösungen zu kommen. Beispiele von weit verbreiteten Heuristischen Optimierungsverfahren sind Tabu Search, Genetische Algorithmen oder Simulated Annealing.

Solche Optimierungsverfahren haben das Ziel, den Lösungsraum geschickt zu durchsuchen, um eine geeignete Lösung zu finden, bei der die Optimierungsziele eingehalten werden. Hierfür werden unter Berücksichtigung des oder der aktuellen Lösungskandidaten die nächsten Kandidaten für die Optimierung ausgesucht. Bei diesen neuen Lösungskandidaten ändert sich mindestens der Wert eines Freiheitsgrades. Jeder einzelne Lösungskandidat entspricht einem Zug, und die Menge der nächsten Lösungskandidaten ist die Nachbarschaft.

Heuristische Optimierungsverfahren sind nach dem Auffinden eines lokalen Optimums in der Lage, die Suche global fortsetzen zu können. Zudem eignen sich diese Verfahren dazu, mehrere unterschiedliche Ziele gemeinsam zu optimieren (multikriterielle Optimierung) [2]. Wissenschaftliche Untersuchungen haben gezeigt, dass die Art des heuristischen Optimierungsverfahrens nicht ausschlaggebend ist für eine „gute“ Optimierung, da bei einem richtigen Einsatz alle zu guten Ergebnissen kommen [3].

Der große Zeitfresser bei der Optimierung von eingebetteten Systemen ist der Aufwand für die Bewertung eines Kandidaten. Daraus ergeben sich zwei wesentliche Anforderungen für einen sinnvollen Einsatz eines Optimierers: Erstens, eine schnelle Bewertung. Zweitens, eine reduzierte Nachbarschaft, welche nur eine Auswahl von sinnvollen Zügen enthält, d.h. Züge, die mit einer hohen Wahrscheinlichkeit zu einer Verbesserung führen. Hierdurch lässt sich eine zielgerichtete Optimierung realisieren.

Sinnvolle Züge

Was ein sinnvoller Zug ist, wird im Folgenden anhand der Taskprioritäten eingehend betrachtet. Es handelt sich um ein System mit insgesamt 7 Tasks. In **Bild 2** ist für ein prioriätenbasiertes Scheduling mit unterbrechbaren Tasks die Worst-Case-Antwortzeit (WCRT) der Task T_20Bms zu sehen. Die WCRT liegt mit 24,71 ms deutlich über der Deadline von 20 ms. In dem Diagramm ist zudem die kumulierte Worst-Case-Verdrängung (siehe blaue Balken) solcher Tasks und ISRs aufgetragen, welche zu dieser Antwortzeit beigetragen haben. Beispielsweise hat Tasks T_20Ams einen Anteil von 7,995 ms, die Tasks T_10ms von 6 ms. Es gibt vier weitere Tasks mit einer niedrigeren Priorität als die Tasks T_20Bms.

Die Worst-Case-Antwortzeit der Tasks T_20Bms soll eingehalten werden, indem die Prioritäten der anderen Tasks geändert werden. Bei insgesamt 7 Tasks ergeben sich somit 5040 mögliche Züge. Bezieht man das Wissen mit ein, dass nur eine Änderung der beiden höherprioren Tasks zum Ziel führen kann, ergeben sich bis zu drei Züge.

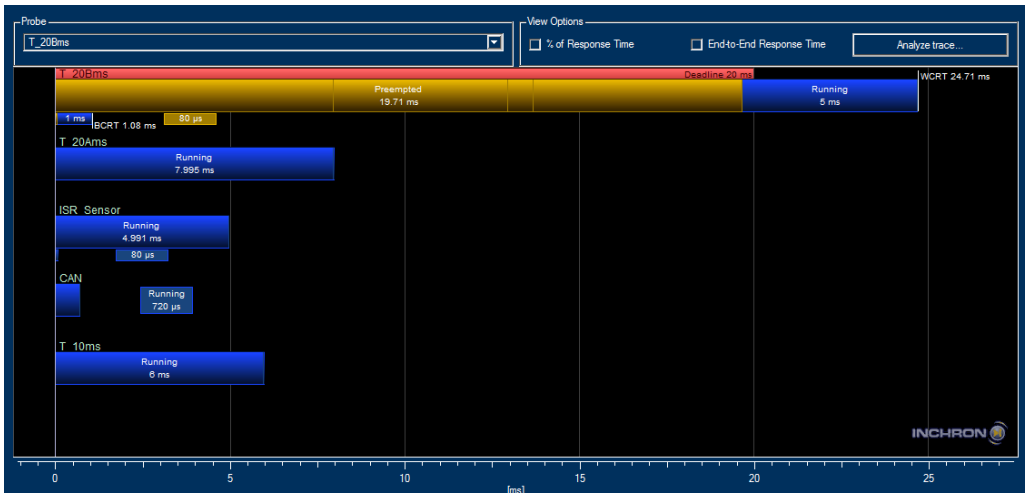

Bild 2. Beispiel für eine Worst-Case-Analyse für ein prioriätenbasiertes Scheduling mit unterbrechbaren Tasks. (Bild: INCHRON)

Sinnvolle Züge lassen sich auch für die Lastoptimierung von Single- und Multi-Core CPUs oder die Verkleinerung der Latenzzeiten von Wirkketten durch eine optimierte Reihenfolge der Wirkkettenschritte ermitteln. Hierfür werden neben dem Systemmodell auch Ergebnisse aus der Bewertung benötigt. Durch einen kombinierten Einsatz der Bewertungsverfahren Simulation und Worst-Case-Analyse, wie es bei der INCHRON Tool-Suite mit dem Simulator chronSIM und dem Validator chronVAL der Fall ist (siehe Kasten), wird die Anzahl der sinnvollen Züge nochmals deutlich reduziert. Dies ist ein wichtiger Baustein für eine zielgerichtete Optimierung.

Zwei Bewertungsverfahren

Die beiden weitverbreiteten Bewertungsverfahren sind die Simulation und die Worst-Case-Analyse. chronVAL realisiert eine schnelle und hinreichend genaue Worst-Case Bewertung durch ein approximatives Echtzeitbewertungsverfahren. Der Simulationskern von chronSIM ist hingegen für eine schnelle Simulation auch von großen verteilten Systemen ausgelegt.

Die Simulation hat vereinfacht gesagt den großen Vorteil, dass das dynamische Verhalten des Systems im Detail betrachtet und somit verstanden werden kann. Allerdings wird – wie auch bei herkömmlichen Testverfahren – nicht immer der schlimmste Fall gefunden. Dies ist die Stärke der Worst-Case-Analyse. Die Worst-Case-Analyseverfahren müssen jedoch immer auf der sicheren Seite sein, was zu einer Überabschätzung und gegebenenfalls zu einem teureren System führt. Ab ASIL A empfiehlt sich deshalb, eine Worst-Case-Analyse durchzuführen, spätestens ab ASIL C ist dies zwingend erforderlich. Da es auch bei höheren ASIL-Einstufungen i.d.R. auch QM-Anteile gibt, führt eine Kombination aus Simulation und Worst-Case-Analyse zu einem kostengünstigen und korrekten System.

Durch die zunehmende Vernetzung im Fahrzeug wird eine steuergeräteübergreifende Optimierung immer wichtiger. Die Auslegung von Wirkketten von den Sensoren bis zu den Aktuatoren im Bereich Chassis oder Fahrerassistenzsysteme sind hier nur zwei Beispiele. Das Bewertungsverfahren muss hierbei nicht nur die dynamische Architektur der einzelnen Steuergeräte berücksichtigen, sondern auch die Kommunikation und die dadurch verursachten Latenzzeiten auf den Bussen (z.B. FlexRay, CAN, Ethernet, LIN). Zudem verursachen driftende Uhren der einzelnen Steuergeräte untereinander und gegenüber einem zeitsynchronen Bus wie den FlexRay häufig Fehler, die nur sehr schwer durch

Testen zu finden und zu reproduzieren sind. Driftende Uhren sind deshalb bei der Worst-Case-Analyse und der Simulation einzubeziehen.

Anpassung des Optimierungsverfahrens

Eine automatisierte Optimierung bietet vielfältige Vorteile beim Design eines Systems. Aufgrund der Systemkomplexität sind heuristische Optimierungsverfahren die Mittel der Wahl. Ausschlaggebend für die Erreichung der Optimierungsziele sind ein geeignetes und aussagekräftiges Timing-Modell sowie eine schnelle Bewertung. Hierfür ist die Erzeugung von sinnvollen Zügen notwendig, die mit einer hohen Wahrscheinlichkeit zu einer Verbesserung führen. Dies führt zu einer zielgerichteten Optimierung.

Eine weitere wichtige Anforderung an einen Optimierer besteht darin, den Anwender bei möglichst vielen unterschiedliche Einsatzszenarien zu unterstützen, sowohl bei der dynamischen Architektur einer einzelnen Multi-Core ECU, als auch bei Wirkketten über mehrere Steuergeräte und unterschiedlichen Bussen. Um eine solche Flexibilität und Leistungsfähigkeit zu erreichen, kann eine systemspezifische Auswahl und Anpassung des Optimierungsverfahrens notwendig sein. Hierbei stehen vor allem Abhängigkeiten und Einschränkungen des Systems im Vordergrund. INCHRON hat sich deshalb dazu entschieden, den Software-Code der in chronOPT verwendeten Optimierungsverfahren der Community zur freien Verfügung zu stellen und ihm unter einer GNU GPL-Lizenz freizugeben.

Literatur

[1] R. Münzenberger: Dynamisches Verhalten. Steuergeräteentwicklung im Spannungsfeld von OEM und Zulieferer. Vortrag im Konferenzband 16. Internationaler Fachkongress in der Automobil-Elektronik, 19. und 20. Juni 2012, Ludwigsburg

[2] Frank Slomka: Mehrkriterienoptimierung verteilter Echtzeitsysteme mit Tabu-Search. Dissertation Dokument Nummer: Nr. 353 Technische Fakultät der Universität Erlangen-Nürnberg, 2002

[3] D. Schuckmann, Diplomarbeit im Studiengang Informatik, Universität Oldenburg, Juli 2006

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 und Optimierung sowie bei der Sicherstellung der Echtzeitfähigkeit unterstützt. Konkret gehören dazu Architekturoptimierung, Migration von Single-Core nach Multi-Core, funktionale Sicherheit oder Prozessberatung.

Ingo Houben

ist als Business Development Engineer bei der INCHRON tätig. Seit 2010 beschäftigt 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.

Dr. Karsten Albers

hat sich bereits in seiner Studienarbeit mit dem Thema Optimierung von Hardware-/Software Co-Design-Systemen befasst. Die Erkenntnis, dass die Bewertungsverfahren entscheidend für den Optimierungserfolg sind, führte ihn zur Entwicklung schneller approximativer Echtzeitanalyseverfahren im Rahmen seiner Promotion. Diese hat er bei der INCHRON GmbH in das Werkzeug chronVAL eingebracht.

Prof. Dr. Frank Slomka

ist seit 2007 Professor am Institut für Eingebettete Systeme/Echtzeitsysteme an der Universität Ulm. Nach einem Studium der Elektrotechnik, Fachrichtung Mikroelektronik an der TU Braunschweig war er u.a bei der Bosch Telecom GmbH, an der Universität Erlangen-Nürnberg sowie an der Universität Oldenburg tätig.

((Kasten))

Werkzeuge für die Optimierung

Die Tool-Suite von INCHRON kombiniert die Vorteile des gemeinsamen Einsatzes einer Simulation und Worst-Case Analyse mit der Optimierung:

- chronSIM ist ein Echtzeitsimulator zum Architekturentwurf und Bewertung von robusten Echtzeitsystemen. Das Einsetzspektrum reicht von Single-, Multi-Core über Multi-CPU ECUs bis zu verteilten Systemen.
- chronVAL ist ein Worst-Case Analyse Werkzeug für Systeme mit harten Echtzeitanforderungen. Ab ASIL-A Einstufung empfohlen, ab ASIL-C zwingend erforderlich.
- chronOPT sorgt für eine zielgerichtete Architektur-Optimierung von Single-, Multi-Core, Multi-CPU und verteilten Systemen.