(19) **Bundesrepublik Deutschland**
Deutsches Patent- und Markenamt

(10) **DE 698 32 943 T2** 2006.06.29

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 911 731 B1**
(21) Deutsches Aktenzeichen: **698 32 943.0**
(96) Europäisches Aktenzeichen: **98 308 323.9**
(96) Europäischer Anmeldetag: **12.10.1998**
(97) Erstveröffentlichung durch das EPA: **28.04.1999**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **28.12.2005**
(47) Veröffentlichungstag im Patentblatt: **29.06.2006**

(51) Int Cl.[8]: ***G06F 9/46*** (2006.01)

(30) Unionspriorität:
957298 **24.10.1997** **US**

(84) Benannte Vertragsstaaten:
DE, FR, GB

(73) Patentinhaber:
Compaq Computer Corp., Houston, Tex., US

(74) Vertreter:
Grünecker, Kinkeldey, Stockmair & Schwanhäusser, 80538 München

(72) Erfinder:
Vandoren, Stephen R., Northborough, Massachusetts 01532, US; Steely, Simon C., Hudson, New Hampshire 03051, US; Sharma, Madhumitra, Shrewsbury, Massachusetts 01545, US; Fenwick, David M., Acton, Massachusetts 01545, US

(54) Bezeichnung: **Sequenzsteuerungsmechanismus für ein switch-basiertes Mehrprozessorsystem**

Beschreibung

[0001] Diese Erfindung betrifft im Allgemeinen das Gebiet der Computerarchitektur und im Besonderen verteilte Mehrprozessorsysteme mit gemeinsamen Speichern.

[0002] Wie in der Technik bekannt, ermöglichen symmetrische Mehrprozesssysteme Hochleistungs-Anwendungsprozesse. Typische symmetrische Mehrprozess-Computersysteme enthalten eine Anzahl von Prozessoren, die über einen Bus miteinander gekoppelt sind. Eine Charakteristik eines Mehrprozesssystem ist, dass Speicherraum von allen Prozessoren gemeinsam genutzt wird. In dem Speicher ist ein Betriebssystem oder sind mehrere Betriebssysteme gespeichert und steuert bzw. steuern die Verteilung der Prozesse oder der Threads zwischen den verschiedenen Prozessoren.

[0003] Die Ausführungsgeschwindigkeit einer gegebenen Anwendung kann dadurch, dass verschiedenen Prozessoren ermöglicht wird, verschiedene Prozesse oder Threads gleichzeitig auszuführen, wesentlich erhöht werden. Theoretisch könnte die Leistung eines Systems dadurch verbessert werden, dass lediglich die Anzahl der Prozessoren in dem Mehrprozesssystem erhöht wird. Das kontinuierliche Hinzufügen von Prozessoren über einen bestimmten Sättigungspunkt hinaus führt in der Realität jedoch lediglich zum Vermehren von Kommunikationsengpässen und dadurch zur Begrenzung der Gesamtleistung des Systems.

[0004] Bezug nehmend auf die Fig. 1A, wird beispielhaft ein typisches Mehrprozesssystem nach dem Stand der Technik gezeigt, das acht miteinander gekoppelte Prozessoren enthält. In Betrieb kommuniziert jeder der Prozessoren **3a**–**3h** über den gemeinsam genutzten Verbindungsbus **5** mit den anderen Prozessoren und mit einem gemeinsamen Speicher **4**. Die symmetrische Mehrprozessoranordnung der Fig. 1A ist für Mehrprozessoren, die bis dato hergestellt wurden, adäquat. Mit der Einführung schnellerer Mikroprozessoren ist jedoch eine gemeinsam genutzte Verbindung nicht mehr in der Lage, von dem vollen Leistungspotenzial der gekoppelten Mikroprozessoren Gebrauch zu machen. Weil das einzige Verbindungsglied zwischen den Prozessoren und dem Speicher der gemeinsam genutzte Bus ist, wird der Bus schnell mit Anforderungen der Prozessoren gesättigt und vermehrt dadurch Verzögerungen, da jeder Prozessor versucht, Zugriff auf den Systembus zu bekommen. Deshalb ist, obwohl die Prozessoren mit höheren Geschwindigkeiten arbeiten können, die verfügbare Bandbreite des Systembusses der in Bezug auf die Leistung beschränkende Faktor. Die Kommunikationsbandbreite ist ein Schlüsselfaktor für die Leistung eines SMP-Systems. Da die Bandbreite zwischen Paaren oder Untergruppen von Knoten in dem SMP-System ungleichmäßig sein kann, nutzt die Industrie zum Bestimmen der Kommunikationsbandbreite eines SMP-Systems eine „Bisektionsbandbreitenmessung". Die Bisektionsbandbreite wird in der folgenden Art und Weise bestimmt: Alle möglichen Arten der Teilung des Systems in zwei Teile gleicher Rechenleistung (gleiche Anzahl von Prozessoren) werden ermittelt. Für jede Teilung wird die Bandbreite, die zwischen den beiden Teilungen aufrechterhalten werden kann, festgestellt. Das Minimum aller Bandbreiten, die aufrechterhalten werden können, ist die Bisektionsbandbreite der Verbindungen untereinander. Die Mindestbandbreite zwischen den beiden Teilungen zeigt die Bandbreite an, die durch das Multiprozessorsystem in dem Fall von Worst-case-Kommunikationsmustern aufrechterhalten werden kann. Infolgedessen ist eine große Bisektionsbandbreite erwünscht.

[0005] Um die Sättigungsprobleme des Busses zu überwinden, wurden nach dem Stand der Technik verschiedene Verbindungsarchitekturen oder „Topologien° genutzt. Diese Topologien enthalten Maschen, Tori, Hyperkuben und erweiterte Hyperkuben.

[0006] Als ein Beispiel wird in der Fig. 1B eine Maschennetzwerkverbindung als System **7** gezeigt. Die Hauptvorteile des Maschennetzwerkes sind seine Einfachheit und das leichte Verdrahten. Jeder Knoten ist mit einer geringen Anzahl von Nachbarknoten verbunden. Die Maschenverbindung weist jedoch drei wesentliche Nachteile auf. Erstens müssen Nachrichten im Durchschnitt eine große Anzahl von Knoten durchqueren, um zu ihrer Zieladresse zu kommen, was zu einer hohen Kommunikationslatenz führt. Zweitens vergrößert sich die Bisektionsbandbreite für eine Maschentopologie nicht so gut, wie bei anderen Topologien. Schließlich sind, da jede der Nachrichten verschiedene Pfade innerhalb des Maschennetzwerkes durchqueren kann, innerhalb des SMP-Systems keine natürlichen Ordnungspunkte vorhanden und deshalb sind die Cache-Kohärenz-Protokolle, deren Anwendung erforderlich ist, oft sehr komplex.

[0007] Die Torus- und Hyperkubus-Topologie und die Topologie des erweiterten Hyperkubus sind jeweils Topologien, in denen die Knoten in verschiedenen komplexen Anordnungen untereinander verbunden sind, beispielsweise in einer Torus-Anordnung oder in einer Kubusanordnung. Die Torus- und Hyperkubus-Verbindungen oder die Verbindungen des erweiterten Hyperkubus sind komplexer als die Verbindungen durch die Maschennetzwerke, bieten jedoch eine bessere Latenz und Bandbreite als die Maschenverbindungen. Die Torus-

und Hyperkubus-Topologie und die Topologie des erweiterten Hyperkubus stellen jedoch, wie auch die Maschenverbindungen, keine natürlichen Ordnungspunkte bereit und infolgedessen muss für jedes dieser Systeme ein komplexes Cache-Kohärenz-Protokoll implementiert werden.

[0008] In Mehrprozesssystemen mit gemeinsamem Speicher verwenden die Prozessoren üblicherweise eigene Caches, um Daten zu speichern, von denen als wahrscheinlich festgestellt ist, dass zukünftig auf diese zuzugreifen ist. Da die Prozessoren Daten aus ihrem eigenen Cache lesen können und die Daten in dem eigenen Cache aktualisieren können, ohne diese zurück in den Speicher zu schreiben, wird ein Mechanismus gebraucht, der sicherstellt, dass der eigene Cache jedes Prozessors konsistent oder kohärent bleibt. Der Mechanismus, der verwendet wird, um die Kohärenz von Daten in dem SMP-System sicherzustellen, wird als das Cache-Kohärenz-Protokoll bezeichnet.

[0009] Das Cache-Kohärenz-Protokoll ist neben der Topologie, der Bandbreite und der Latenz der physikalischen Verbindungen ein Schlüsselfaktor der Systemleistung. Cache-Kohärenz-Protokolle können auf verschiedene Arten Latenzen, Flaschenhälse, Leistungsschwäche oder Komplexität einführen.

[0010] Die Latenz der Last und der Speichervorgänge wird oft direkt von dem Aufbau des Protokolls beeinflusst. Beispielsweise wird in einigen Protokollen der Speichervorgang nicht als ausgeführt betrachtet, bis alle ungültigen Nachrichten bis zu ihren Zielprozessoren gekommen sind und die Bestätigungsnachrichten den ganzen Weg zurück zu dem Ursprungsprozessor verfolgt haben. Die Latenz des Gespeicherten ist hier sehr viel höher als bei dem Protokoll, in dem der Ursprungsprozessor nicht auf die Ungültigen warten muss, um an seine Zieladresse zu kommen. Des Weiteren verbrauchen die Bestätigungen einen wesentlichen Anteil der Systembandbreite.

[0011] Flaschenhälse werden oft auf Grund der starken Belegung der Controller verursacht. „Belegung" ist ein Ausdruck der Technik, der den Zeitraum anzeigt, für den ein Controller nicht verfügbar ist, nachdem er eine Anforderung empfangen hat. In einigen Protokollen ist ein Controller, wenn er eine Anforderung, die einem Speicherort entspricht, empfangen hat, für weitere Anforderungen für denselben Speicherort nicht verfügbar, bis bestimmte Bestätigungen, die der ersten Anforderung entsprechen, in dem Verzeichnis ankommen. Wenn der Controller miteinander in Konflikt stehende Anforderungen mit einer höheren als der durchschnittlichen Rate empfängt, kommt es zu einem Flaschenhals.

[0012] Der Aufbau des Cache-Kohärenz-Protokolls beeinflusst außerdem die Komplexität. Einige Protokolle führen beispielsweise zu Verklemmungen und Fairnessproblemen, denen dann mit zusätzlichen Mechanismen zu begegnen ist. Dies resultiert in zusätzlichem Hardwareaufwand.

[0013] Es ist erwünscht, ein symmetrisches Mehrprozessorsystem bereitzustellen, das die Latenz der Operationen minimiert, eine große Kommunikationsbandbreite und geringere Controller-Belegung bereitstellt und das auf eine große Anzahl von Prozessoren vergrößert werden kann.

[0014] GB-A-2 1881 77 legt ein Computersystem für das gemeinsame Nutzen einer Datenquelle offen. Das System arbeitet durch das Senden einer Anforderungssperre für die gemeinsame Datenquelle zu dem Sperrmanager auf dem Host, der der Master der Datenquellengruppe ist, die zu sperren ist, und für diese wird Exklusivsteuerung ausgeführt.

[0015] US 5.060 144 legt eine Sperrsteuerungs-Statusanzeige für ein Multi-Host-Prozessorsystem offen, das für jeden Host-Prozessor einen Datensatz-Sperrprozessor und einen Pufferspeicher verwendet.

[0016] EP-A-0 121 700 legt eine Mehrprozessor-Speicherungsserialisierungsvorrichtung offen, die mehreren Prozessoren ermöglicht, Befehle auf Gespeichertes zuzugreifen, gleichzeitig auszuführen, ohne die Leistung materiell zu beeinflussen. Dies wird dadurch erreicht, dass die Menge des Gespeicherten auf ein Minimum, beispielsweise eine Seite, gesperrt wird.

[0017] Die vorliegende Erfindung wird vorteilhaft in einem symmetrischen Mehrprozessorsystem angewendet, in dem mehrere Mehrprozessorknoten wenigstens einen Prozessor und einen Teil eines gemeinsamen Speichers, die über einen Switch miteinander gekoppelt sind, enthalten. In jedem der Multiprozessorknoten wird ein Transaktions-Tracking-Verzeichnis (TTT) unterhalten. Das TTT kann in einem globalen Port des Knotens, der den Knoten mit dem Switch verbindet, vorhanden sein oder alternativ auf jedem der wenigstens einen der Prozessoren des Multiprozessorknotens vorhanden sein.

[0018] Das TTT wird verwendet, um eine Reihenfolge der Anforderungen, die von diesem Mehrprozessorknoten ausgegeben und empfangen werden, zu bestimmen und durchzusetzen. Gemäß einem Aspekt der Erfindung wird das TTT verwendet, um die Reihenfolge der Anforderungen, die zu dem Mehrprozessorknoten zurückgesendet werden, in der folgenden Art und Weise zu bestimmen. Jede der Anforderungen wird in eine Anzahl von Transaktionen unterteilt, wobei jede der Transaktionen auf einem verschiedenen virtuellen Kanal ausgeführt wird. Wenigstens einer der Kanäle ist geordnet, jedoch können Return-Daten auf den anderen Kanälen außerhalb der Reihenfolge empfangen werden. Um die Kohärenz aufrechtzuerhalten, ist erwünscht, dass die zu einer gemeinsamen Adresse ausgegebenen Transaktionen in Reihenfolge behandelt werden. Gemäß einem Aspekt der Erfindung wird auf dem geordneten Kanal ein Marker-Paket zu dem TTT gesendet, um anzuzeigen, dass die mit einer Adresse verbundenen Daten noch immer weiterverarbeitet werden. Mit einer derartigen Anordnung kann das TTT sicherstellen, dass die weiteren Anforderungen auf dem geordneten Kanal, die dem Marker-Paket folgend empfangen wurden, entweder ignoriert oder verzögert werden, bis die Daten empfangen werden.

[0019] Gemäß einem Aspekt der vorliegenden Erfindung umfasst ein Computersystem eine Vielzahl von gekoppelten Multiprozessorknoten, jeder der Knoten umfasst wenigstens einen Prozessor und einen Teil eines gemeinsamen Speichers, das Computersystem ist gekennzeichnet durch:
einen Tracking-Mechanismus, verbunden mit der Vielzahl von Prozessoren in jedem der Vielzahl von Multiprozessorknoten, zum Bestimmen einer Position einer Anforderung zu einer Adresse eines dezentralen Teils eines gemeinsamen Speichers, ausgegeben von wenigstens einem der Prozessoren in dem einen der Vielzahl von Multiprozessorknoten,
relativ zu einer Vielzahl von anderen Anforderungen, die durch den wenigstens einen der Prozessoren in der Vielzahl von gekoppelten Multiprozessorknoten zu der Adresse ausgegeben wurde.

[0020] Gemäß einem weiteren Aspekt der vorliegenden Erfindung wird ein Verfahren zum Aufrechterhalten der Reihenfolge zwischen einer Vielzahl von Anforderungen, die zu einer gemeinsamen Adresse in einem Mehrprozessorcomputersystem ausgegeben werden, bereitgestellt.

[0021] Das Multiprozessorcomputersystem enthält eine Vielzahl von Multiprozessorknoten, die über einen Switch gekoppelt sind, wobei jeder der Multiprozessorknoten wenigstens einen Prozessor und einen Teil eines gemeinsamen Speichers umfasst. Das Verfahren enthält den Schritt des Führens eines Adressverzeichnisses von Anforderungen, die von jedem der Mehrfachprozessorknoten an den Switch weitergeleitet werden, um eine relative Reihenfolge von Anforderungen zu den jeweiligen Adressen in einem Teil eines gemeinsamen Speichers eines Mehrprozessorknotens zu identifizieren, wobei eine Adresse in dem Verzeichnis geführt wird, bis die mit der Adresse verbundenen Anforderung erfüllt ist.

[0022] Die oben angeführten und weitere Merkmale der Erfindung werden durch Bezugnahme auf die folgende Beschreibung, die in Verbindung mit den begleitenden Zeichnungen steht, offensichtlicher, wobei in den Zeichnungen

[0023] Fig. 1A und Fig. 1B Blockdiagramme von zwei Mehrprozessorcomputersystemen nach dem Stand der Technik sind,

[0024] Fig. 2 ein Blockdiagramm eines Ausführungsbeispiels eines Mehrprozessorknotens eines Ausführungsbeispiels der Erfindung ist, das einen Switch umfasst,

[0025] Fig. 3 ein Blockdiagramm ist, das den Datenpfad des Switchs der **Fig.** 1 zeigt, der eine Anzahl von Simultan-Eingabepuffern umfasst,

[0026] Fig. 4A ein Blockdiagramm eines Ausführungsbeispiels eines der Simultan-Eingabepuffer der Fig. 3 ist,

[0027] Fig. 4B ein Blockdiagramm einer Implementierung der Logik zum Steuern des Simultan-Eingabepuffers der **Fig.** 4 ist,

[0028] Fig. 5 ein Blockdiagramm eines zweiten Ausführungsbeispiels des einen der Simultan-Eingabepuffer der Fig. 3 ist,

[0029] Fig. 6 ein Blockdiagramm des Mehrprozessorknotens der Fig. 2 ist, der zum Anschließen in einem größeren Netzwerk von gleichartigen Knoten erweitert ist,

[0030] Fig. 7A ein Ausführungsbeispiel eines SMP-Systems ist, das mehrere den Mehrprozessorknoten der Fig. 6 gleichartige Mehrfachknoten verwendend implementiert ist,

[0031] Fig. 7B ein weiteres Ausführungsbeispiel eines SMP-Systems ist, das mehrere den Mehrprozessorknoten der Fig. 6 gleichartige Mehrfachknoten verwendend implementiert ist,

[0032] Fig. 8 ein Blockdiagramm eines globalen Ports der Fig. 6 ist,

[0033] Fig. 9 einen Eintrag in einem Verzeichnis der Mehrprozessorknoten der Fig. 6 darstellt,

[0034] Fig. 10 ein Transaktions-Tracking-Verzeichnis (TTT) zur Verwendung in dem globalen Port der Fig. 8 darstellt,

[0035] Fig. 11 ein Blockdiagramm eines hierarchischen Schalters zum Koppeln der Mehrprozessorknoten in der Fig. 7A ist,

[0036] Fig. 12A ein Blockdiagramm eines Ausführungsbeispiels einer Verbindungslogik für den hierarchischen Switch ist, der Verklemmung eliminiert,

[0037] Fig. 12B ein Flussdiagramm des Betriebs der Verbindungslogik der Fig. 12A ist,

[0038] Fig. 13 ein Flussdiagramm des in der Verbindungslogik der Fig. 11 verwendeten Verfahrens ist, um Stromsteuerung zum Unterbinden, dass Daten von einem der Mehrprozessorknoten übertragen werden, durchzusetzen,

[0039] Fig. 14 ein Zeitablaufdiagramm zum Darstellen der Übertragung von Adressen und Datenpaketen auf den Bussen von und zu dem hierarchischen Switch ist,

[0040] Fig. 15 ein Blockdiagramm eines Ausführungsbeispiels der Pufferlogik zum Aufrechterhalten der Reihenfolge in dem hierarchischen Switch ist,

[0041] Fig. 16 ein Blockdiagramm eines weiteren Ausführungsbeispiels der Pufferlogik zum Aufrechterhalten der Reihenfolge für den hierarchischen Switch ist,

[0042] Fig. 17 ein Flussdiagramm zum Darstellen einer Methode des Betriebs der Pufferlogik der Fig. 16 ist,

[0043] Fig. 18 ein Blockdiagramm eines weiteren Ausführungsbeispiels der Pufferlogik zum Aufrechterhalten der Reihenfolge in dem hierarchischen Switch ist,

[0044] Fig. 19 ein Verzeichnis ist, das die Translation der Prozessorbefehle in Netzwerkbefehle zur Verwendung in dem SMP der Fig. 7A oder Fig. 7B darstellt,

[0045] Fig. 20A–Fig. 20H eine Anzahl von Kommunikationsströmen zum Übertragen von Paketen zwischen Knoten in dem SMP der Fig. 7A oder Fig. 7B darstellt,

[0046] Fig. 21 ein Blockdiagramm ist, das das Layout eines Speichermoduls zur Verwendung in dem Mehrprozessorcomputersystem der Fig. 2 oder Fig. 6 darstellt,

[0047] Fig. 22 ein Zeitablaufdiagramm ist, das die Steuerlogik, die durch das Speichermodul der Fig. 21 für verzögerte Schreiboperationen verwendet wird, darstellt,

[0048] Fig. 23 ein Flussdiagramm ist, das die Verwendung von diskreten Transaktionen, die zum Aufrechterhalten der Cache-Kohärenz auf den Kanälen abgebildet werden, in einem Ausführungsbeispiel der Erfindung darstellt,

[0049] Fig. 24 ein Blockdiagramm ist, das eine Implementierung einer gemeinsamen Warteschlangenstruktur zum Verwalten der virtuellen Kanäle in dem SMP der Fig. 7A oder Fig. 7B darstellt,

[0050] Fig. 25 ein Blockdiagramm ist, das eine Implementierung der einzelnen Kanalpufferung in den Knoten und den hierarchischen Switchs des SMPs der Fig. 7A oder Fig. 7B darstellt,

[0051] Fig. 26 ein Blockdiagramm zum Darstellen der Probleme ist, die entstehen können, wenn ein Teil der Reihenfolge zwischen den virtuellen Kanälen nicht eingehalten wird,

[0052] Fig. 27A–Fig. 27C Blockdiagramme sind, die die Strom- und die Reihenfolgebeschränkungen auf dem Q1-Kanal zum Bereitstellen kohärenter Kommunikation in dem SMP der Fig. 7A oder Fig. 7B darstellen,

[0053] Fig. 28A und Fig. 28B Blockdiagramme sind, die die Mehrdeutigkeitsprobleme, die entstehen können, weil der grobe Vektor Bits der Verzeichniseinträge des SMPs der Fig. 7A und Fig. 7B darstellt,

[0054] Fig. 29 ein Blockdiagramm ist, das das Verfahren darstellt, das verwendet wird, um zu verhindern, dass im Ergebnis des in der **Fig.** 28 beschriebenen Problems Datenmehrdeutigkeit entsteht,

[0055] Fig. 30 ein Blockdiagramm zum Darstellen eines Kohärenzproblems ist, das durch Pakete auf verschiedenen Kanälen, die außerhalb der Reihenfolge empfangen werden, entsteht,

[0056] Fig. 31 ein Blockdiagramm ist, das die Verwendung von Füll-Markern zum Verhindern des in der Fig. 29 beschriebenen Kohärenzproblems darstellt,

[0057] Fig. 32 ein Eintrag in dem TTT ist, der den Status eines Befehls während des in Bezug auf die Fig. 31 beschriebenen Stroms wiedergibt,

[0058] Fig. 33A und Fig. 33B Blockdiagramme sind, die den Arbeitsschritt des Änderns in dirty Befehle in dem SMP-System darstellen,

[0059] Fig. 34 ein Blockdiagramm ist, das die Verwendung von Schattenbefehlen, um das in Bezug auf die **Fig.** 33 beschriebene Problem zu beseitigen, darstellt,

[0060] Fig. 35 ein Eintrag in das TTT ist, der den Status eines Befehls während des in Bezug auf die Fig. 34 beschriebenen Stroms darstellt, und

[0061] Fig. 36 ein Flussdiagramm ist, das die zulässige sequenzielle Ordnung von Befehlen in dem Beispiel, das in der Fig. 35 beschrieben wird, darstellt.

[0062] Gemäß einem Ausführungsbeispiel der Erfindung enthält ein hierarchisches symmetrisches Mehrprozessorsystem (SMP-System) eine Anzahl von SMP-Knoten, die über einen Hochleistungs-Switch miteinander gekoppelt sind. Folglich wirkt jeder der SMP-Knoten als ein Baustein in dem SMP-System. Im Folgenden werden zuerst die Komponenten und der Betrieb eines SMP-Knoten-Bausteins beschrieben, gefolgt von einer Beschreibung des Betriebs des SMP-Systems und einer anschließenden Beschreibung eines Cache-Kohärenz-Protokolls, das verwendet wird, um in dem großen SMP-System Speicherkohärenz aufrechtzuerhalten.

SMP-Knoten-Baustein

[0063] Im Folgenden auf die Fig. 2 Bezug nehmend, enthält ein Mehrprozessorknoten **10** vier Prozessormodule **12a**, **12b**, **12c** und **12d**. Jedes Prozessormodul umfasst eine zentrale Recheneinheit (CPU). In einem bevorzugten Ausführungsbeispiel werden von der Digital Equipement Corporation® gefertigte Alpha® 21264 Prozessorchips verwendet, obwohl alternativ andere Typen von Prozessorchips, die in der Lage sind, das unten beschriebene Kohärenzprotokoll zu unterstützen, verwendet werden können.

[0064] Der Mehrprozessorknoten **10** enthält einen Speicher **12**, der eine Anzahl von Speichermodulen **13a**–**13d** enthalten kann. Der Speicher kann eine Speicherkapazität von 32 Gigabytes bereitstellen, wobei jedes der vier Speichermodule 8 Gigabyte speichert. Jedes der Speichermodule ist in eine Anzahl von Speicherblöcken aufgeteilt, wobei jeder Block beispielsweise 64 Bytes Daten enthalten kann. Die Daten werden generell in Blöcken aus dem Speicher abgerufen.

[0065] Zusätzlich enthält der Mehrprozessorknoten **10** ein I/O-Prozessormodul (IOP-Modul) zum Steuern der Übertragung von Daten zwischen externen Vorrichtungen (nicht gezeigt) und dem Mehrprozessorknoten **10** über einen gekoppelten I/O-Bus **14a**. In einem Ausführungsbeispiel der Erfindung kann der I/O-Bus gemäß dem PCI-Protokoll (Peripheral Computer Interconnect protocol) betrieben werden. Das IOP-Modul **14** enthält einen IOP-Cache **14c** und einen IOP-Tag-Speicher **14b**. Der IOP-Cache **14c** stellt die Zwischenspeicherung für Daten aus dem Speicher **13** bereit, die auf dem PCI-Bus **14a** zu den externen Vorrichtungen übertragen

werden. Der IOP-Tag-Speicher **14b** ist ein 64-Einträge-Tag-Speicher zum Speichern von Kohärenzinformationen für Daten, die zwischen den externen Vorrichtungen, den Prozessoren und dem Speicher bewegt werden.

[0066] Die Kohärenz der in dem Speicher **13** des Mehrprozessorknotens gespeicherten Daten wird mittels eines Duplikat-Tag-Speichers (DTAG) **20** aufrechterhalten. Der DTAG **20** wird von allen Prozessoren **12a**–**12d** gemeinsam genutzt und ist in vier abgegrenzte Teile des Speichers (im Folgenden als Speicherbänke bezeichnet) unterteilt, wobei jede Speicherbank Statusinformationen speichert, die den Daten, die durch einen zugehörigen der Prozessoren verwendet werden, entsprechen.

[0067] Der DTAG, der Speicher und das IOP werden mit einem Logikbus gekoppelt, der als der Verteilerbus **17** bezeichnet wird. Die durch den Prozessor ausgegebenen Speicherblockanforderungen werden über den lokalen Switch **15** zu dem Verteilerbus **17** geleitet. Der DTAG **20** und das IOP **14** schlagen den Status des Blocks in den Caches des Prozessors und des IOPs nach und aktualisieren ihren Status für den Speicherblock atomar. Der Verteilerbus **17** wirkt als ein Serialisierungspunkt für alle Speicherreferenzen. Die Reihenfolge in der die Speicheranforderungen auf dem Verteilerbus erscheinen, ist die Reihenfolge, in der die Prozessoren die Ergebnisse der Anforderung entgegennehmen.

[0068] Die Prozessormodule **12a**–**12d**, die Speichermodule **13a**–**13d** und die IOP-Module **14** sind über einen lokalen Port-Switch **15** miteinander gekoppelt. Jedes der Schnittstellenmodule **12a**–**12d**, **13a**–**13d** und **14** ist mittels einer gleichen Anzahl von bidirektional taktweitergeleiteten Datenlinks **16a**–**16i** verbunden. In einem Ausführungsbeispiel leitet jeder der Datenlinks 64 Bits Daten und 8 Bits Fehlerkorrekturcode (ECC), einen davon auf jeder Flanke des Systemtakts, der mit einer Rate von 150 MHz arbeitet, weitet. Infolgedessen ist die Datenbandbreite jedes der Datenlinks **16a**–**16i** 2,4 Gigabytes/sek.

[0069] Der lokale Switch **15** enthält einen Quad-Switch-Adressensteuerungschip (QSA-Chip) **18** und einen Quad-Switch-Daten-Slice-Chip (QSD-Chip) **19**. Der QSA-Chip **18** enthält einen Verteiler (QSA-Verteiler) **11** zum Steuern der Adressenpfade zwischen den Prozessormodulen, dem IOP und dem Speicher. Zusätzlich stellt der QSA-Chip **18** Steuerung für den QSD-Chip **19** bereit, um den Datenstrom durch den lokalen Switch **15** zu steuern, wie unten beschrieben wird.

[0070] Der QSD-Chip **19** stellt eine Switch-Verbindung für alle Datenpfade zwischen den Prozessormodulen, den Speichermodulen und dem IOP bereit. Obwohl in der Fig. 2 nicht gezeigt, würden, wie unten beschrieben wird, wenn der Mehrprozessorknoten **10** mit den anderen Mehrprozessorknoten über einen globalen Port gekoppelt wäre, der QSD und der QSA zusätzlich für den globalen Port eine Switch-Verbindung bereitstellen. Jeder der Prozessoren kann über den globalen Port Daten von einer der verfügbaren Ressourcen, wie den Speichereinrichtungen **13a**–**13d**, weiteren Prozessoren **12a**–**12d**, dem IOP **14** oder alternativen Ressourcen in anderen Mehrprozessorknoten, anfordern. Infolgedessen sollte der lokale Switch **15** fähig sein, gleichzeitig Eingänge von einer Vielzahl von Ressourcen aufzunehmen, während er die hohe Busbandbreite von 2,4 Gigabytes aufrechterhält.

[0071] Der lokale Switch kann mehrere gleichzeitige Transaktionen behandeln. Da jede Transaktion üblicherweise mehrere Ressourcen verwendet (wie zum Beispiel Speicherbänke, Datenpfade, Warteschlangen), können die Steuerfunktionen des lokalen Switchs sehr komplex sein. Beispielsweise kann eine Transaktion im Stadium 0 der Transaktion eine verfügbare Speicherbank, die Verfügbarkeit des Datenpfades von der Speicherbank zu dem Prozessor in Stadium 1 und die Verfügbarkeit des Datenpfades zu dem Prozessorport in Stadium 2 erforderlich machen. Der lokale Switch-Verteiler (QSA-Verteiler **11** in dem QSA **18**) vermittelt derartig zwischen den Anforderungen, dass sobald eine Transaktion initiiert ist, die in jedem Stadium der Transaktion erforderlichen Ressourcen wie erforderlich verfügbar sind.

[0072] Signifikanter, der Verteiler garantiert durch Sicherstellen, dass bestimmten Anforderungen nicht versagt wird, über einen langen Zeitraum Zugriffskontrolle zu gewinnen (potenziell idenfinit), während andere Fortschritte machen, dass alle Anforderungen und Prozessoren fairen Zugriff auf die Ressourcen erhalten. Als Beispiel wird eine Transaktion T, die drei Ressourcen A, B und C erfordert betrachtet.

[0073] Die Transaktion T könnte die Zugriffsverteilung nicht gewinnen, bis garantiert wäre, dass alle drei Ressourcen in den adäquaten Stadien der Transaktion verfügbar sind. Wenn der Verteiler seine Entscheidung nur auf der Verfügbarkeit von Ressourcen basiert, dann ist es möglich, dass T für einen langen Zeitraum nicht erfolgreich ist, während andere Transaktionen, die nur eine von A, B oder C (zusammen mit anderen Ressourcen D, E usw.) erfordern; kontinuierlich die Zugriffsverteilung gewinnen.

[0074] Das Garantieren fairer Zugriffsverteilung in einem Switch mit einer großen Anzahl von gleichzeitigen Anforderungen, von denen jede, um fertig gestellt zu werden, mehrere Ressourcen verwendet, erfordert einen komplexen Rechenaufwand und führt wahrscheinlich zur Vermehrung von Verzögerungen in dem Hochgeschwindigkeits-Datenpfad. In der hierin offen gelegten Vorrichtung nimmt der QSA-Verteiler **11** vor der Ablaufplanung einer bestimmten Transaktion die Verteilung nur einer Ressource (der Speicherbank) vor. Eine zweite Ressource, die eine Warteschlange ist, die zu den Prozessoren führt, muss zum Zeitpunkt der Verteilung durch den QSA-Verteiler **11** für die erste Ressource nicht auf Verfügbarkeit geprüft werden. Dies deshalb, weil die Architektur des QSDs garantiert, dass die Datenpfade und die Warteschlangenschlitze, die zu der Warteschlange führen, immer verfügbar sind. In dem QSA-Verteiler **11** kann die faire Ressourcenverteilung ohne großen Aufwand bereitgestellt werden.

[0075] Gemäß einem Ausführungsbeispiel der Erfindung ist der QSD fähig, simultan Eingaben aus allen Quellen (Prozessoren, Speicher, IOP und globaler Port) zu empfangen, ohne Vorausverteilung der Puffer, die zu den entsprechenden Zieladressen führen, erforderlich zu machen. Alle Datenquellen können dann unabhängig Daten zu dem Switch weiterleiten, ohne dass über Zugriff auf den Datenpfad oder die Warteschlangenschlitze in dem Schalter entschieden werden müsste, weil der QSD eine Anzahl von Simultan-Eingabepuffern enthält, die in der Lage sind, Daten aus allen Quellen im Wesentlichen simultan zu empfangen. Im Folgenden werden zwei Ausführungsbeispiele von Simultan-Eingabepuffern beschrieben.

Simultan-Eingabepuffer-Switch

[0076] Wie oben beschrieben, dienen die Prozessoren **12a**–**12d**, das IOP **14** und die Speichereinrichtungen **13a**–**13d** jeweils als Ressourcen zum Behandeln von Anforderungen aus den Prozessoren und dem IOP in dem Mehrprozessorknoten. Die Daten werden zwischen jedem der Ressourcenelemente und den anfordernden Elementen in Form von Paketen übertragen. Jedes Paket enthält 512 Bits Daten und 64 Bits ECC. Wie oben beschrieben, trägt jeder der Datenlinks 64 Bits Daten und 8 Bits ECC auf jeder Flanke eines 150-MHz-Taktes. Infolgedessen sind QSD-extern acht Datenübertragungszyklen pro Paket vorhanden. QSD-intern werden die Daten jedoch nur auf einer Flanke des Taktes gesammelt. Infolgedessen werden von den Datenlinks für jeden Taktzyklus der QSD-internen Logik potenziell 128 Bits Daten empfangen. Da jedes Paket 512 Bits Daten und 64 Bits ECC umfasst, sind QSD-intern vier Datenübertragungszyklen für jedes Paket vorhanden, wobei mit jedem QSD-Taktzyklus 128 Bits Daten und 16 Bits ECC aus einem Prozessor, IOP oder einer Speichereinrichtung zu dem QSD übertragen werden.

[0077] Im Folgenden Bezug nehmend auf die Fig. 3, wird der QSD **19** detaillierter gezeigt und enthält fünf Simultan-Eingabepuffer (SIBs) **25a**–**25e**. Jeder SIB ist einem der anfordernden Elemente, beispielsweise den Prozessoren **12a**–**12d** oder dem IOP zugeordnet. Jeder SIB steuert den Datenpfad zum Übertragen von Paketen zwischen seinem zugehörigen anfordernden Element und den anderen Ressourcenelementen in dem Knoten, d. h. den Prozessoren **12a**–**12d**, den Speichern **13a**–**13d**, dem IOP **14** und zweckmäßigerweise dem globalen Port. Der globale Port wirkt als Verbindung zwischen allen weiteren Mehrprozessorknoten untereinander und wird unten im Einzelnen beschrieben. Die SIBs ermöglichen simultanes Empfangen von Paketen durch den Anfordernden von jeder Ressource, die mit dem Switch gekoppelt ist, ohne zwischen den Anfordernden die Verteilung des Switchs erforderlich zu machen.

[0078] Wie zuvor beschrieben, ist der QSA-Verteiler **11** gekoppelt, um die Steuerung für den Switch **19** bereitzustellen. In dem QSA-Verteiler **11** ist ein Hauptverteiler **27** enthalten. Der Hauptverteiler **27** verwaltet die Datenbewegung zwischen den Ressourcen (den Prozessoren **12a**–**12d**, den Speichern **13a**–**13d**, dem IOP **14**) und dem Switch **19**. Jeder der Prozessoren **12a**–**12d** und das IOP **14** gibt auf den Leitungen **28a**–**28e** Anforderungen auf Zugriff aus, die zu dem Hauptverteiler **27** weitergeleitet werden. Der Hauptverteiler wiederum leitet die Anforderungen zu den zugehörigen Ressourcen weiter, wenn jede Ressource in der Lage ist, eine Anforderung zu empfangen. Sobald eine Ressource die Anforderung empfangen hat, ist für den Switch **19** keine Zugriffsverteilung erforderlich, weil jeder der SIBs in der Lage ist, die Eingaben aller Eingänge im Wesentlichen simultan, d. h. innerhalb desselben Datenzyklus, zu empfangen.

[0079] In dem QSA-Verteiler **11** ist außerdem eine Anzahl von einzelnen Verteilern **23a**–**23d** enthalten. Jeder der Verteiler **23a**–**23d** wird verwendet, um jeweils einen Datenpfad zwischen einem zugehörigen der Prozessoren **12a**–**12d** und dessen zugehörigem SIB **25b**–**25e** zu verwalten.

[0080] Ein gleichartiger Verteiler (nicht gezeigt) ist in dem IOP **14** zu Verwalten des Datenpfades zwischen IOP **14** und SIB **25a** enthalten. Da jeder Prozessor in der Lage ist, Daten aus seinem zugehörigen SIB zu empfangen, leitet der zugehörige Verteiler die Daten auf dem gekoppelten Datenpfad weiter.

[0081] Dementsprechend kann durch das Verwenden von simultanen Eingabepuffern innerhalb des Switchs **19** der zwischen einem Anfordernden und einer Ressource verteilte Datenpfad in zwei unterschiedliche Abschnitte unterteilt werden, einen ersten Zugriffsverteilungsabschnitt, in dem der Hauptverteiler eine Ressource in Beantwortung einer Anforderung von einem Prozessor unabhängig von der Verfügbarkeit des anfordernden Prozessors Daten von der gekoppelten Ressource zu empfangen verteilt, und einen zweiten Zugriffsverteilungsabschnitt, in dem der dem Prozessor zugehörige Verteiler den Prozessor für den Zugriff des Prozessors auf von dem Switch weitergeleitete Daten verteilt. Mit einer derartigen Anordnung kann, weil die Zugriffsverteilung abgetrennt ist, sichergestellt werden, dass auf jede der gekoppelten Ressourcen fairer Zugriff bereitgestellt wird.

[0082] Im Folgenden Bezug auf die Fig. 4A Bezug nehmend, wird ein ausführlicheres Schaubild des SIBs **25a** gezeigt, das einen Eingangsverteiler **36** gekoppelt enthält, um MUX-Auswahlsignale<31:0> auf der Leitung **36a** zu acht gekoppelten Multiplexern **34a**–**34h** bereitzustellen, wo vier der Mux-Auswahlsignale zu jedem der acht Multiplexer weitergeleitete werden, um einen von neun Ausgängen an jedem Multiplexer auszuwählen. Alle SIBs **25a**–**25d** sind gleichartig aufgebaut und infolgedessen wird nur einer davon ausführlich beschrieben. Wie oben beschrieben, sind mit dem SIB potenziell zehn Ressourcen gekoppelt. Eine der zehn Ressourcen ist eine Anforderungsvorrichtung, die den Ausgang des SIBs empfängt, während die anderen neun Ressourcen Eingang in den SIB bereitstellen. Deshalb empfängt jeder der Multiplexer **34a**–**34h** den Eingang von neun mit dem SIB gekoppelten Ressourcen. Die Eingänge von drei der gekoppelten Prozessoren werden auf den Leitungen Px, Py und Pz empfangen. Weitere Eingänge von entweder dem vierten Prozessor (wenn der SIB der IOP-Vorrichtung zugehört) oder von dem der IOP-Vorrichtung (wenn der SIB einem der Prozessoren zugehört) werden auf der Leitung PW/IOP empfangen. Die Eingänge aus den Speicherbänken **13a**–**13d** werden jeweils auf den Leitungen mom0, mem1, mem2 und mem3 empfangen und Eingang von dem globalen Port wird auf der Leitung globaler Port empfangen.

[0083] Jeder Multiplexerausgang jedes der Multiplexer **34a**–**34h** ist mit einer der acht Speicherbänke eines Puffers **32** gekoppelt. Jede Speicherbank hat acht Einträge, wobei jede 128 Bits Daten und 16 Bits ECC speichert. Infolgedessen wird jedes Datenpaket, das in dem SIB empfangen wird, in vier verschiedene Speicherbänke in dieselbe Zeile des Puffers **32** geschrieben. Wie unten beschrieben, erhält der Eingangsverteiler **36** den Status der Bits, um die Speicherbänke des Puffers anzuzeigen, die zum Speichern von Daten verfügbar sind. Folglich wählt der Eingangsverteiler **26** bei jedem Zyklus dieser von einer Ressource oder von mehreren Ressourcen empfangenen 128 Bits Paketdaten einen der möglichen neun Ressourceneingänge in jedem der Multiplexer **34a**–**34d** zum Weiterleiten des Zyklus von Paketdaten zu der zugehörigen Speicherbank **32a**–**32h** in Abhängigkeit von dem Verfügbarkeitsstatus der Speicherbänke aus. Der Eingangsverteiler stellt außerdem auf der Leitung **36b** eine Bypass-Datenleitung zu dem Multiplexer **30** bereit. Wenn die Statusbits in dem Eingangsverteiler anzeigen, dass alle der Speicherbänke **32a**–**32h** leer sind, kann eine der neun Ressourcen über den Eingangsverteiler direkt zu dem zugehörigen Anfordernden überbrückt werden.

[0084] Jede der Speicherbänke **32a**–**32h** ist mit dem Multiplexer **30** gekoppelt. Der Multiplexer **30** wird durch den Ausgangsverteiler **38** gesteuert. Wenn der mit dem SIB **25a** verbundene Anfordernde bereit ist, die Daten aus dem SIB zu empfangen und ein Teil eines Pakets in den SIB geschrieben worden ist, leitet der Ausgangsverteiler einen der acht Eingänge aus den Speicherbänke **32a**–**32h** zu dem Anfordernden. Alternativ leitet der Ausgangsverteiler die Bypass-Daten auf der Leitung **36b** zu dem Anfordernden weiter, wenn keine der Speicherbänke zur Übertragung anstehende Daten hat und auf der Leitung **36** von dem Eingangsverteiler Daten verfügbar sind.

[0085] In Betrieb wird, wenn die ersten 128 Bits der Paketdaten auf der SIB-Seite empfangen werden, eine der acht Speicherbänke zum Speichern der ersten 128 Bits der Paketdaten ausgewählt.

[0086] Gemäß einem Ausführungsbeispiel der Erfindung wird während der nächsten drei Zyklen, die diese 128 Bits Paketdaten empfangen werden, die Speicherbank, die an die Speicherbank, die zuvor zum Schreiben ausgewählt war, angrenzt, zum Schreiben der nächsten 128 Bits Paketdaten verwendet. Wenn beispielsweise Speicherbank **32a** als eine verfügbare Speicherbank zum Schreiben des ersten Zyklus der Paketdaten aus der Quelle mem0 verwendet werden würde, würde der zweite Zyklus von Paketdaten in die Speicherbank **32b** geschrieben werden, der dritte in die Speicherbank **32c** und der vierte in die Speicherbank **32d**. Die Auswahl, welche Speicherbank zum Schreiben von folgenden Zyklen von Paketdaten zu verwenden ist, wird folglich auf einer Rotationsbasis, die mit der durch den Eingangsverteiler ausgewählten Speicherbank beginnt und mit einer angrenzenden Speicherbank für das sich daran anschließende Paketschreiben fortsetzt, erfolgen. Im Ergebnis wird das empfangene Paket über vier Speicherbänke in einer gewöhnlichen Zeile des Puffers **32** verteilt.

[0087] Weil acht Speicherbänke bereitgestellt werden und weil in einem Ausführungsbeispiel der Erfindung die Höchstanzahl von Ressourcenauslesungen, die an jedem der Anfordernden unerledigt sind, acht ist, kann sichergestellt werden, dass wenigstens eine Speicherbank für jede Ressource für jeden Schreibzyklus verfügbar ist. Deshalb könnte jede der Speicherbänke **32a**–**32h**, wenn zu einem gegebenen Zeitpunkt alle acht unerledigten Antworten durch den Switch empfangen werden würden, verwendet werden, um den ersten Paketdatenzyklus des Schreibens mit der Auswahl der Speicherbänke, die für die nächsten drei Zyklen rotieren, aufzunehmen.

[0088] In einem Ausführungsbeispiel der Erfindung arbeitet jeder Puffer in einem SIB unter dem FIFO-Protokoll (First-In First-Out protocol). Weil zwei Teile des Pakets simultan empfangen werden können, wird für diese eine Reihenfolge ausgewählt, mit der sie in den Switch gelesen werden. Da die Logik in dem Anfordernden, die die Ressourcen verteilt, nicht mit dem SIB kommuniziert und nicht mit anderen Anfordernden kommuniziert, um den Zugriff zu verteilen, wird einer Standardregel gefolgt, um die Datenintegrität sicherzustellen: beispielsweise eine Regel wie „Daten von einer niedrigeren Anzahl von Eingangsressourcen werden immer vor Daten von einer höheren Anzahl von Eingangsressourcen in den Switch geschrieben", wobei den Ressourcen eine feststehende Prioritätsanzahl zugewiesen wird.

[0089] Wie oben erwähnt, wurde in dem Ausführungsbeispiel des in der Fig. 4A gezeigten SIBs, die Verwendung von acht Speicherbänken beschrieben, weil acht der Anzahl der unerledigten Speicherabfragen entspricht, die ein Anfordernder zu jedem gegebenen Zeitpunkt haben kann. Wenn jedoch die Designbeschränkungen erfordern, dass weniger Speicherbänke bereitgestellt werden, kann das Design durch einen Fachmann in dieser Technik einfach modifiziert werden, um zu ermöglichen, dass mehrere Einheiten von Daten unter Verwendung von Verschachtelung oder einer ähnlichen Technik simultan in verschiedene Speicherorte einer gewöhnlichen Speicherbank geschrieben werden. Deshalb ist die vorliegende Erfindung nicht auf das bestimmte, in der Fig. 4A dargestellte Ausführungsbeispiel, beschränkt.

[0090] Wie oben beschrieben, unterhält der Verteiler im Betrieb Statusinformationen in Bezug auf die Verfügbarkeit der Eintragungen in die Speicherbank, um eine adäquate Speicherbank für das Schreiben der Daten aus einer Ressource auszuwählen.

[0091] Eine exemplarische Ausführung eines Eingangsverteilers **36** zum Steuern der Eingänge in den SIB ist in der Fig. 4B gezeigt. In der Fig. 4B wird, obwohl oben neun Eingänge beschrieben wurden, aus Gründen der Übersichtlichkeit die Logik zum Steuern des Schreibens von nur zwei Ressourceneingaben gezeigt. Wenn die eingegebenen Paketdaten auf den Leitungen **35** empfangen werden, wird ein Anzeigesignal, wie zum Beispiel „Eingang1", zu einer Signalspeicherkette **40** weitergeleitet, die vier Signalspeicher, Flip-Flops oder ähnliche Statuseinrichtungen umfasst. Die Signalspeicherkette **40** wird als ein Zählmechanismus verwendet. Für den Zweck dieses Beispiels wird vorausgesetzt, dass die Daten in vier aufeinander folgenden Übertragungszyklen empfangen werden. Während der vier Datenübertragungszyklen propagiert das Eingang1-Signal durch die Signalspeicherkette. Mit der Signalspeicherkette ist ein ODER-Gatter **46** gekoppelt. Während der Eingangsverteilerwert durch die Signalspeicherkette **40** propagiert, wird der Ausgang des ODER-Gatters **46** aktiviert.

[0092] Der Ausgang des ODER-Gatters **46** stellt zu einem Verschieberegister ein Wechselzeichen bereit. Das Schieberegister umfasst acht Bit-Speicherstellen, eine für jede der Speicherbänke des SIBs. Das Schieberegister **48** wird bei Eingangsempfang des Eingang1-Signal-Samples mit einem Vektor der Speicherbankauswahllogik **44** geladen. Der von der Speicherbankauswahllogik **44** empfangene Bit-Vektor hat nur ein Bit-Set, wobei die relative Speicherstelle des Bits innerhalb des Vektors der Speicherbank anzeigt, an welcher das Schreiben der Paketdaten zu beginnen ist.

[0093] Die Speicherbankauswahllogik **44** steuert infolgedessen die Zieladresse des ersten Zyklus von Paketdaten. Die Speicherbankauswahllogik **44** empfängt als einen Eingang einen verfügbaren Vektor **42** mit den relativen Speicherstellen der Bits in dem verfügbaren Vektor, der die zugehörigen Puffer, die nicht in der Lage sind, die Schreibdaten zu empfangen, anzeigt.

[0094] Wenn die Speicherbankauswahllogik ein Bit zu dem Schieberegister **48** bereitstellt, wird der Wert des Schieberegisters **48** zu einem Demultiplexer **49** weitergeleitet. Der Demultiplexer **49** empfängt außerdem als Eingang eine numerische Darstellung des Eingangs der Multiplexer **34a**–**34h**, mit dem die Eingang1-Quelle verbunden ist. Beispielsweise empfängt der Demultiplexer **49** einen Eingangswert „1", anzeigend, dass die Eingang1-Ressourcendaten durch den Multiplexer **3a**, der einen Multiplexer-Auswahlwert von „1" verwendet, weitergeleitet werden würden. In Abhängigkeit von der Speicherstelle des Bits in dem Schieberegister, das die ausgewählte Speicherbank anzeigt, wird der Wert 1 zu der adäquaten Speicherstelle des Mux-AUSWAHL-Si-

gnals **36a**, <31:0>, propagiert. Jeder der Demultiplexer für jede Eingangsressource treibt alle der Mux-AUSWAHL-Signale, wobei ihre Ausgänge geodert sind, bevor die Signale die Multiplexer **34a**–**34h** treiben.

[0095] Nach dem Schreiben der Speicherbankeinträge werden die Inhalte des Schieberegisters durch das ODER-Gatter **50** zusammengeodert und als der VERFÜGBARE SPEICHER-BANKVEKTOR **42** gespeichert. Dieser wird während des nächsten Zyklus zum Bestimmen durch die Speicherbankauswahllogik **44**, welche Speicherbänke für eingehende Schreiben verfügbar sind, verwendet.

[0096] Jeder Zyklus, in dem der Zeichenwechsel auf der Leitung **46a** aktiviert wird, resultiert darin, dass das Bit des Schieberegisters **48** nach rechts verschoben wird. Während das Bit nach rechts verschoben wird, wird der Auswahlwert in dem Mux-Auswahlsignal<31:0> ebenso nach rechts verschoben, wodurch veranlasst wird, dass die Eingangswertquelle für den nächsten Schreibvorgang zu der nächsten angrenzenden Speicherbank verschoben wird.

[0097] Folglich wird durch das Verwenden eines SIBs innerhalb des lokalen QSD-Switchs ein unkomplizierter und effizienter Verbindungsmechanismus bereitgestellt, der in Lage ist, sicherzustellen, dass mehrere gleichzeitig empfangene Eingänge die Zieladressen ihrer Anfordernden erreichen. Mit einer derartigen Anordnung ist, sobald eine Quelle den Zugriff auf eine Ressource verteilt hat, jede erforderliche Zugriffsverteilung, die durch die Quelle durchgeführt werden muss, ausgeführt. Die Quelle kann sich auf die Tatsache verlassen, dass die Ressource immer in der Lage sein wird, Zugriff auf den Switch-Puffer **32** zu erhalten. Das Ermöglichen, dass die Quellen-Verteiler zum Verwalten einer Ressource unabhängig voneinander arbeiten können, stellt einen Mechanismus bereit, der mit minimalem Aufwand eine faire Zugriffsverteilung bereitstellt. Weil der SIB in der Lage ist, Daten für die Höchstanzahl von ausstehenden Auslesungen des Anfordernden selbst dann zu speichern, wenn die Daten gleichzeitig von allen der Ressourcen empfangen werden, besteht zusätzlich kein Bedarf für die Zugriffsverteilung zwischen den Ressourcen für den Puffer **32** und die Gesamtkomplexität der Ressourcenlogik wird verringert.

[0098] Im Folgenden Bezug nehmend auf die Fig. 5, wird ein zweites Ausführungsbeispiel eines Simultan-Eingabepuffers (SIB) **61** beschrieben, der entweder mit einem Prozessor oder einer IOP-Einrichtung (jede Vorrichtung zum Anfordern, die einen Cache enthält) gekoppelt werden kann, wie in der Fig. 3 gezeigt. Der SIB **61** enthält neun Multiplexer **60a**–**60i**, von denen acht mit einem jeweiligen von acht Puffern **62a**–**62h** gekoppelt sind.

[0099] Der neunte Multiplexer **60i** wird verwendet, um einen Bypasspfad bereitzustellen, wie unten beschrieben wird. Die Multiplexer **60a**–**60i** empfangen neun Eingänge einschließlich vier Eingänge der gekoppelten Speichereinrichtungen mem0–mem3, eines Eingangs des globalen Ports und drei Eingänge der gekoppelten Prozessaren auf den Leitungen Px, Py und Pz und eines Eingangs entweder des IOPs (wenn die dem SIB zugehörige Einrichtung ein Prozessor ist) oder eines weiteren Prozessors (wenn die dem SIB zugehörige Einrichtung das IOP ist) auf der Leitung PW/IOP.

[0100] Jeder der Puffer **62a**–**62h** enthält vier 128-Bit-Einträge. Dementsprechend speichert jeder der Eingangspuffer ein 512-Bit-Paket Information, das in vier 128-Bit-Teilen in aufeinander folgenden Zyklen an dem SIB empfangen wird. An jeden der Puffer ist jeweils ein Multiplexer **64a**–**64h** vier bis eins gekoppelt. Diese Multiplexer werden verwendet, um einen der vier Einträge der zugehörigen Puffer zum Weiterleiten durch einen Multiplexer **66** zu dem Ausgang des SIBs auszuwählen.

[0101] Wie oben in Bezug auf die Fig. 4A beschrieben, sind acht Puffer enthalten, weil in einem Ausführungsbaispiel der Erfindung jeder Anfordernde zu jedem gegebenen Zeitpunkt höchstens acht ausstehende Lesereferenzen für verschiedene Ressourcen haben kann. Infolgedessen ist dies keine Beschränkung der Erfindung, obwohl in der Fig. 5 acht Puffer gezeigt werden. Stattdessen ist die gewählte Anzahl der Puffer von den Pufferungseigenschaften des zugehörigen Prozessors oder der IOP-Einrichtung abhängig.

[0102] In Betrieb wählt der Eingangsverteiler **67**, während von jeder der gekoppelten Ressourcen Eingang empfangen wird, eine der Eingangsleitungen jedes der Multiplexer zum Weiterleiten des Datenpakets zu einem freien Puffer. Während der Dauer eines Paketschreibens einer gegebenen Ressource wird derselbe Puffer gewählt, so dass alle Teile des Pakets in einem einzelnen Paket bleiben. Sobald wenigstens ein Teil des Pakets in den Puffer geschrieben wurde, kann es an den Multiplexer **66** zum Weiterleiten zu dem zugehörigen Anfordernden, wenn der Anfordernde bereit ist, bereitgestellt werden. Alternativ, wenn keine Paketdaten in einem der Puffer sind, kann zum Weiterleiten der Paketdaten direkt durch den Multiplexer **60i** über den Multiplexer **66** ein Bypasspfad zu dem Ausgang verwendet werden.

[0103] Weil acht Puffer bereitgestellt werden, ist die SIB-Vorrichtung **61** in der Lage, Daten von jeder der gekoppelten Ressourcen im Wesentlichen gleichzeitig (d. h. in demselben Zyklus) zu empfangen. Durch das Verwenden eines SIBs in dem QSD, wie in dem vorhergehenden Ausführungsbeispiel, ist keine Verteilung des SIBs zwischen den Anfordernden erforderlich. Im Ergebnis ist die Verfügbarkeit des lokalen Switchs garantiert, wenn die Ressource bereit ist, diesen zu verwenden. Zusätzlich wird ein Verteilungsschema bereitgestellt, das inhärent fair ist, weil im Ergebnis der Verteilung des Switchs keine Anforderung zu einer Ressource durch andere Anforderungen zu anderen Ressourcen blockiert wird. Dementsprechend ist eine faire und relativ einfache Struktur bereitgestellt, die ermöglicht, die maximale Busbandbreite aufrechtzuerhalten, während der Verteilungsaufwand minimiert wird.

[0104] Folglich wird der Multiprozessorknoten **10** bereitgestellt, der die Verarbeitungsressourcen durch das Implementieren eines lokalen Switchs, der zum Unterstützen einer hohen Busbandbreite einen Simultan-Eingabepuffer verwendet, optimal nutzt. Zusätzlich ist, weil in dem Verteilerbus **13** eine Reihenfolge von Referenzen serialisiert wird, ein zentraler Ordnungspunkt bereitgestellt, der das Aufrechterhalten der Kohärenz des Speichers des Multiprozessorknotens **10** vereinfacht. Während die Möglichkeit zum Erhöhen der Rechenleistung durch das Erhöhen der Annzahl von mit dem lokalen Switch gekoppelten Prozessormodulen vorhanden ist, stellt die Anordnung mit vier Prozessoren pro lokalem Switch ein System mit hoher Leistung und geringer Latenz zu geringen Kosten bereit.

Großes symmetrisches Mehrprozessorsystem

[0105] Die Anzahl von Prozessoren, die in einem monolithischen Mehrprozessorknoten enthalten sein können, ist durch zwei Faktoren begrenzt. Zum einen ist die Anzahl von Prozessoren, die über einen lokalen Switch zusammengekoppelt werden können, durch die Anzahl von Anschlussstiften begrenzt, die auf Chips, die den lokalen Switch bilden, vorhanden sind. Zweitens ist die Datenbandbreite begrenzt, die durch einen einzelnen, monolithischen Switch unterstützt wird. Infolgedessen kann durch das Erhöhen der Anzahl von gekoppelten Prozessoren über einen bestimmten Punkt hinaus kein Leistungsgewinn erzielt werden.

[0106] Gemäß einem Ausführungsbeispiel der vorliegenden Erfindung kann ein großes symmetrisches Mehrprozessorsystem durch das Verbinden einer Vielzahl von Multiprozessorknoten über einen hierarchischen Switch bereitgestellt werden. Beispielsweise können acht der Mehrprozessorknoten über den hierarchischen Switch gekoppelt werden, um ein symmetrisches Mehrprozessorsystem (SMP) bereitzustellen, das 32 Prozessormodule, acht IOP-Vorrichtungen und 256 Gigabytes Speicher enthält. Für den Zweck dieser Beschreibung wird ein SMP, das wenigstens zwei Mehrprozessorknoten enthält, als ein großes SMP bezeichnet. Wie unten ausführlicher beschrieben, kann durch das Koppeln einer kleinen Anzahl von Prozessoren, die einen lokalen Switch an einem SMP-Knoten verwenden, und anschließendes Koppeln einer Anzahl von Knoten, die einen hierarchischen Switch nutzen, zu einem großen SMP ein skalierbares Hochleistungssystem realisiert werden.

[0107] Um die Mehrprozessorknoten in ein hierarchisch geschaltetes Netzwerk zu koppeln, wird der Mehrprozessorknoten erweitert, um eine Globalport-Schnittstelle zu enthalten. Beispielsweise, im Folgenden auf die Fig. 2 Bezug nehmend, koppelt ein lokaler Switch **110** vier Prozessormodule, vier Speichermodule und ein IOP-Modul. Gleiche Elemente in den Fig. 2 und Fig. 6 haben dieselben Bezugsnummern. Der lokale Switch **110** des Mehrprozessorknotens **100** ist ein 10-Port-Switch, der 9 Ports **116a**–**116i** hat, die gleichartig wie die Ports **16a**–**16i** der Fig. 2 aufgebaut sind. Ein zusätzlicher Port **116j** stellt über den globalen Link **132** einen vollduplex taktweitergeleiteten Datenlink **120** bereit.

[0108] Der globale Port koppelt einen Mehrprozessorknoten mit dem hierarchischen Switch und realisiert infolgedessen ein großes SMP. Im Folgenden auf die Fig. 7A Bezug nehmend, wird in einem Ausführungsbeispiel der Erfindung beispielsweise ein großes SMP-System **150**, das acht Knoten **100a**–**100h**, die über einen hierarchischen 8 × 8-Switch **155** zusammengekoppelt sind, gezeigt. Jeder der Knoten **100a**–**100h** ist im Wesentlichen mit dem in der Fig. 6 gezeigten Knoten **100** identisch.

[0109] Jeder der Knoten **100a**–**100h** ist durch einen jeweiligen hierarchischen Switch **170a**–**170h** mit dem Switch **155** gekoppelt. In einem Ausführungsbeispiel werden die Datenlinks **170a**–**170h** mit einer Taktgeschwindigkeit von 150 MHz betrieben und unterstützen folglich 2,4 GBytes/sek. Datenbandbreite zum Übertragen von Daten zu und von dem Switch **155**. Dies versieht den Switch mit einem Maximum von 38,4 GBytes/sek. Rohverbindungsdatenbandbreite und 19,2 GBytes/sek. Bisektionsdatenbandbreite.

[0110] Das große SMP-System ist ein verteiltes System mit gemeinsamem Speicher, wobei jeder der Mehrprozessorknoten **100a**–**100h** entweder einen adressierbaren Teil des Gesamtsystemspeichers oder einen un-

terteilten Teil des physikalischen Speichers enthält. In einem Ausführungsbeispiel der Erfindung sind in dem Gesamtsystemspeicher 2^{43} physikalische Speicherstellenadressen vorhanden. Ein Ausführungsbeispiel des SMP-Mehrprozessorsystems **100** unterstützt zwei Adressenformate, die als „großes Format" und „kleines Format" bezeichnet werden. Das große Format bildet physikalische 43-Bit-Adressen ab, auf denen die Prozessoren in jedem Knoten direkt in eine physikalische 43-Bit-Adresse zur Verwendung in dem Mehrprozessorsystem arbeiten. Durch das Adressieren des großen Formats können die Bits<38:36> der physikalischen Speicheradresse als eine Knotenidentifikationsnummer verwendet werden. Die Adressenbits 38:36 dekodieren den Home-Knoten eines I/O-Adressraums. wobei „Home" den physikalischen Mehrprozessorknoten bezeichnet, auf dem die Speicher- und I/O-Vorrichtungen, die dem Speicherraum oder dem I/O-Raum zugehörig sind, vorhanden sind.

[0111] Der Adressiermodus des kleinen Formats setzt voraus, dass nicht mehr als vier Knoten in dem Mehrprozessorsystem vorhanden sind. Das kleine Format ermöglicht den Prozessoren, in jedem Knoten in einem physikalischen 36-Bit-adressierten System zu arbeiten. Bei einem kleinen Format identifizieren die Bits 34:33 der physikalischen Adresse die Home-Knotennummer von Daten oder einer I/O-Vorrichtung.

[0112] Obwohl die CPU unter Verwendung einer physikalischen 36-Bit-Adresse arbeitet, verwendet das Mehrprozessorsystem jedoch konsistent physikalische 43-Bit-Adressen zum Bestimmen der Datenspeicherorte, wobei die Bits 37:36 der physikalischen Adresse die Home-Knotennummer von Daten oder eine I/O-Vorrichtung kennzeichnen. Dementsprechend wird zwischen den durch die CPU ausgegebenen Adressen des kleinen Formats und denen, die über die Datenleitungen **13a**–**13h** zu dem hierarchischen Switch **155** übertragen werden, Translation durchgeführt.

[0113] Die dargestellte Anordnung des Mehrprozessorsystems **150** ist in der Lage, den cachekohärenten gemeinsamen Speicher mit hoher Bandbreite zwischen 32 Prozessoren bereitzustellen.

[0114] Ein weiteres Ausführungsbeispiel eines großen SMPs gemäß der Erfindung wird in der Fig. 7B gezeigt, in der zwei Mehrprozessorknoten **100a** und **100b** ohne Verwendung des hierarchischen Switchs zusammengekoppelt sind. Stattdessen sind die beiden Mehrprozessorknoten durch das Zusammenkoppeln der Ausgänge ihrer globalen Ports direkt gekoppelt.

[0115] Unabhängig davon, ob die Zwei-Knoten-Ausführung der Fig. 7B oder die Mehrknoten-Ausführung der Fig. 7A verwendet wird, ist das Ergebnis ein Mehrprozessorsystem mit einem großen Adressraum und einer großen Verarbeitungsleistung.

[0116] In beiden Ausführungsbeispielen sind die Speicheradressräume und die I/O-Adressräume physikalisch zwischen allen Knoten **100a**–**100h** in Segmenten verteilt. Jeder Knoten in dem System enthält einen Teil des Hauptspeichers, auf den unter Verwendung der oberen drei Bits der physikalischen Adresse des Speicherraums zugegriffen wird. Infolgedessen bildet jede Speicher- oder I/O-Adresse in einer (und nur in einer) Speicherstelle oder I/O-Vorrichtung in nur einem der Knoten ab. Die oberen drei Adressbits stellen infolgedessen eine Knotennummer zum Identifizieren des „Home-Knotens" des Knotens, in den die Speicher- oder I/O-Adresse abbildet, dar.

[0117] Jeder Mehrprozessorknoten kann auf Teile des gemeinsamen Speichers, die in ihrem Home-Knoten oder in anderen Mehrprozessorknoten gespeichert sind, zugreifen. Wenn ein Prozessor auf einen gemeinsamen Speicherblock, für den der Home-Knoten der eigene Knoten des Prozessors ist, zugreift (lädt oder speichert), wird die Referenz als eine „lokale" Speicherreferenz bezeichnet. Wenn die Referenz zu einem Block verweist, für den der Home-Knoten ein anderer als der eigene Knoten des Prozessors ist, wird die Referenz als eine „Remote"- oder „globale" Speicherreferenz bezeichnet. Weil die Latenz eines lokalen Speicherzugriffs verschieden von der eines Remote-Speicherzugriffs ist, wird über das SMP-System ausgesagt, dass es eine NUMA-Architektur (Architektur des nicht gleichförmigen Speicherzugriffs) hat. Da das System des Weiteren kohärente Caches bereitstellt, wird das System als eine cache-kohärente NUMA-Architektur bezeichnet.

[0118] Die hierin offen gelegte cache-kohärente NUMA-Architektur enthält mehrere Aspekte, die zu ihrer hohen Leistung und geringen Komplexität beitragen. Ein Aspekt der Konstruktion ist das Einhalten der Reihenfolge zwischen Nachrichten und die Ausnutzung dieser Reihenfolge. Durch das Sicherstellen, dass Nachrichten in Übereinstimmung mit bestimmten Reihenfolgeeigenschaften durch das System fließen, können die Latenzen der Operationen signifikant verringert werden. Beispielsweise erfordern Speichervorgänge nicht, dass Invalidate-Nachrichten an ihre ultimativen Zieladressenprozessoren geliefert werden, bevor das Speichern als ausgeführt betrachtet wird, stattdessen wird die Speicherung als ausgeführt betrachtet, sobald die Invalida-

te-Nachrichten zu bestimmten geordneten Warteschlangen, die zu dem Zieladressenprozessor führen, gesendet wurden.

[0119] Zusätzlich eliminiert die Konstruktion durch das Garantieren, dass bestimmte Reihenfolgen eingehalten werden, den Bedarf für Bestätigungsmeldungen oder Erledigungsmeldungen. Für die Nachrichten ist garantiert, dass sie ihre Zieladressen in der Reihenfolge, in der sie in bestimmte Warteschlangen eingereiht wurden, erreichen. Infolgedessen besteht kein Bedarf, mit einer Bestätigung zu antworten, die eliminiert wird, wenn die Nachricht ihre Zieladresse erreicht. Dies verbessert die Bandbreite des Systems.

[0120] Zusätzlich werden das Ordnen von Ereignissen und das Ordnen von Nachrichten verwendet, um einen „Hot-Potato-Betrieb" zu erreichen. Durch Ausnutzung der Reihenfolge in bestimmten Warteschlangen sind Controller, wie zum Beispiel der Verzeichnis- oder DTAG-Controller, in der Lage, Anforderungen mit einer einzigen Kontrolle zurückzuziehen. Negativbestätigung und Wiederholung einer Anforderung auf Grund von Konflikten mit anderen Anforderungen sind nicht erforderlich. Als eine Konsequenz des „Hot-Potato-Betriebs" werden Fairness- und Aushungerungsprobleme eliminiert.

[0121] Der zweite Aspekt der Konstruktion sind die verwendeten virtuellen Kanäle. Virtuelle Kanäle sind ein Schema zum Kategorisieren von Nachrichten in Kanälen", wobei die Kanäle physikalische Ressourcen gemeinsam nutzen können (und infolgedessen „virtuell" sind), jedoch jeder Kanal unabhängig von den anderen Kanälen flussgesteuert ist.

[0122] Virtuelle Kanäle werden verwendet, um durch das Eliminieren von Flussbhängigkeits- und Ressourcenabhängigkeitszyklen zwischen den Nachrichten in dem System Systemblockade in dem Cache-Kohärenz-Protokoll zu beseitigen. Dies steht im Gegensatz zu Cache-Kohärenz-Protokollen in NUMA-Mehrprozessoren nach dem Stand der Technik, die Mechanismen zum Erfassen von Verklemmungen anwenden und anschließend Verklemmungssituation durch negativ bestätigende Auswahlnachrichten und das Wiederholen entsprechender Befehle lösen.

[0123] Obwohl später hierin eine ausführliche Beschreibung der Kanäle erfolgt, wird im Folgenden die Verwendung der Kanäle kurz beschrieben. Wie oben erwähnt, werden die Nachrichten innerhalb des großen SMP-Systems unter Verwendung von logischen Datenpfaden, die als „Kanäle" bezeichnet werden, gelenkt. In einem Ausführungsbeispiel der Erfindung sind die folgenden Kanäle enthalten: ein Q0-Kanal zum Befördern von Transaktionen von einem anfordernden Prozessor zu dem Verteilerbus auf dem Home-Knoten, der der Adresse der Transaktion entspricht, ein Q1-Kanal zum Befördern von Transaktionen von einem Home-Verteilerbus zu einem Prozessor oder zu mehreren Prozessoren und dem IOP und ein Q2-Kanal zum Befördern von Datenfülltransaktionen von einem Besitzerprozessor zu dem anfordernden Prozessor. Ein Q0Vic-Kanal kann zum Befördern von Victim-Transaktionen von einem Prozessor zu einem Speicher zum Schreiben modifizierter Daten bereitgestellt werden. Zusätzlich kann der Q0Vic-Kanal verwendet werden, um Q0-Transaktionen zu befördern, die hinter den Victim-Transaktionen bleiben müssen. Schließlich wird ein QIO-Kanal bereitgestellt, um die IO-Raum-Transaktionen von einem Prozessor zu einem IOP zu transportieren.

[0124] Die Kanäle bilden eine Hierarchie, die unten gezeigt wird:
(niedrigster) QIO > Q = Vic → Q0 → Q1 – Q2 (höchster)

[0125] Wie im Folgenden noch beschrieben wird, sollten die Nachrichten, um Verklemmung zu vermeiden, in jedem Kanal niemals auf Grund von Nachrichten in einem niedrigeren Kanal blockiert werden. Weitere Einzelheiten in Bezug auf die Konstruktion und die Implementierung von Mechanismen, die die Ordnungseigenschaften und die virtuellen Kanäle bereitstellen und anwenden, werden später hierin beschrieben.

[0126] Folglich kann, wie in den **Fig. 7A** und **Fig. 7B** gezeigt, durch das Zusammenkoppeln jeder Anzahl von SMP-Knoten der **Fig. 2** ein großes SMP-System bereitgestellt werden. Der Betrieb eines großen SMP-Systems, wie dem, das in den **Fig. 7A** und **Fig. 7B** gezeigt ist, wird unten bereitgestellt und in drei Teilen beschrieben. Zuerst werden die Hardwarekomponenten, die in dem großen SMP enthalten sein können, beschrieben. Anschließend wird das Cache-Kohärenz-Protokoll, das die kohärente gemeinsame Datennutzung zwischen den Prozessoren in dem SMP bereitstellt, beschrieben. Zusätzlich werden die Implementierung und die Verwendung von virtuellen Kanälen einschließlich der Unterstützungsmechanismen, die für die virtuellen Kanäle in dem hierarchischen Switch bereitgestellt werden, beschrieben.

Hardwarekomponenten des großen SMPs

[0127] In jedem der Mehrprozessorknoten werden mehrere Elemente zum Implementieren der kohärenten gemeinsamen Datennutzung bereitgestellt. Wiederum auf die Fig. 6 Bezug nehmend, enthalten diese Elemente das Verzeichnis **140**, den DTAG **20**, den IOP-Tag **14b**, den globalen Port **120** und zusätzlich eine Hierarchie von Serialisierungspunkten, die ermöglicht, dass eine Reihenfolge von Referenzen aufrechterhalten wird, um das Cache-Kohärenz-Protokoll zu erleichtern. Jedes dieser Elemente wird im Folgenden ausführlicher beschrieben.

Der globale Port

[0128] Der globale Port ermöglicht, dass der Mehrprozessorknoten **100** über einen hierarchischen Switch **170** direkt mit einem oder mit mehreren gleichartig aufgebauten Mehrprozessorknoten gekoppelt wird. Weil jeder Mehrprozessorknoten **100** als ein symmetrisches Mehrprozessorsystem arbeitet, werden der verfügbare Adressraum und die Verarbeitungsleistung erhöht, je mehr Knoten in das System eingefügt werden.

[0129] Im Folgenden auf die Fig. 8 Bezug nehmend, wird ein erweitertes Blockdiagramm eines globalen Ports **120** gezeigt. Der globale Port enthält eine Transaktions-Tracking-Verzeichnis (TTT) **122**, einen Victim-Cache **124**, die Paketwarteschlangen **127**, **122**, **123** und **125** zum Speichern von Paketen, die von dem Mehrprozessorknoten zu dem hierarchischen Switch weitergeleitet wurden, und eine Paketwarteschlange **121** zum Speichern von Paketen, die aus dem hierarchischen Switch empfangen wurden. Der globale Port **120** kommuniziert über den Verteilerbus **130** und zwei zugewiesene Ports aus dem lokalen Switch, zum Beispiel dem GP-Link-In **132b** und dem GP-Link-Out **132a**, mit der weiteren Logik in dem Knoten.

[0130] Das TTT verfolgt die ausstehenden Transaktionen in dem Mehrprozessorknoten, d. h. jene Transaktionen, die von dem Knoten über den globalen Port ausgegeben wurden und die auf Beantwortung von einem anderen Mehrprozessorknoten oder von dem hierarchischen Switch warten. Jedes Mal, wenn entsprechende Antworten in dem Knoten empfangen wurden, wird der TTT-Eintrag gelöscht. Das TTT besteht aus zwei Teilen: dem Q0-TTT und dem Q1-TTT, wobei Q0 und Q1 Pakete bezeichnen, die auf den Q0- und Q1-Kanälen wie oben beschrieben unterwegs sind. Die Einzelheiten, wie dem TTT ein Eintrag zugeteilt wird und wann dieser zurückgezogen wird, werden im Folgenden ausführlicher beschrieben.

[0131] Der globale Port **120** enthält ebenso den Victim-Cache **124**. Der Victim-Cache **124** speichert aus jedem der Prozessoren des Mehrprozessorknotens empfangene und für den Speicher eines anderen Mehrprozessorknotens bestimmte Victim-Daten. Victim-Daten sind Daten, die in einer Cache-Speicherstelle in dem Prozessor gespeichert waren und durch diesen Prozessor modifiziert wurden. Wenn neue Daten in dem Prozessor empfangen werden, die an der Speicherstelle, die die modifizierten Daten speichert, gespeichert werden müssen, werden die modifizierten Daten als Victim-Daten bezeichnet.

[0132] Der Victim-Cache **124** stellt vorübergehendes Speichern von Victim-Daten, die von einem Prozessor zu einem Speicher auf einem Remote-Prozessorknoten zielen, bereit. Wenn die Gelegenheit des Übertragens der Victim-Daten über den globalen Port zu einem anderen Knoten zu übertragen, vorhanden ist, wird ein Multiplexer **167** geschaltet, um die Daten aus dem Victim-Cache **124** auf dem Ausgangsteil des Busses **170** bereitzustellen. Das Bereitstellen eines Victim-Caches an dem globalen Port ermöglicht den Prozessoren, ihren jeweiligen Victim-Datenpuffer zu leeren, ohne die einzelnen Prozessoren die Speicherschreiblatenz des Gesamtsystems abwarten zu lassen. Stattdessen können Victim-Schreibvorgänge durch den globalen Port derartig gesteuert werden, dass das Schreiben immer dann durchgeführt wird, wenn ein verfügbarer Datenzyklus vorhanden ist. Es verbleiben einige Steuerprobleme, die die Angemessenheit des Freigebens von Daten aus dem Victim-Cache betreffen, diese Probleme werden jedoch unten beschrieben.

DTAG und IOP-Tag

[0133] Der DTAG und der IOP-Tag sind ebenso in dem kleinen SMP-System enthalten, werden jedoch weiter unten ausführlicher beschrieben. Der DTAG **20** speichert Statusinformationen für jeden der Datenblöcke, die in den Caches der Prozessoren des Mehrprozessorknotens gespeichert sind. Gleichermaßen speichert der IOP-Tag **14** Statusinformationen für jeden Datenblock, der in dem IOP gespeichert ist. Während das Verzeichnis grobe Informationen bereitstellt, die identifizieren, welcher der Mehrprozessorknoten Kopien der Daten speichert, können der DTAG und der IOP-Tag verwendet werden, um eine genauere Angabe dahingehend, welcher der Prozessoren innerhalb eines Mehrprozessorknotens Kopien der Daten speichert, bereitstellen. Deshalb werden, sobald eine Referenz den Mehrprozessorknoten erreicht hat, der DTAG und der IOP-Tag ver-

wendet, um zu bestimmen, welche Prozessoren in dem Knoten adressiert werden sollten.

[0134] Wie in der Fig. 6 gezeigt, werden der DTAG **20** und der IOP-Tag **14b** mit dem Verteilerbus **130** zum Überwachen der Adressen, die auf den mit dem QSA-Chip **18** gekoppelten Speicherbereich verweisen, gekoppelt. Der DTAG ist in vier Segmente unterteilt, die den vier Prozessoren **12a**–**12d** entsprechen. Jeder der Prozessoren enthält einen Cache (nicht gezeigt) zum vorläufigen Speichern einer Datenteilmenge aus dem Speicher **13**. Jedem Cache ist ein Tag-Speicher zum Speichern der oberen Adressbits (Tags) eines in dem Cache jedes Prozessors gespeicherten Speicherblocks zugehörig. Jedes Segment des DTAGs **20** enthält Daten, die den Zustand der Cache-Tags des zugehörigen Prozessors anzeigen. Das Speichern einer Kopie der Tags in dem DTAG **20** extern zu den Verarbeitungseinheiten ermöglicht dem System, auf dem Verteilerbus empfangene Befehle zu filtern und nur diejenigen Lese- und Befehle Invalidate, die mit den Daten in dem Cache des Prozessors verbunden sind, zu dem jeweiligen Prozessor weiterzuleiten. Der IOP-Tag **14a** speichert die oberen Adressbits jeder der in dem IOP-Cache **14c** gespeicherten Datenblöcke. Die IOP-Tag-Speicherung ist den Tag-Speicherungen, die in jedem der Prozessoren **12a**–**12d** unterhalten werden gleichartig.

[0135] Jeder Eintrag in den DTAG **20** und in den IOP-Tag **14a** enthält eine Anzahl von Statusbits, die einen der vier folgenden Zustände anzeigen: Dirty, Clean, Dirty_Not_Probed, Dirty_Probed. Diese Statusbits eines Eintrags in den IOP-Tag zeigen einen der folgenden Zustände an: Valid und Dirty. Ein Valid-Bit zeigt an, dass die in dem entsprechenden Eintrag des zugehörigen Caches gespeicherten Daten mit den im Speicher gespeicherten Daten übereinstimmen. Ein Dirty-Bit zeigt an, dass die in dem entsprechenden Eintrag des zugehörigen Caches gespeicherten Daten durch den zugehörigen Prozessor modifiziert wurden und nicht mit den im Speicher gespeicherten Daten übereinstimmen.

[0136] Auf den DTAG **20** und auf den IOP-Tag **14b** wird jedes Mal dann, wenn auf dem Verteilerbus eines Mehrprozessorknotens **100** ein Befehl erscheint, zugegriffen. Wenn auf den DTAG-Zugriff für Prozessor eins mit einem Ungültig-Status geantwortet wird, dann speichert der Prozessor eins aus dem Knoten keine gültige Kopie der Daten, die der Speicheradresse zugehörig sind. Wenn mit einem Gültig-Status auf den Zugriff zu dem IOP-Tag **14a** geantwortet wird, dann speichert der IOP-Cache **14c** eine gültige Kopie der Daten. Wenn in Reaktion auf einen DTAG-Zugriff auf den Prozessor eins mit einem Clean-Status geantwortet wird, zeigt dies an, dass der Prozessor eine nicht modifizierte Kopie der der Speicheradresse entsprechenden Daten hat, dass jedoch keine Versuche von irgendeinem anderen Prozessor unternommen wurden, die Daten zu lesen. Wenn in Reaktion auf den DTAG-Zugriff mit einem Status Dirty_Not_Probed geantwortet wird, zeigt dies an, dass der Prozessor eine modifizierte Kopie der der Speicheradresse entsprechenden Daten hat und dass wenigstens ein Prozessor versucht hat, die Daten zu lesen, seit der Prozessor die Daten zuletzt modifiziert hat.

Betrieb des Verzeichnisses

[0137] Im Allgemeinen wird das Verzeichnis verwendet, um Besitzerinformationen für jeden Speicherblock in dem zugehörigen Mehrprozessorknoten (dem Home-Knoten) bereitzustellen, wobei ein Speicherblock generell die kleinste Datenmenge ist, die zwischen dem Speicher und einem Prozessor in dem SMP-System übertragen wird. In einem Ausführungsbeispiel ist ein Block bis zu der Größe eines Pakets, 512 Bits (64 Bytes) Daten, entsprechend. Zusätzlich zeigt das Verzeichnis an, welcher Mehrprozessorknoten Kopien der Speicherblockdaten speichert. Infolgedessen identifiziert das Verzeichnis für die Lesebefehlstypen die aktuellste Version der Daten. Für die Victim-Befehlstypen, bei denen ein modifizierter Datenblock zurück in den Speicher geschrieben wird, wird das Verzeichnis geprüft, um zu bestimmen, ob der modifizierte Datenblock aktuell ist und in den Speicher geschrieben werden sollte. Deshalb ist das Verzeichnis der erste Zugriffspunkt für jede Referenz auf einen Speicherblock an dem zugehörigen Mehrprozessorknoten, unabhängig davon, ob die Referenz durch einen Prozessor in einem Remote-Mehrprozessorknoten oder in einem lokalen Mehrprozessorknoten ausgegeben wurde.

[0138] Das Verzeichnis speichert einen 14-Bit-Eintrag für jeden 64-Byte-Datenblock (im Folgenden als eine Cache-Zeile bezeichnet) des Speichers **13** in dem entsprechenden Knoten **100**. Wie der Speicher **13** wird auch das Verzeichnis physikalisch durch die Knoten in dem System verteilt, so dass sich, wenn sich eine Speicheradresse auf dem Knten N befindet, der entsprechende Verzeichniseintrag ebenso auf dem Knoten N befindet.

[0139] Im Folgenden Bezug nehmend auf die Fig. 9, wird ein Ausführungsbeispiel eines Verzeichniseintrags **140a** gezeigt, der ein Besitzer-ID-Feld **142** und ein Knotenpräsenzfeld **144** enthält. Das Besitzer-ID-Feld **142** umfasst 6 Bits Besitzerinformation für jeden 64-Byte-Block. Die Besitzer-ID spezifiziert den aktuellen Besitzer des Blocks, wobei der Besitzer entweder einer der 32 Prozessoren in dem System, einer der acht I/O-Prozessoren oder der Speicher ist. Die acht Bits der Knotenpräsenzinformation zeigen an, welcher der acht Knoten

in dem System die aktuelle Version jeder Cache-Zeile erlangt hat. Die Knotenpräsenz ist ein Grobvektor, wobei eines der Bits den kumulativen Status von vier Prozessoren an demselben Knoten darstellt. In dem Fall von gemeinsam genutzten Daten kann mehr als das Knotenpräsenzbit eingerichtet werden, wenn mehr als ein Knoten wenigstens einen Prozessor, der die Information speichert, hat.

[0140] Gelegentlich können bestimmte Teile der Zustandsinformation von entweder dem DTAG oder dem Verzeichnis erhalten werden. In derartigen Fällen ist die Statusinformation aus dem DTAG zu bevorzugen, da sie wesentlich schneller abgerufen wird. Wenn beispielsweise der Besitzerprozessor eine Speicheradresse in dem Home-Knoten für die Adresse ist, muss der DTAG verwendet werden, um die Besitzer-ID bereitzustellen.

[0141] Für Informationen über Referenzen, die aus Leistungsgründen nicht durch den DTAG bedient werden, ist das Verzeichnis **140** der zentrale Punkt für alle Kohärenzentscheidungen und führt als solcher eine Anzahl von Funktionen aus. Das Verzeichnis identifiziert den Besitzer eines Datenspeicherblocks. Der Besitzer kann entweder ein Prozessor oder der Speicher sein. Die Besitzerinformation aus dem Verzeichnis wird von Lesetypbefehlen (beispielsweise lesen, lesen – modifizieren) verwendet, um die Quelle der aktuellsten Version des Datenblocks zu bestimmen. Die Besitzerinformation wird außerdem verwendet, um zu bestimmen, ob die Victim-Daten zurück in den Speicher geschrieben werden sollten, wie unten ausführlicher beschrieben werden wird.

[0142] Zusätzlich zum Identifizieren der Besitzer aller Daten für alle Lesetypbefehle wird das Verzeichnis verwendet, um Clean-to-Dirty- und Shared-to-Dirty-Befehle aus dem Prozessor zu klären. Ein Clean-to-Dirty-Befehl wird durch einen Prozessor ausgegeben, wenn er eine aktuelle Cache-Zeile in ihren Clean-Status modifizieren will. Ein Shared-to-Dirty-Befehl wird ausgegeben, wenn er eine Cache-Zeile in den Dirty-Shared-Zustand modifizieren will. Die Befehle werden zu dem Home-Verteilerbus weitergeleitet, wobei das Verzeichnis bestimmt, ob der Prozessor eine aktuelle Version der Cache-Zeile hat. Falls ja, ist der Befehl erfolgreich und dem Prozessor wird ermöglicht, die Cache-Zeile zu modifizieren. Andernfalls versagt der Befehl und der Prozessor muss zuerst die aktuelle Version der Cache-Zeile erlangen. Diese Speichertyparbeitsschritte verwenden die Knotenpräsenzinformation in dem Verzeichnis, um Erfolg oder Versagen festzustellen.

[0143] Wie oben erwähnt, identifizieren die Präsenzbits des Verzeichnisses den Mehrprozessorknoten mit Kopien jedes Datenblocks, wenn die Speichertypbefehle ausgegeben werden. Speicherbefehle zeigen an, dass die Inhalte der Cache-Zeile aktualisiert werden. Durch das Prüfen des Präsenzbits **114** des zugehörigen Verzeichniseintrags, wenn ein Speicherbefehl in dem Verzeichnis **140** empfangen wird, werden die Knoten mit ihren Präsenzbits verwendet, um die Mehrprozessorknoten mit Kopien der Cache-Zeile in dem Knoten zu identifizieren, so dass die Cache-Zeilen in jedem dieser Knoten ungültig gemacht werden können.

[0144] Dementspechend arbeiten das Verzeichnis und der DTAG in Verbindung, um für jeden der Datenblöcke in dem Speicher des lokalen Multiprozessors und für jeden der in den Caches des lokalen Prozessors gespeicherten Datenblöcke Statusinformationen bereitzustellen. Das Verzeichnis in dem Home-Knoten stellt Grobinformation über den Status der Kopie eines Cache-Blocks bereit. Anschließend gehen die Befehle Invalidate zu denjenigen durch das Verzeichnis identifizierten Knoten, in denen auf den DTAG zugegriffen wird, um die Kopieinformationen weiter zu verfeinern. Infolgedessen zeigt der DTAG in diesen Knoten an, welcher Prozessor in den jeweiligen Knoten Kopien der Zeile in dem Cache speichert.

Das TTT

[0145] Das TTT wird verwendet, um die von einem Mehrprozessorknoten ausstehenden Transaktionen zu verfolgen, d. h. Referenzen, die auf Reaktionen von anderen Mehrprozessorknoten oder dem hierarchischen Switch warten. Informationen über ausstehende Transaktionen werden durch das Cache-Kohärenz-Protokoll bei der Weiterverarbeitung von anschließenden Befehlen zu in Beziehung stehenden Speicheradressen verwendet.

[0146] Im Folgenden Bezug nehmend auf die **Fig. 10**, wird ein Ausführungsbeispiel des TTTs **122** gezeigt, das ein Adressfeld **152**, ein Befehlsfeld **154**, ein Befehlsgeber-ID-Feld **156** und eine Anzahl von Statusbits **158**, einschließlich der Bits **158a**–**158c** umfasst. Das Adressfeld **152** speichert die Adresse der Cache-Zeile für eine Transaktion, die aktuell in Gang ist, während das Befehlsfeld den Befehl speichert, der der Cache-Zeile für die Transaktion, die aktuell in Gang ist, zugehörig ist. Das Befehlsgeber-ID-Feld **156** speichert die Prozessornummer des Prozessors, der den in dem Befehlsfeld gespeicherten Befehl initiiert hat. Die Statusbits **158** reflektieren den Status des Befehls, der abgewickelt wird. Alternativ können die Statusbits verwendet werden, um verschiedene Eigenschaften des Befehls, der aktuell abgewickelt wird, zu reflektieren.

[0147] Beispielsweise wird ein Füllstatus-Bit **158a** aktualisiert, wenn in Beantwortung eines Lesetypbefehls Fülldaten empfangen werden. Ein Schattenstatus-Bit **158b** wird eingerichtet, wenn der Befehl, der über den globalen Port ausgegeben wird, ein Schattentypbefehl (der unten ausführlicher beschrieben wird) ist. Das ACK-Statusbit **158c** wird eingerichtet, wenn als Reaktion eine Nachricht, die eine Beantwortung des Bestätigungstyps erwartet, empfangen wurde. Wenn die Antwort eintrifft, wird das Bit gelöscht. Es ist zu beachten, dass nicht alle der Statusbits, die in dem TTT enthalten sein können, gezeigt wurden. Stattdessen können diejenigen Statusbits enthalten sein, die eine Relevanz für spätere Beschreibungen haben. Zusätzlich ist vorgesehen, dass andere Statusbits alternativ bereitgestellt werden, wie zum Aufrechterhalten der Speicherkohärenz erforderlich, und die vorliegende Erfindung sollte nicht auf eine bestimmte Zuweisung von Bits in dem TTT beschränkt sein.

[0148] Folglich werden das Verzeichnis, der DTAG, der IOP-Tag und das TTT jeweils verwendet, um die Kohärenz der Cache-Zeilen in dem SMP-System zu erhalten (im Folgenden als Cache-Kohärenz bezeichnet). Jede dieser Komponenten ist über Schnittstellen mit dem globalen Port verbunden, um kohärente Kommunikation zwischen den Mehrprozessorknoten, die mit dem hierarchischen Switch **155** gekoppelt sind, bereitzustellen.

Serialisierungspunkte

[0149] Zusätzlich zu den obigen Elementen wird die Kohärenz der gemeinsamen Datennutzung durch das Bereitstellen eines Serialisierungspunktes in jedem Mehrprozessorknoten aufrechterhalten. In einem Ausführungsbeispiel der Erfindung ist der Serialisierungspunkt in jedem Mehrprozessorknoten der Verteilerbus **130**. Alle Q0-Referenzen, ob von einem lokalen Prozessor oder von einem Remote-Prozessor ausgegeben, werden durch den QSA zu dem Verzeichnis **140** und dem DTAG **20** auf dem Verteilerbus **130** weitergeleitet. Sobald die Referenzen auf das Verzeichnis und/oder den DTAG zugegriffen haben, werden die sich ergebenden Q1-Kanalbefehle in strikter Reihenfolge auf dem Verteilerbus ausgegeben, wobei die Reihenfolge die Serialisierungsordnung der Referenzen ist. Durch das Bereitstellen eines Serialisierungspunktes in jedem der Mehrprozessorknoten wird das Kohärenzprotokoll zur gemeinsamen Datennutzung, das in dem SMP implementiert ist, wesentlich vereinfacht.

[0150] Zusätzlich zu der Bereitstellung eines Serialisierungspunktes in jedem der Mehrprozessorknoten stellt der hierarchische Switch **155** einen zweiten Serialisierungspunkt in dem SMP-System bereit. Wie unten noch ausführlicher beschrieben wird, stimmt der hierarchische Switch mit bestimmten Ordnungsregeln überein, die sicherstellen, dass die in dem ersten Serialisierungspunkt eingeführte Kohärenz in dem großen SMP-System aufrechterhalten bleibt.

Globaler Port/Schnittstelle des hierarchischen Switchs

[0151] Im Folgenden Bezug nehmend auf die Fig. 11, ist ein Blockdiagramm des hierarchischen Switchs **155** gezeigt, der acht Eingangsports **155i0**–**155i7** und acht Ausgangsport **155o0**–**155o7** enthält. Die Eingangsports **155i0**–**155i7** empfangen Pakete von den globalen Ports jeder der gekoppelten Mehrprozessorknoten. Die Ausgangsports **155o0**–**155o7** des hierarchischen Switchs leiten Pakete zu den globalen Ports jedes der gekoppelten Mehrprozessorknoten weiter.

[0152] In einem Ausführungsbeispiel der Erfindung ist jedem Eingangsport ein Puffer **160a**–**160h** zur Pufferung der empfangenen Pakete zugehörig. Obwohl das Ausführungsbeispiel in der Fig. 11 einen Puffer für jeden Eingang darstellt, können die Puffer alternativ von jeder Anzahl von Eingangsports gemeinsam genutzt werden. Wie oben erwähnt, kann jedes der Pakete jedem von fünf Kanälen zugehörig sein. In einem Ausführungsbeispiel der Erfindung, das unten beschrieben wird, sind Teile jedes Eingangspuffers **160a**–**160h** für das Speichern von Paketen von bestimmten Kanälen zugeordnet. Dementsprechend wird die Flusssteuerung von den globalen Ports zu dem hierarchischen Switch **155** auf einer Kanalbasis durchgeführt. Durch Steuerung des Datenflusses in den Schalter auf Kanalbasis und durch das Zuordnen von Teilen der Eingangspuffer zum Auswählen einer der Kanäle, stellt der Switch die Kommunikation zwischen Mehrprozessorknoten in dem SMP-System frei von Verklemmung bereit.

[0153] Zusätzlich zu dem Bereitstellen einer von Verklemmung freien Kommunikation ist der hierarchische Switch **155** außerdem ausgelegt, um die Ordnungszwänge des SMP-Systems zu unterstützen, um die Speicherkohärenz sicherzustellen. Die Ordnungszwänge werden durch das Steuern der Reihenfolge von Paketen, die aus dem hierarchischen Switch **155** heraus zu den globalen Ports der zugehörigen Mehrprozessorknoten weitergeleitet weiden, auferlegt. Pakete von jedem der Eingangspuffer **160a**–**160h** können über die Multiplexer

182a–**182h** zu jedem der Ausgangsports weitergeleitet werden. Wie unten ausführlicher beschrieben wird, ist der Switch **155** außerdem zum Massenversenden der Pakete in der Lage. Dementsprechend können Pakete aus einem Eingangspuffer in jeder Anzahl von Ausgangsports weitergeleitet werden. Durch das Durchsetzen der Reihenfolge an den globalen Ausgangsports kann die in jedem Mehrprozessorknoten erhaltene Serialisierungsordnung aufrechterhalten werden, um in dem SMP-System eine gesamtkohärente gemeinsame Datennutzung bereitzustellen.

Vermeidung von Blockierungen in dem hierarchischen Switch

[0154] Wie oben erwähnt, leitet jeder der acht Knoten der Fig. 7A Daten zu dem hierarchischen Switch weiter und es kann der Fall eintreten, dass alle Knoten gleichzeitig Daten weiterleiten. Die Pakete werden in einer Anzahl von verschiedene Kanaltypen (Q0, Q0Vic, Q1, Q2 und QIO) aufgeteilt, die auf virtuellen Kanälen weitergeleitet wird, wobei ein virtueller Kanal im Wesentlichen ein Datenpfad ist, dem Pakete eines bestimmten Typs zugeordnet sind, der eine gemeinsame Verbindung mit anderen Kanälen zusammen nutzen kann, jedoch an beiden Enden der Verbindung unabhängig gepuffert ist. Weil zwischen dem globalen Port jeder der Knoten und dem hierarchischen Switch nur ein Datenpfad vorhanden ist, werden alle Pakete von verschiedenen virtuellen Kanälen unter Verwendung des einen Datenpfades in den hierarchischen Switch geschrieben.

[0155] Da jeder der acht Knoten **100a**–**100h** in der Lage ist, Daten zu dem hierarchischen Switch zu senden, ist eine Form der Steuerung erforderlich, um angemessen sicherzustellen, dass alle Nachrichten in dem Switch empfangen werden und in einer adäquaten Reihenfolge aus dem Switch heraus weitergeleitet werden. Zusätzlich ist eine Aufgabe der vorliegenden Erfindung, sicherzustellen, dass Pakettypen höherer Ordnung nicht durch Pakettypen niedriger Ordnung blockiert werden, um zu garantieren, dass in dem symmetrischen Mehrprozesssystem keine Verklemmung eintritt. In einem Ausführungsbeispiel der Erfindung ist die Reihenfolge der Pakete von der höchsten Ordnung zu der niedrigsten Ordnung Q2, Q1, Q0, Q0Vic und QIO.

[0156] Gemäß diesem Aspekt der Erfindung wird eine Flusssteuerung der Pakete, die an den Eingangsports des Switchs ankommen, bereitgestellt, die sicherstellt, dass die oben aufgestellte Regel zum Verhindern der Verklemmung immer erfüllt wird. Des Weiteren müssen die in dem Switch verfügbaren Puffer immer optimal ausgenutzt werden und eine maximale Bandbreite muss aufrechterhalten werden.

[0157] Gemäß einem Ausführungsbeispiel der Erfindung wird eine Steuervorrichtung zum Steuern des Schreibens von Daten in den hierarchischen Switch bereitgestellt, die durch das Bereitstellen von jedem Pakettyp zugewiesenen Schlitzen in einem Puffer des hierarchischen Switchs implementiert wird. Der Puffer enthält außerdem eine Anzahl von generischen Schlitzen, die zum Speichern von Paketen jedes Typs verwendet werden können. Durch das Bereitstellen von zugewiesenen Pufferschlitzen in dem hierarchischen Switch kann Systemblockade durch das Garantieren, dass Pakete höherer Ordnung immer einen verfügbaren Pfad durch den Switch haben, vermieden werden. Zusätzlich kann durch das Überwachen der Anzahl von generischen Schlitzen und von zugewiesenen Schlitzen, die verfügbar ist, und durch Überwachen der Anzahl von verschiedenen Pakettypen, die in dem Puffer gespeichert ist, ein einfaches Flusssteuerschema implementiert werden, um Knoten vom Schreiben in den Puffer des hierarchischen Switchs abzuhalten, wenn der Puffer seine Kapazität erreicht.

[0158] Im Folgenden Bezug nehmend auf die Fig. 12A, wird als ein Beispiel der Steuerlogik zur Verwendung beim Steuern des Schreibens durch mehrere Knotenquellen ein gemeinsamer Zieladressenpuffer bereitgestellt. In dem Blockdiagramm der Fig. 12A werden durch ein Beispiel die globalen Ports **120a** und **120b** von zwei verschiedenen Knoten gezeigt.

[0159] In der Fig. 12A werden die Teile der globalen Ports **120a** und **120b** der Knoten **100a** und **100b** jeweils detaillierter gezeigt und enthalten einen Puffer **135**, der die Einträge **135a** und **135b** zum jeweiligen Speichern von Q0/Q0Vic, Q1, Q2 und der Pakete des generischen Typs (entweder Q0-, Q0Vic-, Q1-, Q2-Pakete oder QIO-Pakete) zum Übertragen zu dem hierarchischen Switch **155** enthält. Ein Multiplexer **167a** ist mit dem Puffer **135** gekoppelt, um einen der Pakettypen zum Weiterleiten unter Verwendung eines Auswahlsignals von dem GP-Verteiler **134** über den Link zu dem hierarchischen Switch auszuwählen.

[0160] Zusätzlich enthält jeder globale Port ein zugeordnetes Zählregister **136**. Das zugeordneten Zählregister speichert eine Zählung für jeden Q0-/Q0Vic-, Q1- und Q2-Kanaltyp des Pakets der Anzahl der Pakete dieses Kanaltyps, die aktuell in dem hierarchischen Switch **155** anhängig ist. Die Zählung wird inkrementiert, wenn das Paket des jeweiligen Kanaltyps zu dem hierarchischen Switch übertragen ist, und dekrementiert, wenn das Paket aus dem hierarchischen Switch übertragen wird.

[0161] In einem Ausführungsbeispiel der Erfindung enthält der hierarchische Switch **155** einen Puffer für jede der acht Eingangsquellen. In der Fig. 12A wurden nur zwei Puffer **160a** und **160b**, die jeweils den zwei globalen Ports **120a** und **120b** entsprechen, gezeigt. In einem Ausführungsbeispiel der Erfindung sind wenigstens (m – 1) × n zugewiesene Schlitze in jedem der Puffer **160a** und **160b**, wobei m der Anzahl von virtuellen Kanaltypen entspricht, die zugeordnete Einträge in dem Puffer haben, und n der Anzahl von Knoten entspricht, die den Puffer gemeinsam nutzen. In dem Ausführungsbeispiel der Fig. 12A enthält jeder der Puffer acht Einträge. Fünf der Einträge sind generische Einträge und können jeden Pakettyp speichern, der von dem globalen Port **135** weitergeleitet wird. Jeder der restlichen drei Einträge wird dem Speichern eines bestimmten Pakettyps zugeordnet, wobei ein Eintrag zum Speichern der Q0-/Q0Vic-Pakete zugeordnet ist, ein Eintrag zum Speichern der Q1-Pakete zugeordnet ist und ein Eintrag dem Speichern von Q2-Pakettypen zugeordnet ist.

[0162] Obwohl diese zugeordneten Einträge als sich an einem feststehenden Ort in den Puffern **160a** und **160b** befindend gezeigt wurden, kann in der Realität jede Pufferstelle die zugewiesene Pufferstelle sein, d. h., dass unabhängig von der Stelle des Eintrags für jeden bestimmten Pakettyp immer ein zugeordneter Eintrag in dem Puffer vorhanden ist.

[0163] Der hierarchische Switch enthält zusätzlich für jeden Puffer **160a** und **160b** jeweils einen zugeordneten Zähler **162a** und **162b** und ein Flag-Register **163a** und **163b**. In dem Ausführungsbeispiel in der Fig. 12A enthält der zugeordnete Zähler vier Einträge, drei Einträge zum Speichern der Anzahl von Q0-/Q0Vic-, Q1- und Q2-Paketen, die aktuell in dem Puffer **160a** gespeichert sind, und einen Eintrag zum Speichern einer Zählung der Anzahl von verwendeten generischen Einträgen in dem Puffer. Das Flag-Register umfasst drei Bits, wobei jedes Bit einem der Q0-/Q0Vic-, Q1- oder Q2-Pakettypen entspricht, und zeigt an, ob der zugehörige zugeordnete Zähler null ist (d. h., ob der zugeordnete Eintrag für diesen Pakettyp verwendet wurde). Folglich sind die Werte des Flag-Registers entweder eins, anzeigend, dass wenigstens ein Paket dieses Typs in dem Puffer gespeichert ist, oder null, anzeigend, dass keine Pakete dieses Typs in dem Puffer gespeichert sind.

[0164] Zusätzlich enthält der hierarchische Switch **155** für jeden Puffer **160a** und **160b** jeweils eine Transitzählung **164a** und **164b**. Die Transitzählung führt für jede Quelle die Anzahl von ausstehenden Paketen jedes Typs, die sich während eines gegebenen Datenzyklus im Transit befinden kann.

[0165] Die Anzahl von Paketen, die sich in einem gegebenen Datenzyklus im Transit befinden kann, steht in direkter Beziehung mit der Flusssteuerungslatenz des hierarchischen Switchs und des globalen Ports. Von dem hierarchischen Switch wird ein Flusssteuersignal zu dem globalen Port weitergeleitet, um dem globalen Port zu signalisieren, das Senden von Daten zu dem hierarchischen Switch einzustellen. Die Flusssteuerungslatenz (L) wird als die Anzahl der Datenübertragungszyklen gemessen, die zwischen der Aktivierung eines Flusssteuersignals durch den hierarchischen Switch und dem Anhalten der Datensendungen durch den globalen Port anwächst.

[0166] Der hierarchische Switch enthält außerdem eine Schreibsteuerlogik **166a** und **166b** zum Steuern des Schreibens der jeweiligen Puffer **168a** und **168b**. Die Schreibsteuerlogik steuert den Datenfluss in den zugehörigen Puffern durch das Aktivieren des Flusssteuersignals auf der Leitung **168a** und der Bestätigungssignale (ACK-Signale)<3:0> auf den Leitungen **168b**. Das Flusssteuersignal und die ACK-Signale werden jeden Datenzyklus gesendet. Wie oben erwähnt, wird das Flusssteuersignal verwendet, um das Übertragen von Paketdaten durch den gekoppelten globalen Port anzuhalten. Die ACK-Signale<3:0> auf den Leitungen **168b** enthalten ein Bit für jeden der zugewiesenen Pakettypen und werden verwendet, um dem gekoppelten globalen Port zu signalisieren, dass ein Paket dieses Typs von dem zugehörigen Puffer freigegeben wurde. Die ACK-Signale werden infolgedessen von der Zählung des globalen Ports verwendet, um die Werte in dem zugeordneten Zähler **136** zu inkrementieren.

[0167] Die Schreibsteuerlogik aktiviert die Flusssteuerung, wenn festgestellt ist, dass die Gesamtmenge der verfügbaren generischen Einträge in dem Puffer nicht ausreichend groß ist, um alle der möglichen Pakete, die im Transit zu dem hierarchischen Switch sind, unterzubringen. Die Anzahl der verfügbaren generischen Schlitze kann durch die Gleichung 1 unten bestimmt werden.

Gleichung 1:

Generische Zählung = Puffergröße – # der in dem Puffer verwendeten generischen Einträge – # nicht aktivierter Flags

[0168] Sobald die Anzahl der verfügbaren generischen Einträge bestimmt wurde, wird das Flusssteuersignal

aktiviert, wenn die Gleichung 2 gilt.

Gleichung 2:

Generische Zählung >= Transitzählung · Anzahl der den Knoten verwendenden Puffer

[0169] Dementsprechend überwacht die Schreibsteuerlogik **166** die Anzahl von generischen und zugewiesenen Schlitzen, die in Verwendung sind, die Transitzählung und die Puffergesamtgröße, um festzustellen, wann ein Flusssteuersignal zu aktivieren ist.

[0170] Die Aktivierung des Flusssteuersignals stoppt nicht alle Sendungen durch einen globalen Port eines Quellenknotens. Der globale Port kann immer zugeordnete Paketdaten zu dem hierarchischen Switch übertragen, wenn der zugewiesene Schlitz, der dem zugeordneten Pakettyp entspricht, in dem Puffer des hierarchischen Switchs vertügbar ist. Infolgedessen kann der globale Port, wenn die Werte einer der zugeordneten Zählungen in dem zugeordneten Zähler gleich null sind, immer Paketdaten des entsprechenden zugeordneten Pakettyps senden.

[0171] Dementsprechend garantiert das Bereitstellen von zugewiesenen Einträgen in dem Puffer effektiv, dass der Forschritt der Pakete eines Typs durch den hierarchischen Switch nicht von dem Fortschritt eines anderen Pakets durch den hierarchischen Switch abhängig ist.

[0172] Die Verwendung von zugewiesenen und generischen Schlitzen in den Puffern **160a** und **160b** ermöglicht, dass eine Mindestanzahl von Schlitzen für jeden Pakettyp reserviert ist. Durch das Verfolgen der Anzahl von Paketen, die im Transit sind, kann die Flusssteuerung in einer präzisen Art und Weise erfolgen. Sowohl die Pufferausnutzung als auch die Bandbreite werden maximiert. Wenn beispielsweise nur generische Schlitze vertügbar sind, kann die Flusssteuerung für einen Zyklus verlassen werden und dann in dem nächsten Zyklus wieder durchgesetzt werden. Im Ergebnis können innerhalb des Zeitraums bis zu x mehr Nachrichten empfangen werden.

[0173] Im Folgenden Bezug nehmend auf die Fig. 12B, wird ein Flussdiagramm gezeigt, das das durch den globalen Port verwendete Verfahren zu Weiterleiten von Daten zu dem hierarchischen Switch darlegt. Der Prozess wird unter Bezugnahme auf einen Pakettyp beschrieben, obwohl er gleichermaßen auf andere Pakettypen anwendbar ist. In dem Schritt **169** wird in dem GS-Verteiler **134** festgestellt, ob in einem der Puffer **135a**–**135d** ein Paket vorhanden ist oder nicht, das zu dem hierarchischen Switch weiterzuleiten ist. Wenn ein Paket vorhanden ist, wird in dem Schritt **171** der Status des Flusssteuersignals durch den Verteiler **134** bewertet. Wenn das Flusssteuersignal aktiviert ist, wird in dem Schritt **172** die zugeordnete Zählung für den bestimmten Pakettyp, der zu dem hierarchischen Switch zu senden ist, geprüft, um festzustellen, ob sie gleich null ist oder nicht. Wenn die zugeordnete Zählung nicht gleich null ist, dann ist der zugeordnete Eintrag in dem Puffer für diesen Pakettyp bereits in Verwendung und der Prozess geht zu dem Schritt **170** zurück, in dem er zwischen den Schritten **169**, **171** und **172** in einer Schleife bleibt, bis die zugeordnete Zählung für diesen Pakettyp gleich null ist oder bis das Flusssteuersignal deaktiviert ist. Wenn in dem Schritt **172** festgestellt wird, dass die zugeordnete Zählung gleich null ist, dann aktiviert der GP-Verteiler **134** in dem Schritt **173** das adäquate Auswahlsignal zu dem Multiplexer **167**, um das erwünschte Paket zu dem hierarchischen Switch **155** weiterzuleiten. In dem Schritt **174** wird die zugeordnete Zählung, die dem ausgewählten Pakettyp entspricht, in dem zugeordneten Zählregister **136** in dem globalen Port und in dem zugeordneten Zählregister **162a** in dem hierarchischen Switch **155** inkrementiert und das zugehörige Flag in dem Flag-Register **163a** wird aktiviert.

[0174] Wie oben beschrieben, wird das Flag-Register **163a** zusammen mit der generischen Zählung und der Transitzählung verwendet, um den Status des Flusssteuersignals für den nächsten Datenzyklus zu bestimmen. Im Folgenden Bezug nehmend auf die Fig. 13, ist ein Ausführungsbeispiel zum Steuern der Aktivierung des Flusssteuersignals durch den hierarchischen Switch gezeigt. In dem Schritt **175** wird das Flag-Register **163a** geprüft, um die Anzahl der zugeordneten Zähleinträge, die gleich null sind, zu zählen. Wie oben erwähnt, zeigt die Anzahl von Nullen die Anzahl der potenziell zugeordneten Pakete an, die durch jeden der Knoten, die mit dem Puffer gekoppelt sind, selbst dann weitergeleitet werden könnten, wenn das Flusssteuersignal aktiviert ist.

[0175] Dementsprechend würden, wenn einer der zugewiesenen Schlitze für einen der Knoten in dem Beispiel der Fig. 11 verwendet werden würde, alle Einträge des Flag-Registers gleich null sein und infolgedessen anzeigen, dass drei Pufferstellen vorhanden sind, die für die zugewiesenen Pakete reserviert werden sollten.

[0176] Nachdem die Werte in dem Flag-Register **163a** geprüft wurden, wird in dem Schritt **176** unter Verwendung der oben dargestellten Gleichung 1 die Gesamtanzahl der verfügbaren generischen Schlitze festgestellt. Als Nächstes wird in dem Schritt **177** die Transitzählung für jeden Knoten festgestellt. Wie oben erwähnt, zeigt die Transitzählung die Anzahl von Nachrichten an, die für jeden gegebenen Datenzyklus im Transit zwischen dem globalen Port und dem hierarchischen Switch sein kann. Die Worst-case-Transitzählung ist gleich der Flusssteuerungslatenz L mal der Anzahl von Knoten, die Puffer N nutzen. Jedoch berücksichtigt die Feststellung der Transitzählung gemäß einem Ausführungsbeispiel der Erfindung, ob das Flusssteuersignal für vorhergehende Zyklen aktiviert war oder nicht. Wenn das Flussteuersignal in einem vorhergehenden Zyklus aktiviert war, sind keine Pakete im Transit zwischen dem globalen Port und dem hierarchischen Switch. Wenn die Flussteuerung beispielsweise für die vorhergehenden J Perioden null war, können bis zu J × n Nachrichten im Transit sein. Wenn jedoch das Flusssteuersignal für einen Zeitraum von J – 1 des vorhergehenden Datenzyklus null war, sind nur (J – 1) × n Nachrichten im Transit.

[0177] Infolgedessen bestimmt das Ausführungsbeispiel der Erfindung durch das Prüfen der Gesamtlatenz zwischen der Quelle (globaler Port) und der Zieladresse (hierarchischer Switch) und durch das Prüfen der Wechselbeziehung zwischen der Quelle und der Zieladresse in den vorhergehenden Datenzyklen intelligent die Anzahl von Paketen im Transit. Nachdem die Transitzählung für jeden Knoten ausgeführt ist, wird in dem Schritt **178a** unter Verwendung der oben dargestellten Gleichung 2 festgestellt, ob ausreichend verfügbare generische Einträge in dem Puffer vorhanden sind, um die ausstehenden zugeordneten Pakete und die Pakete, die im Transit sind, unterzubringen. Wenn die Gesamtanzahl der verfügbaren generischen Pakete kleiner als die Anzahl von Paketen im Transit mal der Anzahl von Knoten, die den Puffer gemeinsam nutzen, ist, dann wird in dem Schritt **178** das Flussteuersignal zu dem globalen Port **120a** aktiviert, um das Weiterleiten von Daten zu dem hierarchischen Switch **155** auszuschließen. Wenn jedoch die Gesamtzählung anzeigt, dass die Anzahl der potenziell empfangenen Pakete durch den Puffer **160a** untergebracht werden kann, dann wird das Flussteuersignal nicht aktiviert und der Prozess kehrt für den nächsten Datenzyklus zu dem Schritt **175** zurück.

[0178] Dementsprechend wird durch das Verfolgen der Anzahl von Nachrichten, die im Transit sind, und der Anzahl von vorhergehenden Zyklen, in denen das Flussteuersignal aktiviert war, die Flusssteuerung fein geregelt, um sicherzustellen, dass die Verwendung des Daten-Links, der den globalen Port mit dem hierarchischen Switch koppelt, maximiert wird.

[0179] Obwohl die Pufferschreibsteuerlogik und das Verfahren, die in den Fig. 11–Fig. 13 in Bezug auf das Übertragen von Daten von den Knoten zu dem hierarchischen Switch beschrieben wurden, sollte beachtet werden, dass die vorliegende Erfindung nicht auf ein derartiges Konstrukt beschränkt ist. Stattdessen kann ein Ausführungsbeispiel der Erfindung in jeder Umgebung verwendet weiden, in der mehrere Quellen zum Speisen eines gemeinsamen Empfängers vorhanden sind und in der Verklemmung verhindert werden muss.

Mechanismen in dem hierarchischen Switch zum Unterstützen von Kanalordnungszwängen

[0180] Das Auslesen von Daten aus dem hierarchischen Switch involviert im Wesentlichen das Weiterleiten von Daten aus einem Eingangspuffer in eine Anzahl von Ausgangsquellen, so dass sowohl die Ordnung der Pakete als auch die Datenabhängigkeiten zwischen den Paketen erhalten bleiben. Wie oben erwähnt, werden die Pakete auf einer Vielzahl von Kanälen geliefert. Den Paketen auf den verschiedenen Kanälen sind bestimmte Ordnungszwänge oder Abhängigkeiten zugehörig. In einem Ausführungsbeispiel der Erfindung ist ein Ordnungszwang, dass alle Pakete auf dem Kanal in einer Reihenfolge gehalten werden. Eine weitere Paketordnungszwangsabhängigkeit ist, dass Pakete, die auf Kanälen höherer Priorität unterwegs sind, nicht von Paketen, die auf Kanälen niedrigerer Priorität unterwegs sind, blockiert werden sollten, wobei die Priorität der Kanäle von höchster zu niedrigster Q2, Q1, Q0, Q0Vic und QIO ist. Das Aufrechterhalten der Ordnung wird unter Verwendung verschiedener Techniken, die unten beschrieben werden, durchgängig durch das SMP-System erreicht. In dem hierarchischen Switch werden drei Grundrichtlinien befolgt, um sicherzustellen, dass die Datenabhängigkeiten und die Q1-Kanalordnung eingehalten werden. Diese Grundrichtlinien sind folgende:

[0181] Richtlinie 1: Wenn mehrere Q1-Pakete an einem gegebenen Eingangsport eines hierarchischen Switchs empfangen werden, die auf einen gemeinsamen Ausgangsport zielen, erscheinen die Q1-Pakete in derselben Reihenfolge an dem Ausgangsport, in der sie an dem Eingangsport erschienen sind.

[0182] Richtlinie 2: Wenn Q1-Pakete von mehreren Eingangsports an dem hierarchischen Switch durch zu gemeinsamen Ausgangsports massenversendet werden, erscheinen die Q1-Pakete in derselben Ordnung an allen Ausgangsports, auf die sie zielen.

[0183] Richtlinie 3: Wenn geordnete Verzeichnisse von Q1-Paketen von mehreren Eingangsports des hierarchischen Switchs auf mehrere Ausgangsports zielen, erscheinen die Q1-Pakete an den Ausgangsports in einer Art und Weise, die mit einer einzelnen gemeinsamen Ordnung aller eingehenden Q1-Pakete konsistent ist. Jeder Ausgangsport kann einige oder alle der Pakete in der gemeinsamen geordneten Liste übertragen.

[0184] Zusätzlich zu dem Aufrechterhalten der Gesamtsystemordnung für den Zweck der Kohärenz ist außerdem erwünscht, die Pakete, die von dem Switch ausgegeben werden, so zu ordnen, dass die Leistung des Adress- und des Datenbusses vollständig umgesetzt wird. Im Folgenden Bezug nehmend auf die Fig. 14, wird ein exemplarisches Zeitablaufdiagramm gezeigt, das die Verwendung der Adress- und Datenbusstruktur des HS-Links **170** zeigt.

[0185] Der HS-Link **170** ist durch zwei Paare von unidirektionalen Adress- und Datenbussen mit jedem der Mehrprozessorknoten **100** gekoppelt. Der Datenbus befördert 512 Bit Datenpakete und der Adressbus befördert 80 Bit Adresspakete. Das Übertragen von Datenpaketen erfordert das Zweifache der Anzahl von Zyklen wie das Übertragen der Adresspakete. Einige Befehle, wie zum Beispiel ein Schreibbefehl, enthalten sowohl ein Adress- als auch ein Datenpaket. Beispielsweise entspricht in der Fig. 14 das Adresspaket **179a** dem Datenpaket **179d**. Wenn jeder Befehl sowohl ein Adress- als auch ein Datenpaket enthalten würde, wäre jeder zweite Adressschlitz auf dem Adressbus im Leerlauf. Jedoch enthalten viele Befehle, wie zum Beispiel ein Lesebefehl, nur Adresspakete und erfordern keinen Schlitz auf dem Datenbus zum Übertragen von Datenpaketen. Dementsprechend ist erwünscht, um die Gesamtsystemleistung zu verbessern, einen Switch zu haben, der Pakete zum Weiterleiten aus dem Bus in einer solchen Reihenfolge aussucht, dass sowohl der Datenteil als auch der Adressteil „gepackt" werden, d. h., dass eine Adresse und Daten in jedem möglichen Zeitschlitz der Adress- und Datenteile des HS-Links vorhanden sind. Wenn die Adressen und Daten auf dem HS-Link „gepackt" werden, wird der HS-Link optimal ausgenutzt.

[0186] Zum Implementieren eines hierarchischen Switchs, der in der Lage ist, Daten aus mehreren Quellen über mehrere Eingangsports simultan zu empfangen und die Daten über mehrere Ausgangsports zu mehreren Zieladressen weiterzuleiten, gleichzeitig die Datenabhängigkeiten einzuhalten, die Systemordnung aufrechtzuerhalten und die Datenübertragungsrate zu maximieren, wird eine Vielzahl von Ausführungsbeispielen bereitgestellt. Die verschiedenen Ausführungsbeispiele werden unter Bezugnahme auf die Fig. 15–Fig. 18 beschrieben.

[0187] Im Folgenden Bezug nehmend auf die Fig. 15, wird ein Ausführungsbeispiel eines Switchs, der die oben beschriebenen Ordnungszwänge implementieren kann, gezeigt. Wie in der Fig. 11 dargestellt, enthält der Switch **155** eine Vielzahl von Puffern **160a**–**160h**. Jeder der Eingangspuffer ist ein Puffer mit einem Schreibport und acht Leseports und gekoppelt, um Pakete von acht jeweiligen Eingängen zu empfangen. Der Switch enthält ebenso acht Ausgangsports, obwohl nur die Logik für einen Ausgangsport, Ausgangsport<0>, gezeigt ist. Die Logik der restlichen Ausgangsports ist gleichartig und wird aus Gründen der Übersichtlichkeit hier nicht weiter dargestellt.

[0188] In einem Ausführungsbeispiel der Erfindung enthält jeder Eintrag jedes Puffers ein Kanalfeld **185** zum Identifizieren des Kanals eines Pakets, das in dem Eintrag des Puffers gespeichert ist. Zusätzlich enthält jeder Eintrag eine Reihe von Link-Indexen **186**. Jeder Link-Index ist ein Index zu einem der Einträge in den Eingangspuffern **160a**–**160h**. Die Link-Indexe werden verwendet, um eine Kettenliste-Adressierstruktur bereitzustellen, um auf aufeinanderfolgende Pakete auf demselben Kanal aus dem Puffer **160a** in Übereinstimmung mit den Paketordnungszwängen zuzugreifen. In jedem Eintrag sind drei verknüpfte Indexe L1, L2 und L3 vorhanden, wobei jeder Link-Index den Ort der Eintragung in ein bis drei geordneten Listen kennzeichnet.

[0189] Jeder Eintrag enthält außerdem Abhängigkeits-Flags **189**. Die Abhängigkeits-Flags werden verwendet, um die Abhängigkeiten von zwei Kanälen zu markieren. Das Abhängigkeits-Flag F1 wird gesetzt, wenn das Paket in dem entsprechenden Eintrag ein Paket ist, das entweder auf einem Q1-, einem QIO-Kanal oder einem Q0Vic-Kanal unterwegs ist. Das Abhängigkeits-Flag F2 wird gesetzt, wenn das Paket in dem entsprechenden Eintrag ein Paket ist, das entweder auf einem Q0-Kanal oder einem Q0Vic-Kanal unterwegs ist. Die Abhängigkeits-Flags unterstützen eine Ordnung der Weiterverarbeitung von Paketen in der folgenden Art und Weise.

[0190] Konzeptionell werden die empfangenen Pakete in fünf geordnete Warteschlangen aufgeteilt, die eine Q2-Kanalwarteschlange, eine kombinierte Q1-/QIO-/Q0Vic-Kanalwarteschlange, eine kombinierte Q0-/Q0Vic-Kanalwarteschlange, eine Q0Vic-Kanalwarteschlange und eine QIO-Kanalwarteschlange enthalten. Infolgedessen kann ein Paket in mehr als einer Warteschlange enthalten sein. Die Kopfzeiger enthalten

einen Zeiger **187a**–**187e** für jede der Warteschlangen. Die Kopfzeiger werden verwendet, um in den Puffern **160a**–**160h** einen Index zum Identifizieren des nächsten Pakets in dem Puffer, das dieser Warteschlange entspricht, bereitzustellen. Die Kopfzeiger **187** enthalten infolgedessen einen Q2-Kopfzeiger **187a**, einen Q1-/QIO-/Q0Vic-Kopfzeiger **187b**, einen Q0-/Q0Vic-Kopfzeiger **187c**, einen Q0Vic-Kopfzeiger **187d** und einen QIO-Kopfzeiger **187e**. Wenn ein Paket zuerst in einen Eingangspuffer geschrieben wird, wird es in einer oder in mehreren der geordneten Warteschlangen platziert. Wenn es in mehr als einer geordneten Warteschlange platziert wird, dann wird ein Abhängigkeits-Flag aktiviert oder werden mehrere Abhängigkeits-Flags **189** aktiviert. Der Kanaltyp und die Abhängigkeits-Flags werden geprüft, um einen adäquaten Eintrag in dem Puffer zu wählen, um derartig auszugeben, dass die Kanalabhängigkeiten eingehalten werden.

[0191] Jeder der Einträge jedes der acht Eingangspuffer **160a**–**160h** wird zu dem Multiplexer **182** weitergeleitet. Der Multiplexer **182** wählt in Reaktion auf ein Auswahlsignal von dem Manager **180** eines der Pakete von einem der Eingangspuffer. Der Manager **180** wählt Einträge aus den 64 möglichen Leseports der Eingangspuffer **160a**–**160h** als Ausgänge für den zugehörigen Ausgangsport. Der Manager **180** wählt die Pakete so aus, dass eine Gesamtsystemordnung und die Kanalabhängigkeiten eingehalten werden.

[0192] Während auf einem der Eingangspuffer **160a**–**160h** ein Paket empfangen wird, wird der Kanaltyp in das Kanalfeld des Eintrags geschrieben und jedes diesem Eintrag zugehörige Flag wird in dem Flag-Feld **189** aktiviert. Wie oben erwähnt, sind für jeden Eintrag in den Eingangspuffer drei Link-Indexe vorhanden, von denen jeder einer der drei geordneten Warteschlangen entspricht. In einem Ausführungsbeispiel der Erfindung werden die mehreren Link-Indexe zum Massenversenden der Pakete zu drei verschiedenen Ausgangsports verwendet. Wenn ein massenversendetes Paket in dem Puffer gespeichert wird, wird es auf mehr als einer der verketteten Listen platziert, wobei die verketteten Listen jeweils den verschiedenen Ausgangsports entsprechen. Im Ergebnis kann jeder der den verschiedenen Ausgangsports zugehörige Ausgangsmanager unter Verwendung von verschiedenen verketteten Listen-Indexen auf denselben Eingangspuffereintrag zugreifen.

[0193] Wie oben erwähnt, sind die Link-Indexwerte Pufferindexwerte zum Adressieren des nächsten Pakets des entsprechenden Typs in den Puffern **160a**–**160h**. Dementsprechend wird der Link-Indexwert nicht geschrieben, bis ein nachfolgendes Paket des entsprechenden Typs in den Puffer geschrieben wird. Wenn das nachfolgende Paket in den Puffer geschrieben wird, wird die Adresse des nachfolgenden Pakets in den verketteten Index des vorhergehenden Pakets geschrieben, wodurch in dem nächsten Paket dieses Kanaltyps ein Listen-Index bereitgestellt wird. Weil jeder der Einträge zusätzlich zu dem Schreiben der Adressen in dem vorhergehenden Eintrag drei mögliche Link-Indexfelder enthält, wird ein Zwei-Bit-Feld (nicht gezeigt) zusammen mit der Adresse gespeichert, um dem Eintrag zu ermöglichen, den adäquaten der drei Link-Indexe zum Konstruieren der geordneten Liste zu identifizieren.

[0194] Der Manager **180** wählt eines der Pakete in den Puffern **160a**–**160h** zum Weiterleiten zu dem Ausgangsport in der folgenden Art und Weise. Wie oben erwähnt, speichern die Kopfzeiger den Puffer-Index, der der Oberseite jeder der Warteschlangen entspricht. Wenn Pakete für einen gegebenen Kanal weiterverarbeitet werden, wählt der Manager den Eintrag, der durch den entsprechenden Kopfzeiger angezeigt wird. Wenn ein Flag **189** gesetzt ist oder mehrere Flags **189** gesetzt sind und Pakete in dieser Warteschlange, die den Kanälen mit höherer Priorität zugehörig ist, nicht weiterverarbeitet wurden, kann das Paket nicht weiterverarbeitet werden, bis alle vorhergehenden Pakete in der Warteschlange, die eine höhere Priorität haben, weiterverarbeitet worden sind.

[0195] Wenn der Ausgangsmanager beispielsweise Pakete des Q0-Typs weiterverarbeitet, prüft er die Einträge, die durch die Q1-, QIO-, Q0Vic- und Q0-/Q0Vic-Kopfzeiger angezeigt werden. Wenn das Paket ein Q0-Kanalpaket ist, jedoch das Verarbeiten von Q1-Paketen noch nicht ausgeführt ist, kann der Eintrag nicht weiterverarbeitet werden. Die Weiterverarbeitung von Paketen kann durch das Bereitstellen von Verarbeitungs-Flags mit jedem der Flags F1 und F2 angezeigt werden, die anzeigen, dass entweder die Kanal-Q1-Pakete oder die Kanal-Q0-Pakete bereits weiterverarbeitet wurden. Sobald die Weiterverarbeitung aller Pakete in der Warteschlange, die die Kanäle höherer Priorität hat, eingetreten ist (wie durch die Verarbeitungs-Flags angezeigt), ist das dem Eintrag zugehörige Paket frei zur Weiterverarbeitung.

[0196] Wenn ein Eintrag zu Weiterverarbeitung gewählt wird, wählt der Manager den Kopfzeiger, der mit der Warteschlange, in der der Eintrag ist, assoziiert ist, als den Puffer-Index. Der Puffer-Index wird zu dem Multiplexer **182** weitergeleitet und der Puffereintrag wird an den Ausgangsport weitergeleitet. Die Link-Indexe werden zurück zu dem Kopfzeiger geleitet und der Kopfzeiger wird mit dem Puffer-Listen-Index des nächsten Pakets in dieser Warteschlange aktualisiert.

[0197] Dementsprechend verwendet das Switch-Ausführungsbeispiel der Fig. 15 eine verkettete Verzeichnisdatenstruktur, geordnete Warteschlangen und Flags zum Bereitstellen der Pakete zu einem Ausgangsport, so dass die Gesamtsystemordnung erhalten bleibt. Zusätzlich stellt die verkettete Verzeichnisdatenstruktur, die die mehrfachen Link-Indexe enthält, einen einfachen Mechanismus zum Massenversenden der Pakete bereit, während die Massenversendungs-Paketordnungsregeln eingehalten werden.

[0198] Das Ausführungsbeispiel der Fig. 15 verwendet Flags und geordnete Warteschlangen, um sicherzustellen, dass die Kanalordnung erhalten bleibt. Im Folgenden Bezug auf die Fig. 16 nehmend, wird ein zweites Ausführungsbeispiel eines Switchs gezeigt, der in der Lage ist, Ausgangsdaten entsprechend vorgegebenen Ordnungsabhängigkeiten bereitzustellen. In dem Ausführungsbeispiel der Fig. 16 ist für jeden Ausgangsport des Switchs ein Puffer **200** bereitgestellt. Der Puffer **200** kann gekoppelt sein, um Eingänge von jedem der Puffer **160a**–**160h** (Fig. 11) auf einem Eingangspaket-Empfangspfad **201** zu empfangen, wobei die Pakete von den Eingangspuffern zu dem adäquaten Puffer des Ausgangsports in Abhängigkeit von der Zieladresse der Pakete weitergeleitet werden. In einem Ausführungsbeispiel der Erfindung ist der Puffer als ein kollabierender FIFO implementiert, obwohl andere Pufferungsarchitekturen, die dem Fachmann in dieser Technik bekannt sind, verwendet werden können.

[0199] Der Puffer **220** wird als eine Vielzahl von Paketen speichernd gezeigt, die zu dem Switch weiterzuleiten sind. Der Puffer **200** in dieser Beschreibung speichert Pakete, die auf fünf verschiedenen Kanälen zu übertragen sind: Q0, Q1, Q2, Q3 und Q4. Es sollte beachtet werden, dass die Kanäle Q0–Q4 den zuvor beschriebenen Kanälen Q0, Q1, Q2, Q0Vic und QIO nicht analog sind. Stattdessen werden sie lediglich für den Zweck der Beschreibung des Ausgabevorgangs des Switchs verwendet. Die Pakete Q0–Q4 stellen infolgedessen generische Pakete auf verschiedenen Kanälen dar, wobei die Kanalabhängigkeiten entsprechend den Pfeilen in dem Flussdiagramm der Fig. 16A definiert sind. In dem Diagramm der Fig. 16A zeigt ein Pfeil, der von einem Kanal auf einen anderen Kanal gerichtet ist, dass die Pakete in dem ersten Kanal nicht zu einem Ausgangsport weitergeleitet werden dürfen, während ein Paket in dem zweiten Kanal ist, das vor dem Paket in dem ersten Kanal empfangen wurde, und zur Weiterverarbeitung durch den Switch anhängig ist. Beispielsweise werden in der Fig. 16A die Pakete in dem Kanal Q0 gezeigt, um abhängig von der Weiterverarbeitung der Pakete in dem Kanal Q3 zu sein, und folglich ist ausgesagt, dass die Pakete in dem Kanal Q0 Pakete in den Kanal Q3 „schoben". Die zusätzlichen durch das Flussdiagramm der Fig. 16A dargestellten Abhängigkeiten zeigen an, dass die Pakete in Kanal Q1 Pakete in die Kanäle Q2 und Q3 schoben. Wieder sollte beachtet werden, dass die durch das Flussdiagramm der Fig. 16A dargestellten Abhängigkeiten nicht die Abhängigkeiten der zuvor beschriebenen Q0-, Q1-, Q2-, Q0Vic- und QIO-Kanäle darstellen. Wie hierin später beschrieben wird, sind die Abhängigkeiten der Pakete in den Q0-, Q1-, Q2-, Q0Vic- und QIO-Kanälen komplex und deshalb wurden für ein einfacheres Erklären des Betriebs des Puffers **200** die generischen Pakete und Abhängigkeiten bereitgestellt.

[0200] Wie oben erwähnt, werden die Eingangspakete an jedem der Eingangspuffer **160a**–**160h** des Switchs in Reihenfolge empfangen und in der Reihenfolge, die von der durch das Paket angezeigten Zieladresse abhängig ist, zu den Ausgangspuffern, wie dem Puffer **200**, weitergeleitet. Jeder Eintrag in jedem Ausgangspuffer, wie zum Beispiel der Eintrag **200a**, enthält ein Quellen- und ein Zieladressfeld, die die sendenden und empfangenden Knoten für das Paket anzeigen, ein Kanalfeld, das den Kanal, auf dem das Paket übertragen wird, anzeigt, und eine Reihe von Bits **206a**–**206e**. Die Bitreihe **206a**–**206e** enthält ein Bit für jeden Kanal, der Pakete durch den hierarchischen Switch weiterleitet. Beispielsweise enthält die Bitreihe in dem Ausführungsbeispiel der Fig. 16 ein Bit für jeden Kanal Q0, Q1, Q2, Q3 und Q4.

[0201] Die Schreiblogik **205**, die mit dem Eingangspaket-Empfangspfad für den Ausgangsport gekoppelt ist, steuert die Einstellung jeder Bitreihe gemäß dem Kanal des empfangenen Pakets und gemäß den Abhängigkeiten zwischen den Kanälen, die in dem Flussabhängigkeitsdiagramm der Fig. 16A gezeigt werden. Wie unten detaillierter beschrieben, kann die Schreibsteuerlogik die Bits ebenso durch das entweder statische oder dynamische Erkennen der Abhängigkeiten aktualisieren. Wenn die Abhängigkeiten statisch erkannt werden, werden die für die Kanäle definierten Abhängigkeiten ungeachtet weiterer Pakete, die in dem Puffer sind, angewendet. Wenn die Abhängigkeiten dynamisch erkannt werden, werden die Abhängigkeiten für die Kanäle unter Berücksichtigung der Kanal- und Adressziele der weiteren Pakete in dem Puffer **200** angewendet.

[0202] Mit jeder der Reihen von Bits ist eine entsprechende Suchmaschine **208a**–**208e** gekoppelt. Jede Suchmaschine sucht die zugehörige Spalte von Bits, um einen Eintrag in dem Puffer **200**, der das entsprechende Bit der Spaltengruppe hat, auszuwählen. Der ausgewählte Eintrag wird für jede Spalte (oder jeden Kanal) durch eine Reihe von Signalen S4–S0 zu einem Ausgangspuffer-Manager **202** angezeigt. Unter Verwendung der Auswahlsignale, die von jeder der Suchmaschinen empfangen werden, in Verbindung mit den bekannten

Datenabhängigkeiten zwischen den Kanälen, wählt der Ausgangspuffer-Manager eines der Pakete aus dem Ausgangspuffer **200**, um dieses zu dem Ausgang des globalen Ports bereitzustellen.

[0203] In Betrieb, wenn auf dem Eingangspaket-Empfangspfad **201** ein Paket empfangen wird, wird der Kanal durch die Schreibsteuerlogik **205** bewertet und das Bit in der Reihe von Bits **206a**–**206e**, das diesem Kanal entspricht, wird durchgesetzt. In der Fig. 15 wird das Bit, dass eingestellt wird, um den Typ des Pakets anzuzeigen durch ein „⊗" angezeigt und wird als Kanalidentifizierungs-Flag bezeichnet. Demgemäß ist in der Fig. 16 das Paket1 ein Q3-Paket. Gemäß dem Ausführungsbeispiel der Fig. 15 wird zusätzlich zum Aktivieren des Bits, das den Kanal des Eintrags anzeigt, zusätzlich für jeden der Kanäle, die das Paket auf diesem Kanal schieben, ein Bit aktiviert. Jedes dieser Bits wird als ein Abhängigkeits-Flag bezeichnet und durch ein „x" in der Fig. 16 angezeigt. Deshalb wird für Paket2, das ein Q0-Kanal-Paket ist, das dem Q3-Kanalpaket zugehörige Bit zusätzlich aktiviert, da, wie in dem Flussdiagramm der Fig. 16A gezeigt, die Q0-Pakete die Q3 Pakete schieben.

[0204] Während die Pakete in dem Puffer **200** gespeichert werden und die ihnen zugehörige Reihe von Bits **206a**–**206e** aktiviert wird, wählt jede der Suchmaschinen **208a**–**208e**, die mit jeder Spalte von Bits assoziiert ist, den ersten Eintrag in dem Puffer, der ein Bitset aufweist. Deshalb würde der für die Suchmaschine **208a** ausgewählte Wert auf Paket2 zeigen, der für die Suchmaschine **208b** ausgewählte Wert würde auf Paket3 zeigen usw.

[0205] Die Signale S0–S4 werden zu dem Manager weitergeleitet. Der Manager **202** wählt in Reaktion auf die Aktivierung der Auswahlsignale durch die Suchmaschinen und zusätzlich zu den in dem System vorhandenen Abhängigkeiten eines der Pakete aus. Gemäß einem Ausführungsbeispiel der Erfindung wird beispielsweise ein solches Paket wie Paket2, das auf dem Kanal Q0 ist, nicht zu dem Switch weitergeleitet, bis sowohl die Suchmaschine für Kanal Q0 (**208a**) als auch die Suchmaschine für Kanal Q3 (**208d**) dasselbe Paket auswählen. Dementsprechend wählt der Manager **202**, wann immer mehrere Flags für ein gegebenes Paket gesetzt sind, dieses Paket nicht für die Ausgabe aus, bis die den Flags entsprechenden Suchmaschinen eingerichtet sind, um beide das gegebene Paket auszuwählen.

[0206] Gemäß einem alterriativen Ausführungsbeispiel der Erfindung könnte die Suchmaschine, wenn die Suchmaschine einen Eintrag auswählt, weil ihr Abhängigkeits-Flag gesetzt war, das Abhängigkeits-Flag löschen, und den Puffer weiter durchsuchen, um den nächsten Eintrag zu wählen, in dem entweder das Abhängigkeits-Flag oder das Identitäts-Flag gesetzt ist. Mit einer derartigen Anordnung wird die Weiterverarbeitung der Pakete verbessert, weil die Suchmaschinen nicht abgewürgt werden, wenn sie zur Weiterverarbeitung durch andere Kanäle anstehen.

[0207] Der Effekt des Aktivierens der mehreren Flags zum Kennzeichnen der Abhängigkeiten unterstützt die Aufrechterhaltung einer Gesamtsystemordnung von Paketen, die durch den Switch propagieren. Beispielsweise ist in der Fig. 16 die Beziehung zwischen den Q0-Paketen und den Q3-Paketen so, dass die Q0-Pakete jedes vorhergehende Q3-Kanalpaket vor der Ausführung schieben. Folglich sollte ein Q0-Kanalpaket, das nach einem Q3-Kanalpaket empfangen wurde, nicht vor dem Q3-Paket ausgeführt werden. Paket1 ist ein Q3-Kanalpaket, das vor dem Paket2 (Q0-Kanalpaket) empfangen wurde. Durch Einstellen des Bits **206d** für Paket2 kann sichergestellt werden, dass das Paket2 (Q0-Kanalpaket,) nicht vor dem Paket1 (Q3-Kanalpaket) über den Ausgangsport ausgegeben wird, da der Manager **208** das Q0-Paket solange nicht auswählen wird, bis das S3- als auch das S0-Signal Paket2 auswählen. Im Ergebnis wird durch das Aktivieren von Bits für jedes Paket, das durch ein Paket auf einen gegebenen Kanal geschoben wird, der Kanal effektiv blockiert, bis die Pakete, die durch den gegebenen Kanal geschoben werden, weiterverarbeitet sind. Im Ergebnis wird die Gesamtsystemordnung aufrechterhalten.

[0208] Wie oben erwähnt, kann die Puffersteuerlogik der Fig. 16 betrieben werden, um die Abhängigkeiten entweder statisch oder dynamisch zu erkennen. Statische Abhängigkeiten sind jene Abhängigkeiten, die von dem Flussdiagramm der Fig. 16A gezeigt werden. Dynamische Abhängigkeiten werden durch das Bewerten der Inhalte der Puffer, um zu bestimmen, ob aktuell zwischen Paketen in dem Puffer statische Abhängigkeit vorhanden ist, erkannt. Die statischen Abhängigkeiten werden verwendet, um Ordnungsregeln bereitzustellen, die sicherstellen, dass die Speicherdaten in dem SMP nicht die Kohärenz verlieren. Jedoch ist die Datenkohärenz nur dann betroffen, wenn die Pakete auf denselben Block von Speicherdaten zugreifen. Deshalb untersuchen die dynamischen Abhängigkeiten die Inhalte des Puffers mit größerer Detailtiefe durch das Prüfen der Adresse der Pakete, die bereits in dem Puffer sind, um festzustellen, ob bereits eine Abhängigkeit zwischen zwei Paketen auf verschiedenen Kanälen besteht oder nicht.

[0209] Ein Vorteil des dynamischen Erkennens der Abhängigkeiten zwischen Paketen in dem Puffer **200** ist, dass dies die Zeit, die erforderlich ist, um die Pakete in dem Puffer weiterzuverarbeiten, verringert. Als ein Beispiel die Beschreibung des Betriebs mit Paket1 und Paket1 oben verwendend, besteht, wenn das Q0-Paket2 und das Q3-Paket1 nicht in dieselbe Adresse abbilden, kein Problem zuzulassen, dass das Q0-Paket vor dem Q3-Paket weiterverarbeitet wird. Die Verzögerungszeit, die beim Warten auf die Weiterverarbeitung des vorhergehenden Q3-Pakets anfällt, ist eliminiert, wodurch die Gesamtleistung des SMP-Systems verbessert wird.

[0210] Im Folgenden Bezug nehmend auf die Fig. 17, stellt ein Flussdiagramm exemplarisch den Arbeitsvorgang der Auswahl eines Pakets zur Weiterverarbeitung durch das Erkennen der dynamischen Abhängigkeiten dar. In dem Schritt **220** wird ein Paket an dem Puffer **200** empfangen. In dem Schritt **222** wird das Bit für den Kanal des Pakets durch die Schreibsteuerlogik **205** in die Reihe der Bits **206** gesetzt. In dem Schritt **224** werden die vorhergehend in dem Puffer **200** gespeicherten Pakete geprüft, um zu bestimmen, ab andere Pakete auf dem Kanal, der das Paket schiebt, an demselben Speicherblock sind. Wenn sie an demselben Speicherblock sind, dann werden in dem Schritt **226** die Bits, die den Paketen auf dem Kanal, der das Paket schiebt, entsprechen und die sich in demselben Speicherblock befinden, aktiviert. Dementsprechend, das Beispiel der Fig. 16 für Paket2 verwendend, wird das Bit für Pakettyp Q3 nur aktiviert, wenn das Paket1 auf denselben Speicherblock wie Paket2 zugreift. Dementsprechend kann durch das dynamische Erkennen der Abhängigkeiten die Speicherkohärenz aufrechterhalten werden, während die Gesamtsystemleistung verbessert wird.

[0211] Im Folgenden Bezug nehmend auf die Fig. 18, wird ein weiteres Ausführungsbeispiel eines Verfahrens zum Ausgeben von Daten, die von mehreren Eingangsquellen empfangen wurden, zu mehreren Ausgangsquellen unter Aufrechterhaltung der Gesamtsystemordnung gezeigt. Das Ausführungsbeispiel der Fig. 18 enthält dieselben Elemente, wie jene der Fig. 16. Jedoch aktualisiert die Schreibsteuerlogik der **Fig.** 209 der Fig. 18 jede Bitreihe **206a**–**206e** durch das Analysieren der Abhängigkeiten von Paketen auf eine unterschiedliche Art und Weise. Wie in der Fig. 16 wird für jedes Paket eines der Reihe von Bits gesetzt, um anzuzeigen, dass das Paket von dem zugehörigen Kanal ist. Anstatt jedoch zusätzliche Bits für alle Pakete der Kanäle, die der Kanal schiebt, einzurichten, werden die Bits für die Pakete in dem Kanal, der Pakete dieses Kanals schiebt, gesetzt.

[0212] Dementsprechend werden in dem Ausführungsbeispiel der Fig. 18 zusätzlich zum Setzen des Kanalidentifikations-Flags zusätzliche Bits für alle Kanäle, die durch das Paket maskiert oder blockiert sind, gesetzt. Beispielsweise ist in dem Beispiel der Fig. 18 Paket1 ein Q3-Kanalpaket. Pakete auf dem Kanal Q3 blockieren die Ausführung von Q1- und Q0-Paketen, bis das Q3-Paket ausgeführt ist, wie durch das Abhängigkeitsflussdiagramm der Fig. 18A gezeigt. Dementsprechend werden für das Paket1 die Bits **206d**, **206b** und **206a** gesetzt. Das Paket2 ist jedoch ein Q0-Paket, das die Ausführung eines anderen Pakets nicht blockiert. Im Ergebnis wird für das Paket2 nur das Bit **206** gesetzt.

[0213] Die Switch-Implementierung der Fig. 18 stellt infolgedessen ein abgeändertes Verfahren zum Weiterleiten von Daten zu einem Ausgangsport bereit, während die Systemordnung durch statisches Erkennen von Abhängigkeiten aufrechterhalten bleibt. Es sollte beachtet werden, dass die Pufferimplementierung der Fig. 18 nicht verwendet werden kann, um dynamische Abhängigkeiten zu erkennen, da ein solches Vorgehen das Kennen der Adressen von Daten, bevor diese Daten in den Puffer **200** geschrieben werden, erfordern würde. Jedoch können alle beschriebenen statischen und dynamischen Methoden verwendet werden, um sicherzustellen, dass die Abhängigkeiten zwischen Paketen eingehalten werden.

[0214] Demnach wurden drei Ausführungsbeispiele eines Switchs beschrieben, der in der Lage ist, Daten aus mehreren Quellen über mehrere Eingangsports simultan zu empfangen und die Daten über mehrere Ausgangsports zu mehreren Zieladressen weiterzuleiten, gleichzeitig die Datenabhängigkeiten einzuhalten, die Systemordnung aufrechtzuerhalten und die Datenübertragungsrate zu maximieren. In einem Ausführungsbeispiel wurde ein Pufferschema mit verketteter Liste beschrieben, in dem die Ordnungsabhängigkeiten durch die Verwendung von mehreren Warteschlangen, die Flags speichern, untergebracht werden und in dem die Warteschlangen ausgewählt werden, um die Abhängigkeiten zu identifizieren. In einem zweiten und in einem dritten Ausführungsbeispiel enthält ein Ausgangspuffer, der Daten in Reihenfolge von einem Eingangspuffer des Switchs empfängt, eine Bitreihe, die verwendet wird, um Pakete eines bestimmten Typs zu blockieren, um sicherzustellen, dass die Abhängigkeits- und Kohärenzzwänge durchgesetzt werden. In allen Ausführungsbeispielen werden die Ordnungsabhängigkeiten durch die Verwendung von geordneten Warteschlangen, die Flags enthalten, die gesetzt werden, um potenzielle Abhängigkeitskonflikte zu markieren, verfolgt. Durch das Verwenden einer geordneten Liste von Flags zum Identifizieren der Abhängigkeiten wird die Komplexität der Operationen, die durch den Manager durchzuführen sind, um die Ordnung aufrechtzuerhalten und die Kohärenz sicherzustellen, vereinfacht, während gleichzeitig die Busausnutzung maximiert wird.

[0215] Das Cache-Kohärenz-Protokoll eines Ausführungsbeispiels der Erfindung ist ein Write-Invalidate-Protokoll, das auf Besitz basiert. „Write-Invaldate" impliziert, dass, wenn ein Prozessor eine Cache-Zeile modifiziert, er alte Kopien in den Caches anderer Prozessoren annulliert, anstatt diese mit neuen Werten zu aktualisieren. Das Protokoll wird als ein „Besitzprotokoll" bezeichnet, weil für jede Cache-Zeile immer, ob es der Speicher oder einer der Prozessoren oder IOPs in dem System ist, ein identifizierbarer Besitzer vorhanden ist. Der Besitzer der Cache-Zeile ist, wenn angefordert, für das Liefern des aktuellen Wertes der Cache-Zeile verantwortlich. Ein Prozessor/ein IOP können eine Cache-Zeile „exklusiv" oder gemeinsam besitzen. Wenn ein Prozessor exklusiver Besitzer einer Cache-Zeile ist, kann er diese aktualisieren, ohne das System zu informieren. Andernfalls muss er das System informieren und potenziell die Kopien in den Caches anderer Prozessoren/IOPs annullieren.

[0216] Vor einer ausführlichen Beschreibung wird das Cache-Kohärenz-Protokoll beschrieben und eine Einführung in den gesamten Kommunikationsablauf, der in dem hierarchischen Netzwerk verwendet wird, wird bereitgestellt.

[0217] Wie in Bezug auf die Fig. 7A beschrieben, enthält das große SMP-System **150** eine Anzahl von Knoten, die über einen Switch **155** gekoppelt sind. Jeder der Prozessoren in jedem der Knoten generiert Befehle, um auf Daten in einem Speicher zuzugreifen. Die Befehle können zur Gänze innerhalb des Quellenknotens behandelt werden oder können basierend auf der Adresse und dem Anforderungstyp zu anderen Knoten in dem System übertragen werden.

[0218] Der Adressraum ist in den Speicherraum und den IO-Raum unterteilt. Die Prozessoren und das IOP verwenden eigene Caches, um nur Daten für Speicherraumadressen zu speichern und IO-Raumdaten werden in den eigenen Caches nicht zwischengespeichert. Infolgedessen ist das Cache-Kohärenz-Protokoll nur von den Speicherraumbefehlen betroffen.

[0219] Eine Schlüsselkomponente jedes Cache-Kohärenz-Protokolls ist ein Ansatz zur Serialisierung von Lasten und Speicherungen. Ein Cache-Kohärenz-Protokoll muss allen Lasten eine Ordnung auferlegen und in jeder Speicheradresse X speichern. Die Ordnung ist derartig, dass alle „Speicherungen" in X geordnet sind, es sollten eine erste Speicherung, eine zweite Speicherung, eine dritte Speicherung usw. vorhanden sein. Die i-te Speicherung aktualisiert die Cache-Zeile wie durch die (I – 1)-te Speicherung bestimmt. Des Weiteren ist mit jeder Last eine aktuellste Speicherung assoziiert, aus der die Last den Wert der Cache-Zeile bekommt. Im Folgenden wird dies als Lastspeicherungs-Serialisierungsordnung bezeichnet.

[0220] Es ist eine Eigenschaft des hierin beschriebenen Protokolls, dass der Verteilerbus für eine Adresse X der „Serialisierungspunkt" für alle Lasten ist und X speichert. Das bedeutet, dass die Reihenfolge, in der Anforderungen zu X an dem Home-Verteilerbus für X ankommen, die Reihenfolge ist, in der die entsprechenden Lasten und Speicherungen serialisiert werden. Die meisten Protokolle für große SMP-Systeme nach dem Stand der Technik weisen diese Eigenschaft nicht auf und sind infolgedessen weniger effizient und komplexer.

[0221] In dem in der Fig. 2 gezeigten kleinen SMP-Knotensystem ist ein Verteilerbus vorhanden. Dieser Verteilerbus ist der Serialisierungspunkt für alle Speicherlasten und Speicherungen in dem kleinen SMP. Der mit dem Verteilerbus gekoppelte DTAG erfasst alle Zustände, die durch das kleine SMP-Protokoll erforderlich werden. In dem großen SMP-System erfasst das DIR auf dem Home-Verteilerbus den Grobzustand für das Protokoll und die TTTs und DTAGs erfassen die Zustandsinformationen auf einer detailtieferen Ebene.

[0222] Wenn an dem Home-Verteilerbus eine Anforderung R ankommt, werden der DIR-, der DTAG- und der TTT-Zustand geprüft, Probe-Befehle zu anderen Prozessoren und/oder Antwortbefehle zu dem Quellenprozessor können generiert werden. Des Weiteren werden die Zustände des DIRs, des DTAGs und des TTTs atomar aktualisiert, um die „Serialisierung" der Anforderung R zu reflektieren. infolgedessen wird eine Anforderung Q mit der angeforderten Adresse, die der des Rs gleich ist und die an dem Home-Verteilerbus nach der Anforderung R eingeht, in dem hierarchischen Switch nach R erscheinen.

[0223] Demnach ist der Home-Verteilerbus definiert, um der „Serialisierungspunkt" für alle Anforderungen zu einer Speicheradresse zu sein. Für jede Speicheradresse X werden Speicherungen erscheinen, die in der Reihenfolge ausgeführt werden müssen, in der die entsprechenden Anforderungen (RdMods oder CDTs) in dem Home-Verteilerbus ankommen. Die Lasten zu der Adresse X werden die Version X, die der Speicherung X entspricht, die zuletzt von dem Home-Verteilerbus serialisiert wurde, sein.

[0224] In der folgenden Einführung in das Cache-Kohärenz-Protokoll bezeichnet der Ausdruck „System" alle Komponenten des großen SMP-Systems, einschließlich der Prozessoren und der IOPs. Die Prozessoren und das System interagieren durch das Senden von „Befehlspaketen" oder einfach von „Befehlen". Befehle können in drei Typen unterteilt werden:
Anforderungen, Probes und Antworten.

[0225] Die Befehle, die von dem Prozessor zu dem System ausgegeben werden, und jene, die von dem System zu den Prozessoren ausgegeben werden, sind eine Funktion der Speichersystemschnittstelle des gegebenen Prozessors. Für den Zweck der Beschreibung des Betriebs des SMPs werden Anforderungen und Befehle gemäß der Alpha®-System-Schnittstellendefinition der Digital Equipment Corporation beschrieben, obwohl selbstverständlich ebenso andere Typen von Prozessoren verwendet werden können.

[0226] Anforderungen sind Befehle, die von einem Prozessor ausgegeben werden, wenn er im Ergebnis des Ausführens eines Lade- oder Speichervorgangs eine Kopie von Daten erhalten muss. Anforderungen werden außerdem verwendet, um den exklusiven Besitz eines Datenelements aus dem System zu gewinnen. Anforderungen enthalten Lesebefehle, Lese-/Modifizierbefehle (RdMod-Befehle), Change-to-Dirty-Befehle, Victim-Befehle und Evict-Befehle (bei denen eine Cache-Zei1e aus dem jeweiligen Cache entfernt wird). Probe-Befehle sind Befehle, die von dem System zu einem Prozessor oder zu mehreren Prozessoren ausgegeben werden, die Daten- und/oder Cache-Tag-Status Aktualisierungen anfordern. Probe-Befehle enthalten weitergeleitete Lesebefehle (FRd-Befehle), weitergeleitete Lese-/Modifizierbefehle (FRdMod-Befehle) und Befehle Invalidate. Wenn ein Prozessor P eine Anforderung zu dem System ausgibt, könnte das System einen Probe oder mehr Probes zu anderen Prozessoren auszugeben haben. Wenn P eine Kopie einer Cache-Zeile (mit einer Leseanforderung) anfordert, wird das System einen Probe zu dem Besitzerprozessor (falls vorhanden) senden. Wenn P einen exklusiven Besitz einer Cache-Zeile anfordert (mit einer CDT-Anforderung) sendet das System Invalidate-Probes zu einem Prozessor oder zu mehreren Prozessoren mit Kopien der Cache-Zeile. Wenn P sowohl eine Kopie der Cache-Zeile als auch den exklusiven Besitz der Cache-Zeile anfordert (mit einer RdMod-Anforderung), sendet das System einen FRd-Befehl zu einem Prozessor, der aktuell eine Dirty-Kopie der Cache-Zeile von Daten speichert. Als Antwort auf den FRd-Befehl wird die Dirty-Kopie der Cache-Zeile an das System zurückgesendet. Ebenso wird durch das System ein FRdMod-Befehl zu einem Prozessor, der eine Dirty-Kopie einer Cache-Zeile speichert, ausgegeben. Als Antwort auf den FRdMod-Befehl wird die dirty Cache-Zeile zu dem System zurückgesendet und die in dem Cache gespeicherte Dirty-Kopie wird annulliert. Durch das System kann ein Befehl Invalidate zu einem Prozessor, der eine Kopie der Cache-Zeile in seinem Cache speichert, ausgegeben werden, wenn die Cache-Zeile durch einen anderen Prozessor zu aktualisieren ist.

[0227] Antworten sind Befehle von dem System zu den Prozessoren/IOPs, die die durch den Prozessor angeforderte Daten oder eine der Anforderung entsprechende Bestätigung tragen. Für Lese- und RdMod-Befehle, ist die Antwort jeweils ein Füll- oder FillMod-Befehl, von denen jeder die angeforderten Daten trägt. Für die CDT-Befehle ist die Antwort ein CDT-Success- oder ein CDT-Failure-Befehl, der den Erfolg oder das Versagen des CDTs anzeigt. Für Victim-Befehle ist die Antwort ein Victim-Release-Befehl.

[0228] Im Folgenden Bezug nehmend auf die Fig. 19, wird eine Tabelle zum Darstellen der Beziehung zwischen Anforderungen und dem Status der jeweiligen Cache-Zeile in den einzelnen Prozessoren gezeigt.

[0229] Die Fig. 19 stellt außerdem die sich ergebenden Befehle des Probe-Typs für jede der Anforderungen und jeden der Zustände der Cache-Zeile dar. Die Spalten **300** und **300a** zeigen die durch den Prozessor ausgegebenen Anforderungen, die Spalten **305** und **305a** zeigen den Status der Cache-Zeile in anderen Prozessoren in dem System und die Spalten **320** und **320a** zeigen die sich ergebenden Probe-Befehle, die durch das System generiert werden.

[0230] Die Tabelle der Fig. 19 geht davon aus, dass ein Prozessor, der als Prozessor A bezeichnet wird, eine Anforderung an das System ausgibt. Der Befehl des Prozessors A interagiert anschließend mit einem Prozessor oder mit mehreren Prozessoren, als Prozessor B bezeichnet. Wenn eine durch den Prozessor A adressierte Cache-Zeile in dem Cache des Prozessors B gespeichert ist, wie unter Verwendung des DTAGs und/oder der Verzeichnisinformation festgestellt, dann wird der Cache-Status des Prozessors B bestimmen, ob ein Probe-Befehl zu dem Prozessor B ausgegeben werden muss und welcher Typ von Probe-Befehl ausgegeben werden sollte.

[0231] Im Folgenden werden das Cache-Kohärenz-Protokoll und die Mechanismen detaillierter beschrieben. Die von den Befehlspaketen genommenen Pfade, die Quellen der Statusinformation für jeden Befehlstyp und

die sich ergebenden Operationen sind in der Beschreibung enthalten. Alle Befehle stammen entweder von einem Prozessor oder einem IOP, wobei der ausgebende Prozessor des IOPs als „Quellenprozessor" bezeichnet wird. Die Adresse, die in der Anforderung enthalten ist, wird als die angeforderte Adresse bezeichnet. Der „Home-Knoten" der Adresse ist der Knoten, dessen Adressraum die angeforderte Adresse abbildet. Die Anforderung wird als „lokal" bezeichnet, wenn der Quellenprozessor in einem der Home-Knoten der angeforderten Adresse ist, andernfalls wird sie als „globale" Anforderung bezeichnet. Der Verteilerbus auf dem Home-Knoten wird als der Home-Verteilerbus bezeichnet. Das „Home-Verzeichnis" ist das der angeforderten Adresse entsprechende Verzeichnis. Das Home-Verzeichnis und der Speicher sind folglich mit dem Home-Verteilerbus für die angeforderte Adresse gekoppelt.

[0232] Eine aus einem Prozessor oder einem IOP entstammende Speicheranforderung wird zuerst auf den Home-Verteilerbus geleitet. Die Anforderung wird über den lokalen Switch geleitet, wenn die Anforderung lokal ist, sie geht über den hierarchischen Switch, wenn sie global ist. In dem letzteren Fall durchquert sie den lokalen Switch und den GP-Link, um zu dem GP zu kommen, anschließend geht sie über den HS-Link zu dem hierarchischen Switch, dann über den GP und den lokalen Switch in dem Home-Knoten zu dem Home-Verteilerbus.

[0233] Es ist zu beachten, dass die globale Anforderungen nicht zuerst auf dem Verteilerbus des Home-Knotens erscheinen, sondern stattdessen über den GP-Link direkt zu dem HS-Link geleitet werden. Bei Protokollen nach dem Stand der Technik greift eine globale Anforderung auf den Status des Home-Knotens zu, bevor sie zu einem anderen Knoten ausgesendet wird. Die vorliegende Erfindung verringert die Durchschnittslatenz der globalen Anforderungen durch das Ausgeben von globalen Anforderungen direkt zu dem HS.

[0234] Im Folgenden Bezug nehmend auf die Fig. 20A–Fig. 20J, werden exemplarische Flussdiagramme einer Anzahl von grundlegenden Speichertransaktionen bereitgestellt.

Lokales Lesen

[0235] In der Fig. 20A wird eine Anforderung von einem Quellenprozessor **320** zu dem Home-Verteilerbus weitergeleitet. Das Verzeichnis **322** bestimmt, welcher Prozessor den Speicherblock besitzt. Wenn der lokale Speicher **323** der Besitzer ist, wird von dem Home-Verteilerbus ein kurzer Füll-Befehl zu dem Quellenprozessor **320** ausgegeben.

Globales Lesen

[0236] In der Fig. 20B ist vorausgesetzt, dass der Prozessor **320** des Knotens **325** einen Lese-Befehl zu einer Cache-Zeile des Speichers ausgibt, dessen Home ein Knoten **326** ist. Der globale Lesebefehl wird durch den Schalter **324** zu dem Home-Verteilerbus und zu dem Verzeichnis **321** über den Pfadweg, der durch die Linie **327** angezeigt wird, geleitet. Wenn der Speicher **330** des Knotens **326** der Besitzer der Cache-Zeile ist, dann werden die Daten von dem Knoten **326** durch den Knoten **326**, eine Shortfill-Antwort ausgebend, zu dem Knoten **325** zurückgesendet.

[0237] Wenn ein anderer Prozessor/anderes IOP aktuell die Cache-Zeile besitzt, werden verschiedene Schritte unternommen, um die angeforderte Cache-Zeile zu erhalten. Im Folgenden Bezug nehmend auf die Fig. 20C, wird der Lesebefehl, wenn der Prozessor **320** einen Lesebefehl zu einer Cache-Zeile in dem Speicher des Homes des Knotens **326** ausgibt, wieder über den Pfadweg **327** zu dem Home-Verteilerbus und dem Verzeichnis geleitet. Der Eintrag des Verzeichnisses **321** enthält, wie oben erwähnt, für jede Cache-Zeile des Speichers 14 Bits Statusinformation, die die Besitzerinformation enthält. In diesem Fall identifiziert die Besitzerinformation als den Besitzer den Prozessor **342** in dem Knoten **328**.

[0238] Als Antwort auf die Anzeige des Verzeichnisses, dass der Knoten **328** die erforderliche Cache-Zeile besitzt, treten zwei Ereignisse ein. Zuerst gibt der Home-Knoten, Knoten **326**, einen FR-Probe zu dem Besitzerprozessor **342** aus, wie durch die Linie **329** angezeigt wird. Gleichzeitig damit sendet der Home-Knoten **326** eine Fill-Marker-Antwort zu dem Prozessor **320**; wie durch die Linie **331** angezeigt. Die Rolle der Fill-Marker-Antwort wird in einem späteren Abschnitt beschrieben.

[0239] Als Antwort auf den weitergeleiteten Lesebefehl gibt der Prozessor **342** einen Füllbefehl zu dem Prozessor **320** aus, wobei der Füllbefehl die betreffende Cache-Zeile enthält. Dieser Typ der Antwort auf eine Leseabforderung wird als Long Fill bezeichnet, weil eine Abfolge von drei Befehlen erforderlich ist, um die Daten zurückzusenden. Infolgedessen können die Lesetransaktionen in zwei Typen unterteilt werden: Einen Short

Fill, der eine Antwort von dem Speicher ist, und einen Long Fill, der eine Antwort von dem Besitzer eines Prozessors ist.

Lokaler RdMod

[0240] Im Folgenden Bezug nehmend auf die Fig. 20D, kann dieser entnommen werden, dass eine lokale Lese-Maddifiziere-Transaktion gleichartig wie eine lokale Lese-Transaktion arbeitet, mit der Ausnahme, dass (1) zu allen Prozessoren, die eine Kopie der aktuellen Version der Cache-Zeile erhalten haben, Invalidate Probes gesendet werden und (2) FRMod und FillMod anstelle von Frds und Fills zu dem Besitzer gesendet werden.

[0241] In der Fig. 20D zeigt das Verzeichnis in dem Home-Knoten an, dass ein lokaler Prozessor oder Speicher den Block besitzt. In dem Home-Verteilerbus identifiziert das Verzeichnis **322** alle externen Knoten, die die aktuelle Version des Blocks erhalten haben. Ein Befehl Invalidate mit allen in einem Multi-Cast-Vektor identifizierten entsprechenden Knoten wird zu dem HS **324** gesendet. Der HS massenversendet die Invalidate-Nachrichten zu allen in dem Vektor identifizierten Knoten. Die Invalidate-Nachrichten gehen zu dem Verteilerbus in jedem der Knoten, wo der DTAG diese weiter filtert und Invalidate-Probes nur zu jenen Prozessoren oder IOPs sendet, die als die aktuelle Version der Cache-Zeile aufweisend identifiziert sind.

Globaler RdMod

[0242] Im Folgenden Bezug nehmend auf die Fig. 20E, kann dieser entnommen werden, dass eine Lese-Modifiziere-Transaktion gleichartig wie die Lesetransaktionen, die in Bezug auf die Fig. 20A und Fig. 20B beschrieben wurden, arbeitet. Ein Befehl Lesen/Modifizieren (RdMod) wird zuerst von dem Prozessor **320** zu dem Home-Verteilerbus und dem Home-Verzeichnis **321** der Cache-Zeile geleitet. Wenn der Speicher in dem Knoten **326** die Cache-Zeile speichert, dann wird von dem Prozessor **326** ein Befehl Short Fill Modify, der die Reset-Daten enthält, zu dem Prozessor **320** weitergeleitet. Im Ergebnis dieser Transaktion wird das Verzeichnis **321** aktualisiert. Der Lese-Modifizier-Befehl zeigt an, dass der Prozessor **320** exklusiven Besitz der Cache-Zeile erfordert, so dass er die Inhalte der Cache-Zeile modifizieren kann. Deshalb gibt der Knoten **326** zusätzlich zu dem Befehl Short Fill Modify außerdem Befehle Invalidate zu allen Prozessoren aus, die eine aktuelle Kopie der Cache-Zeile erhalten haben. Das DIR identifiziert die Knoten, auf denen ein Prozessor oder mehrere Prozessoren eine Kopie der aktuellen Version der Cache-Zeile erhalten hat bzw. haben. Die Präsenzbits des DIRs enthalten diese Informationen. Der DTAG identifiziert alle Home-Knoten-Prozessoren, die eine Kopie der Cache-Zeile erhalten haben. Befehle invalidate, dass diese ihr jeweiliges DIR-Präsenzbit zu setzen haben, werden zu allen Knoten gesendet. In jedem der Knoten, der einen Befehl Invalidate empfängt, wird der DTAG aktiviert, um festzustellen, welche Prozessoren aktuell eine Kopie der Cache-Zeile speichern. Die Befehle Invalidate werden nur an diese Prozessoren gesendet. Der IOP-Tag wird verwendet, um zu bestimmen, ob das IOP eine Kopie hat, falls ja, empfängt das IOP ebenso einen Invalidate-Probe-Befehl.

[0243] In dem Fall, in dem ein anderer als der anfordernde Prozessor der Besitzer ist, generiert der Home-Knoten einen Füllen/Modifizieren Marker, einen weitergeleiteten Befehl Lesen/Modifizieren und null oder mehr Befehle Invalidate als einen Befehl. An dem Knoten wird der Befehl zu allen Zieladressenknoten massenversendet. An jedem Zieladressenknoten wird der Befehl in seine Komponenten getrennt und der globale Port jedes Knotens bestimmt, welche Handlung an dem jeweiligen Knoten ausgeführt werden sollte. In dem Beispiel oben wird ein weitergeleiteter RdMod-Befehl durch den Prozessor **342** weiterverarbeitet und ein Marker Füllen/Modifizieren wird durch den Prozessor **320** weiterverarbeitet. Zusätzlich werden Befehle Invalidate an dem Home-Knoten, an dem Knoten, der den Marker Füllen/Moditfizieren und an dem Knoten, der den weitergeleiteten Befehl Modifizieren empfängt, in Übereinstimmung mit ihren DTAG-Einträgen durchgeführt. In Reaktion auf den weitergeleiteten RdMod-Befehl werden die dirty Daten von dem Prozessor **342** über den Befehl langes Füllen/Modifizieren zu dem Prozessor **320** weitergeleitet.

[0244] Infolgedessen kann der Befehl RdMod entweder zwei oder drei Knotenverbindungen oder Hops durchführen. In einem Ausführungsbeispiel der Erfindung resultieren nur die Befehle des Lesetyps (Lesen und Lesen/Modifizieren) in drei Hops, wobei der dritte Hop ein Befehl des Fülltyps ist (entweder Füllen oder Füllen/Modifizieren). Jedoch kann die Erfindung durch adäquates Zuordnen von den zugefügten Befehlen in den virtuellen Kanalwarteschlangen, die unten beschrieben werden, leicht modifiziert werden, um andere Transaktionen zu enthalten, die drei oder mehr Hops erforderlich machen.

CTDs

[0245] Im Folgenden Bezug nehmend auf die Fig. 20G und Fig. 20H, wird jeweils der Basisfluss für

Clean-to-Dirty (CTD) und Invalidate-to-Dirty (ITD) gezeigt. In der Fig. 20G wird ein Clean-to-Dirty von dem Prozessor **320** zu dem Verzeichnis **321** in dem Home-Knoten ausgegeben. Entweder wird ein Bestätigungsbefehl (ACK) oder eine Nichtbestätigungsbefehl (NACK) zu dem Prozessor **320** zurückgesendet, abhängig davon, ob die saubere Cache-Zeile, die der Prozessor **320** aktualisieren will, aktuell oder veraltet ist. Dementsprechend wird über den CTD ausgesagt, dass er erfolgreich ist oder versagt. Zusätzlich werden Befehle Invalidate zu allen Knoten, die durch das Vorhandensein des Bits des Verzeichnisses **321** als eine Kopie der Cache-Zeile aufweisend angezeigt werden, gesendet, wenn der CTD erfolgreich ist.

[0246] Wie in der Fig. 20H gezeigt, arbeitet der Befehl ITD im Wesentlichen gleichartig wie der CTD. Jedoch versagt der ITD nie. Ein ACK wird immer zu dem Prozessor **320** zurückgesendet und Befehle Invalidate werden zu anderen Knoten in dem System gesendet, die eine Kopie der Cache-Zeile von Daten speichern.

Lokale und globale Write Victims

[0247] Wie oben beschrieben, leitet der Befehl Write Victim dirty Daten von dem Cache des Prozessors zurück in den adäquaten Home-Speicher weiter. Im Folgenden Bezug nehmend auf die Fig. 20I bis Fig. 20J, kann diesen entnommen werden, dass der Fluss für die Write Victims in Abhängigkeit davon, ob der Home-Speicher in demselben Knoten wie der Prozessor ist, der den Befehl Write Victim ausgibt, oder nicht, geringfügig verschieden ist. Wie in der Fig. 20I gezeigt, gibt der Prozessor **320**, wenn der Home-Knoten der Knoten des Prozessors ist, den Befehl Write Victim aus und die Daten werden direkt zu dem Speicher desselben Knotens weitergeleitet.

[0248] Wie in der Fig. 20J gezeigt, werden die Daten jedoch, wenn die Victim-Daten in einem anderen Home als der Prozessor sind, in zwei Stufen übertragen. Zuerst wird die Cache-Zeile aus dem Cache (oder Victim-Puffer) heraus weitergeleitet und in dem Victim-Cache (Fig. 6, Element **124**) an dem globalen Port des Knotens des Prozessors gespeichert. Der Victim-Cache antwortet dem Prozessor mit einem Victim-Freigabesignal, das anzeigt, dass der Prozessor den Victim-Puffereintrag erneut nutzen kann. Dann, wenn die verfügbare Bandbreite auf dem Switch vorhanden ist, werden die Victim-Daten von dem Victim-Cache über einen Befehl Write Victim zu dem Speicher des Home-Prozessors weitergeleitet.

[0249] Es sollte beachtet werden, dass die Victim-Daten, die durch den Quellenprozessor P zu dem Home-Speicher gesendet werden, zu dem Zeitpunkt, zu dem sie zu dem Speicher gelangen, veraltet sein können. In einem solchen Fall wird über den Victim-Befehl aus- gesagt, dass er versagt hat, und der Home-Speicher wird nicht aktualisiert. Dieses Szenario tritt ein, wenn sich ein anderer Prozessor in dem Intervall zwischen dem Aneignen des Besitzes der Cache-Zeile durch P und dem Erreichen des Home-Verzeichnisses durch Ps Victim den Besitz der Cache-Zeile aneignet. In einem derartigen Fall muss ein Befehl Invalidate oder ein Befehl FrdMOd Probe für die Cache-Zeile zu dem Prozessor P gesendet worden sein, bevor Ps Victim den Home-Verteilerbus erreicht hat.

[0250] Um zu bestimmen, welche Victim-Daten in den Speicher geschrieben werden sollten, wird der Verzeichniseintrag für die anfordernde Adresse nachgeschlagen, wenn ein Befehl Write Victim auf dem Home-Verteilerbus erscheint. Wenn das Verzeichnis anzeigt, dass der Quellenprozessor noch immer der Besitzer der Cache-Zeile ist, dann ist der Victim-Befehl erfolgreich und aktualisiert den Speicher. Andernfalls versagt er und aktualisiert den Speicher nicht. In beiden Fällen wird, sobald die Entscheidung für ein Victim in dem Verzeichnis **321** getroffen ist, ein Victim-Ack-Befehl zu dem globalen Port des Knotens **325** zurückgesendet, um dem Victim-Cache zu ermöglichen, den zugehörigen Eintrag zu löschen.

[0251] In einem Ausführungsbeispiel der Anordnung wird der DTAG verwendet, um in dem Fall, in dem der Befehl Write Victim lokal ist, über den Erfolg oder das Versagen eines Befehls Write Victim zu entscheiden. In diesem besonderen Fall (dem einer lokalen Anforderung Write Victim) sind sowohl der DTAG als auch das DIR in der Lage, die Informationen bereitzustellen, die gebraucht werden, um den Erfolg oder das Versagen des Befehls Write Victim festzustellen. Der DTAG wird deshalb anstelle des DIRs verwendet, weil der DTAG-basierte Mechanismus bereits in der Hardware des kleinen SMP-Knotens bereitgestellt ist.

[0252] Bei der Beschreibung des Cache-Kohärenz-Protokolls wurden die gebräuchlichsten Operationen und Befehlstypen beschrieben. Die Mechanismen werden in den folgenden Abschnitten ausführlicher beschrieben.

[0253] Wie oben erwähnt, können in einem Ausführungsbeispiel der Erfindung für die Effizienz zwei oder mehr in Beziehung stehende Pakete kombiniert werden. Das kombinierte Paket wird dann an dem HS oder dem Verteilerbus an einem Knoten in seine Komponenten geteilt. Beispielsweise teilt sich eine FrdMod-Nach-

richt zu dem HS in eine FrdMod-Nachricht zu dem Knoten mit dem Besitzerprozessor, in Invalidate-Nachrichten zu Knoten mit Kopien der Cache-Zeile und eine FillMarkerMod-Nachricht zu dem Quellenknoten. Die FrdMod-Nachricht zu dem Besitzer des Prozessorsknotens teilt sich des Weiteren an dem Verteilerbus des Knotens in eine FrdMod-Nachricht zu dem Besitzerprozessor und null oder mehr Invalidate-Nachrichten zu anderen Prozessoren in dem Knoten.

Verzögerte Schreibpufferung zum Aufrechterhalten der Victim-Kohärenz

[0254] Wie oben in Bezug auf die Fig. 20I und Fig. 20J beschrieben, können die Victim-Daten, die zu dem Home-Speicher gesendet werden, zum Zeitpunkt ihres Eintreffens auf Grund eines dazwischenkommenden Befehls invalidate oder FrdMod-Probe-Befehls für die Cache-Zeile, bevor der Write-Victim-Befehl den Home-Verteilerbus erreicht hat, veraltet sein. Ein Verfahren zum Bestimmen, ob die Victim-Daten in den Speicher geschrieben werden sollten, ist ist dem Verzeichniseintrag für jeden Victim-Schreibbefehl nachzuschlagen. Wenn das Verzeichnis anzeigt, dass der Prozessor, der der Victim-Schreibbefehl ausgibt, der Dirty-Besitzer ist, dann sollte der Victim-Befehl zugelassen werden, um durchgesetzt zu werden. Andernfalls sollte er versagen. Dieses Vorgehen ist erwünscht, weil der Bedarf für aufwendiges Vergleichen der Logikstrukturen zum Übereinstimmen der Victim-Schreibbefehle zwischen dem Prozessor und dem Serialisierungspunkt mit Probe-Befehlen zwischen dem Serialisierungspunkt und dem Prozessor umgangen wird.

[0255] Während dieser Ansatz das Aufrechterhalten der Datenkohärenz vereinfacht, kann er Leistungsnachteile in Form von verringerter Speicherbandbreite verursachen. Gemäß diesem Schema muss das System jedes Mal, wenn es einen Victim-Schreibbefehl ausführt, zuerst auf den Verzeichnisstatus zugreifen, dann den Status evaluieren und schließlich, basierend auf dem Status, einen DRAM-Schreibbefehl der Victim-Daten ausführen. Da auf den Speicher und das Verzeichnis automatisch zugegriffen wird, würde der gesamte Victim-Schreibzyklus, wenn das System gemäß der Methode nach dem Stand der Technik ausgeführt würde, gleich der Summe der Verzeichnis-Nachschlagezeit, der Statusbewertungszeit und der DRAM-Schreibzeit sein. Ein derartiges System würde in Bezug auf jene Systeme, deren gesamter Victim-Zyklus nur aus einem DRAM-Schreiben besteht, schwere Leistungseinbußen erleiden.

[0256] Ein Ausführungsbeispiel der Erfindung überwindet das Problem der Verschlechterung der Speicherbankausnutzung durch das Bereitstellen eines Schreibverzögerungspuffers an jeder Speicherbank. Jedes Mal, wenn zu dem Speichersystem ein Victim-Schreibbefehl ausgegeben wird, antwortet das Speichersystem durch das parallele Ausführen der folgenden Funktionen: Speichern der Victim-Schreibdaten in einem Schreibverzögerungspuffer in der Zielspeicherbank und Markieren der Blocks als „nicht schreibbar" oder „ungültig", Zugreifen auf den mit dem Victim-Schreibbefehl assoziierten Verzeichnisstatus und Ausführen, anstelle des aktuellen Victim-Schreibbefehls, eines DRAM-Schreibbefehls eines zuvor gepufferten Victim-Schreibbefehls, der als „schreibbar" oder „gültig" markiert ist. Dann, wenn der Zugriff auf das Verzeichnis ausgeführt ist, zeigt der dem Victim-Schreibbefehl zugehörige Status an, dass der Victim-Schreibbefehl erfolgreich sein soll und der Schreib-Verzögerungspuffer, in dem sich die Victim-Datenl befinden, geht in den „schreibbaren" oder „gültigen" Status über. Der „schreibbare" oder „gültige" Status eines Datenblocks in dem Schreibverzögerungspuffer zeigt an, dass die Daten in dem Puffer eine aktuellere Version der Cache-Zeile sind, als die Version, die in den DRAMs des Speichers gespeichert sind. Wenn der Puffer als „schreibbar" oder „gültig" markiert ist, werden seine Daten im Ergebnis der anschließenden Ausgabe eines Victim-Schreibbefehls zu dem Speichersystem in den DRAM geschrieben.

[0257] Durch das Ausführen des Verzeichnisnachschlagens parallel zu dem DRAM-Schreiben eines zuvor ausgegebenen Victim-Schreibbefehls verringert dieses Ausführungsbeispiel die Victim-Gesamtzykluszeit auf die einer DRAM-Schreibzeit. Da dieses Ausführungsbeispiel „schreibbare" oder „gültige" Datenblöcke für viele Zyklen in den Schreibverzögerungspuffern hält, in denen nachfolgende Referenzen zu dem Pufferblock zu dem Speicherblock ausgegeben werden können, enthält der Schreibverzögerungspuffer ein assoziatives Adressregister. Die Adresse des Victim-Schreibblocks wird gleichzeitig mit dem Speichern der zugehörigen Daten in dem Schreibverzögerungspuffer in dem assoziativen Adressregister gespeichert. Wenn nachfolgende Referenzen zu dem Speichersystem ausgegeben werden, identifiziert das Speichersystem durch das Mittel eines Adressabgleichens mit dem Adressregister jene, die in dem Schreibverzögerungspuffer Adressen blockieren. Durch diese Einrichtung wird das Speichersystem alle Referenzen zu Blocks in den Schreibverzögerungspuffern mit den aktuelleren Daten aus den Puffern, anstatt mit den veralteten Daten in dem Speicher der DRAMs, bedienen.

[0258] Die oben beschriebene Technik des Bereitstellens von Schreibverzögerungspufferung der Victim-Daten kann ebenso in auf einem Snoopy-Bus basierenden Systemen, die keinen direkten DTAG-Status enthalten,

diesen jedoch verwenden, um die Gültigkeit eines Datenblocks zu bestimmen, verwendet werden.

[0259] Im Folgenden Bezug nehmend auf die Fig. 21, wird ein Ausführungsbeispiel eines Speichersteuersystems zum Bereitstellen von verzögerten Schreiboperationen gezeigt, das einen Speicher-Kontroller **332**, der gekoppelt ist, um ein Signal Owner_Match auf der Leitung **140** von dem Verzeichnis **140** zu empfangen, enthält. Zusätzlich empfängt der Speicher-Kontroller **332** einen Eingang von dem QS-Verteiler **11** (der ebenso das Verzeichnis **140** speist) zum Verfolgen der Befehle, die in das Verzeichnis eingegeben werden.

[0260] Der Speicher-Kontroller **332** enthält einen Schreibverzögerungspuffer **336**. Jeder Eintrag in den Schreibverzögerungspuffer **336** enthält einen Datenteil **336a**, einen Flag-Teil **336b** und einen Adressteil **336c**. In einem Ausführungsbeispiel der Erfindung enthält der Schreibverzögerungspuffer, um die Komplexität der Ausführung zu verringern, nur jeweils einen Adress-, Daten- und Flag-Eintrag, obwohl die Erfindung nicht auf eine derartige Konfiguration beschränkt ist.

[0261] Der Schreibverzögerungspuffer arbeitet wie folgt. In Betrieb, während ein Befehl, eine Adresse und Daten in dem Verteilerbus **130** empfangen werden, werden diese zu dem Verzeichnis **140** und auch zu dem Kontroller **332** weitergeleitet. Der Speicher-Kontroller **332** speichert den Befehl, die Adresse und die Daten in dem Schreibverzögerungspuffer **336** für eine Transaktionsperiode (hier 18 Taktzyklen). Während der Transaktionsperiode wird auf das Verzeichnis **140** zugegriffen und die Ergebnisse des Zugriffs werden auf der Leitung Owner_Match **140a** durchgesetzt. Die Leitung Owner_Match wird aktiviert, wenn der Verzeichniseintrag anzeigt, dass die Prozessor-ID des Prozessors, der versucht, den Speicher zu aktualisieren, tatsächlich die Besitzerin der Cache-Zeile der Daten ist. Das Signal Owner_Match wird verwendet, um das Flag **336b** des Schreibverzögerungspuffereintrags **336** zu setzen. in der darauf folgenden Transaktionsperiode werden, wenn der Speicherbus verfügbar ist und wenn das Flag **336b** gesetzt ist, die gespeicherten Daten in den Speicher geschrieben. In einem Ausführungsbeispiel der Erfindung werden nur Schreibvorgänge gepuffert und ein eingehender Lesevorgang wird zugelassen, um auf den Speicherbus zuzugreifen, ohne verzögert zu werden. Darauf folgende Leseoperationen an Victim-Daten, die in dem Schreibverzögerungspuffer gespeichert sind, werden von dem Schreibverzögerungspuffer bedient.

[0262] Im Folgenden Bezug nehmend auf die Fig. 22, wird ein Zeitablaufdiagramm des Betriebs einer Schreibverzögerungsoperation gezeigt. Zu der Zeit T0 wird auf dem Verteilerbus ein Vorgang Read0 empfangen. Diese Leseoperation wird sofort in dem Speicher propagiert, um auf den DRAM **334** zuzugreifen. Zu der Zeit T1 wird eine Operation Write1 auf dem Verteilerbus empfangen. Während dieses T1-Zyklus wird auf das Verzeichnis **140** zugegriffen und bei Abschluss des T1-Zyklus wird das Signal Owner_Match, eine Übereinstimmung der WRITE1-Adresse anzeigend, aktiviert. Im Ergebnis wird das Flag **336b** des Schreibverzögerungspuffereintrags gesetzt. Zu der Zeit T2 wird ein Lesevorgang empfangen und zu dem Speicher vor der Operation WRITE1 weitergeleitet. Während der Zeit T3 wird das der Operation WRITE1 entsprechende Flag aktiviert und wenn die nächste Operation WRITE3 in dem Schreibverzögerungspuffer empfangen wird, wird die Operation WRITE1 zum Speicher für Behandlung durch den DRAM **334** weitergeleitet.

[0263] Es sollte beachtet werden, dass zum Auslesen des lokalen Speichers alternativ die DTAGs verwendet werden können, um das Flag-Bit in dem Schreibverzögerungspuffer zu setzen. Eine der Cache-Zeilen von einem lokalen Speicher kann in einem der Caches der Prozessoren in dem lokalen Knoten gespeichert werden. Wenn einer der Prozessoren eine Cache-Zeile zu einem Victim macht und die Cache-Zeile in den Schreibverzögerungspuffer geschrieben wird, können die DTAG-Einträge für jede Cache-Zeile geprüft werden, um zu bestimmen, ob die Cache-Zeile in einem der Prozessoren vorhanden war oder nicht. Wenn die Cache-Zeile in einem der Prozessoren anwesend war, wird das Gültigkeits-Bit des DTAG-Eintrags geprüft, um sicherzustellen, dass die Kopie, die der Prozessor zu einem Victim macht gültig war. Wenn in dem DTAG ein Treffer ist und die Cache-Zeile gültig war, kann der DTAG das Flag in dem Schreibverzögerungspuffer setzen, um zu veranlassen, dass die Cache-Zeile in den lokalen Speicher geschrieben wird. Dies ermöglicht einfachen snoopy-bus-basierten Systemen (d. h. ohne Verzeichnis) denselben Vereinfachungsalgorithmus anzuwenden.

[0264] Die Speichersteuerlogik der Fig. 21 ermöglicht infolgedessen, dass READ-Operationen sofort in einem READ-Zyklus ausgeführt werden und dass eine WRITE-Operation für jeden WRITE-Zyklus ausgeführt wird (selbst, wenn es ein verzögertes Schreiben ist). Im Ergebnis wird ein gleichförmiger Datenstrom zu den DRAMs weitergeleitet, ohne dass Verzögerungen im Ergebnis der Zugriffe auf das Verzeichnis eintreten und die Leistung wird erhöht, während die Kohärenz aufrechterhalten bleibt. Obwohl die Schreibverzögerungspuffer-Technik hierin in Bezug auf Victim-Schreibvorgänge beschrieben wurde, kann sie in jedem System verwendet werden, in dem der Kohärenzzustand zentralisiert und stationär ist, um die Speicherleistung zu verbessern.

[0265] Danach kann festgestellt werden, dass viele Speicherreferenzen zwischen den Prozessoren, den Verzeichnissen, den Speichern und den DTAGs übertragen werden, um das Cache-Kohärenz-Protokoll zu implementieren. Zusätzlich kann jede Speicherreferenz eine Anzahl von Transaktionen oder Hops zwischen den Knoten, in denen Nachrichten für die Speicherreferenz übertragen werden, bevor die gesamte Referenz ausgeführt ist, enthalten. Wenn die Abhängigkeiten zwischen den Nachrichten verursachen, dass eine Referenz indefinit blockiert wird, verklemmt das Mehrprozessorsystem.

[0266] Wie zuvor kurz beschrieben, bewerkstelligt ein Ausführungsbeispiel der Erfindung den Verkehr zwischen den Knoten und hält durch die Verwendung einer virtuellen Kanalflusssteuerung die Datenkohärenz ohne Verklemmung aufrecht. Virtuelle Kanäle wurden zuerst zum Bereitstellen von verklemmungsfreiem Routing in Verbindungssysteme eingeführt. Gemäß dem Ausführungsbeispiel der Erfindung können die virtuellen Kanäle zusätzlich verwendet werden, um Ressourcenverklemmungen in einem Cache-Kohärenz-Protokoll für Computersysteme mit gemeinsamen Speichern zu verhindern.

[0267] Nach dem Stand der Technik werden in Bezug auf Cache-Kohärenz-Protokolle zwei Lösungswege bereitgestellt. Für Systeme, die eine kleine Anzahl von Prozessoren und eine kleine Anzahl von aktuell ausstehenden Anforderungen haben, werden Warteschlangen und Puffer bereitgestellt, die groß genug sind, um die größtmögliche Anzahl von Antworten, die zu einem Zeitpunkt während der Ausführung vorhanden sein können, zu enthalten. Das Bereitstellen von ausreichend vielen Warteschlangen und genügend Pufferraum garantiert, dass Nachrichten, um Fortschritt zu machen, nie von anderen Nachrichten abhängig sind.

[0268] In größeren Systemen mit einer großen Anzahl von ausstehenden Anforderungen, ist es nicht praktisch, Puffer und Warteschlangen bereitzustellen, die groß genug sind, um die maximale Anzahl von möglichen Antworten zu enthalten. Dementsprechend wurde das Problem durch die Verwendung von Zweikanal-Interconnects, die über einen Verklemmungserfassungs- und Verklemmungsauflösungsmechanismus gekoppelt waren, gelöst. Zuerst verwendet die Interconnect-Verbindung (logische Pfade, die verwendet werden, um zwei Nachrichten zwischen Systemkomponenten, wie zum Beispiel Prozessor und Speicher, zu bewegen) zwei Kanäle, einen Anforderungskanal (oder Kanal niedrigerer Ordnung) und einen Antwortkanal (oder Kanal höherer Ordnung). Die Kanäle sind typischerweise physikalisch, das bedeutet, sie verwenden verschiedene Puffer und Warteschlangen. Zweitens wird typischerweise eine Heuristik angewendet, um eine potenzielle Verklemmung zu erfassen. Beispielsweise kann ein Kontroller eine potenzielle Verklemmung signalisieren, wenn eine Warteschlange voll ist und für einige Zeit keine Nachricht aus der Warteschlange entfernt wurde. Drittens wird ein Verklemmungsauflösungsmechanismus angewendet, bei dem Auswahlnachrichten negativ bestätigt werden, um so Ressourcen freizumachen und infolgedessen zu ermöglichen, dass andere Nachrichten Fortschritte machen. Negative Bestätigungsnachrichten verursachen, dass der entsprechendene Befehl zurückgezogen wird.

[0269] Die oben beschriebene Lösung für das große System hat zwei grundsätzliche Probleme: ein Fairness-/Aushungerungsproblem und ein Problem der Leistungseinbuße. Weil einige der Nachrichten negativ bestätigt werden, ist es möglich, dass einige Befehle für eine lange Zeit nicht ausgeführt werden (potenziell indefinit). Wenn nicht garantiert ist, dass ein Befehl innerhalb eines gegebenen Zeitraums ausgeführt wird, erhält die Ressource, die den Befehl ausgibt, keinen fairen Zugriff auf die Systemdaten. Außerdem kann die Ressource, weil sie keinen fairen Zugriff auf die Systemdaten bekommt, nach Daten aushungern und potenziell das System verklemmen. Da die meisten Nachrichten negativ bestätigt sein könnten und infolgedessen darin versagen, zu ihren Zieladressen zu kommen, müssen Protokollmeldungen, wie zum Beispiel Invalidate-Nachrichten, eine Bestätigung erzeugen, um anzuzeigen, dass sie ihre Zieladresse erfolgreich erreichen. Des Weiteren muss der Kontroller warten, bis alle Bestätigungen empfangen wurden, bevor er die entsprechende Befehlsausführung berücksichtigen kann. Dieser Nichtdeterminismus führt sowohl zu einem Nachrichten-Overhead als auch zu, untypischer Latenz, die die Gesamtleistung des Cache-Kohärenz-Protokolls verringert.

[0270] Gemäß einem Ausführungsbeispiel der Erfindung wird ein Cache-Kohärenz-Protokoll verwendet, das einen systematischen und deterministischen Ansatz zur Vermeidung von Verklemmung verwendet. Anstatt eine potenzielle Verklemmung zu erfassen und dann korrigierend einzugreifen, wird die Verklemmung durch die Ausführung eliminiert. Dementsprechend besteht kein Bedarf für einen Verklemmungserfassungs- und Verklemmungsauflösungsmechanismus. Zweitens werden, da die Nachrichten nie negativ bestätigt werden, Bestätigungen für Protokollmeldungen, wie Invalidate-Nachrichten, nicht erforderlich und deshalb werden die Bandbreite und die Latenz verbessert.

[0271] Für den Zweck des Erklärens der Verwendung von virtuellen Kanälen wird zuerst eine dienliche Ter-

minologie bereitgestellt.

[0272] Abhängigkeit: Eine Nachricht M1 ist als „abhängig" von Nachricht M2 definiert, wenn M1 keinen Fortschritt machen kann, ohne dass M2 Fortschritt macht. Des Weiteren wird Abhängigkeit als transitiv definiert. Für die Implementierung des Cache-Kohärenz-Protokolls der vorliegenden Erfindung sind wenigstens zwei Klassen von Abhängigkeiten vorhanden: Ressourcenabhängigkeiten und Flussabhängigkeiten. M1 wird als „ressourcenabhängig" von M2 definiert, wenn M1 keinen Fortschritt machen kann, bis M2 eine Ressource, wie zum Beispiel einen Warteschlangenschlitz, freimacht. M1 wird als „flussabhängig" von M2 definiert, wenn das Cache-Kohärenz-Protokoll erfordert, dass M1 keinen Fortschritt macht, bis M2 Fortschritt macht. Beispielsweise kann das Cache-Kohärenz-Protokoll erfordern, dass M1 blockiert, bis das Verzeichnis einen bestimmten Status erreicht, und es ist M2, die den Verzeichnisstatus auf den erwünschten Wert setzt. M1 wird als abhängig von M2 definiert, wenn eine Kette von entweder Ressourcen- oder Flussabhängigkeiten von M1 zu M2 vorhanden ist.

[0273] Abhängigkeitszyklus: Ein „Abhängigkeitszyklus" ist definiert, zwischen einer Reihe von Nachrichten M1, M2 (≥ 2) vorhanden zu sein. Wenn der Fortschritt von M1 von dem Fortschritt von M2 abhängig ist, hängt der von M2 von dem von M3 ab, der von Mk-1 hängt von dem von Mk ab und schließlich hängt der von Mk von dem von M1 ab. Ein System von Nachrichten verklemmt, wenn irgendeine Teilmenge der Nachrichten einen Abhängigkeitszyklus bildet. Da M1 von Mk abhängig ist, die wiederum von M1 abhängig ist, kann keine der Nachrichten in dem Zyklus Fortschritt machen.

[0274] Das hierin offen gelegte Verfahren und die hierin offen gelegte Vorrichtung verwenden virtuelle Kanäle, um Verklemmungen in dem Cache-Kohärenz-Protokoll deterministisch zu vermeiden. Im Folgenden werden sowohl die benötigten Hardwaremechanismen als auch die Reihe von Regeln, die bei der Ausführung des Cache-Kohärenz-Protokolls befolgt wird, beschrieben.

[0275] In einem Ausführungsbeispiel definiert das Cache-Kohärenz-Protokoll, dass alle Speichervorgänge in höchstens drei Phasen ausgeführt sein müssen. In der ersten Phase wird eine Nachricht oder werden mehrere Nachrichten zwischen den Komponenten des Systems übertragen. Deshalb wird jede Phase ebenso als ein „Hop" bezeichnet. Die Hops werden mit 0, 1 und 2 nummeriert. In Hop-0 wird eine Anforderung von einem Prozessor oder einem IO-Prozessor zu dem Home-Verzeichnis geleitet. In Hop-1 werden die Nachrichten, die durch das Home-Verzeichnis generiert wurden, zu einem Prozessor oder IO-Prozessor oder zu mehreren Prozessoren oder IO-Prozessoren geleitet. In Hop-2 bewegen Sich die Nachrichten von einem Besitzerprozessor zu dem Quellenprozessor. Die Hops werden in der **Fig. 23** dargestellt.

[0276] Es ist eine gewollte Eigenschaft des Cache-Kohärenz-Protokolls, dass alle Operationen in einer vorbestimmten Anzahl von Hops ausgeführt werden. In einem hierin beschriebenen Ausführungsbeispiel ist die vorbestimmte Anzahl drei, obwohl die Erfindung nicht auf eine bestimmte Anzahl von Hops beschränkt ist, solange die Anzahl, die ausgewählt wird, relativ gering und konsistent ist. Diese Eigenschaft ist der Schlüssel, der garantiert, dass alle Nachrichten ohne jeden Mechanismus zum Erfassen von Verklemmungen und versagenden und wiederholt gesendeten Nachrichten zum Lösen der Verklemmung zu ihren Zieladressen geleitet werden können.

[0277] Wie oben erwähnt, ist die Höchstanzahl von Hops in diesem Ausführungsbeispiel drei. Das System stellt folglich drei Kanäle bereit, die jeweils mit Q0, Q1 und Q2 gekennzeichnet werden. Die Kanäle sind logisch unabhängige Datenpfade durch die System-Interconnect-Verbindung. Die Kanäle können physikalisch oder virtuell sein (oder teilweise physikalisch und teilweise virtuell). Wenn sie physikalisch sind, hat jeder Kanal durchgängig durch das System eine verschiedene Warteschlange und verschiedene Pufferressourcen. Wenn sie virtuell sind, nutzen die Kanäle die Warteschlangen und Pufferressourcen unter den unten genannten Einschränkungen und Regeln.

[0278] Die drei Kanäle bilden eine Hierarchie. Q0 ist der Kanal niedrigster Ordnung, gefolgt von Q1, und Q2 ist der Kanal höchster Ordnung. Eine Nachricht in Kanal Qi kann nie von einer Nachricht in einem Kanal, der geringer als Qi ist, abhängig sein.

[0279] Einem Ausführungsbeispiel der Erfindung ist zusätzlich ein QIO-Kanal hinzugefügt, um Flussabhängigkeitszyklen zwischen den Antwortnachrichten von dem IO-System und Speicherraumbefehlen von dem IO-System zu eliminieren.

[0280] Schließlich wird in einem Ausführungsbeispiel der Erfindung ein Q0Vic-Kanal für Victim-Nachrichten

und darauf folgende abhängige Nachrichten, ausgegeben während Victim-Nachrichten ausstehend sind, angewendet.

[0281] Wie zuvor in Verbindung mit den **Fig.** 10A–Fig. 20H beschrieben, kann ein gegebenes Befehlspaket, das zu dem Switch ausgegeben wird, eine Zahlenreihe von diskreten Transaktionen generieren. In einem Ausführungsbeispiel der Erfindung wird jede diskrete Transaktion für ein gegebenes Befehlspaket einem Kanal zugeordnet. Die Kanäle stellen essentiell eine geordnete Struktur zum Definieren der Ausführungsphasen und der Abhängigkeiten eines gegebenen Befehlspakets bereit.

[0282] Im Folgenden Bezug nehmend auf die Fig. 22, stellt ein Flussdiagramm beispielsweise die Zuordnung von Kanälen für die diskreten Transaktionen, die in den **Fig.** 10A–Fig. 20J gezeigt werden, dar. Die diskreten Transaktionen werden durch die folgende Nomenklatur beschrieben: die erste Transaktion in einer Reihe von Transaktionen, die aus einer Referenz resultiert, wird als Q0- oder Q0Vic-Transaktion bezeichnet, die zweite Transaktion der Reihe von Transaktionen ist eine Q1-Transaktion und die dritte Transaktion in der Reihe von Transaktionen ist eine Q2-Transaktion.

[0283] Ein Q0- oder Q0Vic-Kanal trägt Anfangsbefehle von Prozessoren und IOPs, die das Verzeichnis noch nicht ausgesucht haben. Infolgedessen ist die Zieladresse eines Q0-/Q0Vic-Pakets immer das Verzeichnis. Der Q0Vic-Kanal ist speziell für Write-Victim-Befehle reserviert, während der Q0-Kanal alle anderen durch den Prozessor oder den IOP initiierten Befehle transportiert.

[0284] Ein in dem Schritt **380** ausgegebener Befehl kann versuchen, Daten oder den Aktualisierungsstatus zu erhalten. Der Status ist immer in dem Home-Verzeichnis, das der Adresse der Daten entspricht, verfügbar. In dem Schritt **382** wird auf das Home-Verzeichnis zugegriffen und es wird festgestellt, ob die verfügbare Cache-Zeile im Besitz des Speichers (relativ zu dem Verzeichnis) oder in dem eines anderen Prozessors ist. In beiden Fällen wird über den Q1-Kanal eine Antwort ausgegeben. Wenn in dem Schritt **382** festgestellt ist, dass der Status oder die Daten in dem zweiten Knoten verfügbar sind, dann ist in dem Schritt **384** die Antwort auf dem Q1-Kanal zurück zu dem ersten Knoten gerichtet. Die Q1-Transaktionen umfassen ShortFill, Short Fill Mod, VicACK, CTD-ACK/NACK usw.

[0285] Wenn in dem Schritt **382** festgestellt ist, dass der Home-Knoten die Daten nicht besitzt, dass die Daten jedoch dirty sind und im Besitz eines anderen Prozessors sind, dann wird in dem Schritt **386** auf dem Q1-Kanal eine Q1-Typ-Transaktion entweder eines weitergeleiteten Lesens oder eines weitergeleiteten Lesens/Modifizierens zu einem Remote-Knoten ausgegeben.

[0286] Wenn als Antwort auf eine Statusprüfung in dem Home-Knoten oder als Antwort auf ein Lesen/Modifizieren angezeigt ist, dass andere Knoten Daten, die ihren Status zu dirty geändert haben, gemeinsam nutzen, wird in dem Schritt **388** eine Q1-Typ-Transaktion Invalidate zu den anderen betroffenen Knoten weitergeleitet.

[0287] Infolgedessen dient der Q1-Kanal zum Transportieren von Paketen, die in ihrem zweiten „Hop" sind, wobei der erste „Hop" das Verzeichnis ist. Die Zieladresse des zweiten „Hops" ist immer ein Prozessor, wobei der Prozessor entweder in einem Knoten, der den ursprünglichen Befehl initiiert, ist oder in einem anderen Remote-Knoten in dem System ist.

[0288] Ein Q2-Kanal transportiert entweder eine Transaktion Long Fill oder eine Transaktion Long Fill Mod. Der Q2-Kanal transportiert die Daten aus dem dritten Knoten durch einen dritten „Hop" zurück zu dem Knoten, der den ursprünglichen Befehl initiiert hat. Die Zuordnung der Befehle in Befehle des Q0-/Q0Vic-, Q1- und Q2-Typs kann in einem SMP-System verwendet werden, um verklemmungsfreie Nachrichtenübermittlung in der folgenden Art und Weise bereitzustellen. Obwohl das Flussdiagramm der Fig. 23 die Wechselwirkung zwischen vier virtuellen Kanälen darstellt, können in einem Ausführungsbeispiel der Erfindung für den Zweck der Aufrechterhaltung der Cache-Kohärenz fünf virtuelle Kanäle verwendet werden. Der zusätzliche Kanal enthält einen QIO-Kanal. Generell trägt der QIO-Kanal alle Lese- und Schreibvorgänge, einschließlich des Steuerstatusregisterzugriffs (CRS-Zugriff), zu IO-Adressräumen.

[0289] Im Folgenden auf die Tabelle 2 Bezug nehmend, wird eine Liste von exemplarischen Befehls-Mappings in Kanalpfade bereitgestellt:

QIO	Alle IO-Raum-Anforderungen zu CPU	RdByteIO. RdWordIO, WrByteIO,WrWordIO
Q0	Alle Speicherraum-Anforderungen von CPU oder IOP	Rd. RdMod, Fetch, CTD, ITD, Vic, RdVic, RdModVic
Q0Vic	Alle Speicherraum-Anforderungen von CPU oder IOP die Daten übertragen	WrVic, Full Cache line Write, QV_Rd, QV_RdMod, QV_Fetch
Q1	Alle weitergeleiteten Befehle	FRd, FRdMod, Ffetch
	Alle Schattenbefehle	SFRd, SFRdMod, SFEtch, Sinval, Ssnap
	Short Fills	SFill, SfilMod
	Alle Arten von Fill Marker	FM, FMMod, Pseudo-FM, PSeudo-DMMod, FRdMod with FM
	Andere	CTD-ACK,CTD-NACK,ITD-ACK, Vic-ACK, VicRel
	IO-Raum-Antworten	IOFillMarker, IOWriteAck
	Consig, die in Beziehung stehen	Invl-Ack, LoopComSig
Q2	Long Fills	Fill, FillMod
	IO-Raum-Fills	IOFill

[0290] Eine Implementierung von virtuellen Kanälen in einem switchbasierten System involviert die Verwendung von physikalisch verschiedenen Warteschlangen, Puffern oder Pfaden für jeden Kanal. Alternativ können die Warteschlangen, Puffer und Pfade gemeinsam von den Kanälen genutzt werden und sind infolgedessen wirklich „virtuell". In einem Ausführungsbeispiel der Erfindung wird eine Kombination dieser Techniken verwendet, um die Hardware optimal auszunutzen.

[0291] Im Folgenden Bezug nehmend auf die Fig. 24, wird ein Beispiel gezeigt, wie ein Puffer von mehr als einem virtuellen Kanal genutzt werden kann. Der gezeigte Puffer **400** enthält eine Anzahl von Schlitzen". Jeder der Schlitze ist zur Verwendung durch nur einen der Kanäle zugeordnet. Beispielsweise umfasst der Schlitz **402** eine Anzahl von Puffereinträgen, die den Befehlen des Q2-Typs zugeordnet sind, Schlitz **404** umfasst eine Anzahl von Puffereinträgen, die den Befehlen des Q1-Typs zugeordnet sind, usw.

[0292] Die restlichen Schlitze **410** können durch Nachrichten für jeden der Kanäle verwendet werden und werden deshalb als gemeinsam genutzte oder generische" Schlitze bezeichnet. Für jeden Kanal ist ein Belegtsignal bereitgestellt. Das Belegtsignal zeigt an, dass ein Puffer nicht in der Lage ist, weitere Nachrichten zu speichern und deshalb nichts zu diesem Puffer übertragen werden sollte.

[0293] Zwischen dem Zeitpunkt, zu dem das Belegtsignal in einer gegebenen Ressource für einen gegebenen Kanal aktiviert wird, und dem Zeitpunkt, zu dem die Vorrichtung, die Befehle zu dieser Ressource ausgibt, als Antwort auf das Belegtsignal das Ausgeben einstellt, ist eine Latenzperiode. Während dieser Wartezeit ist es möglich, dass ein Befehlspaket oder mehrere Befehlspakete zu der Ressource ausgegeben werden könnten und deshalb sollte die Ressource so ausgelegt sein, dass keiner dieser Befehle fallengelassen wird.

[0294] Deshalb könnte der Empfänger, nachdem er das Belegt-Flusssteuersignal aktiviert, noch immer M Nachrichten annehmen, wobei M in der Gleichung 3 unten definiert ist:

Gleichung 3:

M = (Flusssteuerungslatenz in Frame-Takten)/(Paketlänge in Frame-Takten)

[0295] Der Wert von „M" definiert dabei die Anzahl von pro Kanal zugeordneten Schlitzen, die verfügbar ist.

[0296] Im Folgenden Bezug nehmend auf die Fig. 25, wird ein Ausführungsbeispiel bereitgestellt, in dem virtuelle Kanäle implementiert werden, die für jeden Kanal separate Ressourcen nutzen. Teile der zwei Knoten

420 und **424** werden über einen hierarchischen Switch (HS) **422** zusammengekoppelt gezeigt.

[0297] Der globale Port **420** ist gekoppelt, um auf dem Bus **421a** Eingangsdaten von dem hierarchischen Switch **422** zu empfangen und um auf dem Bus **421** Daten zu dem Switch **422** zu übertragen. Gleichermaßen ist der globale Port **424** gekoppelt, um auf dem Bus **423a** Daten zu dem Switch **422** zu übertragen und auf dem Bus **423b** Daten von dem Switch **422** zu empfangen.

[0298] Die Datenbusse **421a**, **421b**, **423a** und **423b** übertragen jeder alle Kanalbefehlstypen. Ein Warteschlangenmechanismus, wie der Warteschlangenmechanismus **425**, ist an jedem Eingangs- und Ausgangsanschluss jeder Ressource bereitgestellt. Der Warteschlangenmechanismus umfasst eine Anzahl von einzelgesteuerten Puffern **425a**–**425e**, wobei jeder Puffer zugeordnet ist, um nur einen Typ von Kanalbefehl zu speichern. Der Puffer **425a** speichert nur Q0-Kanalbefehle, der Puffer **425b** speichert nur Q0Vic-Kanalbefehle usw.

[0299] Während die Befehlspakete an der Schnittstelle jeder Ressource empfangen werden, wird der Befehlstyp analysiert und das Paket wird zu dem adäquaten Puffer weitergeleitet.

[0300] Wenn die Befehlspakete bereit sind, um zu dem adäquaten Prozessor oder IOP des Knotens weitergeleitet zu werden, werden sie von dem adäquaten Puffer ausgewählt und über den Verteilerbus und den QSA (**Fig. 6**) weitergeleitet. Fünf Suchmaschinen, eine für jeden Kanal, sind vorhanden, um die nächste Nachricht für den jeweiligen Kanal zu lokalisieren.

[0301] In dem oben beschriebenen Schema wird jeder Kanal einzeln flussgesteuert und systemdurchgängig ist außer für den niedrigsten Kanal ein Schlitz für jeden Kanal in der Hierarchie reserviert. Dies garantiert, dass ein Kanal nie auf Grund von Ressourcenabhängigkeiten durch einen niedrigeren Kanal blockiert werden kann. Die Bewegung der höheren Kanalnachrichten wird nicht auf Grund von Belegung von Ressourcen durch niedrigere Kanalnachrichten blockiert.

[0302] Das oben beschriebene Schema zum gemeinsamen Nutzen eines physikalischen Puffers zwischen virtuellen Kanälen ist ein einfacheres. Ein fortgeschritteneres Schema wurde zuvor im Zusammenhang mit dem hierarchischen Switch beschrieben.

Virtuelle Kanäle: Regeln für die Zugriffsverteilung und das Kohärenz-Protokoll-Design

[0303] Die Hardwaremechanismen allein sind nicht adäquat, um verklemmungsfreie Nachrichtenübermittlung in dem Kohärenzprotokoll zu garantieren, weil sie sich nur dem Ressourcenabhängigkeitsteil des Problems zuwenden. Um alle ressourcen- und flussabhängige Zyklen zu eliminieren, wird eine Anzahl von zusätzlichen Zugriffsverteilungs- und Kohärenz-Protokoll-Designregeln festgelegt.

[0304] Zuerst sollte der Fortschritt einer Nachricht nicht von dem Fortschritt einer niedrigeren Kanalnachricht abhängig sein, wobei Q2 ein Kanal höherer Ordnung ist und Q0 ein Kanal niedriger Ordnung ist. Die Verteiler sollten die Flusssteuerung jedes Kanals unabhängig von den anderen steuern. Wenn beispielsweise für Q1 ein Belegt-Flusssteuersignal aktiviert ist, jedoch für Q2 nicht, sollten die Verteiler Q2 Fortschritt machen lassen. Alle Suchmaschinen, die verwendet werden, um eine Ressource für ausstehende Befehlspakete zu suchen, müssen dieselbe Eigenschaft sicherstellen.

[0305] Zweitens, jede Ressource, die von zwei oder mehr Kanälen gemeinsam genutzt wird, muss einige zugeordnete Schlitze für jeden der höheren Kanäle inkorporieren und sollte den Kanälen höherer Ordnung ermöglichen, Fortschritt zu machen, wenn Kanäle niedriger Ordnung blockiert sind.

[0306] Drittens, alle Kanalbefehle müssen konsistent arbeiten. Der Endpunkt eines Q0-Befehls ist immer ein Verzeichnis. Der Endpunkt eines Q1-Befehls und eines Q2-Befehls ist immer ein Prozessor. Wenn Transaktionen an einem Endpunkt fortsetzen wollen, müssen Sie sich zu einem höheren Kanal bewegen. Wenn beispielsweise eine Q0-Nachricht ein Verzeichnis erreicht, kann sie keine Q0-Nachrichten generieren, sondern muss Q1- oder Q2-Nachrichten generieren. Deshalb kann eine Nachricht nicht verzweigen oder in eine niedrigere Kanalnachricht konvertieren.

[0307] Für Transaktionen, die sich an anderen Punkten verzweigen, können nur Nachrichten desselben oder eines höheren Kanals erzeugt werden. Wenn beispielsweise eine weitergeleitete Lesen-Modifizieren-Nachricht (eine Q1-Nachricht) eine weitergeleitete Lesen-Modifizieren-, eine Invalidate- und eine Füllen-Modifizieren-Markierungsnachricht erzeugt, sind alle diese Nachrichten Q1-Nachrichten.

[0308] Folglich werden eine Vorrichtung und ein Verfahren zum Bereitstellen von virtuellen Kanälen in entweder einem busbasierten oder einem switchbasierten System bereitgestellt. Durch das Verwenden der virtuellen Kanäle und der oben beschriebenen Ordnungszwänge kann garantiert werden, dass Referenzen, sobald sie durch das Verzeichnis bedient werden, ausgeführt werden. Im Ergebnis werden die aufwendigen Protokolle nach dem Stand der Technik, die NACKS (in denen ein Prozessor dem anderen anzeigt, dass ein Prozess nicht ausgeführt wurde) und wiederholt ausgeführte Operationen eliminiert.

[0309] Obwohl Ausführungsbeispiele mit bis zu fünf unabhängigen Kanälen gezeigt wurden, sollte verstanden werden, dass ein Ausführungsbeispiel der vorliegenden Erfindung nicht auf eine gegebene Anzahl von Kanälen und nicht auf ein symmetrisches Mehrprozesssystem beschränkt ist. Stattdessen sollte die ausgewählte Anzahl von Kanälen die Anzahl sein, die zum Absichern einer kohärenten Kommunikation, die jedem Kanal inhärenten Steuer- und Hardware-Overheads gegeben, erforderlich ist. Das Steuerverfahren für die virtuellen Kanäle und die zugehörige Vorrichtung ermöglichen deshalb in jedem Mehrprozesssystem Hochleistungskommunikation ohne Verklemmung.

Betrieb der Verzeichnisse beim Aufrechterhalten der Kohärenz

[0310] Bis hierhin wurde eine Basiskommunikationsfabrik dargestellt und eine Basissteuerungsstruktur zum Ermöglichen des freien Kommunikationsflusses zwischen Knoten in dem SMP-System wurde bereitgestellt. Der Schlüssel für die Kohärenz ist jedoch das Sicherstellen, dass die frei fließenden Befehle von jedem der Prozessoren in dem System in der richtigen Ordnung „behandelt" werden. Der Mechanismus, der den Serialisierungspunkt für alle Befehle in dem SMP-System bereitstellt, ist das Verzeichnis in jedem Knoten.

[0311] Wie oben beschrieben, greifen alle Q0-Typ-Befehle zuerst auf das Home-Verzeichnis der bestreffenden Speicheradresse zu. Das Sicherstellen, dass für jeden Befehl zuerst auf das Verzeichnis zugegriffen wird, ermöglicht, dass jeder Befehl von einer gemeinsamen Quelle in Reihenfolge gesichtet wird.

[0312] In einem Ausführungsbeispiel der Erfindung ist die Serialisierungsordnung die Ordnung, in der die Q0-Befehle für X auf dem Verteilerbus erscheinen, nachdem sie aus dem Verzeichnis die Zugriffsverteilung für die Adresse X gewonnen haben. Ein Befehl des Ladetyps wird festgelegt, wenn der entsprechende Lesebefehl auf das Home-Verzeichnis zugreift. Ein Befehl des Speichertyps wird festgelegt, wenn entweder der entsprechende Befehl Lesen/Modifizieren auf das Verzeichnis zugreift oder wenn der entsprechende Befehl Clean-to-Dirty auf das Verzeichnis zugreift und auf dem Verteilerbus erscheint.

[0313] Als Beispiel sei vorausgesetzt, das die unten gezeigte Sequenz von zehn Befehlen durch verschiedene Prozessoren (#P) zu einem gemeinsamen Home-Verzeichnis ausgegeben wird, wobei X_i der Teil der Cache-Zeile X ist.

Tabelle 4

1	P1: Speichere X_1 (1)
2	P2: Lade X_1
3	P3: Lade X_1
4	P5: Lade X_1
5	P1: Speichere X_2 (2)
6	P2: Speichere X_1 (3)
7	P4: Lade X_1
8	P5: Lade X_2
9	P6: Lade X_1
10	P2: Speichere X_1 (4)

[0314] Die Version der Cache-Zeile wird im Ergebnis jedes Speichervorgangs aktualisiert. Infolgedessen erzeugt der Befehl eins die Version eins, der Befehl fünf erzeugt die Version zwei, der Befehl sechs erzeugt die Version drei und der Befehl zehn erzeugt die Version vier.

[0315] Die Serialisierungsordnung stellt sicher, dass jede Sequenz von Ereignissen, die das Verzeichnis erreicht; die richtige Version jeder Cache-Zeile X erhält. Beispielsweise sollten die Befehle zwei bis einschließlich vier die Version eins erhalten. Wenn der Befehl fünf des Prozessors P1 die Speicherung durchführt, sollte er alle Invalidate-Nachrichten zu allen Versionen einer Cache-Zeile (in den Prozessoren P2, P3 und P5) senden. Gleichermaßen sollte er, wenn der Befehl sechs des Prozessors P2 X mit den Daten der Version drei aktuali-

siert, die Daten der Version zwei des Prozessors P1 Invalidate unterziehen. Die Prozessoren P4, P6 und P7 erhalten die Daten der Version drei, die später durch die Speicherung des Prozessors P8 der Daten der Version vier Invalidate unterzogen werden.

[0316] Es genügt festzustellen, dass in einem System zu jeder gegebenen Zeit eine Anzahl von Last- und Speichervorgängen für eine gemeinsame Cache-Zeile X im Gang sein kann. Das System behandelt diese Befehle in einer solchen Art und Weise, dass die Lasten und Speicherungen durch das Verzeichnis in einer serialisierten Ordnung weiterverarbeitet werden.

[0317] Zum Sichern der Aufrechterhaltung der System-Serialisierungsordnung und gleichzeitig der Aufrechterhaltung der Datenkohärenz wird eine Anzahl von Techniken verwendet. Diese Techniken umfassen das strikte Ordnen von Q1-Kanalbefehlen, von CTD-Eindeutigkeiten, von Shadow-Befehlen, von Fill Markern und von Victim-Schreibverzögerungspufferung. Im Folgenden wird jede Technik detailliert beschrieben.

Q1-Kanalordnung

[0318] Die erste Maßnahme, die ergriffen wird, um die Kohärenz aufrechtzuerhalten, ist sicherzustellen, dass sich alle Nachrichten, die sich auf dem Q1-Kanal bewegen, d. h. jene, die von dem Verzeichnis gesendet werden, in einer FIFO-Ordnung bewegen. Das bedeutet, dass die Nachrichten des Q1-Typs, die von dem Verzeichnis zu einem anderen Prozessor oder dem IOP weitergeleitet werden, in der Ordnung weitergeleitet werden, in der die Befehle in dem Verzeichnis serialisiert wurden.

[0319] Als Beispiel wird in dem exemplarischen Teilsystem der Fig. 26 vorausgesetzt, dass der erste Prozessor P1 (**431**) in dem Knoten **430** eine Cache-Zeile X dirty in seinem Cache speichert. Der Prozessor P16 (**433**) in dem Knoten **432** gibt einen Befehl X Lesen auf dem Q0-Kanal aus, der zu dem Home-Verzeichnis **437** in dem Knoten **436** weitergeleitet wird. Ebenso gibt der Prozessor P17 in dem Knoten **432** einen Inaval-to-Dirty-Befehl auf dem Kanal Q0 aus, der ebenso zu dem Home-Verzeichnis **437** von X in dem Knoten **436** weitergeleitet wird. Als Antwort auf das Empfangen des Befehls X Lesen wird in Überseinstimmung mit dem Verzeichniseintrag auf dem Q1-Kanal ein weitergeleiteter Befehl X Lesen zu dem Prozessor P1 (**431**) gesendet. Als Antwort auf das Empfangen des IDTs wird in Übereinstimmung mit dem Status des Verzeichniseintrages ein Invalidate-Befehl zu dem hierarchischen Switch **435** gesendet, der die weitergeleiteten Invalidate-Befehle auf dem Kanal Q1 zu dem Prozessor P1 und Prozessor P16 weiterleitet.

[0320] Infolgedessen werden gleichzeitig ein Inval-X-Befehl und ein weitergeleiteter Befehl X Lesen als Q1-Kanalbefehle zu dem P1 gesendet.

[0321] Wenn den Befehlen auf dem Kanal Q1 ermöglicht würde, außerhalb der Reihenfolge ausgeführt zu werden, könnte eintreten, dass Invalidate vor dem Read erfolgt. Als Folge dessen würden die Fill-Daten nicht zu dem Prozessor **16** gesendet werden und alle weiteren Vorgänge würden unberechenbar sein.

[0322] Durch das Halten der Befehle auf dem Q1-Kanal in Ordnung, wird der Read-Befehl durch den P1 vor dem Empfang des Invalidate-Befehls behandelt und die Kohärenz bleibt aufrechterhalten.

[0323] In einem Ausführungsbeispiel der Erfindung wird die FIFO-Ordnung nur für einen Kanal Q1 aufrechterhalten, wobei die FIFO-Ordnung bedeutet, dass alle Nachrichten, die derselben Speicheradresse entsprechen, in der FIFO-Ordnung bleiben. Die vorliegende Erfindung ist jedoch nicht darauf beschränkt, lediglich die Ordnung für den Q1-Kanal aufrechtzuerhalten, sondern kann erweitert werden, um das Aufrechterhalten von Ordnung für jede Kombination von Kanälen zu umfassen.

[0324] Eine Methode zum Implementieren der oben beschriebenen Ordnungsprozedur wird durch den QS-Verteiler **11** in dem QSA-Chip (Fig. 6) durchgeführt. Der QS-Verteiler serialisiert alle Q0-Transaktionen zu dem Speicherraum des Knotens. Im Ergebnis wird ein serieller Fluss von Q1-Paketen erzeugt, der über den globalen Port und den hierarchischen Switch sowohl zu dem lokalen Prozessor in dem Knoten als auch zu Prozessoren, die dezentral zu dem Knoten sind, gerichtet ist.

[0325] Die erste Ordnungsregel sagt Folgendes aus: Alle Q1-Pakete, die durch einen gegebenen QS-Verteiler erzeugt werden, werden in serieller Ordnung erzeugt. Alle Prozessoren, auf die einige oder alle der Q1-Pakete von einem gegebenen QS-Verteiler zielen, empfangen diese Q1-Pakete in der Reihenfolge, in der diese durch den QS-Verteiler generiert wurden.

[0326] Um diese Regel zu abzusichern, hält der QSA-Chip die Ordnung bei allen Q1-Paketen aufrecht, die zu und von einem gekoppelten Prozessor in dem Knoten übertragen werden. Die Logik in dem globalen Port hält die FIFO-Ordnung bei allen Paketen aufrecht, die zwischen dem hierarchischen Switch und dem QSA-Chip übertragen werden. Zusätzlich hält der hierarchische Switch Ordnung bei allen Q1-Paketen von jedem gegebenen Eingang zu jedem gegebenen Ausgang aufrecht.

[0327] Es ist zu beachten, dass diese Regel keine bestimmte Ordnung zwischen Q1-Paketen von einem QS-Verteiler und Q1-Paketen von dem QS-Verteiler eines anderen Knotens zuweist. Die Q1-Pakete, die von anderen Knoten empfangen werden, werden über den hierarchischen Switch mit den Q1-Paketen die durch den Home-Knoten erzeugt werden, wie folgt serialisiert: Alle Q1-Pakete, die auf Prozessoren in Remote-Knoten zielen, werden durch den QS-Verteiler der Remote-Knoten weiterverarbeitet. Diese Q1-Pakete werden durch den hierarchischen Switch mit durch den Remote-Knoten erzeugten Q1-Paketen serialisiert. Alle Empfänger von Q1-Paketen von einem gegebenen QS-Verteiler müssen die Q1-Pakete in derselben Ordnung, in der sie in dem QS-Verteiler serialisiert wurden, empfangen.

[0328] Im Folgenden Bezug nehmend auf die Fig. 27A, wird ein Blockdiagramm gezeigt, dass das Ordnen einer Anzahl von Q0- und Q1-Befehlen, die durch das SMP gemäß den oben beschriebenen Ordnungsrichtlinien weiterverarbeitet werden, dargestellt. Es sei vorausgesetzt, dass der Px in Knoten **440** einen Befehl Q0a ausgibt, Prozessor Py einen Befehl Q0b ausgibt und Prozessor Pz einen Befehl Q0c ausgibt. Gleichzeitig damit empfängt der QS-Verteiler **441** von dem globalen Port **443** Q1-Nachrichten von den Prozessoren Pr und Pq.

[0329] Diese Nachrichten werden wie folgt geordnet: Der QS-Verteiler **441** verarbeitet den Q0a, den Q0b und den Q0c weiter, um Q1a-, Q1b- und Q1c-Antworten zu erzeugen. Diese Erzeugten Q1-Befehle werden mit den eingehenden Q1-Befehlen kombiniert, um einen geordneten Fluss von Befehlen zu dem FIFO **442** zum Weiterleiten zu den lokalen Prozessoren bereitzustellen. Die Ordnung der FIFO-Befehle reflektiert die Ordnung der durch den QS-Verteiler weiterverarbeiteten Befehle.

[0330] Die Q1a-, Q1b- und Q1c-Befehle werden zu dem globalen Port **443** zur Übertragung zu einem Remote-Knoten weitergeleitet. Der Ausgangspuffer **444** des globalen Ports speichert diese Befehle in derselben Ordnung, in der sie durch den QS-Verteiler weiterverarbeitet wurden. Diese Ordnung wird durch den hierarchischen Switch **446** unter Verwendung der Methoden, die oben in Bezug auf die Fig. 14–Fig. 19 beschrieben wurden, aufrechterhalten, wählend die Nachrichten zu der Remote-CPU **454** weitergeleitet werden.

[0331] Die Fig. 27A stellt außerdem weitere Ordnungsrichtlinien, die in dem hierarchischen Switch gelten, dar. Wie zuvor erwähnt, erhält der hierarchische Switch die Ordnung durch das Sicherstellen aufrecht, dass mehreren Paketen, die an einem gegebenen Eingangsport des hierarchischen Switchs erscheinen und die auf einen gemeinsamen Ausgangsport des hierarchischen Switchs zielen, in derselben Ordnung an dem Ausgangsport erscheinen, in der sie an dem Eingangsport erschienen sind.

[0332] Im Folgenden Bezug nehmend auf die Fig. 27B, ist der hierarchische Switch, wie oben beschrieben, für das Multicasting von Eingangsnachrichten, d. h. das Senden eines empfangenen Q1-Pakets zu mehr als einem Zieladressenknoten, verantwortlich. Ein Beispiel eines Pakets, dass durch den Switch massenversendet wird, ist das Invalidate-Paket. Wenn mehrere Pakete, die von verschiedenen hierarchischen Switch-Ports eingegeben werden, zu gemeinsamen Ausgangsports massenversendet werden, sollten die Q1-Pakete in derselben Ordnung an allen Ausgangsports erscheinen. Wenn beispielsweise sowohl Paket eins als auch Paket zwei an dem hierarchischen Switch **460** empfangen wurden, dann ist eine zulässige Methode des Multicastings der zwei Nachrichten zu den Prozessoren **464** und **466**, dass die Nachricht zwei beide Prozessoren vor der Nachricht eins erreicht. Eine weitere zulässige Methode wäre, beide Nachrichtenpakete eins beide Prozessoren vor den Nachrichtpaketen zwei erreichen zu lassen. Jedoch sollten beide Prozessoren die Nachrichten nicht in einer verschiedenen Ordnung empfangen.

[0333] Eine weitere Ordnungsregel, die der hierarchische Switch zu befolgen hat, ist sicherzustellen, dass, wenn geordnete Listen von Q1-Paketen von mehreren Eingangsports auf gemeinsame Ausgangsports zielen, die Q1-Pakete an den Ausgangsports in einer Art und Weise erscheinen, die mit einer einzelnen gemeinsamen Ordnung aller eingehenden Q1-Pakete konsistent ist.

[0334] Beispielsweise wird in der Fig. 27C in dem Eingangsport **461** Paket zwei vor Paket vier empfangen. Gleichermaßen wird in dem Eingangsport **462** Paket eins vor Paket drei empfangen. Die Gesamtordnung dieser Befehle muss zur Vermeidung von Verklemmung aufrechterhalten werden. Eine zulässige Ordnung die

Ausgangspakete bereitzustellen, ist, das Paket drei zuerst zu dem Knoten **464** übertragen zu lassen und Paket eins zuerst zu dem Knoten **466** übertragen zu lassen. Diese Übertragung wird in der Fig. 27C dargestellt. Eine weitere zulässige Ausgabe wäre, die Pakete zwei und vier zuerst durch den Empfängerprozessor empfangen zu lassen. Wenn jedoch ein Prozessor das Paket drei zuerst empfängt und ein weiterer Prozessor zuerst das Paket vier empfängt, dann kann Verklemmung eintreten, während die blockierten Prozessoren den Empfang ihrer weiteren Pakete ihrer Originalsequenz erwarten.

[0335] Deshalb werden Regeln bereitgestellt, die sicherstellen, dass die Ordnung in dem Q1-Kanal aufrechterhalten bleibt. In einem Ausführungsbeispiel der Erfindung ist aus Leistungsgründen erwünscht, zu ermöglichen, die Q0- und Q2-Pakete außerhalb der Ordnung weiterzuverarbeiten. Um Datenkohärenz sicherzustellen, werden mehrere Kohärenzmechanismen bereitgestellt, die im Folgenden beschrieben werden.

Change to Dirty Eindeutigkeit

[0336] Wie oben erwähnt, werden nur die Befehle des Q1-Typs in einer Serialisierungsordnung in dem Verzeichnis definiert. In einem Ausführungsbeispiel der Erfindung sind die Q0- und Q2-Befehle nicht geordnet. Daher werden Sicherheitsmaßnahmen ergriffen, um sicherzustellen, dass in dem Verzeichnis im Ergebnis des relativen Timings der empfangenen Q0- und Q2-Befehle keine Kohärenzprobleme entstehen.

[0337] Ein Kohärenzproblem, das entsteht, resultiert aus der Struktur der Verzeichniseinträge. Wie in der Fig. 9 gezeigt, enthält jeder Verzeichniseintrag ein Besitzerfeld und ein Präsenzbit für jeden Knoten. Das Präsenzbit ist ein Grobvektor, der das Vorhandensein von Daten in einem der vier Prozessoren des zugehörigen Knotens darstellt. Arbeitsschritte durch jeden der vier Prozessoren können dazu führen, dass das Präsenzbit gesetzt wird. Infolgedessen besteht eine bestimmte Mehrdeutigkeit, in welchem Prozessor in dem Knoten das Präsenzbit gesetzt ist. Diese Mehrdeutigkeit kann in bestimmten Fällen zu Kohärenzproblemen führen.

[0338] Im Folgenden als Beispiel Bezug nehmend auf die Fig. 28A und Fig. 28B, wird ein Blockdiagramm von zwei Knoten **470** und **472** gezeigt. Der Knoten **470** (Knoten-ID drei des globalen Systems) enthält die Prozessoren P12, P13, P14 und P15, während der Knoten **472** (Knoten-ID sieben des globalen Systems) die Knoten P28, P29, P30 und P31 enthält.

[0339] Der Status des Verzeichniseintrags für eine gegebene Cache-Zeile X in verschiedenen Zeitperioden T0–T3 ist in der Verzeichnisstatustabelle **455** in der Fig. 28B angegeben. in diesem Beispiel ist der Home-Knoten jeder Cache-Zeile X ein anderer Knoten als der Knoten **470** oder **472**.

[0340] Zu der Zeit T0 ist der Speicher der Besitzer der Cache-Zeile X, wie durch die Besitzer-ID **80** angezeigt. Außerdem speichert zu der Zeit T0 der Prozessor **30** an der Knoten-ID sieben eine Clean-Kopie der Cache-Zeile X.

[0341] Zu der Zeit T1 sendet Prozessor **14** einen Speicherbefehl, der in den Block Read X Mod translatiert ist und zu dem Home-Verzeichnis der Cache-Zeile X weitergeleitet wird. Weil der Speicher Besitzer ist, kann der Prozessor **14** die Daten aus dem Speicher erhalten und wird Besitzer der Cache-Zeile. Zu Knoten sieben wird eine Invalidate-Nachricht übertragen, um die ältere Version der Cache-Zeile X ungültig zu machen, und das Präsenzbit von Knoten sieben wird gelöscht. Zusätzlich setzt Prozessor P14 sein Knotenpräsenzbit **456** (Bit drei). Die Cache-Zeile X wird von dem Home-Speicher zum Speichern und zum Modifizieren an den Prozessor **14** gesendet.

[0342] Zu der Zeit T2 gibt ein weiterer Prozessor, wie zum Beispiel der Prozessor **31**, einen Read-Befehl für Cache-Zeile X aus. Der Lesebefehl erhält die Daten über einen Fill von dem Prozessor P14. Infolgedessen zeigt das Verzeichnis zu der Zeit T2 an, dass sowohl die Knoten-ID drei (Prozessor **14**) als auch die Knoten-ID sieben (Prozessor **31**) eine Kopie der Cache-Zeile X speichern, wie durch die Knotenpräsenzbits **458** und **456** angezeigt.

[0343] Wenn zu einer Zeit T3 durch den Prozessor P30 ein CTD ausgegeben wird, ist der Status der Cache-Zeile X, von verschiedenen Prozessoren in dem System aus gesehen, aus dem folgenden Grund nicht kohärent. Wenn der CTD das Verzeichnis erreicht, liest er den Verzeichniseintrag für X und bestimmt, dass das Präsenzbit **458** für seinen Knoten, Knoten-ID sieben, bereits eingerichtet ist. Im Ergebnis setzt der Prozessor P30 anschließend voraus, dass seine CTD-Anforderung erfolgreich war. Der Prozessor **30** annulliert die Kopie der Cache-Zeile X des Prozessors **14** und aktualisiert das Besitzerfeld in dem Verzeichnis. Diese Handlung kann zu unberechenbaren Ergebnissen führen, da der Prozessor P14 eine aktuellere Version der Daten als

der Prozessor P30 speichert.

[0344] Ein Problem ist, dass der Prozessor **30** noch immer eine veraltete Version der durch den Prozessor **14** erzeugten Cache-Zeile speichert und dass Prozessor **14** mitgeteilt wurde, die aktuellste Version der Daten zu annullieren. Eine derartige Situation könnte ernsthafte Kohärenzprobleme innerhalb des SMP-Systems verursachen.

[0345] Zum Korrigieren des oben beschriebenen Problems sind einige Methoden dienlich. Eine Methode ist, das Präsenzfeld des Verzeichniseintrags zu erweitern, um ein Bit für jeden Prozessor in dem System bereitzustellen. Infolgedessen wird die Auflösung von der Knotenebene in die Prozessorebene geändert. Diese Auflösung würde jedoch die Größe des Verzeichnisses unerwünscht vergrößern.

[0346] Ein Ausführungsbeispiel der Erfindung stellt eine einfachere Methode zum Verhindern des oben beschriebenen Eindeutigkeitsproblems durch das Verlangsamen der CTD-Befehle, wenn eine ausstehende Referenz zu derselben Adresse im Transit für diesen Knoten ist, bereit. Wenn eine ausstehende Anforderung für dieselbe Adresse vorhanden ist, wird der CTD zurückgehalten, bis diese vorhergehende Anforderung zurückgezogen wird. Das Transaktions-Tracking-Verzeichnis (TTT) der Fig. 10 eines gegebenen Knotens wird verwendet, um ausstehende Globalreferenzen für diesen Knoten zu überwachen. Zusätzlich sind Anforderungen, die empfangen werden, nachdem der CTD empfangen wurde, fehlgeschlagen.

[0347] Wie unter Bezugnahme auf die Fig. 10 beschrieben, ist das TTT eine vollständig assoziative, mehrfunktionale Steuerstruktur. Das TTT führt zwei Hauptaufgaben aus. Es speichert die Adressen aller Remote-Referenzen, die durch den ihnen zugehörigen Knoten ausgegeben werden. Infolgedessen speichert das TTT einen Informationseintrag für jeden Remote-Zugriff, der durch einen Knoten ausgegeben wird, bis die Transaktion als ausgeführt betrachtet wird. Zusätzlich stellt das TTT in Reaktion auf Anforderungen zu lokalen Adressen Kohärenzinformationen in Bezug auf die transitiven Kohärenzzustände bereit. Folglich ist das TTT ein Verzeichnis zum Verfolgen des Status der Zugriffe, während diese im Transit sind.

[0348] Andere Prozesssysteme ermöglichen einer Referenz zu jeder gegebenen Cache-Zeile, zu jedem Zeitmoment im Transit zu sein. Darauf folgende Referenzen zu einer Cache-Zeile im Transit werden blockiert, bis die Referenz im Transit ausgeführt ist.

[0349] Im Gegensatz dazu ermöglicht das SMP der vorliegenden Erfindung, auf Grund der Serialisierung der Befehle in dem Verzeichnis und der Kanalordnungsregeln; dass zu jedem gegebenen Zeitpunkt mehrere Referenzen zu derselben Cache-Zeile im Umlauf sind. Im Ergebnis wird die Gesamtleistung des SMP-Systems verbessert.

[0350] Das TTT **522** wird durch die Logik in dem QSA-Chip verwendet, um den Status der Transaktionen festzustellen, die über den globalen Port ausgegeben wurden. Bevor er die Antwort zu dem globalen Port ausgibt, greift der QSA erst auf das TTT zu, um festzustellen, welche Referenzen zu derselben Cache-Zeile ausstehend sind. Eine Referenz steht aus, wenn sie in Reaktion auf die letzte empfangene Transaktion nicht aus dem TTT zurückgezogen wurde.

[0351] Wie eine Referenz aus dem TTT zurückgezogen wird, ist von dem Typ der Referenz, der in dem Befehlsfeld **584** angezeigt wird, abhängig. Beispielsweise erfordern Referenzen Read X, die bis zu dem globalen Port zur Speicherung in dem TTT gekommen sind, dass sowohl das Statusbit Fill Here **588a** als auch das Statusbit Fill Marker Here **588b** zu empfangen sind. (Fill Marker werden unten ausführlicher beschrieben.) Für Referenzen des Statustyps, wie zum Beispiel CTD oder ITD, ist das Setzen des ACK/NACK-Bits **588c** in das TTT ausreichend, um diesen Eintrag zurückzuziehen.

[0352] Im Folgenden Bezug nehmend auf die Fig. 29, stellt ein Flussdiagramm die Verwendung des TTTs zum Eliminieren von mehrdeutigen Verzeichniseinträgen dar. In dem Schritt **500** wird eine Cache-Zeile in dem Speicher in ihrem Home-Knoten gespeichert und der Prozessor **30** des Knotens sieben speichert eine Kopie der Daten. In dem Schritt **502** wird durch den Prozessor P14 ein ReadMod X ausgegeben. Im Ergebnis wird Invalidate zu dem Knoten sieben weitergeleitet. In dem Schritt **504** gibt der Prozessor P31 einen Befehl Rd X aus, der einen Eintrag in dem TTT in dem Knoten sieben mit dem folgenden Status erzeugt.

Adresse	Befehls-ID	Status	

		Fill	Fmark	Shadow	ACK/NACK
X	Lese 31				

[0353] In dem Schritt **506** gibt der Prozessor P30 einen CTD X aus. Der QSA-Chip prüft die Adresse der CTD-Anweisung, bestimmt, dass er ein Remote-CTD ist, und leitet ihn zu dem globalen Port und über den GP-Link zu dem TTT weiter. Der Inhalt des TTTs ist dann wie unten gezeigt.

Adresse	Befehls-ID	Status			
		Fill	Fmark	Shadow	ACK/NACK
X	Lese 30				
X	Lese 31				

[0354] Wie in Bezug auf die Fig. 6 erwähnt, nutzt der globale Port die Informationen aus dem TTT, um festzustellen, welche Befehle zum Senden aus dem hierarchischen Switch heraus zugelassen sind. In einem Ausführungsbeispiel der Erfindung wird der globale Port, wenn das TTT bestimmt, dass ein hängiges Read im Transit ist, vom Weiterleiten der CTD zu dem Switch ausgeschlossen, bis die Leseergebnisse zurückgesendet wurden.

[0355] In dem in dem Flussdiagramm der Fig. 29 beschriebenen Beispiel wird durch das TTT eine ausstehende Leseanforderung zu der Adresse X identifiziert. Im Ergebnis wird in dem Schritt **508** der CTD aufgehalten, bis der Read-Befehl nicht länger ausstehend ist.

[0356] Der Read steht aus, bis sowohl ein Fill als auch ein Fill Marker zu dem Knoten sieben zurückgesendet sind. Während dieser Zeitperiode erreicht die durch den ReadMod in dem Schritt **502** ausgegebene Invalidate-Nachricht den Knoten sieben und aktualisiert die DTAGs des jeweiligen Knotens. Wenn die Invalidate-Nachricht für X das TTT erreicht, markiert das TTT jeden CTD, der in der TTT gehalten wird, als fehlgeschlagen und er wird sofort freigegeben. Wenn in dem Schritt **510** der CTD noch immer in dem TTT ist, wird er über den globalen Port gesendet.

[0357] Dementsprechend können durch das Verwenden des TTTs zum angemessenen Aufhalten von fehlgeschlagenen CTD-Befehlen durch Mehrdeutigkeit der Präsenzbits in dem Verzeichnis verursachte Kohärenzprobleme eliminiert werden.

Fill Marker

[0358] Die meisten Antworten zu einem Prozessor sind in dem Q1-Kanal und werden infolgedessen gemäß den oben beschriebenen Regeln in Ordnung gehalten. Jedoch unterliegen Nachrichten, die auf dem Q2-Kanal empfangen werden, diesem Ordnungszwang nicht. Die Nachrichten des Q2-Typs enthalten Fills und Fill Modifies.

[0359] Weil die Ankunft von Nachrichten des Q2-Typs nicht die Serialisierungsordnung, die im Verzeichnis vorhanden ist, reflektiert, ist in den Antwortdaten eine potenzielle Mehrdeutigkeit vorhanden. Wenn beispielsweise ein Invalidate auf dem Kanal Q1 unterwegs ist und ein FillMod auf dem Q2, sollte es eine Methode zum Bestimmen geben, welcher Arbeitsschritt zuerst zu erfolgen hat, damit die Kohärenz aufrechterhalten werden kann.

[0360] Im Folgenden Bezug nehmend auf die Fig. 30, werden beispielsweise zwei Knoten, **520** und **530**, gezeigt. Es werden nur die Teile der Knoten gezeigt, die für den Zweck der Erklärung gebraucht werden.

[0361] Es sei vorausgesetzt, dass der Prozessor P2 (**524**) und der Prozessor P4 (**534**) eine Kopie der Cache-Zeile X speichern. Der Home-Knoten der Cache-Zeile X ist der Knoten **532**.

[0362] In der folgenden Beschreibung werden die Kanäle, die durch die folgenden Pakete verwendet werden, durch das Verwenden verschiedener Linien angezeigt. Die Q0-Befehle sind durch einfache Pfeillinien ange-

zeigt, Q1-Befehle sind durch doppelte Pfeillinien angezeigt und Q2-Befehle sind durch gestrichelte Pfeillinien angezeigt.

[0363] Vorausgesetzt, der Prozessor P4 gibt einen CTD X aus, um den exklusiven Besitz der Cache-Zeile X zu erhalten. Als Antwort gibt das Verzeichnis **542**, gemäß den Verzeichnispräsenzbits und dem DTAG (nicht gezeigt), einen Invalidate-Befehl zu Knoten **520** aus. Dieser Invalidate wird den DTAG in dem Knoten **520** auf dem Kanal Q1 aktualisieren und einen Invalidate-Probe zu allen Prozessoren (hier Prozessor P2), die eine Kopie haben, senden.

[0364] Der Prozessor P1 gibt dann einen ReadMod X zu dem Home-Verzeichnis **542** von X aus. Wie oben erwähnt, ist X aktuell im Besitz des Prozessors P4 und deshalb wird gemäß dem Kohärenz-Protokoll ein weitergeleiteter ReadMod X zu dem Prozessor P4 weitergeleitet. Als Antwort gibt Prozessor P4 einen FillMod zu Prozessor P1 auf dem Q2-Kanal aus.

[0365] Weil die Kommunikation auf dem Q2-Kanal nicht mit der Q1-Kommunikation serialisiert ist, besteht eine Möglichkeit, dass der Q2-FiflMod den Prozessor P1 vor dem Invalidate von CTD X den Knoten **520** erreicht. Der Effekt würde sein, dass gültige Daten in den Cache des P1 geschrieben werden würden, jedoch bald danach die DTAGs eingerichtet werden würden, um jede Kopie von X in dem Knoten zu annullieren, und ein Invalidate würde zu P2 und zu P1 gesendet werden. Der Invalidate entspricht jedoch nur der Version in P2 und nicht der späteren in P1. Das System würde jetzt in einem nicht kohärenten Zustand sein. Das Verzeichnis **544** zeichnet P1 als den Besitzer auf, obwohl P1 Invalidate unterzogen wurde.

[0366] Ein Ausführungsbeispiel der Erfindung überwindet dieses Problem durch Verwenden von Fill Markern und des TTTs (Fig. 10) in dem globalen Port jedes Knotens.

[0367] Ein Fill Marker oder ein Fill Marker Mod ist ein Paket, das in Reaktion auf eine Anforderung Read oder ReadMod für Daten, die aktuell nicht in dem Speicher in dem Home-Knoten gespeichert sind, erzeugt wird. Das bedeutet, der Fill Marker oder der Fill Marker Mod wird gleichzeitig wie der weitergeleitete Read oder der weitergeleitete Read Mod erzeugt. Infolgedessen sind Fill Marker und Fill Marker Mods Q1-Kanalbefehle. Während der weitergeleitete Read oder weitergeleitete Read Mod zu dem Prozessor, der eine Cache-Zeile speichert, weitergeleitet werden, ist die Zieladresse des Fill Markers oder Fill Marker Mods der Prozessor, von dem der ursprüngliche Read oder Read Mod stammt.

[0368] Die Fill Marker ermöglichen dem Absenderprozessor, die Serialisierungsordnung, die in dem Verzeichnis eintritt, festzustellen. Im Folgenden Bezug nehmend auf die Fig. 31, hilft die Anwendung von Fill Markern dem oben dargestellten Problem wie folgt ab. Wie zuvor sei angenommen, dass der Prozessor **53A** einen CTD X zu dem Home-Verzeichnis von X ausgibt, was dazu führt, dass ein Invalidate **550** auf dem Q1-Kanal zu dem Knoten **520** gesendet wird.

[0369] Wenn der Prozessor P1 (**522**) den Read Mod X zu dem Remote-Verzeichnis ausgibt, wird für diese Anforderung ein TTT-Eintrag generiert. Ein beispielhafter TTT-Eintrg für diese Anforderung ist in der Fig. 32 gezeigt. Es ist zu beachten, dass der TTT-Eintrag kein Statusbit Fill Here und Fill Marker Here enthält. Jedes dieser Bits wird in Reaktion auf das repräsentative Paket, das an dem globalen Port des Knotens **520** empfangen wird, gesetzt. Der TTT-Eintrag wird nicht gelöscht, bis sowohl der Fill als auch der Fill Marker zurückgesendet sind.

[0370] Wieder Bezug auf die Fig. 31 nehmend, wird der Read Mod X von dem Prozessor **522** in einem Frd-Mod X zu dem Prozessor **53A** resultieren. Gleichzeitig wird auf dem Kanal Q1 ein Fill Marker Mod X **552** zurück zu dem Prozessor P1 weitergeleitet. Sowohl der Invalidate als auch der Fill Mod Marker sind auf demselben Q1-Kanal.

[0371] Vorausgesetzt, der Fill Mod **554** auf dem Kanal Q2 erreicht den Knoten **520** vor dem Invalidate. In Reaktion auf das Zurücksenden entweder des Fill Mods oder des Fill Mod Markers wird der Duplicate-Tag-Status in globalen Referenzen aktualisiert. Infolgedessen veranlasst der Fill Mod, dass der DTAG-Status für X aktualisiert wird, um den Besitz von X durch den Prozessor **1** zu reflektieren.

[0372] Angenommen, der Invalidate **550** ist der nächste Befehl, der den Knoten **520** erreicht. Auf das TTT wird zugegriffen, um den Status des weitergeleiteten Befehls Read festzustellen. An diesem Punkt hat das TTT das Bit Fill Here gesetzt, das Bit Fill Marker ist jedoch nicht gesetzt. Infolgedessen stellt das TTT eine Anzeige in Bezug auf das relative Timing des Invalidate-Vorgangs und den Vorgang des Remote Reads bereit. Auf

Grund der Serialisierung der Q1-Befehle kann abgeleitet werden, dass der Invalidate zeitlich vor dem RdMod X von dem Prozessor **522** in dem Verzeichnis erzeugt wurde und infolgedessen der Fill Mod eine neuere Version ist und der Invalidate nicht für die Kopie der Daten des Prozessors **522** gilt. Im Ergebnis wird der DTAG-Eintrag für den Prozessor **1** nicht annulliert.

[0373] Obwohl das Ausführungsbeispiel oben das TTT als in dem globalen Port vorhanden zeigt, könnte gemäß einem anderen Ausführungsbeispiel jeder der Prozessoren jeder der Knoten den Status von Remote-Anforderungen zu gemeinsamen Adressen durch das Überwachen der Anforderungen in dem Verzeichnis verfolgen. Danach würden die Fill Marker durch das Verzeichnis zu den zugehörigen Prozessoren weitergeleitet werden, anstatt lediglich zu dem TTT weitergeleitet zu werden.

[0374] Folglich ist klar, dass das TTT zwei Zwecken dienen kann. Durch das Überwachen der Befehlstypen, die aus den Mehrprozessorknoten gesendet werden, kann das TTT das Weiterleiten bestimmter Befehle (wie zum Beispiel des CTDs) verbieten, bis weitere Befehle zu derselben Adresse ausgeführt sind. Zusätzlich kann das TTT, durch das Bereitstellen eines Markierungsmechanismus, der dem TTT anzeigt, wann eine Anforderung in den Q2-Kanal übergegangen ist (wie der Fill Marker), verwendet werden, um eine Anzeige des relativen Timings zwischen Befehlen, die auf verschiedenen Kanälen zurückgesendet wurden (d. h. Q2-Fill- und Q1-Befehle), bereitzustellen, und kann dementsprechend Befehle ausschließen, die den Speicher beim Weiterleiten zu einem Prozessor korrumpieren könnten.

Shadow-Befehle

[0375] Wie aus der bisherigen Beschreibung offensichtlich, sind lokale Zugriffe typischerweise wesentlich schneller als Remote-Zugriffe. Infolgedessen wird in dem SMP-System sowohl das Erfolgen lokaler Zugriffe als auch von Remote-Zugriffen gleichzeitig zugelassen. Jedoch kann in einigen Fällen der Vorgang eines lokalen Zugriffs Verklemmungsprobleme für einen Remote-Zugriff verursachen. Im Folgenden Bezug nehmend auf die **Fig. 33A**, sei beispielsweise vorausgesetzt, dass ein Prozessor **562** einen Rd X zu einer Cache-Zeile X ausgibt. Der Home-Knoten der Cache-Zeile X ist der Knoten **560**. Das Verzeichnis in dem Knoten **560** zeigt an, dass der Prozessor **582** aktuell die Cache-Zeile X besitzt. Infolgedessen wird der weitergeleitete Rd X zu dem Prozessor **582** gesendet.

[0376] Danach sei vorausgesetzt, dass der Prozessor **564** in dem Knoten **560** einen CTD X ausgibt. Wie zuvor erwähnt, ist die Cache-Zeile X zu dem Knoten **560** lokal und wenn der CTD erfolgreich ist, leitet er einen Invalidate zu dem Prozessor P1 (und außerdem zu dem gezeigten Prozessor P5). Unter kurzer Bezugnahme auf die **Fig. 33B** enthält, wie im Einzelnen in der mitanhängigen Anmeldung mit dem Titel Distributed Data Dependency Stall Mechanism, Anwaltnummer PD96-0149, von VanDoren u. a., mit demselben Datum wie die vorliegende beantragt und hierin durch Bezugnahme einbezogen, jeder der Prozessoren, wie der Prozessor P1, Logik zum Abwürgen von Probes zu einem Cache, wenn für denselben Cache-Ort ein ausstehender Lesebefehl vorhanden ist. Das Beispiel oben gegeben, würde der Effekt des Reads X sein, die Adresse X in der MAF (Miss Address File) **574** zu speichern. Die Inhalte der MAF werden mit eingehenden Probes verglichen und wenn eine Übereinstimmung zwischen der Adresse eines eingehenden Probes und der MAF vorhanden ist, wird die Probe-Warteschlange abgewürgt.

[0377] Die Probe-Warteschlange wird freigegeben, wenn die Fill-Daten von dem Prozessor **582** zurückgesendet werden. Wenn jedoch Transaktionen desselben Typs (d. h. Durchführen eines Remote-Reads Y und Ausgeben eines VTDs Y durch P6) in dem Knoten **580** erfolgen, kann die Probe-Warteschlange des Prozessors P5 hängig der Erfüllung der Anforderung Read Y abgewürgt werden.

[0378] Wenn die P5-Probe-Warteschlange mit dem weitergeleiteten Read X von Prozessor P1 hinter dem durch P6 erzeugten Invalidate zu derselben Zeit abgewürgt wird, zu der die P1-Probe-Warteschlange mit dem weitergeleiteten Read Y von P5 hinter dem durch P2 erzeugten Invalidate abgewürgt wird, kann Verklemmung eintreten.

[0379] Es sind einige Strategien vorhandenen, die dazu dienen, um dieses Verklemmungsproblem zu verhindern. Erstens können alle Referenzen dezentral gemacht werden, d. h., alle Referenzen (selbst die des Home-Knotens) können zu dem Switch weitergeleitet werden, bevor sie zu dem Home-Knoten weitergeleitet werden. Wenn alle Referenzen dezentral gemacht werden, dann würde, den oben dargelegten zentralen Ordnungsregeln entsprechend, keine Verklemmungssituation eintreten. Eine zweite Lösung ist, alle Referenzen zu einer gegebenen Cache-Zeile abzuwürgen, sobald eine Referenz zu dieser Cache-Zeile dezentral gesendet wird. Diese Lösungen wirken sich jedoch drastisch auf die Leistung von vorhergehenden lokalen Operationen

aus und werden deshalb nicht bevorzugt.

[0380] Ein Ausführungsbeispiel der Erfindung überwindet das Verklemmungsproblem, das durch die Vermischung von lokalen Referenzen und Remote-Referenzen aufgeworfen wird, durch das Verwenden von Befehls-Shadowing. Sobald eine lokale Referenz zu einer Cache-Zeile X zu einem Remote-Prozessor weitergeleitet ist, werden anschließend alle folgenden Referenzen zu dieser Cache-Zeile dezentral zu dem hierarchischen Switch weitergeleitet, um zentral geordnet zu werden, bis die lokalen Referenzen und alle darauf folgenden Referenzen dieser Cache-Zeile ausgeführt wurden. Infolgedessen veranlasst jede frühere Referenz zu einer Cache-Zeile, die noch immer beschattet wird, dass die gegenwärtige Referenz zu der Cache-Zeile ebenso beschattet wird.

[0381] Im Folgenden Bezug nehmend auf die Fig. 34 und Fig. 35, wird das Beispiel oben mit der Verwendung von Shadow-Befehlen beschrieben. Die Fig. 35 stellt die Inhalte des TTTs für dieses Beispiel dar. Zuerst gibt der Prozessor P1 einen Rd X zu dem Verteiler aus. Dies resultiert, wie zuvor, in einen Frd X zu Prozessor P5, der in der TTT aufgezeichnet wird. Darauf folgend gibt Prozessor P2 einen CTD X zu dem Verteiler aus. Der Verteiler untersucht das TTT, stellt fest, dass ein ausstehender lokaler Read zu einem Remote-Prozessor weitergeleitet ist und leitet den Invalidate X aus dem globalen Port zu dem Prozessor P5 weiter. Um diesen Vorgang zu reflektieren, wird außerdem ein Eintrag in dem TTT erzeugt, wobei ein Shadow-Bit gesetzt wird.

[0382] Gleichzeitig findet in dem Knoten **580** eine gleichartige Serie von Transaktionen statt. Der Prozessor P5 gibt einen Rd Y aus, der zu dem Knoten **560** weitergeleitet wird und durch Enthalten der P5-Adresse in dem TTT angemeldet wird. Der Prozessor P6 gibt darauf folgend einen CTD Y aus. Der Verteiler in dem Knoten **580** vergleicht die CTD-Adresse mit dem ausstehenden Read in dem TTT und „beschattet" den CTD Y über den globalen Port. Für den CTD Y wird in dem TTT ein Eintrag erzeugt, wobei der Eintrag sein Shadow-Bit in dem TTT setzt, anzeigend, dass der CTD Y eine lokale Referenz war, die dezentral weitergeleitet wurde, um die richtige Ordnung der Anforderungen zu Y sicherzustellen.

[0383] Wie oben beschrieben, ist dann ein Problem vorhanden, wenn in beiden Knoten der Frd in der Probe-Warteschlange hinter dem Invalidate ist. Weil die Invalidate-Befehle jetzt zentral geordnet sind, weil sie an einem gemeinsamen Punkt serialisiert werden, d. h. in dem hierarchischen Switch, kann nicht eintreten, dass beide invalidate-Befehle nicht vor den beiden weitergeleiteten Read-Befehlen zu ihren Probe-Warteschlangen weitergeleitet werden können. Im Folgenden Bezug nehmend auf die Fig. 36, wird die Eingangssequenz von Befehlen, die in den hierarchischen Switch **568** eingegeben werden, gezeigt. Die zulässigen Ausgangsserialisierungsordnungen sind als die Ordnungen a–f definiert. Es ist zu beachten, dass gemäß den Ordnungsregeln des Q1-Kanals die Serialisierungsordnung des Eingangs der Pakete in den hierarchischen Switch an dem Ausgang des hierarchischen Switchs aufrechterhalten bleibt. Deshalb geht in dem obigen Fall der Frd dem zugehörigen Invalidate-Befehl voraus, während sie zu dem Zieladressknoten übertragen werden.

[0384] Einer der Knoten kann noch immer einen Invalidate, gefolgt von dem weitergeleiteten Read-Befehl, in der Probe-Warteschlange empfangen. Beispielsweise kann unter Verwendung der Serialisierungsordnung die Probe-Warteschlange des Prozessors P5 durch den Invalidate Y abgewürgt werden und der Frd X kann Fill-hängig abgewürgt werden.

[0385] Es ist jedoch zu beachten, dass in diesem Beispiel der Frd Y nicht hinter dem Invalidate X ist und deshalb in der Lage ist, Fill-Daten zum Freimachen der Probe-Warteschlange des P5 bereitzustellen.

[0386] Wenn Daten für eine Remote-Referenz zurückgesendet werden, wird der der Referenz entsprechende TTT-Eintrag fallen gelassen. Es können andere Referenzen in dem TTT, das die Originalreferenz beschattet, sein. Während diese Befehle von dem hierarchischen Switch empfangen werden, werden die TTT-Einträge für jeden der beschatteten Befehle ebenso fallen gelassen. Schließlich, wenn alle Remote-Zugriffe und alle beschatteten Zugriffe ausgeführt sind, müssen alle darauf folgenden lokalen Referenzen zu dieser Cache-Zeile nicht mehr beschattet werden.

[0387] Dementsprechend können durch die Verwendung von Shadow-Befehlen ressourcenabhängige Verklemmungen, die aus der Koexistenz von lokalen Befehlen und Remote-Befehlen resultieren, ohne eine wesentliche Vergrößerung des Hardwareaufwands eliminiert werden. Es sollte beachtet werden, dass, obwohl das obige Beispiel die Verwendung von weitergeleiteten Read- und CTD-Befehlen involviert, die Shadow-Befehlsmethode gleichermaßen auf andere Befehlstypen anwendbar ist. Wann immer eine Referenz zu einer lokalen Adresse X und eine frühere Nachricht zu der Adresse X zu einem Remote-Prozessor weitergeleitet wurde (wie durch das TTT angezeigt) oder eine frühere Referenz zu X noch immer beschattet wird, wird die ge-

genwärtige Referenz ebenso beschattet.

[0388] Zusätzlich kann die Methode in anderen Architekturtypen angewendet werden, die sogar mehr Hierarchiestufen enthalten, als die einfache Mehrprozessoren-Switch-Hierarchie, die oben beschrieben wurde. Beispielsweise kann die Methode oben für Computersysteme, die mehrere Hierarchieebenen enthalten, verwendet werden, wobei die Befehle in Abhängigkeit von der Hierarchieebene einer vorhergehenden ausstehenden Referenz zu der Cache-Zeile zu der adäquaten Hierarchiestufe weitergeleitet werden.

[0389] Somit wurden eine Architektur und ein Kohärenzprotokoll zur Verwendung in einem großen SMP-System beschrieben. Die Architektur des SMP-Systems enthält eine hierarchische Switch-Struktur, die ermöglicht, dass eine Anzahl von Mehrprozessorknoten mit dem Switch gekoppelt wird, um mit einer optimalen Leistung zu arbeiten. In jedem Mehrprozessorknoten wird ein Simultanpufferungssystem bereitgestellt, das allen Prozessoren des Mehrprozessorknotens ermöglicht, mit Spitzenleistung zu arbeiten. Die Knoten nutzen einen Speicher gemeinsam, wobei sich ein Teil des Speichers in jedem der Mehrprozessorknoten befindet.

[0390] Jeder der Mehrprozessorknoten enthält eine Anzahl von Elementen zum Aufrechterhalten der Speicherkohärenz, einschließlich eines Victim-Caches, eines Verzeichnisses und einer Transaktions-Tracking-Tabelle. Der Victim-Cache ermöglicht das selektive Aktualisieren von Victim-Daten, die für Speicher, die in einem dezentralen Mehrprozessorknoten gespeichert sind, bestimmt sind, und verbessert dadurch die Gesamtleistung des Speichers. Ebenso wird die Speicherleistung dadurch zusätzlich verbessert, dass jeder Speicher einen Schreibverzögerungspuffer enthält, der in Verbindung mit dem Verzeichnis verwendet wird, um Victims, die in den Speicher zu schreiben sind, zu identifizieren.

[0391] Ein mit dem Verzeichnis jedes Knotens gekoppelter Verteilerbus stellt einen zentralen Ordnungspunkt für alle Nachrichten dar, die durch das SMP-System übertragen werden.

[0392] Gemäß einem Ausführungsbeispiel der Erfindung umfassen die Nachrichten eine Anzahl von Transaktionen und jede Transaktion wird abhängig von der Weiterverarbeitungsphase der Nachricht einer Anzahl von verschiedenen virtuellen Kanälen zugewiesen. Die Verwendung der virtuellen Kanäle sichert infolgedessen, durch das Bereitstellen einer einfachen Methode zum Aufrechterhalten der Systemordnung, das Aufrechterhalten der Datenkohärenz. Durch die Verwendung der virtuellen Kanäle und der Verzeichnisstruktur werden Cache-Kohärenzprobleme, die früher zu Verklemmung führten, vermieden.

Patentansprüche

1. Computersystem, umfassend eine Vielzahl von gekoppelten Multiprozessorknoten (**10**), jeder der Knoten umfasst wenigstens einen Prozessor (**12a**) und einen Teil eines gemeinsamen Speichers (**13**), das Computersystem ist gekennzeichnet durch:
einen Tracking-Mechanismus (**122**), verbunden mit der Vielzahl von Prozessoren in jedem der Vielzahl von Multiprozessorknoten (**10**), zum Bestimmen einer Position einer Anforderung zu einer Adresse eines dezentralen Teils eines gemeinsamen Speichers (**13**), ausgegeben von wenigstens einem der Prozessoren (**12a**) in dem einen der Vielzahl von Multiprozessorknoten (**10**), relativ zu einer Vielzahl von anderen Anforderungen, die durch den wenigstens einen der Prozessoren (**12a**) in der Vielzahl von gekoppelten Multiprozessorknoten (**10**) zu der Adresse ausgegeben wurde.

2. Computersystem nach Anspruch 1, des Weiteren an jedem der Vielzahl von Multiprozessorknoten (**10**) umfassend:
einen Serialisierungspunkt (**130**), gekoppelt an den Tracking-Mechanismus (**122**), zum Bereitstellen einer Eingangsreihenfolge von Anforderungen zu dem Teil des gemeinsamen Speichers (**13**) an dem entsprechenden Multiprozessorknoten (**10**).

3. Computersystem nach Anspruch 2, wobei jeder der Prozessoren (**12a**) von jeder der Vielzahl von Multiprozessorknoten eine Vielzahl von Anforderungen ausgibt, jede der Anforderungen eine Vielzahl von Transaktionen umfasst, wobei ein entsprechender einer Vielzahl von Kanälen (Q2, Q1, Q0, Q0Vic und QIO) jeder der Vielzahl von Transaktionen jeder der Vielzahl von Anforderungen zugewiesen ist und wobei ein erster der Kanäle zum Übertragen von Transaktionen zu dem Serialisierungspunkt (**130**) ist.

4. Computersystem nach Anspruch 3, wobei ein zweiter der Kanäle zum Übertragen der von dem Serialisierungspunkt (**130**) ausgegebenen Anforderungen ist und wobei Transaktionen zu gemeinsamen Adressen auf wenigstens dem zweiten der Kanäle streng geordnet sind.

5. Computersystem nach Anspruch 4, wobei der Tracking-Mechanismus (**122**) eine Referenzreihenfolge bestimmt, wie durch die zugehörige Transaktion auf dem wenigstens einen geordneten Kanal angezeigt, so dass die Reihenfolge für Transaktionen auf anderen der Kanäle (Q2, Q1, Q0, Q0Vic und QIO) rekonstruiert werden kann.

6. Computersystem nach Anspruch 5, umfassend des Weiteren für jede der Anforderungen, die eine dritte Transaktion enthält, eine Einrichtung, die dem Tracking-Mechanismus (**122**) des mit dem gemeinsamen Teil des Speichers (**13**) verbundenen Multiprozessorknotens, der der Anforderung entspricht, anzeigt, dass die Anforderung eine dritte Transaktion aufweist.

7. Computersystem nach Anspruch 6, wobei die Einrichtung zum Anzeigen einen Befehl umfasst, der auf dem wenigstens einen geordneten Kanal zu dem Tracking-Mechanismus (**122**) ausgegeben wird.

8. Computersystem nach Anspruch 7, wobei der Tracking-Mechanismus (**122**) des Weiteren eine Vielzahl von Eintragungen enthält, jede der Eintragungen zum Speichern einer Adresse einer Referenz, die einen Speicherort auf einem anderen der Multiprozessorknoten adressiert, und jeder der Einträge des Weiteren eine Vielzahl von Statusbits (**158**) zum Anzeigen des Status der zugehörigen Anforderung enthält.

9. Computersystem nach Anspruch 8, wobei die Statusbits (**158**) des Weiteren umfassen:
ein erstes Bitset zum Anzeigen, ob der Befehl auf dem geordneten Kanal, der anzeigt, dass die Anforderung eine dritte Transaktion hat, zu dem Multiprozessorknoten (**10**) zurückgesendet wurde.

10. Computersystem nach Anspruch 9, wobei die Statusbits (**158**) des Weiteren enthalten:
ein zweites Bitset zum Anzeigen, ob die dritte Transaktion zu dem Multiprozessorknoten zurückgesendet ist,
eine Einrichtung zum Beseitigen eines Eintrags, der sowohl das erste Bit als auch das zweite Bit des Statusbitsets aufweist, aus dem Transaktionsverzeichnis.

11. Computersystem nach Anspruch 10, des Weiteren umfassend:
eine Einrichtung zum Ignorieren einer Anforderung, die zu einer in dem Tracking-Mechanismus (**122**) gespeicherten Adresse ausgegeben wird, die empfangen wird, bevor das erste Bit in dem Tracking-Mechanismus (**122**), das der Adresse entspricht, eingerichtet ist, um den Empfang des Befehls auf dem geordneten Kanal anzuzeigen.

12. Computersystem nach Anspruch 11, wobei die ignorierte Anforderung eine ungültige Anforderung ist.

13. Computersystem nach Anspruch 11, wobei die Einrichtung zum Ignorieren der zu der Adresse ausgegebenen Anforderung die Anforderung nur dann ignoriert, wenn der Prozessor (**12a**), der die Anforderung ausgegeben hat, dem Prozessor (**12a**) entspricht, der veranlasst hat, dass die Adresse in den Tracking-Mechanismus (**122**) eingegeben wurde.

14. Computersystem nach Anspruch 10, des Weiteren umfassend:
eine Einrichtung zum Verzögern einer Referenz, die zu einer in dem Tracking-Mechanismus (**122**) gespeicherten Adresse ausgegeben wurde, bis der Befehl auf dem geordneten Kanal empfangen wird, wobei die Referenz vor dem ersten Bit in dem Tracking-Mechanismus (**122**), das der Adresse entspricht, eingerichtet ist.

15. Computersystem nach Anspruch 14, wobei die Referenz ferner verzögert wird, bis eine erwünschte Version der Daten, die der Adresse zugehörig sind, zu dem Multiprozessorknoten zurückgesendet ist.

16. Computersystem nach Anspruch 13, wobei die Referenz ferner verzögert wird, bis eine erwünschte Version der Daten, die der Adresse zugehörig sind, zu einem der Vielzahl der Prozessoren (**12a**), der veranlasst hat, dass die Adresse in den Tracking-Mechanismus (**122**) eingetragen wurde, zurückgesendet ist.

17. Verfahren zum Ordnen der Reihenfolge zwischen einer Vielzahl von Anforderungen, die zu einer gemeinsamen Adresse in einem Multiprozessor-Computersystem ausgegeben werden, das Multiprozessor-Computersystem umfasst eine Vielzahl von Multiprozessorknoten (**10**), die über einen Switch gekoppelt sind, jeder der Multiprozessorknoten (**10**) umfasst wenigstens einen Prozessor (**12a**) und einen Teil eines gemeinsamen Speichers (**13**), das Verfahren ist durch den folgenden Schritt gekennzeichnet:
Führen eines Verzeichnisses von Adressen der Anforderungen, die von jedem der Multiprozessorknoten (**10**) zu dem Switch weitergeleitet wurden, um eine relative Reihenfolge der Anforderungen zu den jeweiligen Adressen in einem Teil des gemeinsamen Speichers (**13**) eines dezentralen Multiprozessorknotens zu bestim-

men, wobei eine Adresse in dem Verzeichnis geführt wird, bis die mit der Adresse verbundene Anforderung erfüllt ist.

18. Verfahren nach Anspruch 17, wobei jeder der Vielzahl von Multiprozessorknoten einen Serialisierungspunkt (**130**) zum Bereitstellen einer Eingangsreihenfolge von Anforderungen zu dem Teil des gemeinsamen Speichers (**13**) an dem entsprechenden Multiprozessorknoten umfasst.

19. Verfahren nach Anspruch 18, des Weiteren die folgenden Schritte umfassend: jeder wenigstens einer der Prozessoren (**12a**) jeder der Vielzahl von Multiprozessorknoten gibt eine Vielzahl von Anforderungen aus, jede der Anforderungen umfasst eine Vielzahl von Transaktionen, jede der Vielzahl der Transaktionen wird auf einem entsprechenden einer Vielzahl von Kanälen (Q2, Q1, Q0, Q0Vic und QIO) übertragen, wobei ein erster der Vielzahl von Kanälen (Q2, Q1, Q0, Q0Vic und QIO) die Transaktionen zu dem Serialisierungspunkt (**130**) überträgt.

20. Verfahren nach Anspruch 19, wobei ein zweiter der Vielzahl von Kanälen (Q2, Q1, Q0, Q0Vic und QIO) Transaktionen, die von dem Serialisierungspunkt (**130**) ausgegeben wurden, überträgt und wobei die Transaktionen zu gemeinsamen Adressen auf dem zweiten der Vielzahl von Kanälen (Q2, Q1, Q0, Q0Vic und QIO) streng geordnet sind.

21. Verfahren nach Anspruch 20, wobei das Adressenverzeichnis der Anforderungen eine Referenzreihenfolge bestimmt, wie durch die zugehörigen Transaktionen auf dem wenigstens einen geordneten Kanal angegeben, so dass die Reihenfolge für Transaktionen in den anderen der Kanäle (Q2, Q1, Q0, Q0Vic und QIO) rekonstruiert werden kann.

22. Verfahren nach Anspruch 21, des Weiteren den folgenden Schritt enthaltend:
für jede der Anforderungen, die eine dritte Transaktion enthält, dem Adressenverzeichnis in dem Multiprozessorknoten, der mit dem gemeinsamen Teil des Speichers (**13**), der der Anforderung entspricht, verbunden ist, anzeigen, dass die Anforderung eine dritte Transaktion hat.

23. Verfahren nach Anspruch 22, wobei der Schritt des Anzeigens des Weiteren einen Schritt des Ausgebens eines Befehls, der auf dem wenigstens einen geordneten Kanal zu dem Adressenverzeichnis ausgegeben wird, umfasst.

24. Verfahren nach Anspruch 23, wobei das Adressenverzeichnis eine Vielzahl von Eintragungen, jede der Eintragungen zum Speichern einer Adresse einer Referenz, die einen Speicherort auf einem anderen der Multiprozessorknoten adressiert, und eine Vielzahl von Statusbits (**158**) zum Anzeigen des Status der zugehörigen Anforderung enthält.

25. Verfahren nach Anspruch 24, wobei die Statusbits (**158**) des Weiteren umfassen:
ein erstes Bitset zum Anzeigen, ob der Befehl auf dem geordneten Kanal, der anzeigt, dass die Anforderung eine dritte Transaktion hat, zu dem Multiprozessorknoten zurückgesendet wurde.

26. Verfahren nach Anspruch 25, wobei die Statusbits (**158**) des Weiteren umfassen:
ein zweites Bitset zum Anzeigen, ob die dritte Transaktion zu dem Multiprozessorknoten zurückgesendet ist, und wobei das Verfahren den Schritt des Beseitigens eines Eintrags, der sowohl das erste Bit als auch das zweite Bit des Statusbitsets aufweist, aus dem Transaktionsverzeichnis enthält.

27. Verfahren nach Anspruch 26, des Weiteren die folgenden Schritte umfassend:
Ignorieren einer Anforderung, die zu einer in dem Adressenverzeichnis gespeicherten Adresse ausgegeben wird, die empfangen wird, bevor das erste Bit in dem Adressenverzeichnis, das der Adresse entspricht, eingerichtet ist, um den Empfang des Befehls auf dem geordneten Kanal anzuzeigen.

28. Verfahren nach Anspruch 27, wobei die ignorierte Anforderung eine ungültige Anforderung ist.

29. Verfahren nach Anspruch 28, wobei der Schritt des Ignorierens der zu der Adresse ausgegebenen Anforderung die Anforderung nur dann ignoriert, wenn der Prozessor (**12a**), der die Anforderung ausgegeben hat, dem Prozessor (**12a**) entspricht, der veranlasst hat, dass die Adresse in das Adressenverzeichnis eingegeben wurde.

30. Verfahren nach Anspruch 29, des Weiteren den folgenden Schritt umfassend:

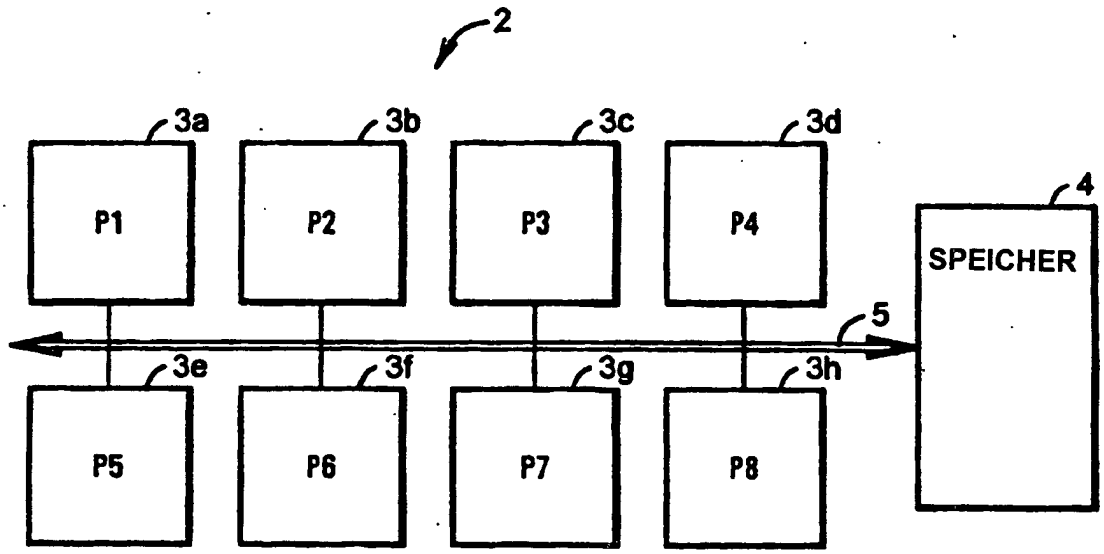
Verzögern einer Referenz, die zu einer in dem Adressenverzeichnis gespeicherten Adresse ausgegeben wurde, die empfangen wird, bevor das erste Bit in dem Tracking-Mechanismus (**122**), das der Adresse entspricht, eingerichtet ist, bis der Befehl auf dem geordneten Kanal empfangen wird.

31. Verfahren nach Anspruch 30, wobei die Referenz ferner verzögert wird, bis eine erwünschte Version der Daten, die der Adresse zugehörig sind, zu dem Multiprozessorknoten zurückgesendet ist.

32. Verfahren nach Anspruch 31, wobei die Referenz ferner verzögert wird, bis eine erwünschte Version der Daten, die der Adresse zugehörig sind, zu einem der Vielzahl der Prozessoren (**21a**), der veranlasst hat, dass die Adresse in das Adressenverzeichnis eingetragen wurde, zurückgesendet ist.

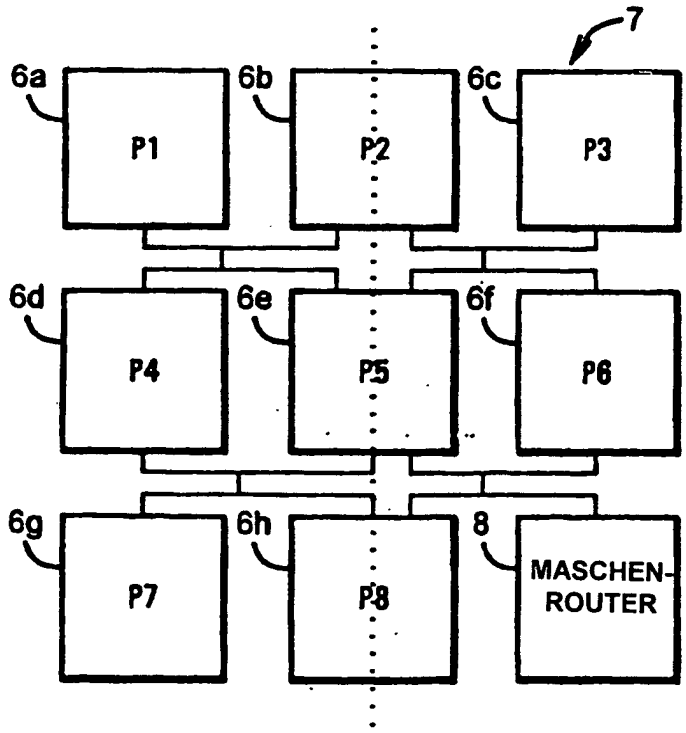
Es folgen 37 Blatt Zeichnungen

Anhängende Zeichnungen



(STAND DER TECHNIK)

FIG. 1A



(STAND DER TECHNIK)

FIG. 1B

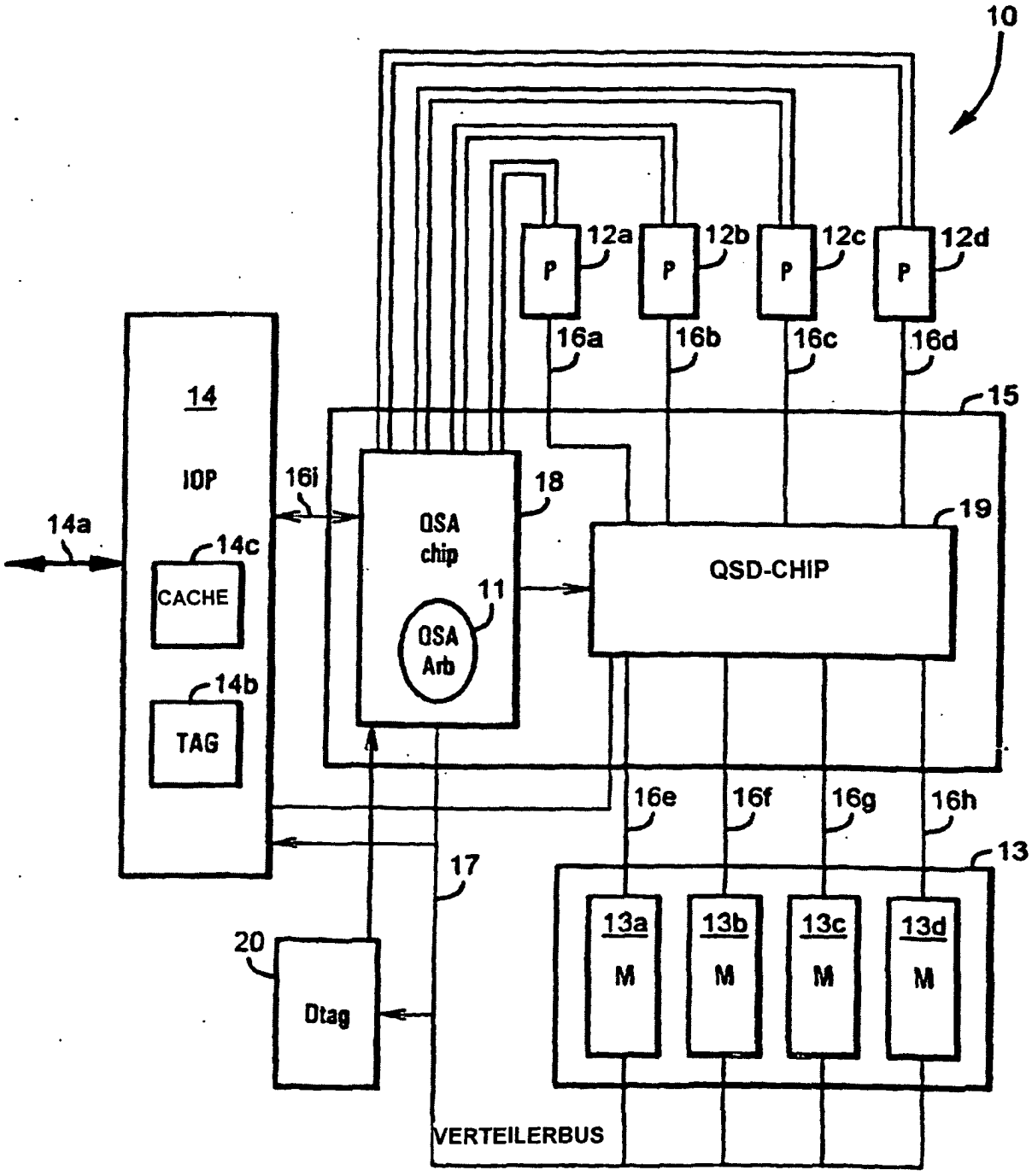

10
P
12a
12b
12c
12d
16a
16b
16c
16d
15
14
IOP
16i
18
19
14a
14c
CACHE
OSA
chip
11
OSA
Arb
QSD-CHIP
14b
TAG
16e
16f
16g
16h
13
17
13a
13b
13c
13d
M
20
Dtag
VERTEILERBUS

FIG. 2

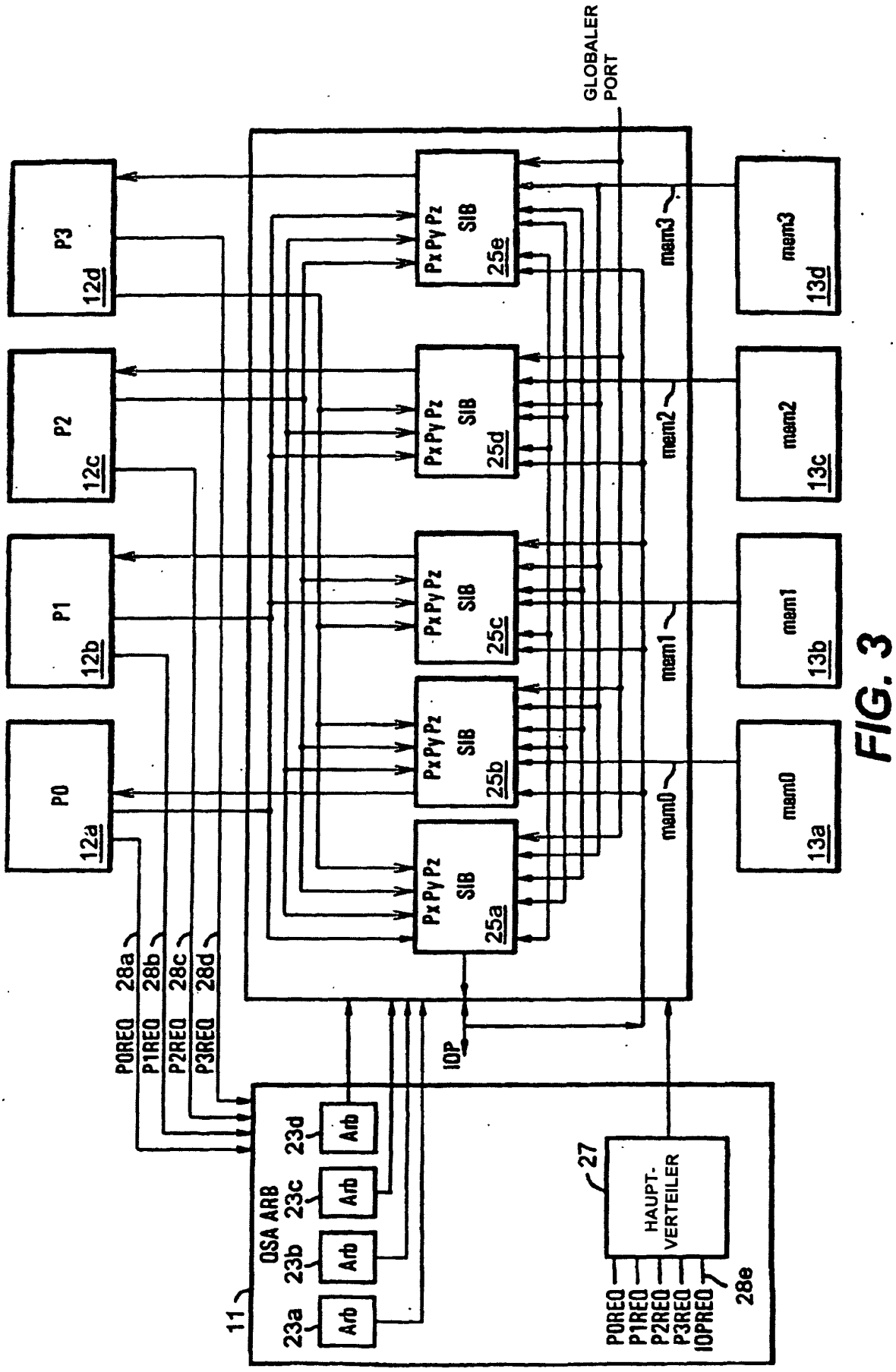

GLOBALER PORT
P3
12d
P2
12c
P1
12b
P0
12a
PxPyPz SIB
25e
25d
25c
25b
25a
mem3
13d
mem2
13c
mem1
13b
mem0
13a
28a
28b
28c
28d
P0REQ
P1REQ
P2REQ
P3REQ
IOP
11
OSA ARB
23a
23b
23c
23d
Arb
27
HAUPT-VERTEILER
28e
P0REQ
P1REQ
P2REQ
P3REQ
IOPREQ

FIG. 3

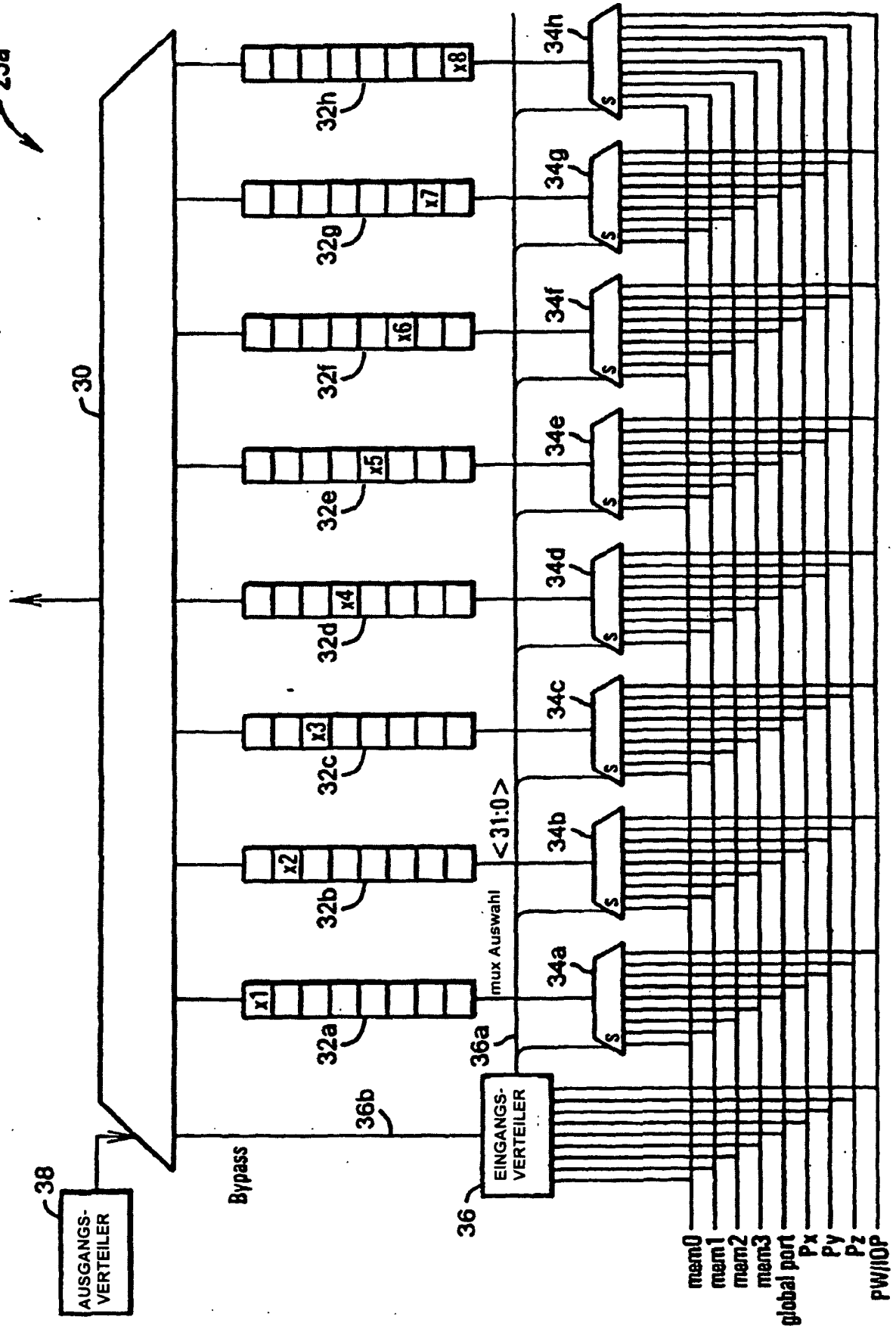

25a
30
38
AUSGANGS-VERTEILER
Bypass
36b
36
EINGANGS-VERTEILER
36a
mux Auswahl
<31:0>
32a
32b
32c
32d
32e
32f
32g
32h
x1
x2
x3
x4
x5
x6
x7
x8
34a
34b
34c
34d
34e
34f
34g
34h
S
mem0
mem1
mem2
mem3
global port
Px
Py
Pz
PW/IOP

FIG. 4A

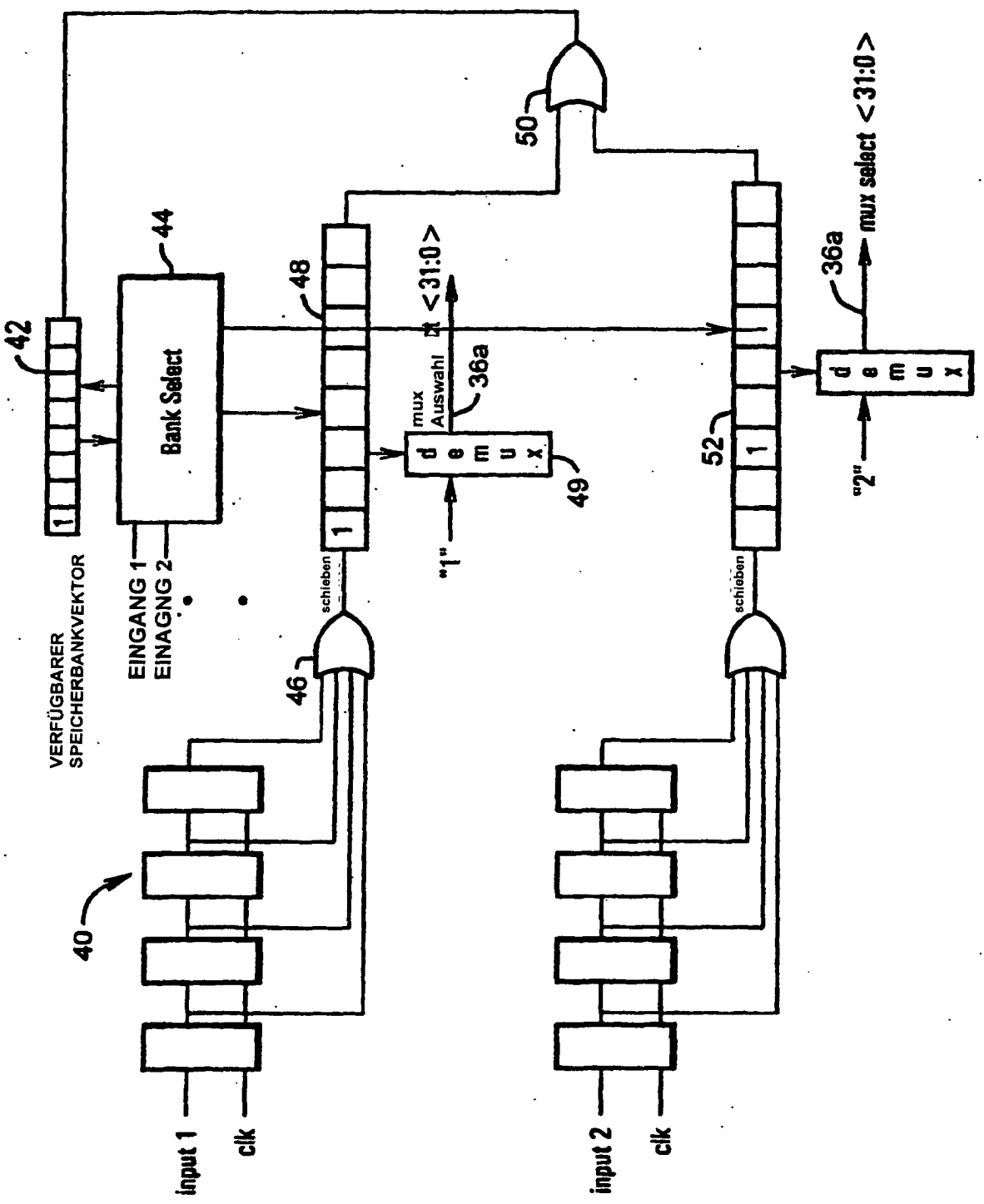

42
Bank Select
44
48
50
52
36a
mux Auswahl
<31:0>
mux select <31:0>
"1"
"2"
49
46
schieben
VERFÜGBARER SPEICHERBANKVEKTOR
EINGANG 1
EINAGNG 2
40
input 1
clk
input 2
clk

FIG. 4B

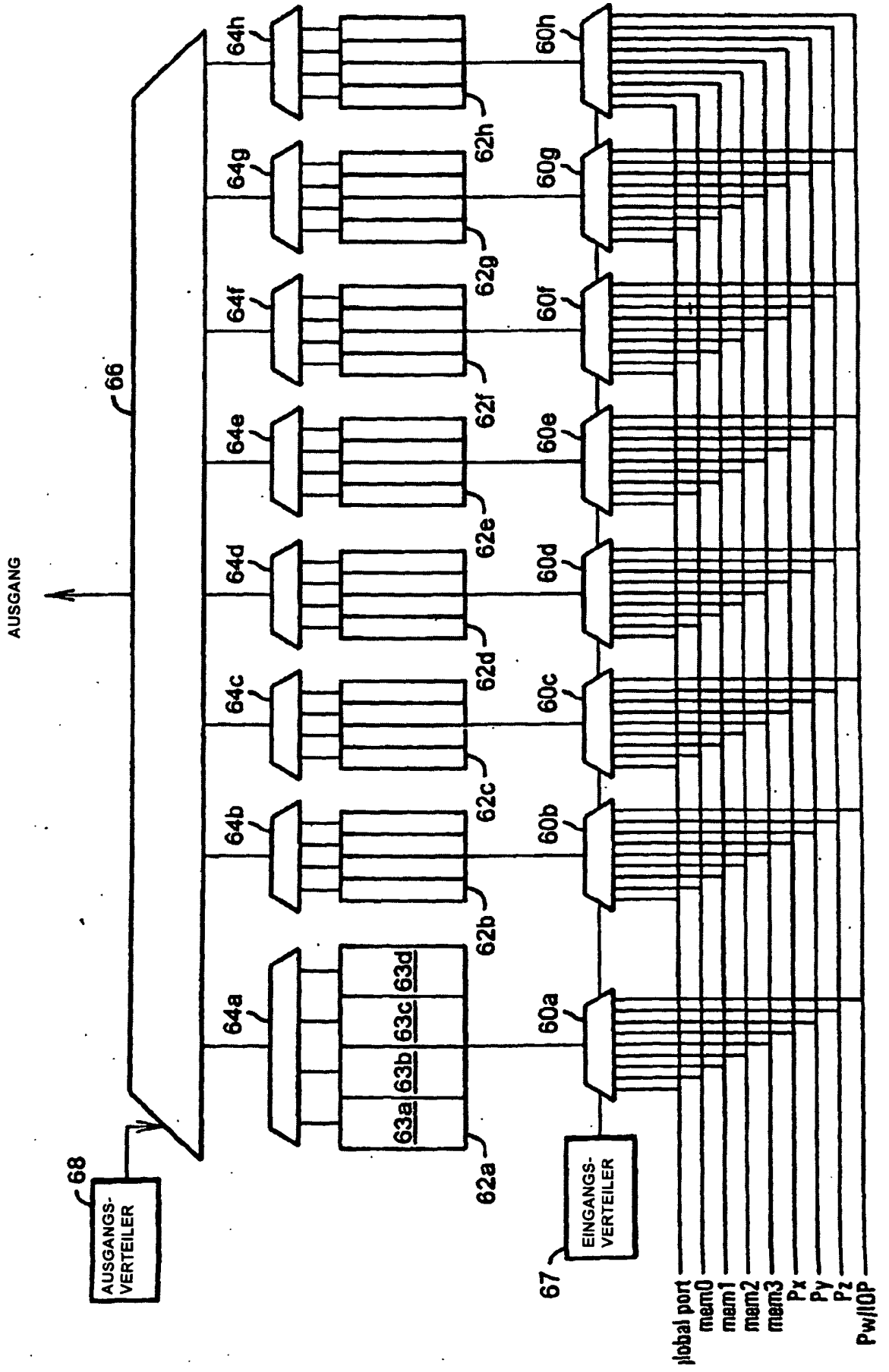

AUSGANG
66
68
AUSGANGS-VERTEILER
64a
64b
64c
64d
64e
64f
64g
64h
62a
62b
62c
62d
62e
62f
62g
62h
63a
63b
63c
63d
60a
60b
60c
60d
60e
60f
60g
60h
67
EINGANGS-VERTEILER
global port
mem0
mem1
mem2
mem3
Px
Py
Pz
Pw/IOP
71

FIG. 5

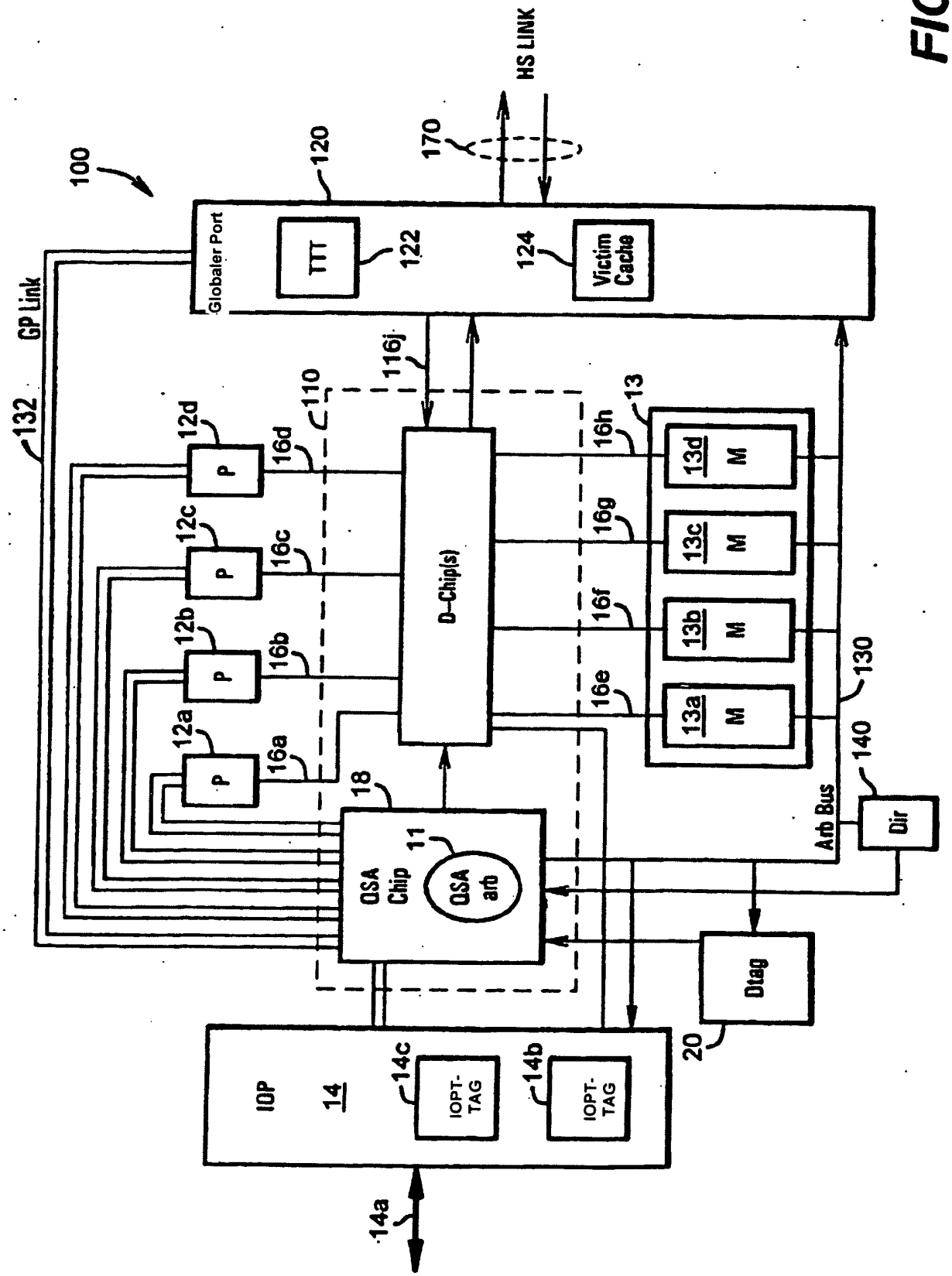

100
120
170
HS LINK
Globaler Port
TTT
122
124
Victim Cache
GP Link
132
116j
110
13
12d
16d
16h
13d
M
12c
16c
16g
13c
M
D-Chip(s)
12b
16b
16f
13b
M
130
12a
16a
16e
13a
M
140
18
Arb Bus
Dir
11
QSA Chip
QSA arb
Dtag
20
IOP
14
14c
IOPT-TAG
14b
IOPT-TAG
14a

FIG. 6

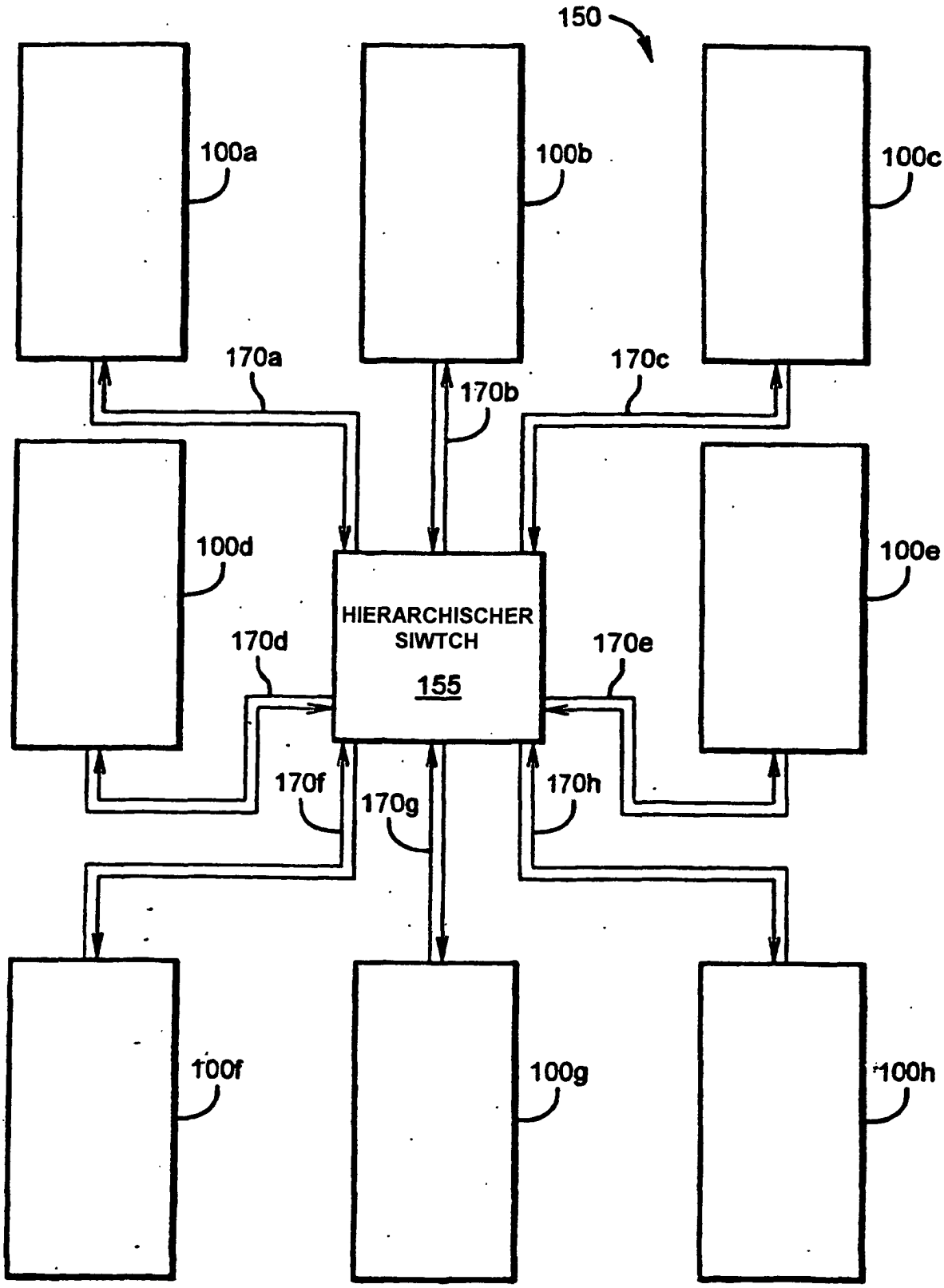

150
100a
100b
100c
170a
170b
170c
100d
100e
HIERARCHISCHER SIWTCH
155
170d
170e
170f
170g
170h
100f
100g
100h

FIG. 7A

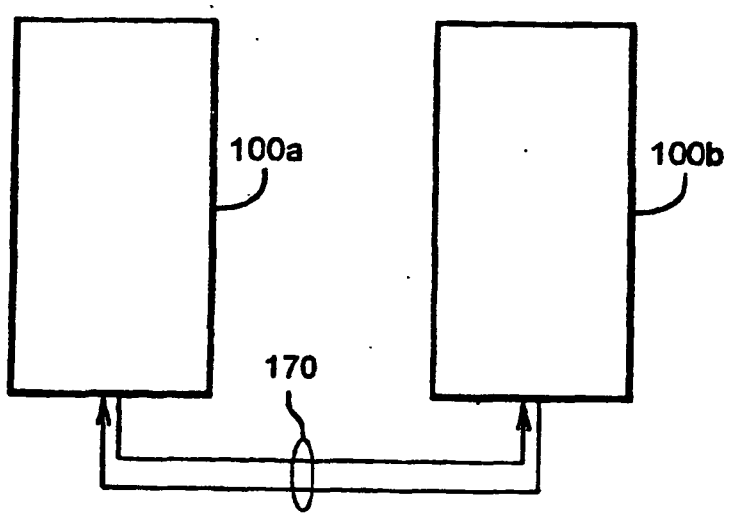


FIG. 7B

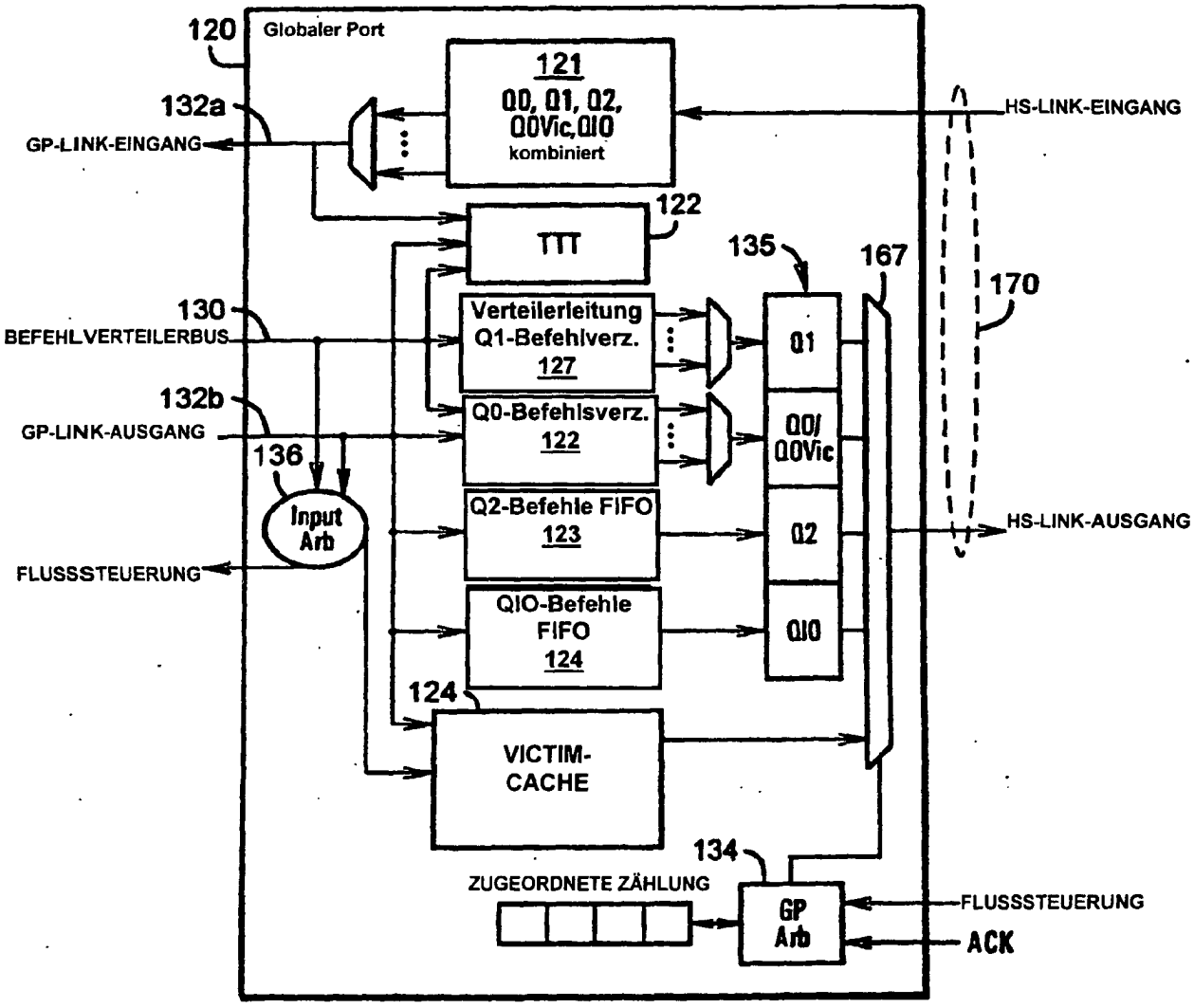

120
Globaler Port
121
Q0, Q1, Q2,
Q0Vic,Q10
kombiniert
132a
GP-LINK-EINGANG
HS-LINK-EINGANG
122
TTT
135
167
170
130
BEFEHLVERTEILERBUS
Verteilerleitung
Q1-Befehlverz.
127
Q1
132b
GP-LINK-AUSGANG
Q0-Befehlsverz.
122
Q0/
Q0Vic
136
Input
Arb
Q2-Befehle FIFO
123
Q2
HS-LINK-AUSGANG
FLUSSSTEUERUNG
Q10-Befehle
FIFO
124
Q10
124
VICTIM-
CACHE
134
ZUGEORDNETE ZÄHLUNG
GP
Arb
FLUSSSTEUERUNG
ACK

FIG. 8

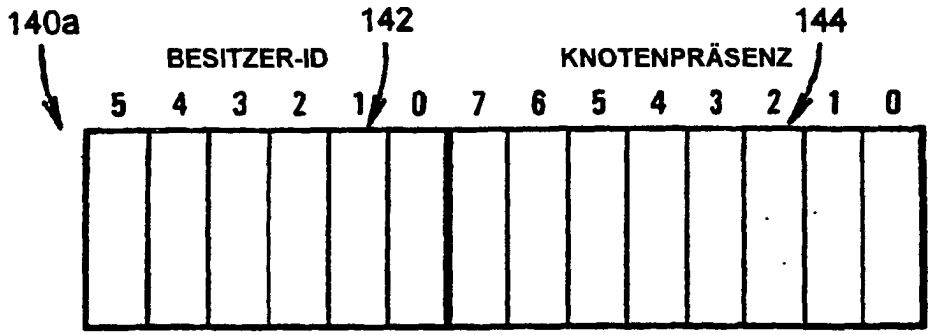


FIG. 9

TRANSAKTIONS-TRACKING-VERZEICHNIS (TTT)

122 (FIGUR 6)

152 Adresse	154 Befehl	156 Befehls-ID	158 Status-Bits					
			Fill here	Fill Marker Here	Shadow	Ack/Nack here	Fetch	Loop Consig
			158a	158d	158b	158c		

FIG. 10

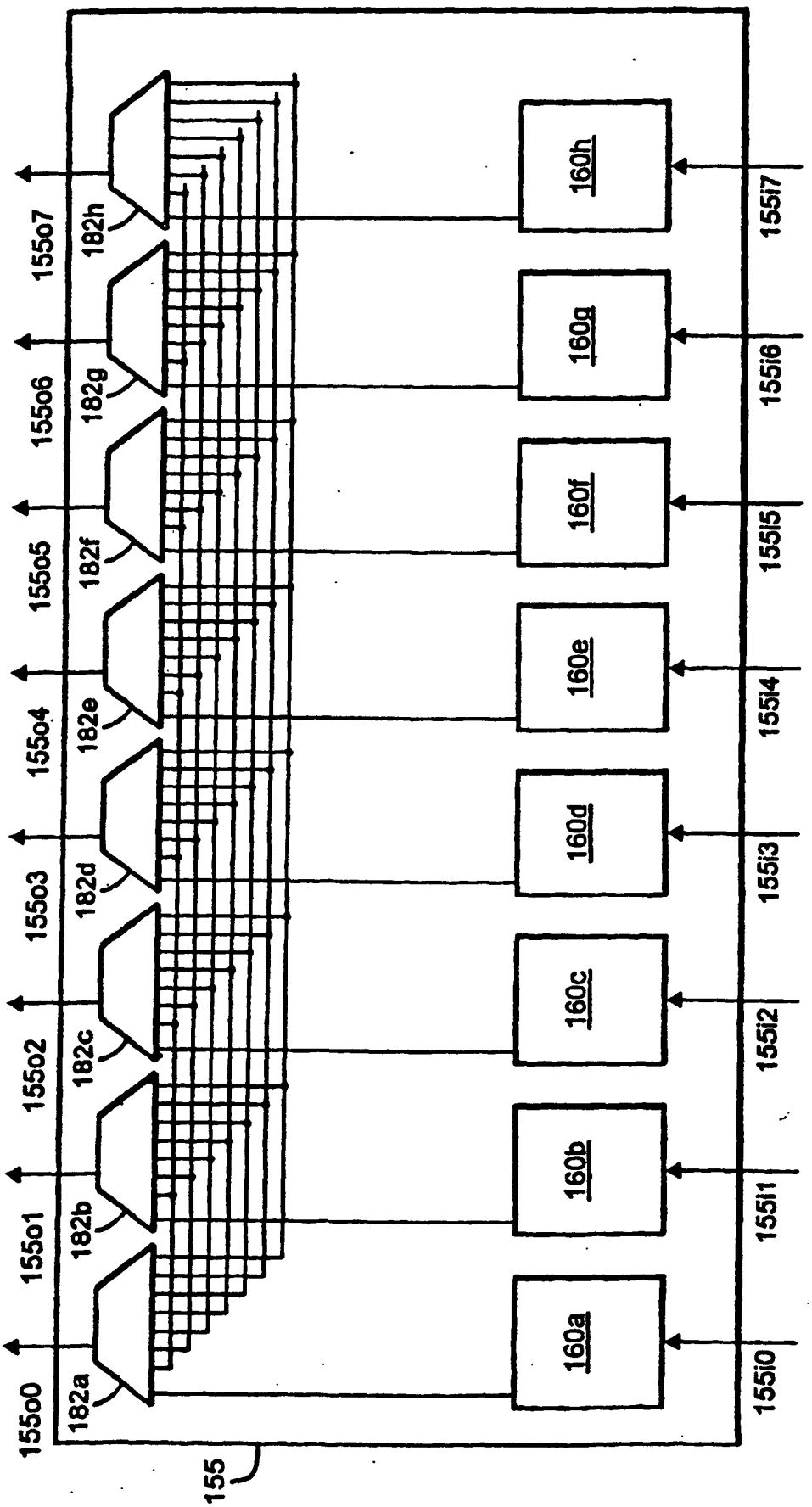

155o0
155o1
155o2
155o3
155o4
155o5
155o6
155o7
182a
182b
182c
182d
182e
182f
182g
182h
160a
160b
160c
160d
160e
160f
160g
160h
155i0
155i1
155i2
155i3
155i4
155i5
155i6
155i7
155

FIG. 11

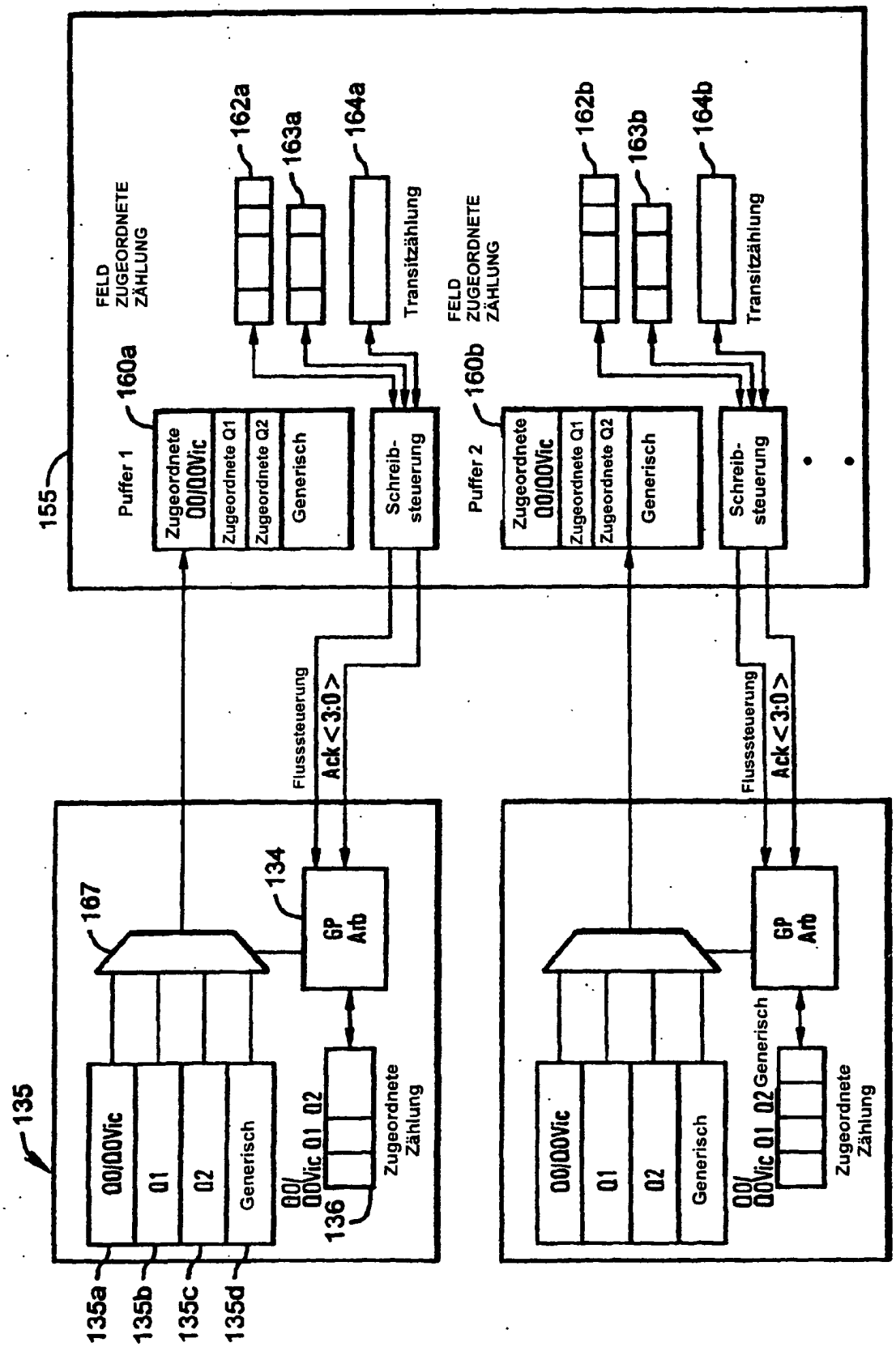

155
Puffer 1
160a
Zugeordnete 00/00Vic
Zugeordnete Q1
Zugeordnete Q2
Generisch
Schreib-steuerung
FELD ZUGEORDNETE ZÄHLUNG
162a
163a
164a
Transitzählung
Puffer 2
160b
Zugeordnete 00/00Vic
Zugeordnete Q1
Zugeordnete Q2
Generisch
Schreib-steuerung
FELD ZUGEORDNETE ZÄHLUNG
162b
163b
164b
Transitzählung
Flusssteuerung
Ack<3:0>
Flusssteuerung
Ack<3:0>
135
167
134
GP Arb
00/00Vic
Q1
Q2
Generisch
00/00Vic Q1 Q2
Zugeordnete Zählung
136
GP Arb
00/00Vic
Q1
Q2
Generisch
00/00Vic Q1 Q2 Generisch
Zugeordnete Zählung
135a
135b
135c
135d

FIG. 12A

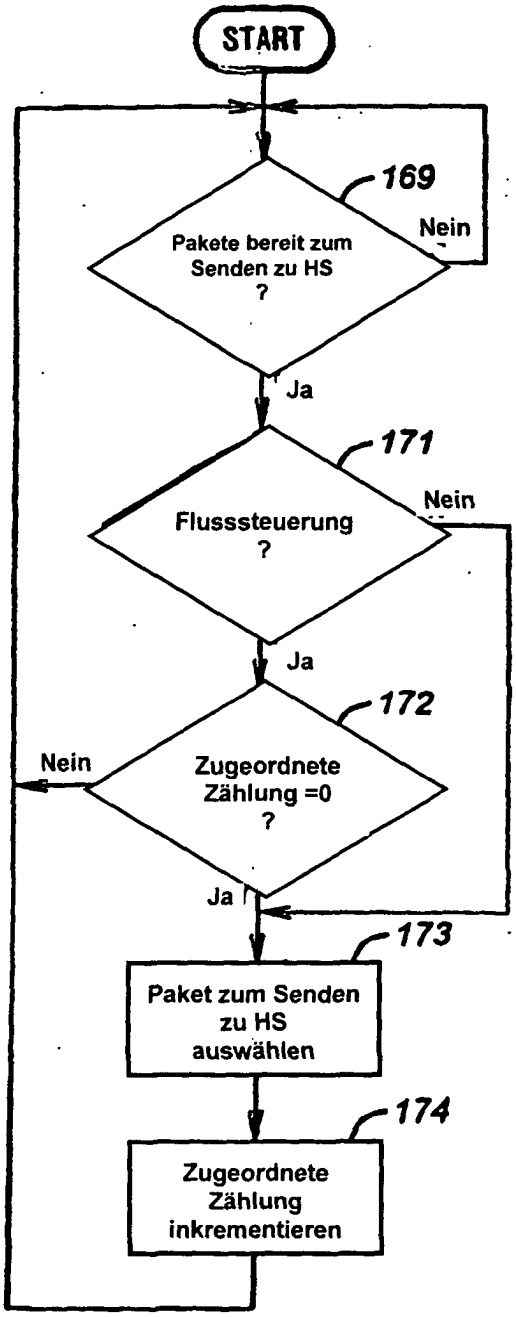


FIG. 12B

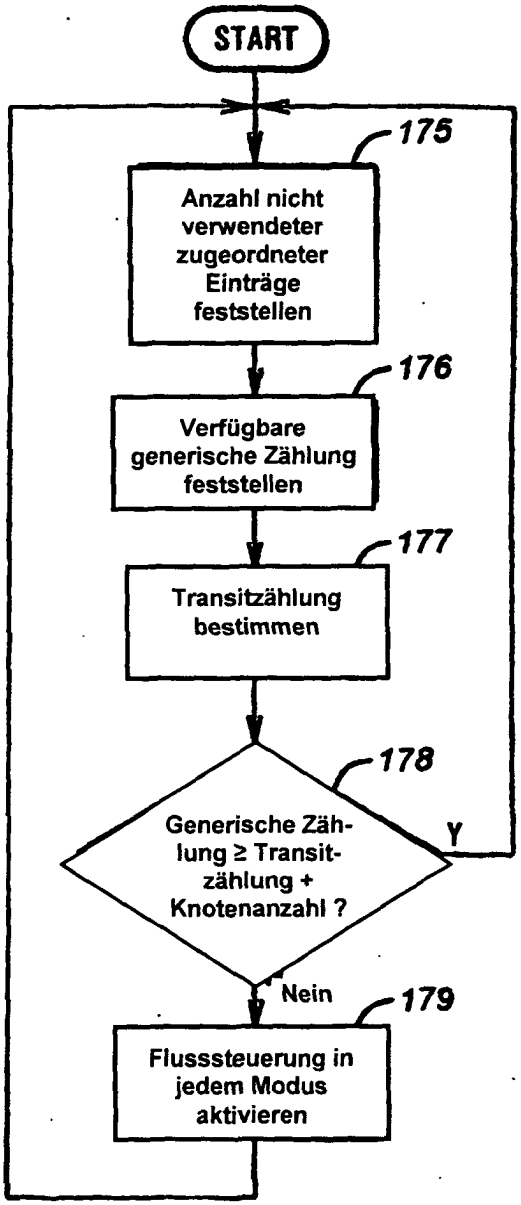


FIG. 13

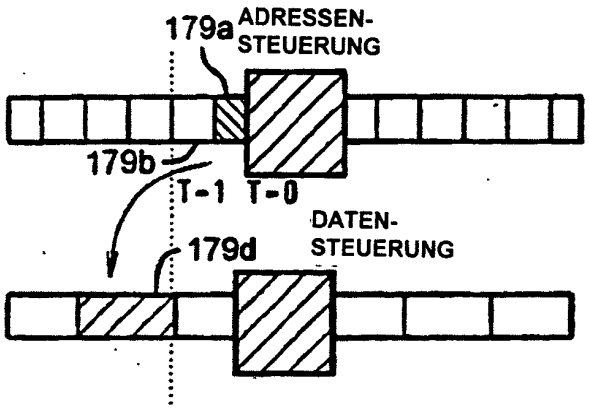

179a
ADRESSEN-
STEUERUNG
179b
T-1
T-0
DATEN-
STEUERUNG
179d

FIG. 14

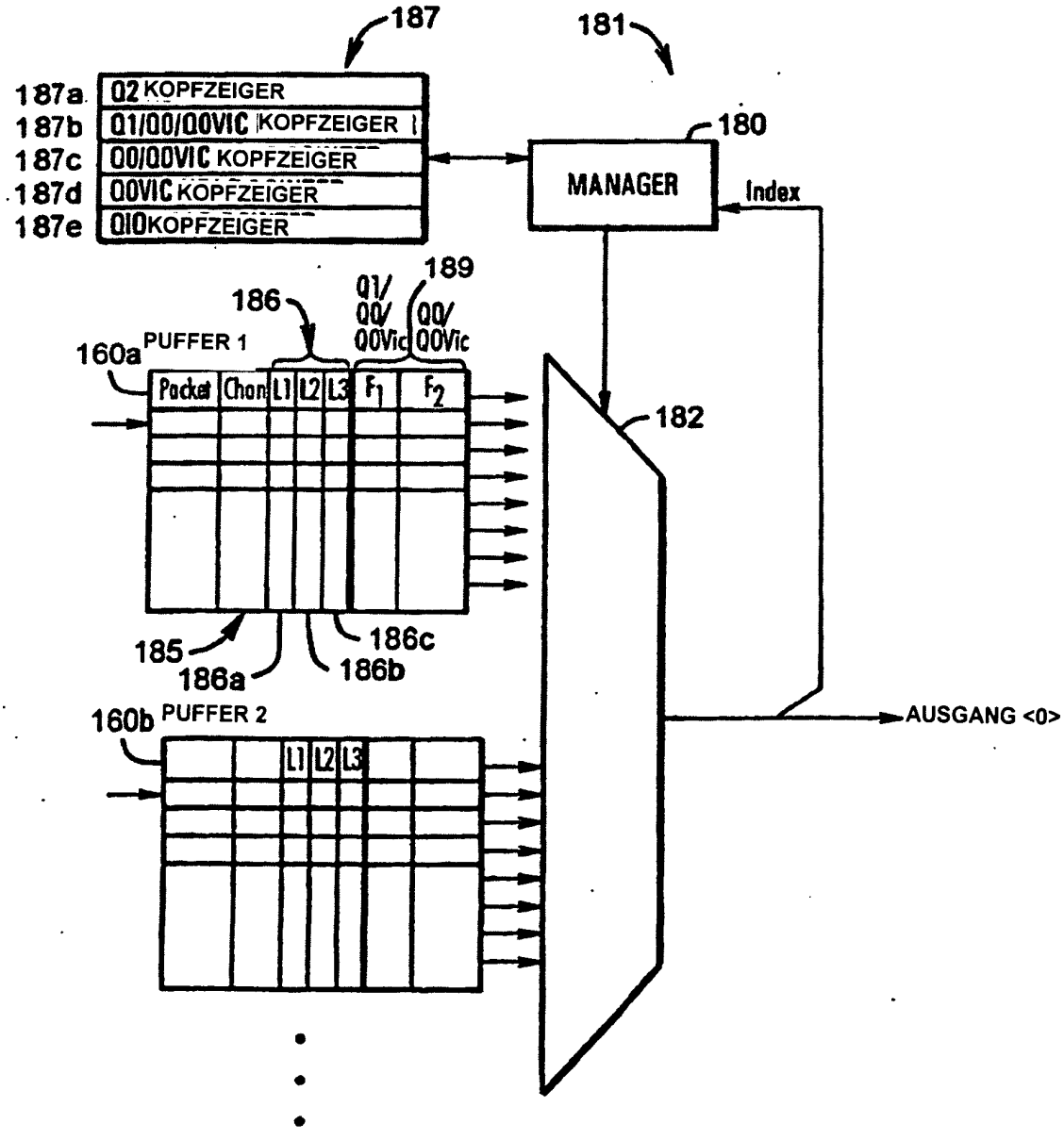

187
181
187a
187b
187c
187d
187e
Q2 KOPFZEIGER
Q1/Q0/QOVIC KOPFZEIGER
Q0/QOVIC KOPFZEIGER
QOVIC KOPFZEIGER
Q1OKOPFZEIGER
180
MANAGER
Index
189
186
Q1/
Q0/
QOVic
Q0/
QOVic
160a
PUFFER 1
Packet
Chan
L1
L2
L3
F1
F2
182
185
186a
186b
186c
160b
PUFFER 2
AUSGANG <0>

FIG. 15

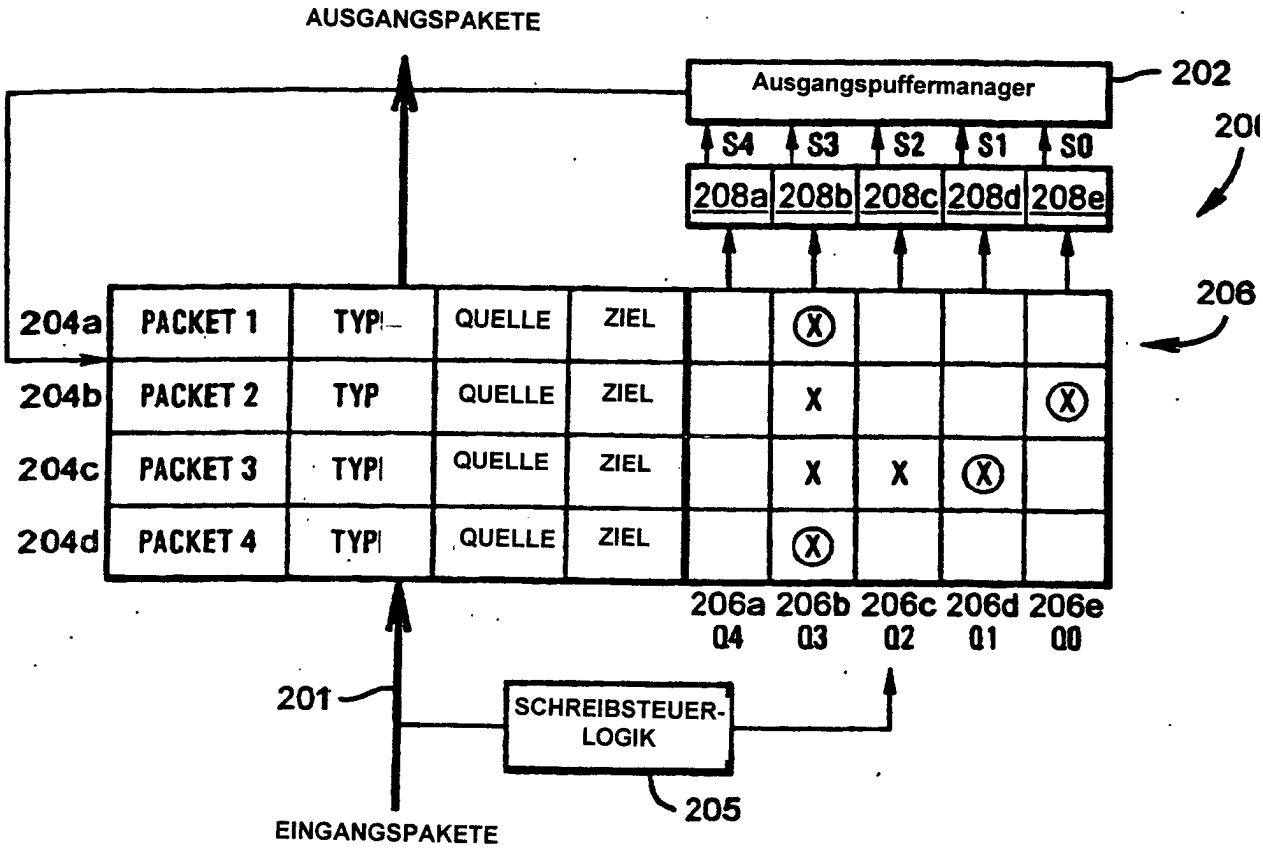


FIG. 16

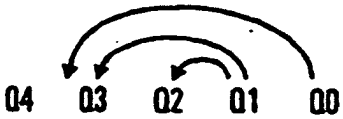


FIG. 16A

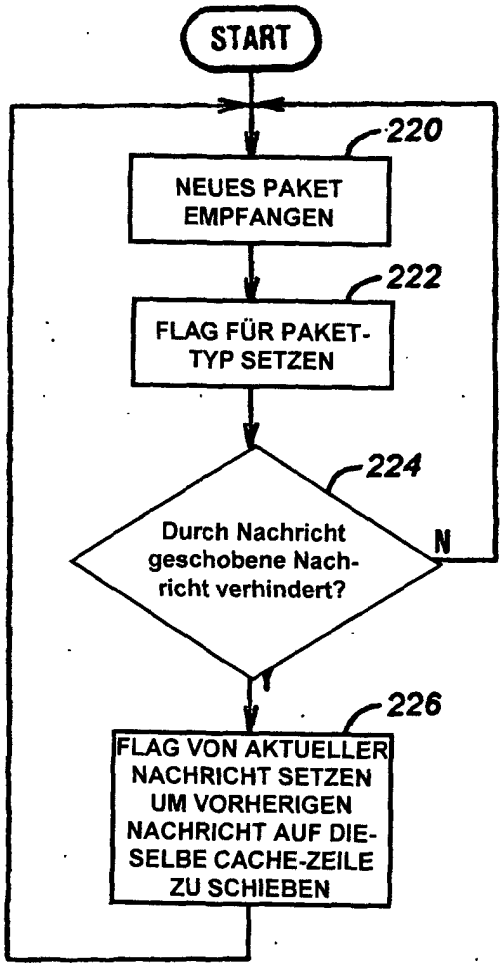

START
220
NEUES PAKET EMPFANGEN
222
FLAG FÜR PAKET-TYP SETZEN
224
Durch Nachricht geschobene Nachricht verhindert?
N
226
FLAG VON AKTUELLER NACHRICHT SETZEN UM VORHERIGEN NACHRICHT AUF DIESELBE CACHE-ZEILE ZU SCHIEBEN

FIG. 17

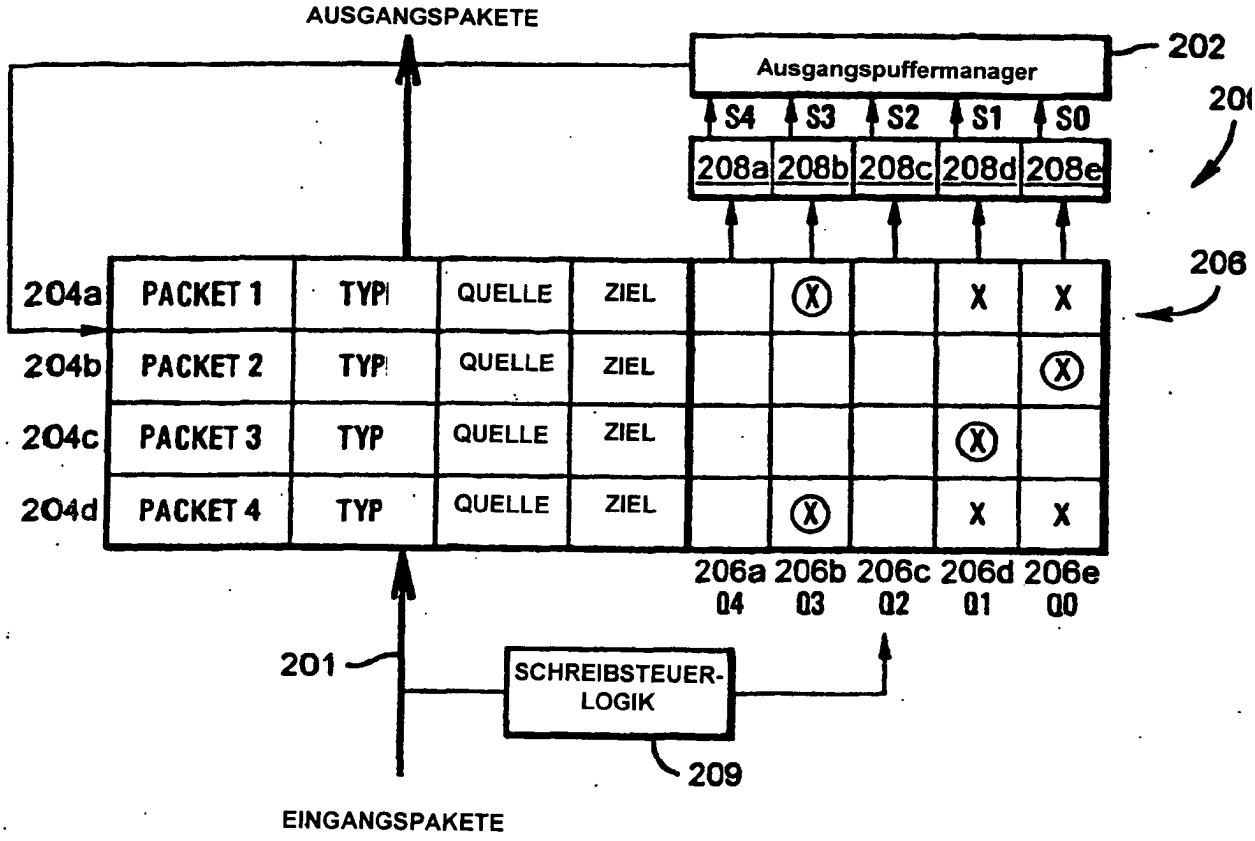


FIG. 18

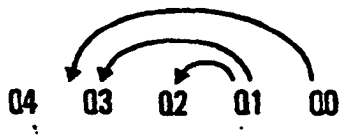


FIG. 18A

300	305	320
BLOCK LESEN	Invalid	
	Clean	
	Dirty	Weitergeleitet Read
	Dirty–Shared	Weitergeleitet Read
BLOCK LESEN MODIFIZIEREN	Invalid	
	Clean	Invalidate
	Dirty	Weitergeleitet Read Mod
	Dirty–Shared	Weitergeleitet Read Mod
ABRUFEN	Invalid	
	Clean	
	Dirty	Weitergeleitet Read
	Dirty–Shared	Weitergeleitet Read
BLOCK LESEN VICTIM	Invalid	
	Clean	
	Dirty	Weitergeleitet Read
	Dirty–Shared	Weitergeleitet Read
BLOCK LESEN VICTIM MODIFIZIEREN	Invalid	
	Clean	Invalidate
	Dirty	Weitergeleitet Read Mod
	Dirty–Shared	Weitergeleitet Read Mod
ABRUFEN BLOCK VICTIM	Invalid	
	Clean	
	Dirty	Weitergeleitet Read
	Dirty–Shared	Weitergeleitet Read
Victim	Any State	
Clean Victim	Any State	

300a	305a	320a
Clean–to–Dirty	Invalid	
	Clean	Invalidate
	Dirty	N/A
	Dirty–Shared	Invalidate
Shared–to–Dirty	Invalid	
	Clean	Invalidate
	Dirty	N/A
	Dirty–Shared	N/A
STC Change –to–Dirty	Invalid	
	Clean	Invalidate
	Dirty	N/A
	Dirty–Shared	Invalidate
Inval–to–Dirty	Invalid	
	Clean	Invalidate
	Dirty	Invalidate
	Dirty–Shared	Invalidate
Full–Block Schreiben	Invalid	
	Clean	Invalidate
	Dirty	Invalidate
	Dirty–Shared	Invalidate

FIG. 19

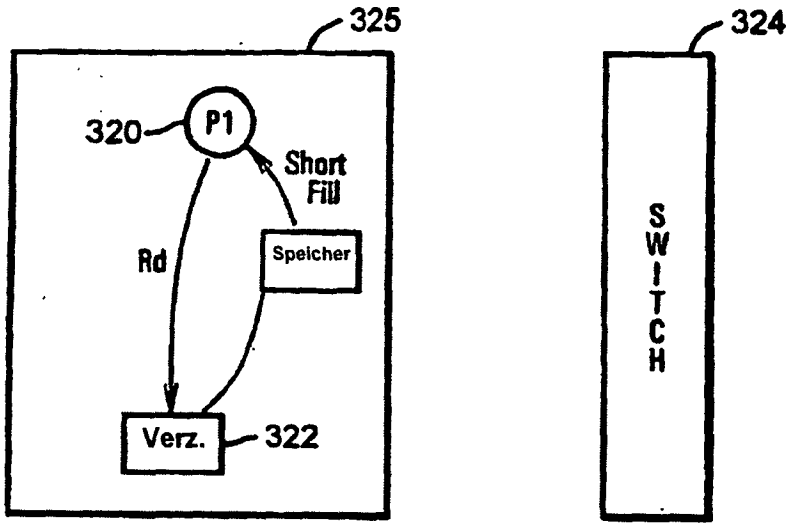


FIG. 20A

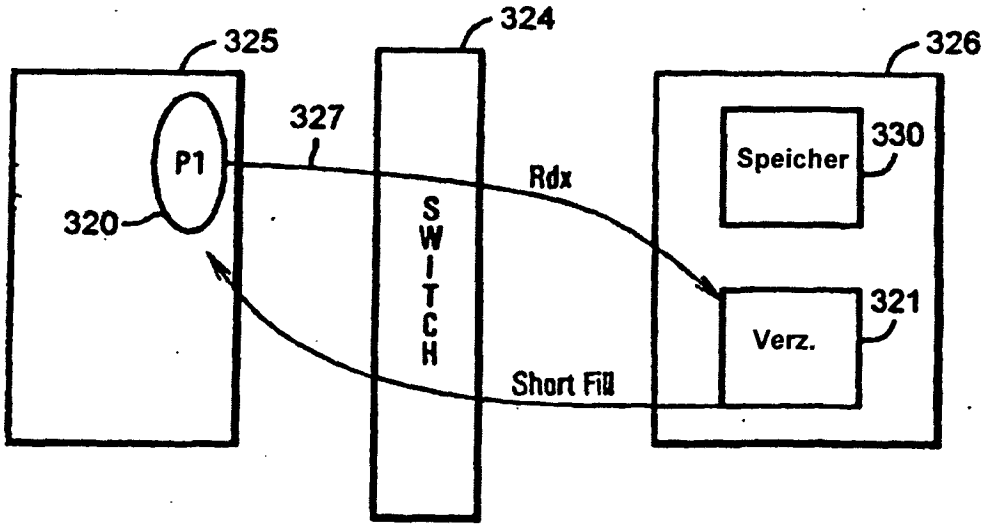


FIG. 20B

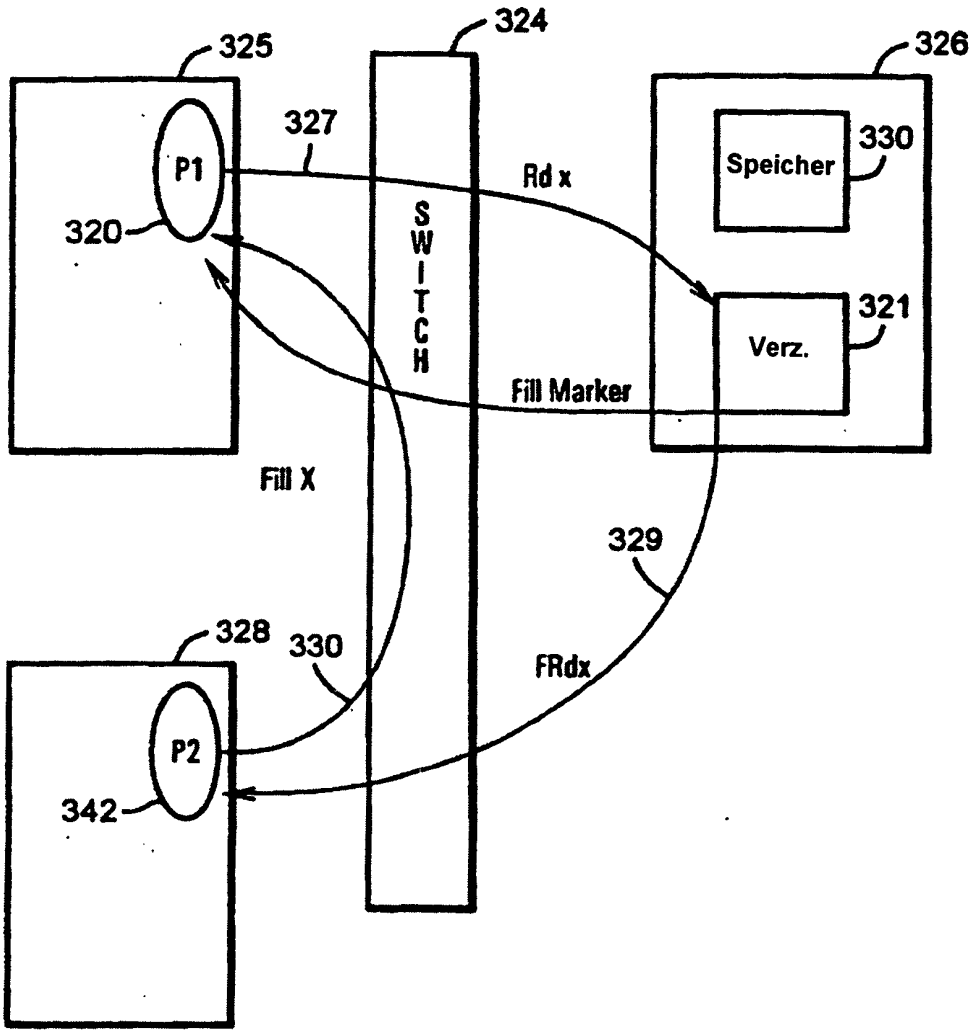


FIG. 20C

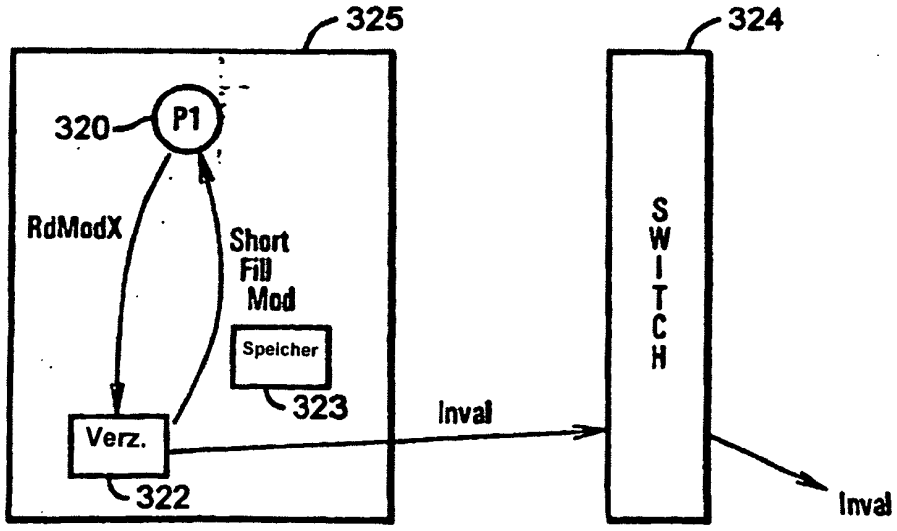


FIG. 20D

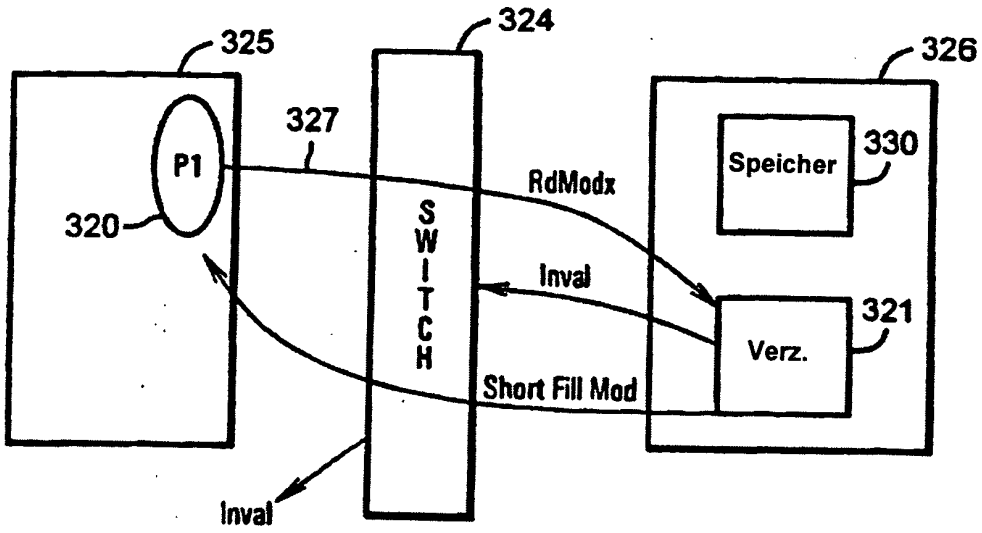


FIG. 20E

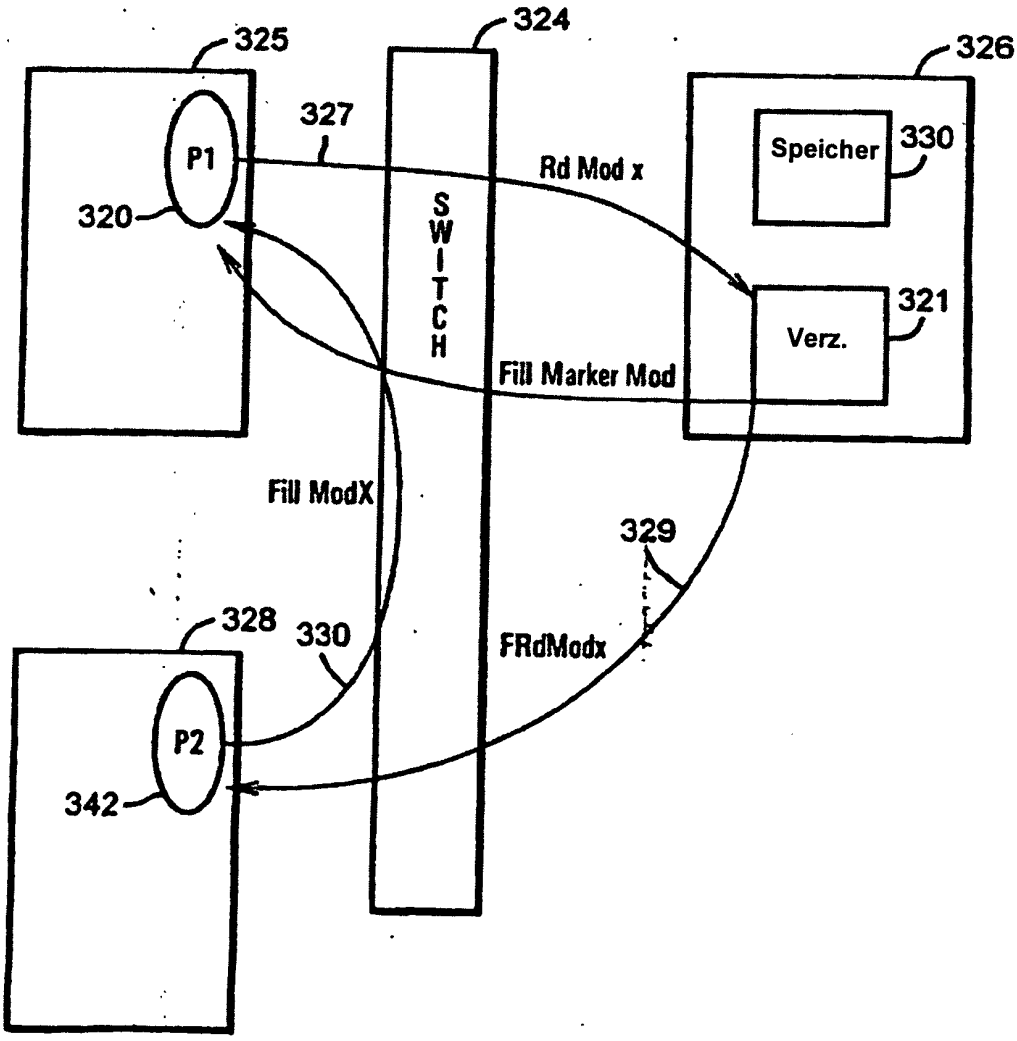


FIG. 20F

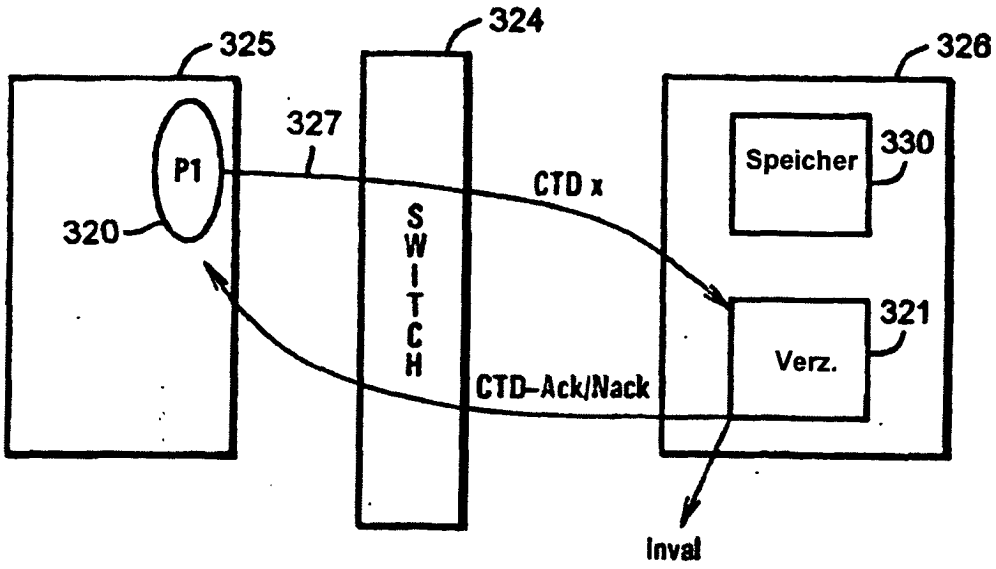


FIG. 20G

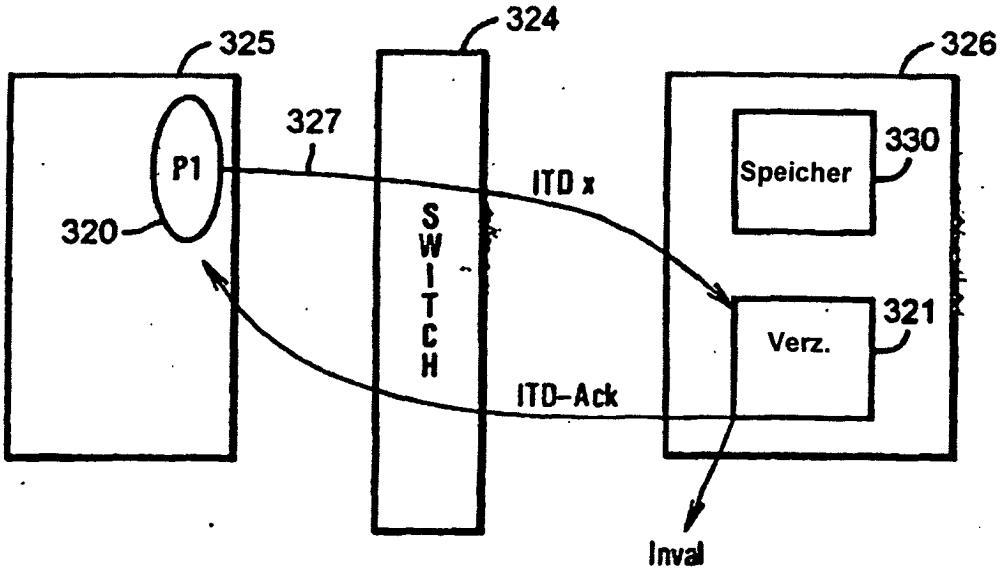


FIG. 20H

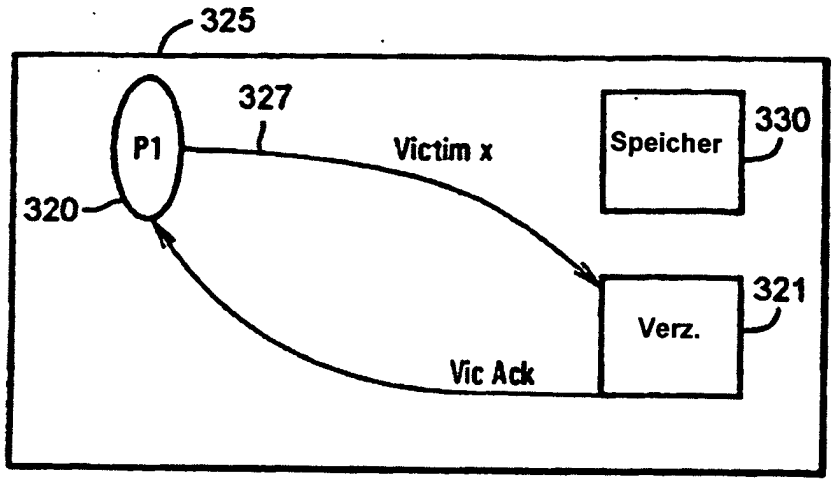


FIG. 20I

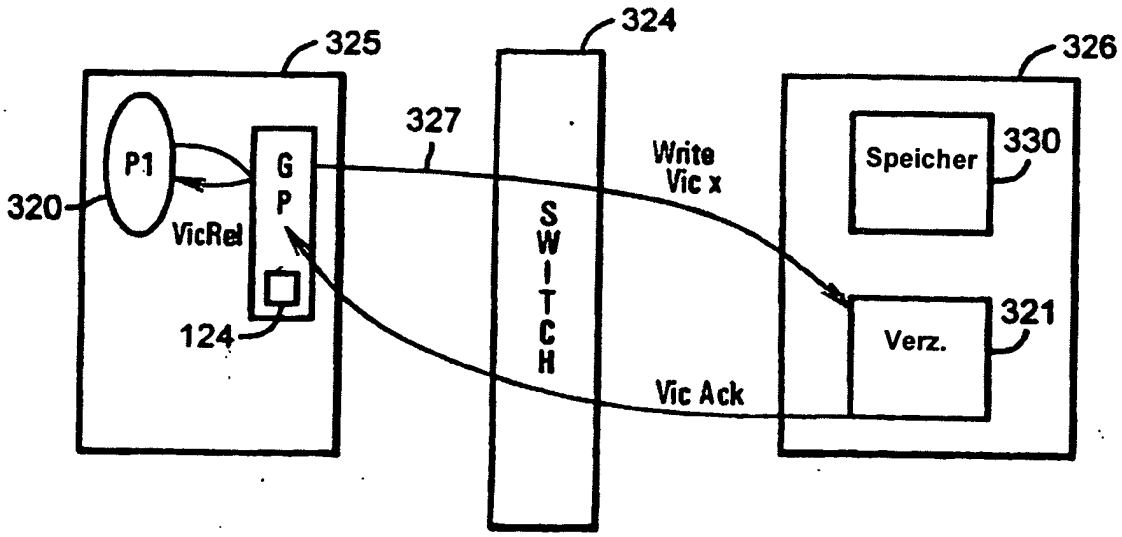


FIG. 20J

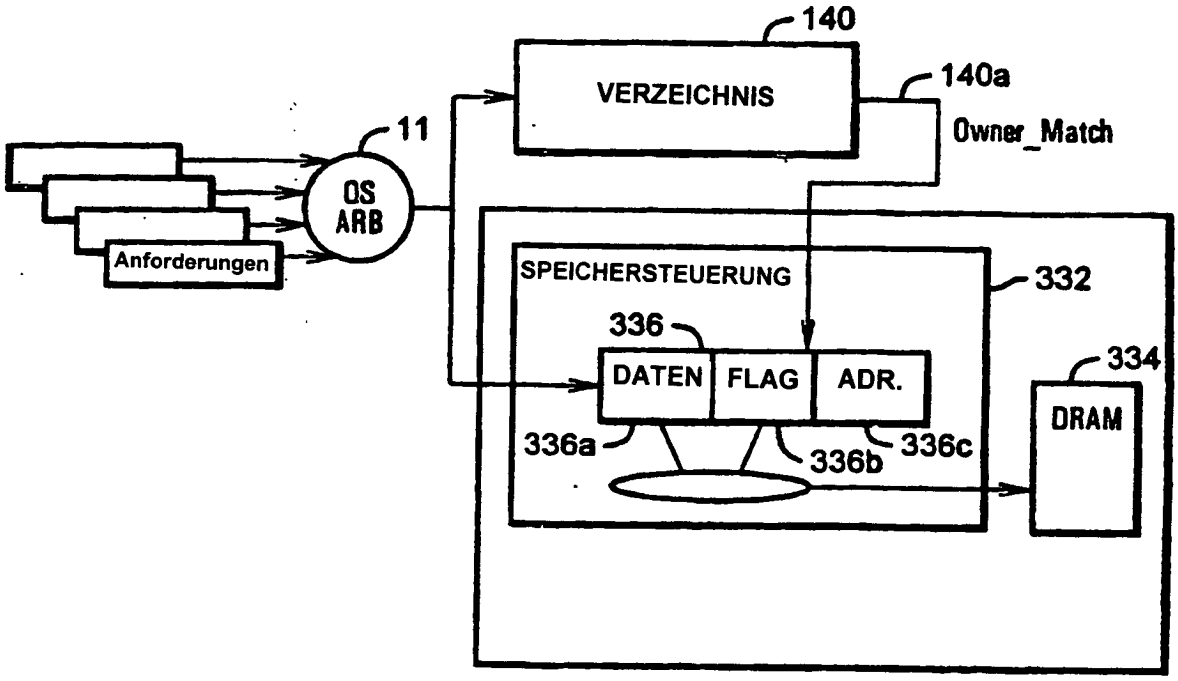


FIG. 21

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Arb_Bus	Read0	Write1	Read2	Write3	Read4	Write5	Read6	Read7	Write8		Write9
Owner_Match		match1		match3		mismatch					
Memory_Bus	Read0		Read2	Write1	Read0	Write3	Read6	Read7	*No write executed*		Write8

FIG. 22

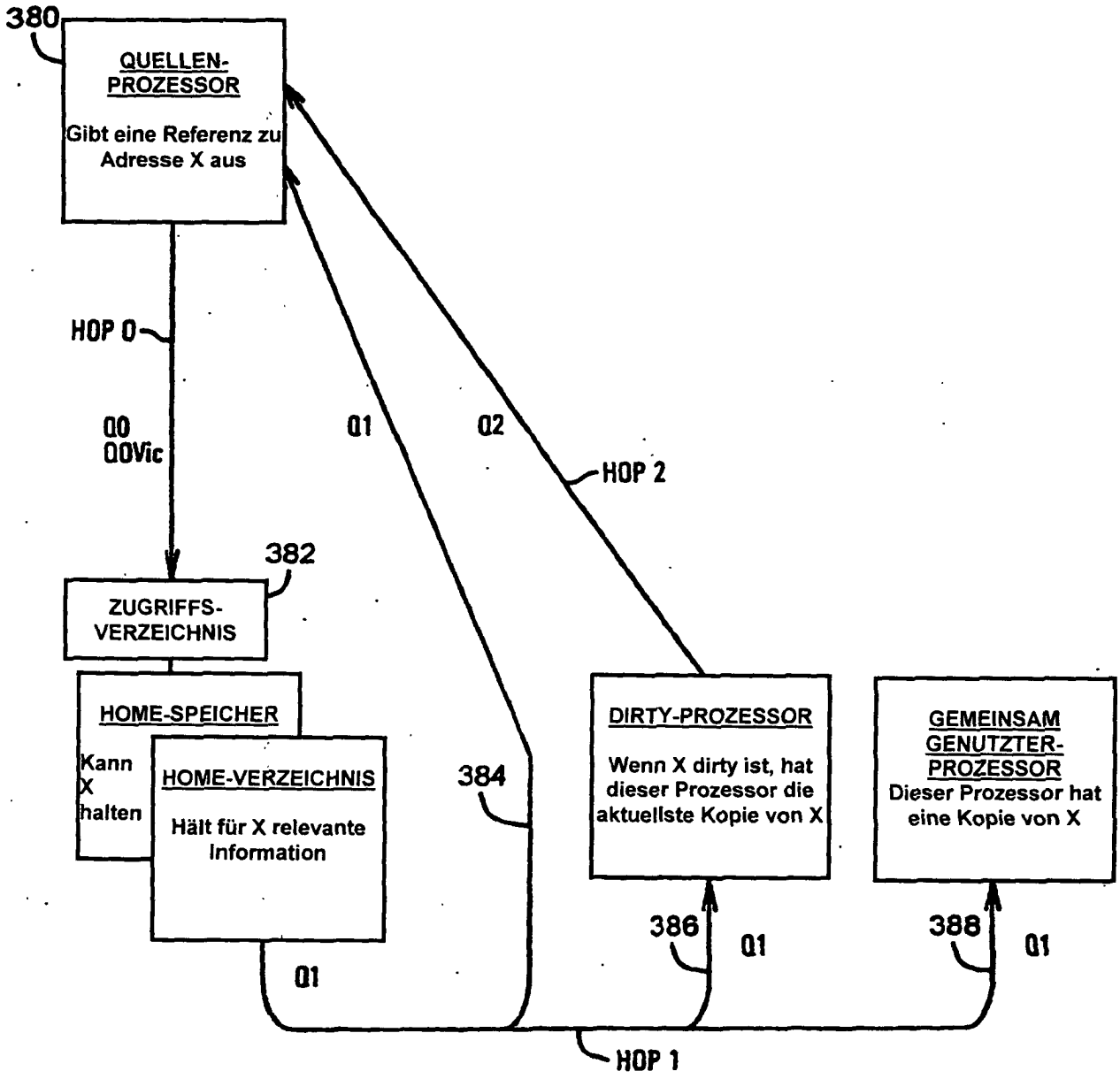

380
QUELLEN-PROZESSOR
Gibt eine Referenz zu Adresse X aus
HOP 0
Q0
Q0Vic
Q1
Q2
HOP 2
382
ZUGRIFFS-VERZEICHNIS
HOME-SPEICHER
Kann X halten
HOME-VERZEICHNIS
Hält für X relevante Information
384
DIRTY-PROZESSOR
Wenn X dirty ist, hat dieser Prozessor die aktuellste Kopie von X
GEMEINSAM GENUTZTER-PROZESSOR
Dieser Prozessor hat eine Kopie von X
386
Q1
388
Q1
Q1
HOP 1

FIG. 23

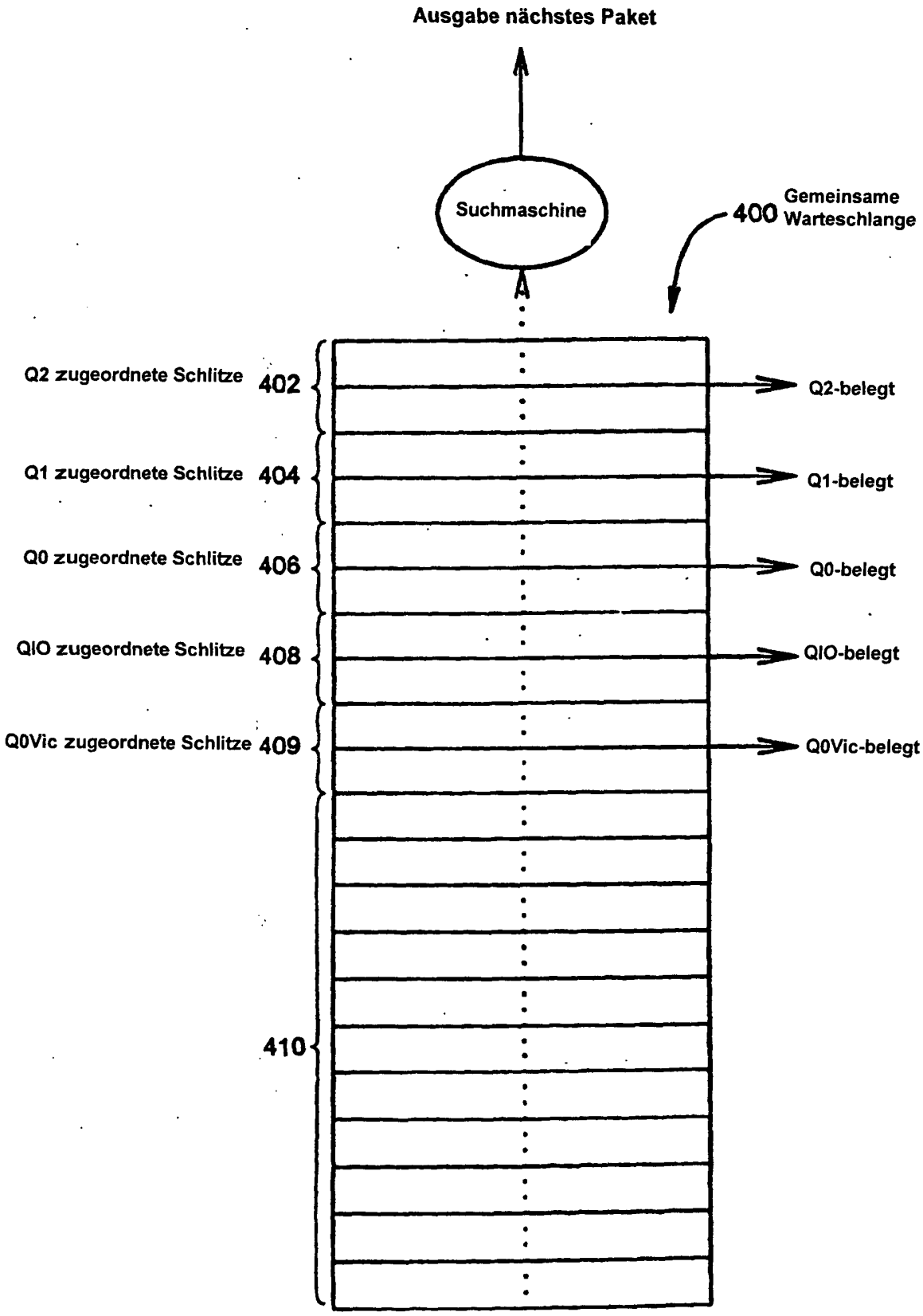

Ausgabe nächstes Paket
Suchmaschine
400
Gemeinsame Warteschlange
Q2 zugeordnete Schlitze
402
Q2-belegt
Q1 zugeordnete Schlitze
404
Q1-belegt
Q0 zugeordnete Schlitze
406
Q0-belegt
QIO zugeordnete Schlitze
408
QIO-belegt
Q0Vic zugeordnete Schlitze
409
Q0Vic-belegt
410

FIG. 24

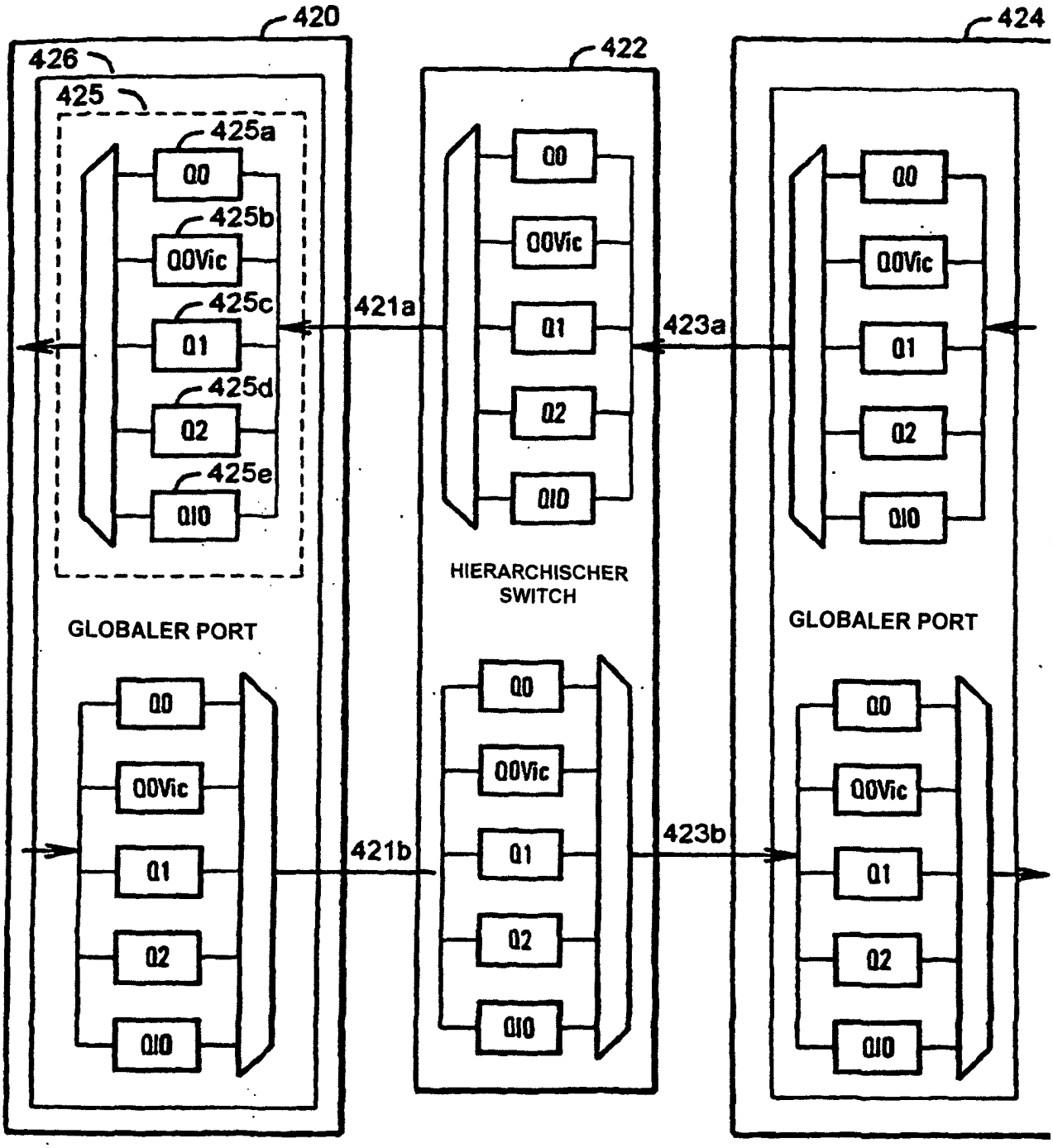

420
426
425
425a
Q0
425b
Q0Vic
425c
Q1
425d
Q2
425e
Q10
GLOBALER PORT
421a
421b
422
Q0
Q0Vic
Q1
Q2
Q10
HIERARCHISCHER SWITCH
423a
423b
424
Q0
Q0Vic
Q1
Q2
Q10
GLOBALER PORT

FIG. 25

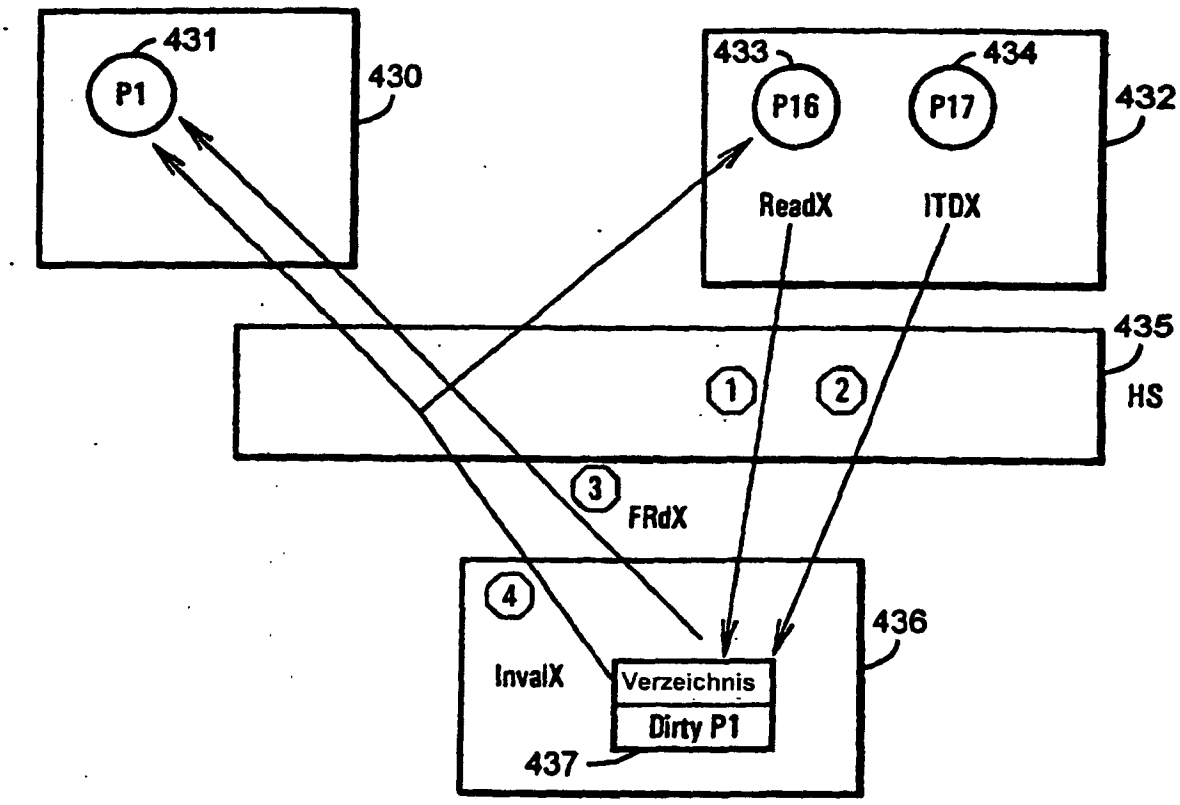

431
P1
430
433
P16
434
P17
432
ReadX
ITDX
435
HS
1
2
3
FRdX
4
436
InvalX
Verzeichnis
Dirty P1
437

FIG. 26

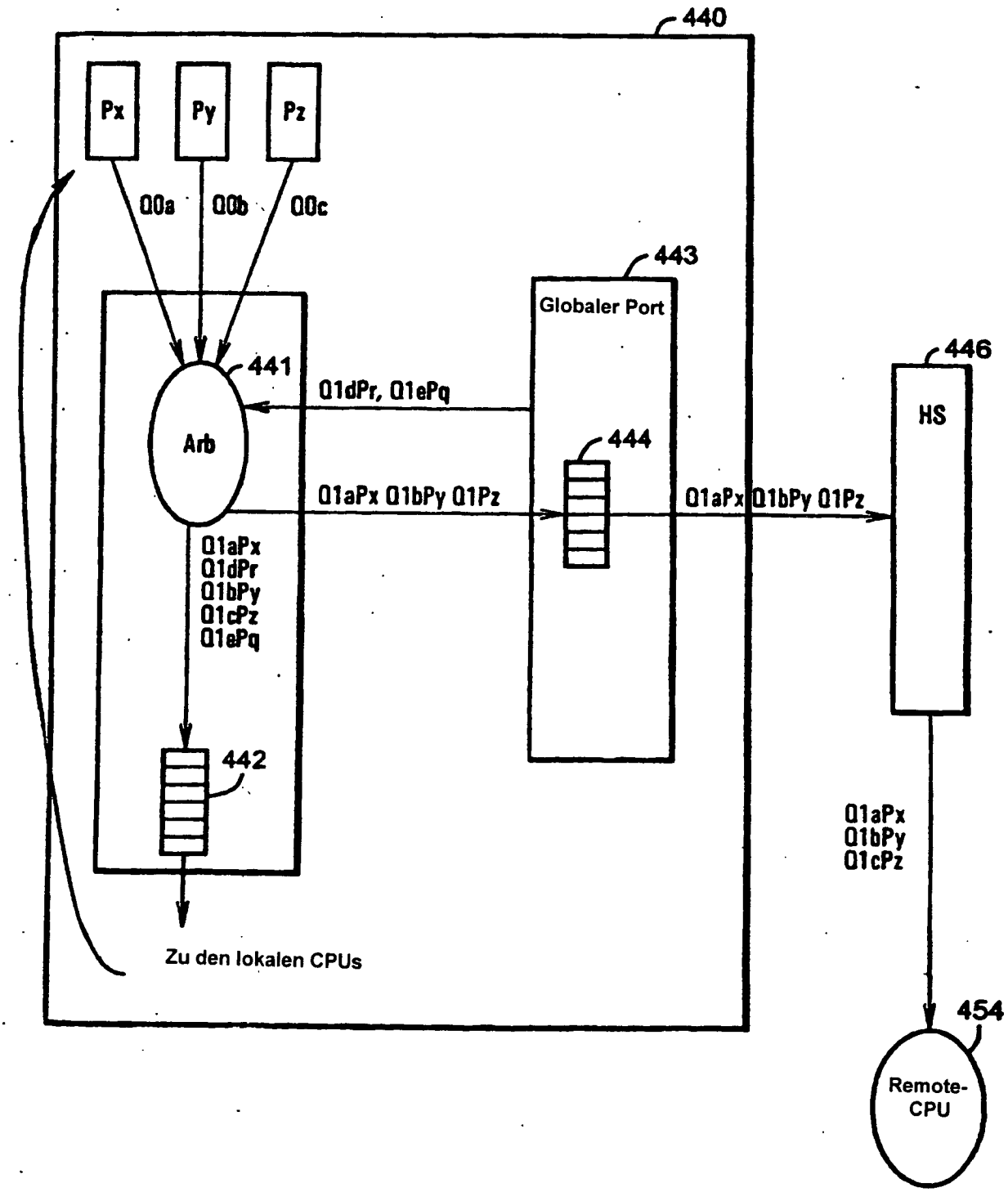

440
Px
Py
Pz
Q0a
Q0b
Q0c
443
Globaler Port
441
Arb
Q1dPr, Q1ePq
444
Q1aPx Q1bPy Q1Pz
Q1aPx Q1bPy Q1Pz
446
HS
Q1aPx
Q1dPr
Q1bPy
Q1cPz
Q1ePq
442
Q1aPx
Q1bPy
Q1cPz
454
Remote-
CPU
Zu den lokalen CPUs

FIG. 27A

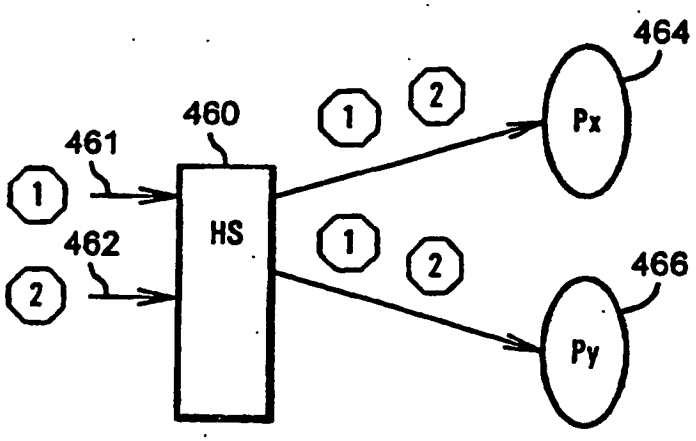


FIG. 27B

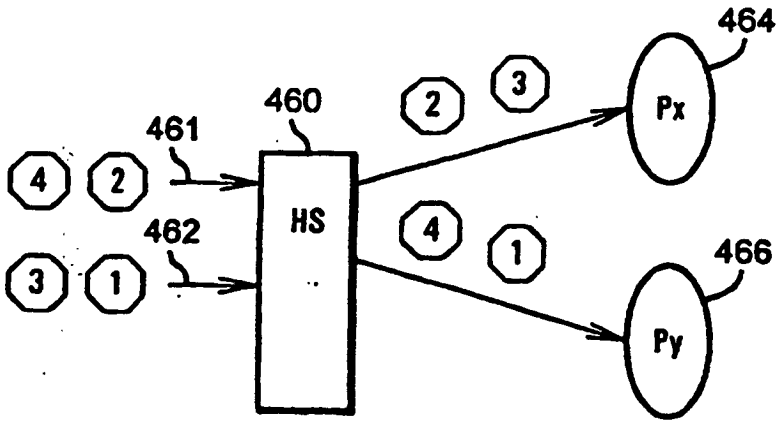


FIG. 27C

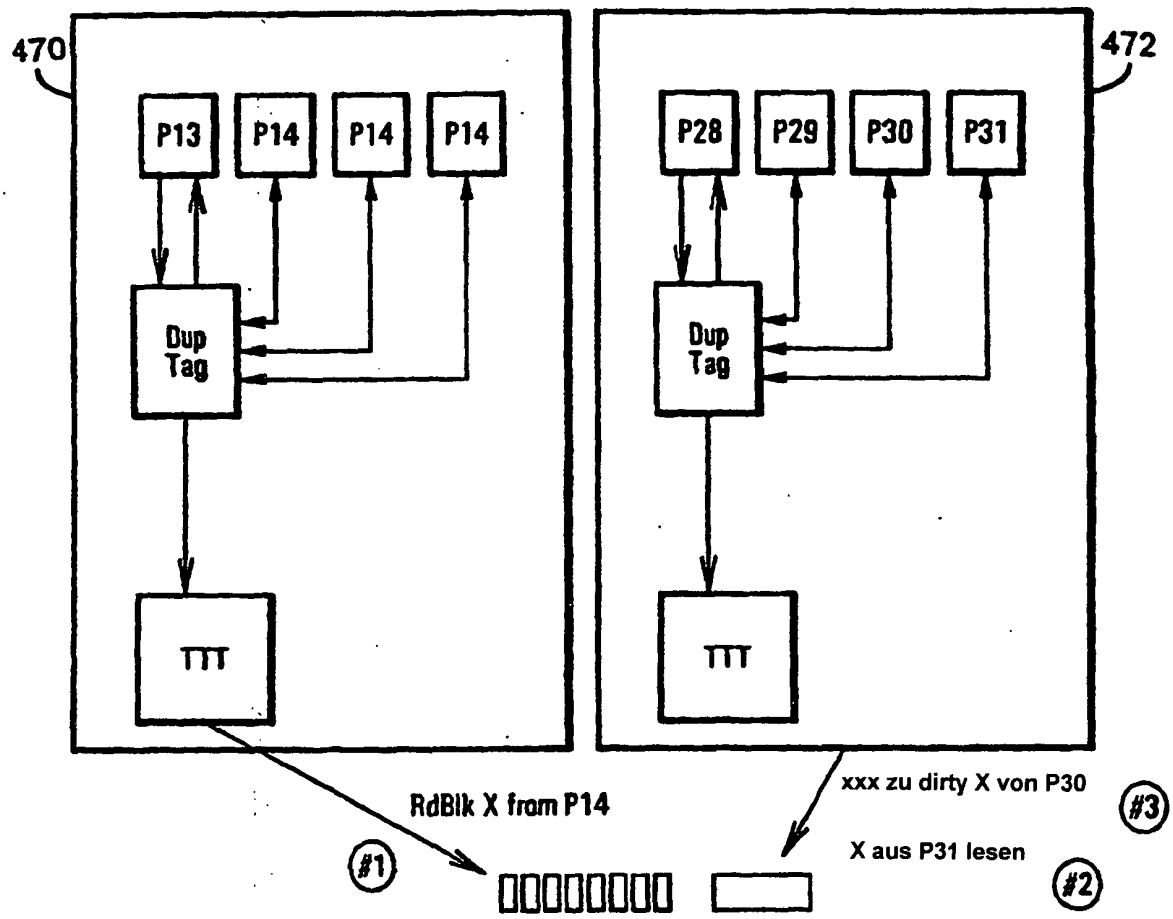


FIG. 28A

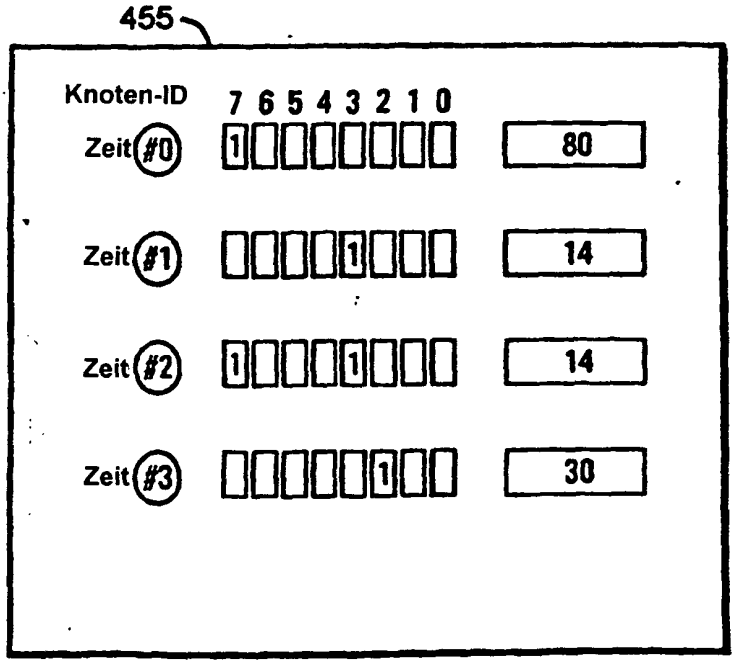


FIG. 28B

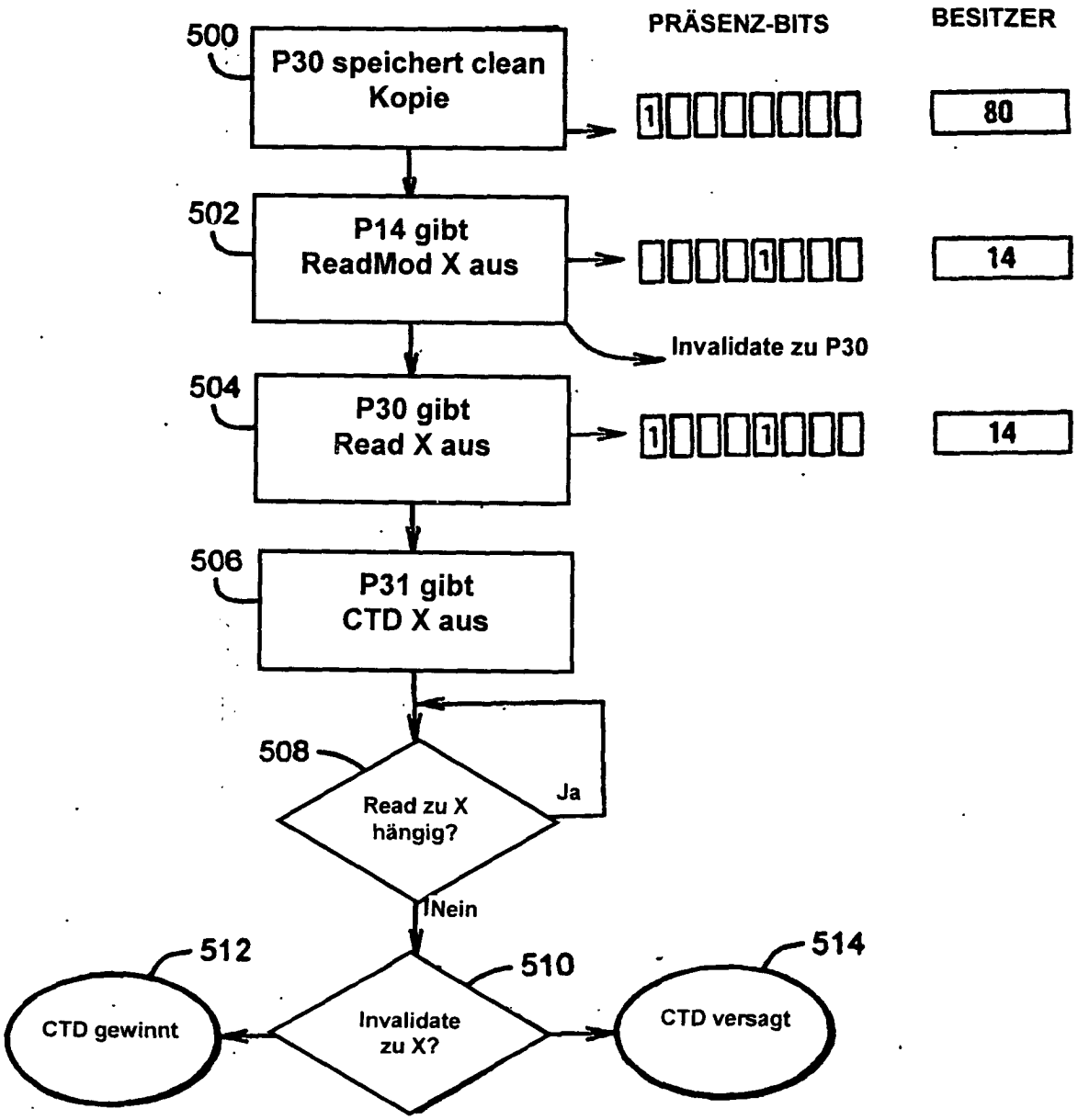

PRÄSENZ-BITS
BESITZER
500
P30 speichert clean Kopie
1
80
502
P14 gibt ReadMod X aus
1
14
Invalidate zu P30
504
P30 gibt Read X aus
1
1
14
506
P31 gibt CTD X aus
508
Read zu X hängig?
Ja
Nein
512
510
514
CTD gewinnt
Invalidate zu X?
CTD versagt

FIG. 29

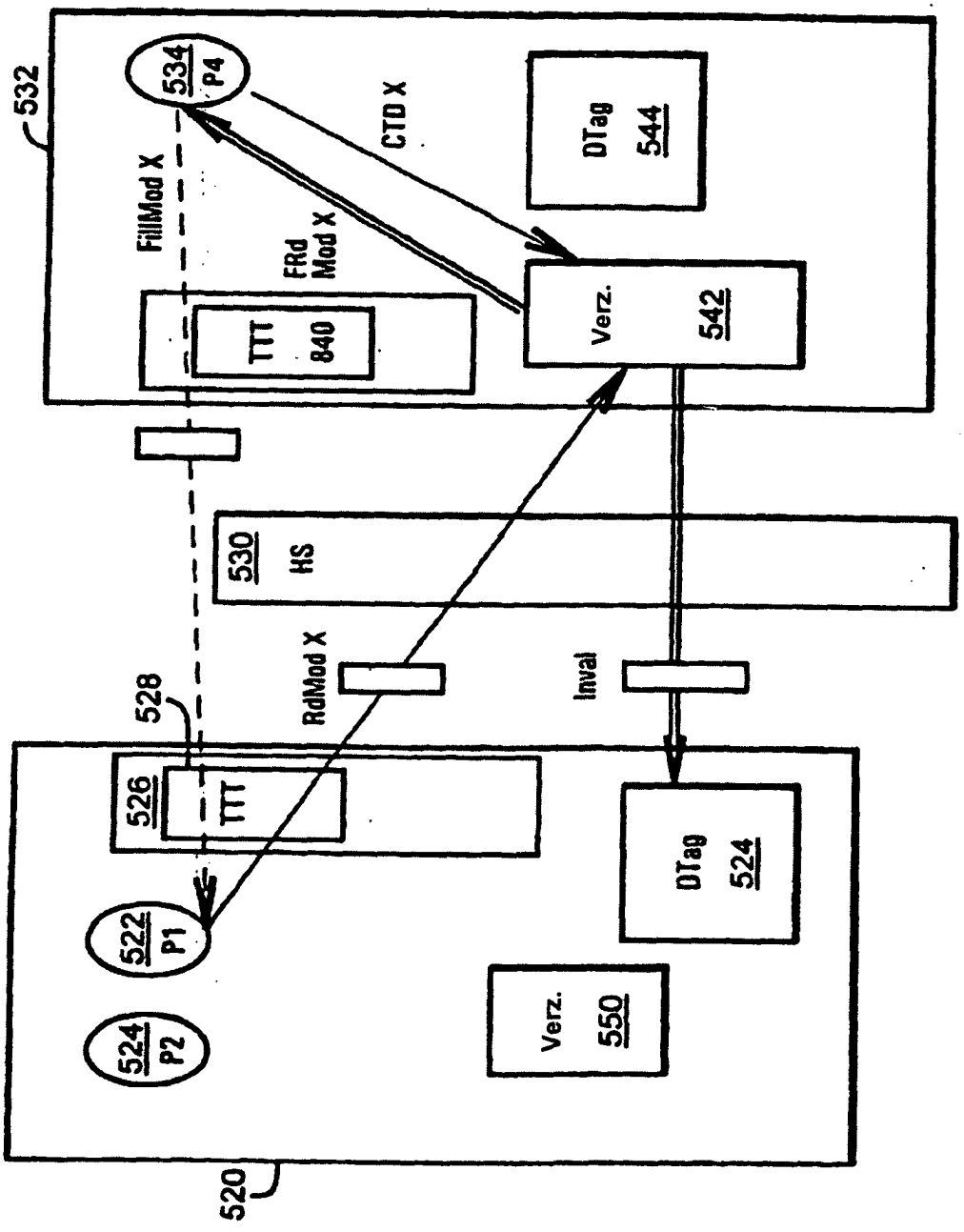

532
534
P4
CTD X
DTag
544
FillMod X
FRd
Mod X
TTT
840
Verz.
542
530
HS
RdMod X
Inval
528
526
TTT
DTag
524
522
P1
524
P2
Verz.
550
520

FIG. 30

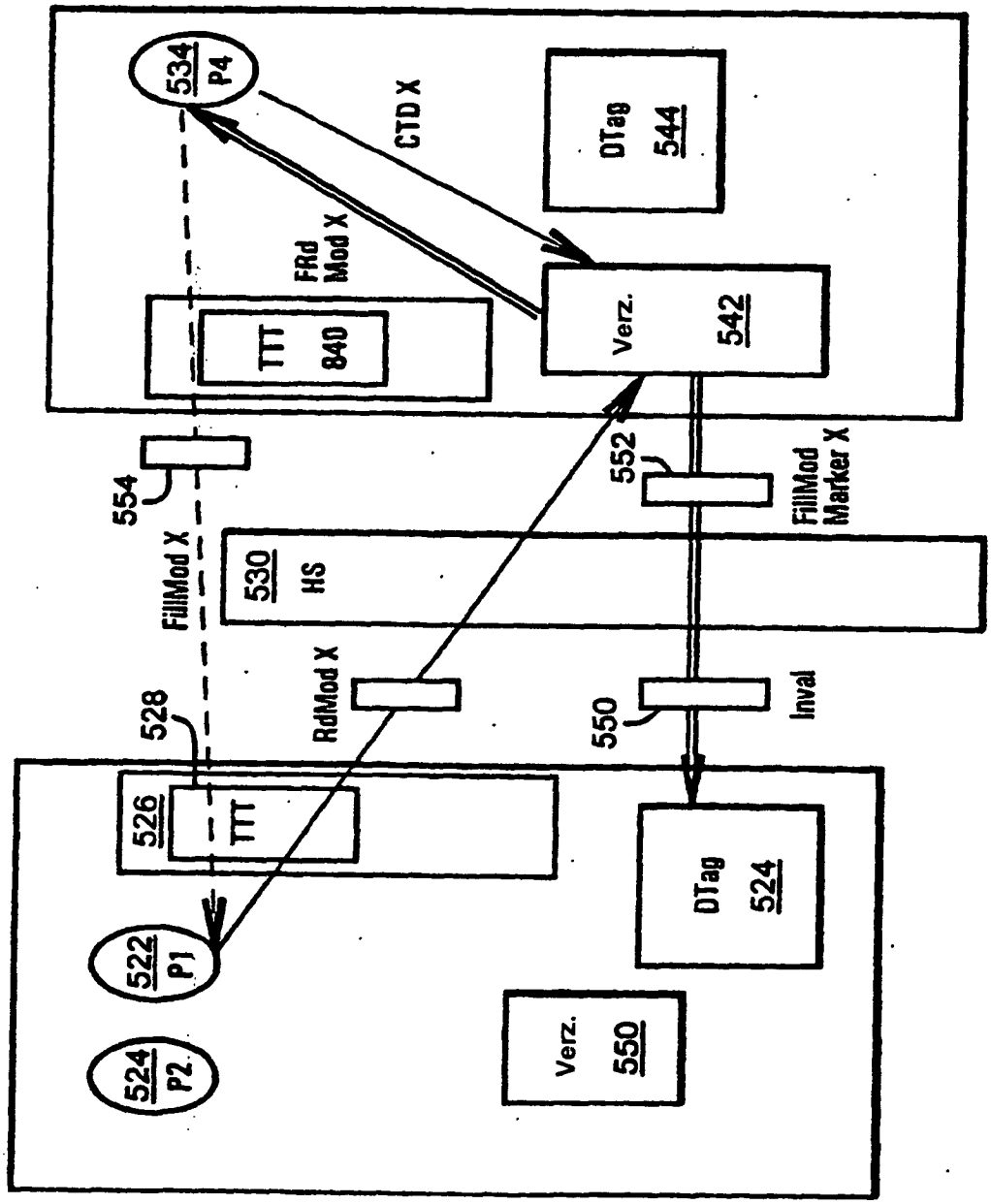

534
P4
CTD X
DTag
544
FRd
Mod X
TTT
840
Verz.
542
554
552
FillMod
Marker X
FillMod X
530
HS
RdMod X
528
550
Inval
526
TTT
DTag
524
522
P1
524
P2
Verz.
550

FIG. 31

Adresse	Befehl	Befehls-ID	Status-Bits					
			Fill here	Fill Marker Here	Shadow	Ack/Nack here	Fetch	Loop Consig
X	RdMod	P1						

FIG. 32

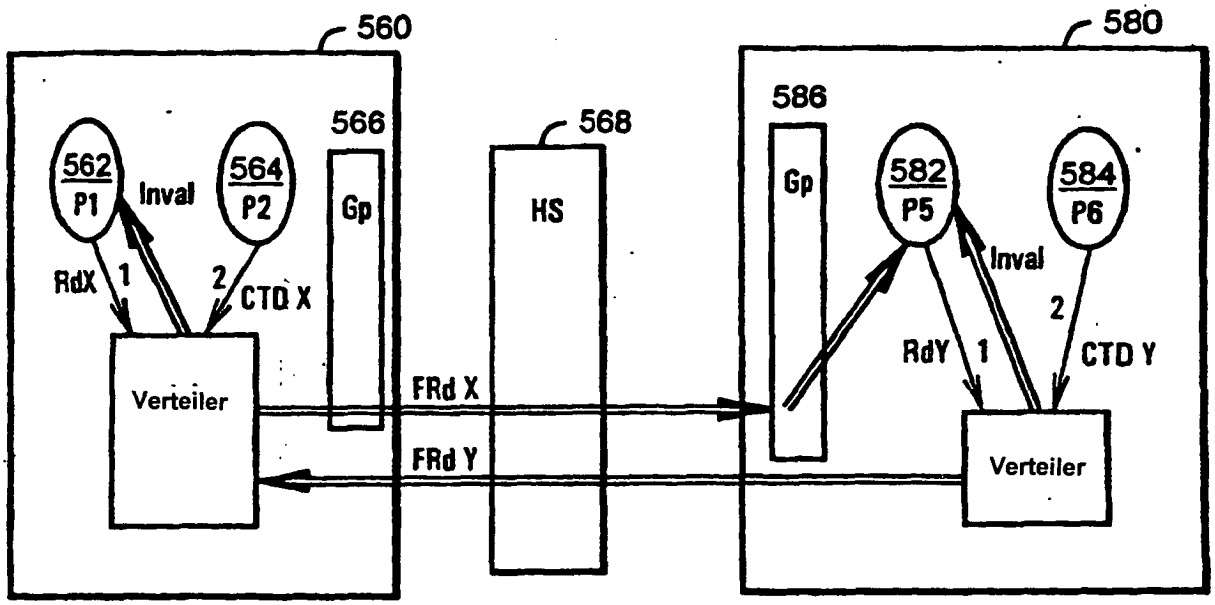


FIG. 33A

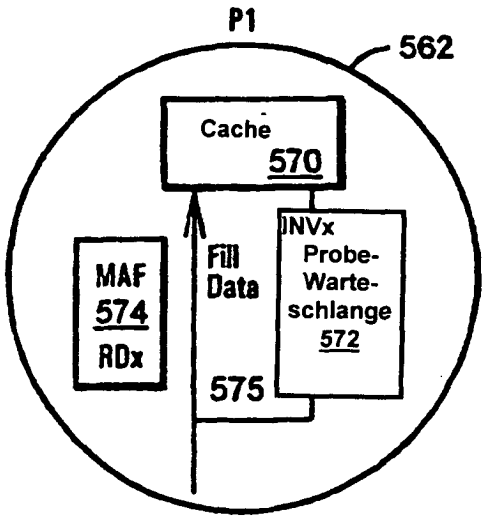


FIG. 33B

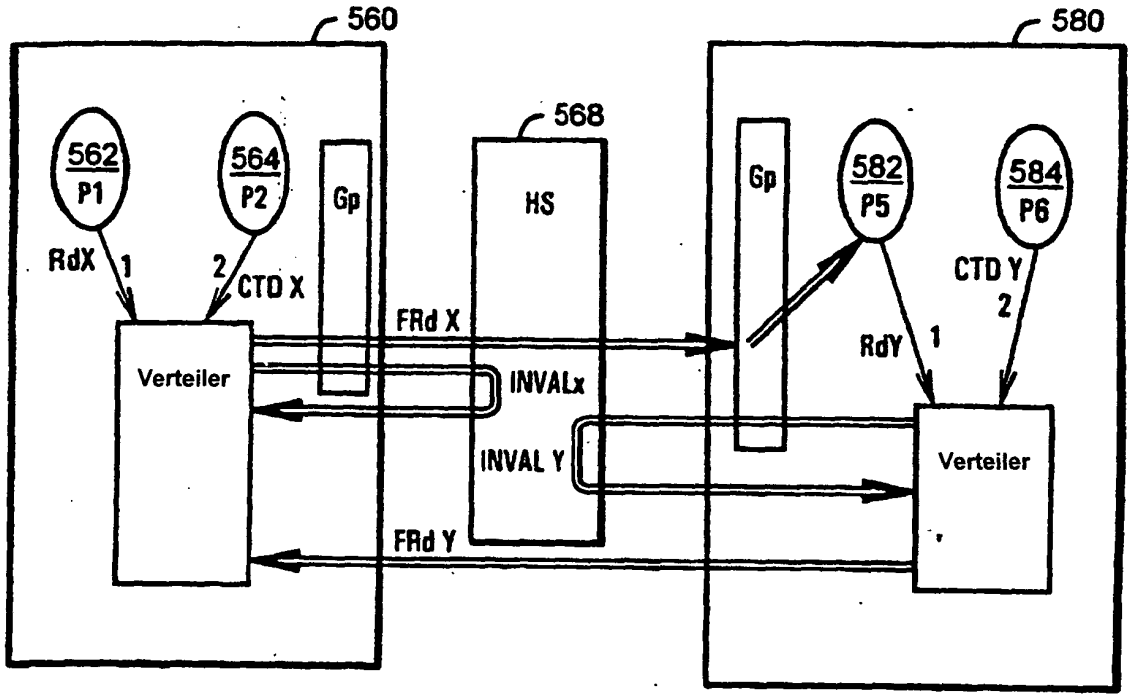


FIG. 34

Adresse	Befehl	Befehls-ID	Status-Bits					
			Fill here	Fill Marker Here	Shadow	Ack/Nack here	Fetch	Loop Consig
X	FRd	P1						
X	INVAL	P2			X			

FIG. 35

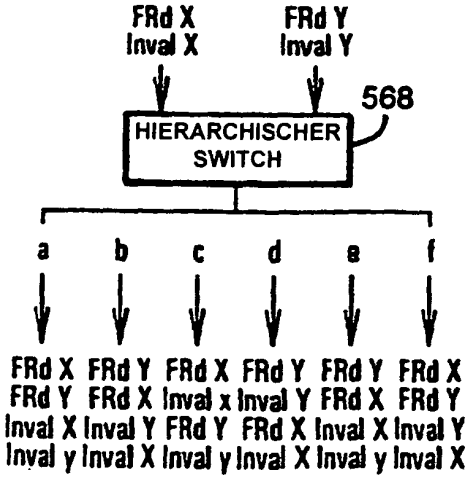


FIG. 36