Hardwarebasiertes Task-Scheduling für Betriebssysteme mit harten Echtzeitanforderungen

Jens Hildebrandt, Dirk Timmermann

1 Einleitung

Der Scheduler eines Betriebssystems hat gleich in doppelter Hinsicht Einfluß auf dessen Echtzeitverhalten. Zum einen hat der Scheduler durch die in ihm umgesetzte Zuteilungsstrategie für die richtige Ablaufreihenfolge der auszuführenden Echtzeit-Tasks zu sorgen, zum anderen trägt er als eine der am häufigsten aufgerufenen Funktionen nicht unwesentlich zum Betriebssystem-Overhead bei, entzieht also den Anwendungstasks Rechenzeit. Die dementsprechend wünschenswerten kurzen Scheduler-Laufzeiten stehen aber im Kontrast zu dem gerade bei leistungsfähigen und flexiblen Scheduling-Verfahren erforderlichen Berechnungsaufwand. Da es sich dabei zum großen Teil um parallelisierbare Operationen handelt, bietet es sich an, diese Verfahren durch den Einsatz spezieller Hardware mit mehreren parallel arbeitenden Berechnungseinheiten zu beschleunigen. Abschnitt 2 gibt daher einen Überblick über mögliche Realisierungsvarianten einer Scheduling-Hardware, während sich Abschnitt 3 mit dem Konzept und Abschnitt 4 mit der Beispielimplementation eines Scheduling-Coprozessors beschäftigen. Eine Zusammenfassung der dabei gewonnenen Erkenntnisse und Erfahrungen bildet Abschnitt 5.

2 Realisierungsformen hardwareunterstützten Task-Schedulings

Der Scheduler eines Echtzeit-Betriebssystems muß neben der Kenntnis der Task-Prioritäten bzw. der zu ihrer Berechnung notwendigen Taskparameter auch – zur Durchführung von Taskwechseln - Zugriff auf die Speicherbereiche der einzelnen Tasks und die Prozessor-Register haben. Letzteres ist in der Regel nicht von außen möglich, so daß ein Hardware-Scheduler, sofern er nicht in die CPU integriert ist, dazu immer die Hilfe von Software, die auf dem Prozessor ausgeführt wird, benötigt. Dies stellt also eine obere Grenze dessen dar, was sich an Scheduler-Funktionalität in Hardware auslagern läßt. Die gesamte Task-Verwaltung, also Anlegen und Löschen von Tasks, Taskzustands- und Ressourcenverwaltung usw., läßt sich hingegen zusammen mit dem Scheduler in eine Hardware integrieren, die mit dem oder den Anwendungsprozessoren gemeinsam auf den Speicher zugreift. Dadurch werden die Taskwechselzeiten erheblich verkürzt, da auf dem Anwendungsprozessor nur noch die Prozessorregister gesichert und neu geladen werden müssen, während alle Berechnungen des Schedulers parallel zur Taskabarbeitung erfolgen. Nachteilig bei dieser Realisierungsform ist neben dem relativ großen Hardwareaufwand jedoch die enge Kopplung der Scheduler-Hardware an das Betriebssystem. Dies läßt sich umgehen, wenn das Betriebssystem selbst komplett in Hardware realisiert wird [1],[3] oder Betriebssystem und Hardware-Scheduler gemeinsam entwickelt werden, wie beispielsweise das Betriebssystem SPRING [5] mit dem zugehörigen SPRING Scheduling Coprocessor [2].

Eine Minimalvariante hingegen stellt die Auslagerung lediglich der parallelisierbaren Prioritätenberechnungen in Hardware dar. Hierbei muß die Scheduler-Funktion des Betriebssystems die zur Berechnung der Prioritäten nötigen Taskparameter an die Scheduler-Hardware übergeben und nach erfolgter Berechnung die Taskprioritäten von dort einlesen, um anhand dieser Werte eine Scheduling-Entscheidung treffen zu können. Der relativ einfachen Hardware in dieser Variante steht ein sehr hoher Kommunikationsaufwand zwischen Betriebssystem und Hardware-Scheduler gegenüber. Hierdurch wird die durch die parallele Prioritätenberechnung eingesparte Zeit teilweise wieder aufgebraucht, im Extremfall kann so eine Lösung sogar langsamer als ein reiner Software-Scheduler sein.

Für eine effektive und flexible Hardware-Unterstützung des Schedulings von Echtzeit-Tasks muß also ein eine Lösung gefunden werden, die sich mit einer Vielzahl von Betriebssystemen bei nur geringfügiger Anpassung der Software einsetzen läßt und ein günstiges Verhältnis von Hardware-Beschleunigung zu Kommunikationsaufwand bietet. Das Konzept eines solchen Scheduling-Coprozessors soll im folgenden Abschnitt beschrieben werden.

3 Konzept eines universellen Scheduling-Coprocessors

Der hier vorgestellte Scheduling-Coprozessor erweitert die im vorigen Abschnitt erwähnte Minimallösung in zwei Aspekten, die den Kommunikationsaufwand zwischen Betriebssystem und Coprozessor erheblich verringern. Zum einen werden die Taskparameter und Zustände im Coprozessor gespeichert und stehen somit sofort für die Prioritätenberechnung zur Verfügung. Desweiteren werden die Prioritäten im Coprozessor nicht nur berechnet sondern auch verglichen und lediglich die Nummer des Tasks mit dem, je nach Zuteilungsverfahren, kleinsten bzw. größten Prioritätswert an das Betriebssystem zurückgeliefert. Die Kommunikation zwischen Software und Hardware beschränkt sich zum Zeitpunkt des Schedulings also lediglich auf ein Startsignal für den Coprozessor und einen Lesezugriff auf dessen Ergebnisregister, sobald das Ende der Berechnungen von der Hardware angezeigt wird. Das Betriebssystem hat lediglich die im Coprozessor gespeicherten Taskparameter und –zustände jederzeit auf dem aktuellen Stand zu halten. Dazu müssen diejenigen Betriebssystemfunktionen, die diese Parameter verändern, um ein Kopieren der neuen Werte in den Coprozessor erweitert werden. Eine weitere Vereinfachung tritt ein, wenn man zeitabhängige Taskparameter von der Scheduler-Hardware berechnen läßt. Der Coprozessor benötigt dazu ein periodisches Eingangssignal als Zeitbasis, das in geeigneter Weise mit der Systemzeit synchronisiert ist. Somit bleiben, abgesehen von der Phase der Initialisierung der Tasks im Coprozessor, zur Laufzeit nur die aktualisierten Taskzustände zum Coprozessor zu übertragen.

Aus dem Vorangegangenen lassen sich bereits die zwei Hauptbestandteile des Scheduling-Coprozessors ableiten: die Task-Module und der Vergleicher. Aufgabe der Taskmodule, die jedes für sich einen Echtzeittask im Coprozessor repräsentieren, ist es, den jeweiligen Taskzustand und die zur Berechnung der Priorität notwendigen Parameter zu speichern und zu aktualisieren. Weiterhin wird in diesen Modulen die Task-Priorität berechnet und dem Vergleicher zur Verfügung gestellt. Die Parameter- und Zustandsspeicher innerhalb der Task-Module müssen zu Initialisierung und Aktualisierung von außen über ein Businterface zugänglich sein. Der Vergleicher hat die Aufgabe, die in den Task-Modulen berechneten

Prioritätswerte zu vergleichen und basierend auf dem angewandten Scheduling-Algorithmus aus diesem Vergleich den nächsten auszuführenden Task zu bestimmen. Je nach Realisierungsform der Vergleicher-Komponente ist gegebenenfalls zusätzlich ein Dekoder erforderlich, der aus deren Ergebniswert die Identifikationsnummer des Tasks in einer für das Betriebssystem direkt verwertbaren Form generiert und über das Businterface zur Verfügung stellt. Abbildung 1 zeigt ein Blockschema der hier beschriebenen Scheduling-Coprozessor-Architektur.

4 Beispielimplementierung

Basierend auf dem oben beschriebenen Konzept wurde eine prototypische Implementierung eines dynamisch-prioren Scheduling-Verfahrens durchgeführt. Zum Einsatz kam der Enhanced Least-Laxity-First (ELLF) Algorithmus [4]. Bei diesem Verfahren wird für alle Tasks die Differenz zwischen verbleibender Rechenzeit und Zeit bis zum Erreichen des Endtermins berechnet. Derjenige Task wird als nächster ausgeführt, bei dem diese Differenz am kleinsten ist oder, sofern dies auf mehrere Tasks zutrifft, der von diesen Tasks die früheste Deadline besitzt. In den Task-Modulen müssen folglich neben dem Task-Zustand die Laufzeit und die Deadline gespeichert und bei jedem Zeittakt aktualisiert werden. Weiterhin wird die Differenz aus diesen beiden Werten gebildet und zusammen mit dem Endtermin dem Vergleicher zur Verfügung gestellt. Daneben erfüllen die Task-Module noch eine Überwachungsfunktion, indem sie ein Fehlersignal auslösen, sobald die Differenz zwischen Laufzeit und Endtermin kleiner als Null wird, da zu diesem Zeitpunkt der Task keine Chance mehr hat, seine Deadline einzuhalten. Als Vergleicher wurde bei dieser Implementierung ein serielles Design ausgewählt, bei dem, beginnend mit dem MSB, die Prioritätswerte aller Taskmodule gleichzeitig bitweise eingelesen und verglichen werden. Diese Realisierungsform wurde gewählt, da hierdurch ein großer Vergleicher-Baum vermieden werden konnte. Außerdem ist so die Laufzeit des Vergleichers nur abhängig von der Bitbreite der Prioritätswerte statt von der Anzahl rechenbereiter Tasks, wie dies etwa beim sequentiellen Vergleich jeweils zweier Prioritäten der Fall wäre. Die Bearbeitungszeit des Schedulers ist damit konstant, was für die Scheduling-Analyse in der Entwurfsphase eines Echtzeitsystems von großer Bedeutung ist.

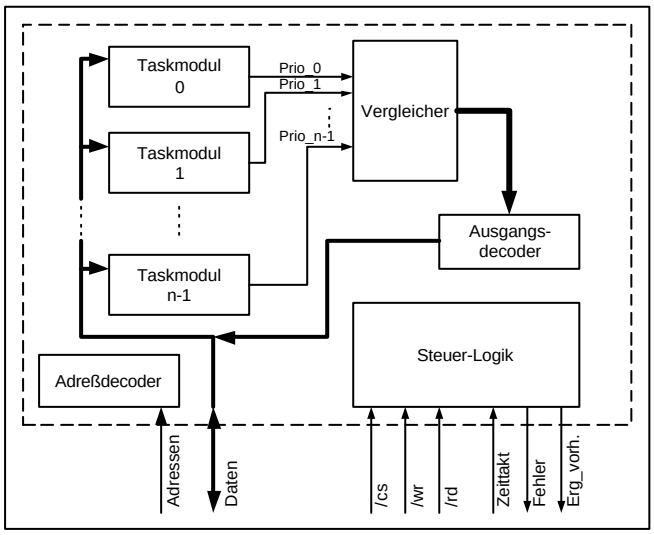

Abbildung 1 Scheduling-Coprozessor

Die hier beschriebene Implementierung erfolgte in einem FPGA vom Typ Virtex-1000 der, je nach Bitbreite der Parameter und Anzahl enthaltener Taskblöcke, zu maximal 11,6% ausgenutzt wurde. Für die Scheduling-Operation ergab sich dabei eine Ausführungszeit von

$$N = 2 \cdot w + 2 \qquad (1)$$

Taktzyklen, wobei w die Bitbreite der Task-Parameter darstellt. Für Schreib- und Lesezugriffe über das 16 Bit breite Businterface werden 3 Taktzyklen benötigt. Tabelle 1 zeigt die Größe und

Geschwindigkeit des Designs für unterschiedliche Bitbreiten und Taskblock-Anzahlen.

Task-Anzahl	Bitbreite	Gatterequivalente	Ausführungszeit bei f_{max}, µs	Max. Taktrate f_{max}, MHz
16	12	70199	0.750	34.650
16	16	76849	1.005	33.838
20	12	78234	0.774	33.575
20	16	86540	1.019	33.340
32	16	116852	1.079	31.488

Tabelle 1: Realisierungsvarianten des Scheduling-Coprozessors (in Virtex-1000 FPGA)

5 Zusammenfassung

Die in den vorangegangenen Abschnitten beschriebene Coprozessor-Architektur für die Umsetzung verschiedener Scheduling-Algorithmen stellt eine Möglichkeit zur schnellen und deterministischen Berechnung von dynamischen Prioritäten dar, wodurch diese effektiv in Echtzeit-Betriebssystemen nutzbar werden. Aufgrund seines universellen Charakters und seines auf die Kernfunktionalität eines jeden prioritätsbasierten Schedulers ausgerichteten Funktionsumfangs eignet sich dieses Design für die Einbindung in verschiedenste Hard- und Software-Umgebungen.

Literatur

[1] Adomat, J. et al.: Real-Time Kernel in Hardware RTU: A Step Towards Deterministic and High-Performance Real-Time Systems. Proceedings of the Euromicro RTS'96 Workshop. IEEE Computer Society, Los Alamitos (1996) 164-168

[2] Burleson, W. et al.: The Spring Scheduling Coprocessor: A Scheduling Accelerator. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 7, No. 1, (March 1999) 38-47

[3] Furunäs, J. et al.: RTU94 – Real Time Unit 1994 – Reference Manual. Technical Report, Mälardalens Real-Time Research Centre (1998)

[4] Hildebrandt, J., Golatowski, F., Timmermann, D.: Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems. Proceedings of the 11th Euromicro Conference on Real Time Systems. IEEE Computer Society, Los Alamitos (1999) 208-215

[5] Stankovic, J., Ramamritham, K.: The spring kernel: A new paradigm for real-time systems. IEEE Transactions on Software Engineering. IEEE Computer Society, Los Alamitos (May 1992) 54-71

Verfasser

Jens Hildebrandt, Prof. Dirk Timmermann
Universität Rostock
Fachbereich Elektrotechnik und Informationstechnik
Institut f. Angewandte Mikroelektronik und Datentechnik
Richard-Wagner-Str. 31
D-18119 Rostock