# Prozessoren tauschen Nachrichten über Dual-ported-RAMs aus

## Dr. R. Brause beschreibt den fehlertoleranten Attempto

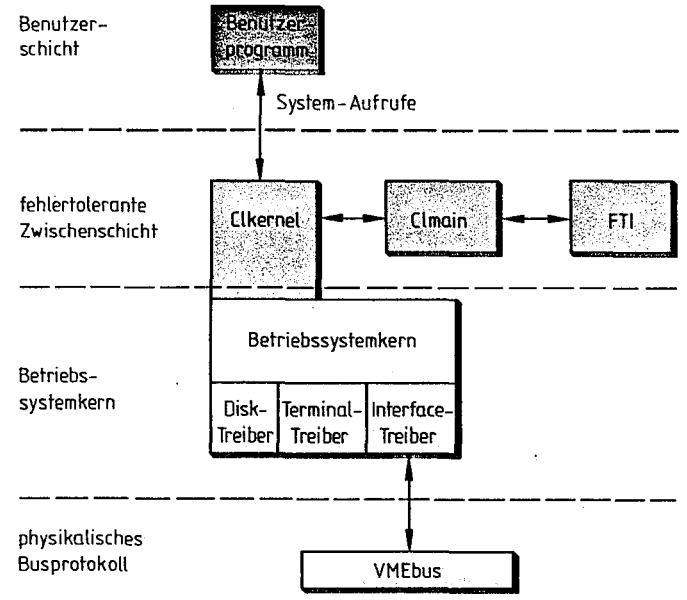
*Dr. R. Brause ist am Lehrstuhl von Prof. Dr. M. Dal Chin, Fachbereich Informatik (20), an der Johann Wolfgang Goethe-Universität in Frankfurt als wissenschaftlicher Mitarbeiter beschäftigt.*

Die Notwendigkeit fehlertoleranter Rechner ist heute unumstritten. Die wenigen, bis jetzt vorhandenen oder neu erscheinenden oder angekündigten, industriellen Produkte weisen aber aus unserer Sicht noch immer erhebliche Mängel auf:

- O Die Fehlertoleranzmechanismen müssen vom Anwender in seinen Programmen berücksichtigt werden (kostspielige Programmänderungen!); Standardsoftware von dritter Seite ist deshalb kaum vorhanden.
- O Die erarbeitete Lösung ist meist sehr speziell, nicht modular und damit auch nicht portierbar.
- O Die benutzte Hard- und Software entspricht keinem industriellen Standard.
- O Die Fehlertoleranz umfaßt meist nur Teile des Gesamtsystems.
- O Das System ist (relativ zur nicht-fehlertoleranten Version) zu teuer.

Im Gegensatz zu den Industrielabors war unsere Arbeitsgruppe in der Situation der Anwender. Als wir begannen, zur Erprobung unserer Fehlertoleranz-Konzepte den fehlertoleranten Rechner Attempto zu entwickeln, entschieden wir uns zu folgenden Vorgaben:

(1) Die Fehlertoleranzmechanismen sollen für den Anwender transparent sein, d. h. alle, auch absolut gebundene Programme sollten ohne Änderung fehlertolerant ablauffähig sein.




**Aufbau der Software:** Die fehlertolerante Zwischenschicht (FTL) besteht aus der Kommunikationsinstanz (CIkernel und CImain) sowie der Fehlertoleranzinstanz (FTI)

(2) Der Anwender kann entscheiden, wie er die Rechenleistung zwischen Fehlertoleranz und Durchsatz aufteilt.

(3) Die Fehlertoleranzmechanismen müssen modular und hardwareunabhängig gewählt werden.

(4) Die für die Fehlertoleranzmaßnahmen nötige Zusatzsoftware soll portabel sein.

(5) Die Zusatzsoftware ist modular und strukturiert in einer höheren Programmiersprache zu schreiben.

(6) Die Hardware-Komponenten müssen Standardteile, keine Spezialanfertigungen sein.

(7) Die Software (Betriebssystem, Utilities) muß Standard sein.

(8) Die Fehlertoleranz erstreckt sich bis auf die Ein- und Ausgabeleitungen.

Mit diesen Vorgaben entschieden wir uns, als Hardware-Standard für Einplatinenrechner an einem Multi-Master-Bus (z. Z. VMEbus) zu verwenden, auf denen Unix als Betriebssystem läuft.

Sämtliche zusätzliche nötige Software ist, soweit unbedingt für den Unix-Kern nötig, in „C" geschrieben; der Hauptteil allerdings in der höheren, fehlervermeidenden Programmiersprache Modula-2 [9]. Fehlertoleranz wird dadurch erreicht, daß derselbe Benutzerjob auf mehreren Rechnern ausgeführt wird und hinterher die Ausgaben verglichen werden (Fehlermaskierung). Danach wird die durch Vergleichstest ermittelte, korrekte Ausgabe von einem Einplatinenrechner unter Kontrolle der anderen ausgegeben.

(*Fortsetzung auf S. 65*)

## Das Betriebssystem des Rechners Attempto

Das Betriebssystem von Attempto besteht aus identischen, autonomen, lokalen Betriebssystemen, deren Kern von einem Unix-ähnlichen Einprozessor-Betriebssystem mit Multitask-Funktionen gebildet wird. Darauf baut eine Zwischenschicht auf, welche die Fehlertoleranzeigenschaften bereitstellt und für das Benutzerprogramm transparent ist. Vom Betriebssystemkern wird sie als normaler Prozeß mit hoher Priorität behandelt.

Mit der Einführung der Zwischenschicht ist es nun möglich, auch der schwierigen ersten Vorgabe zu genügen. An das Betriebssystem wird für die Inter-Prozessor-Koordination nur die Anforderung gestellt, daß Nachrichten in der gleichen zeitlichen Reihenfolge an die Fehlertoleranzschicht weitergegeben werden, in der sie an das Betriebssystem von den Geräte-Händlern übergeben werden.

Die in Modula-2 programmierte Fehlertoleranzschicht (FTL) besteht aus der Kommunikationsinstanz (CI) und der Fehlertoleranzinstanz (FTI). Wenn ein Benutzerprogramm eine Dienstleistung des Systems anfordert (z. B. Ein-/Ausgabe), so geschieht dies durch einen System-Aufruf an das zugrunde liegende Betriebssystem. Die Kommunikationsinstanz bewirkt eine Umleitung der System-Calls zu der eingeschobenen Zwischenschicht. Die CI teilt sich auf in einen dem Betriebssystemkern angehefteten Teil CIkernel, der die System-Calls in Nachrichten umformt und dem entsprechenden Teil CImain, der diese Nachrichten liest und der FTI zur fehlertoleranten Ausführung der Aktionen des Benutzerprogramms (lesen, schreiben usw.) übergibt.

Die lokale FTI ist verantwortlich für folgende Betriebssystemaufgaben:

- O lokale Interpretation der Benutzerkommandos an Attempto
- O Management der systemweiten Resourcen (z. B. Terminals)
- O Kontrolle der Daten von und zu den Ein- und Ausgabegeräten
- O systemweite Konsistenz der Systemtafeln.

Außerdem werden von der FTI folgende Funktionen für Fehlertoleranzzwekke bereitgestellt:

- O dezentrales Dispatching der Benutzerprogramme
- O Vergleich der Ausgabedaten der lokalen Kopien eines Benutzerpro- ▷

## Industrielle Ein-/Ausgabe

VMEbus-Systeme werden überwiegend in industriellen Anwendungen eingesetzt. Besonders wichtig sind in diesem Zusammenhang die Ein-/Ausgabe-Möglichkeiten zum industriellen Prozeß. In Heft 4 (Erscheinungstermin: 1.8.1988) soll schwerpunktmäßig zu diesem Themen berichtet werden. Die Redaktion sucht vor allem noch Applikationsberichte, in denen die Anwender kritisch über ihre Erfahrungen informieren. Redaktionskontakt: Holger Zeltwanger, Postfach 910341, 8500 Nürnberg 91, Tel. (0911) 335908.

gramms und anschließende Diagnose bei Nichtübereinstimmung
- O Überwachung bei der Ausgabe der Daten
- O Fehlerinterpretation und -behandlung.

Aus Gründen der Fehlertoleranz besitzt jede FTI dazu ihre eigene Systemtafel, die alle aktuellen Angaben über Kollegen zur Programmausführung, anstehende Benutzerprogramme, Zustand der globalen Resourcen sowie aller Einplatinenrechner im System enthält.

Der Anwender sieht den Arbeitsplatzrechner als Einbenutzer-/Multitasking-System; die Realisierung als Mehrprozessor-System ist ihm verborgen. Das System bietet die Möglichkeit, die Fehlertoleranzeigenschaften im Gegensatz zu den in [3], [8] erwähnten Rechnern individuell für jede Anwendung zu wählen. Beispielsweise bedeutet ein Eintippen von "MEINPROGRAMM #2#", daß "MEINPROGRAMM" mit dem Fehlertoleranzgrad t=2 ausgeführt werden soll. Dies bedeutet, daß bei der Ausführung des Programms maximal „t" Defekte (und damit transiente oder permanente Fehler auf maximal „t" Einplatinencomputern) toleriert werden in dem Sinne, daß keine fehlerhaften Ergebnisse ausgegeben werden. Der Benutzer kann dem System soviele Programme gleichzeitig ausführen lassen, wie es die Systemkapazität erlaubt: mehrere Programme mit geringem oder wenige mit hohem Fehlertoleranzgrad.

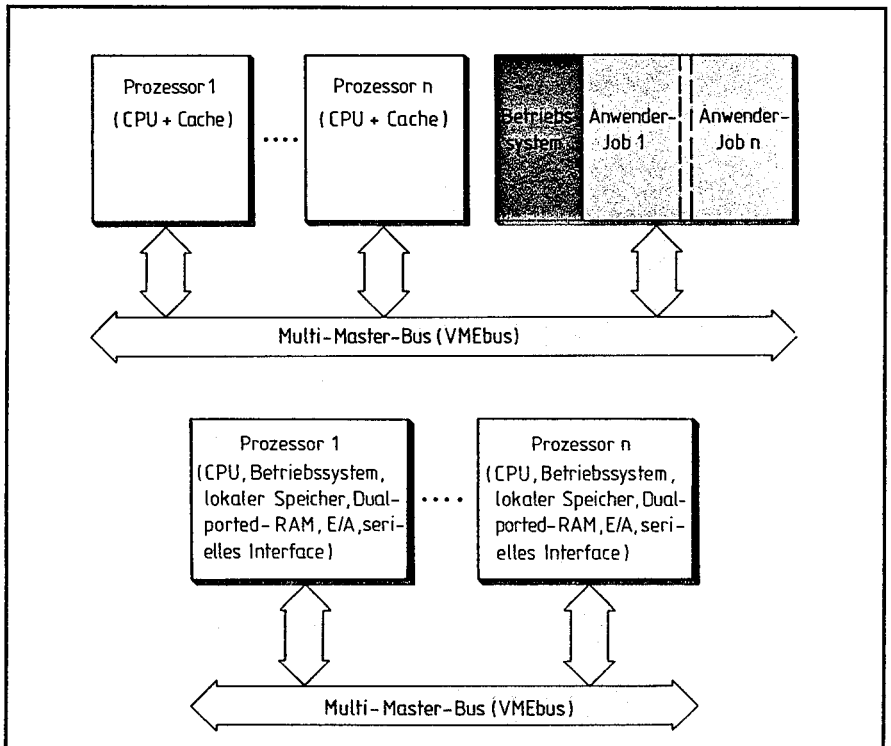
Hat der Benutzer spezifiziert, welches Programm und, einschließlich des Fehlertoleranzgrades, wie viele Kopien von dem Programm von verschiedenen Prozessoren ausgeführt werden sollen, so muß nun der Auftrag koordiniert ausgeführt werden. Das „Dispatching" wird nicht vorher deterministisch festgelegt wie beim Pre-Scheduling bei [8] und [3], sondern nach dem Prinzip der Anziehung (attraction-principle) während der Laufzeit durchgeführt. Im Gegensatz zu der zentralen Hardware-Version dieses Prinzips in Pluribus [6] verwenden wir eine dezentrale Software-Version: Jeder freie Prozessor bewirbt sich um das in seiner Systemtafel „noch zu bearbeiten" gekennzeichnete in der Eingabereihenfolge nächste Benutzerprogramm. Empfängt er eine Bewerbung für ein solches Programm, so trägt er den Bewerber dafür ein. Ist er selbst der Bewerber, so beginnt er mit der Ausführung des Programms. Alle Bewerbungen für bereits „besetzte" Programme werden ignoriert, außer der eigenen. Diese führt zur Bewerbung um das nächste freie Programm.

Vor jeder Ausgabeoperation vergleichen alle Prozessoren, die dasselbe Programm bearbeiten, die zur Ausgabe anstehenden Daten. Im Unterschied zu Standard-Mehrheitsentscheidungen muß bei unserer Fehlerdiagnose nicht jeder Subrechner alle Ergebnisse miteinander vergleichen. Stattdessen ist jedem Fehlertoleranzgrad ein sogenannter Testgraf zugeordnet, durch den festgelegt ist, welche Ausgaben zu vergleichen sind. Die Diagnose basiert auf den Ergebnissen dieser Vergleiche. Damit gewinnt jeder Einplatinen-Computer Informationen über den Zustand der Kollegen. Die eigentliche Ausgabeoperation wird daraufhin von dem Prozessor durchgeführt, der als erster genügend Bestätigung erhalten hat. Die anderen intakten Kollegen überwachen diese Ausgabe. Die Mechanismen zur Diagnose und Festlegung des ausgebenden Prozessors sind detaillierter in [1],[2] beschrieben. Da wir primär transiente Fehler annehmen, ist es nicht ratsam, bei jedem auftretenden Fehler den Einplatinenrechner auszutauschen. Stattdessen führt jeder eine eigene Fehlerfrequenzliste, in der auftretende Nichtübereinstimmungen der Resultate für spätere Auswertungen notiert werden.

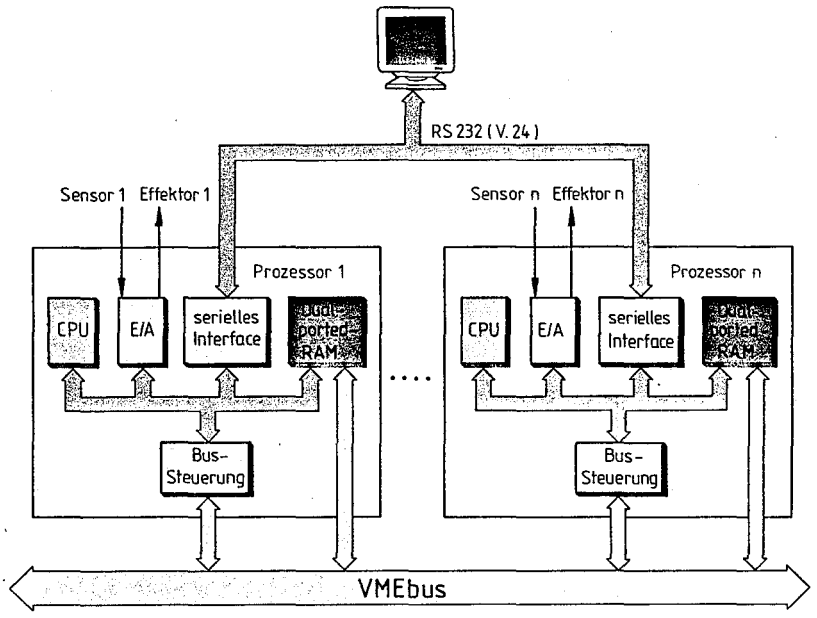
## Die Kopplung der Prozessoren

Bei dem Entwurf des Mehrprozessorsystems stellte sich die Frage, ob die Prozessoren fest oder lose miteinander gekoppelt werden sollen. Bei der festen Kopplung der Prozessoren arbeiten alle Prozessoren mit einem gemeinsamen Speicherbereich, in dem das Betriebssystem und die Benutzerprogramme liegen. Damit der Systembus nicht die Rechenleistung empfindlich begrenzt, ist zusätzlich auf jedem Prozessor-Board ein schneller Cache enthalten. Diese Lösung hat folgende Nachteile:
- O bei Ausfall des Systembusses fällt das Gesamtsystem aus,
- O defekte Prozessoren können die Daten der anderen korumpieren und damit das System ebenfalls wertlos machen,
- O Standard-Programme (Lader, Debugger, usw.) müssen umgeschrieben werden, um Dateninkonsistenz zwischen Cache und Hauptspeicher zu vermeiden.

In dem lose gekoppelten System sind diese Nachteile vermieden. Der Systembus wird nur noch zur Kommunikation (Austausch von Nachrichten zwischen den Prozessoren) verwendet; fällt er aus, so kann die Kommunikation auch




**Feste und lose Koppelung:** Bei festgekoppelten Mehrprozessorsystemen ist ein gemeinsamer Speicher vorhanden, bei losen Koppelungen wird der Systembus nur zum Nachrichtenaustausch verwendet




**Der Rechner Attempto:** Das fehlertolerierende Mehrprozessorsystem hat zwar nur einen zentralen Bus, den VMEbus, da aber die Sensoren und Effektoren redundant ausgelegt sind und direkt an die Prozessorkarten geführt werden, kann das System auch bei Ausfall des VMEbus korrekt weiterarbeiten

über einen anderen Kommunikationsweg (z. B. den seriellen Bus in VMEbus-Systemen) abgewickelt werden und dem Benutzer der Ausfall mitgeteilt werden. Das Betriebssystem und das jeweilige Anwenderprogramm sind für jeden Prozessor direkt zugreifbar, ohne damit die anderen Prozessoren (Rechnerleistung!) oder seine Daten zu tangieren.

Der Nachteil dieser Lösung ist allerdings auch evident: Wurden vorher zur Koordination der Prozessoren bei globalen Ressourcen (z. B. E/A-Einheiten) Monitor-Konstrukte im gemeinsamen Speicher eingesetzt, so geschieht nun die Koordination durch Nachrichtenaustausch. Dies benötigt mehr Rechnerleistung. Da sich aber Kommunikationsprotokolle sehr gut modular von den normalen Rechneraktivitäten abtrennen lassen, kann man den Nachteil relativ einfach durch einen besonderen Kommunikationsprozessor eliminieren.

Bei der üblichen Lösung wird die Eingabe von einem Interface empfangen und dann aktiv (DMA) oder passiv an die Prozessoren verteilt. Damit hängt aber wieder die Funktion des Gesamtsystems nur von einer einzigen Komponente ab. Das Gleiche gilt auch für die Ausgabe. Um dieses Problem zu vermeiden, werden im Attempto-Rechner die Eingabedaten an jeden einzelnen Subrechner herangeführt. In diesem Fall kommen die Eingabedaten vom Benutzerterminal, dessen serielle Schnittstelle parallel mit allen seriellen Schnittstellen der Einplatinenrechner verbunden ist.

Bei besonders fehlertoleranten Prozeßsystemen ist ein derartiger singulärer Datenbus sicher bedenklich. Bei der abgebildeten Alternative kommen die Daten vom gleichen Prozeß, aber von verschiedenen Sensoren, und wirken mit getrennten Leitungen wieder auf unterschiedliche Effektoren. Überlagern sich diese Stellglieder nur in der Wirkung, so kann für die Gesamtregelung auch der Ausfall eines Effektors toleriert werden.

## Die Inter-Prozessor-Koordination

Um einen Systemzusammenbruch bei Defekt des VMEbus oder des gemeinsamen Speichers zu verhindern, hat in Attempto jedes Prozessor-Board seine eigene Systemtafel, die durch Nachrichtenaustausch mit denen der anderen Prozessoren konsistent gehalten wird. Dabei tritt aber das Problem auf, daß die Verarbeitung einer Nachricht selbst wieder vom Zustand der Systemtafel abhängt; so kann beispielsweise ein Prozessor nur dann für einen Job in die Systemtafel eingetragen werden, wenn die Kollegenliste noch nicht vollständig ist. Eine Rückfrage führt in diesem Fall zu erhöhter Kommunikation und belastet das System zusätzlich.

Wie kann man die Systemtafeln trotzdem auf allen Subsystemen konsistent halten? Anstatt nur eine, zentrale, fehleranfällige Systemtafel zu schaffen, wählten wir eine andere Lösung, nämlich die zeitliche Reihenfolge der Nachrichten zur Veränderung der Systemtafeln bei allen Prozessoren gleich zu halten. Damit sind bei gleichem initialem Ausgangszustand die Systemtafeln auch ohne Rückantwort jederzeit konsistent. Dieses Grundprinzip mußte nun möglichst effektiv mit der vorhandenen Kommunikationshardware, dem VMEbus, implementiert werden.

Eine Möglichkeit, die zeitliche Reihenfolge der Nachrichten auf allen Prozessoren identisch zu halten, bietet ein Broadcast-Bus. Bei dieser Lösung wird jede Nachricht von allen CPUs im System empfangen. Da der VMEbus von verschiedenen Prozessoren nur nacheinander benutzt werden kann, ist damit eine eindeutige Reihenfolge der Nachrichten gegeben. Das Senden einer Nachricht entspricht dabei einer atomaren, ununterbrechbaren Aktion auf dem Broadcast-Bus.

Vom Standpunkt der Fehlertoleranz ist ein solcher Broadcast-Bus aber nicht unbedenklich. Es ist dabei nicht möglich, zwischen zwei Prozessoren Nachrichten auszutauschen, ohne daß ein im System befindlicher, defekter Prozessor diese lesen oder verfälschen kann. Es wäre deshalb besser, ein Kommunikationssystem zu verwenden, bei dem der Sender nur dem eine Nachricht übermittelt, der sie auch erhalten soll.

Da außerdem der VMEbus für eine Hardware-Broadcast-Eigenschaft eine zusätzliche, nicht-triviale Hardware-Änderung auf jeder CPU-Platine benötigt, entschieden wir uns im Einklang mit der Vorgabe, nur Standard-Hardware zu verwenden, für ein anderes physikalisches Kommunikationsprotokoll. Dazu benutzen wir die Eindeutigkeit der Adreßkennung auf dem VMEbus.

Die Interprozessor-Synchronisation und Kommunikation basiert in Attempto auf einem Nachrichtenaustausch, der über „ports" (spezielle Speicherbereiche des Dual-ported-RAM) abgewikkelt wird. Auf jeder CPU-Karte existiert ein „port" für den Empfang von Nachrichten von jedem Prozessor des Systems, auch für sich selbst. Auf die Bedeutung des „ports" für Nachrichten an sich selbst („pseudo-port") zur Erhaltung der Reihenfolge der Nachrichten sei besonders hingewiesen. ▷

Der sichere Austausch von Daten wird somit durch eine höhere Busbelastung erkauft – nämlich dann, wenn eine Nachricht an mehrere zu versenden ist. Da jeder Prozessor sofort feststellen kann, ob er eine neue Nachricht erhalten hat, entfällt dafür im Unterschied zu einem Broadcast-Bus die Notwendigkeit, jede einzelne Nachricht zu lesen, um den Empfänger festzustellen.

Allerdings stellt sich ein weiteres Problem: wenn eine Nachricht von einem Prozessor an alle anderen geht, darf das Versenden einer Nachricht an mehrere Empfänger nicht durch einen anderen Prozessor unterbrochen werden, da sonst die Empfangsreihenfolge der beiden Nachrichten auf allen Prozessoren nicht mehr identisch ist. Eine übliche, aber nicht fehlertolerante Möglichkeit, dieses Problem zu lösen, ist die Sperrung des VMEbus während der gesamten Sendedauer einer Nachricht an alle Empfänger (ein defekter Prozessor könnte damit den Bus dauerhaft blokkieren). Stattdessen entschieden wir uns für folgende Lösung: Jedem Prozessor am Kommunikationsbus ist eine Interrupt-Signalleitung zugeordnet, die er aktiviert, nachdem er eine Nachricht zu allen Empfängern gesendet hat.

Dabei wird das folgende logische Protokoll einer asynchronen Nachrichtenübertragung benutzt: Der Interrupt aktiviert die Port-Handler aller Prozessoren. Falls eine CPU-Platine zu den Empfängern einer Nachricht gehört, leitet dessen Handler sie an das Betriebssystem weiter, sendet als ACK-Signal eine Nachricht mit der Signatur der empfangenen Nachricht an deren Absender und löscht den „Header" der Nachricht zum Zeichen, sie gelesen zu haben. Bekommt der Absender keine Empfangsbestätigung (time-out) oder eine solche mit falscher Signatur, so versucht er es mehrmals. Bei Mißerfolg erkennt er auf Bus-Prozessorfehler und kommuniziert nicht mehr mit der betreffenden Einheit (über diesen Bus).

Die Reihenfolge, in der die Nachrichten vom lokalen Betriebssystem registriert werden, ist somit nicht vom Eintreffen der Nachrichten auf den verschiedenen Prozessoren bestimmt, sondern von der Reihenfolge der dazugehörigen Interrupts. Der „Broadcast" wird also hier durch ein überall gleichzeitig anliegendes Interruptsignal ersetzt. Im Vergleich zu den üblichen Broadcast-Systemen ist für den Nachrichtenaustausch eine Interruptleitung für jede CPU-Karte nötig.

Die Forderung, auf allen Subrechnern die gleiche zeitliche Reihenfolge der Nachrichten zu garantieren, wird damit zu der Forderung, auf allen Prozessoren die Bearbeitung der Interrupts in der gleichen Reihenfolge sicherzustellen. Im folgenden werden vier Probleme geschildert, die bei der Realisierung dieser Forderung auftreten und Lösungsmöglichkeiten angegeben.

*Fehlende Interruptsequenz-Hardware*

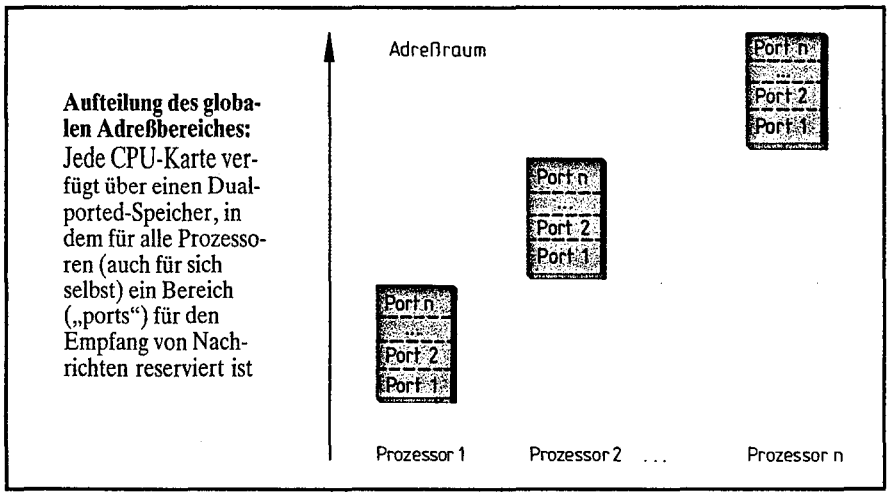
Jede Interrupt-Service-Routine (ISR) benötigt eine gewisse Mindestdauer, z. B. die Zeit zum Umladen der Register und Initialisieren der Routine. Erfolgen innerhalb dieser Zeitspanne erneut Interrupts, so können diese nicht sofort bearbeitet werden, sondern werden in einem Interruptregister als Ereignis gespeichert. Diese Register erlauben aber keine Aussage mehr über das zeitliche Eintreffen der Ereignisse.

Würde man statt der konventionellen nun spezielle Hardware entwickeln, die auch die zeitliche Reihenfolge der Interrupt-Ereignisse beachtet (Aufbau einer FIFO), so wäre es trotzdem nicht möglich, die Unterschiede der fertigungsmäßig und thermisch bedingten Offset-Spannungen der Interrupteingänge zu beseitigen. Dies bedeutet für zwei gleichzeitig generierte Interrupts, daß sie je nach Offset früher oder später registriert und damit unterschiedlich eingeordnet werden.

Das Problem der identischen zeitlichen Reihenfolge läßt sich stattdessen mit Hilfe der Standard-Hardware befriedigend lösen, indem man den Interruptleitungen der Kommunikation Prioritäten zuordnet. Dies induziert eine feste, andere Ordnung der Abarbeitung der Interrupts als die der zeitlichen Reihenfolge. Da diese Ordnung auf allen Prozessoren identisch ist, ist die Konsistenz der Systemtafeln gewahrt. Durch den Nachrichten-Rückantwortmechanismus sind zwei Interrupts in kurzem Abstand für denselben Empfänger auf derselben Leitung (vom selben Sender) ausgeschlossen, so daß auch keine Interrupts von intakten Prozessoren verlorengehen können.

*Verschiedene ISR-Abarbeitungszeiten*

Wenn beim Bearbeiten der Interrupts Routinen benutzt werden, die verschiedene zeitliche Längen haben (z. B. wenn ein Prozessor eine Nachricht erhält, der andere aber nicht), so ist eine korrekte Reihenfolge ebenfalls nicht gewährleistet.




Nach der Registrierung eines Interrupts zur Kommunikation darf während einer Zeitspanne, die zum Entnehmen einer Nachricht aus einem „port" ausreicht, kein weiterer Interrupt generiert werden. Vor dem Auslösen eines Interrupts ermittelt deshalb jeder Prozessor, ob diese Zeitspanne seit dem letzten Interrupt schon vergangen ist.

*Interrupts bei der Zeitabfrage*

Ein Interrupt kann gerade in der Situation erfolgen, in der ein Prozessor vor dem Senden ermittelt hat, daß die oben geforderte Zeitspanne verstrichen ist. Würde er nach dem Abarbeiten der ISR an dieser Stelle das Programm weiter bearbeiten, so würde er ohne weitere Zeitabfrage den Interrupt fürs Senden sofort (im Gegensatz zur Lösung mit unterschiedlichen ISR-Abarbeitungszeiten) auslösen.

Dieses Problem kann dadurch vermieden werden, daß die Instruktionen zwischen der Zeitabfrage und dem Auslösen des Interrupts ununterbrechbar (clear interrupt) abgearbeitet werden.

*Verzögerte Interrupts*

Führt ein Prozessor gerade selbst einen Befehl in einem nichtunterbrechbaren Codestück aus und es tritt ein Interrupt auf, so kann der Prozessor den Interrupt erst verzögert bearbeiten. Erfolgt innerhalb dieser Zeit ein weiterer Interrupt, so bearbeitet der eine Prozessor die Interrupts gemäß ihren Prioritäten, die anderen aber möglicherweise nach der zeitlichen Reihenfolge. Dies führt zu Inkonsistenz der Nachrichten.

Alle Prozessoren müssen vor Abarbeitung des aktuellen Interrupts eine gewisse „Karenzzeit" abwarten. Damit ist sichergestellt, daß alle eventuellen Interrupts auch eingetroffen sind. Nach diesem Zeitabschnitt können keine Interrupts mehr von intakten Einheiten eintreffen, da sich alle Prozessoren im „interrupted"-Zustand befinden. Treffen trotzdem zusätzliche Interrupts ein, so stammen sie von defekten (nicht synchronisierten) Boards. Über ein Maskierungsbit im Interrupt-Controller kann der Interrupt eines als defekt erkannten Prozessors wirkungslos gemacht werden.

Da unsere Implementation nicht für zeitkritische Anwendungen gedacht ist, erlaubt der beschriebene, in Software ausgeführte Mechanismus zur Synchronisation im Unterschied zu SIFT [8] und FTMP [5] den Verzicht auf eine globale Systemuhr mit ihren Problemen [4]. □

**Literatur**


[1] E. Ammann, R. Brause, M. Dal Cin, E. Dilger, J. Lutz, T. Risse: Attempto: A Fault-Tolerant Multiprocessor Working Station. Design and Concepts. Proc. FTCS-13. Milano 1983, S. 10–13.

[2] M. Dal Cin, E. Dilger (Eds.): Self-Diagnosis and Fault-Tolerance. Attempto-Verlag. Tübingen 1981.

[3] G. Färber: Task-Specific Implementation of Fault-Tolerance in Process Automation. In [2] S. 84–102.

[4] S. G. Frison, J. H. Wensley: Interactive Consistency and its Impact on the Design of TMR Systems. Proc. FTCS-12, Santa Monica, 1982, S. 228–234.

[5] A. L. Hopkins et al., FTMP: A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft Control, Proc. IEEE, Vol. 66/10 1978, S. 1221–1239.

[6] D. Katsuki et al., PLURIBUS- an Operational Fault-Tolerant Multiprocessor, Proc. IEEE, Vol. 66/10 1978, S. 1146–1159.

[7] Y. Tohma: The SAFE-Project, Tokyo Institute of Technology, private Mitteilung.

[8] J. H. Wensley et al., SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control, Proc. IEEE, Vol. 66, Oct. 1978, S. 1240–1255.

[9] N. Wirth: Programming in Modula-2. Springer-Verlag 1982.