

Bild 1: Echtzeitprobleme werden oft erst bei der Integration entdeckt und führen zu aufwändigen Re-Designs. Rechtzeitige Echtzeitsimulation in frühen Designphasen hilft solche Iterationen deutlich zu verkürzen.

OPTIMIERUNG VERNETZTER ECHTZEITSYSTEME AUF SYSTEMEBENE

# Die Echtzeit rechtzeitig beherrschen

Das Echtzeitverhalten von Steuergeräten zu beherrschen wird mit steigender Zahl von Funktionen und Vernetzungen immer schwieriger. Zudem ist eine Überprüfung der Echtzeitfähigkeit erst mit dem Source-Code auf dem Target möglich. Für den Systemarchitekten führt dies zu langen und teuren Iterationszyklen. Mit einer Modellierung des Echtzeitverhaltens aber kann man die Performance des Systems frühzeitig abschätzen und auch während der Entwicklungsphase überprüfen und optimieren.

Steuergeräte in Kraftfahrzeugen haben verschiedene Aufgaben abzuwickeln, die mit unterschiedlicher Häufigkeit auftreten. Alle müssen korrekt und planmäßig erledigt werden und sollen vorhersagbare Reaktionszeiten zeigen. Im simpelsten Fall werden die Funktionen stur nacheinander aufgerufen. Die Reaktionszeit entspricht dann der Zykluszeit und die ist, konstante Ausführung vorausgesetzt, vorhersagbar. Sollen mehrere Aufgaben parallel erledigt werden, bieten sich starre Zeitscheiben und Prioritätsebenen an, die mit Hilfe eines Schedulers wiederum gutes Echtzeitverhalten ermöglichen.

Doch so simpel ist die Welt meist nicht. Reaktionen auf die Außenwelt sollen schnell erfolgen, Funktionen werden dynamisch angefordert und sicherheitsrelevante Aufgaben haben immer Vorrang. Wird zum Beispiel ein starres Zeitscheibensystem gemeinsam mit einem dynamischen Kurbelwellensignal oder einem asynchronen FlexRay-Bus betrieben, werden Interferenzen und Schwebungen der verschiedenen Zeittakte auftreten, die schwer vorhersagbar sind. Und hat man erst einmal verteilte Systeme, wo Sensor, Regler und Aktor auf verschiedenen Steuergeräten

residieren, ist ein Zeitverhalten kaum mehr vorhersagbar. Mit Hilfe von Oversampling, Überwachungsmechanismen und verschiedenen Scheduling-Verfahren versucht man den teils divergierenden Zielen dennoch gerecht zu werden. Dabei hat der Systemarchitekt die Aufgabe, das Gesamtsystem so zu optimieren, dass sowohl alle geforderten Aufgaben darauf abgearbeitet werden können als auch die Ressourcen dabei möglichst sparsam genutzt werden. Denn einfach den nächst größeren Prozessor zu wählen, steht außer Frage.

Mit steigender Zahl von Funktionen und Vernetzungen wird diese Optimierungsaufgabe beliebig komplex. Annahmen werden getroffen, Erfahrungswerte wiederverwendet und schlussendlich wird das Ziel iterativ erreicht. Eine solche Iteration bei Echtzeitsystemen bedeutet, dass eine Hard- und Software-Architektur definiert wird, die Applikation implementiert und auf der prototypisch aufgebauten Hardware zum Laufen gebracht wird. Ist das erst einmal erreicht, kann nach einiger Instrumentierung das Echtzeitverhalten schließlich auf einem HiL überprüft werden. Denn das Echtzeitverhalten einer Software kann erst auf dem Zielsystem unter realistischen Betriebsbedingungen

gemessen werden. Liegt der Fokus hingegen auf der Funktionalität, hat man es deutlich leichter und kann schon viel früher im Entwicklungsprozess prüfen und iterieren.

Den Einwand, dass man ja selten ‚auf der grünen Wiese‘ arbeitet und meist ein Vorgängermodell hat, kann man nur bedingt gelten lassen. Auch bei Erweiterungen eines weitgehend unveränderten Systems musste der Architekt erst einmal mit dem Vorgänger iterativ lernen, wie die Performance des Systems ist. Und er wird bei Änderungen des neuen Systems diese Optimierung wiederum mit Iterationen durchführen müssen.

### Echtzeitsimulator *chronSim*

Abhilfe schafft hier eine Abstraktion und Modellierung des Echtzeitverhaltens bereits in frühen Phasen des Systementwurfs. Der Systemarchitekt erstellt ein Echtzeitmodell, in dem das Zeitverhalten der einzelnen Komponenten des Betriebssystems, der Basissoftware und der Applikationen abgebildet werden. Das Gesamtsystem läuft auf dem Echtzeitsimulator *chronSim* ab und reagiert auf Ereignisse der darin mitsimulierten Außenwelt z. B. via Interrupts oder einer Restbussimulation. Abhängig von der Anregung des Systems werden die entsprechenden Datenpfade durchlaufen und deren Ausführungszeiten berücksichtigt. Ein Modell an Stelle des realen Systems zu verwenden, ermöglicht es, Änderungen in der Soft- und Hardware-Architektur schnell auf ihre Auswirkungen auf die Leistungsfähigkeit zu untersuchen. Jetzt kann der Systemarchitekt in kurzen Iterationszyklen verschiedene Alternativen prüfen und das System schnell optimieren. Ist so eine robuste Architektur gefunden, wird die erste Implementierung schon deutlich näher am Performance-Ziel liegen, als herkömmlich entworfene Systeme in dieser Phase. Später, im Lauf der fortschreitenden Implementierung werden auch die Modelle verfeinert und das Echtzeitverhalten ständig überprüft.

Echtzeitmodelle für *chronSim* beschreiben den Rechenzeitbedarf und Ablauf des Sour-

ce-Codes auf dem Target. Sie werden in C geschrieben, um zum einen eine vertraute Notation zu bieten und zum anderen einen fließenden Übergang vom Modell zum realen Source Code zu ermöglichen. Denn *chronSim* ist in der Lage auch Target C-Code auszuführen. Die Modelle nutzen die Betriebssystemaufrufe des gewählten Target-OS, z. B. OSEK, das vom Simulator zur Verfügung gestellt wird. Daher wird ein Echtzeitmodell auch Task-Modell genannt. Die Tasks und ISR (Interrupt Service Routinen) werden mit Standard-OSEK-Notation geschrieben und deren Rechenzeit darin als Delays annotiert. Das DELAY ist ein Schlüsselwort für den Simulator, um den Zeitbedarf der an dieser Stelle geplanten Algorithmen zu definieren. Die so erstellten Tasks und ISR werden vom Simulator gescheduled und in übersichtlichen Diagrammen dargestellt. So können die

```

TASK(TASK_TT_5ms){
  DELAY(300, unit_us); ①
  Schedule(); ②
  while(!finished()) { ③
    DELAY(100, unit_us);
  }
  exectime = 10*data_size; ④
  DELAY(exectime, unit_us);
  DELAY(gaussian(500, 10), unit_us); ⑤
  TerminateTask();
}

```

①: konstante Ausführungszeit

②: Aufruf des Schedulers

③: Ablaufabhängige Ausführungszeit

④: Datenabhängige Ausführungszeit

⑤: Wahrscheinlichkeitsverteilung der Ausführungszeit

Bild 2: Das Echtzeitverhalten des Source-Codes im Task-Modell. Ausführungszeiten werden mit dem DELAY-Kommando für den Simulator definiert.

© automotive

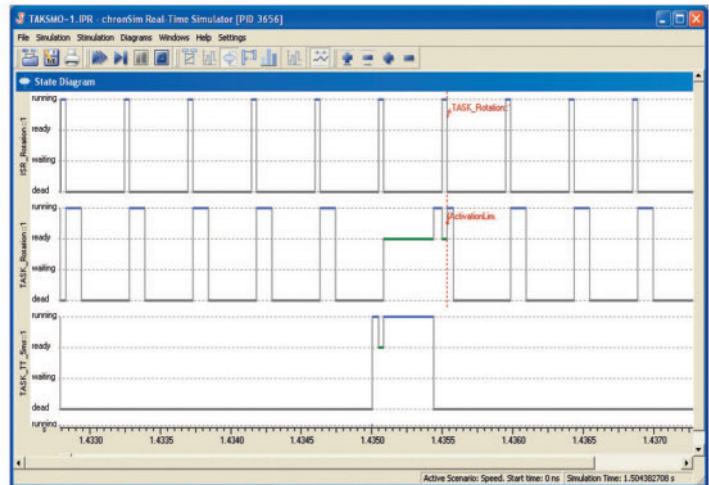


Bild 3: Die Abbildung zeigt ein Task-Zustandsdiagramm, in dem ein zyklischer Kurbelwellen-Interrupt eine Task aktiviert, die aber von einer höher priorisierten Task (dritte Kurve) verdrängt wird. Das Resultat, eine Mehrfachaktivierung, kann der Systemarchitekt mit chronSim schon analysieren, bevor eine Hardware vorhanden ist.

© automotive

Abläufe der Tasks und ISR mit denen der realen Systeme sehr einfach verglichen werden.

Die im DELAY spezifizierten Ausführungszeiten legt man in frühen Phasen zunächst ad hoc fest. Der Systemarchitekt trifft Annahmen oder setzt hier Vorgaben ein. Hat man ein vergleichbares Vorgängersystem, können aus gemessenen Laufzeiten am Prüfstand reelle Zeiten für das Modell gewonnen werden. Je weiter die Entwicklung fortschreitet, desto feingranularer werden Ausführungszeiten ermittelt und im Task-Modell eingesetzt. Abläufe und kausale Zusammenhänge, die zu unterschiedlichen Datenpfaden und damit Ausführungsreihenfolgen/-zeiten führen, können mit allen Mitteln, die C bietet, im Task-Modell abgebildet werden. Wird beispielsweise nach Empfang eines bestimmten Signals via CAN eine Berechnung ausgeführt, wird im Task-Modell eine Case-Struktur eben diese Fallunterscheidung abbilden.

Diese dynamische Reaktion auf die Außenwelt wird der Simulator nach dem entsprechenden Stimulus ausführen. Um das Modell zu triggern, stehen verschiedene Möglichkeiten zur Wahl. Über komfortable Eingabemasken können zyklische, schwankende und stochastische Interrupts ausgelöst werden. Durch Import von Trace-Daten können Anregungsszenarien aus realen Messungen auf dem Simulator nachvollzogen werden. Mit der Restbussimulation aus CANdb- oder Fibex-Daten wird das System auf Datenkommunikation reagieren können. Diese Stimuli sorgen dafür, dass das Echtzeitmodell, vergleichbar mit einem Prototyp auf dem HiL, unter realistischen Bedingungen ausgeführt wird und entsprechendes Echtzeitverhalten nachvollziehbar macht.

### Verteilte Uhren

Einen kritischen Aspekt in Echtzeitsystemen stellen die Uhren dar. Solange alle Abläufe und Anregungen auf einer

stabilen, globalen Zeitachse zyklisch auftreten ist ein System leichter planbar. In der Realität hat man mehrere Steuergeräte mit eigenen Uhren, zufällige Ereignisse und immer häufiger den Echtzeitbus FlexRay, der eine weitere Zeitbasis einbringt. Driften zwei scheinbar synchrone, zyklische Ereignisse auseinander, wird es früher oder später zu (Schwebungs-)Effekten kommen, die eventuell nur selten in der Praxis auftreten werden. Aber sind sie deswegen harmlos? In chronSim werden diese verteilten Uhren in einem patentierten Verfahren berücksichtigt. Der unterschiedliche Stand und Gang der Uhren zueinander wird modelliert. Hier bietet die Echtzeitsimulation sogar die Chance, solche seltenen Fälle gezielt zu provozieren indem man die Uhren künstlich stark driften lässt und Phasenlagen der Zyklen zueinander verschiebt. Man erkennt kritische Systemzustände und kann beurteilen, ob das System damit korrekt umgehen kann. Die Echtzeitmodellierung von vernetzten Steuergeräten ermöglicht es, das Echtzeitverhalten dieser komplexen Systeme frühzeitig verständlich zu machen. Das ‚Henne-Ei-Problem‘, erst die Architektur prüfen zu können, wenn die Hard- und Software fertig ist, wird aufgehoben. Dabei spart man langwierige Iterationszyklen mit teuren Prüfstandszeiten und aufwändigen Re-Designs ein. (la)

Bearbeitet nach Unterlagen der Inchron GmbH, 14482 Potsdam.