***Prozessorzeit optimal ausnutzen***

# Gerecht verteilt

**In allen Bereichen versucht man, soviel Leistung wie möglich aus einer vorhandenen Hardware zu erhalten; es sollen so viele Features vom Prozessorsystem ausgeführt werden wie irgendwie möglich. Das Problem liegt darin, festzustellen, wann alle Ressourcen aufgebraucht sind. Ein Verfahren namens »Deadline Monotonic Analysis« bietet eine Lösung.**

Dr. Andrew Coombes
***ist Produktmanager bei LiveDevices***

Am schönsten wäre es, hätte der Entwickler die Möglichkeit, sein System zu definieren, dann die entsprechenden Ablaufzeiten der einzelnen Tasks grob festzulegen und danach über eine Analyse herauszufinden, ob alles in diesem Zeitplan noch »rund« läuft, also die angenommene Prozessorleistung und Taktfrequenz ausreicht. Bei verschachtelten Prioritäten und Interrupts im System wird dies oft so komplex, dass man meist den sicheren Weg wählt. Dabei nimmt man den schlimmsten Fall aller Aktivitäten an, gibt noch einmal ausreichend Prozente an Sicherheit dazu, und die erste Definition liegt fest.

Genaueres ergibt sich später im Entwicklungszyklus, wenn das System bereits läuft, und bei den Tests. Es wird also die Erfahrung aus anderen Projekten sowohl bezüglich der Ablaufzeiten als auch der worst-case-Zeiten als Basis genommen.

## *Statisches Scheduling*

Die konservativste Art der Implementierung besteht darin, ein solches System mit statischem Scheduling aufzubauen. Dabei baut man ein festes Zeitraster auf, und bringt die entsprechenden Tasks mit der jeweiligen Ablauflänge darin unter.

Mit angenommenen zusätzlichen Sicherheitszeiten kombiniert ergibt sich dann ein sicherer Ablauf. Als Beispiel sei ein einfaches System mit fünf Tasks gezeigt, bei dem ein solches festes Zeitraster eingesetzt wird. Im Zeitraster muss für Task 1 jede ms ein Slot mit der entsprechenden Ablauflänge bereitgestellt werden.

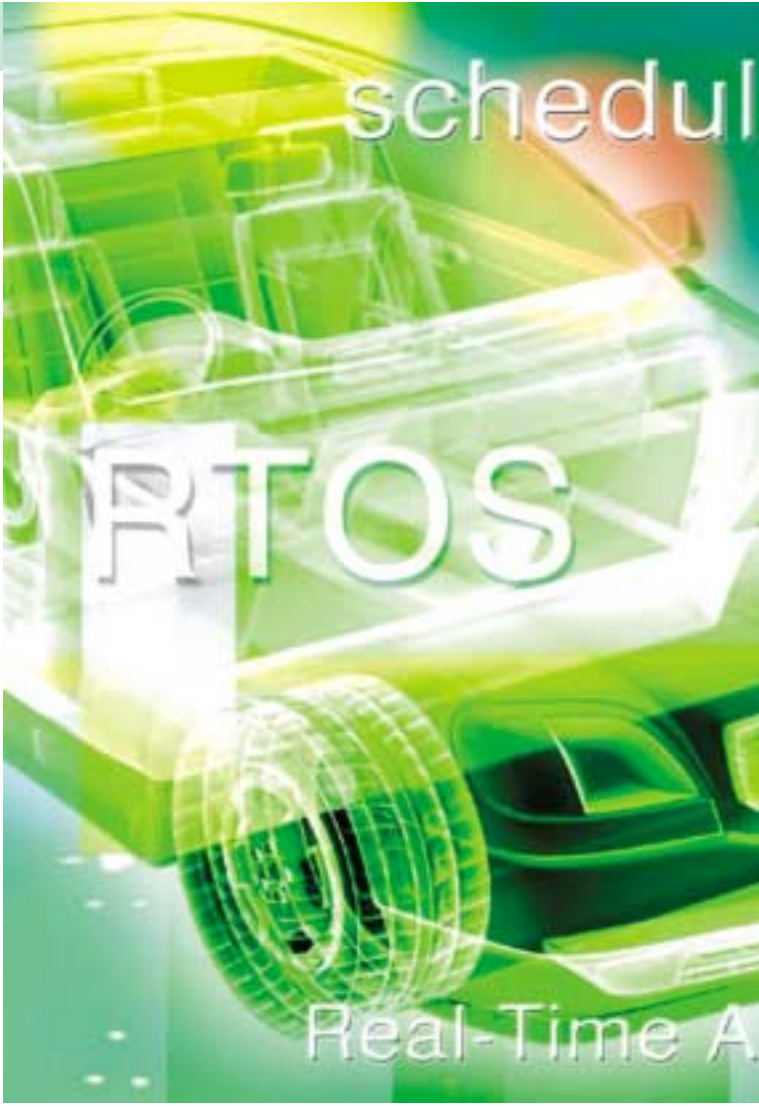
Der Einfachheit halber definiert man also ein 1-ms-Raster.

Task 2 muss alle 2 ms ausführbar sein, damit muss in jedem zweiten Raster ein Slot belegt werden. Die anderen drei Tasks müssen auf jeden Fall einmal innerhalb der 4 ms untergebracht sein. Task 5 hat mit seinen 900 µs in keinem der Slots mehr Platz. Die Routine muss also in zwei Teilen (T5a und T5b) in zwei Slots ausgeführt werden. Addiert man alle Ablaufzeiten, ergibt sich eine aktive Zeit von 2,8 ms.

Damit bleiben etwa 1,2 ms ungenutzt, die zwar als Prozessorleistung zur Verfügung stehen, aber diese 30% sind bei diesem Ansatz Leerlauf.

Betrachtet man alle Zeitfenster, dann steht in allen 4-ms-Blöcken »Freizeit« zur Verfügung; man könnte also den Prozessor auf jeden Fall um Einiges langsamer laufen lassen, um den Stromverbrauch zu reduzieren, man könnte in der verbliebenen Zeit neue Features unterbringen, oder vielleicht ist es mit den reduzierten Anforderungen sogar möglich, einen kostengünstigeren Prozessor einzusetzen.

Reale Systeme sind sehr viel komplexer als das hier beschriebene Beispiel. Zusätzlich sind dann Interrupts zu berücksichtigen und es sind




dann auch viel mehr Tasks auf die verschiedenen Zeitfenster zu verteilen. Ablaufverfolgung und Debugging sind deutlich schwieriger zu realisieren.

## Der Prozessor wird ausgebremst

Noch komplizierter wird es bei Updates, Codeänderungen und Erweiterungen. Hier ist dann der Zusammenhang oft nicht mehr klar, und man muss viel Zeit investieren, um die genauen Hintergründe zu verstehen.
Alle diese Optimierungen führen aber auf jeden Fall dazu, dass man die Grenze der Prozessorleistung erreicht und vielleicht sogar überschreitet, und damit kann oder wird es zu Problemen bei der Programmausführung kommen. Eine bindende Aussage ist oft wegen der Komplexität des Zusammenspiels von Hardware und Software einfach nicht möglich. Verlässt man sich dann auf das Testen, kann der Worst-Case eventuell gar nicht getestet worden sein, da er nicht bekannt ist. Wie nahe kann man also an die Grenze gehen, bzw. wieviel Reserve sollte man zur Sicherheit einbauen?

## Mitgerechnet

Die beste Lösung besteht darin, von dem festen Raster komplett abzugehen und die einzelnen Tasks mit fest eingestellten Prioritäten unterbrechbar zu gestalten. Um dann die Prozessorzyklen aber soweit wie möglich ausnutzen zu können, sind zwei Elemente erforderlich, nämlich ein Echtzeitbetriebssystem (RTOS), das selbst sehr klein ist und auch wenig zusätzliches RAM als Arbeitsbereich erfordert, und ein Werkzeug, mit dem man messen und nachweisen kann, ob alle Tasks mit der erforderlichen Wiederholrate ohne Probleme ablaufen. Zusätzlich muss angezeigt werden, wieviel »Luft« noch ist, damit eventuell weitere Optimierungen möglich sind.
Wie mit einer Schraubzwinge wird also alles zusammengeschoben, bis alle »Luft« raus ist.
Damit kann ein vorgegebener Funktionsumfang nach Taktfrequenz, Prozessorfamilie und Controllertyp optimiert werden. Alle auftretenden Tasks sind in Wiederholrate und Ablauflänge klar definiert, sodass das Werkzeug den »längsten Weg« ausrechnen kann und diese Worst-case-Taskfolge damit auch bekannt ist und getestet werden kann.
Mit Hilfe der »Deadline Monotonic Analysis« (DMA) lässt sich bestimmen, ob auch im schlimmsten Falle alle Tasks beendet werden können.
Das Verfahren (siehe Kasten) analysiert, ob alle im System vorkommenden Tasks mit der entsprechenden Wiederholrate entsprechend den Vorgaben ablaufen können. Hier ist der Begriff der Interferenzzeit wichtig. Diese ist dann zu bestimmen, wenn es zu einem gegebenen Task nur einen Task mit höherer Priorität gibt, und besteht aus der Anzahl der Male, in denen der »wichtigere« Task den niedriger priorisierten unterbrechen kann.
Die gesamte Interferenz setzt sich dann aus den einzelnen Interferenzen zusammen.

Gleichung 1

$$I_i = \sum_{\forall k \in hp(i)} \left\lceil R_i / T_k \right\rceil C_k$$

Gleichung 2

$$R_i^0 = C_i$$

$$R_i^{n+1} = C_i + \sum_{\forall k \in hp(i)} \left\lceil R_i / T_k \right\rceil C_k$$

## Werkzeug der Wahl

Ein Werkzeug, das diese Möglichkeiten bietet, ist Real-Time Architekt (RTA) von LiveDevices. Da das Analysewerkzeug genaue Kenntnisse über die zeitlichen Abläufe innerhalb des Echtzeitbetriebssystems haben muss, wird RTA nur in Kombination mit dem Echtzeitbetriebssystem ausgeliefert.
Das Tool vereinigt die Vorteile eines OSEK-konformen Echtzeitbetriebssystems und eines DMA-basierten Analysetools für die Ablaufkontrolle in einem einzigen Produkt. Die OSEK-Konformität ist für Anwendungen außerhalb des Automobilbereiches nicht unbedingt erforderlich – die Vorteile sind aber auch in anderen Anwendungen wichtig.
RTA basiert auf dem Modell MBA (Model-Build-Analyse). Dabei entsteht ein Modell der Applikation mit festgelegten Zeiten und Deadlines.
Diese basieren zunächst auf Erfahrungswerten und können im Laufe des Projekts vom Anwender sukzessive durch die echten, also gemessenen Daten ersetzt werden.

## Aus Erfahrung lernen

Ausgehend von diesem Modell wird die Applikation immer mehr verfeinert, bis der endgültige Applikationscode fertig ist. In jedem Schritt lässt sich überprüfen,


ob alle vorgegebenen Deadlines immer noch eingehalten werden.
Auf diese Weise kann man sofort feststellen, mit welcher Änderung der Software plötzlich Fehler auftreten. RTA bietet vier verschiedene Möglichkeiten zur Analyse der Applikationssoftware:

- Die Analyse des Zeitplans prüft, ob das System bezüglich der zeitlichen Abläufe funktionsfähig ist,
- die Empfindlichkeitsanalyse zeigt an, wieviel zeitliche Reserve in den einzelnen Tasks steckt,
- die Minimierung der Prioritätsebenen gibt Hinweise darauf, ob Stack-Speicher eingespart werden kann und
- die Minimierung der Taktfrequenz weist darauf hin, ob ein langsamerer Prozessor eingesetzt werden kann, mit Vorteilen bei Kosten und EMV.

## Korrekte Berechnung

Ist der Entwickler mit dem Applikationscode zufrieden, kann er mit Hilfe von RTA die Konfigurationsdaten und den Code für das auch in der Analyse angenommene RTOS erzeugen.
Damit ist sichergestellt, dass es in der Endanwendung keine Unterschiede zwischen dem berechneten Verhalten und dem Verhalten der Kombination aus Betriebssystem und Applikation gibt. (mc)

**Deadline Monotonic Analysis**

DMA, also Deadline Monotonic Analysis ist eine Technik, mit der man berechnen kann, ob auch im Worst-Case alle Tasks beendet werden können. Ein Vorläufer war die »Rate Monotonic Analysis« (RMA), bei der aber nur bis zu etwa 70% der Prozessorzeit ausgenutzt werden konnte. DMA analysiert sicher, ob alle im System vorkommenden Tasks mit der entsprechenden Wiederholrate entsprechend der Vorgaben ablaufen können. Zusätzlich wird angezeigt, wieviel der Prozessorzeit noch ungenutzt ist und optimiert werden kann.
Im Folgenden seien $C_i$ die maximale Ablaufzeit der Task, $D_i$ die maximal zulässige Deadline, $T_i$ die Periode der Task und $R_i$ die Worst-case-Antwortzeit. $C_i$ kann entweder als Vorgabe oder als die exakt im System gemessene Ablaufzeit bei der entsprechenden Taktfrequenz festgelegt werden. $R_i$ wird über die DMA-Technik errechnet, $T_i$ und $D_i$ sind über die Systemanforderungen definiert. $R_i$ besteht aus der Summe von Interferenzzeit $I_i$ und Taskablaufzeit $C_i$. $I_i$ ist die Zeit, die als Taskübergangszeit verloren geht.
Gibt es zum Beispiel nur eine Task mit höherer Priorität, ist es nötig, die Interferenzzeit zu berechnen. Diese besteht aus der Anzahl der Male, in denen die höher priorisierte Task k die niedrigere (i) unterbrechen kann, multipliziert mit der Ausführungszeit von k, vergleiche Gleichung 1. Dabei ist hp(i) die Anzahl der Tasks, die höher priorisiert sind als i.
Dann gilt: $R_i = C_i + I_i$.
So ergibt sich eine rekursive Beziehung, siehe Gleichung 2. DMA als mathematische Form geht von einigen Voraussetzungen aus. So wird eine Task während der Ausführung niemals verzögert, wenn sie auf eine andere Task warten muss. Außerdem führt das Echtzeitbetriebssystem das Scheduling und die Umschaltung zu anderen Tasks in Nullzeit aus. LiveDevices hat diese mathematischen Formeln im Produkt Real-Time Architekt so erweitert, dass diese Beschränkungen nicht mehr zutreffen. Damit wird die Analyse für reale Systeme anwendbar.

**LiveDevices/**
**AK Elektronik**
**Telefon** 0 82 50/99 95 0
**Fax** 0 82 50/99 95 20