Integration eines echtzeitfähigen Prozessorarchitekturkonzeptes in eine Rapid-Analysis-Umgebung

Dipl.-Ing. Steffen Dolling, Dipl.-Ing. Frank Golatowski*, Prof. Dr. Dirk Timmermann
Universität Rostock, Fachbereich Elektrotechnik und Informationstechnik
Institut fürAngewandte Mikroelektronik und Datentechnik

Kurzfassung

In diesem Beitrag beschreiben wir eine Architektur für harte eingebettete Echtzeitsysteme. Die Architektur ist flexibel und wurde konzipiert, um eine Trennung zwischen den reinen Verarbeitungs- und den Verwaltungsfunktionen durchzuführen. Das Ziel besteht in der Minimierung des Systemoverheads. Die Grundlage dafür bildet eine Systemstruktur, die aus Standardprozessoren und anwendungsspezifischen Coprozessor-Elementen besteht. Die Architektur ist rekonfigurierbar und ermöglicht die Adaptierung verschiedener Scheduling- und Ressourcenzuweisungsstrategien. Mit der Rekonfigurierbarkeit ergibt sich die Möglichkeit, bei unterschiedlichen Applikationsprofilen aus verschiedenen Zuteilungsstrategien die für die entsprechende Applikation optimale Variante auszuwählen. Darüber hinaus sind die verfügbaren Standardwerkzeuge für den Programmentwurf nutzbar. Die eigentliche Prozeßablauftsteuerung ist flexibel und abhängig von den physikalischen Anforderungen des zu steuernden Echtzeitprozesses.
Zur Unterstützung der Auswahl des geeigneten Verfahrens erfolgt der Einsatz eines Rapid-Analyse-Verfahrens. Dabei erfolgt auf der Basis einer einfachen Spezifikation die Bestimmung der Durchführbarkeit (feasibility) einer Applikation. Interaktiv kann Einfluß auf den Aufbau des zugehörigen Prozeßsatzes genommen werden. Der Spezifikation schließt sich eine Schedulinganalyse an. Dabei ist es möglich, verschiedene Schedulingverfahren als Basis zu wählen, um einen geeigneten Ablauf des Prozeßsatzes zu finden. Ziel ist es, für eine Anwendung einen geeigneten Prozeßsatz aufzustellen und die Durchführbarkeit mit unterschiedlichen Schedulingverfahren zu finden, die teilweise in Software bzw. Hardware realisiert werden.
In diesem Beitrag stellen wir das Systemkonzept und beispielhaft die Architekturkonzeption eines modifizierten Echtzeit-Schedulingprozessors vor.

1 Einleitung

Kommerzielle Echtzeitbetriebssysteme verwenden in der Regel einen prioritätsbasierten preemptiven Scheduler. Dieser Algorithmus ist vielseitig einsetzbar und flexibel und ist durch Eigenschaften wie freie Vergabe von Prioritäten, Änderung der Prioritätenwerte zur Laufzeit und Realisierung von Mechanismen zur Prioritätenvererbung bei kommunizierenden Prozessen charakterisiert. Mit diesem Verfahren ist die Umsetzung bekannter Analyse- Verfahren möglich, z.B. der Rate Monotonic Analyse (RMA), auch wenn damit ein großer Aufwand verbunden ist, da es keine direkte Betriebssystemunterstützung für dieses Verfahren in kommerziellen Echtzeitbetriebssystemen gibt. Bei einer ratenmonotonen Prioritätenzuteilung erfolgt die Prioritätenverteilung an periodische Prozesse entsprechend der Periode der Prozesse: je kürzer die Periode, desto größer ist deren Priorität. Entsprechend [8] ist das ratenmonotone Verfahren optimal unter den Verfahren mit einer festen Prioritätenzuteilung. Optimal bedeutet hierbei, daß wenn es für einen Prozeßsatz einen gültigen Prozeßablauf (Schedule) gibt, bei dem alle Endtermine eingehalten werden können, dann kann dieser Prozeßsatz mit dem ratenmonotonen

Verfahren zugeteilt werden. Als nachteilig erweist sich diese Prioritätenzuweisung, wenn es Prozesse gibt, deren Prioritätenzuweisung von diesem Verfahren abweicht. Eine flexiblere Prioritätenzuweisung ist mit dem Deadline-Monotonic Scheduling (DMS) möglich. Für dieses Verfahren wird in [2] gezeigt, daß es sich um ein optimales Verfahren handelt, und zwar für Systeme, in denen die Deadline einer Anforderung kleiner als die Periode des Prozesses ist.
Die Vergabe von Prioritäten ist ein wichtiger Designpunkt beim Entwurf von Echtzeitsystemen. Wir stellen später ein Werkzeug vor, mit dem eine Unterstützung bei der Priorisierung von Prozeßsätzen gegeben werden kann. Dabei wird garantiert, daß harte Echtzeitbedingungen (Endtermine) eingehalten werden. Alternativ zur Verwendung eines prioritätsbasierten preemptiven Schedulers kann der Einsatz eines Schedulers erfolgen, der eine Auswahl der Prozesse nach den Verfahren „earliest deadline first“ (EDF) bzw. „least laxity“ (LL) unterstützt. Durch den Algorithmus erhält derjenige Prozeß die CPU, dessen Endtermin zeitlich am nächsten liegt bzw. dessen Schlupf am kleinsten ist. Damit sind bei diesen Algorithmen allein harte Termine entscheidend für das Scheduling. Diese lassen sich wiederum auf Prioritäten abbilden, die dynamisch vom System bestimmt werden müssen. Diese dynamischen Scheduling-Verfahren sind optimal unter den dynamischen Verfahren. Mit diesen Verfahren ist theoretisch eine 100%-ige Auslastung möglich. Ein weiterer Vorteil ist, daß die Angabe von Prioritäten nicht im Vordergrund steht, sondern daß der Scheduler in Abhängigkeit von der verfügbaren Rechenzeit und dem nächsten Endtermin eine Schedulingentscheidung trifft. Damit werden die Kontextwechsel minimiert.
Bei der praktischen Umsetzung eines solchen Schedulers muß das System mit einer hohen Granularität arbeiten, um eine hohe Auflösung zu ermöglichen. Damit steigt aber der Overhead, den ein solcher dynamischer Betriebssystemscheduler mit sich bringt, da der Rechenzeitaufwand mit der Auflösung und der Anzahl der Prozesse wächst. Demzufolge ist es sinnvoll, das Scheduling parallel in Hardware auszuführen.
Wir stellen in diesem Beitrag eine mögliche Hardwarearchitektur eines Echtzeit-Coprozessors vor, der als Erweiterung und als Alternative zu bestehenden Architekturen verwendet werden soll. Dieses Coprozessorkonzept ist Bestandteil einer Entwurfsmethodik, die auf das Einhalten harter Zeitbedingungen zielt.
Nachfolgend werden beispielhaft aktuelle Arbeiten beschrieben, in denen ein Entwurf eines Echtzeitsystems vorgenommen wird und bereits sehr frühzeitig die Einhaltung harter Echtzeitbedingungen berücksichtigt werden. Die Methodik zur Erreichung dieses Ziels geht dort vom Scheduling von Prozessen oder Aktivitäten aus.
Altenbernd [1] beschreibt ein zweigeteiltes Prozessorkonzept. Dieses besteht aus einem *schedule-driven* (SD) und einem *OS (operating system)-driven* (OD) Prozessor und soll die Vorteile, die die Systeme im einzelnen bieten, durch die Vorkonfigurierung des eingesetzten Echtzeitbetriebssystems verbinden. Dabei werden harte Echtzeit-Prozesse, deren Laufzeitverhalten als bekannt vorausgesetzt wird, als zyklische Exekutive auf dem SD-Prozessor ausgeführt, mit dem Ziel einen minimalen Overhead zu erreichen. Der OD-Prozessor dient zur Ausführung anderer z.B. sporadischer oder Nicht-Echtzeit-Aktivitäten. Während ein Großteil der Arbeiten auf dem Gebiet von Multiprozessorsystemen sich mit den Problemen der Lastbalancierung auseinandersetzt, besteht das Ziel dieses Konzeptes in der Minimierung der Overheads.
Cavalcante [3] beschreibt ein Codesign-System, mit dem entschieden werden kann, welche Elemente einer Applikation, deren Spezifikation in „Timed-Petri-Nets“ vorliegt, in Hardware und welche in Software zu realisieren sind. Die Partitionierung erfolgt in zwei Schritten. Zuerst wird bestimmt inwieweit jeder Prozeß in einer Softwareimplementierung ohne Ressourcenzwänge durchführbar ist. Danach wird untersucht inwieweit sich Prozesse in die Hardware ausgliedern lassen. Die Auswahl der Prozesse die in Hardware implementiert werden sollen, erfolgt auf der Basis einer Kostenfunktion. Gibt es für einen Prozeßsatz, der vollständig in Hardware realisiert ist, keine gültige Lösung, so ist die Partitionierung fehlgeschlagen. Praktisch realisiert werden Softwareprozesse in C++ und Hardwareprozesse in Hardware C.
Beim Entwurf mit dem Hardware-Software Cosynthesesystem Chinook [4] wird der Einfluß harter Zeitbedingungen auf das Scheduling in reaktiven Echtzeitsystemen untersucht. Chinook verfügt über Methoden, um das Scheduling-Problem in reaktiven Systemen zu lösen. Dazu wird das Scheduling auf Systemebene und auf Geräteebene betrachtet. Damit soll unterbunden werden, daß

hochpriore preemptive Prozesse Geräteprotokolle unterbrechen und in einem inkonsistenten Zustand zurücklassen. Ausgangsbasis des Entwurfs in Chinook ist die Spezifikation des Systems in einer Hardwarebeschreibungssprache und eine Bibliothek von Prozessor- und Gerätespezifikationen.
Die vollständige Neuentwicklung von Prozessorarchitekturen, die speziell auf Echtzeitbelange zugeschnitten sind, ist eine weitere Möglichkeit. So wird in [11] eine echtzeitfähige Prozessorstruktur mit minimalem Befehlssatz vorgestellt. Eine vollständige Neuentwicklung ermöglicht die Konzeption einer Real-Time Prozessorarchitektur und eines Befehlssatzes unter Beachtung der Echtzeitanforderungen. Damit besteht hier ein ganzheitlicher Entwicklungsansatz und damit ein Vorteil gegenüber einer reinen Softwarelösung, die auf bestehende Standard-Hardwareplattformen aufsetzt. Nachteilig wirkt sich aus, daß bei dieser sehr komplexen Aufgabe zahlreiche Elemente konzipiert und implementiert werden müssen, die bereits in Standardelementen vorhanden sind und nicht in unmittelbaren Zusammenhang mit den Real-Time-Bedingungen stehen (z.B. ALU-Entwurf). Ebenso scheint eine vollständige Hardwareimplementierung der Betriebssystemfunktionalität als nicht günstig und zu aufwendig, was sich darin zeigt, daß in den bekannten Entwürfen nur prioritätsbasierte Scheduling-Verfahren als zentrale Elemente eines Real-Time-Kerns integriert sind.
Der gleiche Ausgangspunkt - die funktionelle Abspaltung der eigentlichen Verarbeitung vom Betriebssystemmanagement, die gleichzeitig mit einer physischen Trennung in eine allgemeine CPU und eine Real-Time Unit (RTU) verbunden ist - liegt den Konzepten FASTCHART [13] und FASTHARD [10] zugrunde. Die RTU's sind dabei sehr einfach gehalten, was z.B. in einem Rate-Monotonic-Scheduling-Algorithmus mit festen Prioritäten und dem Ausschluß dynamischer Schedulingverfahren seinen Ausdruck findet.

2 Integration des Konzeptes der Trennung von Verarbeitung und Verwaltung in EVASCAN

Da in einem Echtzeitsystem nicht allein die Bereitstellung eines Ergebnisses entscheidend ist, sondern auch der Zeitpunkt, an dem dieses Ergebnis vorliegt, sollte die Berücksichtigung der Zeitschranken bereits in der frühen Phase des Entwurfs erfolgen. Um diesem Anliegen gerecht zu werden, haben wir das Evaluierungs- und Schedulinganalyse- Tool EVASCAN entwickelt. EVASCAN besteht aus den folgenden Modulen:

1. SPEZIFIKATIONS-EDITOR
2. SCHEDULABILITY-ANALYSER
3. EXECUTION MODUL
4. EVALUIERUNG UND VERGLEICH

Zunächst erfolgt ein Grobentwurf eines Real-Time Systems in einer einfachen Spezifikationsform. Diese ist im derzeitigen Stadium tabellenorientiert. Die zugehörigen Informationen enthalten Aussagen über die Art des Prozesses (periodisch, aperiodisch, sporadisch, deren Laufzeiten, Prioritäten, Verwendung von Synchronisationsmodulen). Zum Erstellen dieser Test- und Spezifikationsfiles dient der *Spezifikations-Editor*. Diese Spezifikation ist ausgerichtet an den physikalischen Anforderungen und überführt diese in eine mögliche Implementationsform. Aus den in den Spezifikationsfiles enthaltenen Informationen läßt sich mit dem *Schedulability-Analyser* bestimmen, inwieweit ein Prozeßsatz, bei Verwendung alternativer Schedulingstrategien und Implementierungsprotokolle, durchführbar ist. Die Durchführbarkeit (Schedulability) bezieht sich dabei auf das Einhalten der Zeitschranken und wird bestimmt durch die Analyse der Auslastung bzw. der Antwortzeiten von Prozeßsätzen. Am Ende der Analyse erfolgt eine Ergebnispräsentation und die graphische Anzeige der Prozeßverläufe. Hier werden die entsprechenden verpaßten Deadlines hervorgehoben. Der Nutzer hat interaktiv die Möglichkeit, in die Prozeßverläufe einzugreifen und die Charakteristik der Prozesse zu verändern. Während im *Schedulability-Analyser* eine Berechnung der Vorgänge erfolgt, kann der Entwickler mit dem *Execution-Modul* das Verhalten

der beschriebenen Spezifikation auf einem realen System untersuchen. Dieses Modul führt eine synthetische Arbeitslast unter realen Bedingungen auf einem System aus. Im *Evaluierungs-Modul* werden die umfangreichen Daten einer Auswertung zugeführt. Neben der Protokollierung der Ergebnisse ist eine Visualisierung der realen Prozeßabläufe auf einem realen Betriebssystem und demzufolge ein Vergleich mit den im Schedulability-Analyser ermittelten Abläufen möglich.
Mit dieser Entwicklungsunterstützung verfolgen wir das Ziel, eine vorhandene Hardware-Plattform und das zugehörige Echtzeitbetriebssystem zu verwenden (z.B. [6]). Das zugehörige Wissen über das einzusetzende System ist dem Entwurfssystem bekannt. Ist die Verwendung einer Standardarchitektur möglich, soll deren Einsatz erfolgen. Erweist sich diese als nicht ausreichend leistungsfähig, kommt ein Scheduling-Coprozessor zum Einsatz. Dabei verfolgen wir nicht das Ziel des vollständigen Neuentwurfs eines Echtzeitprozessors. Mit diesem Scheduling-Coprozessor soll eine Trennung von Verarbeitungs- und Verwaltungseinheit erfolgen und somit der Systemoverhead minimiert werden. Die entsprechend notwendige Neukonzeption der Hardware-Struktur (Bild1) wird im folgenden Abschnitt dargelegt. Diese Architektur ist in den Entwicklungspfad unseres Evaluierungs- und Schedulinganalyse-Tools EVASCAN [7] integriert.

3 Struktur eines eingebetteten Real-Time-Systems für harte Echtzeitanforderungen

Eine komplexe Betriebsmittelverwaltung, in der auch neue Verfahren der Real-Time-Theorie implementiert sind, sowie eine laufzeitoptimierte Verarbeitung der Prozeßdaten, die der Anforderung nach Einhaltung von harten Zeitbedingungen gerecht wird, kann nur durch Parallelität erreicht werden. Ein grundlegendes Architekturmerkmal ist entsprechend die physische Auslagerung verschiedener Funktionen des Betriebssystem - in unserem Falle des Schedulers- in eine Coprozessorkomponente. Diese arbeitet parallel zur Verarbeitungseinheit.
Auf der Verarbeitungseinheit werden einzelne Tasks zur Prozeßdatenverarbeitung sowie die restlichen Betriebssystemfunktionen ausgeführt. Sie ist damit mit der notwendigen Prozeß-E/A sowie einem frei programmierbaren Prozessor einschließlich seiner peripheren Elemente ausgestattet. Ebenso werden das notwendige Interrupt-Hardware-Interface zum technischen Prozeß und das Kommunikations-Interface zur Umgebung bereitgestellt. Die gesamte Struktur ist auf die speziellen Anforderungen der technischen Anwendung ausgerichtet. Der Daten- bzw. Steuersignalaustausch wird über eine enge Kopplung der Verarbeitungseinheit und des Coprozessors erreicht. Diese ist auf der Basis von FPGA-Strukturen realisierbar. Im Koppelmodul werden durch den Coprozessor notwendige Parameter bereitgestellt. Diese können durch die ausführende Task aktualisiert werden. Die Taskumschaltung erfolgt über einen Interrupt der Verarbeitungseinheit und wird durch den Coprozessor initiiert.
Der Coprozessor muß zeitgerecht, das heißt mit geringem Systemoverhead arbeiten und sollte durch den Anwender konfigurierbar sein, was z.B. die Wahl des Scheduling-Verfahrens betrifft. Bei einer Implementierung des gesamten Betriebssystems als auch einzelner Betriebssystemfunktionen als reine Hardware- oder Softwaremodule wird eine ungenügende Performance erreicht. Deshalb ist eine Zerlegung der einzelnen Funktionen in ihre algorithmischen Bestandteile notwendig. Auf dieser Ebene ist nun eine wesentlich günstigere Hardware-Software- Partitionierung möglich. Das heißt, Elemente wie z.B. Entscheidungsverfahren sollten als Softwarelösung auf einem frei programmierbaren Prozessor ausgeführt werden, während arithmetische Prozesse leicht in eine Hardware zu portieren sind. Der Coprozessor besteht somit aus einem Standard-Prozessor einschließlich seiner notwendigen Peripherie (Speicher etc.) und mehreren Hardwareelementen, die auf der Basis von FPGA's realisiert sind. Während die CPU Träger der sogenannten High-Level-Funktionalität in Form der Softwareroutinen ist, werden auf dem FPGA- Feld die Low-Level-Algorithmen mit dem durch parallele Hardware verbundenen Beschleunigungsgewinn implementiert. Dies sind z.B. arithmetische Basisfunktionen für verschiedene Schedulingverfah-

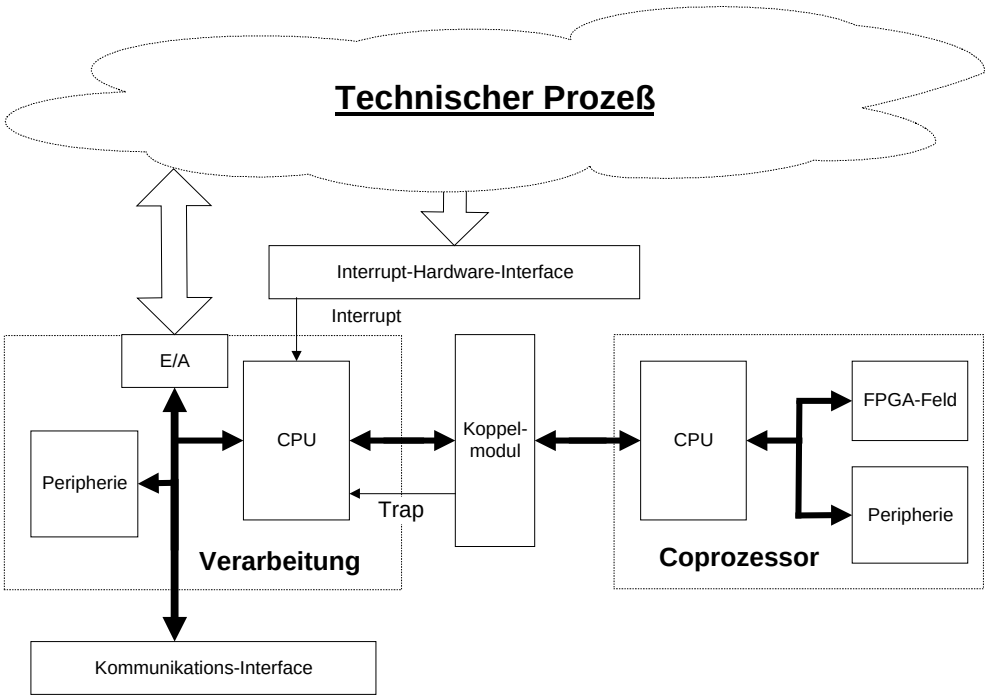

Bild 1: Architektur eines eingebetteten Real-Time-Systems für harte Zeitbedingungen

ren oder hochauflösende Timerstrukturen für Synchronisationsmechanismen. Die Kommunikationsschnittstelle muß einen schnellen und deterministischen Datenaustausch gewährleisten. Entsprechend sind industrielle parallele Bussysteme (z.B. VME, MULTIBUS) oder Feldbussysteme mit einem deterministischen Verhalten (z.B. CAN-Bus) einzusetzen.

4 Konzept für die algorithmische Trennung des Least-Laxity-First-Schedulings

Das Prinzip der Zerlegung einzelner Echtzeitbetriebssystemaufgaben in ihre algorithmischen Bestandteile und die daraus resultierende Hardware-Software-Partitionierung soll konzeptionell am Least-Laxity-First (LLF) -Scheduling erläutert werden. Dieses Auswahlverfahren soll dabei in ein Echtzeitsystem eingebunden sein, das folgenden Voraussetzungen und Bedingungen genügt, die für eine besserer Erklärung teilweise vereinfacht wurden.
Das System ist statisch und ermöglicht die Ausführung eines Prozeßsatzes mit maximal 32 periodischen Tasks. Für diese gilt das allgemeine Zustandsmodell, das die Zustände „Rechnend“, „Bereit“, „Wartend“ und „Suspendiert“ sowie die Übergänge beinhaltet.
Im Zusammenhang mit dem LLF-Scheduling ist jeder periodische Prozeß entsprechend [9] durch verschiedene charakteristische Angaben gekennzeichnet. *TP* ist die Ausführungsperiode (= Ereignisperiode), in der sich das auslösende Ereignis (jeweils am Periodenanfang) und der zugehörige Prozeß wiederholen. *EP* kennzeichnet den Endtermin (= Deadline), bis zu dem nach Eintritt des Ereignisses die Antwort erfolgt sein muß, woraus $EP \le TP$ folgt. Zur Vereinfachung wird $EP = TP$ gesetzt. *RP* ist die Rechenzeit, die ein Prozeß im ungünstigsten Fall zur Ausführung benötigt. Es muß $RP \le EP$ gelten. Die bis zum Zeitpunkt *t* vom Prozeß *P* in der laufenden Periode bereits verbrauchte Rechenzeit wird durch *RP(t)* dargestellt. *EP(t)* ist die Restfrist, das heißt die bis zum nächsten Endtermin verbleibende Zeit, wobei $EP(t) = EP - t$ gilt. Die maximale Zeit, die der Prozeß *P* noch warten darf ohne daß ihm bis zum Endtermin Rechenzeit fehlt, ist der Spielraum (Schlupf, slack, laxity) *SP(t)* des Prozesses *P*. Zu einem Zeitpunkt *t* bekommt der Prozeß *P* den Prozessor, für den unter allen lauffähigen Prozessen $SP(t) = EP(t) - (RP - RP(t))$ minimal ist.

Das System ist durch drei Zeitebenen gekennzeichnet. Der Hardwaretakt *CLK* liegt oberhalb von 40 MHz und dient als Grundtakt der Hardware-Module des LLF-Schedulers. Der Tick *TC* liefert den Takt des LLF-Schedulers. Dieser wird durch den Nutzer festgelegt und sollte eine Größe von 10^3 bis 10^4 *CLK*-Perioden aufweisen. Mit jeder *TC*-Periode erfolgt ein LLF-Auswahlverfahren. Der Realtimetakt *RT*, der ebenfalls durch den Anwender einstellbar ist, liefert die Zeitscheibe zur Ausführung einer Task, wobei *RT* ein ganzzahliges Vielfaches von *TC* sein muß. Verhältnisse zwischen 1:10 und 1:100 erweisen sich dabei als günstig. Lastet die jeweils rechnende Task die Periode *RT* vollständig aus, kann ein Taskwechsel mit dem nächsten *RT*-Takt erfolgen. Wird dagegen eine rechnende Task vor Ablauf der *RT*-Periode beendet, kann ein Taskwechsel mit dem nächsten *TC*-Takt erfolgen. Über die Zeitscheibe *RT* läßt sich damit ein angemessenes Verhältnis zwischen Task-Ausführungszeit und Task-Wechselzeit realisieren, womit sich auch die beim LLF-Verfahren möglichen Trashing-Effekte einschränken lassen. Über den Tick *TC* wird die Feinkörnigkeit des Systems erhöht. Jeder periodischen Task sind durch den Anwender die charakteristischen Größen *EP* und *RP* zuzuordnen. Um ein anforderungsgerechtes Scheduling zu erreichen, müssen diese Zeiten ein Vielfaches des Betriebssystem-Ticks *TC* sein.
Das LLF-Verfahren läßt sich in die folgenden funktionalen Bestandteile partitionieren:

- Bestimmung der charakteristischen Größen *SP(t), EP(t), RP(t)* für jeden Prozeß *P*
- Ermittlung der Prozesse mit den kleinsten Spielräumen
- Auswahl des nächsten auszuführenden Prozesses
- Fehlererkennung (Verpassen einer Deadline)
- Fehlerbehandlung
- Ausführen des Taskwechsels

Die Bestimmung der charakteristischen Größen bezieht sich, wie oben gezeigt, auf einfache arithmetische Gleichungen, die für alle Prozesse parallel auszuführen sind. Demzufolge wird hier eine Hardware-Lösung realisiert, die aus 32 parallel arbeitenden, identischen Modulen besteht. Dabei werden zum Beginn der Ausführungen durch eine Initialisierungs-Routine jedem Modul die durch den Nutzer festgelegten Kenngrößen *EP* und *RP* übergeben. Wie bereits erwähnt, sind die für das Scheduling charakteristischen Größen ein ganzzahliges Vielfaches des Betriebssystem-Ticks *TC*. Dieser wird systemintern mit einer binären Eins bewertet. Daraus resultiert, daß mit jeder Periode von *TC* der Wert *EP(t)* um Eins heruntergezählt werden muß. Durch einen Vergleich wird geprüft, wann *EP(t)* den Wert 0 erreicht hat, was mit dem Beginn einer neuen Taskperiode gleichzusetzen ist. Zu diesem Zeitpunkt muß dann wieder der Ausgangswert *EP* anstelle von *EP(t)* die Berechnungsgrundlage bilden. Der Wert *RP - RP(t)* dagegen wird mit dem Tick *TC* dekrementiert, wenn die Task „rechnend“ ist und bleibt in den anderen Taskzuständen konstant. Ist die Bedingung *RP = RP(t)* erfüllt, wird ein „Endesignal“ generiert. Damit wird die Task bis zum Ende deren Taskperiode aus dem Auswahlverfahren ausgeschlossen. Mit dem Beginn einer neuen Taskperiode erfolgt wieder das Rücksetzen auf *RP*. Der Schlupf *SP(t)* ergibt sich durch die Subtraktion $EP(t) - (RP - RP(t))$. Unterschreitet *SP(t)* den Wert Null, so entspricht dies dem Verpassen einer Deadline und führt zur Generierung eines Fehlersignals. Der entsprechende Datenfluß ist in Abbildung 2 gezeigt. Dieser läßt sich mit drei *n*-Bit-Registern, zwei *n*-Bit-Zustandsmaschinen und einem *n*-Bit-Subtrahierer realisieren. Der Wert *n* bestimmt dabei das maximale Vielfache der Werte *SP(t), EP, EP(t), RP und RP(t)* gegenüber *TC* (z.B.: $n = 10$; $EP_{max} = 1024 * TC$). Die Werte *RP - RP(t)* und *EP(t)* werden durch den jeweiligen Zustand der sequentiellen Maschinen repräsentiert, die mit dem Takt *TC* getrieben werden.
Die Bestimmung der Prozesse mit den kleinsten Spielräumen erfolgt durch ein iteratives Verfahren. Dafür werden zwei 32-Bit-Register *A* und *B* benötigt, deren Bitpositionen jeweils mit einem der oben beschriebenen Taskmodule über je eine Signalleitung korrespondieren. Am Anfang einer *TC*-Periode werden diese Register initialisiert. Register *B* wird dabei auf Null gesetzt, während Register *A* für alle bereiten Tasks bzw. die aktive Task auf den jeweiligen Bitpositionen mit Null

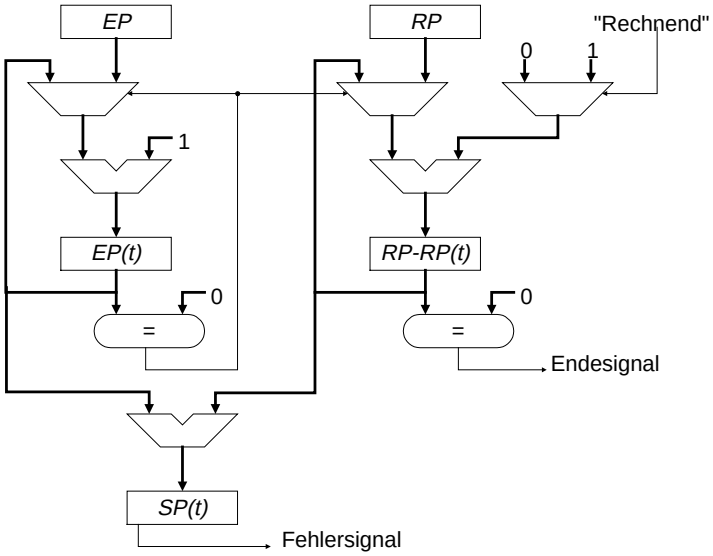

Bild 2: Bestimmung von *EP(t)* und *SP(t)* für eine Task

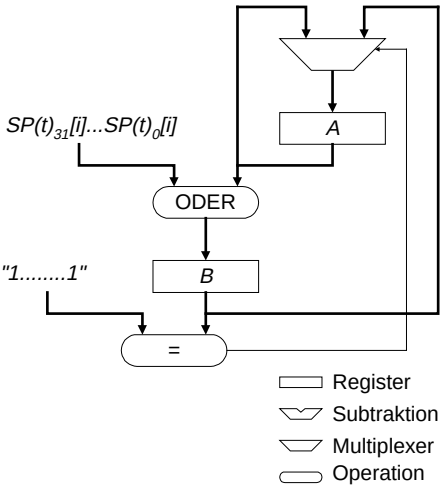

Bild 3: Bestimmung von $SP(t)_{min}$

und für alle nicht bereiten Tasks auf den jeweiligen Bitpositionen mit Eins initialisiert wird. Anschließend werden *n* identische Iterationen entsprechend dem Datenfluß in Bild 3 durchlaufen. Im Iterationsschritt *i* werden dabei die 32 gleichwertigen Bitpostionen $SP(t)_{31}[i]...SP(t)_0[i]$ (für $i = 0$ die MSB's) mit den jeweils korrespondierenden Bitstellen des Registers *A* ODER-verknüpft. Das Ergebnis der Operation wird in Register *B* zwischengespeichert und durch einen Vergleich wird geprüft, ob alle Bitpositionen mit Eins belegt sind. In diesem Fall wird der Inhalt des Registers *A* beim Übergang vom Iterationsschritt *i* zum Schritt *i+1* nicht geändert. Andernfalls wird der Inhalt von *B* in *A* übernommen. Nach *n* Rekursionen ist die Task bzw. sind die Tasks mit dem kleinsten Schlupf *SP(t)* im Register *A* durch eine Null gekennzeichnet.
Aus den Prozessen mit den kleinsten Spielräumen wird nun derjenige ausgewählt, der als nächstes zur Ausführung kommt. In diesen Entscheidungsprozeß sind verschiedene Kriterien, die vom jeweiligen Systemzustand oder den Taskzuständen abhängig sind, einbezogen (z.B. zwei oder mehrere Tasks mit dem gleichen numerischen Wert *SP(t)*). Die Ausführung dieses Algorithmusses wird durch eine Software-Routine auf dem Mikrocontroller des Coprozessors realisiert.
Ein Vorteil des LLF-Verfahrens ist das frühzeitige Erkennen von Deadlines, die verpaßt werden. Die Fehlerbehandlung, die schon vor dem Eintreffen des Ereignisses (Überschreiten der Deadline) beginnen kann, wird wiederum durch einen Software-Algorithmus ausgeführt. Somit kann der Nutzer entsprechend der Anwendung, des Prozesses usw., flexibel die Reaktion auf die Verletzung von Zeitbedingungen programmieren.
Der Taskwechsel wird ebenfalls über den Mikrocontroller des Coprozessors initiiert. Dies erfolgt nach der Auswertung der Rückgabeparameter der gerade rechnenden Task sowie den beschriebenen Taskauswahlverfahren zu Beginn einer Periode von *RT* oder *TC*. Ausgelöst wird die Taskumschaltung durch einen Interrupt der Verarbeitungs-CPU (siehe Bild 1).

5 Zusammenfassung

In diesem Beitrag haben wir die Konzeption eines Echtzeit-Schedulingprozessors beschrieben, der über eine hohe Granularität verfügt und der das dynamische Scheduling in Echtzeitsystemen unterstützt. Diese Architektur ist integriert in den Entwicklungspfad beim Entwurf von Echtzeitsystemen. Mit dem Coprozessorkonzept ist es möglich, aufwendige Schedulingalgorithmen von der verarbeitenden CPU fernzuhalten. Somit lassen sich verschiedene Protokolle implementieren, deren Implementierung in einem Betriebssystemscheduler zu aufwendig und mit einem großen Overhead verbunden ist. Im weiteren Verlauf der Arbeiten sollen dynamische Serveralgorithmen

für die Anwendung in dynamischen Echtzeit-Schedulern untersucht werden [12], mit denen aperiodische Aktivitäten mit harten und weichen Endterminen berücksichtigt werden.
Der eigentliche Systementwurf und die Evaluierung auf Durchführbarkeit (feasibility) eines Prozeßsatzes erfolgt mit Unterstützung von EVASCAN. Beim weiteren Ausbau von EVASCAN als Entwicklungssystem soll die Verwendung alternativer Eingabespezifikationen erfolgen. Für dynamisches Scheduling in harten Echtzeitsystemen wird der Einsatz des Ereignisstrommodells als Beschreibungsform untersucht [5].

Literatur

[1] P. Altenbernd, „Timing Analysis, Scheduling, and Allocation of Periodic Hard Real-Time Tasks," Dissertation, Universität-GH Paderborn, 1996

[2] N. Audsley, A. Burns, M. Richardson, A. Wellings, „Hard real-Time Scheduling: The Deadline Monotonic Approach", Proceedings 8th Workshop on Real-Time Operating Systems and Software, 1991

[3] S. V. Cavalcante, „A Hardware-Software Codesign Environment for Real-Time Embedded Applications", EMSYS96, OMI 6th Annual Conference, Berlin, In: Müller-Schloer et al. (Eds.) Embedded Microprocessor Systems, IOS Press, S.253-362, 1996

[4] P. Chou, E.A. Walkup, G. Borriello, „Scheduling for Reactive Real-Time Systems", IEEE Micro, August 1994, S.37-47.

[5] G. Färber, R. Huber, H. Thielen, "Wirklichkeitsnahe Prozeßzeit-Modellierung zur Konstruktion effizienter Realzeitsysteme", Abschlußbericht des DFG-Forschungsvorhabens Fa 109/10-1, Lehrstuhl für Prozeßrechner, Technische Universität München, 1997.

[6] Gallmeister, B. O., "Programming for the Real World: POSIX.4", O´Reilly & Associates, Inc., 1995

[7] F. Golatowski, D. Timmermann, „EVASCAN: Methodik zur Evaluierung, Analyse und zum Test von Echtzeitsystemen und Echtzeitbetriebssystemen", 14. ITG/GI-Fachtagung ARCS´97 Architektur von Rechensystemen, Rostock, 1997

[8] C.L. Liu, J.W. Layland, „Scheduling Algorithms for Hard Real-Time Environments", Journal of the ACM, vol. 20, no. 1, S. 46-61, 1973

[9] R. Kern, „Prozeßauswahl und Ablaufplanung in Echtzeit-Systemen", Elektronik, 14, 1992, S. 26 – 32

[10] L. Lindh, „A Real-Time Kernel implemented in one chip", Fifth Euromicro Workshop on Real-Time Systems, Tagungsband S. 251 - 254, Oula, 1993

[11] H.-P. Meske, „Eine echtzeitfähige Prozessorarchitektur mit minimalem Befehlssatz", Tagung Echtzeit '94, Tagungsband S. 49 - 57, Hamburg, 1994

[12] M. Spuri, G. Buttazzo, „Scheduling Aperiodic Tasks in Dynamic Priority Systems", The Journal of Real-Time Systems 10(2), S. 179-210,1996

[13] F. Stanischewski: „FASTCHART - Ein Echtzeitprozessorkonzept", Tagung Echtzeit '95, Tagungsband S. 139 - 144, Karlsruhe, 1995