SOFTWAREKONZEPTE DES FEHLERTOLERANTEN ARBEITSPLATZRECHNERS ATTEMPTO

R.Brause, E.Ammann, M.Dal Cin, E.Dilger, J.Lutz, T.Risse

Zusammenfassung: In diesem Artikel werden Konzepte beschrieben, welche die Entwicklung des Betriebs-Systems für den fehlertoleranten Arbeitsplatz-Rechner ATTEMPTO bestimmen. Besondere Bedeutung kommt dabei einer konsistenten Kooperation von Prozessoren zu.

Abstract: In this paper concepts are decribed which determine the development of the operating system for the fault-tolerant working station ATTEMPTO. Especially we discuss the consistent cooperation of processors.

1. Einführung

ATTEMPTO *) ist ein experimenteller, fehlertoleranter Arbeitsplatz-Rechner, der zur Zeit in Tübingen entwickelt wird. Folgende Ziele werden verfolgt:

1) Entwurf eines Rechners, der dem Benutzer die Möglichkeit bietet, ein seinen jeweiligen Erfordernissen angemessenes Verhältnis zwischen Fehlertoleranz und Rechenleistung zu wählen.

2) Bei der Implementierung soll gezeigt werden, daß es möglich ist,
   a) mit kommerzieller Hardware (Single-Board-Computer SBC) ohne nennenswerte Eigenentwicklung,
   b) in einer höheren Programmiersprache und
   c) mit einem üblichen Betriebssystemkern
   einen fehlertoleranten Rechner aufzubauen.

3) Der Rechner soll auch als Testbett für die Erprobung von Diagnosealgorithmen und Rekonfigurationsstrategien dienen. Deshalb müssen die Fehlertoleranzmechanismen modular und hardwareunabhängig gewählt werden können.

*) A TesTable Experimental MultiProcessor system with fault-TOlerance: ATTEMPTO, Motto des Gründers der Universität Tübingen

ACM Proc. Microcomputing II, Teubner-Verlag 1983 pp. 328-341

2. Die Benutzersicht von ATTEMPTO

Der Benutzer sieht den Arbeitsplatzrechner als Single-User-, Multi-Tasking-System; die Realisierung als Multiprozessorsystem ist ihm verborgen. Das System bietet die Möglichkeit, die Fehlertoleranzeigenschaften im Gegensatz zu [3],[7],[8] individuell für jede Anwendung zu wählen. Dies geschieht durch Festlegen eines Fehlertoleranzindex. Die Angabe des Fehlertoleranzindexes t bedeutet, daß bei der Ausführung eines Programms transiente oder permanente Fehler in maximal t SBCs toleriert werden, in dem Sinne, daß keine fehlerhaften Ergebnisse ausgegeben werden. Syntaktisch ist der Index Teil des Programmnamens. Beispielsweise bedeutet ein Eintippen von 'MEINPROGRAMM (2)', daß 'MEINPROGRAMM' mit Toleranzgrad t=2 ausgeführt werden soll.
Der Benutzer kann das System soviele Programme gleichzeitig ausführen lassen, wie es die Systemkapazität erlaubt: mehrere Programme mit geringem oder wenige mit hohem Fehlertoleranzindex.

3. Die Hardwarekonfiguration des Systems

Die Hardware-Struktur der Implementierung besteht aus handelsüblichen Single-Board-Computern (Intel iSBC 86/12A) mit Dual-Port-Memory, die durch einen Multi-Master-Bus zur Inter-Prozessor-Kommunikation miteinander verbunden sind. Das Benutzer-Terminal ist über eine RS232-Schnittstelle mit allen SBCs verbunden.

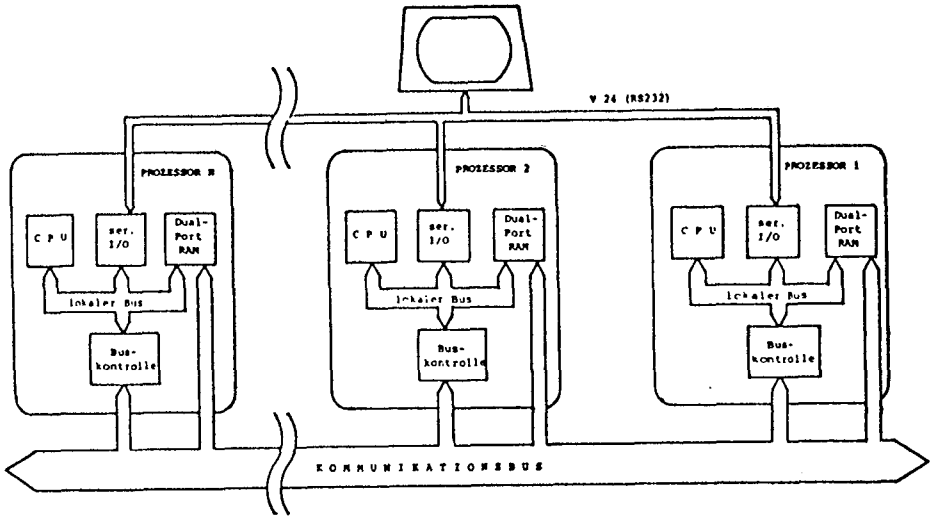

Abb.1 Hardware-Konfiguration

4. Systemsoftware

Das Betriebssystem von ATTEMPTO besteht aus identischen, autonomen, lokalen Betriebssystemen ATOS (ATTEMPTOs local Operating System), eines auf jedem SBC. Den Kern von ATOS bildet ein UNIX-ähnliches Einprozessor-Mehrprozeß-Betriebssystem (OS). Darauf baut eine Zwischenschicht von Systemfunktionen auf, welche die Fehlertoleranzeigenschaften bereitstellt. 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 OS von den Device-Handlern übergeben werden.
Um Fehler tolerieren zu können, werden Kopien des Benutzerprogramms asynchron auf mehreren SBCs parallel abgearbeitet. Die Ergebnisse werden anschließend verglichen (s.4.2.2).
Zum jetzigen Zeitpunkt kommunizieren die lokalen Betriebssysteme nur zum Zwecke der Fehlertoleranz und der Resourcenverwaltung.

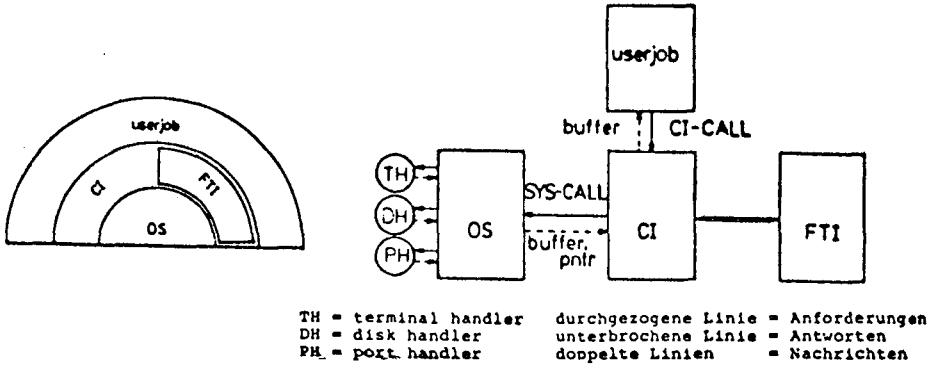

Abb. 2 Software-Konfiguration

Die Zwischenschicht besteht aus der Kommunikationsinstanz (CI) und der Fehlertoleranzinstanz (FTI).
Diese Zwischenschicht ist transparent.

Als Systemprogrammiersprache wird Modula-2 verwendet [9], da es die einzige zur Zeit für uns verfügbare Programmiersprache ist, die sich sowohl für Systemzwecke eignet als auch in der Sprachdefinition und der Programmierumgebung das Prinzip der Modularisierung und der wohldefinierten Schnittstellen unterstützt. Damit werden Projektmanagement, insbesondere Dokumentation und Validierung unseres Systems, entscheidend erleichtert.

4.1 Die Kommunikationsinstanz (CI)

Wenn ein Benutzerprogramm eine Dienstleistung des System anfordert (z.B. Eingabe/Ausgabe), so geschieht dies durch einen sogenannten CI-Call. Die Kommunikationsinstanz überprüft die Syntax und die Zulässigkeit der Anforderungen und schützt dadurch das System. Sie stellt auch die Möglichkeiten bereit zur Kommunikation zwischen Betriebssystemkern (OS) und FTI sowie zwischen zwei FTI auf verschiedenen SBCs.

4.2 Die Fehlertoleranzinstanz (FTI)

Die lokale FTI ist verantwortlich für folgende Betriebssystemaufgaben:

- lokale Interpretation der Benutzerkommandos an ATTEMPTO
- Management der systemweiten Resourcen (z.B.Terminal)
- Kontrolle der Daten von und zu den Ein- und Ausgabegeräten
- Systemweite Konsistenz der Systemtafeln

Außerdem werden von der FTI folgende Funktionen für Fehlertoleranzzwecke bereitgestellt:

- dezentrales Dispatching der Benutzerprogramme
- Vergleich der Ausgabedaten der lokalen Kopien eines Benutzerprogramms und anschließende Diagnose bei Nichtübereinstimmung
- Überwachung bei der Ausgabe der Daten
- Fehlerinterpretation und -behandlung

Aus Gründen der Fehlertoleranz besitzt jede FTI ihre eigene Systemtafel, den sogenannten Systemkontrollblock (SCB), vgl. Abb.3.

```
TYPE
 ticks    = CARDINAL;
 tSCB     = RECORD
              myself          :[1..N];
              Statustable     :ARRAY[1..N] OF (ok, faulty, notexistent);
              FaultFreqs      :ARRAY[1..N] OF CARDINAL;
              Resources       :ARRAY[1..resmax] OF (free, locked);
              JCBqueue        :tQueue;
            END;

 tSBCset = SET OF [1..N];        (* Menge aller Single-Board-Computer *)
 tJCB     = RECORD
              JobName         :STRING;
              SI              :INTEGER;
              Colleagues      :tSBCset;
              CoStartTimes    :ARRAY[1..N] OF ticks;
            END;

 tMsgKey = (UserJobWrite,UserJobInputRequest,NewJob,Signatur,Key,...);
 tMessage= RECORD
              Sender          :[1..N];
              Receiver        :tSBCset;
              MsgKey          :tMsgKey;
              data            :...;
            END;
```

Abb. 3 Datenstrukturen

Der SCB enthält eine doppelt verzeigerte Liste (JCBqueue) von Referenzen auf Job-Kontrollblöcke (JCB). Jeder Job-Kontrollblock charakterisiert ein Benutzerprogramm mit seinem Namen und der Angabe der SBCs, die es ebenfalls ausführen, den sogenannten 'Kollegen'.

4.2.1 Dispatching der Benutzerprogramme

Es findet kein Pre-Scheduling wie bei [8] und [3] statt. Stattdessen wird Dispatching 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 als 'noch zu bearbeiten' gekennzeichnete in der Eingabereihenfolge nächste Benutzerprogramm. Wenn ein Prozessor ein Benutzerprogramm beginnt, ist er in allen Systemtafeln mit Hilfe des Synchronisations-Mechanismus von Abschnitt 5 vermerkt.

Im einzelnen wird der Dispatcher bei folgenden drei Ereignissen aktiv:

i) Der Benutzer gibt das Kommando ein, ein neues Programm zu starten. In diesem Fall wird ein eventuell laufendes Benutzerprogramm unterbrochen und das Kommando interpretiert. Dazu wird ein neuer Job-Kontrollblock erzeugt, in den der Name des Benutzerprogramms, die Zahl der zur Ausführung nötigen Kollegen, Zeiger zu den I/O- Buffern dieses Benutzerprogramms etc. eingetragen werden.
Falls ein Benutzerprogramm dazu unterbrochen wurde, wird es nun fortgesetzt. Anderfalls wird nach ii) verfahren.

ii) Ein Benutzerprogramm wurde abgearbeitet.
Zuerst werden die entsprechenden Einträge in der JCB-Liste entfernt und der Speicherplatz freigegeben. Dann durchsucht die FTI die JCB-Liste nach Jobs, die noch keine vollständige Kollegenliste besitzen. Ist ein solcher Job gefunden, so bewirbt sich die FTI für diesen Job. Dazu teilt sie ihre Absicht, diesen Job zu bearbeiten, allen Prozessoren, ein-

schließlich sich selbst (Pseudo- Port, Abschnitt 5), mit. Wenn diese Nachricht bei ihr selbst ankommt und in der Zwischenzeit keine andereren Prozessoren durch Nachrichten die Kollegenliste des Jobs vervollständigt haben, startet die FTI den Job.
Mit dem Synchronisierungsmechanismus aus Abschnitt 5 ist gewährleistet, daß die Eintragungen in der JCB-Liste auf allen SBCs in derselben Reihenfolge erfolgen und damit systemweit konsistente JCBs erzeugen,

iii) Die Nachricht liegt vor, daß ein Prozessor einen Job bearbeiten will. Der laufende Benutzerjob wird unterbrochen. Falls der gewünschte Job noch keine vollständige Kollegenliste besitzt, notiert die FTI den Sender als Kollegen und die Empfangszeit der Nachricht als Startzeit des Jobs. Die Kenntnis der Startzeit eines Jobs dient dazu, eine Zeitüberwachung (time-out) der Kollegen zu ermöglichen.

4.2.2 Vermeidung von fehlerhafter Ausgabe

Die FTI garantiert folgendermaßen, daß nur korrekte Ergebnisse ausgegeben werden:

Vor jeder Ausgabeoperation vergleichen alle Prozessoren, die dasselbe Programm bearbeiten, die zur Ausgabe anstehenden Daten. Die Daten werden dazu auf eine genormte Länge komprimiert (Signatur-Bildung). Jedem Fehlertoleranzindex ist ein sogenannter Testgraph zugeordnet, durch den festgelegt ist, welche Signaturen zu vergleichen sind. Die Diagnose basiert auf den Ergebnissen dieser Vergleiche. Damit gewinnt jeder SBC Information 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 SBC auszutauschen. Stattdessen führt jeder SBC eine eigene Fehlerfrequenzliste, in der auftretende Nichtübereinstimmungen der Resultate notiert werden. Für Service-Zwecke kann der Benutzer ein Rekonfigurationsprogramm starten, das auf allen SBCs des Systems abgearbeitet wird und zu einer Aktualisierung der dezentralen, systemweiten Systemtafeln mittels der lokalen, eventuell unterschiedlichen Fehlerfrequenzlisten führt.

5. Inter-Prozessor Koordination

In manchen Multiprozessorsystemen werden die Einzelaktivitäten der Prozessoren durch eine einzige, meist in einem 'common memory' gelagerte Systemtafel synchronisiert. Diese Lösung hat den gravierenden Nachteil, daß bei einer fehlerhaften Systemtafel (defekter Speicher) oder fehlerhaften Zugriffswegen (defekter Bus) das System zusammenbricht.

5.1 Konsistenz der Systemtafeln

Um solch einen Systemzusammenbruch zu verhindern, hat in ATTEMPTO jeder SBC seine eigene Systemtafel, die er durch Nachrichtenaustausch mit denen der anderen SBCs konsistent hält. 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. Deshalb wählten wir eine andere Lösung, nämlich die zeitliche Reihenfolge der Nachrichten zur Veränderung der Systemtafeln bei allen SBCs gleich zu halten. Damit sind bei gleichem initialem Ausgangszustand die Systemtafeln auch ohne Rückantwort jederzeit konsistent.

5.2 Nachrichten-Austausch in ATTEMPTO

Eine Möglichkeit, die zeitliche Reihenfolge der Nachrichten auf allen SBCs identisch zu halten, bietet ein Broadcast-Bus. Bei dieser Lösung wird jede Nachricht von allen

SBCs im System empfangen. Da dieser Bus von verschiedenen Prozessoren nur nacheinander benutzt werden kann, ist damit eine eindeutige Reihenfolge der Nachrichten gegeben.
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 im Bussystem unserer Implementation (MULTIBUS) für eine Broadcast-Eigenschaft eine zusätzliche, nicht-triviale Hardware-Änderung auf jeder SBC-Platine nötig wäre, 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 MULTIBUS (s. Abb. 4).

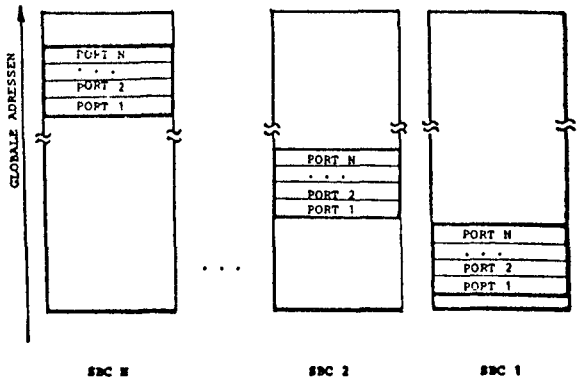

Abb. 4 Aufteilung des globalen Adressbereichs

Die Interprozessor-Synchronisation und Kommunikation basiert in ATTEMPTO auf einem Nachrichtenaustausch, der über 'ports' (spezielle Speicherbereiche des Dual-Port RAM) abgewickelt wird. Auf jedem SBC existiert ein port für den Empfang von Nachrichten von jedem SBC des Systems. Auf die Bedeutung des ports für Nachrichten an sich selbst ('pseudo-port') wurde bereits in 4.2.1 hingewiesen.
Der sicherere 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.

5.3 Koordination des Nachrichtenaustauschs

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 SBCs nicht mehr identisch ist.
Eine übliche, aber nicht fehlertolerante Möglichkeit, dieses Problem zu lösen, ist die Sperrung des Busses während der gesamten Sendedauer einer Nachricht an alle Empfänger (ein defekter Prozessor könnte damit den Bus dauerhaft blockieren).
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 SBCs. Falls ein SBC zu den Empfängern einer Nachricht gehört, leitet dessen Handler sie an das OS 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 ATOS registriert werden, ist somit nicht vom Eintreffen der Nachrichten auf den verschiedenen SBCs bestimmt, sondern von der Reihenfolge der dazugehörigen Interrupts. Im Vergleich zu den üblichen Broadcast-Systemen ist für den Nachrichtenaustausch eine Interruptleitung für jeden SBC nötig.

Die Forderung, auf allen SBCs die gleiche zeitliche Reihenfolge der Nachrichten zu garantieren, wird damit zu der Forderung, auf allen SBCs 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.

a) 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 SBCs identisch ist, ist die Konsistenz der Systemtafeln gewahrt. Durch den Nachrichten-Rückantwortsmechanismus sind zwei Interrupts in kurzem Abstand für den selben Empfänger auf der selben Leitung (vom selben Sender) ausgeschlossen, so daß auch keine Interrupts von intakten Prozessoren verloren gehen können.

b) Unterschiedliche 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 Interrupt 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.

c) 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 für's Senden sofort (im Gegensatz zu b) auslösen.

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

d) Verzögerte Interrupts

Führt ein Prozessor gerade selbst eine Instruktion 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 (nichtsynchronisierten) Boards. Über ein Maskierungsbit im Interrupt-Controller kann der Interrupt eines als defekt erkannten SBC 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 allen ihren Problemen [4].

6. Schluß und Ausblick

Wir glauben, daß ATTEMPTO als universaler, fehlertoleranter (wenn auch nicht ultra-zuverlässiger) Arbeitsplatz-Rechner eine angemessene Antwort auf die zunehmende Nachfrage nach zuverlässigen Rechen-Systemen darstellt. Die hier vorgestellten Überlegungen (insbesondere die zur Inter-Prozessor-Kommunikation) werden Ausgangspunkt sein für die Validierung gerade der Fehler-Toleranz-Eigenschaften des Systemes - eine Aufgabe, die wir für unverzichtbar halten.

Diese Arbeit wurde von der Deutschen Forschungsgemeinschaft gefördert.

Referenzen

[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) 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] 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) 228-234

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

[6] D.Katsuki et al.,PLURIBUS- an Operational Fault-Tolerant Multiprocessor, Proc. IEEE, Vol 66/10 (1978) 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), 1240-1255.

[9] N.Wirth, Programming in Modula-2, Springer-Verlag (1982).

Anschrift der Autoren:
Institut für Informationsverarbeitung
Universtät Tübingen
Köstlinstr.6
D-74 Tübingen