Ein ‘Stabiler’ Speicher
für ein
speichergekoppeltes Multiprozessorsystem

Der Technischen Fakultät der
Universität Erlangen-Nürnberg

zur Erlangung des Grades

D O K T O R - I N G E N I E U R

vorgelegt von

A n d r e a s G r y g i e r

Erlangen - 1995

Als Dissertation genehmigt von
der Technischen Fakultät der
Universität Erlangen-Nürnberg

Tag der Einreichung:	07. November 1994
Tag der Promotion:	22. Dezember 1994
Dekan:	Prof. Dr. Franz Durst
Berichterstatter:	Prof. Dr. Mario Dal Cin
	Prof. Dr. Erik Maehle

Danksagung:

Ich möchte mich ganz herzlich bedanken bei Herrn Prof. Dr. Mario Dal Cin für die Betreuung und die Begutachtung dieser Arbeit, ebenso bei Herrn Prof. Dr. Erik Maehle für die bereitwillige Übernahme des Zweitgutachtens.

Mein weiterer Dank gilt allen Kollegen, die mich mit Kritik und Anregungen bei dieser Arbeit unterstützt haben. Auch das Personal im Labor möchte ich dabei erwähnen, das mir alle nötige Unterstützung bei der Implementierung des Stabilen Speichers zukommen ließ.

Besonderer Dank gilt Manfred Semmler, der wichtige Teile des Stabilen Speichers implementiert und gepflegt hat.

Diese Arbeit wurde im Rahmen des Sonderforschungsbereichs 182 “Multiprozessor- und Netzwerkkonfigurationen” von der Deutschen Forschungsgemeinschaft unterstützt.

Eine Anmerkung zur Schreibweise:

In dieser Arbeit wird der Begriff “Stabiler Speicher” als Eigenname verwendet.

Daher wird in diesem Zusammenhang das Adjektiv “stabil” groß geschrieben.
Auf die Anführungsstriche wird dabei verzichtet.

für

Karin,
Clarissa
und
Jonathan

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Motivation

Computer sind aus dem heutigen Leben nicht mehr wegzudenken. Sie erledigen schon so viele Aufgaben, daß sie zum größten Teil zum unverzichtbaren Teil der industrialisierten Menschheit geworden sind. Die fortschreitende Miniaturisierung und die immer größer werdende Leistungsfähigkeit der Rechnereinheiten scheinen bisher noch nicht an ihre Grenzen gestoßen zu sein. Innerhalb weniger Jahre vervielfachen sich Prozessorleistungen und Speicherkapazitäten.

Viele Aufgaben, die mit einem langdauernden Berechnungsverfahren und mit einem enormen Speicherbedarf verbunden sind, wie z.B. umfangreiche Simulationen aus der Strömungsmechanik, ließen sich bisher kaum mit einem vertretbaren Aufwand bearbeiten. Als Konsequenz davon wurde die Aufgabe in kleinere Stücke zerteilt, die dann stückweise mit den zur Verfügung stehenden Mitteln bearbeitet wurde. Unter der Zerstückelung leiden aber im allgemeinen die gewonnen Ergebnisse, da oft in den kleineren Szenarien nicht mehr alle Abhängigkeiten berücksichtigt werden können, die in der Gesamtkonstellation bestehen. Die Vereinfachung des Modells führt dann oft zu Ergebnissen, die oft im Detail nicht sehr befriedigend sind.

Die fortschreitende Entwicklung bezüglich der Rechenleistung und der Größe des verfügbaren Arbeitsspeichers ermöglicht es, diesen Nachteil zumindest stückweise zu beheben. Die Modelle wurden ja nur deshalb so weit vereinfacht, damit sie überhaupt mit vertretbarem Aufwand bearbeitet werden konnten. Mit leistungsfähigeren Maschinen können aber die realitätsnäheren, aber komplizierteren Modelle für die Beschreibung der realen Welt in vertretbarer Zeit berechnet werden.

Man dreht sich also im Kreis. Je leistungsfähiger die Rechner werden, um so größer werden die Ansprüche, die zur Bewältigung der Aufgabe gestellt werden. Somit wird es wohl nie dazu kommen, daß die aufwendigen Aufgaben für einen Rechner ausgehen.

Multiprozessoren bieten nun die Möglichkeit, die Leistungsfähigkeit eines Rechners noch bedeutend zu vergrößern. Das betrifft sowohl die verfügbare Verarbeitungsleistung als auch den gesamten Arbeitsspeicher. Es ist also kein Wunder, daß sie bei rechenintensiven und speicherintensiven Aufgaben eingesetzt werden. Und auch hier ist man oft genug gezwungen, lange Rechenzeiten für die Bewältigung der Aufgabe in Kauf zu nehmen.

Mit der Größe eines Systems wächst aber nicht nur die Rechenleistung sondern im allgemeinen auch die Wahrscheinlichkeit dafür, daß ein Fehler im Rechner auftritt. Das bezieht sich auf die Hardware des Systems wie auch auf die Software, die auf diesem Rechner eingesetzt wird. Fehlertoleranzmaßnahmen sind also erforderlich, um langlaufende Prozesse vor den schwerwiegenden Folgen eines Fehlers zu bewahren.

Voraussetzung dafür, Fehlertoleranzmaßnahmen überhaupt wirkungsvoll einsetzen zu können, ist die Entdeckung des Fehlers [ANL81]. Je früher dies nach dem Auftreten des Fehlers geschieht, um so weniger kann sich im allgemeinen der Fehler bereits ausgebreitet und zu Folgefehlern geführt haben. Eine genaue Fehlerdiagnose ist somit leichter durchzuführen. Besonders

nützlich ist, wenn Informationen über den Fehler und seine Umgebung aufgezeichnet werden, die zu einem späteren Zeitpunkt ausgewertet werden können. Diese Aufzeichnungen zeigen oft ein genaueres Bild vom Fehler, als es durch die Untersuchung von Fehlerauswirkungen zu ermitteln ist.

Nach der Fehlerdiagnose können dann geeignete Maßnahmen zur Fehlerbehebung durchgeführt werden. Dafür können zwei verschiedene Methoden angewandt werden [ECH90] [GÖR89]:

Bei einer Vorwärtsfehlerbehebung wird versucht, die Auswirkungen des Fehlers zu kompensieren, um wieder einen korrekten Zustand zu erreichen. Für eine Kompensation ist es sehr wichtig, einen Fehler schnell zu erkennen, um mit recht einfachen Mitteln eine geeignete Fehlerbehebung durchführen zu können. Würde dieser Fehler erst später entdeckt werden, so könnte sich der Fehler schon ausgebreitet haben, indem der fehlerhafte Wert Einfluß auf andere Werte (Variablen) genommen hat.

Dagegen wird bei einer Rückwärtsfehlerbehebung nach dem Entdecken eines Fehlers auf einen vorher erreichten Zustand (“Sicherungspunkt”) zurückgesetzt und das Programm an dieser Stelle wiederaufgesetzt. Im einfachsten Fall ist der Aufsetzpunkt der Beginn des Programms. Bei kurz laufenden Programmen ist dies eine akzeptable und einfache Vorgehensweise, bei lang laufenden Programmen jedoch oft schon allein aus wirtschaftlichen Gründen eine untragbare Situation. Läuft diese Anwendung auch noch auf einem Multiprozessorsystem, das eine deutlich kürzere MTBF (Mean Time Between Failure) hat als seine einzelnen Knoten (je ein Monoprozessor), so besteht die Gefahr, daß sie auch bei weiteren Versuchen vor ihrem Programmende von einem neuen Fehler getroffen wird.

Um eine Anwendung nicht in einer Sisyphusarbeit enden zu lassen, kann als Vorsorgemaßnahme ab und zu der erreichte Zwischenzustand abgespeichert werden, auf den man nach dem Auftreten eines Fehlers zurückgreifen kann. So besteht die Möglichkeit, von diesem Sicherungspunkt aus die Anwendung weiterlaufen zu lassen statt von vorne starten zu müssen. Auf diese Weise können auch Anwendungen bearbeitet werden, die deutlich länger für ihre Bearbeitung benötigen, als es die MTBF eines großen Multiprozessorsystems zulassen würde [HÖN94].

Beispielsweise ist bei einem Multiprozessorsystem, das aus 256 Knoten mit je einer MTBF von 2 Jahren besteht, die Gesamt-MTBF etwa 3 Tage, wenn man unabhängige Ausfälle und eine Einsatzfähigkeit des Gesamtsystems nur bei Funktionstüchtigkeit aller Knoten annimmt.

Es ist somit verständlich, daß der Sicherungspunkt eine sehr hohe Bedeutung hat. Er darf keinesfalls verloren gehen oder zerstört bzw. beschädigt werden, um den Wiederanlauf einer Anwendung von einem Sicherungspunkt aus nicht zu gefährden. Hier kommt die Idee des “Stabilen Speichers” ins Spiel. Er sollte so ausgelegt sein, daß er auch unter Einfluß von Fehlern in der Lage ist, die abgelegten Daten sicher aufzubewahren. Erste Vorschläge für ein Stabiles System, bei dem der Stabile Speicher ein sehr wichtiger Bestandteil ist, wurden von Lampson in den achtziger Jahren veröffentlicht [LAM88]. Seine Vorstellung von einem Stabilen Speicher basiert auf der Verwendung von Plattenspeichern, auf denen der Prozessor alle wichtigen Daten

(für einen eventuellen Wiederanlauf) hält und aktualisiert.

Für Lampson war es zum einen besonders wichtig, daß die Daten sicher auf die Platte gelangen und dort unversehrt bleiben, und zum anderen, daß ein Prozessor-Crash, der sich auch während eines Updates eines Datenblocks im Stabilen Speicher ereignen kann, letztendlich nicht zu verlorenen Daten führt. Da der Update eines Datenblocks nicht als atomare Aktion ablaufen kann, sondern aus einer unterbrechbaren Hintereinanderausführung einzelner Schreibaktionen besteht, ist zur Aufrechterhaltung eines konsistenten Datenblocks eine mehrfache Speicherung der Daten und ein kontrollierter Ablauf des Update-Vorgangs nötig. Eine fehlerkorrigierende Kodierung z.B. mit ECC ist nicht ausreichend.

Die Gedanken Lampsons sollten nun auch für Multiprozessorsysteme übertragen werden können. Für ein an der Universität Erlangen-Nürnberg entwickeltes speichergekoppeltes Multiprozessorsystem MEMSY, das für langlaufende Anwendungen gedacht ist, sollte ein Stabiler Speicher eingerichtet werden. Besonderes Kennzeichen von MEMSY ist die umfangreiche Verbindungstopologie, über die die einzelnen Knoten mit Kommunikationsspeichern zum Datenaustausch verbunden sind. Da hier auch einige Ausfälle von Übertragungswegen toleriert werden können, sollte der Stabile Speicher hier integriert werden [DAL94].

Die Zugriffe über diese Verbindungswege sind aber im Gegensatz zu den Zugriffen bei Lampsons Vorschlag einfache Speicherzugriffe. Diese können, wenn der Prozessor oder auch andere an der Übertragung beteiligte Hardware nicht fehlerfrei arbeiten, viel leichter zu einem Fehler in den abgespeicherten Daten führen, als dies beim Zugriff auf einen Plattenspeicher zu erwarten ist, da für diese Zugriffe praktisch kein Protokoll zu beachten ist. Daher muß der Stabile Speicher zusätzlich zu den von Lampson beschriebenen Fehlern auch mit anderen Fehlersituationen fertig werden. Vor allem muß bei den Zugriffen dafür Sorge getragen werden, daß ein Prozessor nur auf seine eigenen Daten Zugriff hat und nicht auf andere. Denn so etwas war beim Entwurf der Kommunikationsspeicher absichtlich erlaubt worden.

Besondere Aufmerksamkeit muß dem Verändern und Löschen der abgelegten Daten geschenkt werden, da diese Operationen die sicher geglaubten Daten zerstören können. Daher muß sich der Stabile Speicher gerade bei einer solchen Aktion sicher sein, daß dieser “Auftrag” zu Recht gegeben wurde. Das bedeutet vor allem, daß der Stabile Speicher in der Lage sein muß, die gewünschte Aktion auf ihre “Glaubwürdigkeit” hin zu beurteilen. So wird ein neues Protokoll nötig, dessen Einhaltung vom Stabilen Speicher ständig überwacht und als “korrekte Arbeitsweise des Auftraggebers inklusive der Verbindungs-Hardware” interpretiert werden kann.

Mit Hilfe des Protokolls sollen neben Hardware-Fehlern auch Fehler in der Software erkannt werden, die bei Zugriffen auf den Stabilen Speicher eine Rolle spielt. Auch aus diesem Grund muß immer der korrekte Ablauf in seinen Zugriffsfolgen überwacht werden können.

Die angesprochenen Schutzmaßnahmen gehen somit weit über das hinaus, was schon jetzt in vielen Rechnern zum Schutz vor Fehlern in den abgelegten und übertragenen Daten auf vielfältige Weise eingesetzt wird. Aber auch das Zusammenwirken der verschiedenen bereits vorhan-

denen Schutzmechanismen und der noch zu integrierenden Mechanismen (vor allem zur Überwachung des Prozessors / Prozeßablaufs) mit den Möglichkeiten, die der Stabile Speicher bietet, ist für die gesamte Fehlertoleranz-Situation in einem Multiprozessor sehr wichtig.

Die Einrichtung eines Stabilen Speichers stellt für den Benutzer die Möglichkeit dar, wichtige Daten unter einen besonderen Schutz stellen zu können. Gegen verschiedene Fehlerursachen (Übertragungsfehler, spontan veränderte Bits in abgespeicherten Daten) muß er dann keine eigenen Vorkehrungen treffen, da diese bereits vom System behandelt werden.

Das Multiprozessorsystem MEMSY (Modular erweiterbares Multiprozessor-System), das an der Universität Erlangen-Nürnberg konzipiert wurde, gehört zu der Klasse von Systemen, für die eine große Prozessorzahl vorgesehen ist. In der gegenwärtigen Implementierung ist es mit 20 Knoten (mit insgesamt 80 Prozessoren) zwar eher ein kleines Multiprozessorsystem, ist aber von der Topologie her so entworfen worden, daß dieses System beliebig groß skaliert werden kann. Dabei bleibt die Topologie vollständig erhalten. Vor allem muß keine Clusterung eingeführt werden. Daher kann die Arbeitsweise eines sehr großen Systems dieser Art direkt vom kleinen (vorhandenen) System übernommen werden.

Für den Datenaustausch zwischen den Prozessoren stehen Kommunikationsspeicher zur Verfügung, die regelmäßig im Gesamtsystem angeordnet sind. Dabei hat jeder Prozessorknoten Zugriff auf mehrere Kommunikationsspeicher. Ebenso kann jeder Kommunikationsspeicher von mehreren Prozessorknoten erreicht werden.

Vor diesem Hintergrund wurde der Einsatz eines Stabilen Speichers für MEMSY konzipiert. Dabei sollte der Ansatz verfolgt werden, den Stabilen Speicher mit Hilfe von RAM-Bausteinen zu implementieren. Auf Grund ihrer kurzen Zugriffszeit im Vergleich zu Plattenzugriffen versprechen sie einen auch für den Anwender akzeptablen, geringen Overhead.

In dieser Arbeit werden nun zunächst Lampsons Ansatz und eine daraus abgeleitete Implementierung mit RAM-Speichern vorgestellt (Kapitel 2). Anschließend wird die Struktur des Multiprozessorsystems MEMSY erläutert, wobei besonders auf den Datentransfer zwischen den Prozessoren und den Kommunikationsspeichern eingegangen wird (Kapitel 3). Dies mündet in die Überlegungen, auf welche Weise der Stabile Speicher in dieses Multiprozessorsystem integriert werden kann (Kapitel 4). Danach werden die möglichen Fehlersituationen besprochen, mit denen bei MEMSY am ehesten gerechnet werden muß, und die für den Stabilen Speicher eine Rolle spielen (Kapitel 5). Den größten Raum nimmt dann die Vorstellung des Stabilen Speichers ein. Neben der Struktur dieses Stabilen Speichers werden auch die vorgenommenen Maßnahmen zur Fehlertoleranz näher erläutert (Kapitel 6). Anschließend werden Ergebnisse vorgestellt, die aus der Implementierung des Stabilen Speichers gewonnen wurden (Kapitel 7).

2 Ansätze für einen Stabilen Speicher

Grundlegende Überlegungen zu einem Stabilen Speicher gehen auf einen Algorithmus zurück, der von Lampson für ein stabiles Gesamtsystem entwickelt wurde [LAM88]. Die Grundüberlegung dabei ist, über ein reales System, das vielen Fehlermöglichkeiten ausgesetzt ist, ein virtuelles System zu konstruieren, das ein fehlertolerantes Verhalten zeigt.

Als Einsatzgebiet betrachtete Lampson Transaktionssysteme, die auf Rechnern bearbeitet werden, z.B. Buchungssysteme. Dies sind i.a. komplexere Vorgänge, die im Rechner nicht atomar ablaufen können. In Fehlersituationen kann es vorkommen, daß sie an einer Stelle abgebrochen werden, in der weder die Situation vor dem Start der Aktion noch die Situation nach der Fertigstellung dieser Aktion vorliegt. Eine solche Situation muß nach der Wiederaufnahme des Betriebs erkannt und geeignet behandelt werden. Dazu muß es möglich sein, entweder in den Zustand vor dem Start der Aktion oder in den Zustand nach Beendigung der Aktion zu gelangen.

Eine sehr wichtige Rolle spielt dabei der Plattenspeicher, in dem die bearbeiteten Daten abgelegt sind. Aus einem “normalen” Speicher muß nun ein (virtuell) “stabiler” Speicher geschaffen werden, der die oben gestellten Anforderungen erfüllt.

Dem Vorschlag Lampsons ist der nächste Abschnitt gewidmet. Anschließend wird eine Realisierung vorgestellt, die sich an Lampsons Vorschlag orientiert, jedoch auf einem anderen Speichermedium (RAM) basiert. Da auf dieses in anderer Weise zugegriffen wird als auf einen Plattenspeicher, hatte dies einige Folgen für die Umsetzung von Lampsons Vorschlag. Dies mündet dann in eine Überleitung zum Einsatz eines Stabilen Speichers für das Multiprozessorsystem MEMSY.

2.1 Der Stabile Speicher von Lampson

Prozessoren und Speicher - hier werden Plattenspeicher betrachtet - sind mehr oder weniger anfällig für Fehler. Üblicherweise greift der Prozessor mit den Operationen GET und PUT auf die Platte zu. Dabei wird jedesmal ein Block transportiert. Um dabei aufgetretene Fehler entdecken zu können, ist jeder Block mit redundanter Information ausgestattet, z.B. mit einer Checksumme.

Fehler können nun in allen aufgeführten Bereichen auftreten, im Prozessor, auf der Übertragungsstrecke oder im Plattenspeicher. Für eine Untersuchung von Fehlersituationen muß man sich zunächst über die möglichen Fehlerereignisse klar werden. So sind für jede Teileinheit unterschiedliche Fehler zu erwarten:

- Der Prozessor kann einen Crash erleiden. Das bedeutet, daß der Prozessor plötzlich angehalten wird und alle seine Zustandsinformationen “vergißt”, die er in seinem flüchtigen lokalen Speicher gehalten hat. Dagegen wird eine Fehlersituation, bei der der Prozessor

über einen längeren Zeitraum hinweg fehlerhaft arbeitet, nicht angenommen.

- Auf der Übertragungsstrecke können Fehler in der Weise auftreten, daß der transportierte Block beim Schreiben oder beim Lesen verfälscht wird. Davon können einzelne Bits betroffen sein oder ganze Worte.

- Im Speicher kann sich ein abgelegter Block von einem korrekten Zustand in einen fehlerhafteten umwandeln (spontanes Fehlerereignis), ohne daß ein Speicherzugriff stattgefunden hat. Dies kann z.B. durch transiente Fehler passieren, wobei einzelne Bits umkippen. Bei nachfolgenden Leseoperationen würde man immer zum Schluß kommen, daß der Block fehlerhaft ist.

- Eine weitere Fehlersituation wäre, daß Schreiboperationen nicht stattfinden.

Um diese Fehler handhaben zu können, müssen einige Annahmen bezüglich der Fehlerhäufigkeiten gemacht werden. So wird angenommen, daß während eines längeren Zeitabschnitts höchstens ein Fehler auftreten darf. Der Zeitabschnitt muß dabei so groß gewählt sein, daß dieser Fehler erkannt und seine Auswirkungen beseitigt werden können. Dann ist es nicht nötig, sich mit vielen aufeinanderfolgenden Fehlern zu befassen. Bei den heutigen Rechnern kann diese Annahme mit gutem Gewissen aufrechterhalten werden.

Ein relativ weit verbreiteter Fehler ist die Störung der Übertragungsstrecke. In den meisten Fällen liegen hier transiente Fehler vor. Somit lassen sich solche Fehler oft durch einfaches Wiederholen des Zugriffs beheben. Zur Tolerierung solcher Fehler konstruierte Lampson zunächst aus den normalen GET- und PUT-Operationen eines Plattenspeicherzugriffs die Operationen CAREFUL_GET und CAREFUL_PUT. Die Operation CAREFUL_GET besteht aus der einfachen GET-Operation, die im Fall, daß ein Übertragungsfehler aufgetreten ist, wiederholt wird. Um eine Verklemmung auszuschließen, ist die Anzahl an Wiederholungsversuchen auf eine feste Zahl beschränkt (z.B. 3). Für die CAREFUL_PUT-Operation wird zunächst ein PUT ausgeführt, auf das ein GET folgt. Die zurückgelesenen Daten werden mit denen verglichen, die geschrieben werden sollten. Erkennt man eine Differenz, so wird dieser Vorgang (PUT und GET) wiederholt. Auch hier ist die maximale Zahl an Wiederholungsversuchen auf eine Obergrenze festgelegt. Mit CAREFUL_PUT lassen sich auch nicht erfolgte Schreibversuche erkennen.

Das Ergebnis der Operationen CAREFUL_GET und CAREFUL_PUT sollte, sofern höchstens ein transienter Fehler auf der Übertragungsstrecke vorlag, ein erfolgreicher Speicherzugriff sein. War der Speicherzugriff jedoch nicht erfolgreich, so liegt offenbar ein Fehler vor, der den abgelegten Datenblock betraf. Daher müssen mächtigere Operationen konstruiert werden, die auch die restlichen Fehlersituationen (Fehler im Speicher, Prozessor-Crash) meistern. Nach außen hin sollen diese Operationen immer erfolgreich sein, solange kein Prozessor-Crash erfolgt. Andernfalls, so wird gefordert, muß das Transaktionskonzept eingehalten werden (Wiederherstellung des alten Zustands). Dazu ist es nötig, die Datenblöcke im stabilen Speicher doppelt abgelegt zu haben. Sie bilden ein Paar, das fast immer denselben Inhalt haben soll. Nur während einer Schreib-Operation (Update) oder bei einem Speicherfehler dürfen sie unterschiedliche In-

halte tragen. Entsteht in einer Hälfte ein Fehler, so kann dieser (nach seiner Entdeckung) durch Kopieren aus der anderen Hälfte behoben werden. Die zweite Hälfte ist nach Voraussetzung noch korrekt, da in der vorgegebenen Zeitspanne höchstens ein Fehler auftreten darf. Daraus ergibt sich die Konstruktion der Operationen STABLE_GET und STABLE_PUT:

Bei einem STABLE_GET wird zunächst mit einem CAREFUL_GET von der ersten Hälfte gelesen. Ist der Block fehlerfrei übertragen worden, so ist nichts weiter zu tun. Ist aber ein Fehler entdeckt worden, so wird dieser Block mit einem CAREFUL_GET aus der zweiten Hälfte gelesen. Diese ist nach Voraussetzung noch fehlerfrei. Das Resultat der Operation STABLE_GET ist daher immer erfolgreich.

Bei einem STABLE_PUT wird zunächst ein CAREFUL_PUT auf die erste Hälfte ausgeführt. War dies erfolgreich, so wird auch auf die zweite Hälfte ein CAREFUL_PUT durchgeführt. Konnte auch dies erfolgreich beendet werden, so ist die Aktion STABLE_PUT erfolgreich beendet worden. Dadurch, daß die zweite CAERFUL_PUT-Aktion erst dann gestartet werden darf, wenn die erste erfolgreich beendet worden war, wird sichergestellt, daß zu jedem Zeitpunkt ein korrekter Datenblock verfügbar ist.

Das Erkennen eines Fehlers darf nicht zu lange dauern, damit die Anforderung an das Zeitintervall, in dem höchstens ein Fehler auftreten darf, nicht zu hoch werden. Während dieser Zeit müssen sämtliche Datenblockpaare auf Gleichheit überprüft und gegebenenfalls korrigiert werden können. Dazu wird die Operation CLEANUP eingeführt. Sie muß regelmäßig für alle benutzten Datenblockpaare gestartet werden. Sie vergleicht die Inhalte von Datenblockpaaren miteinander. Sind beide Blöcke fehlerfrei und tragen die gleiche Information, so ist nichts weiter zu tun. Ist ein Block fehlerfrei, der korrespondierende aber nicht, so wird der fehlerfreie Block auf den anderen mit CAREFUL_PUT kopiert. Unter der Voraussetzung, daß während einer längeren Zeit höchstens ein spontanes Fehlerereignis auftritt, das den korrekten Block in einen fehlerbehafteten Block umwandelt, lassen sich auftretende Fehler mit Hilfe von CLEANUP beheben. Aus dieser Vorgehensweise läßt sich erkennen, daß das angesprochene Zeitintervall mindestens so lang sein muß wie man für die Überprüfung aller Datenblöcke im Stabilen Speicher und für deren Korrektur maximal brauchen würde. Damit dem Prozessor auch für “Nutzarbeit” noch Zeit bleibt, sollte die Zeitspanne deutlich höher liegen.

Die Operation CLEANUP soll auch direkt nach einem Prozessor-Crash eingesetzt werden. In dieser Situation müssen zunächst alle Datenblockpaare auf Gleichheit untersucht werden. Bei jedem Datenblockpaar können als Ergebnis drei verschiedene Situationen vorliegen:

1.) beide Datenblöcke sind in Ordnung und tragen dieselbe Information;

2.) ein Datenblock ist in Ordnung, der andere nicht;

3.) beide Datenblöcke sind zwar in Ordnung, tragen aber unterschiedliche Inhalte.

Im ersten Fall ist alles in Ordnung, da der Datenblock zum Zeitpunkt des Crashes offenbar nicht bearbeitet worden war.

Im zweiten Fall war eine Hälfte des Datenblocks während des Crashes offenbar in einer Update-Phase, und diese Aktion wurde unterbrochen. Nach der oben dargestellten Vorgehensweise der Operation STABLE_PUT muß die andere Hälfte noch in Ordnung sein. So kann die defekte Hälfte durch Kopieren der anderen Hälfte korrigiert werden. Als Ergebnis erhält man entweder den Zustand, in dem das Datenblockpaar schon die Update-Aktion erfolgreich ausgeführt hat, oder den Zustand, in dem sich das Datenblockpaar vor der Update-Aktion befunden hat.

Die dritte Situation kann nur auftreten, wenn der Crash bei einer Update-Operation auf dem Stabilen Speicher genau zwischen den beiden CAREFUL_PUT-Phasen aufgetreten ist. In dieser Situation kann man beim CLEANUP beliebig entscheiden, ob für beide Blöcke der alte oder der neue Inhalt durch CAREFUL_PUT gebildet werden soll.

Mit dem Starten der Operation CLEANUP nach einem Prozessor-Crash kann erneut mit dem Abmessen der Zeitspanne begonnen werden, bis zu der ein erneutes CLEANUP für alle Datenblockpaare gestartet werden muß.

Abschließend sei noch auf einige Verhältnisse hingewiesen, die speziell bei einer Implementierung mit Plattenspeichern beachtet werden müssen. Sie betrifft die Auswahl der Lage für die Datenblockpaare.

Falls zwei Plattenspeicher zur Verfügung stehen, kann die Verteilung der Datenblockpaare in der Weise vorgenommen werden, daß die beiden Hälften auf unterschiedlichen Platten zu liegen kommen. Dann kann ein Fehler, der nur eine Platte betrifft, auch nur eine der beiden Hälften treffen. Die andere Hälfte bleibt unversehrt und kann somit als Quelle bei einem anschließenden CLEANUP dienen.

Steht nur eine Platte zur Verfügung, so sind einige Vorkehrungen zu treffen, damit auch in diesem Fall mit sehr hoher Wahrscheinlichkeit nur eine Datenblockhälfte von einem Fehler betroffen ist. Beim Betrieb der Platte sind Fehler typisch, die gleich eine ganze Reihe von Blöcken in Mitleidenschaft ziehen können. So werden bei einer Beschädigung einer Platte überwiegend solche Blöcke betroffen, die sich z.B. auf derselben Oberfläche eines Plattenstapels befinden oder solche, die sich auf demselben Zylinder befinden. Daher muß bei der Auswahl der Seitenpaare berücksichtigt werden, daß die beiden Datenblockhälften nicht in die gleichen Schadenseinzugsbereiche gelegt werden. Dann kann ein solcher Fehler nur einen der beiden Datenblockhälften betreffen.

2.2 Die “Fault Tolerant Machine” (FTM)

Die grundlegenden Überlegungen von Lampson haben zu verschiedenen Implementierungsansätzen geführt. Zum einen lassen sich die Vorstellungen Lampsons direkt auf Plattenspeicher anwenden. Dieser Algorithmus kann auch auf andere Speichermedien übertragen werden, z.B. auf RAM-Bausteine. Einer davon ist der Ansatz von Banâtre und anderen [BAN91] [MHP94] für eine “Fault Tolerant Machine” (FTM).

2.2.1 Struktur der FTM

Für die FTM wird ein üblicher Rechner mit einem “Stable Transactional Memory” (STM) ausgestattet, der die Aufgabe eines Stabilen Speichers übernimmt. Die Architektur der FTM soll jeweils einen Fehler in den Teileinheiten Prozessor, Platte und Bussystem tolerieren können. Dabei wird der STM so ausgelegt, daß ein einzelner interner Fehler behoben werden kann.

Die minimale FTM-Konfiguration ist eine Zwei-Prozessor-Maschine. Sie besteht aus zwei Prozessoren, zwei STMs und zwei Gruppen von Peripheriegeräten. Jeder Prozessor hat einen Zugang zu seinem eigenen STM und einen Zugang zum STM des anderen Prozessors. Jeder STM ist also mit einem Port mit dem zugehörigen Prozessor verbunden und mit einem zweiten Port, dem “Backup-Port”, mit dem zweiten Prozessor verbunden. An jedem Port befindet sich ein intelligenter Controller (C). Dieser kann die Zugriffe des Prozessors auf den STM überwachen. Die Peripheriegeräte sind ebenfalls an zwei Busse angeschlossen. Einer verbindet sie mit dem ersten Prozessor, der andere mit dem zweiten Prozessor. In Abb. 1 ist die Minimalkonfiguration dargestellt.

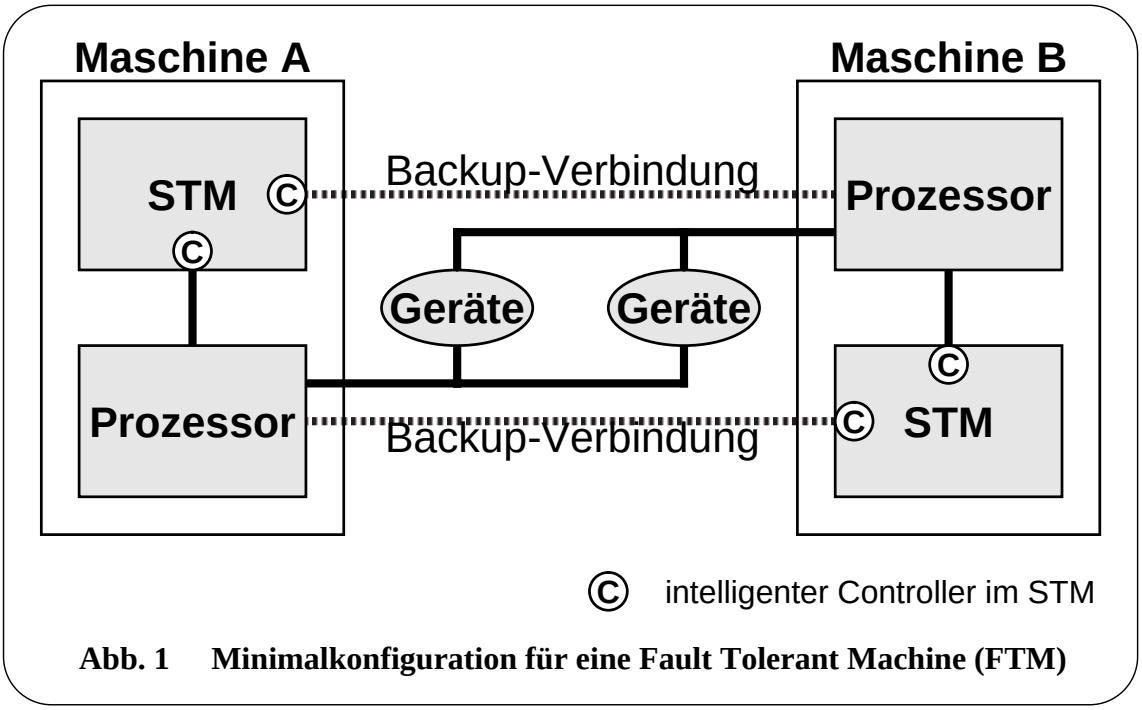


Abb. 1 Minimalkonfiguration für eine Fault Tolerant Machine (FTM)

Die beiden Maschinen A und B sind zunächst einmal eigenständige Rechner. Durch die dargestellte Kopplung über die STMs und die Peripheriegeräte bilden sie jedoch ein Backup-Paar. Sie sollen sich gegenseitig ersetzen können, wenn einer von ihnen durch einen Crash ausfällt. Die Erkennung eines Crashes basiert hier auf der Überwachung eines Watchdog-Timers, der von einem korrekt arbeitenden Prozessor regelmäßig zurückgesetzt wird. Erkennt der Controller den abgelaufenen Watchdog-Timer, so kann der STM über den anderen Controller den Backup-Prozessor von dem Ausfall des anderen Prozessors informieren, damit dieser dessen Arbeit übernehmen kann.

Die Minimalkonfiguration kann zu einem größeren Multiprozessorsystem erweitert werden. Dabei bilden immer zwei Prozessoren mit ihren STMs und Peripheriegeräten in der beschriebenen Weise einen Fehlertoleranz-Verbund (siehe linken Teil der Abb. 2). Für die Kommunikation der Prozessoren untereinander gibt es zwei Busverbindungen. An den einen sind die A-Maschinen aller Fehlertoleranz-Verbunde angeschlossen, an den anderen die B-Maschinen. Die Maschinen sind absichtlich nicht alle an einen Bus verbunden, damit im Fall, daß ein Prozessor aufgrund einer Fehlersituation den Bus nicht mehr freigibt, ein weiterer Bus zur Verfügung steht. Dann können immerhin noch die Prozessoren der anderen Gruppe miteinander kommunizieren.

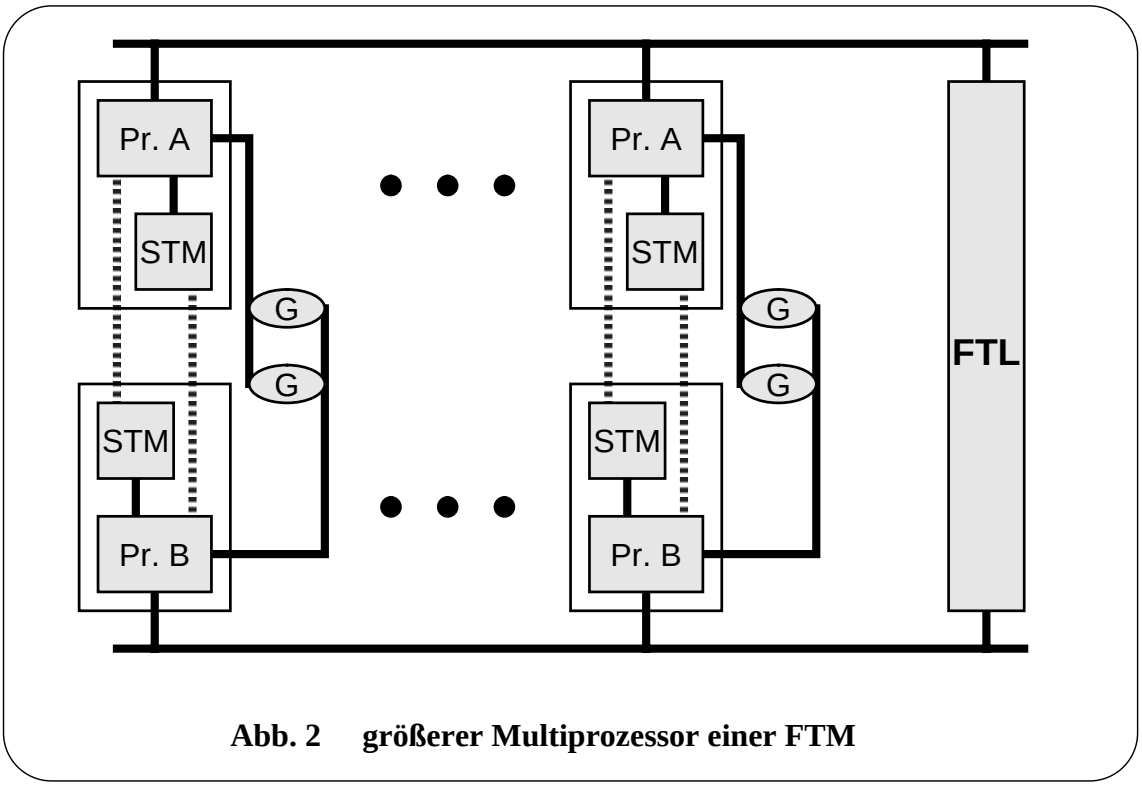


Abb. 2 größerer Multiprozessor einer FTM

Bis hierher stellt sich die FTM als zweigeteiltes Multiprozessorsystem dar: alle A-Maschinen bilden ein lose gekoppeltes System und ebenso alle B-Maschinen. Um aber dennoch ein einheitliches Multiprozessorsystem zu erhalten, bei dem alle Prozessoren untereinander über eine Busverbindung miteinander verbunden sind, werden die beiden Busse über einen “Fault Tolerant Link” (FTL) virtuell zu einem Bus verknüpft. Der FTL arbeitet im fehlerfreien Betrieb als ein paralleles bidirektionales Verbindungssystem, das die Transfers von einem Bus zum anderen durchführt. Um Synchronisationsprobleme zwischen den beiden Bussen zu vermeiden, stehen im FTL Puffer zur Verfügung. Im Fall eines dauerhaften Busfehlers trennt der FTL die beiden Busse voneinander, so daß wieder ein zweigeteiltes Multiprozessorsystem entsteht.

2.2.2 Struktur des STM

Das Stable Transactional Memory (STM) ist implementiert in zwei Speicherboards (siehe Abb. 3). Jeder Speicher entspricht einer Hälfte eines Stabilen Speichers. Update-Operationen werden, entsprechend der Vorgehensweise beim STABLE_PUT, immer zuerst im ersten Speicher ausgeführt und erst danach im zweiten. Der Speicher ist mit Error Correcting Code (ECC) -Mechanismen ausgestattet, die es erlauben, einzelne Bitfehler beim Lesen zu tolerieren. Handelte es sich bei diesem Fehler um einen transienten Fehler, so läßt sich das korrigierte Wort in den Speicher zurückschreiben.

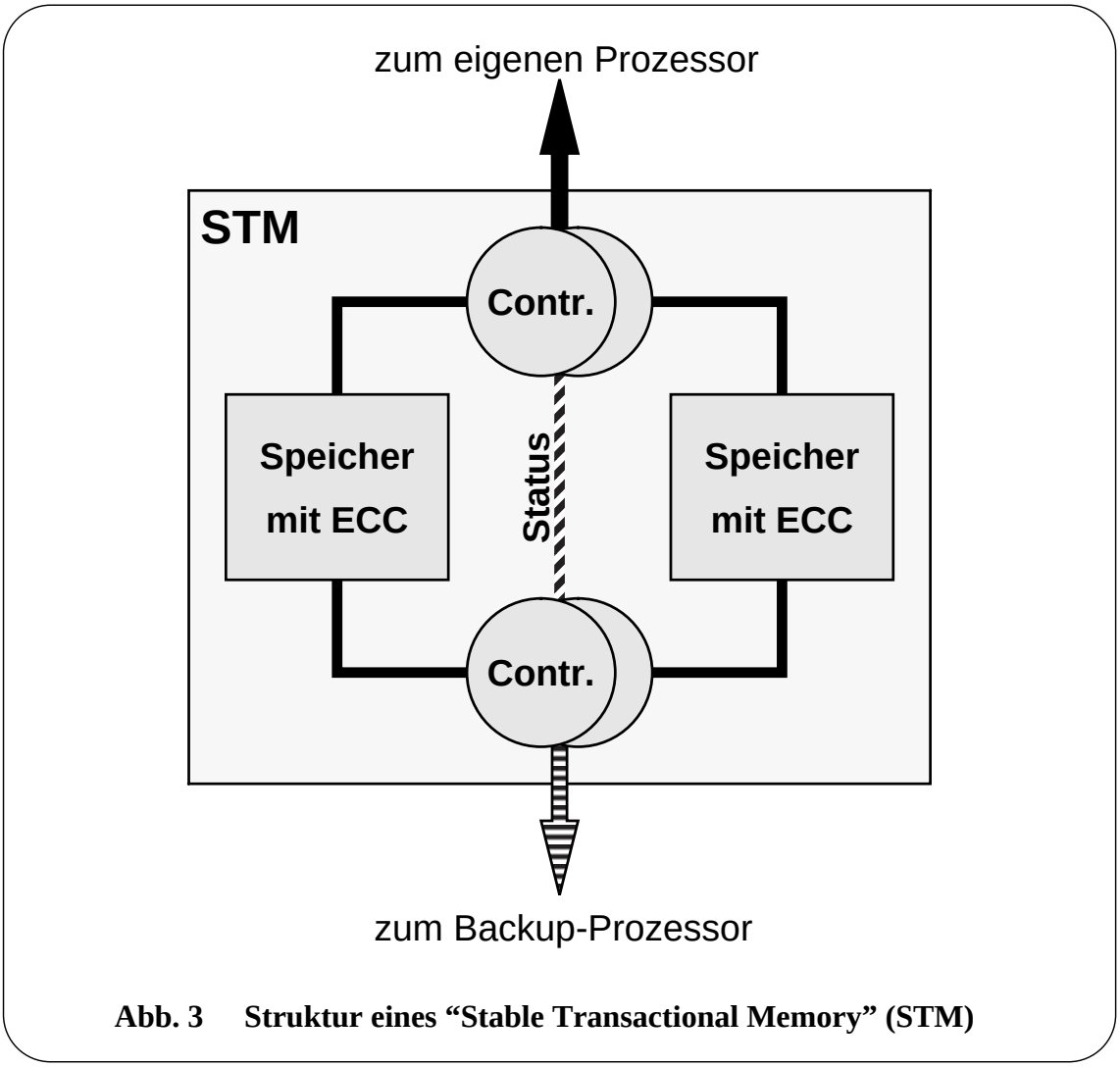


Abb. 3 Struktur eines “Stable Transactional Memory” (STM)

Innerhalb des Speichers trägt jeder Speicherchip ein bestimmtes Bit vieler (z.B. 1 Mega) Worte. Fällt ein solcher Chip aus, so kann zwar kein korrigiertes Wort mehr in den Speicher zurückgeschrieben werden, dennoch kann aus der verbleibenden Information das korrekte Wort gewonnen werden. Im STM ist zusätzlich eine Bit Steering Technik [HAR90] vorgesehen. Dazu werden alle Worte aus dem Speicherbereich geholt, der den defekten Chip betrifft, und korrigiert. Das Ergebnis wird in den Speicher zurückgeschrieben, wobei der defekte Chip durch einen Re-

serve-Chip ersetzt wird. Anschließend kann ein Operator beauftragt werden, den defekten Chip gegen einen neuen auszutauschen.

Zugriffe der Prozessoren werden immer über einen intelligenten Controller abgewickelt. Er hat Zugriffsmöglichkeiten sowohl auf die erste als auch auf die zweite Speicherhälfte. Da dem Controller eine so wichtige Schlüsselrolle zufällt, ist es unbedingt erforderlich, daß ein Ausfall von ihm nicht zu einer Katastrophe führen kann, bei der abgelegte Daten zerstört werden könnten. Dazu ist der Controller als Master-Checker-Paar mit Fail-Stop-Verhalten aufgebaut. Um nach dem Ausfall des Controllers noch auf die abgelegten Daten zugreifen zu können, ist ein zweiter Controller nötig, ebenfalls ein Master-Checker-Paar mit Fail-Stop-Verhalten. Er wird auf der Backup-Seite des STM plaziert. Über ihn kann der Backup-Prozessor auf diesen Stabilen Speicher zugreifen. Um zu erkennen, ob der Partner-Controller defekt oder intakt ist, tauschen die Controller untereinander ihren Status aus. Als Konsequenz aus dem Vorhandensein von zwei Zugriffswegen pro STM müssen die Speicherhälften auch jeweils zwei Ports besitzen.

2.2.3 Arbeitsweise des STM

Die Arbeitsweise des STM orientiert sich an den Vorgaben des Stabilen Speichers von Lampson. Das betrifft die Zugriffe auf die “stabilen Strukturen” im STM, das sind die im STM gespeicherten Objekte. Diese Strukturen werden hier STM-Objekte genannt. Darüber hinaus werden noch weitere Ziele verfolgt, die den Umgang des Anwenders mit dem STM erleichtern sollen. Da man bei solchen Erleichterungen leicht Gefahr läuft, die Möglichkeit einer unbeabsichtigten Änderung von STM-Objekte zuzulassen, wird besonderes Augenmerk darauf gelegt, dies zu verhüten.

Folgende Anforderungen sollen für den STM erfüllt sein:

- Atomare Transaktionen auf den STM-Objekten, wobei auch mehrere STM-Objekte gleichzeitig bearbeitet werden können;
- Einfaches Programmier-Interface;
- Autonomie des STM;
- Schutzvorkehrungen vor unbeabsichtigten Änderungen an STM-Objekten;
- Schneller Zugriff auf die STM-Objekte.

Schnelle Zugriffe auf die STM-Objekte werden erreicht, indem der STM direkt an den lokalen Bus angeschlossen ist. An diesem befindet sich auch der lokale Speicher des Rechners. So können Zugriffe auf den STM ähnlich schnell ablaufen wie Zugriffe auf den lokalen Speicher. Im Falle einer Schreib-Operation sind nur die in einer Hälfte des STM abgelegten Daten erreichbar. Der Update der zweiten Hälfte wird vom intelligenten Controller ausgeführt.

Da aber üblicherweise Zugriffe auf den lokalen Speicher ohne größere Schutzvorkehrungen ablaufen, birgt dies für den STM die Gefahr, daß schon bei kleineren transienten Störungen irrtümlich auf ihn zugegriffen wird, oder daß eine falsche Stelle in ihm adressiert wird. Um die Auswirkungen solcher Fehler von vornherein zu verringern, sind die STM-Objekte üblicherweise geschlossen und müssen vor dem Beginn einer Aktion erst geöffnet werden. Nach Abschluß der Aktion sollten sie wieder geschlossen werden, damit unberechtigte Zugriffe auf diese sofort erkannt werden können. Die maximale Zahl der Objekte, die zu jedem Zeitpunkt geöffnet sein dürfen, ist begrenzt. Diese Zahl sollte nach Banâtres Angaben aus Gründen der möglichen Parallelbearbeitung von Transaktionen größer als 3 sein. Er empfiehlt 8-16 gleichzeitig geöffnete STM-Objekte. Für die Beseitigung eines dennoch entstandenen Fehlers muß das standardmäßige CLEANUP eingesetzt werden.

Ein STM-Objekt besteht aus einem zusammenhängenden Block, der die abgelegte Information trägt, und aus einem Deskriptor, der unter anderem die Ober- und Unter-Grenze des Bereichs enthält, in dem ausschließlich dieser Block abgelegt ist. Bevor ein Zugriff auf ein Objekt erfolgen darf, muß es geöffnet werden. Dabei werden die beiden Grenzen in Register eingetragen. Von diesen Registern gibt es nur eine beschränkte Zahl. Die Inhalte dieser Register repräsentieren die derzeit geöffneten Objekte. Bei jedem Zugriff auf Objekte im STM wird die angelegte Adresse mit den Grenzen der geöffneten Objekte verglichen. Ist die aktuelle Adresse in einem solchen Bereich enthalten, so wird der Zugriff auf dem STM durchgeführt. Ist sie aber nicht enthalten, so wird der Zugriff abgewiesen und dies dem Prozessor mit einer Exception signalisiert. Beim Schließen des STM-Objekts werden die Grenzen wieder aus den Registern entfernt. Dadurch ist dieses Objekt vor weiteren Zugriffen geschützt. Ferner wird wieder Platz für die Bearbeitung anderer Objekte geschaffen.

Die Ausführung von Operationen im STM muß innerhalb von Transaktion vollzogen werden. Diese stellt sicher, daß alle Aktionen im STM gemäß des Transaktionskonzepts ausgeführt werden. Dazu wird ein Zwei-Phasen-Commit-Protokoll abgearbeitet. Aktionen werden zunächst nur im ersten Speicher ausgeführt. Dabei ist es möglich, mehrere STM-Objekte zu verändern, je nach geöffneten Objekten. Diese werden in einem Transaktions-Deskriptor angegeben. Erst wenn alle Aktionen im ersten Speicher erfolgreich waren, werden die Veränderungen im zweiten Speicher nachvollzogen. Waren die Aktionen im ersten Speicher nicht erfolgreich, so werden sie durch Kopieren der alten Information vom zweiten in den ersten Speicher rückgängig gemacht. Auch nach Prozessor-Crashes können, wie bei Lampson beschrieben, hier defekte Strukturen wiederhergestellt werden.

Ähnlich wie STM-Objekte werden Transaktionen erzeugt und später auch wieder gelöscht. Ihre Konfigurationsdaten werden ebenfalls im STM abgelegt. Diese enthalten unter anderem die Angabe der benötigten Objekte.

Um schon während der Zugriffe auf die verschiedenen Datenstrukturen im STM mehr Kontrolle zu haben, ist jedes Wort mit einem Tag ausgestattet. Es zeigt an, welcher Art die in diesem Wort abgelegte Information ist.

Dabei wird unterschieden zwischen:

- freien Plätzen,
- dem Start eines Objekt-Deskriptors,
- dem Start eines Transaktions-Deskriptors und
- einer sonst belegten Stelle.

Diese Angaben werden bei allen Zugriffen auf den STM überprüft. Beispielsweise dürfen bei der Erzeugung eines STM-Objekts nur freie Plätze benutzt werden. Ebenso muß z.B. bei der oben angesprochenen Open-Operation der anzugebende Parameter, der das zu öffnende Objekt bezeichnet, auf den Beginn eines Objekt-Deskriptors zeigen. Ähnliches gilt auch für den Start einer Transaktion. Mit diesen Überprüfungen zur Laufzeit lassen sich Zugriffsfehler wenigstens zum Teil sofort erkennen, bevor ein Schaden (z.B. durch falsche Programmierung oder Adressierung) eingetreten ist.

Bei einem Multiprozessorsystem ist es oft notwendig, eine Gesamt-Transaktion aus mehreren Einzel-Transaktionen zusammenzusetzen. Dies gilt, wenn Teile der Gesamt-Transaktion in verschiedenen STMs durchgeführt werden müssen. Die gesamte Aktion soll nur dann als erfolgreich betrachtet werden können, wenn alle einzelnen Teil-Transaktionen erfolgreich waren. Dazu wird ein zusätzliches Kommando eingeführt, ein “Prepare_to_Commit”, das alle beteiligten Transaktionen am Ende der ersten Phase im Fall eines erfolgreichen Update im 1. Speicher an einen Koordinator schicken. Ansonsten senden sie “Abort”. Wenn dieser Koordinator von allen Transaktionen ein “Prepare_to_Commit” erhalten hat, sendet er das Kommando “Commit”, ansonsten “Abort”. Die beteiligten Transaktionen warten auf diese Mitteilung und führen diese Order nach Erhalt aus. Auf diese Weise läßt sich ein verteiltes Commit-Protokoll verwirklichen.

2.3 Absicht zur Implementierung eines Stabilen Speichers für MEMSY

Im nächsten Abschnitt wird näher auf das Multiprozessorsystem MEMSY eingegangen. Dabei interessiert vor allem die Interknotenkommunikation über gemeinsame Speicher, die Kommunikationsspeicher. Aus dieser Betrachtung heraus werden anschließend diejenigen Fehler betrachtet, die an dieser Stelle am ehesten zu erwarten sind. Besonders solche sind dann von Interesse, die von einer Hardware erkannt und gegebenenfalls korrigiert werden können. Dies mündet dann in die Beschreibung für den Entwurf eines Stabilen Speichers für MEMSY.

3 Das Multiprozessorsystem MEMSY

Die Beschäftigung mit speichergekoppelten Multiprozessoren hat an der Universität Erlangen-Nürnberg schon Tradition, wie es die Projekte um den EGPA-Rechner [HÄN76] und um das DIRMU-System [HMW85] zeigen. In Fortsetzung dieser Reihe wurde das Multiprozessorsystem MEMSY entworfen. An ihm werden auf vielen Gebieten verschiedene Untersuchungen bezüglich Multiprozessoren durchgeführt [FRH89].

Standen zunächst vor allem Untersuchungen zur Leistungsfähigkeit (Rechenleistung, Verbindungsstrukturen [HIL92]) im Vordergrund, so wurden in jüngerer Zeit in immer stärkerem Maß auch Fehlertoleranz-Aspekte betrachtet. Neben den Untersuchungen zur schnellen Fehlererkennung werden Maßnahmen zur Fehlerbehebung, insbesondere Wiederaufsetzverfahren mit Hilfe von Sicherungspunkten, untersucht [DCH91]. Zur Anwendung von Sicherungspunkten müssen drei Dinge funktionieren:

- Der Sicherungspunkt muß regelmäßig nach einer gewissen Zeit erstellt wird, die deutlich unter der MTBF des Rechners liegt.
- Die Daten des Sicherungspunkts müssen ohne Fehler abgespeichert und auch mindestens bis zur nächsten Sicherungspunkterstellung fehlerfrei erhalten werden.
- Das Lesen der Daten muß auch dann noch möglich sein, wenn nach dem Auftreten eines Fehlers ein Prozessor inaktiv wird. Dann muß ein anderer Prozessor seine Rolle übernehmen können.

An MEMSY werden auch Untersuchungen zur Adaptierbarkeit von Betriebssystemen an größenvarianten (speichergekoppelten) Multiprozessoren durchgeführt. Schließlich werden auf der Anwenderseite die Möglichkeiten zur Parallelisierung von Algorithmen untersucht, die auf speichergekoppelten Multiprozessoren wie z.B. MEMSY laufen sollen. Dabei soll der Anwender in gewissem Maß Einblick in die Topologie von MEMSY haben, damit er die umfangreichen Kommunikationsmöglichkeiten innerhalb des Systems vorteilhaft nutzen kann.

An dieser Stelle soll nun die Topologie von MEMSY vorgestellt werden. Anschließend werden die “zu erwartetenden” Fehler diskutiert, die dann bei der Integration eines Stabilen Speichers berücksichtigt werden müssen.

MEMSY besteht aus einer Menge von Knotenrechnern [DGH93a]. Die einzelnen Knotenrechner sind auf zwei Ebenen verteilt, auf die untere (A-) Ebene und auf die obere (B-) Ebene. Innerhalb einer Ebene sind die Knotenrechner feldartig angeordnet, wobei die Ränder torusartig geschlossen sind. Das bedeutet, daß alle Knotenrechner innerhalb einer Ebene, auch die im Bild am Rand erscheinenden, vier Nachbarknoten besitzen.

Zwischen den beiden Ebenen gibt es ebenfalls eine feste Nachbarschaftsbeziehung. Vier Knoten aus der A-Ebene, die in einem Quadrat angeordnet sind, und ein Knoten aus der B-Ebene bilden zusammen eine Pyramide. Die Kanten der Pyramide stellen die Nachbarschaftsbeziehun-

gen dar. Setzt man mehrere solche Pyramiden feldartig nebeneinander, so erhält man ein größeres Multiprozessorsystem. Beispiele für ein MEMSY mit fünf bzw. mit zwanzig Knoten sind unten dargestellt (siehe Abb. 4).

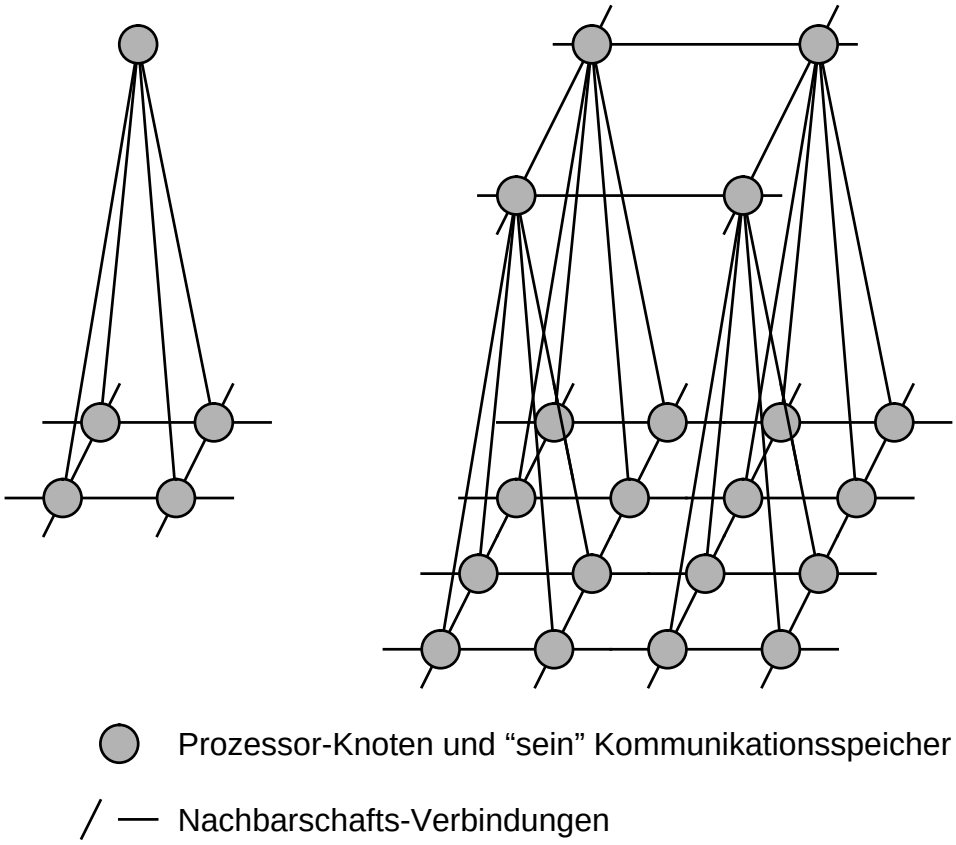


Abb. 4 **MEMSY mit 5 Knoten (Elementar-Pyramide)** **MEMSY mit 20 Knoten : 4 Knoten in der B-Ebene, 16 Knoten in der A-Ebene**

Ein einzelner Knoten besteht aus einem Prozessor-Knoten und einem Kommunikationsspeicher. Innerhalb jeder Ebene hat jeder Prozessor Zugriff auf seinen eigenen Kommunikationsspeicher und auf die Kommunikationsspeicher seiner vier Nachbarn (siehe Abb. 5). Die Prozessoren in der B-Ebene haben zusätzlich Zugriff auf die Kommunikationsspeicher ihrer untergeordneten vier Knoten. Über die Kommunikationsspeicher können die einzelnen Knoten Informationen mit ihren Nachbarknoten austauschen.

Entsprechend der Nachbarschaftsbeziehungen, die durch die Verbindungen zu den Kommunikationsspeichern gegeben ist, existiert auch eine Interruptkopplung zwischen den Prozessoren. Diese besteht immer aus einer gegenseitigen Interrupt-Möglichkeit, auch wenn die Zugriffsmöglichkeiten der beteiligten Prozessoren zu den Kommunikationsspeichern nur einseitig sind (B-Prozessoren können auf die Kommunikationsspeicher der A-Ebene zugreifen, die A-Prozes-

soren aber nicht auf die Kommunikationsspeicher in der B-Ebene).

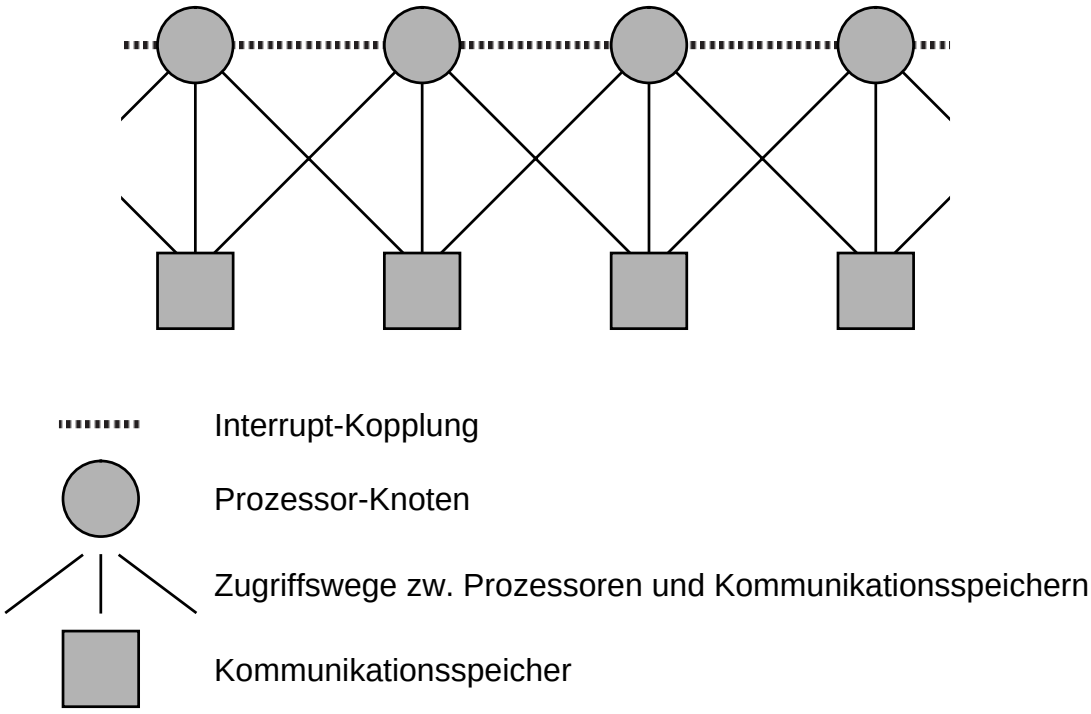


Abb. 5 Verbindung der einzelnen Knoten innerhalb einer Ebene (dargestellt ist eine einzelne Reihe)

3.1 Die Schnittstelle für die Speicherkopplung

Für MEMSY wurden die Prozessor-Knoten aus üblichen Motorola-Systemen der M88K-Reihe errichtet. Sie enthalten als Prozessoren bis zu viermal den MC88100-Chip und bis zu achtmal den zugehörigen Cache- und Memory-Management-Bausteinen MC88200 oder MC88204 (CMMU). Weiterhin sind die Systeme mit lokalem Speicher (16 bzw. 32 MByte Hauptspeicher) und mit allen üblichen Serviceeinrichtungen ausgestattet, die für ein Single-Computer-System erforderlich sind.

Für den Einsatz als speichergekoppeltes Multiprozessorsystem war die Ergänzung des Motorola-Systems um ein Interface erforderlich [SFB93]. Dafür mußte eine geeignete Hardware-Schnittstelle mit Steckverbinder gesucht und ein entsprechendes “Prozessorinterface” (PI) ent-

wickelt werden (siehe Abb. 6).

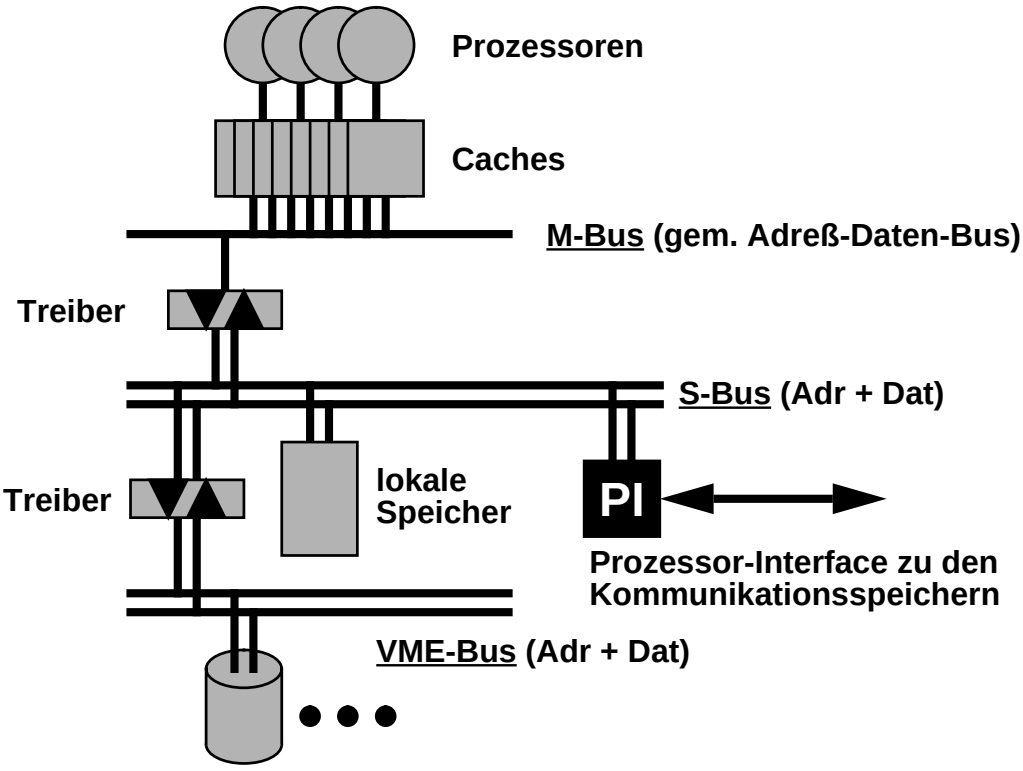


Abb. 6 Schnittstellen innerhalb des M88K-Systems

Vom Prozessor aus gesehen ist die erste Schnittstelle der M-Bus (Memory Bus). Die nächste Schnittstelle ist der S-Bus (Slave Bus), über den das Prozessorboard mit dem lokalen Speicherboard und dem System Controller Board verbunden ist (Kerneinheit). Schließlich gibt es den VME-Rückwand-Bus, über den sämtliche Teilsysteme eines Prozessorknotens miteinander verbunden sind [MVM89]. Die Entscheidung fiel zugunsten der S-Bus-Schnittstelle aus. Diese hat gegenüber den beiden anderen Varianten einige Vorteile, die hier kurz dargestellt werden.

Die S-Bus-Schnittstelle besteht aus zwei üblichen 96-poligen Steckern, die häufige Steckvorgänge erlauben. Dies ist besonders in der Erprobungsphase ein oft vorkommender Vorgang. Dagegen hat der M-Bus-Stecker (3 x 100 Pins) einen wesentlich empfindlicheren Aufbau, so daß häufige Steckvorgänge nicht ratsam sind. Ferner ist zu berücksichtigen, daß der Anschluß weiterer Schaltkreise an einer Schnittstelle eine zusätzliche elektrische Belastung darstellt, so daß sich die Zeitverhältnisse ändern können. Da die M-Bus-Schnittstelle in diesem Punkt sehr enge Toleranzen einhalten muß, wurde von einer Implementierung des Prozessorinterfaces an dieser Stelle abgesehen.

Der VME-Bus schneidet unter diesen Gesichtspunkten wesentlich günstiger ab und würde den oben genannten Forderungen genügen. Daß der VME-Bus dennoch nicht als Schnittstelle für

das Prozessorinterface genommen wurde, hat mehrere Gründe:

Zum einen liegt die VME-Bus-Schnittstelle am weitesten vom Prozessor entfernt, d.h. es müssen mehr Durchlaufzeiten durch verschiedene Stufen (ICs) abgewartet werden als bei den beiden anderen Schnittstellen, bevor die Signale diese Schnittstelle erreichen. Und zum zweiten verfügt der VME-Bus nicht über die Möglichkeit, einen Übertragungsfehler in seinem Einzugsbereich erkennen zu lassen. Bei M- und S-Bus werden immerhin Paritybits mit übertragen [MVM89].

Diese dargelegten Gründe führten dazu, den S-Bus als Schnittstelle für das Prozessorinterface auszuwählen. Mit seinem Aufbau befassen sich die folgenden Abschnitte.

3.2 Die Schnittstellen des Prozessorinterface

Das Prozessorinterface bildet die Schnittstelle zwischen dem Prozessorknoten (Motorola-System) und den Übertragungswegen zu den Kommunikationsspeichern. Aufgabe dieser Schnittstelle ist, die Signale, die das Motorola-System bei einem Speicherzugriff am S-Bus anbietet, in die Form umzuwandeln, die die Kommunikationsspeicher erwarten. Ebenso müssen die Signale von den Kommunikationsspeichern in entsprechender Weise an das Motorola-System weitergeleitet werden. Die betroffenen Signale am S-Bus sind die des Adreßbusses, die des Datenbusses und einige Steuersignale. In Tab. 1 ist eine Gegenüberstellung der Signale des S-Busses und die der Übertragungsstrecke angegeben.

Aus dieser Gegenüberstellung der vorhandenen bzw. benötigten Signale ist erkennbar, daß verschiedene Signalumwandlungen erforderlich sind.

Auf der S-Bus-Seite liegen getrennte Adreß- und Daten-Busse vor, während auf der Übertragungsstrecke zu den Kommunikationsspeichern ein im Zeitmultiplex betriebener Adreß-Daten-Bus vorhanden ist. Auf der S-Bus-Seite wird nur die Dateninformation mit 4 Paritybits (für 4 Bytes je ein Paritybit) gegen Übertragungsfehler abgesichert. Auf der Übertragungsstrecke zu den Kommunikationsspeichern aber wird sowohl die Adreßinformation als auch die Dateninformation mit 4 Paritybits geschützt. Daher muß im Prozessorinterface zumindest eine Parity-Erzeugung vorhanden sein. Das Checken der Paritybits bildet die Grundlage für die Fehlererkennung. Dies findet immer am Ende einer Übertragungsstrecke statt. Beim Schreiben werden sowohl die Adresse als auch die Daten vom Prozessor zum Kommunikationsspeicher transportiert und somit beide am Kommunikationsspeicher überprüft. Beim Lesen wird nur die Adresse zum Kommunikationsspeicher transportiert und dort gecheckt. Die Daten werden dagegen im Prozessorinterface auf gültige Parity kontrolliert.

Signale am S-Bus	Signale an der Übertragungsstrecke
Adreßbus:	gemultiplexter Adreß-Daten-Bus:
30 Adreßbits (SA[2..31])	32 Informationsbits (AD[0..31])
Datenbus:	4 Paritybits (ADP[0..3])
32 Datenbits (SD[0..31])	
4 Paritybits (SDP[0..3])	
Steuersignale:	Steuersignale:
4 Byte Enable (BE[0..3])	4 Write Byte (WB[0..3])
1 Read/Write (RD)	1 Adreß Strobe (ASTR)
1 Last Data Transfer (LDT)	1 Data Strobe (DSTR)
1 Lock (LK)	1 Lock (ACTIV)
1 Adreß-/Daten-Phase (AP)	
3 Reset (global, lokal, Power Up)	
4 Select (MEMB[0..3])	
Antwortsignale:	Antwortsignale:
1 Wait (entspricht Ready, RDY)	1 Fertig (MRDY)
1 Fehler (ERR)	1 Fehler (PERR)

Tab. 1 Gegenüberstellung der Signale am S-Bus und an der Übertragungsstrecke

Bezüglich der Steuer- und Antwort-Signale müssen die Protokolle von S-Bus und von der Übertragungsstrecke zu den Kommunikationsspeichern gegenseitig umgesetzt werden. Zwar werden beide Verbindungen für Speicherzugriffe eingesetzt, ihnen liegen aber unterschiedliche Signalfolgen zugrunde. Zum besseren Verständnis wird die Funktionsweise dieser Signale an den Lese- und Schreib-Operationen erklärt, die über diesen Bus abgewickelt werden.

3.2.1 Signale bei Speicherzugriffen am S-Bus

Am S-Bus wird ein einfaches Speicherzugriffsprotokoll abgearbeitet. Dies muß in ein anderes, wenn auch ähnliches Speicherzugriffsprotokoll für die Kommunikationsspeicher umgesetzt werden.

Die möglichen Speicherzugriffsoperationen sind:

- Schreiben eines einzelnen Wortes (oder eines Halbwortes oder eines Bytes)
- Lesen eines einzelnen Wortes (oder eines Halbwortes oder eines Bytes)
- Burst-Schreiben (Schreiben von 4 Worten, die an aufeinanderfolgenden Adressen liegen)
- Burst-Lesen (Lesen von 4 Worten, die an aufeinanderfolgenden Adressen liegen)
- Durchführung eines Read Modify Write - Befehls (RMW-Befehls) : XMEM-Operation

Zunächst wird der Ablauf beim Schreiben bzw. Lesen eines einzelnen Wortes erklärt. Die drei anderen Zugriffsoperationen lassen sich danach leicht daraus ableiten.

3.2.1.1 Einzelwort-Zugriff auf den Speicher über den S-Bus

Die Durchführung eines Speicherzugriffs wird von der CMMU gesteuert. Der Prozessor stößt die CMMU dazu nur an und wartet auf ihre Fertigmeldung von ihr.

Das Motorola-M88K-System verfügt über einige *Select*-Signale (MEMB[0..3]), von denen beim Speicherzugriff genau ein Signal aktiviert wird (MEMBi = 0). Es gibt an, auf welchem Speicherboard dieser Zugriff durchgeführt werden soll. Dieses Signal bleibt so lange aktiv, bis der Zugriff beendet ist. Während dieser Zeit liegt die Adresse auf dem Adreßbus an. Auch das *Schreib-Lese*-Signal (RD) ist von Beginn des Zugriffs bis zum Ende gültig. Die restlichen Steuersignale sind nicht während der ganzen Übertragungsphase gültig, da sie auf Leitungen auf dem S-Bus angeboten werden, die im Zeitmultiplex betrieben werden. Ein Signal, das *Adreß-Daten-Phase*-Signal (AP), entscheidet, welche Bedeutung die restlichen Steuersignale haben. In der Adreßphase (AP = 1) ist das *Lock*-Signal gültig. Es gibt an, ob es sich um einen Einzelwort-Transfer oder um einen Burst-Transfer handelt (LK = 0), oder ob ein RMW-Befehl ausgeführt wird (LK = 1). In der Adreßphase gibt es noch drei andere Steuersignale, die für die Cache-Kohärenz von Bedeutung sind. Für den Entwurf des Prozessorinterfaces sind sie aber nicht von Bedeutung. In der Datenphase (AP = 0) gibt das *Last Data Transfer* - Signal (LDT) an, ob nach der Übertragung eines Wortes noch weitere Transfers folgen (Burst-Transfer: LDT = 0) oder nicht (LDT = 1). Ferner geben in der Datenphase 4 *Byte Enable* - Signale (BE[0..3]) an, welche Bytes aus dem übertragenen Wort am Ziel relevant (BEi = 1) sind. Da über die Übertragungsstrecke aber immer 4 Byte (= 1 Wort) gleichzeitig transportiert werden, sind die Byte Enable - Signale für die Durchführung des Transfers selbst nicht von Bedeutung. Erst der Empfänger des Wortes (beim Schreiben der Speicher, beim Lesen der Prozessor bzw. die CMMU) wertet diese Information aus.

Nach dem Start einer Übertragung liegt auf dem S-Bus als Antwort “Warte” (Wait-Signal) vor. Die CMMU wartet nun solange, bis eine andere Antwort ausgegeben wird, i.a. “Ready” oder “Error”. Im Fall eines Lesezugriffs kann die CMMU nun die Daten vom S-Bus übernehmen. Im Fall eines Schreibzugriffs hat der Speicher die Daten zu diesem Zeitpunkt bereits übernommen. Als letzte Aktion setzt der Adreßdekoder das Select-Signal wieder auf inaktiv [MVM89] (siehe

auch Abb. 7).

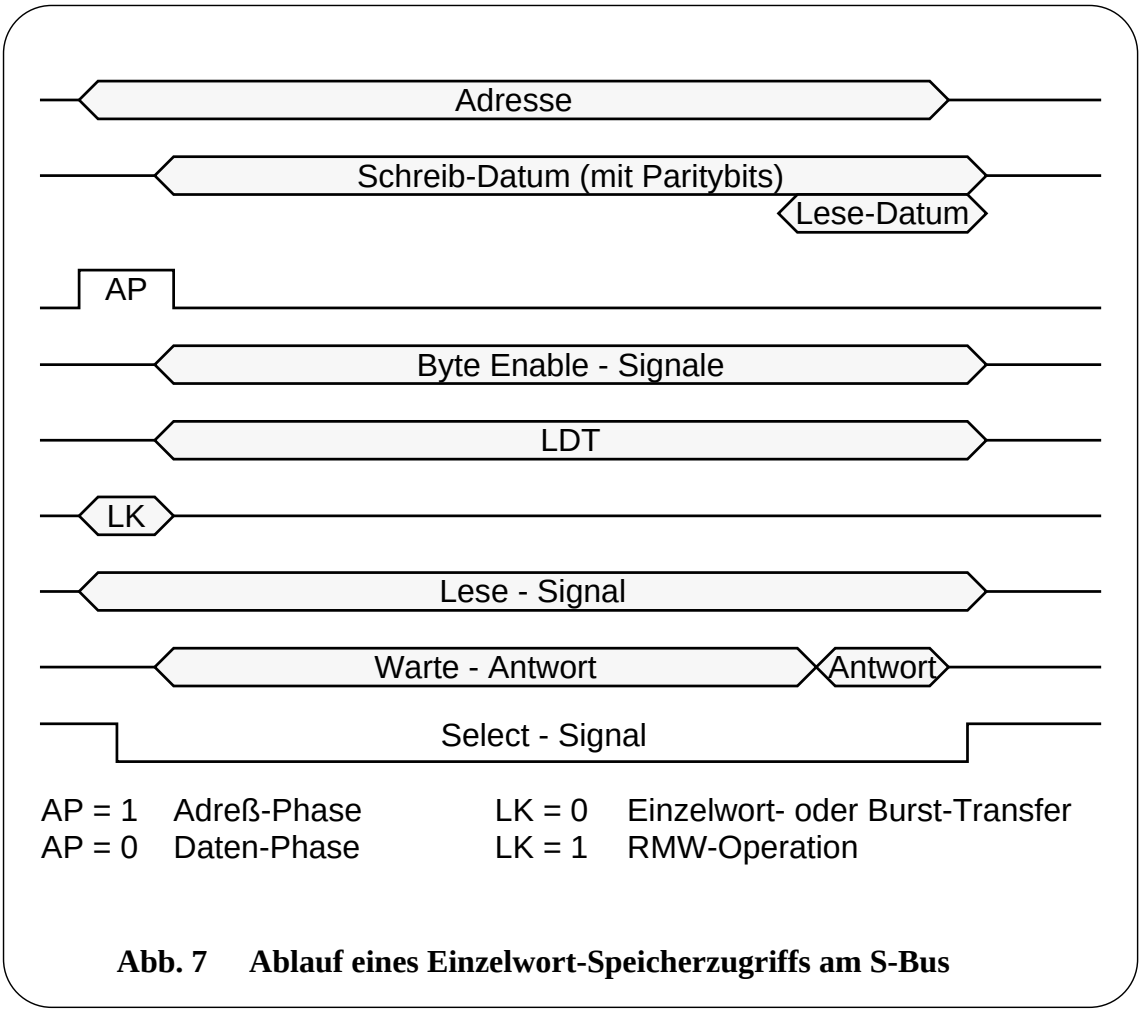


Abb. 7 Ablauf eines Einzelwort-Speicherzugriffs am S-Bus

3.2.1.2 Burst-Zugriff auf den Speicher über den S-Bus

Bei einem Burst-Zugriff werden vier Worte, die an aufeinanderfolgenden Adressen liegen, zwischen dem Speicher und der CMMU transportiert. Diese Fähigkeit setzt die CMMU ein, um eine Cache-Zeile möglichst schnell zwischen Speicher und Cache zu transportieren. Diese Fähigkeit kann allerdings nicht unmittelbar in einem Anwenderprogramm genutzt werden, sondern bleibt dem Betriebssystem vorbehalten.

Das Erscheinungsbild eines Burst-Zugriffs auf dem S-Bus ähnelt sehr dem Einzelwort-Transfer. Das Select-Signal bleibt während des ganzen Vier-Wort-Transfers aktiv; und auch das Lock-Signal wird hier nicht gesetzt. Nur am LDT-Signal kann der Empfänger (der Speicher) den Burst-Zugriff erkennen. Beim Transfer der ersten drei Worte wird “nicht LDT” angezeigt; und erst beim 4. Transfer erscheint “LDT”.

Innerhalb des Burst-Zugriffs wird das Ende jeder einzelnen Wortübertragung genauso wie beim Einzelwort-Transfer durch das Wait-Signal gekennzeichnet.

3.2.1.3 Read-Modify-Write - Operation (RMW)

Der Prozessor MC88100 besitzt als einzige RMW-Operation den “XMEM”-Befehl. Bei seiner Ausführung wird ein Wort aus dem Speicher geholt und in ein Register geladen. Der ursprüngliche Registerinhalt wird in die angegebene Speicherzelle geschrieben. Der Speicher ist während des ganzen Vorgangs für andere Prozessoren bzw. CMMUs gesperrt.

Die Steuersignale auf dem S-Bus zeigen das gleiche an wie bei einem Einzelwort-Lesezugriff und nach einer kurzen Idle-Phase das eines Schreibzugriffs. Der Unterschied zwischen den RMW- und den Einzelwort-Zugriffen ist am Lock-Signal erkennbar. Dieses ist ausschließlich bei RMW-Zugriffen gesetzt. Allerdings ist das Lock-Signal nur während der Adreßphase (AP = 1) am S-Bus sichtbar.

An dieser Stelle ist anzumerken, daß das Select-Signal für diese Speicherzugriffe nicht während der gesamten Aktion, sondern nur während der Lese- und der Schreib-Phase gesetzt ist. In der dazwischenliegenden Modify-Phase ist es nicht gesetzt. Daher kann die Durchführung einer RMW-Operation ausschließlich am anfangs gesetzten Lock-Signal erkannt werden.

3.2.2 Steuersignale für die Kommunikationsspeicher

Die Datenübertragung zwischen Prozessorinterface und den Kommunikationsspeichern verläuft über einen Parallelbus, über den im Zeitmultiplex Adressen und Daten übertragen werden. Dabei sind folgende Übertragungsarten möglich:

- Schreiben eines einzelnen Wortes
- Lesen eines einzelnen Wortes
- Burst-Schreiben (Schreiben von 4 Worten, die an aufeinanderfolgenden Adressen liegen)
- Burst-Lesen (Lesen von 4 Worten, die an aufeinanderfolgenden Adressen liegen)
- atomare Durchführung einer Read Modify Write - Operation mit einem Wort

Die Übertragungen einzelner Bytes oder Halbworte wird (wie beim S-Bus) in der Weise durchgeführt, daß immer ein ganzes Wort übertragen wird, aber der Empfänger aus der Byte Enable-Information die relevanten Bytes erkennen kann. Bei der Überprüfung werden die Paritybits der irrelevanten Datenbytes ignoriert.

Für die Übertragung eines einzelnen Wortes ist zunächst nur die Transfer-Richtung (lesend oder schreibend) wichtig. Im Fall eines Schreibzugriffs ist für den Kommunikationsspeicher die Byte Enable-Information wichtig. Dies zusammen läßt sich auf einfache Weise in 4 Steuersignalen, den *Write Byte* - Signalen (WB[0..3]) kodieren, wie die folgende Tabelle (Tab. 2) veranschaulicht: Aus dem Kommunikationsspeicher wird immer dann gelesen, wenn

alle Write Byte-Signale auf 1 gesetzt sind. Geschrieben wird immer dann, wenn mindestens ein WBi-Signal auf 0 gesetzt ist. Gegenüber einem separaten Schreib-Lese-Signal und 4 Byte Enable-Signalen hat diese Kodierung praktisch keine Einschränkung. Nur eine Schreiboperation mit 0 Bytes kann nicht kodiert werden, was auch nicht notwendig ist.

Operation	S-Bus-Signale		Signale zur Übertragungsstrecke
	RD	BE[0..3]	WB[0..3]
Lesen, 1 Wort :	1	1 1 1 1	1 1 1 1
Lesen, 1 Halbwort :	1	1 1 0 0	1 1 1 1
	1	0 0 1 1	1 1 1 1
Lesen, 1 Byte :	1	1 0 0 0	1 1 1 1
	1	0 1 0 0	1 1 1 1
	1	0 0 1 0	1 1 1 1
	1	0 0 0 1	1 1 1 1
Schreiben, 1 Wort :	0	1 1 1 1	0 0 0 0
Schreiben, 1 Halbwort :	0	1 1 0 0	0 0 1 1
	0	0 0 1 1	1 1 0 0
Schreiben, 1 Byte :	0	1 0 0 0	0 1 1 1
	0	0 1 0 0	1 0 1 1
	0	0 0 1 0	1 1 0 1
	0	0 0 0 1	1 1 1 0

Tab. 2 Gegenüberstellung der Schreib-Lese-Steuersignale auf dem S-Bus und der für die Kommunikationsspeicher benötigten WBi-Signale

Prinzipiell könnten auch noch andere Schreiboperationen mit den WBi-Signalen kodiert werden wie z.B. das Schreiben von 3 Bytes oder das Schreiben von zwei nicht zusammenhängenden Bytes. Das Prozessorinterface und die Kommunikationsspeicher lassen das zu. Jedoch kommt solch ein Auftrag bei dem Motorola-Prozessor bzw. bei der CMMU nicht vor.

In den beiden folgenden Abschnitten werden Einzelwort- und Mehrwort-Transfers, wozu Burst- und RMW-Operationen gehören, näher betrachtet.

3.2.2.1 Einzelwort-Transfer auf der Übertragungsstrecke

Da es sich bei der Übertragungsstrecke um einen im Zeitmultiplex betriebenen Bus handelt, müssen die beiden Phasen, in denen eine Adresse (Adreß-Phase) bzw. ein Wort (Daten-Phase) übertragen wird, durch Steuersignale unterschieden werden.

In der Adreß-Phase wird das ASTR-Signal vom Prozessorinterface ausgegeben. Der Beginn der Daten-Phase wird vom Prozessorinterface durch das Aktivieren des DSTR-Signals angezeigt. Bis zu diesem Zeitpunkt hat der Kommunikationsspeicher keine Möglichkeit, auf das Zeitverhalten des Zugriffsablaufs Einfluß zu nehmen. Dies bedeutet vor allem, daß er die übertragene Adresse bei sich ablegen muß. Erst in der Daten-Phase bestimmt er durch das Ausgeben des Fertig-Signals (MRDY), wann der Transfer vom Prozessorinterface beendet werden kann. Für den Kommunikationsspeicher ist der Zugriff mit dem Aussenden des MRDY-Signals beendet, d.h. auch auf der Übertragungsstrecke ist der Transfer dann beendet (siehe auch Abb. 8).

Zu Beginn des Zugriffs wird zunächst die Adresse auf die Übertragungsstrecke gelegt. Kurz nach dem Erscheinen der Adresse wird das ASTR-Signal aktiviert, um dem Kommunikationsspeicher anzuzeigen, daß er jetzt die Adresse übernehmen muß. Die fallende Flanke des ASTR-Signals kann dabei als Zeitpunkt zur Übernahme der Adresse verwendet werden. Aus diesem Grund ist das ASTR-Signal gegenüber der Zeit, in der die Adresse auf dem Bus getrieben wird, leicht verschoben. Nach etwa 120 nsec wird das ASTR-Signal wieder inaktiv. Die Adresse wird zu diesem Zeitpunkt schon nicht mehr getrieben (siehe Abb. 8).

Zu Beginn der Datenphase erscheint das DSTR-Signal für die Dauer eines Taktes (40 nsec). Während dieses Zeitraums müssen die WBi-Signale gültig sein, denn sie werden zu dieser Zeit vom Kommunikationsspeicher übernommen. Diese Signale entscheiden die Transferrichtung (lesend oder schreibend).

Beim Schreibzugriff wird zu Beginn der Daten-Phase, d.h. gleich im Anschluß an die Adresse, das zu transferierende Wort auf den Bus gelegt. Das Prozessorinterface wartet nun so lange, bis der Kommunikationsspeicher das MRDY-Signal ausgibt. Dabei ist der Zeitpunkt von Bedeutung, zu dem das MRDY-Signal eine steigende Flanke besitzt (Wechsel von 0 auf 1). Das Prozessorinterface beendet nun den Schreibzugriff, indem es das RDY-Signal über den S-Bus an die CMMU weitergibt.

Bei einem Lesezugriff (WB[0..3] = 1111) wird zu Beginn der Daten-Phase die Transferrichtung des Busses umgekehrt. Das Prozessorinterface erwartet das ausgegebene Wort vom Kommunikationsspeicher. Bei der steigenden Flanke des MRDY-Signals wird das Wort im Prozessorinterface übernommen und die Paritybits auf Korrektheit überprüft. Ist alles in Ordnung, so wird dieses Wort über den S-Bus an die CMMU mit einem RDY-Signal weitergegeben. Ist ein Übertragungsfehler an einem falschen Paritybit erkannt worden, so wird an Stelle des RDY-Signals das ERR-Signal an die CMMU weitergegeben.

Eine weitere Situation für die Beendigung eines Zugriffs auf einen Kommunikationsspeicher mit einer ERR-Antwort ist dann gegeben, wenn dieser ein PERR-Signal ausgibt. Dies wird immer dann aktiviert, wenn bei der Übertragung vom Prozessorinterface zum Kommunikationsspeicher ein Parity Error aufgetreten ist, den die Logik am Speicher erkennt. Die möglichen Fehler sind in diesem Fall ein Fehler bei der Übertragung der Adresse oder ein Fehler bei der Übertragung des zu schreibenden Wortes. Wichtig ist in diesem Fall, daß das PERR-Signal (fallende Flanke des PERR-Signals) vom Kommunikationsspeicher spätestens bei der steigenden

Flanke des MRDY-Signals aktiviert wird. Es darf aber auch früher erscheinen. Dann kann der Speicherzugriff bereits früher abgebrochen werden. Eine sich daraus ergebende Konsequenz ist, daß gewährleistet sein muß, daß wirklich nur im Fall eines erkannten Übertragungsfehlers das PERR-Signal aktiv werden darf. In allen anderen Fällen, d.h. auch in dem Fall, daß gerade kein Zugriff über diese Übertragunsstrecke abgewickelt wird, muß das PERR-Signal auf inaktiv gesetzt sein.

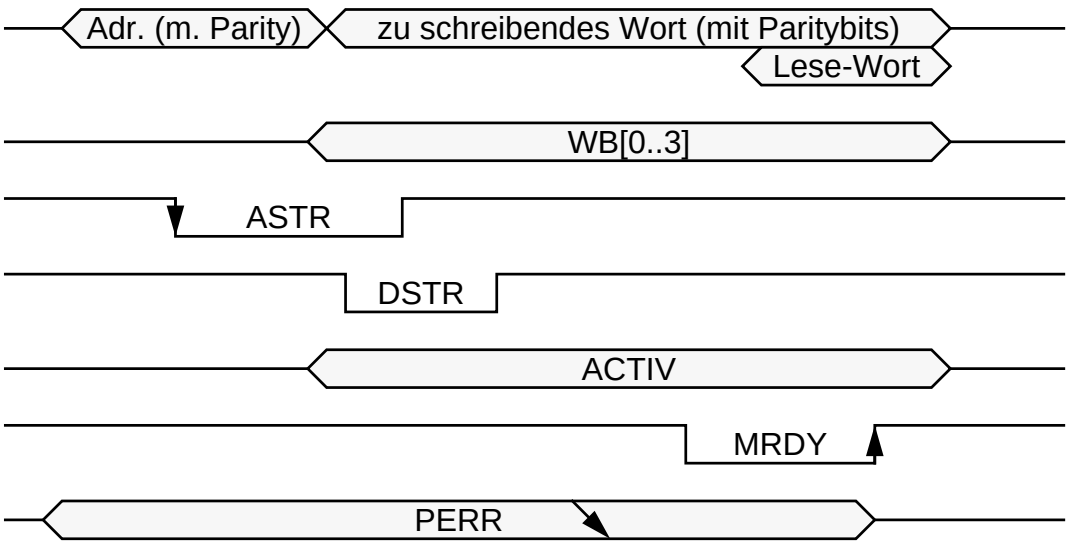


Abb. 8 Ablauf eines Einzelwort-Speicherzugriffs auf der Übertragungsstrecke zwischen dem Prozessorinterface und den Kommunikationsspeichern

3.2.2.2 Burst-Transfer und RMW-Operation auf der Übertragungsstrecke

Zwischen dem Prozessorinterface und den Kommunikationsspeichern gibt es ein weiteres Steuersignal, das bisher noch nicht erwähnt worden ist, das ACTIV-Signal. Es dient dazu, dem Kommunikationsspeicher zu signalisieren, daß nach dem gerade laufenden Transfer (ein Wort lesen oder schreiben) mindestens ein weiterer Transfer folgt, der zu der laufenden Operation gehört. Dadurch können das Burst-Schreiben und -Lesen und RMW-Operationen als nicht unterbrechbare Aktionen mit dem Kommunikationsspeicher durchgeführt werden. Im Fall des Einzelwort-Transfers ist das ACTIV-Signal auf inaktiv (= 1) gesetzt. In den anderen angegebenen Fällen wird es ab dem ersten DSTR-Signal vom Prozessorinterface auf aktiv (= 0) gesetzt. Im Fall eines Burst-Transfers bleibt es bis zum vorletzten (3.) Wort-Transfer gesetzt und wird während des letzten (4.) Transfer zurückgenommen (siehe Abb. 9).

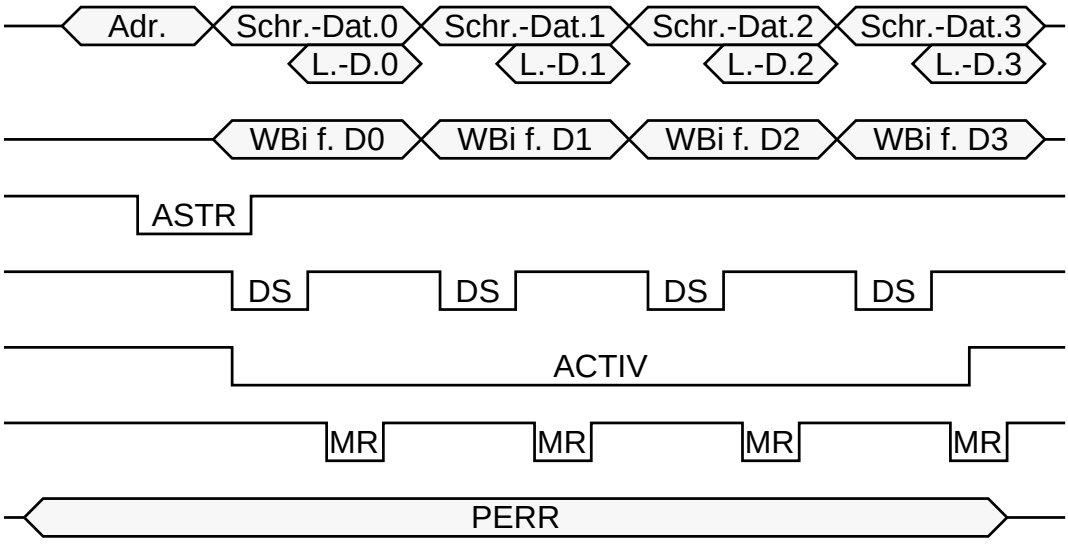


Abb. 9 Burst-Zugriff über die Übertragungsstrecke zum Kommunikationsspeicher

Bei der Durchführung einer RMW-Operation wird das ACTIV-Signal ebenfalls gesetzt und erst während der Schreib-Operation zurückgenommen (siehe Abb. 10).

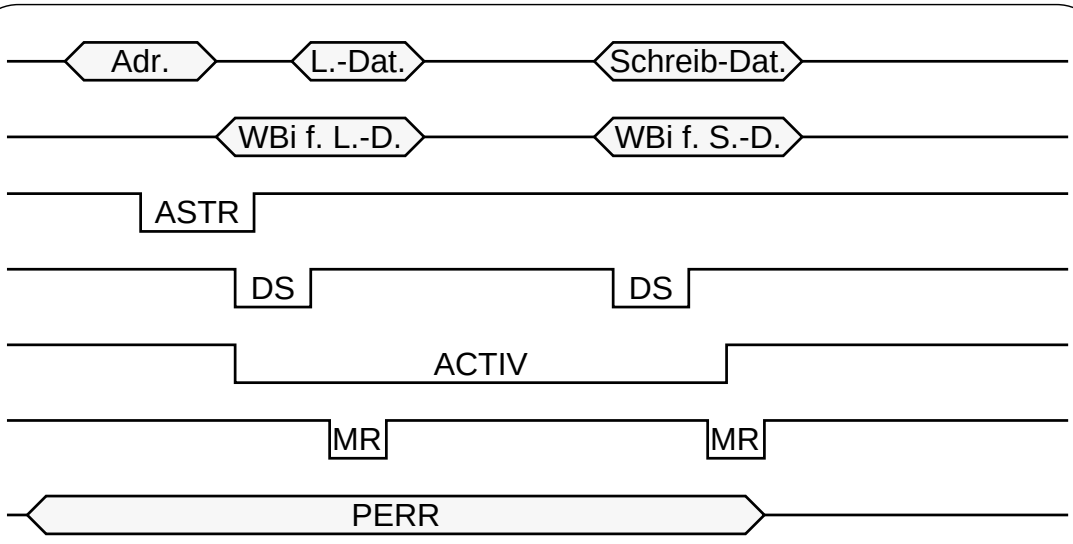
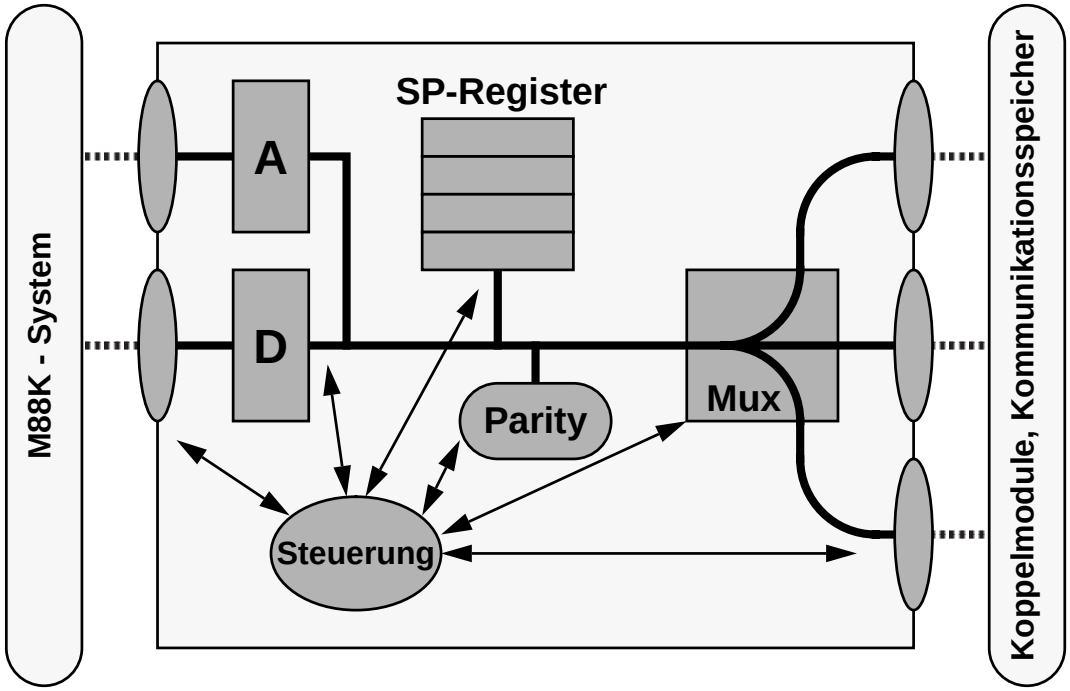


Abb. 10 RMW-Operation über die Übertragungsstrecke zum Kommunikationsspeicher

3.3 Die Struktur des Prozessorinterface

Das Interface, das es dem Prozessor ermöglicht, auf die Kommunikationsspeicher zuzugreifen, hat zwei Schnittstellen, eine zum S-Bus, an den es an das Motorola-System angeschlossen ist, und eine zur Übertragungsstrecke, über die die Kommunikationsspeicher erreichbar sind. Dazwischen erfolgt die Umsetzung der Signale und eine Pufferung der übertragenen Daten. Die Pufferung ist notwendig, da der Kommunikationsspeicher bei einer Lese-Operation nur eine gewisse Zeit lang die Daten treibt. Diese Zeit ist nicht ausreichend, um die Daten sicher an den Prozessor bzw. die CMMU zu übergeben. Die Pufferung hat auch den Vorteil, daß genau das empfangene Wort auf Parity überprüft wird, welches kurz danach an den Prozessor übergeben wird, und nicht ein Wort, das noch Störungen von der Übertragungsstrecke her ausgesetzt ist.

Einen Überblick über das Prozessorinterface (PI) gibt Abb. 11.



A: Adreß-Schnittstelle
D: Daten-Schnittstelle
Mux: Multiplexer (Wegewahl)
Parity: Parity-Erzeugung / -Überprüfung

Abb. 11 Prozessorinterface

Innerhalb der Schnittstelle werden sowohl für Adressen als auch für Schreibdaten die Paritybits erzeugt und zur Übertragung bereitgestellt. Hintereinander werden dann Adresse und (bei

Schreib-Operationen) das zu übertragende Wort auf den Übertragungsweg gelegt.

Auf der Seite, die den Kommunikationsspeichern zugewandt ist, stehen drei Ports zur Verfügung. An diese kann je ein Kommunikationsspeicher oder ein “Koppelmodul” (siehe auch Abschnitt 3.5) angeschlossen werden. Das Koppelmodul ermöglicht weitere Wegeverzweigungen zu den Kommunikationsspeichern.

Die Auswahl, über welchen Port ein Zugriff geführt werden soll, wird durch zwei Bits aus der übertragenen physikalischen Adresse ermittelt. Von den vier Möglichkeiten, die mit zwei Bit dargestellt werden können, werden drei Kombinationen (‘00’, ‘01’, ‘10’) zur Wegewahl bei einem Zugriff auf die Kommunikationsspeicher verwendet. In diesen Fällen werden die Steuersignale und die Adreß- und Daten-Informationen in der vorher beschriebenen Weise (über die Koppelmodule) zum ausgewählten Kommunikationsspeichern übertragen. An den anderen beiden Ports dieses Interfaces bleiben alle Signale in Ruheposition.

Die vierte Kombination (‘11’) dient zur Auswahl von Zugriffen, die sich direkt auf dieses Interface beziehen. Innerhalb des Interfaces befinden sich nämlich einige 32 Bit breite Spezial-Register (SP-Register), die vom Prozessor angesprochen werden können. Es sind dies:

- ein Kommando-Register,
- ein Status-Register,
- ein Fehleradreß-Register,
- ein Zähler-Register und
- eine Meßschnittstelle (Register, dessen Ausgang an einen Stecker geführt ist).

Diese Spezialregister haben verschiedene Funktionen. Zum einen dienen sie als Konfigurations- und Zustandsregister, zum anderen als Informationsregister, die im Fehlerfall ausgewertet werden können.

In das Kommandoregister können einige Angaben eingetragen werden, die den Zugriff über dieses Interface regeln. Nach dem Einschalten des Motorola-Systems soll es nämlich noch nicht möglich sein, über diese Schnittstelle auf die umgebenden Kommunikationsspeicher zuzugreifen. Dies dient dazu, die vorhandenen Startup-Routinen, die das Motorola-System in dieser Situationen durchführt, nicht zu stören. Gleichzeitige Zugriffe auf Speicher, die von mehreren Motorola-Systemen erreichbar sind, könnten bei der “Suche nach vorhandenem Speicher” ohne Wissen, daß sich dieses System in einem speichergekoppelten Multiprozessorsystem befindet, zu einer falschen Diagnose führen. Erst nach erfolgreichem Abschluß dieser Aktionen soll das Betriebssystem, das über das Multiprozessorsystem und über die Kommunikationsspeicher Bescheid weiß, die Zugriffe über diese Schnittstelle ermöglichen. Die “Zuschaltung der Kommunikationsspeicher” erfolgt durch einen einfachen Speicherzugriff auf das Kommandoregister, indem ein bestimmtes Bit gesetzt wird. Ebenso kann auch eine “Abschaltung dieser Zugriffsmöglichkeiten” durch Löschen dieses Bits erreicht werden.

Als zweites läßt sich im Kommandoregister einstellen, ob ein beim Lesen erkannter Parityfehler an den Prozessor weitergeleitet werden soll oder nicht. Eine Abschaltung ist üblicherweise nur für Testzwecke vorgesehen. Im Normalfall sollte die Weiterleitung eingeschaltet sein. Die Abschaltung ist daher vorgesehen worden, weil die CMMU, über die diese Daten noch laufen werden, ebenfalls die Möglichkeit hat, den Parityfehler zu erkennen. Ob sie dieses Wissen über den Fehler an den Prozessor weitergibt oder nicht, kann durch Konfigurierung eingestellt werden [MOT2: SCR, System Control Register]. Mit einer vom Prozessorinterface aufgezwungenen “Fehler-Antwort” hätte sie diese Möglichkeit nicht mehr.

Die eingetragenen Einstellungen lassen sich aus dem Statusregister auch abfragen (lesen). Weiterhin enthält das Statusregister auch Angaben über den zuletzt aufgetretenen Fehler, der bei einem Zugriff über dieses Interface registriert worden ist. Dabei werden folgende Informationen über den Zugriff abgespeichert:

- es war eine Lese- oder ein Schreiboperation;
- der Parityfehler wurde im Prozessorinterface erkannt; dies kann sich nur beim Lesen ereignen, wenn das übertragene Wort das Interface mit unpassenden Paritybits erreichte;
  ob der Fehler, der dies verursachte, bei der Übertragung oder während der Lagerung im Speicher auftrat, ist damit noch nicht geklärt;
- der Parityfehler wurde vom angesprochenen Kommunikationsspeicher erkannt; dies kann sowohl die Adresse als auch das zu schreibende Wort betroffen haben;
  beim Lesen wird das ausgegebene Wort dagegen nicht vom Kommunikationsspeicher sondern vom Prozessorinterface überprüft;
- das ausgewählte Ziel: Kommunikationsspeicher über Port A oder über Port B oder über Port C oder ein internes Spezialregister;
- das angesprochenen Spezialregister existiert nicht, oder die erfolgte Zugriffsart ist hier nicht erlaubt (z.B. Lesen von der Meßschnittstelle);
  auch Burst- und XMEM-Zugriffe auf Spezialregister sind nicht erlaubt;
- der Speicherzugriff wurde vom Kommunikationsspeicher nicht (mit einem Signal) beantwortet, d.h. ein Timeout-Fehler trat auf;
- das Statusregister wurde seit der letzten Mitteilung eines Fehlers an den Prozessor bereits gelesen;
- seit dem letzten Lesen des Spezialregisters haben sich im Zusammenhang mit diesem Interface mehrere Fehler ereignet;
  das Statusregister enthält in diesem Fall die Informationen über den letzten Fehler;
- es kam zu einem Timing-Fehler, so daß die LOCK-Information der CMMU vom Prozessorinterface nicht ausgewertet werden konnte;
  dieser Fall kann unter Umständen auftreten, wenn das Motorola-System, dessen Takt das Prozessorinterface übernimmt, mit einem schnelleren Takt als 25 MHz betrieben wird.

Nach dem Auftreten eines Fehlers können diese Eintragungen im Statusregister wichtige Hinweise auf den Fehler geben. Für den Fall, daß der mißlungene Speicherzugriff wiederholt werden soll, wird im Fehleradreßregister die beim Fehler gültige physikalische Adresse eingetragen. Dieser Eintrag bleibt so lange bestehen, bis sich der nächste Fehler mit Beteiligung dieses Interfaces ereignet. Das Fehleradreßregister kann gelesen werden, um die Adresse für einen folgenden (Wiederholungs-) Zugriff verwenden zu können.

Als weiteres Spezialregister existiert das Zählerregister. Es enthält vier 7-bit-Zähler mit je einer Überlaufanzeige (Zählbereich: “0 - 127” oder “mehr als 127”). Dieses Register kann durch einen Schreibzugriff (mit beliebigem Wert) komplett gelöscht werden. In dem Register werden automatisch die (seit dem letzten Löschen) aufgetretenen Parityfehler gezählt. Dabei ist je ein Zähler für die Kommunikationsspeicher vorgesehen, die sich hinter den drei Ports befinden, und einer für die Parityfehler, die im Interface selbst erkannt werden. Gelesen werden die Zähler gemeinsam durch einen einzigen Lesezugriff. Dabei gibt es zwei Möglichkeiten. In einem Fall bleiben die Werte in den Zählern unverändert, und im anderen Fall werden sie automatisch, d.h. atomar mit dem Lesezugriff, gelöscht. Diese beiden Leseversionen unterscheiden sich gegenüber dem Prozessor nur durch Zugriff auf unterschiedliche Adressen innerhalb des Interfaces.

Diese Zähler sind für den Einsatz als “Langzeitüberwachung” bei Zugriffen auf die Kommunikationsspeicher gedacht. Solange sich nicht allzu viele Übertragungsfehler ereignen (kein Zählerüberlauf), kann man mit ihnen einen Eindruck von der Fehleranfälligkeit der Übertragungswege gewinnen. Die Zähler müssen nicht nach jedem fehlerhaften Zugriff gelöscht werden. Sie sind nicht als Datenbasis für die Untersuchung nach einem aufgetretenen Fehler gedacht, obgleich sie, nach jedem untersuchten Fehler gelöscht, dazu dienen können.

Ferner ist zu bemerken, daß diese Zähler unabhängig davon die Fehler registrieren, ob eine Weitermeldung der fehlerhaft gelesenen Worte an die CMMU durchgeführt werden soll oder nicht. Dagegen werden die Informationen über aufgetretene Fehler im Status- und im Fehleradreßregister nur dann eingetragen, wenn auch tatsächlich der Fehler weitergeleitet werden soll. Dies ist so gewählt worden, damit der Prozessor auch immer die Information zum mitgeteilten Fehler im Status- und im Fehleradreßregister vorfinden kann.

Als weiteres Spezialregister ist eine Meßschnittstelle integriert. Auf dieses Register kann nur geschrieben werden. Ein einfacher Schreibzugriff auf dieses Register lädt ein 32-bit-Wort in dieses Register. Die Ausgänge des Registers sind mit einem Stecker verbunden, an den außerhalb des Rechners ein Meßgerät oder eine Anzeigeeinheit angeschlossen werden kann. Auch ein Signal, das (als Trigger) den Zeitpunkt angibt, wann eine neue Information in das Register geschrieben worden ist, kann am Stecker abgegriffen werden. Diese Einrichtung dient in erster Linie zur Unterstützung des Hybrid-Monitoring [HOF93]. Aber auch für Testzwecke eignet sich diese Schnittstelle sehr gut zur Anzeige.

An dieser Stelle soll ein Nachteil erwähnt werden, der die Auswertung der verschiedenen Informationen aus den Spezialregistern nach dem Auftreten eines Fehlers betrifft. An der gewählten Schnittstelle (S-Bus) kann die Identität des Prozessors, der einen Zugriff initiiert hat, nicht fest-

gestellt werden. Kommt es vor, daß zwei Prozessoren kurz hintereinander Zugriffe auf die Kommunikationsspeicher durchführen, die beide mit einem Error beantwortet werden, so ist die Aussagekraft des Statusregisters in Frage gestellt. Denn dessen Inhalt bezieht sich nur auf den letzten Fehler, und der Prozessor, der den Fehler untersucht, kann i.a. nicht feststellen, ob er als letzter vom Fehler betroffen war oder nicht. Als Anhaltspunkt kann zwar dienen, daß im Statusregister zu erkennen ist, ob seit dem letzten Lesen dieses Registers kein Fehler, genau ein Fehler oder mehr als ein Fehler aufgetreten ist; eine Gewißheit darüber, daß er selbst von der aufgezeichneten Fehlersituation betroffen ist, existiert aber nicht. Nur wenn sichergestellt wäre, daß nur einer der bis zu vier Prozessoren des Knotens zur Zeit über das Prozessorinterface zugreift, wäre eine sichere Aussagekraft dieses Registers und des Fehleradreßregisters gegeben. Dieser Nachteil wird immer wieder einmal auftauchen. An diesen Stellen wird dann kurz auf diesen Hinweis verwiesen.

3.4 Der Kommunikationsspeicher

Der Kommunikationsspeicher dient dem Austausch von Daten benachbarter Knoten. Damit von verschiedener Seite aus auf ihn zugegriffen werden kann, ist er mit drei Ports ausgestattet. Diese ähneln sehr den drei Ports des eben beschriebenen Prozessorinterfaces. Über diese Ports greifen die angeschlossenen Prozessoren auf den Kommunikationsspeicher zu (siehe Abb. 12).

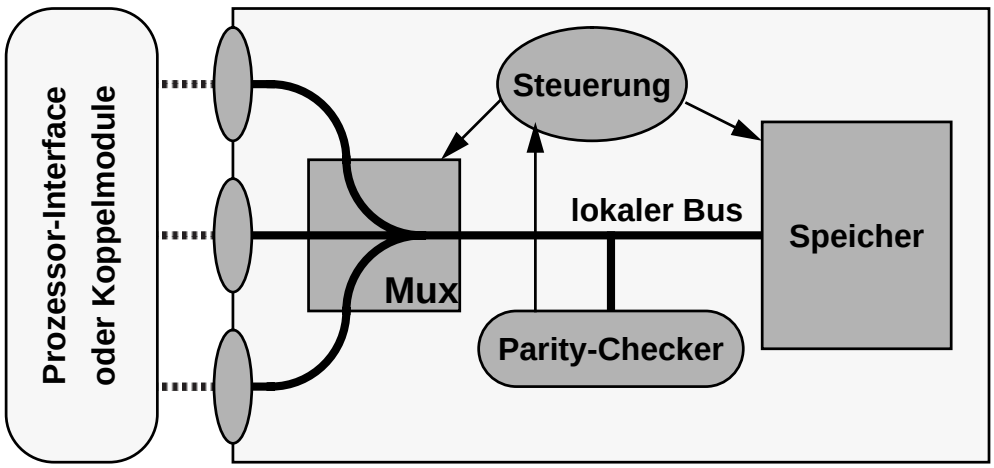


Abb. 12 Kommunikationsspeicher

Bis zum Port erfolgen die eintreffenden Zugriffswünsche verschiedener Prozessoren unabhängig voneinander. Der hinter den Ports liegende Speicher kann die Zugriffe aber immer nur hintereinander bearbeiten. Bei gleichzeitig eintreffenden oder bei sich zeitlich überlappenden Zugriffswünschen verschiedener Prozessoren muß daher eine Hintereinanderausführung der Zugriffe stattfinden. Das wird von einer internen Steuerung geregelt. Auf diese hat der Zugreifer

keinen Einfluß. Da diese Steuerung als Zustandsautomat aufgebaut ist, muß auch eine Synchronisierung der übergebenen Signale erfolgen.

Die vom Prozessor übertragene Adresse wurde schon unabhängig von der Zuteilung für diesen Zugriff im Port eingefangen. Dieses Einfangen ist der Auslöser zur Bewerbung um die Zuteilung des Speicherzugriffs. Hat sich die Steuerung für einen bestimmten Port entschieden, so wird die hier gewünschte Operation im Speicher ausgeführt. Bei einem Schreibzugriff treibt das Prozessorinterface das zu übertragende Wort so lange, bis eine Antwort in Form von Steuersignalen (“Ready” oder “Error”) das Ende des Zugriffs signalisiert. Bei einem Lesezugriff wartet das Prozessorinterface so lange, bis durch die Steuersignale der Empfang des übertragenen Wortes angezeigt wird. Entsprechend wird auch bei Mehrwortzugriffen (Burst) und bei der Durchführung des XMEM-Befehls (Read-Modify-Write-Befehl) durch die Steuersignale angezeigt, wann ein neues Wort übertragen werden soll.

Für den Prozessor, der diesen Zugriff durchführt, sieht der Zugriff auf den Kommunikationsspeicher genauso aus wie ein Zugriff auf seinen Privatspeicher. Von der Zeitverzögerung, die durch den Zugriff auf einen Kommunikationsspeicher entsteht (Synchronisationsstufen, Warten am Kommunikationsspeicher), merkt der Prozessor anhand der empfangenen Steuersignale nichts.

Detailliertere Informationen über den Kommunikationsspeicher und dessen Konfliktauflösungsstrategie bei gleichzeitig vorliegenden Zugriffen sind in [HIL92] zu finden.

3.5 Das Koppelmodul

Für die bei MEMSY erforderlichen Verbindungen zur Herstellung der gewünschten Topologie sind die Ports am Prozessorinterface und am Kommunikationsspeicher zahlenmäßig nicht ausreichend. Daher wurde ein Koppelmodul entwickelt, mit dem die benötigte Konnektivität hergestellt werden kann.

Das Koppelmodul hat nach außen vier Prozessor-Ports (P-Port), an die je ein Prozessorinterface angeschlossen werden kann, und vier Speicher-Ports (M-Port), die mit je einem Kommunikationsspeicher verbunden werden können. Es bilden immer ein P-Port und ein M-Port ein Paar.

Ein solches Port-Paar ist mit Hilfe von speziellen Treiberbausteinen (“Bus-Schalter”) realisiert, die vier Busse miteinander verbinden können [AMD29]. Diese Bausteine haben unter anderem die Fähigkeit, gleichzeitig je zwei der Busse zum Datentransport miteinander zu verbinden (z.B. Bus A — Bus C und gleichzeitig Bus D — Bus B).

Untereinander sind die vier Busschalter zu einem Ring verbunden. Zwei von deren vier Bussen sind mit einem P- und einem M-Port des Koppelmoduls verknüpft, und die zwei anderen bilden

zusammen mit den anderen entsprechenden Bussen vier interne Verbindungswege (siehe Abb. 13).

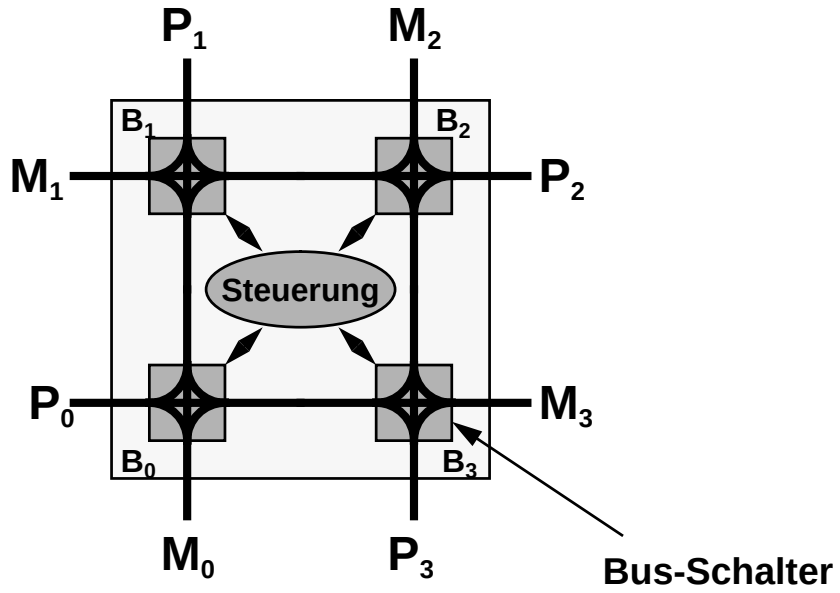


Abb. 13 Koppelmodul

Solange keine Konfliktsituationen bezüglich der Nutzung von Wegen bestehen, können mehrere (bis zu vier) Verbindungen gleichzeitig innerhalb des Koppelmoduls geschaltet und benutzt sein. Für das Auftreten von Konflikten kommen zwei Gründe in Frage:

- mindestens zwei Prozessoren wollen über das Koppelmodul auf denselben Kommunikationsspeicher (M-Port) zugreifen;
- zwei Zugriffe sollen gleichzeitig über dieselbe interne Teilstrecke laufen.

In beiden Fällen kann nur einer der Zugreifer sofort zum gewünschten Kommunikationsspeicher durchgeschaltet werden. Der andere Zugreifer muß warten, bis der Weg wieder frei geworden ist.

Kontrolliert werden die einzelnen Zugriffe im Koppelmodul von einer zentralen Steuerung, die ständig einen genauen Überblick über die gewünschten Zugriffe und die verfügbaren Wege hat. Unter Umständen schaltet sie für einen Transfer auch einen “Umweg”, der zwar über mehrere interne Wege geht, aber dafür gleichzeitig zu einem anderen Transfer ablaufen kann, mit dem dieser bei Wahl des kürzeren Weges in Konflikt stehen würde. So sind beispielsweise die Verbindung P_0 — M_3 über B_0 — B_3 und P_3 — M_0 über B_3 — B_2 — B_1 — B_0 möglich. In dieser Situation können sogar noch gleichzeitig die Verbindungen P_1 — M_1 über B_1 und P_2 — M_2 über B_2 benutzt werden. Das sind insgesamt vier gleichzeitig nutzbare Verbindungen.

Mit diesem Koppelmodul besteht zum größten Teil auch die Möglichkeit, einen intern ausgefallenen Weg durch “Umleitung” zu umgehen. Damit kann ein Teilausfall des Koppelmoduls oft noch dadurch toleriert werden, indem ein Alternativweg benutzt wird. Dazu müssen andere Adreßbereiche bei Zugriffen auf die Kommunikationsspeicher benutzt werden. Dadurch ist es in den meisten Fällen noch möglich, von jedem Prozessor aus noch alle angeschlossenen Kommunikationsspeicher zu erreichen.

Gegenüber dem Prozessorinterface und dem Kommunikationsspeicher stellen sich die übertragenen Signale so dar, wie es diese beiden Einheiten voneinander erwarten. Daher ist es für einen Speicherzugriff unerheblich, ob er über ein Koppelmodul geführt wird oder nicht. Nur im Zeitverhalten ist eine weitere Verzögerung dieses Zugriffs zu bemerken, denn auch hier muß eine Synchronisierung der übertragenen Signale stattfinden.

Weitere Informationen über das Koppelmodul sind in [HIL91] und [HIL92] zu finden.

3.6 Realisierung der MEMSY-Topologie mit Hilfe der Koppelmodule

Die MEMSY-Topologie, bestehend aus Knoten (Rechner) und Kanten (Verbindungswege), ist innerhalb einer Ebene eine Gitterstruktur, die an jedem Rand zur gegenüberliegenden Seite geschlossen ist. (Torus). Betrachtet man die Kanten zwischen den Knoten als Seitenränder von Quadraten, und färbt man diese Quadrate abwechselnd mit heller und dunkler Farbe ein, so entsteht ein Schachbrettmuster. Bei Feldern, die eine gerade Anzahl an Knoten an jeder Seite besitzen (z.B. 4 x 4 - Felder), läßt sich dieses Schachbrettmuster über den Rand hinaus zur gegenüberliegenden Seite verbinden.

Betrachtet man nun nur die dunklen Felder und interpretiert diese als Koppelmodule, wobei die vier Kanten die internen Verbindungswege darstellen, so läßt sich bereits die benötigte Anordnung der Koppelmodule innerhalb einer Ebene von MEMSY erkennen. An allen Eckpunkten der Felder befindet sich jeweils ein Prozessorknoten und ein Kommunikationsspeicher. Diese sind über insgesamt zwei Ports mit den beiden an dieser Stelle zusammenstoßenden Koppelmodulen verbunden sind (siehe Abb. 14). Auf diese Weise lassen sich A- und B-Ebene bilden [GRY90].

Für die Verbindung zwischen den Ebenen kann ebenfalls je ein Koppelmodul pro Elementarpyramide eingesetzt werden, an das die vier Kommunikationsspeicher der A-Ebene mit jeweils ihrem dritten Port an die M-Ports und der dritte Port des B-Prozessors an einen P-Port des Koppelmoduls angeschlossen wird. Damit ist die erforderliche Verbindungsstruktur für MEMSY geschaffen: Alle Prozessoren einer Ebene können (über zwei Koppelmodule) auf den eigenen Kommunikationsspeicher und den der vier Nachbarn in den Himmelsrichtungen zugreifen; und jeder B-Prozessor kann zusätzlich die vier Kommunikationsspeicher seiner untergeordneten A-Prozessoren erreichen.

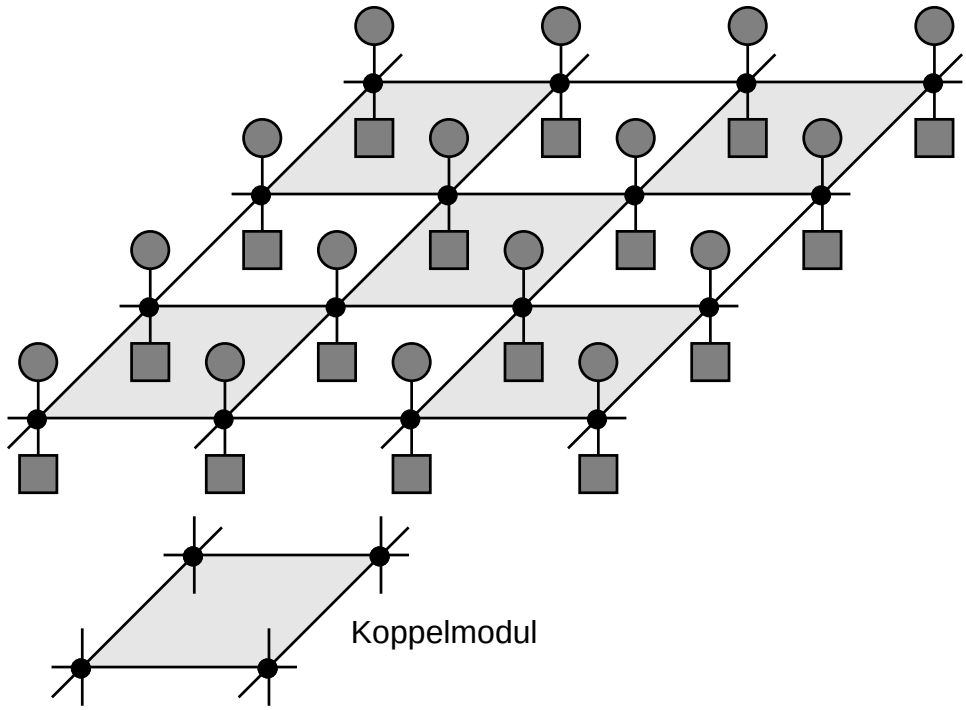


Abb. 14 Realisierung der MEMSY-Topologie mit Koppelmodulen (dargestellt ist nur eine Ebene)

Im nächsten Kapitel wird auf die Platzierung des Stabilen Speicher innerhalb von MEMSY eingegangen.

4 Integration des Stabilen Speichers in MEMSY

Für die Platzierung des Stabilen Speichers innerhalb des Multiprozessorsystems MEMSY muß ein geeigneter Ort gefunden werden. Dabei sind folgende Punkte zu berücksichtigen:

- Es müssen mehrere Wege zum Stabilen Speicher vorhanden sein, um auch bei Ausfall einer Verbindung weiterhin Zugriff auf diesen zu haben.
- Auf der gesamten Übertragungsstrecke muß eine Erkennung von Übertragungsfehlern gewährleistet sein.
- Am Stabilen Speicher müssen Schutzmöglichkeiten eingerichtet werden können, die es defekten Prozessoren nicht erlauben, Veränderungen in den im Stabilen Speicher abgelegten Daten vorzunehmen.
- Mehrere Prozessoren sollen einen Stabilen Speicher gemeinsam nutzen können, um diesen zusätzlichen Aufwand beschränken zu können.
- Der Stabile Speicher sollte möglichst große Unabhängigkeit von fehlerverursachenden Einheiten haben.

Das Multiprozessorsystem MEMSY besitzt eine umfangreiche Verbindungsstruktur. In ihr können jeweils mehrere Prozessorknoten auf gemeinsame Speicherbereiche (Kommunikationsspeicher) zugreifen. Weiterhin ist das Verbindungsnetzwerk intern so ausgelegt, daß es von jedem Prozessorknoten aus je zwei Wege gibt, über die der Zugriff zu seinen erreichbaren Kommunikationsspeichern ablaufen kann. Dies ergibt sich aus der Implementierung des Koppelmoduls, mit dem die Verbindungsstruktur implementiert worden ist (siehe auch Abschnitt 3.6, MEMSY-Topologie). Da diese Verbindungsstruktur bereits in hohem Maße die Voraussetzungen für die Umgebung des Stabilen Speichers bietet, wurde für diesen der Platz an der Seite eines Kommunikationsspeichers ausgewählt. Somit ist kein zusätzliches Verbindungssystem für den Zugriff auf den Stabilen Speicher erforderlich.

Das Koppelmodul verfügt über die Eigenschaft, von jedem angeschlossenen Prozessorknoten aus einen Weg zu allen angeschlossenen Kommunikationsspeichern bereitstellen zu können (siehe Abb. 15). Somit können die vier Kommunikationsspeicher aus der Grundfläche einer Elementarpyramide als gemeinsamer Speicher der zugehörigen Prozessorknoten genutzt werden. Dies geht über die ursprünglich geforderte Nachbarschaftsverbindung von MEMSY hinaus, denn nun gibt es über das Koppelmodul innerhalb der Ebene auch eine Zugriffsmöglichkeit entlang einer Diagonalen.

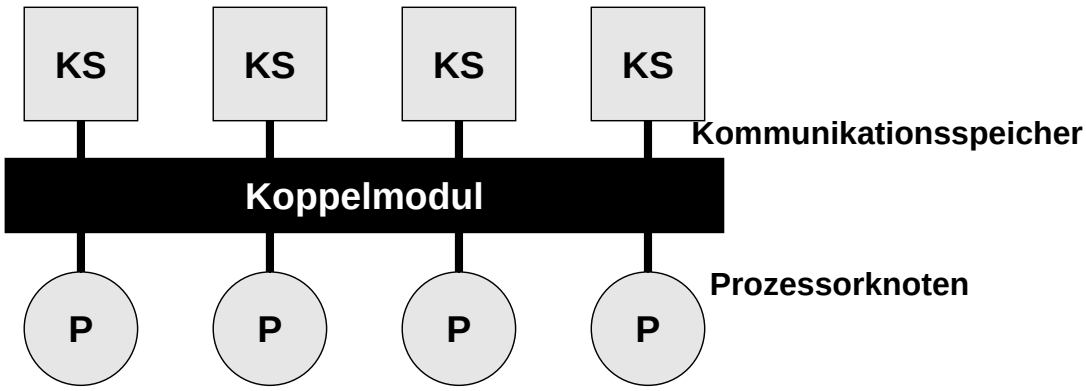


Abb. 15 vollständige Verbindungen über ein Koppelmodul

Zwischen den vier A-Knoten und dem zugehörigen B-Knoten ist ebenfalls ein Koppelmodul eingesetzt. Über dieses greift der B-Prozessor auf die vier Kommunikationsspeicher seiner untergebenen A-Prozessoren zu. Damit hat auch der B-Prozessorknoten, wie gefordert, einen direkten Zugriff auf diese vier Kommunikationsspeicher (siehe Abb. 16).

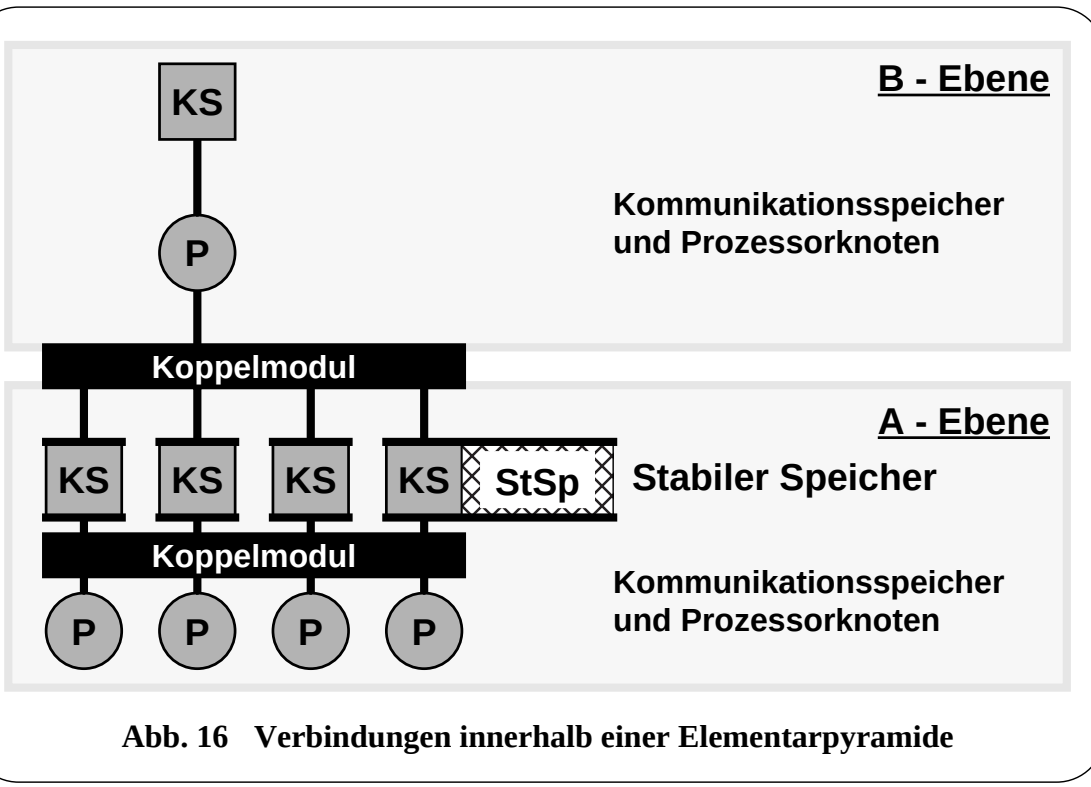


Abb. 16 Verbindungen innerhalb einer Elementarpyramide

Auf Grund der geschilderten Umstände ist es möglich, einen beliebigen Kommunikationsspeicher dieser vier auszuwählen und ihn mit einem zusätzlichen Speicher, dem Stabilen Speicher,

auszustatten. Somit reicht, zumindest bezüglich der Zugriffsmöglichkeiten, ein einziger Stabiler Speicher in einer Elementarpyramide für diese fünf Knoten aus. Dadurch kann auch der zusätzlich erforderliche Implementierungsaufwand eingeschränkt werden. Ferner bildet der Stabile Speicher auf diese Weise einen eigenständigen Fehlereingrenzungsbereich, der sich außerhalb der Prozessorknoten befindet.

Auf die Kommunikationsspeicher erfolgen die Zugriffe der Prozessoren genauso wie übliche Hauptspeicherzugriffe. Dazu mußten bereits bei jedem Prozessor entsprechend der von ihm aus erreichbaren Kommunikationsspeicher zusätzliche Adreßbereiche eingerichtet werden. Bei den Zugriffen auf die Kommunikationsspeicher entscheidet das Koppelmodul anhand der übertragenen Adresse, welcher Weg für diesen Transfer geschaltet werden muß. Für jeden möglichen Kommunikationsspeicher wurde ein Adreßraum von 16 MByte reserviert. Dieser Bereich wird aber in der aktuell implementierten Version nicht voll genutzt, da jeder Kommunikationsspeicher nur über 4 MByte verfügt. Die restlichen 12 MByte blieben bislang ungenutzt. Und genau in diesen Bereich wird der Stabile Speicher integriert. Hier belegt er einen Bereich von 4 MByte innerhalb dessen die beteiligten Prozessoren den Stabilen Speicher ansprechen können.

Es ist klar, daß 4 MByte nicht ausreichend sind, um den erforderlichen Speicherbedarf für die abzulegenden Sicherungsdaten oder ähnlich wichtiger Daten der umgebenen Prozessoren aufnehmen zu können. Daher und zum Schutz der Daten vor unbefugtem Zugriff (z.B. durch fehlerhaft arbeitende Prozessoren) wird in dem angesprochenen Adreßbereich eine Speicherschnittstelle eingebettet, über die die Prozessoren nur einen mittelbaren Zugriff auf die Daten haben. Die Zugriffe der Prozessoren erfolgen dann auf einige wenige Auftrags-, Parameter- und Transfer-Register. Eine in dieser Schnittstelle integrierte Steuerung führt dann die gewünschten Zugriffe für den Datentransport aus.

5 Fehlermöglichkeiten in MEMSY

In jeder Hardware und auch in jeder Software gibt es die Möglichkeit, daß ein oder mehrere Fehler auftreten. Auch die sorgfältigste Arbeit kann dies in umfangreichen Implementierungen, sei es Hard- oder Software, nicht verhindern. Daher ist es unerläßlich festzustellen, welche Fehler am wahrscheinlichsten auftreten werden. Dann können Maßnahmen zu ihrer Abwehr oder Tolerierung gefunden werden.

5.1 Fehler in der Software

Software-Fehler sollen hier zum größten Teil ausgeklammert werden, da sie aus Sicht der Hardware, ohne Wissen über das laufende Programm, kaum erkannt werden können. Bemerkbar machen sich solche Fehler am ehesten dem Anwender, der sich wundert, warum sein Programm nicht das tut, was er von ihm erwartet. In diesen Fällen bleibt ihm meistens nichts anderes übrig, als sein Programm zu analysieren. Es gibt eine ganze Reihe von Hilfsmitteln, die einem das Erstellen von Software erleichtern und so von vornherein die Gefahr des Auftretens von nicht gewünschten Programmabläufen verringern. Aus Sicht der Hardware ist dies aber nicht als Fehler erkennbar.

Dennoch gibt es einige Möglichkeiten, die es der Hardware gestatten, Fehler in der Software wenigstens zum Teil zu erkennen. So gibt es in den meisten Rechnern eine Memory Management Unit (MMU), eine Hardware, welche u.a. die Zugriffe des Prozessors in seine Außenwelt kontrolliert. Dabei wird dafür gesorgt, daß der Prozessor nur in einen Teil seines ihm potentiell zur Verfügung stehenden Adreßraums zugreifen kann. Dies wird zum Schutz der verschiedenen Tasks oder Jobs untereinander eingesetzt, die gleichzeitig im Adreßraum eines Prozessors existieren [MOT2]. Die MMU ist konfigurierbar und wird von der Betriebssoftware programmiert. Somit ist der damit erzielbare Schutz ganz von der korrekten Programmierung der MMU abhängig. Alle Zugriffe, die die MMU passieren, werden von der dahinterliegenden Hardware i.a. nicht mehr auf ihre Berechtigung hin kontrolliert. Beim Einsatz des Stabilen Speichers für MEMSY wird sich zeigen, daß hier auch nach der MMU noch spezielle Hardware für Kontrollvorgänge eingesetzt wird, da bei Zugriffen auf den Stabilen Speicher in viel feinerem Maß die Erlaubnis zum Zugriff überprüft werden muß, als dies von der MMU durchgeführt werden kann.

5.2 Fehler in der Hardware

Bei Hardware-Fehlern wird allgemein zwischen transienten und permanenten Fehlern unterschieden. Permanent ist ein Fehler dann, wenn eine Teileinheit dauerhaft gestört oder defekt ist. Die Möglichkeiten zur Behebung im laufenden Betrieb sind meistens sehr gering. Der Fehler müßte maskiert werden können, wie dies beispielsweise bei TMR-Schaltungen (Triple Modular

Redundancy) möglich ist. Ansonsten bleibt nur der Ersatz oder die Reparatur übrig. In den meisten Fällen wird der laufende Betrieb dabei unterbrochen werden müssen.

Transiente Fehler entstehen aus kurzzeitigen Störungen. Sie treten deutlich öfter auf als permanente Fehler [SIE82] [DCP94]. Transiente Fehler können ein vorübergehendes (zeitlich sehr begrenztes) Fehlverhalten auslösen (z.B. Transport falscher oder fehlerhafter Daten) oder auch zu einem Erscheinungsbild führen, das wie ein vorläufig dauerhafter Fehler aussieht (z.B. verändertes Bit im Speicher).

Die Auswirkungen eines Fehler müssen sich nicht sofort nach dem Auftreten des Fehlers zeigen. Im allgemeinen bleiben sie unerkannt, bis die fehlerbehafte Einheit verwendet werden soll. Aber auch beim nächsten Zugriff auf eine Einheit mit einem permanenten Fehler ist nicht sicher, daß der Fehler entdeckt wird. Beispielsweise kann ein permanenter Fehler in einer Speicherbaugruppe existieren, der dazu führt, daß nicht die richtige Adresse an die Speicherchips angelegt wird. Dies kann durch eine schadhafte Verbindung zwischen dem Adreßregister und den Speicherchips oder durch einen Fehler im Adreßdekoder hervorgerufen werden [SPH94]. Bei Schreibzugriffen kann es dann vorkommen, daß Daten, die an verschiedene Adressen in diesem Speicher geschrieben werden sollen, am selben Speicherplatz eingetragen werden. Beim späteren Lesen wird dieser Fehler unter Umständen gar nicht bemerkt, wenn der zuerst geschriebene Wert für den weiteren Programmablauf irrelevant ist. Ähnliche Auswirkungen können z.B. auch dann auftreten, wenn die vom Prozessor gelieferte Adresse zu einem falschen Zeitpunkt in das Register übernommen wird. Dann erfolgt der Speicherzugriff auch an eine falsche Adresse.

Für die Fehlererkennung ist es wichtig, daß Redundanz vorhanden ist. Am weitesten verbreitet ist in einem Computer das Erzeugen und Checken von Parity-Information. Dazu wird zu einer Gruppe von Bits, meistens eines Byte, ein zusätzliches Bit erzeugt, so daß die Gesamtzahl der dann vorliegenden gesetzten Bits eine gerade Zahl bzw. eine ungerade Zahl (bei Even Parity bzw. bei Odd Parity) ist. Wird der auf diese Weise gebildete Wert zwischen zwei Teileinheiten übertragen, so kann der Empfänger überprüfen, ob noch die benutzte Parity vorliegt. Durch diese Methode lassen sich Übertragungsfehler erkennen, bei denen als Fehlerursache das Umkippen eines einzelnen Bits innerhalb eines Bytes angenommen wird. Auf gleiche Weise lassen sich auch Fehler erkennen, bei denen während der Lagerung in einem Speicher einzelne Bits umgekippt sind [WAK78].

Ob die Auswirkung eines Fehlers sofort erkannt und gegebenenfalls korrigiert werden kann oder nicht, hängt entscheidend davon ab, wann die möglicherweise gestörte Information überprüft wird. Eine während einer Übertragung aufgetretene Störung kann sofort erkannt werden, sofern Einrichtungen dafür zur Verfügung stehen. Darauf kann der Prozessor oder auch der angesprochenen Speicher sofort reagieren. Dagegen wird eine transiente Störung im Speicher im allgemeinen erst viel später entdeckt, nämlich beim nächsten Lesen des betroffenen Wortes. Ist außer dem Parity keine weitere Redundanz vorhanden, läßt sich der Fehler dann nicht mehr korrigieren.

Für die Bewertung der Auswirkung von Fehlern ist es auch wichtig zu wissen, ob sich der Fehler

auf die weitere Bearbeitung der Aufgabe auswirken wird oder nicht. Ist beispielsweise eine Variable durch einen Fehler geändert worden, die in der weiteren Berechnung nicht mehr benötigt wird, so hat dieser Fehler keinen Einfluß. Anders ist es, wenn die Variable noch genutzt wird. Dann kann die fehlerhafte Veränderung einen falschen Ablauf der weiteren Berechnungen hervorrufen. Wie stark die Auswirkungen sind, läßt sich ohne Wissen über das Programm aber nicht vorhersagen.

5.3 Abwehrmaßnahmen gegen Fehler in der Motorola-Hardware

Das Multiprozessorsystem MEMSY ist aus verschiedensten Teilen aufgebaut. Den Kern des Systems bildet ein handelsübliches System von Motorola, das selbst aus mehreren Komponenten besteht. Erweitert wurde das System durch eigene Entwicklungen, die im wesentlichen aus den Kommunikationsspeichern mit der Verbindungshardware und aus der Interruptkopplung bestehen.

Das Motorola-System ist standardmäßig mit einigen Fehlererkennungsmechanismen ausgestattet. Die meisten wirken gegen Fehler, die in der Software auftreten können. Dagegen ist gegen Hardware-Fehler nur ein geringer Schutz vorhanden. Fehler dieser Art werden offenbar nicht so häufig erwartet.

5.3.1 Maßnahmen gegen Software-Fehler

Das Betriebssystem eines Rechners ist ein sehr umfangreiches Programmpaket. Meistens wird es von vielen Personen in Teamarbeit erstellt und immer wieder verändert. Daher ist es nicht verwunderlich, daß sich hier Fehler einschleichen können. Das Betriebssystem ist aber gleichzeitig das wichtigste Programm, das der Rechner für seine Arbeit benötigt. Somit sind die Rechner gegenüber Fehlern, die einer unkorrekten Arbeitsweise des Betriebssystems entspringen können, sehr anfällig. Da das Betriebssystem im Gegensatz zu den Anwenderprozessen meistens alle Zugriffsrechte auf alle Ressourcen besitzt, lassen sich Fehler, die bei ihm auftreten, i.a. nicht von einer Hardware kontrollieren.

In den meisten Rechnern laufen auch mehrere Programme gleichzeitig im Time-Sharing Betrieb oder bei Multiprozessorsystemen auch im Space-Sharing Betrieb. Die Programme dürfen sich in ihrer Arbeitsweise gegenseitig nicht beeinträchtigen. So gehören Zugriffe in unerlaubte (“fremde”) Adreßbereiche zu den problematischsten Fehlern, da es kaum möglich ist, im nachhinein die Fehlerursache zu diagnostizieren. Zur Vermeidung solch katastrophaler Fehler verfügen die meisten Rechner über einige Schutzvorkehrungen.

Die wichtigste Rolle spielt dabei die Memory Management Unit (MMU). In diesem Baustein werden Adreßumrechnungen vorgenommen, die eine logische Adresse, die der Prozessor ausgibt, in eine physikalische Adresse umwandeln. Bei den Umrechnungen werden gleichzeitig verschiedene Schutzvorkehrungen überprüft, welche die Zugriffe auf diesen Speicherbereich

betreffen. Im Fall einer Schutzverletzung werden entsprechende Fehlersignale gesetzt. Es sind dies:

- Segment Fault,
- Page Fault,
- Write-Protection Violation,
- Supervisor Violation (oder auch “Privilege Violation” genannt).

Das Auftreten eines solchen Flags ruft eine sofortige Reaktion der MMU hervor, und die vom Prozessor gewünschte Aktion auf den Speicher wird nicht ausgeführt. Außerdem wird diese Situation dem Prozessor sofort angezeigt, so daß dieser seine vorgesehene Arbeitsweise unterbrechen und eine Maßnahme zur Fehlerdiagnose und -behebung einleiten kann.

Die Wahl, welche Abschnitte aus dem Speicher auf welche Weise geschützt werden, wird zuvor vom Betriebssystem durch Programmieren der MMU eingerichtet. Die Hardware (MMU) ist dann in der Lage, im laufenden Betrieb die Rechte zu überprüfen. Ein sorgfältiges Programmieren der MMU ist daher sehr wichtig für die Abwehr von Fehlern.

Die MMU, die im Motorola-System eingesetzt wird, hat neben den Umrechnungs- und Schutzaufgaben zusätzlich einen Cache Controller sowie einen Cache zusammen auf einem einzigen Chip integriert. Daher wird dieser Baustein als CMMU (Cache / Memory Management Unit) bezeichnet. In den an der Universität Erlangen-Nürnberg vorhandenen Multiprozessorsystemen vom Typ MEMSY gibt es verschiedene CMMU-Bausteine. Die älteren haben je einen Cache der Größe von 16 KByte integriert (Baustein MC88200), während die neueren mit einem 64 KByte großen Cache (Baustein MC88204) ausgestattet sind. [MOT2] [MOT3]

Neben den Fehlern, die sich auf Speicherzugriffe beziehen, werden vom Prozessor auch einige Ausnahmen bei der Datenverarbeitung erkannt (z.B. divide by 0) und als Exceptions signalisiert [MOT1]. Für die Betrachtungen von Speicherzugriffen spielen sie jedoch keine Rolle. Daher werden sie hier nicht weiter betrachtet.

5.3.2 Maßnahmen gegen Hardware-Fehler

Neben den oben dargestellten Schutzvorkehrungen vor Software-Fehlern erkennt das Motorola-System zum Teil auch Hardware-Fehler, die sich auf Übertragungsstrecken und in den verschiedenen Speichern des Systems ereignen. Einige interne Busse sind mit Redundanz ausgestattet, indem die 32 Bit breiten Verbindungen mit vier Paritybits versehen sind, für jedes Byte ein Paritybit. Dieser Schutz ist jedoch nicht durchgängig. Auf dem P-Bus, der Verbindung zwischen dem Prozessor und den CMMUs, gibt es keinen Parityschutz. Der Bus auf der anderen Seite der CMMUs, der M-Bus, ist dagegen mit Paritybits ausgestattet. Hier handelt es sich um einen zwischen Adressen und Daten gemultiplexten Bus. Somit wird auch beides mit Paritybits versehen. Sie können auf der Empfängerseite gecheckt werden. Ob dies aber durchgeführt wird, ist frag-

lich. Empfangene Daten, sie stammen aus Lesevorgängen, werden in der CMMU überprüft.

An dem M-Bus liegen allerdings noch nicht die lokalen Speicher des Systems. Diese liegen am S-Bus, der vom M-Bus noch durch eine Schnittstelle getrennt ist (vergleiche Abb. 6 in Abschnitt 3.1, Schnittstellen). Der S-Bus verfügt im Gegensatz zum M-Bus über einen separaten Adreßbus und einen Datenbus. Von diesen ist nur der Datenbus mit vier Paritybits abgesichert. Bei Schreibzugriffen auf den lokalen Speicher werden die übertragenen Paritybits ohne Überprüfung im Speicher abgelegt. Erst bei einem späteren Lesevorgang werden sie von der CMMU auf korrekte Parity kontrolliert.

Die Fehler, die von dem Parityschutz erkannt werden können, sind solche, bei denen sich maximal ein Bit eines Bytes seit der Erzeugung der Paritybits verändert hat. Die Stelle, an der dieser Fehler aufgetreten ist, läßt sich aus der Überprüfung der Paritybits aber nicht eindeutig ermitteln. Sowohl auf dem Übertragungsweg (von Platine zu Platine: über einige Treiberstufen und kurze Wege) als auch innerhalb des Speichers kann die Veränderung vorgekommen sein. Im Motorola-System wird die Parity-Information beim Zugriff auf den Hauptspeicher nur beim Lesen und dabei auch nur bei den Daten überprüft, nicht aber bei der übertragenen Adresse. Dies ist auch bei vielen anderen Computern so üblich. Zur sicheren Fehlererkennung darf dann beim Schreiben, Lagern und Lesen insgesamt höchstens ein Bit pro Byte in den Daten verändert sein. Die CMMU gibt die Information, daß ein Fehler erkannt worden ist, als “Transaction Fault” an den Prozessor weiter. Sie selbst markiert in ihrem System Status Register (SSR) diesen Fehler als “Bus Error” oder als “Copyback Error” [MOT2]. Diese Information kann anschließend von einer Fehlerbehandlungs-Routine zur Diagnose verwendet werden.

Andere Fehler, z.B. Mehrbit-Fehler in einem Byte, können nicht mit Sicherheit sofort erkannt werden. Sie machen sich möglicherweise erst viel später bemerkbar. Dann sind aber kaum noch Möglichkeiten für eine sichere Fehlerdiagnose gegeben.

Auch über den VME-Bus können Fehlerinformationen weitergeleitet werden, die in entfernt liegenden Einheiten entdeckt worden sind. Dabei gibt es Signale für eine Zeitüberschreitung (Timeout) bei Zugriffen über den VME-Bus, für Parityfehler, die innerhalb einer Speicherplatine entdeckt worden ist, für Störungen in der Stromversorgung (ACFAIL) und für sonstige Systemfehler (SYSFAIL) [MVM89]. Bei der Implementierung des Prozessorinterface spielten diese Signale aber keine Rolle.

5.4 Abwehrmaßnahmen gegen Fehler in der Erweiterungs-Hardware

In der Erweiterungs-Hardware wurden mehrere Einrichtungen implementiert, mit deren Hilfe Übertragungsfehler erkannt und an den Prozessor zurückgemeldet werden können. Dabei wird auch sichergestellt, daß ein begonnener Zugriff auf jeden Fall beendet wird, damit der Prozessor hier nicht in einen Deadlock gerät.

5.4.1 Parity-Fehler

Die Möglichkeit zum Überprüfen von Übertragungsfehlern mit Hilfe von Paritybits wurde auch für den Transfer zwischen Prozessoren und Kommunikationsspeichern übernommen und ausgeweitet. Die Verbindungen bestehen aus 32-bit-Bussen, die mit vier Paritybits ausgestattet sind. Bei einem Zugriff werden Adressen und Daten im Multiplex-Betrieb übertragen.

Alle Informationen, die das Prozessorinterface zum Übertragungsweg hin verlassen (das sind die Adressen und die zu schreibenden Worte), werden von diesem mit gültigen Paritybits versehen. Am Kommunikationsspeicher wird die Parity überprüft. Daten, die aus dem Kommunikationsspeicher gelesen werden, werden mitsamt den dort vorliegenden Paritybits an das Prozessorinterface übertragen und dort überprüft. Ein an dieser Stelle erkannter Parityfehler ist dann entweder während der Lagerung im Speicher oder im Verlauf des Lesens auf dem Übertragungsweg aufgetreten. Diese Überprüfungen werden bei allen Zugriffsarten ausgeführt, die der Kommunikationsspeicher bearbeiten kann (Ein- und Mehr-Wort-Transfers, RMW-Operationen). Insbesondere die Überprüfung der am Kommunikationsspeicher ankommenden Adressen und Daten übertrifft deutlich die entsprechenden Vorkehrungen innerhalb des Motorola-Systems bei Zugriffen des Prozessors auf seinen Hauptspeicher.

Das Koppelmodul, das zwischen Prozessorinterface und Kommunikationsspeicher liegt, gibt die 36-bit-Worte nur weiter und kontrolliert keine Parity.

Im Fall eines erkannten Adreßfehlers wird im Kommunikationsspeicher keine Aktion ausgeführt. In einer älteren Version des Kommunikationsspeichers konnte der Speicherzugriff nicht mehr gestoppt werden; allerdings wurde dann bei einem gewünschten Schreibzugriff stattdessen ein Lesezugriff ausgeführt. Das verhinderte ein Schreiben auf eine unbekannte Adresse.

Im Fall eines Datenfehlers kann das Schreiben nicht mehr abgebrochen werden. Dieser bezüglich der Fehlerausbreitung unerwünschte Effekt ist die Folge von Optimierungen, die den Zugriff auf den Kommunikationsspeicher möglichst schnell ablaufen lassen sollen. Dies geschah unter den Überlegungen, daß Übertragungsfehler gegenüber erfolgreichen Zugriffen sehr selten auftreten. So ist bei Schreiboperationen bereits begonnen worden, den Speicherbausteinen Steuersignale zuzuführen, noch bevor der Datencheck vollzogen war. Ein Abbruch ab diesem Zeitpunkt würde die vorgeschriebenen Signal-Zeitverhältnisse beim Speicher verletzen, so daß sein Verhalten nicht vorhersagbar wäre. Da die Transferadresse jedoch korrekt empfangen wurde, ist der Ort der fehlerhaften Operation bekannt. Daher bestehen noch Chancen, diese Operation durch Wiederholung des Schreibzugriffs zu korrigieren.

Entdeckt der Kommunikationsspeicher einen Übertragungsfehler, so gibt er ein Error-Signal an das Prozessorinterface zurück. Dieses wird zusammen mit anderen Informationen in einem Statusregister des Prozessorinterfaces eingetragen. Entdeckt dagegen das Prozessorinterface den Parityfehler, so wird ein anderes Statusbit gesetzt. In beiden Fällen wird die aktuelle physikalische Adresse im Fehleradreßregister des Prozessorinterfaces eingetragen.

Der Prozessor bzw. seine CMMU wird sofort durch entsprechende Antwortsignale über den fehlgeschlagenen Zugriff informiert. Dabei wird in der CMMU unter anderem auch die aktuelle Transferadresse vermerkt [MOT2: SAR, System Address Register]. Jetzt liegt es am Betriebssystem, wie schnell es auf den Fehler reagiert. Zur Fehlerdiagnose kann neben den Eintragungen in der CMMU auch das Fehleradreßregister und das Statusregister aus dem Prozessorinterface benutzt werden. Die Fehlerdiagnose sollte rasch erfolgen, da die Informationen über den Fehler nur so lange in den Registern der Prozessorinterface aufgehoben werden, bis sich ein neuer Fehler zwischen Prozessor und Kommunikationsspeicher ereignet.

An dieser Stelle tritt wieder das Problem auf, das mit den mehrfach vorhandenen Prozessoren innerhalb eines Prozessorknotens verknüpft ist. Ein anderer Prozessor aus diesem Knoten kann zur ähnlichen Zeit einen Übertragungsfehler auslösen. Dabei wird der Inhalt des Fehleradreß- und des Statusregisters verändert, und der überprüfende Prozessor findet in diesen Registern möglicherweise nicht mehr die bei ihm aufgetretene Fehlersituation vor (siehe Abschnitt 3.3, Prozessorinterface).

5.4.2 Timeout-Fehler

Als weitere Möglichkeit für die Auslösung eines Fehlers tritt die Überschreitung einer vorgegebenen Zeit bei der Durchführung eines Zugriffs auf (Timeout). An drei Stellen laufen Timer, die bei Zeitüberschreitung eine vorzeitige Beendigung eines laufenden Transfers veranlassen können:

- beim Kommunikationsspeicher (Timeout etwa 11 µsec),
- beim Koppelmodul (Timeout etwa 15 µsec),
- im Prozessorinterface (Timeout einstellbar zwischen 163 µsec und 1310 µsec).

Der Kommunikationsspeicher und das Koppelmodul geben in dieser Fehlersituation eine Error-Antwort an das Prozessorinterface zurück. Dies läßt sich im Prozessorinterface nicht von der Situation unterscheiden, in der vom Kommunikationsspeicher ein Übertragungsfehler (Parity-Error) zurückgemeldet wird, da hierfür kein eigenes Signal auf der Übertragungsstrecke vorgesehen ist. Erreicht stattdessen das Prozessorinterface seinen Timeout, so wird diese Information im Statusregister als Timeout-Fehler vermerkt. An den Prozessor bzw. an die CMMU wird in beiden Fällen eine Error-Antwort zurückgegeben.

Beim Kommunikationsspeicher kann der Grund für das Erreichen der Timeout-Zeit in der Nichteinhaltung des Übertragungsprotokolls durch das Prozessorinterface, durch das Koppelmodul oder durch den Kommunikationsspeicher in Frage kommen. Als weitere Möglichkeit kann die Störung oder der Ausfall der Übertragungsstrecke für die Steuersignale in Betracht gezogen werden. Oder die Steuerung des Kommunikationsspeichers ist gestört worden und befindet sich in einer Deadlock-Situation. Durch den Timeout kann sie sich daraus befreien.

Beim Koppelmodul kann der Grund darin liegen, daß die Übertragung des Antwortsignals

(“Ready” oder “Error”) vom Kommunikationsspeicher wegen einer Störung nicht erkannt wurde oder wegen eines Hardware-Fehlers verloren ging. Außerdem kann es sein, daß kein Speicher am ausgewählten M-Port des Koppelmoduls angeschlossen ist. Nach Ablauf der Timeout-Zeit wird der Weg selbständig wieder freigegeben.

Auch beim Prozessorinterface kann eine Störung der Steuersignale oder das Fehlen des Kommunikationsspeichers bzw. des Koppelmoduls dazu führen, daß kein Ende des Speicherzugriffs zurückgemeldet wird. In diesem Fall beendet der Timer den Zugriff.

Die Zeiten dieser Timeout-Zähler sind so gewählt, daß als erstes der Kommunikationsspeicher, dann das Koppelmodul und zuletzt das Prozessorinterface reagieren würde. Damit wird die beauftragende Einheit (der Prozessor) einen laufenden Transfer mit Sicherheit erst *nach* einer Zeitdauer abbrechen, die die ausführende Einheit (Kommunikationsspeicher) am längsten für die Bearbeitung (inklusive Timeout) benötigt. So kann es nicht vorkommen, daß Kommunikationsspeicher oder Koppelmodul noch mit der Abwicklung eines Transfers beschäftigt sind, während der Prozessor diesen Zugriff längst als beendet angesehen hat. Ein weiterer Zugriff des Prozessors auf diesen Kommunikationsspeicher könnte sonst zu einem weiteren Fehler in Form eines Timeouts oder eines Protokollfehlers führen.

Die Timeout-Zeit, die im Prozessorinterface bei einem Zugriff maximal gewartet werden soll, muß deutlich größer sein als die Zeit, die der Kommunikationsspeicher längstens mit einem Zugriff beschäftigt ist. Das ergibt sich daraus, daß sich bei Zugriffen von mehreren Prozessorknoten auf denselben Kommunikationsspeicher deutlich längere Zugriffszeiten ergeben können als bei nicht in Konflikt stehenden Zugriffen, weil auf die Beendigung der zuvor an die Reihe kommenden Zugriffe gewartet werden muß. Dies muß auch für den Fall zutreffen, daß alle Zugriffe erst durch Erreichen des Timeouts beim Kommunikationsspeicher oder beim Koppelmodul beendet werden. Daher muß ein Prozessorinterface in der Lage sein, mehrere fremde Zugriffe, die vor ihm an die Reihe kommen, auch bei extrem langer Bearbeitungszeit abwarten zu können. Daher beträgt seine Timeout-Zeit mindestens das zehnfache der Timeout-Zeit der anderen Komponenten. Dies ist etwa soviel wie die Anzahl der Prozessorknoten, die Zugriff auf einen Kommunikationsspeicher haben.

Die fairen Arbitrierungsprotokolle von Kommunikationsspeicher und Koppelmodul sorgen weiterhin dafür, daß es bei mehreren vorliegenden Zugriffswünschen nicht dazu kommt, daß ein Zugreifer benachteiligt wird und bedeutend länger als die anderen warten müßte [HIL92].

5.4.3 Signal-Fehler

Aus den bisherigen Ausführungen wurde deutlich, daß Störungen auf den Steuersignalen, die zwischen Prozessoren und Kommunikationsspeichern ausgetauscht werden, schwerwiegende Auswirkungen auf das Zugriffsprotokoll haben können. Um dem entgegenzuwirken, werden für die Übertragung der “logischen” Steuersignale physikalisch Differenzsignale verwendet. Dabei wird jedes einzelne Steuersignal durch je zwei Leitungen übertragen, deren Potentialdifferenz

entweder positiv oder negativ ist, je nach dem zu übertragenden logischen Signal. Eine Störung auf der Übertragungsstrecke wirkt sich i.a. so aus, daß das Potential aller Leitungen in die gleiche Richtung verschoben wird. Dabei bleibt die Differenz erhalten und kann somit auf der Empfängerseite störungsfrei zurückgewonnen werden. Nachteilig macht sich bei der Realisierung nur bemerkbar, daß die Signallaufzeiten durch den Einsatz eines Sende- und eines Empfänger-Bausteins insgesamt um etwa 50 nsec verlängert werden [TIC90].

5.5 Weitere Maßnahmen

Die standardmäßigen Implementierungen zur lokalen Fehlererkennung mit kurzer Latenzzeit im Motorola-System sind die Exceptions, die vom Prozessor erkannt werden, und die Parity-Absicherung beim Datentransfer. Weitere Maßnahmen zur Fehlertoleranz müssen durch Fehlerbehandlungs-Routinen oder durch vorbeugende Maßnahmen und Redundanz erreicht werden.

Die Implementierung der für die MEMSY-Topologie benötigten Verbindungswege zwischen Prozessoren und Kommunikationsspeichern mit Hilfe der Koppelmodule erfolgte in der Weise, daß es für jeden Prozessor zwei Wege gibt, über die er seinen ihm zugeordneten Kommunikationsspeicher erreichen kann. Die Wegewahl wird auf Basis der angegebenen physikalischen Adresse vorgenommen. Will man erreichen, daß statt dem Standardweg der Alternativweg für die Übertragung gewählt wird, so muß eine andere physikalische Adresse angegeben werden. Dies muß in die Adreßübersetzungstabellen eingetragen werden, welche die CMMU benutzt. Das müßte gegebenenfalls von einer Fehlerbehandlungs-Routine ausgeführt werden [HIL92].

Andere Maßnahmen können softwaremäßig implementiert werden, wie z.B. die Erzeugung und Überprüfung von Checksummen. Mit ihrer Hilfe ist es möglich, bei der Übertragung von Blökken solche Fehler zu entdecken, die durch die Übertragung eines falschen Wortes innerhalb eines Blocks entstehen [WAK78].

Auch die Überwachung eines zuvor festgelegten möglichen Programmablaufs kann zur Erkennung eines fehlerhaften Verhaltens herangezogen werden. Hier tun Watchdogs ihre Arbeit [MIC92] [PMH93]. Von ihnen gibt es sehr unterschiedliche Ausprägungen. Die einfachsten erwarten vom Prozessor nur, daß dieser periodisch spätestens nach einer vorher festgesetzten Zeit ein Lebenszeichen schickt (“I‘m alive” - Message). Wenn dieses Lebenszeichen ausbleibt, interpretiert der Watchdog dies als einen Fehler des Prozessors. Aufwendigere Watchdogs kontrollieren den Ablauf des zu beobachteten Programms genauer. Dies kann auf der Basis von Signaturen geschehen, die dieses Programm an den Watchdog überträgt [DGH93b]. Solche Watchdogs gehören zwar nicht zur Standardausstattung von MEMSY, ihr Einsatz und ihr Einfluß auf das Systemverhalten wird aber untersucht. Der Watchdog kann ein separater Prozessor oder ein Prozeß sein, der auf dem Prozessor des Anwendersystems läuft [HÖN94].

Eine noch stärkere Kontrolle würde durch eine direkte Beobachtung des Instruktionsbusses di-

rekt am Prozessor erfolgen können. Dabei wird jede Instruktion, die der Prozessor zur Bearbeitung aus einem Speicher holt, beobachtet und mit dieser Kenntnis die korrekte Bearbeitungsreihenfolge kontrolliert. Bei den heute benutzten Prozessoren ist dies aber oft nicht möglich, da Prefetch-Verfahren oder auf dem Prozessorchip befindliche Caches die benötigte Kontrolle nicht mehr erlauben [MPD94].

5.6 Fehlermodell für den Stabilen Speicher

In den vorangegangenen Abschnitten wurden verschiedene Möglichkeiten für das Auftreten eines Fehlers in einem Rechner betrachtet. Dabei wurde auch speziell auf die diesbezüglichen Eigenschaften von MEMSY eingegangen. Auch zu bereits vorhandenen und den noch möglichen Einrichtungen zur Fehlererkennung und -behebung sind bereits einige Angaben gemacht worden.

In diesem Abschnitt sollen nun die Fehler behandelt werden, die der Stabile Speicher tolerieren soll.

Im Vordergrund steht dabei der Schutz der abgelegten Daten im Stabilen Speicher. Sie müssen erstens innerhalb des Stabilen Speichers über eine lange Zeit unverändert gespeichert werden können. Und zweitens muß dafür gesorgt werden, daß sich bei Zugriffen auf diese Daten keine Fehler ereignen, die zu “unbeabsichtigtem Verlust oder Veränderung” führen. Hier spielt vor allem eine Rolle, daß Übertragungsfehler und eine fehlerhafte Arbeitsweise von Prozessoren rechtzeitig entdeckt werden, bevor sich zerstörerische Auswirkungen auf die Daten ergeben können. Auch muß der Stabile Speicher interne Fehler tolerieren können.

Die bei Speicherung und Zugriff beteiligten Einheiten sind der Prozessor, der Übertragungsweg und der Speicher. Die für die Fehlertoleranz erforderlichen Vorkehrungen sind dabei sehr unterschiedlich. Je nach spezifischen Fehlerausprägungen müssen an den verschiedenen Stellen unterschiedliche Schutzmaßnahmen getroffen werden.

5.6.1 Prozessor-Fehler

Zunächst werden Prozessor-Fehler betrachtet. Das ideale Verhalten eines Prozessors beim Auftreten eines Fehlers wäre ein Fail-Stop-Verhalten. Das bedeutet, daß er nach dem Fehler keine weitere Aktion mehr ausführt. Somit ist die Fehlerausbreitung sehr eingeschränkt. Für die Einhaltung dieses Verhaltens ist jedoch einiger Hardware-Aufwand nötig, der i.a. nicht bei der Konstruktion des Prozessors oder des Computers bereitgestellt wird. Diese Einheit müßte als Master-Checker-Paar mit einem zuverlässigen Vergleicher aufgebaut sein, der sofort nach einer festgestellten Diskrepanz die beiden Prozessoren stoppen könnte. Da solch eine Einrichtung beim verwendeten Motorola-System (und auch bei den meisten anderen Multiprozessorsystemen) nicht implementiert ist, kann es durchaus vorkommen, daß der Prozessor auch nach dem Auftreten eines (vor allem transienten) Fehlers noch weiterarbeitet. Dabei ist nicht sicherge-

stellt, daß der aufgetretene Fehler oder ein daraus hervorgegangener Folgefehler überhaupt entdeckt wird. Es könnten beispielsweise falsche Daten erzeugt worden sein. Ob dies aber zu einem falschen Programmablauf oder zu deutlich falschen Ergebnissen führen wird, die von einem Watchdog-Mechanismus bzw. von einem Akzeptanztest erkannt werden, ist zum Teil fraglich. Selbst bei einer erfolgreichen Fehlererkennung kann dieser Zeitpunkt sehr weit vom Auftreten des Fehlers entfernt sein. Betraf der Fehler eine Aktion, die der Prozessor im Stabilen Speicher ausgeführt hat, so kann diese schon längst abgeschlossen sein, bevor auf Grund des außerhalb des Stabilen Speichers erkannten Fehlers eine Gegenmaßnahme ergriffen werden kann. Daher muß der Stabile Speicher eigene Möglichkeiten besitzen, mit deren Hilfe er ohne äußere Unterstützung durch andere Fehlererkennungsmechanismen auch Prozessorfehler rasch entdecken kann. Diese Mechanismen müssen schnell genug reagieren können, bevor eine fehlerhafte Veränderung der gespeicherten Daten nicht mehr von alleine, d.h. ohne Hilfe durch den Prozessor, rückgängig gemacht werden kann.

Der Stabile Speicher wird mit mehreren Prozessoren zusammenarbeiten müssen, d.h. mehrere Prozessoren werden Zugriff auf den Stabilen Speicher haben. Dies soll schon aus dem Grund möglich sein, damit im Fall eines dauerhaften Ausfalls eines Prozessors ein “Ersatz-Prozessor” die Möglichkeit hat, auf die Daten des ausgefallenen Prozessors zuzugreifen. Im Normalfall soll aber jeder Prozessor nur seine eigenen Daten erreichen können. Der Stabile Speicher muß also in der Lage sein, die Prozessoren voneinander zu unterscheiden. Die Fehlersituation, daß ein Prozessor auf die Daten eines anderen zugreifen möchte (außer im “Ersatz-Fall”), muß erkannt werden. Dafür ist ein Schutzmechanismus mit Hilfe von Paßwörtern vorgesehen.

Weiterhin muß sichergestellt werden können, daß ein mit inkonsistentem Inhalt übergebener Datenblock als solcher erkannt werden kann, bevor eine endgültige Abspeicherung im Stabilen Speicher erfolgt. Da der Stabile Speicher keine Informationen darüber hat, wie ein “korrekter Inhalt” (semantische Kontrolle) eines abzuspeichernden Datenblocks aussieht, muß dies mit formalen Methoden erreicht werden können. Daher wird die Erzeugung und Überprüfung eines Checkworts eingesetzt [WAK78]. Der Prozessor muß beim Schreiben für den “Netto-Datenblock” ein Checkwort bilden und als letztes übertragen. Im Stabilen Speicher wird auf gleicher Weise die Checkwort-Bildung betrieben. Sollte das übertragene Checkwort und das vom Stabilen Speicher erzeugte Checkwort nicht übereinstimmen, so muß es innerhalb des Stabilen Speichers möglich sein, diese Aktion rückgängig zu machen.

Mit diesem Verfahren wird eine gute Fehlerüberdeckung erreicht, wenn der Fehler darin besteht, daß einzelne korrekte Worte durch andere Worte ersetzt werden. Dies kann z.B. dann auftreten, wenn der Prozessor aus einem falschen Register heraus überträgt, oder wenn eine wegen eines zuvor aufgetretenen Übertragungsfehlers eingeleitete Wort-Wiederholung nicht oder fehlerhaft abläuft. Entsprechende Versuche mit 32-bit-Zahlen zeigten, daß alle falsch zusammengestellten Blöcke entdeckt wurden. Dabei machte es keinen Unterschied, ob ein einfacher Checksummenalgorithmus, der nur eine Addition ohne Berücksichtigung eines Übertrags macht und ein 32-bit-Ergebnis produziert, oder ein aufwendigerer Algorithmus mit Berücksichtigung des Übertrags und mit Speicherung als 64-bit-Zahl angewandt wurde.

Unerkannt blieben die Fehler eher dann, wenn eine einzelne Bitstelle immer auf einen festen Wert gezwungen wurde. Bei diesen Bitspaltenfehlern konnte keiner dieser Checksummenalgorithmen überzeugen. Diese Fehler werden aber viel eher vom Parity-Checker entdeckt. Bei einer Vertauschung zweier Worte innerhalb des Datenblocks sind allerdings beide Fehlererkennungsmechanismen machtlos.

Als Fazit läßt sich sagen, daß bei einer Wortbreite von 32 Bit das Risiko sehr gering ist, daß ein Fehler unerkannt bleibt, der durch das Einschleusen eines oder mehrerer falscher Worte an Stelle des richtigen Wortes in den mit einer Checksumme gesicherten Datenblock erzeugt wird. Das größte Risiko dabei ist immer noch jenes, bei dem schon bei der Erzeugung des Datenblocks und der zugehörigen Checksumme mit falschen Daten gearbeitet wurde. Führt dieser Prozessor keine fehlerhaften Zugriffe gegenüber dem Stabilen Speicher aus, kann der Stabile Speicher daran kein vorhergehendes Fehlverhalten dieses Prozessors erkennen. Hier ist der Stabile Speicher überfordert. Andere Einheiten, wie z.B. eine Kontrollflußüberwachung durch einen Watchdog, könnten hier eingreifen [MIH91]. Wichtig wäre dabei, daß das Anhalten dieses Prozessors erfolgt, bevor er einen Auftrag an den Stabilen Speicher erteilt hat. An diesem Beispiel wird deutlich, wie wichtig eine durch mehrere Einheiten vollzogene Fehlererkennung und das “Unschädlichmachen” des betroffenen Prozesses oder Prozessors ist.

Als weiteren Fehler muß der Stabile Speicher auch einen plötzlichen Stop eines Prozessors tolerieren können. Bei einem Wiederanlauf muß sichergestellt werden können, daß eine unterbrochene Aktion wieder bereinigt werden kann. Dies wird, wie die spätere Beschreibung der Implementierung zeigen wird (siehe Abschnitt 6.4.3, Datentransfer), nur während der Übertragung eines Datenblocks nötig sein. Hierfür gibt es die Möglichkeit, dem Stabilen Speicher anzuzeigen, daß der derzeit laufende Transfer nun abgebrochen werden soll. Alle anderen Aktionen werden mit einem einzigen Schreibzugriff gestartet; und diese sind aus Sicht des Prozessors atomar durchführbar. Ein Crash ereignet sich daher entweder davor (noch kein Start) oder dahinter (Start ist erfolgt, die Ausführung wird vom Stabilen Speicher allein durchgeführt).

Die erkennbaren Fehler des Prozessors, die der Stabile Speicher tolerieren soll, sind demnach:

- Zugriff auf fremde Daten,
- fehlerhafte Bearbeitung eines Auftrags-Protokolls,
- Übertragung fehlerhafter Datenblöcke und
- Totalausfälle eines Prozessors.

5.6.2 Fehler auf der Übertragungsstrecke

Neben den Prozessoren gibt es weitere Teileinheiten, bei denen sich Fehler ereignen können, so die Übertragungsstrecke. Das Multiprozessorsystem MEMSY verfügt über umfangreiche Hardware zwischen den Prozessoren und dem Stabilen Speicher. Ein Zugriff eines Prozessors bzw. einer CMMU läuft innerhalb des Motorola-Systems durch mehrere Stufen (Treiber) und gelangt

zu den Boards des Hauptspeichers und an das Prozessorinterface. Nachdem durch Steuersignale entschieden worden ist, daß dieser Zugriff über das Interface zu den Kommunikationsspeichern geführt werden soll, werden zwei Bits aus der Adresse zur Wegewahl benutzt. Ähnliches passiert (durch Interpretation von weiteren vier Bits) in dem sich anschließenden Koppelmodul. Danach ist der Kommunikationsspeicher erreicht. Hier entscheiden zwei weitere Bits, ob der Zugriff diesem Kommunikationsspeicher oder dem angeschlossenen Stabilen Speicher gelten soll. Wird irgendwo auf diesem Weg eine falsche Entscheidung getroffen, so erreicht der Zugriffswunsch einen falschen Speicher, möglicherweise einen falschen Stabilen Speicher. Aus Sicht des Stabilen Speichers liegt hierbei die gleiche Situation vor wie beim Zugriff durch einen falschen Prozessor. Dieser Fehler muß entdeckt werden, bevor dieser Zugriff nicht mehr rückgängig zu machende Auswirkungen auf die abgelegten Daten haben kann.

Der an dieser Stelle wirksame Schutz wird vor allem durch die Vergabe eines eng begrenzten Adreßbereichs innerhalb der zur Verfügung stehenden 4 MByte und durch die Überprüfung eines Paßwortes erreicht. Dazu ist es notwendig, daß die Vergabe dieser Adreßbereiche und der Paßworte in den verschiedenen Stabilen Speichern des Multiprozessorsystems bereits bei der ersten Erteilung unabhängig voneinander in den allermeisten Fällen zu unterschiedlichen Ergebnissen kommt. Das soll auch dann erreicht werden, wenn das Anwendersystem in allen Knoten in derselben Programmablaufsituation, d.h. zur selben Zeit, ein Paßwort von einem Stabilen Speicher anfordert.

Übertragungsfehler, bei denen der Wert eines einzelnen Bit verändert worden ist, müssen ebenfalls sofort am Stabilen Speicher erkannt werden. Wenn dabei die Bits für die Wegesteuerung betroffen sind, kann dies auch die Ursache für eine falsche Wegewahl sein. Einzelne veränderte Bits können durch Parity-Sicherung erkannt werden. Dabei sollen sowohl die Daten als auch die Adressen mit diesem Schutz versehen sein. Denn gerade unerkannte Adreßfehler können zu sehr schwer zu diagnostizierenden Fehlern führen, wenn man nicht mehr herausfinden kann, an welcher falschen Stelle eine Schreiboperation stattgefunden hat.

Für Fehler, die sich auf dem Übertragungsweg zwischen Prozessor bzw. CMMU und dem Stabilen Speicher ereignen, müssen folgende Fehlerarten toleriert werden können:

- falsche Wegewahl (und dadurch Übertragung zu falschem Ziel),
- Einzelbit-Fehler bei Adressen und Daten.

5.6.3 Fehler innerhalb des Stabilen Speichers

Eine weitere und noch kritischere Fehlerquelle ist der Stabile Speicher selbst. Hier kann keine äußere Kontrolle einen Fehler verhindern oder korrigieren. Daher müssen innerhalb des Stabilen Speichers Vorkehrungen getroffen werden, so daß eine Gefährdung der Daten vor unbeabsichtigter Zerstörung so klein wie möglich gehalten werden kann.

Die wichtigsten Ziele sind dabei:

- unmittelbare Eigenkontrolle der zentralen Ablaufsteuerung im Stabilen Speicher; Fehler müssen sofort erkannt werden und dürfen sich nicht ausbreiten;
- im Fall eines innerhalb des Stabilen Speichers aufgetretenen dauerhaften, aber begrenzten Fehlers muß es noch möglich sein, die Arbeitsweise fortzusetzen, solange sich kein zweiter Fehler ereignet; die Prozessoren müssen aber über diesen Fehler informiert werden können, damit sie in angemessener Zeit noch geeignet reagieren können, z.B. durch Verlagern ihrer Daten in einen anderen (stabilen) Speicher;
- spontane Veränderungen in den abgespeicherten Daten müssen intern behebbar sein; von außen darf dieser Fehlereinfluß nicht bemerkt werden;
- im Fall eines Versagens, einen neuen Zustand eines Objekts zu erzeugen (Update) muß der alte Zustand (vor Beginn der Update-Operation) wiederhergestellt werden können.

Der Ablauf einer Zugriffsoperation wird auf die Weise ablaufen, daß sie vom Anwenderprozeß mit wenigen Zugriffen angestoßen wird, innerhalb des Stabilen Speichers aber umfangreiche Aktionen zur Folge haben wird. Da diese von außen nicht überwacht werden können, muß intern für eine umfassende Fehlererkennung gesorgt werden. Mit einer TMR-Schaltung (Triple Modular Redundancy) mit einem Voter lassen sich aufgetretene Fehler eines der drei Subsysteme an den Ausgängen der Schaltung sofort erkennen und (im Gegensatz zu einem Master-Checker-Paar) auch ohne Unterbrechung der aktuellen Bearbeitung maskieren. Bis zum Auftreten eines weiteren solchen Fehlers ist eine Weiterarbeit noch möglich. Erst ein weiterer Fehler macht eine Fortführung des TMR-Systems unmöglich.

Gegen Fehler, die sich in die abgespeicherten Daten einschleichen, kann eine zweifache Speicherung der Daten und deren regelmäßige Kontrolle auf paarweise Gleichheit wirken. Bei einer Diskrepanz muß festgestellt werden, welche Daten noch korrekt sind und welche nicht. Anschließend können die korrekten Daten auf den fehlerbehafteten Teil übertragen werden. Hier kann auch auf die für die Erkennung von Übertragungsfehlern verwendete Technik des Parity-Schutzes zurückgegriffen werden, wenn die Paritybits zusammen mit den “Nutzdaten” abgespeichert werden.

Mit diesem Verfahren lassen sich sporadisch auftretende “Umwelteinflüsse” auf die gespeicherten Daten beheben. Die angesprochenen Umwelteinflüsse werden Alpha-Teilchen zugeschrieben, die im Chipgehäuse entstehen. Diese können die Ladungssituation einer Speicherzelle eines dynamischen Speichers (DRAM) beeinflussen und damit die abgespeicherte Information verändern. Dieser Fehler wird im Gegensatz zu den Übertragungsfehlern mit gekippten Bits allerdings erst bei einer anschließenden Leseoperation durch das Auftreten eines Parity-Fehlers entdeckt. Für das Funktionieren dieses Verfahrens ist es wichtig, daß über einen Zeitraum, der deutlich länger ist als der Turnus von einem Check des Objekts bis zu seinem nächsten Check, höchstens nur vereinzelt Bits umkippen. Dies ist bei den heute gebräuchlichen Speicherchips (DRAM) auch der Fall [CYP92], denn die Hersteller der Speicherchips sind sich dieser Fehler-

quelle bewußt und beugen ihr durch geeignete Maßnahmen im Chipgehäuse zum Schutz vor Alpha-Teilchen vor. Daher dürfte dieser Fehler nur noch sehr selten vorkommen. Schon ein einfacher Versuch, bei dem ein Kommunikationsspeicher mit bekannten Werten beschrieben wurde und anschließend lange Zeit unangetastet blieb, konnte dies bestätigen. Nach etwa 170 Stunden (1 Woche) hatte sich in den 4 MByte noch kein Bit verändert.

Diese Forderung ist deutlich schwächer als die Forderung von Lampson, bei der über einen solchen Zeitraum nur ein oder wenige Bits kippen dürfen, was bereits zu einem ungültigen Block führen würde. Diese hier geltende schwächere Forderung kommt daher zustande, daß hier jedes Wort mit vier Paritybits abgelegt ist. Dann sind auch mehrere umkippende Bits innerhalb eines Objekt-Paares kein Grund für eine unwiderrufbare Zerstörung des gespeicherten Objekts, solange sie nicht im selben Wortpaar auftreten. Unter Umständen können sogar mehrere gekippte Bits im selben Wortpaar wieder repariert werden, solange sie nicht gleichzeitig in den entsprechenden Bytes auftreten.

Mit Hilfe der zweifachen Abspeicherung läßt sich auch während des Updates eines Objekts sicherstellen, daß im Fall eines fehlerhaften Datenblocks der alte Objektzustand wiederhergestellt werden kann. Diese Vorgehensweise wurde auch bei anderen Rechnern angewendet [BER88] [BOP93].

Eine weitere Fehlerursache kann der Ausfall eines Speicherchips sein. Da die Chips im Hauptspeicher von MEMSY und in den Kommunikationsspeichern (und auch im Stabilen Speicher) so organisiert sind, daß sie je ein Bit von vielen Worten aufnehmen, macht sich dieser Fehler in der gleichen Weise bemerkbar wie ein einzelnes umgekipptes Bit. Allerdings ist in diesem Fall gleich ein ganzer Speicherbereich betroffen. Eine doppelte Speicherung in verschiedenen Speicherbänken ermöglicht auch in diesem Fall, daß beim Lesen dem Prozessor ein korrekter Datenblock geliefert werden kann. Für das Schreiben und für die regelmäßige Kontrolle auf paarweise Gleichheit sieht die Situation aber nicht so gut aus. In diesem Fall muß ein anderer Speicherplatz innerhalb des Stabilen Speichers gefunden werden, der den defekten Chip nicht benutzt. Sofern ein solcher Platz gefunden werden kann, soll eine Verlagerung ohne Einfluß und Wissen des Anwenderprozesses möglich sein. Ansonsten muß der Anwenderprozeß wie im Fall eines dauerhaften Defekts der zentralen Steuerung über die Situation informiert werden, damit dieser geeignet reagieren kann.

Als weitere Fehlermöglichkeit kommt auch in Frage, daß der Speicherzugriff selbst fehlerhaft abläuft. Handelt es sich dabei um transiente Fehler, so können diese i.a. durch eine Wiederholung des Zugriffs korrigiert werden. Liegt jedoch ein permanenter Fehler vor, muß die Hardware ausgewechselt werden. Zur Fehlererkennung kann beim Lesen eines Wortes bzw. eines Datenblocks die Auswertung der Parity-Information und der Checksumme herangezogen werden. Ein beim Schreiben eines Wortes auftretender Zugriffsfehler läßt sich durch ein anschließendes Lesen und Vergleich mit dem ursprünglichen Wort (Kontrollesen) erkennen. Auch ein auftretender Timeout kann auf einen Zugriffsfehler hindeuten.

6 Ein Stabiler Speicher für MEMSY

In diesem Abschnitt wird der Stabile Speicher vorgestellt, der für das Multiprozessorsystem MEMSY entwickelt worden ist. Die Anforderungen an diesen Stabilen Speicher orientieren sich an dem im letzten Kapitel vorgestellten Fehlermodell. Daraus leitet sich seine Struktur ab. Bei der Vorstellung seiner Arbeitsweise wird der Datenübertragung für den Update eines "Stabilen Objekts" besondere Beachtung geschenkt. Als Stabiles Objekt wird die im Stabilen Speicher abgelegte Kopie des Objekts bezeichnet, das der Anwenderprozeß übergeben hat. Daran schließen sich die detaillierteren Beschreibungen der einzelnen Teilkomponenten an. Die tolerierbaren Fehler und die Abwehrmaßnahmen dieses Stabilen Speichers werden bei den jeweiligen Abschnitten genauer dargestellt und am Ende dieses Kapitels zusammengefaßt.

6.1 Anforderungen an den Stabilen Speicher

Die Anforderungen, die an den Stabilen Speicher gestellt werden, orientieren sich an verschiedenen Gesichtspunkten.

Zunächst einmal muß es im Stabilen Speicher möglich sein, ein aus der Sicht des Anwendersystems atomaren Update eines Stabilen Objekts durchführen zu können. Dies lehnt sich an die Vorschlägen von Lampson an. Da es sich hier um eine Implementierung handelt, bei der das Speichermedium aus RAM-Bausteinen besteht, wird es bezüglich des Ablaufs und der Fehlererkennungsmöglichkeiten einige Abweichungen zu den Vorschlägen von Lampson geben. Aber die Eigenschaft, daß die zur Verfügung gestellten Operationen nur auf ganze Stabile Objekte wirken dürfen und nicht auf einzelne Worte innerhalb des Objekts, ändert sich nicht.

Ebenso muß die Persistenz der Stabilen Objekte gewährleistet sein. Das bedeutet, daß ein einmal im Stabilen Speicher abgelegtes Objekt so erhalten werden kann, daß auch nach sehr langer Zeit dieses Objekt noch ohne Veränderung gelesen werden kann. Diese Zeit sollte deutlich über der MTBF des Gesamtsystems liegen. Üblicherweise wird ein Sicherungspunkt im Stabilen Speicher zwar sehr viel kürzer aufgehoben werden müssen als diese angegebene Zeit. Dennoch wird eine sehr lange Zeit gefordert, da dieser Stabile Speicher nicht nur in der Lage sein soll Sicherungspunkte aufzunehmen sondern auch andere Daten sicher speichern soll, die nicht ab und zu einmal verändert werden. Bei so langen Zeitvorgaben kann man nicht davon ausgehen, daß einmal geschriebene Daten im Speicher stets unverändert bleiben. Es ist eher zu erwarten, daß während einer größeren Zeitspanne, ähnlich der oben angedeuteten MTBF, ein Bit auf Grund eines (transienten) Fehlers seinen Wert verändert. So müssen innerhalb des Stabilen Speichers einige Vorkehrungen dazu getroffen werden, die einen aufgetretenen Fehler rechtzeitig erkennen, so daß er ohne Zutun von außen sicher behoben wird.

Neben diesen Anforderungen müssen weitere Forderungen erfüllt sein, um die Datensicherheit der Stabilen Objekte zu gewährleisten.

Sehr wichtig ist, daß defekte Prozessoren nicht den Inhalt des Stabilen Speichers zerstören können. Dies geht über die Annahmen von Lampson hinaus, der als Prozessor-Fehler nur einen (plötzlichen) Crash vorgesehen hat. Hier wird auch ein über längere Zeit fehlerhaft arbeitender Prozessor angenommen. Die Dauer, die ein Prozessor bis zu seiner Entdeckung fehlerhaft arbeiten darf, wird nicht eingeschränkt. Verschiedene Fehlererkennungsmechanismen, die bereits in das System eingebaut sind bzw. noch werden, sollen zwar einen Prozessorfehler möglichst rasch entdecken und den betreffenden Prozessor stillegen, so daß kaum eine Ausbreitung des Fehlers erfolgt. Dennoch sollen vom Stabilen Speicher aus keine Anforderungen an diese Einheiten gestellt werden. Da nämlich beim Stabilen Speicher, wäre er ohne eigene Schutzvorkehrungen ausgestattet, schon ein einfacher Speicherzugriff zur Zerstörung eines Stabilen Objekts führen könnte, müßten diese anderen Fehlertoleranzmechanismen schon sehr schnell, nämlich bereits während des Speicherzugriffs, reagieren können. Dies ist aber bei Einheiten, die nicht wenigstens aus Master-Checker-Paaren aufgebaut sind, eine viel zu hohe Forderung. Aus diesem Grund wird der Stabile Speicher so angelegt, daß kein Prozessor direkten Zugriff auf die Stabilen Objekte hat. Dies ist auch ein Unterschied zu der Einrichtung bei der FTM, bei der nach dem Öffnen eines Stabilen Objekts der Prozessor direkten Zugriff auf die erste Kopie hat (siehe Abschnitt 2.2). Der hier implementierte Stabile Speicher arbeitet dagegen ganz autonom vom Prozessor. Der Stabilen Speicher nimmt vom Prozessor nur die Aufträge mit Parametern und gegebenenfalls die Daten entgegen. Die Ausführung dieser Aktion erfolgt dann alleine durch den Stabilen Speicher. Der Prozessor wird zur Datenübertragung nur die Möglichkeit haben, in einen Puffer zu schreiben oder aus diesem zu lesen. Die Stabilen Objekte bleiben dabei noch unberührt.

Weiterhin muß die Stabile Speicher in der Lage sein, quasi gleichzeitig für mehrere Prozessoren tätig zu sein. Das ergibt sich daraus, daß auf Grund der nicht zu unterschätzenden Kosten der beabsichtigten Implementierung nicht jeder Prozessorknoten einen eigenen Stabilen Speicher bekommen wird. Daher werden mehrere Prozessoren Zugang zum selben Stabilen Speicher haben. Ferner ist anzunehmen, daß bei einer verteilten Anwendung alle daran beteiligten Prozessoren etwa zur gleichen Zeit ihren Sicherungspunkt anlegen wollen [CHL85] [DHL89]. Dabei sollen sie sich am Stabilen Speicher gegenseitig nicht übermäßig behindern. Vor allem das gleichzeitige Ablegen einer größeren Datenmenge soll simultan ausgeführt werden können.

Bei dieser nebenläufigen Arbeitsweise besteht die Gefahr, daß das Fehlverhalten eines Prozessors die Arbeit der intakten Prozessoren mit dem Stabilen Speicher derart stört, daß sich bei ihrer Datenübertragung Fehler einstellen, obwohl sie selbst korrekt arbeiten. Dies muß erkannt werden, bevor es zu einer nicht mehr rückgängig zu machenden Datenveränderung im Stabilen Speicher kommen kann. Dies verlangt nach besonderen Schutzmaßnahmen des Stabilen Speichers, die die einzelnen Prozessoren voreinander schützen.

Der Stabile Speicher hat eine besondere Funktion in dem Multiprozessorsystem. Ihm werden Daten anvertraut, die man für besonders schutzwürdig hält. Damit er seinem Namen gerecht werden kann, muß dafür gesorgt werden, daß er selbst nicht einen Defekt erleidet, der zum Verlust der Daten führen kann. Um sicherzustellen, daß sich interne Fehler nicht ausbreiten können,

müssen interne Fehler sofort erkannt werden können.

Es würde aber nichts nützen, wenn der Stabile Speicher dann dauerhaft in einen STOP-Zustand gehen würde. Die abgelegten Daten wären dann nicht mehr erreichbar und für den Anwender verloren. Das muß verhindert werden. Daher wird die Steuerung des Stabilen Speichers in TMR-Technik (Triple Modular Redundancy) errichtet. Durch die Maskierung des Fehlers kann die Arbeit im Stabilen Speicher ohne Unterbrechung fortgesetzt werden. An einer geeigneten Stelle kann dann versucht werden, den ausgefallenen Teil der Steuerung wieder neu aufzusetzen. War ein transienter Fehler die Ursache für den Ausfall, so können anschließend wieder alle drei Teileinheiten der Steuerung arbeiten. War dagegen ein permanenter Fehler die Ursache, so hat sich der Stabile Speicher in einen nicht mehr stabilen Speicher gewandelt. Dann muß es noch möglich sein, die bisher abgelegten Daten zu lesen und anderweitig sicher abzulegen.

Einige weitere Forderungen betreffen die Voraussetzungen, die dem Stabilen Speicher in seiner Umgebung zur Verfügung gestellt werden müssen.

In einem Speicher abgelegt Daten nützen nichts, wenn der Speicher von den Prozessoren abgeschnitten ist. Daher sollte es mindestens zwei Wege geben, auf die ein Prozessor den Stabilen Speicher erreichen kann. Diese Forderung ist bei MEMSY zu einem großen Teil schon durch die bereits bestehende Verbindungsstruktur der Speicherkopplung erfüllt. Aus diesem Grund wird hier kein weiteres Verbindungssystem errichtet.

Als nächstes muß die Möglichkeit zur Fehler-Erkennung gegeben sein. Hier wird die bestehende Parity-Überprüfung verwendet. Weitere Maßnahmen zur Erkennung von Ablauf-Fehlern (Protokoll-Fehler) müssen noch eingerichtet werden. Dafür müssen im Stabilen Speicher bzw. in der Schnittstelle zwischen ihm und den Prozessoren entsprechende Schutzmaßnahmen eingerichtet werden. Die Rückmeldung eines erkannten Protokollfehlers oder eines Datentransferfehlers an den Prozessor muß ebenfalls möglich sein. Das in der Erweiterungshardware vorhandene Error-Signal sollte nicht dazu verwendet werden, denn dieses ist für Fehler gedacht, die sich auf dem Übertragungsweg ereignen. Daher muß eine neue Fehler-Rückmeldeeinrichtung vorgesehen werden. Da das bestehende Verbindungsnetz nicht mehr verändert werden soll, wird die Fehlerrückmeldung über Einträge in Status- oder Ergebnis-Registern erfolgen. Über dieses wird auch die Rückmeldung der oben genannten kritischen Zustände erfolgen.

6.2 Struktur des Stabilen Speichers

Die dargestellten Forderungen führen zu folgender Struktur für den Stabilen Speicher. Der Stabile Speicher wird eine eigenständige Einheit sein, ein Coprozessor, der über eine Speicherschnittstelle an den Bus des Kommunikationsspeichers angeschlossen wird. Aus Sicht der Prozessoren läßt sich dieser Stabile Speicher genauso ansprechen wie der Kommunikationsspeicher. Allerdings findet bei einem solchen Zugriff mehr statt als ein einfacher Datentransfer. Vor allem werden die Zugriffswünsche der Prozessoren auf ihre Berechtigung hin überprüft.

Der Stabile Speicher hat eine Unterteilung in drei Bereiche (siehe auch Abb. 17):

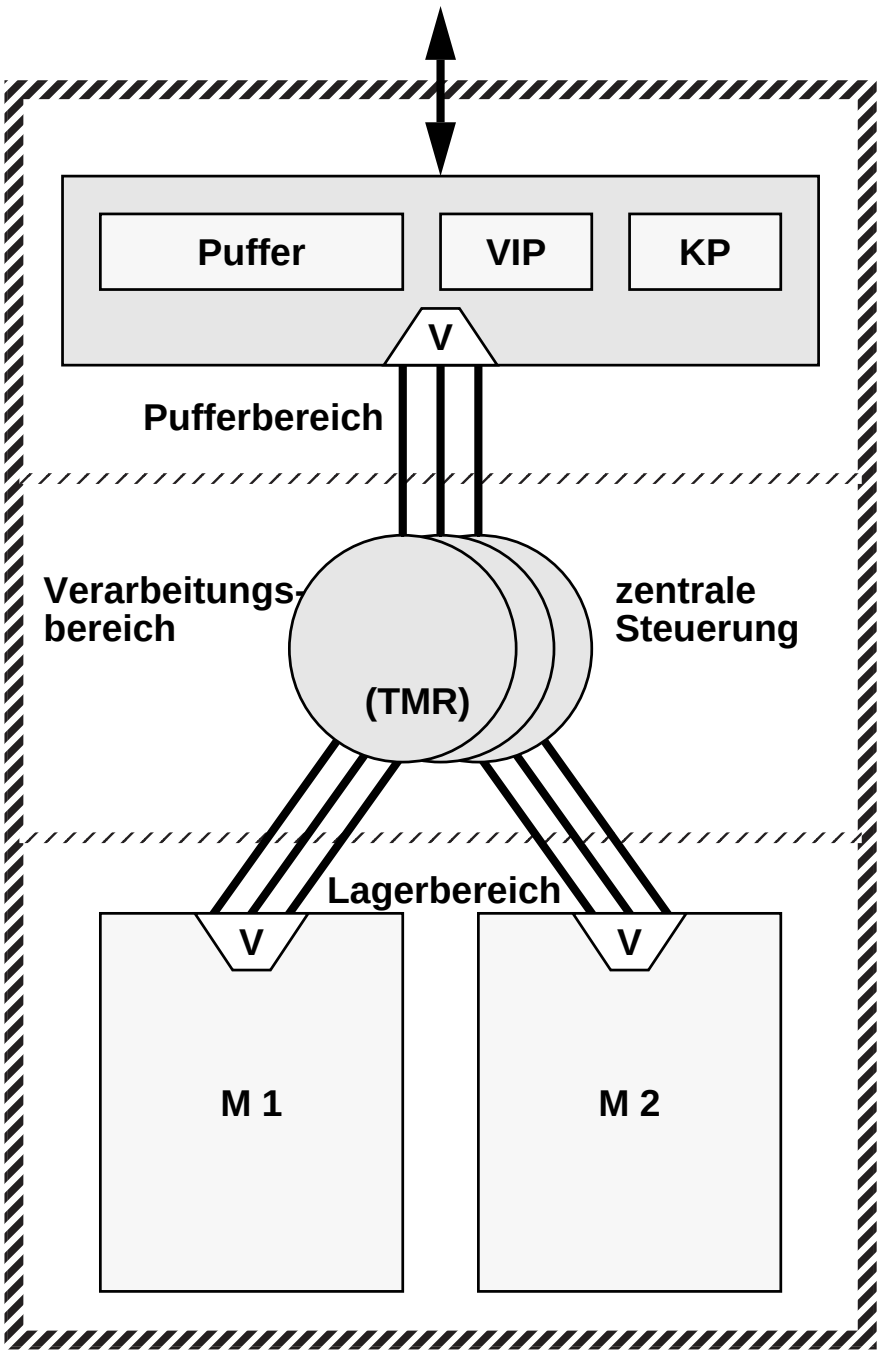
- Pufferbereich:
  Der Pufferbereich bildet die Schnittstelle zwischen dem Anwendersystem (Prozessoren und Kommunikationsspeicher) und dem Stabilen Speicher. Die Prozessoren können bis hier hin zugreifen und ihre Aufträge ablegen, um dem Stabilen Speicher mitzuteilen, was er tun soll. Ebenso steht hier ein Pufferspeicher zur Verfügung, über den die Stabilen Objekte ausgetauscht werden. Hier werden auch Datenstrukturen abgelegt und aktuell gehalten, mit deren Hilfe über die Zugriffsberechtigung entschieden werden kann. Die Aktionen in diesem Bereich werden von einer “Puffersteuerung” ausgeführt.

- Verarbeitungsbereich / zentrale Steuerung:
  Als nächstes schließt sich die zentrale Steuerung des Stabilen Speichers an. Sie hat die Aufgabe, die im Pufferbereich abgelegten Aufträge auszuführen. Ihre Vorgehensweise orientiert sich am Algorithmus von Lampson. Zusätzlich müssen auch noch weitere Auftragsarten, vor allem Verwaltungsaufgaben, bearbeitet werden. Der zentralen Steuerung obliegt auch die Kontrolle, ob der angegebene Auftrag und die übergebenen Parameter konform zu den Angaben über die Stabilen Objekte sind. Auf Grund der zentralen Bedeutung dieser Steuerung für die Funktionsweise des Stabilen Speichers wird sie zum Schutz gegen Ausfälle in TMR-Technik aufgebaut. Im Fall, daß eine der drei Einheiten (vorübergehend) ausfällt, kann eine Fortführung der Arbeit zumindest zeitweise (bis zum Auftreten des nächsten Fehlers) noch aufrechterhalten werden.

- Lagerbereich / Objektspeicher:
  Der dritte Bereich ist der Lager-Bereich für die Stabilen Objekte. Er besteht aus zwei “Objektspeichern”. Ein Stabiles Objekt wird als zweifache Kopie in diesen beiden Objektspeichern abgelegt. Auf diesen Bereich hat nur die zentrale Steuerung des Stabilen Speichers Zugriff. Dies soll die Stabilen Objekte vor unkontrollierten Veränderungen durch die Anwenderprozesse schützen.

Die Aufgaben, welche die Puffersteuerung und die zentrale Steuerung übernehmen, sind von sehr unterschiedlicher Natur. In der zentralen Steuerung stehen die Behandlung eines Stabilen Objekts und dessen sichere Aufbewahrung im Vordergrund. Dem gegenüber sieht der Ablauf, der sich im Pufferbereich abspielt, ganz anders aus. Hier ist es wie beim Kommunikationsspeicher an der Tagesordnung, daß mehrere Prozessoren einzelne Worte schreiben oder lesen wollen. Jeder Zugriff kann ein anderes Objekt betreffen, und Zugriffe zur Auftragsvergabe und zum Datenaustausch erfolgen ohne erkennbare Ordnung durcheinander, da jeder Prozessor sich in einer anderen Bearbeitungsphase befinden kann.

In dieser Situation muß die Puffersteuerung in der Lage sein, einen größtmöglichen Schutz vor unberechtigten Zugriffen auf die abgespeicherten Objekte gewährleisten zu können. Dabei spielen “Kommunikationspages” innerhalb des Pufferbereichs eine große Rolle.



V: synchronisierender Voter
VIP: Vermittlungs- und Informations-Page
KP: Kommunikations-Page

Abb. 17 Struktur des Stabilen Speichers bei MEMSY

In den nachfolgenden Abschnitten werden zunächst die grundsätzliche Arbeitsweise und die möglichen Aufträge vorgestellt. Daran schließen sich die Beschreibungen der drei Unterbereiche (Pufferbereich, zentrale Steuerung, Lagerbereich) an. In allen Bereichen werden immer wieder die getroffenen Schutzvorkehrungen gegen verschiedene Fehler besprochen. Schließlich ist ein Abschnitt den Votern gewidmet.

Zum Schluß werden einige Forderungen gestellt, die das Host-System (hier das Motorola-System) erfüllen muß, damit nach dem Auftreten eines (transienten / permanenten) Prozessorfehlers gewährleistet ist, daß im Anschluß daran auch weiterhin auf die abgelegten Stabilen Objekte zugegriffen werden kann.

6.3 Arbeitsweise des Stabilen Speichers und die Aufträge

In diesem Abschnitt wird zunächst in groben Zügen die grundsätzliche Arbeitsweise vorgestellt, wie die Anwenderprozesse den Stabilen Speicher veranlassen können, Aktionen auf den Stabilen Objekten auszuführen.

Ein Stabiles Objekt ist repräsentiert als ein eindimensionale Feld von 32-bit-Werten, dessen Länge vom Anwender festgelegt worden ist. Ergänzt wird jedes Stabile Objekt mit einer Checksumme (siehe Abb. 18). Zur Erkennung von Übertragungsfehlern sind alle Worte mit 4 Paritybits versehen. Bei Schreiboperationen werden Parity und die Checksumme vom Anwender übertragen und vom Stabilen Speicher überprüft. Bei Leseoperationen ist es umgekehrt.

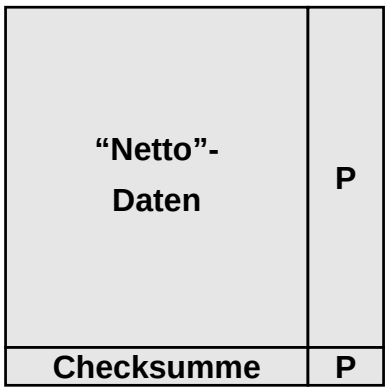


Abb. 18 Stabiles Objekt

Anwenderprozesse, die den Stabilen Speicher nutzen wollen, müssen die Möglichkeit haben, Stabile Objekte im Stabilen Speicher anzulegen, zu schreiben, zu lesen und auch zu löschen. Dies erfolgt in Form von Aufträgen, die die Anwenderprozesse an den Stabilen Speicher übergeben. Zu diesem Zweck können die Prozessoren des Motorola-Systems auf einige Register innerhalb des Pufferbereichs des Stabilen Speichers zugreifen, in die sie den Auftrag und die da-

zugehörigen Parameter ablegen können (ein “Auftragsregister” und drei “Parameterregister”). Der Austausch der Datenblöcke erfolgt über einen Pufferspeicher. Diese Einrichtungen befinden sich alle innerhalb des Pufferbereichs.

Sobald alle notwendigen Angaben eingetragen und im Fall eines Schreibauftrags auch die Daten vom Prozessor in den Pufferspeicher übertragen worden sind, wird der Auftrag von der Puffersteuerung in eine Warteschlange (Auftrags-FIFO) eingetragen. Die zentrale Steuerung des Stabilen Speichers entnimmt ihn daraus und bearbeitet den Auftrag.

Bei Leseaufträgen wird das Stabile Objekt von der zentralen Steuerung in den Puffer eingetragen. Nach Abschluß des Auftrags durch den Stabilen Speicher kann der Prozessor den Datenblock dort lesen. Für die Durchführung des Datenaustausches stehen zwei Register zur Verfügung, ein “Transferregister” und ein “Wiederholungsregister”. Ihre Benutzung wird in Abschnitt 6.4.3 (Datentransfer) genauer erklärt.

Zum Abschluß trägt die zentrale Steuerung in einem “Ergebnisregister” ein, ob dieser Auftrag erfolgreich durchgeführt werden konnte oder nicht. Im Fall einer fehlgeschlagenen Ausführung wird darin auch der Grund dafür vermerkt und zusätzlich, in welchem Zustand sich das Stabile Objekt nun befindet. So kann sich z.B. am Ende eines Schreibauftrags das Stabile Objekt noch im alten Zustand befinden, so als wäre der Auftrag nicht erteilt worden. Dies geschieht dann, wenn der übergebene Datenblock vom Stabilen Speicher nicht akzeptiert worden ist.

Treten mehrere oder schwerwiegendere Fehler innerhalb des Stabilen Speichers auf, und sind die verschiedenen Schutzmechanismen nicht mehr in der Lage, die Fehler zu korrigieren, so besteht die Gefahr, daß die Stabilen Objekte im Stabilen Speicher zerstört werden können. Ein Beispiel für einen schwerwiegenden Fehler wäre der Ausfall einer der beiden Objektspeicher. In solchen Fällen müssen die zugreifenden Anwenderprozesse möglichst bald darüber informiert werden. Daher werden auch solche Schadensinformationen im Ergebnisregister vermerkt.

Die Menge der möglichen Aufträge besteht zum Teil aus solchen, die verwaltungstechnischer Art sind (z.B. CREATE, DELETE), und solchen, mit denen Stabile Objekte verändert, gelesen oder überprüft (WRITE, READ, VERIFY) werden. Es sind dies:

- INFO : Der Anwender erhält allgemeine Informationen über den Stabilen Speicher (z.B. Konfigurationsdaten).
- ACCESS : Der Anwender möchte in Zukunft an der Nutzung des Stabilen Speichers teilnehmen. Mit diesem Auftrag meldet er sich beim Stabilen Speicher an.
- RELEASE : Der Anwender gibt die Nutzung des Stabilen Speichers wieder auf. Alle Stabilen Objekte, die er noch im Stabilen Speicher stehen hat, werden gelöscht. Der Speicherplatz kann wieder anderweitig genutzt werden.
- CREATE : Der Anwender möchte ein neues Stabiles Objekt im Stabilen Speicher erzeugen. Dazu muß Speicherplatz belegt werden.

- DELETE : Der Anwender möchte ein Stabiles Objekt löschen. Damit wird im Stabilen Speicher wieder Platz frei.
- WRITE : Der Anwender möchte auf ein schon bestehendes Stabiles Objekt schreiben, d.h. einen Update ausführen. Dabei darf die Größe des Objekts nicht verändert werden.
- READ : Der Anwender möchte ein bestehendes Stabiles Objekt lesen.
- VERIFY : Ein Stabiles Objekt soll überprüft werden. Diese Operation entspricht im wesentlichen dem Cleanup von Lampson. Da hier aber jedes Wort mit Parity ausgestattet ist, gibt es beim Auftreten von Bitfehlern deutlich mehr Korrekturmöglichkeiten als bei Datenblöcken, die nur durch eine Checksumme geschützt sind. Dieser Auftrag wird vom Stabilen Speicher automatisch ausgeführt, wenn nichts weiteres zu tun ist. Diese Aktion kann aber auch explizit vom Anwender angestoßen werden.
- WRITE_BI : Jedem Stabilen Objekt kann der Anwender eine Kurzinformation mitgeben. Diese darf (im hier implementierten Fall) maximal 20 Bit umfassen. Mit dieser Operation kann diese Benutzerinformation beschrieben werden.
- READ_BI : Mit dieser Operation kann die Benutzerinformation gelesen werden.

Zum Teil gibt es auch Kombinationen aus diesen Aufträgen, z.B. <CREATE and WRITE> oder <WRITE and WRITE_BI>. Ein solcher Auftrag setzt sich aus der Hintereinanderausführung der beiden Teile zusammen. Folgende Aufträge können ausgeführt werden:

INFO,
ACCESS,
RELEASE,
CREATE,
CREATE and WRITE,
CREATE and WRITE_BI,
CREATE and WRITE and WRITE_BI,
DELETE,
WRITE,
WRITE and WRITE_BI,
READ,
READ and READ_BI,
VERIFY,
WRITE_BI,
READ_BI.

Die Aufträge INFO und ACCESS unterscheiden sich allerdings in der Benutzung von den anderen. Bei ihnen wird nur von bestimmten Adressen gelesen, während bei den übrigen Aufträgen bestimmte Angaben zu den betroffenen Stabilen Objekten an den Pufferbereich übergeben werden. Genauer wird dies in den Abschnitten 6.4.2 (Ablauf ohne Datentransfer) und 6.4.4 (VIP) behandelt.

Ein Lese- oder Schreib-Auftrag an den Stabilen Speicher betrifft immer das ganze Stabile Objekt. Auch wenn nur Teile eines Stabilen Objekts benötigt werden, besteht nur die Möglichkeit, das gesamte Stabile Objekt zu transferieren (in diesem Fall: lesend) und anschließend außerhalb des Stabilen Speichers die gewünschten Teile des Objekts zu entnehmen und zu verwenden. Auch beim Schreiben muß ein ganzes Objekt transferiert werden. Dies ist für die Berechnung und Überprüfung der Checksumme notwendig. Die Arbeit mit dem Stabilen Speicher wird dadurch kaum eingeschränkt, da man bei der vorgesehenen Verwendung (Aufnahme von Sicherungsdaten) leicht mit dem Transfer von kompletten Objekten auskommen kann.

In bestimmten Situationen (z.B. bei Recovery nach dem Ausfall eines Prozessors) kann es aber wünschenswert sein, eine Kurzinformation über das Stabile Objekt erhalten zu können ohne das ganze Objekt lesen zu müssen. Dazu dient die Benutzerinformation. Sie wird separat vom Stabilen Objekt gespeichert und wird auch bei einer Checksummenbildung über das Stabile Objekt nicht berücksichtigt. So kann sie aus dem Stabilen Speicher entnommen werden ohne das gesamte Objekt transportieren zu müssen. Dies geht deutlich schneller als die oben beschriebene Möglichkeit. Entsprechend einfach kann diese Information auch ohne Zugriff auf das abgespeicherte Stabile Objekt verändert werden.

In den folgenden Ausführungen werden die Aufträge danach unterschieden, ob sie mit einem Datentransport verbunden sind oder nicht. Diejenigen ohne Datentransport sind:

RELEASE,
CREATE,
CREATE and WRITE_BI,
DELETE, VERIFY,
WRITE_BI,
READ_BI.

Der Transfer der Benutzerinformation wird nicht als Auftrag mit Datentransfer betrachtet, da hierfür kein Pufferplatz zur Verfügung gestellt werden muß. Dieser Datenaustausch erfolgt über ein bestimmtes Register, das in einer Kommunikationspage vorgesehen ist.

Die Aufträge mit Datentransport sind alle, die den Anteil “WRITE” oder “READ” enthalten:

CREATE and WRITE,
CREATE and WRITE and WRITE_BI,
WRITE,
WRITE and WRITE_BI,

READ,
READ and READ_BI.

Wird in den folgenden Ausführungen von “WRITE-Aufträgen” oder von “READ-Aufträgen” gesprochen, so gilt das dort beschriebene dann auch für die entsprechenden anderen Aufträge mit WRITE- oder READ-Anteil.

6.4 Pufferbereich

Der Pufferbereich ist die einzige Einheit des Stabilen Speichers, auf die von zwei Seiten aus zugegriffen wird. Zum einen sind es die Prozessoren aus dem Host-System (Motorola-System), die den Stabilen Speicher nutzen wollen. Auf der anderen Seite ist es die zentrale Steuerung des Stabilen Speichers, die zur Bearbeitung des Auftrags vor allem den Datenaustausch durchführt. Dabei werden Datenblöcke zwischen dem Lagerbereich und dem Pufferbereich transportiert. Um einen möglichst reibungslosen Ablauf gewährleisten zu können, bei dem sich die beiden Zugreifer kaum behindern, ist der Pufferbereich in drei Unterbereiche aufgeteilt:

- Zugriffskontrolle,
- Pufferspeicher,
- Voter.

Der zentralen Steuerung sind der Voter und der Pufferspeicher zugewandt, dem Anwendersystem der Bereich Zugriffskontrolle (siehe Abb. 19). Der Grund für diese Anordnung liegt darin, daß die von der zentralen Steuerung übertragenen Adressen und Daten immer gevotet werden müssen und dann vor allem Zugriffe auf den Pufferspeicher erfolgen. Dagegen müssen die Angaben, die vom Anwendersystem kommen, zunächst in der Zugriffskontrolle auf ihre Berechtigung überprüft werden.

Bei der Ausführung der READ- und WRITE-Aufträge eines Anwenders müssen in der Datenübertragungsphase auch Zugriffe auf den Pufferspeicher durchgeführt werden. Es zeigt sich aber, daß auch in diesem Fall die Aktivitäten in der Zugriffskontrolle einen deutlich höheren zeitlichen Anteil haben als der reine Zugriff auf den Pufferspeicher (siehe Abschnitt 7.1.6, Zugriffskonflikte).

Dem gegenüber muß auch die zentrale Steuerung des Stabilen Speichers gelegentlich auf den Bereich der Zugriffskontrolle zugreifen, da sich hier das Auftrags-FIFO und die Angaben zu den erteilten Aufträgen in einem Speicher (VIP, KP) befinden. Die Zahl dieser Zugriffe ist im Verhältnis zu der Zahl der Zugriffe auf den Pufferspeicher sehr gering. Daher behindern sich die beiden Zugreifer nur sehr selten.

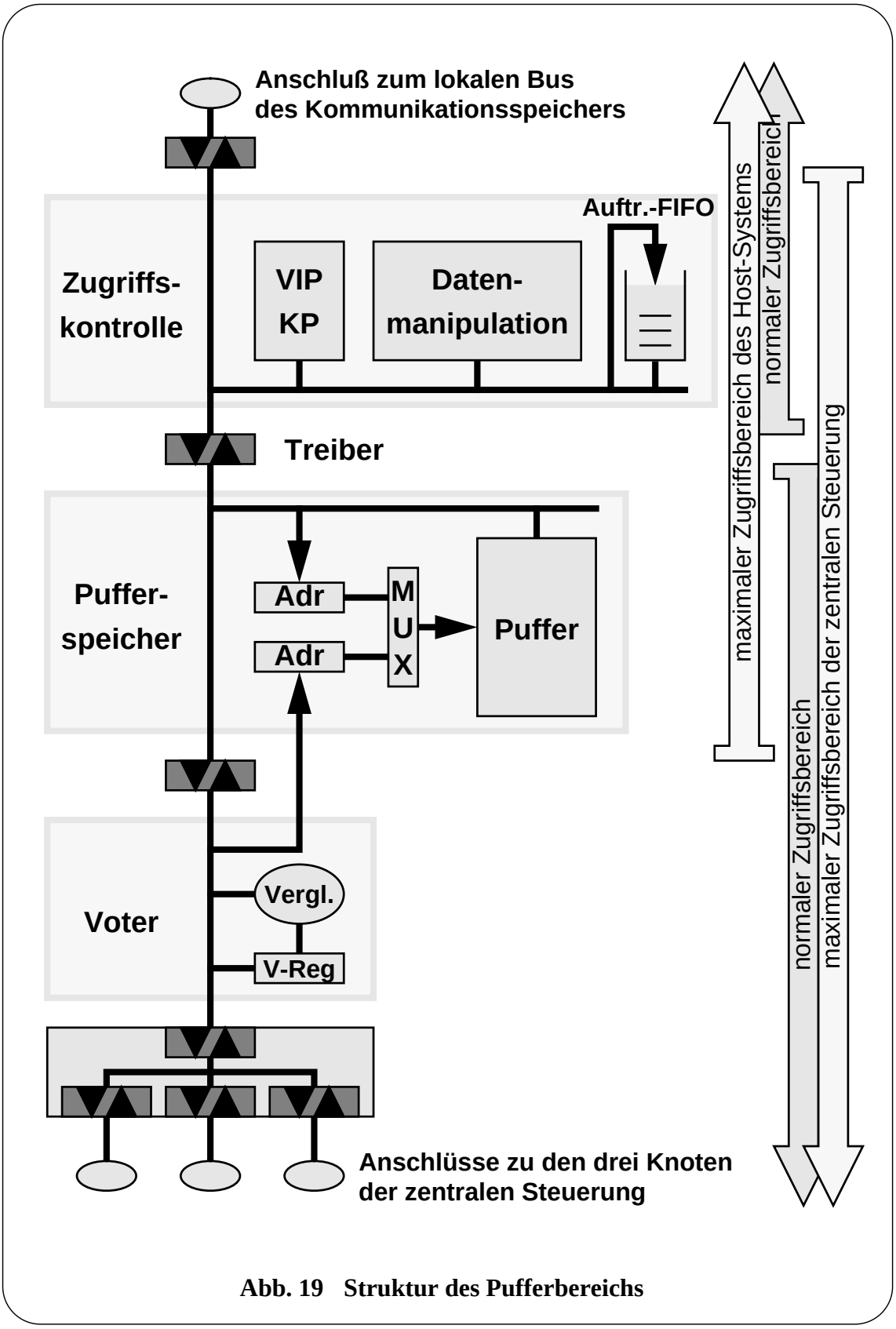


Abb. 19 Struktur des Pufferbereichs

In Abhängigkeit von der vom Anwender angegebenen Adresse müssen im Pufferbereich unterschiedliche, und in den meisten Fällen mehrere Einzelaktionen hintereinander ausgelöst werden. Diese Abläufe, die bei jedem einzelnen Anwenderzugriff durchgeführt werden müssen, werden von einer “Puffersteuerung” ausgelöst. Diese ist nicht mit Hilfe eines Prozessors sondern mit programmierbaren Logik-Bausteinen realisiert worden [MACH94] [LAT91]. Auf Grund der vielen unterschiedlich zu steuernden Aktionen mußten mehrere Logik-Bausteine eingesetzt werden, die einen großen Teil einer Platine belegen. Ansonsten werden Standardbausteine (ICs) eingesetzt [TIT89] [FAST85]. Die “Programme” für die Logik-Bausteine wurden mit Hilfe eines Entwicklungstools erstellt [LOG93] und programmiert. Dies alles konnte im Labor des IMMD 3 an der Universität Erlangen-Nürnberg durchgeführt werden. Die Platinenentwicklung und ihre Fertigung mußten dagegen außerhalb dieses Instituts erfolgen.

6.4.1 Schutzmechanismen in der Zugriffskontrolle

Aus der Sicht der Prozessoren stellt sich der Stabile Speicher als ein Adreßbereich dar, der in einem Teil eines 16 MByte-Bereichs liegt, der für einen Kommunikationsspeicher reserviert ist.

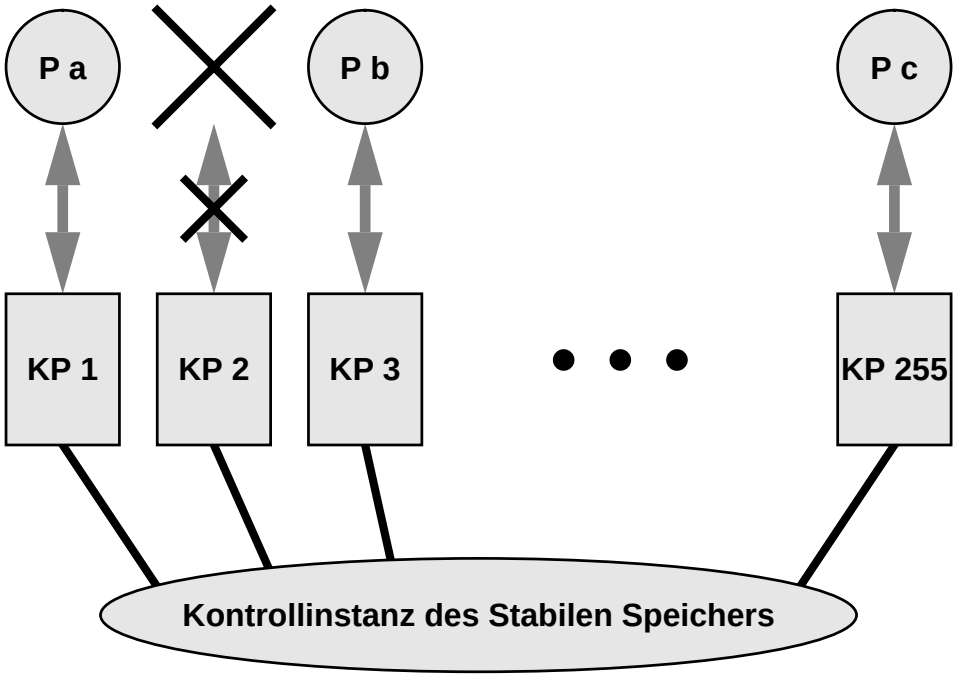
Andersherum stellt sich für den Stabilen Speicher die Welt um ihn herum etwas unübersichtlich dar. Dem Stabilen Speicher ist es nicht möglich, die einzelnen Prozessoren direkt voneinander zu unterscheiden. Der Grund liegt darin, daß die Prozessoren (bzw. die CMMU) für den Zugriff auf einen Kommunikationsspeicher eine bestimmte physikalische Adresse angeben müssen, die dieses Ziel beschreibt. Da nun alle Prozessoren, die denselben Stabilen Speicher ansprechen wollen, dasselbe Ziel adressieren, geben sie auch den gleichen Adreßbereich (4 MByte-Bereich) an. Es wird auch kein Identifikationsmerkmal automatisch mit angegeben, das der Stabile Speicher zur Unterscheidung der Prozessoren und deren Authentisierung nutzen könnte. So muß sich der Stabile Speicher anderer Mittel bedienen, um die Prozessoren auseinanderzuhalten.

6.4.1.1 Kommunikationspages

Zur Kommunikation der Prozessoren mit dem Stabilen Speicher ist nur ein kleiner Adreßbereich nötig. Einige wenige Register (höchstens 8) genügen, über die der Informationsaustausch zwischen Anwenderprozeß und Stabilem Speicher durchgeführt werden kann. Dazu werden sogenannte “Kommunikationspages” (KP) eingerichtet. Ziel ist, daß der Stabile Speicher je einem zugreifenden Prozeß eine Kommunikationspage zuordnet.

An dieser Stelle soll der Begriff des “*Auftraggebers*” eingeführt werden, um von den Begriffen Prozessor und Prozeß zu abstrahieren. Denn der Stabile Speicher ordnet die Kommunikationspages immer Auftraggebern zu, ohne wissen zu können, ob sich dahinter unterschiedliche Prozessoren oder unterschiedliche Prozesse befinden. Daher wird im Verlauf dieser Arbeit fast ausschließlich von Auftraggebern gesprochen. Nur bei Aspekten des Betriebssystems wird der Begriff “Prozeß” und bei Aspekten der Hardware der Begriff “Prozessor” wieder verwendet.

Der Stabile Speicher interpretiert die Zugriffe, die ihn über die gleiche Kommunikationspage erreichen, als solche, die zum gleichen Auftraggeber gehören. Zugriffe, die ihn über verschiedene Kommunikationspages erreichen, ordnet er unterschiedlichen Auftraggebern zu. In Abb. 20 ist die Sichtweise des Stabilen Speichers auf seine Auftraggeber dargestellt.



P a ... P c : Auftraggeber (Prozessoren / Anwender-Prozesse);
KP 1 ... KP 255 : Kommunikationspages (KP);
(KP 1, KP 3, KP 255 sind zugeordnet, KP 2 ist nicht zugeordnet)

Abb. 20 Zuordnung der Kommunikationspages zu den Auftraggebern aus Sicht des Stabilen Speichers

Innerhalb des Stabilen Speichers sind 256 Kommunikationspages von einer Größe von je 64 Worten eingerichtet. Die Zahl 64 ist zwar deutlich größer als die oben angegebenen maximal 8 benötigten Worte pro Page, der restliche Platz kann aber für die Speicherung interner Daten genutzt werden, welche die Kommunikationspages betreffen.

Die 256 Kommunikationspages müssen von den Auftraggebern adressiert werden können. Dazu werden sie als gleich große Adreßbereiche innerhalb des zur Verfügung stehenden 4 MByte-Bereichs aufgeteilt. Daher kann eine Kommunikationspage (gegenüber den Auftraggebern) maximal einen Adreßraum von 16 KByte belegen. Das ist die Größenordnung von Pages aus der Speicherverwaltung des Host-Systems. Das legt den Gedanken nahe, eine Kommunikationspage aus Sicht der Auftraggeber genau in einer Page des Host-Systems unterzubringen.

Um hier eine Anpassung an verschiedene Systeme zu ermöglichen, wurde die Möglichkeit eingerichtet, eine von vier möglichen Pagegrößen zu nutzen. Mit zwei Schaltern lassen sich die Pagegrößen einstellen, die das Host-System (hier das Motorola-System) benutzt:

- 2 KByte
- 4 KByte
- 8 KByte
- 16 KByte

Je nach eingestellter Größe werden unterschiedliche Gruppen von 8 Bit aus der übertragenen physikalischen Adresse für die Kennzeichnung einer Kommunikationspage verwendet (siehe Abb. 21). Aus dem Anwenderprogramm heraus kann die eingestellte Größe abgefragt werden (siehe auch Abschnitt 6.4.4, VIP). Innerhalb einer Kommunikationspage wird bei allen möglichen Pagegrößen nur zwischen 256 Bytes (= 64 Worte) unterschieden, wobei die niedrigsten 8 Adreßbits betrachtet werden. Die weiteren (höheren) Adreßbits bleiben unberücksichtigt.

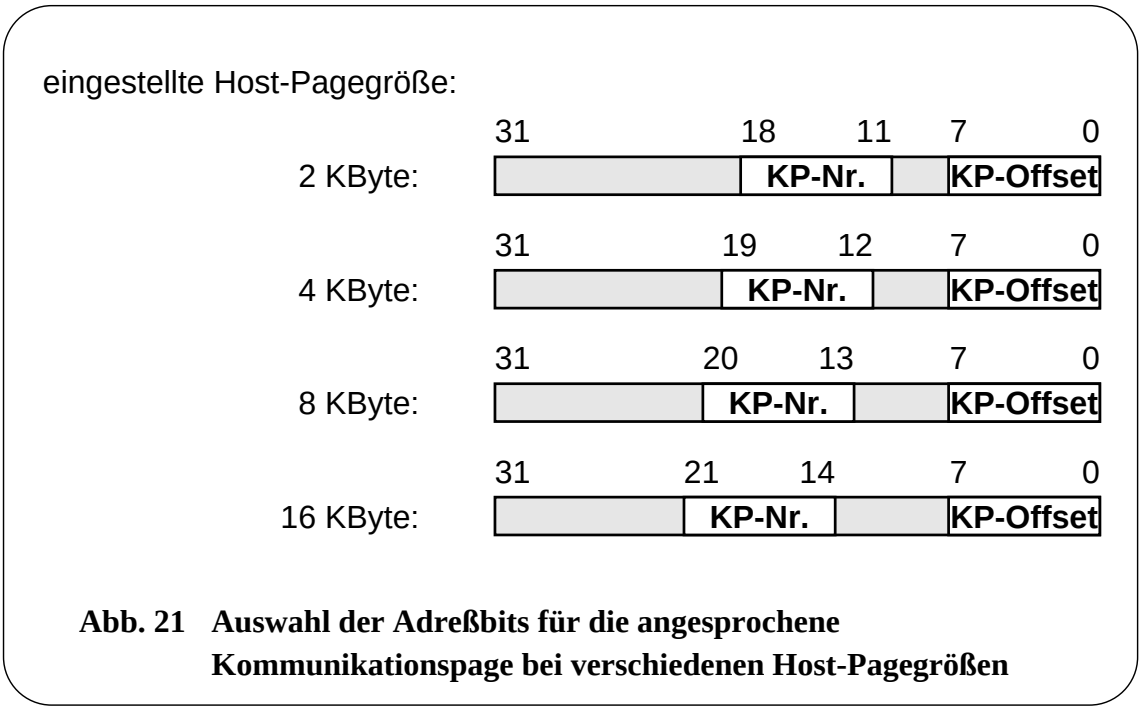


Abb. 21 Auswahl der Adreßbits für die angesprochene Kommunikationspage bei verschiedenen Host-Pagegrößen

Die Verwendung der im Motorola-System benutzten Pagegröße hat noch eine praktische Nebenwirkung. Wird nämlich im Motorola-System dafür gesorgt, daß das Betriebssystem für den Zugriff auf den Stabilen Speicher nur eine einzige Page pro Auftraggeber bereitstellt und die anderen Pages im verwendeten 4 MByte-Bereich sperrt, so kann bereits an der Schnittstelle des Prozessors an die Außenwelt (CMMU) ein Schutzmechanismus wirken, der Zugriffe auf die anderen Pages von vornherein verhindert. Die Nichtbenutzung des größten Teils dieses 4 MByte-Bereichs bedeutet keine Einschränkung für den Auftraggeber, denn dieser gehört zu dem Bereich, den er sowieso für den Stabilen Speicher bereitstellen muß.

Alle Auftraggeber müssen einen entsprechenden 4 MByte-Bereich freihalten. So ist es dem Stabilen Speicher möglich, einen solchen Bereich nach eigenen Vorstellungen aufzuteilen. Dies ermöglicht es ihm, sich ein eigenes Bild von den außen befindlichen Auftraggebern zu machen. Diese Abbildung erfolgt über die Kommunikationspages 1 - 255.

Über eine besondere Page, die Vermittlungs- und Informations-Page (VIP) müssen sich die zukünftigen Auftraggeber zunächst einmal beim Stabilen Speicher anmelden. Dabei bekommen sie eine bisher freie Kommunikationspage und ein Paßwort zugewiesen. Die VIP liegt als Kommunikationspage 0 auch im bereits angesprochenen 4 MByte-Bereich. Alle weiteren Kommunikationen mit dem Stabilen Speicher zur Auftragsbearbeitung müssen die Auftraggeber nun über die zugewiesene Kommunikationspage durchführen. Diese Zuordnung bleibt so lange erhalten, bis der Auftraggeber dem Stabilen Speicher (über die zugewiesene Kommunikationspage) mitteilt, daß er die Zusammenarbeit beenden möchte (RELEASE-Auftrag). Dann wird diese Kommunikationspage wieder freigegeben und kann einem anderen Auftraggeber zur Verfügung gestellt werden. Auch die möglicherweise noch vorhandenen Stabilen Objekte dieses Auftraggebers werden dabei wieder aus dem Stabilen Speicher entfernt. In Abschnitt 6.4.4 befinden sich weitere Einzelheiten über die VIP.

6.4.1.2 Schutzeinrichtungen in der Kommunikationspage

Zur Auftragsvergabe muß jeder Auftraggeber die dazu benötigten Angaben in die Auftrags- und Parameterregister eintragen. Dies erfolgt durch einfache Schreibzugriffe auf die Kommunikationspage. Dabei muß sichergestellt sein, daß die mitgeteilten Angaben nur vom zugeordneten Auftraggeber in die Kommunikationspage eingetragen werden. Da die Auftraggeber dieselben Verbindungswege benutzen, besteht hier die Gefahr, daß ein Auftraggeber den Stabilen Speicher über eine falsche Kommunikationspage erreicht. Das kann passieren, wenn kein Pageschutzmechanismus verwendet wird, wenn dieser falsch programmiert worden ist, oder wenn auf der Übertragungsstrecke ein falscher Weg geschaltet worden ist. Dies muß abgewendet werden können. Dieser Schutz ist deshalb sehr wichtig, weil nach einer erfolgreichen Auftragsvergabe die zentrale Steuerung des Stabilen Speichers völlig selbständig die Ausführung des Auftrags übernimmt. Der Auftraggeber hat dann keinen Einfluß mehr darauf.

Der Schutz bei der Auftragsvergabe wird durch die Vergabe eines Paßwortes erreicht. Dieses muß bei allen Schreiboperationen zur Auftragsvergabe angegeben werden. Ein Zugriff (eines anderen Auftraggebers) mit einem falschen Paßwort kann daher wirkungslos gemacht werden, bevor ein Eintrag in den Auftrags- und Parameterregistern durchgeführt wird.

Die einzelnen Angaben, die zur Bearbeitung eines Auftrags erforderlich sind, sind hier zusammengestellt:

- die Angabe über den Auftrag (z.B. WRITE),
  (Die Anzahl der möglichen Aufträge ist sehr begrenzt. Zur Unterscheidung müßten etwa 4-5 Bits ausreichen. Um aber flexibler zu bleiben, wurden 8 Bit gewählt.)

- die Angabe des zu bearbeitenden Stabilen Objekts (= Objekt-Nummer),
  (In der aktuelle Implementierung sind maximal etwa 4000 Stabile Objekte für einen Stabilen Speicher vorgesehen. Daher sind 12 Bit dafür ausreichend. Es gibt keine prinzipiellen Schwierigkeiten, die Maximalzahl auf 65000 zu erhöhen, wobei dann 16 Bit benötigt würden. Allerdings müßten dann einige Implementierungsdetails bei der verwaltungstechnischen Bearbeitung der Stabilen Objekte innerhalb des Stabilen Speichers verändert werden.)
- die Angabe über die Länge des Stabilen Objekts (= Objekt-Länge),
  (Es werden Objekte von maximal 1 Megawort = 4 MByte zugelassen. Für die Beschreibung der Länge sind somit 20 Bit ausreichend.)
- und eine Kennzeichnung, die der Benutzer völlig frei für ein Stabiles Objekt vergeben kann (20 Bit Benutzerinformation).

Jede dieser Angaben ist höchstens 20 Bit breit. Die restlichen 12 Bit eines 32-bit-Wortes können daher für ein Paßwort verwendet werden. Somit gibt es über 4000 Möglichkeiten für das Paßwort.

Welche der Angaben benötigt werden, ist bei den Aufträgen verschieden. Meistens wird die Nummer des Stabilen Objekts und dessen Länge erforderlich sein. Soll die Benutzerinformation geschrieben werden (WRITE_BI), so ist auch diese anzugeben. Beim Lesen der Benutzerinformation (READ_BI) wird diese vom Stabilen Speicher im entsprechenden Parameterregister ausgegeben. Beim CREATE-Auftrag wird eine vom Stabilen Speicher ermittelte Objektnummer in dem Parameterregister ausgegeben, in das bei den anderen Aufträgen die Objektnummer einzutragen ist.

Die Datenstrukturen für das Auftragsregister und die drei Parameterregister sind in Abb. 22 dargestellt. Der schattierte Bereich gibt die Stellen in dem zu schreibendem Wort an, an denen eine beliebige Bitkombination stehen kann. Die dort gemachten Angaben werden weder überprüft noch für eine Auswertung herangezogen.

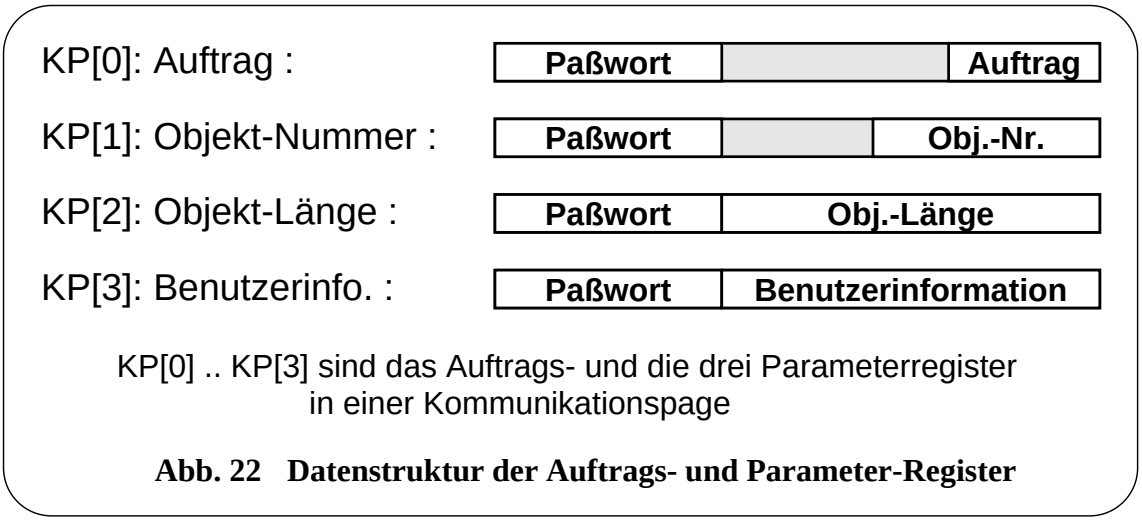


Abb. 22 Datenstruktur der Auftrags- und Parameter-Register

Aus der Zuordnung zwischen Kommunikationspage-Nummer und dem Paßwort läßt sich keine Gesetzmäßigkeit ableiten, denn diese erfolgt quasi zufällig. Diese Kombination kann und wird üblicherweise auch bei jeder Neuzuteilung anders sein. Selbst nach einer Freigabe und einer anschließenden Wiedervergabe dieser Kommunikationspage wird das neue Paßwort mit sehr hoher Wahrscheinlichkeit (> 99%) anders sein als vorher (siehe auch Abschnitt 6.4.4, VIP).

Mit der Angabe und der Überprüfung des Paßworts bei jedem Schreibzugriff, der das Ablegen eines Auftrags betrifft, kann mit sehr großer Wahrscheinlichkeit (> 99%) sichergestellt werden, daß nur der berechtigte Auftraggeber in der Lage ist, einen vollständigen Auftrag in seiner Kommunikationspage abzulegen. Dabei wird angenommen, daß ein defekter Auftraggeber einen zufälligen Wert für das Paßwort angibt.

Dieser Schutzmechanismus ist somit in der Lage, bei “versehentlichen” Zugriffen auf eine fremde Kommunikationspage von vornherein eine unbeabsichtigte Änderung fremder Auftrags-Angaben zu verhindern. Solch versehentliche Zugriffe könnten beispielsweise dann auftreten, wenn ein falscher Übertragungsweg geschaltet wurde, wenn ein oder mehr veränderte Bits nicht erkannt wurden, oder wenn die Anwendersoftware bzw. das Betriebssystem bei der Umrechnung der übergebenen Kommunikationspage-Nummer in einen Eintrag in der Pageverwaltung einen Fehler gemacht hat. Zur letzten Fehlermöglichkeit wird auf den Abschnitt 6.4.4 verwiesen, in der die VIP beschrieben wird.

Diese Schutzvorkehrung kann allerdings nicht verhindern, daß ein Auftraggeber durch Probieren das Paßwort einer fremden Kommunikationspage herausfindet. Dies würde aber ein absichtliches Suchen nach einem fremden Paßwort sein. Der Schutz davor ist aber auch nicht seine Aufgabe.

Die Vorgehensweise des Stabilen Speicher zum Schutz der Kommunikationspages vor unberechtigten Zugriffen erinnert an den Ablauf, der bei Geldautomaten durchgeführt wird. Auch dort muß sich der Bankkunde mit Hilfe seiner ec-Karte und einem Paßwort als Eigentümer eines Kontos (Kommunikationspage) ausweisen. Das Ausspionieren eines fremden Paßwortes nach der oben genannten Methode kann bei Geldautomaten kaum vorgenommen werden, denn nach einigen Versuchen mit falschen Paßwörtern behält der Automat die ec-Karte. Damit gibt es für den (falschen) Kunden keine Möglichkeit mehr, mit Hilfe der ec-Karte das (fremde) Konto zu benutzen. Neue Versuche können erst dann durchgeführt werden, wenn eine aufwendige Prozedur erfolgreich durchlaufen worden ist, in der die Berechtigung des Kunden anhand von Ausweisen überprüft worden ist.

Eine solche Vorgehensweise wäre beim Stabilen Speicher nicht sinnvoll. Denn hier benutzt der Auftraggeber keine physikalisch vorhandene Karte sondern nur das Wissen über seine Zuordnung zu einer Kommunikationspage (und sein Paßwort). Das bedeutet, daß jeder Auftraggeber eine “angebliche” Zuordnung zu einer Kommunikationspage angeben kann, unabhängig von der Existenz einer nur einmal vorhandenen Berechtigungskarte. Würde man nun eine Kommunikationspage sperren, nachdem mehrmals von einem fremden Auftraggeber versucht worden ist, mit einem falschen Paßwort auf die Kommunikationspage zuzugreifen, so sähe der korrekt

arbeitende Auftraggeber plötzlich und ohne erkennbaren Grund seine Kommunikationspage gesperrt. Dieser Einfluß eines fehlerhaft arbeitenden Auftraggebers auf einen korrekt arbeitenden Auftraggeber sollte vermieden werden. Daher wird hier eine Kommunikationspage nach fehlerhaften Zugriffen nicht gesperrt.

Stattdessen werden bei der Übernahme des Auftrags von der zentralen Steuerung weitere Schutzmechanismen eingesetzt, die einen Auftrag mit nicht zusammenpassenden Angaben erkennen. Dieser Auftrag wird dann nicht ausgeführt. Der Auftraggeber wird durch Einträge im Ergebnisregister darüber informiert.

Weiterhin erlaubt die Puffersteuerung des Stabilen Speichers das Schreiben eines neuen Auftrags und der Parameter nur dann, wenn die angesprochene Kommunikationspage auch belegt ist und derzeit kein Auftrag zur Bearbeitung im Auftrags-FIFO eingetragen, von der zentralen Steuerung aber noch nicht beendet worden ist. Sollte der Auftraggeber dennoch Zugriffe auf die Auftrags- oder Parameterregister machen, so wird dies schon von der Puffersteuerung erkannt und abgewiesen. Diese Schutzmaßnahme sorgt dafür, daß von den Auftraggebern nur in kontrollierter Weise auf Ressourcen des Pufferbereichs zugegriffen werden kann.

6.4.2 Ablauf der Aufträge ohne Datentransfer im Stabilen Speicher

Hier wird nun der Ablauf der Aufträge dargestellt, die keinen Datentransfer benötigen. Dabei wird auch auf die Stellen aufmerksam gemacht, an denen bei Aufträge mit Datentransfer zusätzliche Abläufe durchgeführt werden müssen. Die Details des Datentransfers sind im nächsten Abschnitt (6.4.3) beschrieben.

Mit dem erfolgreichen Schreiben des Auftragsregisters werden Auftrags- und Parameterregister gegen weitere Zugriffe des Auftraggebers gesperrt, damit sich dieser Datensatz nicht mehr verändern kann. Erst nach der Beendigung des Auftrags durch die zentrale Steuerung wird die Sperre wieder aufgehoben.

Der Zeitpunkt der Übergabe eines Auftrags an die zentrale Steuerung ist das Schreiben des Auftragsregisters durch den Auftraggeber. Dabei erfolgt ein Eintrag der Kommunikationspage-Nummer in das Auftrags-FIFO. Für einen korrekten Ablauf der Auftragsübergabe ist es daher notwendig, daß der Eintrag ins Auftragsregister erst *nach* dem Schreiben der Parameter in die Parameterregister erfolgt.

Bei WRITE-Aufträgen würde an dieser Stelle das Schreiben des Datenblocks folgen (siehe Abschnitt 6.4.3). Auch die interne Übergabe dieses Auftrags würde in diesem Fall erst *nach* dem Schreiben des letzten Wortes des Datenblocks ausgeführt werden.

Bei der Übergabe der Kommunikationspage-Nummer ins Auftrags-FIFO kann der Auftraggeber allerdings nicht erkennen, ob und wieviele Aufträge im FIFO bereits vorliegen. Daher kann er nicht abschätzen, wie lange es bis zur Beendigung des Auftrags dauern wird.

Prinzipiell gibt es zwei Möglichkeiten für die Feststellung, wann der Auftrag beendet worden ist. Zum einen könnte der Stabile Speicher den Auftraggeber mit einem Interrupt von der Beendigung seines Auftrags benachrichtigen. Und als zweite Möglichkeit könnte der Auftraggeber immer wieder beim Stabilen Speicher nachfragen, ob sein Auftrag schon fertig ist.

Die erste Alternative sieht eleganter aus als die zweite, denn sie verhindert, daß der Auftraggeber ein “busy wait” durchführen muß. Denn dies könnte bei intensiver Nutzung dazu führen, daß das Verbindungsnetzwerk zwischen den Prozessoren und den Kommunikationsspeichern einer übermäßigen Belastung ausgesetzt wäre.

Bei der anderen Alternative müßte eine Interruptverbindung zwischen dem Stabilen Speicher und den Prozessoren vorhanden sein. Dies wurde hier nicht vorgesehen, da die Einführung einer solchen Verbindung dazu führen würde, daß ein zusätzliches umfangreiches Verbindungssystem zu dem bereits bestehenden eingerichtet werden müßte. Denn bei der bestehenden Topologie kann ein Kommunikationsspeicher, und damit auch der dort platzierte Stabile Speicher, von 8 Prozessorknoten erreicht werden. Doch eine zusätzliche Verkabelung sollte ja vermieden werden. Somit hat der Stabile Speicher keinen “direkten Draht” zu den Auftraggebern.

Dafür bleibt man aber flexibler, was die mögliche Nutzung des Stabilen Speichers durch die Prozessoren betrifft. Denn somit gilt für die Nutzung des Stabilen Speichers nur, daß ein Prozessor die Möglichkeit hat, einen normalen Speicherzugriff auf ihn durchführen zu können. Wieviele Prozessoren dann Zugriff auf den Stabilen Speicher haben, ist somit nur von der (bereits bestehenden) Verbindungstopologie des Multiprozessorsystems abhängig und nicht von der Struktur des Stabilen Speichers. Folglich ist die zunächst ungünstiger erscheinende Version mit dem “Nachfragen” verwendet worden.

Dazu existiert in jeder Kommunikationspage das Ergebnisregister (KP[4]). Auf dieses sind nur Lesezugriffe erlaubt. Sein Inhalt gibt an, wie es um den Fortschritt des Auftrags bestellt ist. Mögliche Werte für das Ergebnisregister sind:

- IDLE : der Auftrag wartet noch in der Warteschlange auf seine Bearbeitung.
- BUSY : der Auftrag wird gerade bearbeitet.
- ERG_OK : der Auftrag ist fertig und wurde erfolgreich zu Ende geführt;
  (im Fall einer READ-Operation kann nun der Datenblock aus dem Puffer gelesen werden.)
  dies ist der Initial-Zustand nach der Zuteilung einer Kommunikationspage.
- ERG_OLD : der Auftrag ist fertig, konnte aber nicht ausgeführt werden;
  die alte Situation wurde wiederhergestellt.
- FAIL : das Stabile Objekt wurde irreparabel korrumpiert;
  für den Fall FAIL gibt es einige unterschiedliche Werte, die die Ursache für die Fehlersituation beschreiben.
  (Diese Situation sollte im Stabilen Speicher eigentlich nicht vorkommen.

In diesem Fall hat aber eine Anhäufung von Fehlersituationen dazu geführt, daß der Stabile Speicher das Stabile Objekt nicht mehr korrekt bearbeiten konnte.)

Für diese Angaben sind 20 Bit im Ergebnisregister reserviert (Bits 19 .. 0).

Die folgenden Werte werden zusätzlich zu den bisher genannten Angaben im Ergebnisregister eingetragen. Sie betreffen das Ergebnis bei der Suche nach einem Platz im Pufferspeicher zur Übertragung eines Objekts.

- P_FOUND : es wurde ein Pufferabschnitt mit passender Größe zur Aufnahme des zu übergebenden Datenblocks bereitgestellt.
- P_FULL : es kann derzeit (wegen Überlast) kein Puffer bereitgestellt werden; der Auftrag wird abgewiesen; er kann später wiederholt werden; dazu ist nur erforderlich, daß der Auftrag erneut ins Auftragsregister geschrieben wird; die Eintragungen in den Parameterregistern bleiben bestehen.

Für diese Angaben sind 4 Bit im Ergebnisregister reserviert (Bits 23 .. 20).

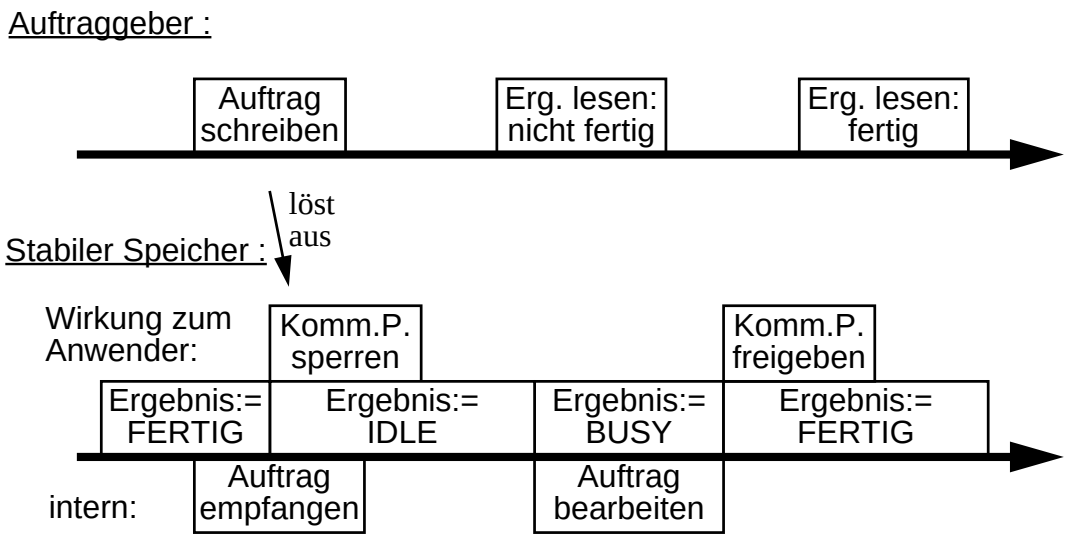
Solange im Ergebnisregister noch der Wert IDLE oder BUSY steht, ist sein Auftrag noch nicht fertig. Schreibzugriffe, die während dieser Zeit auf die Auftrags- und Parameterregister gewünscht werden, werden nicht durchgeführt. Dies wird durch eine Hardware im Pufferbereich erreicht, die bei jedem Zugriff (auf die Auftragsregister) die Berechtigung dazu kontrolliert. Erst wenn im Ergebnisregister ein Wert steht, der Aufschluß über den Erfolg der angestoßenen Operation gibt (FERTIG-Meldungen: ERG_OK, ERG_OLD oder FAIL-Informationen), ist der Auftrag vom Stabilen Speicher beendet worden. Ab jetzt akzeptiert die Kommunikationspage auch wieder einen neuen Auftrag.

Bei READ-Aufträgen würde ab dieser Stelle das Lesen des Datenblocks durch den Auftraggeber folgen (siehe Abschnitt 6.4.3). Die Freigabe der Kommunikationspage geschieht in diesem Fall erst *nach* dem Lesen des letzten Wortes des Datenblocks.

Für den Stabilen Speicher ist es nicht wichtig, ob der Auftraggeber das Ergebnis auch liest oder nicht. Er sperrt die Zugriffe auf Auftrags- und Parameterregister dieser Kommunikationspage, wenn das Auftragsregister beschrieben wird, und gibt sie wieder frei, wenn (mit Ausnahme des READ-Auftrags) die zentrale Steuerung den Auftrag fertig bearbeitet hat.

Was der Auftraggeber während der Wartezeit auf das Ende der Auftragsbearbeitung durchführt, ist dem Programmierer bzw. dem Betriebssystem überlassen. Eine Möglichkeit, die Bearbeitungszeit sinnvoll zu nutzen, wäre, in dieser Zeit die Daten für einen neuen Auftrag vorzubereiten.

In Abb. 23 ist der zeitliche Ablauf eines Auftrags ohne Datentransfer skizziert.



"Ergebnis:=FERTIG" bedeutet,
daß irgendein Ergebnis (über den letzten Auftrag) vorliegt. Ob dieser Auftrag erfolgreich bearbeitet werden konnte oder nicht, ist in diesem Zusammenhang nicht entscheidend.

Abb. 23 zeitlicher Ablauf der Auftragsbearbeitung

Neben den Statusangaben zum aktuellen Auftrag enthält das Ergebnisregister noch weitere Information über den Schadenszustand des Stabilen Speichers. Dafür sind 8 Bit vorgesehen (siehe Abb. 24). Diese Angaben sagen etwas darüber aus, ob der Stabile Speicher noch fehlerfrei arbeitet, oder ob sich ein Schaden in ihm ereignet hat. Dieser könnte dazu führen, daß der Stabile Speicher auf Dauer nicht mehr sicher arbeiten kann.

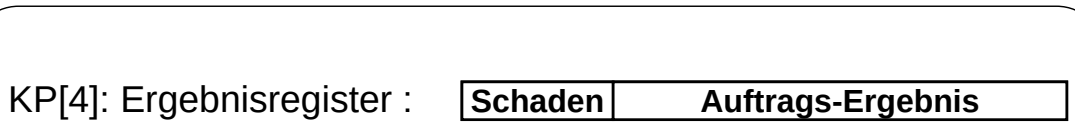


Abb. 24 Datenstruktur des Ergebnisregisters

Das Auftreten eines Schadens im Stabilen Speicher wird sicher eine große Seltenheit sein, denn dann müssen zuvor bereits mehrere Maßnahmen zur Behebung einzelner Fehler versagt haben. Entdeckt ein Auftraggeber solch eine Schadensmeldung, so sollte er geeignet darauf reagieren. Solange der Lesezugriff auf Stabile Objekte noch möglich ist, kann der Auftraggeber beispielsweise seine Objekte noch aus dem nicht mehr stabilen Speicher holen und in einem anderen Speicher unterbringen. Solange der Schaden nicht die Pufferplatine des Stabilen Speichers be-

trifft, ist dies noch möglich. Nähere Einzelheiten zu Schadenssituationen sind in Abschnitt 6.5 beschrieben.

Da von einem Schaden alle Auftraggeber dieses Stabilen Speichers betroffen sind, sollten über den Auftraggeber, der den Schaden entdeckt hat, alle anderen Knoten in seiner Nachbarschaft informiert werden, soweit dies von ihm aus möglich ist. Alternativ dazu kann ein Auftraggeber auch sein Ergebnisregister regelmäßig abfragen, auch wenn derzeit kein Zugriff auf seine Stabilen Objekte notwendig ist.

6.4.3 Datentransfer zwischen dem Auftraggeber und dem Stabilen Speicher

Bei der Kommunikation mit dem Stabilen Speicher wird meistens ein Datentransport gewünscht. Mit den Aufträgen WRITE und READ werden Daten zwischen dem Auftraggeber und dem Stabilen Speicher ausgetauscht.

Zur Übertragung müssen die Daten vom Auftraggeber als ein Block vorliegen, der aus einer Folge von Worten besteht (`array[länge] of integer`). Jedes Wort ist ein 32-bit-Wert. Die Länge ist eine Zahl zwischen 1 und 1 Mega (= 2^{20}). Der Block setzt sich zusammen aus einer Folge von Worten, die die “Netto”-Information beinhalten, und aus einem Checkwort. Für die bei der Auftragserteilung anzugebende Länge gilt die Netto-Länge. Dieser Wert liegt somit zwischen 0 und 2^{20}-1. Er läßt sich genau in der dafür vorgesehenen Stelle (20 bit) im Parameterregister KP[2] bei der Auftragsvergabe eintragen.

Für den Datenblock besteht auf Grund seiner Größe nicht die Möglichkeit, ihn auch nur annähernd komplett in dem Adreßbereich unterzubringen, der von der Kommunikationspage zur Verfügung gestellt wird. Daher wurde eine andere Möglichkeit für den Austausch eines Datenblocks geschaffen.

Im Pufferbereich gibt es einen 4 MWort großen Speicher, den Puffer. Dieser wird verwendet, um Datenblöcke aufzunehmen, die zwischen dem Auftraggeber und dem Stabilen Speicher transferiert werden müssen. Der Puffer ist dafür eingerichtet, Blöcke bis zu einer maximalen Größe von 1 MWorte (1 MWorte = 4 MByte) aufnehmen zu können.

Die Auswahl des aktuell bereitgestellten Pufferplatzes ist Teil der Auftragsvergabe, die während des Schreibzugriffes auf das Auftragsregister KP[0] durch den Auftraggeber ausgeführt wird. Da hierbei recht enge Zeitgrenzen (Timeout des Kommunikationsspeichers) eingehalten werden müssen, ist es erforderlich, daß die Auswahl des Pufferplatzes sehr einfach durchgeführt werden kann. Aus diesem Grund ist der Puffer in mehrere Abschnitte unterteilt, die feste, aber unterschiedliche Größen besitzen. Wird für einen Datentransfer Pufferplatz benötigt, so wird ein geeigneter freier Bereich ausgewählt und für diesen Auftrag zur Verfügung gestellt.

Die verfügbaren Pufferplätze und ihre Größen sind:

- 1 mal 1 M Worte (= 2^{20} Worte)
- 4 mal 256 K Worte (= 2^{20} Worte)
- 16 mal 64 K Worte (= 2^{20} Worte)
- 16 mal 16 K Worte (= 2^{18} Worte)
- 16 mal 4 K Worte (= 2^{16} Worte)

Der Auftraggeber hat keine Möglichkeit, den tatsächlich bereitgestellten Pufferplatz zu bestimmen. Er gibt nur an, wie groß der Platz für sein Objekt sein muß. Ausschließlich diese Längenangabe bestimmt, aus welchem Puffergrößenbereich der Pufferplatz reserviert wird. Beispielsweise erfolgt der Transfer eines 60 KWorte großen Objekts über einen Pufferplatz der Größe 64 KWorte.

Diese 53 Pufferplätze sollten ausreichen, um allen gleichzeitig zugreifenden Auftraggebern Puffer zur Verfügung zu stellen. Nur bei der Vergabe der beiden größten Bereiche (1 x 1 MW, 4 x 256 KW) könnte es unter Umständen Schwierigkeiten geben. Bei zu vielen gleichzeitigen Wünschen nach diesen großen Bereichen muß eine Wartezeit in Kauf genommen werden. Dagegen dürfte es bei den anderen Bereichen kaum zu Wartezeiten kommen.

Die Entscheidung, ob ein Pufferplatz vergeben werden kann oder nicht, erfolgt während des Schreibzugriffs durch den Auftraggeber auf das Auftragsregister KP[0]. Das Resultat dieser Entscheidung (P_FULL / P_FOUND) wird im Ergebnisregister eingetragen. Der Auftraggeber kann dies somit sofort nach dem Schreiben des Auftrags abfragen.

Konnte ein Pufferabschnitt bereitgestellt werden, so steht dem weiteren Auftragsablauf nicht mehr im Weg. Bei einem WRITE-Auftrag kann der Auftraggeber nun auf diesen zugreifen. Bei einem READ-Auftrag steht der Pufferplatz nun bereit, damit die zentrale Steuerung bei der Auftragsbearbeitung das angeforderte Stabile Objekt hier übergeben kann.

Für den Zugriff auf den Pufferplatz wird dem Auftraggeber nun nicht der bereitgestellte Adreßbereich mitgeteilt, sondern es wird ihm nur ein Tor zu diesem Bereich geöffnet (siehe Abb. 25). Dieses Tor ist eine einziges Register aus dem Bereich seiner Kommunikationspage, das Transferregister (KP[5]). Zugriffe auf diese Stelle werden als Schreiben (bei WRITE-Aufträgen) bzw. als Lesen (bei READ-Aufträgen) innerhalb eines FIFOs behandelt, das sich innerhalb des ausgewählten Bereichs befindet. Die aktuelle Adresse für den Puffer wird bei Auftragserteilung in der Puffersteuerung erzeugt und in der zugehörigen Kommunikationspage abgelegt. Sie ist die Summe aus der Basisadresse dieses Pufferbereichs und der Objektlänge. Bei jedem Zugriff des Auftraggebers auf das Transferregister wird diese aktuelle Adresse zum Zugriff auf den Puffer verwendet, anschließend automatisch dekrementiert und wieder in der Kommunikationspage abgelegt. Dabei wird gleichzeitig festgestellt, ob das Blockende schon erreicht worden ist oder nicht. Ist das Ende erreicht worden, wird dieser Pufferabschnitt für weitere Zugriffe ge-

sperrt. Dies ist Teil der “Datenmanipulation” aus Abb. 19 in Abschnitt 6.4 (Pufferbereich).

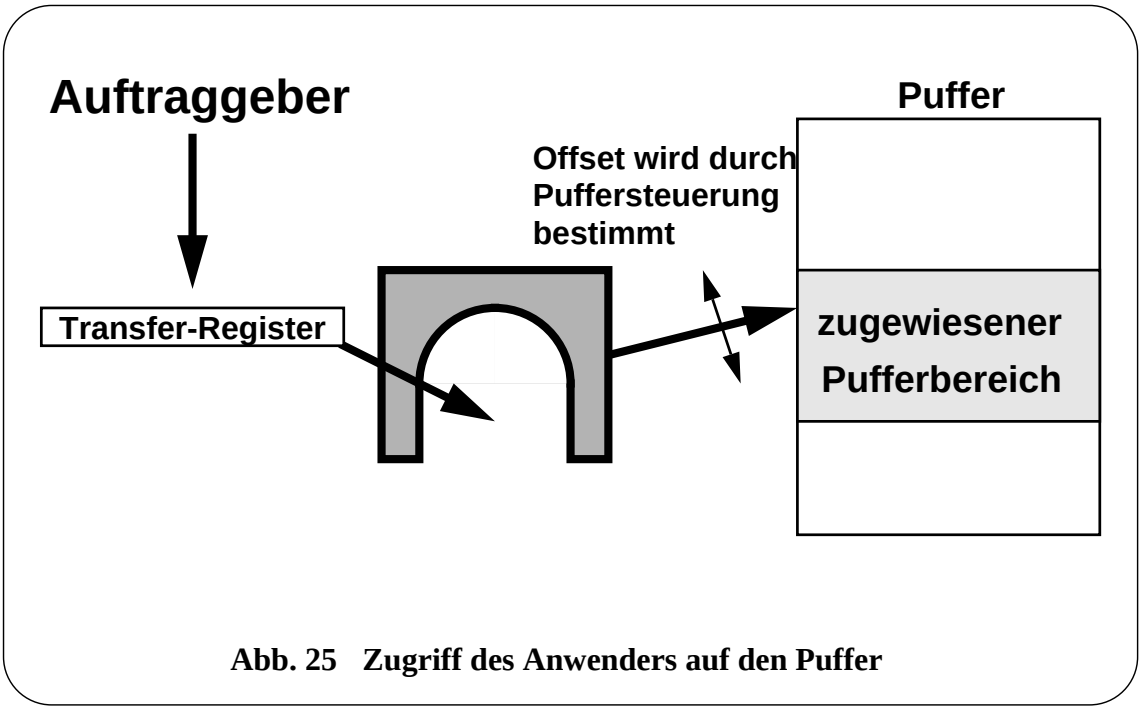


Abb. 25 Zugriff des Anwenders auf den Puffer

Die Übertragung des Datenblocks zwischen dem Prozessor des Host-Systems (Motorola-System) und dem Pufferbereich des Stabilen Speichers ist mit der Gefahr verbunden, daß ein anderer Auftraggeber Zugriffe auf diesen Datenblock macht, obwohl er dazu keine Erlaubnis hat. An dieser Stelle wird nicht die Überprüfung eines Paßwortes verwendet. Denn das hätte bedeutet, daß ein Auftraggeber beim Schreiben des Datenblocks seine üblicherweise in 32-bit-Worten vorliegenden Daten in kleinere Teile (z.B. 20- oder 16-bit-Daten) aufspalten müßte, bevor er diese zum Stabilen Speicher transportiert. Dies wäre eine unnötige Behinderung bei der Zusammenarbeit mit dem Stabilen Speicher. So wird auf die Angabe des Paßwortes verzichtet.

Um dennoch einen akzeptablen Schutz für die übertragenen Datenblöcke aufrechterhalten zu können, wird der zugehörige Pufferabschnitt nur für den Zeitabschnitt geöffnet, wie es für die Übertragung des Datenblocks zwischen Auftraggeber und Stabilem Speicher erforderlich ist. Bei WRITE-Aufträgen ist dies unmittelbar nach der Bereitstellung des Pufferabschnitts. Nach der Übertragung des letzten Wortes wird der Pufferabschnitt von der Puffersteuerung automatisch gesperrt. Weitere Zugriffe eines beliebigen Prozessors auf diesen Pufferabschnitt sind dann nicht mehr möglich. Die Freigabe erfolgt von der zentralen Steuerung, nachdem das Stabile Objekt komplett neu beschrieben oder der alte Zustand wiederhergestellt worden ist. Zu diesem Zeitpunkt ist auch der WRITE-Auftrag vollständig ausgeführt worden.

Bei READ-Aufträgen öffnet die zentrale Steuerung den Pufferabschnitt, nachdem sie die Übertragung der Kopie des angeforderten Stabilen Objekts aus dem Lagerbereich in den Pufferbereich abgeschlossen hat. Nachdem das letzte Wort vom Auftraggeber aus dem Pufferabschnitt gelesen worden ist, wird dieser Pufferabschnitt von der Puffersteuerung wieder gesperrt. Aus

Sicht des Stabilen Speichers ist der READ-Auftrag in diesem Moment beendet worden.

Zusätzlich zu dieser Zeitbegrenzung ist der Zugriff auf den Pufferabschnitt je nach Auftragsart auf Schreiben bzw. Lesen beschränkt. Ein Zugriff mit der falschen Transferrichtung durch einen (fehlerhaft) zugreifenden Auftraggeber wird von der Puffersteuerung verhindert.

Trotz dieser beschränkten Öffnung des Pufferbereichs ist es möglich, daß ein fremder (defekter) Auftraggeber während dieser Zeit Zugriffe auf diesen Pufferabschnitt durchführt. Dies kann aus den bereits früher erwähnten Gründen von der Puffersteuerung nicht erkannt werden. Zur Eindämmung dieser Gefahr erfolgt die Kontrolle des gesamten Datenblocks durch das Überprüfen der mitgelieferten Checksumme. Um die Möglichkeiten für die Bildung der Checksumme nicht von vornherein sehr zu begrenzen, wird dies nicht von der Puffersteuerung ausgeführt, da diese rein als Hardware implementiert wurde. Stattdessen erfolgt die Überprüfung während der Übertragung des Datenblocks vom Puffer in den ersten Objektspeicher. Dies wird von der zentralen Steuerung des Stabilen Speichers ausgeführt wird, die aus einem programmgesteuerten Prozessor besteht. Ungültige Datenblöcke werden dabei rechtzeitig erkannt. “Rechtzeitig” bedeutet in diesem Fall, daß bei einer nicht konformen Checksumme der alte Zustand des Stabilen Objekts durch Kopieren aus dem zweiten Objektspeicher wiederhergestellt werden kann.

Beim Lesen des Blocks durch den Auftraggeber stellt dieser den Checksummenfehler fest. Da im Stabilen Speicher dafür gesorgt wurde, daß bei einem READ-Auftrag nur eine gültige Kopie eines Stabilen Objekt im Puffer abgelegt wird, muß der Fehler beim Lesen aus dem Puffer entstanden sein. Dies kann passieren, wenn ein anderer (fehlerhaft arbeitender) Auftraggeber Daten aus dem übergebenen Datenblock “stiehlt”, indem er in der Zeit, in der dieser Pufferabschnitt zum Lesen geöffnet war, über das Transferregister aus dem fremden Pufferabschnitt gelesen hat. Der Auftraggeber, von dem der READ-Auftrag stammte, wird dann einen Checksummenfehler entdecken. Um dennoch an das korrekte Stabile Objekt zu kommen, müßte er denselben Auftrag erneut stellen.

Steht bei der Auftragsübergabe des Auftraggebers kein passender Pufferplatz zur Verfügung, so wird dies im Ergebnisregister als P_FULL vermerkt. Der Auftraggeber muß das Schreiben des Auftrags zu einem späteren Zeitpunkt wiederholen. Die Bestimmung einer Wartezeit ist dabei allein Sache des Auftraggebers. Darauf nimmt der Stabile Speicher keinen Einfluß. Dieser erlaubt nur nach Bereitstellung des Pufferplatzes die weitere Fortführung des Auftrags.

In dieser Situation, in der kein Pufferplatz zur Verfügung steht, wäre statt des Wartens auch eine andere Fortführung möglich. Da aus Sicht des Stabilen Speichers der Auftrag noch nicht angenommen wurde, was aus der P_FULL-Mitteilung erkennbar ist, könnte nun auch ein anderer Auftrag gestartet werden. Dazu müßten nur andere Einträge in die Parameter- und ins Auftragsregister erfolgen. Eine solche Vorgehensweise erfordert allerdings mehr Intelligenz von der Software, die für den Start der Aufträge verantwortlich ist.

Die FIFO-Arbeitsweise des Puffers erfordert eine besondere Vorgehensweise beim Auftreten eines Übertragungsfehlers, der sich zwischen dem Pufferbereich des Stabilen Speichers und dem Auftraggeber ereignet. Dabei wird angenommen, daß der Übertragungsfehler transienter Natur ist, so daß eine Wiederholung des Einzelwort-Transports erfolgversprechend erscheint. Es sind zwei Situationen zu unterscheiden:

- Schreibzugriff auf das Transferregister,
- Lesezugriff auf das Transferregister.

Bei einem Schreibzugriff ist der Stabile Speicher ständig in der Rolle des Empfängers. Er überprüft die Adresse und den übertragenen Wert auf Parity-Fehler. Bei Entdeckung eines Fehlers wird der gewünschte Zugriff nicht durchgeführt. Es erfolgt auch keine Änderung bei der aktuellen Transferadresse. Dem Auftraggeber wird durch das Error-Signal mitgeteilt, daß ein Übertragungsfehler aufgetreten ist. Im Stabilen Speicher ändert sich dabei kein Eintrag. Daher wird beim wiederholten Zugriff auf das Transferregister das zu schreibende Wort an der richtigen Stelle im FIFO-Puffer eingetragen.

Etwas anders ist die Situation bei einem fehlerhaften Lesezugriff auf das FIFO. Zunächst befindet sich der Stabile Speicher bei der Adreßübertragung in der Empfängerrolle und führt bei einem hier erkannten Fehler keine Aktion aus. Der Anwender erhält ebenfalls das Error-Signal. Diese Situation ist die gleiche wie beim Schreibzugriff. Dagegen wird eine Störung, die die übertragenen Daten auf dem Weg vom Stabilen Speicher zum Prozessor verändert, nur vom empfangenden Prozessor als Parityfehler erkannt, vom Stabilen Speicher aber nicht bemerkt. Hier ist der Auftraggeber gefordert, dem Stabilen Speicher erkennbar zu machen, daß beim nächsten Zugriff auf den FIFO-Puffer das soeben ausgegebene Wort erneut übertragen werden soll. Dazu gibt es innerhalb der Kommunikationspage das Wiederholungs-Register (KP[6]), das nun an Stelle des Transferregisters adressiert werden muß. Dieses Register enthält immer das letzte aus dem FIFO-Puffer gelesene Wort. Dabei muß die vom Anwender erhaltene Adresse korrekt gewesen sein; d.h. *ausschließlich* bei einem Lesezugriff, bei dem der Übertragungsfehler beim gelesenen Wort aufgetreten ist, führt der nachfolgende Zugriff auf das Wiederholungsregister zum gewünschten Resultat. Betraf der Übertragungsfehler dagegen die Adresse, so muß beim Wiederholungszugriff das Transferregister adressiert werden.

Zur Klärung der Art des aufgetretenen Fehlers reicht das Error-Signal auf dem Übertragungsweg allein nicht aus, denn es zeigt nur irgendeinen Übertragungsfehler an. Die Unterscheidung, ob der Fehler bei der Adreß- oder bei der Datenübertragung aufgetreten ist, läßt sich aus dem Statusregister im Prozessorinterface ermitteln. Nach Auswertung dieser Information kann richtig auf den Übertragungsfehler reagiert werden:

- erneuter Schreibzugriff auf den Puffer über das Transferregister, oder
- erneuter Lesezugriff auf den Puffer über das Transferregister bei Adreßfehler,
- Lesezugriff auf das Wiederholungsregister bei Lese-Datenfehler.

Auch an dieser Stelle besteht die bereits angesprochene Gefahr, daß mehrere Übertragungsfehler, die im gleichen Prozessorknoten aufgetreten sind, zu ungeklärten Fehlersituationen führen können (siehe Abschnitt 3.3, Prozessorinterface).

Der Inhalt des Transfer- und des Wiederholungsregisters ist jeweils ein 32-bit-Wort (siehe Abb. 26). Auf das Transferregister sind sowohl Schreib- als auch Lese-Zugriffe erlaubt, allerdings nur dann, wenn ein entsprechender Transfer zur Abwicklung eines Auftrags durchgeführt werden soll und der Pufferplatz zur Verfügung steht. Auf das Wiederholungsregister sind dagegen nur Lesezugriffe erlaubt. Schreibzugriffe auf dieses werden dagegen nicht ausgeführt.

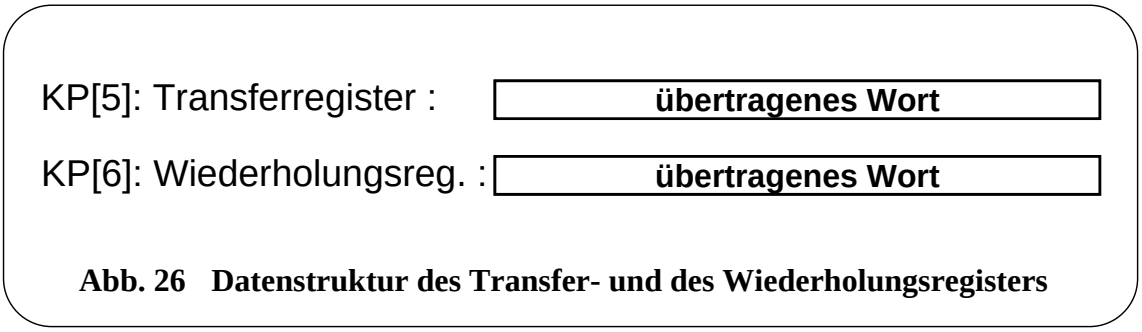


Abb. 26 Datenstruktur des Transfer- und des Wiederholungsregisters

Aus dem Wiederholungsregister kann im Gegensatz zum Transferregister zu jeder Zeit gelesen werden. Damit ist sichergestellt, daß auch dann noch aus dem Wiederholungsregister gelesen werden kann, wenn sich der Übertragungsfehler beim Transfer des letzten Wortes des Datenblocks ereignet hat und der Pufferabschnitt bereits wieder gesperrt ist.

Zusammenfassend ist in Abb. 27 ein einfacher Ablauf für einen WRITE-Auftrag als C-Programm dargestellt.

```
WRITE_Auftrag ()
{
ergebnis = KP[4] & 0x000FFFFF; /* bereit fuer neuen Auftrag ? */
if ((ergebnis==IDLE)||(ergebnis==BUSY)) { return(); }

KP[1] = (Passwort | Objektnummer);
KP[2] = (Passwort | Objektlaenge);
KP[3] = (Passwort | Benutzerinfo);
/* und als letztes: */
KP[0] = (Passwort | WR_Auftrag);
/* Die Angabe von KP[3] ist nur bei "WRITE_and_WRITE_BI" noetig */
/* Unter den Angaben Passwort, Objekt-... ist */
/* die entsprechend formatierte Version zu verstehen */

ergebnis = KP[4] & 0x00F0000;
while (ergebnis==P_FULL) {
     warte(gewisse_Zeit);
     KP[0] = (Passwort | WR_Auftrag);
     ergebnis = KP[4] & 0x00F0000;
     }

/* Pufferplatz ist nun vorhanden */
/* Schreibe Objekt mit Checksumme in den Puffer */
for (i=0; i<=Objektlaenge; i++) { KP[5] = Objekt[i]; }

/* Falls nicht auf das Ergebnis gewartet werden soll, */
/* kann hier "return()" erfolgen; */
/* ansonsten: */
do {
     warte(gewisse_Zeit);
     ergebnis = KP[4] & 0x000FFFFF;
     }
while ((ergebnis==IDLE)||(ergebnis==BUSY));

auswertung_des_ergebnisses(); /* OK oder FAIL */

}
```

Abb. 27 Programm für WRITE-Auftrag

Die Angaben von Objektnummer, Objektlänge, Benutzerinfo und Auftrag sind dabei maximal 20 Bit lang. Das 12 Bit lange Paßwort ist linksbündig im geschriebenen Wort eingetragen.

Die physikalischen Adressen, unter denen der Auftraggeber die Register KP[0] bis KP[5] ansprechen kann, ist zum einen abhängig von dem Adreßbereich, unter dem der Auftraggeber den Stabilen Speicher sieht, und zum anderen von der vergebenen Kommunikationspage. Wie er diese erhält, ist im Abschnitt 6.4.4 über die VIP beschrieben.

Ähnlich ist die Vorgehesweise beim READ-Auftrag (siehe Abb. 28).

```
READ_Auftrag ()
{
ergebnis = KP[4] & 0x000FFFFF; /* bereit fuer neuen Auftrag ? */
if ((ergebnis==IDLE)||(ergebnis==BUSY)) { return(); }

KP[1] = (Passwort | Objektnummer);
KP[2] = (Passwort | Objektlaenge);
/* und als letztes: */
KP[0] = (Passwort | RD_Auftrag);
/* Unter den Angaben Passwort, Objekt-... ist */
/* die entsprechend formatierte Version zu verstehen */

ergebnis = KP[4] & 0x00F0000;
while (ergebnis==P_FULL) {
     warte(gewisse_Zeit);
     KP[0] = (Passwort | RD_Auftrag);
     ergebnis = KP[4] & 0x00F0000;
     }

/* ergebnis == IDLE oder BUSY oder eine Fehlermeldung */
/* Der Stabile Speicher ist nun an der Reihe */
do {
     warte(gewisse_Zeit);
     ergebnis = KP[4] & 0x000FFFFF;
     }
while ((ergebnis==IDLE)||(ergebnis==BUSY));

/* Lese Objekt mit Checksumme aus dem Puffer */
for (i=0; i<=Objektlaenge; i++) { Objekt[i] = KP[5]; }

auswertung_des_ergebnisses(); /* OK oder FAIL */

}
```

Abb. 28 Programm für READ-Auftrag

Am Ende eines Auftrags kann der Auftraggeber das erhaltene Ergebnis auswerten. Für die Arbeitsweise des Stabilen Speichers ist dies aber nicht notwendig. Das Ergebnisregister gibt dem Auftraggeber Aufschluß darüber, wie sein Auftrag bearbeitet wurde. Neben dem gewünschten Ausgang (“erfolgreiche Ausführung”) gibt es noch einige andere Ergebnisse, die anzeigen, aus welchem Grund ein Auftrag nicht ausgeführt werden konnte. Einige Dinge beziehen sich auf die übergebenen Angaben des Auftraggebers. Wenn z.B. das angegebene Stabile Objekt nicht existiert oder zu einer anderen Kommunikationspage (d.h. zu einem anderen Auftraggeber) gehört, führt der Stabile Speicher nichts aus und gibt im Ergebnisregister eine entsprechende Mitteilung aus. Ebenso wird verfahren, wenn die angegebene Länge nicht mit der im Stabilen Speicher abgelegten Information übereinstimmt.

Andere Angaben beziehen sich auf den Mißerfolg bei der Bearbeitung des Stabilen Objekts im Stabilen Speicher. Die Gründe dafür können zum einen darin liegen, daß der Auftraggeber bei der Übergabe des Datenblocks falsche Angaben gemacht hat. Beispielsweise paßt die im Stabilen Speicher berechnete Checksumme nicht mit der überein, die der Auftraggeber übergeben hat. Das kann sowohl auf einen Defekt dieses oder eines benachbarten Auftraggebers hindeuten als auch auf ein Defekt auf dem Übertragungsweg.

Bei der Übertragung des Datenblocks zwischen dem Auftraggeber und dem Pufferbereich des Stabilen Speichers kann sich eine Situation einstellen, in der der Auftraggeber auf Grund eines Fehlers außer Tritt gerät und nicht mehr herausfinden kann, bei welchem Wort er inzwischen angekommen ist. Um in solchen Fällen einen einfachen Ausstieg aus dem aktuell laufenden Auftrag zu ermöglichen, existiert ein Abbruchregister in der Kommunikationspage (KP[7]). Das Schreiben des Paßwortes (linksbündig) auf dieses Abbruchregister bewirkt, daß der Pufferplatz wieder freigegeben wird. Ein Abbruch ist allerdings nur dann möglich, wenn bei einem Auftrag *mit* Datentransfer die Transferphase an der Reihe ist. In allen anderen Fällen hat der Schreibzugriff auf das Abbruchregister keine Wirkung. Der Erfolg dieser Aktion kann auch im Ergebnisregister abgefragt werden.

Diese Einschränkung, einen Abbruch nur in der Transferphase zuzulassen, ist nötig, um unklare Situationen zu vermeiden. Da der Stabile Speicher nach einer Auftragserteilung (Eintrag im Auftrags-FIFO) bis zur Auftragsfertigstellung keinen Kontakt zum Auftraggeber herstellt, würde ein Abbruch während dieser Phase eine unterschiedliche Sicht zur Folge haben. Der Stabile Speicher wäre z.B. noch mit einem Update eines Stabilen Objekts beschäftigt, während der Auftraggeber dieses nach der vollständigen Übergabe der Daten abbrechen wollte. Ein solches Rückgängigmachen darf es aber nicht geben. Dies würde nämlich bedeuten, daß der Auftraggeber die aus seiner Sicht erfolgreiche Übertragung seines Datenblocks nachträglich rückgängig machen wollte, ohne daß hierfür Gründe vorliegen würden, die die Funktionsweise des Stabilen Speichers betreffen.

Abschließend wird nun der Aufbau einer Kommunikationspage zusammengefaßt:

Die Kommunikationspage kann als zusammenhängender Block von 8 Worten interpretiert werden, wie es bereits in der Bezeichnung KP[0] .. KP[7] deutlich wurde. Auf diese Register darf nur wortweise zugegriffen werden. Byte- und Halbwort-Zugriffe werden ebenso abgewiesen wie Blockmoves, wie sie beim Transport von Cachelines vorkommen. Auch eine Ausführung des XMEM-Befehls auf diese Register wird nicht erlaubt. In allen diesen Fällen wird der Transfer mit einem Error-Signal beendet.

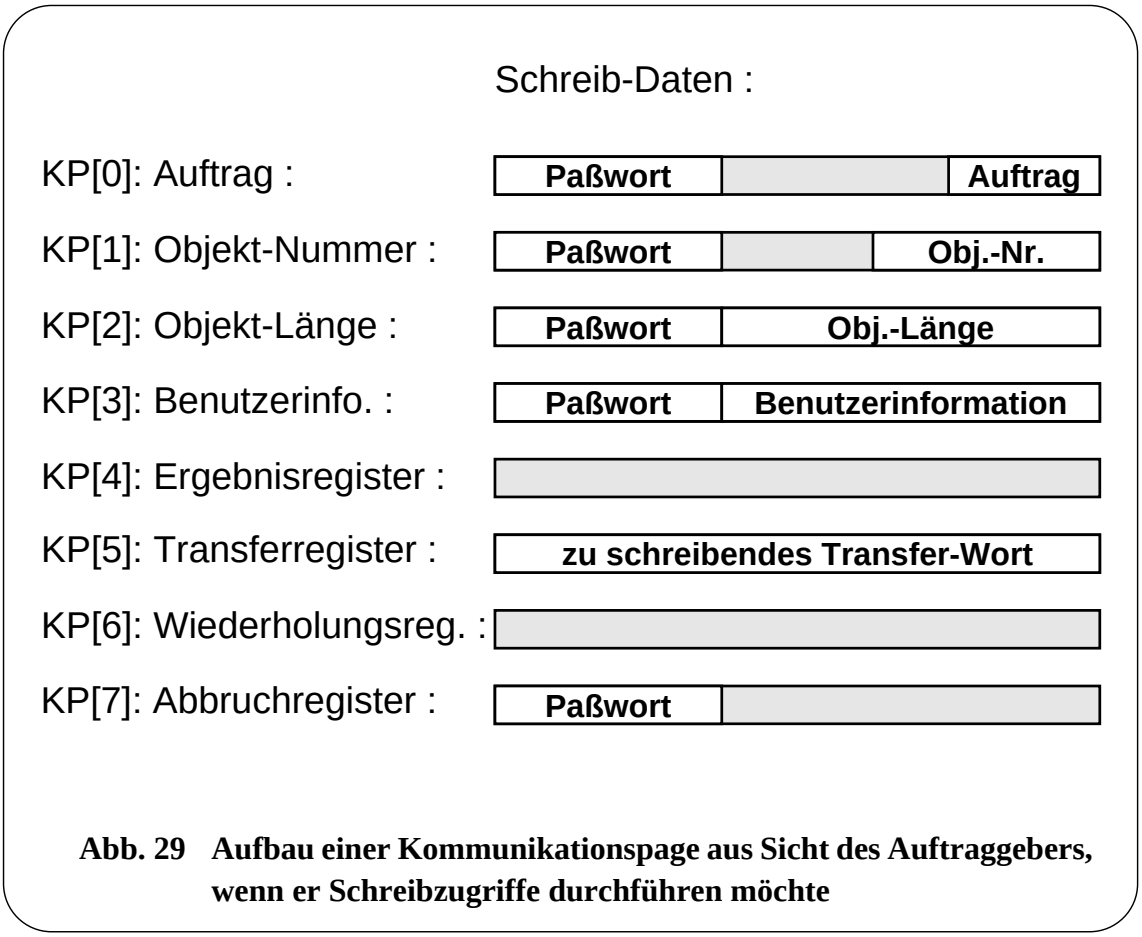


Abb. 29 Aufbau einer Kommunikationspage aus Sicht des Auftraggebers, wenn er Schreibzugriffe durchführen möchte

Auf diese 8 Register in der Kommunikationspage gibt es zwei Sichtweisen. Beim Lesen herrscht ein anderes Format vor als beim Schreiben (siehe Abb. 29 und Abb. 30). Der größte Unterschied ist der, daß beim Schreiben in den meisten Fällen ein Paßwort angegeben werden muß, und beim Lesen wird an dieser Stelle Null ausgegeben. Das dient dem Schutz der Auftraggeber untereinander. Denn dadurch hat kein Auftraggeber die Möglichkeit, das Paßwort anderer Auftraggeber auf einfache (zufällige) Weise zu erfahren. Auch der Auftrag und die Objekt-Länge kann nicht gelesen werden. Dagegen muß das Lesen der Objekt-Nummer und der Benutzerinformation eines Objekts möglich sein, da dies bei der Bearbeitung der Aufträge CREATE und READ_BI notwendig ist. In allen Bereichen eines gelesenen Wortes, die kein Informationsfeld enthalten, wird Null ausgegeben.

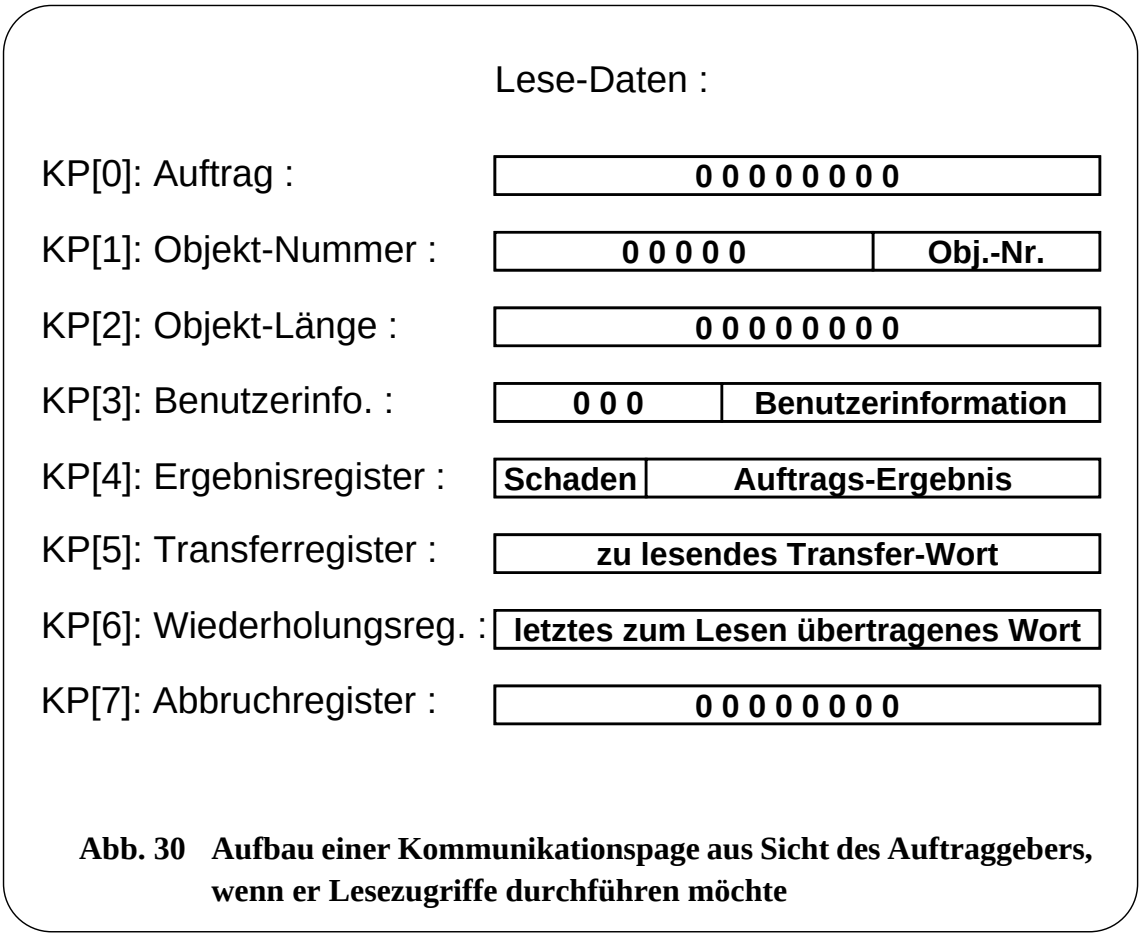


Abb. 30 Aufbau einer Kommunikationspage aus Sicht des Auftraggebers, wenn er Lesezugriffe durchführen möchte

Intern stellt sich die Kommunikationspage dem Stabilen Speicher etwas anders dar. Hier ist es zwar auch ein zusammenhängender Block von 8 Worten. Deren Inhalt ist aber geringfügig anders als es sich für den Auftraggeber darstellt. Dies ist in Abb. 31 dargestellt.

Die vier Auftragsregister haben dieselbe Struktur. Das Paßwort ist mit abgespeichert.

Das Ergebnisregister ist nur mit den untersten 24 Bit belegt; die oberen 8 Bit tragen einen beliebigen Wert. In der Hardware der Puffersteuerung und in der Software der zentralen Steuerung wird dafür gesorgt, daß Null eingetragen ist. Das allgemeine Ergebnisregister ist ein getrennt liegendes Register, das von der zentralen Steuerung separat geschrieben wird. Es ist aus Sicht des Stabilen Speichers nicht Teil einer Kommunikationspage. Dieses Register wird nur bei Lesezugriffen des Auftraggebers an Stelle der obersten 8 Bit des Ergebnisregisters aus der Kommunikationspage eingeblendet.

Im Transferregister ist die aktuelle Adresse (für Worte) untergebracht, der beim Datentransport zwischen Auftraggeber und Pufferspeicher benötigt wird. Bei der Übergabe eines WRITE-Auftrags an die zentrale Steuerung ist hier die Basisadresse des Pufferabschnitts eingetragen. Bei der Beendigung eines READ-Auftrags durch die zentrale Steuerung muß hier von ihr die Start-

adresse des Datenblocks eingetragen werden, bevor der Auftraggeber den Datenblock aus dem Pufferspeicher lesen kann.

Im Wiederholungsregister wird immer das zuletzt aus dem Puffer gelesene Wort eingetragen, so wie es auch der Anwender sieht.

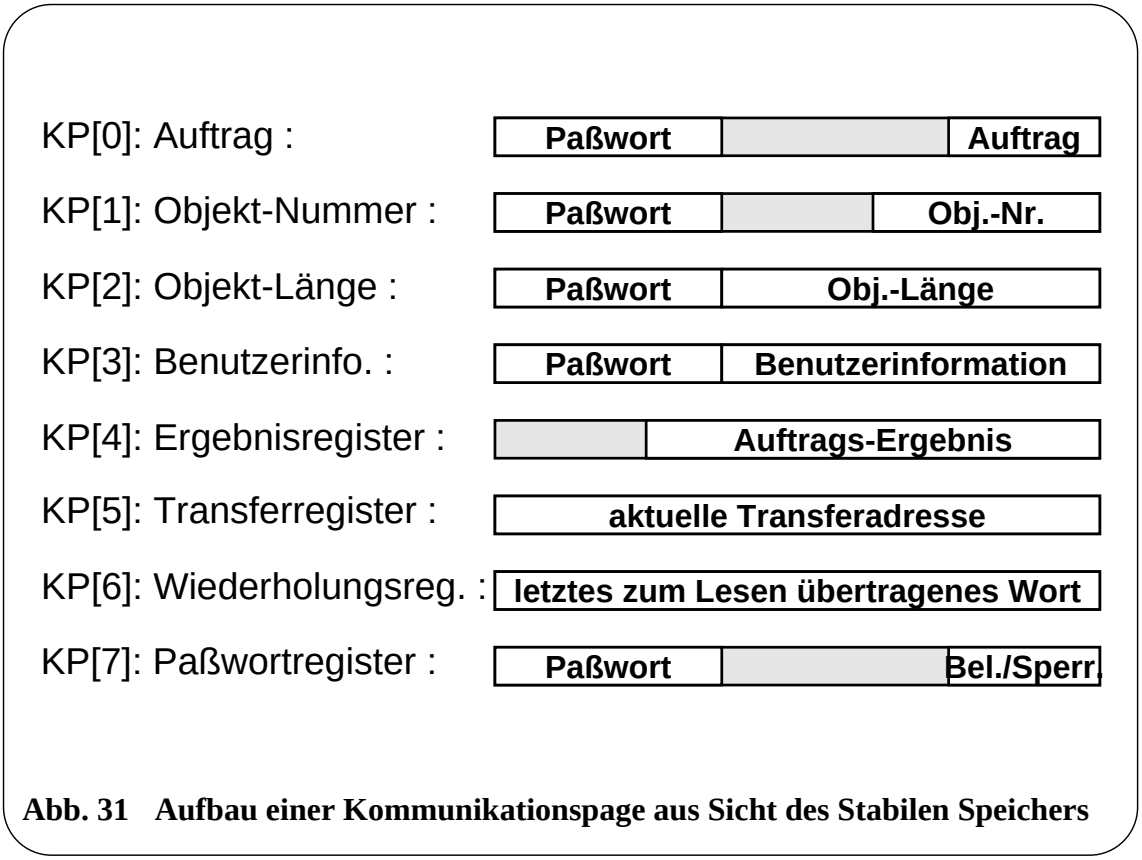


Abb. 31 Aufbau einer Kommunikationspage aus Sicht des Stabilen Speichers

Das Abbruchregister wird vom Auftraggeber nicht in der Weise benutzt, daß er etwas darin ablegen möchte, sondern so, daß ein Schreibzugriff darauf nur als Aufforderung verstanden wird, einen Abbruch des aktuellen Auftrags durchzuführen. Das dabei übertragene Paßwort wird dabei nirgends abgespeichert sondern steht nur zum Vergleich mit dem vergebenen Paßwort zur Verfügung. Da dieses Register somit beim Abbruch nicht als Speicherplatz benötigt wird, steht es hier für einen anderen Zweck zur Verfügung. Hier werden bei der Anmeldung des Auftraggeber (ACCESS) das vergebene Paßwort eingetragen und die Informationen, die Angaben darüber machen, ob diese Kommunikationspage belegt ist oder nicht, und ob derzeit lesend oder schreibend über diese Kommunikationspage auf den Pufferspeicher zugegriffen werden darf oder nicht. Es ist wichtig, daß diese Informationen im Pufferbereich abgelegt sind, damit sie bei jedem Zugriff eines Auftraggebers schnell verfügbar sind. Sie müssen jedesmal zum Vergleich herangezogen werden. Die Belegt-Information und die Angabe über die derzeit erlaubten Transferzugriffe werden sogar innerhalb des Wortes fehlertolerant abgelegt, “belegt” als “111”, frei als “000” (binär). Somit kann ein Einbit-Fehler nicht dazu führen, daß die Zugriffsmöglichkeit auf diese Kommunikationspage geändert wird. Ähnliches gilt auch für die Information über die Sperrung des Pufferzugriffs. Hier werden in fünf Bit die Informationen “Zugriff nur auf

Auftragsregister erlaubt", "Zugriff weder auf Auftragsregister noch auf Puffer erlaubt", "nur Schreibzugriff auf Puffer erlaubt", "nur Lesezugriff auf Puffer erlaubt" fehlertolerant codiert. Auch hier ist ein Einbit-Fehler tolerierbar. Die Arbeitsweise des Stabilen Speichers ist so ausgelegt, daß solche Einzelbitfehler nach spätestens der gleichen Frist erkannt und korrigiert werden, wie entsprechende Fehler bei den Datenblöcken durch den regelmäßigen Check erkannt und behoben werden.

Die zentrale Steuerung kann auch auf die restlichen Einträge einer Kommunikationspage (KP[8] .. KP[63]) schreibend und lesend zugreifen. In der derzeitigen Implementierung wird dieser Speicherplatz aber nicht genutzt.

6.4.4 Die Vermittlungs- und Informations-Page (VIP)

Bisher wurde beschrieben, auf welche Weise ein Auftraggeber Zugriffe auf den Stabilen Speicher durchführen kann. Dazu ist die Benutzung einer zugewiesenen Kommunikationspage notwendig. In diesem Abschnitt wird erläutert, wie einem zukünftigen Auftraggeber eine Kommunikationspage zugeteilt wird.

Diese Zuteilung erfolgt über eine besondere Page, die Vermittlungs- und Informationspage (VIP). Vom Adreßraum her ist sie den Kommunikationspages zugeordnet. Sie vervollständigt die 256 Pages. Ansprechbar ist die VIP im Adreßraum des Stabilen Speichers an der Stelle, an der die Kommunikationspage mit der Nummer 0 liegen würde. Ihre physikalische Basisadresse läßt sich aus der physikalischen Basisadresse desjenigen Kommunikationsspeichers bilden, an den dieser Stabile Speicher angeschlossen ist. Bezogen auf eine 32-bit-Zahl werden die vordersten 8 Bit der Adresse zum Ansprechen des Kommunikationsspeichers übernommen, gefolgt von zwei gesetzten Bits. Alle anderen Adreßbits sind Null. Lautete die Basisadresse des Kommunikationsspeichers z.B. hexadezimal "81000000", so ist die Basisadresse des Stabilen Speichers, und damit auch die Basisadresse der VIP gleich "81C00000".

Im Gegensatz zu den anderen Kommunikationspages besteht für die VIP kein situationsabhängiger Zugriffsschutz. Allerdings sind vom Auftraggeber aus nur Lesezugriffe auf sie erlaubt. Bei Schreibzugriffen auf sie wird nichts eingetragen.

Die VIP ist 64 Worte groß. Die erste Hälfte davon nimmt ein Abschnitt ein, der für Informationsdienste verwendet wird. In diesem Bereich sind Daten abgelegt, die zum einen dem Zugreifer kenntlich machen, daß sich in diesem 4 MByte großen Adreßraum die Schnittstelle zu einem Stabilen Speicher befindet. Weiterhin sind Angaben hinterlegt, die Aussagen über die Größe und die aktuelle Belegung dieses Stabilen Speichers machen. Diese Angaben können vom Auftraggeber interpretiert werden. Diese Einträge sind in Abb. 35 zu finden.

In der zweiten Hälfte befindet sich der Vermittlungsteil. Die meisten Worte tragen Informationen, die vom Auftraggeber nicht interpretiert werden können. Zugriffe darauf sind zwar möglich, die gelesenen Informationen stammt aber aus internen Verwaltungsdaten des Stabilen

Speichers, die der Puffervergabe dienen. Es ist nicht vorgesehen, daß der Auftraggeber diese Daten interpretiert. Beim Lesen werden sie jedoch genauso ausgegeben wie die Daten aus dem Informationsteil.

Das letzte Wort im Vermittlungsteil (VIP[63]) ist das “Vermittlungsregister”. Es wird von einer besonderen Hardware gespeist, die eine neue Kommunikationspage-Nummer und ein Paßwort ausgibt. Diese Einheit erzeugt in einer quasi zufälligen Reihenfolge zu jedem Takt eine neue Kommunikationspage-Nummer (KP-Nummer) und ein Paßwort. Die Folge der erzeugten Bitkombinationen ist zwar deterministisch; da ihr Wechsel aber zu jedem Takt erfolgt, der Auftraggeber aber schon allein auf Grund der nicht deterministisch langen Zugriffe auf seine lokalen DRAM-Speicher zeitlich nie so exakt arbeitet, erfolgt die Vergabe von Kommunikationspage und Paßwort quasi zufällig. Dieser Eindruck wird dadurch noch verstärkt, daß die Reihenfolge der erzeugten Teilworte (Kommunikationspage-Nummer und Paßwort) nicht aus einer auf- oder absteigenden Folge von Zahlen besteht. Stattdessen wird für jedes Teilwort je ein rückgekoppeltes Shiftregister benutzt. Bei diesen Rückkopplungen wurde darauf geachtet, daß ihnen primitive Polynome zugrunde liegen [WIS78]. Dann erhält man Zyklen, die die Länge 2^n-1 haben. Das bedeutet, daß bis auf den Wert Null alle möglichen Zahlen vertreten sind. Für die Anzahl der KP-Nummern gilt n=8, d.h. es gibt 255 Kommunikationspages. Für das Paßwort gilt n=12, d.h. 4095 verschiedene Paßwörter kommen vor.

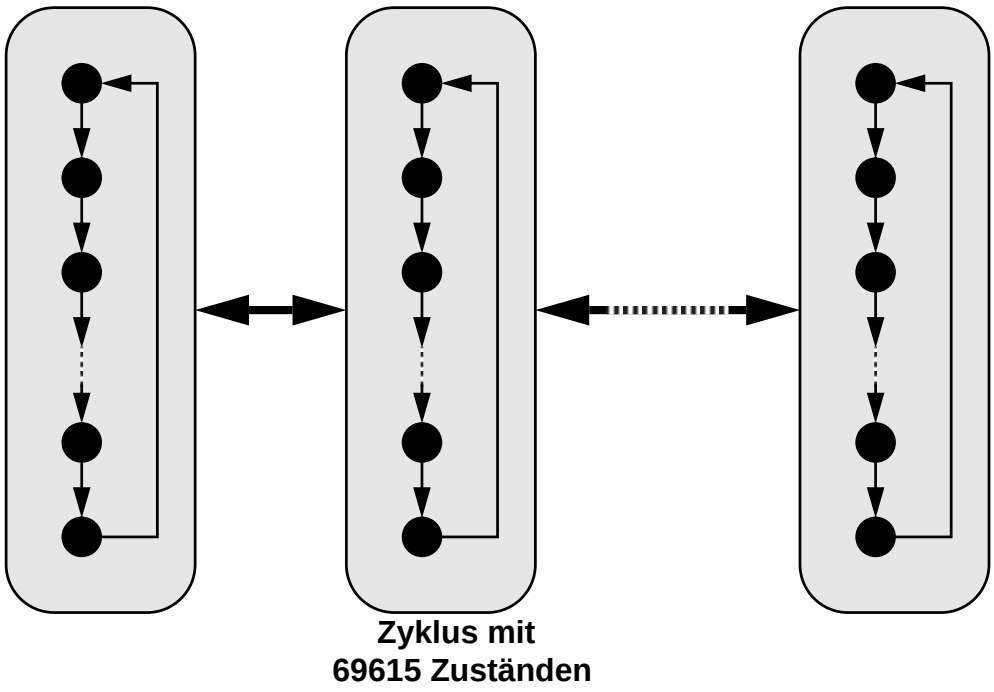


Abb. 32 Zustandsmenge und Übergänge der Kombinationen zwischen KP-Nummer und Paßwort

Die beiden Zyklen werden gleichzeitig gestartet und laufen mit derselben Taktfrequenz (hier 32 MHz). Da die Zyklen unterschiedlich lang sind, wird es zu jeder KP-Nummer auch unterschiedliche Paßwörter geben. Um zu ermitteln, nach welcher Taktzahl sich der gesamt Zyklus wiederholt, muß das kleinste gemeinsame Vielfache (kgV) der beiden Zyklenlängen berechnet werden. Daraus ergeben sich 69615 Takte, das sind 17 x 4095 Takte bzw. 273 x 255 Takte. Mit anderen Worten, solange man sich in einem Zyklus befindet, können zu jeder KP-Nummer 273 verschiedene Paßwörter vergeben werden. Um die Kombinationsmöglichkeiten zwischen KP-Nummer und Paßwort noch deutlich zu erhöhen, wird bei jeder Vergabe einer neuen Kommunikationspage der Lauf des Paßwort-Zyklus einige Takte länger angehalten als der Zyklus für die KP-Nummer. Dadurch wird erreicht, daß man sich anschließend in einem neuen gemeinsamen Zyklus von KP-Nummer und Paßwort mit 69615 Zuständen befindet. So kann im Laufe der Zeit zu jeder Kommunikationspage ein beliebiges Paßwort zugeordnet werden. Insgesamt sind das 4095 x 255 = 1044225 Kombinationen (siehe Abb. 32).

Bei der Vergabe der Kommunikationspage-Nummer und des Paßwortes ist es notwendig, daß diese Informationsausgabe (KP-Nummer und Paßwort) atomar erfolgt. Da für den Auftraggeber nur einfache Lesezugriffe in Frage kommen, muß für die Vergabe eine Datenstruktur gewählt werden, die mit 32 Bit auskommt. Für die KP-Nummer sind 8 Bit erforderlich, für das Paßwort weitere 12 Bit. Ferner muß noch mitgeteilt werden, ob überhaupt noch eine Kommunikationspage vergeben werden konnte. Dafür ist ein Bit nötig, verwendet wurden aber vier Bit. Der Auftrag ACCESS besteht somit allein aus dem Lesen des Vermittlungsregisters. Dessen Datenstruktur ist in Abb. 33 angegeben.

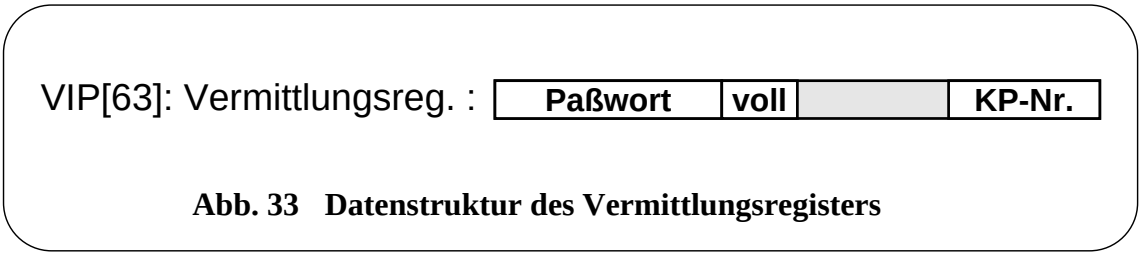


Abb. 33 Datenstruktur des Vermittlungsregisters

Nach dem Lesezugriff auf das Vermittlungsregisters muß der Auftraggeber entscheiden, ob das übergebene 32-bit-Wort zur Vergabe einer neuen Kommunikationspage geführt hat oder nicht. Im Erfolgsfall ist der 4-bit-Bereich “voll” gleich Null. Dann können die KP-Nummer und das Paßwort in lokale Variablen des Auftraggebers übertragen werden, die dann bei den folgenden Zugriffen auf die eigene Kommunikationspage verwendet werden. Es ist dabei sehr wichtig, daß beim Auftrag ACCESS nur einmal auf das Vermittlungsregister zugegriffen wird. Ansonsten würden mehrere Kommunikationspages zugeteilt werden. Deren Angaben würden entsprechend dem beabsichtigten Zweck untereinander natürlich nicht zusammenpassen.

Aus der Angabe zur KP-Nummer muß für die weiteren Zugriffe noch die physikalische Basisadresse der übergebenen Kommunikationspage ermittelt werden. Diese Adresse wird aus der physikalischen Basisadresse der benutzten VIP gebildet. Diese muß als Hexadezimalzahl die Form "XXC00000” besitzen. (XX bezeichnet 2 Hexadezimalstellen, das C das hexadezimale C

und jede 0 eine hexadezimale 0.) Der zweite Teil der Basisadresse wird aus der übergebenen KP-Nummer ermittelt. Diese muß noch um einige Bitstellen nach links verschoben werden. Die Anzahl der Bitstellen läßt sich aus einem Eintrag im Informationsteil der VIP entnehmen. In VIP[26] ist die eingestellte Pagegröße (in Bytes) für das Host-System eingetragen (siehe auch Abb. 35). Sie sollte mit der tatsächlichen Pagegröße des Host-Systems übereinstimmen, damit der Pageschutzmechanismus der Host-Speicherverwaltung wirken kann. Ist in VIP[26] z.B. die Hexadezimalzahl “1000” eingetragen, so muß um 12 Bit nach links verschoben werden. Dies ist die Pagegröße des M88K-Systems.

```
/*im Bereich eines Stabilen Speichers: */
unsigned long int VIP[64];

/* lokal im Prozessorknoten: */
long int lokale_vip, Passwort, voll, kpnr, hpg;
unsigned long int BasisKP, BasisVIP;

/* ACCESS: */
lokale_vip = VIP[63]; /* Lesen nur 1 x durchfuehren !!! */

voll = lokale_vip & 0x000F0000 ;
if (voll == 0) { /* Kommunikationspage wurde vergeben */
     Passwort = lokale_vip & 0xFFF00000 ;
     kpnr = lokale_vip & 0x000000FF ;
     hpg = VIP[26]; /* eingestellte Pagegr. d. Host-Systems */
     hpg >>= 1;
     while (hpg != 0) {
          hpg >>= 1;    kpnr <<= 1;
          }
     BasisKP = BasisVIP | kpnr;
     }
else {
     /* Stabiler Speicher kann nicht benutzt werden */
     }
```

Abb. 34 Programm für den Auftrag ACCESS und “Zubereitung” der lokalen Parameter “Passwort” und “BasisKP”

Ein Problem ergibt sich aber noch an dieser Stelle. Ereignet sich beim Transport des aus dem Vermittlungsregister gelesenen Wortes unterwegs ein Übertragungsfehler, so hat der Stabile Speicher eine Kommunikationspage vergeben, der Auftraggeber aber keine erhalten. Da in diesem Fall kein Auftraggeber das 32-bit-Wort abfragen und anschließend benutzen kann, ist es niemandem möglich diese vergebene Kommunikationspage wieder freizugeben. Auch der Sta-

bile Speicher darf dies nicht ohne korrekten Auftrag des Auftraggebers tun. Daher wird diese Kommunikationspage in Zukunft bis zum Abschalten des Stabilen Speichers nicht mehr verwendet werden können. Daher sollte nach einem Übertragungsfehler, der beim Lesen des Vermittlungsregisters auftritt, in der Fehlerbehandlung äußerst vorsichtig reagiert werden:

Ist anhand des Statusregisters im Prozessorinterface erkennbar, daß das gelesene Wort mit einem Parityfehler behaftet war, so sollte nicht durch oftmaliges Wiederholen des Lesezugriffs versucht werden, doch noch dieses Wort (d.h. den Zugang zum Stabilen Speicher) zu bekommen. Denn dann würden mehrere Kommunikationspages umsonst belegt werden. Akzeptabel wäre noch, eine sehr begrenzte Zahl (z.B. 3) weiterer Lesezugriffe auf das Vermittlungsregister durchzuführen. Ist dies erfolgreich, was bei einem vorherigen transienten Fehler auf der Übertragungsstrecke sehr wahrscheinlich ist, so wurden nur wenige Kommunikationspages umsonst vergeben. Führten diese Zugriffe beim transportierten Wort aber erneut zu Übertragungsfehlern, so sollten weitere Versuche unterbleiben, diesen Stabilen Speicher mit dem Auftrag ACCESS zu erreichen.

Ist dagegen im Statusregister des Prozessorinterfaces erkennbar, daß die Adreßübertragung mit einem Parity-Error quittiert wurde, so wurde auch beim Stabilen Speicher kein Zugriff durchgeführt und somit auch keine Kommunikationspage vergeben. Weitere Versuche können in diesem Fall bedenkenlos durchgeführt werden.

Eine andere Fehlerart kann eintreten, die ebenfalls zum Verlust von bisher unbelegten Kommunikationspages führen kann. Liest ein Auftraggeber ständig das Vermittlungsregister, so eignet er sich damit viele oder sogar alle noch verfügbaren Kommunikationspages an. Aus der Sicht des Stabilen Speichers (Pufferbereichs) ist dies nicht von einer korrekten Arbeitsweise mehrerer Auftraggeber zu unterscheiden. Auf diese Situation kann hindeuten, wenn innerhalb des Informationsteils der VIP in VIP[24] (Zahl der freien Kommunikationspages) eingetragen ist, daß alle Kommunikationspages benutzt sind. Allerdings ist diese Information mit großer Vorsicht zu behandeln, da es auch bei einwandfreier Arbeitsweise des Stabilen Speichers und aller angeschlossenen Auftraggeber zu dieser Situation kommen kann.

In allen Situationen, in denen es zum Verlust von bislang unbelegten Kommunikationspages kommen kann, ist sichergestellt, daß die bisher beim Stabilen Speicher angemeldeten Auftraggeber und ihre Stabilen Objekte keine Einbußen bezüglich ihres Schutzes vor unerlaubtem Zugriff erfahren. Diese Auftraggeber können auch noch normal weiterarbeiten. Aber zusätzliche Auftraggeber können möglicherweise diesen Stabilen Speicher nicht mehr benutzen.

In der folgenden Liste (Abb. 35) sind alle Daten zusammengestellt, die in der VIP von den Auftraggebern gelesen werden können.

VIP[0..3]	S T A B L E _ _ S T O R A G E _
VIP[4..7]	M E M S Y _ _ _ V E R _ 1 . 0 _
VIP[8..11]	A N D R E A S _ G R Y G I E R _
VIP[12..15]	M A N F R E D _ S E M M L E R _
VIP[16..19]	U N I _ E R L A N G E N _ _ _ _
VIP[20..23]	I M M D _ 3 _ _ 1 9 9 4 _ _ _
VIP[24]	Zahl der noch freien Kommunikationspages
VIP[25]	Zahl der maximal verfügbaren Kommunik.-Pages
VIP[26]	eingestellte Anwender-Page-Größe (in Bytes)
VIP[27]	Zahl der noch freien Pages im Stabilen Speicher
VIP[28]	Größe einer Page im Stabilen Speicher (in Worten)
VIP[29]	Zahl der maximal verfügbaren Pages im St. Sp.
VIP[30..31]	0
VIP[32..47]	Info über Belegung der Kommunikationspages
VIP[48..55]	0
VIP[56..60]	Info über Belegung der Pufferbereiche
VIP[61..62]	(reserviert, unbekannte Werte)
VIP[63]	**Paßwort** | | **KP-Nr.**

Abb. 35 Struktur der VIP

Um beim Start (Hochfahren) des Systems herauszufinden, ob sich im Bereich eines bereits gefundenen Kommunikationsspeichers ein Stabiler Speicher befindet, kann eine entsprechende “Such-Software” versuchen, in dem Bereich der VIP des vermuteten Stabilen Speichers entsprechende Daten zu finden (lesen). Dabei kann nach der Zeichenkette gesucht werden, die durch die Werte von VIP[0..23] gebildet werden. Das letzte Byte (innerhalb von VIP[23]) hat den Wert Null. Das ist bei der Programmiersprache C das Endekennzeichen eines Strings (Zeichenkette) [KER90].

Ein Stabiler Speicher kann an der Zeichenkette “STABLE__STORAGE_” erkannt werden, die in VIP[0..3] steht. Dieser Eintrag läßt sich auch bei einem Schreibzugriff des (zukünftigen) Auftraggebers nicht verändern. Die restlichen Werte VIP[4..23] können ebenfalls nicht verändert werden. Bei einer anderen Version dieses Stabilen Speichers können sie jedoch andere Werte enthalten als in Abb. 35 angegeben ist. Ist eine solche Zeichenkette an dieser Stelle gefunden worden, so kann die Such-Software davon ausgehen, daß sich hier ein Stabiler Speicher befindet. Die restlichen Angaben (VIP[24..62]) tragen Informationen, die sich zum Teil im Laufe der Zeit ändern können. Für den Anwender sind vor allem die Angaben von VIP[24..31] interessant, da hier Werte über die aktuelle Belegung des Stabilen Speichers zu finden sind. Eine Auswertung der Angaben aus VIP[32..62] durch den Anwender ist dagegen nicht vorgesehen.

Für Testzwecke besteht aber die Möglichkeit, sämtliche im Pufferbereich gespeicherten Daten vom M88K-System aus nach Belieben zu lesen und zu verändern. Dazu ist allerdings die manuelle Umstellung eines Schalters am Stabilen Speicher nötig. Da dieser Modus aber im Normalbetrieb nicht erlaubt ist, wird in dieser Arbeit nicht auf die Testmöglichkeiten eingegangen.

6.5 Die Steuerungs-Einheit für diesen Stabilen Speicher

Die bisherige Beschreibung betraf vor allem die Schnittstelle, wie sich der Stabile Speicher dem Auftraggeber gegenüber darstellt. Wenn ein Auftrag im Pufferbereich abgelegt worden ist, muß er zur Bearbeitung der zentralen Steuerung des Stabilen Speichers vorgelegt werden. Von dieser handelt dieser Abschnitt. Dabei werden zunächst die Übergabe und Bearbeitung eines Auftrags aus Sicht der Steuerung beschrieben, dann die Struktur der Speicherverwaltung in groben Zügen aufgezeigt, Schutzmechanismen gegen eigenen Ausfall erläutert und der interne Aufbau dargestellt.

6.5.1 Übergabe und Bearbeitung der Aufträge

Über das Auftrags-FIFO werden die Aufträge der zentralen Steuerung des Stabilen Speicher übergeben. Dazu wird dort die KP-Nummer von der Puffersteuerung eingetragen. Zu diesem Zeitpunkt hat der Auftraggeber alle Angaben für diesen Auftrag gemacht. Bei Transfer-Aufträgen (WRITE und READ) ist auch ein Platz im Pufferspeicher bereitgestellt, und bei einem WRITE-Auftrag dort bereits der vollständige Datenblock eingetragen worden.

Die zentrale Steuerung muß nun den Auftrag mitsamt den Parametern, die in der angegebenen Kommunikationspage abgelegt wurden, zunächst auf korrekte Angaben überprüfen. Dazu gehört die Kontrolle, ob das angegebene Stabile Objekt auch existiert und dieser Kommunikationspage zugeordnet ist. Weiterhin wird die übergebene Objektlänge mit der Angabe verglichen, die sich der Stabile Speicher beim Erzeugen des Objekts in seiner Objektbeschreibung eingetragen hat. Eine weitere Überprüfung betrifft die Angabe des Auftrags. Durch die vorgesehenen 8 Bit für die Auftragsangabe könnten 256 verschiedene Aufträge gekennzeichnet werden. Da es aber wesentlich weniger gibt, existieren viele Kodierungen, die keinen Auftrag bezeichnen. Dies erkennt die zentrale Steuerung auch als Fehler.

Ist eine der Angaben falsch, so wird der Auftrag abgebrochen und eine entsprechende Fehlermitteilung im Ergebnisregister der entsprechenden Kommunikationspage abgelegt. Der Auftrag ist dann für den Stabilen Speicher abgeschlossen. Ein eventuell belegter Pufferplatz wird wieder freigegeben.

Sind alle Angaben korrekt, so beginnt die zentrale Steuerung mit der Bearbeitung dieses Auftrags. Je nach Auftrag sind sehr unterschiedliche Dinge zu tun. Der Ablauf der Transferaufträge orientiert sich grundsätzlich an der Vorgehensweise von Lampson [LAM88]. Darüber hinaus werden zusätzliche Maßnahmen ergriffen, so daß noch mehr Fehler toleriert werden können als bei Lampson. So muß zu Beginn eines WRITE-Auftrags sichergestellt sein, daß wenigstens eine der beiden (alten) Kopien fehlerfrei ist. Daher wird als erstes eine der beiden Kopien überprüft. Ist sie fehlerfrei, so wird sie bei der Ausführung des WRITE-Auftrags als “Backup-Kopie” (2.Kopie) verwendet. Ist sie aber nicht korrekt, so wird auch die andere Kopie überprüft. Ist diese fehlerfrei, so wird sie als Backup-Kopie verwendet. “Fehlerfrei” bedeutet, daß weder ein Parityfehler noch ein Checksummenfehler vorliegt. Im Fall, daß beide Kopien fehlerhaft sind, wird mit einem VERIFY versucht, einen korrekten Datenblock (für die Backup-Kopie) aus den beiden fehlerbehafteten Blöcken herzustellen. Mißlingt auch dies, wird versucht, den WRITE-Auftrag auch ohne alte Version des Datenblocks auszuführen. Solange dies erfolgreich ist, hat dieser Datenblock einen neuen korrekten Inhalt bekommen. Gelingt dies jedoch nicht, so ist dieses Stabile Objekt in diesem Stabilen Speicher verloren. Für den Auftraggeber wird in jedem Fall eine entsprechende Erfolgs- bzw. Fehlermeldung im Ergebnisregister hinterlegt.

Während der Durchführung eines WRITE-Auftrags wird immer zuerst nur eine Kopie des Stabilen Objekts vollständig verändert. Erst wenn dies erfolgreich beendet worden ist, wird auch die zweite Kopie in gleicher Weise verändert. War bei den Arbeiten an der ersten Kopie ein Fehler aufgetreten, so wird die noch unverändert gebliebene zweite Kopie wieder auf die erste übertragen. Damit wird sichergestellt, daß zu jedem Zeitpunkt ein konsistentes Objekt (alt oder bereits neu) im Lagerbereich vorliegt.

Die umfangreichen Reparaturfähigkeiten bei einem VERIFY sind möglich durch die Verwendung von RAM-Bausteinen mit byteweiser Parityabsicherung. Dies erlaubt eine viel detailliertere Fehlerlokalisierung und damit auch deutlich mehr Fähigkeiten zur Fehlerbehebung als dies bei Daten auf Plattenspeichern mit einer einzigen Good-Bad-Entscheidung über einen Daten-

block möglich ist. So kann bei Parityfehlern der Ort des Fehlers bis auf das Byte genau lokalisiert werden. Außerdem ist es möglich, sehr viele gleichzeitig vorliegende Parityfehler zu beheben.

Bei READ-Aufträgen kopiert die zentrale Steuerung das Stabile Objekt zunächst aus dem ersten Objektspeicher in den bereitgestellten Pufferabschnitt. Der Adreßbereich läßt sich aus den Angaben ermitteln, die vor bzw. während der Aufragsübergabe in dem Parameterregister KP[2] (Objektlänge) vom Auftraggeber und im Transferregister KP[5] (Startadresse) von der Puffersteuerung eingetragen wurde. Genauso konnte bei WRITE-Aufträgen der Adreßbereich im Pufferspeicher von der zentralen Steuerung gefunden werden. Wird bei der Übertragung des Datenblocks ein Parityfehler oder ein Checksummenfehler festgestellt, so wird der Transfer wiederholt, als Quelle jedoch der zweite Objektspeicher benutzt. Ist dieser Transfer fehlerfrei abgelaufen, so ist der Auftrag für die zentrale Steuerung beendet. Ist aber auch beim zweiten Transfer ein Fehler erkannt worden, so muß ein VERIFY durchgeführt werden, wie dies bereits beim WRITE-Auftrag beschrieben wurde. Ist dies erfolgreich, so erhält der Auftraggeber ein konsistentes Objekt im Pufferbereich. Ansonsten wird ihm im Ergebnisregister der Fehler mitgeteilt.

Im Fall, daß ausschließlich in einer Kopie (in der ersten Kopie) Fehler entdeckt worden sind, wird der Zeitpunkt für die Korrektur zunächst verschoben. Sie erfolgt im Rahmen einer regelmäßig stattfindenden VERIFY-Durchführung, welche die zentrale Steuerung immer dann anstößt, wenn gerade kein Auftrag von einem Auftraggeber vorliegt. Dabei werden der Reihe nach alle Stabilen Objekte berücksichtigt. Jedoch haben neue Aufträge Vorrang vor dieser selbständig gestarteten VERIFY-Ausführung.

Bei den Aufträgen CREATE, DELETE und RELEASE werden nur Speicherverwaltungsaufgaben durchgeführt (siehe auch Abschnitt 6.5.2), bei denen Speicherbereiche belegt bzw. freigegeben werden. Dazu werden einige Einträge in “Objektbeschreibungen” vorgenommen. Diese sind wie die Stabilen Objekte doppelt in den beiden Objektspeichern abgelegt. Sie werden genauso regelmäßig überprüft und gegebenenfalls korrigiert. Allerdings sind sie nicht mit einer Checksumme gesichert, da sie nicht daraufhin überprüft werden müssen, ob sie vom Auftraggeber als “konformer Datenblock” übertragen worden sind. Die wichtigsten Angaben der Objektbeschreibung bleiben während der “Lebensdauer eines Stabilen Objekts” unverändert, wie ihre Größe, ihre Objektnummer, ihr Besitzer und ihre Lage innerhalb der Objektspeicher. Daher besteht bei ihnen nicht die Gefahr wie bei den Stabilen Objekten, daß während eines Updates sowohl im alten Zustand wie auch im neuen Zustand aufgetretene Fehler zu einem Verlust dieser Informationen führt.

Zusätzlich sind in der Objektbeschreibung noch einige Angaben gemacht, die sich während einer Auftragsbearbeitung ändern, wie der Zustand des Objekts während eines Updates. Diese Informationen machen es der zentralen Steuerung leichter, nach einem aufgetretenen Fehler den Zustand des Objektes zu erkennen. Sollte diese Information aber selbst von einem Fehler betroffen sein, so kann auf die Untersuchung der Checksummen der beiden Objektkopien zurück-

gegriffen werden.

Die Aufträge WRITE_BI und READ_BI sind sehr einfach auszuführen. Bei ihnen wird nur ein Transfer eines Wortes zwischen dem Parameterregister KP[3] und einem Eintrag in der Objektbeschreibung in jeder der beiden Objektspeicher durchgeführt.

Für die korrekte Zusammenarbeit mit der Puffersteuerung muß die zentrale Steuerung die “Sperr-Einträge”, welche die Puffersteuerung als Grundlage für die Berechtigung eines Auftraggeberzugriffs auf einzelne Register in der Kommunikationspage benutzt, am Ende ihrer Auftragsbearbeitung aktualisieren. Bei fast allen Aufträgen muß der Zugriff auf die Auftrags- und Parameterregister wieder erlaubt, der Zugriff auf das Transferregister wieder verboten werden. Nur bei einem READ-Auftrag muß die zentrale Steuerung nach der Übertragung des angeforderten Objekts in den Pufferspeicher stattdessen das Lesen vom Pufferspeicher erlauben. Die Zugriffssperre auf Auftrags- und Parameterregister bleibt bestehen. Deren Freigabe und die Sperre des Transferregisters wird von der Puffersteuerung nach der Übertragung des letzten Wortes an den Auftraggeber durchgeführt.

Bei jeder Auftragsbeendigung setzt die zentrale Steuerung die Einträge der Objektnummer und der Objektlänge des gerade bearbeiteten Auftrags in der Kommunikationspage auf einen ungültigen Wert (z.B. Null). Dadurch kann verhindert werden, daß ein Auftrag ohne Angabe der erforderlichen Parameter zur Ausführung gelangt. Ansonsten könnte z.B. “versehentlich” das Löschen des Objekts erfolgen, das im zuvor ausgeführten Auftrag angesprochen war.

Ausnahmen von dieser Vorgehensweise gibt es nur bei den Aufträgen CREATE und READ_-BI, bei denen als Rückgabewert die Objektnummer bzw. die Benutzerinformation in das entsprechende Register eingetragen wird, damit dies der Auftraggeber dort lesen kann. Da aber auch in diesen Fällen die Objektlänge auf den Wert Null gesetzt wird, kann bei einem nachfolgenden Auftrag schlimmstenfalls ein Objekt mit der Länge Null gelöscht oder neu beschrieben werden.

6.5.2 Struktur der Speicherverwaltung

Jeder der beiden Speicher des Lagerbereichs für die Stabilen Objekte ist organisatorisch in gleichgroße Pages aufgeteilt. Auf Basis dieser Einteilung läßt sich eine einfache Speicherverwaltung aufbauen. Jedes Objekt wird je nach seiner Größe in einer oder mehreren Pages untergebracht. Diese Pages müssen sich nicht in aufeinanderfolgenden Adreßbereichen befinden. Außerdem teilt kein Objekt seine Page mit einem anderen Objekt. Folglich können maximal so viele Stabile Objekte angelegt werden wie Pages zur Verfügung stehen.

In dieser Implementierung wurde die Page-Größe so gewählt, daß in einer Page genauso viele Worte stehen, wie es Pages gibt. Bei einer Pagegröße von N Worten gibt es somit N Pages. Die Kapazität eines Speichers ist also immer N^2 Worte. Dies läßt sich bei Verwendung der üblichen DRAM-Bausteine leicht erreichen. In der vorliegenden Implementierung enthält jeder Speicher

16 MWorte (Megaworte), das sind 64 MByte. Es gibt also 4096 Pages mit einer Größe von je 4096 Worten. Als Basisgröße wurde ein 32-bit-Wort genommen und kein Byte. Das erklärt sich daraus, daß dieser Stabile Speicher an einen Rechner angeschlossen ist, bei dem der Datentransport von Prozessor bis zu den Kommunikationsspeichern und damit auch bis zum Stabilen Speicher auf der Basis von 32-bit-Worten ausgeführt wird. Ferner werden voraussichtlich auch die abzulegenden Datenstrukturen aus 32-bit-Worten oder sogar aus 64-bit-Worten bestehen. Mit dem Verzicht auf Byte-Zugriffe wird zudem die Speicherzugriffs-Hardware einfacher.

Die zunächst willkürlich erscheinende Wahl der Pagegröße in Abhängigkeit von der Speichergröße wirkt sich bei der Speicherverwaltung positiv aus. Denn nun kann für die Verwaltungsdaten, die ebenfalls hier gespeichert werden, eine feste Anzahl von Pages vergeben werden. Diese Zahl ist unabhängig von der verwendeten Speicher- und Page-Größe. Ausschlaggebend ist nur noch, wieviele Angaben pro Objekt eingetragen werden sollen. Die Strategie zur Speicherverwaltung ist somit unabhängig von der Speichergröße geworden. Bei einem größeren Speicher erhöht sich mit der Pagegröße in gleichem Maße auch die Anzahl der maximal abzulegenden Objekte. Es müssen aber nicht mehr Pages zur Verwaltung bereitgestellt werden. Damit ist die Speicherverwaltung für alle möglichen Speichergrößen (mit: Speichergröße = 2^{2n} Worte) vorbereitet. Einige Einschränkungen gibt es jedoch für die Mindestgröße einer Page und damit auch für den Speicher. Da einige Informationen, die in den Kommunikationspages eingetragen sind, aus Schutzgründen auch im Lagerbereich des Stabilen Speichers abgelegt werden, muß (im vorliegenden Fall) die Page mindestens 1024 Worte groß sein.

Folgende Informationen sind für die Verwaltung eines Stabilen Objekts nötig:

- Besitzer dieses Objekts (= KP-Nummer),
- Größe dieses Objekts (in Worten),
- Benutzerinformation,
- einige Angabe, die etwas über den Zustand des Objekts aussagen, wie “Objekt belegt”, “Objekt in Ordnung”, “Objekt ist gerade in Bearbeitung einer WRITE-Operation”, ...
- Zeiger auf die vorherige und auf die nächste Page, die Teil dieses Objekts ist,
- Zeiger auf das vorherige und auf das nächste Objekt dieses Eigentümers.

Für diese Informationen wurden 4 Worte benutzt. Diese geringe Zahl an Worten wurde erreicht, da in den meisten 32-bit-Worten mehrere der aufgeführten Informationen untergebracht werden konnten. Daher reichen 4 Pages aus, um auch für die größtmögliche Zahl an Stabilen Objekten gerüstet zu sein.

Weiterhin gibt es für jede Kommunikationspage zwei Einträge, die das vergebene Paßwort und die Nummer des ersten Stabilen Objekts dieses Eigentümers tragen. Diese Informationen konnten auch innerhalb der eben genannten Pages abgelegt werden. Somit sind insgesamt nur 4 Pages für alle Verwaltungsinformationen nötig. Weitere Einzelheiten finden sich in [SEM93].

6.5.3 Schutz der zentralen Steuerung vor eigenem Ausfall

An den Stabilen Speicher werden sehr hohe Erwartungen bezüglich seiner fehlerfreien Arbeitsweise gestellt. Um seinem Namen gerecht werden zu können, muß er in der Lage sein, nicht nur Fehlersituationen bei den Auftraggebern zu erkennen, sondern auch eigenes Fehlverhalten ohne Verzögerung zu erkennen und es nicht zu einer fehlerhaften Fortführung seiner Arbeitsweise kommen zu lassen. Denn ein Fehler kann auch hier nicht ganz ausgeschlossen werden. Auf Grund der "Allmächtigkeit" dieser Steuerung wäre es sogar möglich, daß hier entstandene Fehler zu einer völligen Zerstörung der Stabilen Objekte führen könnten, ohne daß dies ein Auftraggeber rechtzeitig bemerken würde und Alarm schlagen könnte. Daher ist an dieser Stelle eine hochwirksame Fehlererkennung und -korrektur notwendig.

Voruntersuchungen zum Hardware-Entwurf des Stabilen Speichers haben gezeigt, daß innerhalb des Stabilen Speichers umfangreiche Arbeiten auszuführen sind [ROB93]. Schon allein der normale Ablauf zum Schreiben und Lesen und zur Verwaltung der Stabilen Objekte führt zu vielen unterschiedlichen Bearbeitungsschritten. Aber auch die möglichen Fehlersituationen, in die die Stabilen Objekte kommen können, müssen auf unterschiedliche Weise behandelt werden. Für deren Ausführung muß eine umfassend programmierbare Hardware zur Verfügung stehen. Ansätze, die eine Hardware ohne Verwendung von Prozessorchips vorsahen, hatten daher keine Aussicht auf Erfolg. So wurde die zentrale Steuerung mit Hilfe von Transputern T805 aufgebaut und in der Programmiersprache Occam-2 programmiert [OCC88] [TDS88] [TDD89]. Es wird hier vorausgesetzt, daß die zentrale Steuerung korrekt programmiert worden ist. Die Erfüllung dieser Forderung läßt sich nur sehr schwer nachweisen, da das implementierte Programmsystem umfangreich ist. Nur durch ausgedehnte Tests, die gezeigt haben, daß das Programm in der erwünschten Weise arbeitet, kann man sich von der korrekten Arbeitsweise überzeugen.

Für das unverzügliche Erkennen eines Fehlverhaltens und eine unterbrechungsfreie Fortführung der Bearbeitung wurde die zentrale Steuerung als TMR-System aufgebaut. Über Voter werden die produzierten Ausgaben jedes Knotens miteinander verglichen und nur das als Ergebnis weitergegeben, das von mindestens zwei Knoten erzeugt worden ist. Damit kann der Ausfall eines Knotens (bei einem einwandfrei arbeitenden Voter) sofort erkannt und maskiert werden. Eine sofort einsetzende Fehlerdiagnose und -behebung ist in diesem Fall nicht nötig. Es ist stattdessen möglich, dies erst zu einem späteren, geeigneteren Zeitpunkt durchzuführen. Solange noch zwei Knoten fehlerfrei arbeiten, kann zunächst ohne Unterbrechung und ohne Verlust der gegenseitigen Kontrolle der Prozessoren untereinander (wie bei einem Master-Checker-System) mit der Bearbeitung fortgefahren wird, bis der aktuelle Auftrag fertig bearbeitet worden ist.

Nach der Fertigstellung eines Auftrags wird versucht, den ausgefallenen Knoten wieder in die Arbeit zu integrieren. Dies wird über das Zurücksetzen (Reset) aller Knoten erreicht. Da zu diesem Zeitpunkt aber kein Stabiles Objekt mitten in der Bearbeitung ist, kann diese Fehlerbehebung ohne störenden Einfluß auf die gewünschte Bearbeitung im Stabilen Speicher durchge-

führt werden. Ein extra VERIFY ist daher nicht nötig.

Diese Vorgehensweise gelingt unter der Annahme, daß mit sehr hoher Wahrscheinlichkeit im Verlauf der restlichen Bearbeitungszeit für das aktuelle Stabile Objekt kein weiterer Fehler mehr auftritt. Ansonsten würde der Voter jede weitere Arbeit unmöglich machen. Dann bliebe nur noch das Zurücksetzen aller Knoten als Ausweg offen. Ein aus diesem Grund veranlaßter Abbruch der Bearbeitung hinterläßt i.a. ein unvollständig bearbeitetes Objekt. Nach dem Reset muß daher ein VERIFY erfolgen, wobei mit Hilfe der Checksumme ermittelt werden kann, welche Kopie korrekt ist. Ob dabei der alte oder der neue Zustand des Stabilen Objekts wiederhergestellt werden konnte, läßt sich aus der aktuellen Verwaltungsinformation zu diesem Objekt entnehmen. Hier ist noch vor dem Beginn der ersten Updatephase sicher (zweifach) eingetragen worden, welche Kopie als “Backup-Kopie” benutzt wird.

Beim Einsatz von TMR ist es notwendig, daß die drei Knoten synchron laufen. Nun kann es aber vorkommen, daß auf Grund unterschiedlicher Signallaufzeiten ein Knoten z.B. ein Steuersignal etwas früher oder etwas später bekommt als die beiden anderen Knoten. Als Folge davon ist es möglich, daß ein Knoten ab jetzt gegenüber den anderen Knoten um einen Takt versetzt läuft und somit auch seine folgenden Steuersignale um einen Takt verschoben ausgeben wird. Dabei besteht die Gefahr, daß dieser Knoten von einem Voter als asynchron angesehen wird. Ein Ausschluß dieses Knotens von der weiteren Berechnung (Anhalten des Knotens) würde aber in dieser Situation unnötig sein, da diese Asynchronität nicht auf einem Defekt des Knotens beruht. Dieses unnötige Ausscheiden des Knotens muß verhindert werden, da ansonsten für die weitere Auftragsabwicklung nur noch ein Master-Checker-Paar als Steuerung vorhanden wäre. Aus diesem Grund werden alle Zugriffe der Steuerung auf die angeschlossenen Speicherplatinen, das sind die Pufferplatine und die beiden Objektspeicher, über je einen synchronisierenden Voter geführt. Dieser kann Zugriffsaktionen, deren Start zeitlich leicht differieren, als “gleichzeitig eingetroffen” betrachten und diese Zugriffe zu einem gleichzeitigen Ende führen (siehe auch Abschnitt 6.7, Voter). Da diese Zugriffe über die bei der Bearbeitung aller Aktionen auf dem Stabilen Speicher sehr häufig und regelmäßig durchgeführt werden, erfolgt auch ständig diese Synchronitätskontrolle und, falls nötig, die Synchronisierung.

Die ständige Sorge um die Aufrechterhaltung der Synchronität ist auch bei anderen Rechnern mit einem TMR-Kern als ein zentraler Punkt erkannt worden, wie z.B. bei der “Integrity S2”. Dieser Rechner ist dafür ausgelegt, Unix basierenden Anwendungen eine sehr hohe Verfügbarkeit zu bieten. Hier laufen die einzelnen Knoten (wie im vorgestellten Stabilen Speicher) nicht unbedingt taktsynchron, sondern müssen sich spätestens nach einer eng begrenzten Zeit wieder synchronisieren [JEW91]. Besonders in den Situationen, in denen Ausnahmen (wie Fehlerereignisse) zu behandeln sind, wurde besonderes Augenmerk auf die Synchronität des TMR-Kerns gelegt.

Alle Schreibzugriffe der zentralen Steuerung auf den Pufferspeicher und auf die beiden Objektspeicher werden von einem Lesezugriff und einem Vergleich gefolgt, um die erfolgreiche Übertragung des Wortes zu kontrollieren. Gegebenenfalls werden bis zu drei Wiederholungen durch-

geführt, um transiente Fehler tolerieren zu können, die bei den angesprochenen Speicherplatinen aufgetreten sind. Schlagen jedoch alle Wiederholungsversuche fehl, wird ein Ausfall der angesprochenen Platine angenommen.

6.5.4 Reaktion der zentralen Steuerung bei Ausfall einer Speicherplatine

Der Ausfall einer ganzen Platine des Stabilen Speichers ist ein schwerwiegendes Fehlerereignis. Unter günstigen Umständen kann dieser Fehler toleriert werden. Allerdings darf dann i.a. kein weiterer Fehler aufgetreten sein. Ist eine Reparatur oder ein Zurücksetzen erfolgreich, so handelt es sich um einen vorübergehenden Ausfall, im anderen Fall um einen dauerhaften Schaden.

Ein vorübergehender Ausfall der zentralen Steuerung ist gleichzusetzen mit einer Asynchronität von mehr als einem Knoten der TMR-Steuerung und läßt sich durch Zurücksetzen der drei Prozessoren beheben. Hier muß zunächst ein VERIFY auf dem aktuell bearbeiteten Stabilen Objekt ausgeführt werden, bevor mit der normalen Arbeitsweise fortgefahren werden kann.

Bei vorübergehendem Ausfall eines Objektspeichers läßt sich nach Wiederinbetriebnahme dieses Speichers der Zustand des intakt gebliebenen Objektspeicher hineinkopieren. Dies ist erfolgreich, solange in keinem Objekt ein Fehler vorlag, und solange ein im Rahmen eines WRITE-Auftrags übertragener Datenblock vom Auftraggeber fehlerfrei im Pufferspeicher eingetragen wurde. Im letzten Fall ist wichtig, daß ein laufender Transfer eines Datenblocks (Pufferspeicher $\rightarrow$ Objektspeicher) nicht durch den Ausfall des anderen Objektspeichers unterbrochen wird.

Der vorübergehenden Ausfall des Pufferbereichs läßt sich ebenfalls tolerieren, da alle wichtigen Daten, die im Pufferbereich in den Kommunikationspages und in der VIP vorliegen, auch in den Objektspeichern gesichert sind. Nach Wiederinbetriebnahme des Pufferbereichs können diese Daten dort wieder eingetragen werden. Die Stabilen Objekte in den Objektspeichern sind während der ganzen Zeit durch das von der zentralen Steuerung ausgeführte VERIFY geschützt. Der Ausfall des Pufferbereichs wird im Gegensatz zu Ausfällen der anderen Platinen von den Auftraggebern sofort erkannt, sofern sie aktuell mit einer Kommunikation mit diesem Stabilen Speicher beschäftigt sind. Unterbrochene Aufträge müssen von ihnen dann neu erteilt werden.

Während die vorübergehenden Ausfälle (als einzig aufgetretener Fehler) zu verkraften sind, kann ein dauerhafter Ausfall zum vollständigen Verlust aller abgelegten Stabilen Objekte führen. Denn nur in dem Fall, daß zumindest der Pufferbereich, die zentrale Steuerung und einer der Objektspeicher in Ordnung sind, können die Auftraggeber ihre abgelegten Objekte noch erreichen. In dieser Situation erlaubt die zentrale Steuerung jedoch nur noch solche Aufträge, bei denen keine Objekte geschrieben oder erzeugt werden. Nur noch das Lesen (READ), das Löschen (DELETE) und das Überprüfen (VERIFY) der Stabilen Objekte wird dann durchgeführt. Außerdem kann der Auftrag zum Freigeben der Kommunikationspage (RELEASE) noch ausgeführt werden. Detailliertere Angaben dazu sind in [SEM93] zu finden. Somit ist es den Auftraggebern zumindest in den meisten Fällen noch möglich, ihre Objekte aus diesem (nicht mehr)

Stabilen Speicher herauszuholen und an einer anderen Stelle abzuspeichern.

6.5.5 Aufbau eines Knotens der zentralen Steuerung

Jeder Knoten ist mit einem lokalen zwischen Adressen und Daten gemultiplexten Bus ausgestattet. An ihm sitzen der Prozessor T805 und Programm- und Datenspeicher. Weiterhin gibt es in jedem Knoten einen Parity-Checker/-Generator, der je nach Transferrichtung das übertragene Wort (Adresse bzw. Datum) mit passenden Paritybits versieht bzw. die Parity überprüft.

Zugriffe innerhalb eines Knotens laufen alle ohne Synchronisierungsvorgänge zwischen den drei Knoten ab. Werden die drei Knoten aber mit dem gleichen Programm und mit den gleichen Daten (im Datenspeicher) versorgt und synchron gestartet, so laufen sie dennoch synchron. Der Grund liegt darin, daß alle knoteninternen Zugriffe mit einem festen Timing versehen sind. Dies ist möglich, da die internen Speicher alle aus SRAMs bestehen, bei denen Zugriffe nicht durch ein erforderliches Refresh (in unvorhersagbarer Weise) verlängert werden müssen.

Nach außen hin gibt es eine Schnittstelle zu einem Rückwandbus, an den auch alle Speicherplatinen angeschlossen sind. Weiterhin ist eine separate Synchronisierungseinheit für die drei Knoten (Austausch- oder Synchronisierungslogik) über diesen Bus erreichbar. Erst die Zugriffe, die über eine dieser Schnittstellen führen, werden über Voter geleitet und dabei leichte zeitliche Differenzen auf die oben beschriebene Weise synchronisiert.

In Abb. 36 ist der Aufbau eines Knotens dargestellt.

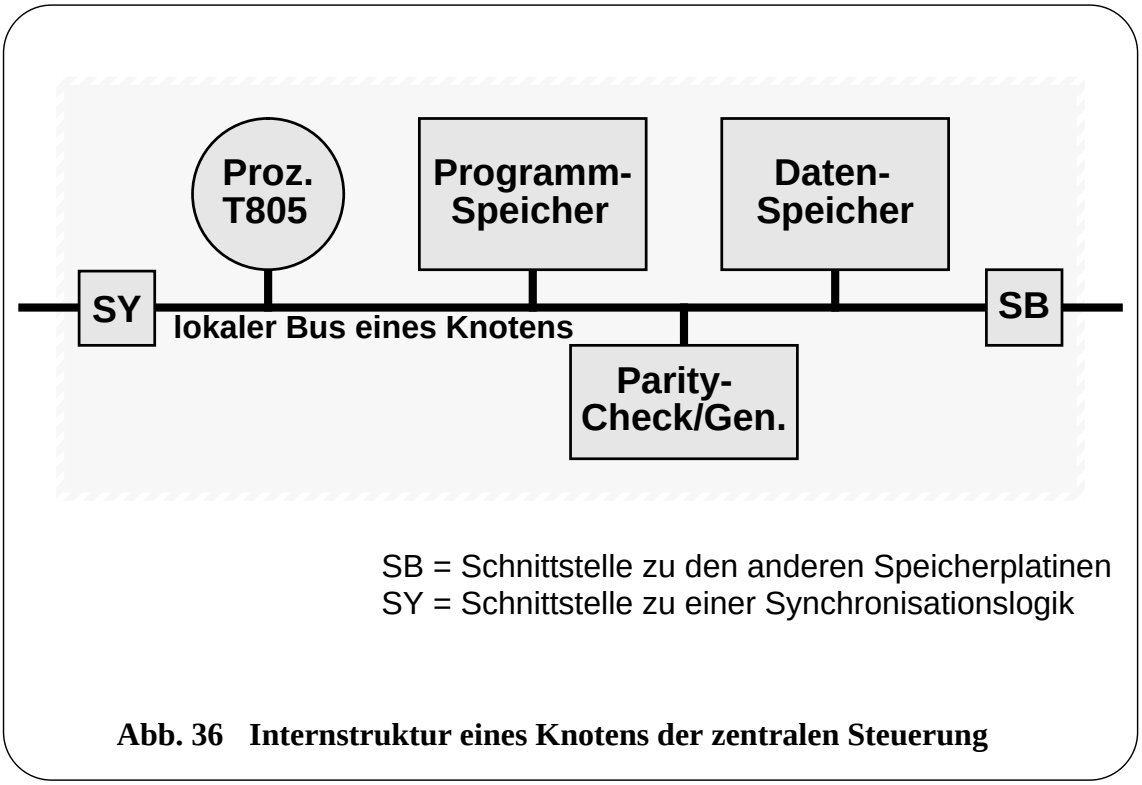


Abb. 36 Internstruktur eines Knotens der zentralen Steuerung

6.5.6 Die gemeinsame Synchronisations- und Austausch-Einheit der Knoten

Ein Problem kann dann auftauchen, wenn der Programmablauf in den Knoten eine Programmverzweigung erreicht und die Möglichkeit besteht, daß sich die Knoten unterschiedlich entscheiden würden. Dann würden sie sofort ihre Synchronität verlieren. Dies muß verhindert werden.

Entscheidungsgrundlage für Programmverzweigungen sind immer knoteninterne Daten. Sie können aus Daten stammen, die sich nur innerhalb des Knotens befinden (z.B. Schleifenzähler) oder aus temporär gebildete Daten, die aus vorher von anderen Speicherplatinen gelesenen Daten abgeleitet wurden. Ein Fehler, der sich dabei nur in einem Knoten ereignet, kann dazu führen, daß dieser Knoten den Programmablauf in eine andere Richtung lenken wird als die anderen. Zunächst wird das nicht erkannt. Erst beim nächsten Zugriff über einen Voter kann die Auswirkung des Fehlers bemerkt werden. Wegen der 2-aus-3-Entscheidung am Voter führt dies außerhalb des Knotens nicht zu einer fehlerhaften Aktion.

Wird der Knoten, der nicht mehr dasselbe tut wie die beiden anderen Knoten, vom Voter als asynchron erkannt, so meldet der Voter dies an die zentrale Steuerung zurück. Eine Stopp-Logik hält dann den entsprechenden Knoten an. Nach Abschluß der Bearbeitung eines Auftrags wird wieder versucht, diesen Knoten wieder in die gemeinsame Arbeit zu integrieren.

Die drei Knoten der TMR-Schaltung arbeiten intern ohne gegenseitige Beeinflussung. Erst Zugriffe, die über ihre Grenzen hinausgehen wie z.B. der Zugriff auf eine der Speicherplatinen, werden über die Voter geleitet und gegebenenfalls synchronisiert.

Bei lokalen Entscheidungen der drei Knoten besteht die Gefahr, daß sie auch dann, wenn sie die gleichen Daten nach gleichen Kriterien beurteilen, dennoch auf Grund eines lokalen Fehlers zu unterschiedlichen Ergebnissen kommen können. Dies kann beispielsweise passieren, wenn sie mit einem gemeinsamen Zugriff auf den Objektspeicher (über Voter geführt) einen Wert holen, dieser aber bei einem der drei Knoten auf Grund einer Störung auf einer Übertragungsstrecke fehlerhaft ankommt.

Folgt daraus dann ein unterschiedlicher Programmablauf, so führt dies i.a. bereits nach kurzer Zeit dazu, daß ein Knoten angehalten wird und nur noch zwei Knoten synchron weiterarbeiten. Hier besteht die große Gefahr, daß ein weiterer Fehler zu einem totalen Ausfall der zentralen Steuerung führen kann. Dies ist bereits geschildert worden.

Um zu vermeiden, daß bereits ein transienter Fehler, der auf ein einzelnes übertragenes Wort wirkt, zu einem so kritischen Zustand führt, ist auf der Steuerplatine eine Einrichtung geschaffen worden, mit deren Hilfe eine gemeinsame Entscheidung aller drei Knoten herbeigeführt werden kann. Selbst wenn einer der drei Knoten fehlerhafte Daten empfangen hatte und daher lokal eine andere Entscheidung getroffen hätte als die beiden anderen, kann er überstimmt werden und fährt mit den gleichen Programmschritten fort wie die beiden anderen.

Diese Einheit kann auch zur Synchronisation der drei Knoten untereinander eingesetzt werden, da sie ähnlich wie die Voter bei leichten zeitlichen Differenzen der drei Zugreifer beim Start der Aktion ein gleichzeitiges Zugriffsende erzeugt. Da hierbei kein Zugriff auf eine andere Platine nötig ist, die möglicherweise auf Grund einer Störung gerade nicht erreichbar ist, kann über diese Synchronisations- und Austausch-Einheit im Bedarfsfall eine Synchronisation vorgenommen werden (siehe Abb. 37).

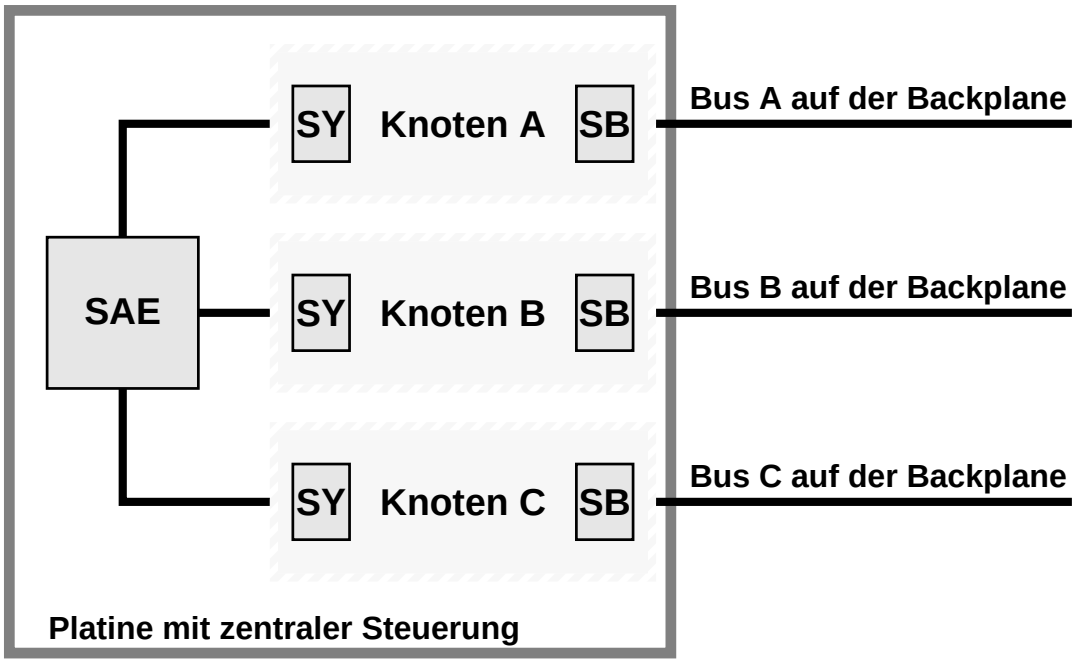


Abb. 37 zentrale Steuerung als TMR mit Synchronisations-Einheit

6.6 Der Lagerbereich

Im Lagerbereich sind zwei Speicher implementiert, in denen die Stabilen Objekte abgelegt werden. Je eine Kopie dieser Objekte wird in den beiden Speichern abgelegt. Dann kann ein Ausfall einer Platine nur eine der beiden Kopien eines Stabilen Objekts betreffen.

Zugang zu diesen Speichern hat nur die zentrale Steuerung des Stabilen Speichers. Dabei greifen die drei Knoten über einen Voter auf den Speicher zu (siehe Abb. 38). Die gewünschte Aktion (Lesen bzw. Schreiben bei der angegebenen Adresse) wird nur ausgeführt, wenn der Voter bei den übertragenen Adressen (und Daten) mindestens zwei gleiche Zugriffswünsche festgestellt worden sind.

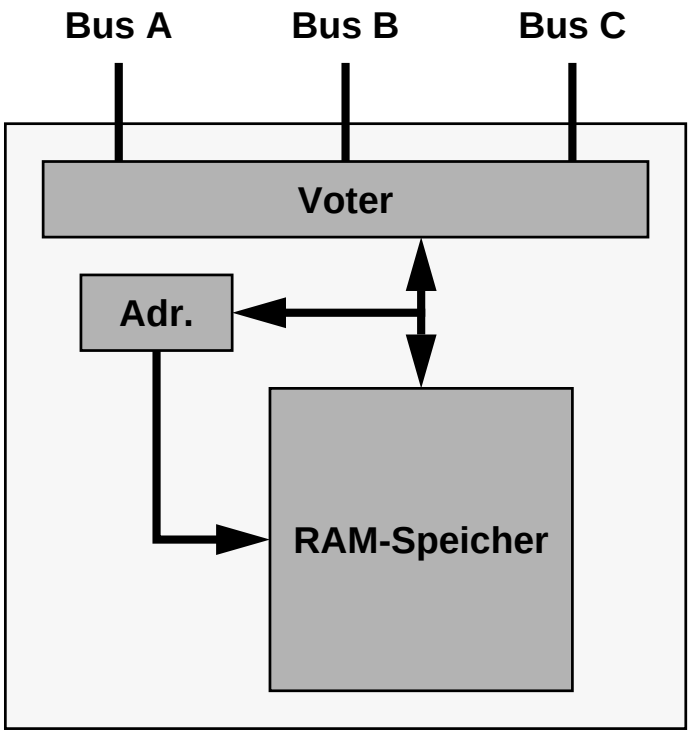


Abb. 38 Speicherplatine des Lagerbereichs

Bei der Implementierung des Speichers mußte zwischen dem Einsatz einer Parity-Absicherung und einer ECC-Codierung (Error Correcting Code) entschieden werden.

Für den Einsatz von ECC spricht, daß es hier mehr Möglichkeiten gibt, aufgetretene Einzelbitfehler zu tolerieren. Bei einer Wortbreite von 32 Bit reichen 7 weitere Bits aus, um eine Kodierung mit einem Hammingabstand von 4 zu finden. Damit lassen sich alle Einzelbitfehler korrigieren und alle 2-Bit-Fehler sicher erkennen [HAM86] [PEW72]. Einzelbitfehler (durch Umwelteinflüsse) sind bei den abgespeicherten Daten gegenüber den Mehrbitfehlern am ehesten zu erwarten.

Dennoch könnte auch unter der Annahme, daß ein abgespeichertes Wort höchstens mit einem Einzelbitfehler behaftet sein kann, nicht auf die Verwendung einer zweifachen Speicherung verzichtet werden. Denn bei der Ausführung einer WRITE-Operation könnte man nicht mehr den alten Zustand herstellen, wenn dies auf Grund eines anderen Fehlers (z.B. Checksummenfehler wegen Übertragung eines falschen Wortes, oder wegen der Transferstörung durch einen anderen fehlerhaft arbeitenden Prozessor) notwendig sein sollte. In diesem Fall ist man auf die Existenz einer zweiten, in sich konsistenten Kopie angewiesen. Auch der Ausfall eines Speichers ließe sich mit nur diesem einen Speicher nicht mehr tolerieren.

Die Verwendung von Parity-Absicherung kann dagegen voll von der geforderten doppelten

Speicherung der Stabilen Objekte profitieren. Im Fall eines Bitfehlers (auch eines Mehrbitfehlers) in einem Wort kann durch Kopieren aus dem entsprechenden Wort aus der zweiten Kopie der Fehler behoben werden, solange nicht auch dieses von einem Fehler betroffen ist. Außerdem können die Daten ohne Transformation (ECC ↔ Parity) zwischen dem Stabilen Speicher und dem Anschluß an das Motorola-System transportiert werden. Die Transformations-Hardware kann somit eingespart werden. Es hatte sich auch gezeigt, daß im Gegensatz zu Parity-Checkern und -Generatoren kaum Hardware für ECC-Generierung bzw. -Check angeboten wird. Eine eigene Implementierung wäre sehr aufwendig gewesen. Aus diesen Gründen wurde die Parity-Absicherung auch innerhalb des Stabilen Speichers im Lagerbereich verwendet.

Jeder der implementierten Speicherplatinen ist mit 4 MBit-Chips aufgebaut und hat eine Gesamtkapazität von 64 MByte. Das sind 16 MWorte. Intern ist der Speicher in 36 x 4 x 4 MBit organisiert. Die Wortbreite von 36 Bit erklärt sich aus der Abspeicherung jedes 32-bit-Wortes mit 4 Paritybits. In der Tiefe gibt es somit 4 Blöcke zu je 4 MWorten. Das bedeutet, daß bei Ausfall eines Speicherchips ein Viertel des Gesamtspeichers betroffen ist. Die technologische Weiterentwicklung der Speicherchips zu immer höherer Kapazität bringt zwar Vorteile bei der Implementierung eines großen Speicherbereichs mit sich; bezüglich der Situation bei Ausfall eines Speicherchips ergibt sich aber eher eine Verschlechterung. Bei Ausfall eines Speicherchips ist dann nämlich ein größerer Teil der abgelegten Daten betroffen. Daher sollte bei weiter wachsender Speicherkapazität pro Chip auch die Kapazität des Stabilen Speichers entsprechend vergrößert werden und nicht zum Einsatz deutlich weniger Speicherchips übergegangen werden.

Eine Möglichkeit, die Folgen des Ausfalls eines Speicherchips auf eine kleinere Zahl von Worten zu beschränken, wäre, jedes Wort nicht in 32 + 4 Chips zu verteilen sondern beispielsweise nur in 8 + 1 Chips. In diesem Fall würde sich der Ausfall eines Speicherchips im Vergleich zur ersten Lösung nur auf ein Viertel der Worte auswirken können. Diese Vorgehensweise hätte aber einen deutlichen Nachteil beim Abspeichern und beim Lesen jedes Wortes. In diesem Fall müßten intern vier Zugriffe statt einem einzigen durchgeführt werden, bis das Wort abgespeichert bzw. bereitgestanden wäre. Bei DRAM-Bausteinen macht das schon eine spürbare Verlängerung der gesamten Zugriffszeit aus. Außerdem würde die interne Ablaufkontrolle aufwendiger werden. Ein Ausfall eines Speicherchips kommt dagegen so selten vor, daß auch bei der Verwendung von 144 Chips (36 x 4) pro Speicherplatine für diesen Stabilen Speicher erst nach mehreren Monaten mit einem Ausfall zu rechnen ist. Zum Vergleich: Das Multiprozessorsystem MEMSY enthält in 20 Kommunikationsspeichern 720 Speicherchips (20 x 36). Im Verlauf von über einem Jahr ist bislang noch kein Speicherchip ausgefallen.

Bei Abwägung dieser Gründe wurde eine Entscheidung zugunsten der ersten Lösung (nur ein Speicherzugriff pro Wort) getroffen.

Im Lagerbereich werden neben den Stabilen Objekten auch die Daten abgelegt, die für die Verwaltung dieser Objekte notwendig sind. Das sind Angaben über den Eigentümer (Kommunikationspage-Nummer), die Länge des Objekts und Angaben über den aktuellen Zustand des Ob-

jekts (gerade unbenutzt, oder: 1.Kopie bei Update, und anderes). Diese Informationen werden durch den gleichen Mechanismus geschützt wie die Stabilen Objekte.

Weiterhin sind auch die Daten abgelegt, die aus den Einträgen aus den Kommunikationspages stammen. Somit kann auch nach einem vorübergehenden Ausfall des Pufferbereichs der Inhalt der Kommunikationspages wiederhergestellt werden. Das ist sehr wichtig für die Erreichbarkeit der abgelegten Stabilen Objekte, denn sonst könnten die Auftraggeber keine weiteren Aufträge mehr an den Stabilen Speicher stellen und auch die gesicherten Daten nicht mehr nutzen. Da sich der Ausfall des Pufferbereichs auch während der Ausführung eines Auftrags ereignen kann, muß diese Situation durch ein VERIFY bereinigt werden, wobei mit Hilfe der Checksumme ermittelt werden kann, welche der beiden abgelegten Kopien eines Stabilen Objekts in diesem Moment korrekt ist.

6.7 Die Voter

Bei einer TMR-Schaltung (Triple Modular Redundancy) spielt der Voter eine entscheidende Rolle. Er fällt die Entscheidung über das, was von der Dreifacheinheit nach außen gegeben wird. Aus den drei Inputs bildet er eine Mehrheitsentscheidung. Für eine korrekte Funktionsweise müssen mindestens zwei Inputs dieselben Werte besitzen (siehe Abb. 39).

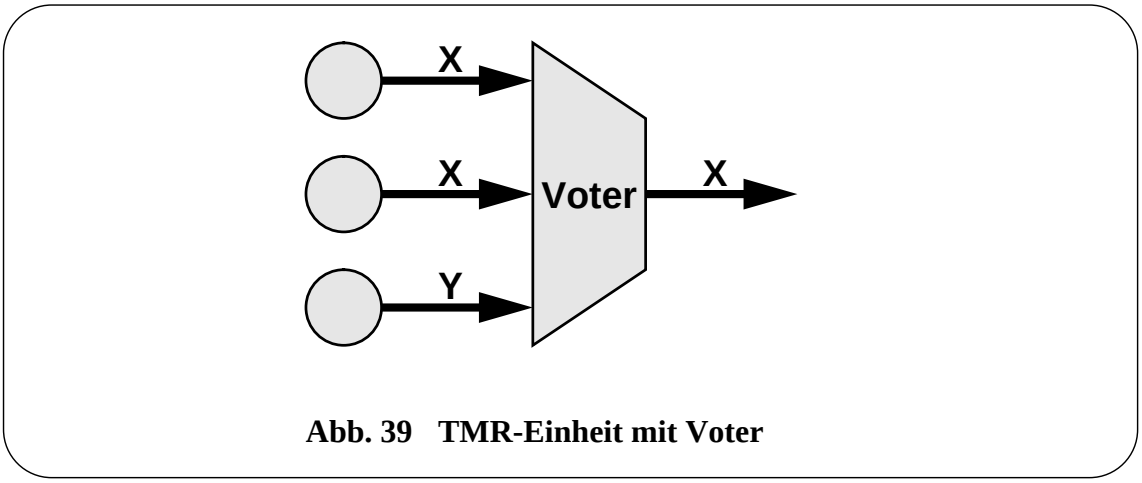


Abb. 39 TMR-Einheit mit Voter

Oft sind die Voter als reine Kombinatorik aufgebaut. Um Störungen auf den Übertragungsleitungen leichter ausblenden zu können, wurde im hier vorliegenden Fall eine andere Vorgehensweise ausgewählt.

Die Voter liegen auf den Platinen, die den Lagerspeicher (auch “Objektspeicher” genannt) implementiert haben, und auf der Platine, die den Pufferbereich beherbergt. Die Zugriffe der zentralen Steuerung stammen von einer weiteren Platine. Da die Voter somit erst unmittelbar vor einem Speicher liegen, lassen sich auch Fehler maskieren, die sich auf der Übertragungsstrecke (von Platine zu Platine) zwischen der zentralen Steuerung und dem Speicher ereignen.

Die Voter sind intern nicht dafür eingerichtet, gleichzeitig zwei oder drei Vergleiche von je zwei 36-bit-Zahlen (32 Bit Information und 4 Bit Parity) und einiger Steuersignale durchzuführen. Die dafür notwendige Hardware wäre zu umfangreich geworden und hätte nicht mehr auf der Platine Platz gefunden. Der implementierte Voter ist nur in der Lage, einen Vergleich zweier 36-bit-Zahlen in einem Takt auszuführen. Als Hardware ist dann nur ein üblicher Vergleicher, ein Register zur Zwischenspeicherung des einen Vergleichwertes und eine Steuerung notwendig, die immer zwei Wertepaare zum Vergleich zusammenstellt (siehe Abb. 40).

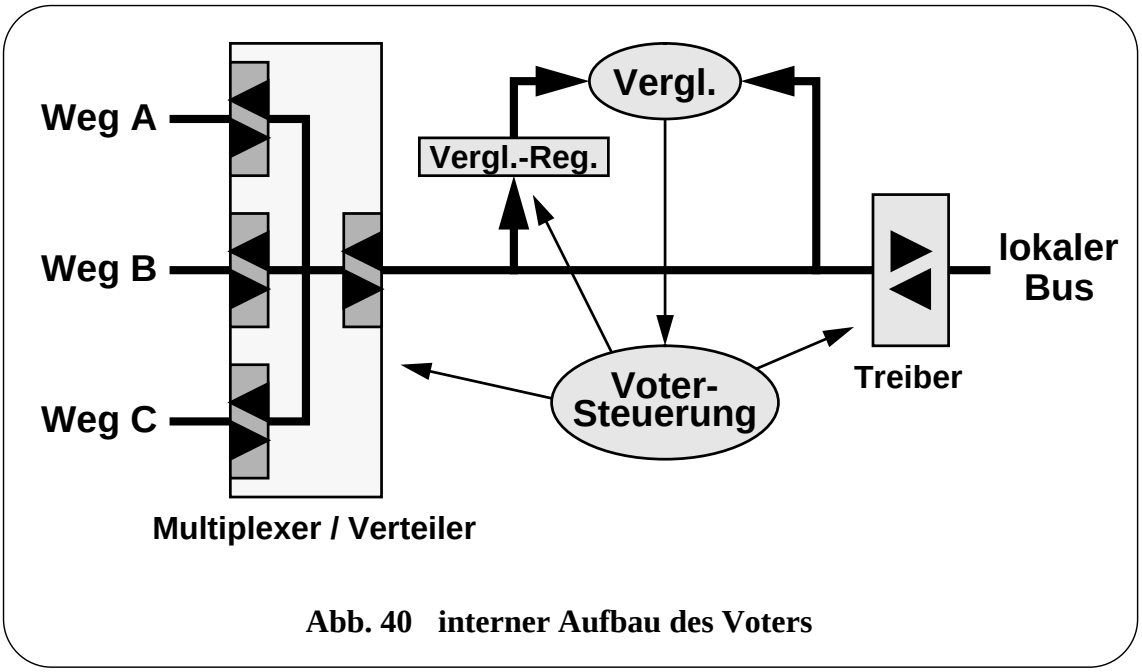


Abb. 40 interner Aufbau des Voters

Bei den Steuersignalen wird etwas anders vorgegangen. Sie werden getrennt registriert und beeinflussen die Votersteuerung. Dabei kann festgestellt werden, ob drei gleichartige Zugriffswünsche (lesend bzw. schreibend) vorliegen oder nicht. Sind drei gleichartige Zugriffe gewünscht, so werden hintereinander die Vergleiche der Zahlenpaare B und A, anschließend von B und C und, wenn noch nötig, der Vergleich zwischen A und C durchgeführt. Der dritte Vergleich ist nur dann erforderlich, wenn die ersten beiden Vergleiche nicht zu einem positiven Ergebnis gekommen sind. Am Ende des Votings können dann 3 verschiedene Situationen vorliegen:

- alle drei Werte sind gleich, oder
- nur zwei Werte sind identisch, einer (A oder B oder C) unterscheidet sich von ihnen, oder
- alle drei Werte sind unterschiedlich.

Das erste Ergebnis ist sicher das erwünschte. Die zweite Möglichkeit führt, obwohl einer der Zugreifer mit einem falschen Wert arbeitet, noch zu einer richtigen Aktion (Maskierung des einen fehlerhaften Zugriffs). Dagegen zeigt das dritte Ergebnis einen nicht zu maskierenden Fehler an.

Im gesamten Stabilen Speicher existiert auf Grund des Umfangs und der damit verbundenen Störanfälligkeit kein zentraler Takt. Daher muß zwischen den einzelnen Platinen synchronisiert werden. Die Inputs werden mit eigenem Takt empfangen und zwischengespeichert. Aus Sicht der Voter kann es dabei zu synchronen Zugriffen und zu asynchronen Zugriffen der drei Knoten des TMR-Systems kommen.

6.7.1 Synchroner Zugriff über die Voter

Solange die drei Knoten völlig synchron laufen, treffen gleichzeitig gleichartige Zugriffswünsche am Speicher ein. In diesem Fall müssen die Adressen und, bei Schreiboperationen, auch die Daten miteinander verglichen werden. In der vorliegenden Hardware handelt es sich um einen zwischen Adressen und Daten gemultiplexten Bus. Da diese bezüglich jedes Knotens über denselben Bus übertragen werden, müssen sie hintereinander verglichen werden. Dazu wird zunächst die Adresse auf jedem der drei Wege am Eingang des Voters in einem Latch eingefangen. Bei einem Schreibzugriff treibt anschließend jeder der drei Prozessoren das zu übertragende Wort.

Bei Lesezugriffen wird nur über die angelegten Adressen gevotet. Das gelesene Wort wird dann im Broadcast gleichzeitig über alle Wege (A,B,C) zurücktransportiert. Dabei wird von der Eigenschaft des verwendeten Treiberbausteins Gebrauch gemacht, daß ein Input gleichzeitig auf mehrere Ausgänge getrieben werden kann [AMD29]. Von dieser Vorgehensweise wird nur abgewichen, wenn der Voter zuvor (bei der Adresse) bemerkt hatte, daß nicht mehr über alle Wege ein gleichartiger Zugriffswunsch (lesend bzw. schreibend) vorlag. Dann läuft ein Prozessor aus der TMR-Schaltung bereits asynchron. Der Broadcast wird dann nur noch auf den beiden noch funktionstüchtig gebliebenen Wegen durchgeführt. Dieser asynchrone Fall wird weiter unten behandelt.

Im synchronen Fall wird, nachdem beim Voten der Adressen und gegebenenfalls beim Voten der angelegten Daten der erste oder zweite Fall der oben beschriebenen Situationen ermittelt wurde, der angegebene Zugriff auf den hinter dem Voter (und dem Treiber) liegenden Speicher auch durchgeführt. Bei der dritten Situation wird dagegen keine Aktion im Speicher gestartet.

In allen drei Fällen erzeugt der Voter eine Statusinformation, die an alle drei Knoten zurückgeleitet wird. Aus dieser geht hervor, zu welchem Ergebnis der Voter beim Vergleich gekommen ist. Die oben angegebenen Voterergebnisse werden als folgende Statusinformationen den Knoten mitgeteilt:

- 0 0 0 0 : alle Knoten arbeiten synchron und liefern identische Werte.
- 0 0 0 1 : alle Knoten arbeiten synchron,
  aber Knoten A hat gegenüber B und C unterschiedliche Werte.
- 0 0 1 0 : alle Knoten arbeiten synchron,
  aber Knoten B hat gegenüber A und C unterschiedliche Werte.

- 0 1 0 0 : alle Knoten arbeiten synchron,
  aber Knoten C hat gegenüber A und B unterschiedliche Werte.
- 0 1 1 1 : alle Knoten arbeiten synchron,
  aber alle Knoten haben in der Adreß- oder Daten-Phase
  unterschiedliche Werte geliefert.
- 0 0 1 1 : alle Knoten arbeiten synchron,
  aber die Knoten A und B hatten (bei einer Schreiboperation) in der Adreß- bzw. Daten-Phase je einmal einen unterschiedlichen Wert zu C geliefert.
- 0 1 0 1 : alle Knoten arbeiten synchron,
  aber die Knoten A und C hatten (bei einer Schreiboperation) in der Adreß- bzw. Daten-Phase je einmal einen unterschiedlichen Wert zu B geliefert.
- 0 1 1 0 : alle Knoten arbeiten synchron,
  aber die Knoten B und C hatten (bei einer Schreiboperation) in der Adreß- bzw. Daten-Phase je einmal einen unterschiedlichen Wert zu A geliefert.

Die erzeugten Werte 0000 und 0111 entsprechen genau dem ersten bzw. dritten oben beschriebenen Fall. Die anderen sechs Ergebnisse (0001 - 0110) geben wieder, was der Voter bei einzelnen Fehlern entdeckt hat. Diese unterteilen sich in solche, bei denen Übertragungs- oder Zugriffsfehler nur an einem Knoten auftreten (0001, 0010, 0100), und in solche, bei denen sich in der Adreß- und in der Datenphase beim Schreiben je einmal bei verschiedenen Knoten ein Fehler ereignet hat (0011, 0101, 0110). Da in den letzten Fällen die zwei Fehler zu unterschiedlichen Zeitpunkten auftraten, konnten dennoch immer zwei gleiche Werte für die übertragene Adresse bzw. für das übertragene Wort gefunden werden. Somit kann in allen diesen sechs Fällen der Fehler durch den Voter maskiert werden.

6.7.2 Asynchroner Zugriff über die Voter

Bislang wurden nur drei synchron arbeitende Knoten betrachtet. Dies bedeutet, daß sie alle noch die gleichen Folgen von Zugriffen zur gleichen Zeit ausführen. Die dabei entstandenen und entdeckten Fehler sind dabei von dem Typ eines Übertragungsfehlers. Es kann aber auch ein Fehler auftreten, der einen Knoten in einen anderen Programmzweig führt als die anderen beiden Knoten. Ursache dafür könnte z.B. ein Zugriffsfehler beim Lesen des nächsten Befehls sein. Da solch ein Zugriff rein lokaler Natur ist und nicht über die Voter geführt wird, wird dies auch nicht sofort bemerkt. Erst beim nächsten Zugriff, der über die Knotengrenze hinausgeht, kann ein Voter dies bemerken. Dann können drei verschiedene Situationen vorliegen:

Im ersten Fall wollen die beiden anderen noch korrekt arbeitenden Knoten auch gerade einen gleichartigen (lesenden bzw. schreibenden) Zugriff wie der des fehlerhaften Knotens über diesen Voter durchführen, aber mit einer anderen Adresse und/oder mit einem anderen zu schreibenden Wort. Dann sieht das Bild für den Voter so aus, als würde zwar der eine Knoten andere Werte anlegen, der Zugriffszeitpunkt wäre aber der gleiche. In diesem Fall erkennt der Voter

wie bereits beschrieben auf “Synchron, ein Knoten falsch”.

Im zweiten Fall treffen ebenfalls zur gleichen Zeit drei Zugriffswünsche ein, jedoch will der fehlerhaft arbeitende Knoten einen andersartigen Zugriff ausführen als die beiden anderen Knoten. Diese Situation (z.B. 2 x Schreiben, 1 x Lesen) erkennt der Voter bereits als Asynchronität des einen Knotens und antwortet in seiner Statusinformation mit “Asynchron, ein Knoten falsch”.

Der dritte Fall unterscheidet sich von allen bisherigen dadurch, daß die Zugreifer zu unterschiedlichen Zeitpunkten einen Zugriff über den Voter durchführen wollen. Für einen erfolgreichen Zugriff müssen dabei mindestens zwei Zugreifer dieselbe Aktion durchführen wollen. Ein einzeln auftretender Zugriffswunsch kann daher als fehlerhaft interpretiert werden. Aus diesem Grund muß der Voter erst dann reagieren, wenn mindestens zwei Zugriffswünsche vorliegen.

Um alle diese synchron und asynchron auftretenden Zugriffswünsche in der gewünschten Weise behandeln zu können, wurden die Voter so implementiert, daß jeder Voter erst dann reagiert, wenn mindestens zwei gleichartige Zugriffswünsche vorliegen. Um dabei die Gefahr zu bannen, daß ein nur geringfügig aus dem Takt gelaufener “Nachzügler” (siehe Abschnitt 6.5.3, Schutz der zentralen Steuerung) unnötigerweise aus der weiteren Bearbeitung ausgeschlossen wird, wartet der Voter noch eine festgelegte kurze Zeit (einstellbar bis zu 12 Takten, entspricht 375 nsec), ob nicht auch der dritte Knoten an diesem Transfer beteiligt sein möchte. Wenn nicht, so wird in der Statusmitteilung des Voters vermerkt, daß die Zugriffe der Knoten nicht synchron waren, und auch, welcher Knoten asynchron war (“Asynchron, ein Knoten falsch”). Die in diesen Situationen zurückgelieferten Statusinformationen sind:

- 1 0 0 1 : Knoten A arbeitet nicht mehr synchron zu den beiden anderen.
- 1 0 1 0 : Knoten B arbeitet nicht mehr synchron zu den beiden anderen.
- 1 1 0 0 : Knoten C arbeitet nicht mehr synchron zu den beiden anderen.

Eine Stop-Logik bei den Knoten der zentralen Steuerung reagiert in diesen Fällen so, daß der entsprechende Knoten angehalten wird. Die TMR-Einheit kann zwar in dieser Situationen noch alle Aufgaben durchführen; für sie besteht aber ab diesem Zeitpunkt die Gefahr, daß der Ausfall eines weiteren Knotens zum völligen Zusammenbruch der Aktivität der zentralen Steuerung des Stabilen Speichers führt. Denn dann kann der Voter keine Mehrheitsentscheidung mehr treffen. In diesen Fällen liefert der Voter als Statusmitteilung folgenden Wert zurück:

- 1 1 1 1 : alle Knoten arbeiten nicht mehr synchron.

Für die zentrale Steuerung des Stabilen Speichers bleibt in dieser Situation nichts mehr übrig, als die aktuelle Arbeit abzubrechen und zu versuchen, die Knoten wieder in einen Zustand zu versetzen, in dem sie alle, oder zumindest zwei der drei Knoten, wieder synchron arbeiten. Solange die Ursachen für den Ausfall eines Knotens transienter Natur waren, wird der Neustart gelingen, falls sich nicht gerade in dieser Situation ein weiterer Fehler ereignet. Lag jedoch ein

einziger permanenter Fehler vor, so kann es sein, daß nur zwei Knoten wieder synchron in Gang kommen. Dabei bleibt die zentrale Steuerung aber sehr anfällig für weitere auftretende Fehler. Lagen aber mehrere permanente Fehler bei verschiedenen Knoten vor, so wird die zentrale Steuerung nicht mehr in der Lage sein, neu zu starten. Die in diesem Stabilen Speicher abgelegten Daten wären nach dieser Häufung von Fehlern verloren, wenn keine Reparatur der zentralen Steuerung erfolgt.

6.7.3 Alternative Arbeitsweise für Testzwecke

Neben der oben dargestellten Arbeitsweise wurde für Testzwecke noch eine weitere Möglichkeit zur Reaktion der Voters implementiert. In der Entwicklungsphase des Stabilen Speichers war es notwendig, Zugriffe von einem einzelnen Knoten über die Voter durchführen zu können. In der normalen Arbeitsweise würde dies dazu führen, daß der Voter nur einen einzelnen Zugriffswunsch erkennt und nicht reagiert. (Das ist die Vorkehrung, um asynchron laufenden Knoten keinen Zugriff auf die hinter den Votern liegenden Speicher zu gestatten.) Um die Testsituation zu erkennen, verfügt jeder Voter über einen weiteren Input, der ihm angibt, welche Knoten zugreifen werden. Dazu wurden vier Möglichkeiten eingerichtet:

- Normalbetrieb: über alle Knoten soll gevotet werden.
- Testbetrieb A: nur Zugriffe von Knoten A werden durchgeführt.
- Testbetrieb B: nur Zugriffe von Knoten B werden durchgeführt.
- Testbetrieb C: nur Zugriffe von Knoten C werden durchgeführt.

Die Reaktionen, die der Voter in der Statusinformation darstellt, sind im Testbetrieb:

- 1 1 1 0 : Zugriff von Knoten A ist durchgeführt worden.
- 1 1 0 1 : Zugriff von Knoten B ist durchgeführt worden.
- 1 0 1 1 : Zugriff von Knoten C ist durchgeführt worden.

Im Testbetrieb kann der Voter nicht zwischen verschieden Inputs vergleichen. So werden Adressen und Daten nur vom angegebenen Knoten zum Speicher transportiert bzw. umgekehrt. Nur die Verzögerung zum Warten auf andere Knoten ist in diesem Fall aktiv. Da aber kein weiterer Zugreifer beachtet wird, wird immer die eingestellte Zeitspanne (0, 4, 8 oder 12 Takte; dies entspricht bei 32 MHz etwa 0, 125, 250, 375 nsec) lang gewartet.

Diese Voterergebnisse vervollständigen die Liste der zurückgelieferten Statusinformationen. Damit sind alle Kombinationen mit 4 Bits bis auf “1000” mit einer Bedeutung belegt. Der Wert “1000” wird aber nicht von den Votern ausgegeben.

6.8 Zusammenfassung der tolerierbaren Fehler

Der hier vorgestellte Stabile Speicher ist in der Lage, verschieden Hardware-Fehler sowie einige Software-Fehler zu tolerieren, die in den Prozessoren des Host-Systems und auf den Übertragungswegen zum Stabilen Speicher auftreten können. Bei der Hardware sind dies:

- Einbit-Fehler auf den Übertragungswegen (Parity);
- Übertragung über falsch geschalteten Weg (Paßwort, Checksumme);
- Ausfall einer der beteiligten Einheit beim Zugriff auf den Stabilen Speicher während eines einzelnen Wort-Transfers (Timeout).

Die Tolerierung von Störungen, die sich auf den Übertragungswegen ereignen, wird dagegen mit Maßnahmen zur Fehlerverhinderung betrieben (Differenzübertragung der Steuersignale).

Software-Fehler des Auftraggebers können nur sehr eingeschränkt als solche vom Stabilen Speicher erkannt werden. Diese beziehen sich auf die Auftragserteilung:

- Verletzung des Protokolls, das den Ablauf der Auftragserteilung und die Datenübertragung betrifft (Sperre der Auftrags- und Parameterregister, Sperre des Pufferbereichs);
- Auftragsvergabe ohne Angabe von Parametern (Löschen der Parameterregister am Ende einer Auftragsbearbeitung);
- Übertragung eines Datenblocks mit inkonsistentem Inhalt (Checksumme).

Andere Software-Fehler haben gegenüber dem Stabilen Speicher die gleiche Erscheinungsform wie Hardware-Fehler, z.B. das (versuchte) Schreiben von Auftrags-, Parameter- und Abbruchregister mit einem falschen Paßwort oder die Übertragung zum / vom Transferregister einer falschen oder unbelegten Kommunikationspage.

Neben diesen außerhalb des Stabilen Speichers auftretenden Fehler können auch Hardware-Fehler toleriert werden, die sich in seinem Innern ereignen:

- transienter Fehler bei Speicherzugriff (Kontrollesen, Transferwiederholung);
- spontane Veränderungen einzelner Bits während der “Lagerung” (zweifache Abspeicherung mit Paritybits);
  während der Ausführung eines WRITE-Auftrags darf sich dagegen insgesamt höchstens ein Fehler in den Übertragungsdaten oder ein Fehler in der Backup-Kopie ereignet haben;
- Ausfall eines Knotens der TMR-Steuerung ohne Beeinträchtigung der aktuellen Auftragsbearbeitung (Voter);
- Ausfall / Störung eines Busses zwischen zentraler Steuerung und Pufferplatine bzw. Objektspeichern (Voter);
- vorübergehender Ausfall der gesamten zentralen Steuerung, wenn das anschließende Zu-

rücksetzen (Reset) der Prozessoren des TMR-Verbundes wieder zur vollen Funktionstüchtigkeit dieser drei Prozessoren führt;
können nur zwei Prozessoren wieder gleichzeitig gestartet werden, so ist der Stabile Speicher nur noch eingeschränkt arbeitsfähig; dies wird in der "Schadensinformation" dem Auftraggeber mitgeteilt;

- vorübergehender Ausfall eines Objektspeichers;
  dies führt zu einem eingeschränkten Betrieb;
- vorübergehender Ausfall des Pufferbereichs oder der Verbindung zwischen Stabilem Speicher und Kommunikationsspeicher;
  erst nach Wiederinbetriebnahme des Pufferbereichs bzw. der Störungsbeseitigung auf dem Übertragungsweg können die Auftraggeber wieder zugreifen.

Der dauerhafte Ausfall des Pufferbereichs bzw. die dauerhafte Störung des Übertragungsweges kann dagegen nicht toleriert werden. Allerdings bleibt die Sicherheit der Stabilen Objekte gewährleistet. Nach einer Reparatur ist der Stabile Speicher wieder nutzbar, ohne daß die Stabilen Objekte zwischenzeitlich Schaden genommen hätten.

Ein weiterer Punkt betrifft die Energieversorgung. Der Stabile Speicher ist mit einer eigenen Stromversorgung ausgestattet und somit unabhängig vom Host-System. Daher kann hierfür eine unterbrechungsfreie Stromversorgung eingerichtet werden. Beim Prototyp ist dies zwar nicht erfolgt, jedoch besteht kein Hindernis dafür. Sogar separate Versorgungen für die zentrale Steuerung, für den Pufferbereich und für jeden einzelnen Objektspeicher sind problemlos möglich, da die Kommunikation untereinander ausschließlich asynchron über den Rückwandbus erfolgt (siehe Abb. 41). Bei Ausfall einer Platine sind die herausführenden Steuersignale auf "inaktiv" gesetzt.

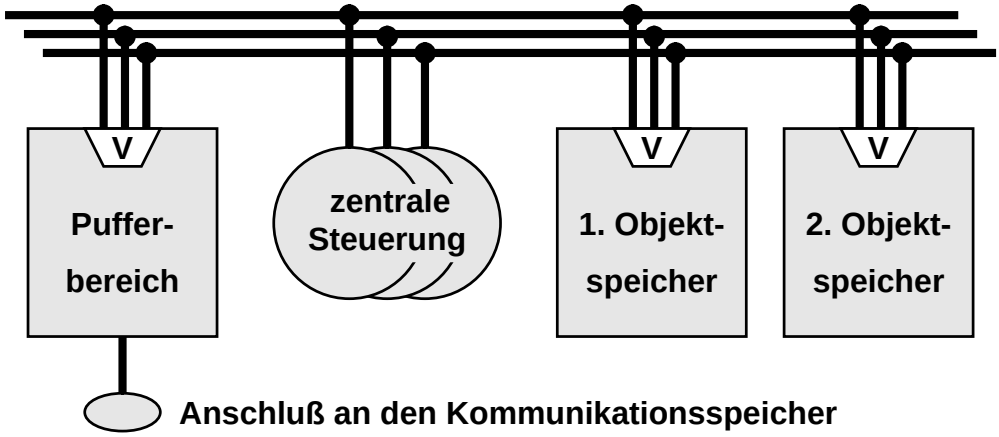


Abb. 41 physikalische Struktur des Stabilen Speichers

6.9 Vorsorgemaßnahmen im Host-System gegen Knotenausfälle

Die gemeinsame Nutzung eines Stabilen Speichers durch mehrere Auftraggeber, die auch in verschiedenen Prozessorknoten sitzen können, war der Grund dafür, daß Schutzmechanismen eingeführt wurden, die es einem Auftraggeber praktisch unmöglich machen sollen, auf abgelegte Daten fremder Auftraggeber zugreifen zu können. So erhalten die Auftraggeber nur dann Zugriff auf die Stabilen Objekte, wenn sie genaue Angaben bezüglich der Kommunikationspage-Nummer, des Paßwortes und der Objektnummer und -länge machen können. Ist auch nur eine dieser Angaben falsch, so wird der versuchte Zugriff vom Stabilen Speicher abgewiesen. Dadurch soll verhindert werden, daß ein falscher oder ein fehlerhaft arbeitender Auftraggeber das Objekt manipulieren kann.

Ein Auftraggeber würde bei Verlust dieser Zugriffsinformationen aber in eine Lage kommen, in der er seine eigenen Stabilen Objekte nicht mehr ansprechen, verwenden und auch nicht mehr löschen könnte. Diese Gefahr besteht, wenn diese Informationen nur in einem flüchtigen Speicher (in der Nähe des Prozessors) aufbewahrt werden würde. Nach einem vorübergehenden Ausfall des Prozessorknotens, der durch einen Reset des Prozessorsknotens behoben werden kann, ist nicht damit zu rechnen, daß diese Informationen noch im Hauptspeicher des Prozessors zu finden sind. Es ist auch nicht möglich, im Stabilen Speicher nach diesen Informationen zu suchen, denn das würde die eingangs erwähnten Schutzvorkehrungen umgehen.

Daher müssen die genannten Informationen im Bereich des Prozessorknotens außerhalb des Hauptspeichers aufgehoben werden, wo danach gesucht werden kann, z.B. auf einer lokalen Magnetplatte. Die dabei abzulegenden Daten können z.B. aus der Kommunikationspage-Nummer, dem Paßwort und aus einer Liste aller Objektnummern mit ihrer Längenangabe bestehen. Diese Liste muß ständig aktuell gehalten werden. Ein Zugriff auf diese Daten ist allerdings nicht bei jedem Zugriff auf ein Stabiles Objekt nötig, sondern nur bei der erfolgreichen Vergabe einer Kommunikationspage und eines Paßwortes (Auftrag ACCESS) und beim Anlegen und Löschen eines Stabilen Objekts (CREATE, DELETE). Solange ein Stabiles Objekt nur wiederholt mit neuen Daten beschrieben wird (WRITE), ist dagegen kein Zugriff auf diese Datenstruktur auf der Platte nötig. Gleiches gilt auch für das Lesen des Stabilen Objekts (READ).

Bei MEMSY existiert in jedem Prozessorknoten eine lokale Platte, auf die nur die in diesem Knoten befindlichen Prozessoren zugreifen können, nicht aber Prozessoren aus den benachbarten MEMSY-Knoten. Daher sind diese Informationen vor fehlerhaft arbeitenden Nachbarprozessoren geschützt. Allerdings besteht auch hier wieder das in Abschnitt 3.3 (Prozessorinterface) angesprochene Problem, daß mehrere Auftraggeber innerhalb eines Knotens existieren können, so daß zwischen ihnen kein Schutz gewährleistet werden kann.

Bis jetzt wurde angenommen, daß ein Prozessorknoten nur vorübergehend ausfällt und anschließend erfolgreich neu gestartet werden kann. Nun wird vorgestellt, welche Vorkehrungen im Host-System erforderlich sind, um auch bei einem dauerhaften Ausfall eines Prozessorknotens auf die Stabilen Objekte des betroffenen Auftraggebers zugreifen zu können.

Ein dauerhafter Ausfall eines Prozessorknotens von MEMSY bedeutet, daß ein anderer Prozessorknoten die Arbeit des ausgefallenen übernehmen muß. Da bei dem hier vorgestellten Stabilen Speicher keine Zuordnung zweier Prozessorknoten durch Hardware existiert, ähnlich der in Abschnitt 2.2 vorgestellten FTM [BAN91], muß auf andere Weise ein “Ersatz-Prozessorknoten” bestimmt werden.

An dieser Stelle wird davon Gebrauch gemacht, daß der Stabile Speicher die zugreifenden Auftraggeber nicht unmittelbar voneinander unterscheiden kann, sondern nur auf der Basis der benutzten Kommunikationspage eine Zuordnung herstellt. Daher kann ein zweiter Auftraggeber dem Stabilen Speicher vorspielen, er sei der ursprüngliche (Original-) Auftraggeber. Dazu muß er nur alle Auftragsangaben korrekt übergeben. Somit besteht für die Bestimmung des Ersatz-Prozessorknotens nur die Bedingung, daß dieser ebenfalls auf denselben Stabilen Speicher zugreifen kann wie der “Original-Prozessorknoten”.

Damit der Ersatz-Auftraggeber im Bedarfsfall die Rolle des ursprünglichen Auftraggebers übernehmen kann, muß der Original-Auftraggeber ihm die aktuellen Informationen zu den bestehenden Stabilen Objekten mitteilen. Dies kann über die üblichen Datenaustauschmöglichkeiten von MEMSY erfolgen. In der eingesetzten Software muß allerdings sichergestellt sein, daß die Verwendung dieser Informationen durch den Ersatz-Auftraggeber ausschließlich nach dem Ausfall des Original-Auftraggebers erfolgt, da der Stabile Speicher diese Alias-Funktion nicht erkennen kann.

7 Meßergebnisse

Der implementierte Stabile Speicher ist in das Multiprozessorsystem MEMSY eingebettet. Hier soll er für mehrere Prozessoren wichtige Daten aufnehmen. In erster Linie ist dabei an Sicherungsdaten gedacht, die im Fall eines bei den Prozessoren aufgetretenen Fehlers für einen Wiederanlauf des Systems benötigt werden. Dabei sollte der Zeitbedarf für eine Sicherung nicht zu hoch werden, da diese Prozedur auch bei anhaltender Fehlerlosigkeit als Vorsorgemaßnahme immer wieder durchgeführt werden muß.

Daher stand als erstes die Leistungsfähigkeit des Stabilen Speichers im fehlerfreien Fall im Vordergrund der Untersuchungen. Dazu wurden Messungen der Übertragungsraten zwischen Prozessor und Kommunikationsspeicher und zwischen Prozessor und Stabilem Speicher durchgeführt.

In der weiteren Fortführung dieses Abschnitts wurden dann einige Ergebnisse aus Fehlersituationen betrachtet und zum Teil den fehlerfreien Fällen gegenübergestellt.

7.1 Leistungsfähigkeit des Stabilen Speichers beim Transfer von Datenblöcken

Bei der fehlerfreien Durchführung der Aufträge können drei bzw. vier Phasen betrachtet werden:

- 1. Phase:
  Der Auftraggeber überprüft, ob der Stabile Speicher bereit ist, einen Auftrag anzunehmen. Besteht dazu die Erlaubnis, so übergibt er einen Auftrag an den Pufferbereich des Stabilen Speichers. In Fall von WRITE-Aufträgen folgt nun der Datentransfer vom Prozessor in den Pufferbereich des Stabilen Speichers, vorausgesetzt, es steht zu diesem Zeitpunkt ein Pufferplatz zur Verfügung. Am Ende dieser Phase wird der Auftrag in das Auftrags-FIFO eingetragen. Sofern der Auftraggeber zu diesem Zeitpunkt nicht daran interessiert ist, welchen Verlauf sein Auftrag genommen hat, kann er diesen Auftrag als beendet ansehen. Ansonsten beginnt für ihn eine Wartezeit, bis das Ergebnis über diesen Auftrag vorliegt.

- 2. Phase:
  Wartephase, in der der Auftrag in dem Auftrags-FIFO bis an die erste Stelle rutscht und am Ende von der zentralen Steuerung des Stabilen Speichers entnommen wird.

- 3. Phase:
  Der Auftrag wird von der zentralen Steuerung ausgeführt. Bei Aufträgen von verwaltungstechnischer Art wie CREATE und DELETE ist kein weiterer Datentransfer nötig. Im Fall von WRITE- und READ-Aufträgen erfolgt ein Datentransfer zwischen dem Puf-

ferbereich und den Objektspeichern. Am Ende dieser Phase wird im Ergebnisregister eine Information über den Verlauf (korrekt oder fehlerhaft) dieser Aktion eingetragen. Für die zentrale Steuerung des Stabilen Speichers ist zu diesem Zeitpunkt der Auftrag beendet. Der Auftraggeber kann nun den Verlauf des Auftrags ablesen. Bei fast allen Aufträgen ist nun alles beendet, was diesen Auftrag betrifft.

- 4. Phase:
  Nur bei einem READ-Auftrag schließt sich eine vierte Phase an. In ihr liest der Auftraggeber den Datenblock aus dem Pufferbereich.

Interessant für Messungen waren vor allem die Auftragsübergabe und die Situationen, in denen Datenblöcke transferiert werden müssen. Zwischen dem Auftraggeber und dem Stabilen Speicher erfolgt dies in den Phasen 1 und 4. Für den Ablauf in der zentralen Steuerung des Stabilen Speichers ist dagegen die Phase 3 interessant. Da alle Phasen nicht überlappend sondern hintereinander ausgeführt werden, konnten diese Messungen unabhängig voneinander vorgenommen werden.

Für die Phase 2 waren Messungen nicht sehr aussagekräftig, da die Wartezeit hauptsächlich von den Aufträgen abhängt, die von den verschiedenen Auftraggebern gleichzeitig an denselben Stabilen Speicher übergeben werden. Die zentrale Steuerung führt diese hintereinander aus. Es lassen sich jedoch Abschätzungen dafür angeben, die aus den Ergebnissen der zuvor erwähnten Messungen abgeleitet werden können.

Als Vergleichswerte wurden Zeiten bzw. Transferraten verwendet, die bei Zugriffen auf den Kommunikationsspeicher auftreten, an den der Stabile Speicher angeschlossen ist. Dies erscheint aus folgendem Grund gerechtfertigt:

Bei allen Verfahren, bei denen mit Hilfe von Sicherungsdaten versucht wird, nach einem Ausfall eines Prozessors bei einem bereits erreichten Zwischenergebnis mit der Berechnung wiederaufzusetzen, spielt die sichere Aufbewahrung der Sicherungsdaten eine entscheidende Rolle. Um auch den dauerhaften Ausfall eines einzelnen Prozessors oder eines einzelnen Speichers oder einer einzelnen Kommunikationslinie (Übertragungsweg) tolerieren zu können, ist eine mehrfache Speicherung der Sicherungsdaten unerläßlich. Alternativ zu der Speicherung im gesondert implementierten Stabilen Speicher ließe sich in einem Multiprozessorsystem wie MEMSY auch eine Speicherung in mehreren Kommunikationsspeichern oder in den Privatspeichern benachbarter Knoten realisieren. Diese Vorgehensweise wird in [LEH90] für das speichergekoppelte Multiprozessorsystem DIRMU vorgestellt. Die dadurch verminderten Schutzmöglichkeiten vor den betrachteten unterschiedlichen Fehlern seien hier einmal ausgeklammert. Zur Datenübertragung steht bei MEMSY das umfangreiche Verbindungssystem mit den Kommunikationsspeichern zur Verfügung. Zum Transport sind somit für jede Kopie eines Datenblocks ein bzw. zwei Transfers zwischen dem Privatspeicher eines Knotens und einem Kommunikationsspeicher notwendig.

Die Untersuchungen zur Transfer-Leistungsfähigkeit des Stabilen Speichers, des Kommunika-

tionsspeichers und des Koppelmoduls sollen vor allem klären, mit welchen Übertragungszeiten der Auftraggeber beim Abspeichern der Sicherungsdaten zu rechnen hat.

7.1.1 Zugriffe einzelner Prozessoren auf den Kommunikationsspeicher

Mehrere Prozessoren können unabhängig voneinander Zugriffe auf die Kommunikationsspeicher starten. Dabei können sich die Zugriffswünsche am Kommunikationsspeicher zeitlich überlappen. In Abschnitt 3.4 über den Kommunikationsspeicher wurde bereits erläutert, daß die Steuersignale synchronisiert werden und eine Arbitrierung erfolgen muß, bevor der Zugriff durchgeführt werden kann. Die Zeit, die für einen solchen Zyklus benötigt wird, begrenzt die Zugriffsrate am Kommunikationsspeicher.

Zwischen den Prozessoren und den Kommunikationsspeichern befinden sich bei MEMSY die Koppelmodule, die die nachbarschaftlichen Verbindungen herstellen. Auch hier wird jeder Zugriff vor allem wegen der Synchronisation und der Arbitrierung und zu einem geringen Teil wegen des Durchlaufs durch mehrere Treiberstufen zeitlich verzögert (siehe Abschnitt 3.5, Koppelmodul).

Der Einsatz eines einzelnen Stabilen Speichers innerhalb einer Elementarpyramide in MEMSY führt dazu, daß von mehreren Prozessoren Zugriffe auf denselben Speicheranschluß eines Koppelmoduls ausgeführt werden. Damit ergeben sich im Koppelmodul Konflikte durch Zugriff auf denselben Speicher. Folglich wird dies die Begrenzung bei der maximalen Zugriffsrate auf den angeschlossenen Kommunikationsspeicher sein.

Untersucht wurde der Ablauf einer Schleife innerhalb eines Datentransfer-Programms. Dabei wird ein Datenblock aus dem Privatspeicher eines Motorola-Rechners in einen Kommunikationsspeicher übertragen. Mit Hilfe eines Logikanalysators konnte die Dauer gemessen werden, während der der Prozessor bzw. die CMMU mit der Übertragung eines einzelnen Wortes in den Kommunikationsspeicher beschäftigt ist. Auch die Zyklusdauer von einer Übertragung zur nächsten wurde auf diese Weise ermittelt.

Die Schleife des Testprogramms hatte folgenden Inhalt:

```
for ( i=0; i<Blocklänge; i++ )
    Kommunikationsspeicher[i] = Privatspeicher[i];
```

Der Ablauf dieser Schleife besteht aus einem Transfer eines Wortes vom Privatspeicher in ein Register im Prozessor, aus einem Transfer vom Register in den Kommunikationsspeicher und aus einer Schleifenbearbeitung.

Dieses Programm wurde auf zwei Hardware-Konfigurationen gemessen. Die erste Konfiguration bestand aus einer direkten Kopplung eines Motorola-Systems mit einem Kommunikationsspeicher. In der zweiten Konfiguration wurde noch ein Koppelmodul dazwischengeschaltet. Damit sollte die Verzögerung ermittelt werden, die durch das Koppelmodul erzeugt wird.

In Abb. 42 ist der prinzipielle Ablauf der Schleife dargestellt. Dabei fällt auf, daß der Kommunikationsspeicher mit einer geringeren Zeitdauer an dem Speicherzugriff beschäftigt ist als der Prozessor. Das ergibt sich daher, daß im Prozessorinterface zunächst entschieden werden muß, über welchen Port der Zugriff erfolgen soll. Dazu muß die für diesen Speicherzugriff angegebene Adresse untersucht werden. Dann erst werden am ausgewählten Port die Steuersignale zum Start des Speicherzugriffs im Kommunikationsspeicher erzeugt. Diese werden zum Kommunikationsspeicher übertragen und dort synchronisiert. Erst zu diesem Zeitpunkt kann die Steuerung des Kommunikationsspeichers auf diesen Zugriffswunsch reagieren. Da das Ready-Signal, das der Kommunikationsspeicher zum Prozessor schickt, im Prozessorinterface ebenfalls synchronisiert werden muß, beendet der Prozessor den Zugriff später als der Kommunikationsspeicher. Durch den Einsatz des Koppelmoduls verstärkt sich dieser Effekt noch weiter, da auch hier sowohl bei den Signalen auf dem Hinweg als auch bei den Signalen auf dem Rückweg synchronisiert werden muß.

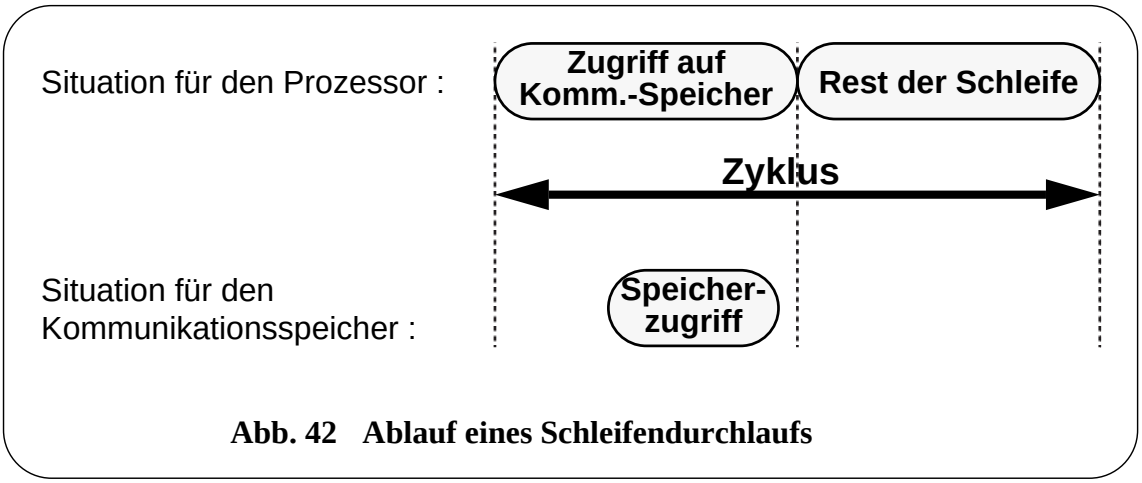


Abb. 42 Ablauf eines Schleifendurchlaufs

Das alles hat zur Folge, daß der Kommunikationsspeicher in der Lage ist, mehr Zugriffswünsche zu bearbeiten als ein einziger Prozessor erzeugen kann. Weitere Prozessoren, die ihre Zugriffe über die anderen Ports am Kommunikationsspeicher führen, können dies nutzen. Diese Überlappung der Zugriffe wird zwar bei konkurrierenden Zugriffen über ein Koppelmodul auf einen Kommunikationsspeicher nicht genutzt; sie macht aber deutlich, daß bei intensiven Zugriffen mehrerer Prozessoren auf denselben Kommunikationsspeicher die Prozessoren selbst bzw. das Koppelmodul der begrenzende Faktor sind und nicht der Kommunikationsspeicher.

Der regelmäßig auftretende Refresh der DRAM-Module macht sich in den Messungen kaum bemerkbar. Er tritt etwa alle 11 µsec auf und belegt den Kommunikationsspeicher für die Dauer von etwa 0,3 µsec. Er tritt wie ein weiterer Zugreifer auf den Kommunikationsspeicher auf, der asynchron zu den Prozessoren arbeitet. Da er aber keinen laufenden Zugriff verdrängt und sein Anteil an der Speicherbelegung unter 3% liegt (0,3 µsec von 11 µsec), macht sich der Refresh praktisch nicht im Zeitverhalten der Prozessoren bemerkbar. Daher wird diese Zeit nicht weiter berücksichtigt. In Tab. 3 sind die Meßergebnisse für die beiden Hardware-Konfigurationen (ohne / mit Koppelmodul) dargestellt. Ferner enthält die Tabelle auch die Zugriffszeit des Pro-

zessors auf den Privatspeicher. Die Zeiten sind in der Regel auf 0,1 µsec genau angegeben, obwohl genauere Meßergebnisse (auf 0,01 µsec genau) vorliegen. Der Grund dafür liegt darin, daß sich die einzelnen Messungen wegen der notwendigen Synchronisationen zwischen Prozessor und Kommunikationsspeicher unterscheiden. Die gemessenen Abweichungen betrugen bis zu 0,06 µsec in beiden Richtungen. Um nicht eine größere Genauigkeit vorzugaukeln, wurden typische Werte ermittelt und auf 0,1 µsec genau gerundet. Da sich die meisten Zeitwerte in dieser und auch in den weiteren Tabellen über 1,0 µsec bewegen, bedeutet diese Rundung aber keinen Nachteil für die Genauigkeit.

Zugriff auf Komm.-Speicher	ohne Koppelmodul	mit Koppelmodul
Zyklus-Zeit	2,0 µsec	2,4 µsec
Zugriff auf Kommunikationssp.	0,8 µsec	1,2 µsec
Rest der Schleife	1,2 µsec	1,2 µsec
Speicherzugriff	0,45 µsec	0,45 µsec
Zugriff auf Privatspeicher	0,2 µsec	0,2 µsec

Tab. 3 Zugriff eines Prozessors auf den Kommunikationsspeicher

7.1.2 Zugriffe einzelner Prozessoren auf den Stabilen Speicher

Zugriffe der Prozessoren auf den Pufferbereich des Stabilen Speichers werden durchgeführt wie Zugriffe auf einen Kommunikationsspeicher. Allerdings leitet der Kommunikationsspeicher den Zugriff an den Stabilen Speicher weiter und erwartet seinerseits von ihm ein Ready- oder ein Error-Signal. Dieses Signal und bei Lesezugriffen auch die Daten übermittelt er dann dem Prozessor. Da auch an der Schnittstelle zwischen Kommunikationsspeicher und dem Stabilen Speicher wieder synchronisiert werden muß, wird hier ebenfalls ein Zeitverlust zu beobachten sein.

Bei den Zugriffen auf den Pufferbereich des Stabilen Speichers werden intern mehrere Aktionen ausgeführt. Das liegt zum einen an den notwendigen Überprüfungen der Auftraggeberzugriffe auf ihre Rechtmäßigkeit. Zum anderen werden durch diese Zugriffe auch sehr unterschiedliche Aktionen ausgelöst. Zu den einfachsten Aktionen gehören Leseoperationen aus der VIP und Lese- bzw. Schreiboperationen, welche die Parameterregister einer Kommunikationspage betreffen (KP[1..3]), sofern die Zugriffe erlaubt sind. Recht einfach gestalten sich auch die Lese-Zugriffe auf das Ergebnisregister KP[4] und auf das Wiederholungsregister KP[6]. Hier muß nur der entsprechende Inhalt des Registers zum Auftraggeber übertragen werden. Beim Ergebnisregister wird dabei in das oberste Byte der Inhalt des Schadensregisters eingeblendet.

Aber auch unzulässige Zugriffe auf Register führen zu sehr einfachen Aktionen: bei Schreibzugriffen wird nichts getan, bei Leseoperationen eine Null zurückgeliefert.

Zugriffe auf andere Register einer KP führen dagegen zu deutlich umfangreicheren Aktionen. So bedeutet das Schreiben auf das Auftragsregister, daß neben dem Eintrag des Auftrags in die KP[0] noch folgende weiteren Aktionen auszuführen sind (in Phase 1):

- Sperren der Auftrags- und Parameter-Register gegen weitere Zugriffe,
- bei Transfer-Aufträgen:
  Suchen und Belegen eines Pufferbereichs und Eintrag der Startadresse in KP[5],
- bei Schreib-Aufträgen (WRITE und CREATE_WRITE):
  Pufferbereich zum Schreiben freigeben,
- Eintrag der aktuellen Situation in das Ergebnisregister,
- bei allen Aufträgen mit Ausnahme der Schreib-Aufträge:
  Eintrag der KP-Nummer in das Auftrags-FIFO;
  die zentrale Steuerung erhält daraus die Information über die vorliegenden Aufträge.

Ähnlich aufwendig gestalten sich die Zugriffe auf das Transferregister KP[5]. Hier müssen folgende Aktionen durchgeführt werden (in Phasen 1 und 4):

- Überprüfung, ob der gewünschte schreibende bzw. lesende Zugriff zur Zeit erlaubt ist oder nicht; im Fall der Erlaubnis wird mit den folgenden Aktionen weitergefahren.
- Schreib- bzw. Lese-Zugriff auf den Puffer, wobei die gespeicherte aktuelle Transferadresse verwendet wird; nur bei dieser Aktion muß im Pufferbereich ein Zugriff über die Zugriffskontrolle hinaus in den Pufferspeicher erfolgen (siehe Abschnitt 6.4, Pufferbereich);
- Dekrementieren dieser Adresse und Zurückschreiben ins Register,
- bei Lese-Transfers:
  Übergabe des aus dem Puffer erhaltenen Wortes an den Kommunikationsspeicher zur Weitergabe an den Prozessor und Eintrag ins Wiederholungsregister KP[6],
- Feststellung, ob dieser Transfer die Übertragung des letzten Wortes aus dem Objekt ist; falls dies so ist, muß mit den folgenden Aktionen weitergefahren werden.
- Sperrung des Pufferbereichs (bei Lese- und Schreib-Transfers),
- bei Lese-Transfers muß die Sperre der Auftrags- und Parameter-Register wieder aufgehoben werden, da in diesem Moment die gesamte READ-Operation des Anwenders beendet wird;
- bei Schreib-Transfers muß die KP-Nummer ins Auftrags-FIFO eingetragen werden.

Der Schreib-Zugriff auf das Abbruchregister KP[7] führt ebenfalls zu mehreren internen Aktionen:

- Überprüfung, ob in der augenblicklichen Situation ein Abbruch erlaubt ist (ist nur während der Schreib- oder der Lese-Phase bei Datentransfers erlaubt),
- Eintrag der aktuellen Situation im Ergebnisregister;
  falls der Abbruch erlaubt ist, wird mit folgenden Aktionen fortgefahren:
- Freigabe des benutzten Pufferbereichs und Sperrung weiterer Zugriffe über das Transferregister;
- Aufhebung der Sperre bei Zugriffen auf Auftrags- und Parameter-Registern.

Der Auftrag ACCESS erfolgt als Lese-Zugriff auf das Vermittlungsregister der VIP (VIP[63]). Dies hat auch eine Reihe interner Aktionen zur Folge, denn hier muß ein neuer Teilnehmer in eine Kommunikationspage eingetragen werden:

- Lesen der Zahl der noch freien Kommunikationspages;
  ist sie noch größer als Null, so wird wie folgt weitergefahren:
- Dekrementieren dieser Zahl und Zurückschreiben in das entsprechende Register,
- Anhalten und Lesen der rückgekoppelten Schieberegister, die der Erzeugung eines Paßwortes und einer KP-Nummer dienen (“gewürfelte” KP-Nummer),
- Untersuchung, ob die auf diese Weise ermittelte Kommunikationspage noch frei ist;
  ist sie frei, so kann der nächste Punkt übersprungen werden;
- Suchen einer noch unbelegten Kommunikationspage,
- Eintrag einer “Belegt”-Kennung in die gefundene Kommunikationspage (KP[7]),
- Aktualisierung einer Datenbasis, in der über die belegten Kommunikationspages Buch geführt wird,
- Rückgabe des Paßwortes, der KP-Nummer und einer “Frei”-Kennung an den Prozessor;
  konnte keine Kommunikationspage mehr vergeben werden, wird eine “Voll”-Kennung zurückgegeben,
- Fortführung der zyklischen Shifts in den rückgekoppelten Schieberegistern für die Erzeugung einer neuen KP-Nummer und eines Paßwortes.

Diese Aktionen müssen während eines einzigen Speicherzugriffs eines Prozessors durchgeführt werden. Neben dem allgemeinen Wunsch, daß diese Aktionenfolge möglichst schnell durchgeführt werden sollen, besteht besonders bei den Stellen, bei denen ein Suchalgorithmus erfolgen muß, ein zwingender Grund für eine schnelle Bearbeitung dieser Aktionen. Denn hier besteht die Gefahr, daß sich die Durchführungsdauer derart verlängert, daß die Timeout-Schranke des Kommunikationsspeichers (ca. 11 µsec) oder des Koppelmoduls (ca. 15 µsec) erreicht wird. In diesen Fällen würden diese Einheiten den Zugriff abbrechen. Das würde auch dann geschehen, wenn der Grund für die lange Dauer in der Implementierung des Algorithmus liegt und nicht in einer defekten Hardware.

Suchalgorithmen müssen bei der Suche nach einer noch unbelegten Kommunikationspage und bei der Suche nach einem freien Pufferplatz während der Auftragsvergabe (Schreiben auf KP[0]) durchgeführt werden. Zur Verkürzung der Suchdauern wurden für die Pufferplätze und für die Kommunikationspages kleine Datenbasen angelegt, bei denen je ein Bit Aufschluß über deren aktuelle Belegung geben. So wurden immer 16 Bits zu einem Wort zusammengefaßt und abgespeichert, wie Abb. 43 zeigt.

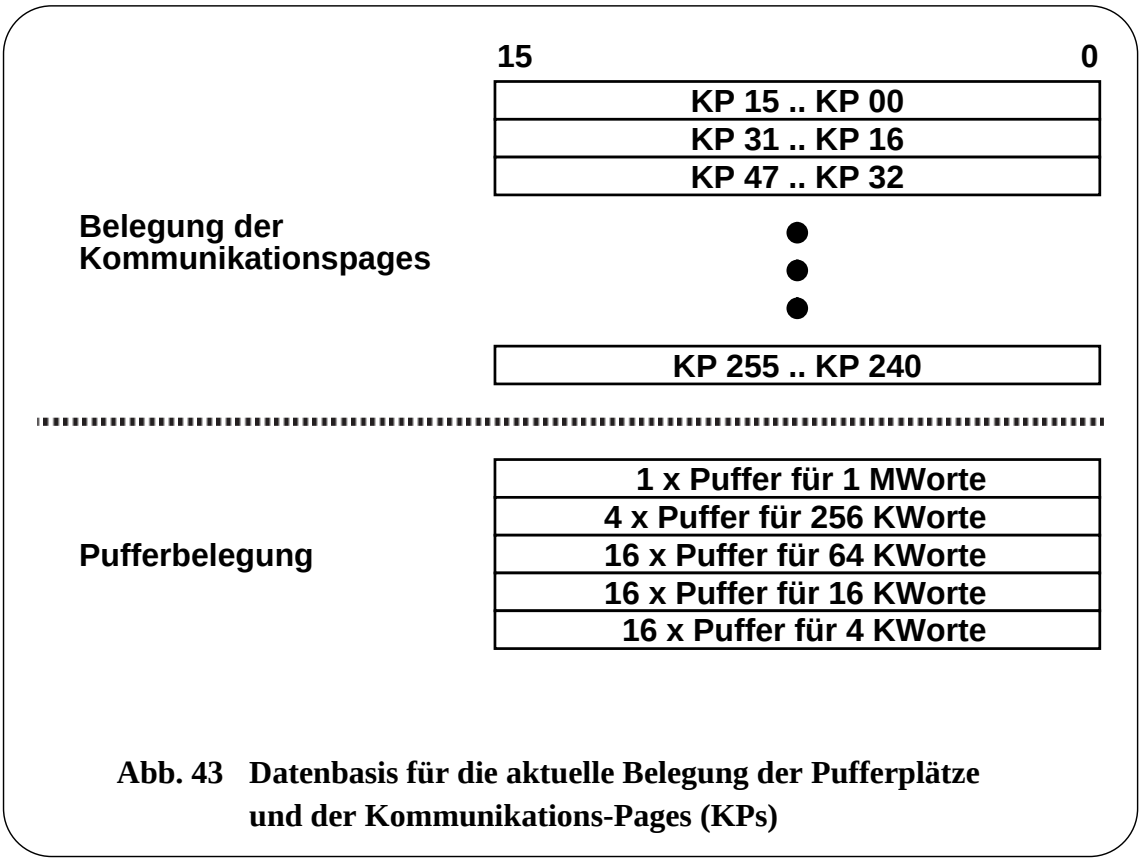


Abb. 43 Datenbasis für die aktuelle Belegung der Pufferplätze und der Kommunikations-Pages (KPs)

Die Kommunikationspages, die nicht zur Verteilung bereitstehen (z.B. KP 0), werden bereits bei der Initialisierung dieser Datenbasis mit einem “Belegt”-Eintrag versehen. Bei der Belegung für Puffergrößen, die nicht 16-mal zur Verfügung stehen, ist diese Vorgehensweise ebenfalls möglich. Diese Worte werden innerhalb des reservierten Bereichs der VIP abgelegt (VIP[32..60]).

Zur Feststellung, ob innerhalb einer so angeordneten Sechzehnergruppe ein freier Platz vorhanden ist, wurde eine Hardware erstellt, die innerhalb eines Taktes die unvollständige Belegung anzeigt und gleichzeitig die Position der “niedrigsten Null” innerhalb dieser Gruppe ausgibt. Diese Hardware übernimmt auch das Setzen dieses ausgewählten Bits. Diese Hardware ist Bestandteil der “Datenmanipulation” aus Abb. 19 in Abschnitt 6.4 (Pufferbereich).

Für die Freigabe eines Pufferbereichs besteht auch die Möglichkeit, ein bestimmtes Bit zu löschen. Die Position des Bits kann eindeutig auf Grund der angegebenen aktuellen Pufferadresse (während des Transfers des letzten Wortes eines Blocks) bestimmt werden.

Auf ganz ähnliche Weise erfolgt auch die Freigabe einer belegten Kommunikationspage. Im Rahmen der Ausführung des RELEASE-Auftrags überträgt die zentrale Steuerung des Stabilen Speichers die auszutragende KP-Nummer an diese Hardware, die ihrerseits das entsprechende Bit löscht. Die Position des Bits (unter 256 Bits) kann eindeutig aus der KP-Nummer ermittelt werden. In diesem Fall muß kein Suchalgorithmus ausgeführt werden.

Die unterschiedlichen Aktionenfolgen werden durch eine programmierte Steuerung ausgeführt. Dabei müssen meistens mehrere Zugriffe auf die Einträge einer Kommunikationspage oder der VIP erfolgen. Die Anzahl der Zugriffe kann als Maß für die Ausführungsdauer einer Aktionenfolge dienen. Den meisten Zugriffen der Auftraggeber läßt sich eine bestimmte feste Zahl an Aktionen zuordnen. Einige wenige sind dagegen abhängig von den aktuellen Zuständen im Pufferbereich. So muß beim ACCESS-Zugriff, wobei dem Auftraggeber ein neues Paßwort und eine neue KP-Nummer zugewiesen wird, unter Umständen ein Algorithmus ausgeführt werden, bei dem mehrere Einträge in der Datenbasis für die KP-Belegung untersucht werden müssen. Die Suchlänge bestimmt daher die tatsächliche Ausführungsdauer. Dagegen gibt es bei der Suche nach einem geeigneten Pufferplatz, die im Rahmen einer Auftragsvergabe (Schreibzugriff auf KP[0]) erfolgt, keine unterschiedliche Suchdauer, da hier immer nur ein einziger Eintrag aus der Datenbasis für die Pufferbelegung untersucht werden muß. Dennoch ergeben sich auch bei der Auftragsvergabe unterschiedliche Ausführungsdauern, da in Abhängigkeit davon, ob der übergebene Auftrag mit einem Datentransfer verbunden ist oder nicht, verschiedene Aktionen ausgeführt werden müssen.

Werden vom Auftraggeber Zugriffe auf den Pufferbereich gestartet, die zum aktuellen Zeitpunkt nicht erlaubt sind, so wird ebenfalls eine andere Aktionenfolge ausgeführt als unter normalen Umständen, in denen diese Zugriffe erlaubt sind. Hier sind die wichtigsten ungültigen Zugriffe aufgeführt:

- Zugriff auf eine unbelegte Kommunikationspage;
- Schreibzugriff auf Auftrags- oder Parameter- oder Abbruch-Register mit falschem Paßwort;
- Zugriffe auf das Auftrags- und auf die Parameter-Register, wenn der Auftrag schon ins Auftrags-FIFO übergeben worden ist;
- Zugriff auf das Transfer-Register, wenn derzeit keine Erlaubnis dazu besteht;
  Schreib- bzw. Lese-Zugriff wird nur in den Transferphasen während eines Transferauftrags erlaubt;
- Zugriff auf das Abbruch-Register außerhalb der Transferphasen;
- Zugriff auf generell verbotene Register, z.B. Schreiben auf VIP, Lesen von KP[0], ...

In Tab. 4 sind für jeden Auftraggeber-Zugriff die Anzahl der Zugriffe auf die KP-Einträge, das Schreiben ins Auftrags-FIFO, der Transfer mit dem Pufferspeicher und die Zeit vermerkt, die die Steuerung mit diesen Aktionen beschäftigt ist. Die in den Fehlersituationen gültigen Anga-

ben sind mit einem “F” gekennzeichnet.

	Lese-Zugriff	Schreib-Zugriff
	Zahl der Aktionen ↓ FIFO-Zugriff ↓ Puffer-Transfer ↓ benötigte Zeit	Zahl der Aktionen ↓ FIFO-Zugriff ↓ Puffer-Transfer ↓ benötigte Zeit
VIP [0] : VIP [62]	2 - - 1,0 µsec	0 - - 0,75 µsec (**F**)
VIP [63]	7 - - 2,0 µsec : : 22 - - 4,8 µsec	0 - - 0,75 µsec (**F**)
KP [0]	1 - - 0,9 µsec (**F**)	falscher Auftrag: 1 - - 0,9 µsec Auftrag ohne Dat.-Transf.: 4 1 - 1,6 µsec Auftrag mit Dat.-Transfer: 8 1 - 2,3 µsec
KP [1]	2 - - 1,0 µsec	2 - - 1,0 µsec
KP [2]	1 - - 0,9 µsec (**F**)	2 - - 1,0 µsec
KP [3]	2 - - 1,0 µsec	2 - - 1,0 µsec
KP [4]	2 - - 1,0 µsec	1 - - 0,75 µsec (**F**)
KP [5]	4 - 1 1,6 µsec	3 - 1 1,4 µsec
KP [6]	2 - - 1,0 µsec	1 - - 0,75 µsec (**F**)
KP [7]	1 - - 0,9 µsec (**F**)	Abbr. in falscher Situation: 3 - - 1,2 µsec Abbr. in richtiger Situation: 7 - - 2,0 µsec
KP [8] : KP [63]	1 - - 0,9 µsec (**F**)	0 - - 0,75 µsec (**F**)

Tab. 4 Zugriff eines Prozessors auf die VIP und die KP des Stabilen Speichers

Die Fehlersituation “Zugriff auf eine unbelegte Kommunikationspage” ist in Tab. 4 nicht vermerkt. In diesem Fall dauert die Ausführung eines gewünschten Lesezugriffs 0,9 µsec. Dabei wird immer eine Null an den Prozessor zurückgeliefert. Ein beabsichtigter Schreibzugriff auf eine unbelegte Kommunikationspage wird nicht ausgeführt. Zur Erkennung und zur kontrollierten Beendigung des Zugriffs ohne Ausführung ist die Steuerung 0,8 µsec lang beschäftigt.

Die aufgeführten Zeiten berücksichtigen nur die Zeit, die die Steuerung selbst für die internen Aktionen benötigt. Zur Ermittlung der Zeitdauer, während der der Prozessor mit dem Zugriff auf den Stabilen Speicher beschäftigt ist, müssen noch Zeiten hinzugefügt werden, die bei einem Zugriff über die verschiedenen Stufen vor allem wegen der nötigen Synchronisation und der Auswahl von Übertragungswegen vergehen:

- Umsetzung der lokalen Signale des Motorola-Systems in die Signale auf dem Übertragungsweg: 0,35 µsec.
- Verzögerung durch das Koppelmodul: 0,4 µsec.
- Overhead im Kommunikationsspeicher: 0,35 µsec.

Bei Einsatz des Stabilen Speichers ohne Verwendung des Koppelmoduls müssen folglich alle in der Tab. 4 angegebenen Zeiten um 0,7 µsec erhöht werden. Bei Verwendung des Koppelmoduls erhöhen sich die Zeiten um insgesamt 1,1 µsec.

Zur Beantwortung der Frage, ob bei einem der möglichen Auftraggeber-Zugriffe die Gefahr besteht, daß der beteiligte Kommunikationsspeicher oder das Koppelmodul eine langdauernde Bearbeitung vorzeitig mit Timeout beenden wird, müssen zu der in der Tabelle angegebenen Zeit noch die Zeit für den Overhead am Kommunikationsspeicher und die zusätzliche Zeit für das Koppelmodul berücksichtigt werden. Danach ergibt sich auch für den langwierigsten Fall (ACCESS), daß dieser in einer Zeit unter 6 µsec abgeschlossen sein wird. Da die Timeout-Zeiten bei 11 µsec bzw. 15 µsec liegen, besteht keine Gefahr eines irrtümlich ausgelösten Timeout.

An dieser Stelle sind noch einige Bemerkungen zum ACCESS-Zugriff angebracht. Die kürzeste Zeit von 2,0 µsec innerhalb des Stabilen Speichers wird erreicht, wenn die “gewürfelte” KP-Nummer bisher nicht belegt ist, da in diesem Fall kein Suchalgorithmus gestartet werden muß. Ist die gewürfelte KP-Nummer aber bereits belegt, so wird zunächst in der Sechzehnergruppe nach einer freien KP-Nummer gesucht, zu der auch die gewürfelte KP-Nummer gehört. Wird hier eine freie KP-Nummer gefunden, so wird diese belegt. Dieser Vorgang benötigt nicht mehr Zeit als der zuerst beschriebene Fall. Das liegt daran, daß auch im Fall der direkten Wahl einer freien Kommunikationspage ein Eintrag in die entsprechende Sechzehnergruppe der Datenbasis für die KP-Belegung erfolgen muß. Da diese beiden Fälle auch dann noch mit sehr hoher Wahrscheinlichkeit (über 95%) auftreten, wenn bereits 102 von 255 Plätzen belegt sind (das sind 40% der Kommunikationspages), wird üblicherweise kaum eine längere Zugriffszeit beim Auftrag ACCESS zu beobachten sein. Da die Auftraggeber für einen Stabilen Speicher normalerweise nur aus fünf Prozessorknoten kommen, bedeuten 102 belegte Plätze, daß in jedem Prozessorknoten etwa 20 Auftraggeber vorhanden sind, eine Zahl, die recht hoch gegriffen ist.

Ähnlich zu dem im letzten Abschnitt behandelten Testprogramm wurde auch hier ein typischer Ablauf bei der Übertragung eines Datenblocks aus dem Privatspeicher eines Prozessors des Motorola-Systems in den Pufferbereich des Stabilen Speichers gemessen. Zur Ermittlung der Übertragungsleistung wurde nur der Datentransfer genauer untersucht. Die zuvor nötige Übergabe des Auftrags WRITE blieb hier unberücksichtigt, da sie im Gegensatz zu den Transferoperationen nur einmal ausgeführt werden muß.

Die Schleife des Testprogramms hatte folgenden Inhalt:

```
for ( i=0; i<Blocklänge; i++ )
    KP[5] = Privatspeicher[i];
```

Der Ablauf dieser Schleife besteht aus einem Transfer eines Wortes vom Privatspeicher in ein Register im Prozessor, dem Transfer vom Register in das Transferregister KP[5] und aus einer Schleifenbearbeitung.

Auch in diesem Fall wurde das Programm auf zwei Hardware-Konfigurationen gemessen, einmal ohne und einmal mit Verwendung des Koppelmoduls. In Tab. 5 sind die Ergebnisse zusammengefaßt.

Zugriff auf Stabilen Speicher	ohne Koppelmodul	mit Koppelmodul
Zyklus-Zeit	3,3 µsec	3,7 µsec
Zugriff auf Stabilen Speicher	2,1 µsec	2,5 µsec
Rest der Schleife	1,2 µsec	1,2 µsec
KS mit Zugriff beschäftigt	1,75 µsec	1,75 µsec
Aktionenfolge im Stabilen Sp.	1,4 µsec	1,4 µsec

Tab. 5 WRITE-Zugriff eines Prozessors auf den Stabilen Speicher

Die Zeit von 1,75 µsec, die der Kommunikationsspeicher bei einem WRITE-Auftrag mit jedem übertragenem Wort beschäftigt ist, zeigt, daß der Pufferbereich des Stabilen Speichers (zusammen mit dem Kommunikationsspeicher) in der Lage ist, maximal etwa 571000 Worte pro Sekunde (ca. 2,2 MByte/sec) aufnehmen zu können. Gegenüber dem Kommunikationsspeicher (0,45 µsec, über 2222000 Worte pro Sekunde, etwa 8,4 MByte/sec) ist das eine Verringerung etwa um den Faktor vier. Das macht den zusätzlichen Synchronisationsaufwand zwischen dem Kommunikationsspeicher und dem Stabilen Speicher sowie den Schutzaufwand deutlich.

Berücksichtigt man noch das Koppelmodul, über das in beiden Fällen zugegriffen wird, so betragen die Zeiten 2,15 µsec für den Stabilen Speicher gegenüber 0,85 µsec für den Kommunikationsspeicher. Über das Koppelmodul können somit bis zu 465000 Worte in den Puffer des Stabilen Speichers geschrieben werden, während in den Kommunikationsspeicher bis zu 1176000 Worte übertragen werden können. Das Verhältnis ist in diesem Fall nicht mehr 1 : 3,9 sondern 1 : 2,5 .

Die bisherigen Messungen brachten Aufschluß über die Leistungsfähigkeit des Pufferbereichs des Stabilen Speichers beim Zugriff eines einzelnen Prozessors. Da aber zu erwarten ist, daß bei einem typischen Multiprozessorprogramm mit regelmäßigem Schreiben von Sicherungsdaten mehrere Prozessoren gleichzeitig die Absicht haben, Datenblöcke in den Stabilen Speicher zu schreiben, wurden weitere Messungen vorgenommen. Dabei greifen bis zu vier Prozessoren über ein Koppelmodul auf den Pufferbereich zu.

7.1.3 Zugriffe mehrerer Prozessoren auf den Kommunikationsspeicher

Die bisherigen Beobachtungen von Zugriffen der Prozessoren auf die Kommunikationsspeicher von MEMSY legten die Vermutung nahe, daß vor allem das Koppelmodul der begrenzende Faktor bei der Übertragung der Daten in den Kommunikationsspeicher ist. Daher wurde zur weiteren Untersuchung das zuvor beschriebene Datentransferprogramm in einer Konfiguration eingesetzt, bei der bis zu vier Prozessoren über dasselbe Koppelmodul auf denselben Kommunikationsspeicher zugreifen (siehe Abb. 44).

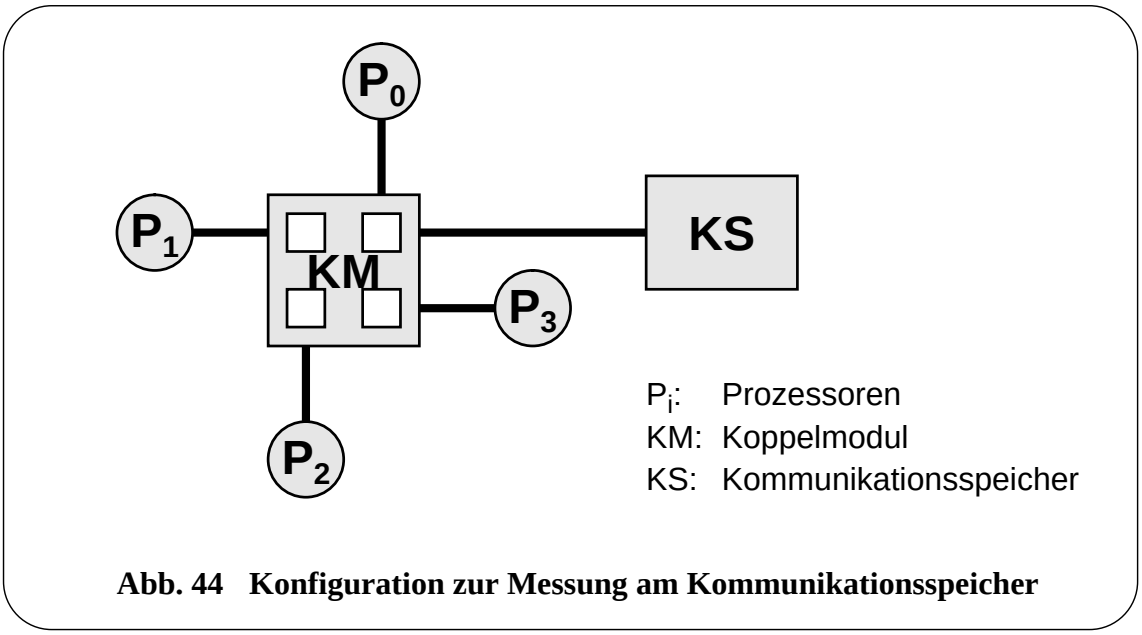


Abb. 44 Konfiguration zur Messung am Kommunikationsspeicher

Dabei wurde gemessen, wie lange der Datentransfer eines Prozessors dauert, wenn er allein Zugriffe auf den Kommunikationsspeicher ausführt, oder wenn gleichzeitig ein bis drei weitere Prozessoren ein gleichartiges Programm ausführen. Um zu verhindern, daß der gemessene Datentransfer zum Teil ohne Einfluß durch die anderen Prozessoren abläuft, mußten die anderen

Prozessoren einen deutlich größeren Datenblock transportieren. Dadurch konnte sichergestellt werden, daß der gemessene Datentransfer vollständig unter Konkurrenz der anderen Prozessoren stattgefunden hat.

Bei den dabei gewonnenen Meßergebnissen konnte beobachtet werden, daß die durchschnittlichen Zugriffszeiten davon abhingen, ob der Prozessor an derselben Ecke des Koppelmoduls angeschlossen war wie der Kommunikationsspeicher oder nicht. Dies machte sich umso stärker bemerkbar, je mehr Prozessoren gleichzeitig über das eine Koppelmodul auf diesen Kommunikationsspeicher zugreifen wollten. Im Einzelfall wurden Unterschiede bis zu 27% festgestellt. Daher wurden in den folgenden Tabellen die erhaltenen Meßergebnisse nach den Prozessoren P_0 und P_{1-3} getrennt aufgeführt.

		Gesamtzeit [µsec]	**Zeit / Wort** [µsec]	**Worte / Zeit** [1 / sec]
1 Prozessor	**P_0**	**162 450**	**2,479**	**403 400**
	P_1,P_2,P_3	**166 000**	**2,533**	**394 800**
2 Prozessoren	**P_0**	**166 400**	**2,539**	**393 800**
	P_1,P_2,P_3	**167 000**	**2,548**	**392 400**
3 Prozessoren	**P_0**	**176 000**	**2,686**	**372 400**
	P_1,P_2,P_3	**176 000 177 000 178 000**	**2,686 2,701 2,716**	**372 400 370 300 368 200**
4 Prozessoren	**P_0**	**191 500**	**2,922**	**342 200**
	P_1,P_2,P_3	**211 000 224 000 243 500**	**3,220 3,418 3,716**	**310 600 292 600 269 100**

Tab. 6 Zugriff mehrerer Prozessoren auf den Kommunikationsspeicher mit einer Blocklänge von 65536 Worten (64 KWorte)

Um auch hier keine größere Meßgenauigkeit vorzutäuschen als tatsächlich vorlag, wurden die gewonnenen Meßwerte und ihre daraus abgeleiteten Größen auf drei bis vier relevante Stellen gerundet. Bei sehr unterschiedlichen Einzelergebnissen wurden mehrere Werte in der Tabelle angegeben. Diese Schwankungen sind vor allem dadurch zu erklären, daß im Koppelmodul bei jeder Messung nicht wiederholbare Vorgänge bezüglich der Arbitrierung stattfanden. Schon sehr kleine Differenzen im zeitlichen Zusammenspiel der zugreifenden Prozessoren konnten

unter bestimmten Umständen größere Unterschiede im Gesamtablauf eines Prozessors hervorrufen.

7.1.4 Zugriffe mehrerer Prozessoren auf den Stabilen Speicher

Für die Messung der Schreibtransfer-Zugriffe mehrerer Prozessoren auf einen Stabilen Speicher wurde die im letzten Abschnitt vorgestellte Hardware-Konfiguration um den Stabilen Speicher erweitert (siehe Abb. 44).

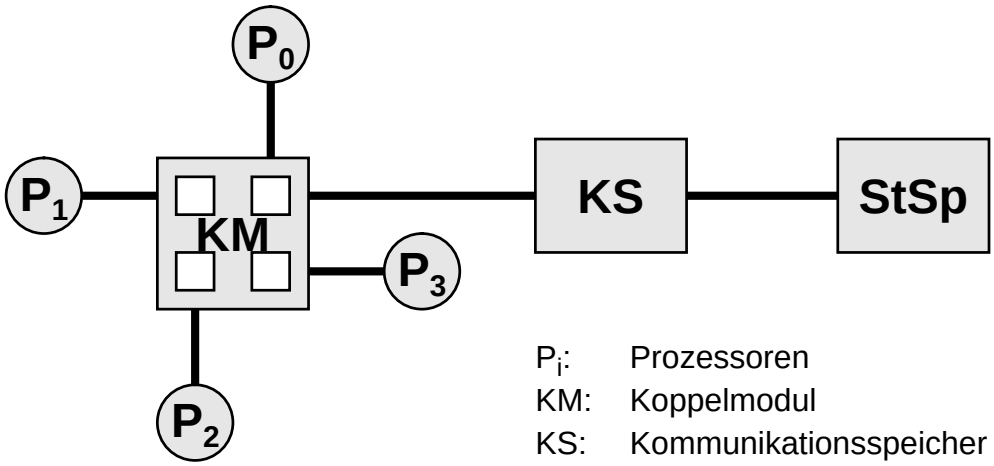


Abb. 45 Konfiguration zur Messung am Stabilen Speicher

Auch hier wurde das bereits vorgestellte Datentransferprogramm zum Schreiben eines Blocks in den Pufferbereich des Stabilen Speichers verwendet. In Tab. 7 sind die gewonnenen Ergebnisse dargestellt. Auch hier wurden bezüglich der dargestellten Meßergebnisse die gleichen Rundungen vorgenommen wie in der letzten Tabelle (Tab. 6), in der die Ergebnisse bei Zugriffen auf den Kommunikationsspeicher dargestellt wurden.

Bei der in Abschnitt 7.1.2 bereits vorgestellten Messung des Zugriffs eines einzelnen Prozessors auf den Pufferbereich war zu erkennen, daß die Vorgänge im Pufferbereich den dominierenden Anteil an der Gesamtdauer des Zugriffs haben. Dagegen spielt hier das Koppelmodul keine so große Rolle. Bei den nun durchgeführten Messungen mit mehreren gleichzeitig auf einen Stabilen Speicher zugreifenden Prozessoren wurde dies besonders deutlich. Bereits ab zwei zugreifenden Prozessoren hat die durchschnittliche Übertragungszeit für ein Wort um etwa 15% zugenommen. Deutlich stärker ist dies bei drei und vier gleichzeitig arbeitenden Prozessoren zu beobachten. Hier erhöht sich die mittlere Übertragungszeit um bis zu 134% gegenüber dem Ausgangswert.

Besonders auffallend war bei diesen Messungen der starke Unterschied in den Programmlaufzeiten bei drei Prozessoren. Je nachdem, ob der Prozessor und der Kommunikationsspeicher mit dem verbundenen Stabilen Speicher an derselben Ecke des Koppelmoduls angeschlossen sind oder nicht, ergaben sich sehr unterschiedliche Meßwerte. Der Grund für dieses Verhalten konnte nicht ermittelt werden. Aber auch alle Wiederholungen dieser Messung zeigten dieses Ergebnis.

		Gesamtzeit [µsec]	**Zeit / Wort** [µsec]	**Worte / Zeit** [1 / sec]
1 Prozessor	**P_0**	**248 550**	**3,793**	**263 700**
	P_1,P_2,P_3	**251 500**	**3,838**	**260 600**
2 Prozessoren	**P_0**	**288 150**	**4,379**	**227 400**
	P_1,P_2,P_3	**288 500** **290 000**	**4,402** **4,425**	**227 200** **226 000**
3 Prozessoren	**P_0**	**294 350**	**4,491**	**222 600**
	P_1,P_2,P_3	**536 000** **566 000** **568 300**	**8,179** **8,636** **8,672**	**122 300** **115 800** **115 300**
4 Prozessoren	**P_0**	**581 650**	**8,875**	**112 700**
	P_1,P_2,P_3	**581 650**	**8,875**	**112 700**

Tab. 7 WRITE-Zugriff mehrerer Prozessoren auf den Stabilen Speicher mit einer Blocklänge von 65536 Worten (64 KWorte)

In der folgenden Grafik (Abb. 46) sind die Ergebnisse aus den Zugriffen auf den Kommunikationsspeicher und auf den Stabilen Speicher gegenübergestellt. Hierbei wird besonders deutlich, daß die zuvor ermittelte maximale Übertragungskapazität zwischen Prozessoren und Stabilem Speicher (465000 Worte/sec) bei mehreren Zugreifern deutlich früher erreicht wird als bei den Zugriffen auf den Kommunikationsspeicher (117600 Worte/sec).

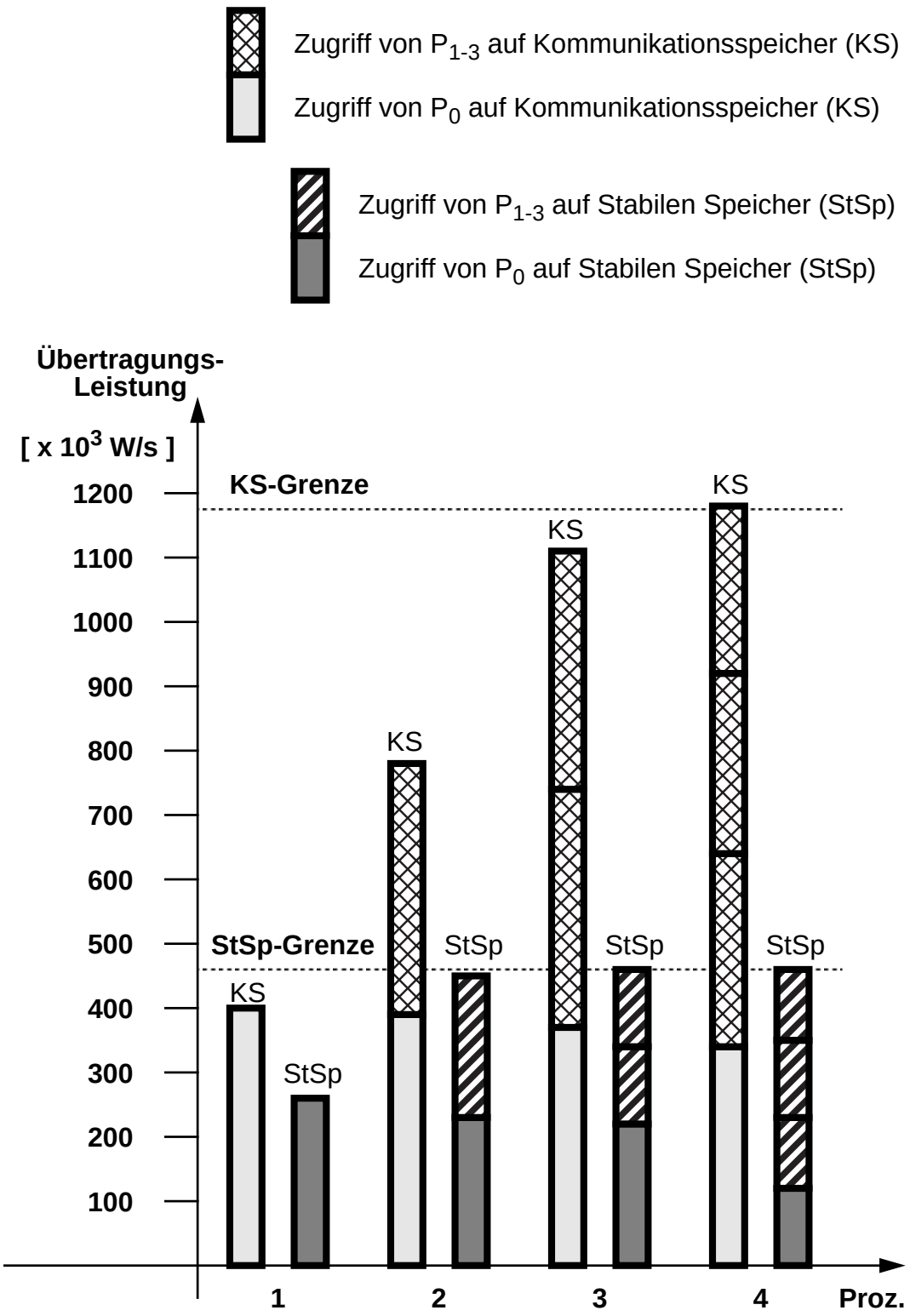


Abb. 46 Übertragungsleistung bei Transferoperationen in den Kommunikationsspeicher bzw. in den Stabilen Speicher

7.1.5 Bearbeitung der Aufträge durch die zentrale Steuerung

Die einzelnen Aufträge, die ein Auftraggeber dem Stabilen Speicher übergeben kann, unterscheiden sich sehr in ihrem Aufwand, der zu ihrer Ausführung aufgebracht werden muß. Die Aufträge CREATE, DELETE und RELEASE sind verwaltungstechnischer Art. Hier ist die Speicherverwaltung des Stabilen Speichers gefordert, um neue Speicherbereiche zur Verfügung zu stellen bzw. belegte Plätze wieder freizugeben. Ein Datentransfer findet hierbei nicht statt. Dennoch sind diese Aufträge abhängig von der Objektlänge. Das betrifft auch RELEASE, da hierbei alle Stabilen Objekte, die zu einer Kommunikationspage gehören, wieder gelöscht werden müssen. Die Aufträge WRITE_BI und READ_BI sind ebenfalls mit nur geringem Aufwand verbunden, denn hier muß unabhängig von der Objektlänge nur ein einzelnes Wort zwischen der Kommunikationspage und einem bereits bestehenden Stabilen Objekt transferiert werden.

Ein deutlich größerer Aufwand ist i.a. mit den WRITE- und READ-Aufträgen verbunden, denn hier muß ein Datenblock zwischen dem Pufferbereich und den Objektspeichern transportiert werden. Daher ist der Zeitbedarf für die Bearbeitung dieser Aufträge abhängig von der transportierten Datenmenge. Ähnliches gilt auch für den VERIFY-Auftrag.

Bei den vorgenommenen Messungen wurde für alle Auftragsarten die maximale Objektlänge von 1 MWort angenommen. Somit wurde die Maximaldauer der Aufträge untersucht. In Tab. 8 sind in der ersten Spalte die dabei ermittelten Zeiten angegeben. Dabei zeigte sich, daß die WRITE- und READ-Aufträge und der Auftrag VERIFY unerwartet lange Ausführungsdauern besaßen (5,5 - 16,5 sec). Daher wurden diese Aufträge genauer untersucht.

Bei den angesprochenen Aufträgen muß berücksichtigt werden, daß auch im fehlerfreien Fall pro übertragenem Wort vom Auftraggeber mehr als ein interner Transfer zwischen dem Pufferbereich und den Objektspeichern von der zentralen Steuerung ausgeführt werden muß.

Im einzelnen sind dies bei:

- WRITE : 7 Transfers:
  Lese altes Objekt, um herauszufinden, daß noch eine gültige Kopie vorliegt,
  Lese neues Objekt aus dem Puffer,
  Schreibe Objekt in den ersten Objektspeicher,
  Lese aus dem ersten Objektspeicher, um den Erfolg dieser Aktion zu prüfen,
  Lese erneut neues Objekt aus dem Puffer,
  Schreibe Objekt in den zweiten Objektspeicher,
  Lese aus dem zweiten Objektspeicher, um den Erfolg dieser Aktion zu prüfen.
- READ : 3 Transfers:
  Lese Objekt aus einem Objektspeicher,
  Schreibe Objekt in den Puffer,
  Lese aus dem Puffer, um den Erfolg dieser Aktion zu prüfen.

- CREATE_and_WRITE : 6 Transfers:
  Ablauf wie bei WRITE; nur das erste Lesen entfällt,
  da dieses Objekt bislang noch nicht angelegt ist.
- VERIFY : 2 Transfers:
  Lese aus dem ersten Objektspeicher,
  Lese aus dem zweiten Objektspeicher.

Diese Transferphasen sind Zugriffe auf Speicherbereiche, bei denen der Zugriff synchronisiert werden muß. Messungen ergaben, daß bei der zentralen Steuerung pro Schreibtransfer etwa 1,1 µsec und pro Lesetransfer etwa 0,9 µsec vergehen. Dagegen benötigen die sonstigen Aktionen, die bei einem solchen Aufträgen notwendig sind, wie die Schleifenbearbeitung, die Adreßerzeugung, die Überprüfung der Checksumme und der Vergleich im Anschluß an das Kontrolllesen, deutlich weniger Zeit, da sie lokal in der zentralen Steuerung ablaufen. Unter dieser Annahme dürften bei einem WRITE-Auftrag mit 1 MWort Datentransfer etwa 7 sec allein für die Transfers und weitere etwa 4 sec für die sonstigen Bearbeitungen veranschlagt werden. Diese etwa 11 sec wurden aber deutlich übertroffen (16,5 sec).

Um diesem Phänomen auf den Grund zu gehen, wurde der Ablaufs der innersten Programmschleife des WRITE-Auftrags genauer betrachtet. Dies ist der Transfer eines Wortes aus dem Pufferspeicher in einen Objektspeicher. Hier werden für ein Wort zwei Lesezugriffe und ein Schreibzugriff auf Puffer- bzw. Objektspeicher ausgeführt. Dabei stellte sich heraus, daß bei jedem Schleifendurchlauf neben den drei erwarteten internen Transfers zusätzlich 12 Lesezugriffe auf den Programmspeicher erfolgten. Dieser liegt außerhalb des Transputers und ist aus SRAM-Bausteinen aufgebaut. Bei einer Schleifendauer von 7,25 µsec entfielen 2,9 µsec auf diese zusätzlichen Zugriffe. Sie hatten damit einen Zeitanteil von 40% innerhalb der Schleife. Dem gegenüber entfielen auf die drei Zugriffe für den Transfer eines Worts vom Pufferspeicher in einen Objektspeicher und das Kontrollesen 2,9 µsec, was einen Zeitanteil von ebenfalls 40% bedeutet. Diese Situation hatte sich ergeben, da kein Instruktionscache zur Verfügung stand. Für die sonstigen Berechnungen innerhalb einer Schleife werden somit etwa 1,45 µsec benötigt (Zeitanteil von 20%).

Zur Verkürzung der Zeitdauer für einen Schleifendurchlauf wurde die Schleifenbearbeitung durch Optimierungen direkt im Assemblercode verkürzt. Dies verbesserte die Bearbeitungszeit jedoch nur um etwa 10%.

Eine deutlichere Reduzierung der Zeit konnte durch die Verlagerung des (optimierten) Assemblercodes in den auf dem Transputerchip befindlichen 4 KByte großen Speicher erreicht werden. Dadurch wird dieser Programmteil so schnell abgearbeitet, als würde der Code in einem Cache liegen. In diesem Fall benötigt die Schleife statt der ursprünglichen 7,25 µsec nur noch 4,50 µsec: 2,9 µsec für den Transfer (64%) und 1,6 µsec für die sonstigen Berechnungen (36%). Die Gesamtlaufzeit für die Bearbeitung des WRITE-Auftrags innerhalb der zentralen Steuerung verkürzte sich dadurch um 29,7% von 16,5 sec auf 11,6 sec.

Ähnliche Zeitreduzierungen wurden auch in den anderen mit einem Datentransfer verbundenen Aufträgen erreicht. Dies gilt auch für den VERIFY-Auftrag, der zwar nicht zu einem Datentransfer zwischen dem Auftraggeber und dem Stabilen Speicher führt, innerhalb der zentralen Steuerung aber mit einem zweifachen Zugriff auf das gesamte Stabile Objekt ausgeführt wird. In Tab. 8 sind ebenfalls die optimierten Fassungen dieser Aufträge (in der 2. Spalte) angegeben.

Alle oben angesprochenen Angaben beziehen sich auf den fehlerfreien Fall. Im Fehlerfall kann es zu sehr unterschiedlichen und i.a. auch deutlich längeren Bearbeitungszeiten kommen. Bei diesen Situationen wurde auch auf eine Optimierung und auf die Verlagerung des Programmcodes in den Transputerchip verzichtet, da der hierfür erforderliche Programmcode zu groß für das On-Chip-RAM ist.

	Bearbeitungsdauer (bei 1 M Worten)	
	nicht optimiert	**optimiert**
RELEASE (kein Obj. zu löschen)	**0,3 msec**	
CREATE	**15,5 msec**	
DELETE	**15,7 msec**	
CREATE_and_WRITE	**13 703,7 msec**	**9 770,0 msec**
WRITE	**16 512,0 msec**	**11 642,2 msec**
READ	**6 771,4 msec**	**4 872,5 msec**
VERIFY	**5 528,0 msec**	**3 768,0 msec**
WRITE_BI	**0,4 msec**	
READ_BI	**0,3 msec**	

Anmerkung: Die Meßgenauigkeit betrug etwa ±0,06 msec.

Tab. 8 Auftragsbearbeitung durch die zentrale Steuerung

Die vier verbesserten Transferoperationen wurden auch mit verschiedenen Blocklängen im Bereich von 1 KWorten bis 1 MWorten gemessen, um feststellen zu können, ob die Ausführungs-

dauern allein von der Objektlänge abhängen oder nicht. Dabei konnte für den Bereich zwischen 10 KWorten und 1 MWorten eine exakte lineare Abhängigkeit festgestellt werden. Nur im Bereich noch kleinerer Objektlängen wurde davon nur um bis zu 20% abgewichen. Daher kann für jeden dieser Aufträge eine nahezu konstante Bearbeitungsleistung angegeben werden (siehe Tab. 9).

	Zeit / Wort [µsec]	**Worte / Zeit** [1 / sec]
CREATE_and_WRITE	**9,317**	**107 300**
WRITE	**11,103**	**90 100**
READ	**4,647**	**215 200**
VERIFY	**3,593**	**278 300**

Tab. 9 Bearbeitungsrate der zentralen Steuerung bei den optimierten Transfer-Operationen

Die ermittelten Ergebnisse sehen deutlich schlechter aus als die Werte, die bei der Übertragung vom Prozessor in den Pufferbereich des Stabilen Speichers gemessen wurden. Dabei ist aber zu berücksichtigen, daß bei den Zugriffen der zentralen Steuerung auf die Objektspeicher und auf den Pufferspeicher genauso wie bei Zugriffen von Arbeitgebern auf den Stabilen Speicher eine Synchronisierung der Steuersignale nötig ist. Dies führt zu Zeitverlusten. Auch die Arbeit des Voters nimmt eine gewisse Zeit in Anspruch. Selbst wenn keine Unterschiede in den Adressen und (bei Schreibzugriffen) in den Daten vorliegen und es auch keinerlei zeitliche Diskrepanz bei der Signalisierung des gewünschten Speicherzugriffs gibt, werden bereits 0,2 µsec für das Voten der Adresse und bei Schreibzugriffen weitere 0,2 µsec für das Voten der Daten benötigt. Da sich dies bei jedem Speicherzugriff ereignet, wäre der Einsatz eines Voters, der die drei Vergleiche für 32-bit-Worte in einem Takt ausführen könnte, sehr vorteilhaft. Leider war ein solcher Chip (auch als 8-bit-Version) nicht verfügbar.

Ferner kommt hinzu, daß pro Wort, das von einem Auftraggeber in den Pufferspeicher geschrieben wurde, mehrere Worte (2 - 7) von der zentralen Steuerung zwischen den Objektspeichern, dem Pufferspeicher und dem Transputer transportiert werden müssen. Zieht man dieses in Betracht, so läßt sich für die verschiedenen Auftragsarten die “Zahl der internen Transfers” pro Zeiteinheit bestimmen. Aus den Werten der Tab. 9 und der zu Beginn dieses Abschnitts gemachten Angaben zu der Anzahl der internen Transfers lassen sich somit die in Tab. 10 dargestellten “Internen Transferraten” ableiten.

Nach diesen Werten wird bei der Ausführung der transportintensiven Aufträge im Durchschnitt etwa 1,56 µsec pro internem Transfer benötigt. Dabei entfallen etwa 1,0 µsec (64%) auf einen Speicherzugriff inklusive Synchronisation und Voting und die restlichen 0,56 µsec (36%) auf sonstige lokale Aktivitäten.

	Worte / Zeit [1 / sec]	int. Transf.	Transf. / Zeit [1 / sec]	Zeit / Transf. [µsec]
CREATE_WR	107 300	6	643 800	1,553
WRITE	90 100	7	630 700	1,586
READ	215 200	3	645 600	1,549
VERIFY	278 300	2	556 600	1,797

Tab. 10 Interne Transferraten der zentralen Steuerung

Weiterhin wurde untersucht, ob die einzelnen Knoten des TMR-Verbundes ab und zu aus der Taktsynchronität herauslaufen, so daß die Synchronisierungsfähigkeit der Voter erforderlich ist oder nicht. Dabei hat sich gezeigt, daß tatsächlich bei jedem zweiten bis dritten Zugriff eine Synchronisierung erfolgte. Die maximale Abweichung betrug dabei in fast allen Fällen nur einen Takt. Größere Abweichungen waren fast immer mit dem Ausbrechen eines Transputers aus der synchronen Arbeitsweise verbunden. Die Wiederaufnahme der synchronen Arbeitsweise dieses Transputers zu den beiden anderen Transputern nach Abschluß einer Auftragsbearbeitung erfolgte innerhalb einer Sekunde.

7.1.6 Zugriffskonflikte am Pufferspeicher

Wenn die Prozessoren des Host-Systems Zugriffe auf das Transferregister ausführen, wird von der Puffersteuerung jedesmal intern ein Zugriff auf den Pufferspeicher durchgeführt. Andererseits greift auch die zentrale Steuerung häufig auf den Pufferspeicher zu, wenn sie mit der Bearbeitung eines READ- oder WRITE-Auftrags beschäftigt ist. Diese Zugriffe auf den Pufferspeicher stehen somit in Konkurrenz. Ob die daraus erwachsende gegenseitige Behinderung von Bedeutung ist oder nicht, läßt sich durch die Untersuchung der beiden parallel durchgeführten Vorgänge ermitteln.

Greift ein Auftraggeber auf das Transferregister zu, so ist der Kommunikationsspeicher damit für eine Zeit von 1,75 µsec (bei WRITE) bzw. von den 1,95 µsec (bei READ) damit beschäftigt (siehe Tab. 5 und Tab. 4). Innerhalb des Pufferbereichs wird für die Arbitrierung und den Zu-

griff auf den Pufferspeicher etwa 0,4 µsec benötigt. Das sind 23% bzw. 20,5% der Zeit, die mindestens für einen Zugriff auf das Transferregister vergeht. Dieser Anteil ist die Obergrenze für den Fall, daß alle angeschlossenen Auftraggeber intensiv mit einer Datenübertragung zu diesem Stabilen Speicher beschäftigt sind.

Dem gegenüber greift die zentrale Steuerung des Stabilen Speichers bei der Bearbeitung der READ-Aufträge zweimal und bei der Ausführung der WRITE-Aufträge einmal pro Durchlauf durch die innerste Schleife auf den Pufferspeicher zu. Auch hier wird für jeden Zugriff auf den Pufferspeicher inklusive Arbitrierung etwa 0,4 µsec benötigt. Die Schleife hat eine Zykluszeit von 4,5 µsec (siehe Abschnitt 7.1.5). Somit beträgt der Zeitanteil der zentralen Steuerung an der konfliktträchtigen Situation 0,4 bzw. 0.8 µsec von 4,5 µsec. Das sind 9% bzw. 18%.

Die Gegenüberstellung dieser Zahlen zeigt, daß auch unter den ungünstigsten Umständen nur in sehr wenigen Fällen (unter 5%) mit einem Konflikt beim Zugriff auf den Pufferspeicher zu rechnen ist.

7.1.7 Wartezeit der Aufträge auf ihre Bearbeitung

Die Zeit, die ein Auftrag in der Warteschlange (im Auftrags-FIFO) verbringt, ist schwierig anzugeben. Dies hängt in erster Linie davon ab, welche Aktivitäten die verschiedenen Auftraggeber etwa zur gleichen Zeit an den Stabilen Speicher übergeben.

Solange immer nur ein Auftraggeber einen Auftrag vom Stabilen Speicher bearbeiten läßt, fällt praktisch keine Wartezeit an. Nur ein gerade laufender VERIFY für ein Stabiles Objekt, das der Stabile Speicher selbst gestartet hat, muß zuvor abgewartet werden. Die Gesamt-Ausführungsdauer für die Bearbeitung dieses Auftrags läßt sich somit aus der Objektlänge und den Angaben in den Tabellen Tab. 8 und Tab. 9 berechnen.

In den Situationen, in denen mehrere Auftraggeber Aufträge an den Stabilen Speicher übergeben, lassen sich nur grobe Abschätzungen angeben, wie lange dieser in der Warteschlange verbleibt. Um Aussagen über die voraussichtliche Wartezeit machen zu können, werden folgende Annahmen gemacht:

- der Stabile Speicher wird von vier Auftraggebern gleichzeitig benutzt;
- diese vier Auftraggeber erteilen annähernd zur gleichen Zeit gleichartige Aufträge mit gleich langen Objekten; diese Annahme entspricht etwa der Situation, in der in den vier Prozessorknoten nach einer globalen Synchronisation ein Sicherungspunkt angesprochen (meistens geschrieben) werden soll;
- der Stabile Speicher ist zu Beginn des ersten Auftrags mit einem (selbständig gestarteten) VERIFY eines Stabilen Objekts beschäftigt, das die gleiche Größe besitzt wie die Objekte, die nun bei den Aufträgen benutzt werden;
- der Verlauf aller Aufträge erfolgt fehlerfrei.

Aus dem annähernd gleichzeitigen Start der vier gleich großen Aufträge ergibt sich, daß sie alle etwa zur gleichen Zeit in das Auftrags-FIFO eingetragen werden. Ihre Reihenfolge ist dabei völlig zufällig. Somit muß ein Auftrag die Bearbeitung von 0 - 3 anderen Aufträgen abwarten, bis er selbst an der Reihe ist. Damit ergibt sich eine mittlere Wartezeit von 1,5 Auftragsbearbeitungen. Erhöht wird dieser Wert durch eine zusätzliche Wartezeit, die sich aus der noch ausstehenden Beendigung des zunächst laufenden VERIFY-Auftrags ergibt. Im Mittel muß die Zeit für den halben VERIFY-Auftrag abgewartet werden, bevor der erste neue Auftrag gestartet werden kann. Diese Zeit ist zwar für jeden der folgenden Aufträge gleich lang, gemessen an den Bearbeitungszeiten der verschiedenen “Nutz”-Aufträge haben sie aber einen unterschiedlichen Anteil. Aus den Werten in Tab. 9 läßt sich ableiten, daß sich bei den folgenden “Nutz”-Aufträgen folgende Werte für die gesamte mittlere Wartezeit ergeben:

- CREATE_and_WRITE : 1,5 + 0,19 = 1,69 Auftragsbearbeitungen
- WRITE : 1,5 + 0,16 = 1,66 Auftragsbearbeitungen
- READ : 1,5 + 0,39 = 1,89 Auftragsbearbeitungen
- VERIFY : 1,5 + 0,50 = 2,00 Auftragsbearbeitungen

Die Maximalwerte sind gerade doppelt so groß.

Alle anderen Aufträge sind verwaltungstechnischer Art und benötigen für ihre Bearbeitung nur sehr wenig Zeit. Kommt vor ihnen ein Auftrag mit Datentransfer an die Reihe, so müssen sie im Verhältnis zu ihrer eigenen Bearbeitungszeit unverhältnismäßig lange warten. Daher macht es keinen Sinn, für sie entsprechende Zeitangaben zu machen wie bei den oben dargestellten Aufträgen.

7.2 Fehlersituationen

Der Stabile Speicher muß mit verschiedenen Fehlersituationen zurechtkommen. Dazu wurden verschiedene fehlerhafte Zugriffe provoziert, welche die (fehlerhaften) Auftraggeber in dem Stabilen Speicher ausführen wollten :

- Zugriff der Prozessoren mit Übertragungsfehlern (einzeln umgekipptes Bit),
- Zugriff auf unerlaubte Register,
- Zugriff der Prozessoren mit falschen Daten (Paßwort),
- Zugriffe auf das Transfer-Register in einer falschen Situation oder in der falschen Richtung.

In allen Fällen reagiert die Puffersteuerung wie erwartet mit der Nichtausführung der gewünschten Operation. Die Puffersteuerung gibt ein Ready-Signal zurück, führt aber nichts aus. Bei einem Übertragungsfehler wird auch das Error-Signal aktiviert. Die Ausführungszeiten sind

dabei kürzer als bei der Ausführung mit korrekten Daten.

Weitere Untersuchungen beschäftigten sich mit Fehlersituationen, die innerhalb des Stabilen Speichers betrachtet werden:

- falsche Angaben zum Auftrag,
- Übertragung eines fehlerhaften Datenblocks,
- Veränderungen in den abgelegten Daten,
- vorübergehender Ausfall der zentralen Steuerung,
- vorübergehender Ausfall eines Speicherbereichs (Pufferbereich, Objektspeicher).

Diese Fehler wurden alle erkannt und konnten geeignet behandelt werden. Dabei war sichergestellt, daß zum Abschluß der Fehlerbehebung immer ein konformes Stabiles Objekt in zweifacher Kopie vorlag, was hier die Voraussetzung für eine umfassende Fehlertoleranz ist.

Nun wird näher auf die oben aufgeführten Fehlersituationen eingegangen.

Zur Feststellung der korrekten Arbeitsweise des Auftraggebers vergleicht die zentrale Steuerung als erstes alle Angaben, die der Auftraggeber zum Objekt gemacht hat, mit den Angaben, die der Stabile Speicher bei der Erzeugung des angesprochenen Stabilen Objekts bei sich abgelegt hatte. Besteht eine Diskrepanz, oder handelte es sich um einen ungültigen Auftrag, so wird dies im Ergebnisregister der betreffenden Kommunikationspage eingetragen und der Auftrag beendet. Das angesprochene Stabile Objekt wird dabei nicht verändert. Dafür ist die zentrale Steuerung weniger als eine Millisekunde lang beschäftigt.

Die Übertragung eines fehlerhaften Datenblocks bei einem WRITE-Auftrag wird von der zentralen Steuerung mit Hilfe der Checksumme überprüft. Zur Feststellung eines konformen Datenblocks wird während der Übertragung des Datenblocks vom Pufferspeicher in den ersten Objektspeicher die Checksumme von der zentralen Steuerung neu berechnet und mit der übertragenen Checksumme verglichen. Der Zeitpunkt der Fehlererkennung befindet sich somit am Ende der ersten Update-Phase. Ist eine Korrektur nötig, so muß nun die Backup-Kopie auf die fehlerhafte Kopie übertragen werden, um den alten Objektzustand wiederherzustellen. Dieser Transfer (2.Kopie $\rightarrow$ 1.Kopie) erfordert genauso viele interne Transfers, wie nun bei einer erfolgreichen Ausführung des WRITE-Auftrags benötigt worden wäre (Pufferbereich $\rightarrow$ 2.Kopie). Daher ist die Dauer für diesen Ablauf identisch zu der erfolgreichen Auftragsausführung (siehe auch Abschnitt 7.1.5, Bearbeitung durch die zentrale Steuerung).

Bei READ-Aufträgen könnte von der zentralen Steuerung auf die Überprüfung der Checksumme des auszugebenden Datenblocks verzichtet werden, denn das auszugebende Stabile Objekt, das zuvor beschrieben worden war, wurde dabei bereits korrekt abgespeichert. Erfolgte dieser READ-Auftrag auf ein angelegtes, aber bislang unbeschriebenes Objekt, so wird dies auf Grund der Eintragungen in der Objektbeschreibung von der zentralen Steuerung erkannt und eine entsprechende Information im Ergebnisregister eingetragen. Die Ausführungsdauer beträgt im zu-

letzt beschriebenen Fall unter einer Millisekunde.

Dennoch wurde die Überprüfung der Checksumme bei READ-Aufträgen beibehalten, da sie sich kaum in der Ausführungszeit bemerkbar gemacht hat. Denn für Meßzwecke hatte dies einen Vorteil, da mit ihrer Hilfe die Reaktion der zentralen Steuerung auf einzelne Bitfehler überprüft und ihr Zeitverhalten ermittelt werden konnte. Beim Auftreten von Parityfehlern wird nämlich ein ganz ähnlicher Ablauf ausgeführt, wie es nun beim Auftreten eines internen Checksummenfehlers der Fall ist. Ein Checksummenfehler läßt sich in einem Testmodus leicht durch das nachträgliche Schreiben eines Wortes provozieren, während dies bei den Paritybits nicht möglich ist.

Der ähnliche Ablauf beim Auftreten von Parityfehlern und der beim Checksummenfehler rührt daher, daß auftretende Parityfehler zwar bei jeden Zugriff der zentralen Steuerung in jeder einzelnen Einheit der TMR-Steuerung entdeckt werden, aber nicht unmittelbar an den jeweiligen Prozessor weitergegeben werden. Somit führen sie nicht unmittelbar zu einer Wiederholung des Transfers. Stattdessen wird diese Information über einen Parityfehler von den drei Einheiten der TMR-Steuerung bei jedem Speicherzugriff über einen separaten Voter (3 x 1 Bit) geführt und das Mehrheitsergebnis mit dem bisherigen Resultat verknüpft. Am Ende eines Datenblocktransfers wird diese Information ausgewertet und das Summenbit für die Parityfehler wieder zurückgesetzt. War kein Fehler aufgetreten, so ist das Gewünschte eingetreten. Die hierbei ermittelte Ausführungszeit ist in den Tabellen 8 - 10 eingetragen. Ist dagegen ein Fehler festgestellt worden, so wird der gesamte Transfer wiederholt, als Quelle jedoch der andere Objektspeicher verwendet. Ist diese Kopie des Stabilen Objekts korrekt, so ergibt sich eine doppelt lange Ausführungszeit für die Bearbeitung des READ-Auftrags.

War jedoch auch die zweite Kopie mit einem Parityfehler behaftet, so muß bereits während der Ausführung des READ-Auftrags korrigiert werden. Um die genaue Position für das umgekippte Bit zu finden, wird das gesamte Stabile Objekt aus beiden Objektspeichern nochmals transferiert, nun aber nach jedem einzelnen Wort das Summenbit für den Parityfehler überprüft.

Das Programm für diese intensive Fehlerbehebung steht im Gegensatz zur schnellen READ-Routine nicht im On-Chip-RAM des Transputers. Aus diesem Grund und wegen der Überprüfung nach jedem Wort kam es zu einer deutlichen Verschlechterung der resultierenden Transferleistung um den Faktor fünf bis sechs. Unter der Annahme, daß umgekippte Bit sehr selten auftreten, ist diese Vorgehensweise jedoch akzeptabel.

Ganz ähnliche Vorgänge laufen auch bei der Ausführung des VERIFY-Auftrags mit einem fehlerhaften Datenblock ab.

Bei den vorübergehenden Ausfällen einer Teilkomponente (zentrale Steuerung, Pufferbereich, ein Objektspeicher) steht nicht eine schnelle Ausführung im Vordergrund sondern eine sichere Bearbeitung. Daher wurde in diesem Fall auf Zeitangaben verzichtet.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

In dieser Arbeit wurde ein Stabiler Speicher vorgestellt, der für das speichergekoppelte Multiprozessorsystem MEMSY entwickelt worden ist. Da als Speichermedium RAM-Bausteine eingesetzt wurden, konnte ein schneller Zugriff gewährleistet werden.

Die Topologie von MEMSY und seine Realisierung hatten großen Einfluß auf die Details bei der Implementierung des Stabilen Speichers. Dennoch sind viele Details in der Problemstellung nicht MEMSY-typisch sondern treten auch bei anderen speichergekoppelten Multiprozessorsystemen auf. Daher lassen sich die meisten hier gefundenen Lösungen auch auf andere Systeme übertragen.

Charakteristisch für diesen Stabilen Speicher ist seine Zweiteilung bei der Bearbeitung von Zugriffswünschen der Auftraggeber. Während sich im Pufferbereich bei jedem Speicherzugriff die Auftraggeber ständig abwechseln können, wird von der zentralen Steuerung immer nur ein Auftrag, also i.a. ein Stabiles Objekt, komplett (über eine längere Zeit) bearbeitet. Für die Auftraggeber entsteht vor allem bei der Datenübertragung zum Schreiben eines Objekts der Eindruck einer ungehinderten Ansprechbarkeit des Stabilen Speichers; und für das sichere Abspeichern eines Datenblocks ist nur ein einmaliger Transfer von den Auftraggebern nötig. Alle weiteren Arbeiten wie das doppelte Ablegen und die Kontrolle über das erfolgreiche Schreiben werden vom Stabilen Speicher ohne Mitwirkung des Auftraggebers durchgeführt.

Der Nachteil dieser Lösung ist allerdings, daß bis zur Beendigung des Auftrags eine Wartezeit in Kauf genommen werden muß, auf deren Länge der Auftraggeber keinen Einfluß hat, und die er auch nur sehr schwer abschätzen kann. Will er sich von der ordnungsgemäßen Abwicklung des Auftrags überzeugen, muß er diese Zeit abwarten. Gleiches gilt auch für das Lesen eines Stabilen Objekts.

Bei der internen Bearbeitung konzentriert sich der Stabile Speicher jeweils auf ein einziges Objekt. Daher ist in einer Fehlersituation i.a. nur ein einziges Stabiles Objekt davon betroffen und muß innerhalb des Stabilen Speichers (ohne Einwirkung durch einen Auftraggeber) korrigiert werden.

Mit der Speicherkopplung waren jedoch einige Probleme bei der Identifizierung der zugreifenden Prozessoren verbunden. Die größte Herausforderung stellte sich darin, daß auf Grund der Verwendung eines schaltbaren Verbindungssystems für den Stabilen Speicher keine Möglichkeit bestand, die einzelnen zugreifenden Prozessoren direkt voneinander zu unterscheiden. Dies machte einen Paßwortschutz erforderlich und die Überprüfung der Angaben des Auftraggebers zu seinem Auftrag. Dadurch konnten fehlerhafte Zugreifer bereits vor dem Start einer Aktion auf gespeicherte Stabile Objekte mit hoher Wahrscheinlichkeit erkannt und deren Zugriff wir-

kungslos gemacht werden.

Deutlich größer war die Latenzzeit zum Erkennen eines fehlerhaft übertragenen Datenblocks. Dennoch konnte bei Schreibzugriffen durch die zweifache Speicherung des Datenblocks im Stabilen Speicher der alte Zustand des Stabilen Objekts wiederhergestellt werden. Diese doppelte Abspeicherung war auch die Basis für die Rekonstruktion eines zum Teil fehlerhaften Objekts. Die Fehler konnten dabei vom auftraggebenden Prozessor selbst kommen, der auf Grund eines eigenen Defekts den Datenblock fehlerhaft übertragen hat; oder ein defekter Nachbarprozessor oder eine Störung bei der Wegewahl vom Prozessor zum Stabilen Speicher war für den Fehler verantwortlich. Auch spontane Veränderungen in den abgelegten Daten und der vorübergehende Ausfall eines Objektspeichers konnte damit toleriert werden.

Der wegen seiner “Allmacht” sensibelste Teil des Stabilen Speichers gegenüber unerkannten Fehlern ist die zentrale Steuerung. Hier wurde durch den Einsatz eines TMR-Systems eine unmittelbar wirkende Fehlertoleranzmaßnahme implementiert. Ein Ausfall eines Knotens der TMR-Schaltung kann somit toleriert werden, ohne die Bearbeitung eines Auftrags in einem Zustand unterbrechen zu müssen, in dem ein Stabiles Objekt nicht als zweifache Kopie vorliegt. Erst nach dessen “Sicherstellung” wird eine Fehlerbehebungsmaßnahme eingeleitet.

Somit kann bei praktisch allen auftretenden Einfachfehlern zunächst ein definierter und für das Stabile Objekt sicherer Bearbeitungszustand erreicht werden, bevor eine Fehlerbehebung gestartet wird. Erst beim Auftreten mehrerer oder massiver Fehler (z.B. vorübergehender Ausfall einer Platine) ist man gezwungen, in einem nicht sehr sicheren Objektzustand eine Fehlerdiagnose und -behebung durchzuführen.

Nicht gefeit ist dieser Stabile Speicher allerdings vor Fehlern, die dazu führen, daß das auf den Prozessoren des Host-Systems ausgeführte Programm auf Grund eines einzigen dort aufgetretenen Fehlers einen ungültigen Pfad einschlägt. Werden dabei Zugriffe auf den Stabilen Speicher ausgeführt, kann dies der Stabile Speicher (unter Umständen) nicht als Fehler erkennen, solange alle dabei gemachten Angaben zueinander konform sind.

Die Gefahr ist deshalb verhältnismäßig groß, weil dabei mit hoher Wahrscheinlichkeit gültige Parameterdaten verwendet werden, aber zu diesem Zeitpunkt falsche Aufträge vergeben werden können.

Hier muß z.B. ein Watchdog dafür sorgen, daß dieser Fehler rechtzeitig bemerkt wird und ein Auftrag an den Stabilen Speicher in fehlerhafter Situation unterbleibt. “Rechtzeitig” muß in diesem Fall sehr eng begrenzt sein, denn bereits ein Auftrag der Form “DELETE (Objekt-Nummer, Objekt-Länge)” kann innerhalb weniger Speicherzugriffe erteilt werden und läßt sich nach seiner Übergabe an den Stabilen Speicher weder aufhalten noch rückgängigmachen.

8.2 Ausblick

Betriebssystem-Entwickler sehen in der Forderung, die Fertigstellung eines Auftrags immer wieder abfragen zu müssen, eine nicht adäquate Kooperation des Anwenders bzw. des Betriebssystems mit dem Stabilen Speicher. Daher ist die Frage neu zu überdenken, ob nicht doch eine Interruptmöglichkeit vom Stabilen Speicher zu den Prozessoren des Host-Systems geschaffen werden sollte. Ein Interrupt sollte immer dann an alle Prozessoren erfolgen, wenn ein Auftrag beendet worden ist, oder wenn sich ein Schaden im Stabilen Speicher ereignet hat.

Mit der zusätzlichen Verbindung büßt man allerdings ein, daß allein die bereits bestehenden Zugriffsmöglichkeiten der Prozessoren auf diesen Speicherbereich, also die Topologie des Host-Systems, die Kooperation mit dem Stabilen Speicher bestimmt. Nun müßten möglicherweise sehr viele, vielleicht auch überflüssige, separate Verbindungen geschaffen werden.

Im Bereich der Protokolle, die eine sichere Aufbewahrung und einen sicheren Update der Stabilen Objekte gewährleisten, und bei der Datenübertragung kann in Hardware und Software viel erreicht werden. “Kleinere Fehler” wie das Umkippen einzelner Bits, der Übertragung eines falsch zusammengesetzten Datenblocks oder der Versuch eines fehlerhaften Prozessors, Operationen auf den Stabilen Objekten durchzuführen, können hierbei erkannt und (durch Zugriffsverhinderung oder durch Reparatur) behandelt werden.

Problematischer sind jedoch die “Großen Fehler”, bei denen ganze Teileinheiten ausfallen können. Besonders kritisch sind beim Stabilen Speicher die Objektspeicher. So kann ein Stromausfall zum vollständigen Verlust der Daten führen. Gegen solch einen Fehler können Magnetplatten helfen, der Zugriff auf diese ist jedoch im Vergleich zu RAM-Bausteinen sehr langsam, da im wesentlichen die Plattenrotation und die Armbewegungen die Transferkapazitäten begrenzen [SIE93] [PAH90].

Alternativ könnte die Verwendung von Flash-Speichern, die wie DRAM-Speicher lesbar sind und bei Stromausfall nicht ihre gespeicherte Information verlieren, Vorteile bringen. Der Nachteil ist jedoch, daß das Schreiben eines Flash-Speichers nur blockweise und mit geringerer Geschwindigkeit durchgeführt werden kann. Dazu müßten die Zugriffsfolgen auf die Objektspeicher anders organisiert werden als bisher, bei denen auf jedes Schreiben ein Kontrollesen folgt. Allerdings könnte der bereits vorhandene DRAM-Pufferspeicher die Rolle eines Schreibpuffer übernehmen [OKO94].

9 Literatur

[AMD29] Advanced Micro Devices: “Am29C982 / Am29C983, Multiple Bus Exchange Handbook”, Sunnyvale, California, 1988.

[ANL81] Anderson, T.; Lee, P.A.: “Fault Tolerance, Principles and Practice”, Prentice Hall, Englewood Cliffs, New Jersey, 1981.

[BAN91] Banâtre, M.; Muller, G.; Rochat, B.; Sanchez, P.: “Design Decisions for the FTM: A General Purpose Fault Tolerant Machine”, In Proceedings of 21th International Symposium on Fault-Tolerant Computing Systems, Montreal, Canada, S. 71-78, Juni 1991.

[BER88] Bernstein, P.A.: “Sequoia: A Fault-Tolerant Tightly Coupled Multiprocessor for Transaction Processing”, Computer, Vol. 21, Nr. 2, S. 37-45, Februar 1988.

[BOP93] Bowen, N.S.; Pradhan, D.K.: “Processor- and Memory-Based Checkpoint and Rollback Recovery”, Computer, Vol. 26, Nr. 2, S. 22-31, Februar 1993.

[CHL85] Chandy, K.M.; Lamport, L.: “Distributed Snapshots: Determining Global States of Distributed Systems”, ACM Transactions on Computer Systems, Vol. 3, Nr. 1, S. 63-75, Februar 1985.

[CYP92] Cypress Semiconductor: “CMOS/BiCMOS Data Book”, Cypress Semiconductor Corporation, San Jose, California, USA, 1992.

[DAL94] Dal Cin, M.; Dalibor, S.; Eckert, T.; Grygier, A.; Hessenauer, H.; Hildebrand, U.; Hönig, J.; Hofmann, F.; Hohl, W.; Linster, C.U.; Michel, E.; Pataricza, A.; Sieh, V.; Thiel, T.; Turowski, S.: “Architecture and Realization of the Modular Expandable Multiprocessor System MEMSY”, Proceedings of Conference on Massively Parallel Computing Systems (MPCS’94), Ischia, Mai 1994.

[DCH91] Dal Cin, M.; Hohl, W. (Hrsg.): “Fault-Tolerant Computing Systems, Tests, Diagnosis, Fault Treatment”, Proceedings of 5th International Conference on Fault-Tolerant Computing Systems, Nürnberg, Informatik Fachberichte 283, Berlin, Springer, 1991.

[DCP94] Dal Cin, M.; Pataricza, A.: “Increasing Dependability in Multiprocessors”, Proceedings of the 8th Symposium on Microcomputer and Microprocessor Applications, Budapest, S. 55-64, Oktober 1994.

[DGH93a] Dal Cin, M.; Grygier, A.; Hessenauer, H.; Hildebrand, U.; Hofmann, F.; Linster, C.U.; Thiel, T.; Turowski, S.: “MEMSY - A Modular Expandable Multiprocessor System”, in Bode, A.; Dal Cin, M. (Eds.): “Parallel Computer Architectures - Theory, Hardware, Software, Applications”, Lecture Notes in Computer Science (LNCS) 732, Berlin, Springer, S. 15-30, 1993.

[DGH93b] Dal Cin, M.; Grygier, A.; Hessenauer, H.; Hildebrand, U.; Hönig, J.; Hohl, W.; Michel, E.; Pataricza, A.: “Fault Tolerance in Distributed Shared Memory Multiprocessors”, in: Bode, A.; Dal Cin, M. (Eds.): “Parallel Computer Architectures - Theory, Hardware, Software, Applications”, Lecture Notes in Computer Science (LNCS) 732, Berlin, Springer, S. 31-48, 1993.

[DHL89] Dal Cin, M.; Hildebrand, U.; Hohl, W.; Lehmann-Emilius, L.; Michel, E.: “Mechanismen zur Fehlerdiagnose und -behebung für die MEMSY-Hochleistungsstruktur”, Proceedings SFB Workshop ‘Grundlagen verteilter und paralleler Systeme’, Pommersfelden 16.-17. November 1989, Arbeitsberichte des IMMD, Band 22, Nr. 13, S. 113-130, November 1989.

[ECH90] Echtle, K.: “Fehlertoleranzverfahren”, Studienreihe Informatik, Berlin, Springer, 1990.

[FAST85] Fairchild: “FAST: Fairchild Advanced Schottky TTL, Data Book”, Fairchild Camera and Instrument Corporation, Maine, Portland, USA, 1985.

[FRH89] Fritsch, G.; Henning W.; Hessenauer, H.; Klar, R.; Linster, C.U.; Oehlrich, C.W.; Schlenk, P.; Volkert, J.: “Distributed Shared Memory Multiprocessor Architecture MEMSY for High Performance Parallel Computations”, Computer Architecture News, Dezember 1989.

[GÖR89] Görke, W.: “Fehlertolerante Rechensysteme”, Handbuch der Informatik, Band 2.1, Oldenbourg Verlag, München, 1989.

[GRY90] Grygier, A.: "Kommunikationsstrukturen in Multiprozessorsystemen am Beispiel des MEMSY-Hochleistungssystem", erschienen in: Ecker, K. (Hrsg.), "Workshop über Parallelverarbeitung", Lessach (Österreich), 18.9.-22.9.1989, Informatik-Bericht 90/1, Institut für Informatik, Clausthal-Zellerfeld, S. 91-102, 1990.

[HÄN76] Händler, W.; Herzog, U.; Hofmann, F.; Schneider, H.J.: “A General Purpose Array with Broad Spectrum of Applications”, Computer Arcitecture, Informatik Fachberichte 4, Berlin, Springer, S. 311-335, 1976.

[HAM86] Hamming, R.W.: “Coding and Information Theory”, 2. Edition, Prentice Hall, Englewood Cliffs, New Jersey, 1986.

[HAR90] Hardell, W.R.; Hicks, D.A.; Howell, L.C.; Maule, W.E.; Montoye, R.; Tuttle, D.P.: “Data Cache and Storage Control Units” , IBM RISC System/6000 Technology, S. 44-51, IBM, 1990.

[HIL91] Hildebrand, U.: “A Fault Tolerant Interconnection Network for Memory-Coupled Multiprocessor Systems”, Proceedings of 5th International Conference on Fault-Tolerant Computing Systems, Nürnberg, Informatik-Fachberichte 283, Berlin, Springer, S. 360-371, 1991.

[HIL92] Hildebrand, U.: “Konzeption, Bewertung und Realisierung einer dynamischen Netzwerkkomponente für speichergekoppelte Multiprozessorsysteme”, Dissertation, Arbeitsberichte des IMMD, Band 25, Nr. 5, Erlangen, Juli 1992.

[HMW85] Händler, W.; Maehle, E.; Wirl, K.: “DIRMU Multiprocessor Configurations”, Proceedings of International Conference on Parallel Processing, St. Charles, Illinois, S. 652-656, 1985.

[HÖN94] Hönig, J.: “Softwaremethoden zur Rückwärtsfehlerbehebung in Hochleistungs-parallelrechnern mit verteiltem Speicher”, Dissertation, IMMD, Erlangen, Juli 1994.

[HOF93] Hofmann, R.: “The Distributed Hardware Monitor ZM4 and Its Interface to MEMSY”, in: Bode, A.; Dal Cin, M. (Eds.): “Parallel Computer Architectures: Theorie, Hardware, Software, Applications”, Lecture Notes in Computer Science (LNCS) 732, Berlin, Springer, S. 66-79, 1993.

[JEW91] Jewett, D.: “Integrity S2: A Fault-Tolerant Unix Platform”, FTCS 21, IEEE, S. 512-519, 1991.

[KER90] Kernighan, B.W.; Ritchie, D.M.: “Programmieren in C”, 2,Ausgabe, deutsche Übersetzung von Schreiner, A.T. und Janich, E., Carl Hanser-Verlag, München und Prentice Hall-Verlag, London, 1990.

[LAM88] Lampson, B.W.; Paul, M.; Siegert, H.J.: “Distributed Systems - Architecture and Implementation”, Lecture Notes in Computer Science (LNCS) 105, Berlin, Springer, 4th printing, S. 246-265, 1988.

[LAT91] Lattice: “GAL Data Book”, Lattice Semiconductor Corporation, Hillsboro, Oregon, USA, 1991.

[LEH90] Lehmann-Emilius, L.: “Rekonfiguration und Rückwärtsfehlerbehebung für Multiprozessoren mit begrenzter Nachbarschaft - eine Untersuchung zur verteilten Recovery”, Dissertation, Arbeitsberichte des IMMD, Band 23, Nr. 2, Erlangen, Februar 1990.

[LOG93] Isdata: “LOGiC Benutzerhandbuch”, Isdata, Karlsruhe, 1993.

[MACH94] Advanced Micro Devices: “MACH 1 and 2 Family Data Book, High-Density EE CMOS Programmable Logic”, Advanced Micro Devices, 1994.

[MHP94] Muller, G.; Hue, M.; Peyrouze, N.: “Performance of Consistent Checkpointing in a Modular Operating System: Results of the FTM Experiment”, in: Echtle, K.; Hammer, D.; Powell, D. (Eds.): “Dependable Computing - EDCC-1”, Lecture Notes in Computer Science (LNCS) 852, Berlin, Springer, S. 491-508, 1994.

[MIC92] Michel, E.: “Fehlererkennung mit Überwachungsrechnern in Multiprozessor-systemen”, Dissertation, Arbeitsberichte des IMMD, Band 25, Nr. 6, Erlangen, September 1992.

[MIH91] Michel, E.; Hohl, W.: “Concurrent Error Detection Using Watchdog Processors in the Multiprocessor System MEMSY”, Proceedings of 5th International Conference on Fault-Tolerant Computing Systems, Nürnberg, Informatik Fachberichte 283, Berlin, Springer, S. 54-64, 1991.

[MOT1] Motorola: “MC88100 RISC Microprocessor, User’s Manual”, 2. Edition, Prentice Hall, Englewood Cliffs, New Jersey 07632.

[MOT2] Motorola: “MC88200 Cache/Memory Management Unit, User’s Manual”, 2. Edition, Prentice Hall, Englewood Cliffs, New Jersey 07632.

[MOT3] Motorola: “MC88204, Technical Summary, 64K-Byte Cache/Memory Management Unit (CMMU)”, Motorola MC88204/D, 1991.

[MPD94] Majzik, I.; Pataricza, A.; Dal Cin, M.; Hohl, W.; Hönig, J.; Sieh, V.: “Hierarchical Checking of Multiprocessors Using Watchdog Processors”, in: Echtle, K.; Hammer, D.; Powell, D. (Eds.): “Dependable Computing - EDCC-1”, Lecture Notes in Computer Science (LNCS) 852, Berlin, Springer, S. 386-403, 1994.

[MVM89] Motorola: “MVME188 VMEmodule, RISC Microcomputer, User’s Manual”, Motorola MVME188/D1, September 1989.

[OCC88] Inmos: “OCCAM 2, Reference Manual”, Inmos Limited, Prentice Hall, New York, 1988.

[OKO94] Okoth, I.: “Smarte Flash-Konzepte”, in: Design & Elektronik, 20.Ausgabe, S. 22-23, Oktober 1994.

[PAH90] Patterson, D.A.; Hennessy, J.L.: “Computer Architecture”, Morgan Kaufmann Publishers, San Mateo, California, S. 506-521, 1990.

[PEW72] Peterson, W.W.; Weldon, E.J.: “Error-Correcting Codes”, 2. Edition, MIT Press, Cambridge, Massachusetts, 1972.

[PMH93] Patericza, A.; Majzik, I.; Hohl, W.; Hönig, J.: “Watchdog Processors in Parallel Systems”, Microprocessing and Microprogramming 39, S. 69-74, 1993.

[ROB93] Robbert, G.: “Software-Simulation eines Stabilen Speichers”, Studienarbeit im Fach Informatik am IMMD 3 der Universität Erlangen-Nürnberg, März 1993.

[SEM93] Semmler, M.: “Die Steuerung für einen Stabilen Speicher”, Diplomarbeit im Fach Informatik am IMMD 3 der Universität Erlangen-Nürnberg, Dezember 1993.

[SFB93] “Arbeits- und Ergebnisbericht des SFB182”, S. 114-120, Erlangen, 1993.

[SIE82] Siewiorek, D.P.; Swarz, R.S.: “The Theory and Practice of Relieable System Design”, Digital Press, 1982.

[SIE93] Sieh, V.: “Transparente Zustandssicherung und -wiederherstellung im Memsos-Betriebssystem”, Diplomarbeit im Fach Informatik am IMMD 3 der Universität Erlangen-Nürnberg, März 1993.

[SPH94] Sieh, V.; Patericza, A.; Sallay, B.; Hohl, W.; Hönig, J.; Benyó, B.: “Fault Injection Based Validation of Fault-Tolerant Multiprocessors”, Proceedings of the 8th Symposium on Microcomputer and Microprocessor Applications, Budapest, S. 85-94, Oktober 1994.

[TIC90] Texas Instruments: “Interface Circuits Data Book”, Texas Instruments, 1990.

[TIT89] Texas Instruments: “The TTL Data Book, Vol.1 + Vol.2”, Texas Instruments, 1989.

[TDS88] Inmos: “Transputer Developement System”, Inmos Limited, Prentice Hall, New York, 1988.

[TDD89] Inmos: “The Transputer Developement and iq Systems Databook”, Inmos Limited, 1989.

[WAK78] Wakerly, J.F.: “Error Detecting Codes, Self-Checking Circuits and Applications”, North Holland, New York, 1978.

[WIS78] Williams, F.J.; Sloane, N.J.A.: “The Theory of Error-Correcting Codes”, S. 80 ff, North Holland, Amsterdam, 1978.

Lebenslauf

von

Andreas Grygier

Geburtstag, Geburtsort:	9. Mai 1960	in Bad Brückenau
Familienstand:	verheiratet seit dem 8. September 1989, zwei Kinder	
Schulausbildung:	1966 - 1970 :	4 Jahre Grundschule in Bad Brückenau
	1970 - 1979 :	9 Jahre Gymnasium in Bad Brückenau, Abschluß mit Abitur
Wehrdienst:	1979 - 1980 :	Grundwehrdienst als Jäger
Hochschulausbildung:	1980 - 1987 :	Informatik-Studium an der Universität Erlangen-Nürnberg mit Schwerpunkt Rechnerarchitektur, Abschluß mit Diplomprüfung
Berufstätigkeit:	1987 - 1995 :	wissenschaftlicher Mitarbeiter am Institut für Mathematische Maschinen und Datenverarbeitung III (IMMD 3) der Universität Erlangen-Nürnberg