

(Quelle: Live Devices)

der Codierung eine zu hohe Taktfrequenz des ausführenden Mikroprozessors erfordert, um den zeitgerechten Ablauf des Codes zu ermöglichen. Es gilt also, möglichst von Anfang an allen Routinen der Applikation eine bestimmte und von allen akzeptierte Ablaufzeit zuzuordnen und damit böse Überraschungen zu vermeiden.

Die Problematik besteht darin, dass bei der Komplexität und den in vieler Hinsicht asynchronen Interaktivitäten externer Ereignisse unter der Kontrolle des Echtzeit-Betriebssystems niemand sicher sein kann, ob alle Deadlines unter allen Umständen eingehalten werden können. Die einzige Alternative ist eine Entwicklung unter strikter Berücksichtigung der Zeitvorgaben während des gesamten Entwicklungsprojektes.

► Welcher Fall ist „der schlimmste“?

Die erste Definition der Ablaufzeiten wird sicherlich auf der Erfahrung der Entwickler basieren – auf jeden Fall müssen aber die Beteiligten aller Gruppen mit den Vorgaben einverstanden sein. Aber wie kann sichergestellt werden, dass die getroffenen Annahmen auch ein System beschreiben, das unter allen Anforderungen einen zeitgerechten Ablauf garantiert? Gerade bei Interrupts kann viel „Vorarbeit“ erforderlich sein, bis die eigentliche Aufgabe angegangen werden kann; eine genaue Berechnung der auftretenden Antwortzeiten ist damit fast unmöglich. Unterbrochene Interrupts mit mehreren Prioritätsebenen sorgen dafür, dass meist die Ablaufsicherheit einer Applikation erst beim Testen herausgefunden wird, wenn Worst-case-Szenarios durchgespielt werden.

Die Hauptfrage bleibt aber dennoch unbeantwortet: Welche Kombination von Eingangs- und Ausgangsparametern und Interruptprioritäten ist eigentlich der sogenannte „schlimmste Fall“? Kein Test kann sicherstellen, dass dieser Fall abgedeckt ist. Ein kurzer Ausflug in die Theorie zeigt dies ganz deutlich (*Bild 2*). Die Wahrscheinlichkeitsverteilung der möglichen Antwortzeiten eines Embedded-Systems: Die minimale Antwortzeit wird zwar niemals unterschritten, ob aber auch die Worst-

Software-Entwicklung mit der Stoppuhr

Echtzeit-Verhalten von Anfang an im Griff durch Deadline-based Monotonic Analysis

Komplexe Echtzeit-Systeme mit vielen Tasks sind so unüberschaubar, dass man sich bei einem Test nicht sicher sein kann, ob der ungünstigste Fall wirklich mitgetestet wurde. Eine „Schedulability“-Analyse ist ein zuverlässiges Hilfsmittel zur Vorhersage, ob alle vorgegebenen Zeitschranken auch in jedem Fall eingehalten werden können.

Von Dr. Andrew Coombes

Bei der Entwicklung von Hardware wird davon ausgegangen, dass die einzelnen Komponenten schnell genug sind, um die später entwickelte Software abzuarbeiten. Müssen jedoch Software- und Hardware-Komponenten in komplexen Echtzeit-Systemen zusammenspielen, können Situationen auftreten, die die Unterscheidung von harter und weicher Echtzeit erforderlich machen. Der Ansatz von Live Devices (<http://www.livedevices.com/>) und der Einsatz von

Real-Time Architect stellen sicher, dass ein hartes Echtzeit-Verhalten unter Einhaltung aller Zeitbedingungen garantiert werden kann.

In komplexen Mikroprozessorsystemen müssen für die Entwicklung heutzutage Entwicklergruppen eingesetzt werden, die das Design in viele Einzelblöcke aufspalten, von der Architekturdefinition über die Codierung bis hin zu Test und Freigabe (*Bild 1*). Es hilft nicht viel, wenn die Spezifikation Ablaufdetails definiert, die dann später bei

Case-Antwortzeit während eines Testlaufes auftritt, bleibt unsicher. Das Rezept, um diesem Problem beim Software-Design aus dem Weg zu gehen, lautet „Deadline-based Monotonic Analysis“. Real-Time Architect von Live Devices ist die Implementierung genau dieses mathematischen Modells, angepasst an die Anforderungen der Systementwicklung.

Beim Einsatz von Real-Time Architect wird nicht nur das Echtzeit-Verhalten nach der Codierung verifiziert, sondern es wird schon im Stadium des Code-Entwurfes das Worst-case-Verhalten statisch durchgerechnet und dokumentiert. Da das zeitliche Verhalten aller Komponenten vorher definiert wurde und das eingesetzte Echtzeit-Betriebssystem entsprechend dieser Ablauf-Anforderungen aufgebaut ist, kann das komplette zeitliche Verhalten auf der Basis dieser Vorgaben durchgerechnet werden. Es ist damit am Anfang noch nicht geklärt, ob es auch codierbar ist, aber für alle Beteiligten liegen exakte Zeitabläufe vor, an denen man sich orientieren muss.

Alle Routinen bestehen sozusagen aus einer Aneinanderreihung von einzelnen Black-Boxes, beschrieben durch Grenzwerte wie Ablaufzeit, Deadlines, Prioritätsebene und Unterbrechbarkeit. Bei der Codierung müssen lediglich diese Rahmendaten eingehalten werden. Lassen sich bestimmte Deadlines dann nicht mehr einhalten, weil zum Beispiel der Code zu lang ist, können Teile des Gesamtmodells angepasst werden, und ein neuer Lauf von Real-Time Architect gibt entweder ein „ok“ oder zeigt die Problemfälle an. Dieser Ansatz passt auch zum Entwicklungsablauf in Bild 1, denn unabhängig davon, wo man sich im Entwicklungszyklus befindet, gelten in allen Bereichen dieselben Vorgaben. Als „Nebeneffekt“ ergibt sich eine Minimierung der Fehlerkosten: Ein Fehler im Design wird um so teurer, je später er im Projektverlauf entdeckt wird, denn er muss ja dann in allen „Statio-

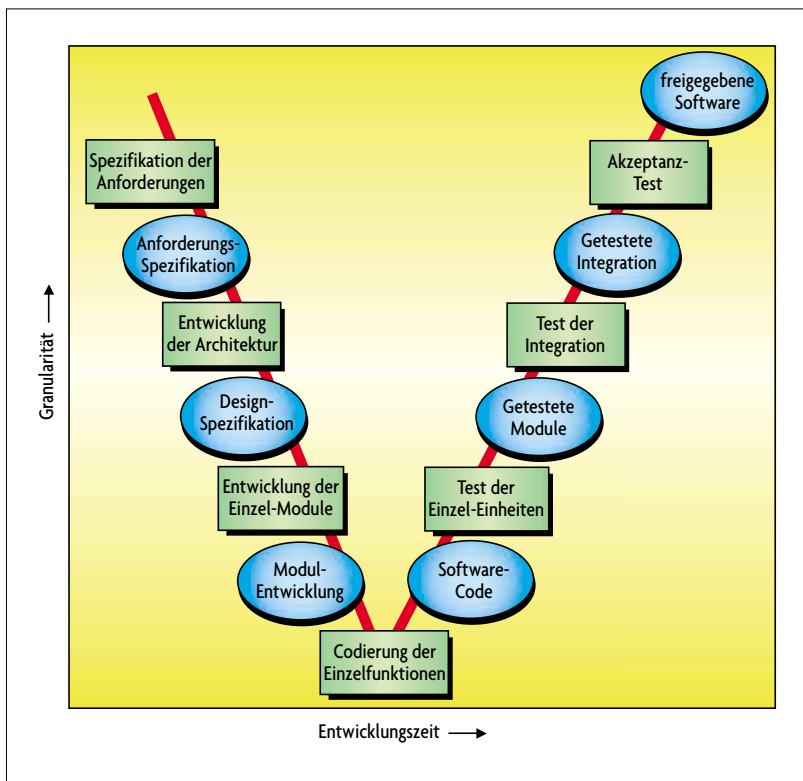
nen“ korrigiert werden. Der hier beschriebene Ansatz und die sofortige Kontrollmöglichkeit stellen sicher, dass er sich nicht fortpflanzt, sondern sofort lokal korrigiert werden kann.

► Optimierungen als „Nebenprodukt“

Da alle zeitlichen Abläufe der Applikation von Anfang an vordefiniert sind und lediglich während des Designs an

die aktuellen Anforderungen angepasst werden müssen, kann der Code am Schluss leicht verifiziert werden. Durch die Definition aller Ablaufmodalitäten wird sichergestellt, dass alle so definierten Deadlines immer eingehalten werden, wobei das eingesetzte Echtzeit-Betriebssystem bereits mit in die Berechnungen einbezogen ist. Durch die konsequente zeitliche Ablaufplanung ergeben sich aber auch Möglichkeiten für Optimierungen:

Bild 1. Die Entwicklung von größeren Software-Projekten vollzieht sich im sog. V-Modell. Real-Time Architect unterstützt diesen Ablauf, indem die zeitliche Ablaufplanung der Software von Anfang an berücksichtigt wird.



- Die Auslastungs-Analyse gibt an, wieviel Reserven der Prozessor noch hat. Sind es noch 2 % oder sogar 30 %? Diese Information kann als Basis für die Implementierung neuer Features oder als „Reserve-Block“ für spätere Software-Erweiterungen in das Design einbezogen werden.
- Minimierung der Prioritätsebenen: Je komplexer das Design, desto mehr Prioritätsebenen sind meist erforderlich. Das kostet Speicher im Stack und Zeit bei der Ausführung. Durch Umbau der Struktur können eventuell Ebenen gespart werden, wenn die Verifikation die Einhaltung der Zeitschranken garantiert.
- Optimierung der Taktfrequenz: Da über Real-Time Architect die komplette erforderliche Prozessorleistung für die Applikation bekannt ist, kann der Entwickler entscheiden, ob eventuell sogar ein langsamerer Prozessor eingesetzt werden kann, mit Vorteilen bei Kosten und EMV.

Deadline-based Monotonic Analysis

Bei den meisten Echtzeit-Systemen ist es unmöglich, alle Kombinationen der Eingangsaktivitäten, der Ablaufsequenzen der einzelnen Tasks und der Interrupts zu testen.

Beim Einsatz preemptiver Echtzeit-Betriebssysteme, z.B. im Automobilbereich, können damit Ablauffehler beim Testen unentdeckt bleiben, die dann im Produktionsmodell auf der Straße „weitergetestet“ werden. Bei der hohen Stückzahl des entsprechenden Modells multipliziert mit der Laufleistung auf der Straße ergeben sich Fahrsituationen, die im Test eventuell nicht abgedeckt waren. Diese Fehler führen zu hohen Servicekosten; im schlimmsten Fall ist eine Rückrufaktion notwendig, um fehlerhafte Elektronik durch neue mit geänderter Software zu ersetzen.

Der Einsatz der Schedulability-Analyse bietet eine effektive Methode, um Software-Qualität und Zuverlässigkeit zu steigern, da eine statische Analyse der



Real-Time Architect zeigt die Ausführungszeit der verschiedenen Tasks an.

Ablaufgenauigkeit ermöglicht wird. Die Software kann damit so entwickelt werden, dass das Zeitverhalten vorhersehbar ist (s. Bild). Dieser Fokus auf vorhersehbare und funktionale Genauigkeit führt zu Softwarecode, der neben der hohen Qualität auch die Wartung vereinfacht.

Schedulability-Analyse ist eine mathematische Technik, die alle Ablaufmuster, Zeitbeschränkungen, Interrupt-Prioritäten und die Ausführungszeiten von Tasks und Interrupt-Handlern dazu verwendet, um daraus die Worst-case-Antwortzeiten zu

berechnen. Aus den Ergebnissen folgt dann, ob alle Deadlines immer eingehalten werden. Dabei werden alle möglichen Szenarien einbezogen, einschließlich der Einflüsse von Interaktionen bezüglich gemeinsamer Speicherressourcen in verschiedenen Prioritätsebenen und Blockierungen.

Zuerst wird auf der höchsten Prioritätsebene ausgerechnet, wieviel Rechenleistung erforderlich ist. Aus Taskwiederholungszeit und Tasklänge kann dann berechnet werden, wieviel Prozessorzeit dies zusammen mit den anderen Tasks in der nächstniedrigen Ebene ergibt. Dieser Rechenvorgang wird bis zur niedrigsten Prioritätsebene hin durchgerechnet und damit sind alle Summenzeiten einschließlich der Handler-Overheads bekannt und es ergeben sich Lösungen auf die wichtigen Fragen:

- Werden alle Deadlines eingehalten?
- Wieviel Prozessorzeit ist noch ungenutzt?
- Kann die Zahl der Prioritätsebenen reduziert werden?
- Kann die Taktrate des Prozessors reduziert werden?

Genauer zur Deadline Monotonic Analysis unter <http://www.livedevices.com/realtime/schedulability.shtml>

Der Designer hat damit gerade bei Neuentwicklungen die Möglichkeit, die vielen Freiheitsgrade bezüglich Prozessor-typ, Taktfrequenz, RAM und ROM auszuprobieren und zu optimieren, bleibt jedoch garantiert immer innerhalb des Rahmens der vorgegebenen Spezifikation. Bei späteren Modifika-

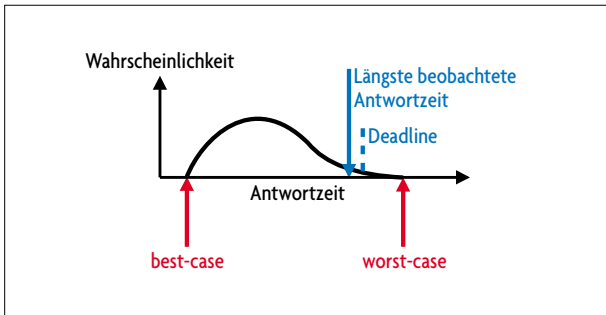


Bild 2. Wahrscheinlichkeitsverteilung der möglichen Antwortzeiten eines Embedded-Systems. Die minimale Antwortzeit wird im Test niemals unterschritten. Ob aber die Worst-Case-Antwortzeit jemals getestet wurde, ist unsicher.

tionen liegt die komplette Ablaufstruktur bereits vor und erleichtert schon von Anfang an Aussagen, wieviel „Platz“ noch frei ist, ohne dass Prozessor oder Taktfrequenz verändert werden müssen.

► Gut geplant ist halb entwickelt

Mit Real-time Architect haben Entwickler zum ersten Mal die Möglichkeit, auch beim Einsatz preemptiver Echtzeit-Betriebssysteme die Einhaltung von Deadlines garantieren zu können. Neu verfügbare Hardware wie zum Beispiel Prozessoren, bei denen per Software die interne Taktfrequenz und die Betriebsspannung den Anforderungen entsprechend umgeschaltet und angepasst werden können, sowie Multiprozessorsysteme, speziell im Automobilbereich, bieten nur durch den Einsatz dieser Techniken die Möglichkeit, die Übersicht zu behalten und die Einhaltung der Deadlines zu garantieren. *jk*



Dr. Andrew Coombes

studierte an der Universität von York/UK und arbeitete dort im Bereich sicherheitskritische Systeme. Seit 1999 bei Live Devices, ist er jetzt als Product Manager verantwortlich für die Produktstrategie von Real-Time Architect, einem Produkt zum Berechnen des RTOS-Echtzeit-Verhaltens.

► E-Mail: andy.coombes@livedevices.com

Literatur

- [1] Burns, A.; Wellings, A.: Real-time systems and their programming languages. Addison Wesley, 1996.
- [2] Tracey, N.: Echtzeit-Verhalten im Griff. Realisierung eines „harten“ Echtzeit-Systems mit der Deadline-Monotonic-Analysis-Methode. *Elektronik* 2001, H. 24, S. 98ff.
- [3] Tindell, K.: Echtzeit-Betriebssystem für Ein-Chip-Systeme. *Elektronik* 1999, H. 18, S. 130ff.