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

(10) **DE 697 31 936 T2** 2005.06.23

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 854 611 B1**
(21) Deutsches Aktenzeichen: **697 31 936.9**
(96) Europäisches Aktenzeichen: **97 310 656.0**
(96) Europäischer Anmeldetag: **30.12.1997**
(97) Erstveröffentlichung durch das EPA: **22.07.1998**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **15.12.2004**
(47) Veröffentlichungstag im Patentblatt: **23.06.2005**

(51) Int Cl.[7]: **H04L 12/44**
H04L 12/40, H04L 12/56

(30) Unionspriorität:
774605 **30.12.1996** **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:
Walker, William J., Houston, Texas 77070, US; Kotzur, Gary B., Spring, Texas 77388, US; Hareski, Patricia E., Houston, Texas 77070, US; Mayer, Dale J., Houston, Texas 77070, US; Witkowski, Michael L., Tomball, Texas 77375, US

(54) Bezeichnung: **Netzwerkkoppelfeld mit Mehrfachbusarchitektur**

Beschreibung

[0001] Die vorliegende Erfindung betrifft das Gebiet von Netzwerk-Vorrichtungen und insbesondere einen Netzwerkschalter mit einer Mehrfachbus-Architektur.

[0002] Es gibt viele verschiedene Arten von Netzwerken und Netzwerksystemen, die sich in Dateien und Ressourcen teilen oder anderweitig eine Kommunikation zwischen zwei oder mehr Computern ermöglichen. Netzwerke können auf der Basis verschiedener Merkmale und Funktionen kategorisiert werden, z. B. Nachrichtenkapazität, Bereich, über den die Knoten verteilt sind, Knoten- oder Computertypen, Knotenbeziehungen, Architektur oder Struktur basierend auf Kabeltyp und Datenpaketformat, Zugriffsmöglichkeiten usw. Zum Beispiel bezieht sich der Bereich eines Netzwerks auf die Entfernung, über die die Knoten verteilt sind, z. B. Lokale Netzwerke (LANs) in einem Büro oder einer Etage eines Gebäudes, Weitbereichs-Netzwerke (WANs), die eine Hochschulanlage, eine Stadt oder einen Staat überspannen, Globale Netzwerke (GANs), die nationale Grenzen überspannen, usw.

[0003] Die Struktur eines Netzwerks bezieht sich im Allgemeinen auf die verwendete Verkabelung oder Medien und Medienzugang sowie die Paketstruktur der über die Medien zu sendenden Daten. Verschiedene Strukturen sind üblich, einschließlich Ethernet, das Koaxialkabel, verdrillte Kabelpaare oder faseroptische Kabel zum Betrieb bei 10 Megabit pro Sekunde (Mbps) (z. B. 10Base-T, 10Base-F) verwendet, oder schnelles Ethernet, das bei 100 Mbps (100Base-T, 100Base-FX) arbeitet. ARCnet (Attached Resource Computer Network) ist eine relativ billige Netzwerkstruktur, die Koaxialkabel, verdrillte Kabelpaare oder faseroptische Kabel zum Betrieb bei 2.5 Mbps verwendet. Tokenring-Topologien verwenden spezielles IBM-Kabel oder faseroptisches Kabel zum Betrieb bei 1–16 Mbps. Natürlich sind viele andere Arten von Netzwerken bekannt und verfügbar.

[0004] Jedes Netzwerk enthält im Allgemeinen zwei oder mehr Computer, oft als Knoten oder Stationen bezeichnet, die durch ausgewählte Medien oder verschiedene andere Netzwerk-Vorrichtungen miteinander verbunden sind, um die Daten zwischen den Knoten weiterzuleiten, zu übertragen, zu wiederholen, zu übersetzen, zu filtern usw. Der Begriff "Netzwerk Vorrichtung" betrifft gewöhnlich die Computer und ihre Netzwerk-Schnittstellenkarten (NICs) sowie verschiedene andere Vorrichtungen auf dem Netzwerk, z. B. Repeater, Brücken, Schalter, Router, Brouter, um einige Beispiele zu nennen. Ein Netzwerk, das entsprechend einem gegebenen Kommunikationsprotokoll arbeitet, kann unter Verwendung von einem oder mehreren Repeatern, Brücken oder Schaltern erweitert werden. Ein Repeater ist eine Hardware-Vorrichtung, die auf der physikalischen Schicht arbeitet und jedes empfangene Paket an jeden anderen Port weitersendet. Eine Brücke arbeitet auf der Datenverbindungsschicht des OSI-Referenzmodells und erhöht die Effizienz durch Filtern von Paketen, um die Menge an unnötiger Paketausbreitung auf jedem Netzwerksegment zu verringern.

[0005] Ein Netzwerkschalter funktioniert ähnlich, doch effizienter, wie eine Multiport-Brücke, die eine Vielzahl von Ports zum Verbinden mit mehreren ähnlichen Netzwerken zum Dirigieren des Netzwerkverkehrs unter den Netzwerken enthält. Ein Netzwerkschalter umfasst gewöhnlich eine mit den Ports über einen Bus verbundene Schaltmatrix und einen Speicher, um Netzwerkdaten, z. B. Ethernet-Pakete oder dergleichen, vorübergehend zu speichern. Gewöhnlich ist ein erhebliches Verarbeitungsvermögen erforderlich, um den Verkehr zu dirigieren und andere Aufgaben durchzuführen, z. B. Initialisierung, Konfigurierung, statistische Überwachung und Netzwerk-Verwaltung, um einige Beispiele zu nennen. Die Netzwerk-Verwaltung umfasst die Speicherverwaltung, Ausführung des Spannbaum-Algorithmusses nach dem IEEE (Institute of Electrical and Electronics Engineers) 802.1 Standard, Unterhaltung und Verwaltung der Verwaltungs-Informationsbasis-(MIB) oder MIB II Struktur usw.

[0006] Typische Schalter-Architekturen weisen einen Primärbus für allen Netzwerk- und Prozessorverkehr auf. Solche Overhead-Funktionen benötigen wenigstens einen mit dem Bus verbundenen Prozessor oder dergleichen, um die Ports, die Schalterstruktur und den Speicher zu überwachen und zu verwalten. Die Overhead-Funktionen benötigen erhebliche Prozessorzeit und Busbandbreite, die mit dem normalen Netzwerkverkehr in Konflikt geraten, wodurch die Leistung des Schalters verlangsamt und verschlechtert wird. Eine solche Leistungsverschlechterung führt oft zu einer bedeutsamen Zahl von fallen gelassenen Paketen, besonders bei starker Belastung.

[0007] Es besteht der Wunsch, einen Netzwerkschalter mit verbesserter Kapazität bereitzustellen, um den Netzwerkverkehr auch bei starker Belastung zu handhaben. Es ist daher erwünscht, einen Netzwerkschalter bereitzustellen, der den Netzwerkverkehr handhaben kann, während er auch Netzwerk-Overhead-Funktionen, z. B. Initialisierung, Konfigurierung, Überwachung und Netzwerk-Verwaltung, durchführt.

[0008] WO 9613922 A offenbart ein Computernetzwerk-Schaltsystem mit einem Schaltstruktur-Schaltkreis, der eine Vielzahl von Ports umfasst, die mit einem entsprechenden einer Vielzahl von LAN-Segmenten verbunden sind. Der Schaltstruktur-Schaltkreis dient zum Empfangen von Anforderungen für Datenübertragungsoperationen von der Vielzahl von Ports und zum Priorisieren der Anforderungen während einer Synchronisationsperiode. Es wird offenbart, dass jedes LAN-Segment nach einem unterschiedlichen LAN-Kommunikationsprotokoll arbeiten kann.

[0009] Gemäß US-A-5546385 wird ein Netzwerkschalter bereitgestellt, der umfasst: eine Vielzahl von Netzwerkports zum Empfangen und Senden von Daten, wobei jeder umfasst: eine Netzwerk-Schnittstelle, eine Datenbus-Schnittstelle und eine Prozessorport-Schnittstelle, wobei der Netzwerkschalter weiter einen Datenbus, der mit der Datenbus-Schnittstelle jedes der Vielzahl von Netzwerk-Ports verbunden ist, einen Prozessor, der mit einem Prozessorbus verbunden ist, der mit der Prozessorport-Schnittstelle jedes der Vielzahl von Netzwerkports verbunden ist, und einen Speicher umfasst, der mit einem Speicherbus verbunden ist.

[0010] Die vorliegende Erfindung ist gekennzeichnet durch einen Schalter-Manager, der mit dem Datenbus, dem Prozessorbus und einem weiteren Prozessorbus verbunden ist, der mit jedem der Vielzahl von Netzwerkports verbunden ist, und mit dem Speicherbus verbunden ist, um den Datenfluss zwischen der Vielzahl von Netzwerkports und dem Speicher zu steuern und dem Prozessor zu ermöglichen, auf jeden der Vielzahl von Netzwerk-Ports und den Speicher zuzugreifen, wobei der Speicher-Manager umfasst:
eine Datenbus-Schnittstelle, die mit dem Datenbus verbunden ist und umfasst:
eine Abfragelogik, die periodisch abfragt, um den Status jedes der Vielzahl von Netzwerk ports zu bestimmen, und
eine Steuerlogik, die mit der Abfragelogik verbunden ist, wobei die Steuerlogik den Datenfluss zwischen der Vielzahl von Netzwerkports, dem Prozessor und einer Speicherbus-Schnittstelle steuert,
wobei die Speicherbus-Schnittstelle mit dem Speicherbus und der Datenbus-Schnittstelle verbunden ist, und
eine Prozessorbus-Schnittstelle, die mit dem Prozessorbus, der Datenbus-Schnittstelle und der Speicherbus-Schnittstelle verbunden ist.

[0011] Auf diese Weise hat der Prozessor direkten und unabhängigen Zugriff auf die Netzwerk ports zur Überwachung, Bestimmung des Status, Konfiguration und Verwaltung, ohne wertvolle Bandbreite des Datenbusses zu verbrauchen.

[0012] Die Datenbus-Schnittstelle enthält vorzugsweise Empfangs- und Sendepuffer zum Übertragen von Daten, wenigstens eine Zustandsmaschine zum periodischen Abfragen der Ports, um ihren Status zu bestimmen, und Steuerlogik zum Steuern des Datenflusses zwischen den Ports und zwischen den Ports und dem Speicher. Die Speicherbus-Schnittstelle kann eine Speichersteuerung zum Steuern von Speicherzyklen des Speichers und einen Arbiter umfassen, um den Zugriff auf den Speicher durch die Speichersteuerung zu steuern. Die Speicherbus-Schnittstelle kann auch eine Empfangssteuerung zum Steuern des Datenflusses von der Datenbus-Schnittstelle zum Speicher und eine Sendesteuerung zum Steuern des Datenflusses vom Speicher zu der Datenbus-Schnittstelle enthalten. Die Speicherbus-Schnittstelle kann werter eine Auffrischungssteuerung zum Aufrechterhalten des Zustands des Speichers über den Speicherbus enthalten, um so den Prozessor von Auffrischungsfunktionen zu befreien.

[0013] Der Prozessorbus kann einen zwischen den Schalter-Manager und den Prozessor geschalteten Prozessorabschnitt und einen zwischen den Schalter-Manager und jeden der Ports geschalteten Portabschnitt enthalten. Die Prozessorbus-Schnittstelle des Schalter-Managers kann eine Prozessor-Schnittstelle, die mit dem Prozessor durch den Prozessorabschnitt des Prozessorbusses verbunden ist, und eine Port-Schnittstelle enthaften, die mit der Prozessor-Schnittstelle und mit jedem der Netzwerkports durch den Portabschnitt des Prozessorbusses verbunden ist. Die Prozessor- und Portbusabschnitte können die gleiche Große haben. In der gezeigten und hierin beschriebenen Ausführung weisen jedoch die Prozessor und Portabschnitte des Prozessorbusses verschiedene Breiten auf, wo die Prozessor-Schnittstelle eine Zustandsmaschine enthält, die Zyklen zwischen den Prozessor- und Portabschnitten des Prozessorbusses übersetzt. Jeder der Netzwerkports kann einen oder mehr Statistik-Zähler zum Verfolgen des Status und Betriebs seines entsprechenden Ports enthalten, wobei die Zähler verbunden und daher für den Portabschnitt des Prozessorbusses ohne werteres verfügbar sind. Auf diese Weise hat der Prozessor unabhängigen und vollständigen Zugriff auf jeden der Ports zum Durchführen von Overhead-Funktionen während des Betriebs, ohne Aktivitäten auf dem Datenbus zu stören.

[0014] Die Prozessorbus-Schnittstelle kann ferner dem Prozessor gestatten, auf den Datenbus und den Speicher durch die Speicherbus-Schnittstelle zuzugreifen. Das heißt, die Prozessorbus-Schnittstelle enthält geeig-

nete Sende- und Empfangspuffer und eine erste Steuerung zum Steuern des Datenflusses zwischen der Prozessorbus-Schnittstelle und der Datenbus-Schnittstelle und eine zweite Steuerung zum Steuern des Datenflusses zwischen der Prozessorbus-Schnittstelle und der Speicherbus-Schnittstelle.

[0015] Bei der hierin beschriebenen einzelnen Ausführung eines erfindungsgemäßen Netzwerkschalters umfasst die Vielzahl von Netzwerkports eine Gruppe von Ports, die entsprechend einem ersten mit dem ersten Datenbus verbundenen Protokoll arbeiten, und eine zweite Gruppe von Ports, die entsprechend einem zweiten Protokoll arbeiten. Ein zweiter Datenbus wird zum schnittstellenmäßigen Verbinden der zweiten Gruppe von Ports bereitgestellt, und eine Brückenvorrichtung ist zwischen den ersten und den zweiten Datenbus geschaltet. In der gezeigten Ausführung arbeitet die erste Gruppe von Ports entsprechend dem Ethernet-Standard bei 10 Mbps, während die zweite Gruppe entsprechend dem Ethernet-Standard bei 100 Mbps arbeitet, obwohl einzusehen ist, dass die vorliegende Erfindung nicht auf ein bestimmtes Protokoll oder Datenübertragungsgeschwindigkeit begrenzt ist.

[0016] Ein erfindungsgemäßes Netzwerksystem umfasst eine Vielzahl von Netzwerken, wobei jedes wenigstens eine Datenvorrichtung zum Senden und Empfangen von Datenpaketen und einen Netzwerkschalter, wie oben beschrieben, enthält, der mit den Netzwerken zum Übertragen der Datenpakete verbunden ist.

[0017] Die vorliegende Erfindung kann besser verstanden werden, wenn die folgende ausführliche Beschreibung der bevorzugten Ausführung in Verbindung mit den folgenden Zeichnungen in Betracht gezogen wird. Inhalt der Zeichnungen:

[0018] Fig. 1 ist ein vereinfachtes Schaltbild eines Netzwerksystems, das einen erfindungsgemäßen Netzwerkschalter enthält.

[0019] Fig. 2 ist ein genaueres Blockschaltbild des Netzwerkschalters von Fig. 1.

[0020] Fig. 3A ist ein Blockschaltbild einer exemplarischen Vierfach-Kaskaden-Vorrichtung von Fig. 2 zur Implementierung der Ports des Netzwerkschalters.

[0021] Fig. 3B ist ein Diagramm, das die Signale der in Fig. 3A gezeigten einzelnen Vierfach-Kaskaden-Vorrichtung veranschaulicht.

[0022] Fig. 3C ist ein exemplarisches Timing-Diagramm, das das Prozessor-Lesetiming der Vierfach-Kaskaden-Vorrichtung von Fig. 3A veranschaulicht.

[0023] Fig. 3D ist ein exemplarisches Timing-Diagramm, das das Prozessor-Schreibtiming der Vierfach-Kaskaden-Vorrichtung von Fig. 3A veranschaulicht.

[0024] Fig. 3E ist ein exemplarisches Timing-Diagramm, das das Prozessor-Burst-Lesezugriffstiming der Vierfach-Kaskaden-Vorrichtung von Fig. 3A veranschaulicht.

[0025] Fig. 3F ist ein exemplarisches Timing-Diagramm, das eine Pufferstatusabfrage jedes der Ports von Fig. 3A veranschaulicht.

[0026] Fig. 3G ist ein exemplarisches Timing-Diagramm, das einen gleichlaufenden Lese- und Schreibzyklus auf dem HSB von Fig. 2 veranschaulicht.

[0027] Fig. 3H ist ein Flussdiagramm, das eine Prozedur zum Ausführen eines gleichlaufenden Lese- und Schreibzyklusses auf dem HSB von Fig. 2 veranschaulicht.

[0028] Fig. 4 ist ein Blockschaltbild des Schalter-Managers von Fig. 2.

[0029] Fig. 5A ist ausführlicheres Blockschaltbild des Bussteuerungsblocks von Fig. 4.

[0030] Fig. 5B ist ein Diagramm, das Puffer in dem Speicher des Bussteuerungsblocks von Fig. 5A veranschaulicht.

[0031] Fig. 5C ist ein Zustandsdiagramm, das die Arbeitsweise der Empfangsabfrage-Zustandsmaschine in dem Bussteuerungsblock von Fig. 5A veranschaulicht.

[0032] Fig. 5D ist ein Zustandsdiagramm, das die Arbeitsweise der Sendeabfrage-Zustandsmaschine in dem Bussteuerungsblock von Fig. 5A veranschaulicht.

[0033] Fig. 6 ist ein ausführlicheres Blockschaltbild des Speichersteuerungsblocks von Fig. 4.

[0034] Fig. 7A–Fig. 7E sind ausführlichere Blockschaltbilder des Prozessorsteuerungsblocks von Fig. 4.

[0035] Fig. 8A ist ein vereinfachtes Blockschaltbild der Thunder-LAN-Portschnittstelle (TPI) von Fig. 2.

[0036] Fig. 8B ist ein ausführlicheres Blockschaltbild der TPI.

[0037] Fig. 8C ist ein Blockschaltbild, das die Konfiguration und Funktionalität jedes der Thunder-LANs (TLANs) von Fig. 2 veranschaulicht.

[0038] Fig. 8D ist ein Diagramm, das das allgemeine Format einer Steuerliste zum Ausführen durch jedes der TLANs veranschaulicht.

[0039] Fig. 8E ist ein Diagramm, das eine Definition von TPI-Peripheriekomponenten-Verbindungs-(PCI)Konfigurationsregistern veranschaulicht, die von der mit dem PCI-Bus von Fig. 2 verbundenen TPI verwendet werden.

[0040] Fig. 8F ist ein Diagramm, das die Definition der von der TPI verwendeten TPI-Steuerregister veranschaulicht.

[0041] Fig. 8G ist ein Flussdiagramm, das PCI-Initialisierungsoperationen der CPU von Fig. 2 veranschaulicht.

[0042] Fig. 8H ist ein Flussdiagramm, das eine Empfangsoperation für jedes der TLANs veranschaulicht.

[0043] Fig. 8I ist ein Flussdiagramm, das einen Empfangsdaten-Übertragungsvorgang über den Hochgeschwindigkeitsbus (HSB) von Fig. 2 veranschaulicht.

[0044] Fig. 8J ist ein Flussdiagramm, das einen Sendedaten-Übertragungsvorgang über den HSB veranschaulicht.

[0045] Fig. 8K ist ein Flussdiagramm, das einen Sendevorgang für jedes der TLANs veranschaulicht.

[0046] Fig. 9A–Fig. 9H sind Blockschaltbilder, die die Organisation des Speichers von Fig. 2 veranschaulichen.

[0047] Fig. 10 ist ein exemplarisches Blockschaltbild, das mehrere Sendepaketstrecken, die ein Rundsendepaket einschließen, veranschaulicht.

[0048] Fig. 11A und Fig. 11B sind Blockschaltbilder, die die Organisation des statischen Speichers von Fig. 6 veranschaulichen.

[0049] Fig. 12A ist ein Flussdiagramm, das die allgemeine Arbeitsweise des Netzwerkschalters von Fig. 2 zum Empfangen von Datenpaketen im Speicher und zum Senden von Datenpaketen in einer Durchschalt-Betriebsart veranschaulicht.

[0050] Fig. 12B ist ein Flussdiagramm, das die allgemeine Arbeitsweise des Netzwerkschalters von Fig. 2 zum Senden von Datenpaketen aus dem Speicher veranschaulicht.

[0051] Fig. 13 ist ein Flussdiagramm, das eine Hash-Lookup-Operation des Schalter-Managers von Fig. 2 veranschaulicht.

[0052] Fig. 14 ist ein Flussdiagramm, das eine Hash-Lookup-Operation zum Suchen von Hash-Tabelleneinträgen im Speicher von Fig. 2 veranschaulicht.

[0053] Fig. 1 zeigt ein vereinfachtes Netzwerkdiagramm eines Netzwerks **100**, das einen nach der vorliegen-

den Erfindung implementierten Netzwerkschalter **102** enthält. Der Netzwerkschalter **102** umfasst einen oder mehr "A" Ports **104**, jeweils zum Verbinden und Kommunizieren mit einem von mehreren "A" Netzwerken **106** durch ein geeignetes Mediensegment **108**. Jedes Mediensegment **108** ist irgendeine Art von Medium, z. B. ein verdrilltes Kabel oder faseroptisches Kabel usw. Die Ports **104** ermöglichen bidirektionale Kommunikation oder Datenfluss zwischen dem Netzwerkschalter **102** und jedem der Netzwerke **106**. Ein solcher bidirektionaler Datenfluss ist entsprechend einer von mehreren Betriebsarten, z. B. Halbduplex oder Vollduplex. Wie in Fig. 1 gezeigt, gibt es "j" + 1 Netzwerke **106**, die einzeln mit A-Netzwerk0, A-Netzwerk1, ..., A-Netzwerkj bezeichnet sind, wo jedes Netzwerk **106** mit dem Netzwerk schalter **102** durch einen entsprechenden der j + 1 Ports **104**, einzeln bezeichnet mit A-Port0, A-Port1, ..., A-Portj, verbunden ist. Der Netzwerkschalter **102** kann jede gewünschte Zahl von Ports **104** zum Verbinden mit bis zu einer zugehörigen Zahl von Netzwerken **106** enthalten. Bei der hier beschriebenen Ausführung ist j eine Ganzzahl gleich 23 für insgesamt 24 Ports zum Verbinden von bis zu Netzwerken **106**, wobei diese Ports kollektiv als Ports **104** oder einzeln als Ports Port0, Port1, Port2, ..., Port23 bezeichnet werden.

[0054] In ähnlicher Weise umfasst der Netzwerkschalter **102** weiter einen oder mehr "B" Ports **110**, jeder zum Koppeln an ein und Verbinden mit einem "B" Netzwerk **112** durch ein geeignetes Mediensegment **114**. Jedes Mediensegment kann wieder jede Art von Medium zum Verbinden von Netzwerk-Vorrichtungen sein, z. B. ein verdrilltes Kabelpaar ein faseroptisches Kabel usw. Die Ports **110** sind ebenfalls bidirektional, um einen Datenfluss zwischen dem Netzwerkschalter **102** und den Netzwerken **112** in einer ähnlichen Weise wie für die Ports **104** beschrieben zu ermöglichen. In der gezeigten Ausführung gibt es "k" + 1 Ports **110**, einzeln bezeichnet mit B-Port0, B-Port1, ..., B-Portk, zum Verbinden von bis zu k + 1 Netzwerken, bezeichnet mit B-Netzwerk0, B-Netzwerk1, ..., B-Netzwerkk. Der Netzwerkschalter **102** kann jede gewünschte Zahl von Ports **110** zum Verbinden mit bis zu einer zugehörigen Zahl von Netzwerken **112** enthalten. In der gezeigten spezifischen Ausführung ist k eine Ganzzahl gleich 3 für insgesamt 4 Ports **110** zum Verbinden von bis zu vier Netzwerken **112**. Die "A" Typ Ports und Netzwerke arbeiten bei einem unterschiedlichen Netzwerkprotokoll und/oder Geschwindigkeit als die "B" Typ Ports und Netzwerke. In der gezeigten spezifischen Ausführung arbeiten die Ports **104** und Netzwerke **106** entsprechend dem Ethernet-Protokoll bei 10 Megabit pro Sekunde (Mbps), während die Ports **110** und Netzwerke **112** entsprechend dem Ethernet-Protokoll bei 100 Mbps arbeiten. Die Ports B-Port0, B-Port1, ..., B-Port3 werden hier kollektiv als die Ports **104** und einzeln als Port24, Port25, ..., Port27 bezeichnet.

[0055] Die Netzwerke **106** und **112** enthalten eine oder mehr Datenvorrichtungen oder Datenendgeräte (DTE), die die Eingabe oder Ausgabe von Daten erlauben, oder jede Art von Netzwerk-Vorrichtung, um eine oder mehr Datenvorrichtungen miteinander zu verbinden. Jedes der Netzwerke, z. B. A-Netzwerk0 oder B-Netzwerk1, usw, kann daher einen oder mehr Computer, Netzwerk-Schnittstellenkarten (NICs), Workstations, Datei-Server, Modems, Drucker oder jede andere Vorrichtung enthalten, die Daten in einem Netzwerk empfängt oder sendet, z. B. Repeater, Schalter, Router, Hubs, Konzentratoren usw. Zum Beispiel sind, wie in Fig. 1 gezeigt, mehrere Computersysteme oder Workstations **120**, **122** und **124** mit dem entsprechenden Segment **108** von A-Netzwerkj verbunden. Die Computersysteme **120**, **122** und **124** können miteinander oder mit anderen Vorrichtungen von anderen Netzwerken durch den Netzwerkschalter **102** kommunizieren. Jedes Netzwerk **106** und **112** stellt daher eine oder mehrere durch ein oder mehrere Segmente verbundene Datenvorrichtungen dar, wobei der Netzwerkschalter **102** Daten zwischen irgendwelchen zwei oder mehr Datenvorrichtungen in jedem der Netzwerke **106** und **112** überträgt.

[0056] Der Netzwerkschalter **102** empfängt gewöhnlich Information von Datenvorrichtungen, die mit jedem der Ports **104** und **110** verbunden sind, und leitet die Information an einen oder mehr der anderen Ports **104** und **110**. Der Netzwerkschalter **102** filtert auch die Information durch Wegwerfen oder sonstwie Ignorieren von Information, die von einer Datenvorrichtung in einem Netzwerk **106** oder **112** empfangen wird und nur für Datenvorrichtungen in diesem gleichen Netzwerk bestimmt ist. Die Daten oder Information sind in der Form von Paketen, wo die einzelne Form jedes Pakets von dem durch ein gegebenes Netzwerk unterstützten Protokoll abhängt. Ein Paket ist ein vordefinierter Block von Bytes, der gewöhnlich aus Vorspann, Daten und Nachspann besteht, wobei das Format eines gegebenen Pakets von dem Protokoll abhängt, das das Paket erzeugte. Der Vorspann enthält gewöhnlich eine Zieladresse, die die Zieldatenvorrichtung identifiziert, und Quellenadresse, die eine Datenvorrichtung identifiziert, die das Paket hervorbringt, wobei die Adressen typisch Medienzugangssteuer-(MAC)Adressen sind, um die Einmaligkeit in der Industrie zu sichern. Ein für eine Zielvorrichtung bestimmtes Paket wird hierin als ein Unicast-Paket bezeichnet. Der Vorspann enthält weiter ein Gruppenbit, das angibt, ob das Paket ein für mehrfache Zielvorrichtungen bestimmtes Multicast- oder Broadcast-(BC)Paket ist. Wenn das Gruppenbit auf logisch eins (1) gesetzt wird, wird es als ein Multicast-Paket betrachtet, und wenn alle Zieladressenbits auch auf logisch 1 gesetzt sind, ist das Paket ein BC-Paket. Zu Zwecken der vorliegenden Erfindung werden jedoch Multicast- und BC-Pakete gleich behandelt und werden im Folgenden als BC-Pakete

bezeichnet.

[0057] Fig. 2 zeigt ein genaueres Blockschaltbild des Netzwerkschalters **102**. In der gezeigten Ausführung enthält der Netzwerkschalter **102** sechs ähnliche Vierfach-Steuerungs- oder Vierfach-Kaskaden-(QC)Vorrichtungen **202**, wobei jede vier der Ports **104** einschließt. Die QC-Vorrichtungen **202** können in jeder gewünschten Weise implementiert werden, z. B. als in ein einziges ASIC-(anwendungsspezifische integrierte Schaltung) Gehäuse integriert oder als getrennte integrierte Schaltungs-(IC)Chips, wie gezeigt. In der gezeigten Ausführung arbeitet jeder Port **104** bei 10 Mbps bei Halbduplex für einen Gesamtdurchsatz von 20 Mbps pro Port bei Vollduplex. Dies ergibt insgesamt 480 Mbps für alle sechs der QC-Vorrichtungen **202** bei Vollduplexbetrieb. Jede der QC-Vorrichtungen **202** enthält vorzugsweise eine mit einem QC/CPU-Bus **204** verbundene Prozessor-Schnittstelle und eine mit einem Hochgeschwindigkeitsbus (HSB) **206** verbundene Bus-Schnittstelle. Der HSB **206** enthält einen Datenabschnitt **206a** und verschiedene Steuer- und Statussignale **206b**. Der HSB **206** ist ein 32-Bit 33 MHz Bus zum Übertragen von über ein Gigabit von Daten pro Sekunde.

[0058] Der HSB **206** und der QC/CPU-Bus **204** sind weiter mit einem Ethernet-Paketschalter-Manager (EPSM) **210** verbunden, der in der gezeigten Ausführung als ASIC implementiert ist, obwohl die vorliegende Erfindung nicht auf eine bestimmte physikalische oder logische Implementierung begrenzt ist. Der EPSM **210** ist weiter mit einem Speicher **212** durch einen 32-Bit Speicherbus **214** verbunden, der einen Daten- und Adressenabschnitt **214a** und Steuersignale **214b** enthält. Der Speicher **212** enthält vorzugsweise 4 bis 16 Megabyte (MB) an dynamischem Direktzugriffspeicher (DRAM), obwohl, wenn gewünscht, mehr Speicher abhängig von den Bedürfnissen der einzelnen Anwendung hinzugefügt wird. Der EPSM **210** unterstützt jede von wenigstens drei verschiedenen Arten von DRAMs zur Implementierung des Speichers **212**, einschließlich schneller Seitenmodus-(FPM)Einzel-Inline-Speichermodule (SIMMs), die mit etwa 60 Nanosekunden (NS) arbeiten, Erweiterte Datenausgabe-(EDO)Modus DRAM SIMMs oder Synchronmodus- DRAM SIMMS. Synchrone DRAMs benötigen gewöhnlich einen 66 MHz Takt zum Erreichen einer Stoßdatenrate von 66 MHz oder 266 MB pro Sekunde. EDO DRAMs können mit entweder einem 33 oder 66 MHz Takt arbeiten, erreichen aber eine maximale Stoßdatenrate von 33 MHz oder 133 MB pro mit jeder Taktrate. FPM DRAMs können auch mit einem 33 oder 66 MHZ Takt arbeiten und erreichen eine maximale Stoßrate von 16 MHZ oder 64 MB pro Sekunde mit einem 33 MHz Takt und eine Stoßrate von 22 MHz oder 88 MB pro Sekunde mit einem 66 MHz Takt.

[0059] Der Speicherbus **214** umfasst einen Speicherdatenbus MD[31:0], Datenparitätssignale MD_PAR[3:0], Reihen- und Spaltenadresssignale MA[11:0], ein Schreibfreigabesignal MWE*, Bankauswählsignale RAS[3:0]*/SD_CS*[3:0], die entweder Reihensignale für FPM DRAM und EDO DRAM oder Chipauswahlen für synchrone DRAM sind, Speicherbyte-Steuersignale CAS[3:0]*/SD_DQM[3:0], die Spaltensignale für FPM und EDO oder DQM für synchrone DRAM sind, ein Reihensignal SD_RAS* nur für synchrone DRAM, ein Spaltensignal SD_CAS* nur für synchrone DRAM, ein Serial-Eingang-SIMM/DIMM-Anwesenheits-Erfassungssignal PD_SERIAL_IN und ein Parallel-Eingang-SIMM/DIMM-Anwesenheits-Erfassungssignal PD_LOAD*.

[0060] Der HSB **206** ist mit einer Thunder LAN (TLAN) Portschnittstelle (TPI) **220** verbunden, die weiter mit einem Peripheriekomponenten-Verbindungs-(PCI) Bus **222** verbunden ist, der Daten und Adresssignale **222a** und zugehörige Statussignale **222b** umfasst. Der PCI-Bus **222** ist mit vier TLANs **226** verbunden, die in jeder gewünschten Weise implementiert werden können. Die TLANs **226** sind vorzugsweise die TNETE100 ThunderLAN™ PCI Ethernet™ Controller, hergestellt von Texas Instruments, Inc, (TI), wo jedes einen der Ports **110** einschließt. Für den EPSM **210** arbeitet die TPI **220** in einer ähnlichen Weise auf dem HSB **206** als eine andere QC-Vorrichtung **202** zum Anschließen von vier Ports. Der EPSM **210** "sieht" daher effektiv sieben (7) Vierfach-Portvorrichtungen. In Bezug auf den PCI-Bus **220** emuliert die TPI **220** einen Standard-PCI-Bus in dem erforderlichen Ausmaß zum richtigen Betrieb der TLANs **226**, die normalerweise mit PCI-Speichervorrichtungen verbunden sind. Der PCI-Bus **222** muss daher nicht voll PCI-kompatibel sein. Der PCI-Bus **222** ist mit einem Prozessor oder zentralen Verarbeitungseinheit (CPU) **230** verbunden, die mit einem lokalen Prozessorbus **232** zum Verbinden der CPU **230** mit dem lokalen RAM **234**, einem lokalen Flash-RAM **236** und, wenn gewünscht, mit einer seriellen Portschnittstelle **238** verbunden ist. Die serielle Portschnittstelle **238** ist vorzugsweise ein UART oder dergleichen. In der gezeigten Ausführung ist die CPU eine 32-Bit, 33 MHz i960RP CPU von Intel, obwohl die CPU **230** jeder andere geeignete Prozessor sein kann.

[0061] Die CPU **230** handhabt gewöhnlich die Initialisierung und Konfigurierung der TPI **220** und des EPSM **210** beim Einschalten des Netzwerkschalters **102**. Die CPU **230** überwacht und gewinnt auch Statistiken und verwaltet und steuert die Funktionen der verschiedenen Vorrichtungen des Netzwerkschalters **102** während des Betriebs. Die CPU **230** aktualisiert ferner die Hash-Tabellendaten im Speicher **212** durch den EPSM **210**. Der EPSM **210** steuert jedoch den Zugriff auf den Speicher **212** und führt die DRAM-Auffrischungszyklen durch, um dadurch Auffrischungsoperationen von der CPU **230** zu entfernen. Andernfalls würde die CPU **230**

6–8 Buszyklen benötigen, um jeden Auffrischungszyklus durchzuführen, was wertvolle Prozessor-Ressourcen verbrauchen würde. Die CPU **230** agiert auch als ein zusätzlicher Netzwerkport für verschiedene Zwecke und wird hierin oft als Port28 bezeichnet. Die Ports **104**, **110** und die CPU **230** schließen daher kollektiv die Ports Port0–Port28 ein.

[0062] Die CPU **230** ist werter mit dem EPSM **210** durch einen CPU-Bus **218** verbunden, der einen Adress- und Datenabschnitt **218a** und zugehörige Steuer- und Statussignale **218b** umfasst. Der Adress- und Datenabschnitt **218a** wird vorzugsweise zwischen Adress- und Datensignale gemultiplext. Insbesondere umfasst der CPU-Bus **218** einen Adress/Daten-Bus CPU_AD[31:0], einen Adress-Strobe CPU_ADS* von der CPU **230**, Datenbyte-Freigaben CPU_BE[3:0], ein Lese/Schreib-Auswählsignal CPU_WR*, einen Burst-Letzte-Daten-Strobe CPU_BLAST*, ein Daten-Bereit-Signal CPU_RDY* und wenigstens ein CPU-Unterbrechungssignal CPU_INT*. In dieser Offenbarung bezeichnen normale Signalnamen anders als Daten- und Adresssignale positive Logik, wo das Signal als geltend gemacht betrachtet wird, wenn es hoch ist oder auf logisch eins (1) ist, und Signalnamen gefolgt von einem Stern (*) bezeichnen negative Logik, wo das Signal als geltend gemacht betrachtet wird, wenn es tief ist oder auf logisch null (0) ist: Die Funktions-Definition ist im Allgemeinen einfach und gewöhnlich durch den Signalnamen bestimmbar.

[0063] Fig. 3A ist ein Blockschaltbild einer exemplarischen QC-Vorrichtung **202** zur Implementierung von vier der Ports **104**, wobei die Vorrichtung sechsmal dupliziert ist, um die 24 Ports Port0–Port23 zu implementieren. Eine bestimmte Vorrichtung ist der L64381 Quad Cascade Ethernet Controller von LSI Logic Corporation (LSI). Eine verbesserte Vorrichtung ist der QE110 Quad Cascade Ethernet Controller, auch von LSI, der zusätzliche Merkmale und Fähigkeiten enthält, wie hierin beschrieben. Es wird jedoch angemerkt, dass die vorliegende Erfindung nicht auf eine bestimmte Vorrichtung zum Implementieren der Ports **104** begrenzt ist. In der gezeigten Ausführung enthält jede QC-Vorrichtung **202** einen Ethernet-Kern **300** für jeden der Ports **104**, wo der Ethernet-Kern **300** voll synchron ist und eine Medien-Zugriffssteuerung, einen Manchester-Coder/Decoder und verdrillte Paar/AUI (Attachment Unit Interface) Transceiver enthält. Jeder Ethernet-Kern **300** ermöglicht bidirektionale Kommunikation mit einem verbundenen Netzwerk **106** auf einem entsprechenden Segment **108**, und jeder ist mit einem entsprechenden 128-Bit Empfangs-FIFO (First-in-First-out) **302** und einem 128-Bit Sende-FOFO **304** verbunden. Jeder Ethernet-Kern **300** ist auch mit einem Block von Statistik-Zählern **306** verbunden, wo jeder Block von Statistik Zählern **306** 25 Zähler zum Bereitstellen von On-Chip-Wartung enthält. Die Zähler in jedem Block von Statistik-Zählern **306** erfüllen vorzugsweise die Forderungen des einfachen Netzwerk Verwaltungsprotokolls (SNMP). Jeder der FIFOs **302**, **304** ist weiter mit Busschnittstellenlogik **308** verbunden, die mit dem HSB **206** verbunden ist, um bidirektionalen Datenfluss zwischen jeder QG-Vorrichtung **202** und dem EPSM **210** zu ermöglichen. Jede QC-Vorrichtung **202** enthält Konfigurations- und Steuerlogik **310**, die programmierbares Konfigurieren ermöglicht, wie z. B. Einfügen von Quellenadressen, Einfügen einer Rahmenprüfsequenz (FCS), sofortiges Neusenden bei Kollision, Busübertragungsgröße und Sendepuffer-Schwellengröße.

[0064] Die Konfigurations- und Steuerlogik **310** und alle Blöcke von Statistik-Zählern **306** und die FOFOs **302**, **304** sind mit dem QC/CPU-Bus **204** verbunden. Der EPSM **210** stellt eine getrennte Schnittstelle zwischen dem CPU-Bus **218** und dem QC/CPU-Bus **204** bereit. Auf diese Weise hat die CPU **230** vollen Zugang, um die Aktivitäten jeder der QC-Vorrichtungen **202** und somit jedes der Ports **104** zu initialisieren, zu überwachen und zu modifizieren. Der QE110 Quad Cascade Ethernet Controller enthält eine zusätzliche Verbindung **320** zwischen der Konfigurations- und Steuerlogik **310** zum Erfassen eines Rückstau-Anzeichens, um eine Hemmungssequenz geltend zu machen, um ein grade gesendetes Paket zu beenden, wenn das Rückstau-Anzeichen rechtzeitig empfangen wird. Das Rückstau-Anzeichen ist vorzugsweise ein auf dem HSB **206** ausgeführter Rückstau-Zyklus, obwohl jedes von mehreren Verfahren benutzt werden kann, um einen Rückstau anzuzeigen, z. B. ein getrenntes Signal oder dergleichen.

[0065] Es wird angemerkt, dass die Hemmungssequenz während der ersten 64 Bytes des Datenpakets gesendet werden sollte, das an einem Port empfangen wird, als "früh" oder rechtzeitig zu betrachten ist. Die ersten 16 Byte (4 DWORDs) sind nötig, bevor eine später beschriebene Hash-Lookup-Prozedur durch den EPSM **210** durchgeführt wird. Jedes Datenbit wird in etwa 100 ns über Ethernet 10Base-T übertragen; sodass die ersten 16 Byte in etwa 13 µs übertragen werden. 64 Byte werden in etwa 51 µs empfangen, sodas der Netzwerkschalter **102** etwa 38 µs hat, um die ersten 16 empfangenen Bytes zu übertragen, die Hashing-Prozedur durchzuführen, den Rückstau-Zyklus auszuführen und schließlich die Hemmungssequenz geltend zu machen. Da ein Hash-Lookup etwa 1–2 µs braucht, um zu vollenden, ist fast immer genug Zeit vorhanden, um die Hemmungssequenz rechtzeitig zu senden. Das rechtzeitige Geltendmachen der Hemmungssequenz ist jedoch nicht garantiert, sodass die Möglichkeit besteht, dass Pakete infolge einer Schwellenverletzungsbedingung fallen gelassen werden. Wenn der Rückstau-Zyklus spät ausgeführt wird, weist der Port den Rückstau-Zyklus zu-

rück, und der Netzwerkschalter **102** lässt das Paket fallen, wenn er außerstande ist, das Paket anzunehmen. Der Netzwerkschalter kann dieses Paket annehmen, da eine Schwellenbedingung ein frühes Anzeichen ist und daher Speicher vorhanden sein kann, um das Paket zu speichern.

[0066] Wenn der Rückstau-Zyklus in einer rechtzeitigen Weise ausgeführt wird, und wenn der Port in Halbduplex arbeitet, macht die Konfigurations- und Steuerlogik **310** als Reaktion einen Kollisionsbefehl an einem der Ethernet-Kerne **300** eines angegebenen Ports **104** geltend. Der Ethernet-Kern **300**, der den Kollisionsbefehl empfängt, macht dann die Hemmungssequenz geltend, um ein Paket zu beenden, das durch diesen Port **104** empfangen wird. Wenn der Rückstau-Zyklus in dem 64-Byte Fenster ausgeführt wird, zeigt der Port dem EPSM **210** an, dass der Rückstau-Zyklus für diesen Port ausgeführt wird, durch Geltendmachen eines Abort-Signals ABORT_OUT* auf dem HSB **206**. Wenn der Rückstau-Zyklus außerhalb des 64-Byte Fensters liegt und daher nicht rechtzeitig geltend gemacht wird, wird das ABORT_OUT* Signal nicht geltend gemacht, und der EPSM **210** lässt das Paket fallen. EPSM **210** lässt das Paket meistens fallen, wenn ein Versuch, Rückstau geltend zu machen, fehlschlägt. Obwohl erwünscht ist, so wenig Pakete wie möglich zur maximalen Effizienz fallen zu lassen, wird ein fallen gelassenes Paket schließlich auf höheren Netzwerkstufen in der hervorbringenden Datenvorrichtung erfasst und ist daher für den Gesamtbetrieb des Netzwerksystems **100** nicht fatal. Die hervorbringende Vorrichtung erkennt, dass das Paket fallen gelassen wurde und sendet ein oder mehr Pakete einschließlich des fallen gelassenen Pakets neu.

[0067] Die Busschnittstellenlogik **308** enthält vorzugsweise Lese-Latches **324** und Schreib-Latches **326** zur Implementierung eines gleichlaufenden Lese- und Schreibzyklusses auf dem HSB **206**, wie weiter unten beschrieben. Diese Latches speichern PORT_NO[1:0] Signale, die auf dem HSB **206** bei bestimmten Zyklen eines ersten Taktsignals (CLK_1) geltend gemacht werden. Das Signal CLK_1 ist der Haupttakt für den HSB **206** und arbeitet in der gezeigten Ausführung typisch bei etwa 30–33 MHz. Da das Signal CLK_1 der Haupttakt ist, wird es im Folgenden einfach als CLK-Signal bezeichnet. Ein zweites Taktsignal CLK_2 wird auch für die Schnittstelle zum Speicher **212** verwendet und arbeitet bei der doppelten Frequenz des CLK-Signals oder bei etwa 60–66 MHz.

[0068] Fig. 3B ist ein Diagramm, das die Signale der in Fig. 3A gezeigten einzelnen Vierfach-Kaskaden-Vorrichtung **202** veranschaulicht. Die Signale sind in mehrere Funktions- und Busabschnitte geteilt, die mit dem QC-Bus **204** verbundene Prozessor-Schnittstellensignale, mit den vier Ports **104** verbundene Netzwerk-Schnittstellensignale, Statussignale, Takt- und Prüfsignale, mit dem HSB-Bus **206** verbundene Busschnittstellensignale und gemischte Signale umfassen.

[0069] Den QC-Bus **204** betreffend schreibt der EPSM **210** Daten in die Register und Zähler **306**, **310** der QC-Vorrichtung **202** durch Datensignale PDATA[15:0] und liest Daten daraus aus. Das Signal READ* wird für einen Schreibvorgang hoch und für einen Lesevorgang tief gesetzt. Das einzelne Register in der QC-Vorrichtung **202** durch eine auf ADR[5:0] Signalen geltend gemachte Adresse bestimmt. Geltendmachen eines Adress-Strobesignals ADRS_STROBE* zusammen mit dem entsprechenden der mehreren Chipauswählsignale CHIP_SELECTm* veranlasst die QC-Vorrichtung **202**, die ADRS-Signale zu speichern. Ein an den Signalnamen angehängtes "m" bezeichnet gewöhnlich mehrfache Signale eines einzelnen Typs. Zum Beispiel gibt es sechs getrennte CHIP_SELCT[5:0]* Signale, jedes zum Adressieren einer betreffenden der sechs QC-Vorrichtungen **202**. Ein Signal PREADY* wird durch die QC-Vorrichtung **202** für einen Zyklus eines CLK-Signals während eines Schreibzyklusses nach der steigenden CLK-Flanke, auf der die verlangten Daten gespeichert werden, tief geltend gemacht. Für einen Schreibzyklus macht die QC-Vorrichtung **202** PREADY* für einen CLK-Zyklus tief geltend, nachdem sie Daten auf den PDATA-Bus gelegt hat.

[0070] Fig. 3C ist ein exemplarisches Timing-Diagramm, das einen Prozessor-Lesezyklus für eine QC-Vorrichtung **202** veranschaulicht, und Fig. 3D ist ein exemplarisches Timing-Diagramm, das einen Prozessor-Schreibzyklus veranschaulicht. Fig. 3D ist ein exemplarisches Timing-Diagramm, das einen Prozessor-Stoßlesezugriffszyklus für eine QC-Vorrichtung **202** veranschaulicht. Diese Timing-Diagramme sind nur exemplarisch und werden gezeigt, um eine allgemeine Funktionalität und nicht ein bestimmtes Timing oder bestimmte Signaleigenschaften zu veranschaulichen.

[0071] Zurück auf Fig. 3B verweisend umfassen die Netzwerk-Schnittstellensignale die negativen und positiven Kollisionsschwellensignale, das Kollisionsbezugssignal, das serielle Daten-Ein-Signal, die negativen und positiven Manchester-codierten Datensignale, die positiven und negativen Datenschwellensignale, das Datenschwellenbezugssignal, die positiven und negativen Präemphasesignale und die Verdrillte-Paar/AUI-Modusauswählsignale für jeden der mit [3:0] bezeichneten vier Ports der QC-Vorrichtung **202**. Jede QC-Vorrichtung empfängt das CKL-Signal und hat einen CLOCK_20 MHz Eingang, der ein 20 MHz Taktsignal empfängt, um

80, 20 und 10 MHz interne Taktsignale zur Verwendung durch die Ports **104** zu erzeugen. Jeder Ethernet-Kern **300** erfasst eine auf dem entsprechenden Segment **108** auftretende Kollision und sendet eine Hemmungssequenz entprechend dem Ethernet CSMA/CD-(Carrier Sense Multiple Access/Collision Detect) Verfahren.

[0072] Die mit dem HSB **206** verbundenen Busschnittstellensignale betreffend beendet eine QC-Vorrichtung vorzeitig ein ganzes Paket durch Geltendmachen des Signals ABORT_OUT*. Der EPSM **210** beendet vorzeitig den laufenden Buszyklus durch Geltendamchen eines Abortsignals ABORT_IN*. In einer Ausführung sind die QC-Vorrichtungen **202** QE110 Vorrichtungen, die ersonnen sind, dem EPSM **210** zu ermöglichen, ein Paket, das empfangen wird, vorzeitig zu beenden, durch Ausführen eines Rückstau-Zyklusses auf dem HSB **206**. Dieser einzelne Typ von Rückstau-Fähigkeit ist ein "Paket für Paket" oder dynamischer "pro Port" Rückstau, der das Rückweisen eines Pakets erlaubt, das an einem Port empfangen wird. L64381-Vorrichtungen umfassen ein Selbseinfügungs-Rahmenprüfsequenzsignal (AI_FCS_IN*), das werter unten beschrieben wird. QE110-Vorrichtungen ersetzen das AI_FCS_IN* Signal mit einem Signal FBPN*, das benutzt wird, um die gleichen Funktionen wie das Siggnal AI_FCS_IN* auszuführen, aber auch benutzt wird, um einen Rückstau-Zyklus und eine erhöhte Paketflut anzuzeigen. Natürlich können alternative Verfahren verwendet werden, um den hierin beschriebenen dynamischen Rückstau zu implementieren. Das heißt, der EPSM **210** macht das Signal FBPN* während eines Lesezyklusses geltend, um einen Rückstau-Anforderungszyklus auszuführen. Wenn das Signal ABORT_OUT* durch die entsprehende QC-Vorrichtung **202** während der Datenphase des Lesezyklusses geltend gemacht wird, dann wurde die Rückstau-"Anforderung" durch die QC-Vorrichtung **202** gewährt, die dann eine Hemmungssequenz geltend macht, um das Paket vorzeitig zu beenden. Wenn das Signal ABORT_OUT* nicht geltend gemacht wird, lässt der EPSM **210** das Paket fallen.

[0073] Der EPSM **210** macht ein Status-Strobesignal STROBE* an allen QC-Vorrichtungen **202** und der TPI **220** geltend, die alle mit dem Status ihrer vier Ports **104** oder **110** (im Fall der TPI **220**) in gemultiplexter Weise auf Signale PKT_AVAILm* und BUF_AVAILm* antworten, wenn das Signal STROBE* auf der steigenden Flanke des CLK-Signals geltend gemacht wird. Es gibt ein getrenntes Signal für jede QC-Vorrichtung **202**, einen Satz für die TPI **220** und einen ähnlichen Satz für die CPU **230**, die für einige Operation als ein weiterer Port agiert. Das heißt, die Signale PKT_AVAILm* und BUF_AVAILm* enthalten Signale PKT_AVAIL[5:0]* und BUF_AVAIL[5:0]* für die QC-Vorrichtungen, Signale TPI_PKT_AVAIL* und TPI_BUF_AVAIL*, andernfalls als PKT_AVAIL[6]* und BUF_AVAIL[6]* bezeichnet, für die TPI **220** und Signale PCB_PKT_AVAIL* und PCB_BUF_AVAIL*, andernfalls als PKT_AVAIL[7]* und BUF_AVAIL[7]* bezeichnet, die der CPU **230** entsprechen, für insgesamt 8 Signale pro Signaltyp.

[0074] Auf diese Weise umfasst der HSB **206** Signale PKT_AVAIL[0]* und BUF_AVAIL[0]* für die erste QC-Vorrichtung **202**, um auf die vier Ports Port0–Port3 zuzugreifen, der HSB **206** umfasst Signale PKT_AVAIL[1]* und BUF_AVAIL[1]* für die nächste QC-Vorrichtung **202**, um auf die nächsten vier Ports Port0–Port7 zuzugreifen, usw., die TPI **220** umfasst Signale PKT_AVAIL[6]* und BUF_AVAIL[6]*, um auf die Ports Port24–Port27 zuzugreifen, und der EPSM **210** enthält interne Signale PKT_AVAIL[7]* und BUF_AVAIL[7]* für die CPU **230**. Bis zu vier Bits werden auf jedem der Signale entsprechend den vier Ports, getrennt durch jeweilige Zyklen des CLK-Signals, gemultiplext.

[0075] Als Reaktion auf das STROBE* Signal enthält die Busschnittstellenlogik **308** Portstatuslogik **303** zum Multiplexen von vier Statusbits auf einem betreffenden der BUF_AVAIL[5:0]* Signale, um anzuzeigen, ob jeder von ihren entsprechenden Sende-FIFOs **304** für den betreffenden Port genug freien Raum zur Verfügung hat, um Daten zu speichern. Die Portstatuslogik **303** ist entweder für alle vier Ports, wie gezeigt, zentralisiert oder ist unter den Ports verteilt. Die Bestimmung von freiem Raum erfolgt entsprechend einem Konfigurationsregister in der Busschnittstellenlogik **308**, das eine Busübertragungsfeldgröße (TBUS) speichert, die vorzugsweise durch die CPU **230** zu 16, 32 oder Bytes konfiguriert wird. In ähnlicher Weise enthält als Reaktion auf das STROBE* Signal die TPI **220** ähnliche Portstatuslogik **820** (Fig. 8B), die mit dem HSB **206** zum Multiplexen von vier Statusbits auf dem BUF_AVAIL[6]* Signal, um anzuzeigen, ob jeder ihrer internen Sende-FIFOs, unten beschrieben, genug freien Raum hat, um Daten für die entsprechenden der TLANs **226** für die jeweiligen Ports Port24–Port27 zu speichern. Für die CPU **230** oder Port28 macht ein PCB **406** (Fig. 4) in dem EPSM **210** ein einzelnes Statusbit auf dem BUF_AVAIL[7]* Signal geltend, um anzuzeigen, ob ein interner PCB-Sende-FIFO in dem EPSM **210** verfügbaren Raum hat, um Daten für die CPU **230** zu speichern.

[0076] In einer ähnlichen Weise multiplext als Reaktion auf das STROBE* Signal die Portstatuslogik **303** der Busschnittstellenlogik **308** in jeder QC-Vorrichtung **202** vier Statusbits auf einem entsprechenden der PKT_AVAIL[5:0]* Signale, die anzeigen, ob jeder ihrer Empfangs-FIFOs **302** für den betreffenden Port genug Daten entsprechend dem TBUS-Wert hat, um empfangene Daten für eine Busübertragung auf dem HSB **206** zu übertragen. Desgleichen multiplext die TPI **220** vier Statusbits auf dem PKT_AVAIL[6]* Signal, das anzeigt,

ob ihre internen Empfangs-FIFOs genug Daten von den betreffenden Ports Port23–Port27 empfangen haben, um sie auf dem HSB **206** zu übertragen. Für die CPU **230** macht der PCB **406** im EPSM **210** ein einzelnes Statusbit auf dem PKT_AVAIL[7]* Signal geltend, um anzuzeigen, ob ein interner PCB-Empfangs-FIFO im EPSM **210** genug Daten von von der CPU **230** für eine HSB **206** Busübertragung empfangen hat.

[0077] Fig. 3F ist ein exemplarisches Timing-Diagramm, das eine Pufferstatusabfrage der QC-Vorrichtung **202** und der TPI **220**, einschließlich Geltendmachung des STROBE* Signals durch den EPSM **210** und Antwort durch jede der QC-Vorrichtungen **202** auf das Geltendmachen jeweiliger PKT_AVAILm* und BUF_AVAILm* Signale durch die TPI **220** veranschaulicht. Die Verweise auf Port0, Port1, Port2 und Port3 in Fig. 3F sind die vier betreffenden Ports einer bestimmten QC-Vorrichtung **202** oder der TPI **220**. Der PCB **406** antwortet in einer ähnlichen Weise, außer dass sein Port für alle vier Phasen aktiv ist. Das STROBE* Signal ist pegelgetriggert und daher auf der ersten steigenden Flanke des CLK-Signals tief gesamplet. Es wird angemerkt, dass das Timing-Diagramm von Fig. 3F nur exemplarisch ist und gezeigt wird, um die allgemeine Funktionalität und nicht ein bestimmtes Timing oder bestimmte Signaleigenschaften zu veranschaulichen. Zum Beispiel ist das STROBE* Signal periodisch und wird typischerweise für mehr als einen CLK-Zyklus im Betrieb der gezeigten Ausführung tief geltend gemacht.

[0078] Wieder auf Fig. 3B verweisend wird das PORT_BUSY* Signal benutzt, um anzuzeigen, ob der jeweilige Port im Halbduplexmodus sendet oder empfängt, oder wenn der Port im Vollduplexmudus sendet. Lesedatensignale READ_OUT_PKT[5:0]* werden von dem EPSM **210** geltend gemacht, um eine betreffende QC-Vorrichtung **202** zu informieren, Daten von einem betreffenden Empfangs-FIFO **302** auf die Datensignale DATA[31:0] zu legen. In einer ähnlichen Weise werden Schreibdatensignale WRITE_IN_PKT[5:0] durch den EPSM **210** geltend gemacht, um eine betreffende QC-Vorrichtung **202** zu informieren, Daten von den Datensignalen DATA[31:0] in einen betreffenden Sende-FIFO **304** zurückzugewinnen. Auch werden ähnliche Signale PCG_RD_OUT_PKT*, PVB_WR_IN_PKT* und TPI_RFAD_OUT_PKT*, TPI_WRITE_IN_PKT* für die TPI **220** bzw. die CPU **230** eingeschlossen. Alle Lese- und Schreibsignale werden kollektiv als READ_OUT_PKTm* bzw. WRITE_IN_PKTm* bezeichnet. Die PORT_NO[1:0] Bits geben an, welcher einzelne Port **104** für einen auf dem HSB **206** ausgeführten Zyklus adressiert wird.

[0079] Ein Signal SOP* gibt den Start des Pakets an, wenn der Anfang oder Vorspann eines Pakets auf dem HSB **206** übertragen wird. Das AI_FCS_IN* Signal wird typischerweise mit dem SOP* und einem der WRITE_IN_PKTm* Signale geltend gemacht, um eine L64381-Vorrichtung (für eine Implementierung der QC-Vorrichtungen **202**) zu veranlassen, automatisch einen CRC-(zyklische Redundanzprüfung)Wert aus den Daten in dem Paket zu berechnen und den CRC in das FCS-Feld des Pakets einzufügen. Eine QE110 Vorrichtung ersetzt das AI_FCS_IN* Signal mit dem FBPN* Signal, wie früher beschrieben, für zusätzliche Funktionen. Ein Signal EOP* bezeichnet das Ende des Pakets, wenn die letzte Datenübertragung eines Datenpakets auf dem HSB **206** übertragen wird. BYTE_VALID[3:0]* Signale geben an, welche Bytes in dem gegenwärtigen Wort auf den DATA-Signalen gültig sind. Es wird angemerkt, dass ein Datenpaket für ein einzelne Übertragung auf dem HSB **206** gewöhnlich zulang ist, sodass jeder Buszyklus eine Datenmenge kleiner als oder gleich dem TBUS-Wert überträgt.

[0080] Man wird erkennen, dass jede QC-Vorrichtung **202** jeden ihrer vier Port als 10Base-T Ethernet-Ports betreibt. Man wird weiter erkennen, dass der EPSM **210** Zugang hat, um alle Register der QC-Vorrichtungen **202** durch den QC-Bus **204** zu lesen und zu beschreiben. Ferner liest der EPSM **210** Daten aus allen Empfangs-FIFOs **320** und schreibt Daten in alle Sende-FIFOs **304** durch den HSB **206**.

[0081] Fig. 3G ist ein exemplarisches Timing-Diagramm, das einen gleichzeitigen Lese- und Schreibzyklus auf dem HSB **206** veranschaulicht. Der obere Teil des Timing-Diagramms bezeichnet den Zyklustyp, wo zwei gleichlaufende Lese- und Schreibzyklen einer nach dem anderen ausgeführt werden. Die Signale CLK, CLK_2, STROBE*, READ_OUT_PKTm*, WRITE_IN_PKTm*, PORT_NO[1:0], DATA[31:0] und ABORT_OUT* sind auf einer Y Achse (oder Vertikalachse) über Zeit geplottet, die auf einer X-Achse (oder Horizontalachse) des Timing-Diagramms geplottet ist. Es gibt zwei verschiedene Typen von gleichzeitigen Lese- und Schreibzyklen, die abhängig von der einzelnen Ausführung durchgeführt werden. Für den ersten, allgemeinen Typ von gleichzeitigen Zyklen werden, wenn die QC-Vorrichtungen **202** mit den QE110 Vorrichtungen, die die Latches **324**, **326** enthalten, implementiert sind, gleichzeitige Lese- und Schreibzyklen ohne weitere Verbesserung durchgeführt. Alternativ, wenn die QC-Vorrichtungen **202** mit den L64381 Vorrichtungen implementiert sind, werden externe Latches und Auswähllogik (nicht gezeigt) hinzugefügt, um die PORT_NO Signale zu speichern, wenn auf dem HSB **206** geltend gemacht. Ein zweiter, spezieller Typ von gleichzeitigen Lese- und Schreibzyklen wird mit den L64381 Vorrichtungen ohne wertere Verbesserung durchgeführt, aber nur, wenn die PORT_NO Signale gleich sind und nur, wenn die QC-Vorrichtungen **202** verschieden sind.

[0082] Der EPSM **210** bestimmt den Typ des auszuführenden Zyklusses, z. B. Lesen, Schreiben, gleichzeitiges Lesen und Schreiben, Rückstau usw. Ein Lesezyklus wird gewöhnlich durch Geltendmachung eines der READ_OUT_PKTm* Signale angegeben, und ein Schreibzyklus wird gewöhnlich durch Geltendmachung eines der WRITE_IN_PKTm* Signale angegeben. Ein gleichzeitiger Lese- und Schreibzyklus wird durch gleichzeitige Geltendmachung eines READ_OUT_PKTm* Signals und eines WRITE_IN_PKTm* Signals angegeben. Der EPSM **210** führt einen gleichzeitigen Lese- und Schreibzyklus zwischen Ports unter bestimmten Bedingungen durch, z. B. nur wenn beide Ports konfiguriert sind, um im Durchschalt-(CT)Modus, unten ausführlicher beschrieben, zu arbeiten.

[0083] Während des gleichzeitigen Zyklusses macht der EPSM **210** eines der READ_OUT_PKTm* Signale am Anfang des dritten CLK-Zyklusses tief geltend, um eine der QC-Vorrichtungen oder die TPI **220** anzugeben, und macht die geeignete Portnummer auf den PORT_NO[1:0] Signalen während des dritten CLK-Zyklusses geltend, um einen der vier Ports der durch das geltend gemachte READ_OUT_PKTm* Signal identifizierten QC-Vorrichtung **202** anzugeben. Die durch das bestimmte READ_OUT_PKTm* Signal identifizierte QC-Vorrichtung **202** speichert die PORT_NO[1:0] Signale im dritten CLK-Zyklus, um den einzelnen Port, der gelesen wird, zu bestimmen. Zum Beispiel sind die QE110 Vorrichtungen, die die QC-Vorrichtungen **202** implementieren, mit den Lese-Latches **324** konfiguriert, um die Signale PORT_NO[1:0] zu speichern. Außerdem enthält die TPI **220** ähnliche Lese-Latches **819b** (Fig. 8B), um die PORT_NO[1:0] Signale im dritten CLK-Zyklus zu speichern, wenn durch das READ_OUT_PKT[6]* Signal angegeben. Alternativ werden externe Latches für diesen Zweck verwendet, wenn die QC-Vorrichtungen **202** mit den L64381 Vorrichtungen implememtiert sind. An diesem Punkt ist der identifizierte einzelne Port PORT0–PORT27 als der Quellenport für einen Lesezyklus auf dem HSB **206** bezeichnet worden.

[0084] Der EPSM **210** macht eines der WRITE_IN_PKTm* Signale am Anfang des vierten CLK-Zyklusses tief geltend, um die gleiche oder irgendeine andere der QC-Vorrichtungen **202** oder die TPI **220** zu bezeichnen, und macht die geeignete Portnummer auf den PORT_NO[1:0] Signalen während des vierten CLK-Zyklusses geltend, um einen der vier Ports der Vorrichtung zu bezeichnen, die durch das geltend gemachte WRITE_IN_PKTm* Signal bezeichnet wird. Die durch das einzelne WRITE_IN_PKTm* Signal identifizierte QC-Vorrichtung speichert die PORT_NO[1:0] Signale im vierten CLK-Zyklus, um den einzelnen Port, in den geschrieben wird, zu bestimmen. Zum Beispiel sind die QE110 Vorrichtungen, die die QC-Vorrichtungen **202** implementieren, mit den Schreib-Latches **326** konfiguriert, um die Signale PORT_NO[1:0] im viertenCLK-Zyklus zu speichern. Außerdem enthält die TPI **220** ähnliche Schreib-Latches **819b**, (Fig. 8B), um die PORT_NO[1:0] Signale im vierten CLK-Zyklus zu speichern, wenn durch das WRITE_IN_PKT[6]* Signal angegeben. In dieser Weise wird irgendein anderer der Ports Port0–Port27 als der Zielport für einen Schreibzyklus auf dem HSB **206** bezeichnet, wobei der Schreibzyklus zur der gleichen Zeit wie der gerade angegebene Lesezyklus stattfindet. Die Quellen- und Zielports können auf der gleichen QC-Vorrichtung oder zwei Ports der TPI **220** sein, oder können zwischen verschiedenen QC-Vorrichtungen **202** liegen. Ein gleichzeitiger Lese- und Schreibzyklus wird jedoch zwischen einem der Ports **104** der QC-Vorrichtungen **202** und einem der Ports **110** der TPI **220** in der gezeigten Ausführung wegen der Unterschiede in der Datenübertragungsgeschwindigkeit nicht durchgeführt.

[0085] In den folgenden Zyklen des CLK-Signals werden Paketdaten gleichzeitig übertragen oder aus dem Quellenport gelesen und über den HSB **206** direkt in den Zielport geschrieben, ohne in dem EPSM **210** oder dem Speicher **212** gespeichert zu werden. Die Datenübertragung erfolgt in Zyklen 5, 6, 7 und 8 zum Übertragen mehrerer Bytes abhängig von der Ausführung. Zum Beispiel werden bis zu 64 Bytes für L64381 Vorrichtungen übertragen, und bis zu 256 Bytes werden für QE110 Vorrichtungen übertragen. Obwohl vier CLK-Zyklen für die Datenübertragung gezeigt werden, kann die Datenübertragung mit einem, zwei oder vier CLK-Zyklen abhängig davon stattfinden, wie viele Daten übertragen werden. Für neue Pakete wird zuerst ein normaler Lesezyklus durchgeführt, um die Quellen- und Ziel-MAC-Adressen in den EPSM **210** zu bringen, der dann eine werter unten beschriebene Hashig-Prozedur durchführt, um die Zielportnummer, wenn bekannt, zu bestimmen. Sobald die Zielportnummer bekannt ist, und wenn es nur einen Zielport gibt, kann eine gleichzeitige Lese- und Schreib-Operation für jeden Abschnitt oder den ganzen Rest des Pakets, wie gewünscht, durchgeführt werden.

[0086] Der spezielle Typ des gleichzeitigen Lese- und Schreibzyklusses wird durchgeführt, wenn die PORT_NO Signale gleich sind, aber zwischen zwei verschiedenen Ports und daher zwischen zwei verschiedenen QC-Vorrichtungen **202**. Fig. 3G zeigt auch diesen Fall, außer dass die PORT_NO Signale während des ganzen Zyklusses unverändert bleiben. Die Latches **324**, **326** sind nicht erforderlich, da die PORT_NO Signale unverändert bleiben, sodass dieser Typ von Bleizeitigem Zyklus zwischen zwei verschiedenen L64391 Vorrichtungen ohne externe Latches oder Auswähllogik durchgeführt werden kann. Der EPSM **210** bestimmt, dass

die PORT_NO Signale zwischen den Quellen- und Zielports gleich sind, und dass zwei verschiedene QC-Vorrichtungen **202** involviert sind, und lässt dann den gleichzeitigen Zyklus, wie gezeigt, laufen.

[0087] Wein Fig. 3G gezeigt, findet eine zweite, gleichzeitige Lese- und Schreibübertragung im sechsten CLK-Zyklus statt, wo die PORT_NO[1:0] Signale dann im siebten, achten und neunten Zyklus mit dem Lesemodus, der Leseportnummer und der Schreibportnummer geltend gemacht werden. Als Reaktion wird ein READ_OUT_PKTm* Signal für den siebten Zyklus deaktiviert. Desgleichen wird ein WRITE_IN_PKTm* Signal für den achten Zyklus deaktiviert. Dieser zweite, gleichzeitige Zyklus ist entweder eine Fortsetzung des ersten gleichzeitigen Zyklusses zum Bereitstellen von fortlaufenden und aufeinanderfolgenden Daten des gleichen Pakets, oder kann der Beginn eines gänzlich verschiedenen Pakets sein. Die Quellen- und Zielports sind für fortgesetzte Daten für das gleiche Paket die gleichen. Entweder der Quellenport, der Zielport oder beide können jedoch im zweiten, gleichzeitigen Zyklus zum Übertragen von Daten für ein unterschiedliches Paket verschieden sein.

[0088] Fig. 3H ist ein Flussdiagramm, das eine Prozedur zum Ausführen eines gleichzeitigen Lese- und Schreibzyklusses auf dem HSB **206** veranschaulicht. In einem ersten Schritt **330** stellt der EPSM **210** fest, ob ein gleichzeitiger Lese- und Schreibzyklus auf dem HSB **206** zwischen einem Quellenport und einem Zielport ausgeführt werden kann. Im nächsten Schritt **332** macht dann der EPSN **210** die geeigneten Signale geltend, um den Quellenport zu identifizieren. Dies wird durch Geltendmachung der Quellen- oder "Lese"-Portnummer mittels der PORT_NO Signale auf dem HSB **206** und durch Geltendmachung des geeigneten READ_OUT_PKTm* Signals durchgeführt. Im nächsten Schritt **334** erfasst oder speichert die identifizierte Quellenport-Vorrichtung die Identifikationssignale. In dem speziellen gleichzeitigen Zyklus ohne Latches erfasst die QC-Vorrichtung **202** das READ_OUT_PKTm* Signal und dann die PORT_NO Signale auf dem HSB **206** und beginnt, sich auf einen Lesezyklus vorzubereiten. In den allgemeinen gleichzeitigen Zyklen, die Latches verwenden, speichert die angegebene QC-Vorrichtung **202** oder die TPI **220** in Schritt **334** die Leseportnummer und beginnt, sich auf einen Lesezyklus vorzubereiten.

[0089] Im nächsten Schritt **336** macht der EPSM **210** die geeigneten Signale geltend, um den Zielport zu identifizieren. Für den speziellen gleichzeitigen Zyklus macht der EPSM **210** das geeignete WRITE_IN_PKTm* Signal geltend und bewahrt die gleichen PORT_NO Signale. Für den allgemeinen Fall macht der EPSM **210** auch die Ziel- oder "Schreib"-Portnummer auf dem HSB **206** zusammen mit dem geeigneten WRITE_IN_PKTm* Signal im nächsten Schritt **336** geltend. Im nächsten Schritt **338** erfasst oder speichert die identifizierte Zielport-Vorrichtung die Identifikationssignale. In dem speziellen Zyklus ohne Latches erfasst die angegebene QC-Vorrichtung **202** das WRITE_IN_PKTm* Signal und dann die PORT_NO Signale auf dem HSB **206** und beginnt, sich auf einen Schreibzyklus vorzubereiten. Für den allgemeinen Fall speichert die angegebene QC-Vorrichtung **202** oder die TPI **220** die Ziel- oder Schreibportnummer im nächsten Schritt **338**. Schließlich stellt im nächsten Schritt **340** der angegebene Quellenport die Daten auf dem HSB **206** bereit, während der angegebene Zielport die Daten aus dem HSB **206** in einem gleichzeitigen Lese- und Schreibzyklus liest.

[0090] Die gleichzeitige Lese- und Schreib-Operation ist der schnellste Typ von Datenübertragungszyklus, da nur ein einziger Buszyklus für jede Übertragung von Paketdaten benötigt wird. Wie weiter unten beschrieben, benötigt eine normale CT-Betriebsart wenigstens zwei Übertragungen, eine von dem Quellenport zu dem EPSM **210** und eine andere von dem EPSM **210** zu dem Zielport, was zwei getrennte Zyklen auf dem HSB **206** für die gleichen Daten benötigt. Ein gleichzeitiger Lese- und Schreibzyklus benötigt eine einzige direkte Übertragung auf dem HSB **206** für die gleichen Daten, wodurch die Bandbreite des HSB **206** erhöht wird. Andere, langsamere Modi werden bereitgestellt, einschließlich mehrerer Interim-CT- und Speichern-und-Weiterleiten-(SnF)Modi, wo Paketdaten in den Speicher **212** geschrieben werden, bevor sie zu dem Zielport übertragen werden.

[0091] Fig. 4 zeigt ein vereinfachtes Blockschaltbild des EPSM **211**, das den Datenfluss und Konfigurationsregister veranschaulicht. Der EPSM **210** enthält drei Hauptabschnitte, einschließlich eines HSB-Steuerungsblocks (HCB) **402**, eines Speicher-Steuerungsblocks (MCB) **404** und eines Prozessor-Steuerungsblocks (PCB) **406**. Eine QC-Schnittstelle **410** verbindet den HSB **206** und den HCB **402** des EPSM **210**. Ein Satz von Puffern oder FIFOs **412** ist mit der anderen Seite der QC-Schnittstelle **410** verbunden, wo die FIFOs **412** weiter unten beschriebene Empfangs-, Sende- und Durchschalt-FIFOs umfassen. Die andere Seite der FIFOs (ausschließlich eines CT-Puffers, Fig. 5A) ist mit dem MCB **404** durch eine MCB-Schnittstelle **414** verbunden, die mit einer HCB-Schnittstelle **418** in dem MCB **404** durch einen geeigneten Bus **420** verbunden ist. Die HCB-Schnittstelle **418** ist weiter mit einer Speicherschnittstelle **422** verbunden, die mit dem Speicher **212** durch den Speicherbus **214** verbunden ist. Die Speicherschnittstelle **422** ist weiter mit einer Seite einer PCB-Schnitt-

stelle **424** verbunden, deren andere Seite mit einer Seite einer MCB-Schnittstelle **426** in dem PCB **406** durch einen geeigneten MCB-Bus **428** verbunden ist. Die andere Seite der MCB-Schnittstelle **426** ist mit einer Seite eines Satzes von FIFOs **430** verbunden, die weiter mit einer CPU-Schnittstelle **432** in dem PCB **406** verbunden sind. Die CPU-Schnittstelle **432** ist mit dem QC/CPU-Bus **204** und mit dem CPU-Bus **218** verbunden. Die CPU-Schnittstelle **432** ist weiter mit einer Seite eines zweiten Satzes FIFOs **434** in dem PCB **406** verbunden, deren andere Seite mit einer QC/HCB-Schnittstelle **436** verbunden ist. Die andere Seite der QC/HCB-Schnittstelle **436** ist mit der QC-Schnittstelle **410** über einen geeigneten HCB-Bus **438** verbunden.

[0092] Es wird angemerkt, dass die Signale PCB_BUF_AVAIL*, PCG_PKT_AVAIL*, PCB_RD_OUT_PKT* und PCB_WR_IN_PKT* des HCB-Busses **438**, die mit dem PCB **406** und der CPU **230** verbunden sind, in den Signalen BUF_AVAILm*, PKT_AVAILm*, READ_OUT_PKTm* und WRITE_IN_PKTm* enthalten sind. In der gezeigten Ausführung ist der HCB-Bus **438** ähnlich dem HSB **206** und ist im Wesentlichen eine interne Version des HSB **206** im EPSM **210**. Der PCB **406** verhält sich in ähnlicher Weise wie jeder der Ports **104** und die TPI **220** zu dem HCB **402**. Auf diese Weise arbeitet die CPU **230**, durch die Funktion des PCB **406**, als ein zusätzlicher Port (Port28) zu dem HCB **402**.

[0093] Die CPU-Schnittstelle **432** ist mit einer Registerschnittstelle **440** durch einen Bus **442** verbunden, wo die Registerschittstelle **440** werter mit einem Registerbus **444** verbunden ist. Der Registerbus **444** ist mit einem Satz von HCB-Konfigurationsregistern **446** in dem HCB **401** und einem Satz von MCB-Konfigurationsregistern **448** in dem MCB **404** verbunden. In dieser Weise initialisiert und programmiert die CPU **230** die Register in den HCB- und MCB-Konfigurationsregistern **446** und **448** durch die CPU-Schnittstelle **432** und die Registerschnittstelle **440**.

[0094] Die MCB-Konfigurationsregister **448** werden benutzt, eine wesentliche Menge mit den Ports und dem Speicher **212** verbundener Konfigurationsinformation zu speichern. Zum Beispiel enthalten die MCB-Konfigurationsregister **448** Port-Statusinformation, die angibt, ob jeder in einem lernenden (LRN), weiterleitenden (FWD), blockierten (BLK), zuhörenden (LST) oder abgeschalteten (DIS) Zustand ist, Speichersektorinformation, Busbenutzungsinformation des Speicherbusses **214**, Zahl fallen gelassener Pakete, Hash-Tabelleninformation, Speicherschwellen, BC-Schwellen, Identifikation von sicheren Ports, wenn vorhanden, Speichersteuerinformation, MCB-Unterbrechungsquellenbits, Unterbrechungsmaskierungsbits und Abfragequellenbits usw.

[0095] Die Beschreibung des EPSM **210** veranschaulicht, dass die CPU **230** Zugriff auf die QC-Vorrichtungen und den Speicher **212** für Konfigurations- und Steuerzwecke hat. Obwohl der Hauptdatenfluss mit dem HSB **206** mit dem EPSM **210** durch die FIFOs **412** und den Speicher **212** ist, findet ein Datenfluss auch zwischen dem HSB **206** und der CPU **230** durch den HCB-Bus **438** und zugehörige FIFOs und Schnittstellen des EPSM **210** statt.

[0096] Fig. 5A zeigt ein ausführlicheres Blockschaltbild des HCB **402**. Der HCB-Bus **438** ist eine interne Version des HSB **206** zum Verbinden des PCB **406**, wo beide Busse **206**, **438** kollektiv als der HSB **206** bezeichnet werden. Eine Abfragelogik **501** ist mit dem HSB **206**, mit einem Satz von lokalen Registern **506** und mit HCB-Konfigurationsregistern **446** verbunden. Die Abfragelogik **501** empfängt das CLK-Signal und macht periodisch das STROBE* Signal an den QC-Vorrichtungen **202** und der TPI **220** zum Abfragen der Ports **104**, **110** und des PCB **406** geltend. Die Abfragelogik **501** überwacht dann die gemultiplexten PKT_AVAIL_m* und BUF_AVAILm* Signale von den QC-Vorrichtungen und der TPI **220**, wobei die jede QC-Vorrichtung **202** und die TPI **220** den Status ihrer vier Ports **104** bzw. **110** liefert, wie vorher beschrieben. Die TPI **220** antwortet mit den Signalen PKT_AVAIL[6]* und BUF_AVAIL[6]*, und der PCB **406** antwortet mit den Signalen PKT_AVAIL[7]* und BUF_AVAIL[7]*.

[0097] Die Abfragelogik **501** enthält eine Empfangs-(RX)Abfragezustandsmaschine **502**, die die PKT_AVAILm* Signale durchsieht und eine Empfangsliste **509** in den Registern **506** aktualisiert. In einer ähnlichen Weise enthält die Abfragelogik **501** eine Sende-(TX)Abfragezustandsmaschine **503**, die die BUF_AVAILm* Signale durchsieht und eine Sendeliste **510** in den Registern **506** aktualisiert. Wenn ein WT-PRIORITY-Flag in den HCB-Konfigurationsregistern **446** durch die CPU **230** gesetzt wird, verwenden die RX-Abfagezustandsmaschine **502** und die TX-Abfragezustandsmaschine **503** einen Satz von Gewichtsfaktoren **508** in den HCB-Konfigurationsregistern **446** zum Programmieren der Empfangsliste **509** bzw. der Sendeliste **510**, wie weiter unten beschrieben. Die HCB-Konfigurationsregister **446** enthalten auch einen Satz von CT_SnF-Registern **507**, die von der CPU **230** programmiert werden, um die gewünschte Betriebsart zwischen CT und SnF zu bestimmen, wenn der entsprechende Port entweder ein Quellen- oder ein Zielport ist.

[0098] Die Register **506** werden in jeder gewünschten Weise abhängig von der Implementierung des EPSM

210 implementiert, z. B. als Latches, Flipflops, statische RAMs (SRAM), DRAMs usw., und umfassen eine Vielzahl von Status- und Steuerregistern oder Puffern. Die Empfangsliste **509** enthält eine Vielzahl von Registerwerten, die den relativen Empfangsstatus und die Priorität jedes Ports anzeigen. Desgleichen enthält die Sendeliste **510** ein Vielzahl von Registerwerten, die den relativen Sendestatus und die Priorität jedes Ports anzeigen. Ein PRCOUNT-Register **511a** speichert eine PRCOUNT-Nummer, die von der RX-Abfragezustandsmaschine **502** verwendet wird, um jedem Port eine relative Empfangspriorität zuweisen, wenn Paketdaten durch diesen Port von einer externen Netzwerk-Vorrichtung empfangen werden. Alternativ verwendet die RX-Abfragezustandsmaschine **502** einen entsprechenden Gewichtsfaktor von den Gewichtsfaktoren **508**. Desgleichen speichert ein TP-COUNT-Register **511b** eine TPCOUNT-Nummern die von der TX-Abfragezustandsmaschine **503** verwendet wird, um jedem Port eine relative Sendepriorität zuzuweisen, wenn Paketdaten zum Senden durch diesen Port an eine externe Netzwerk-Vorrichtung vorhanden sind und der Fort Raum hat, um Daten zum Senden zu empfangen. Alternativ verwendet die TX-Abfragezustandsmaschine **502** einen entsprechenden Gewichtsfaktor von den Gewichtsfaktoren **508**. Relative Arbitrations-Zählwerte RXNEWCNT, RXACTCNT, TXNEWCNT und TXCTCNT werden in Registern RXNEWCNT **511c**, RXACTCNT **511d**, TXNEWCNT **511e** und TXCTCNT **511f** gespeichert.

[0099] Der HCB **402** enthält Arbitrationslogik **504**, die die Daten in den Registern **506** und **446** durchsieht, um die Typen der auf dem HSB **206** ausgeführten Zyklen zu bestimmen. Eine HSB-Steuerung **505** steuert jeden auf dem HSB **206** ausgeführten Zyklus zum Steuern des Datenflusses zwischen dem EPSM **210** und dem HSB **206**. Die HSB-Steuerung **505** ist mit den Registern **506** zum Modifizieren von Statusbits verbunden. Die HSB-Steuerung **505** empfängt eine Angabe des Typs jedes Zyklusses von der Arbitrationslogik **504**. Die Arbitrationslogik **504** enthält eine Haupt-Arbiter **512**, der mit vier Arbitern verbunden ist, die einen Neupaketempfangs-(RX NW)Arbiter **513**, einen Empfangaktiv-(RX ACT)Arbiter **514**, einen Neupaketsende-(TX NW)Arbiter **515** und einen Sende-Durchschall-(TX CT)Arbiter **516** umfassen. Der Hauptarbiter **512** wählt gewöhnlich zwischen dem RX NW-Arbiter **513**, dem RX ACT-Arbiter **514**, dem TX NW-Arbiter **515** und dem TX CT-Arbiter **516** aus, wo jeder Arbiter schlichtet, um den nächsten Zyklus zu definieren. Der Hauptarbiter **512** verwendet nach Wunsch jedes annehmbare Prioritätsschema. In der gezeigten Ausführung verwendet der Hauptarbiter **512** z. B. ein Umlauf-Prioritätsschema.

[0100] Die FIFOs **412** werden in jeder gewünschten Weise implementiert. In der gezeigten Ausführung implementieren zwei Empfangspuffer RX BUFs **529**, **522** einen RXFIFO, wo Daten aus einem Puffer gelesen werden, während sie in den anderen geschrieben werden, und umgekehrt. Auch werden zwei Sendepuffer TX-BUFs **524**, **526** bereitgestellt und arbeiten in einer ähnlichen Weise wie die RXBUFs **510**, **522**. Die FIFOs **412** enthalten auch wenigstens einen Durchschalt-Puffer CTBUF **528**. Die RXBUFs **520** und **522** sind beide 64-Byte Puffer, die je eine bidirektionale Datenschnittstelle mit dem HSB **206** zum Datenfluss in jeder Richtung und eine unidirektionale Schnittstelle zum Liefern von Daten an den MCB **404** durch eine RXMCB-Schnittstelle **530** enthalten. Die TXBUFs **524**, **526** sind beide 64-Byte Puffer, die zwischen den HSB **206** und eine TXMCB-Schnittstelle **531** geschaltet sind. Die TXBUFs **524**, **526** empfangen Daten von dem MCG **404** durch die TXMCB-Schnittstelle **531** und liefern Daten an den HSB **206**. Der CTBUF **528** ist ein 64-Byte Puffer mit einer bidirektionalen Schnittstelle mit dem HSB **206**. Ein FIFO-Steuerblock **529** ist mit den Registern **506**, der HSB-Steuerung **505**, den RXBUFs **520m 522**, den TXBUFs **524**, **526**, dem CTBUF **528**, der RXMCB-Schnittstelle **530** und der TXMCB-Schnittstelle **513** zum Steuern des Datenflusses durch die FIFOs **520**, **522**, **524**, **526** verbunden, um bestimmte durch die RX, TXMCB-Schnittstellen **530** und **531** geltend gemachte Statussignale zu erfassen und bestimmte Bits in den Registern **506** zu setzen, wie unten weiter beschrieben.

[0101] Der Bus **420** enthält eine Vielzahl von Daten- und Steuersignalen zum Verbinden des HSB **402** mit dem MCB **404** durch die RX, TXMCB-Schnittstellen **530**, **531**, Hash-Anforderungslogik und MCB-Schnittstelle (bezeichnet als HASH REQ LOGIC) **532** und Sende-Arbiter-Anforderungslogik und MCB-Schnittstelle (bezeichnet als TX ARB REQ LOGIC) **533**. Die HSB-Steuerung **505** kopiert den Vorspann jedes neuen Pakets von einem der Ports Port0–Port28 in einen der RXBUFs **520**, **522** und auch in die HASH REG LOGIC **532**. Der Vorspann beträgt wenigstens drei DWORDs (je 32 Bit), die sowohl die Quellen- als auch Ziel-MAC-Adressen enthalten. Die HASH REQ LOGIC **532** verlangt, dass die Hashing-Prozedur durch den MCB **404** ausgeführt wird, und setzt geeignete Bits in den Registern **506**. Die Hashing-Prozedur wird durchgeführt, um die geeignete Aktion zu bestimmen, die für das Paket zu ergreifen ist.

[0102] In der gezeigten Ausführung macht nach Empfangen des Vorspanns eines neuen Pakets die HASH REG LOGIC **532** ein Signal HASH_REQ* am MCB **404** geltend und multiplext die 48-Bit MAC-Ziel- und Quellenadressen und eine 8-Bit Quellenportnummer auf HASH_DA_SA[15:9] Signale. Der MCB **404** erfasst das HASH_REQ* Signal, führt die Hashing-Prozedur durch und macht ein Signal HASH_DONE an der HASH REQ LOGIC **532** geltend. Der MCB **404** macht auch Signale HASH_DSTPRT[4:0], HASH_STATUS[1:9] und ein Si-

gnal HASH_BP*, wenn angebracht, geltend. Die HASH_STATUS[1:0] Signale bezeichnen eines von vier Ergebnissen, die 00b (b bezeichnet eine Binärzahl) = DROP_PKT, um das Paket fallen zu lassen, 01b = GROUP_BC für ein Rundsende-(BC)Paket, 10b = MISS_BC für einen unbekannten Zielport und daher ein BC-Paket, und 11b = FORWARD_PKT, das ein Unicast-Paket an einen einzelnen Zielport bezeichnet, umfassen. Wenn HASH_STATUS[1:0] = FORWARD_PKTn werden die HASH_DSTPRT[4:0] Signale geltend gemacht, wobei eine binäre Portnummer den Zielport für das Paket bezeichnet. Das HASH_BP* Signal wird geltend gemacht, um Rückstau anzuzeigen, wenn Rückstau freigegeben und anwendbar ist, infolge einer Schwellenüberlaufbedingung im Speicher **212**, wie durch den MCB **404** festgestellt.

[0103] Bestimmte Schwellenwerte werden für den ganzen Speicher **212**, für einzelne Typen von Paketen (z. B. BC-Pakete) und auf einer Port-für-Port-Basis festgelegt. Wenn ein Schwellenwert erreicht wird, sodass ein anderes an den Speicher **212** geliefertes Paket eine Schwellenbegingung verletzen würde, entscheidet der Netzwerkschalter **102**, ob das Paket fallen zu lassen ist. Die sendende Vorrichtung erkennt schließlich, dass das Paket fallen gelassen wird, und sendet das Paket neu. Wenn bestimmte Schwellenbedingungen verletzt werden, wird Rückstau freigegeben, und wenn der Quellenport im Halbduplexmodus arbeitet, wird das HASH_BP* Signal geltend gemacht.

[0104] Die HASH_REQ_LOGIC **532** erfasst das HASH_BP* SIGNAL und stellt fest, ob HASH_STATUS[1:0] = DROP_PKT, z. B. die Quellen- und Zielports sind die gleichen. Wenn HASH_STATUS[1:0] = DROP_PKT, ist keine weitere Aktion nötig, da das Paket fallen zu lassen ist. Wenn HASH_STATUS[1:0] nicht gleich DROP_PKT ist, stellt die HASH REQ LOGIC **532** fest, ob HASH_STATUS[1:0] = FORWARD_PKT und das Paket im CT-Modus durch den CT BUF **528** zu übertragen ist, um dadurch möglicherweise den Speicher **212** zu umgehen. Wenn der Zielport beschäftigt ist, oder wenn HASH_STATUS[1:0] nicht angibt, das Paket fallen zu lassen oder das Paket weiterzuleiten, weist die HASH REQ LOGIC **532** die HSB-Steuerung an, einen Rückstau-Zyhlus auf dem Port, der Daten empfängt, auszuführen.

[0105] Während des SnF-Betriebs empfängt der EPSM **210** das ganze Paket und speichert es im Speicher **212**, bevor jeder Teil des Pakets an einen Zielport gesendet wird. Nachdem das Paket gesendet ist, und wenn der Zielport bekannt ist, wird das Paket an den Zielport, wenn vorhanden, entsprechend dem einzelnen benutzten Arbitrationsschema gesendet. Um CT-Betrieb anzuwenden, werden beide Ports für den CT-Modus in den CT_SNF-Registern **507** voreingestellt, beide Ports arbeiten bei der gleichen Geschwindigkeit, und die TBUS-Einstellung für den Zielport ist größer als die oder gleich der TBUS-Einstellung für den Quellenport. Für die einzelne gezeigte Ausführung, die die TLANs **226** verwendet, um die 100 Mbps Ethernet-Ports Port24–Port27 zu implementieren, wird der CT-Modus für die Ports Port24–Port27 nicht durchgeführt, da die TLANs vor dem Senden die Größe des ganzen Pakets benötigen. Außerdem verlangt die gezeigte Ausführung, dass die TBUS-Werte gleich sind. Die vorliegende Erfindung wird durch diese verschiedenen Erwurfserwägungen nicht begrenzt. Während der CT-Betriebsart liefert der EPSM **210** die Daten an die geeignete QC-Vorrichtung **202** zum Senden auf dem angegebenen Zielport, wenn er nicht beschäftigt ist. Die Paketdaten werden durch die FIFOs **412** zwischen den Quellen- und Zielports gepuffert, ohne in den Speicher **212** übertragen zu werden.

[0106] Wenn der Zielport am Anfang eines empfangenen Pakets beschäftigt ist, werden die Daten im Speicher **212** zwischen den Quellen- und Zielports entsprechend der Interim-CT-Betriebsart gepuffert. Der Paketabschnitt steht jedoch sofort zum Senden an einen Zielport zur Verfügung, sodass der Zielport nicht auf das ganze zu empfangende Paket warten muss. Als ein Sicherheitsmechanismus kann die Interim-CT-Betriebsart aufgehoben und der Betrieb für dieses einzelne Paket in den SnF-Modus für das nächste Paket umgeschaltet werden.

[0107] Wenn aus irgendeinem Grund der Zielport außerstande ist, mehr Daten während der Übertragung eines Pakets im CT-Modus anzunehmen, z. B. wenn der Zielport stehen bleibt, wird der Betrieb auf den Mittelpaket-Interim-CT-Modus umgeschaltet. Während des Mittelpaket-Interim-CT-Modus werden die Paketdaten in den FIFOs **412** an den Speicher **212** gesendet und dann an die Zielport gesendet, wenn er verfügbar ist, um mehr Daten zu empfangen. Es wird angemerkt, dass, da andere nachfolgend empfangene Pakete durch andere Ports zum Senden an den gleichen stehen gebliebenen Port empfangen werden können, wobei diese nachfolgenden Pakete in eine entsprechende Sendekette für den Port gestellt werden, der restliche Paketabschnitt des auf den Mittelpaket-Interim-CT-Modus umgeschalteten Pakets zuerst in die Sendekette gestellt wird, um die richtige Reihenfolge zu sichern.

[0108] Ein anderer Modus wird als der adaptive SnF-Modus bezeichnet. Während ein Paket entsprechend dem CT-Betrieb übertragen wird, überwacht und verfolgt die CPU **230** die Aktivität der Ports **104**, **110** und des

PCB **406**, µm festzustellen, ob einer oder mehrere der Ports eine bedeutsame Zahl von Fehlern erfährt, z. B. "Runts", "Overruns", "Jabbers", Spätkollision, FCS-Fehler usw. Ein Runt ist ein Paket kleiner als eine bestimmte Mindestdatenmenge, wobei das Minimum in der gezeigten Ausführung 64 Byte beträgt. Ein Overrun ist ein Paket, das größer ist als eine bestimmte Maximaldatenmenge, die in der gezeigten Ausführung 1,518 Byte entsprechend dem Ethernet-Standard beträgt. Ein Jabber ist ein Paket größer als die Maximalgröße (1,518 Bytes für Ethernet) und enthält einen ungültigen CRC-Wert. Gewöhnlich werden Pakete mit solchen Fehlern fallen gelassen und durch das System nicht verbreitet. Entsprechend dem adaptiven SnF-Modus schaltet, wenn ein Port **104** im CT-Betrieb arbeitet und eine bedeutsame Zahl solcher Fehler erfahren werden, wie durch die CPU **230** bestimmt, die CPU **230** den voreingestellten Modus für den gewünschten Port von CT- auf SnF-Betrieb um, bis alle Fehler korrigiert oder sonstwie beseitigt sind.

[0109] Die Arbeitsweise der Ports **110** jedes TLAN **226** ist ähnlich, außer dass Paketdaten durch die TPI **220** über den HSB **206** zu dem EPSM **210** laufen und vor dem Senden im Speicher **212** gespeichert werden. Die TPI **220** arbeitet effektiv als eine Brücke zwischen dem PCI-Bus **222** und dem HSB **206**. Die TLANs **226** benötigen die Länge des ganzen Pakets, bevor das Pa-ket an ein externes Netzwerk gesendet wird, sodass jedes Paket in seiner Gesamtheit empfangen und im Speicher **212** gespeichert wird, bevor es an eines der TLANs **226** neu gesendet wird. Außerdem, Daten, die durch ein TLAN **226** zum Senden durch eine QC-Vorrichtung **202** empfangen werden, und Daten die durch eine QC-Vorrichtung **202** zum Senden durch ein TLAN **226** empfangen werden, werden infolge des großen Geschwindigkeitsunterschieds zwischen den Vorrichtungen **202**, **226** in der gezeigten Ausführung im SnF-Modus betrieben und im Speicher **212** gespeichert.

[0110] Die RXMCB-Schnittstelle **530** macht ein Signal RX_PKT_AVAIL* an dem MCB **404** geltend, wenn Paketdaten in einem der RXBUFs **520**, **522** liegen und zum Übertragen in den Speicher **212** bereit sind. Paketdaten werden von dem HCB **402** an den MCB **404** auf einem Speicherdaten-Ausgangsbus MemDataOut oder MDO[31:0] übertragen. Ein statisches Signal MEM_EDO wird geltend gemacht, wenn der Typ des Speichers **212** entweder EDO oder Synchron-DRAM ist, und wird für FPM-DRAM nicht geltend gemacht. Die RXMCB Schnittstelle **530** macht auch, wenn angebracht, mehrere andere Signale geltend, während das Signal RX_PKT_AVAIL* geltend gemacht wird. Das heißt, die RXMCB-Schnittstelle **530** multiplext die Quellenportnummer auf RX_SRC_DST[4:0] Signale für einen CLK-Zyklus gefolgt von der Zielportnummer, wenn bekannt, während des nächsten CLK-Zyklusses, während das RX_PKT_AVAIL* Signal geltend gemacht wird. Ferner macht die RXMCB-Schnittstelle **530** die Zahl von DWORDs (minus einem DWORD) auf RX_CNT[5:0] geltend, die sich in dem ausgewählten RXBUF **520** oder **522** befinden.

[0111] Die RXMCB-Schnittstelle **530** macht ein Signal RX_SOP* mit dem RX_PKT_AVAIL* Signal geltend, wenn die Daten der Beginn eines Pakets sind, oder macht ein Signal RX_EOP* mit dem RX_PKT_AVAIL* Signal geltend, wenn die Daten das Ende des Pakets sind. Die RX-MCB-Schnittstelle **530** macht ein Signal RX_CUT_THRU_SOP* mit den RX_PKT_AVAIL* und RX_SOP* Signalen geltend, wenn das Paket im CT-Modus übertragen, aber durch den Speicher **212** gepuffert wird, z. B. für Interim-CT- oder Mittelpaket-CT-Modi. Das heißt, Interim-CT (volles Paket) wird angegeben, wenn (!RX_CUT_THRU_SOP* & !RX_PKT_AVAIL* & !RX_SOP*), und Interim-CT-Mittelpaket wird angegeben, wenn (!RX_CUT_THRU_SOP* & !RX_PKT_AVAIL* & RX_SOP*). Die RXMCB-Schnittstelle **530** macht ein Signal RX_MISS_BC* mit den RX_PKT_AVAIL* und RX_SOP* Signalen geltend, wenn die Zieladresse unbekannt war, und daher das Paket ein BC-Paket ist. Die RXMCB-Schnittstelle **530** macht ein Signal RX_GROUP_BC* mit den Signalen RX_PKT_AVAIL* und RX_SOP* geltend, wenn das GROUP-Bit in dem Paketvorspann gesetzt ist, sodass das Paket wiederum ein BC-Paket ist. Die RXMCB-Schnittstelle **530** macht ein Signal RX_END_BYTE[1:0] mit den RX_PKT_AVAIL* und RX_EOP* Signalen geltende, um den Bytepfad des letzten Bytes in dem Paket anzuzeigen.

[0112] Die RXMCB-Schnittstelle **530** macht ein Signal RX_ERROR* mit den Signalen RX_PKT_AVAIL* und RX_EOP* geltend, wenn der Quellenport während des Übertragens in dem Paket einen Fehler erfasst und durch Geltendmachung des Signals ABORT_OUT* anzeigt. Verschiedene Fehlerbedingungen werden durch die Ports **104**, **110** geprüft, z. B. Erfassen eines FIFO-Overrun, eines Runt-Pakets, eines übergroßen Pakets, eines Rahmenprüfsequenz-(FCS)Fehlers oder eines Phasenverriegelungsschleifen-(PLL)Fehlers. Wenn das Signal RX_ERROR* geltend gemacht wird, lässt der Netzwerkschalter **102** das Paket fallen, wenn es im SnF-Modus übertragen wird.

[0113] Der MCB **404** macht ein Signal RX_ACK* an dem HCB **401** nach Erfassen des geltend gemachten Signals RX_PKT_AVAIL* geltend und nach Speichern der mit dem RX_PKT_AVAIL* Signal geltend gemachten zugehörigen Signale, wie oben beschrieben. Der MCB **404** macht ein Signal RX_STB* geltend, wenn er bereit ist, das nächste DWORD von Daten anzunehmen. Der MCB **404** macht ein Signal RX_PKT_COMPLETE* geltend, wenn er bestimmt, dass der HCB **402** die Daten anfordern kann. Das heißt, der MCB **404** macht das Si-

gnal RX_PKT_COMPLETE* geltend nach Erfassen des durch den HCB **402** für CT-Mo-duspakete geltend gemachten Signals RX_SOP*. Außerdem macht der MCB **404** das RX_PKT_COMPLTE* Signal geltend nach Erfassen des durch den HCB **402** für SnF-Moduspakete geltend gemachten Signals RX_EOP*. Der MCB **404** macht das Signal RX_PKT_COMPLETE* nicht geltend, wenn das Signal RX_ERROR* für ein SnF-Paket geltend gemacht war (dadurch angezeigt, dass das Signal RX_CUT_THRU* nicht mit dem RX_SOP* geltend gemacht wird). Der MCB **404** macht ein Signal RX_PKT_ABORTED* an dem HCB **402** anstelle des Signals RX_PKT_COMPLETE* geltend, wenn das Paket infolge einer Überlaufbedingung des Speichers **212**, wie durch den MCB **404** festgestellt, fallen gelassen wird.

[0114] Die TX ARB REQ LOGIC **533** empfängt eine Anforderung von der Arbitrationslogik **504**, um Paketdaten aus dem Speicher **212** zum Senden durch einen verfügbaren Zielport zurückzugewinnen, wobei die Anforderung typischerweise durch den TXNW Arbiter **515** hervorgebracht wird. Die TX ARB REQ LOGIC **533** macht folglich ein Sendeanforderungssignal TX_ARB_REQ* an dem MCB **404** geltend, während auch die Zielportnummer auf Signalen TX_ARB_PORT[4:0] und eine maximale Übertragungslänge für jeden Datenabschnitt auf Signalen TX_ARB_XSIZE[2:0] geltend gemacht werden. Die maximale Übertragungslänge ist für die TXBUFs **524**, **526** als 000b = 16 Byte, 001b = 32 Byte, 010b = 64 Byte, 011 = 128 Byte und 100 = 256 Byte definiert. Der MCB **404** speichert diese Werte und macht ein Bestätigungssignal TX_ARB_ACK* an der TX ARB REQ LOGIC **533** geltend. Der MCB **404** gewinnt dann die verlangten Daten aus dem Speicher **212** zurück und schreibt die Daten in einen der TXBUFs **524**, **526**.

[0115] Daten werden an die TXBUFs **524**, **526** in den HCB **402** über einen Speicherdateneingangsbus MemDataIn oder MDI[31:0] übertragen. Die TXMCB-Schnittstelle **531** gibt ein Signal TX_BUF_AVAIL* aus, wenn der FIFO-Steuerblock **529** feststellt, dass einer der TXBUFs **524**, **526** verfügbar ist, um Daten von dem MCB **404** zu empfangen. Der MCB **404** gibt ein Strobesignal TX_STB* aus, wenn Daten verfügbar sind, um durch die TXMCB-Schnittstelle **531** des HCB **402** zur Speicherung in dem verfügbaren TXBUF **524** oder **526** abgetastet zu werden. Der MCB **404** gibt mehrere Signale gleichzeitig mit dem TX_STB* Signal zum Identifizieren von Eigenschaften der Daten aus. Das heißt, der MCB **404** macht ein Signal TX_SOP* mit dem TX_STB* Signal für den Beginn oder Start eines Pakets von dem Speicher **212** geltend. Der MCB **404** macht ein Signal TX_AIFCS* mit dem TX_STB* Signal geltend, wenn der Quellenport der PCB **406** ist, der die CPU **230** angibt. Der MCB **404** macht eine Binärzahl auf Signalen TX_CNT[5:0] mit dem TX_STB* Signal geltend, wo die TX_CNT[5:0] Signale die Zahl von DWORDS (minus ein DWORD) angeben, um in den ausgewählten TX FIFO zu schreiben. Der MCB **404** macht ein Signal TX_EOP* mit dem TX_STB* Signal für das Ende des Pakets von dem Speicher **212** geltend. Der MCB **404** macht auch ein Pufferkettenendesignal TX_EOBC* mit den Signalen TX_EOP* und TX_STB* geltend, wenn es für den einzelnen Zielport im Speicher **212** keine Daten mehr gibt. Der MCB **404** macht auch Byteendesignale TX_END_BYTE[1:0] mit den Signalen TX_EOP* und TX_STB* geltend, um den Bytepfad des letzten Bytes in dem Paket anzuzeigen.

[0116] Für BC-Pakete macht der MCB **404** ein Signal BC_PORT_STB* geltend, während eine BC-Bitmap auf den MDI[31:0] Signalen geltend gemacht wird. Der FIFO-Steuerblock **529** erfasst die Geltendmachung des BC_PORT_STB* Signals, verriegelt die MDI[31:0] Signale und speichert das Ergebnis in einem internen BCBITMAP[28:0] Register. Der FIFO-Steuerblock benutzt die Werte in dem BCBITMAP-Register, wenn Bits in ein Feld von Speicherbits TXMEMCYC[28:0] in der Sendeliste **510** gestellt werden.

[0117] Fig. 5B ist ein Diagramm, das mehrere der Register in den Registern **506** veranschaulicht. Die CT_SNF-Register **507** enthalten eine Anordnung von programmierbaren Quellenportmodusbits SRC CT_SNF[28:0], wobei jedes einem der Ports Port28–Port0 entspricht und durch die CPU **230** programmiert wird, um die gewünschte Betriebsart zwischen CT und SnF zu identifizieren, wenn der entsprechende Port ein Quellenport ist. Das heißt, wenn das SRC CT_SNF Bit für einen gegebenen Port gesetzt wird, wird gewünscht, diesen Port im CT-Modus zu betreiben, wenn der Port als ein Quellenport agiert. Wenn das SRC CT_SNF Bit gelöscht wird, wird gewünscht, diesen Port im SnF-Modus zu betreiben, wenn der Port als ein Quellenport agiert. Desgleichen enthalten die CT_SNF-Register **507** eine Anordnung von programmierbaren Zielportmodusbits DEST CT_SNF[28:0], wobei jedes einem der Ports Port28 bis Port0 entspricht und durch die CPU **230** programmiert wird, um die gewünschte Betriebsart zwischen CT und SnF zu identifizieren, wenn der entsprechende Port als ein Zielport für ein Unicast-Paket agiert. Der CT-Modus wird nur gewünscht, wenn sowohl der Quellen- als auch der Zielport für den CT-Modus in den CT_SNF-Registern **507** bezeichnet sind.

[0118] Die Empfangsliste **509** umfasst eine Vielzahl von Registern zum Speichern von entsprechenden Empfangsprioritätszählwerten, bezeichet als RXPORTBUFx[4:0] Zählwerte, wo "x" die Portnummer widerspiegelt. Jeder RXPORTBUFx Zählwert ist in der gezeigten Ausführung fünf Bits zum Priorisieren von bis zu 32 Ports. Die Empfangsliste **509** enthält eine entsprechende Anordnung von Portmaskenbits RXPRTMSK[28:0], wo je-

des RXPRTMSK-Bit durch die RX-Abfragezustandsmaschine **502** gesetzt wird, wenn dieses RXPRTMSK-Bit anfangs auf logisch 0 ist, um anzuzeigen, dass momentan keine Priorität zugewiesen ist, und wenn das betreffende PKT_AVAILm* Signal dann geltend gemacht wird. Zu dieser Zeit weist die RX-Abfragezustandsmaschine **502** eine Prioritätsnummer in dem entsprechenden RXPORT-BUFx Register zu. Die Prioritätsnummer bleibt gültig, bis der Port bedient wird. Während das RXPRTMSK-Bit gesetzt ist, ignoriert die RX-Abfragezustandsmaschine **502** wertere Anforderungen durch Maskieren nachfolgender Geltendmachungen des entsprechenden PKT_AVAILm* Signals. Die HSB-Steuerung **505** löscht das RXPRTMSK-Bit während jeder Lesezyklusübertragung von dem betreffenden Port für dieses Paket außer für die erste Übertragung für ein neues Paket. Die HASH REQ LOGIC **532** löscht das RXPRTMSK-Bit während der ersten Lesezyklusübertragung, wenn das Paket entsprechend der SnF-Betriebsart zu übertragen ist. Die HSB-Steuerung **505** löscht das RX-PRTMSK-Bit während der ersten Schreibzyklusübertragung an den Zielport, wenn das Paket im CT-Modus übertragen wird.

[0119] Die Empfangsliste **509** enthält eine Anordnung von In-Queue-Bit RXINQUE[28:0], die jeweils gesetzt werden, wenn das entsprechende RXPRTMSK-Bit gesetzt wird. Jedes RX-INQUE-Bit zeigt an, ob der Prioritätswert gültig ist, und, wenn ja, dass der entsprechende Port durch die Arbitrationslogik **504** in die Arbitration einzuschließen ist. Das RXINQUE-Bit wird durch einen Arbiter in der Arbitrationslogik **504** gelöscht, wenn der befreffende Port an den Haupt-Arbiter **512** übergeben wird, um als der nächste Port zum Übertragen von Daten für ein neues Paket oder für ein sich fortsetzendes SnF-Paket bedient zu werden.

[0120] Die Empfangsliste **509** enthält eine Anordnung von Speicherbits RXMEMCYC[28:0], die anzeigen, ob der befreffende Port Daten im Speicher **212** empfangen soll. Dies kommt für die SnF-, Interim-CT- und die Interim-Mittelpaket-CT-Betriebsart vor. Die HASH REQ LOGIC **532** setzt ein entsprechendes RXMEMCYC-Bit, wenn der SnF- oder Interim-CT-Modus bestimmt wird. Der Haupt-Arbiter **512** setzt das RXMEMCYC-Bit für Mittelpaket-Interim-CT-Moduspakete, wenn der Zielport nicht anzeigt, dass während des normalen CT-Modus Pufferplatz verfügbar ist. Die HSB-Steuerung **505** löscht das RXMEMCYC-Bit bei der letzten Lesezyklusübertragung von Daten für den betreffenden Port.

[0121] Die Empfangsliste **509** enthält eine Anordnung von aktiven oder CT-Bits RXACTCYC[28:0], die angeben, ob der betreffende Port ein Datenpaket entsprechend der normalen CT-Betriebsart überträgt. Die HASH REQ LOGIC **532** setzt ein entsprechendes RXACTCYC-Bit für CT-Moduspakete. Die HSB-Steuerung **505** löscht das RXACTCYC-Bit bei einem Lesezyklus der letzten Datenübertragung eines Pakets für den entsprechenden Port. Der Haupt-Arbiter **512** löscht das RXACTCYC-Bit, wenn das Bit für den CT-Modus gesetzt ist, und wandelt das Paket in ein Mittelpaket-CT-Paket um.

[0122] Die Sendeliste **510** enthält eine Vielzahl von Registern zum Speichern von entsprechenden Sendeprioritätszählwerten, bezeichnet als die TXPORTBUFx[4:0] Zählwerte, wo "x" die Portnummer widerspiegelt. TX-PORTBUFx Zählwert ist in der gezeigten Ausführung fünf Bits zum Priorisieren von bis zu 32 Ports. Die Sendeliste **510** enthält eine entsprechende Anordnung von Portmaskenbits TXPRTMSK[28:0], wo jedes TX-PRTMSK-Bit durch die TX-Abfragezutandsmaschine **503** gesetzt wird, wenn dieses TXPRTMSK-Bit anfangs auf logisch 0 ist, um anzuzeigen, dass momentan keine Priorität zugewiesen ist, und wenn das betreffende BUF_AVAILm* Signal dann geltend gemacht wird. Zu dieser Zeit weist die TX-Abfragezustandsmaschine **503** eine Prioritätsnummer in dem entsprechenden TXPORTBUFx Register zu. Die Prioritätsnummer bleibt gültig, bis der Port bedient wird. Während das TXPRTMSK-Bit gesetzt ist, ignoriert die TX-Abfragezustandsmaschine **503** wertere Anforderungen durch Maskieren nachfolgender Geltendmachungen des entsprechenden BUF_AVAILm* Signals. Die HSB-Steuerung **505** löscht das TXPRTMSK-Bit während jeder Lesezyklusübertragung von dem betreffenden Port für dieses Paket außer für die erste Übertragung für ein neues Paket. Die HSB-Steuerung **505** löscht das TXPRTMSK-Bit während jeder Schreibzyklusübertragung von Paketdaten an den Zielport.

[0123] Die Sendeliste **510** enthält eine Anordnung von In-Queue-Bit TXINQUE[28:0], die jeweils gesetzt werden, wenn das entsprechende TXPRTMSK-Bit gesetzt wird. Jedes TXINQUE-Bit zeigt an, ob der Prioritätswert gültig ist, und, wenn ja, dass der entsprechende Port durch die Arbitrationslogik **504** in die Arbitration einzuschließen ist. Das TXINQUE-Bit wird durch einen Arbiter in der Arbitrationslogik **504** gelöscht, wenn der befreffende Port an den Haupt-Arbiter **512** übergeben wird, um als der nächste Port zum Übertragen von Daten für ein neues Paket oder für ein sich fortsetzendes SnF-Paket bedient zu werden.

[0124] Die Sendeliste **510** enthält die Anordnung von Speicherbits TXMEMCYC[28:0], die angeben, ob der befreffende Port von dem Speicher **212** empfangene Daten senden soll. Dies kommt für die SnF-, Interim-CT- und die Interim-Mittelpaket-CT-Betriebsart vor. Der FIFO-Steuerblock **529** setzt ein oder mehr TXMEM-

CYC-Bits als Reaktion auf die Geltendmachung des RX_PKT_COMPLETE* Signals durch den MCB **404** nach Empfangen von Daten von dem HCB-**402**. Für Unicast-Pakete wird nur eines der TXMEMCYC-Bits gesetzt. Für BC-Pakete verwendet der FIFO-Steuerblock **529** sein BCBITMAP-Register, um zu bestimmen, welches Bit zu setzen ist. Für SnF-Moduspakete werden die TXMEMCYC-Bits gesetzt, nachdem das ganze Paket an den MCB **404** zur Speicherung im Speicher **212** übertragen ist. Für Interim-CT-Moduspakete, einschließlich Mittelpaket-Interim-Modus-CT-Paketen wird ein TXMEM-CYC-Bit während der ersten Übertragung von Daten an den MCB **404** gesetzt. Die HSB-Steuerung **505** löscht ein TXMEMCYC-Bit bei der letzten Schreibzyklusübertragung von Daten an einen betreffenden Port. Dies kommt vor, wenn der MCB **404** auch das Signal TX_EOBC* geltend macht, um anzuzeigen, dass für diesen Port im Speicher **212** keine Daten mehr vorhanden sind.

[0125] Die Sendeliste **510** enthält eine Anordnung von Sende-CT-Bits TXCTCYC[28:0], die angeben, ob es Daten in einem der RXBUFs **520**, **522** zum direkten Senden an den betreffenden Zielport entsprechend der normalen CT-Betriebsart gibt. Die HASH REQ LOGIC **532** setzt ein entsprechendes TXCTCYC-Bit bei der ersten Datenübertragung des Pakets. Die HSB-Steuerung **505** löscht das TXCTCYC-Bit bei der ersten Schreibzyklusübertragung von Daten an den entsprechenden Zielport.

[0126] Die Sendeliste **510** enthält eine Anordnung von aktiven CT-Bits TXACTCTCYC[28:0], die angeben, ob der betreffende Port beim Übertragen eines Pakets entsprechend der CT-Betriebsart involviert ist. Die HASH REQ LOGIC **532** setzt ein entsprechendes TXACTCYC-Bit, wenn sie feststellt, dass das Paket entsprechend dem CT-Modus zu übertragen ist. Der FIFO-Steuerblock **529** löscht das TXACTCYC-Bit während der ersten Übertragung von Daten an den MCB **404** zur Speicherung im Speicher **212**, wenn das Paket vom CT-Modus in den Mittelpaket-Interim-CT-Modus umgewandelt wird. Die HSB-Steuerung **505** löscht auch das TXACTCYC-Bit während der letzten Datenübertragung eines Pakets.

[0127] Die Gewichtsfaktoren **508** umfassen eine Anordnung von Port-Gewichtsfaktoren PORTWTx [4:0] für jeden der Ports Port0–Port28, wo "x" die einzelne Portnummer angibt. Die PORTWT-Gewichtsfaktoren sind bevorzugt einmalig und vom Benutzer vorprogrammiert, um benutzerprogrammierbare Priorität der Ports bereitzustellen. In der gezeigten Ausführung wird der gleiche Gewichtsfaktor jedem Port für den Empfangs- und den Sendefall zugewiesen, obwohl verschiedene Gewichtsfaktoren für den Sende- und den Empfangsvorgang definiert werden könnten.

[0128] Fig. 5C ist ein Zustandsdiagramm, das die Empfangsabfrageoperation der RX-Abfragezustandsmaschine **502** veranschaulicht. Die Hauptfunktion der RX-Abfragezustandsmaschine **502** ist das Überwachen der PKT_AVAILm* Signale, Zuweisen von Prioritätszählungen RXPORTBUFx und Setzen der RXPRTMSK-Bits in der Empfangsliste **509**. Übergänge zwischen Zuständen basieren auf Übergangen oder Zyklen des CLK-Signals und dem Status des STROBE* Signals. Zu Anfang nach Einschalten und Konfiguration wird die Empfangsprioritäts-Zählzahl RPCOUNT auf null gesetzt, und die RX-Abfragezustandsmaschine **502** wird in einen anfänglichen Leerlaufzustand **550** gebracht. Ferner werden RXINCCNTBY[7:0] Logikbits, die PKT_AVAILm* Signalen entsprechen, gelöscht. Die RX-Abfragezustandsmaschine **502** bleibt im Zustand **550**, während das Signal STROBE* nicht geltend gemacht ist, was der Fall ist, wenn das STROBE* Signal hoch oder auf logisch 1 ist. Wenn das STROBE* Signal tief geltend gemacht wird, geht die Operation in einen CLK-Wartezustand (RxPoll-Wait) **552** über.

[0129] Als Reaktion auf das Abtasten des geltend gemachten STROBE* Signals entworten die QC-Vorrichtungen **202**, die TPI **220** und der PCB **406** jeweils durch Geltendmachen eines entsprechenden der PKT_AVAILm* Signale, sonst als PKT_AVAIL[7:0]* Signale bezeichnet, nach einem CLK-Zyklus. Der Vorgang geht daher nach einem CLK-Zyklus zu Zustand **554**, um das Abfragen aller PKT_AVAIL[7:0]* Signale zu beginnen. Der Vorgang geht bei aufeinanderfolgenden Zyklen des CLK-Signals vom Zustand **554** zum Zustand **556**, dann zum Zustand **558** und dann zum Zustand **560** über. Der Vorgang kehrt vom Zustand **560** zum Zustand **554** zurück und fährt in einer Schleife fort, während das STROBE* Signal geltend gemacht bleibt. Das STROBE* Signal ist jedoch vorzugsweise periodisch und wird für einen CLK-Tyklus negiert und dann für die nächsten drei CLK-Zyklen erneut geltend gemacht. Der Vorgang kehrt daher zum Zustand **550** zurück, wenn das STROBE* Signal in Schritt **560** ungeltend gemacht wird. In jedem der Zustände **554**, **556**, **558** und **560** wird eine anfängliche Arbitrations-Zähllogik-Operation basierend auf einen Zuwachs der RXNEWCNT- und RXA-CTCNT-Zahlen verglichen mit der RPCOUNT-Zahl durchgeführt, um festzustellen, ob irgendwelche der übrigen Logikoperationen durchgeführt werden.

[0130] Wenn die anfängliche Arbitrations-Zähllogik-Operation in Schritt **554** wahr ist, werden neun logische Operationen, bezeichnet 1–9, durchgeführt, wo die ersten acht Operationen den Ports Port0, Port4, Port8,

Port16, Port20, Port24 bzw. Port28 für den ersten Port jeder der QC-Vorrichtungen **202**, der TPI **220** und des PCB **406** entsprechen. Für jede der acht logischen Portoperationen, 1–8, wird ein entsprechendes der PKT_AVAILm* Signale mit einem entsprechenden RXPRTMSK-Bit verglichen, um festzustellen, ob die Anforderung anzunehmen ist. Wenn die Anforderung für einen Port angenommen wird, was vorkommt, wenn das RXPRTMSK-Bit nicht vorher gesetzt wurde, wird diesem Port eine RXPORTBUFx Prioritätsnummer zugewiesen. Ferner wird das entsprechende RXPRTMSK-Bit auf logisch 1 gesetzt, um wertere Anforderungen durch diesen Port zu maskieren, und ein entsprechendes RXINCCNTBY-Bit wird auf logisch 1 gesetzt. Die neunte logische Operation wird durchgeführt, um RPCOUNT zu inkrementieren.

[0131] Wenn für Port0 PKT_AVAIL[0]* nicht geltend gemacht ist oder wenn RXPRTMSK logisch 1 ist, ist die Priorität bereits errichtet worden und wird erst geändert, wenn Port0 bedient wird. Wenn jedoch das PKT_AVAIL[0]* Signal tief geltend gemacht ist und wenn RXPRTMSK[0] logisch 0 ist, wird die entsprechende Prioritätsnummer RXPORTBUD0 gleich dem entsprechenden Gewichtsfaktor RXPORTWT0 gesetzt, wenn ein WTPRIORITY-Flag Priorität entsprechend den Gewichtsfaktoren anzeigt. Wenn jedoch das WTPRIORITY-Flag unwahr ist, wird die Prioritätsnummer RXPORTBUF0 gleich RPCOUNT gesetzt. Dann werden die RXPRTMSK[0]- und RXINCCNTBY[0]-Bits beide auf logisch 1 gesetzt. Das Setzen von RXPRTMSK[0] maskiert weitere Empfangsabfrage-Anforderungen für Port0. Das RXINCCNTBY[0]-Bit entspricht dem PKT_AVAIL[0]* Signal und wird in den übrigen logischen Operationen im Zustand **554** benutzt, um anzuzeigen, dass der Prioritätswert für Port0 gesetzt wurde.

[0132] Wenn in der zweiten logischen Operation, die Port4 entspricht, PKT_AVAIL[1]* nicht tief geltend gemacht ist oder wenn RXPRTMSK[4] logisch 1 ist, ist die Priorität bereits festgelegt worden und wird erst geändert, wenn Port4 bedient wird. Wenn jedoch das PKT_AVAIL[1]* Signal tief geltend gemacht ist und wenn RXPRTMSK[4] logisch 0 ist, wird die entsprechende Prioritätsnummer RXPORTBUF4 gleich dem entsprechenden Gewichtsfaktor RXPORTWT4 gesetzt, wenn ein WTPRIORITY-Flag Priorität entsprechend den Gewichtsfaktoren anzeigt. Wenn jedoch das WTPRIORITY-Flag unwahr ist, wird die Prioritätsnummer RXPORTBUF4 gleich RPCOUNT plus RXINCCNTBY[0] gesetzt. Auf diese Weise wird, wenn WTPRIORITY unwahr ist, RXPORTBUF4 eine Prioritätsnummer von RPCOUNT erteilt, wenn Port0 keine Prioritätsnummer erteilt wurde, oder eine Prioritätsnummer RPCOUNT + 1 erteilt, wenn Port0 eine Priorität erteilt wurde. Dies stellt sicher, dass Port0 und Port4 nicht die gleiche Prioritätsnummer erhalten. Das RXPORTMSK[4]-Bit wird dann auf logisch 1 gesetzt, um wertere Abfrageanforderungen zu maskieren. Auf diese Weise ist die jedem Port erteilte Prioritätsnummer entweder der vorbestimmte Gewichtsfaktor für diesen Port, oder die Prioritätsnummer ist gleich RPCOUNT plus der Zahl von Ports, die eine niedrigere Portnummer haben und eine Prioritätsnummer zur gleichen Zeit erhalten haben.

[0133] Die nächsten sechs logischen Operation sind ähnlich der zweiten logischen Operation. Wenn in der achten logischen Operation, die dem PCB **406** entspricht, PKT_AVAIL[7]* nicht tief gesetzt ist, oder wenn RXPRTMSK[28] gleich logisch1 ist, ist die Priorität bereits festgelegt worden und wird erst geändert, wenn der PCB **406** bedient wird. Wenn jedoch das PKT_AVAIL[7]* Signal tief geltend gemacht ist und wenn RXPRTMSK[28] logisch 0 ist, wird die entsprechende Prioritätsnummer RXPORTBUF28 für den PCB **406** gleich dem entsprechenden Gewichtsfaktor RXPORTWT28 gesetzt, wenn ein WTPRIORITY Flag Priorität entsprechend den Gewichtsfaktoren anzeigt. Wenn jedoch das WTPRIORITY-Flag unwahr ist, wird die Prioritätsnummer RXPORTBUF28 gleich RPCOUNT plus die "Bitsumme" von RXINCCNTBY[6:0] gesetzt. Die Bitsumme von RXINCCNTBY[6:0] ist gleich der Zahl von Prioritätswerten, die in den vorherigen sieben logischen Operationen zugewiesen wurden. Dem PCB **406** wird daher eine Prioritätsnummer gleich dem vorbestimmten Gewichtsfaktor erteilt, oder die Prioritätsnummer ist RPCOUNT plus die Zahl von Ports, die eine niedrigere Portnummer haben und gleichzeitig eine Prioritätsnummer erhalten haben. Eine neunte logische Operation wird im Zustand **554** durchgeführt, um RPCOUNT um die Bitsumme von RXINCCNTBY[7:0] zu erhöhen, die gleich der Zahl von Ports ist, denen im Zustand **554** eine Priorität zugewiesen wurde. Diese Operation stellt sicher, dass RPCOUNT für den nächsten Satz von logischen Operationen im Zustand **556** inkrementiert wird.

[0134] Wenn z. B. alle mit dem ersten gemultiplexten Bit der PKT_AVAIL[7:0]* Signalen verbundenen Ports oder Ports Port0, Port4, Port8, Port12, Port16, Port20, Port24 und Port28 im Zustand **554** gleichzeitig anfordern und RPCOUNT anfangs null ist und keines der entsprechenden RXPTRMSK-Bits vorher gesetzt wurde, und wenn WTPRIORITY unwahr ist, werden den entsprechenden Prioritätswerten RXPORTBUFx (X = 0, 4, 8, 12, 16, 20, 24 und 28) Prioritätsnummern von 0, 1, 2, 3, 4, 5, 6, und 7 im Zustand **554** zugewiesen. Dann wird RPCOUNT gleich 8 gesetzt. Wenn als anderes Beispiel die Ports Port4, Port12 und Port20 die einzigen Ports sind, die Service anfordern, werden den Prioritätsnummern RXPORTBUFx (x = 4, 12, 20) Prioritätsnummern von 0, 1 bzw. 2 zugewiesen, wenn WTPRIORITY unwahr ist, und dann wird RPCOUNT gleich 4 gesetzt. Die Bitsummenoperation stellt sicher, dass jedem Port eine einmalige Prioritätsnummer gegeben wird, wenn meh-

rere Ports gleichzeitig Service anfordern. Auf diese Weise sind die Prioritätsnummern entsprechend einem 'wer zuerst kommt, mahlt zuerst' (FCFS) Prioritätsschema, aber eine bestimmte Reihenfolge ist vorbestimmt, um Priorität festzulegen, um gleichzeitige Zuweisungen zu handhaben.

[0135] Die logischen Operationen in den Zuständen **556**, **558** und **560** sind ähnlich den im Zustand **554** durchgeführten. Im Zustand **556** werden, wenn die anfängliche logische Arbitrationszähloperation wahr ist, acht logische Operationen durchgeführt, einschließlich sieben Operationen, die mit dem zweiten Port jeder der QC-Vorrichtungen **202** und der TPI **220** verbunden sind, basierend auf den PKT_AVAIL[6:0]* Signalen, was die Ports Port1, Port5, Port9, Port13, Port17, Port21 und Port25 einschließt, und die achte logische Operation von Zustand **554** wird für den Port Port28 für die CPU **230** wiederholt. Im Zustand **558** werden sieben mit dem dritten Port jeder der QC-Vorrichtungen **202** und der TPI **220** verbundene logische Operationen basierend auf den PKT_AVAIL[6:0]* Signalen durchgeführt, die die Ports Port2, Port6, Port10, Port14, Port18, Port22 und Port26 einschließen, und die achte logische Operation von Zustand **554** wird für den Port Port28 für die CPU **230** wiederholt. Im Zustand **560** werden sieben mit dem vierten Port jeder der QC-Vorrichtungen **202** und der TPI **220** verbundene logische Operationen basierend auf den PKT_AVAIL[6:0]* Signalen durchgeführt, die die Ports Port3, Port7, Port11, Port13, Port15, Port19, Port23 und Port27 einschließen, und die achte logische Operation von Zustand **554** wird für den Port Port28 für die CPU **230** wiederholt. In jedem der Zustände **556**, **558** und **560** wird eine letzte logische Operation durchgeführt, um RPCOUNT mit die Bitsumme der RXINCCNTBY-Bits in einer ähnlichen Weise wie vorher beschrieben zu aktualisieren.

[0136] Fig. 5D ist ein Zustandsdiagramm, das die Sendeabfrage-Operation der TX Abfragezustandsmaschine **503** veranschaulicht. Die TX-Abfragezustandsmaschine **503** arbeitet in einer ähnlichen Weise wie die RX-Abfragezustandsmaschine **502** und umfasst Zustände **561**, **562**, **564**, **566**, **568** und **570**, die analog zu den Zuständen **550**, **552**, **554**, **556**, **558** und **560** sind. RPCOUNT ist jedoch durch TPCOUNT ersetzt, und die änfängliche logische Arbitrationszähloperation wird basierend auf einem Zuwachs der TXNEWCNT- und TXTACTCNT-Nummern verglichen mit der TPCOUNT-Nummer durchgeführt, um festzustellen, ob irgendwelche der restlichen logischen Operation durchgeführt werden. Die BUF_AVAILm* Signale ersetzen die PKT_AVAILm* Signale, und TXPRTMSK-Bits ersetzen die RXPRTMSK Bits. Ferner wird für jede Portgleichung jedes TXPRTMSAK-Bit mit einem logischen Ausdruck UND-verknüpft, der auf entsprechenden Bits der TXMEMCYC-, TXCTACTCYC- und TXCTCYC-Bitanordnungen basiert. Das heißt, die entsprechenden Bits der TXMEMCYC-, TXCTACTCYC- und TXCTCYC-Bitanordnungen werden miteinander ODER-verknüpft, sodass die Priorität einem Zielport nur zugewiesen wird, wenn Daten in dem EPSM **210** oder dem Speicher **212** zum Senden durch diesen Port vorhanden sind. Ferner ersetzen TXPORTBUFx Prioritätsnummern die RXPORTBUFx Prioritätsnummern. TXPORTWT-Gewichtsfaktoren ersetzen die RXPORTWT-Gewichtsfaktoren, und TXINCCNTBY-Bits ersetzen die RXINCCNTBY-Bits. Auf diese Weise bezeichnet jeder Port und der PCB **406** ein betreffendes der BUF_AVAIL* Signale als Reaktion auf das STROBE* Signal, und die TX-Abfragezustandsmaschine **503** weist eine Prioritätsnummer basierend auf den Gewichtsfaktoren oder FCFS mittels TPCOUNT zu und legt die Priorität entsprechend fest.

[0137] Man wird einsehen, dass die Abfragelogik **501** das STROBE* Signal periodisch oder kontinuierlich umschaltet und die PKT_AVAILm* und BUF_AVAILm* Signale für jeden der Ports **104**, **110** und den PCB **406** überwacht, um jedem der anfordernden Ports Priorität zuzuweisen und die entsprechenden Abfragemaskenbits zu setzen. Die zugewiesene Priorität basiert auf den vorprogrammierten Gewichstfaktoren, wenn WTPRIORITY wahr ist, oder auf FCFS, wenn WTPRIORITY unwahr ist. Die Priorität bleibt statisch, bis der Port bedient wird. Schließlich wird der Port bedient, und das Maskenbit wird gelöscht, wie unten beschrieben.

[0138] Die Arbiter **513**–**516** wählen zwischen den Ports **104**, **110**, und dem PCB **406** basierend auf mehreren Arbitrationsschemas aus, wo das einzelne Arbitrationsschema benutzerprogrammierbar ist. Das erste ist ein Umlauf-Schema, wo die Ports in jeder beliebigen Reihenfolge, z. B. Port0, Port1, ..., Port28 oder dergleichen überprüft werden, oder die Reihenfolge wird durch die in den PORTWTx Registern vorprogrammierten Gewichtsfaktoren **508** ausgewählt. In der gezeigten Ausführung werden die Gewichtsfaktoren **508** benutzt, um die umlaufende Reihenfolge zuzuweisen, und werden in die betreffenden RXPORTBUFx und TXPORTBUFx Zählungen programmiert. Der RX NW Arbiter **513** benutzt und inkrementiert die RXNEWCNT-Prioritätsnummer, der RX ACT Arbiter **514** benutzt und inkrementiert die RXACTCNT-Prioritätsnummer, der TX NW Arbiter **515** benutzt und inkrementiert die TXNEWCNT-Prioritätsnummer, und der TX CT Arbiter **516** benutzt und inkrementiert TXCTCNT-Prioritätsnummer. Für das Umlauf-Schema überprüfen die Arbiter **513**, **514** jeweils die RXINQUE Werte, um die aktiven Empfangsports, die Servive anfordern, zu bestimmen, und vergleichen ihre jeweilige Prioritätsnummer (RXNEWCNT, RXACTNT) mit den Werten in den RXPORTBUFx Zählungen der aktiven Ports, um den nächsten zu bedienenden Port zu bestimmen. Ferner überprüfen die Arbiter **515**, **516** jeweils die TXINQUE Werte, um die aktiven Empfangsports, die Servive anfordern, zu bestimmen, und verglei-

chen dann ihre jeweilige Prioritätsnummer (TXNEWCNT, TXACTNT) mit den Werten in den TXPORT_BUFx Zählungen der aktiven Ports, um den nächsten zu bedienenden Port zu bestimmen. Da die Gewichtsfaktoren eine bestimmte Reihenfolge bestimmen, werden die Ports in einer umlaufenden Weise geordnet.

[0139] Das zweite Arbitrationsschema ist FCFS, wo WTPRIORITY unwahr ist und die Ports basierend auf der Reihenfolge, in der sie Service angefordert haben, bedient werden, wie durch RXPORTBUFx unf TXPORTBUFx Prioritätsnummern angegeben. Das FCFS-Schema arbeitet in ähnlicher Weise wie das Umlauf-Verfahren, außer dass die RXPORTBUFx und TXPORTBUFx Zählungen entsprechend den RPCOUNT- und TPCOUNT-Werten programmiert werden, wie vorher beschrieben. Dann überprüfen die RX-Arbiter **513**, **514** jeweils die RXINQUE Werte, um die aktiven Empfangsports, die Service anfordern, zu bestimmen, und vergleichen dann ihre jeweilige Prioritätsnummer (RXNEWCNT, RXACTCNT) mit den Werten in den RXPORTBUFx Zählungen der aktiven Ports, um den nächsten zu bedienenden Port zu bestimmen. Ferner überprüfen die TX-Arbiter **515**, **516** jeweils die TXINQUE Werte, um die aktiven Empfangsports, die Service anfordern, zu bestimmen, und vergleichen dann ihre jeweilige Prioritätsnummer (TXNEWCNT, TXACTCNT) mit den Werten in den TXPORTBUFx Zählungen der aktiven Ports, um den nächsten zu bedienenden Port zu bestimmen. Da die RPCOUNT- und TPCOUNT-Werte die Reihenfolge bestimmen, werden die Ports in der FCFS-Weise geordnet.

[0140] Ein anderes Schema ist das gewichtete Prioritätsschema, wo WTPRIORITY wahr ist und die RXPORTWTx- und TXPORTWTx-Nummern in entsprechende der RXPORTBUFx- und TXPORTBUFx_Register kopiert und zum Bestimmen der Priorität benutzt werden. Jedoch bestimmen die RX-Arbiter **513**, **514** die Priorität aus einer RX HIGH PRIORITY Nummer, und die TX-Arbiter bestimmen die Priorität aus einer TX HIGH PRIORITY Nummer. Die RX HIGH PRIORITY Nummer wird durch Identifizieren der höchsten Prioritätsnummer (oder der niedrigsten Nummer) in den RXPORTBUFx-Zählungen der aktiven Empfangsports bestimmt, wo die aktiven Empfangsports aus den RXINQUE-Werten bestimmt werden. Desgleichen wird die TX HIGH PRIORITY Nummer durch Identifizieren der höchsten Prioritätsnummer (oder der niedrigsten Nummer) in den TXPORTBUFx-Zählungen der aktiven Sendeports bestimmt, wo die aktiven Sendeports aus den TXINQUE-Werten bestimmt werden. In dieser Weise wird ein aktiver (Service anfordernder) Port mit dem höchsten Gewichtsfaktor jedes Mal ausgewählt, um so das gewichtete Prioritätsschema zu implementieren.

[0141] Der RX NW Arbiter **513** behandelt alle an den Ports Port0–Port28 empfangenen neuen Paketvorspanndaten und sich fortsetzende SnF-Moduspaketdaten, die an einen der RX BUFs **520**, **522** übertragen werden. Der RX NW Arbiter **513** aktualisiert die RXNEWCNT-Nummer und überprüft die Empfangsliste **509**, um festzustellen, welche der Ports Port0–Port28 sein Empfangskriterium erfüllen. Das Empfangskriterium für den RX NW Arbiter **513** wird durch die Ports erfüllt, deren jeweiliges RXINQUE-Bit gesetzt und deren RXACTCYC-Bit nicht gesetzt ist. Das Empfangskriterium für den RX NW Arbiter **513** schließt auch Ports ein, deren jeweilige RXINQUE- und RXMEMCYC-Bits beide gestzt sind. Der RX NW Arbiter **513** schlichtet dann zwischen den Ports, die sein Empfangskriterium erfüllen, und entsprechend einem gewählten Arbitrationsschema, wie vorher beschrieben. Nach Wählen eines Ports und Definieren eines Zyklusses fordert der RX NW Arbiter **513** den Haupt-Arbiter **512** auf, einen Lesezyklus auszuführen. Wenn der RX NW Arbiter **513** das nächste Mal durch den Haupt-Arbiter **512** ausgewählt wird, löscht er das RXINQUE-Bit des zu bedienenden ausgewählten Ports. Der RX NW Arbiter **513** wiederholt fortlaufend diesen Prozess.

[0142] Der TX CT Arbiter **516** überträgt Daten in den RX BUFs **520**, **522** an einen Zielport im normalen CT-Betrieb. Der TX CT Arbiter **516** aktualisiert die TXCTNT-Nummer und überprüft die Sendeliste **510**, um festzustellen, welche der Ports Port0–Port28 sein Sendekriterium erfüllen. Das Sendekriterium für den TX CT Arbiter **516** wird von den Ports erfüllt, deren jeweilige TXINQUE- und TXCTCYC-Bits beide gesetzt sind. Der TX CT Arbiter **516** schlichtet dann zwischen den Ports die sein Sendekriterium erfüllen, und entsprechend dem gewählten Sendekriterium, wie oben beschrieben. Nach Wählen eines Ports und Definieren eines Zyklusses fordert der TX NW Arbiter **516** den Haupt-Arbiter **512** auf, einen Schreibzyklus von dem ausgewählten RX BUF **520** oder **522** an den gewinnenden Zielport auszuführen. Wenn der TX NW Arbiter **516** das nächste Mal durch den Haupt-Arbiter **512** ausgewählt wird, löscht er das TXINQUE-Bit des zu bedienenden ausgewählten Ports. Der TX NW Arbiter **516** wiederholt fortlaufend diesen Prozess.

[0143] Der RX ACT Arbiter **514** überträgt aufeinanderfolgende Paketdaten an den CT BUF **528** von einem Quellenport, der in der normalen CT-Betriebsart arbeitet, außer dem ersten Lesezyklus (der durch den RX NW Arbiter **513** gehandhabt wird). Der RX ACT Arbiter **514** aktualisiert die RXACTCNT-Nummer und überprüft die Empfangsliste **509**, um festzustellen, welche der Ports Port0–Port28 sein Empfangskriterium erfüllen. Das Empfangskriterium für den RX ACT Arbiter **514** wird von den Ports erfüllt, deren jeweilige RXINQUE- und RXACTCYC-Bits gesetzt sind und deren jeweiliges RXMEMCYC-Bit nicht gesetzt ist. Der RX ACT Arbiter **514**

schlichtet dann zwischen den Ports, die sein Empfangskriterium erfüllen, und entsprechend dem gewählten Arbitrationsschema, wie oben beschrieben. Nach Wählen eines Ports und Definieren eines Zyklusses fordert der RX ACT Arbiter **514** den Haupt-Arbiter **512** auf, einen Lesezyklus auszuführen, um Daten von dem ausgewählten Quellenport an den CT BUF **528** zu übertragen. Wenn der RX ACT Arbiter **514** das nächste Mal durch den Haupt-Arbiter **512** ausgewählt wird, löscht er das RXINQUE-Bit des zu bedienenden ausgewählten Ports. Der RX ACT Arbiter **514** wiederholt fortlaufend diesen Prozess.

[0144] Der Haupt-Arbiter **512** folgt jedem CT-Modus-Lesezyklus in den CT BUF **528** mit einem Schreibzyklus, um Daten in dem CT BUF **528** an den durch die HASH REQ LOGIC **532** angegebenen Zielport zu übertragen. Der Haupt-Arbiter **512** stellt fest, ob der Zielport beschäftigt ist, bevor er dem RX ACT Arbiter **514** erlaubt, CT-Daten an den CT BUF **528** zu übertragen. Wenn der Haupt-Arbiter **512** feststellt, dass der Zielport beschäftigt ist, wandelt er die Quellen- und Zielports in den Mittelpaket-Interim-CT-Modus um, indem er das betreffende RX-MEMCYC-Bit setzt und das betreffende RXACTCYC-Bit für den Quellenport löscht.

[0145] Der TX NW Arbiter **515** überträgt Daten von jedem der TX BUFs **524**, **526** an den HSB **206** entsprechend der SnF-Betriebsart. Der TX NW Arbiter **515** aktualisiert die TXNEWCNT-Nummer und überprüft die Sendeliste **510**, um festzustellen, welche der Ports Port0–Port28 sein Sendekriterium erfüllen. Das Sendekriterium für den TX NW Arbiter **515** wird von den Ports erfüllt, deren jeweilige TXINQUE- und TXMEMCYC-Bits gesetzt sind und deren jeweiliges TXACTCTCYC-Bit nicht gesetzt ist. Der TX NW Arbiter **515** schlichtet dann zwischen den Ports, die sein Sendekriterium erfüllen entsprechend dem gewählten Arbitrationsschema. Nach Wählen eines Ports und Definieren eines Schreibzyklusses von einem der TX BUFs **524**, **526** zu dem ausgewählten Zielport fordert der TX NW Arbiter **515** den Haupt-Arbiter **512** auf, den Schreibzyklus auszuführen. Wenn der TX NW Arbiter **151** das nächste Mal durch den Haupt-Arbiter **512** ausgewählt wird, löscht er das TXINQUE-Bit des zu bedienenden ausgewählten Ports. Der TX NW Arbiter **515** wiederholt fortlaufend diesen Prozess.

[0146] Fig. 6 zeigt ein ausführlicheres Blockschaltbild des MCB **404** in dem EPSM **210**. Die MCB-Konfigurationsregister **448** sind in Fig. 6 nicht gezeigt, obwohl sie vorhanden sind und bei Bedarf für viele der Funktionsblöcke, die nun beschrieben werden, zugänglich sind. Der MCB **404** enthält eine Hash-Steuerung **602**, die mit der MCB-Schnittstelle **414** durch einen Bus **420** verbunden ist. Die Hash-Steuerung **602** enthält optional eine Hash-Speichertabelle **603**, die aus dem Speicher **212** zurückgewonnene Daten speichert. Der Hash-Speicher **603** bietet schnelleren Zugriff auf kürzlich aus dem Speicher **212** gezogene Daten, anstatt einen weiteren Speicherzyklus zu benötigen, um kürzlich erlangte Information zurückzugewinnen. Die Hash-Steuerung **602** umfasst Adresse/Länge/Status-(AD/LN/ST) Ausgänge, die mit einem Mehrleitungs-Eingang eines Viereingang-Adress-Multiplexers (Mux) **630** über einen Bus **610** verbunden sind. Die AD/LN/ST-Ausgänge definieren eine Länge für den Speicher **212**, eine Länge der Übertragung zum Bestimmen, ob ein Burst-Zyklus durchzuführen ist oder nicht, und verschiedene Statussignale, z. B. ein Lese/Schreib-(R/W)Signal, Bytefreigaben, ein Seitentreffersignal, eine Spensignal usw. DRAM-Anforderung/Gewährung/Strobe/Steuer-(DRAM RQ/GT/STB/CTL) Signale **628** sind mit einem DRAM-Speicherarbiter **638** und mit DRAM RQ/GT/STB/CTL-Eingängen der Hash-Steuerung **602** verbunden. Der Ausgang des Mux **630** wird an AD/LN/ST-Eingänge einer DRAM-Speichersteuerung **636** geliefert, die weiter mit dem Speicher **212** durch den Speicherbus **214** verbunden ist. Die Hash-Steuerung **602** hat einen Dateneingang (DIN) zum Empfangen von Daten von einem Mem-DataIn-Ausgang der DRAM-Steuerung **636** über einen Datenbus **618**.

[0147] Eine RX HCB-Schnittstelle **601** ist dem Bus **420** verbunden, die die MDO[31:0]-Signale einschließt, und enthält einen Datenausgang zum Liefern von Daten an einen ersten Mehrleitungs-Eingang eines Viereingang-Daten-Mux **632** über einen Bus **620**, wo der Mux **632** seinen Ausgang an MemDataOut-Ausgänge der DRAM-Steuerung **636** liefert. Die RX HCB-Schnittstelle **601** enthält STB/CTL-Eingänge zum Empfangen der Strobe- und Steuersignale der DRAM RQ/GT/STB/CTL-Signale **628**. Eine RX-Steuerung **604** ist mit dem Bus **420** verbunden und hat AD/LN/ST-Ausgänge, die über einen Bus **612** mit dem zweiten Eingang des Mux **630** verbunden sind. Die RX-Steuerung **604** hat einen Datenausgang DOUT, der mit dem zweiten Eingang des Mux **632** über einen Bus **622** verbunden ist, einen Dateneingang DIN, der mit dem Bus **618** verbunden ist, SRAM RQ/GT/STB/CTL-Eingänge zum Empfangen von SRAM RQ/GT/STB/CTL-Signalen, die mit einem statischen RAM/DRAM) verbunden sind, und DRAM RQ/GT/STB/CTL-Eingänge zum Empfangen der DRAM RQ/GT/STB/CTL-Eingänge zum Empfangen der RAM RQ/GT/STB/CTL-Signale **628**.

[0148] Eine TX HCB-Schnittstelle **605** ist mit dem Bus **420** verbunden, der die MDI[31:0]-Signale umfasst, und hat einen Dateningang DIN, der mit dem Bus **618** verbunden ist, und STB/CTL-Eingänge, die Strobe- und Steuersignale der DRAM RQ/GT/STB/CTL-Signale **628** empfangen. Eine TX-Steuerung **606** ist mit dem Bus **420** verbunden und hat AD/LN/ST-Ausgänge, die über einen Bus **614** an den dritten Eingang des Mux **630** ge-

liefert werden, einen Datenausgang DOUT, der mit dem dritten Eingang des Mux **632** über einen Bus **624** verbunden ist, einen Dateneingang DIN, der mit dem Bus **618** verbunden ist, SRAM RQ/GT/STB/CTL-Eingänge zum Empfangen der SRAM RQ/GT/STB/CTL-Signale **654** und DRAM RQ/GT/STB/CTL-Eingänge zum Empfangen der DRAM RQ/GT/STB/CTL-Signale **628**. Die PCB-Schnittstelle **424** hat AD/LN/ST-Ausgänge, die mit dem vierten Eingang des Mux **630** über einen Bus **616** verbunden sind, einen Datenausgang DOUT, der mit dem vierten Eingang des Mux **626** über einen Bus **626** verbunden ist, einen Dateneingang, der mit dem Bus **618** verbunden ist, SRAM RQ/GT/STB/CTL-Eingänge zum Empfangen der SRAM RQ/GT/STB/CTL-Signale **654** und DRAM RQ/GT/STB/CTL-Eingänge zum Empfangen der DRAM RQ/GT/STB/CTL-Signale **628**.

[0149] Die Hash-Steuerung **602**, die RX-Steuerung **604**, die TX-Steuerung **606**, die PCB-Schnittstelle **424**, die RX HCB-Schnittstelle **601** und die TX HCB-Schnittstelle **605** verwenden jeweils das STB-Signal zum Synchronisieren des Datenflusses, wo die Geltendmachung des Strobe-Signals bestimmt, wenn Daten für einen Lesezyklus gültig sind, oder wenn Daten für einen Schreibzyklus zurückgewonnen werden. Die CTL-Signale sind verschiedenartige Steuersignale, wie z. B. ein Signal, das anzeigt, wenn ein Datenzyklus vollendet ist.

[0150] Der DRAM-Arbiter **638** ist weiter mit der DRAM-Steuerung **636** durch Speichersteuersignale (MEMCTL) verbunden und liefert Mux-Steuersignale (MUXCTL) an die Auswähleingänge der Multiplexer **630**, **632**. Die MEMCTL-Signale geben gewöhnlich den Anfang und das Ende jedes Speicherzyklusses an. Auf diese Weise schlichten die Hash-Steuerung **602**, die RX-Steuerung **604**, die TX-Steuerung **606** und die PCB-Schittstelle **424** den Zugriff auf die DRAM-Steuerung **636**, um einen Speicherzyklus in dem Speicher **212** auszuführen, indem betreffende Anforderungssignale geltend gemacht werden. Der DRAM-Arbiter **638** empfängt die Anforderungssignale und macht ein entsprechendes Gewährungs-(GT)Signal an einer der anfordernden Vorrichtungen **602**, **604**, **606** oder **424** gültig, um so Zugriff auf diese Vorrichtung zu gewähren. Sobald Zugriff gewährt ist, macht der DRAM-Arbiter **638** die MUXCTL-Signale an den Multiplexern **630**, **632** geltend und ermöglicht Zugriff der DRAM-Steuerung **636** durch die ausgewählte der Vorrichtungen **602**, **604**, **606** oder **424**, um, wenn gewünscht, Speicherzyklen durchzuführen, und eines der MEMCTL-Signale wird geltend gemacht, um der DRAM-Steuerung **636** den Beginn des Zyklusses anzuzeigen. Die DRAM-Steuerung **636** setzt oder negiert eines der MEMCTL-Signale, um die Vollendung eines Speicherzyklusses anzuzeigen.

[0151] Die Hash-Steuerung **602** kommuniziert mit der HASH REQ LOGIC **532**, um die Hashing-Prozedur durchzuführen, um zu bestimmen, wie ein in der HASH REQ LOGIC **532** gespeicherer neuer Paketvorspann zu behandeln ist. Die Hash-Steuerung **602** erfasst das gesetzte HASH_REQ* Signal, gewinnt die Quellen- und Ziel-Medienzugangssteuer-(MAC)Adressen aus den HASH_DA_SA[15:0] Signalen zurück und führt die Hashing-Prozedur zum Bestimmen der HASH_STATUS[1:0] Signale und zum Bereitstellen der Zielportnummer auf den HASH_DSTPRT[4:0] Signalen durch, wenn vorher im Speicher **212** gespeichert. Die RX-Steuerung **604** und die RX HCB-Schnittstelle **601** steuern und übertragen Daten von den RX BUFs **520**, **522** in den Speicher **212**. Die TX-Steuerung **606** und die TX HCB-Schnittstelle **605** steuern und übertragen hauptsächlich Daten von dem Speicher **212** an die TX BUFs **524**, **526**. Die PCB-Schnittstelle **424** ermöglicht der CPU **230** einen direkteren Zugriff auf Daten im Speicher, einschließlich des Speichers **212** und des SRAM **650**.

[0152] Das SRAM **650** ist mit einer SRAM-Steuerung **652** verbunden, die werter mit der RX-Steuerung **604**, der TX-Steuerung **606** und der PCB-Schnittstelle **424** über einen Bus **653** verbunden ist. Ein SRAM-Arbiter **651** ist mit der SRAM-Steuerung **652** durch Steuersignale SCTL verbunden ist auch mit den SRAM RQ/GT/STB/CTL-Signalen **654** zum Steuern des Zugriffs auf das SRAM **650** durch die PCB-Schnittstelle **424** verbunden. Die TX-Steuerung **606** und die RX-Steuerung **604** steuern über den Bus **653** den Zugriff auf die DRAM-Steuerung **636** in ähnlicher Weise wie der DRAM-Arbiter **638**.

[0153] Der MCB **404** umfasst das SRAM **650** zum Speichern von Paketsteuerregistern und anderen Daten, wie unter weiter beschrieben. Die Paketsteuerregister enthalten einen Satz von Zeigern auf eine RECEIVE SECTOR CHAIN pro Port, eine TRANSMIT PACKET CHAIN pro Port und eine FREEPOOL CHAIN von freien Speichersektoren im Speicher **212**. Die Paketsteuerregister enthalten weiter Steuerinformation und Parameter zum Ermöglichen der Steuerung des Flusses von Paketdaten in dem Netzwerkschalter **102**. Der Speicher **212** enthält einen Paketspeicherabschnitt, der als eine Vielzahl von zusammenhängenden und gleich großen Sektoren organisiert ist. Die Sektoren werden anfangs mit Adresszeigern und dergleichen miteinander verbunden, um die FREEPOOL CHAIN zu bilden. Sobald Paketdaten von einem Port empfangen werden, werden die Sektoren aus der FREEPOOL CHAIN gezogen und der RECEIVE SECTOR CHAIN für diesen Port hinzugefügt. Feiner werden die Paketdaten zu einer oder mehr TRANSMIT PACKET CHAINs für einen oder mehr Zielports verbunden, an die das Paket zum Übertragen zu senden ist. Der Bus **653** ermöglicht der RX-Steuerung **604**, der TX-Steuerung **606** und der CPU-Schnittselle **436**, auf die Paketsteuerregister zuzugreifen, die die Zeiger auf die Paketketten von Daten im Speicher **212** enthalten.

[0154] Die DRAM-Steuerung **636** enthält werter eine Speicher-Auffrischungslogik **660** zum Bewahren der Daten im Speicher **212**. Die Auffrischungslogik **660** kann entsprechend dem mit dem Speicherbus **214** verbundenen Speichertyp arbeiten, einschließlich FPM DRAM, EDO DRAM oder Synchron-DRAM. Auf diese Weise werden Auffrischungsfunktionen von der CPU **230** zum effizienteren Betrieb und zur verbesserten Leistung entfernt. Ein 10-Bit Speicher-Auffrischungszähler (MRC), der sich in den MCB-Konfigurationsregistern **448** befindet, definiert die Zahl von Taktzyklen zwischen Auffrischungsanforderungen. Es ist erwünscht, dass die Periode kleiner als oder gleich 15.625 µs ist. Die Vorgabe ist 208h, wo "h" einen Hexadezimalwert bezeichnet, der eine Auffrischungsperiode von etwa 15.60 µs für einen 30 ns CLK-Zyklus liefert. Bei Timeout macht der MRC-Zähler ein Signal REFREQ an dem DRAM-Arbiter **638** gehend, der eines der MEMCTL-Signale an der DRAM-Steuerung **636** geltend macht, das der Speicher-Auffrischungslogik **660** anzeigt, die Auffrischungszyklen durchzuführen. Die MCB-Konfigurationsregister **448** umfassen ein Speichersteuerregister (MRC), das den Speichertyp, die Geschwindigkeit und Konfiguration des Speichers **212** definiert. Zum Beispiel definieren 2 Bits des MRC, ob der Speichertyp FPM, EDO oder Synchron-DRAM ist. Ein werteres Bit definiert die Speichergeschwindigkeit als entweder 50 oder 60 ns. Andere Bits definieren bestimmte Betriebsarten des gewählten DRAM-Typs und zeigen auch Fehler, z. B. Paritätsfehler, an.

[0155] Fig. 7A zeigt ein ausführlicheres Blockschaltbild des PCB **406**. Der CPU-Bus **218** ist mit CPU-Schnittstellenlogik **700** in der CPU-Schnittstelle **432** verbunden, wo die CPU-Schnittstellenlogik **700** werter durch den Bus **701** mit einer QC/CPU-Schnittstelle **702** zum Anschließen des QC/CPU-Busses **204** verbunden ist. Die CPU-Schnittstellenlogik **700** liefert Daten an einen 16-Byte Empfangspuffer RX BUF **706** in den FIFOs **430**, der Daten auf dem MCB-Bus **428** geltend macht. Der MCB-Bus **428** liefert Daten an einen 16-Byte Sendepuffer TX BUF **708**, ebenfalls in den FIFOs **430**, zum Liefern von Daten an die CPU-Schnittstellenlogik **700**. Die MCB-Schnittstelle **426** steuert den Datenfluss zwischen der CPU-Schnittstellenlogik **700** und dem MCB-Bus **428**. Die CPU-Schnittstellenlogik **700** ist dem RX BUF **706**, dem TX BUF **708** und der MCB-Schnittstelle **426** durch Bussignale **703** verbunden.

[0156] Die CPU-Schnittstellenlogik **700** ist mit der Register-Schnittstelle **440** durch den Bus **442** verbunden, wo die Register-Schnittstelle **440** den Zugriff auf andere Konfigurationsregister in dem EPSM **210** ermöglicht. Die CPU-Schnittstellenlogik **700** ist auch mit einem Satz von PCB-Registern **704** durch den Bus **442** verbunden, zum Definieren des Eingabe/Ausgabe-(E/A)Raumes der CPU **230**, z. B. Unterbrechungsregister, Konfigurationsregister, Paketinformationsregister, speicherbezogene Register, Setup- und Statusregister, Schnittstellen- und Überwachungsregister, Statistikregister, Modusregister, Arbitrationsregister usw.

[0157] Während des Einschaltens und Konfigurierens programmiert die CPU **230** Anfangs- oder Vorgabewerte in den PCB-Registern **704**. Zum Beispiel programmiert die CPU **230** ein PORT SPEED REGISTER in den PCB-Registern **705**, das eine Bitmap ist, die die Geschwindigkeit jedes Ports definiert, die in der gezeigten Ausführung entweder 10 oder 100 MHz beträgt. Ferner wird ein PORT TYP REGISTER programmiert, das eine Bitmap ist, die den Typ von Port zwischen QC und TLAN definiert. Diese Register werden typischerweise während des Betriebs nicht geändert, können aber, wenn gewünscht, umprogrammiert werden.

[0158] Andere Register in den PCB-Registern **704** werden während des Betriebs verwendet. Zum Beispiel enthalten die PCB-Register **704** ein INTERRUPT SOURCE Register und ein POLLING SOURCE Register. Das INTERRUPT SOURCE Register enhält einen Satz von Unterbrechungsbits MCB_INT, MEM_RDY, PKT_AVAIL, BUF_AVAIL, ABORT_PKT und STAT_RDY. Die PKT_AVAIL- und BUF_AVAIL-Unterbrechungsbits entsprechen den PCB_PKT_AVAIL- und PCB_BUF_AVAIL*-Signalen. Wenigstens ein Unterbrechungssignal CPU_INR* wird an die CPU **230** geliefert, die das INTERRUPT SOURCE Register liest, um die Quelle der Unterbrechung zu bestimmen, wenn das CPU_INT* Signal geltend gemacht wird. Das MCB_INT-Unterbrechungsbit zeigt der CPU **230** an, dass in dem MCB **404** eine Unterbrechung aufgetreten ist. Das MEM_RDY-Unterbrechungsbit informiert die CPU **230**, dass die verlangten Daten im Speicher **212** in den FIFOs **430** verfügbar sind. Das PKT_AVAIL-Unterbrechungsbit informiert die CPU **230**, dass Paketdaten für die CPU **230** vorhanden sind. Das BUF_AVAIL-Unterbrechungsbit informiert die CPU **230**, dass Pufferplatz für die CPU **230** vorhanden ist, um Paketdaten zu senden. Das ABORT_PKT-Unterbrechungsbit informiert die CPU **230**, dass das ABORT_IN*-Signal geltend gemacht wurde. Das STAT_RDY-Unterbrechungssignal informiert die CPU **230**, dass die verlangte statistische Information von den QC-Vorrichtungen **202** sich in den FIFOs **430** befindet. Das POLLING SOURCE Register enthält eine Kopie jedes Unterbrechungsbits, falls die Unterbrechungen maskiert werden und der Abfragemodus verwendet wird.

[0159] Die CPU-Schnittstellenlogik **700** liefert Daten an einen 64-Byte Empfangspuffer RX BUF **710** in den FIFOs **434**, die Daten auf dem HCB-Bus **438** geltend machen. Ein Sendepuffer TX BUF **712** in den FIFOs **434** empfängt von dem HCB-Bus **438** zum Liefern der Daten an die CPU-Schnittstellenlogik **700**. Die CPU-Schnitt-

stellenlogik **700** ist mit dem RX BUF **710**, dem TX BUF **712** und der QC/HCB-Schnittstelle **436** durch Bussignale **705** verbunden. Die QC/HCB-Schnittstelle **436** ist mit der CPU-Schnittstellenlogik **700**, den RX und TX BUFs **710**, **712** und dem HCB-Bus **438** zum Steuern von Datenübertragungen zwischen dem HCB **402** und dem PCB **406** verbunden.

[0160] Fig. 7B ist ein ausführlicheres Blockschaltbild der CPU-Schnittstelle **700**. Die CPU-Steuer- und Statussignale **218B** werden durch eine Steuerlogik **713** geltend gemacht, die mit einer CPU-Tracker-Zustandsmaschine **717** und einer Alternativspeichersteuer-Zustandsmaschine **718** verbunden ist. Der Adress- und Datenabschnitt **218a** des CPU-Busses **218** ist ein gemultiplexter Bus, wo Daten von anderen Abschnitten des PCB **406** an eine Datenbus-Freigabelogik **716** zur Geltendmachung auf dem CPU-Adress- und Datenabschnitt **218a** geliefert werden. Dir CPU **230** macht Adressen an einer Adressendecodier/Anforderungs-Erzeugungslogik **714** geltend, die eine Vielzahl von Anforderungssignalen an andere Abschnitte des PCB **406** liefert, einschließlich der CPU-Tracker-Zustandsmaschine **717** und der Alternativspeichersteuer-Zustandsmaschine **718**. Ein Satz von Informations-Latches **715** emügängt Adressen und Daten von der CPU **230** und macht verriegelte Adressen und verriegelte Daten an anderen Abschnitten des PCB **406** geltend, wie unten werter beschrieben. CPU-Steuersignale werden zwischen der Adressendecodier/Anforderungs-Erzeugungslogik **714**, der CPU-Tracker-Zustandsmaschine **717** und der Alternativspeichersteuer-Zustandsmaschine **718** zum Überwachen und Steuern von CPU-Zyklen bereitgestellt.

[0161] Fig. 7C ist ein ausführlicheres Blockschaltbild der QC/CPU-Schnittstellenlogik **702**. Die QC/CPU-Schnittstellenlogik **702** arbeitet allgemein, um eine relativ transparente Schnittstelle zwischen der CPU **230** und den QC-Vorrichtungen **202** herzustellen, z. B. Umwandlung zwischen dem 32-Bit Format der CPU **230** und dem 16-Bit Format der QC-Vorrichtungen **202**. Ein QC REGISTER REQUEST Signal wird von der Adressendecodier/Anforderungs-Erzeugungslogik **714** an eine CPU-Tracker-Zustandsmaschine **720** geliefert, die mit einer Zerlegungs/Zusammensetzungs-Zustandsmaschine **722** zum Umwandeln zwischen 16-Bit und 32-Bit Formaten verbunden ist. Die Zerlegungs/Zusammensetzungs-Zustandsmaschine **722** ist mit einem Satz von Daten, Adressen- und Steuersignaltreibern und Empfängern **724** zum Verbinden mit der CPU-Schnittstelle **700** über den Bus **701** und mit den QC-Vorrichtungen **202** durch den QC/CPU-Bus **204** verbunden. Ein Statistikpuffer **726** empfängt Statistikdaten und andere Information von dem QC/CPU-Bus **204** zum Liefern der Daten an die CPU-Schnittstelle **700** über den Bus **701**. Ein STATISTICS REQUEST Signal wird von der Adressendecodier/Anforderungd-Erzeugungslogik **714** an eine Statistikanforderungs-Zustandsmaschine geliefert, die mit der Zerlegungs/Zusammensetzungs-Zustandsmaschine **722** und einer CPU-Bus-Zustandsmaschine **730** verbunden ist. Die CPU-Bus-Zustandsmaschine **730** ist werter mit der Zerlegungs/Zusammensetzungs-Zustandsmaschine **722** und dem Satz von Daten-, Adressen- und Steuersignaltreibem und Empfängern **724** verbunden. Auf diese Weise hat die CPU **230** eine relativ vollständigen und unabhängigen Zugriff auf die QC-Vorrichtungen **202** zum zu Gewinnen von Statistik und anderer Information der Ports **105** und auch zum Modifizieren der Konfiguration der Ports **104**, ohne den Datenfluss und den Betrieb des HSB **206** zu stören.

[0162] Die CPU **230** fordert den EPSM **210** auf, Statistik- und Statusinformation von den QC-Vorrichtungen **202** durch Schreiben in ein QC STATISTICS INFORMATION Register in den PCB-Registern **704** zu empfangen. Die CPU **230** fordert Statistik-Information an, durch Bereitstellen einer Nummer, die einer der QC-Vorrichtungen **202** entspricht, einer Portnummer, der Nummer des Anfangsregisters für den angegebenen Port und der Zahl für den angegebenen Port zu lesender Register. Wie Fig. 7C gezeigt, bewirkt das Schreiben in das QC STATISTICS INFORMATION Register, dass das QC STATISTICS REQUEST Signal geltend gemacht wird, wo die Statistikanforderungs-Zustandsmaschine **728** die angegebenen Anforderungen auf dem QC/CPU-Bus **204** durch den Satz von Daten-, Adressen- und Steuersignaltreibern und Empfängern **724** vornimmt. Die CPU-Schnittstelle **700** führt die gewünschten Lesezyklen auf der geeigneten QC-Vorrichtung(en) **202** unter Verwendung der geeigneten CHIP_SELECTm* Signale durch und schreibt dann die Information in den Statistikpuffer **726**.

[0163] Sobald alle angeforderten Daten zurückgewonnen und im Statistikpuffer **726** gespeichert sind, aktualisiert die CPU **230** das STAT_RDY-Bit in dem POLLING SOURCE Register in den PCB-Registern **704** und setzt das STAT_RDY-Unterbrechungsbit im INTRERUPT SOURCE Register. Der EPSM **210** macht das CPU_INT* Signal an der CPU **230** geltend, die durch Lesen des INTERRUPT SOURCE Registers **720** reagiert, um die Quelle der Unterbrechung zu bestimmen. Wenn Unterbrechungen maskiert werden, erfasst die CPU **230** das STAT_RDY-Bit im POLLING SOURCE Register während einer Abfrage-Routine. Auf diese Weise stellt die CPU **230** fest, dass die Anforderung durch entweder eine Unterbrechung oder einen Abfragemechanismus, wenn die Unterbrechungen maskiert werden, vollendet ist. Die STA_RDY-Unterbrechung wird, wenn gewünscht, programmatisch unterbrochen, wenn der Abfragemechanismus zu verwenden ist. Die CPU **230** ge-

winnt als Reaktion die ganze Statistikinformation aus dem Statistikpuffer in einem oder mehr aufeinanderfolgenden Prozessorzyklen zurück. Die Prozessorzyklen über den CPU-Bus **218** können reguläre Prozessorzyklen sein, sind aber vorzugsweise Burst-Zyklen zum Übertragen größerer Datenmengen.

[0164] Natürlich können mehrere alternative Ausführungen erwogen werden. In einer ersten alternativen Ausführung gibt die CPU **230** einfach eine Nummer aus, die einer der QC-Vorrichtungen **202** entspricht, und der EPSM **210** sammelt entsprechend alle Daten aller Register **306** aller Ports der QC-Vorrichtungen **202**. In einer zweiten alternativen Ausführung gibt die CPU **230** einfach eine globale Statistikanforderung aus, und die Daten aller Register **306** aller QC-Vorrichtungen **202** werden gesammelt. Es ist jedoch anzumerken, dass die CPU **230** typischerweise Statistikinformation für einen der Ports **104** zu einer Zeit benötigt.

[0165] Man wird einsehen, dass die CPU **230** nur eine einzige Anforderung an den EPSM **210** richten muss, um die ganze Statistikinformation für jeden der Ports zurückzugewinnen. Das heißt, das QC STATISTICS INFORMATION Register wird durch die CPU **230** in einem einzigen Befehl beschrieben, um die Anforderung vorzunehmen. Die CPU **230** ist daher frei für andere Aufgaben, antstatt auf Antworten von den QC-Vorrichtungen **202** zu warten. Stattdessen führt der EPSM **210** alle einzelnen Statistik-Leseanforderungen über den QC/CPU-Bus **204** aus und gewinnt alle Daten. Die CPU **230** wird durch ein Unterbrechungssignal oder einen Abfragemechanismus informiert und ist in der Lage, alle angeforderte Information zurückzugewinnen. Dies resultiert in einer effizienteren Nutzung der Prozessorzeit der CPU **230**.

[0166] Fig. 7D ist ein ausführlicheres Blockschaltbild der Schnittstelle zwischen der CPU-Schnittstelle **700** und dem MCB **404**. Ein Speicheranforderungssignal von der Adressendecodier/Anforderungs-Erzeugungslogik **714** wird einer Speicher-FIFO-Zugriffs-Zustandsmachine **740** zugeführt, die mit Adressenerzeugungslogik **746** und FIFO-Status- und Unterbrechungs-Erzeugungslogik **742** verbunden ist. Ein FIFO-Block **748**, der den RX BUF **706** und den TX BUF **708** enthält, ist mit der Adressenerzeugungslogik **746** und der FIFO-Status- und Unterbrechungs-Erzeugungslogik **742** verbunden. Die Adressenerzeugungslogik **746** und die FIFO-Status- und Unterbrechungs-Erzeugungslogik **742** sind beide mit einem Satz von Daten-, Adressen- und Steuersignaltreibem und Empfängern **744** zum Verbinden mit der CPU-Schnittstelle **700** über den Bus **703** und mit dem MCB **404** durch den MCB-Bus **428** verbunden.

[0167] Fig. 7E ist ein ausführlicheres Blockschaltbild der Schnittstelle zwischen der CPU-Schnittstelle **700** und dem HCB **402**. Ein Paketlese-Anforderungssignal von der Adressendecodier/Anforderungs-Erzeugungslogik **714** wird einer Sendepaket-Zustandsmaschine **760** zugeführt, die mit einem Sendepuffer **762** verbunden ist, der den TX BUF **712** enthält. Ein Paketschreib-Anforderungssignal von der Adressendecodier/Anforderungs-Erzeugungslogik **714** wird einer Empfangspaket-Zustandsmachine **770** zugeführt, die den RX BUF **710** enthält. Der Sendepuffer **762** und der Empfangspuffer **770** sind beide mit einem Satz von Daten-, Adressen- und Steuersignaltreibern und Empfängern **764** zum Verbinden mit der CPU-Schnittstelle **700** über den Bus **705** und mit dem HCB **402** durch den HCB-Bus **438** verbunden.

[0168] Fig. 8A zeigt ein vereinfachtes Blockschaltbild, das die TPI **220** ausführlicher veranschaulicht. Die TPI **220** überträgt Daten zwischen dem HSB **206** und dem PCI-Bus **222**, um Netzwerkdaten zwischen den TLANs **226** und dem EPSM **210** zu übermitteln. Die TPI **220** arbeitet als ein Slave auf dem HSB **206**, antwortet auf Abfragen des EPSM **210** und überträgt Daten an den und von dem EPSM **210** in einer ähnlichen Weise wie die QC-Vorrichtungen **202**. Auf der Seite des PCI-Busses **222** überträgt die TPI **220** Netzwerkdaten an jedes der vier TLANs **226** (Port24, Port25, Port26 und Port27) über den PCI-Bus **222** und empfängt Netzwerkdaten von denselben.

[0169] Die TPI **220** umfasst eine HSB-Steuerung **804**, eine PCI-Bussteuerung **802** und einen Speicher **806**. Die PCI-Bussteuerung **802** verbindet den PCI-Bus **222** entsprechend den PCI-Standards und ermöglicht Datenübertragungen zwischen der TPI **220** und dem PCI-Bus **222**. Die PCI-Standards sind von dem 'Inter Architecture Lab' zusammen mit seinen Industriepartnem definiert. Die HSB-Steuerung **804** verbindet den HSB **206** entsprechend der definierten Operation des HSB **206** und ermöglicht Datenübertragungen zwischen der TPI **220** und dem EPSM **210**. Der Speicher **806** kann zentralisiert oder verteilt sein und umfasst eine Vielzahl von Datenpuffern **807** und einen Steuerlistenspeicher **808**. Die Datenpuffer **807** liefern vorübergehende Speicherung, um die Datenübertragung zwischen dem PCI-Bus **222** und dem HSB **206** zu erleichtern. Der Steuerlistenspeicher **808** ermöglicht den Busmasterbetrieb der TLANs auf dem PCI-Bus **222**.

[0170] Fig. 8B zeigt ein ausführlicheres Bockschaltbild der TPI **220**. Die TPI **220** enthält eine PCI-Bus-Schnittstellenlogik **810**, die weiter Puffer, Treiber und verwandte Schaltungen enthält, um den PCI-Bus **222** schnittstellenmäßig zu verbinden. Der PCI-Bus **222** der vorliegenden Ausführung hat eine Daten-

breite von 32 Bits und arbeitet bei einer Takfrequenz von 33 MHz. Es versteht sich jedoch, dass der PCI-Bus eine andere Datenbreite haben kann und bei jeder gewünschten oder verfügbaren Taktfrequenz, wie z. B. 66 MHz, arbeiten kann. Die TPI **220** enthält einen PCI-Arbiter **811**, der zwischen jedem der TLANs **226**, der TPI **220** und der CPU **230** für Zugriff und Steuerung des PCI-Busses **222** schlichtet. Das heißt, jedes der TLANs **226**, die TPI **220** und die CPU **230** machen ein betreffendes von mehreren Anforderungssignalen REQm geltend, um die Steuerung des PCI-Busses **222** zu verlangen, wo die REQm Signale durch den PCI-Arbiter **811** empfangen werden. Als Reaktion gewährt der PCI-Arbiter **811** einer der anfordernden Vorrichtungen die Steuerung, indem er ein betreffendes Gewährungssignal GNTm geltend macht. Der PCI-Arbiter **811** führt in der gezeigten Ausführung eine Umlauf-Schlichtung durch, obwohl der PCI-Arbiter **811** jedes andere gewünschte Arbitrationsschema verwenden kann. Der PCI-Arbiter **811** macht TLAN-Auswählsignale (TSELm) geltend, um ein bestimmtes TLAN **226** zu identifizieren, nachdem ihm die Steuerung des PCI-Busses **222** gewährt wurde.

[0171] Die TPI **220** enthält eine HSB-Datenübertragungs-Schnittstellenlogik **819**, die Puffer, Treiber und zugehörige Schaltungen enthält, um die TPI **220** schnittstellenmäßig mit dem HSB **206** zu verbinden. Die HSB-Datenübertragungs-Schnittstellenlogik **819** enthält Lese-Latches **819a** und Schreib-Latches **819b** zum Durchführen von gleichzeitigen Lese- und Schreibzyklen auf dem HSB **206**. Die HSB-Datenübertragungs-Schnittstellenlogik **819** enthält eine Portstatuslogik **820** zum Antworten auf EPSM **210** Abfragen und zum Überwachen der auf dem HSB **206** ausgeführten Zyklen. Das heißt, die Portstatuslogik **820** empfängt und erfasst Gelmachungen des STROBE* Signals durch den EPSM **210** und antwortet durch Geltendmachen der PKT_AVAIL[6]* und BUF_AVAIL[6]* Signale in einer gemultiplexten Weise basierend auf dem Datenstatus der TPI **220**. Die Portstatuslogik **820** erfasst auch Lese- und Schreibzyklen auf dem HSB **206**, die für die TPI **220** bestimmt sind, durch Erfassen der READ_OUT_PKT[6]* und WRITE_IN_PKT[6]* Signale. Während Übertragungen von Paketdaten von der TPI **220** an den EPSM **210** über den HSB **206** macht die Portstatuslogik **820** die SOP* und EOP* Signale während des HSB **206** Buszyklusses geltend, wenn der Anfang oder das Ende eines Pakets übertragen wird. Während Übertragungen von Paketdaten von dem EPSM **210** an die TPI **220** über den HSB **206** liest die Portstatuslogik **820** die Signale SOP* und EOP*, um festzustellen, ob die empfangenen Daten der Anfang eines Pakets oder das Ende eines Pakets sind.

[0172] Die Datenpuffer **807** umfassen mehrere bidirektionale FIFO-Datenpuffer **807a**, **807b**, **807c** und **807d** (**807a**–**d**), die jeweils sowohl einen 32 Bit breiten Sendepuffer (TPI TX FOFO) als auch einen 32 Bit breiten Empfangspuffer (TPI RX FIFO) enthalten. In der gezeigten Ausführung entsprechen die Datenpuffer **807a**, **807b**, **807c** und **807d** den Ports Port24, Port25, Port26 bzw. Port27. Jeder TPI RX FIFO empfängt Daten von einem betreffenden TLAN **226** über den PCI-Bus **222**, wo die Daten von der TPI **220** an den EPSM **210** über den HSB **206** gesendet werden. Jeder TPI TX FIFO empfängt Daten von dem EPSM **210** über den HSB **206**, wobei die Daten von der TPI **220** an ein betreffendes TLAN **226** über den PCI-Bus **222** gesendet werden.

[0173] Empfangslisten-Decodierlogik **812** ist mit der PCI-Bus-Schnittstellenlogik **810** verbunden und speichert wenigstens eine Empfangssteuerliste in einem Empfangssteuelistenspeicher (RX CNTL LIST) **808a**, der Teil des Steuerlistenspeichers **808** ist. Die Empfangslisten-Decodierlogik **812** reagiert auf die Geltendmachung einer RECEIVE LIST MEMORY ADDRESS, die als eine Adresse auf dem PCI-Bus **222** geltend gemacht ist, durch Schreiben einer Empfangssteuerliste aus der RX CNTL LIST **808a** als Daten in den PCI-Bus **222**. In der gezeigten Ausführung hält die RX CNTL LIST **808a** eine Empfangssteuerliste zu einer Zeit. Das heißt, jeder TLAN **226** erlangt die Steuerung des PCI-Busses **222** und macht die RECEIVE LIST MEMORY ADDRESS auf dem PCI-Bus **222** geltend und empfängt eine entsprechende Empfangssteuerliste von der RX CNTL LIST **808a**. Die Empfangssteuerliste enthält eine PACKET DATA MEMORY BASE ADDRESS zur Verwendung durch das TLAN **226**, die eine Adresse ist, die angibt, wo die empfangenen Daten zu speichern sind. Als Reaktion auf den Empfang eines Datenpakets von seinem betreffenden Port **110** erlangt das TLAN **226** die Steuerung des PCI-Busses **222** zurück, um Daten aus dem empfangenen Datenpaket unter Verwendung der gespeicherten Adresse in der vorher geholten Empfangssteuerliste an die TPI **220** zu übertragen. Wie weiter unten beschrieben, schlichtet das TLAN **226** und erhält die Steuerung des PCI-Busses **222** und macht die PACKET DATA MEMORY ADDRESS während eines Schreibzyklusses auf dem PCI-Bus **222** geltend.

[0174] Eine Empfangsdaten-Decodierlogik **813**, eine PCI RX FIFO Steuerlogik **817**, der PCI-Arbiter **811** und eine FIFO-Synchronisationslogik **818** steuern den Fluss von empfangenen Daten von der PCI-Bus-Schnittstellenlogik **810** in den entsprechenden TPI TX FIFO. Die PCI RX FIFO Steuerlogik **817** enthält einen Eingang, um Daten von der PCI-Bus-Schnittstellenlogik **810** zu empfangen, und mehrere wählbare Ausgänge, die jeweils mit dem Eingang eines entsprechenden TPI RX FIFO verbunden sind. Der PCI-Arbiter **811** liefert die TSELm Signale an die FIFO-Synchronisationslogik **818**, die entsprechende PCI-Pufferauswählsignale (PBSELm) an der PCI RX FIFO Steuerlogik **817** geltend macht, um den geeigneten TPI RX FIFO basierend auf dem einzelnen TLAN **226**, dem die Steuerung des PCI-Busses **222** gewährt wurde, auszuwählen. Die Emp-

fangsdaten-Decodierlogik **813** empfängt und decodiert die PACKET DATA MEMORY ADDRESS, die durch Ausführen eines Schreibzyklusses auf dem PCI-Bus **222** durch das TLAN **226** geltend gemacht wurde, und macht als Reaktion ein Empfangsfreigabesignal (REN) an der PCI RX FIFO Steuerlogik **817** geltend, um der PCI RX FIFO Steuerlogik **817** zu ermöglichen, Daten zu dem ausgewählten TPI RX FIFO zu leiten.

[0175] Es wird angemerkt, dass ein bidirektionaler Datenfluss zwischen dem PCI-Bus **222** und dem HSB **206** durch die Datenpuffer **807** stattfindet. Der PCI-Bus **222** und der HSB **206** arbeiten in einer Ausführung bei der gleichen Geschwindigkeit, z. B. bei einem 33 MHz Takt, können aber in alternativen Ausführungen bei verschiedenen Taktfrequenzen arbeiten. Zum Beispiel arbeitet in einer anderen Ausführung der HSB **206** bei 33 MHz, während der PCI-Bus **222** bei 66 MHz arbeitet. Die TPI **220** ist so implementiert, dass sie den Datenfluss trotz der unterschiedlichen Taktgeschwindigkeiten handhaben und synchronieren kann. Jeder TPI RX FIFO und TPI TX FIFO der Datenpuffer **807a–d** wird vorzugsweise als ein Ringpuffer implementiert, wobei Zeiger auf beiten Seiten zum Schreiben und Lesen von Daten unterhalten werden. Die FIFO-Synchronisationslogik **818** abeitet allgemein, um die Zeiger auf beiden Seiten jedes FIFO zu synchronisieren, zu unterhalten und zu aktualisieren, um sicherzustellen, dass Daten richtig in den geeigneten TPI FIFO geschrieben oder daraus gelesen werden.

[0176] Wie oben erwähnt, ist jeder TPI RX FIFO als ein Ringpuffer implementiert. Die PCI RX FIFO Steuerlogik **817** enthält mehrere PCI-Empfangszeiger (PCI RX PTRs), einen Zeiger für jeden TPI RX FIFO, um auf die nächste Stelle zu zeigen oder sie zu adressieren, um ein DWORD (32 Bit) in dem ausgewählten TPI RX FIFO zu empfangen. In ähnlicher Weise enthält die HSB RX FIFO Steuerlogik **821**, die sich auf der anderen Seite jedes TPI RX FIFO befindet, mehrere "synchronisierte" PCI Empfangszeiger (PCI RX SPTRs), von denen jeder eine synchronisierte Kopie eines entsprechenden PCI RX PTR ist. Zusammen mit den PBSELm Signalen, um den geeigneten TPI RX FIFO auszuwählen, macht die FIFO-Synchronisationslogik **818** auch ein entsprechendes einer Vielzahl von PCI-Zählsignalen (PCNTm) geltend, um den geeigneten PCI RX PTR in der PCI RX FIFO Steuerlogik **817** synchron zu aktualisieren oder zu inkrementieren. Die FIFO-Synchronisationslogik **818** macht werter ein entsprechendes einer Vielzahl von HSB-Zählsignalen (HCNTm) geltend, um einen entsprechenden PCI RX SPTR in der HSB RX FIFO-Steuerlogik **821** synchron zu aktualisieren oder zu inkrementieren. Auf diese Weise wird ein Zeiger auf beiden Seiten jedes TPI RX FIFO bereitgestellt, um anzuzeigen, wo Daten einzufügen sind.

[0177] Ein PCI TX FIFO Steuerlogik **816** erfasst Daten in jedem der TPI TX FIFOs und veranlasst die TPI **220**, den PCI-Bus **222** aufzufordern und die Steuerung desselben zu erlangen, einen Befehl an ein TLAN **226** entsprechend dem TPI TX FIFO, der Daten zum Senden hat, zu senden. Die PCI TX FIFO Steuerlogik **816** greift auf die Adressen des geeigneten TLAN **226** aus einem Satz von TPI-Steuerregistern **846** zu. Die TPI **220** schreibt einen Befehl in das geeignete TLAN **226** und stellt eine TRANSMIT LIST MEMORY BASE ADDRESS bereit, um das TLAN **226** zu veranlassen, anschließend eine Sendesteuerliste von der TPT **220** unter Verwendung der TRANSMIT LIST MEMORY BASE ADDRESS anzufordern.

[0178] Eine Sendelisten-Decodierlogik **814** ist mit der PCI-Bus-Schnittstellenlogik **810** verbunden und speichert wenigstens eine Sendesteuerliste in einem Sendesteuerlistenspeicher (TX CNTL LIST) **808b**, der Teil des Steuerlistenspeichers **808** ist. Die Sendelisten-Decodierlogik **814** reagiert die auf Geltendmachung der als eine Adresse auf dem PCI-Bus **222** geltend gemachten TRANSMIT LIST MEMORY BASE ADDRESS durch Schreiben einer Sendesteuerliste aus der TX CNTL LIST **808b** als Daten in den PCI-Bus **222**. In der gezeigten Ausführung hält die TX CNTL LIST **808b** eine Sendesteuerliste zu einer Zeit. Auf diese Weise erlangt jedes TLAN **226** die Steuerung des PCI-Busses **222** und macht die TRANSMIT LIST MEMORY BASE ADDRESS auf dem PCI-Bus **222** geltend und empfängt eine entsprechende Sendesteuerliste von der TX CNTL LIST **808b**. Nach Rückgewinnung der Sendesteuerliste führt das TLAN **226** die Sendesteuerung durch, indem es den PCI-Bus **222** auffordert und die Steuerung desselben erlangt, einen Lesezyklus durchzuführen, um die Daten aus dem entsprechenden TPI TX FIFO der TPI **220** mittels der TRANSMIT LIST MEMORY BASE ADDRESS zurückzugewinnen.

[0179] Eine Sendedaten-Decodierlogik **815**, die PCI TX FIFO Steuerlogik **816**, der PCI-Arbiter **811** und die FIFO-Synchronisationslogik **818** steuern den Fluss von Daten von jedem TPI TX FIFOs der Datenpuffer **807** auf den PCI-Bus **222**. Die PCI TX FIFO Steuerlogik **816** enthält einen Ausgang, um Daten an die PCI-Bus-Schnittstellenlogik **810** zu liefern, und mehrere Eingänge, die jeweils mit einem Ausgang eines entsprechenden der TPI TX FIFOS verbunden sind. Wenn ein TLAN **226** einen Lesezyklus auf dem PCI-Bus **222** durchführt, um Daten zu lesen, liefert der PCI-Arbiter **811** die TESLm Signale an die FIFO-Synchronisationslogik **818**, die die PBSELm Signale an der PCI TX FIFO Steuerlogik **816** geltend macht, um den entsprechenden TPI TX FIFO basierend auf dem einzelnen TLAN **226**, das die Steuerung des PCI-Busses **222** hat, aus-

zuwählen. Die Sendedaten-Decodierlogik **815** empfängt und decodiert die durch das TLAN **226** geltend gemachte PACKET DATA MEMORY BASE ADDRESS und macht als Reaktion ein Freigabesignal TEN an der PCI TX FIFO Steuerlogik **816** geltend, um die Übertragung von Daten an den ausgewählten TPI TX FIFO zu ermöglichen. Es wird angemerkt, dass die PBSELm Signale sowohl an die PCI RX FIFO Steuerlogik **817** als auch an die PCI TX FIFO Steuerlogik **816** angelegt werden, und dass die Signale TEN und REN zwischen der PCI RX FIFO Steuerlogik **817** und der PCI TX FIFO Steuerlogik **816** abhängig von dem Typ des Zyklusses und der Richtung des Datenflusses auswählen.

[0180] Jeder TPI TX FIFO ist in der gezeigten Ausführung als ein Ringpuffer implementiert. Die PCI TX FIFO Steuerlogik **816** enthält mehrere PCI-Sendezeiger (PCI TX PTRs), einen Zeiger für jeden TPI TX FIFO, um auf die nächste Stelle zu zeigen oder sie zu adressieren, von der ein DWORD von Daten zu lesen ist. In ähnlicher Weise enthält die HSB TX FIFO Steuerlogik **822**, weiter unten beschrieben, die sich auf der anderen Seite jedes TPI TX FIFO befindet, mehrere "synchronisierte" PCI Sendezeiger (PCI TX SPTRs), von denen jeder eine synchronisierte Kopie eines entsprechenden PCI TX PTR ist. Die FIFO-Synchronisationslogik **818** macht ein entsprechendes der PCNTm Signale geltend, um den geeigneten PCI TX PTR zu inkrementieren, und ein entsprechendes der HCNTm Signale, um den geeigneten PCI TX SPTR jedes Mal zu inkrementieren, wenn ein DWORD von Daten von der PCI TX FIFO Steuerlogik **816** an den PCI-Bus **222** angelegt wird. Auf diese Weise wird ein Zeiger auf beiden Seiten jedes TPI TX FIFO bereitgestellt, um anzuzeigen, wo Daten zu lesen sind.

[0181] Die HSB RX FIFO Steuerlogik **821** hat mehrere wählbare Eingänge, die jeweils mit einem Ausgang eines entsprechenden der TPI RX FIFOs verbunden sind. Die HSB RX FIFO Steuerlogik **821** hat einen Ausgang zum Liefern der Daten an die HSB-Datenübertragungs-Schnittstellenlogik **819** zur Geltendmachung auf dem HSB **206**. Die HSB TX FIFO Steuerlogik **822** hat mehrere wählbare Ausgänge, die jeweils mit einem Eingang eines entsprechenden der TPI TX FIFOS verbunden sind. Die HSB TX FIFO Steuerlogik **822** hat einen Eingang zum Empfangen von Daten der HSB-Datenübertragungs-Schnittstellenlogik **819** von dem HSB **206**.

[0182] Die HSB RX FIFO Steuerlogik **821**, die Portstatuslogik **820** und die FIFO-Synchronisationslogik **818** steuern den Datenfluss zwischen den TPI RX FIFOs und den Datenpuffern **807a–d** und dem HSB **206** während der Datenübertragungen von der TPI **220** an den EPSM **210**. Die Portstatuslogik **820** erfasst die Geltendmachung des READ_OUT_PKT[6]* Signals, das einen Lesezyklus auf dem HSB **206** anzeigt, und decodiert die PORT_NO[1:0] Signale, um den entsprechenden TPI TX FIFO des gewählten Ports zu identifizieren. Das heißt, der EPSM **210** macht PORT_NO[1:0] Signale 00, 01, 10 oder 11 geltend, um den TPI RX FIFO eines der Datenpuffer **807a**, **807b**, **807c** oder **807d** für den Port Port24, PORT25, PORT26 oder PORT27 auszuwählen. Die Portstatuslogik **802** macht Portauswählsignale (PSELm) an der FIFO-Synchronisationslogik **818** geltend, die als Reaktion entsprechende HSB-Auswählsignale (HBSELm) geltend macht, um einen Ausgang der mit dem entsprechenden TPI RX FIFO verbundenen HSB RX FIFO Steuerlogik **821** auszuwählen. Ferner macht die Portstatuslogik **820** ein HSB-Freigabesignal (HREN) geltend, um der HSB RX FIFO Steuerlogik **821** zu ermöglichen, die Daten an die HSB-Datenübertragungs-Schnittstellenlogik **819** zur Geltendmachung auf dem HSB **206** auszugeben.

[0183] Die HSB RX FIFO Steuerlogik **821** enthält einen HSB-Empfangszeiger (HSB RX PTR) für jeden TPI RX FIFO, um die einzelnen Daten in dem TPI RX FIFO zu lokalisieren. Die FIFO-Synchronisationslogik **818** macht ein entsprechendes der HCNTm Signale geltend, um den entsprechenden HSB RX PTR des ausgewählten TPI RX FIFO für jedes aus dem TPI RX FIFO gelesene DWORD zu aktualisieren oder zu dekrementieren. Die PCI RX FIFO Steuerlogik **817** enthält auch einen entsprechenden "synchronisierten" HSB-Empfangszeiger (HSB RX SPTR), der durch die FIFO-Synchronisationslogik **818** durch Geltendmachen eines entsprechenden der PCNTm Signale dekrementiert wird. Auf diese Weise hat die HSB RX FIFO Steuerlogik **821** zwei Zeiger für jeden TPI RX FIFO, einschließlich des PCI RX SPTR, der anzeigt, wo Daten zu schreiben sind, und des HSB RX PTR, der anzeigt, wo Daten zu lesen sind. Die Portstatuslogik **820** greift auch auf diese Zeiger zu, um die Menge an gültigen Daten oder die Zahl gültiger Datenbytes in jedem TPI RX FIFO zu gewinnen. Diese Anzahl wird mit einer entsprechenden RBSIZE (entsprechend dem TBUS-Wert) für dem HSB **206** verglichen, um zu bestimmen, wie die PKT_AVAIL[6]* Signale als Reaktion auf das STROBE* Signal geltend zu machen sind.

[0184] Die HSB TX FIFO Steuerlogik **822**, die Portstatuslogik **820** und die FIFO-Synchronisationslogik **818** steuern den Datenfluss zwischen jedem TPI TX FIFO und dem HSB **206** während Datenübertragungen von der EPSM **210** an die TPI **220**. Die Portstatuslogik **820** erfasst die Geltendmachung des WRITE_IN_PKT[6]* Signals und bestimmt die Portnummer aus den PORT_NO[0:1] Signalen während eines von dem EPSM **210** auf dem HSB **206** ausgeführten Schreibzyklusses. Die Portstatuslogik **820** macht entsprechend die PSELm Signale und ein HSB-Sendefreigabesignal (HTEN) geltend, um den geeigneten TPI TX FIFO zu bezeichnen.

Die FIFO-Synchronisationslogik **818** macht als Reaktion die HBSELm Signale geltend, um den entsprechenden Eingang der HSB TX FIFO Steuerlogik **822** zu dem geeigneten TPI TX FIFO auszuwählen. Das HTEN-Signal ermöglicht der HSB TX FIFO Steuerlogik **822**, die Daten von der HSB-Datenübertragungs-Schnittstellenlogik **819** zur Geltendmachung an dem ausgewählten TPI TX FIFO zu empfangen.

[0185] Die HSB TX FIFO Steuerlogik **822** enthält einen HSB-Sendezeiger (HSB TX PTR) für jeden TPI TX FIFO, um die einzelne Datenstelle in dem TPI TX FIFO aufzufinden, um Daten zu schreiben. Die FIFO-Synchronisationslogik **818** macht ein entsprechendes der HCNTm Signale geltend, um den entsprechenden HSB TX PTR des gewählten TPI TX FIFO für jedes in den gewählten TPI TX FIFO geschriebene DWORD zu aktualisieren oder zu inkrementieren. Ferner enthält die PCI TX FIFO Steuerlogik **816** einen entsprechenden "synchronisierten" HSB-Sendezeiger (HSB TX SPTR), der durch die FIFO-Synchronisationslogik **818** durch Geltendmachen eines der PCNTm Signale inkrementiert wird. Auf diese Weise hat die HSB TX FIFO Steuerlogik **822** zwei Zeiger für jeden TPI TX FIFO, einschließlich des PCI TX SPTR, der anzeigt, wo Daten zu lesen sind, und des HSB TX PTR, der anzeigt, wo Daten zu schreiben sind. Die Portstatuslogik **820** greift auch auf diese Zeiger zu, um die Menge an verfügbarem Platz oder die Zahl in jedem TPI TX FIFO vorhandener leerer Datenbytes zu gewinnen. Dieser Wert wird mit einer entsprechenden XBSIZE (entsprechend den TBUS-Wert) für den HSB **206** verglichen, um zu bestimmen, wie die BUF_AVAIL[6]* Signale als Reaktion auf das STROBE* Signal geltend zu machen sind.

[0186] Ein Satz von TPI PCI Konfigurationsregistern **835** ist in der TPI **220** bereitgestellt und mit der PCI-Bus-Schnittstellenlogik **810** zum Zugriff über den PCI-Bus **222** verbunden. Ferner sind die TPI-Steuerregister **846** bereitgestellt und mit verschiedenen Vorrichtungen in der TPI **220** und der PCI-Bus-Schnittstellenlogik **810** zum Zugriff über den PCI-Bus **222** verbunden. Der Inhalt und die Struktur dieser Register **846** und **835** werden unten werter beschrieben. Die HSB-Datenübertragungs-Schnittstellenlogik **819** enthält auch ein PACKET SIZE Anhängerregister **819c**. Die HSB-Datenübertragungs-Schnittstellelogik **819** erfasst und speichert das erste DWORD jedes von dem EPSM **210** gesendeten Datenpakets in dem PACKET SIZE Anhängerregister **819c** und schreibt dann den Inhalt des PACKET SIZE Registers **819c** in die TX CNTL LIST **808b** der Sendelisten-Decodierlogik **814**.

[0187] Fig. 8C ist ein Blockschaltbild, das die Konfiguration und Funktionalität jedes der TLANs **226** veranschaulicht. Das TLAN **226** enthält einen Ethernet-Port **110**, eine PCI-Busschnittstelle **824** und einen zwischen den Ethernet-Port **110** und die PCI-Busschnittstelle **824** geschalteten Speicher **825**. Der Ethernet-Port **110** enthält eine geeignete Buchse, um einen kompatiblen Stecker eines 100 Mb Ethernet-Segments **114** aufzunehmen, um Paketdaten von einem entsprechenden Netzwerk **112** zu empfangen und Paketdaten an dasselbe zu senden. Der Ethernet-Port **110** liefert empfangene Paketdaten an Datenpuffer **826** im Speicher **825**. Der Ethernet-Port **110** gewinnt Paketdaten aus den Datenpuffern **826** zurück und sendet die Paketdaten an ein Ethernet-Segment **114**.

[0188] Das TLAN **226** enthält einen Satz von Registern **828** in dem Speicher **825** zum Steuern seines Betriebs. Die Register **828** umfassen ein Befehlsregister **828a** zum Ermöglichen einer externen Vorrichtung, Befehle durch den PCI-Bus **222** einzuführen. Die Register **828** umfassen werter ein Kanalparameterregister **828b** zum Speichern einer Adresse, um auf eine Befehlsliste von einem externen Speicher durch den PCI-Bus **222** zuzugreifen. Das Befehlsregister **828a** enthält ein GO-Bit (nicht gezeigt), das das TLAN **226** anweist, eine Befehlsliste zurückzugewinnen und auszuführen. Das Befehlsregister **828a** enthält auch ein RX/TX-Bit (nicht gezeigt), das das TLAN **226** anweist, eine Empfangsbefehlsliste (für den RX-Fall) oder eine Sendebefehlsliste (für den TX-Fall) zurückzugewinnen und auszuführen. Der Speicher **825** enthält einen Listenpuffer **827** zum Speichern gegenwärtiger Steuerlisten, wo der Listenpuffer **827** weiter einen Empfangssteuerlistenpuffer **827a** zum Speichern der gegenwärtigen Empfangssteuerliste und einen Sendesteuerlistenpuffer **827b** zum Speichern der gegenwärtigen Sendesteuerliste enthält.

[0189] Die PCI-Busschnittstelle **824** enthält geeignete Logik zum Verbinden mit dem PCI-Bus **222**, um Datenübertragungen zwischen der TPI **220** und dem TLAN **226** durch Arbeiten als ein Busmaster des PCI-Busses **222** während der Datenübetragung zu steuern. Eine externe Vorrichtung, z. B. die TPI **220** oder die CPU **230**, schreibt eine Adresse in das Kanalparameterregister **828b** und schreibt einen Befehl in das Befehlsregister **828a**. Das TLAN **226** macht als Reaktion ein REQm Signal geltend, um für den PCI-Bus **222** zu schlichten. Wenn ein GNTm Signal empfangen wird, führt das TLAN **226** einen Zyklus auf dem PCI-Bus **222** aus, um eine angegebene Befehlsliste zurückzugewinnen und im Listenpuffer **827** zu speichern. Der Befehl wird als ein Sendebefehl angesehen, wenn das RX/TX-Bit für TX gesetzt ist, und als Empfangsbefehl, wenn ds RX/TX-Bit für RX gesetzt ist.

[0190] Um Empfangsvorgänge einzuleiten, schreibt die CPU **230** die RECEIVE LIST MEMORY BASE ADDRESS in das Kanalparameterregister **828b** und einen Empfangsbefehl in das Befehlsregister **828a** jedes TLAN **226**. Das TLAN **226** fordert als Reaktion den PCI-Bus **222** auf, eine Empfangssteuerliste mittels der RECEIVE LIST MEMORY BASE ADDRESS zurückzugewinnen. Die TPI **220** liefert eine Empfangssteuerliste an das TLAN **226**, und das TLAN **226** wartet dann, um Daten zu empfangen, bevor die Empfangssteuerliste ausgeführt wird. Die Empfangssteuerliste enthält einen Vorwärtszeiger als die nächste Adresse für das TLAN **226**, die es benutzt, um die nächste Empfangssteuerliste zurückzugewinnen, um eine Steuerlistenverkettung zu errichten. In der bevorzugten Ausführung lädt jedoch die TPI **220** den Vorwärtszeiger jeder Empfangssteuerliste mit der gleichen RECEIVE LIST MEMORY BASE ADDRESS. Wenn Daten von dem Port **110** an die TPI **220** empfangen werden, schlichtet die PCI-Busschnittstelle **824** und erlangt die Steuerung des PCI Busses **222** und führt die Empfangssteuerliste in ihrem Empfangssteuerlistenpuffer **827a** aus, um Daten über den PCI-Bus **222** an die TPI **220** zu übertragen. Sobald die Übertragung eines ganzen Datenpakets vollendet ist, verwendet das TLAN **226** die RECEIVE LIST MEMORY BASE ADDRESS in dem Vorwärtszeigter der momentanen Empfangssteuerliste, um eine wertere Empfangssteuerliste anzufordern.

[0191] Für Sendevorgänge erfasst die TPI **220** zu sendende Daten von einem ihrer TPI TX FIFOs und schlichtet und erlangt als Reaktion die Steuerung des PCI-Busses **222**. Die TPI **220** schreibt dann die TRANSMIT LIST MEMORY BASE ADDRESS in das Kanalparameterregister **828b** und einen Sendebefehl in das Befehlsregister **828a** jedes TLAN **226**. Das TLAN **226** fordert als Reaktion den PCI-Bus **222** auf, eine Sendesteuerliste mittels der TRANSMIT LIST MEMORY BASE ADDRESS zurückzugewinnen. Sobald die Sendesteuerliste empfangen ist, speichert das TLAN **226** die Sendesteuerliste in seinem Sendesteuerlistenpuffer **828b** und führt dann die gespeicherte Sendesteuerliste aus, um Paketdaten zu empfangen. Die Sendesteuerliste enthält auch einen Vorwärtszeiger, der normalerweise als die nächste Adresse für das TLAN **226** benutzt wird, um die nächste Sendesteuerliste zurückzugewinnen, um eine Steuerlistenverkettung zu errichten. In der gezeigten Ausführung lädt jedoch die TPI **220** den Vorwärtszeiger jeder Sendesteuerliste mit einem Nullwert. Nach Ausführen der Sendesteuerliste in seinem Sendesteuerlistenpuffer **827a** wartet daher das TLAN **226**, bis die TPI **220** einen werteren Sendebefehl schreibt.

[0192] Fig. 8D zeigt ein Diagramm, das eine Steuersignalliste **830** veranschaulicht, die das Format für Sende- und Empfangssteuerlisten ist und auch das Format für die RX CNTL LIST **808a** und die TX CNTL LIST **808b** ist. Die Steuerliste **830** enthält ein FORWARD_POINTER Feld **831**, ein PACKET_SIZE Feld **832a**, ein CSTAT Feld **832b**, ein COUNT Feld **833** und ein DATA_POINTER Feld **834**. Jedes Feld ist 32 Bits mit Ausnahme des PACKET_SIZE Feldes **832a** und des CSTAT Feldes **832b**, die 16-Bit Felder sind.

[0193] Das FORWARD_POINTER Feld **832** wird gewöhnlich benutzt, um Steuerlisten miteinander zu verketten. Für Empfangsvorgänge führt das TLAN **226** durch die TPI **220** von der RX CNTL LIST **808a** bereitgestellte Steuerlisten wieder und wieder aus, da das FORWARD_POINTER Feld **831** in jedem Fall die gleiche RECEIVE LIST MEMORY BASE ADDRESS ist. Auf diese Weise verwendet jedes TLAN **226** die RECEIVE LIST MEMORY BASE ADDRESS in dem FORWARD_POINTER Feld **831** seiner momentanen Empfangssteuerliste, um die nächste Empfangssteuerliste anzufordern, wenn das nächste Datenpaket von einem Netzwerk **112** empfangen wird. Die TPI **220** muss daher für Empfangsvorgänge keine Vorgangsstartbefehle an die TLANS **226** ausgeben. Für Sendevorgänge führt das TLAN **226** jedes Mal Sendesteuerlisten aus der gleichen TX CNTL LIST **808b** aus. Die TPI **220** setzt jedoch das FORWARD_POINTER Feld **831** auf einen Nullwert (0000h), sodass die TPI **220** und ein betreffendes TLAN **226** einen Sendevorgang ausführen, wenn durch die TPI **220** eingeleitet. Wenn Daten in einem der TPI TX FIFOs erfasst werden und die TPI **220** gegenwärtig keine Sendevorgänge auf dem betreffenden TLAN-Port eines TPI TX FIFOs durchführt, gibt die TPI **220** einen Sendebefehl an ein betreffendes TLAN **226** aus, um einen Sendevorgang einzuleiten. Das betreffende TLAN **226** gewinnt die Sendesteuerliste aus der TX CNTL LIST **808b** zurück, führt die Sendesteuerliste aus und kehrt dann in einen Vorgabezustand zurück, wenn der Nullwert in dem FORWARD_POINTER Feld **831** angetroffen wird.

[0194] Das PACKET_SIZE Feld **832a** gibt gewöhnlich die Größe eines Datenpakets an. Für Empfangsvorgänge setzt die TPI **220** anfangs das PACKET_SIZE Feld **832a** in der RX CNTL LIST **808a** auf null. Nachdem das TLAN **226** eine Übertragung eines vollständigen Datenpakets an die TPI **220** vollendet hat, führt das TLAN **226** einen letzten Ein-DWORD-Schreibvorgang in das PACKET_SIZE Feld **832a** und das CSTAT Feld **832b** der RX CNTL LIST **808a** durch. Das PACKET_SIZE Feld **832a** wird mit der tatsächlichen Paketdatengröße geladen, und ein Rahmen-Vollendet-Bit in dem CSTAT Feld **832b** wird gesetzt. Für Sendevorgänge wird das PACKET_SIZE Feld **832a** der TX CNTL LIST **808b** mit der Größe eines von der TPI **220** an ein TLAN **226** zu sendenden Datenpakets geladen. Die HSB-Datenübertragungs-Schnittstellenlogik **819** schreibt die Paketgroße DWORD im PACKET_SIZE Registeranhänger **819c** in die TX CNTL LIST **808b** in der Sendelisten-Deco-

dierlogik **814**. Die TPI **220** schreibt dann den Sendebefehl in das entsprechende TLAN **226**, wie vorher beschrieben, und der Inhalt der TX CNTL LIST **808b** wird an ein TLAN **226** als eine Sendesteuerliste, wenn verlangt, geliefert.

[0195] Das CSTAT Feld **832b** wird verwendet, um Befehls- und Statusinformation zwischen der TPI **220** und den TLANs **226** zu übermitteln. Die TPI **220** setzt anfangs das CSTAT Feld **832b** der CNTL LIST **808a** auf null. Wenn eine Paketdatenübertragung von einem TLAN **226** in einen betreffenden TPI RX FIFO vollendet wurde, setzt die TPI **220** das Rahmen-Vollendet-Bit des CSTAT Feldes **832b** (Bit 14) in der RX CNTL LIST **808a**, um darzustellen, dass die Paketdatenübertragung vollendet wurde. Die TPI **220** informiert die Portstatuslogik **820**, dass das Paket vollständig ist, um eine Übertragung über den HSB **206** an den EPSM **210** einzuleiten. Die Portstatuslogik **820** zeigt dann an, dass Daten in einem betreffenden TPI RX FIFO zur Übertragung an den EPSM **210** als Reaktion auf eine Abfrage durch den EPSM **210** verfügbar sind. Dies gilt, selbst wenn die Menge an Paketendedaten nicht den RBSIZE- oder TBUS-Wert erfüllt, da das Ende des Pakets übertragen werden muss.

[0196] Die TPI **220** setzt das CRC-(zyklische Redundanzprüfung) Bit in dem CSTAT Feld **832b** der TX CNTL LIST **808b** basierend auf dem Sataus des AI_FCS_IN* (oder FBPN) Signals während des Empfangs eines Datenpakets von dem EPSM **210**. Die TPI **220** setzt das CRC-Bit, um anzuzeigen, ob das Datenpaket in einer CRC benutzte Daten enthält. Ein Ethernet-Datenpaket, das CRC einschließt, enthält zusätzlich zu den Paketdaten vier Bytes von CRC-Daten, die zur Fehlerprüfung verwendet werden.

[0197] Das DATA_POINTER Feld **834** spezifiziert die durch ein TLAN **226** während eines Datenübertragungsvorgangs geltend zu machende PCI-Adresse. Die Adresse ist vorzugsweise für Sende- und Empfangsvorgänge die gleiche und ist die PACKET DATA MEMORY BASE ADDRESS.

[0198] Während eines Empfangsvorgangs macht ein TLAN **226** die PACKET DATA MEMORY BASE ADDRESS geltend, und die Empfangsdaten-Decodierlogik **813** decodiert die Adresse, und ein Schreibzyklus auf dem PCI-Bus **222** ermöglicht der PCI RX FIFO Steuerlogik **817**, den Empfang von Paketdaten in einem ausgewählten TPI RX FIFO zu erlauben. Während eines Sendevorgangs macht ein TLAN **226** die PACKET DATA MEMORY BASE ADDRESS geltend, und die Sendedaten-Decodierlogik **815** decodiert die Adresse, und ein Lesevorgang ermöglicht der PCI TX FIFO Steuerlogik **816**, die Übertragung von Paketdaten aus einem ausgewählten TPI TX FIFO durchzuführen.

[0199] Das COUNT Feld **833** spezifiziert eine vorhandene Datenmenge oder die Menge an verfügbarem Pufferplatz bei dem momentanen Wert des DATA_POINTER Feldes **834**. Während eines Datenempfangsvorgangs setzt die Empfangslisten-Decodierlogik **812** das COUNT Feld **833** auf einen in ein RCV_DATA_COUNT Register **847b** (Fig. 8F) der TPI-Steuerregister **846** geschriebenen Wert. Der Wert von dem RCV_DATA_COUNT Register **847b** bestimmt die größte durch die TPI **220** zu empfangende Paketgröße. Als Vorgabe beträgt dieser Wert 1518 Bytes, was die größte ETHERNET-Datenpaketgröße mit vier Bytes von CRC ist. Während eines Datensendevorgangs setzt die TPI **220** das COUNT Feld **833** auf den gleichen Wert wie das PACKET_SIZE Feld **832a**.

[0200] Fig. 8E ist ein Diagramm, das eine Definition der von der TPI **220** eingesetzten TPI-PCI-Konfigurationsregister **835** veranschaulicht. Die TPI-PCI-Konfigurationsregister **835** umfassen Register, die allen PCI-Busarchitekturen gemeinsam sind, sowie für die TPI **220** einmalige, zusätzliche Register. Register, die allen PCI-Bussen gemeinsam sind, umfassen ein DEVICE_ID Register **836a**, ein VENDOR_ID Register **836b**, ein STATUS Register **837a**, ein COMMAND Register **837b**, ein CLASS_CODE Register **838a**, ein REV_ID Register **838b**, ein BIST Register **839a**, ein HDR_TYPE Register **839b**, ein LATENCY Register **839c**, ein CACHELSZ Register **839d**, ein MAXLAT Register **845a**, ein MINGNT Register **845b**, ein INTPIN Register **845c** und ein INTLINE Register **845d**. Für die TPI **220** einmalige Register umfassen ein TPI CONTROL **10** BASE ADDRESS Register **840** ein TPI CONTROL MEMORY BASE ADDRESS Register **841**, ein TRANSMIT LIST MEMORY BASE ADDRESS Register **842**, ein RECEIVE LIST MEMORY BASE ADDRESS Register **843** und ein PACKET DATA MEMORY BASE ADDRESS Register **844**.

[0201] Nach Initialisierung enthält das TPI CONTROL **10** BASE ADDRESS Register **840** eine TPI CONTROL IO BASE ADDRESS für die TPI-Steuerregister **846**. Das TPI CONTROL MEMORY BASE ADDRESS Register **841** enthält eine TPI CONTROL MEMORY BASE ADDRESS für die TPI-Steuerregister **846**. Auf diese Weise sind die TPI-Steuerregister **846** sowohl im I/O- als auch im Speicherraum des PCI-Busses **220** zugänglich. Das TRANSMIT LIST MEMORY BASE ADDRESS Register **842** enthält die TRANSMIT LIST MEMORY BASE ADDRESS für die TX CNTL LIST **808b**, die von der Sendelisten-Decodierlogik **814** decodiert wird. Das RE-

CEIVE LIST MEMORY BASE ADDRESS Register **843** enthält die RECEIVE LIST MEMORY BASE ADDRESS für die RX CNTL LIST **808a**, die von der Empfangslisten-Decodierlogik **812** decodiert wird. Das PACKET DATA MEMORY BASE ADDRESS Register **844** enthält die PACKET DATA MEMORY BASE ADDRESS, die den Datenpuffern **807** der TPI **220** entspricht. Die PACKET DATA MEMORY BASE ADDRESS wird sowohl von der Sendedaten-Decodierlogik **815** als auch der Empfangsdaten-Decodierlogik **813** decodiert.

[0202] Fig. 8F ist ein Diagramm, das die Definition der durch die PTI **220** verwendeten TPI-Steuerregister **846** veranschaulicht. Die TPI-Steueregister **846** umfassen ein RCV_DATA_COUNT Register **847b**, ein XBSIZE3 Register **848a**, ein XBSIZE2 Register **848b**, ein XBSIZE1 Register **848c**, ein XBSIZE0 Register **848d**, ein RBSIZE3 Register **849a**, ein RBSIZE2 Register **849b**, ein RBSIZE1 Register **849c**, ein RBSIZE0 Register **849d**, ein NET_PRI3 Register **850a**, ein NET_PRI2 Register **850b**, ein NET_PRI1 Register **850c**, ein NET_PRI0 Register **850d**, ein TLAN0 MEMORY BASE ADDRESS Register **851**, ein TLAN1 MEMORY BASE ADDRESS Register **852**, ein TLAN2 MEMORY BASE ADDRESS Register **853** und ein TLAN3 MEMORY BASE ADDRESS Register **845**.

[0203] Das RCV_DATA_COUNT Register **847b** speichert die von der TPI **220** behandelte Maximalgröße von empfangenen Datenpaketen. Die TPI **220** gewinnt diesen Wert zurück und legt ihn in das COUNT Feld **833** der RX CNTL LIST **808a**. Jedes der XBSIZE Register **848a–d** hält eine Sendestoßgröße in DWORDs für betreffende Ports, nämlich XBSIZE0 für Port24, XBSIZE1 für Port25, XBSIZE2 für Port26, XBSIZE3 für Port27. Die XBSIZE Sendestoßgrößenwerte werden von der HSB TX FIFO Steuerlogik **822** und der Portstatuslogik **820** der TPI **220** verwendet, wenn festgestellt wird, ob genug Paketpufferplatz in einem betreffenden TPI TX FIFO vorhanden ist, um Daten von dem EPSM **210** für den betreffenden Port anzufordern. Jedes der RBSIZE Register **849a–d** hält betreffende HSB-Empfangsstoßgrößen in DWORDs für die betreffenden Ports, nämlich RBSIZE0 für Port24, RBSIZE1 für Port25, RBSIZE2 für Port26 und RBSIZE3 für Port27. Die RBSIZE Empfangsstoßgrößenwerte werden von der HSB RX FIFO Steuerlogik **821** und der Portstatuslogik **820** verwendet, wenn festgestellt wird, ob genug Paketdaten in einem betreffenden TPI RX FIFO vorhanden sind, um eine Übertragung von empfangenen Daten von dem betreffenden Port an den EPSM **210** anzufordern. In der gezeigten Ausführung sind die in den XBSIZE- RBSIZE-Registern **848**, **849** gespeicherten Werte einander gleich und gleich dem TBUS-Wert. Die XBSIZE Register **848** und die RBSIZE Register **849** werden jedoch, abhängig von der Ausführung, mit jedem gewünschten Stoßübertragungswert programmiert.

[0204] Die NET_PRI Register **850** halten betreffende Netzwerk-Prioritätswerte für die Ports, nämlich NET_PRI0 für Port24, NET_PRI1 für Port25, NET_PRI2 für Port26 und NET_PRI3 für Port27. Diese Werte werden von der Sendelisten-Decodierlogik **814** verwendet, um die Sendepriorität der betreffenden Ports festzulegen. Das TLAN0 MEMORY BASE ADDRESS Register **851** hält eine PCI-Speicheradresse, bezeichnet als TLAN0 MEMORY BASE ADDRESS, für Port24. Das TLAN1 MEMORY BASE ADDRESS Register **852** hält eine PCI-Speicheradresse, bezeichnet als TLAN1 MEMORY BASE ADDRESS, für Port25. Das TLAN2 MEMORY BASE ADDRESS Register **853** hält eine PCI-Speicheradresse, bezeichnet als TLAN2 MEMORY BASE ADDRESS, für Port26. Das TLAN3 MEMORY BASE ADDRESS Register **854** hält eine PCI-Speicheradresse, bezeichnet als TLAN3 MEMORY BASE ADDress für Port27. Jedes dieser Register wird beim Einschalten durch die CPU **230** nach Bestimmen der Adressen jedes der TLANs **226** initialisiert. Diese Werte werden an die PCI TX FIFO Steuerlogik **816** geliefert und von dieser verwendet, um jeden Sendebefehl auf dem PCI-Bus **222** auszugeben, um Paketsendevorgänge zu starten.

[0205] Fig. 8G ist ein Flussdiagramm, das PCI-Initialisierungsoperationen der CPU **230** beim Initialisieren, Starten oder Rücksetzen des Netzwerkschalters **102** veranschaulicht. Im ersten Schritt **855** konfiguriert die CPU **230** den PCI-Bus **222**, bildet die TLANs **226** in den PCI-Speicherraum ab und schreibt diese Konfiguration über den PCI-Bus **222** in die TPI-PCI-Konfigurationsregister **835**. Die Schritte zum Konfigurieren des PCI-Busses **222** sind bekannt und werden nicht werter beschrieben.

[0206] Insbesondere ist das DEVICE_ID Register **836a** das Standard-PCI-Geräte-ID-Register und sein Wert wird auf 0x500h gesetzt. Das VENDOR_ID Register **836b** ist das Standard-PCI-Lieferanten-ID-Register und sein Wert wird auf 0x0E11h gesetzt. Das STATUS-Register **837a** ist das Standard-PCI-Gerätestatusregister. Das COMMAND-Register **837b** ist das Standard-PCI-Gerätebefehlsregister. Das CLASS_CODE Register **838a** ist das Standard-PCI-Geräteklassencoderegister und sein Wert wird auf 0x060200h gesetzt. Das REV_ID Register **838b** ist das Standard-PCI-Geräterevisions-ID-Register und sein Wert wird auf 0x00h gesetzt. Das BIST-Register **839a** ist das Standard-PCI-BIST-Statusregister und sein Wert wird auf 0x00h gesetzt. Das HDR_TYPE Register **839b** ist das Standard-PCI-Vorspanntypregister und sein Wert wird auf 0x80h gesetzt. Das LATENCY Register **839c** ist das Standard-PCI-Latenztypregister und wird durch die CPU **230** initialisiert. Das CACHELST-Register **839d** ist das Standard-PCI-Cachelinegrößenregister und wird durch die

CPU **230** initialisiert. Das MAXLAT_register **845a** ist das Standard-PCI-Gerätemaximallatenzregister und sein Wert wird auf 0x00h gesetzt. Das MINGNT-Register **846b** ist das Standard-PCI-Geräteminimalgewähnangsregister und sein Wert wird auf 0x00h gesetzt. Das INTPIN-Register **845c** ist das Standard-PCI-Geräteunterbrechungspinregister und sein Wert wird auf 0x00h gesetzt. Das INTLINE-Register **845d** ist das Standard-PCI-Geräteunterbrechungsleitungsregister und wird von der CPU **230** eingerichtet.

[0207] Ferner schreibt in Schritt **855** die CPU **230** einen Wert von 0xFFFFFFFFh in jedes der folgenden Register: das TPI CONTROL **10** BASE ADDRESS Register **840**, das TPI CONTROL MEMORY BASE ADDRESS Register **841**, das TRANSMIT LIST MEMORY BASE ADDRESS Register **842**, das RECEIVE LIST MEMORY BASE ADDRESS Register **843** und das PACKET DATA MEMORY BASE ADDRESS Register **844**. Nach jedem Schreiben ersetzt die TPI **220** den Wert in jedem Register durch einen Wert, der die Menge an I/O- oder Speicherplatz angibt, der von dem einzelnen angegebenen Register benötigt wird. Die CPU **230** liest als Reaktion jeden neuen Wert in jedem Register und schreibt dann eine Basisadresse in jedes Register zurück, um die Wesenheit in den PCI-I/O- oder Speicherraum abzubilden.

[0208] Das heißt, nach Bestimmen der benötigten Menge an Platz schreibt die CPU **230** die TPI CONTROL IO BASE ADDRESS in das TPI CONTROL IO BASE ADDRESS Register **840**, um den I/O-Raumzugriff der TPI-Steuerregister **846** zu ermöglichen, die CPU **230** schreibt die TPI CONTROL MEMORY BASE ADDRESS in das TPI CONTROL MEMORY BASE ADDRESS Register **841**, um den Speicherraumzugriff der TPI-Steuerregister **846** zu ermöglichen, die CPU **230** schreibt die TRANSMIT LIST MEMORY BASE ADDRESS in das TRANSMIT LIST BASE ADDRESS Register **842**, das der Adresse des TX CNTL LIST **808b** Speicherblocks entspricht, die CPU **230** schreibt die RECEIVE LIST MEMORY BASE ADDRESS in das RECEIVE LIST MEMORY BASE ADDRESS Register **843**, das der Adresse der RX CNTL LIST **808a** entspricht, und die CPU **230** schreibt die PACKET DATA MEMORY BASE ADDRESS in das PACKET DATA MEMORY BASE ADDRESS Register **844**, das der PCI-Adresse des Datenpuffers **807** entspricht.

[0209] Im nächsten Schritt **856a** fragt die CPU **230** jedes TLAN **226**, eines nach dem anderen, auf dem PCI-Bus **222** ab, um die Zahl vorhandener TLANs und ihre entsprechenden PCI-Adressen zu bestimmen. Im nächsten Schritt **856b** initialisiert die CPU **230** die abgefragten TLANs **226** in einen bekannten Ruhezustand. Die CPU **230** stellt dann im nächsten Schritt **857** fest, ob noch mehr TLANs **226** vorhanden sind, und kehrt, wenn ja, zu Schritt **856a** zurück, um das nächste TLAN abzufragen, bis alle TLANs **226** auf dem PCI-Bus **222** initialisiert sind. Zu dieser Zeit sind die Werte der TLAN0 MEMORY BASE ADDRESS, der TLAN1 MEMORY BASE ADDRESS, der TLAN2 MEMORY BASE ADDRESS und der TLAN3 MEMORY BASE ADDRESS bekannt.

[0210] Im nächsten Schritt **858** initialisiert die CPU **230** die TP1-Steuerregister **846** auf die geeigneten Werte, wie oben mit Verweis auf **Fig. 8F** beschrieben. Dies schließt die Werte der TLAN0 MEMORY BASE ADDRESS, der TLAN1 MEMORY BASE ADDRESS, der TLAN2 MEMORY BASE ADDRESS und der TLAN3 MEMORY BASE ADDRESS ein. Im nächsten Schritt **859** beginnt die CPU **230** die Einleitung des Empfangsvorgangs für jedes TLAN **226** durch Schreiben der RECEIVE LIST MEMORY BASE ADDRESS in das Kanalparameterregister **828b**. Die Einleitung des Empfangsvorgangs wird in Schritt **860** vollendet, wo die CPU **230** in das Befehlsregister **828a** jedes TLANs **226** schreibt. In dieser Weise initialisiert, beginnt jedes TLAN **226** sofort einen Empfangsvorgang, indem es den PCI-Bus **222** auffordert, eine Empfangssteuerliste anzufordern.

[0211] **Fig. 8H** ist ein Flussdiagramm, das den Empfangsvorgang des Netzwerkschalters **102** für jedes der TLANs **226** veranschaulicht. Der Vorgang beginnt im ersten Schritt **861a**, wo ein TLAN **226** den PCI-Bus **222** von dem PCI-Arbiter **811** anfordert und die Steuerung erhält. Im nächsten Schritt **861b** macht das TLAN **226** die RECEIVE LIST MEMORY ADDRESS auf dem PCI-Buss **222** geltend, um eine Empfangssteuerliste anzufordern, und die CPU **230** liefert im nächsten Schritt **861c** eine Empfangssteuerliste an das TLAN **226**. Die Empfangssteuerliste enthält die PACKET DATA MEMORY BASE ADDRESS, um das TLAN **226** zu informieren, wohin oder wie ein empfangenes Datenpaket zu senden ist. Im nächsten Schritt **861d** gibt das TLAN **226** die Steuerung des PCI-Busses **222** frei.

[0212] Ein TLAN **226** empfängt ein Datenpaket von einem Netzwerk **112**, wie im nächsten Schritt **862a** angegeben, und verlangt und erhält dann im nächsten Schritt **862b** die Steuerung des PCI-Busses **222**. Das TLAN **226** schreibt dann im nächsten Schritt **862c** einen Datenstoß unter Verwendung der PACKET DATA MEMORY BASE ADDRESS als die Adresse auf den PCI-Bus **222**, während die TPI **220** Daten in einen ausgewählten TPI RX FIFO schreibt, wie im nächsten Schritt **862d** angegeben. Nach Vollendung des Schreibstoßes gibt das TLAN im nächsten Schritt **862e** den PCI-Bus **222** frei. Im nächsten Schritt **865** stellt das TLAN **226** fest, ob das ganze Datenpaket an die TPI **220** gesendet worden ist, was durch eine letzte DWORD-Schreiboperation an-

gezeigt wird. Wenn nicht, kehrt der Vorgang zu Schritt **862b** zurück, wo das TLAN **226** erneut den PCI-Bus **222** auffordert, einen weiteren Datenstoß zu senden.

[0213] Nachdem das TLAN **226** den letzten Teil des Datenpakets gesendet hat, führt es eine letzte Iteration durch, um die TPI **220** über das Ende des Pakets zu informieren. Das heißt, das TLAN **226** führt eine letzte DWORD-Übertragung an das PACKET_SIZE Feld **832a** und das CSTAT Feld **832b** in der RX CNTL LIST **808a** der TPI **220** aus. Diese DWORD-Übertragung aktualisiert die RX CNTL LIST **808a** mit der Paketgröße des gerade vollendeten Datenpakets und aktualisiert das Rahmen-Vollendet-Bit in dem CSTAT Feld **832b**. Die TPI **220** erfasst diese Schreiboperation, wie in Schritt **865** angegeben, und setzt interne Flags, um anzuzeigen, dass der Vorgang abgeschlossen ist, und übergibt den geeigneten Status an die Portstatuslogik **820**, wie in Schritt **866** angegeben.

[0214] Fig. 8I ist ein Flussdiagramm, das einen Empfangsdaten-Übertragungsvorgang von der TPI **220** an den EPSM **210** über den HSB **206** veranschaulicht. Der Vorgang beginnt in einem ersten Schritt **876**, wo die Portstatuslogik **820** der TPI **220** eine Datenmenge in jedem der TPI RX FIFOs, die gleich oder größer ist als die betreffende RBSIZE, wie in den TPI-Steuerregistern **846** bereitgestellt, oder das durch ein TLAN **226** angezeigte EOP für diesen Port erfasst.

[0215] Wie im nächsten Schritt **877** angegeben, antwortet die TPI **220** auf Abfragen des EPSM **210**, indem sie geeignete PKT_AVAIL[6]* Signale in gemultiplexter Form geltend macht, die angeben, ob in jedem der TPI RX FIFOs genug Daten verfügbar sind. Das Abfragen erfolgt unabhängig und wird zur Klärung eingeschlossen. Wenn das PKT_AVAIL[6]* Signal anzeigt, dass in jedem TPI RX FIFO der TPI **220** genug Daten vorhanden sind, leitet schließlich der EPSM 210 im nächsten Schritt **878** einen Lesezyklus auf dem HSB **206** an den spezifizierten Port ein, wenn es genug Pufferplatz in einem verfügbaren Empfangspuffer des EPSM **210** gibt.

[0216] Im nächsten Schritt **879** erfasst die Portstatuslogik **820** der TPI **220** den Lesezyklus auf dem HSB **206** und wählt den geeigneten TPI RX FIFO aus, um Daten bereitzustellen. In Schritt **880** sendet dann die TPI **220** den Datenstoß an den EPSM **210**. Wenn während der Datenübertragung von Schritt **880** die Portstatuslogik **820** feststellt, dass die momentane Datenübertragung über den HSB **206** der Beginn eines Pakets ist, wie im nächsten Schritt **881a** angegeben, macht die TPI **220** in Schritt **881b** das SOP* Signal auf dem HSB **206** während der Datenübertragung geltend. Desgleichen, wenn während der Datenübertragung im Schritt **880** die Portstatuslogik **820** feststellt, dass die momentane Datenübertragung über den HSB **206** das Ende eines Pakets ist, wie im nächsten Schritt **882a** angegeben, macht die TPI **220**, wie durch Schritt **881b** angegeben, das EOP* Signal auf dem HSB **206** während der Datenbertragung geltend. Von Schritt **882a** oder **882b** kehrt der Vorgang zu Schritt **876** zurück.

[0217] Fig. 8J ist ein Flussdiagramm, das einen Sendedaten-Übertragungsvorgang zum Übertragen von Paketdaten von dem EPSM **210** über den HSB **206** an die TPI **220** veranschaulicht. Der Vorgang beginnt beim ersten Schritt **890**, wo die Portstatuslogik **820** der TPI **220** feststellt, dass einer der TPI TX FIFOs ein Menge an verfügbarem Pufferplatz hat, der gleich oder größer ist als die entsprechende XBSIZE. Der Vorgang geht dann zum nächsten Schritt **891**, wo die Portstatuslogik **820** auf eine Abfrage des EPSM **210** durch geeignetes Geltendmachen des BUF_AVAIL[6]* Signals in gemultiplexter Form antwortet, um verfügbaren Pufferplatz in dem entsprechenden TPI TX FIFO anzuzeigen. Wie oben beschrieben, erfolgt die Abfrage unabhängig und wird zur Klärung eingeschlossen. Im nächsten Schritt **892** leitet der EPSM **210** einen Schreibzyklus auf dem HSB **206** an einen Port ein, der dem TPI TX FIFO entspricht, ein, wenn für diesen Port genug Daten zum Senden durch den EPSM **210** verfügbar sind. Im nächsten Schritt **893** erfasst die Portstatuslogik **820** der TPI **220** den Schreibzyklus auf dem HSB **206** und wählt den geeigneten TPI TX FIFO für den angegebenen Port aus. Im nächsten Schritt **894** sendet der EPSM **210** einen Stoß von Daten über den HSB **206** an die TPI **220**, und die TPI **220** schreibt die Daten in den entsprechenden TPI TX FIFO in der TPI **220**.

[0218] Wenn, wie in Schritt **895a** angegeben, die TPI **220** die Geltendmachung des SOP* Signals während des Datenstoßes von Schritt **894** erfasst, wird in Schritt **895b** das erste DWORD von Daten, das die Paketgröße hält, in das PACKET SIZE Anhängerregister **819c** gelegt. Wenn, wie in Schritt **896a** angegeben, die TPI **220** die Geltendmachug des EOP* Signals während des Datenstoßes von Schritt **894** erfasst, setzt die TPI **220** in Schritt **896b** ein Flag in der TPI **220**, um das Ende des Datenpakets anzuzeigen. Von Schritt **896a** oder **896b** kehrt der Vorgang zurück zu Schritt **890**.

[0219] Fig. 8K ist ein Flussdiagramm, das einen Sendevorgang des Netzwerkschalters **102** für jedes der TLANs **226** veranschaulicht. Im ersten Schritt **867** erfasst die TPI **220** Daten in jedem der TPI TX FIFOs und verlangt und erhält als Reaktion die Steuerung des PCI-Busses **222** von dem PCI-Arbiter **811**. Im nächsten

Schritt **868** schreibt die TPI **220** einen Sendebefehl in das Befehlsregister **828a** des entsprechenden TLAN **226**. In Schritt **869** gibt dann die TPI **220** den PCI-Bus **222** frei.

[0220] Im nächsten Schritt **870a** verlangt und erhält das TLAN **226**, das den Sendebefehl empfängt, die Steuerung des PCI-Busses **222** vom PCI-Arbiter **811** und verlangt dann eine Sendesteuerliste von der TPI **220**. Im nächsten Schritt **870b** liefert die TPI **220** die Sendesteuerliste an das TLAN **226**, das die Steuerung des PCI-Busses **222** innehat, wo das TLAN **226** die Sendesteuerliste an seinen Sendesteuerlistenpuffer **827b** liefert. Im nächsten Schritt **870c** gibt das TLAN **226** den PCI-Bus **222** frei, fordert aber sofort den PCI-Bus **222** wieder an, wie in Schritt **870d** angegeben. Sobald das TLAN **226** wieder die Steuerung des PCI-Busses **222** erhält, beginnt es die Ausführung der Sendesteuerliste, wie in Schritt **871a** angegeben, durch Anfordern eines Stoßes von Daten von der TPI **220**. Das heißt, das TLAN **226** macht in Schritt **871a** die PACKET DATA MEMORY BASE ADDRESS auf dem PCI-Bus **222** geltend. Im nächsten Schritt **871b** antwortet die TPI **220** durch Auswählen und Freigeben des entsprechenden TPI TX FIFO und liefert Daten über den PCI-Bus **222** an das TLAN **226**. Nach jedem Datenstoß gibt das TLAN **226** die Steuerung des PCI-Busses **222** frei, wie im nächsten Schritt **871a** angegeben. Wenn die Übertragung eines vollständigen Pakets von Daten nicht vollendet wurde, wie im nächsten Schritt **872** angegeben, kehrt der Vorgang zu Schritt **870d** zurück, wo das TLAN **226** wieder die Steuerung des PCI-Busses **222** anfordert und schließlich zurückgewinnt.

[0221] Wenn die Übertragung des Pakets vollendet war, wie in Schritt **872a** bestimmt, geht der Ablauf zu Schritt **873a**, wo das TLAN **226** an die TPI **220** schreibt, dass die Datenübertragung vollendet ist, und die TPI **220** signalisiert, dass der Vorgang abgeschlossen ist. Das heißt, das TLAN **226** führt einen letzten Ein-DWORD-Schreibvorgang in das CSTAT Feld **832b** der TX CNTL LIST **808b** aus, um ein Rahmen-Vollendet-Bit in dem CSTAT Feld **832** zu setzen. Ferner wird das PACKET_SIZE Feld **832a** der TX CNTL LIST **808b** mit der Größe eines von der TPI **220** an ein TLAN **226** zu sendenden Pakets geladen. Sobald das TLAN **226** den Schreibvorgang vollendet hat, gibt es in Schritt **873b** den PCI-Bus **222** frei. Von Schritt **873b** kehrt der Vorgang zu Schritt **867** für den nächsten Sendevorgang zurück.

[0222] Es ist nun zu erkennen, dass nach Initialisierung durch die CPU **230** die TPI **220** konfiguriert ist, um mit den TLANs **226** zusammenzuarbeiten, um der CPU **230** zu gestatten, andere wichtige Aufgaben und Funktionen des Netzwerkschalters **102** durchzuführen. Die CPU **230** initialsiert PCI-Speicher- und I/O-Raum durch Bestimmen des Typs und der Zahl von Vorrichtungen auf dem PCI-Bus **222** und Zuweisen von entsprechenden Adressenwerten. Die CPU **230** liefert Anfangsadressenwerte der TPI **220** an jedes der TLANs **226** und fügt einen Befehl ein, um Vorgänge einzuleiten. Die TLANs **226** werden konfiguriert, um eine Steuerliste anzufordern und dann die Steuerliste auszuführen, um Daten aus einem Speicher, der sich an einer Adresse in der Steuerliste befindet, zu lesen bzw. in denselben zu schreiben. Die TPI **220** wird konfiguriert, um jede Steuerliste zu aktualisieren und jedem anfordernden TLAN **226** zur Verfügung zu stellen. Ferner wird die TPI **220** konfiguriert, um Sendevorgänge durch Schreiben eines Befehls in das geeignete TLAN **226** einzuleiten, und dann die entsprechende Sendesteuerliste bereitzustellen, wenn anschließend angefordert. Auf diese Weise ist die CPU **230** nach Durchführen der Initialisierung frei, um andere Funktionen des Netzwerkschalters **102** durchzuführen.

[0223] Fig. 9A ist ein Blockschaltbild, das die Organisation des Speichers **212** zeigt. In der gezeigten Ausführung liegt die Größe des Speichers **212** zwischen 4 und 16 MB, obwohl die Speichergröße variieren kann und so groß oder so klein wie gewünscht sein kann. Die Breite der in Fig. 9A–Fig. 9G gezeigten Speicherabschnittsblöcke und daher die Breite jeder Speicherzeile beträgt ein DWORD oder 32 Bit. Der Speicher **212** ist in zwei Hauptabschnitte geteilt, einschließlich eines Hash-Speicherabschnitts **902** und eines Paketspeicherabschnitts **904**. Der Hash-Speicherabschnitt **902** dient als ein Netzwerkgeräte-Identifikationsabschnitt zum Identifizieren eines oder mehr der Netzwerkgeräte in den mit dem Netzwerkschalter **102** verbundenen Netzwerken **106**, **112**. Die Größe des Hash-Speicherabschnitts **902** ist basierend auf der Zahl von Vorrichtungen und zugehöriger Adressen und der gewünschten Einträge programmierbar. In der gezeigten Ausführung enthält der Hash-Speicherabschnitt **902** 256 kB an Speicher zum Unterstützen von wenigstens 8 k ($k = 2^{10} = 1{,}024$) Adressen bis zu 16 k Adressen. Der Hash-Speicherabschnitt **902** kann irgendwo in dem Speicher **212** liegen, und befindet sich in der gezeigten Ausführung am Anfang des Speichers **212**. Die Größe des Paketspeicherabschnitts **904** ist der Rest des Speichers **212**, der nicht für den Hash-Speicherabschnitt **902** verwendet wird.

[0224] Fig. 9B ist ein Blockdiagramm der Organisation des Hash-Speicherabschitts **902** des Speichers **212**. Der Hash-Speicherabschnitt **902** ist mit einer Länge von 16 kB gezeigt, wobei zu verstehen ist, dass die Hash-Speicherabschnittsgröße, wie gewünscht, entweder fest oder programmierbar ist. Der Hash-Speicherabschnitt **902** ist in zwei 128 kB Abschnitte geteilt, einschließlich eines ersten 128 kB Haupt-Hash-Speicherabschnitts **905** für Haupt-Hash-Einträge und eines zweiten 128 kB verketteten Hash-Eintragsabschnitts **908**

für verkettete Hash-Einträge. Jeder der Abschnitte **908**, **908** enthält 8 k Einträge, je 16 Bytes lang.

[0225] Fig. 9C ist ein Diagramm, das die Organisation eines Hash-Tabelleneintrags **910** zeigt, der für die Einträge in dem Hash-Speicherabschnitt **902**, einschließlich des Haupt-Hash-Eintragsabschnitts **906** und des verketteten Hash-Eintragsabschnitts **908**, repräsentativ ist. Jeder Eintrag **910** entspricht einer Netzwerkvorrichtung der mit dem Netzwerkschalter **102** verbundenen Netzwerke **106**, **112**. Jeder der Haupteinräge befindet sich an einer Hash-Adresse, die durch "Haschieren" der MAC-Adresse für diese Vorrichtung bestimmt wird. Das heißt, jeder Netzwerkvorrichtung wird eine 48-Bit Hardware-Adresse, auch bekannt als physikalische Adresse oder MAC-Adresse, zugewiesen, die ein einmaliger nummerischer Wert ist, der jeder Netzwerkvorrichtung während des Herstellungsprozesses oder durch Setzen von Brücken oder Schaltern während der Netzwerkinstallation zugewiesen wird. Ein Teil dieser MAC-Adresse wird dem Hersteller durch das IEEE (Institute of Electrical and Electronics Engineers) zugewiesen und ist allen Komponenten von diesem Hersteller gemeinsam, und der zweite Teil der Hardwre-Adresse ist ein einmalger Wert, der durch den Hardware-Hersteller zugeteilt wird. Die ersten 6 Bytes, oder Bytes 5–0, des Hash-Tabelleneintrags **910** enthalten die MAC-Adresse der mit diesem Eintrag verbundenen Vorrichtung. Der Netzwerkschalter **102** fügt daher einen Hash-Tabelleneintrag für jede Netzwerkvorrichtung hinzu, die ein Datenpaket einschließlich seiner Quellen-MAC-Adresse sendet.

[0226] Jedes von jeder Netzwerkvorrichtung in den Netzwerken **106**, **112** gesendete Datenpaket enthält typischerweise eine Quellen- und eine Ziel-MAC-Adresse, die entsprechend einem oder mehreren Algorithmen haschiert werden. In der gezeigten Ausführung werden zwei Teile jeder MAC-Adresse logisch kombiniert oder verglichen, um eine entsprechende Hash-Adresse zu berechnen. Jeder Teil ist 13 bis 16 Bits, die durch Exklusiv-ODER-(XOR) Logik bitweise kombiniert werden, um 13 bis 16 Bit Hash-Adressen zu bilden. Zum Beispiel werden die ersten 16 Bits einer MAC-Adresse, oder MA[15:0], bitweise mit den nächsten 16 Bits der MAC-Adresse MA[31:16]XOR-verknüpft, um die Hash-Adresse HA[15:0] zu erhalten. In einer Ausführung werden die ersten 13, 14, 15 oder 16 Bits des haschierten Ergebnisses als die Hash-Adresse HA verwendet. Alternativ werden die ersten 13 Bits der MAC-Adresse MA[12:0] mit den nächsten 13 Bits MA[25:13] haschiert, um eine 13-Bit Hash-Adresse MA[12:0] zu erhalten. Oder die ersten 14 Bits der MAC-Adressen MA[13:0] werden mit den nächsten 14 Bits MA[27:14] haschiert, um eine 14-Bit Adresse MA[13:0] zu erhalten, usw. Es versteht sich, dass viele andere verschiedene Hash-Algorithmen bekannt sind und verwendet werden können, um alle bestimmten Kombinationen von Adressenbits zu kombinieren, wie den Fachleuten in der Technik bekannt ist, und dass die vorliegende Erfindung nicht auf ein bestimmtes Hash-Schema begrenzt ist.

[0227] Die Hash-Adresse wird als die wirkliche Adresse oder als eine Offsetadresse verwendet, um jeden der Hash-Einträge des Haupt-Hash-Eintragsabschnitts **906** aufzufinden. Obwohl die MAC-Adressen einmalig sind, kann die Hash-Adresse nicht einmalig sein, sodass zwei verschiedene MAC-Adressen zu der gleichen Hash-Adresse haschieren. Der verkettete Hash-Eintragsabschnitt **908** wird bereitgestellt, um doppelte Hash-Adressen für verschiedene Vorrichtungen zu speichern, wie unten weiter beschrieben. Die Organisation, die einen durch die Hash-Adressen zugänglichen Haupt-Hash-Eintragsabschnitt **906** und einen durch eine im ersten Eintrag des Hauptabschnitts **906** gelegene Link-Adresse zugänglichen verketteten Hash-Eintragsabschnitt **908** umfasst, beseitigt wenigstens eine Verzweigungsoperation. Anstatt eine Liste von Zeigern zu verwenden, um auf die Tabelleneinträge zuzugreifen, wird der erste Eintrag im Speicher **212** in einer einzigen Verzweigungsoperation zurückgewonnen, der zweite Eintrag in einer zweiten Verzweigungsoperation usw. Auf diese Weise liefert die Organisation des Speichers **212** einen effizienteren Zugriff der Hash-Einträge durch Beseitigen wenigstens einer Verzweigungsoperation pro Zugriff.

[0228] Das nächste Byte (6) des Hash-Tabelleneintrags **910** enthält eine binäre Portnummer (Port-Num), die die zugehörige Portnummer identifiziert, mit der die Vorrichtung verbunden ist, wo die Portnummer für Port0 null ist, die Portnummer für Port1 eins ist, die Portnummer für Port28 (für die CPU **230**) 28 ist usw. Das nächste Byte (7) ist ein Steuer- und Alters-Informationsbyte (Control/Age), das ein Gültig-Bit (VALIDENTRY) enthält, das identifiziert, ob der Eintrag gültig ist oder nicht, wo logisch "1" anzeigt, dass der Eintrag gültig ist, und logisch "0" anzeigt, dass der Eintrag nicht gültig ist, ansonsten ein leerer Eintrag genannt. Das Control/Age-Byte enthält eine binäre Altersnummer (AGE), die die vergangene Zeit seit dem letzten mit dieser Vorrichtung verbundenen Quellenzugriff darstellt. Eine Vorrichtung kann betagt sein und durch die CPU **230** aus dem Hash-Eintrag gelöscht werden, nachdem sie für einen vorbestimmten Zeitraum seit dem letzten Quellenzugriff nicht verwendet wurde. Die Messung der vergangenen Zeit wird mit einem von mehreren Verfahren durchgeführt und kann in Sekunden oder Teilen davon, Minuten, Stunden usw. gemessen werden. Der vorbestimmte Zeitraum, bevor eine Vorrichtung betagt ist, ist auch programmierbar. In einer alternativen Ausführung ist die AGE-Nummer ein einzelnes Bit, das benutzt wird, um anzuzeigen, ob die Vorrichtung für "alt" gehalten wird oder nicht, was durch einen Laufzeit-Timer oder dergleichen festgelegt wird.

[0229] Die nächsten vier Bytes (B:8) definieren einen 29-Bit Virtual-LAN-(VLAN)Bitmap-Wert, der Portgruppierungen, wenn verwendet, darstellt. Jedes Bit des VLAN-Wertes entspricht einem betreffenden der Ports und wird gesetzt, wenn die Vorrichtung oder Port mit diesem Port gruppiert wird. Der VLAN-Wert identifiziert daher, mit welchem der anderen Ports die Vorrichtung gruppiert ist. Dies ermöglicht den Netzwerken **106**, **112**, in jeder gewünschten Kombination kombiniert zu werden, um eine Vielzahl verschiedener, mit dem Netzwerkschalter **102** verbundener LANs zu bilden. Wenn z. B. die ersten fünf Ports Port0 bis Port4 miteinander gruppiert werden, ist der VLAN-Wert für jeden 0000001fFh, wo "h" einen Hexadezimalwert bezeichnet. Ein von einer mit Port Port2 verbundenen Vorrichtung gesendetes BC-Paket wird an die Ports Port0, Port1, Port3 und Port4 wiederholt, anstatt an alle anderen Ports des Netzwerkschalters **102** wiederholt zu werden. Ein VLAN-Wert von nur Einsen oder 1FFFFFFFh bezeichnet keine Gruppierungen für diese Vorrichtung. Es wird angemerkt, dass es für eine Vorrichtung möglich ist, mit mehr als einer Gruppe verbunden zu werden. In einer alternativen Ausführung kann ein VLAN-Feld eingeschlossen werden, um mehr als eine von mehreren VLAN-Gruppen, zu denen jede Vorrichtung gehört, so vorhanden, zu identifizieren.

[0230] Die letzten vier Bytes (F:C) jedes Hash-Tabelleneintrags **910** ist eine Link-Adresse (Link A[31:0] oder Link-Adresse), die auf den nächsten Eintrag mit einer identischen Hash-Adresse, so vorhanden, in dem verketteten Hash-Eintragsabschnitt **908** zeigt. Der nächste Eintrag wird an der nächsten verfügbaren Stelle in dem verketteten Hash-Eintragsabschnitt **908** gespeichert. Auf diese Weise wird, wenn zwei MAC-Adressen von zwei verschiedenen Vorrichtungen zu der gleichen Hash-Adresse haschieren, der erste oder "Haupt"-Eintrag in dem Haupt-Hash-Eintragsabschnitt **906** gespeichert, und der zweite Eintrag wird in dem verketteten Hash-Eintragsabschnitt **908** gespeichert, und die Link-Adresse des Haupteintrags zeigt auf den zweiten Eintrag. Wenn eine andere MAC-Adresse zu der gleichen Hash-Adresse wie die ersten zwei haschieren, wird jeder zusätzliche Eintrag im verketteten Hash-Eintragsabschnitt **908** gespeichert und in aufeinanderfolgender Reihenfolge oder mit Link-Adressen miteinander verbunden. Jeder Eintrag folgt dem Format des Hash-Tabelleneintrags **910**. Das Format der Link-Adresse kann in jeder genehmen Weise definiert werden. Die Link-Adresse enthält typischerweise einen Basisadressteil, der auf den Hash-Speicherabschnitt **902** im Speicher **212** zeigt, und einen Offsetteil, der auf den tatsächlichen Eintrag in dem Hash-Speicherabschnitt **902** zeigt. Die unteren Adressbits können, wenn gewünscht, zum Byteabgleich auf null gesetzt werden. Der letzte Eintrag in jeder Kette wird identifiziert, indem ein Teil der Link-Adresse auf null gesetzt wird. Zum Beispiel kann der letzte Eintrag bezeichnet werden, indem die Link-Adressenbits [A31:28] auf null gesetzt werden.

[0231] Fig. 9D ist ein Blockdiagramm, das die Organisation des Paketspeicherabschnitts **904** des Speichers **212** veranschaulicht. In der gezeigten Ausführung ist der Paketspeicherabschnitt **904** als eine Vielzahl von aneinandergrenzenden und gleich großen Sektoren **912** organisiert, wo jeder Sektor **912** einen Sektorinformationsabschnitt, genannt Sektorpräfix **914**, und einen Paketabschnitt **916** mit einem oder mehr Paketdatenblöcken umfasst. Jeder der Sektoren **912** hat vorzugsweise eine Größe von 2 KByte, um so der Seitengröße der Speichervorrichtungen, die den Speicher **212** implementieren, zu entsprechen, um Entwurf und Overhead zu vereinfachen. In der gezeigten Ausführung sind FPM DRAM SIMMs mit 4 KByte Seitengrenzen organisiert, und synchrone DRAM SIMMs sind in 2 KByte Seitengrenzen organisiert. Eine 2 KByte Sektorgröße ist daher für die unterstützten Speichervorrichtungstypen ausreichend. Die Sektoren **912** sind anfangs leer, aber mit Link-Adressen miteinander verkettet, um die FREEPOOL CHAIN von freien Speichersektoren zu bilden.

[0232] Wenn neue Informationspakete von jedem der Ports **104**, **110** empfangen werden, werden ein oder mehr Sektoren **912** von der FREEPOOL CHAIN getrennt und in einer RECEIVE SECTOR CHAIN pro Port miteinander verbunden. Ferner wird jedes Paket mit anderen Paketen in der gleichen oder anderen RECEIVE SECTOR CHAINs verbunden, um eine getrennte TRANSMIT PACKET CHAIN pro Port zu bilden. Auf diese Weise wird eine RECEIVE SECTOR CHAIN für einen Port auch in eine TRANSMIT PACKET CHAIN für einen anderen Port gelegt. Wenn alle Daten im Paketabschnitt **816** eines Sektors **912** an einen Zielport gesendet sind, wird der Sektor von seiner RECEIVE SECTOR CHAIN befreit und wieder mit der FREEPOOL CHAIN verbunden. Die RECEIVE SECTOR und FREEPOOL Chaims werden mittels Link-Adressen oder Zeigern von einem Sektor zu dem nächsten in einer unten weiter beschriebenen Weise implementiert. Alle TRANSMIT PACKET CHAINs werden von einem Paketdatenblock zu dem nächsten für jeden Port mittels Link-Adressen oder Zeigern miteinander verbunden, wie unten beschrieben.

[0233] Fig. 9E ist ein Diagramm, das die Organisation jedes der Sektorpräfixe **914** für jeden Sektor **912** des Paketspeicherabschnitts **904** zeigt. Das Sektorpräfix **914** enthält Information eines entsprechenden Sektors **912** und fungiert werter als ein Link zu einem nächsten Sektor **912**. Es wird angemerkt, dass, obwohl ein Präfix angegeben ist, dieser Informationsteil irgendwo in dem Sektor **912** platziert werden kann. Das erste Byte (0) definiert eine binäre Sektorpaketzählung (SecPktCnt), die die Zahl von Paketen oder Paketstücken im gegenwärtigen Sektor **912** angibt. Die Sektorpaketzählung wird inkrementiert, wenn Paketdaten in den Sektor ge-

speichert werden, und dekrementiert, wenn die Daten zum Senden durch den Zielport gelesen werden. Der Sektor wird an die FREEPOOL CHAIN zurückgegeben, wenn die Sektorpaketzählung SecPktCnt auf null dekrementiert, und wenn der Sektor nicht am Ende der RECEIVE SECTOR CHAIN liegt. Das nächste Byte (1) ist ein Sektorquellenwert (SecSource), der den Quellenport des empfangenen Pakets spezifiziert. Dieser Wert soll eine geeignete Empfangsport-Sektorzählung (RxSecCnt) identifizieren und dekrementieren, wenn der Sektor an die FREEPOOL CHAIN zurückgegeben wird. Die nächsten zwei Bytes (3:2) sind reserviert oder nicht benutzt.

[0234] Die nächsten vier Bytes (7:4) in jedem Sektorpräfix **914** bilden eine nächste Link-Adresse (NextSecLink) zum nächsten Sektor in einer entsprechenden RECEIVE SECTOR CHAIN oder FREEPOOL CHAIN. Die gleiche Link-Adresse wird für beide Zwecke verwendet, obwohl eine andere Link-Adresse auch benutzt werden könnte. In der gezeigten Ausführung ist die NextSecLink-Adresse 32 Bits, einschließlich Basis- und Offsetteilen. Die niedrigstwertigen "n" Bits können auf null gesetzt werden, um die Bytes der NextSecLink-Adresse entsprechend der Sektorgröße anzupassen. Die Ganzzahl "n" ist 12 für 4 KByte Sektoren, 11 für 2 KByte Sektoren, 10 für 1 KByte Sektoren und 9 für 512 Byte Sektoren. In der gezeigten Ausführung ist n 11 für 2 KByte Sektoren usw. Auf diese Weise wird, wenn ein oder mehr Pakete von einem Port **104**, **110** empfangen werden, eine RECEIVE SECTOR CHAIN von Sektoren **912** zugeteilt, um ein oder mehr durch diesen Port empfangene Pakete zu speichern. Die Sektoren **912** werden in einer Kettenform unter Verwendung der Next-Sec-Link-Adresse in dem Sektorpräfix **914** jedes Sektors **912** in der Kette miteinander verbunden. Die Paketdaten werden sequenziell im Paketabschnitt **916** jedes Sektors **912** in jeder RECEIVE SECTOR CHAIN gespeichert. Es wird angemerkt, dass Paketdaten für ein einzelnes Paket Sektorgrenzen in einer RECEIVE SECTOR CHAIN kreuzen können. Die letzten acht Bytes (15:8) des Sektorpräfixes **914** sind reserviert oder unbenutzt.

[0235] Fig. 9F ist ein Diagramm, das die Organisation eines exemplarischen Paketdatenblocks **917** zeigt, der jeden Paketdatenblock im Paketabschnitt **916** repräsentiert. Der Paketdatenblock **917** ist in zwei Teile geteilt, einen Paketblockvorspann **918** und einen Paketdatenabschnitt **920**. Der Paketblockvorspann **918** wird vorzugsweise jedem Paket durch den MCB **404** vorangestellt, um einen Paketdatenblock **917** zu bilden. Die ersten zwei Bytes (1:0) im Paketblockvorpsnn **918** bilden einen 15-Bit binären Paketlängen-(PktLength) Wert, der die Paketlänge in Bytes definiert, und einen 1-Bit Mittelpaket-CT-WERT (MidPktCT), der gesetzt wird, wenn ein CT-Moduspaket infolge eine stehen gebliebenen Ports an der Speicher **212** geleitet wird. Der MCB **404** hängt das erste DWORD, das die PktLength enthält, an das Paket an, wenn es an Ports Port24–Port27 für die TLANs **226** und an Port28 für die CPU **230** gesendet wird. Das nächste Byte (2) des Paketblockvorspanns **918** identifiziert die Quellenport-(SourcePort) Nummer des Pakets, die eine binäre 8-Bit Port-ID-Nummer ist, die die Nummer des mit der Quellenadresse verbundenen Ports identifiziert. Der Quellenport wird auch durch die einzelne RECEIVE SECTOR CHAIN identifiziert, in der das Paket gespeichert ist. Das nächste Byte (4) identifiziert die Zielport-(DestPort) Nummer, die eine binäre 8-Bit Port-ID-Nummer ist, die die Nummer des Zielport in einer ähnlichen Weise wie der SourcePort-Wert identifiziert. Der Ziel wird auch durch die einzelne TRANSMIT PACKET CHAIN identifiziert, zu der das Paket gehört.

[0236] Die Bytes (11:8) des Paketblockvorspanns **918** definieren eine 32-Bit nächste Link-Adresse (NextTxLink) zu dem nächsten Paket oder Paketdatenblock **917** in einer TRANSMIT PACKET CHAIN. Das Ende der TRAMSMIT PACKET CHAIN wird angezeigt, wenn eine Sendepaketzählung (TxPktCnt) auf null dekrementiert ist. Das niedrigstwertige Bit A0 der Next-TxLink-Adresse wird als ein BC-Paketbit (NextPktBC) benutzt, das angibt, ob das nächste Paket rundgesendet wird oder nicht. Wenn NextPktBC = 1, ist das nächste Paket im Rundsendeformat, unten beschrieben, und wenn NextPktBC = 0, ist das Nächste Nicht-Rundsenden. Das zweitniedrigstwertige Bit A1 der NextTxLink-Adresse wird als ein SnF-Paketbit (NextPktSnF) benutzt, das angibt, ob das nächste Paket SnF ist oder nicht. Es wird angemerkt, dass das niedrigstwertige Nibbel (vier Bits) der NextTxLink-Adresse für Byteabgleichzwecke ungeachtet des tatsächlichen Werte des Nibbels als null angenommen werden kann. Wenn z. B. die NextTxLink-Adresse gelesen wird, werden daher die Bits A[3:0] ungeachtet ihres tatsächlichen Werts, z. B. NextPktBC = 1, als null angenommen. Dies erlaubt diesen Bits, für andere Zwecke benutzt zu werden. In der gezeigten Ausführung werden die Datenstrukturen 16-Byte-ausgerichtet, sodass die niedrigstwertigen Bits A[3:0] als null angenommen werden.

[0237] In der gezeigten Ausführung folgt der Paketdatenabschnitt **920** sofort dem Paketblockvorspann **918**, wo die Länge des Datenfeldes im Paketvorspann definiert wird. Es wird jedoch angemerkt, dass die einzelne Reihenfolge jedes Sektors und die einzelnen Stellen von Werten in der gezeigten Ausführung willkürlich sind und der Veranschaulichung dienen, und daher in jeder gewünschten Weise organisiert werden können, ohne den Umfang der vorliegenden Erfindung zu verlassen.

[0238] Wie vorher beschrieben, werden Pakete aus jedem der Ports Port0–Port28 zurückgewonnen und in entsprechenden RECEIVE SECTOR CHAINs der Sektoren **912**, eine RECEIVE SECTOR CHAIN pro Port, gespeichert. Wie in Fig. 9H gezeigt, wird eine erste Empfangssektorkette **930** für Port0 gezeigt, wo ein erster Sektor **931** mit einem anderen Sektor **932** mittels des NextSecLink im Sektorpräfix **914** des Sektors **931** verbunden ist. Wertere Sektoren können, wenn gewünscht, mittels der Link-Adressen in den Sektorpräfixen **914** verbunden werden. Ferner wird eine zweite Empfangssektorkette **940** für Port1 gezeigt, wo ein erster Sektor **941** mit einem anderen Sektor **942** mittels des NextSecLink im Sektorpräfix **914** des Sektors **941** verbunden ist. Für jedes an einem gegebenen Port empfangene Paket wird der Paketblockvorspann **918** direkt hinter dem vorher empfangenen Paketdatenblock **917** in dem Paketabschnitt **916** des momentanen Sektors **912** der entsprechenden RECEIVE SECTOR CHAIN platziert, und der Paketblockvorspann **918** wird von seinem Paketdatenabschnitt **920** gefolgt. Wenn der Paketabschnitt **916** des momentanen Sektors **912** voll wird, während ein Paketdatenblock **917** gespeichert wird, wird ein weiterer Sektor **912** aus der FREEPOOL CHAIN zugewiesen und mit der RECEIVE SECTOR CHAIN für den Port verbunden. Auf diese Weise werden die von einem Port empfangenen Paketdatenblöcke **917** in der entsprechenden RECEIVE SECTOR CHAIN für diesen Port aneinandergrenzend platziert. Ferner kann der Paketabschnitt eines Sektors **912** ganze Pakete und/oder Paketteile enthaften.

[0239] Wie in Fig. 9H gezeigt, werden daher am Port0 empfangene Paketdatenblöcke **934**, **935** und **936** in den Sektoren **931** und **932**, wie gezeigt, platziert. Man beachte, dass der Paketdatenblock **935** die Sektoren **931** und **932** überspannt. In ähnlicher Weise werden am Port1 empfangene Paketdatenblöcke **944** und **945** in den Sektoren **941** und **942**, wie gezeigt, platziert, wo der Paketdatenblock **945** die Sektoren **941** und **942** überspannt.

[0240] Jedes Paket ist auch mit der TRANSMIT PACKET CHAIN von Paketen für jeden Zielport verbunden, wo die Pakete mittels der NextTxLink-Adresse miteinander verbunden sind. Pakete in jeder TRANSMIT PACKET CHAIN werden gewönlich basierend darauf geordnet, wann sie durch den Netzwerkschalter **102** empfangen werden, sodass die Reihenfolge bewahrt wird, wenn sie an den zugehörigen Zielport gesendet werden. Wenn z. B., wie in Fig. 9H gezeigt, die Paketdatenblöcke **934** und **944** vom Port10 zu senden sind, und der Paketdatenblock **934** direkt vor dem Paketdatenblock **944** zu senden ist, zeigt die NextTxLink-Adresse des Paketblockvorspanns **918** des Paketdatenblocks **934** auf den Paketdatenblock **944**. Die NextTxLink-Adresse des Paketblockvorspanns **918** des Paketdatenblocks **944** zeigt auf den als Nächstes zu sendenen Paketdatenblock usw. Die tatsächliche Reihenfolge zum Senden wird bestimmt, wenn ein Paket in eine TRANSMIT PACKET CHAIN eingebunden wird. CT-Moduspakete werden verkettet, wenn der Anfang des Pakets empfangen wird, und SnF-Moduspakete werden verkettet, nachdem das ganze Paket gespeichert ist. Mittelpaket-Interim-CT-Moduspakete werden mit dem Anfang der entsprechenden TRANSMIT PACKET CHIN verkettet, um die richtige Reihenfolge zu sichern.

[0241] Fig. 9G ist ein Blockdiagramm, das einen für BC-Pakete benutzten 128-Byte Paketvorspann **922** zeigt, der den normalen Paketblockvorspann **918** ersetzt. Für BC-Pakete wird der Next-PktBC-Wert im vorherigen Paket gesetzt, um anzuzeigen, dass das momentane Paket ein BC-Paket ist. Es wird angemerkt, dass jede TRANSMIT PACKET CHAIN für alle Ports unterhalten werden sollte, die das BC-Paket zum Senden enthalten. Der BC-Paketvorspann **922** enthält daher eine 4-Byte Link-Adresse (Port# NextTxLink) für jeden Port von Nummer **0**–**28** (einschließlich Ports **104**, **110** und CPU **230**), wo jede NextTxLink-Adresse auf das nächste Paket in der TRANSMIT PACKET CHAIH zeigt, die mit dem entsprechenden Port verbunden ist, der durch die Stelle in der Liste (Port#) identifiziert wird. NextTxLink-Adressen beginnen daher bei Bytes (11:8) und enden bei Bytes (123:120). Der erste NextTxLink-Adresseneintrag (11:8) entspricht dem nächsten Paket im Speicher **212** für den ersten Port Port0, der zweite Eintrag (15:12) ist eine NextTxLink-Adresse zum nächsten Paket im Speicher **212** für den zweiten Port Port1 usw. bis zum letzten Eintrag (Bytes 123:120), der ein NextTxLink zum nächsten Paket für die CPU **230** ist. Jede BC-Link Adresse enthält auch ein nächstes BC-Paket-(NextPktBC) Bit, das anzeigt, ob das nächste Paket in der betreffenden Sendepaketkette ein BC-Paket ist oder nicht, und ein nächstes SnF-Paket-(NextPktSnF) Bit, das anzeigt, ob das nächste Paket in der betreffenden Sendepaketkette ein SnF-Paket ist oder nicht.

[0242] Die ersten vier Bytes (3:0) des BC-Paketvorspanns **922** sind ähnlich den letzten vier Bytes des normalen Paketblockvorspanns **918**, einschließlich der Werte für PktLength, MidPktCT, SourcePort und DestPort, außer dass der MidPktCT-Wert für BC-Pakete null ist. Die nächsten vier Bytes (7:4) des BC-Paketvorspanns **922** ist eine Rundsendeport-Bitmap (BC_Ports), in der jedes der Bits 28:0 einem Port entspricht, der die BC-Paketdaten empfangen wird. Jedes Bit wird gelöscht, wenn das Paket an einen entsprechenden Port gesendet wird. Wenn alle BC_Ports-Bits gelöscht sind, wird die vorher beschriebene SecPktCnt-Zählung folglich auch dekrementiert.

[0243] Fig. 10 ist ein exemplarisches Blockdiagramm, das mehrere Sendepaketlinks zeigt, die jeweils das gleiche BC-Paket **1010** einschließen. In diesem Beispiel sind Ports **1**, **5**, **11** und **12** unter Verwendung der VLAN-Funktion oder dergleichen zusammengruppiert, sodass die Daten des an einem Quellenport, z. B. Port **12**, empfangenen BC-Pakets **1010** in die restlichen Ports **1**, **5** und **11** in dieser Gruppe dupliziert werden. Vier Sendepaketketten **1002**, **1004**, **1006** und **1008** werden für Ports **1**, **5**, **11** und **12** gezeigt. Die Sendepaketketten **1002**, **1004** und **1006** verbinden mehrere generische Nicht-Rundsendepakete **1000** mit dem BC-Paket **1010**. Da Port **12** der Quellenport ist, wird das BC-Paket **1010** auf Port **12** nicht gesendet, sodass es in der Sendepaketkette **1008** nicht enthalten ist. Das BC-Paket **1010** enthält einen BC-Paketvorspann **1012**, der eine Liste von Link-Adressen, eine für jeden Port, enthält, einschließlich einer Link-Adresse **1016**, die auf das nächste Paket **1000** in der Sendepaketkette **1002** von Port **1** zeigt, einer Link-Adresse **1018**, die auf das nächste Paket **1000** in der Sendepaketkette **1004** von Port **5** zeigt, und einer Link-Adresse **1002**, die auf das nächste Paket **1000** in der Sendepaketkette **1006** von Port **11** zeigt. Auf diese Weise wird jede der Sendepaketketten **1002**, **1004** und **1006** aufrechterhalten. Es wird auch angemerkt, dass jede Sendepaketkette ein oder mehr BC-Pakete enthaften kann, die, wie gewünscht, nicht-aufeinanderfolgend oder aufeinanderfolgend vorkommen können.

[0244] Fig. 11A ist ein Blockdiagramm, das MCB-Paketsteuerregister **1102** zeigt, wobei der Satz von Registern im SRAM **650** bereitgestellt und für jeden der 29 Ports **104**, **110**, einschließlich der CPU **230**, des Netzwerkschalters **102** dupliziert wird. Die CPU **230** wird als ein "Port" für bestimmte Zwecke behandelt, z. B. zum Senden und Empfangen von Brückenprotokoll-Dateneinheiten (BPDUs) zu Zwecken der Überspannungsbaum-Prozedur. Jedes MCB-Paketsteuerregister **1102** enthält einen Empfangsabschnitt **1104** und einen Sendeabschnitt **1106**. Im Empfangsabschnitt **1104** ist ein 28-Bit Empfangspaketvorspann-Basiszeiger (Rx-BasePtr) ein Zeiger auf die Basis des momentanen Empfangspaketvorspanns für den entsprechenden Port, der der Anfang der RECEIVE SECTOR CHAIN für diesen Port ist. Wie vorher für den Speicher **212** beschrieben, sind die Datenstrukturen für das SRAM **650** 16-Byte-ausgerichtet, sodass die niedrigstwertigen Bits A[3:0] aller Zeiger als null angenommen werden. Ein 28-Bit momentaner Empfangszeiger (RxCurPtr) ist ein Zeiger auf die momentane Datenspeicherstelle für die RECEIVE SECTOR CHAIN dieses Ports. Die niedrigstwertigen vier Bits des RxCurPtr-Werts sind Steuerbits, einschließlich eines Empfangs-BC-Paketanzeigebits (RxBC), eines Empfangsübertragung-im-Gange-(RxIP) Bits, das als ein Paketanfang-(SOP)Flag benutzt wird, eines Mehrsektorpaket-(MultisecPkt)Bits 1, das angibt. ob das momentane Paket eine Sektorgrenze kreuzt, und eines SnF-Bits 0, das anzeigt, dass der Sende-Link am Ende des Pakets aktualisiert wird. Der Empfangsabschnitt **1104** umfasst werter ein Mittelpaket-CT-Bit (MidCT), einen 16-Bit Empfangspaketlängen-(RxPktLn)Wert gleich der Länge des momentan empfangenen Pakets in Bytes bis zu dem RxCurPtr, eine 16-Bit Empfangsportsektorzählung (RxSecCnt), die die Zahl von gegenwärtig durch den entsprechenden Port benutzten Sektoren angibt, und einen 16-Bit Empfangssektorschwellen-(RxSecThreshold)Wert, der eine CPU-programmierte maximale Zahl von Sektoren identifiziert, die für jeden Port oder jede RECEIVE SECTOR CHAIN erlaubt ist. Der RxSecThreshold-Wert wird benutzt, um zu bestimmen, ob Rückstau für diesen Port anzuwenden ist, indem RxSecThreshold mit RcSecCnt verglichen wird. Wenn Rückstau unterbunden ist, wird der RXSecThreshold-Wert verwendet, um alle werteren an dem entsprechenden Port empfangenen Pakete fallen zu lassen.

[0245] Der Empfangsabschnitt **1104** umfasst weiter einen Sendeende-Queue-Zeiger (EndOfTx-QPtr), der ein 28-Bit Zeiger auf die Basis des letzten Pakets in der TRANSMIT PACKET CHAIN für den entsprechenden Port ist. Schließlich wird ein Sendeende-Queue-BC-(EOQ_BC) Bit gesetzt, um ein Rundsendeformat für das letzte Paket in der TRANSMIT PACKET CHAIN für den entsprechenden Port anzugeben.

[0246] Der Sendeabschnitt **1106** liefert Information für die TRANSMIT PACKET CHAIN für den entsprechenden Port. Ein Sendebasiszeiger (TxBasePtr) ist ein 28-Bit Zeiger auf die Basis des momentanen Sendepaketvorspanns, und ein anderer 28-Bit Momentan-Sendezeiger (TxCurPtr) zeigt auf die momentane Datenrückgewinnungsstelle für den entsprechenden Port. Ein Sende-Broadcast-(TxBC)Bit wird gesetzt, um anzuzeigen, dass der Paketvorspann im Rundsendeformat ist. Ein Senden-im-Gange-(TxIP)Bit wird auf logisch 1 gesetzt, um anzuzeigen, dass ein Senden momentan für den Port im Gange ist, und wird benutzt, SPO anzuzeigen. Eine 8-Bit Sendequellenport-(TxSrcPort)Nummer ist die Quellenportnummer des momentanen Sendepakets, die bei SOP aus dem Paketvorspann gelesen wird. Ein 16-Bit Sendepaketlängen-(TxPktLn)Wert ist gleich den restlichen zu sendenden Bytes für das momentane Sendepaket. Wenn ein Paket zu senden ist, wird der Pkt-Length-Wert im Paketblockvorspann **918** des Pakets in den TxPktLn-Wert im Sendeabschnitt **1106** kopiert, und dann wird der TxPktLn-Wert durch die TX-Steuerung **606** dekrementiert, wenn das Paket gesendet wird. Wenn der TxPktLn auf null dekrementiert ist, erzeugt der EPSM **210** das entsprechende EOP* Signal, um das Ende des Pakets anzuzeigen. Ein 16-Bit Maximal-Paketzahl-(TxPktThreshold)Wert ist gleich der CPU-programmierten maximalen Zahl von Paketen, die für jeden Port in einer Warteschlange eingereiht werden darf. Es wird angemerkt, dass für die CPU **230** bestimmte Pakete nicht der TxPktThreshold- oder TxPktThreshold-Grenze

unterliegen. Schließlich ist eine 16-Bit Sendepaketzählung (TxPktCnt) gleich der Zahl von Paketen, die momentan für den entsprechenden Port in einer Warteschlange eingereiht sind.

[0247] Fig. 11B ist ein Blockdiagramm, das in dem SRAM **650** gelegene Freepool-Paketsteuerregister **1108** zeigt, die mit der FREEPOOL CHAIN von Registern verbunden sind. Jedes Freepool-Register **1108** enthält einen Zeiger (NextFreeSecPtr) auf den nächsten freien Sektor in der FREEPOOL CHAIN, einen Zeiger (LastFreeSecPtr) auf den letzten Sektor in der FREEPOOL CHAIN, eine freie Sektor Zählung (FreeSecCnt) gleich der Zahl von momentan verfügbaren freien Sektoren, einen Freisektor-Schwellen-(FreeSecThreshold)Wert gleich der CPU-programmierten Mindestzahl von Sektoren, die erlaubt ist, bevor ein Speicherüberlauf-Flag (MOF) für Rückstau- oder Filterungs-(Pakete fallen lassen)Zwecke gesetzt wird, eine BC-Paketzählung (BC_PktCnt) gleich der Zahl von BC-Paketen, die momentan im Speicher **212** sind, und eine BC-Paketschwellen-(BC_PKThreshold) Zählung gleich einer CPU-programmieren Maximalzahl im Speicher **212** erlaubter BC-Pakete.

[0248] Fig. 12A ist ein Flussdiagramm, das die Arbeitsweise des Netzwerkschalters **102** zum Empfangen von Datenpaketen in dem Speicher **212** und zum Senden von Datenpaketen in der CT-Betriebsart veranschaulicht. Daten werden typischerweise durch die Ports Port0–Port27 des Netzwerkschalters **102** in der Form von Paketen in Echtzeit oder in ihrer Gesamtheit empfangen und gesendet und werden nicht unterteilt während sie über die Segmente **108**, **114** gesendet werden. Die FIFOs im Netzwerkschalter **102** sind jedoch typischerweise nicht groß genug, um ein ganzes Paket zu speichern. Paketdaten werden daher im Netzwerkschalter **102** von einem FIFO zu einem anderen in Paketabschnitten oder Unterteilungen von Paketen übertragen.

[0249] In einem ersten Schritt **1200** erfasst der EPSM **210** ein neues Paket, das durch einen der Ports **104**, **110** gesendet wird, durch Anzeigen der PKT_AVAILm* Signale. Im nächsten Schritt **1202** wird der Anfangsteil oder Vorspann des Pakets aus dem Quellenport zurückgewonnen und in die HASH REQ LOGIC **532** gelesen, wo der Vorspann die Ziel- und Quellen-MAC-Adressen enthält. Die HASH REQ LOGIC **532** stellt die Ziel- und Quellenadressen und die Quellenportnummer auf den HASH_DA_SA[15:0] Signalen bereit und macht das HASH_REQ* Signal am MCB **404** geltend. Der MCB **404** ruft als Reaktion die die Hashing-Prozedur zum Bestimmen der geeigneten Aktion für das Paket auf, wo die Quellen- und Zieladressen haschiert werden, um zu bestimmen, ob jede der Adressen vorher im Speicher **212** gespeicher wurde. Der MCB **404** macht das HASH_DONE* Signal geltend, wenn genug Information für den HCB **404** verfügbar ist, um die für das Paket geeignete zu ergreifenden Aktion zu bestimmen. Das in Fig. 12A gezeigte Flussdiagramm umfasst zwei Hauptabschnitte für die Ziel- und die Quellenadressen, die getrennt erörtert werden. In der gezeigten Ausführung wird zuerst die Zieladresse haschiert, gefolgt von der Quellenadresse, obwohl die Prozeduren gleichzeitig oder in jeder gewünschten Reihenfolge durchgeführt werden können.

[0250] Für die Zieladresse geht der Vorgang zu Schritt **1204**, wo die Haschierungsprozedur aufgerufen wird, um die Zieladresse zu haschieren. Der Vorgang geht als Reaktion auf das HASH_DONE* Signal von Schritt **1204** zu Schritt **1208**, um die Schwellenbedingung für Unicast- und BC-Pakete zu prüfen. In Schritt **1208** wird festgestellt, ob eine relevante Schwellenbedingung durch das neue Paket verletzt werden würde. Das heißt, wenn die FreSecCnt-Zahl gleich oder kleiner ist als die FreeSecThreshold-Zahl, kann nicht genug Platz vorhanden sein, um das Paket im Speicher **212** zu speichern. Ferner, wenn die RxSecCnt-Zahl größer oder gleich der RxSecThreshold-Zahl ist, kann der Netzwerkschalter **102** bestimmen, das Paket fallen zu lassen. Für BC-Pakete wird die BC_PktThreshold-Zahl mit der BC_Pkt-Cnt-Zahl, die die tatsächlichen Zahl von BC-Paketen ist, verglichen, um festzustellen, ob die Maximalzahl von BC-Paketen bereits empfangen wurde. Für Unicast-Pakete wird die TxSec-Threshold-Zahl mit der TxSecCnt-Zahl für den Zielport verglichen.

[0251] Von Schritt **1208** geht der Vorgang zu Schritt **1205**, wo der HCB **404** aus den HASH_STATUS[1:0] Signalen und aus dem Vergleichen von jeder der Schwellenbedingungen bestimmt, ob das Paket fallen zu lassen ist. Das Paket kann aus einer Vielfalt anderer Gründe, wie vorher beschrieben, fallen gelassen werden, z. B., wenn die Quellen- und Zielports gleich sind. Wenn das Paket fallen zu lassen ist, geht der Vorgang von Schritt **1205** zu Schritt **1207**, wo das Paket entweder fallen gelassen oder Rückstau angewandt wird. Rückstau wird angewandt, wenn die FreeSecThreshold- oder RxSecThreshold-Bedingungen verletzt werden, und wenn Rückstau freigegeben ist und der Quellenport im Halbduplex-Modus arbeitet. Andernfalls wird das Paket fallen gelassen. Für Rückstau führt der EPSM **210** einen Rückstau-Zyklus auf dem HSB **206** aus, was den Quellenport veranlasst, eine Hemmungssequenz in der sendenden Vorrichtung geltend zu machen. Das Paket wird fallen gelassen, wenn die Rückstau-Anzeige durch den Quellenport nicht angenommen wird (wie durch das ABORT_OUT* Signal angezeigt), weil sie zu spät bereitgestellt wird, um die Hemmungssequenz geltend zu machen. Ferner wird das Paket fallen gelassen, wenn die BC_PktThreshold-Bedingung die einzige Schwellenbedingung ist, die verletzt wird. Der Netzwerkschalter **102** fährt fort, den Rest des fallen gelassenen Pakets zu

empfangen, aber das Paket wird nicht gespeichert oder an einen anderen Port gesendet. Von Schritt **1207** geht der Vorgang zu Schritt **1214**, wo die geeigneten Statistikregister in den MCB-Konfigurationsregistern **448** basierend auf der in Schritt **1207** ergriffenen Aktion aktualisiert werden. Die Statistikregister zeigen an, ob das Paket infolge von Überlaufbedingungen fallen gelassen oder rückgestaut wurde. Zum Beispiel wird eine pro Port "fallen gelassenes Paket – kein Puffer" Zählung für den Port inkrementiert, um anzuzeigen, dass ein Paket infolge von Überlaufbedingungen fallen gelassen wird, oder eine "Paket zurückgestaut" Zählung wird inkrementiert, wenn das Paket rückgestaut wird.

[0252] Wenn das Paket nicht fallen gelassen wird, geht der Vorgang von Schritt **1205** zu Schritt **1206**, wo festgestellt wird, ob die Zieladresse im Hash-Speicherabschnitt **902** gefunden wurde, und ob das Paket rundzusenden ist oder nicht. Das Paket wird rundgesendet, wenn die Zieladresse nicht erkannt wird und daher der Zielport nicht bekannt ist, oder wenn das GROUP-Bit in dem Paket gesetzt ist. Wenn die Zieladresse nicht gefunden wird, oder wenn das Paket andernfalls ein BC-Paket ist, wie in Schritt **1206** bestimmt, ist das Paket rundzusenden, und der Vorgang geht zu Schritt **1210**, wo der MCB **404** des EPSM **210**, wenn nötig, einen weiteren Sektor im Speicher **212** für das neue Paket zuteilt. Ein neuer Sektor ist nicht nötig, wenn der gegenwärtige Sektor genug Platz für das Paket hat. Der Vorgang geht zu Schritt **1216**, der angibt, dass der Rest des Pakets, Stoß für Stoß, durch den EPSM **210** gepuffert und in den Speicher **212** übertragen wird. Ungeachetet Porteinstellungen werden BC-Pakete mit SnF-Modus gehandhabt, wo das ganze Paket im Speicher **212** gespeichert wird, bevor es gesendet wird. Von Schritt **1216** geht der Vorgang zu Schritt **1217**, um festzustellen, ob das ABORT_OUT* Signal während des Empfangens des Pakets infolge eines Paketfehlers geltend gemacht wurde. Mehrere Fehlerbedingungen werden durch die Ports Port0–Port27 geprüft, z. B. Erfassen eines FIFO-Überlaufs, eines Runt-Pakets, eines übergroßen Pakets, das Paket hatte eine schlechte FCS (Rahmenprüfsequenz) oder ein PLL-Fehler wurde erfasst. Wenn in Schritt **1217** ein Paketfehler erfasst wird, geht der Vorgang zu Schritt **1219**, wo das Paket aus dem Speicher **212** entfernt wird.

[0253] Wenn in Schritt **1217** keine Paketfehler erfasst werden, geht der Vorgang zu Schritt **1218**, wo die Rundsendeport-Bitmap BC_Ports im Paketvorspann **922** des BC-Pakets mit den aktiven Ports, von denen das BC-Paket zu senden ist, aktualisiert wird. Das BC-Paket wird an alle Ports außer den folgenden Ports gesendet: der Quellenport; jeder Port, der nicht im FORWARDING-Status ist, wenn der Quellenport die CPU **230** ist, oder jeder Port im DISABLED-Status, wenn der Quellenport die CPU **230** ist, und alle Ports mit einer TxPktCnt-Zahl größer oder gleich der entsprechenden TxPktThreshold-Zahl. Wenn VLAN freigegeben ist, wird auch der VLAN-Bitmapwert im Hash-Tabelleneintrag **910** untersucht, was die Ports weiter auf aktive, zugehörige Ports in der VLAN-Gruppe begrenzt. Ferner werden Fehl-BC-Pakete, wo das Paket infolge einer unbekannten Zieladresse rundgesendet wird, entsprechend einem MissBCBitMap-Register befördert. Es wird angemerkt, dass, wenn die resultierende BC_Ports-Bitmap alles Nullen sind, sodass das Paket an keinen Port zu senden ist, diese Entscheidung entweder in Schritt **1205** getroffen und das Paket in Schritt **1207** fallen gelassen wird, oder das Paket in Schritt **1218** aus dem Speicher **212** entfernt wird.

[0254] Der Vorgang geht von Schritt **1218** zu Schritt **1220**, wo das Paket zu der TRANSMIT PACKET CHAIN für jeden Port in der resultierenden BC_port-Bitmap hinzugefügt wird. Das heißt, jede der NextTxLink-Link-Adressen für jeden in der BC_port-Bitmap bezeichneten Port im Paketvorspann **922** wird aktualisiert, um das BC-Paket in die TRANSMIT PACKET CHAINs der geeigneten Ports einzufügen. Alle anderen zugehörigen Register- oder Zählwerte und Statistiken im Netzwerkschalter **102** werden folglich auch aktualisiert, z. B. die BC_PktCnt-Zählung.

[0255] Zurück auf Schritt **1206** verweisend geht, wenn die Zieladresse gefunden ist und das Paket kein BC-Paket ist, der Vorgang zu Schritt **1222**, wo die Hash-Cache-Tabelle **603** aktualisiert wird. Der Vorgang geht dann zum nächsten Schritt **1224**, wo abgefragt wird, ob entweder der Quellenport oder der Zielport für den SnF-Modus eingerichtet ist. Wenn beide Ports für den CT-Modus eingerichtet sind und die anderen CT-Bedingungen erfüllt sind, z. B. gleiche Portgeschwindigkeit, und die TBUS-Einstellung für den Zielport gleich der TBUS-Einstellung für den Quellenport ist, geht der Vorgang zu Schritt **1225**, wo abgefragt wird, ob der Zielport tätig ist. Wenn der Vorgang für den SnF-Modus bezeichnet ist, wie in Schritt **1224** bestimmt, oder wenn für den CT-Modus bezeichnet, aber der Zielport tätig ist, wie in Schritt **1225** bestimmt, sodas der Interim-CT-Modus eingeleitet wird, geht der Vorgang zu Schritt **1226**, wo der MCB **404** des EPSM **210**, wenn nötig, Platz im Speicher **212** für das neue Paket zuweist. Von Schritt **1226** geht der Vorgang zu Schritt **1228**, wo der restliche Teil des Pakets in den EPSM **210** zurückgewonnen und in den Speicher **212** übertragen wird. Wenn ein Paketfehler während des Empfangs des Pakets auftritt, wie in Schritt **1229**, der dem Schritt **1217** ähnlich ist, angedeutet, geht der Vorgang zu Schritt **1219**, um das Paket aus dem Speicher **212** zu entfernen. Andernfalls geht der Vorgang zu Schritt **1230**, wo das Paket zu der TRANSMIT PACKET CHAIN des Zielports hinzugefügt wird und die geeigneten Link-Adressen, Zählungen und CHAINs aktualisiert werden.

[0256] Wenn, wieder auf Schritt **1225** verweisend, der Zielport nicht tätig ist, geht der Vorgang zu Schritt **1231**, wo die Quellen- und Zielports für normalen CT-Betrieb für das momentane Paket bezeichnet werden. Für den normalen CT-Modus wird jeder restliche Paketteil nicht an den Speicher **212** gesendet, sondern wird stattdessen durch den CT BUF **528** zu dem Zielport gepuffert. Der Vorspann des Pakets wird von dem RX FIFO des EPSM **210** direkt zu dem Zielport übertragen. Der nächste Schritt **1232** gibt das Empfangen von Datenpaketteilen im CT BUF **528** und das Übertragen der Paketteile zu dem Zielport an. Während des CT-Betriebs fragt der nächste Schritt **1233**, ob der Zielport oder -Weg tätig oder unverfügbar wird. Die in Schritt **1233** angegebene Abfrage wird ausgeführt, bevor Daten im CT BUF **528** durch den Haupt-Arbiter **512** empfangen werden. Während der Zielport für mehr Daten verfügbar bleibt, verzweigt der Vorgang zu Schritt **1234** um zu fragen, ob das ganze Paket zu dem Zielport übertragen wurde, und geht, wenn nicht, zurück zu Schritt **1232**, um mehr Daten zu senden. Wenn das ganze Paket im CT-Modus übertragen wurde, wie in Schritt **1234** festgestellt, ist der Vorgang für dieses Paket abgeschlossen.

[0257] Wenn der Zielport während der normalen CT-Modusübertragung tätig oder unverfügbar wird, wie in Schritt **1233** festgestellt, geht der Vorgang zu Schritt **1235**, um den restlichen Teil des Pakets im Speicher **212** zu empfangen, um den Mittelpaket-Interim-CT-Modus einzuleiten. Während des Mittelpaket-Interim-CT-Modus wird der restliche Teil des Pakets durch den Speicher **212** gepuffert. Da das Paket mitten in der Übertragung war, werden die restlichen an den Speicher **212** gesendeten Paketdaten am Anfang der TRANSMIT PACKET CHAIN für diesen Port platziert, um die richtige Paketreihenfolge zu sichern, wie im nächsten Schritt **1236** angegeben. Wie in der normalen CT-Betriebsart ist jeder an den Speicher **212** während des Mittelpaket-Interim-CT-Modus gelieferte Datenpaketteil zur Übertragung zu dem Zielport, sobald empfangen, verfügbar.

[0258] Wieder auf Schritt **1202** verweisend geht der Vorgang zu Schritt **1240**, um die Quellenadresse zu haschieren. Der Vorgang geht dann zu Schritt **1242**, wo abgefragt wird, ob die Quellenadresse im Hash-Speicherabschnitt **902** gefunden wurde, und ob das GROUP-Bit im Paket gesetzt wurde. Wenn die Quellenadresse gefunden wurde und das GROUP-Bit nicht gesetzt war, geht der Vorgang zu Schritt **1244**, wo das AGE-Feld des Hash-Speicherabschnitts **902** mit der AGE-Information aktualisiert wird. Zum Beispiel wird der AGE-Wert auf null gesetzt. Es wird angemerkt, dass die Quellen-MAC-Adresse und die Quellenportnummer nicht mehr einem vorherigen Eintrag entsprechen können. Dies könnte z. B. vorkommen, wenn eine Netzwerk- oder Datenvorrichtung von einem Port zu einem anderen bewegt wird. Diese Information wird in Schritt **1244** verglichen und aktualisiert.

[0259] Wenn, zurück auf Schritt **1242** verweisend, die Quellenadresse nicht gefunden wurde, oder wenn das GROUP-Bit gesetzt war, geht der Vorgang zu Schritt **1246**, wo eine Unterbrechung an der CPU **230** erzeugt wird, die die folgenden Schritte durchführt. In Schritt **1248** weist die CPU **230** einen Hash-Tabelleneintrag im Hash-Speicherabschnitt **902** des Speichers **212** oder einen am wenigsten kürzlich benutzten (LRU) Abschnitt der Hash-Cache-Tabelle **603** für die neue Quellenadresse zu. Der Vorgang geht dann zu Schritt **1250**, wo die Werte in dem zugewiesenen Hash-Eintrag, z. B. die Quellen-MAC-Adresse, die Quellenportnummer und die AGE-Information, aktualisiert werden.

[0260] Fig. 12B ist ein vereinfachtes Flussdiagram, das die allgemeine Arbeitsweise des Netzwerkschalters **102** zum Übertragen von Daten aus dem Speicher **212** zu einem oder mehreren Zielports veranschaulicht. Die Übertragungsprozedur gilt allgemein für SnF- und Mittelpaket-Interim-CT-Betriebsarten und für BC-Pakete, wie unten qualifiziert. Ein erster Schritt **1260** stellt allgemein dar, dass Paketdaten entsprechend den vorher beschriebenen Prozeduren im Speicher **212** in einer Warteschlange eingereiht sind. Der Vorgang geht dann zum nächsten Schritt **1262**, wo der MCB **404** dem HCB **402** anzeigt, dass Paketdaten verfügbar sind.

[0261] Für den Mittelpaket-Interim-CT-Modus wird diese Anzeige bereitgestellt, sobald das erste DWORD von Daten an den MCB **404** zur Speicherung im Speicher **212** gesendet wird, da die Daten fast sofort zur Übertragung an einen Zielport zur Verfügung stehen. Für den SnF-Modus wird diese Anzeige jedoch nur bereitgestellt, nachdem das letzte DWORD von Daten für ein Datenpaket an den MCB **404** gesendet ist, da das ganze Paket vor der Übertragung gespeichert wird. Sobald Paketdaten zum Senden verfügbar sind, geht der Vorgang zu Schritt **1264**, wo festgestellt wird, ob der Zielport Pufferplatz zur Verfügung hat, um Paketdaten zum Senden zu empfangen. Schritt **1264** stellt allgemein die durch den EPSM **210** durchgeführte Abfrageprozedur zum Abfragen jedes der Ports **104**, **110** dar, die mit entsprechenden BUF_AVAILm* Signalen antworten, wie vorher beschrieben. Der Vorgang bleibt bei Schritt **1264**, bis der Zielport anzeigt, dass er Pufferplatz zur Verfügung hat, um Paketdaten zu empfangen.

[0262] Wenn der Zielport in Schritt **1264** anzeigt, dass er Pufferplatz besitzt, geht der Vorgang zu Schritt **1266**, wo der HCB **402** die Übertragung von Daten für den Zielport anfordert. Im nächsten Schritt **1268** wird ein Stoß

von Daten vom Speicher **212** an den Zielport zum Senden übertragen. Der Vorgang geht zum nächsten Schritt **1270**, wo abgefragt wird, ob alle Daten im Speicher **212** an den Zielport übertragen wurden. Wenn nicht, geht der Vorgang zu Schritt **1264**, um zu warten, bis der Zielport mehr Pufferplatz für eine weitere Datenübertragung zur Verfügung hat. Schließlich wird das ganze Datenpaket, im Fall des SnF- und Interim-CT-Modus, oder die restlichen Paketdaten im Fall des Mittelpaket-Interim-CT-Modus übertragen, wie in Schritt **1270** festgestellt.

[0263] Der Vorgang geht dann zu Schritt **1272**, wo festgestellt wird, ob das Paket ein BC-Paket ist oder nicht. Wenn das Paket ein BC-Paket ist, geht der Vorgang zu Schritt **1274**, um festzustellen, ob das ganze Paket an alle aktiven Ports übertragen wurde. Wenn nicht, wird der Vorgang für das momentane Paket vollendet. Die Prozedur wird für jeden Port wiederholt, bis das Paket an alle aktiven Ports übertragen ist. Es wird angemerkt, dass die Schritte **1272** und **1274** gezeigt werden, um darzustellen, dass die Schritte **1264** bis **1270** für jeden Zielport für jedes BC-Paket durchgeführt werden. Das ganze BC-Datenpaket bleibt daher im Speicher **212**, bis es an alle aktiven Zielports zum Übertragen gesendet ist. Wenn das Paket kein BC-Paket ist, oder nachdem das ganze Paket an alle aktiven Ports für BC-Pakete gesendet ist, wie in Schritt **1274** angegeben, geht der Vorgang zu Schritt **1276**, wo der Pufferplatz im Speicher **212**, der das BC-Paket hält, freigesetzt wird. Das heißt, die Sektoren, die die Paketdaten halten, werden an die FREEPOOL CHAIN von freien Speichersektoren im Speicher **212** zurückgegeben.

[0264] Fig. 13 ist ein Flussdiagramm, das den Hash-Nachseh-Vorgang des EPSM **210** veranschaulicht. Die Schritte im Flussdiagramm von Fig. 13 werden durch den MCB **404** ausgeführt. Ein Anfangsschritt **1302** erfasst eine Hash-Anforderung, wie durch Geltendmachung des HASH_REQ* Signals angezeigt. Der HCB **402** identifiziert den Vorspann des Pakets als ein neues Paket, bestimmt die Quellen- und Zieladressen und die Quellenportnummer und macht die HASH_DA_SA[15:0] Signale an der Hash-Steuerung **602** des MCB **404** geltend. Der MCB **404** gewinnt dann die Quellen- und Ziel-MAC-Adressen und die Quellenportnummer zurück und führt die Haschierungs-Prozedur durch, die die geeignete Aktion für das Paket bestimmt.

[0265] Der MCB **404** ergreift gewöhnlich eine von vier Aktionen, wobei jede Aktion auf der Quellenportnummer und der Quellen- und Ziel-MAC-Adresse basiert. Das heißt, die Hash-Steuerung **602** bestimmt die HASH_STATUS[1:0] Signale, die auf FORWARD_PKT, um das Paket an den Zielport zu befördern, DROP_PKT, um das Paket fallen zu lassen und zu ignorieren, MISS_BC, wenn die Ziel-MAC-Adresse neu und unbekannt ist, sodass das Paket an alle anderen Ports rundgesendet wird, oder GROUP_BC gesetzt werden, wenn das Paket in eine Untermenge verbundener Ports zu duplizieren und durch diese zu senden ist. Von Schritt **1302** geht der Vorgang zu Schritt **1304**, um festzustellen, ob das Paket fallen zu lassen ist, was durch die folgende Gleichung (1) bestimmt wird:

DropPkt := (SrcState = DIS) oder (!FilterHit & SrcState != FWD) (1)

wo SrcState den Überspannungsbaum-Zustand des Quellenports identifiziert, FilterHit ein Bit ist, das gesetzt wird, wenn die Quellen-MAC-Adresse in einen vorbestimmten Bereich fällt, das Etzeichen "&" die logische UND-Operation darstellt, das Ausrufezeichen "!" eine logische Verneinung bezeichnet, das Symbol "!=" die Funktion "nicht gleich" bezeichnet, und das Symbol ":=" die Funktion "setzen gleich" bezeichnet. Jeder Port hat einen von fünf in den HSB-Konfigurationsregistern **448** bereitgestellten Zuständen, und wie durch die Überspannungsbaum-Funktion der IEEE 802.1 Spezifikation bestimmt, einschließlich Lernen (LRN), Befördern (FWD), Gesperrt (BLK), Hören (LST) und Abgeschaltet (DIS). In der gezeigten Ausführung werden die Zustände BLK UND LST gleich behandelt. Das Paket wird daher fallen gelassen, wenn der Quellenport abgeschaltet ist, oder wenn die Quellen-MAC-Adresse nicht in dem vorbestimmten Filterbereich liegt und der Zustand des Quellenports nicht Befördernd ist.

[0266] Wenn DropPkt wahr ist, wie in Schritt **1304** bestimmt, geht der Vorgang zu Schritt **1305**, wo HASH_STATUS[1:0] Signale auf 00b = DROP_PKT gesetzt werden, um den HCB **402** anzuweisen, das Paket zu ignorieren oder andernfalls fallen zu lassen. Wenn DropPkt unwahr ist, geht der Vorgang zu Schritt **306**, wo das FilterHit-Bis untersucht wird, um festzustellen, ob die Quellen-MAC-Adresse in dem vorbestimmten Bereich liegt. Der vorbestimmte Bereich identifiziert Pakete, die von der CPU **230** stammen oder dafür bestimmt sind, einschließlich Brückenprotokoll-Dateneinheiten (BPDUs), die an die CPU **230** gesendet werden. Wenn FilterHit wahr ist, wie in Schritt **1306** bestimmt, geht der Vorgang zu Schritt **1308**, um den Zielport (DstPrt) zu identifizieren. Wenn das Paket von der CPU **230** kommt (SrcPrt = CPU=, wird der Zielport gleich einem durch die CPU **230** in einem vorherigen Vorgang (Dst := FItrPrt) gesetzten Wert FItrPrt gesetzt. Andernfalls wird das Paket an die CPU **230** gesendet (DstPrt := PORT28). Der Vorgang geht dann von Schritt **1308** zu Schritt **1310**, um nach der folgenden Formel (2) zu bestimmen, ob das Paket zu befördern (FwdPkt) ist.

FwdPkt := (DstPrt != SrcPrt) & ((DstState = FWD) oder (SrcPrt = CPU & DstState != DIS)) (2)

wo DstState der Überspannungsbaum-Zustand des Zielports (DstPrt) ist und "&" die logische UND-Operation bezeichnet. Das Paket wird daher an den Zielport befördert, wenn der Ziel- und Quellenport nicht der gleiche nicht, und wenn der Status des Zielports Befördern ist, oder wenn der Quellenport die CPU **230** und der Status des Zielports nicht Abgeschattet ist. Der Zielport ist auch ohne Hash-Nachsehen bekannt, da er entweder die CPU **230** ist oder durch die CPU **230** als FItrPrt bestimmt wird. Wenn FwdPkt unwahr ist, geht der Vorgang zu Schritt **1305**, um das Paket fallen zu lassen. Andernfalls, wenn FwdPkt wahr ist, geht der Vorgang zu **1312**, wo die HASH_STATUS[1:0] Signale auf 11b = FORWARD_PKT gesetzt werden, um anzuzeigen, dass das Paket an den Zielport zu berfördern ist. Ferner werden die HASH_DSTPRT[4:0] Signale mit der DstPrt-Zielportnummer geltend gemacht.

[0267] Wenn, wieder auf Schritt **1306** verweisend, die Quellenandresse nicht in dem vorbestimmten Bereich und daher außerhalb der gefilterten MAC-Adressen liegt, geht der Vorgang zu Schritt **1314**, um das GROUP-Bit in dem empfangenen Paket zu untersuchen, das angibt, ob das Paket ein BC-Paket ist oder nicht. Wenn GROUP unwahr ist (GROUP-Bit = logisch 0), geht der Vorgang zu Schritt **1316**, um ein Hash-Nachsehen der Ziel-MAC-Adresse (DA) durchzuführen. Die MAC-Adresse wird zuerst haschiert, indem zwei verschiedene Sätze von Bits aus der Adresse genommen werden die zwei Sätze Bit für Bit logisch kombiniert oder verglichen werden, um eine entsprechende 13–16 Bit Hash-Adressee zu bilden, wie vorher beschrieben. Alle Bits der MAC-Adresse können für die Zwecke der Haschierungs-Prozedur gewählt werden. Die tatsächliche Nachschlag-Prozedur wird von einer getrennten Routine oder Funktion durchgeführt, die unten mit Verweis auf das Flussdiagramm von **Fig. 14** beschrieben wird.

[0268] Die Nachschlag-Prozedur in Schritt **1316** gibt, wenn gewünscht, einen oder mehr Werte zurück, einschließlich eines HIT bezeichneten Bits, das als DA_Hit für die Zieladresses oder SA_Hit für die Quellenadresse zurückgegeben wird. Das HIT-Bit bestimmt, ob die haschierte Adresse im Hash-Speicherabschnitt **902** gefunden wurde. Von Schritt **1316** geht der Vorgang zu Schritt **1318**, wo der DA_Hit-Wert untersucht wird, um festzustellen, ob die Adresse gefunden wurde oder nicht. Die Adresse wird im Speicher **212** gefunden werden, wenn die der Ziel-MAC-Adresse entsprechende Vorrichtung vorher ein Paket hervorgebracht hat. Wenn DA_Hit wahr ist, geht der Vorgang zu Schritt **1310**, um festzustellen, ob das Paket zu befördern ist, wie vorher beschrieben. Wenn die Hash-Adresse nicht gefunden wurde und DA_Hit unwahr ist, geht der Vorgang zu Schritt **1320**, wo die HASH_STATUS[1:0] Signale auf 10b = MISS_BC gesetzt werden, um eine neue MAC-Adresse anzuzeigen. Da die mit der Zielvorrichtung verbundene Portnummer noch nicht bekannt ist, wird das Paket an alle anderen aktiven (und wie durch VLAN und andere Logik qualifizierte) Ports rundgesendet, um sicherzustellen, dass das Paket an die geeignete Zielvorrichtung gesendet wird. Schließlich antwortet die Zielvorrichtung auf das Paket mit einem neuen Paket, das die gleiche MAC-Adresse als eine Quellenadresse enthält. Der Netzwerkschalter **102** ist dann in der Lage, die MAC-Adresse mit einem Port und einer Portnummer zu verbinden und den Hash-Speicher abschnitt **902** entsprechend zu aktualisieren. Wenn, wieder auf Schritt **1314** verweisend, das GROUP-Bit wahr (oder logisch 1) ist, geht der Vorgang zu Schritt **1322**, wo die HASH_STATUS[1:0] Signale auf 01b = GROUP_BC gesetzt werden, um anzuzeigen, dass das Paket an alle anderen Ports oder an eine durch die VLAN-Funktion spezifizierte Gruppe von Ports rundzusenden ist.

[0269] Von jedem der Schritte **1305**, **1312**, **1320** oder **1322** geht der Vorgang zu Schritt **1324**, um durch Untersuchen eines SrcLookUp-Wertes festzustellen, ob der Hash-Speicherabschnit **902** nach der Quellen-MAC-Adresse abzusuchen ist. Der SrcLookUp-Wert wird nach der folgenden Gleichung (3) bestimmt:

SrcLookUp := (SrcState = (LRN oder FWD)) & SrcPrt != CPU (3)

die anzeigt, dass die MAC-Quellenadresse gesucht wird, wenn der Quellenport im Lernen- oder Befördern-Modus ist und nicht die CPU **230** ist. Wenn SrcLookUp wahr oder gesetzt ist, wie in Schritt **1324** bestimmt, geht der Vorgang zu Schritt **1326**, wo zwei Werte VLAN und SecurePort untersucht werden. Das VLAN-Bit ist wahr, wenn einer der VLAN-Modi freigegeben ist, aber andernfalls unwahr. SecurePort ist wahr, wenn der Quellenport sicher ist, wo keine neuen Adressen zu dem Hash-Speicherabschnitt **902** hinzugefügt werden und Pakete von unbekannten Quellenadressen fallen gelassen werden. Wenn VLAN nicht wahr ist, und wenn der Port nicht sicher ist, geht der Vorgang zu Schritt **1328**, wo das HASH_DONE* Signal geltend gemacht und vorübergehend geltend gemacht gelassen wird. An diesem Punkt werden die HASH_STATUS- und HASH_DSTPRT-Signale durch den HCB **402** ergriffen.

[0270] Wenn VLAN wahr ist, oder wenn SecurePort wahr ist, wie in Schritt **1326** bestimmt, oder nachdem Schritt **1328** ausgeführt ist, wird die Geltendmachung des HASH_DONE* Signals bis nach dem nächsten Quel-

lenadressen-Nachschlagen verzögert. Der Vorgang geht dann zu Schritt **1330**, wo ein Hash-Nachschlagen auf der Quellen-MAC-Adresse (SA) in der gleichen Weise wie oben für die Ziel-MAC-Adresse beschrieben durchgeführt wird. In Schritt **1330** wird ein Wert SA_Hit wahr zurückgegeben, wenn die Hash-Adresse für die entsprechende Vorrichtung gefunden ist. Von Schritt **1330** geht der Vorgang zu Schritt **1332**, wo ein Wert Src_Hit untersucht wird. Src_Hit ist mit SA_Hit durch die folgende Gleichung (4) verknüpft:

Src_Hit := SA_Hit & (HshPrt = SrcPort) (4)

wo Src_Hit wahr ist, wenn ein Source-Treffer vorkam (SA_Hit ist wahr), und wenn die in dem Eintrag in dem Hash-Speicherabschnitt **902** gefundene Portnummer gleich der tatsächlichen Quellenportnummer ist, wo das Paket empfangen wurde. Wenn die gespeicherte Quellenportnummer nicht gleich der tatsächlichen Qellenportnummer ist, wurde die Vorrichtung wahrscheinlich zu einem anderen Port bewegt, und der Hash-Speicherabschnitt **902** wird durch die CPU **230** aktualisiert, wie unten beschrieben. Wenn Src_Hit wahr ist, geht der Vorgang zu Schritt **1334**, wo das HASH_DONE* Signal geltend gemacht wird, wenn VLAN unwahr ist. Der Vorgang geht dann zu Schritt **1336**, wo die AGE-Zahl der Vorrichtung mit null verglichen wird. Wenn AGR nicht gleich null ist, wird die AGE-Zahl in Schritt **1338** auf Null gesetzt. Wenn die AGE-Zahl null ist, wie in Schritt **1336** bestimmt, oder nachdem sie in Schritt **1338** auf null gesetzt wurde, geht der Vorgang zu Schritt **1340**, wo das VLAN-Bit erneut untersucht wird. Wenn VLAN wahr ist, geht der Vorgang zu Schritt **1342**, wo eine HASH-VLAN-Routine oder Prozedur ausgeführt wird, um in Beziehung stehende Ports zu identifizieren, wie aus dem entsprechenden VLAN-Bitmap-Wert im Hash-Tabelleneintrag **910** bestimmt. Wenn VLAN nicht wahr ist, wie in Schritt **1340** bestimmt, geht der Vorgang zu Schritt **1344**, wo das HASH_DONE* Signal geltend gemacht oder für eine Zeitperiode gepulst, wenn nicht bereits geltend gemacht, und dann negiert wird. Von Schritt **1344** wird der Vorgang für diese Prozedur abgeschlossen. Die Negation des HASH_DONE* Signals beendet das Hash-Nachschlagen des HCB **402**.

[0271] Wenn, wieder auf Schritt **1332** verweisend, Src_Hit unwahr ist, geht der Vorgang zu Schritt **1350**, wo durch Untersuchen eines LearnDisPrt-Wertes festgestellt wird, ob das Lernen des Quellenports abgeschaltet ist. Wenn nicht, geht der Vorgang zu Schritt **1352**, wo neue Information des Pakets in geeignete Register geladen und die CPU **230** unterbrochen wird. Als Reaktion aktualisiert die CPU **230** den Hash-Speicherabschnitt **902** mit einem neuen Hash-Tabelleneintrag **910**. Wenn das Lernen des Quellenports abgeschaltet ist, wie in Schritt **1350** festgestellt, oder nachdem der Hash-Speicherabschnitt **902** in Schritt **902** akualisiert wurde, geht der Vorgang zu Schritt **1354**, um das SecurePort-Bit zu untersuchen. Wenn SecurePort wahr ist, geht der Vorgang zu Schritt **1356**, wo die HASH_STATUS[1:0] Signale in 00b = DROP_PKT geändert werden. In diesem Fall wird das neue Paket fallen gelassen, da die Adresse neu ist und neue Adressen auf sicheren Ports nicht erlaubt sind. Ferner wird, wenn gewünscht, eine Sicherheitsverletzungs-Unterbrechung an der CPU **230** geltend gemacht, um geeignete Maßnahmen als Reaktion auf die Sicherheitsverletzung zu ergreifen. Von Schritt **1356** geht der Vorgang zu Schritt **1344**. Wenn, wieder auf Schritt **1354** verweisend, das SecurePort-Bit unwahr ist, um einen nicht-sicheren Port anzuzeigen, geht der Vorgang zu Schritt **1340**. Wenn, wieder auf Schritt **1324** verweisend, SrcLookUp unwahr ist, geht der Vorgang direkt zu Schritt **1344**.

[0272] Fig. 14 ist ein Flussdiagramm, das eine Hash-Nachschlag-Prozedur zum Suchen aller Hash-Tabelleneinträge **910** im Hash-Speicherabschnitt **902** veranschaulicht. Im ersten Schritt **1402** wird ein Adresswert A gleich der empfangenen Hash-Adresse gesetzt, wie sie z. B. von Schritten **1316** oder **1330** gesendet würde. Der Vorgang geht zu Schritt **1404**, wo der Hash-Tabelleneintrag **910** in dem mit der empfangenen Hash-Adresse verbundenen Haupt-Hash-Eintragsabschnitt **906** gelesen wird. Der Vorgang geht zu Schritt **1406**, wo das VALIDENTRY-Bit gelesen und die MAC-Adresse des neuen Pakets mit der gespeicherten MAC-Adresse verglichen wird. Wenn der Eintrag gültig ist und eine genaue Übereinstimmung zwischen den MAC-Adressen vorkommt, geht der Vorgang zu Schritt **1408**, wo das HIT-Bit auf wahr gesetzt wird, um einen Hash-Treffer anzuzeigen, und der Vorgang kehrt zur aufrufenden Prozedur oder Routine zurück. Andernfalls, wenn der Eintrag nicht gültig ist oder keine Adressübereinstimmung vorkam, geht der Vorgang zu Schritt **1410**, wo das VALIDENTRY-Bit und der EOC-(Kettenende)Wert des Eintrags untersucht werden. Wenn der Eintrag ungültig oder das EOC erreicht ist, kehrt der Vorgang mit HIT-Bit unwahr zurück. Andernfalls wird in Schritt **1412** die Hash-Adresse gleich der Link-Adresse im Hash-Eintrag (Bytes F:C) gesetzt, und der Vorgang kehrt zu Schritt **1404** zurück, um den nächsten verketteten Eintrag im verketteten Hash-Eintragsabschnitt **908** zu versuchen. Der Vorgang wiederholt die Schritte **1404**, **1406**, **1410** und **1412**, bis ein gültiger Eintrag mit einer MAC-Adressübereinstimmung oder ein ungültiger Eintrag gefunden ist oder der EOC-Wert angetroffen wird.

[0273] Die folgende Tabelle (1) zeigt die CPU **230** Eingabe/Ausgabe-(E/A)Raum-Register für eine bestimmte, erfindungsgemäß implementierte Ausführung. Tabelle (1) wird nur als Beispiel bereitgestellt, wo die einzelnen Register in einzelnen Ausführungen implementiert werden können oder nicht, oder ähnliche Register verschie-

dene Nomenklatur aufweisen können.

TABELLE 1: E/A-Raum-Register der CPU 230

Offset(h)	Master	Shadowed	Access (R/W)	Reg_name/Bit_name	Description
0	PCB 406		CPU: R PCB: W MCB: ---- HCB: ----	**Interrupt Source 1** Bit 0: MCB_INT 1: MEM_RDY 2: ABORT_PKT 3: STAT_RDY 4-31: RESERVED	The source of any interrupt(s) to the CPU 230. These interrupts are cleared by the CPU 230 when it acknowledges the interrupt.
4	PCB 406		CPU: R/W PCB: R MCB: ---- HCB: ----	**Interrupt Mask 1** Bit 0: MCB_INT 1: MEM_RDY 2: ABORT_PKT 3: STAT_RDY 4: HASH_MISS 5-31: RESERVED	Interrupts to the CPU 230 which are to be masked.
8	PCB 406		CPU: R/W PCB: R/W MCB: ---- HCB: ----	**Packet Information -RdPkt** Bit 0: SOP 1: EOP 2-15: RESERVED 16-23: Length (for EOP) 24-31: RESERVED	This register is written by the CPU 230.
C	PCB 406		CPU: R/W PCB: R/W MCB: ---- HCB: ----	**Packet Information -WrPkt** Bit 0: SOP 1: EOP 2-5: BE (for SOP) 6-15: RESERVED 16-23: Length 24-31: RESERVED	This register is written by the EPSM 210.
10	PCB 406		CPU: R PCB: R/W MCB: ---- HCB: ----	**SIMM Presence Detect** Bit 0-3: simm1_pd[0..3] 4-7: simm2_pd[0..3] 8-11: simm3_pd[0..3] 12-15: simm4_pd[0..3] 16-31: RESERVED	This register will contain information on the SIMM's through a shift register interface.
14	PCB 406		CPU: R/W PCB: W MCB: ---- HCB: ----	**Polling Source (1 & 2)** Bit 0: MCB_INT 1: MEM_RDY 2: PKT_AVAIL 3: BUF_AVAIL 4: ABORT_PKT 5: STAT_RDY 6: HASH_MISS 7-31: RESERVED	The source of any interrupt(s) to the CPU 230 which have been masked.
18	PCB 406		CPU: R PCB: W MCB: ---- HCB: ----	**Interrupt Source 2** Bit 0: PKT_AVAIL 1: BUF_AVAIL 2-31: RESERVED	The source of any interrupt(s) to the CPU 230. These interrupts are cleared by the CPU 230 when it acknowledges the interrupt.
1c	PCB 406		CPU: R/W PCB: R MCB: ---- HCB: ----	**Interrupt Mask 2** Bit 0: PKT_AVAIL 1: BUF_AVAIL 2-31: RESERVED	Interrupts to the CPU 230 which are to be masked.
20	PCB 406		CPU: R/W PCB: R/W MCB: ---- HCB: ----	**QC Statistics Info** Bit 0-1: Port number 2-4: QC number 5-9: Register number 10-14: Number of Regs. 15-19: Max. number of regs. 20-31: RESERVED	The CPU 230 writing to this register will inform the QC interface to issue a statistics read of the appropriate port.
24	PCB 406		CPU: R PCB: R/W MCB: ---- HCB: ----	**Total Packet Info** Bit 0-15: Packet Length 16-23: Source Port 24-31: Dest. Port	This register is written by the EPSM 210
28	PCB 406		CPU: WO PCB: R/W MCB: ----	**Flush Fifo**	This register when written to will flush the fifo contents and continue to flush until EOP is

Offset(h)	Master	Shadowed	Access (R/W)	Reg_name/Bit_name	Description
			HCB: ----		received.
30	PCB 406	MCB 404 HCB 402	CPU: R/W PCB: R MCB: R HCB: R	**EPSM Setup** Bit 0: TPI installed 1: EXP installed 2: Master Switch Enable 3-4: QcXferSize[1:0] 5-6: TPIXferSize[1:0] 7: AI_FCS 8: DramWrDis 9: SramWrDis 10-12: Epsm Addr Dcd 13: Clk1Sel 14-21: CPU Port Number 22-31: RESERVED	This register holds the general setup parameters.
34	PCB 406	HCB 402	CPU: R/W PCB: ---- MCB: R HCB: R	**Port Speed** Bit 0: Port 0 Speed 1: Port 1 Speed : : 27: Port 27 Speed 28-31: RESERVED	This is the Port Speed Bitmap register. When the bit for a port is reset it is a 10mhz port and when the bit is set it is a 100mhz port. i.e. : 0 = 10mhz 1 = 100mhz Powerup default should contain the correct values.
38	PCB 406	MCB 404 HCB 402	CPU: R PCB: ---- MCB: R HCB: R	**Port Type** Bit 0: Port 0 Type 1: Port 1 Type : : 27: Port 27 Type 28-31: RESERVED	This is the Port Type Bitmap register. When the bit for a port is reset it is a QC port and when the bit is set it is a TLAN port. i.e. : 0 = QC 1 = TLAN Powerup default should contain the correct values.
3c	PCB 406	MCB 404	CPU: R/W PCB: R MCB: R HCB: ----	**MEM Request** Bit 0-23: Mem Address 24: Memory Select 25: Transfer size 26-29: Byte Enables 30: RW 31: Locked Page Hit	This is the register that contains the address and the controls for memory transfers from the CPU 230.
40	PCB 406	HCB 402	CPU: R PCB: ---- MCB: R HCB: R	**EPSM Revision** Bit 0-7: Rev. Number 8-31: RESERVED	This read only register provides the revision number for the EPSM 210.
54	HCB 402		CPU: R/W PCB: ---- MCB: ---- HCB: R	**HCB Utilization Setup** Bit 0-7: Port Number or Total 8-9: Mode 10-31: RESERVED	This register selects the port to be observed for HCB 402 utilization and the mode bits. The possible modes are TX, RX, Both.
58	HCB 402		CPU: R/W PCB: ---- MCB: ---- HCB: R/W	**HCB Utilization** Bit 0-31: Average Time	HCB 402 utilization is the average time the port selected is on the bus.
5c	HCB 402		CPU: R/W PCB: ---- MCB: ---- HCB: R	**Source CT_SNF Per Port** Bit 0: Port 0 1: Port 1 : : 27: Port 27 28-31: RESERVED	This register is a bitmap for the ports to indicate which source ports are able to CT and which are only able to do SnF.
60	HCB 402		CPU: R/W PCB: ---- MCB: ---- HCB: R	**Destination CT_SNF Per Port** Bit 0: Port 0 1: Port 1 : :	This register is a bitmap for the ports to indicate which destination ports are able to CT and which are only able to do SnF.

Offset(h)	Master	Shadowed	Access (R/W)	Reg_name/Bit_name	Description
				27: Port 27 28-31: RESERVED	
64	HCB 402 (High 2 bits of each xfersz)		CPU: R/W PCB: ---- MCB: ---- HCB: R	**XferSize Per Port** Bit 0-3: Port 0 xfersize 4-7: Port 1 xfersize 8-11: Port 2 xfersize 12-15: Port 3 xfersize 16-19: Port 4 xfersize 20-23: Port 5 xfersize 24-27: Port 6 xfersize 28-31: Port 7 xfersize	This register contains the xfersize for the specified port.
68	HCB 402 (High 2 bits of each xfersz)		CPU: R/W PCB: ---- MCB: ---- HCB: R	**XferSize Per Port** Bit 0-3: Port 8 xfersize 4-7: Port 9 xfersize 8-11: Port 10 xfersize 12-15: Port 11 xfersize 16-19: Port 12 xfersize 20-23: Port 13 xfersize 24-27: Port 14 xfersize 28-31: Port 15 xfersize	This register contains the xfersize for the specified port.
6c	HCB 402 (High 2 bits of each xfersz)		CPU: R/W PCB: ---- MCB: ---- HCB: R	**XferSize Per Port** Bit 0-3: Port 16 xfersize 4-7: Port 17 xfersize 8-11: Port 18 xfersize 12-15: Port 19 xfersize 16-19: Port 20 xfersize 20-23: Port 21 xfersize 24-27: Port 22 xfersize 28-31: Port 23 xfersize	This register contains the xfersize for the specified port.
70	HCB 402 (High 2 bits of each xfersz)		CPU: R/W PCB: ---- MCB: ---- HCB: R	**XferSize Per Port** Bit 0-3: Port 24 xfersize 4-7: Port 25 xfersize 8-11: Port 26 xfersize 12-15: Port 27 xfersize 16-19: Port 28 xfersize 20-31: RESERVED	This register contains the xfersize for the specified port.
74	HCB 402		CPU: R/W PCB: ---- MCB: ---- HCB: R	**Arb_Mode** Bit 0-1: Mode Value 2-31: RESERVED	This register contains the arbitration mode value. Arbitration modes available are FCFS, weighted, or round_robin.
78	HCB 402		CPU: R/W PCB: ---- MCB: ---- HCB: R	**HCB Misc Cntl** Bit 0: Enable CT Fifo 1: Enable Rd Extra WS 2: Enable CC Rd/Wr Qc 3: Enable CC Rd/Wr Qe 4: Enable Early AD 5-31: RESERVED	Miscellaneous controls for the HCB 402 subsection.
7c	HCB 402		CPU: R/W PCB: ---- MCB: ---- HCB: R	**Port Shutdown** Bit 0-27: Bitmap	Bitmap of ports to be that are disabled.
80	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Program Port State** Bit 0-1: State Value 2-31: RESERVED	This register tells what state the ports indicated in the port state bitmap register should be changed to. State Value — Condition 00 b — Disabled 01 b — Blocked/ Listening 10 b — Learning 11 b — Forwarding
90	MCB 404		CPU: R/W PCB: ---- MCB: R	**Port State Bitmap** Bit 0: Port 0 1: Port 1	This register indicates which ports are going to change their state. This register in

Offset(h)	Master	Shadowed	Access (R/W)	Reg_name/Bit_name	Description
			HCB: ----	: : 27: Port 27 28-31: RESERVED	combination with program port state register fill the port state registers.
94	MCB 404		CPU: R PCB: ---- MCB: R/W HCB: ----	**Port State #1** Bit 0-1: Port_0_st[1:0] 2-3: Port_1_st[1:0] 4-5: Port_2_st[1:0] 6-7: Port_3_st[1:0] 8-9: Port_4_st[1:0] 10-11: Port_5_st[1:0] 12-13: Port_6_st[1:0] 14-15: Port_7_st[1:0] 16-17: Port_8_st[1:0] 18-19: Port_9_st[1:0] 20-21: Port_10_st[1:0] 22-23: Port_11_st[1:0] 24-25: Port_12_st[1:0] 26-27: Port_13_st[1:0] 28-29: Port_14_st[1:0] 30-31: Port_15_st[1:0]	The two bits for each port tell the arbiter what state the port is in as follows: State Value Condition 00 b Disabled 01 b Blocked/ Listening 10 b Learning 11 b Forwarding
98	MCB 404		CPU: R PCB: ---- MCB: R/W HCB: ----	**Port State #2** Bit 0-1: Port_16_st[1:0] 2-3: Port_17_st[1:0] 4-5: Port_18_st[1:0] 6-7: Port_19_st[1:0] 8-9: Port_20_st[1:0] 10-11: Port_21_st[1:0] 12-13: Port_22_st[1:0] 14-15: Port_23_st[1:0] 16-17: Port_24_st[1:0] 18-19: Port_25_st[1:0] 20-21: Port_26_st[1:0] 22-23: Port_27_st[1:0] 24-31: RESERVED	The two bits for each port tell the arbiter what state the port is in as follows: State Value Condition 00 b Disabled 01 b Blocked/ Listening 10 b Learning 11 b Forwarding
9c	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Destination Miss Broadcast** Bit 0-28: DestMissBC bitmap 29-31: RESERVED	Destination miss broadcast bitmap.
a8	MCB 404		CPU: R/W PCB: ---- MCB: R/W HCB: ----	**Memory Bus Monitor Cntl** Bit 0-14: Monitor Mode 15: Monitor Select 16-23: Monitor Port Select 24-27: Filter Time Scale 28: Monitor Clear 29: Count/Filter Mode 30: Backpress. Enable 31: Alarm	The memory bus 214 monitor control is utilized to setup the monitoring (if any) that is being done on the memory bus 214.
ac	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Memory Bus Monitor Thresholds** Bit 0-7: Alarm Set Threshold 8-15: Alarm Clr Threshold 16-19: RESERVED 20-31: Peak BW	The memory bus 214 monitor thresholds are used to set an alarm and to clear the alarm.
b0	MCB 404		CPU: R PCB: ---- MCB: R/W HCB: ----	**Memory Bus Utilization** Bit 0-31: Percent Utilization	Memory bus 214 utilization register.
b8	MCB 404		CPU: R PCB: ---- MCB: R/W HCB: ----	**Dropped Packets Memory OF** Bit 0-31: Number of packets	The number of packets dropped due lack of memory space because of the memory threshold counters. This register is cleared when read.
bc	MCB		CPU: R	**Dropped Packets BC OF**	The number of broadcast

Offset(h)	Master	Shadowed	Access (R/W)	Reg_name/Bit_name	Description
	404		PCB: ---- MCB: R/W HCB: ----	Bit 0-31: Number of packets	packets dropped due lack of broadcast memory space. This register is cleared when read.
c0	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Hash Table Definition** Bit 0-14: Address[16:2] 15-23: Address[25:17] 24-25: Table size 26: Lock Hash Cycle 27: Vlan Group BC 28: Vlan Miss BC 29: Vlan Unicast 30-31: RESERVED	The address for the base of the hash table. Size of the hash table as described in the register definition.
c4	MCB 404		CPU: R PCB: ---- MCB: R/W HCB: ----	**Rx Sector Count OF** Bit 0-28: Bitmap 29-31: RESERVED	The bitmap of ports that have interrupted the CPU 230 due either a set or clear of receive sector threshold overflow.
c8	MCB 404		CPU: R PCB: ---- MCB: R/W HCB: ----	**Tx Packet Count OF** Bit 0-28: Bitmap 29-31: RESERVED	The bitmap of ports that have interrupted the CPU 230 due to either a set or clear of transmit packet threshold overflow.
cc	MCB 404		CPU: R PCB: ---- MCB: R/W HCB: ----	**Hash Address Low** Bit 0-31: Byte 0-3	The address which was missed when looking in the hash table.
d0	MCB 404		CPU: R PCB: ---- MCB: R/W HCB: ----	**Hash Address High** Bit 0-15: Byte 4-5 16-23: Source Port 24: Port Miss 25-31: RESERVED	The remaining hash address and source port.
d4	MCB 404		CPU: R PCB: ---- MCB: R/W HCB: ----	**Dropped Packets Receive OF** Bit 0-31: Number of packets	The number of packets dropped due to receive memory sectors overflow. This register is cleared when read.
d8	MCB 404		CPU: R PCB: ---- MCB: R/W HCB: ----	**Dropped Packets Transmit OF** Bit 0-31: Number of packets	The number of packets dropped due to transmit memory sectors overflow. This register is cleared when read.
dc	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Dropped Packets Receive** Bit 0-28: Port Bitmap 29-31: RESERVED	This register is the bitmap of ports that have dropped packets due to receive overflow.
e0	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Dropped Packets Transmit** Bit 0-28: Port Bitmap 29-31: RESERVED	This register is the bitmap of ports that have dropped packets due to transmit overflow.
e4	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Learning Disable Ports** Bit 0-27: Learn'g Dis. bitmap 28-31: RESERVED	Learning disable port bitmap.
e8	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Secure Ports** Bit 0-27: Secure port bitmap 28-31: RESERVED	Secure port bitmap.
ec	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Security Violation Stats** Bit 0-31: Count	This register contains the total dropped packets due to port security.
f0	MCB 404		CPU: R/W PCB: ---- MCB: R	**Security Violation** Bit 0-27: Port Bitmap 28-31: RESERVED	This register is the bitmap of ports that have dropped packets due to security.

Offset(h)	Master	Shadowed	Access (R/W)	Reg_name/Bit_name	Description
			HCB: ----		
f4	MCB 404		CPU: R/W PCB: ---- MCB: R/W HCB: ----	**Mem Control** Bit 0-1: Memory Type 2: Memory Speed 3: EDO Test Mode 4: Dbl Link Mode 5: DisRcPgHits 6: DisTxPGHits 7-31: RESERVED	This register contains the memory type, speed etc.
f8	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**RAS Select** Bit 0-31: Rasenx[1:0]	RAS enables for 4M blocks of memory.
fc	MCB 404		CPU: R/W PCB: R MCB: ---- HCB: ----	**Refresh Counter** Bit 0-9: Count 10-31: RESERVED	The refresh counter generates a refresh signal for the memory controller.
100	MCB 404 (bit 4-7)		CPU: R/W PCB: ---- MCB: R HCB: ----	**Filter Control** Bit 0-3: Address Enables[3:0] 4-7: Mask Enables[3:0] 8-31: RESERVED	This register enables address filtering and masking address.
104	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Mask Address Filter Low** Bit 0-31: Bytes 0-3	This register contains mask bits for address filtering.
108	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Mask Address Filter High** Bit 0-15: Bytes 4-5 16-31: RESERVED	This register contains mask bits for address filtering.
10c	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Address Filter 0Low** Bit 0-31: Bytes 0-3	This register contains bytes 0-3 of address filter 0.
110	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Address Filter 0High** Bit 0-15: Bytes 4-5 16-23: Dest. Port 24-31: FilterMask0	This register contains bytes 4-5 of address filter 0.
114	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Address Filter 1Low** Bit 0-31: Bytes 0-3	This register contains bytes 0-3 of address filter 1.
118	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Address Filter 1High** Bit 0-15: Bytes 4-5 16-23: Dest. Port 24-31: FilterMask1	This register contains bytes 4-5 of address filter 1.
11c	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Address Filter 2Low** Bit 0-31: Bytes 0-3	This register contains bytes 0-3 of address filter 2.
120	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Address Filter 2High** Bit 0-15: Bytes 4-5 16-23: Dest. Port 24-31: FilterMask2	This register contains bytes 4-5 of address filter 2.
124	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Address Filter 3Low** Bit 0-31: Bytes 0-3	This register contains bytes 0-3 of address filter 3.
128	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Address Filter 3High** Bit 0-15: Bytes 4-5 16-23: Dest. Port 24-31: FilterMask3	This register contains bytes 4-5 of address filter 3.
12c	MCB 404		CPU: R PCB: ---- MCB: R/W	**MCB Interrupt Source** Bit 0: Security Int 1: Memory Overflow Set	This register contains the source of any interrupt initiated in the MCB 404.

Offset(h)	Master	Shadowed	Access (R/W)	Reg_name/Bit_name	Description
			HCB: ----	2: Memory Overflow Clr 3: Broadcast OF Set 4: Broadcast OF Clr 5: Receive OF 6: Transmit OF 7: Rx Packet Aborted 8: BW Alarm Set 0 9: BW Alarm Clr 0 10: BW Alarm Set 1 11: BW Alarm Clr 1 12-31: RESERVED	
130	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**MCB Interrupt Mask** Bit 0: Security Int 1: Memory Overflow Set 2: Memory Overflow Clr 3: Broadcast OF Set 4: Broadcast OF Clr 5: Receive OF 6: Transmit OF 7: Rx Packet Aborted 8: BW Alarm Set 0 9: BW Alarm Clr 0 10: BW Alarm Set 1 11: BW Alarm Clr 1 12-31: RESERVED	This register contains the masking for any interrupt initiated in the MCB 404.
134	MCB 404		CPU: R/W PCB: ---- MCB: R/W HCB: ----	**MCB Polling Source** Bit 0: Security Int 1: Memory Overflow Set 2: Memory Overflow Clr 3: Broadcast OF Set 4: Broadcast OF Clr 5: Receive OF 6: Transmit OF 7: Rx Packet Aborted 8: BW Alarm Set 0 9: BW Alarm Clr 0 10: BW Alarm Set 1 11: BW Alarm Clr 1 12-31: RESERVED	This register contains the source of any interrupt initiated in the MCB 404 which are masked.
138	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**BackPressure Enable** Bit 0-23: RESERVED 24-27: Port Bitmap 28-31: RESERVED	
13c	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Bonded Port Set 0** Bit 0-27: Port Bitmap 28-31: RESERVED	
140	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Bonded Port Set 1** Bit 0-27: Port Bitmap 28-31: RESERVED	
144	MCB 404		CPU: R/W PCB: ---- MCB: R HCB: ----	**Default Vlan Bitmap** Bit 0-28: Bitmap	
148	MCB 404		CPU: R/W PCB: ---- MCB: ---- HCB: R	**Promiscuous Port** Bit 0-7: Observed Port No. 8-15: Rx Monitor Port No. 16-23: Tx Monitor Port No. 24-31: RESERVED	This register holds the value of the port that is being observed in promiscuous mode. Also contains the ports that the Rx traffic and the Tx traffic appear on.
200-2ff			CPU: R/W PCB: R/W MCB: ---- HCB: ----	**Quad Cascade 0 Regs**	This is the offset for the Quad Cascade registers. This is for QC0.

Offset(h)	Master	Shadowed	Access (R/W)	Reg_name/Bit_name	Description
300-3ff			CPU: R/W PCB: R/W MCB: ---- HCB: ----	**Quad Cascade 1 Regs**	This is the offset for the Quad Cascade registers. This is for QC1.
400-4ff			CPU: R/W PCB: R/W MCB: ---- HCB: ----	**Quad Cascade 2 Regs**	This is the offset for the Quad Cascade registers. This is for QC2.
500-5ff			CPU: R/W PCB: R/W MCB: ---- HCB: ----	**Quad Cascade 3 Regs**	This is the offset for the Quad Cascade registers. This is for QC3.
600-6ff			CPU: R/W PCB: R/W MCB: ---- HCB: ----	**Quad Cascade 4 Regs**	This is the offset for the Quad Cascade registers. This is for QC4.
700-7ff			CPU: R/W PCB: R/W MCB: ---- HCB: ----	**Quad Cascade 5 Regs**	This is the offset for the Quad Cascade registers. This is for QC5.
800-8ff			CPU: R PCB: R/W MCB: ---- HCB: ----	**QC Statistics Buffer**	This is the address space for the statistics buffers just read from the Quad Cascade.
900			CPU: R/W PCB: R/W MCB: ---- HCB: ----	**HCB FIFO - BPDU**	This is address of the fifo to send/receive packet data to/from the HCB 402.
a00			CPU: R/W PCB: ---- MCB: R/W HCB: ----	**MCB DATA FIFO**	This is address of the fifo to send/receive data to/from the MCB 404. 16 Byte Fifo.
b00-fff				**RESERVED For Expansion**	

[0274] Die folgenden Registerdefinitionen werden bereitgestellt, um die Register von Tabelle (1) zu erklären.

Unterbrechungsinformation

[0275] Es gibt drei Unterbrechungspins von dem EPSM **210** an die CPU **230**: CPUINTHASHL, CPUINTPKTL und CPUINTL. Der CPUINTHASHL wird nur geltend gemacht, wenn ein Hash-Miss aufgetreten ist, und wird durch Lesen des Hash-Adresse-Tief-Registers (bei Offset'hcc) gelöscht. Der CPUINTPKTL wird geltend gemacht, wenn entweder ein Paket in dem Paketschnittstellen-FIFO verfügbar ist oder wenn der Paketschnittstellen-FIFO freien Pufferplatz zum Senden von mehr Paketdaten hat. Der CPUINTL wird für vier mögliche Quellen geltend gemacht: Eine dieser Quellen betrifft acht mögliche Quellen im MCB **404**. Die Unterbrechungsquellen werden eine Unterbrechung der CPU **230** bewirken, wenn sie nicht maskiert werden. Damit die Information der Unterbrechungsquelle verfügbar werden kann, ohne die CPU **230** zu unterbrechen, ist ein Abfragemechanismus verfügbar. Das Maskieren einer Unterbrechungsquelle bewirkt, dass die Unterbrechungen von der CPU **230** ferngehalten werden, aber die Information noch in dem Abfragequellenregister verfügbar ist. Wenn z. B. das STAT_RDY-Maskenbit gesetzt ist, wird, wenn die verlangte Statistik verfügbar ist, keine Unterbrechung auftreten, aber die CPU **230** kann noch bestimmen, dass die Statistik bereit ist, durch Lesen des Abfrageregisters gelesen zu werden. Man beachte: Das Unterbrechungsquellenregister wird durch Lesen desselben gelöscht, aber das Abfragequellenregister muss beschrieben werden, um es zu löschen.

[0276] Unterbrechungsquelle 1 Reg. – (Offset = 'h00) Quelle der an die CPU **230** gesendeten Unterbrechung. Dieses Register wird durch den EPSM **210** aktualisiert, und dann wird die Unterbrechung an die CPU **230** gesendet. Wenn die CPU dieses Register liest, wird der Inhalt gelöscht. Ein Wert von 1 in einem Bit zeigt an, dass eine Unterbrechung aufgetreten ist. Die Vorgabe ist 32'h0000_0000.

Bit 0 (W/R) – MCB_INT ist die Unterbrechung, die der CPU **230** sagt, dass eine Unterbrechung im MCB **404** aufgetreten ist, und dass das MCB-Unterbrechungsquellenregister gelesen werden muss, um die Unterbrechung weiter zu verstehen. Vorgabe ist 0.
Bit 1 (W/R) – MEM_RDY ist die Unterbrechung, die der CPU **230** sagt, dass die verlangten Speicherdaten im Pufferraum vorhanden sind. Vorgabe ist 0.

Bit 2 (W/R) – ABORT_PKT ist die Unterbrechung, die der CPU **230** zeigt, dass das ABORT_IN* Signal in den PCB **406** geltend gemacht wurde. Vorgabe ist 0.
Bit 3 (W/R) – STAT_RDY ist die Unterbrechung, die der CPU **230** sagt, dass die verlangte Statistikinformation im PCB **406** Pufferraum bereit ist. Vorgabe ist 0.
Bits 4–31 (RO)-RESERVIERT. Immer als 0 gelesen.

PCB-Registerschnittstelle für Unterbrechungsquellenregister

McbInt (in)	– Eingabe vom MCB, die Bit 0 bestimmt.
MemRdy (in)	– Eingabe vom FIFO, die Bit 1 bestimmt.
AbortPktInt (in)	– Eingabe von der HCB **402** Schnittstelle, die Bit 4 bestimmt.
StatRdyInt (in)	– Eingabe von der QC-Schnittstelle, die Bit 5 bestimmt.
CpuInt_(out)	– das Signal an die CPU **230**, das anzeigt, dass eine Unterbrechung stattgefunden hat

[0277] Unterbrechungsmaske 1 Reg. – (Offset = 'h04) Unterbrechungen, die durch die CPU **230** zu maskieren sind. Ein Wert von 1 in einem Bit zeigt an, dass die Unterbrechung maskiert ist. Vorgabe = 32'h0000_001f.

Bit 0 (W/R)	– Maskiert die McbInt-Unterbrechung an die CPU **230**. Vorgabe ist 1.
Bit 1 (W/R)	– Maskiert die MemRdy-Unterbrechung an die CPU **230**. Vorgabe ist 1.
Bit 2 (W/R)	– Maskiert die AbortPktInt-Unterbrechung an die CPU **230**. Vorgabe ist 1.
Bit 3 (W/R)	– Maskiert die StatRdyInt-Unterbrechung an die CPU **230**. Vorgabe ist 1.
Bit 4 (W/R)	– Maskiert die Hash-Miss-Unterbrechung an die CPU **230**. Vorgabe ist 1.
Bit 5–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

[0278] Unterbrechungsquelle 2 Reg. – (Offset = 'h18) Quelle der an die CPU **230** gesendeten CPU-INTPKTL-Unterbrechung. Dieses Register wird durch den EPSM **210** aktualisiert, und dann wird die Unterbrechung an die CPU **230** gesendet. Wenn die CPU **230** dieses Register liest, wird der Inhalt gelöscht. Ein Wert von 1 in einem Bit zeigt an, dass eine Unterbrechung aufgetreten ist. Vorgabe = 32'h0000_0000.

Bit 0 (W/R)	– PKT-AVAIL ist die Unterbrechung, die der CPU **230** sagt, dass Paketdaten für die CPU **230** vorhanden sind. Vorgabe ist 0.
Bit 1 (W/R)	– BUF_AVAIL ist die Unterbrechung, die der CPU **230** sagt, dass Pufferplatz für die CPU **230** verfügbar ist, um Paketdaten zu senden. Vorgabe ist 0.
Bits 2–13 (RO)	– RESERVIERT. Immer als 0 gelesen.

PCB-Registerschnittstelle für Unterbrechungsquellenregister

PktAvailInt (in)	– Eingabe vom TX FIFO, die Bit 2 bestimmt.
BufAvailInt (in)	– Eingabe vom RX FIFO, die Bit 3 bestimmt.
CpuInt_Pkt (out)	– das Signal an die CPU **230**, das anzeigt, dass eine Paketunterbrechung aufgetreten ist.

[0279] Unterbrechungsmaske 2 Reg. – (Offset = 'h1c) Unterbrechungen, die von der CPU **230** zu maskieren sind. Ein Wert von 1 in einem Bit zeigt an, dass die Unterbrechung maskiert ist. Vorgabe ist 32'h0000_0003.

Bit 0 (W/R)	– Maskiert die PktAvailInt-Unterbrechung an die CPU **230**. Vorgabe ist 1.
Bit 1 (W/R)	– Maskiert die BufAvailInt-Unterbrechung an die CPU **230**. Vorgabe ist 1.
Bits 2–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

[0280] Abfragequellen 1 & 2 Reg. – (Offest = 'h14) Dieses Register enthält die maskierte Unterbrechungsinformation und wird durch die CPU **230** gelöscht, die Einsen schreibt, um die gewünschten Bits zu löschen. Dies erlaubt der CPU **230**, abzufragen anstatt unterbrochen zu werden. Die CPU **230** wird jede Unterbrechungsquelle zu maskieren haben, die sie stattdessen abzufragen wünschen würde.

Bit 0 (W/R)	– MCB_INT ist die Unterbrechung, die der CPU **230** sagt, dass eine Unterbrechung im MCB **404** aufgetreten ist, und dass das Unterbrechungsquellenregister gelesen werden muss, um die Unterbrechung weiter zu verstehen. Vorgabe ist 0.
Bit 1 (W/R)	– MEM_RDY ist die Unterbrechung, die der CPU **230** sagt, dass die verlangten Speicherdaten im Pufferraum vorhanden sind. Vorgabe ist 0.
Bit 2 (W/R)	– PKT_AVAIL ist die Unterbrechung, die der CPU **230** sagt, dass Paketdaten für die CPU **230** vorhanden sind. Vorgabe ist 0.
Bit 3 (W/R)	– BUF_AVAIL ist die Unterbrechung, die der CPU **230** sagt, dass Pufferplatz für die CPU **230** verfügbar ist, um Paketdaten zu senden. Vorgabe ist 0.
Bit 4 (W/R)	– ABORT_PKT ist die Unterbrechung, die der CPU **230** sagt, dass das Abort_ In Signal in den PCB **406** geltend gemacht wurde. Vorgabe ist 0.
Bit 5 (W/R)	– STAT_RDY ist die Unterbrechung, die der CPU **230** sagt, dass die verlangte Statistikinformation im PCB **406** Pufferplatz bereit ist. Vorgabe ist 0.
Bit 6 (W/R)	– HASH_MISS ist die Unterbrechung, die der CPU **230** sagt, dass ein Hash-Miss aufgetreten ist.
Bits 7–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

PCB-Schnittstelle für Abfragequellenregister

McbInt (in)	– Eingabe vom MCB, die Bit 0 bestimmt.
MemRdy (in)	– Eingabe vom Speicher-FIFO, die Bit 1 bestimmt.
PktAvailInt (in)	– Eingabe vom TX FIFO, die Bit 2 bestimmt.
BufAvailInt (in)	- Eingabe vom RX FIFO, die Bit 3 bestimmt.
AbortPktInt (in)	– Eingabe von HCB **402** Schnittstelle, die Bit 4 bestimmt.
StatRdyInt (in)	– Eingabe von QC-Schnittstelle, die Bit 5 bestimmt.
m_HashInt (in)	– Eingabe vom MCB **404**, die Bit 6 bestimmt.

Paketdatenkonfiguration

[0281] Es gibt drei für Paketdatenübertragungen verwendete Reisten: Eines für empfangene Pakete und zwei für Sendepakete. Die empfangenen Pakete sind mit dem ReadOutPkt Signal vom HSB **206** verbunden. Die Sendepakete sind mit dem WriteInPkt Signal vom HSB **206** verbunden. Beachte: Die Begriffe Empfangen und Senden sind auf den HSB **206** bezogen. Die CPU **230** sollte auf das geeignete Register zugreifen, bevor auf den Paketdatenpuffer zugegriffen wird.

[0282] Paketinformations RdPkt Reg. – (Offset = 'h08) Die benötigte Information für das durch die CPU **230** empfangene Datenpaket. Empfange Pakete beziehen sich auf den HSB **206**. Vorgabe = 32'h0000_0000.

Bit 0 (W/R)	– SOP. Start von Paket von der CPU **230**. 1 = SOP.
Bit 1 (W/R)	- EOP. Ende von Paket von der CPU **230**. 1 = EOP.
Bits 2–15 (RO)	– RESERVIRET. Immer als 0 gelesen.
Bits 16–23 (W/R)	– Länge von Daten im FIFO, wenn EOP geltend gemacht ist (Zahl von Bytes.
Bits 24–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

PCB-Registerschnittstelle für Paketinformations RdPkt Register

r_Sop (out)	– Paketstartanzeiger, gegeben an die HSB **206** Schnittstelle.
r_Eop (out)	– Paketendeanzeiger, gegeben an die HSB **206** Schnittstelle.
r_length (out)	– Länge in Bytes von Daten im Puffer, wenn EOP angezeigt wird.

[0283] Paketinformations WrPkt Reg. – (Offset = 'h0c) Die benötigte Information für das durch den HSB **206** gesendete Datenpaket. Sendepaket bezieht sich auf den HSB **206**. Vorgabe ist 32'h0000_0000.

Bit 0 (W/R)	– SOP. Start von Paket vom HSB **206**. 1 = SOP.
Bit 1 (W/R)	– EOP. Ende von Paket vom HSB **206**. 1 = EOP.
Bits 2–5 (W/R)	– Bytefreigaben für mit SOP oder EOP verbundenes DWORD. Gewöhnlich sind alle Bytes freigegeben. 1 = freigegeben.
Bits 6–15 (R/O)	– RESRVIERT. Immer als 0 gelesen.
Bits 16–23 (W/R)	– Länge von Daten im FIFO (Zahl von Bytes)
Bits 24–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

PCB-Registerschnittstelle für Paketinformations Wr Pkt Register

h_SopIn_ (in)	– SOP-Anzeiger von der HSB **206** Schnittstelle
h_EopIn_ (in)	– EOP-Anzeiger von der HSB **206** Schnittstelle
h_ByteAalIn_ (in)	– Bytefreigbaben von der HSB **206** Schnittstelle

[0284] Gesamtpaket-Info – (Offset = 'h24) Dies ist die Information, die der MCB **404** dem Paket hinzufügt, bevor es an die CPU **230** gesendet wird. Dieser Wert wird gesetzt, wenn es ein SOP für ein an die CPU gerichtetes Paket gibt. Vorgabe = 32'h0000_0000.

Bits 0–15 (RO)	– Paketlänge
Bits 16–23 (RO)	– Quellenport
Bits 24–31 (RO)	– Zielport

Speicheranwesenheits-Erfassung

[0285] SIMM/DIMM-Anwesenheits-Erfassungsregister – (Offset = 'h10) Enthält die Information über die SIMMs in dem System. Diese Information wird kurz nach Rücksetzen aus einem Schieberegister auf der Platine geladen.

Bits 0–3 (RO)	– simm1_pd[0...3].
Bits 4–7 (RO)	– simm2_pd[0...3].
Bits 8–11 (RO)	– simm3_pd[0...3].
Bits 12–15 (RO)	– simm4_pd[0...3].
Bits 16–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

PCB-Registerschnittstelle für Anwesenheits-Erfassungsregister

i_PDSerIn (in)	– Serielle Eingabe von Anwesenheitserfassungs-Schieberegistern.

Vierfach-Kaskaden-Statistik-Einstellung

[0286] QC-Statistik-Informationsregister – (Offset = 'h20). Einstellinformation zum Lesen der Vierfach-Kaskaden-Statistikregister. Die CPU schreibt dieses Register, das die Statistik-Lesungen einleitet. Vorgabe = 32'h000b_8000.

Bits 0–1 (W/R)	– Portnummer. Dies ist die Portnummer, dessen Statistik gelesen wird. Der zu lesende Port wird durch diese Nummer und die spezifizierte Vierfach-Kaskade bestimmt.
Bits 2–4 (W/R)	– QC-Nummer. Bezeichnet die Vierfach-Kaskade, auf die zuzugreifen ist. Reservierte Kombinationen 3'b110 und 3'b111.
Bits 5–9 (W/R)	– Registernummer. Dies ist die Nummer des ersten für den spezifizierten Port zu lesenden Registers.
Bits 10–14 (W/R)	– Zahl von Register. Die ist die Zahl zu lesender Register. Beachte: Software ist nötig, um diese Zahl zusammen mit der Registernummer im Bereich zu lesender, verfügbarer Register zu halten.
Bits 15–19 (W/R)	– Maximale Zahl von Registern. Dies ist die maximale Zahl von Statistikregisterrn, die den Vierfach-Kaskaden verfügbar sind. Vorgabe = 6'h17.
Bits 20–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

PCB-Registerschnittstelle für Vierfach-Kaskaden-Statistik-Einstellregister

r_QcStatPortNo (out)	– Portnummer zum Statistik-Lesen. Dies ist ein Wert zwischen 0 und 3. Er wird zusammen mit der QC-Nummer benutzt, um zu bestimmen, welcher Port in dem Schalter beobachtet wird.
r_QcStatQcNo (out)	– Qc-Nummer. Wird mit der obigen Portnummer verwendet.
r_StatRegNo (out)	– Anfangsregisternummer. Dies ist die Nummer des ersten zu lesenden Statistikregisters.
r_NoStatRegs (out)	– Zahl zu lesender Statistikregister.
r_Maxregs (out)	- Maximalzahl von Statistikregistern, die existieren. Diese steht besonders für künftigen Gebrauch zur Verfügung, wenn die Zahl von Statistiken, die unterhalten werden, geändert wird.

EPSM **210** Einstellung

[0287] EPSM-Einstellregister – (Offset = 'h30) Allgemeine Einstellparameter für dem EPSM **210**. Vorgabe = 32'h0007_1000 oder 32'h0007_3000, abhängig von clk1sel-Eingabe.

Bit 0 (W/R)	– TPI installiert. 1 = TPI **220** installiert. Vorgabe = 0. Dieses Bit kann nur geschrieben werden, wenn Master Switch Enable (Bit 2) negiert ist.
Bit 1 (W/R)	– EXP installiert. 1 = Erweiterung installiert. Vorgabe = 0. Dieses kann nur geschrieben werden, wenn Master Switch Enable (Bit 2) negiert ist.
Bit 2 (W/R)	– Master Switch Enable. 1 = ermöglicht Paketverkehr. Vorgabe = 0.
Bits 3–4 (W/R)	– QcXferSize[1:0]. Diese Bits können nur geschrieben werden, wenn Master Switch Enable (Bit 2) negiert ist. 00 = 16 Byte Übertragungsgröße auf dem HSB **206**. 01 = 32 Byte Übertragungsgröße auf dem HSB **206**. 10 = 64 Byte Übertragungsgröße auf dem HSB **206**. 11 = Ungültige Kombination.

Bits 5–6 (W/R)	– PTIXferSize[1:0]. Diese Bits können nur geschrieben werden, wenn Master Switch Enable (Bit 2) negiert ist. 00 = 16 Byte Übertragungsgröße auf dem HSB **206**. 01 = 64 Byte Übertragungsgröße auf dem HSB **206**. 10 = 128 Byte Übertragungsgröße auf dem HSB **206**. 11 = 256 Byte Übertragungsgröße auf dem HSB **206**.
Bit 7 (W/R)	– AIFCS. Dieses Bit wird benutzt, um den Vierfach-Kaskaden zu ermöglichen, die FCS-Bits automatisch einzufügen. Dies wird nur für die Pakete von der CPU **230** verwendet.
Bit 8 (W/R)	– DramWrDis. Dies wird, wenn gesetzt, Schreiben von der CPU **230** in das DRAM unterbinden. Vorgabe = 0.
Bit 9 (W/R)	– SramWrDis. Dies wird, wenn gesetzt, Schreiben von der CPU **230** in das interne SRAM unterbinden. Vorgabe = 0.
Bits 10–12 (W/R)	– EPSM **210** Adressdecodierung. Diese Bits werden benutzt, um den EPSM **210** Registerraum und die Speicherschnittstelle zu decodieren.
Bit 13 (RO)	– clk1sel. 1 = CLK2 Frequenz ist 1X CLK1 Frequenz. 0 = CLK2 Frequenz ist 2X CLK1 Frequenz.
Bits 14–21 (RO)	– CPU Portnummer. Bezeichnet die CPU Portnummer. Vorgabe = 8'h1c.
Bits 22–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

PCB-Registerschnittstelle für EPSM-Einstellregister

clk1sel (in)	– Eingabe von Pin, um zu bestimmen, ob clk1 und clk2 gleiche Raten haben.
r_DramWrDis (out)	– Lässt CPU **230** wissen, dass Sehreibungen in das DRAM unterbunden sind.
r_SramWrDis (out)	– Lässt CPU **230** wissen, dass Schreibungen in das interne SRAM unterbunden sind.
r_EPSMAdrDcd (out)	– Diese 3-Bit Zahl wird mit Adressbits 31:29 auf dem CPU **230** Bus verglichen.

HCB-Registerschnittstelle für EPSM-Einstellregister

r_MstrSwEn (out)	– Sagt dem Arbiter usw., dass der Schalter für Paketverkehr freigegeben ist.
r_TpiInst (out) r_ExpInst (out) r_NonULBCMode[1:0] (out) r_ULBCMode[1:0] (out) r_AIFCS (out)	

MCB-Registerschnittstelle für EPSM-Einstellregister

r_DramWrDis (out)	– Unterbindet CPU-Anforderungen für DRAM-Schreibungen.
r_SramWrDis (out)	– Unterbindet CPU-Anforderungen für Schreibungen in das interne SRAM.

[0288] EPSM-Revisionsregister – (Offset = 'h40) Die Revisionsnummer des EPSM **210**.

Bits 0–7 (RO)	– Die Revisionsnummer des EPSM **210**.
Bits 8–31	– RESERVIERT. Immer als 0 gelesen.

PCB-Registerschnittstelle für EPSM-Revisionsregister

Keine

Porteinstellung

[0289] Portgeschwindigkeitsregister – (Offset = 'h34) Bitmap, die die Geschwindigkeit jedes Ports enthält. 1 = 100 Mhz, 0 = 10 MHz. Vorgabe = 32'h0f00_0000.

Bit 0 (W/R)	– Port **0** Geschwindigkeit.
Bit 1 (W/R)	– Port **1** Geschwindigkeit.
:	
:	
Bit 27 (W/R)	– Port **27** Geschwindigkeit.
Bits 28–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

HCB-Registerschnittstelle für Portgeschwindigkeitsregister

r_PortSpd[27:0] (out)	– Portgeschwindigkeits-Bitmap für HCB **402** Blöcke.

[0290] Porttyp-Register – (Offset = 'h38) Bitmap, die den Typ jedes Ports enthält. 1 = TLAN, 0 = Vierfach-Kaskade. Vorgabe = 32'h0100_000.

Bit 0 (W/R)	– Port **0** Typ
Bit 1 (W/R)	– Port **1** Typ
:	
:	
Bit 27 (W/R)	– Port **27** Typ
Bit 28–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

MCB-Register- & HCB-Registerschnittstelle für Porttyp-Register

r_PortType[27:0] (out)	– Porttyp-Bitmap für den MCB-Bus **404** & HCB-Bus **402**.

CPU-Speicheranforderung

[0291] Die Speicheranforderungen durch die CPU **230** können auf zwei Wegen erfolgen. Das folgende Register wird in beiden Verfahren verwendet; die CPU greift auf das Register nur direkt zu, wenn das Anfangsregister/FIFO-Speicheranforderungsverfahren verwendet wird.

[0292] Speicheranforderungsregister – (Offset = 'h3c) Die CPU schreibt in dieses Register, um Speicher-Lesen oder Schreiben anzufordern. Dieser Anforderungsmechanismus wird benutzt, um entweder auf das externe DRAM oder das interne SRAM zuzugreifen.

Bits 0–23 (W/R)	– Anfangsadresse[25:0] der Übertragung. Für SRAM-Zugriffe sind Bits 23–8 reserviert. Bits 7:0 adressieren die 256 24-Bit Wörter.
Bits 24 (W/R)	– Speicherauswahl. 0 = Zugriff auf externes DRAM (d. h. Paket- & Hash-Speicher). 1 = Zugriff auf linternes SRAM (d. h. Paketsteuerregister).
Bit 25 (W/R)	– Übertragungslänge. 0 = 1 Übertragung (4 Bytes) 1 = 4 Übertragungen (16 Bytes). Anmerkung: Die Startadresse & Übertragungslänge sollten nicht so eingestellt sein, dass die Übertragung eine 2 K Seitengrenze kreuzen würde. Ein Weg, dies zu garantieren, ist, sicherzustellen, dass alle Datenstrukturen (wie Hash-Einträge) 16-Byte-ausgerichtet sind.
Bits 26–29 (W/R)	– Bytefreigabe[3:0]. (1 = geltend gemacht). Nützlich zum Schreiben von Teilwörtern. Wird auch im EDO-Testmodus benutzt, um ohne CAS zu lesen. Zum Schreiben mit Übertragungslänge größer als 1, müssen Bytefreigaben 1111 sein. Diese sind beim Lesen ohne Bedeutung, sofern nicht der EDO-Testmodus eingestellt ist.
Bit 30 (W/R)	– Schreiben/Lesen. 0 = Lesen. 1 = Schreiben.
Bit 31 (W/R)	– Treffer auf gesperrte Seite. Zeigt an, dass eine andere CPU-Anfordederung in der gleichen Speicherseite folgen wird. Der DRAM-Speicher-Arbiter wird dem anderen Anforderer das Speichersystem nicht gewähren, und RAS wird nach dem momentanen Zyklus geltend gemacht bleiben. Wird nur im EDO-Testmodus benutzt. Kein anderer Anforderer, einschließlich Auffrischung, hat Zugriff auf den Speicher, während gesetzt. Sollte niemals in SRAM-Zugriffen (außer für Hardware-Fehlersuche) benutzt werden, da ankommender Paketspeicherverkehr aufhören wird, während das SRAM gesperrt ist.

MCB-Registerschnittstelle für Speicheranforderungsregister

CpuAdr[25:2] (out)	– Übergibt Startadresse Memctl & Mcbsram Modul.
CpuBE[3:0] (out)	– Übergibt Bytefreigaben an Memctl & Mcbsram Modul.
CpuLn[1:0] (out)	– Übergibt Übertragungslänge an Memctl & Mcbsram Modul (00, wenn In = 1, 11, wenn In = 4).
CpuMemSel (out)	– Steuert Mux zwischen externen DRAM (0) 6 internen SRAM (1) Daten.
CpuWr (out)	– an Memctl & Mcbsram Modul geltend gemacht, wenn Schreib/Lese-Bit = 1.
CpuPgHit (out)	– an Memctl & Mcbsram Modul geltend gemacht, wenn Gesperrte-Seite-Treffer-Bit = 1.
CpuReq (out)	– an Memctl & Mcbsram geltend gemacht, wenn das Speicheranforderungsregister geschrieben wird und Speicherauswahl = 0. Muss geltend gemacht bleiben, bis CpuAck geltend gemacht wird.
CpuAck (in)	– wird von Memctl Modul an Mcb-Registern geltend gemacht, wenn Cpu-Req angenommen wird.
CpuInternalReq (out)	– An McbSram geltend gemacht, wenn das Speicheranforderungsregister geschrieben wird und Speicherauswah = 1. Muss geltend gemacht bleiben, bis CpuInternalAck geltend gemacht wird.
CpuInternalAck (in)	– wird vom Mcbsram Modul an Mcb-Registern geltend gemacht, wenn CpuInternalReq angenommen wird.

[0293] Anmerkung: Die Folgende Sequenz sollte benutzt werden, um auf EDO-Speicher zu prüfen:
1: EDO-Testmodusbit im Speichersteuerregister setzen.
2: Ein DWORD in die zu prüfende Bank mit 0000h schreiben.
3: Das gleiche DWORD mit gesetztem Gesperrte-Seite-Treffer-Bit und Bytefreigaben = 1111b lesen. Nach diesem Lesen werden EDO-DRAMs MD tief halten, während FPM-DRAMs MD schweben lassen, und ein Pull-Up-Widerstand auf MD[0] diese Leitung nach etwa 100 ns hoch ziehen wird.
4: Das DWORD mit gelöschtem Gesperrte-Seite-Treffer-Bit und den Bytefreigaben = 0000b erneut lesen. Dies ist ein Lesen ohne CAS geltend gemacht. MD[0] wird für EDO-DRAM tief und für FPM hoch sein.
5: Schritte 1–4 für jede installierte Speicherbank wiederholen. Speichertyp kann nur auf EDO-DRAM gesetzt werden, wenn alle Bänke EDO-DRAM enthalten.
6: EDO-Testmodusbit löschen und den Speichertyp setzen. EDO-Testmodus nicht gesetzt lassen.

Gemischter Port

[0294] Gemischter-Port-Register – (Offset = 'h148) Die Steuerungen und welcher Port im gemischten Modus beobachtet wird, ist in dem Register enthalten. Vorgabe = 32'h0000_0000. Dieses Register kann nur geschrieben werden, wenn Master Switch Enable (EPSM-Einstellregister) negiert ist.

Bits 0–7 (W/R)	– Portnummer, die im gemischten Modus beobachtet wird.
Bits 8–15 (W/R)	– Der Port, der Daten, die empfangen werden, zeigen wird.
Bits 16–23 (W/R)	– Der Port, der Daten, die an den beobachteten Port gesendet werden, zeigen wird.
Bits 24–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

Hochgeschwindigkeits-Busmonitor

[0295] HSB-Benutzungs-Einstellregister – (Offset = 'h54) Die Steuerungen und welcher Port für HSB **206** Benutzung überwacht werden wird. Vorgabe = 32'h0000_0000.

Bits 0–7 (W/R)	– Portnummer oder Total.
Bits 8–9 (W/R)	– Modus.
Bits 10–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

[0296] HSB-Benutzungsregister – (Offset = 'b58) HSB **206** Benutzung ist die mittlere Zeit, die der ausgewählte Port auf dem HSB **206** ist. Vorgabe = 32'h0000_0000:
Bits 0–31 (RO) – Mittlere Zeit, die der ausgewählte Port auf dem HSB **206** ist.

CUT-THRU/STORE-N-FORWARD INFORMATION

[0297] Quellen-CT_SNF-Register – (Offset = 'h5c) Bitmap, die den CT/SnF-Status des Quellenports enthält. 0 = CT, 1 = SNF. Vorgabe = 32'h0000_0000.

Bit 0 (W/R)	– Port **0** Quelle CT_SNF.
Bit 1 (W/R)	– Port **1** Quelle CT_SNF.
:	
:	
Bit 27 (W/R)	– Port **27** Quelle CT_SNF.
Bits 28–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

HCB-Registerschnittstelle für Quellen-CT_SNF-Register

TblSrcPrt (in)	– Der momentane Paketquellenport. 8-Bit Eingabe.
r_RxPortCtSnf (out)	– Der CT_SNF-Status für TblSrcPrt. 1-Bit Ausgabe.

[0298] Ziel-CT_SNF-Register – (Offset = 'h60) Bitmap, die den CT/SnF-Status des Zielports enthält. 0 = CT; 1 = SNF. Vorgabe = 32'h0000_0000.

Bit 0 (W/R)	– Port **0** Ziel CT_SNF.
Bit 1 (W/R)	– Port **1** Ziel CT_SNF.
:	
:	
Bit 27 (W/R)	– Port **27** Ziel CT_SNF.
Bits 28–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

HCB-Registerschnittstelle für Quellen-CT_SNF-Register

TblDstPrt (in)	– Der momentane Paketzielport. (8-Bit Eingabe).
r_TxPortCtSnf (out)	– Der CT_SNF-Satus für TblDstPrt. 1-Bit Ausgabe).

Arbitrationsinformation

[0299] Arbitrationsmodusregister – (Offset = 'h74) Enthält den Arbitrationsmoduswert. Vorgabe = 32'h0000_0000. Dieses Register kann nur geschrieben werden, wenn Master Switch Enable (EPSM-Einstellregister) negiert ist.

Bits 0–1 (W/R)	– Arbitrationsmodus. 2'b00: Wer-zuerst-kommt, -mahlt-zuerst-Arbitrationsmodus. 2'b01: Arbitationsmodus mit gewichteter Priorität. 2'b10: Umlauf-Arbitrationsmodus. 2'b11: Bewirkt auch Wer-zuerst-kommt, -mahlt-zuerst-Modus.
Bits 2–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

HCB-Registerschnittstelle für Arbitrationsmodusregister

r_ArbMode (out)	– Der oben gezeigte 2-Bit Wert, der in Arbitrationsmodulen im HCB **402** benötigt wird.

[0300] Arbitrationsgewichtsregister #1 – (Offset = 'h64) Das Gewicht für Ports **0**–**7** für Arbitrationsmodus mit gewichteter Priorität.

Bits 0–3 (W/R)	– Port **0** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 4–7 (W/R)	– Port **1** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 8–11 (W/R)	– Port **2** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 12–15 (W/R)	– Port **3** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 16–19 (W/R)	– Port **4** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 20–23 (W/R)	– Port **5** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 24–27 (W/R)	– Port **6** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 28–31 (W/R)	– Port **7** Arbitrationsgewicht für Modus mit gewichteter Priorität.

HCB-Registerschnittstelle für Arbitrationsgewichtsregister #1

r_ArbWt0 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **0** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt1 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **1** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt2 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **2** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt3 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **3** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt4 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **4** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt5 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **5** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt6 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **6** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt7 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **7** im gewichteten Arbitrationsmodus verwendet.

[0301] Arbitrationsgewichtsregister #2 – (Offset = 'h68) Das Gewicht für Ports **8**–**15** für Arbitrationsmodus mit gewichteter Priorität.

Bits 0–3 (W/R)	– Port **8** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 4–7 (W/R)	– Port **9** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 8–11 (W/R)	– Port **10** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 12–15 (W/R)	– Port **11** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 16–19 (W/R)	– Port **12** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 20–23 (W/R)	– Port **13** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 24–27 (W/R)	– Port **14** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 28–31 (W/R)	– Port **15** Arbitrationsgewicht für Modus mit gewichteter Priorität.

HCB-Registerschnittstelle für Arbitrationsgewichtsregister #2

r_ArbWt8 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **8** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt9 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **9** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt10 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **10** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt11 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **11** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt12 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **12** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt13 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **13** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt14 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **14** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt15 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **15** im gewichteten Arbitrationsmodus verwendet.

[0302] Arbitrationsgewichtsregister #3 – (Offset = 'h6c) Das Gewicht für Ports **16**–**23** für Arbitrationsmodus mit gewichteter Priorität.

Bits 0–3 (W/R)	– Port **16** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 4–7 (W/R)	– Port **17** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 8–11 (W/R)	– Port **18** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 12–15 (W/R)	– Port **19** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 16–19 (W/R)	– Port **20** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 20–23 (W/R)	– Port **21** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 24–27 (W/R)	– Port **22** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 28–31 (W/R)	– Port **23** Arbitrationsgewicht für Modus mit gewichteter Priorität.

HCB-Registerschnittstelle für Arbitrationsgewichtsregister #3

r_ArbWt16 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **16** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt17 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **17** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt18 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **18** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt19 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **19** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt20 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **20** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt21 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **21** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt22 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **22** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt23 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **23** im gewichteten Arbitrationsmodus verwendet.

[0303] Arbitrationsgewichtsregister #4 – (Offset = 'h70) Das Gewicht für Ports **24**–**28** für Arbitrationsmodus mit gewichteter Priorität.

Bits 0–3 (W/R)	– Port **24** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 4–7 (W/R)	– Port **25** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 8–11 (W/R)	– Port **26** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 12–15 (W/R)	– Port **27** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 16–19 (W/R)	– Port **28** Arbitrationsgewicht für Modus mit gewichteter Priorität.
Bits 20–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

HCB-Registerschnittstelle für Arbitrationsgewichtsregister #4

r_ArbWt24 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **24** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt25 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **25** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt26 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **26** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt27 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **27** im gewichteten Arbitrationsmodus verwendet.
r_ArbWt28 (out)	– Diese vier Bits werden durch den HCB **402** zum Gewichten von Port **28** im gewichteten Arbitrationsmodus verwendet.

HCB 402 Gemischte Steuerung

[0304] HCB Gemischte Steuerung – (Offset = 'h78) Gemischte Steuerungen für den HCB 402. Vorgabe = 32'h0000_0000

Bit 0 (W/R)	– CT FIFO freigeben. 1 = CT FIFO freigegeben.
Bit 1 (W/R)	– Extra Lesewartezustände freigeben. 1 = Wartezustände freigegeben.
Bit 2 (W/R)	– Gleichzeitiges Lesen und Schreiben für Vierfach-Kaskade freigeben.
Bit 3 (W/R)	– Gleichzeitiges Lesen und Schreiben für QE110 freigeben.
Bit 4 (W/R)	– Frühe Adresse freigeben.
Bits 5–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

Port-Abschalten

[0305] Port-Abschalten – (Offset = 'h7c) Bitmap für welche Ports abgeschaltet sind. Vorgabe = 32'h0000_0000.

Bits 0–27 (W/R)	– Bitmap für Ports **0** bis **27**. 1 = Port ist abgeschaltet.
Bits 28–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

Portstatus-Einstellung

[0306] Um den Status eines Ports einzustellen oder zu ändern, müssen zwei Register geschrieben werden. Das erste zu schreibende Register ist das Portstatus-Bitmapregister, das die Bitmap des Ports enthält, der geändert werden wird. Das zweite zu schreibende Register ist das Programm-Portstatusregister, das den Wert des Status enthält und die Programmierung der zwei Portstatusregister einleitet. Der Portstatus der CPU ist immer Befördern und kann niemals geändert werden.

[0307] Portstatus-Bitmapregister – (Offset = 'h90) Bitmap von Ports, deren Staus sich ändern wird. 1 = Ändern dieses Portstatus in einen Wert im Programm-Portstatusregister. Vorgabe = 32'h0000_0000.

Bit 0 (W/R)	– Port **0**. Setzen dieses Bits ermöglicht das Ändern des Status von Port **0**.
Bit 1 (W/R)	– Port **1**. Setzen dieses Bits ermöglicht das Ändern des Status von Port **1**.
:	
:	
Bit 27 (W/R)	– Port **27**. Setzen dieses Bits ermöglicht das Ändern des Status von Port **27**.
Bits 28–31 (RO)	– RESERVIERT. Immer als 0 gelesen.

[0308] Programm-Portstatusregister – (Offset = 'h80) Portstatus. Die CPU schreibt dieses Register, das die Programmierung der Portstatusregister einleitet. Das Portstatus-Bitmapregister muss zuerst beschrieben werden. Vorgabe = 32'h0000_0000.

Bits 0-1 (W/R) - Statuswert. Dieser Wert wird bei Offset 30 in die in der Bitmap angegebenen Ports gelegt.

Statuswert	Bedingung
00b	Abgeschaltet
01b	Gesperrt/Hören
10b	Lernen
11b	Befördern

Bits 2-31 (RO) - RESERVIERT. Immer als 0 gelesen.

[0309] Portstatus #1 Register – (Offset = 'h94) Zustände von Ports **0** bis **15**. Programmiert durch das Programm-Portstatus- und das Portstatus-Bitmapregister. Vorgabe = 32'h0000_0000.

Statuswert	Bedingung
00b	Abgeschaltet
01b	Gesperrt/Hören
10b	Lernen
11b	Befördern

Bits 0-1 (RO) - Port_0_st[1:0]
Bits 2-3 (RO) - Port_1_st[1:0]
Bits 4-5 (RO) - Port_2_st[1:0]
Bits 6-7 (RO) - Port_3_st[1:0]
Bits 8-9 (RO) - Port_4_st[1:0]
Bits 10-11 (RO) - Port_5_st[1:0]
Bits 12-13 (RO) - Port_6_st[1:0]
Bits 14-15 (RO) - Port_7_st[1:0]
Bits 16-17 (RO) - Port_8_st[1:0]
Bits 18-19 (RO) - Port_9_st[1:0]
Bits 20-21 (RO) - Port_10_st[1:0]
Bits 22-23 (RO) - Port_11_st[1:0]
Bits 24-25 (RO) - Port_12_st[1:0]
Bits 26-27 (RO) - Port_13_st[1:0]
Bits 28-29 (RO) - Port_14_st[1:0]
Bits 30-21 (RO) - Port_15_st[1:0]

[0310] Portstatus #2 Register – (Offset = 'h98) Zustände von Ports 16 bis 28. Programmiert durch das Programm-Portstatus- und das Portstatus-Bitmapregister. Vorgabe = 32'h0300_0000

Statuswert	Bedingung
00b	Abgeschaltet
01b	Gesperrt/Hören
10b	Lernen
11b	Befördern

Bits 0-1 (RO) - Port_16_st[1:0]

Bits 2-3 (RO) - Port_17_st[1:0]

Bits 4-5 (RO) - Port_18_st[1:0]

Bits 6-7 (RO) - Port_19_st[1:0]

Bits 8-9 (RO) - Port_20_st[1:0]

Bits 10-11 (RO) - Port_21_st[1:0]

Bits 12-13 (RO) - Port_22_st[1:0]

Bits 14-15 (RO) - Port_23_st[1:0]

Bits 16-17 (RO) - Port_24_st[1:0]

Bits 18-19 (RO) - Port_25_st[1:0]

Bits 20-21 (RO) - Port_26_st[1:0]

Bits 22-23 (RO) - Port_27_st[1:0]

Bits 24-25 (RO) - Port_28_st[1:0] CPU-Port ist immer Befördernd (11)

Bits 26-31 (RO) - RESERVIERT. Immer als 0 gelesen.

MCB-Registerschnittstelle für Portstatus-Einstellregister

SourcePort[7:0] (in)	– Quellenportnummer vom Mcb-Hash-Modul.
m_HashDstprt[7:0] (in)	– Zielportnummer vom Mcb-Hash-Modul.
SrcPrtState[1:0] (out)	– Kombinierte Ausgabe an Mcb-Hash basierend auf Quellenport- und Portstatusregistern.
DstPrtState[1:0] (out)	– Kombinierte Ausgabe an Mcb-Hash basierend auf m_HashDst-Prt- und Portstatusregistern.

Paketspeicherdefinition

[0311] Speichersektor-Informationsregister – (Offset = 'ha0) Paketspeicher besteht aus einer festen Zahl von Sektoren. Dieses Register definiert die Sektorgröße. Die minimale Sektorgröße von 2 KByte stellt sicher, dass das größte Paket (1518 Bytes + Overhead) nicht mehr als eine Sektorgrenzenkreuzung machen kann. Gegenwärtig wird nur die Sektorgröße von 2 KByte unterstützt. Dieses Register kann nur geschrieben werden, wenn Master Switch Enable (EPSM-Einstellregister) negiert ist.

Bits 0–1 (W/R)	– Sektorgröße. Gegenwärtig wird nur die Sektorgröße von 2 KByte unterstützt. 00 = 2 KByte (Vorgabe) 01 = 4 KByte 10 = 8 KByte 11 = 16 KByte
Bits 2–31 (RO)	RESERVIERT. Immer als 0 gelesen.

Spereicherbus-Bandbreitenmonitor

[0312] Speicherbus-Monitorsteuerregister – (Offset = 'ha8) Es gibt zwei unabhängige durch das Register gesteuerte Busmonitoren. Das Monitorauswählbit (24) wird benutzt, um zu wählen, auf welchen Monitor zugegriffen wird. Dieses Bit steuert auch den Zugriff auf das Speicherbusmonitor-Schwellenregister und das Speicherbenutzungsregister. Das Monitorbit kann durch Schreiben nur des hohen Bytes dieses Registers unabhängig gesetzt werden.

Bits 0–9 (W/R)	– Monitormodus [9:0]. Definiert den Typ der zu überwachenden Busaktivtät. Vorgabe ist 10'h3FF (alles überwachen). Zyklustyp (ein oder mehr Bits setzen). Bit 0 – Paket (gesetzt, um paketbezogenen Verkehr zu überwachen). Bit 1 – Hash (gesetzt, um Hash-Lookup-Verkehr zu überwachen). Bit 2 – CPU (gesetzt, um CPU-Zugriffe auf Speicher zu überwachen). Bit 3 – Auffrischung (gesetzt, um Auffrischungszyklen zu überwachen). Pakettyp (muss ein oder beide Bits setzen, wenn Paketbit (0) gesetzt ist). Bit 4 – Unicast (gesetzt, um bekannte Einzel-Adressmoduspakete zu überwachen. Bit 5 – Broadcast (gesetzt, um Pakete mit gesetztem Gruppenbit oder Hash-Miss zu überwachen). Paket Tx/Rx (muss ein oder beide Bits setzen, wenn Paketbit (0) gesetzt ist). Bit 6 – Senden (gesetzt, um sendebezogenen Verkehr zu überwachen). Bit 7 – Empfangen (gesetzt, um empfangsbezogenen Verkehr zu überwachen). Paketdaten/Overhead (muss ein oder beide Bits setzen, wenn Paketbit (0) gesetzt ist). Bit 8 – Daten (gesetzt, um den Datenteil von Paketübertragungen zu überwachen). Bit 9 – Overhead (gesetzt, um den nicht-datenbezogenen Teil von Paketübertragungen zu überwachen, d. h. Bus-Arbitration, Paketspeicherwartung, unbenutzbare Zyklen).
Bits 10–15 (RO)	– Reserviert. Immer als 0 gelesen.
Bits 16–19 (W/R)	– Filterzeitmaßstab. Stellt ungefähre Zeitkonstante für LP-Filterung ein: 0h = 75 ms 4h = 300 ms 8h = Res. Ch = Res. 1h = 600 ms 5h = 2.5 Sek 9h = Res. Dh = Res. 2h = 5 ms 6h = 20 Sek Ah = Res. Eh = Res. 3h = 40 ms 7h = 2.5 Min Bh = Res. Fh = Res. Vorgabe = 0h. Güt nur im Filtermodus.

Bit 20 (W/R)	– Zähl/Filter-Modus (Vorgabe = 0, Filtermodus). 0 = Monitor arbeitet als ein Tiefpassfilter, wie durch Filtermaßstab definiert. Lesen des Busbenutzungsregisters beeinflusst seinen Wert im Filtermodus nicht. 1 = Monitor zählt Buszyklen, filtert aber nicht. Wenn im Zählmodus, wird das Busbenutzungsregister gelöscht, wenn gelesen.
Bit 21 (W/R)	– Timermodus. Gilt nur, wenn im Zählmodus. (Vorgabe = 0). 0 = Nur durch Monitormodusbits definierte Zyklen zählen. 1 = Zähler bei jedem Taktzyklus inkrementieren.
Bit 22 (W/R)	– Rückstau-Freigabe. 1 = Alarm von diesem Monitor verwenden, um Rückstau auf allen Ports freizugeben. Vorgabe = 0, abgeschaltet.
Bit 23 (W/R)	– Broadcast-Steuerfreigabe. 1 = Den Alarm von diesem Monitor verwenden, um von einem Port empfangene Broadcast-Pakete fallen zu lassen. Vorgabe = 0, abgeschaltet.
Bit 24 (W/R)	– Monitorauswahl. 0 = Monitor1 (Vorgabe). 1 = Monitor2.
Bits 25–31 (RO)	– Reserviert. Immer als 0 gelesen.

[0313] Speicherbusmonitor-Schwellen-BW-Register – (Offset = 'hac). Das Monitorauswählbit muss vor dem Zugriff auf dieses Register gesetzt werden.

Bits 0–7 (W/R)	– Alarm-Einstellschwelle. Wenn die Busbenutzung diesen Wert erreicht oder übersteigt, wird das Alarmflag gesetzt und eine CPU-Unterbrechung erzeugt. Rückstau- oder Broadcast-Steuerung wird angewandt, wenn freigegeben. (Vorgabe 08'h00).
Bits 8–15 (W/R)	– Alarm-Löschschwelle. Wenn die Busbenutzung unter diesen Wert fällt, wird das Alarmflag gelöscht und eine CPU-Unterbrechung erzeugt. Rückstau- und Broadcast-Steuerung werden losgelassen (Vorgabe = 8'h00.
Bits 16–23 (RO)	– Spitzen-BW. Max. erfasste Bandbreite seit letztem Lesen. Gelöscht, wenn gelesen.
Bits 24–31 (RO)	- Momentane BW. Momentaner Wert des Busbandbreiten-Benutzungsfilters. Ein Wert von 00h stellt 0% Benutzung dar, und ein Wert von FFh stellt 100% Benutzung dar.

[0314] Speicherbus-Benutzungsregister – (Offset = 'hb0). Das Monitorauswählbit muss vor dem Zugreifen auf dieses Register gesetzt werden.

Bits 0–31 (RO)	– Busbenutzung [31:0]. Speicherbusbenutzungszähler. – Im Zählmodus ist dieser Wert eine Zählung von aktiven Buszyklen seit dem letzten Starten des Zählers. Gelöscht, wenn gelesen. Zähler von beiden Filtern starten gleichzeitig, wenn das Busbenutzungsregister für beide gelesen wurde. – Im Filtermodus ist es nicht nötig, dieses Register zu lesen, da die oberen 8 Bits als momentane BW in das Schwellen-BW-Register kopiert werden. Es ist erwünscht, mehr als 8 Bits für BW zu verwenden. Man beachte, dass der Maximalbandbreitenwert immer 32'hFF00_0000 ist und der Minimalwert abhängig von dem gewählten Zeitmaßstab zwischen 32'h0000-0000 und 32'h00FF_FFFF sein wird. Nicht gelöscht, wenn im Filtermodus gelesen.

MCB-Reisterschnittstelle für Speicherbandbreitenmonitore

Ausgewählte Bandbreite [31:0] (in)	– Speicherbus-Benutzungsregister [31:0] für ausgewählten Monitor. Ferner sind Bits 24–31 momentane BW im Schwellen-BW-Register.
SelectedMaxBW [7:0] (in)	– Spitzen-BW in Schwellen-BW-Registerbits 16–23.
Alarm0 (in)	– Alarmflag für Monitor0. MCB-Register werden Unterbrechungen BWALARMSET0 und BWALARMCLR0 erzeugen, wenn dieses Flag gesetzt oder gelöscht wird.
Alarm1 (in)	– Alarmflag für Monitor1. MCB-Register werden Unterbrechungen BWALARMSET1 und BWALARMCLR1 erzeugen, wenn dieses Flag gesetzt oder gelöscht wird.
r_MonMode0 [9:0] (out)	– Monitormodus für Monitor 0.
r_MonMode1 [9:0] (out)	– Monitormodus für Monitor 1.
r_BwSacle0 [2:0] (out)	– Filter-Zeitmaßstab für Monitor 0.
r_BwSacle1 [2:0] (out)	– Filter-Zeitmaßstab für Monitor 1.
r_CountOnly0 (out)	– Zähl/Filtermodusbit für Monitor 0.
r_CountOnly1 (out)	– Zähl/Filtermodusbit für Monitor 1.
r_Timermode0 (out)	– Timermodusbit für Monitor 0.
r_Timermode1 (out)	– Timermodusbit für Monitor 1.
r_BackPresOnAlarm0 (o)	- Rückstau-Freigabebit für Monitor 0.
r BackPresOnAlarm1 (o)	– Rückstau-Freigabebit für Monitor 1.
r_DropBcPktsOnAlarm0 (o)	– Broadcast-Steuerfreigabebit für Monitor 0.
r_DropBcPktsOnAlarm1 (o)	– Broadcast-Steuerfreigabebit für Monitor 1.
r_FilterSelect (out)	– Monitorauswählbit.
r_AlarmSet0 [7:0] (out)	– Alarmeinstellschwelle für Monitor 0.
r_AlarmSet1 [7:0] (out)	– Alarmeinstellschwelle für Monitor 1.
r_AlarmClr0 [7:0] (out)	– Alarm-Löschschwelle für Monitor 0.
r_AlarmClr1 [7:0] (out)	– Alarm-Löschschwelle für Monitor 1.
ClrBwCtr0 (out)	– Gesetz für einen Takt, wenn das Benutzungsregister für Monitor 0 gelesen wird.
ClrBwCtr1 (out)	– Gesetz für einen Takt, wenn das Benutzungsregister für Monitor 1 gelesen wird.
ClrMaxBW0 (out)	– Gesetzt für einen Takt, wenn das Schwellen-BW-Register für Monitor 0 gelesen wird.
ClrMaxBW1 (out)	– Gesetzt für einen Takt, wenn das Schwellen-BW-Register für Monitor 1 gelesen wird.

Statistik für abgeworfene Pakete

[0315] Pakete, die infolge von Speicherüberlauf, Broadcast-Überlauf, Empfangssektor-Überlauf und Sendesektor-Überlauf abgeworfen werden, werden gezählt. Diese Zählungen und die Bitmaps für den Empfangssektor-Überlauf und Sendesektor-Überlauf werden behalten. Diese Bedingungen bewirken auch Unterbrechungen an der CPU **230**. Die Unterbrechungsinformation wird im MCB-Unterbrechungsquellenregister aufbewahrt.

[0316] Abwerfpaket-Speicherüberlaufregister – (Offset = 'hb8) Dieses Register enthält die Zahl von Paketen, die infolge von Speicherüberlauf abgeworfen wurden, der durch zwei Bedingungen verursacht wird. Diese Bedingungen sind Schwellenüberschreitung während Hash-Lookup und tatsächlicher Speicherüberlauf, wenn ein Paket gespeichert wird, dieser verursacht ein abgebrochenes Paket.
Bits 0-31 (W/R) – Zahl infolge Speicherüberlaufs abgeworfener Pakete.
Abwerfpaket-Broadcast-Überlaufregister – (Offset = 'hbc) Dieses Register enthält die Zahl von Paketen, die infolge von Broadcast-Schwellenüberlauf abgeworfen warfen.
Bits 0–31 (W/R) – Zahl infolge Broadcast-Schwellenüberlaufs abgeworfener Pakete.
Abwerfpaket-Empfangssektor-Überlaufregister – (Offset = 'hd4) Dieses Register enthält die Zahl von Paketen, die infolge von Empfangssektor-Überlauf abgeworfen wurden.
Bits 0–31 (W/R) – Zahl infolge Empfangssektor-Überlaufs abgeworfener Pakete.
Abwerfpaket-Sendesektor-Überlaufregister – (Offset = 'hd8) Dieses Register enthält die Zahl von Paketen, die infolge von Sendesektor-Überlauf abgeworfen wurden.
Bits 0–31 (W/R) – Zahl infolge Sendesektor-Überlaufs abgeworfener Pakete.
Abwerfpaket-Empfangssektor-Bitmapregister – (Offset = 'hdc) Dieses Register enthält die Bitmap von Ports, die Pakete infolge von Empfangssektor-Überlauf abgeworfen haben.
Bits 0–28 (W/R) – Bitmap von Ports, die Überlauf von Empfangssektorgebrauch melden.
Abwertpaket-Sendesektor-Bitmapregister – (Offset = 'he0) Dieses Register enthält die Bitmap von Ports, die Pakete infolge von Sendesektor-Überlauf abgeworfen haben.
Bits 0–28 (W/R) – Bitmap von Ports, die Überlauf von Sendesektorgebrauch melden.

MCB-Registerschnittstelle für Abwerfpaket-Statistik

x_rxPktAbortet_	– Strobe vom XCB, der sagt, wenn Paket ein infolge Speicherüberlaufs abgebrochen wurde.
DropPktStb_MemOF	– Strobe, der sagt, wenn ein Paket abgeworfen wird, weil es den Speicher zum Überlaufen bringt.
DropPktStb_BCOF	– Strobe, der sagt, wenn ein Paket abgeworfen wird, weil die Broadcast-Schwelle überlaufen wird.
DropPktStb_RxOF	– Strobe, der sagt, wenn ein Paket abgeworfen wird, weil die Empfangssektor-Schwelle überlaufen wird.
DropPktStb_TxOF	– Strobe, der sagt, wenn ein Paket abgeworfen wird, weil die Sendesektor-Schwelle überlaufen wird.

Hash-Tabellendefinition

[0317] Hash-Tabellen-Definitionsregister – (Offset = 'hc0) Definiert die Basisadresse und Größe der Haupt-Hash-Eintragstabelle. Wenn mehrfache Kopien der Hash-Tabelle im Speicher gehalten werden, kann dieses Register benutzt werden, um den EPSM **210** zwischen ihnen schalten zu lassen.

Bits 0–14 (RO)	– Primär-Hash-Tabellen-Basisadresse [16:2]. Immer 0.
Bits 15–23 (RO)	– Primär-Hash-Tabellen-Basisadresse [25:17]. Immer 0
Bits 24–25 (W/R)	– Primär-Hash-Tabellengröße [1:0]. (Vorgabe ist 00). 00 = Schlüsselgröße 13 Bits, Tabellengröße 128 KB (8 K 16-Byte Einträge. 01 = Schlüsselgröße 14 Bits, Tabellengröße 256 KB (16 K 16-Byte Einträge. (Basisadressbit 17 wird ignoriert und intern auf 0 gezwungen). 10 = Schlüsselgröße 15 Bits, Tabellengröße 512 KB (32 K 16-Byte Einträge. (Basisadressbits 18–17 werden ignoriert und intern auf 0 gezwungen). 11 = Schlüsselgröße 16 Bits, Tabellengröße 1 MB (64 K 16-Byte Einträge. (Basisadressbits 19–17 werden ignoriert und intern auf 0 gezwungen).
Bit 26 (W/R)	– Hash-Zyklen sperren. Setzen dieses Bits bewirkt, dass Speicherzyklen während eines Hash-Lookup gesperrt werden. Dies minimiert die Hash-Lookup-Zeit auf Kosten des Verzögerns von Paket-Lese- und Schreibübertragungen an den Speicher. Vorgabe ist 0.
Bits 31:27 (RO)	– Reserviert. Immer als 0 gelesen.

MCB-Registerschnittstelle für Hash-Tabellendefinitionsregister

r_HashBaseAdr[25:17] (out)	– Übergibt Basisadresse an Mem-Hash-Modul.
r_HashKeySize[1:0] (out)	– Übergibt Schlüsselgröße an Mem-Hash-Modul.
r_LockHashCycs (out)	– An Mcb-Hash-Modul geltend gemacht, wenn Lock-Hash-Cycles-Bit gesetzt ist.
HashLookupIP (in)	– Gesetzt durch Mcb-Hash-Modul, um anzuzeigen, dass ein Hash-Lookup im Gange ist und alle Schreibungen in das Hash-Tabellendefinitionsregister aufgeschoben werden sollen, bis negiert. MCB-Register können das Register auf jeder steigenden Taktflanke aktualisiern, wenn HashLookUpIP negiert ist.

Quellenwort-Lernen

[0318] Hash-Quellen-Miss-Register-Tief – (Offset = 'hcc) Bytes 3:0 der neuen Quellenadresse sind der Hash-Tabelle hinzufügen. Diese Register werden geladen und eine Unterbrechung wird ausgegeben, wenn eine Hash-SA unbekannt ist oder der Port sich geändert hat und der Quellenport nicht Lernen-Unwirksam gemacht ist. Die Register sind gesperrt, bis das Hash-Quellen-Miss-Reg-Hoch-Register gelesen wird (Tief-Reg muss zuerst gelesen werden). Unbekannte SA's oder Portänderungen, die angetroffen werden, während Register gesperrt sind, werden ignoriert.

Bits 0–7 (RO)	– Byte 0 der der Hash-Tabelle hinzufügenden MAC-Adresse. (Hochwertiges Adressbyte. Gruppenbit = Bit 0).
Bits 8–15 (RO)	– Byte 1 der der Hash-Tabelle hinzufügenden MAC-Adresse.
Bits 16–23 (RO)	– Byte 2 der der Hash-Tabelle hinzufügenden MAC-Adresse.
Bits 24–31 (RO)	– Byte 3 der der Hash-Tabelle hinzufügenden MAC-Adresse.

Hash-Quellen-Miss-Register-Hoch – (Offset = 'hd0) Bytes 5:4 der neuen Quellenadresse und Quellenport-ID

Bits 0–7 (RO)	– Byte 4 der der Hash-Tabelle hinzufügenden MAC-Adresse.
Bits 8–15 (RO)	– Byte 5 der der Hash-Tabelle hinzufügenden MAC-Adresse.
Bits 16–23 (RO)	– Die der Hash-Tabelle hinzufügende Quellenport-ID.
Bits 24–31 (RO)	– Reserviert. Immer als 0 gelsen.

[0319] Lernen-Unwirksam-Portregister – (Offset = 'he4) Bitmapped Lernen-Unwirksam-Portregister. Gilt nicht für CPU.

Bit 0 (W/R)	– Port **0** Lernen Unwirksam. 1 = Unwirksam. Vorgabe = 0.
Bit 1 (W/R)	– Port **1** Lernen Unwirksam. 1 = Unwirksam. Vorgabe = 0.
.....	
Bit 28 (W/R)	– Port **28** Lernen Unwirksam. 1 = Unwirksam. Vorgabe = 0.
Bits 29–31 (RO)	– Reserviert. Immer als 0 gelesen.

MCB-Registerschnittstelle für Quellenport-Lernen

SelectedAdr[47:0] (in)	– Quellenadresse vom Mem-Hash-Modul.
SourcePort[7:0] (in)	– Quellenportnummer vom Mem-Hash-Modul.
SrcMissStb (in)	– Gesetzt durch Mem-Hash-Modul, wenn Hash-SA-Miss aufgetreten ist und SelectedAdr und SourcePort gültig sind. Sollte als Tor zu den Hash-Quellen-Miss-Registern benutzt werden. Memhash wird Haltezeit garantieren.
SrcMissLock (out)	– Geltend gemacht an Memhash, um zu verhindern, dass SrcMissStb geltend gemacht wird.
LearnDisPort (out)	– Geltend gemacht, wenn Lernen-Umwirksam für Port gesetzt. Dies ist eine kombinatorische Ausgabe an MemHash basierend auf dem SourcePort-Eingang und dem Lernen-Umwirksam-Register und wird dauernd bewertet. MemHash weiß, wenn abzutasten ist.

Portsicherheit

[0320] Sicherer-Port-Register – (Offset = 'he8) Bitmapped Sicherer-Port-Register. (Es kann auch erwünscht sein, Lernen-Unwirksam-Bits für Ports mit Sicherheit-Freigegeben zu setzen).

Bit 0 (W/R)	– Port **0** Sicherheit freigegeben. 1 = Freigegeben. Vorgabe = 0.
Bit 1 (W/R)	– Port **1** Sicherheit freigegeben. 1 = Freigegeben. Vorgabe = 0.
.....	
Bit 28 (W/R)	– Port **28** Sicherheit freigegeben. 1 = Freigegeben. Vorgabe = 0.
Bits 29–31 (RO)	– Reserviert. Immer als 0 gelesen.

Sicherheits-Verletzungsregister – (Offset = 'hf0) Bitmapped Sicherheitsverletzung durch Port

[0321] Gelöscht, wenn gelesen. Initialisiert auf 0. Eine Unterbrechung wird ausgegeben, wenn das erste Bits gesetzt ist, und gelöscht, wenn gelesen.

Bit 0 (RO)	– Sicherheitsverletzung Port **0**. 1 = Verletzung aufgetreten.
Bit 1 (RO)	– Sicherheitsverletzung Port **1**. 1 = Verletzung aufgetreten.
.....	
Bit 28 (RO)	– Sicherheitsverletzung Port **28**. 1 = Verletzung aufgetreten.
Bits 29–31 (RO)	– Reserviert. Immer als 0 gelesen.

[0322] Sicherheitsverletzungs-Statistikregister – (Offset = 'hec) Zählung aller Sicherheitsverletzungen auf allen Ports. Gelöscht, wenn gelesen. Initialisiert auf 0.

Bits 0–31 (RO)	– Sicherheitsverletzungszählung [31:0].

MCB-Registerschnittstelle für Portsicherheit

SourcePort[7:0] (in)	– Quellenportnummer vom Mem-Hash-Modul.
SecurePort (out)	– Geltend gemacht, wenn sicherer Modus für Port gesetzt ist. Dies ist ein kombinatorischer Ausgang basierend auf der SourcePort-Eingabe und dem Sicherer-Port-Register und wird ständig bewertet. Mem-Hash weiß, wenn abzutasten ist.
SecViolationStb (in)	– Strobe, der anzeigt, dass eine Sicherheitsverletzung auf dem angegebenen Port aufgetreten ist. Sollte als ein Tor zu dem durch SourcePort angegebenen Sicherheitsverletzungsregisterbit benutzt werden. Mem-Hash wird Haltezeit garantieren.

Speicherkonfiguration

[0323] Speichersteuerregister – (Offset = 'hf4) Gemischte Speichersteuerfunktionen. Dieses Register kann nur geschrieben werden, wenn Master Switch Enable (EPSM-Einstellregister) negiert ist.

Bits 0–1 (W/R)	– Speichertyp 00 = Schnelles Seitenmodus-DRAM (Vorgabe). 01 = EDO-DRAM. 10 = Synchron-DRAM. 11 = Reserviert.
Bit 2 (W/R)	– Speichergeschwindigkeit (0 = 60 ns, 1 = 50 ns). Vorgabe ist 0.
Bit 3 (W/R)	– EDO-Testmodus (1 = Freigabe). Vorgabe ist 0.
Bit 4 (W/R)	– Doppel-Link-Modus. Vorgabe = 0.
Bit 5 (W/R)	– Empfangsseitentreffer unwirksam machen. Vorgabe ist 0.
Bit 6 (W/R)	– Sendeseitentreffer unwirksam machen. Vorgabe ist 0.
Bits 7–31 (RO)	– Reserviert. Immer als 0 gelesen.

MCB-Registerschnittstelle für Speichersteuerregister

r_MemEDO (out)	– Durch mcbregs an memctl-Modul geltend gemacht, wenn Speichertyp 01 ist.
r_MemSync (out)	– Durch mcbregs an memctl-Modul geltend gemacht, wenn Speichertyp 10 ist.
r_Mem50ns (out)	– Durch mcbregs an memctl-Modul geltend gemacht, wenn Speichergeschwindigkeit 1 ist.
r_TestForEDO out)	– Durch mcbregs an memctl-Modul geltend gemacht, wenn EDO-Testmodus 1 ist.

[0324] Speicher-RAS-Auswählregister – (Offset = 'hf8) Definiert, welche RAS-Leitung für jeden 4 M Block an

Speicher geltend zu machen ist. Dieses Register kann nur geschrieben werden, wenn Master Switch Enable (EPSM-Einstellregister) negiert ist.

Bits 0–1 (W/R)	– RAS-Auswahl für 0000000h-03FFFFFh (4 M)
Bits 2–3 (W/R)	– RAS-Auswahl für 0400000h-07FFFFFh (8 M)
Bits 4–5 (W/R)	– RAS-Auswahl für 0800000h0-BFFFFFh (12 M)
Bits 6–8 (W/R)	– RAS-Auswahl für 0C00000h-0FFFFFFh (16 M)
.....	
Bits 30–31 (W/R)	– RAS-Auswahl für 3000000h-3FFFFFFh (64 M)

[0325] RAS-Auswahlen sind wie folgt codiert: 00 = RAS0, 01 = RAS1, 10 = RAS2, 11 = RAS3. Vorgaben sind immer 00.

MCB-Registerschnittstelle für Speicher-RAS-Auswählregister

r_RasSelReg[31:0] (out)	– Übergibt die Daten von mcbregs an memctl-Modul.
Speicherauffrischungs-Zählregister	– (Offset = 'hfc) Definiert die Zahl von CLK-Zyklen zwischen Auffrischungsanforderungen.
Bits 0–9 (W/R)	– Auffrischungszählung [9:0]. Auffrischungszählung mal CLK-Periode muss kleiner oder gleich 15.625 ms sein. Vorgabe ist 208h. (15.60 ms für 30 ns CLK).
Bits 10–31 (RO)	– Reserviert. Immer als 0 gelesen.

MCB-Registerschnittstelle für Speicherauffrischungszählregister

[0326] RefReg (out) – Auffrischungs-Anforderungsstrobe an memctl-Modul. Strobe kann jede Länge haben, da memctl die Anforderung auf der positiven Flanke erfasst. Es wird kein ack zurückgegeben.

MAC-Adressfilterung

[0327] Filterung basierend auf Zieladresse wird bereitgestellt, um Pakete an die und von der CPU **230** zu leiten. Vier Filter werden bereitgestellt, obwohl gegenwärtig nur zwei benötigt werden. Maskieren ist verfügbar, um 'nicht kümmern' in den Adressvergleich einzuschließen, obwohl gegenwärtig kein Bedarf dafür besteht. Zwei Filter sollten eingerichtet werden, eines mit der individuellen Adresse der CPU **230** und das andere mit der BPDU-Multicast-Adresse (für Überspannungsbaum). Wenn ein von einem Port, der nicht die CPU **230** ist, empfangenes Paket eine Filteradresse trifft, wird das Paket an die CPU **230** und nur die CPU **230** befördert (selbst wenn BC oder MC). Wenn ein von der CPU **230** stammendes Paket eine Filteradresse (BPDU-Adresse) trift, wird das Paket an den im Filteradressregister spezifizierten Zielport befördert. Hash-Tabellen-Lookups werden umgangen, wenn ein Paket eine Filteradresse trifft.

Filtersteuerregister	– (Offset = 'h100) Steuert MAC-Zieladressenfilterung.
Bits 0–3 (W/R)	– Adressfilter-Freigaben [3:0]. 1 = Einzel-Zieladressenfilterung für entsprechendes Adressfilterregister [3:0] freigeben. Vorgabe 0.
Bits 4–7 (W/R)	– Adressmasken-Freigaben [3:0]. 1 = Maskieren freigeben, wenn das Adressfilterregister [3:0], mit dem Adressfilter-Maskenregister. Vorgabe 0.
Filtermaskenregister Tief	– (Offset = 'h104) Vorgabe = 0.
Bits 0–7 (W/R)	– Byte 0 der MAC-Adressmaske (1 = Adressbit maskieren).
Bits 8–15 (W/R)	– Byte 1 der MAC-Adressmaske (1 = Adressbit maskieren).
Bits 16–23 (W/R)	– Byte 2 der MAC-Adressmaske (1 = Adressbit maskieren).
Bits 24–31 (W/R)	– Byte 3 der MAC-Adressmaske (1 = Adressbit maskieren).
Filtermaskenregister Hoch	– (Offset = 'h108) Vorgabe 0.
Bits 0–7 (W/R)	– Byte 4 der MAC-Adressmaske (1 = Adressbit maskieren).
Bits 8–15 (W/R)	– Byte 5 der MAC-Adressmaske (1 = Adressbit maskieren).
Bits 16–31 (RO)	– Reserviert. Immer als 0 gelesen.
Filteradressregister 0 Tief	– (Offset = 'h10c)
Bits 0–7 (W/R)	– Byte 0 der zu befördernden MAC-Adresse.
Bits 8–15 (W/R)	– Byte 1 der zu befördernden MAC-Adresse.
Bits 16–23 (W/R)	– Byte 2 der zu befördernden MAC-Adresse.
Bits 24–31 (W/R)	– Byte 3 der zu befördernden MAC-Adresse.
Filteradressregister 0 Hoch	– (Offset = 'h110)
Bits 0–7 (W/R)	– Byte 4 der zu befördernden MAC-Adresse.
Bits 8–15 (W/R)	– Byte 5 der zu befördernden MAC-Adresse.
Bits 16–23 (W/R)	– Zielport. Wenn der Quellenport die CPU **230** ist, spezifiziert dieses Feld, an welchen Port das Paket befördert werden soll, wenn die MAC-Adresse mit der Filteradresse übereinstimmt. Wenn der Quellenport nicht die die CPU **230** ist, wird dieses Feld ignoriert, und Treffer auf die Filter-MAC-Adresse werden an die CPU **230** befördert.
Bits 24–31 (RO)	– Reserviert. Immer als 0 gelesen.
Filteradressregister 1 Tief	– (Offset = 'h114) siehe oben.
Filteradressregister 2 Tief	– (Offset = 'h11c) siehe oben.
Filteradressregister 1 Hoch	– (Offset = 'h118) siehe oben.
Filteradressregister 2 Hoch	– (Offset = 'h120) siehe oben.
Filteradressregister 3 Tief	– (Offset = 'h124) siehe oben.
Filteradressregister 3 Hoch	– (Offset = 'h128) siehe oben.

MCB-Registerschnittstelle für Adressfilterung

SelectedAdr[47:0] (in)	– Zieladresse von Memhash-Modul.
FilterHit (out)	– Geltend gemacht, wenn ein Filteradresstreffer auftritt. Dies ist ein kombinatorischer Ausgang an Memhash basierend auf der SelectedAdr und den Filterregistern und wird ständig bewertet. Memhash weiß, wenn abzutasten ist.
FilterPort[7:0] (out)	– Wenn der Quellenport die CPU **230** ist, ist FilterPort gleich dem Zielportfeld von dem Filterregister, das einen Filtertraffer erzeugt. Wenn der Quellenport nicht die CPU **230** ist, ist FilterPort gleich CpuPort (von dem EPSM-Einstellregister).
SourcePort[7:0] (in)	– Quellenportnummer von Memhash-Modul.
SrcPrtIsCpu	– Geltend gemacht, wenn SourcePort mit Cpu-Port-Nummer im EPSM-Einstellregister übereinstimmt.

MCB-Unterbrechungsinformation

[0328] Es gibt acht Unterbrechungsquellen in dem MCB **404**. Die Unterbrechungsquellen werden bewirken, dass die CPU **230** unterbrochen wird, wenn sie nicht maskiert werden. Damit die Information der Unterbrechungsquelle verfügbar sein kann, ohne die CPU **230** zu unterbrechen, steht ein Abfragemechanismus zur Verfügung. Die Maskierung einer Unterbrechungsquelle bewirkt, dass die Unterbrechungen von der CPU **230** ferngehalten werden, aber die Information noch in dem Abfragequellenregister verfügbar ist.

[0329] MCB-Unterbrechungsquellenregister – (Offset = 'h12c) Quelle der an die CPU **230** gesendeten Unterbrechung. Dieses Register wird durch den EPSM **210** aktualisiert, und dann wird eine Unterbrechung an die CPU **230** gesendet. Wenn die CPU **230** dieses Register liest, wird der Inhalt gelöscht. Ein Wert von 1 in einem Bit zeigt an, dass eine Unterbrechung aufgetreten ist. Vorgabe = 32'h0000_0000.

Bit 0 (W/R)	– Sicherheitsunterbrechung. Wenn eine Sicherheitsverletzung stattfindet, erscheint diese Unterbrechung.
Bit 1 (W/R)	– Speicherüberlauf gesetzt. Wenn der Speicher sich mit Paketen füllt und die Überlaufschwelle durchschritten wird, erscheint diese Unterbrechung.
Bit 2 (W/R)	– Speicherüberlauf gelöscht. Wenn der Speicher sich leert und die Überlaufschwelle durchschritten wird, erscheint diese Unterbrechung.
Bit 3 (W/R)	– Broadcast-OF gesetzt. Wenn die Broadcast-Pakete den Speicher füllen und die Broadcast-Schwelle durchschritten wird, erscheint diese Unterbrechung.
Bit 4 (W/R)	– Broadcast-OF gelöscht. Wenn die Broadcast-Pakete aus dem Speicher ausgeleert werden und die Broadcast-Schwelle durchschritten wird, erscheint diese Unterbrechung.
Bit 5 (W/R)	– Empfangs-OF. Wenn ein Port seinen zugeteilten Platz zum Empfangen von Paketen überschreiten wird, erscheint diese Unterbrechung.
Bit 6 (W/R)	– Sende-OF. Wenn ein Port, der Pakete sendet, seinen zugeteilten Platz überschreiten wird, erscheint diese Unterbrechung.
Bit 7 (W/R)	– Rx-Paket abgebrochen. Wenn ein Paket begonnen hat, gespeichert zu werden und festgestellt wird, dass der Speicher überschritten wird, wird das Paket abgebrochen und diese Unterbrechung erscheint.
Bits 8–31 (RO)	– Reserviert. Immer als 0 gelesen.

MCB-Registerschnittstelle für Unterbrechungsquellenregister

[0330] Unterbrechungs-Maskierungsregister – (Offset = 'h130) Unterbrechungen, die durch die CPU **230** zu maskieren sind. Ein Wert von 1 in einem Bit zeigt an, dass eine Unterbrechung maskiert ist. Vorgabe = 32'h0000_0000.

Bit 0 (W/R)	– Maske für die Sicherheitsunterbrechung.
Bit 1 (W/R)	– Maske für die Speicherüberlauf-Gesetzt-Unterbrechung.
Bit 2 (W/R)	– Maske für die Speicherüberlauf-Gelöscht-Unterbrechung.
Bit 3 (W/R)	– Maske für die Broadcast-OF-Gesetzt-Unterbrechung.
Bit 4 (W/R)	– Maske für die Broadcast-OF-Gelöscht-Unterbrechung.
Bit 5 (W/R)	– Maske für die Empfangs-OF-Unterbrechung.
Bit 6 (W/R)	– Maske für die Sende-OF-Unterbrechung.
Bit 7 (W/R)	– Maske für die Rx-Paket-Abgebrochen-Unterbrechung.
Bits 8–31 (RO)	– Reserviert. Immer als 0 gelesen.

[0331] Abfragequellenregister – (Offset = 'h134) Dieses Register enthält die maskierte Unterbrechungsinformation und wird durch die CPU **230** gelöscht, die eine eins schreibt, um die gewünschten Bits zu löschen. Dies erlaubt der CPU **230**, abzufragen, anstatt unterbrochen zu werden. Die CPU wird jede Unterbrechungsquelle zu maskieren haben, die sie stattdessen abzufragen wünschen würde.

Bit 0 (W/R)	– Sicherheitsunterbrechung. Wenn eine Sicherheitsverletzung stattfindet, erscheint diese Unterbrechung.
Bit 1 (W/R)	– Speicherüberlauf gesetzt. Wenn der Speicher sich mit Paketen füllt und die Überlaufschwelle durchschritten wird, erscheint diese Unterbrechung.
Bit 2 (W/R)	– Speicherüberlauf gelöscht. Wenn der Speicher sich leert und die Überlaufschwelle durchschritten wird, erscheint diese Unterbrechung.
Bit 3 (W/R)	– Broadcast-OF gesetzt. Wenn die Broadcast-Pakete den Speicher füllen und die Broadcast-Schwelle durchschritten wird, erscheint diese Unterbrechung.
Bit 4 (W/R)	– Broadcast-OF gelöscht. Wenn die Broadcast-Pakete aus dem Speicher ausgeleert werden und die Broadcast-Schwelle durchschritten wird, erscheint diese Unterbrechung.
Bit 5 (W/R)	– Empfangs-OF. Wenn ein Port seinen zugeteilten Platz zum Empfangen von Paketen überschreiten wird, erscheint diese Unterbrechung.
Bit 6 (W/R)	– Sende-OF. Wenn ein Port, der Pakete sendet, seinen zugeteilten Platz überschreiten wird, erscheint diese Unterbrechung.
Bit 7 (W/R)	– Rx-Paket abgebrochen. Wenn ein Paket begonnen hat, gespeichert zu werden und festgestellt wird, dass der Speicher überschritten wird, wird das Paket abgebrochen und diese Unterbrechung erscheint.
Bits 8–31 (RO)	– Reserviert. Immer als 0 gelesen.

MCB-Registerschnittstelle für Abfragequellenregister
Rückstau

Rückstau-Freigabe	– (Offset = 'h138) Bitmap zum Freigeben von Rückstau.
Bits 0–23 (RO)	– Reserviert. Immer als 0 gelesen.
Bits 24–27 (W/R)	- Bitmap.
Bits 28–31 (RO)	– Reserviert. Immer als 0 gelsen.

Port-Bondierung

[0332] Es gibt zwei Sätze von bondierten Ports. Daher gibt es zwei Register, um zu sagen, welche Ports miteinander bondiert sind. Anmerkung: Nur zwei Bits in jedem Register sollten gesetzt werden, das heißt, nicht mehr als zwei Port sollten miteinander bondiert werden.

Bondierter Portsatz 0	– (Offset = 'h13c) Diese Bitmap sagt, welche Ports in diesem Statz miteinander bondiert sind.
Bits 0–27 (W/R)	– Bitmap für Satz 0.
Bits 28–31 (R0)	– Reserviert. Immer als 0 gelesen.
Bondierter Portsatz 1	– (Offset = 'h140) Diese Bitmap sagt, welche Ports in diesem Statz miteinander bondiert sind.
Bits 0–27 (W/R)	– Bitmap für Satz 1.
Bits 28–31 (RO)	– Reserviert. Immer als 0 gelesen.

VLAN

Vorgabe VLAN-Register – (Offset = 'h144).

[0333] Nun ist einzusehen, dass ein Multiport-Abfragesystem für einen Netzwerkschalter ein effizientes System zum Bestimmen des Empfangs- und Sendestatus für eine Vielzahl von Netzwerkports bereitstellt. Eine Abfragelogik macht periodisch ein einziges Abfragesignal geltend und empfängt eine Vielzahl von Sende- und Empfangsstatussignalen, um so den Status von mehrfachen Ports zu einer Zeit zu empfangen. Die Abfragelogik aktualisiert Sende- und Empfangslisten entsprechend einer fortlaufenden Verfolgung des Statusses aller Ports. Dies ermöglicht einer Arbitrations- und Steuerlogik, die die Listen durchsieht, zu bestimmen, wenn Daten von einem Quellenport zurückzugewinnen sind und wenn Daten an einen Port zum Senden zu liefern sind.

[0334] Obwohl ein erfindungsgemäßes System und Verfahren in Verbindung mit der bevorzugten Ausführung beschrieben wurde, ist es nicht gedacht, auf die hierin dargelegte Form begrenzt zu sein, sondern ist im Gegenteil gedacht, solche Alternativen, Modifikationen und Gleichwertigkeiten einzuschließen, wie sie vernünftigerweise im Umfang der Erfindung, wie in den anliegenden Ansprüchen definiert, enthalten sein können.

Patentansprüche

1. Netzwerk-Switch (**102**), der umfasst:
eine Vielzahl von Netzwerk-Ports (**104**) zum Empfangen und Senden von Daten, die jeweils enthalten:
eine Netzwerk-Schnittstelle;
eine Datenbus-Schnittstelle (**202**); und
eine Prozessor-Port-Schnittstelle (**202**);
wobei der Netzwerk-Switch des Weiteren einen Datenbus (**206**) umfasst, der mit der Datenbus-Schnittstelle jedes der Vielzahl von Netzwerk-Ports (**104**) gekoppelt ist;
einen Prozessor (**230**), der mit einem Prozessorbus (**218**) gekoppelt ist, der mit der Prozessor-Port-Schnittstelle jedes der Vielzahl von Netzwerk-Ports gekoppelt ist;
einen Speicher (**212**), der mit einem Speicherbus (**214**) gekoppelt ist;
wobei der Netzwerk-Switch gekennzeichnet ist durch:
einen Switch-Manager (**210**), der mit dem Datenbus (**206**), mit dem Prozessorbus (**218**) und mit einem weiteren Prozessorbus (**204**), der mit jedem der Vielzahl von Netzwerk-Ports (**104**) gekoppelt ist, sowie mit dem Speicherbus (**214**) gekoppelt ist, um Datenstrom zwischen der Vielzahl von Netzwerk-Ports (**104**) und dem Speicher (**212**) zu steuern und den Prozessor (**230**) in die Lage zu versetzen, auf die Vielzahl von Netzwerk-Ports (**104**) und den Speicher (**212**) zuzugreifen; wobei der Switch-Manager (**210**) umfasst:
eine Datenbus-Schnittstelle (**410**), die mit dem Datenbus (**206**) gekoppelt ist und enthält;

eine Abfrage-Logik (**501**), die periodisch abfragt, um den Status jedes der Vielzahl von Netzwerk-Ports (**104**) zu bestimmen; und
eine Steuer-Logik (**504**), die mit der Abfrage-Logik (**501**) gekoppelt ist, wobei die Steuer-Logik so betrieben werden kann, dass sie Datenstrom zwischen der Vielzahl von Netzwerk-Ports (**104**), dem Prozessor (**230**) und einer Speicherbus-Schnittstelle (**422**) steuert;
wobei die Speicherbus-Schnittstelle (**422**) mit dem Speicherbus (**214**) und der Datenbus-Schnittstelle (**410**) gekoppelt ist; und
eine Prozessorbus-Schnittstelle (**432**), die mit dem Prozessorbus (**204**), der Datenbus-Schnittstelle und der Speicherbus-Schnittstelle (**422**) gekoppelt ist.

2. Netzwerk-Switch nach Anspruch 1, wobei die Datenbus-Schnittstelle (**410**) umfasst:
einen Empfangs-Puffer (**520**, **522**) zum Empfangen und temporären Speichern von Daten von der Vielzahl von Netzwerk-Ports (**104**);
einen Sende-Puffer (**524**, **526**) zum Empfangen und temporären Speichern von Daten von der Speicherbus-Schnittstelle (**414**);
wobei die Steuer-Logik (**504**), der Empfangs-Puffer (**520**, **522**) und der Sende-Puffer (**524**, **526**) so betrieben werden können, dass sie Datenstrom zwischen der Vielzahl von Netzwerk-Ports (**104**), dem Prozessor (**230**) und der Speicherbus-Schnittstelle (**414**) steuern.

3. Netzwerk-Switch nach Anspruch 1 oder Anspruch 2, wobei die Speicherbus-Schnittstelle (**414**) enthält:
eine Speicher-Steuerung (**636**), die mit dem Speicherbus (**214**) gekoppelt ist, um Speicherzyklen zu steuern; und
einen Zuteiler (**638**), der mit der Speicher-Steuerung (**636**), der Datenbus-Schnittstelle (**418**,**410**) und der Prozessorbus-Schnittstelle (**424**) verbunden ist, um Zugriff auf den Speicher (**212**) über die Speicher-Steuerung (**636**) zu steuern.

4. Netzwerk-Switch nach Anspruch 3, wobei die Speicherbus-Schnittstelle des Weiteren enthält:
eine Empfangs-Steuerung (**604**), die mit der Datenbus-Schnittstelle (**418**, **410**) und der Speicher-Steuerung (**636**) verbunden ist, um Datenstrom von der Datenbus-Schnittstelle zu dem Speicher zu steuern; und
eine Sende-Steuerung (**606**), die mit der Datenbus-Schnittstelle (**418**, **410**) und der Speicher-Steuerung (**636**) gekoppelt ist, um Datenstrom von dem Speicher zu der Datenbus-Schnittstelle (**418**, **410**) zu steuern.

5. Netzwerk-Switch nach Anspruch 1, wobei die Prozessorbus-Schnittstelle (**432**) einschließt:
dass der Prozessorbus (**218**) einen Prozessor-Abschnitt (**218**), der zwischen den Switch-Manager (**210**) und den Prozessor (**230**) gekoppelt ist, sowie einen Port-Abschnitt (**204**) enthält, der zwischen den Switch-Manager (**210**) und die Prozessor-Port-Schnittstelle (**202**) jedes der Vielzahl von Netzwerk-Ports gekoppelt ist;
eine Prozessor-Schnittstelle (**432**), die mit dem Prozessor (**230**) über den Prozessor-Abschnitt (**218**) des Prozessorbusses gekoppelt ist; und
eine Port-Schnittstelle, die mit der Prozessor-Schnittstelle (**432**) und jedem der Vielzahl von Netzwerk-Ports (**104**) über den Port-Abschnitt (**204**) des Prozessorbusses gekoppelt ist.

6. Netzwerk-Switch nach Anspruch 5, wobei die Prozessorbus-Schnittstelle (**434**) eine Bustransfer-Logik zum Umsetzen von Zyklen zwischen dem Prozessor-Abschnitt (**218**) und dem Port-Abschnitt (**204**) des Prozessorbusses enthält.

7. Netzwerk-Switch nach Anspruch 5, wobei die Prozessorbus-Schnittstelle (**432**) des Weiteren umfasst:
einen ersten Empfangs-Puffer (**710**), der mit der Prozessor-Schnittstelle (**432**) und der Datenbus-Schnittstelle (**436**, **410**) gekoppelt ist;
einen ersten Sende-Puffer (**712**), der mit der Prozessor-Schnittstelle (**432**) und der Datenbus-Schnittstelle (**436**, **410**) gekoppelt ist;
eine erste Steuerung (**713**), die mit der Datenbus-Schnittstelle (**436**, **410**), der Prozessor-Schnittstelle, dem ersten Empfangs-Puffer (**710**) und dem ersten Sende-Puffer (**712**) gekoppelt ist, um Datenstrom zwischen der Prozessorbus-Schnittstelle (**432**) und der Datenbus-Schnittstelle (**436**, **410**) zu steuern;
einen zweiten Empfangs-Puffer (**706**), der mit der Prozessor-Schnittstelle (**432**) und der Speicherbus-Schnittstelle (**426**, **422**) gekoppelt ist;
einen zweiten Sende-Puffer (**708**), der mit der Prozessor-Schnittstelle (**432**) und der Speicherbus-Schnittstelle (**426**, **422**) gekoppelt ist; und
eine zweite Steuerung (**718**), die mit der Speicherbus-Schnittstelle (**426**, **422**), der Prozessor-Schnittstelle (**432**), dem zweiten Empfangs-Puffer (**706**) und dem zweiten Sende-Puffer (**708**) gekoppelt ist, um Datenstrom zwischen der Prozessorbus-Schnittstelle (**432**) und der Speicherbus-Schnittstelle (**426**, **422**) zu steuern.

8. Netzwerk-Switch nach einem der Ansprüche 1 bis 7, wobei jeder der Vielzahl von Netzwerk-Ports (**104**) des Weiteren enthält:
eine Vielzahl von Statistik-Zählern (**303**), die mit dem Prozessorbus (**204**) gekoppelt sind, wobei jeder der Vielzahl von Statistik-Zählern (**303**) Status und Funktion eines entsprechenden Ports (**104**) verfolgt.

9. Netzwerk-Switch nach einem der Ansprüche 1 bis 8, wobei der Prozessorbus einen Prozessor-Abschnitt (**218**), der zwischen den Switch-Manager (**210**) und den Prozessor (**230**) gekoppelt ist, sowie einen Port-Abschnitt (**204**) enthält, der zwischen den Switch-Manager (**210**) und jeden der Vielzahl von Netzwerk-Ports (**104**) gekoppelt ist,
wobei der Prozessor-Abschnitt (**218**) des Prozessorbusses, der Datenbus (**206**) und der Speicherbus (**214**) jeweils 32-Bit-Busse umfassen und der Port-Abschnitt (**204**) des Prozessor-Busses einen 16-Bit-Bus umfasst.

10. Netzwerk-Switch nach einem der Ansprüche 2 bis 8, wobei die Vielzahl von Ports (**104**, **110**) umfassen:
eine erste Vielzahl von Ports, die entsprechend einem ersten Protokoll arbeiten;
eine zweite Vielzahl von Ports (**110**), die entsprechend einem zweiten Protokoll arbeiten;
einen zweiten Datenbus (**222**), der mit der zweiten Vielzahl von Ports (**110**) und dem Prozessor (**230**) gekoppelt ist; und
eine Brückenvorrichtung (**220**), die mit dem Datenbus (**206**) und mit dem zweiten Datenbus (**222**) gekoppelt ist.

11. Netzwerk-System, das umfasst:
eine Vielzahl von Netzwerken, die jeweils wenigstens eine Datenvorrichtung zum Senden und Empfangen von Datenpaketen enthalten; und
einen Netzwerk-Switch nach Anspruch 1, der mit dem Datenbus, dem Prozessorbus und dem Speicherbus gekoppelt ist, um Datenstrom zwischen der Vielzahl von Netzwerk-Ports und dem Speicher zu steuern und den Prozessor in die Lage zu versetzen, auf die Vielzahl von Netzwerk-Ports und den Speicher zuzugreifen.

Es folgen 44 Blatt Zeichnungen

Anhängende Zeichnungen

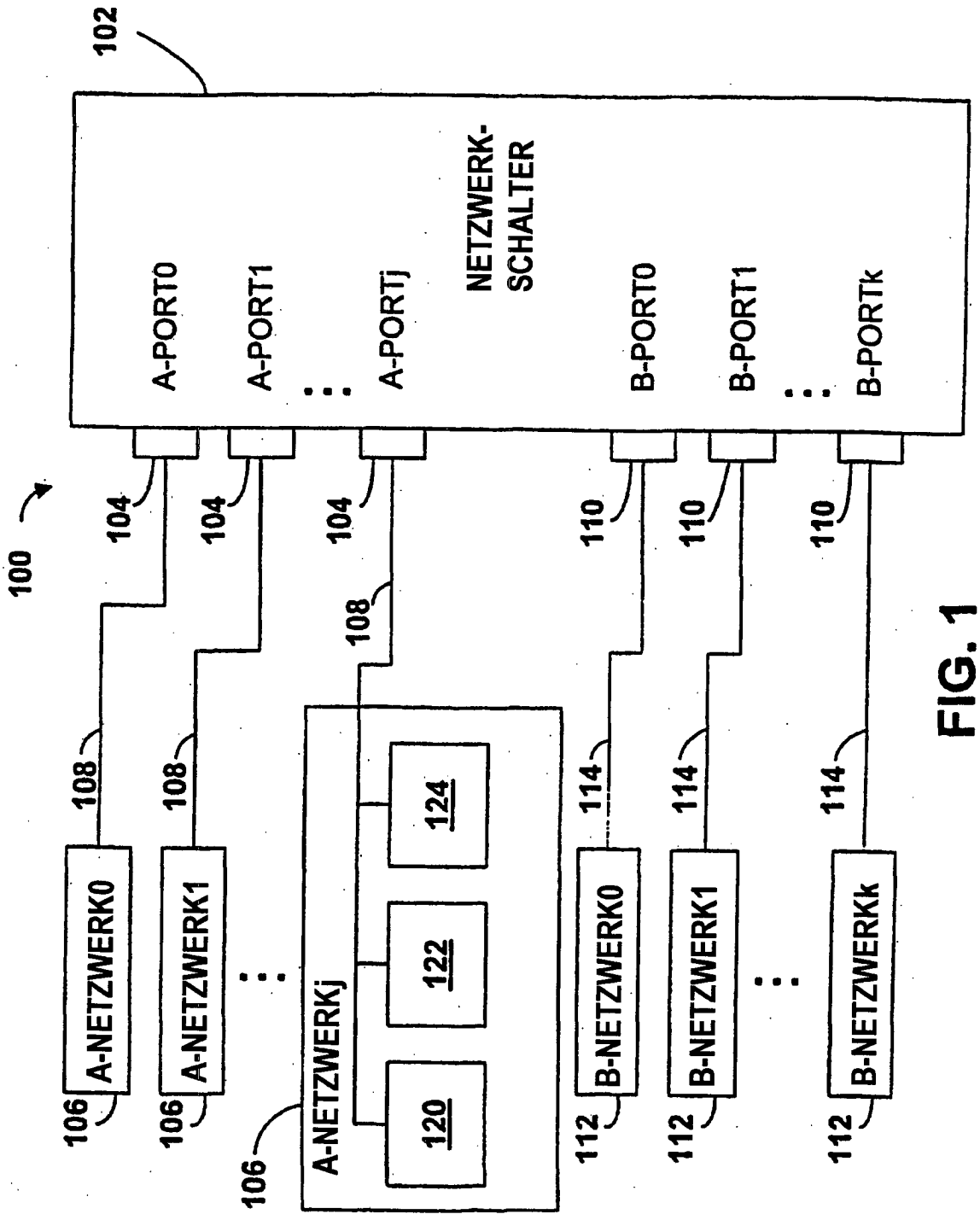


FIG. 1

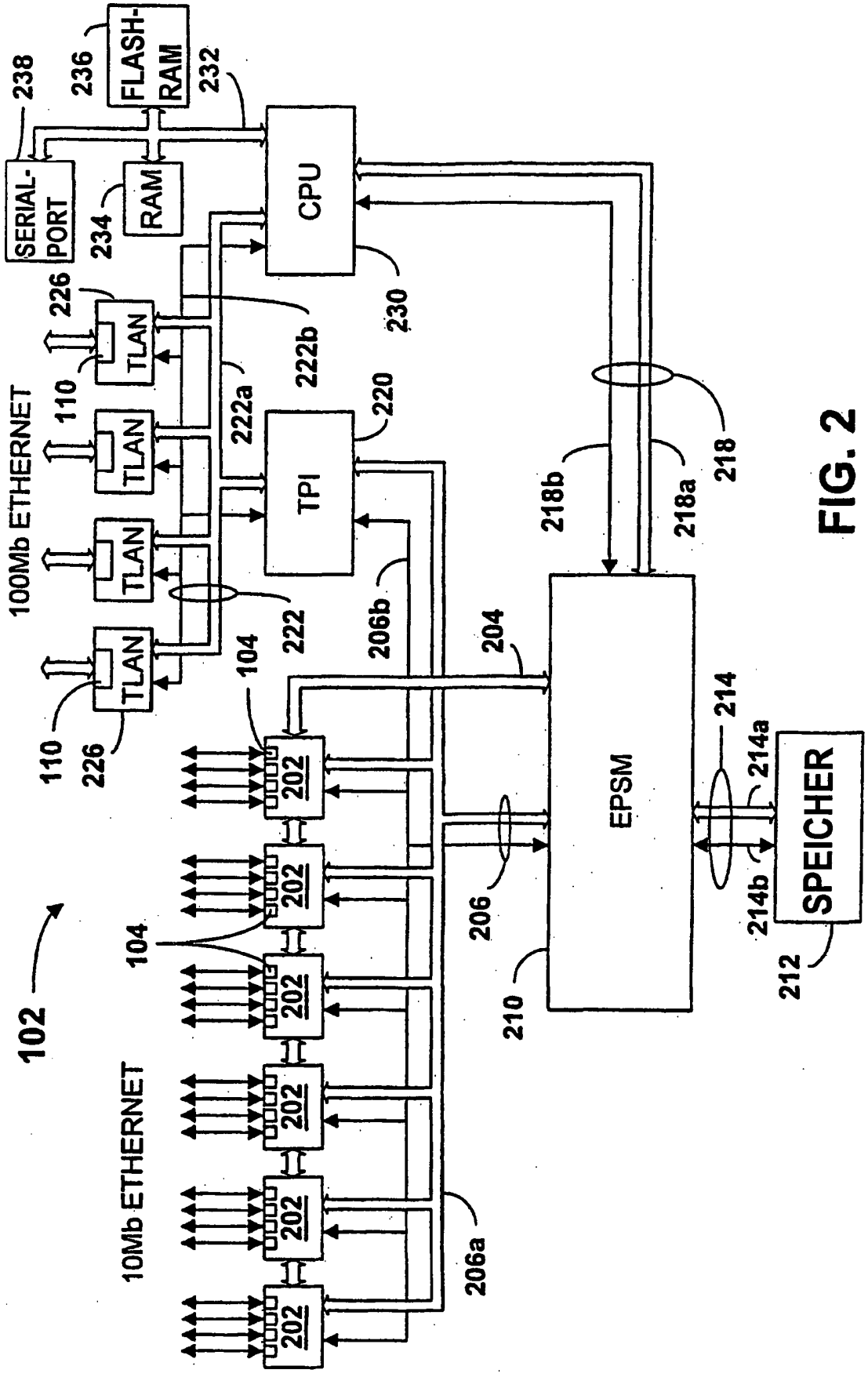

102
100Mb ETHERNET
10Mb ETHERNET
110
226
TLAN
104
202
222
222a
222b
TPI
220
206b
204
206
206a
210
EPSM
214
214a
214b
SPEICHER
212
218
218a
218b
CPU
230
232
FLASH-RAM
236
RAM
234
SERIAL-PORT
238

FIG. 2

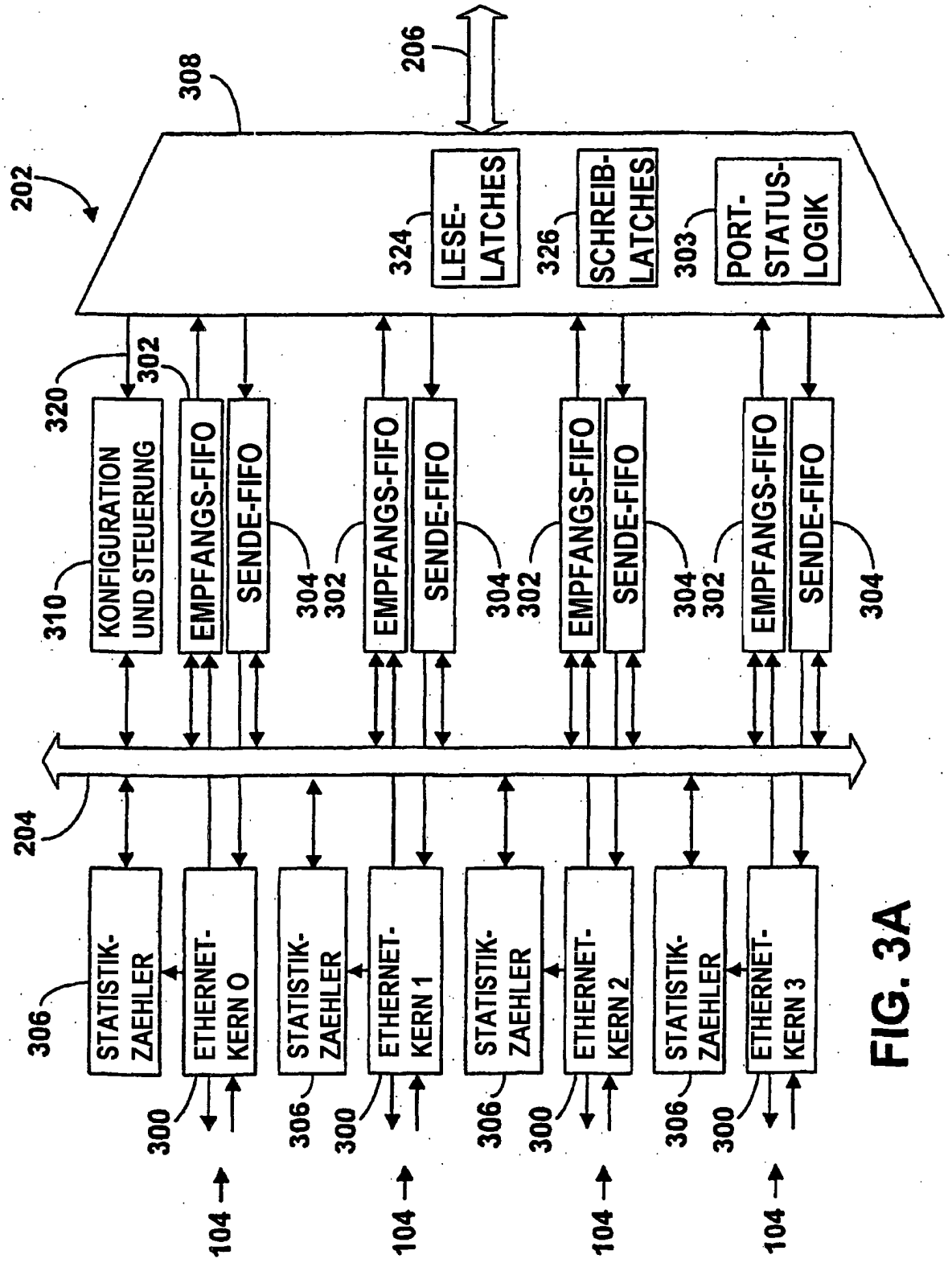

202
206
308
324
LESE-LATCHES
326
SCHREIB-LATCHES
303
PORT-STATUS-LOGIK
320
302
310
KONFIGURATION UND STEUERUNG
EMPFANGS-FIFO
SENDE-FIFO
304
204
306
STATISTIK-ZAEHLER
ETHERNET-KERN 0
ETHERNET-KERN 1
ETHERNET-KERN 2
ETHERNET-KERN 3
300
104

FIG. 3A

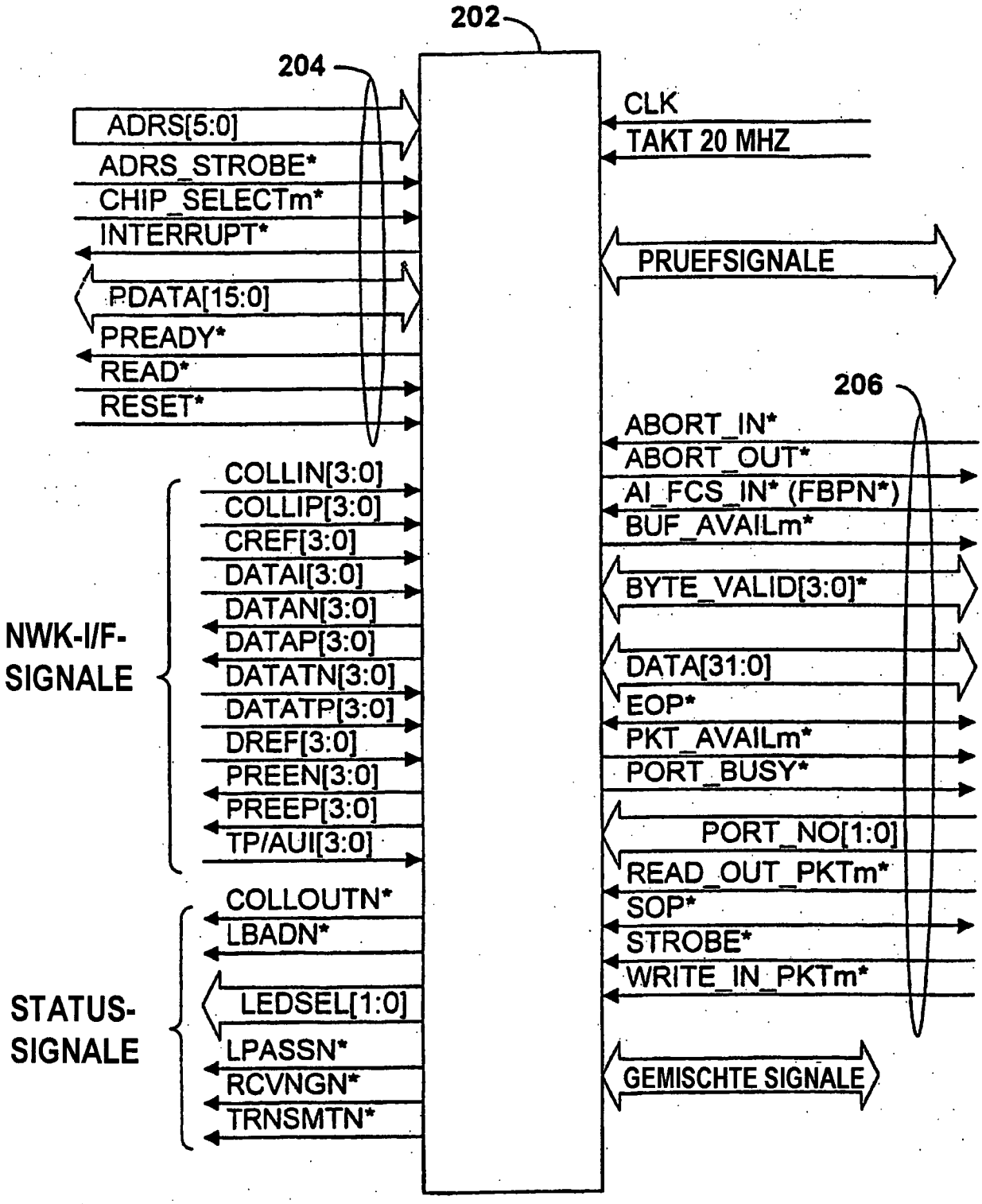

202
204
ADRS[5:0]
ADRS_STROBE*
CHIP_SELECTm*
INTERRUPT*
PDATA[15:0]
PREADY*
READ*
RESET*
CLK
TAKT 20 MHZ
PRUEFSIGNALE
206
ABORT_IN*
ABORT_OUT*
AI_FCS_IN* (FBPN*)
BUF_AVAILm*
BYTE_VALID[3:0]*
DATA[31:0]
EOP*
PKT_AVAILm*
PORT_BUSY*
PORT_NO[1:0]
READ_OUT_PKTm*
SOP*
STROBE*
WRITE_IN_PKTm*
GEMISCHTE SIGNALE
NWK-I/F-SIGNALE
COLLIN[3:0]
COLLIP[3:0]
CREF[3:0]
DATAI[3:0]
DATAN[3:0]
DATAP[3:0]
DATATN[3:0]
DATATP[3:0]
DREF[3:0]
PREEN[3:0]
PREEP[3:0]
TP/AUI[3:0]
STATUS-SIGNALE
COLLOUTN*
LBADN*
LEDSEL[1:0]
LPASSN*
RCVNGN*
TRNSMTN*

FIG. 3B

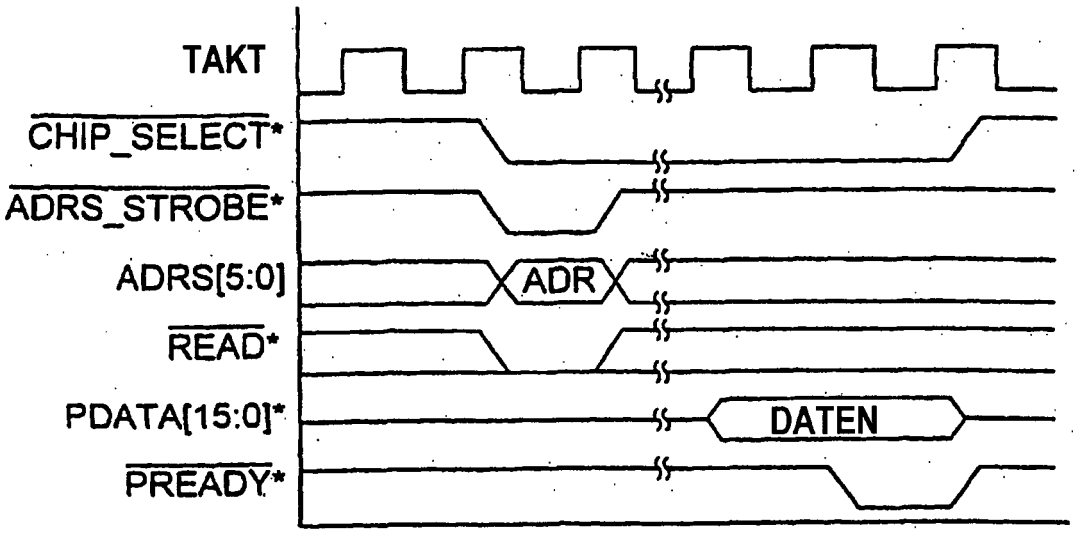

TAKT
CHIP_SELECT*
ADRS_STROBE*
ADRS[5:0]
ADR
READ*
PDATA[15:0]*
DATEN
PREADY*

FIG. 3C

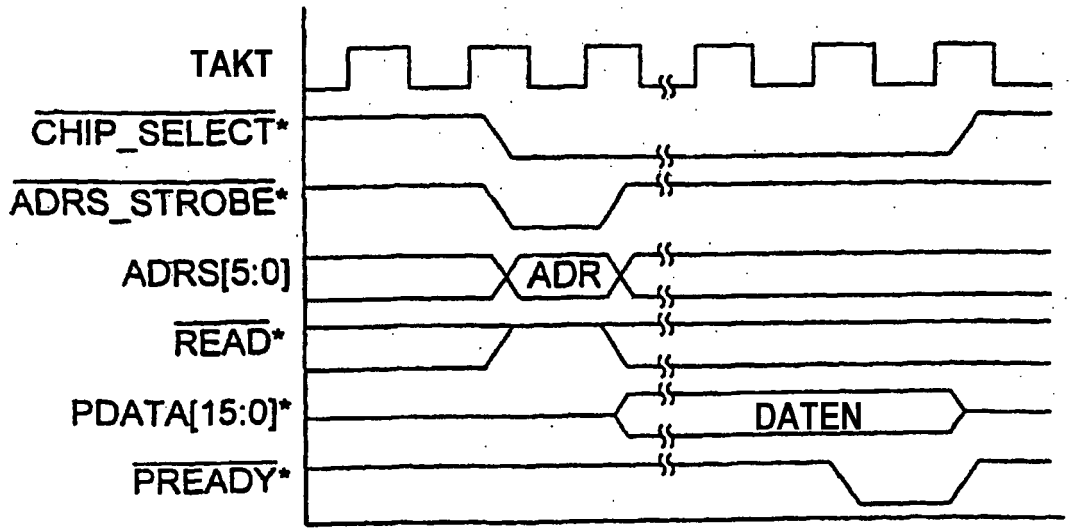

TAKT
CHIP_SELECT*
ADRS_STROBE*
ADRS[5:0]
ADR
READ*
PDATA[15:0]*
DATEN
PREADY*

FIG. 3D

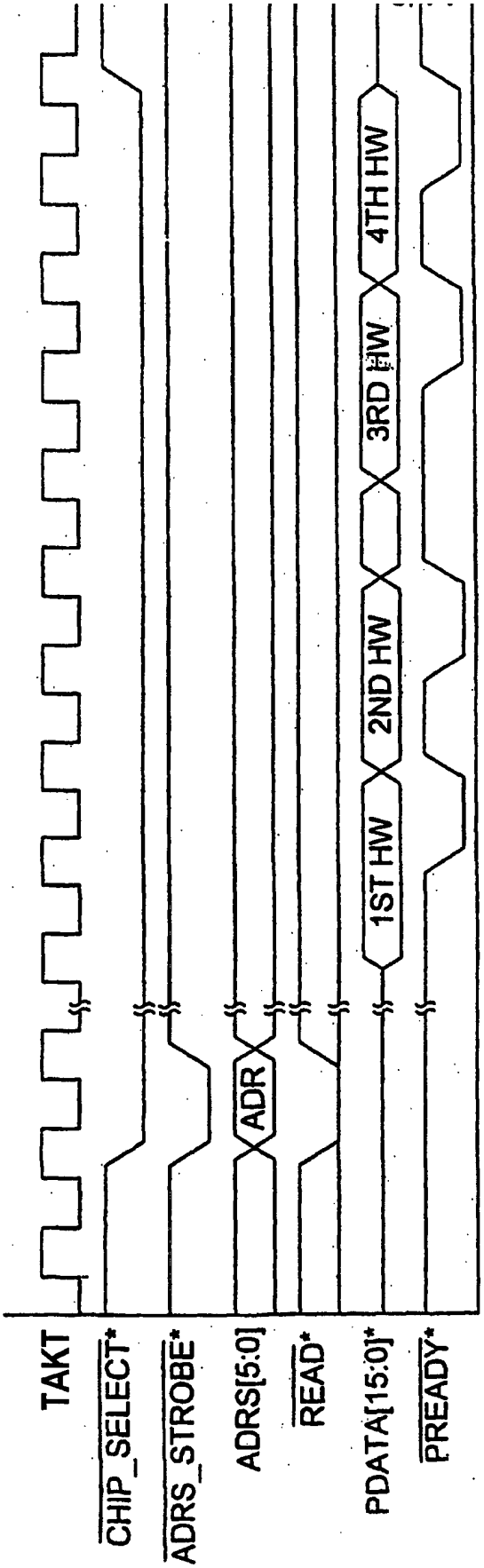

TAKT
CHIP_SELECT*
ADRS_STROBE*
ADRS[5:0]
ADR
READ*
PDATA[15:0]*
1ST HW
2ND HW
3RD HW
4TH HW
PREADY*

FIG. 3E

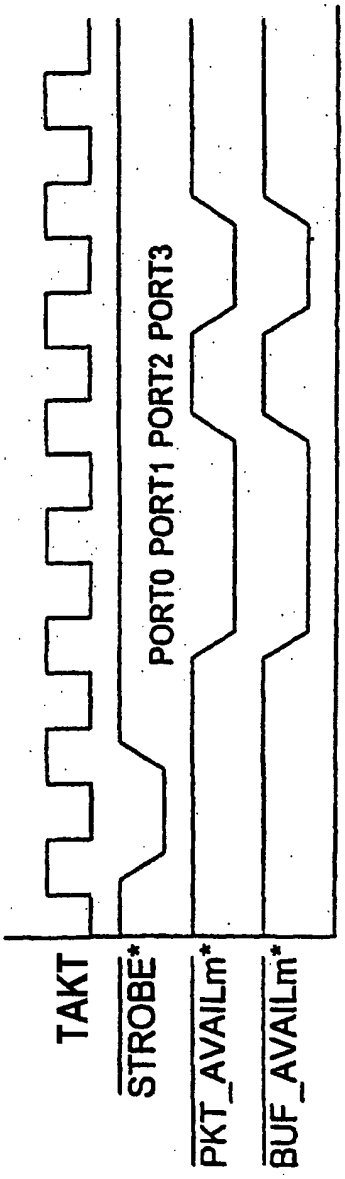

TAKT
STROBE*
PORT0 PORT1 PORT2 PORT3
PKT_AVAILm*
BUF_AVAILm*

FIG. 3F

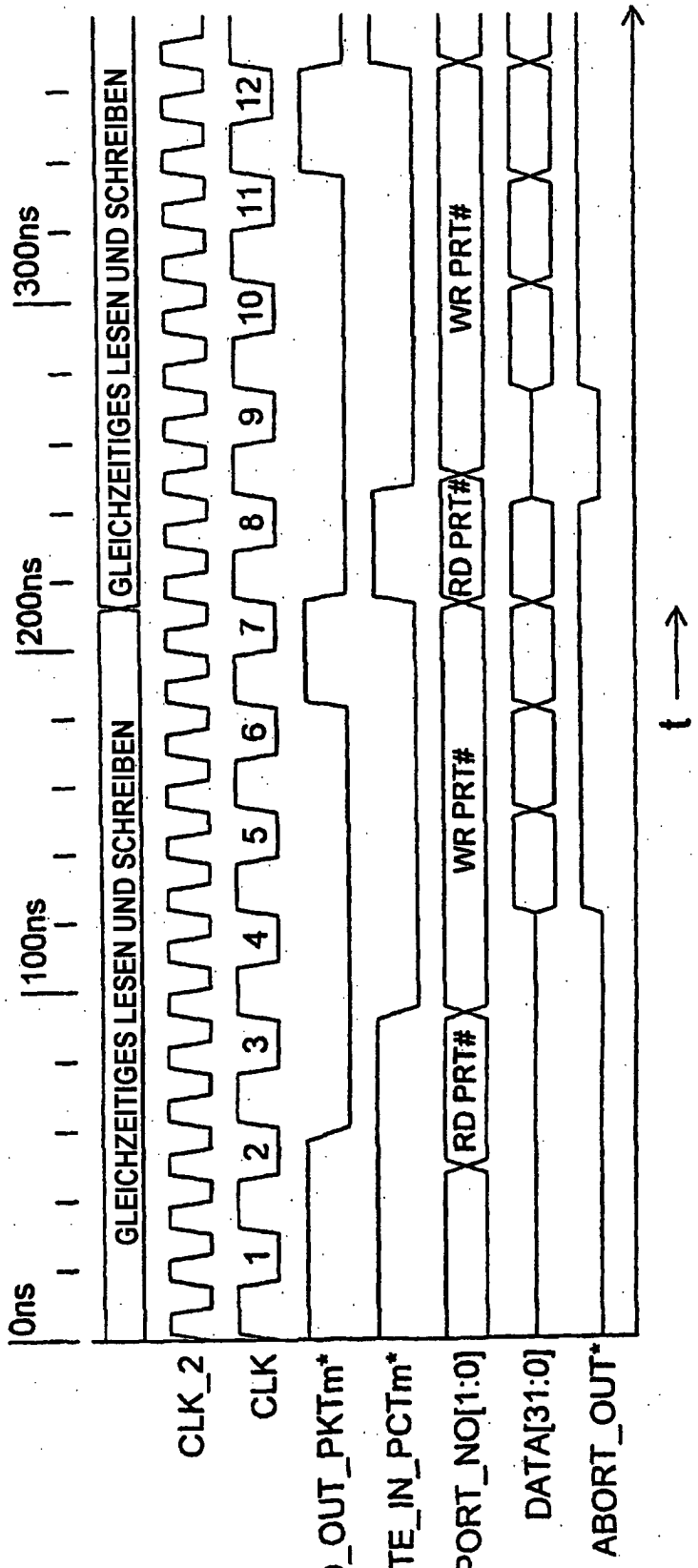

0ns
100ns
200ns
300ns
GLEICHZEITIGES LESEN UND SCHREIBEN
GLEICHZEITIGES LESEN UND SCHREIBEN
1
2
3
4
5
6
7
8
9
10
11
12
CLK_2
CLK
READ_OUT_PKTm*
WRITE_IN_PCTm*
PORT_NO[1:0]
DATA[31:0]
ABORT_OUT*
RD PRT#
WR PRT#
RD PRT#
WR PRT#
t

FIG. 3G

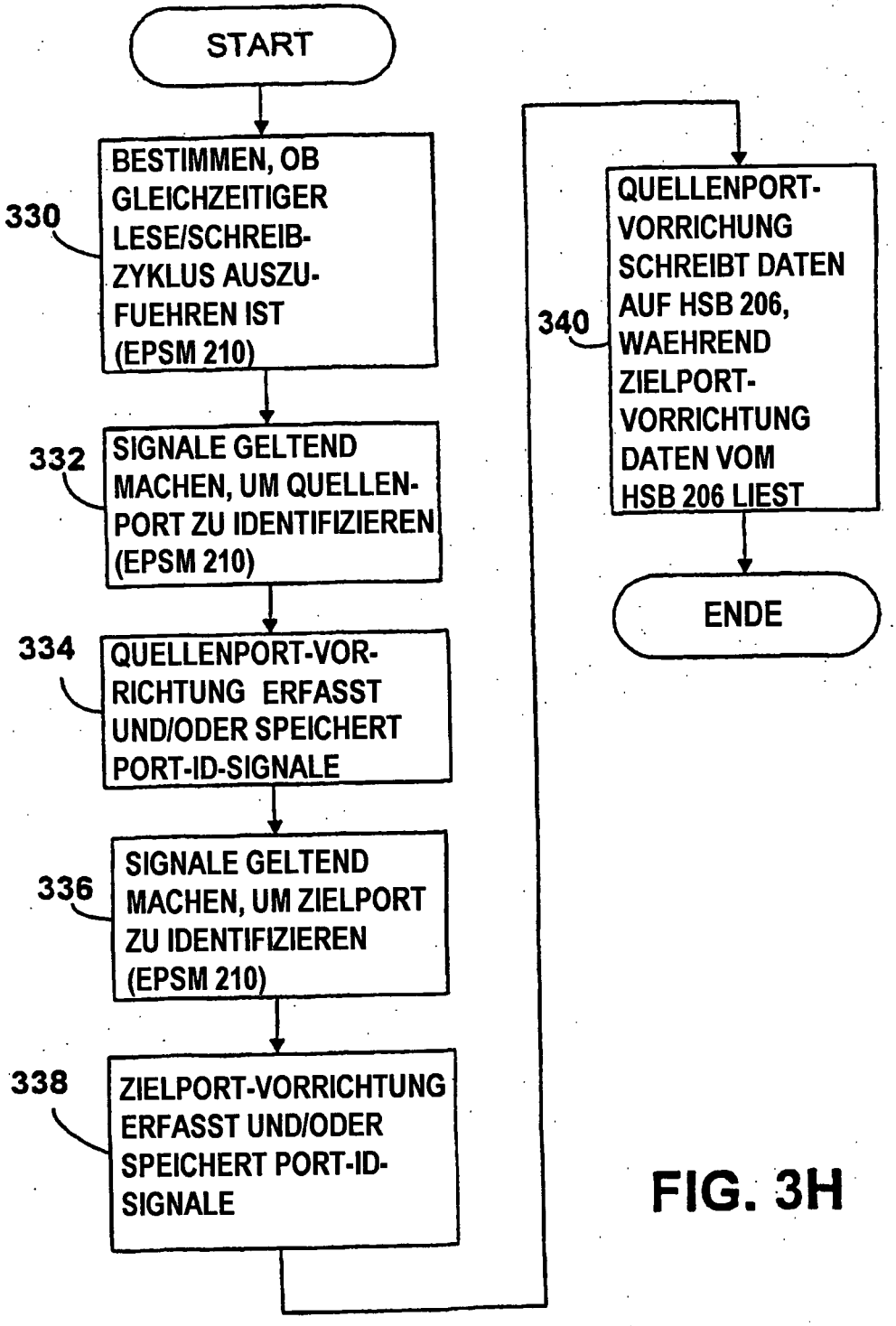

START
330
BESTIMMEN, OB GLEICHZEITIGER LESE/SCHREIB-ZYKLUS AUSZU-FUEHREN IST (EPSM 210)
332
SIGNALE GELTEND MACHEN, UM QUELLEN-PORT ZU IDENTIFIZIEREN (EPSM 210)
334
QUELLENPORT-VOR-RICHTUNG ERFASST UND/ODER SPEICHERT PORT-ID-SIGNALE
336
SIGNALE GELTEND MACHEN, UM ZIELPORT ZU IDENTIFIZIEREN (EPSM 210)
338
ZIELPORT-VORRICHTUNG ERFASST UND/ODER SPEICHERT PORT-ID-SIGNALE
340
QUELLENPORT-VORRICHUNG SCHREIBT DATEN AUF HSB 206, WAEHREND ZIELPORT-VORRICHTUNG DATEN VOM HSB 206 LIEST
ENDE

FIG. 3H

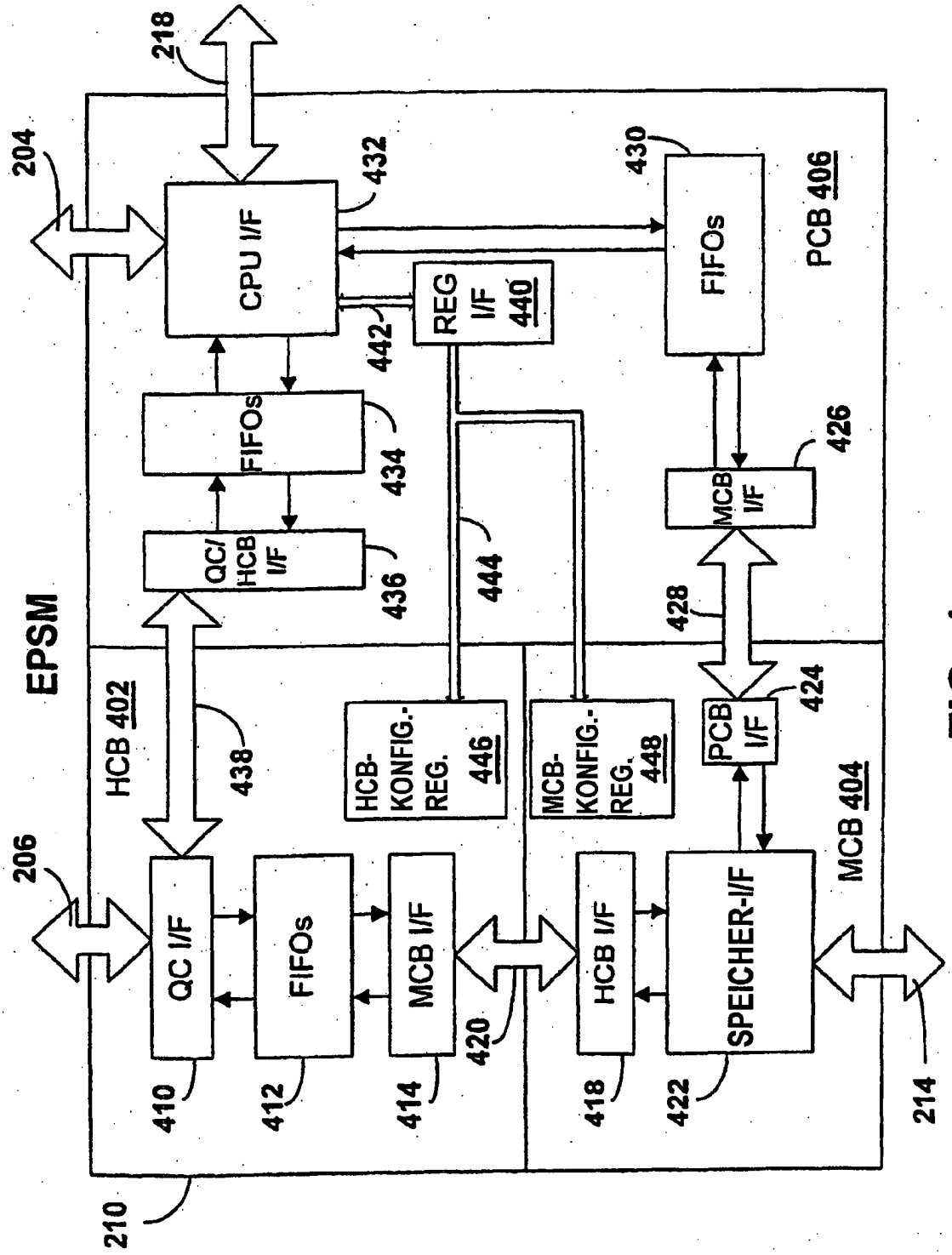

EPSM
218
204
CPU I/F
432
430
FIFOs
PCB 406
REG I/F 440
442
FIFOs
434
426
MCB I/F
QC/ HCB I/F
436
444
428
HCB 402
438
HCB-KONFIG.-REG. 446
MCB-KONFIG.-REG. 448
PCB I/F
424
206
QC I/F
FIFOs
MCB I/F
HCB I/F
SPEICHER-I/F
MCB 404
410
412
414
420
418
422
214
210

FIG. 4

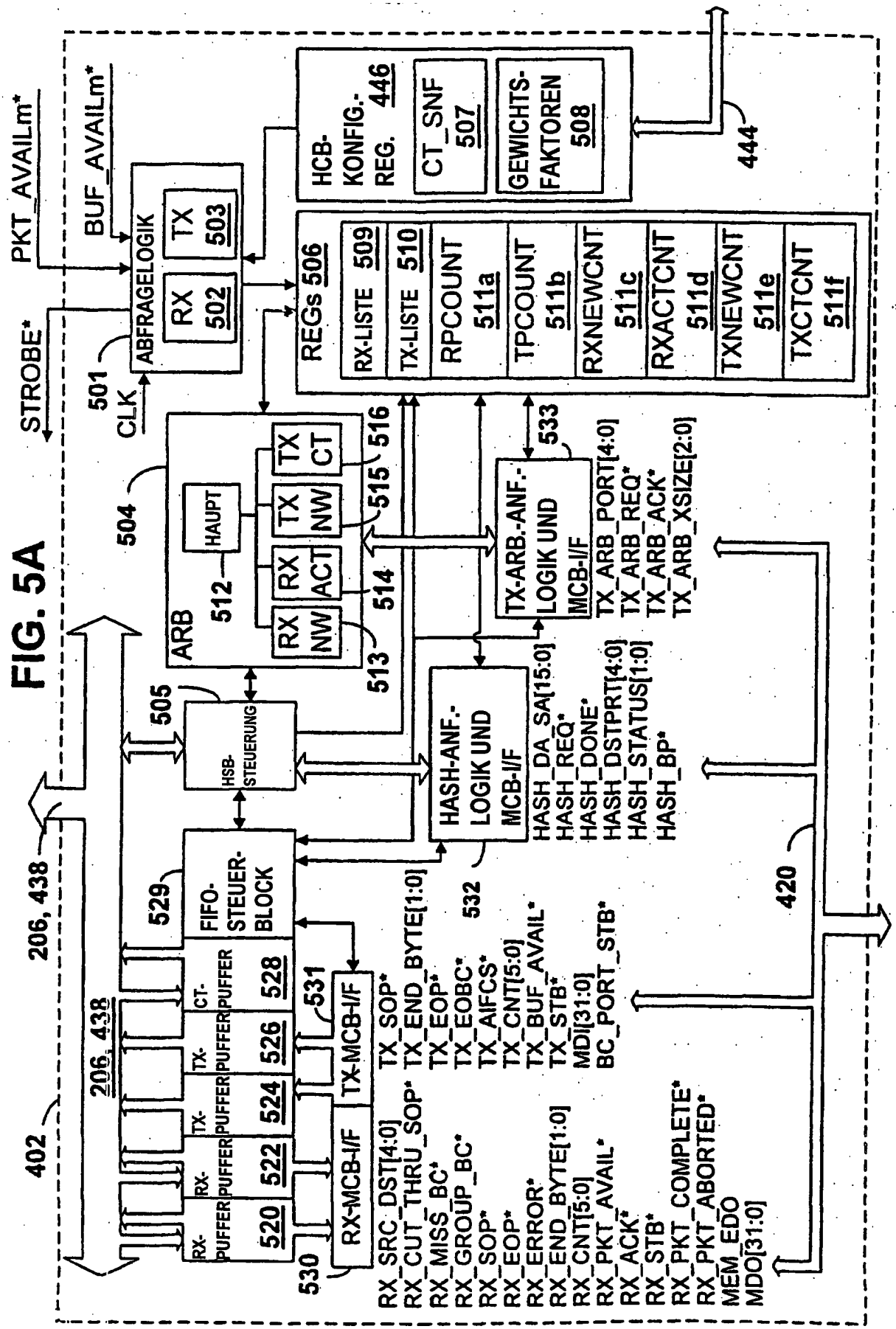

PKT_AVAILm*
BUF_AVAILm*
STROBE*
501
CLK
ABFRAGELOGIK
RX 502
TX 503
HCB-KONFIG.-REG. 446
CT_SNF 507
GEWICHTS-FAKTOREN 508
444
REGs 506
RX-LISTE 509
TX-LISTE 510
RPCOUNT 511a
TPCOUNT 511b
RXNEWCNT 511c
RXACTCNT 511d
TXNEWCNT 511e
TXCTCNT 511f
504
ARB
HAUPT 512
RX NW
RX ACT
TX NW
TX CT
513
514
515
516
533
TX-ARB.-ANF.-LOGIK UND MCB-I/F
TX_ARB_PORT[4:0]
TX_ARB_REQ*
TX_ARB_ACK*
TX_ARB_XSIZE[2:0]
505
HSB-STEUERUNG
HASH-ANF.-LOGIK UND MCB-I/F
HASH_DA_SA[15:0]
HASH_REQ*
HASH_DONE*
HASH_DSTPRT[4:0]
HASH_STATUS[1:0]
HASH_BP*
206, 438
529
FIFO-STEUER-BLOCK
420
402
206, 438
RX-PUFFER 520
RX-PUFFER 522
TX-PUFFER 524
TX-PUFFER 526
CT-PUFFER 528
531
RX-MCB-I/F
TX-MCB-I/F
530
RX_SRC_DST[4:0]
RX_CUT_THRU_SOP*
RX_MISS_BC*
RX_GROUP_BC*
RX_SOP*
RX_EOP*
RX_ERROR*
RX_END_BYTE[1:0]
RX_CNT[5:0]
RX_PKT_AVAIL*
RX_ACK*
RX_STB*
RX_PKT_COMPLETE*
RX_PKT_ABORTED*
MEM_EDO
MDO[31:0]
TX_SOP*
TX_END_BYTE[1:0]
TX_EOP*
TX_EOBC*
532
TX_AIFCS*
TX_CNT[5:0]
TX_BUF_AVAIL*
TX_STB*
MDI[31:0]
BC_PORT_STB*

FIG. 5A

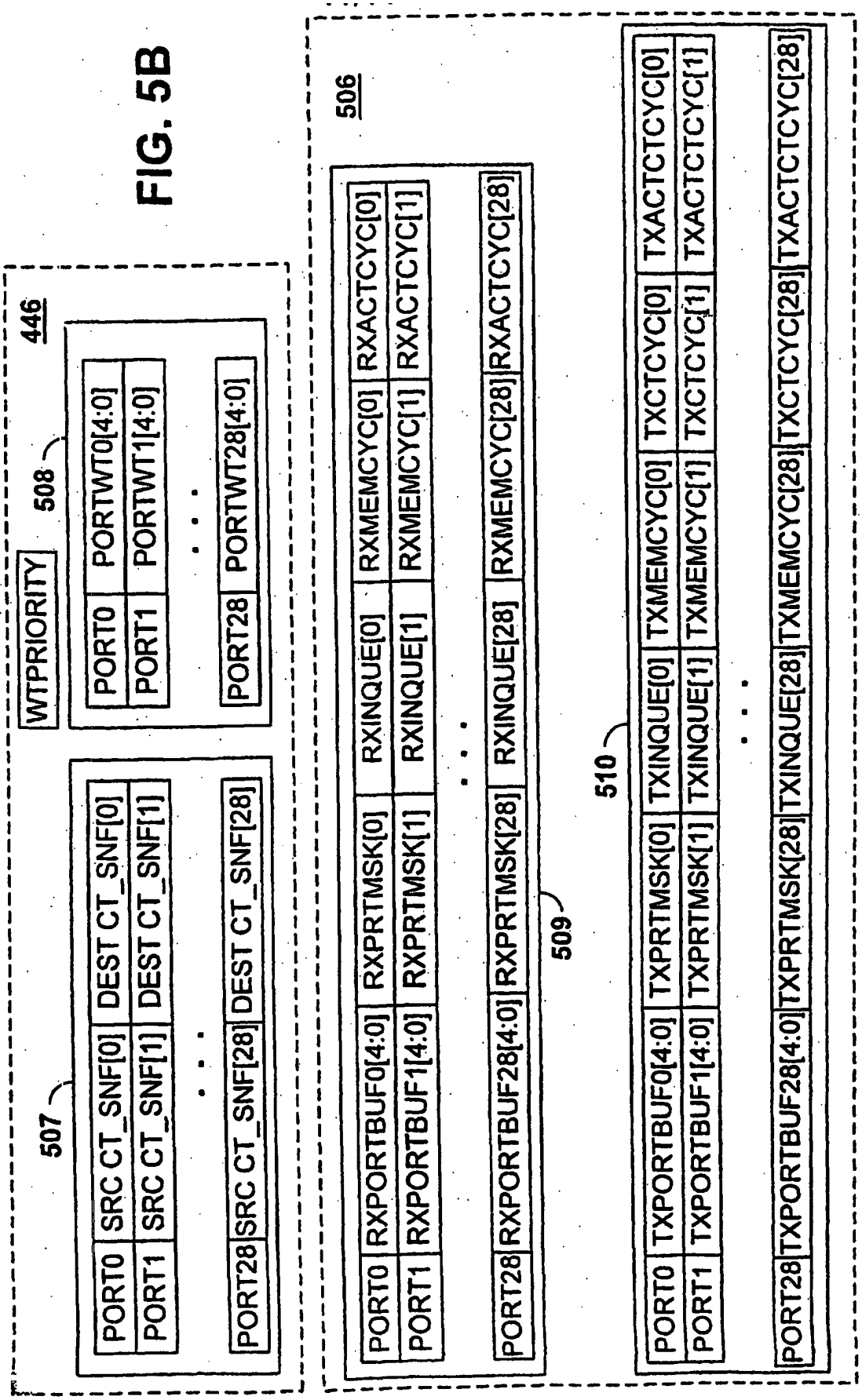

446
WTPRIORITY
508
PORT0
PORTWT0[4:0]
PORT1
PORTWT1[4:0]
PORT28
PORTWT28[4:0]
507
PORT0
SRC CT_SNF[0]
DEST CT_SNF[0]
PORT1
SRC CT_SNF[1]
DEST CT_SNF[1]
PORT28
SRC CT_SNF[28]
DEST CT_SNF[28]
506
PORT0
RXPORTBUF0[4:0]
RXPRTMSK[0]
RXINQUE[0]
RXMEMCYC[0]
RXACTCYC[0]
PORT1
RXPORTBUF1[4:0]
RXPRTMSK[1]
RXINQUE[1]
RXMEMCYC[1]
RXACTCYC[1]
PORT28
RXPORTBUF28[4:0]
RXPRTMSK[28]
RXINQUE[28]
RXMEMCYC[28]
RXACTCYC[28]
509
510
PORT0
TXPORTBUF0[4:0]
TXPRTMSK[0]
TXINQUE[0]
TXMEMCYC[0]
TXCTCYC[0]
TXACTCYC[0]
PORT1
TXPORTBUF1[4:0]
TXPRTMSK[1]
TXINQUE[1]
TXMEMCYC[1]
TXCTCYC[1]
TXACTCYC[1]
PORT28
TXPORTBUF28[4:0]
TXPRTMSK[28]
TXINQUE[28]
TXMEMCYC[28]
TXCTCYC[28]
TXACTCYC[28]

FIG. 5B

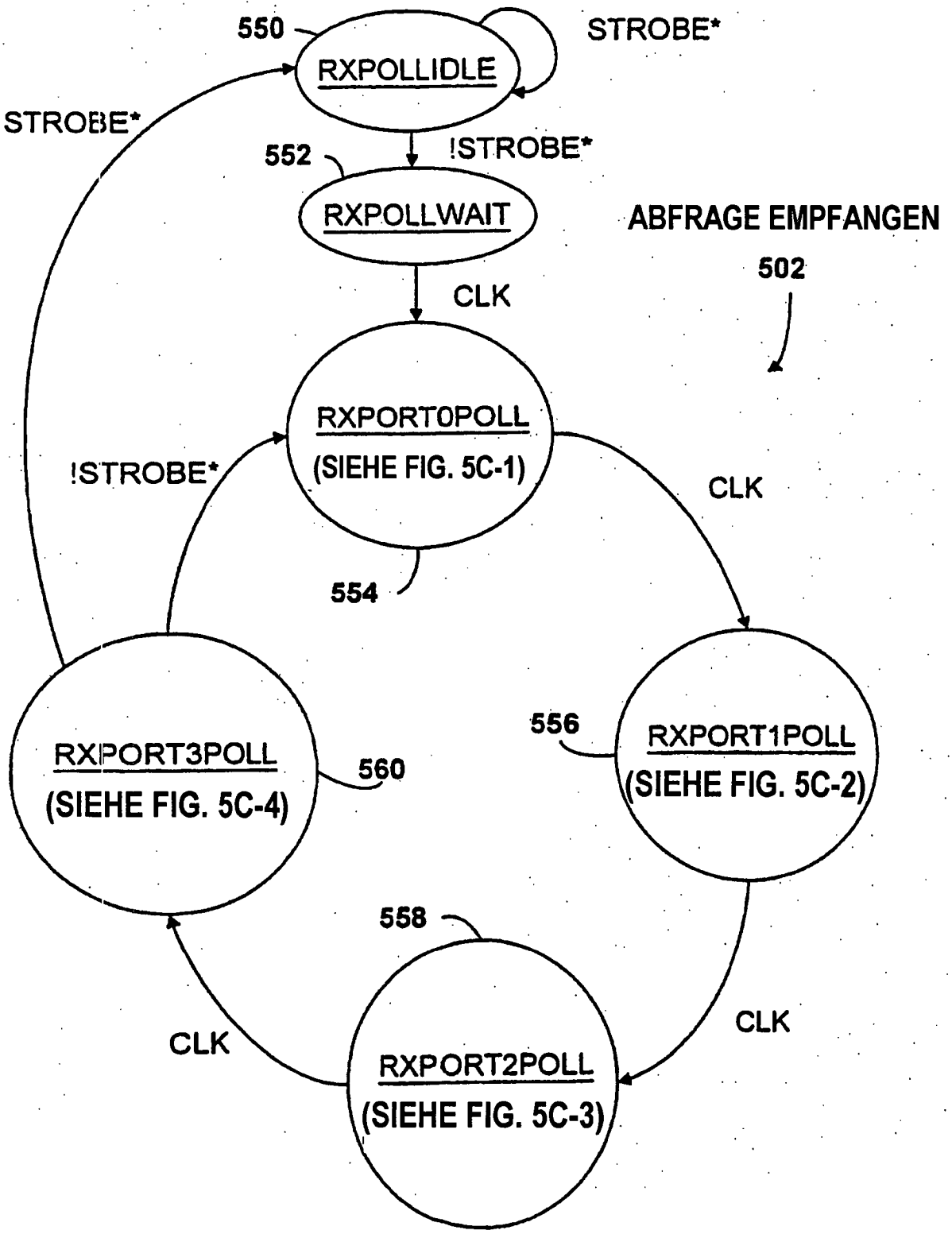

550
RXPOLLIDLE
STROBE*
STROBE*
552
!STROBE*
RXPOLLWAIT
ABFRAGE EMPFANGEN
502
CLK
RXPORT0POLL
(SIEHE FIG. 5C-1)
!STROBE*
CLK
554
RXPORT3POLL
(SIEHE FIG. 5C-4)
560
556
RXPORT1POLL
(SIEHE FIG. 5C-2)
558
CLK
CLK
RXPORT2POLL
(SIEHE FIG. 5C-3)

FIG. 5C

554

```
RXPORT0POLL
IF ((RXNEWCNT+1 != RPCOUNT)
|| (RXACTCNT+1 != RPCOUNT)), THEN
{
1) IF (!PKT_AVAIL*[0] && !RXPRTMSK[0]), THEN
(IF (WTPRIORITY), THEN RXPORTBUF0 = PORTWT[0],
ELSE  RXPORTBUF0 = RPCOUNT;
RXPRTMSK[0]  = 1; RXINCCNTBY[0] = 1);

2) IF (!PKT_AVAIL[1]* && !RXPRTMSK[4]), THEN
(IF (WTPRIORITY), then RXPORTBUF4 = PORTWT[4],
ELSE RXPORTBUF4 = RPCOUNT + RXINCCNTBY[0];
RXPRTMSK[4] = 1; RXINCCNTBY[1] = 1);
. . .
8) IF (!PKT_AVAIL[7]* && !RXPRTMSK[28]), THEN
(IF (WTPRIORITY), THEN RXPORTBUF28 = PORTWT[28],
ELSE RXPORTBUF28 = RPCOUNT +
BITSUM(RXINCCNTBY[6:0]);
RXPRTMSK[28] = 1; RXINCCNTBY[7] = 1);

9) RPCOUNT = RPCOUNT +
BITSUM(RXINCCNTBY[7:0])
}
```

FIG. 5C-1

556

```
RXPORT1POLL
IF ((RXNEWCNT+1 != RPCOUNT)
|| (RXACTCNT+1 != RPCOUNT)), THEN
{
1) IF (!PKT_AVAIL[0]* && !RXPRTMSK[1]), THEN
(IF (WTPRIORITY),
THEN RXPORTBUF1 = PORTWT[1],
ELSE RXPORTBUF1 = RPCOUNT;
RXPRTMSK[1] = 1; RXINCCNTBY[0] = 1);
. . .
7) IF (!PKT_AVAIL[6]* && !RXPRTMSK[25]), THEN
(IF (WTPRIORITY),
THEN RXPORTBUF25 = PORTWT[25],
ELSE RXPORTBUF25 =
RPCOUNT + BITSUM(RXINCCNTBY[5:0]);
RXPRTMSK[25] = 1; RXINCCNTBY[6] = 1);

8) (SAME EQUATION 8 AS IN STATE 554);
9) RPCOUNT = RPCOUNT +
BITSUM(RXINCCNTBY[6:0])
}
```

FIG. 5C-2

558

```
RXPORT2POLL
IF ((RXNEWCNT+1 != RPCOUNT)
|| (RXACTCNT+1 != RPCOUNT)), THEN
{
1) IF (!PKT_AVAIL[0]* && !RXPRTMSK[2]), THEN
(IF (WTPRIORITY),
THEN RXPORTBUF2 = PORTWT[2],
ELSE RXPORTBUF2 = RPCOUNT;
RXPRTMSK[2] = 1; RXINCCNTBY[0] = 1);
. . .
7) IF (!PKT_AVAIL[6]* && !RXPRTMSK[26]), THEN
(IF (WTPRIORITY),
THEN RXPORTBUF26 = PORTWT[26],
ELSE RXPORTBUF26 = RPCOUNT +
BITSUM(RXINCCNTBY[5:0]);
RXPRTMSK[26] = 1; RXINCCNTBY[6] = 1);

8) (SAME EQUATION 8 AS IN STATE 554);
9) RPCOUNT = RPCOUNT +
BITSUM(RXINCCNTBY[6:0])
}
```

FIG. 5C-3

560

```
RXPORT3POLL
IF ((RXNEWCNT+1 != RPCOUNT)
|| (RXACTCNT+1 != RPCOUNT)), THEN
{
1) IF (!PKT_AVAIL[0]* && !RXPRTMSK[3]), THEN
(IF (WTPRIORITY),
THEN RXPORTBUF3 = PORTWT[3],
ELSE RXPORTBUF3 = RPCOUNT;
RXPRTMSK[3] = 1; RXINCCNTBY[0] = 1);
. . .
7) IF (!PKT_AVAIL[6]* && !RXPRTMSK[27]), THEN
(IF (WTPRIORITY),
THEN RXPORTBUF27 = PORTWT[27],
ELSE RXPORTBUF27 = RPCOUNT +
BITSUM(RXINCCNTBY[5:0]);
RXPRTMSK[27] = 1; RXINCCNTBY[6] = 1);

8) (SAME EQUATION 8 AS IN STATE 554);
9) RPCOUNT = RPCOUNT +
BITSUM(RXINCCNTBY[6:0])
}
```

FIG. 5C-4

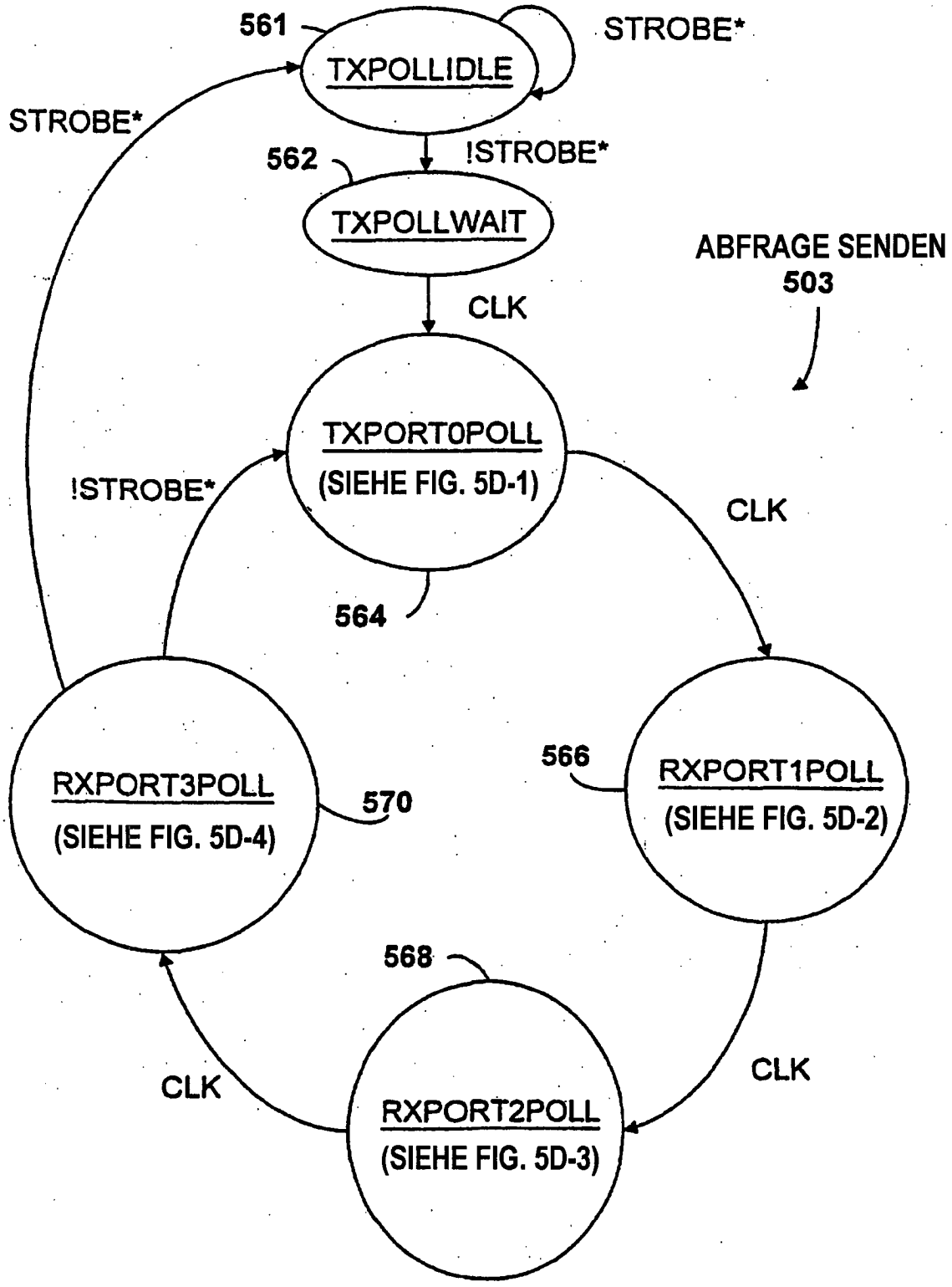

561
TXPOLLIDLE
STROBE*
STROBE*
562
!STROBE*
TXPOLLWAIT
ABFRAGE SENDEN
503
CLK
TXPORT0POLL
(SIEHE FIG. 5D-1)
!STROBE*
CLK
564
RXPORT3POLL
(SIEHE FIG. 5D-4)
570
566
RXPORT1POLL
(SIEHE FIG. 5D-2)
568
CLK
CLK
RXPORT2POLL
(SIEHE FIG. 5D-3)

FIG. 5D

564

```
TXPORT0POLL
IF ((TXNEWCNT+1 != TPCOUNT)
|| (TXCTCNT+1 != TPCOUNT)), THEN
{
1) IF (!BUF_AVAIL*[0] && (!TXPRTMSK[0] &&
(TXMEMCYC[0] || TXCTACTCYC[0] || TXCTCYC[0]))), THEN
(IF (WTPRIORITY), THEN TXPORTBUF0 = PORTWT[0],
ELSE  TXPORTBUF0 = TPCOUNT;
TXPRTMSK[0] = 1; TXINCCNTBY[0] = 1);
2) IF (!BUF_AVAIL[1]* && (!TXPRTMSK[4] &&
(TXMEMCYC[4] || TXCTACTCYC[4] || TXCTCYC[4]))), THEN
(IF (WTPRIORITY), THEN TXPORTBUF4 = PORTWT[4],
ELSE TXPORTBUF4 = TPCOUNT + TXINCCNTBY[0];
TXPRTMSK[4] = 1; TXINCCNTBY[1] = 1);
. . .
8) IF (!BUF_AVAIL[7]* && (!TXPRTMSK[28] &&
(TXMEMCYC[28] || TXCTACTCYC[28] || TXCTCYC[28]))), THEN
(IF (WTPRIORITY), THEN TXPORTBUF28 = PORTWT[28],
ELSE TXPORTBUF28 = TPCOUNT +
BITSUM(TXINCCNTBY[6:0]);
TXPRTMSK[28] = 1; TXINCCNTBY[7] = 1);
9) TPCOUNT = TPCOUNT +
BITSUM(TXINCCNTBY[7:0])
}
```

FIG. 5D-1

566

```
TXPORT1POLL
IF ((TXNEWCNT+1 != TPCOUNT) ||
(TXCTCNT+1 != TPCOUNT)), THEN
{
1) IF (!BUF_AVAIL[0]* && (!TXPRTMSK[1] &&
(TXMEMCYC[1] || TXCTACTCYC[1] || TXCTCYC[1]))),
THEN (IF (WTPRIORITY),
THEN TXPORTBUF1 = PORTWT[1], ELSE TXPORTBUF1 =
TPCOUNT; TXPRTMSK[1] = 1; TXINCCNTBY[0] = 1);
. . .
7) IF (!BUF_AVAIL[6]* && (!TXPRTMSK[25] &&
(TXMEMCYC[25] || TXCTACTCYC[25] || TXCTCYC[25]))), THEN
(IF (WTPRIORITY), THEN TXPORTBUF25 = PORTWT[25],
ELSE TXPORTBUF25 =
TPCOUNT + BITSUM(TXINCCNTBY[5:0]);
TXPRTMSK[25] = 1; TXINCCNTBY[6] = 1);

8) (SAME EQUATION 8 AS IN STATE 564);
9) TPCOUNT = TPCOUNT +
BITSUM(TXINCCNTBY[6:0])
}
```

FIG. 5D-2

568

```
TXPORT2POLL
IF ((TXNEWCNT+1 != TPCOUNT)
|| (TXCTCNT+1 != TPCOUNT)), THEN
{
1) IF (!BUF_AVAIL[0]* && (!TXPRTMSK[2] &&
(TXMEMCYC[2] || TXCTACTCYC[2] || TXCTCYC[2]))),
THEN(IF (WTPRIORITY),
THEN TXPORTBUF2 = PORTWT[2],
ELSE TXPORTBUF2 = TPCOUNT;
TXPRTMSK[2] = 1; TXINCCNTBY[0] = 1);
. . .
7) IF (!BUF_AVAIL[6]* && (!TXPRTMSK[26] &&
(TXMEMCYC[26] || TXCTACTCYC[26] || TXCTCYC[26]))), THEN
(IF (WTPRIORITY),
THEN TXPORTBUF26 = PORTWT[26],
ELSE TXPORTBUF26 = TPCOUNT +
BITSUM(TXINCCNTBY[5:0]);
TXPRTMSK[26] = 1; TXINCCNTBY[6] = 1);

8) (SAME EQUATION 8 AS IN STATE 564);
9) TPCOUNT = TPCOUNT +
BITSUM(TXINCCNTBY[6:0])
}
```

FIG. 5D-3

570

```
TXPORT3POLL
IF ((TXNEWCNT+1 != TPCOUNT) ||
(TXCTCNT+1 != TPCOUNT)), THEN
{
1) IF (!BUF_AVAIL[0]* && (!TXPRTMSK[3] &&
(TXMEMCYC[3] || TXCTACTCYC[3] || TXCTCYC[3]))), THEN
(IF (WTPRIORITY),
THEN TXPORTBUF3 = PORTWT[3],
ELSE TXPORTBUF3 = TPCOUNT;
TXPRTMSK[3]  = 1; TXINCCNTBY[0] = 1);
. . .
7) IF (!BUF_AVAIL[6]* && (!TXPRTMSK[27] &&
(TXMEMCYC[27] || TXCTACTCYC[27] || TXCTCYC[27]))),
THEN (IF (WTPRIORITY),
THEN TXPORTBUF27 = PORTWT[27],
ELSE TXPORTBUF27 = TPCOUNT +
BITSUM(TXINCCNTBY[5:0]);
TXPRTMSK[27] = 1; TXINCCNTBY[6] = 1);

8) (SAME EQUATION 8 AS IN STATE 564);
9) TPCOUNT = TPCOUNT +
BITSUM(TXINCCNTBY[6:0])
}
```

FIG. 5D-4

FIG. 6

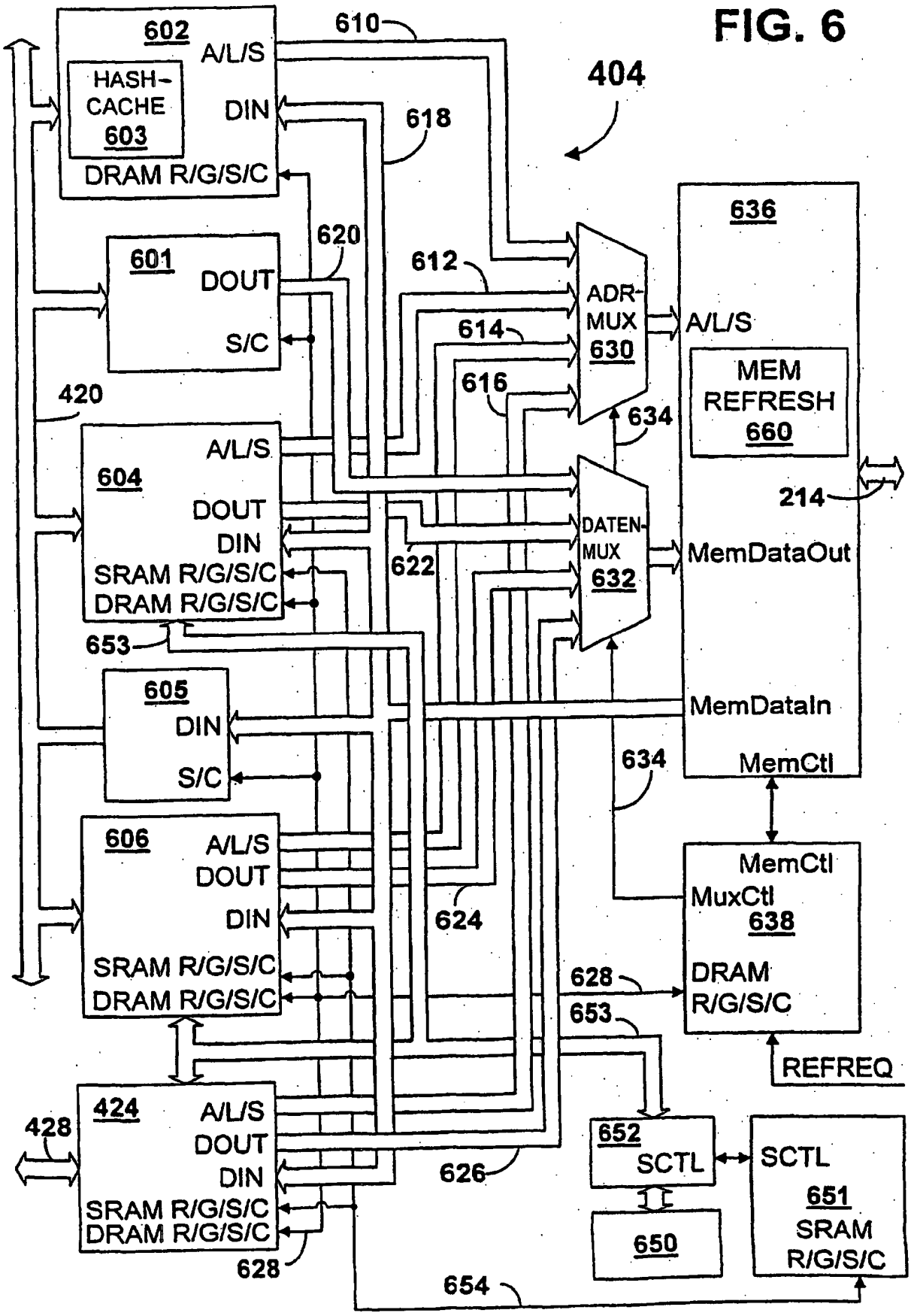

602
A/L/S
HASH-CACHE 603
DIN
DRAM R/G/S/C
610
618
404
620
601
DOUT
S/C
612
614
616
ADR-MUX 630
636
A/L/S
MEM REFRESH 660
634
420
604
A/L/S
DOUT
DIN
SRAM R/G/S/C
DRAM R/G/S/C
622
DATEN-MUX 632
214
MemDataOut
653
605
DIN
S/C
MemDataIn
MemCtl
634
606
A/L/S
DOUT
DIN
SRAM R/G/S/C
DRAM R/G/S/C
624
MemCtl
MuxCtl 638
628
DRAM R/G/S/C
653
REFREQ
424
A/L/S
DOUT
DIN
SRAM R/G/S/C
DRAM R/G/S/C
428
626
652 SCTL
SCTL
651 SRAM R/G/S/C
650
628
654

FIG. 7A

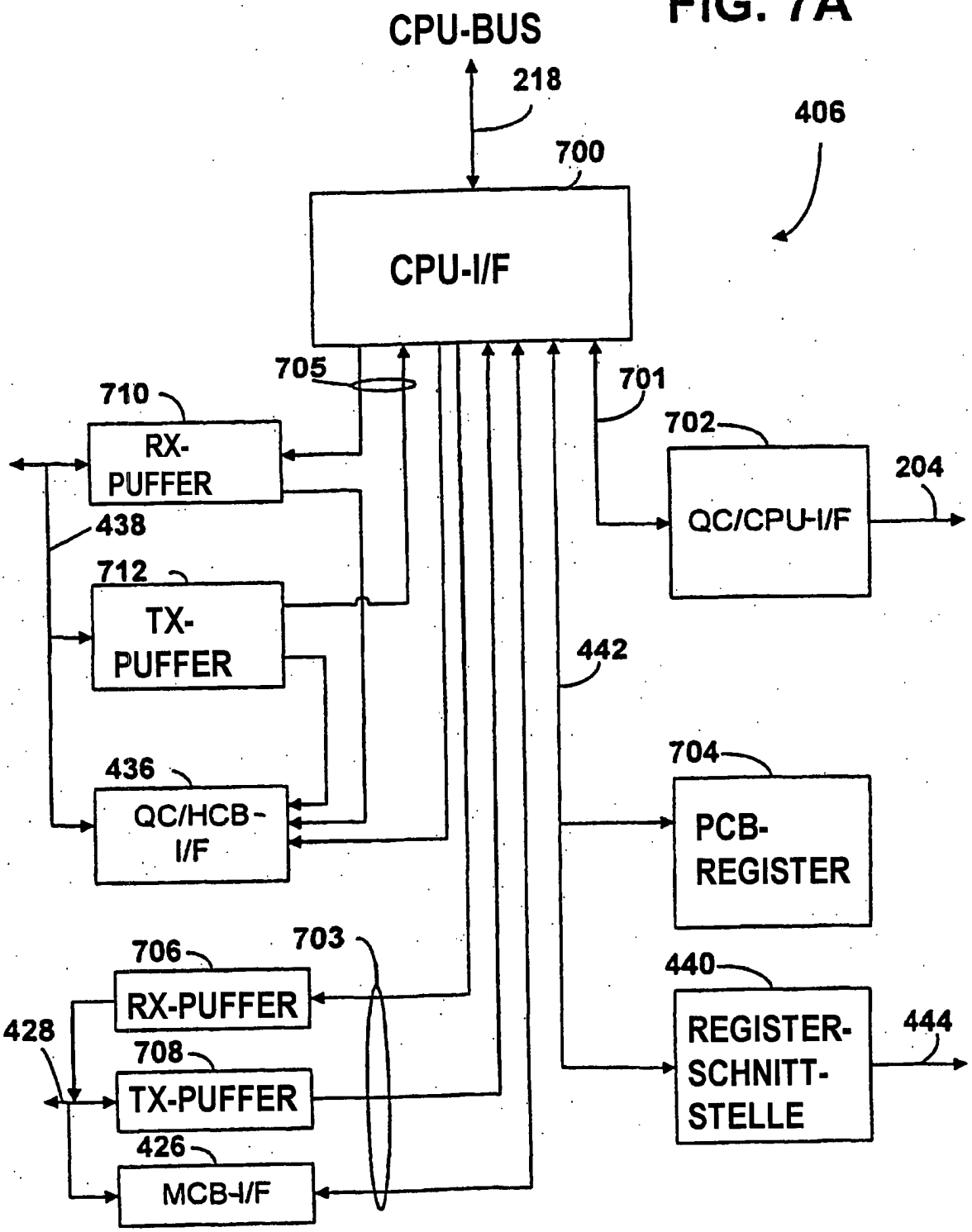

CPU-BUS
218
406
700
CPU-I/F
705
701
710
RX-
PUFFER
702
204
QC/CPU-I/F
438
712
TX-
PUFFER
442
436
QC/HCB-
I/F
704
PCB-
REGISTER
703
706
RX-PUFFER
440
428
708
TX-PUFFER
REGISTER-
SCHNITT-
STELLE
444
426
MCB-I/F

FIG. 7B

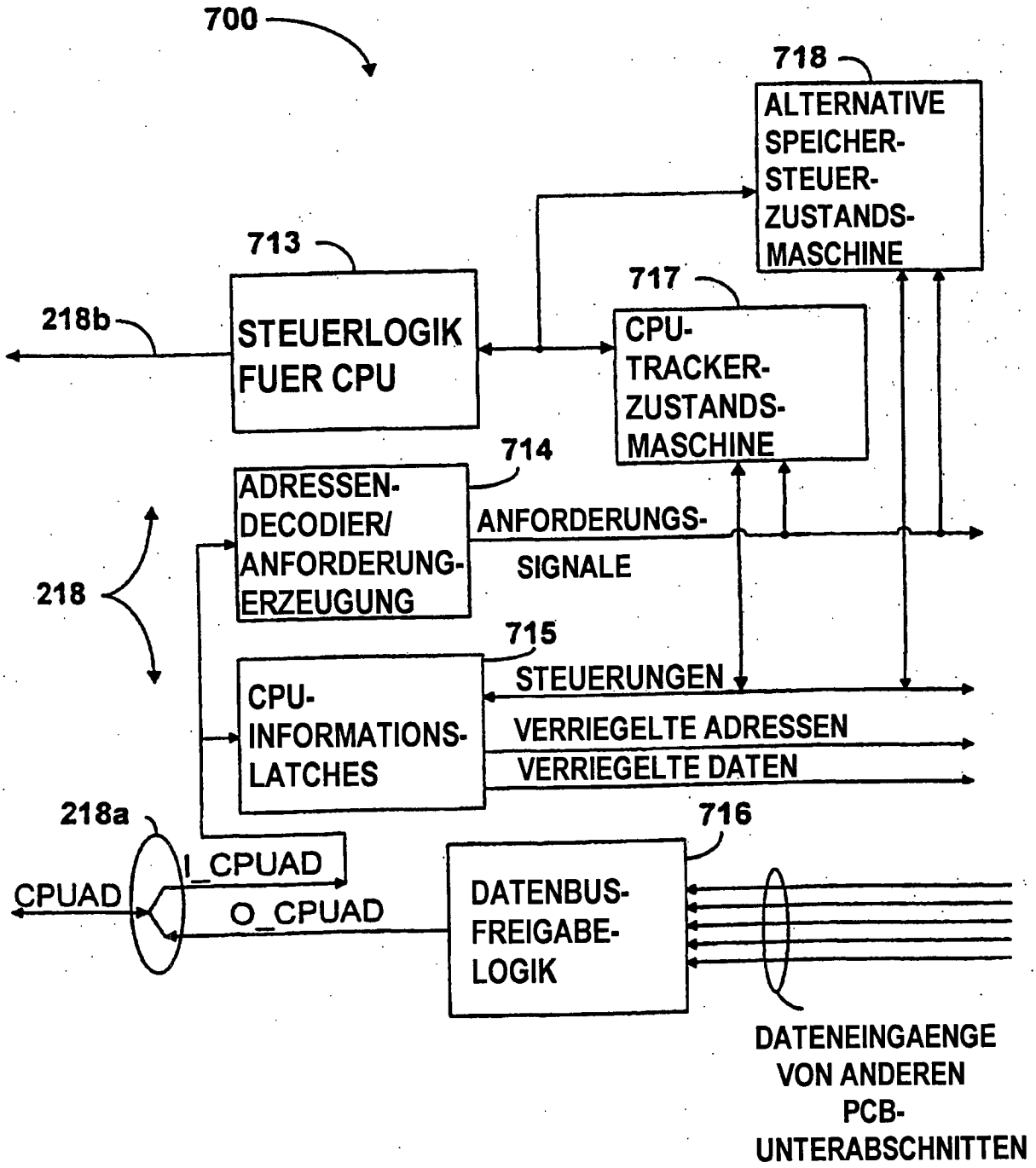

700
718
ALTERNATIVE SPEICHER-STEUER-ZUSTANDS-MASCHINE
713
717
218b
STEUERLOGIK FUER CPU
CPU-TRACKER-ZUSTANDS-MASCHINE
714
ADRESSEN-DECODIER/ ANFORDERUNG-ERZEUGUNG
ANFORDERUNGS-SIGNALE
218
715
STEUERUNGEN
CPU-INFORMATIONS-LATCHES
VERRIEGELTE ADRESSEN
VERRIEGELTE DATEN
218a
716
I_CPUAD
CPUAD
O_CPUAD
DATENBUS-FREIGABE-LOGIK
DATENEINGAENGE VON ANDEREN PCB-UNTERABSCHNITTEN

FIG. 7C

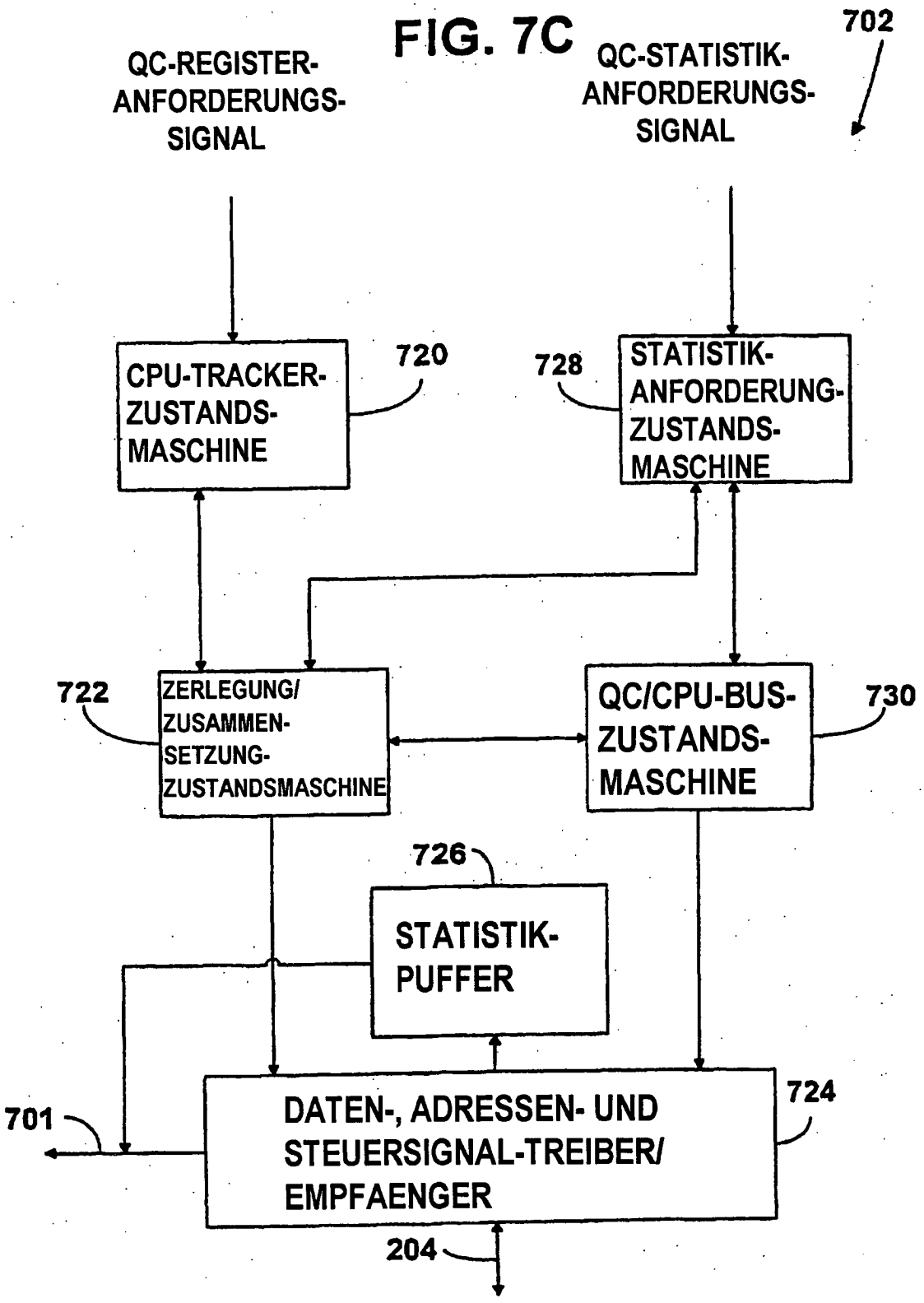

QC-REGISTER-ANFORDERUNGS-SIGNAL
QC-STATISTIK-ANFORDERUNGS-SIGNAL
702
CPU-TRACKER-ZUSTANDS-MASCHINE
720
728
STATISTIK-ANFORDERUNG-ZUSTANDS-MASCHINE
722
ZERLEGUNG/ZUSAMMEN-SETZUNG-ZUSTANDSMASCHINE
QC/CPU-BUS-ZUSTANDS-MASCHINE
730
726
STATISTIK-PUFFER
701
DATEN-, ADRESSEN- UND STEUERSIGNAL-TREIBER/EMPFAENGER
724
204

FIG. 7D

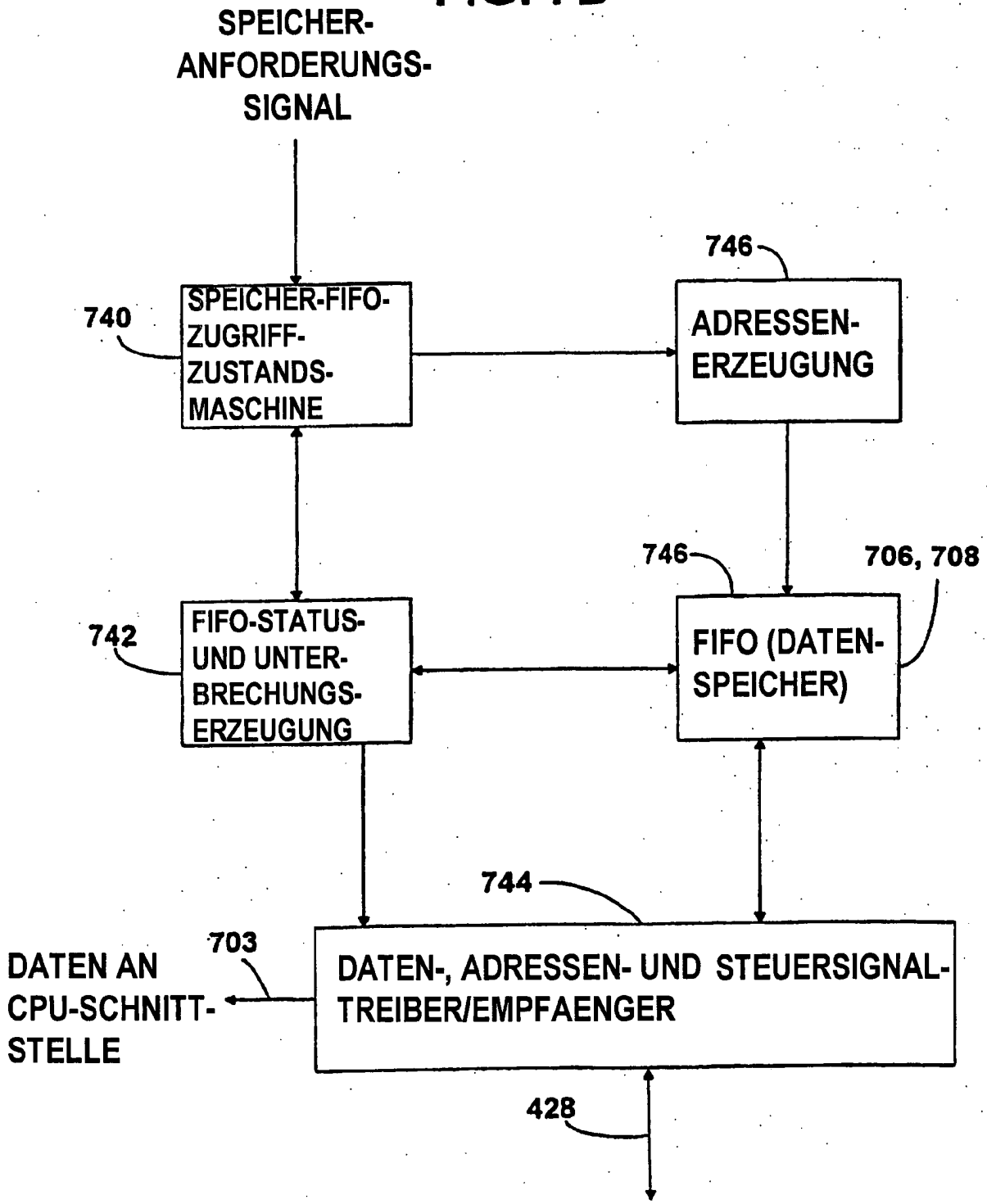
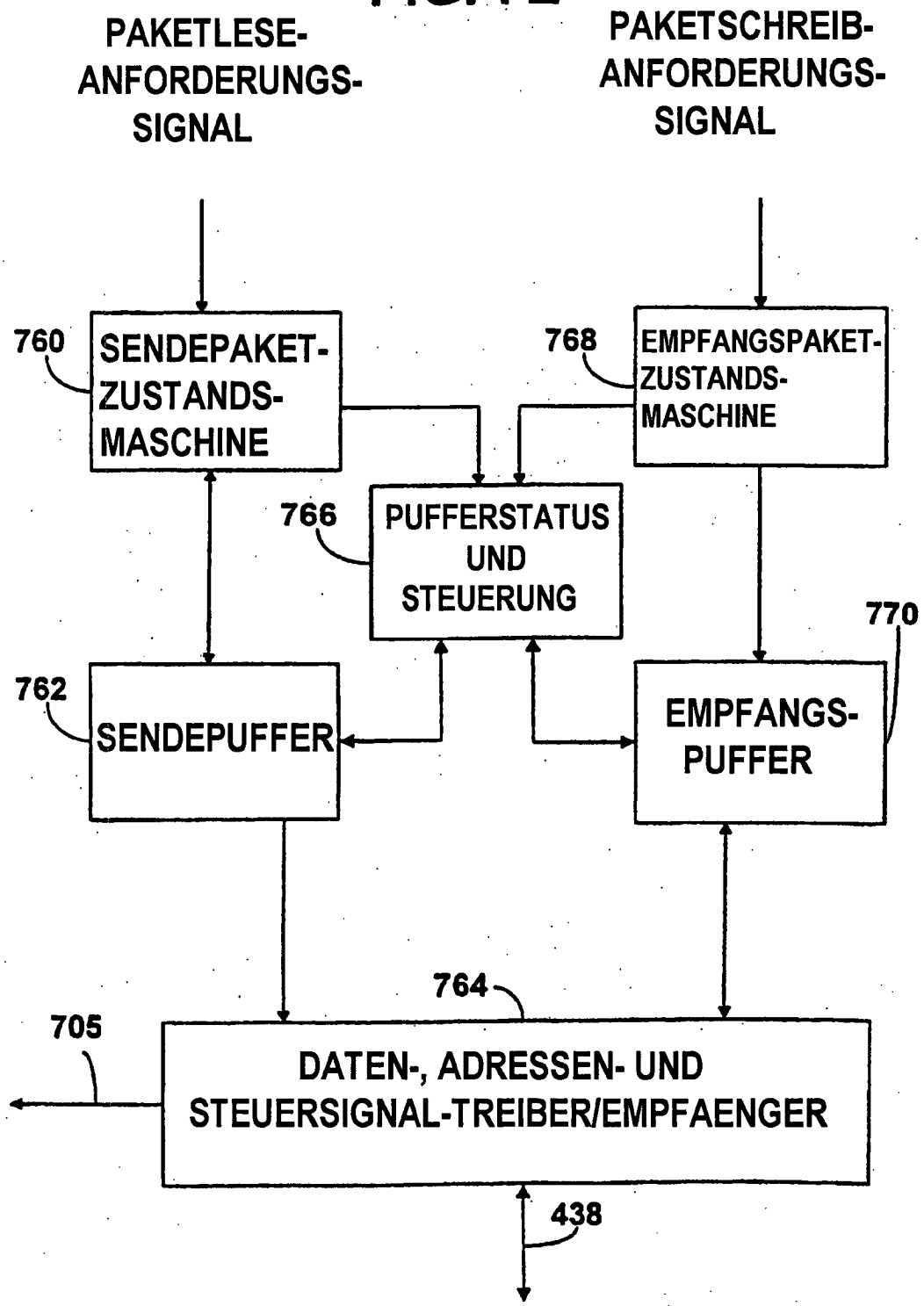

SPEICHER-ANFORDERUNGS-SIGNAL
746
740
SPEICHER-FIFO-ZUGRIFF-ZUSTANDS-MASCHINE
ADRESSEN-ERZEUGUNG
746
706, 708
742
FIFO-STATUS- UND UNTER-BRECHUNGS-ERZEUGUNG
FIFO (DATEN-SPEICHER)
744
703
DATEN AN CPU-SCHNITT-STELLE
DATEN-, ADRESSEN- UND STEUERSIGNAL-TREIBER/EMPFAENGER
428

FIG. 7E


PAKETLESE-ANFORDERUNGS-SIGNAL
PAKETSCHREIB-ANFORDERUNGS-SIGNAL
760
SENDEPAKET-ZUSTANDS-MASCHINE
768
EMPFANGSPAKET-ZUSTANDS-MASCHINE
766
PUFFERSTATUS UND STEUERUNG
770
762
SENDEPUFFER
EMPFANGS-PUFFER
764
705
DATEN-, ADRESSEN- UND STEUERSIGNAL-TREIBER/EMPFAENGER
438

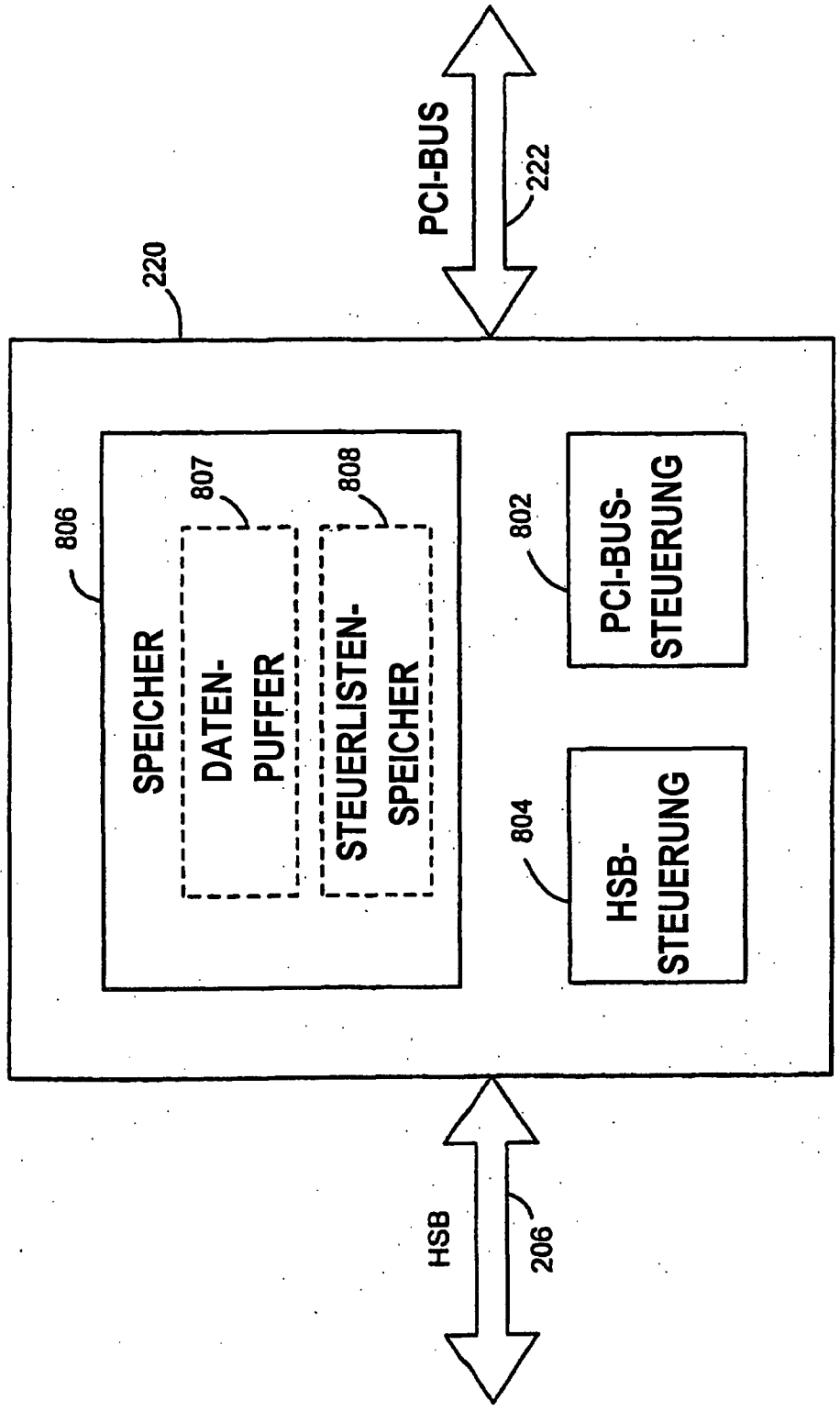

PCI-BUS
222
220
SPEICHER
806
DATEN-
PUFFER
807
STEUERLISTEN-
SPEICHER
808
PCI-BUS-
STEUERUNG
802
HSB-
STEUERUNG
804
HSB
206

FIG. 8A

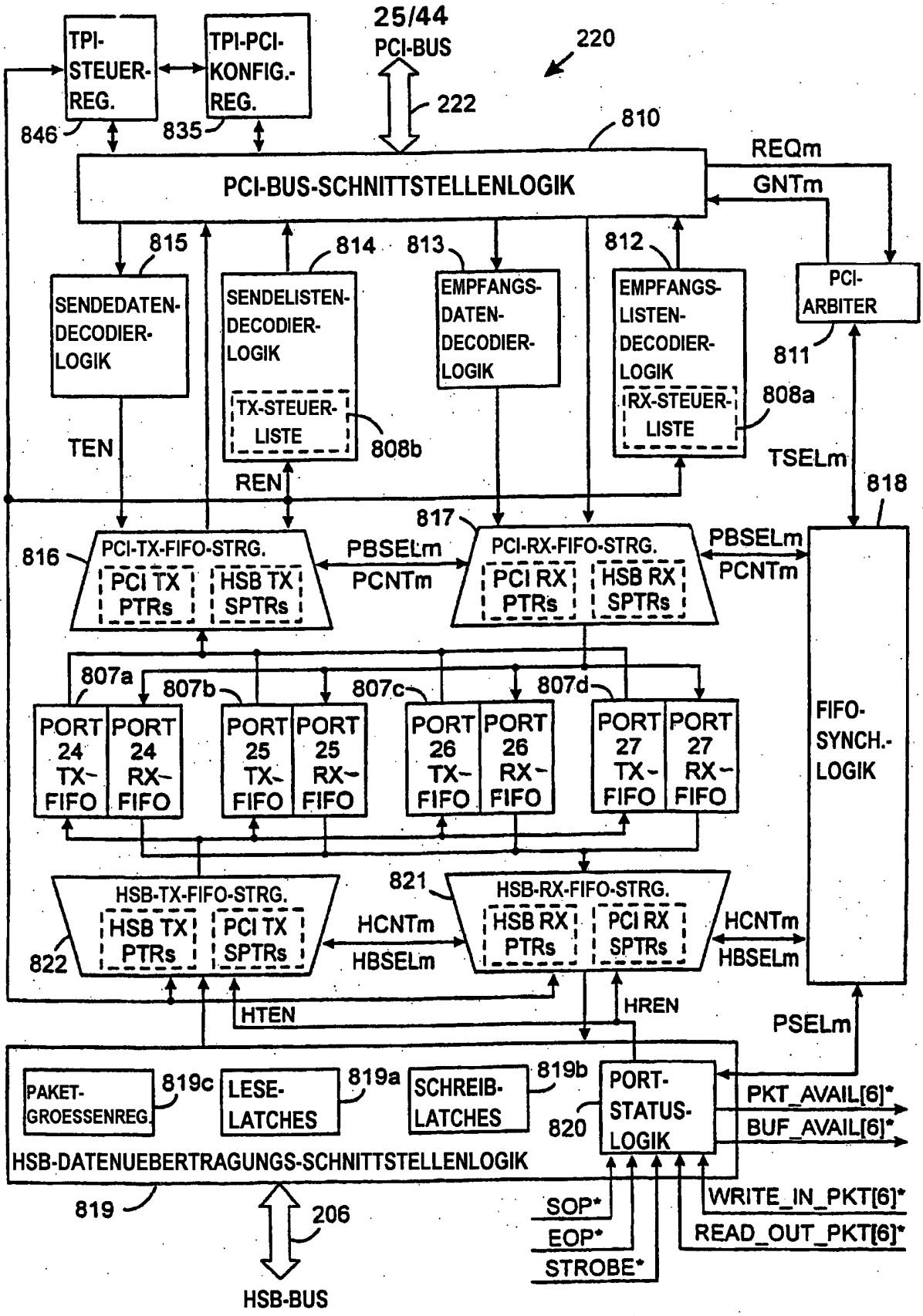

25/44
TPI-STEUER-REG.
TPI-PCI-KONFIG.-REG.
PCI-BUS
220
222
846
835
810
PCI-BUS-SCHNITTSTELLENLOGIK
REQm
GNTm
815
814
813
812
SENDEDATEN-DECODIER-LOGIK
SENDELISTEN-DECODIER-LOGIK
TX-STEUER-LISTE
808b
EMPFANGS-DATEN-DECODIER-LOGIK
EMPFANGS-LISTEN-DECODIER-LOGIK
RX-STEUER-LISTE
808a
PCI-ARBITER
811
TEN
REN
TSELm
818
817
816
PCI-TX-FIFO-STRG.
PCI TX PTRs
HSB TX SPTRs
PBSELm
PCNTm
PCI-RX-FIFO-STRG.
PCI RX PTRs
HSB RX SPTRs
807a
807b
807c
807d
PORT 24 TX-FIFO
PORT 24 RX-FIFO
PORT 25 TX-FIFO
PORT 25 RX-FIFO
PORT 26 TX-FIFO
PORT 26 RX-FIFO
PORT 27 TX-FIFO
PORT 27 RX-FIFO
FIFO-SYNCH.-LOGIK
821
HSB-TX-FIFO-STRG.
HSB TX PTRs
PCI TX SPTRs
822
HCNTm
HBSELm
HSB-RX-FIFO-STRG.
HSB RX PTRs
PCI RX SPTRs
HTEN
HREN
PSELm
PAKET-GROESSENREG.
819c
LESE-LATCHES
819a
SCHREIB-LATCHES
819b
PORT-STATUS-LOGIK
820
PKT_AVAIL[6]*
BUF_AVAIL[6]*
HSB-DATENUEBERTRAGUNGS-SCHNITTSTELLENLOGIK
819
206
HSB-BUS
SOP*
EOP*
STROBE*
WRITE_IN_PKT[6]*
READ_OUT_PKT[6]*

FIG. 8B

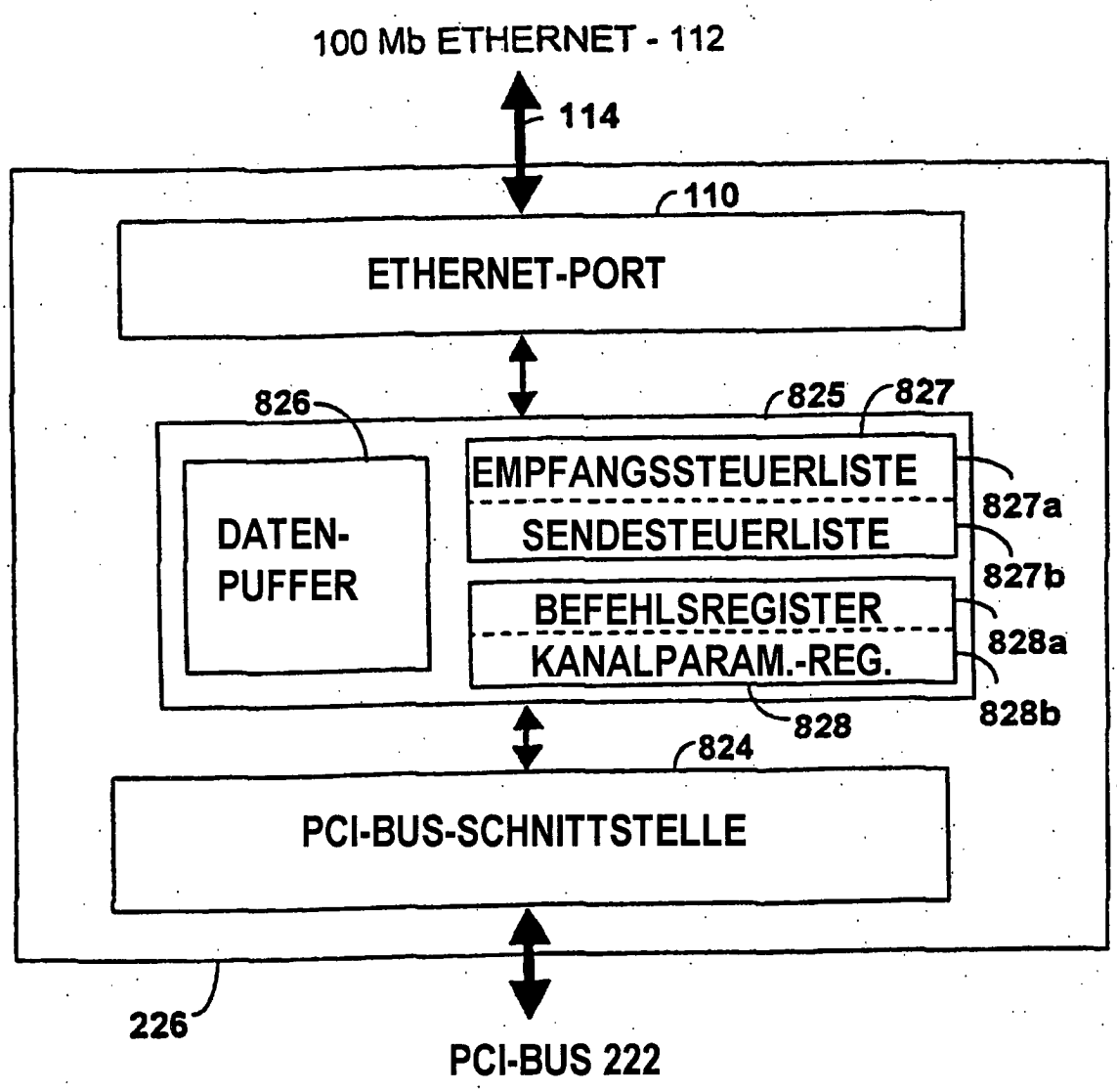

100 Mb ETHERNET - 112
114
110
ETHERNET-PORT
826
825
827
DATEN-
PUFFER
EMPFANGSSTEUERLISTE
SENDESTEUERLISTE
827a
827b
BEFEHLSREGISTER
KANALPARAM.-REG.
828a
828b
828
824
PCI-BUS-SCHNITTSTELLE
226
PCI-BUS 222

FIG. 8C

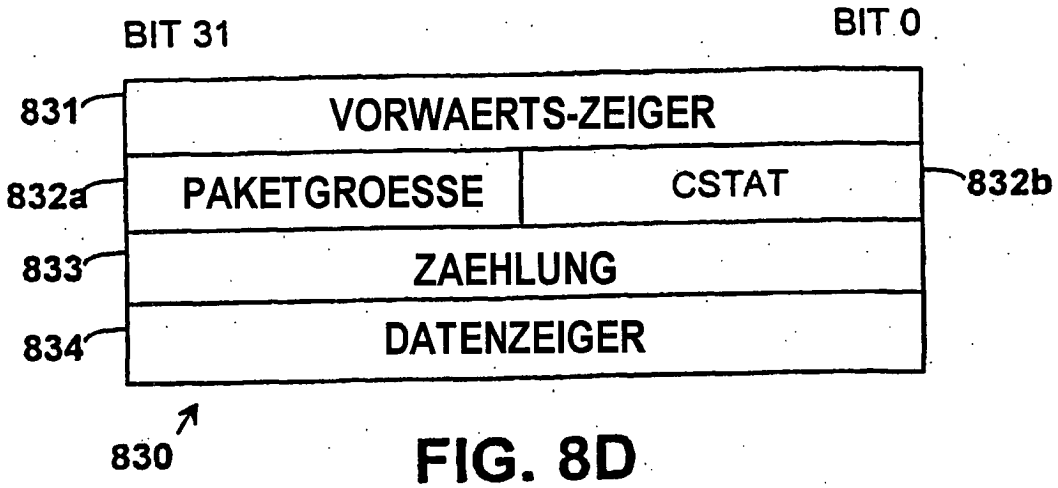

BIT 31
BIT 0
831
VORWAERTS-ZEIGER
832a
PAKETGROESSE
CSTAT
832b
833
ZAEHLUNG
834
DATENZEIGER
830

FIG. 8D

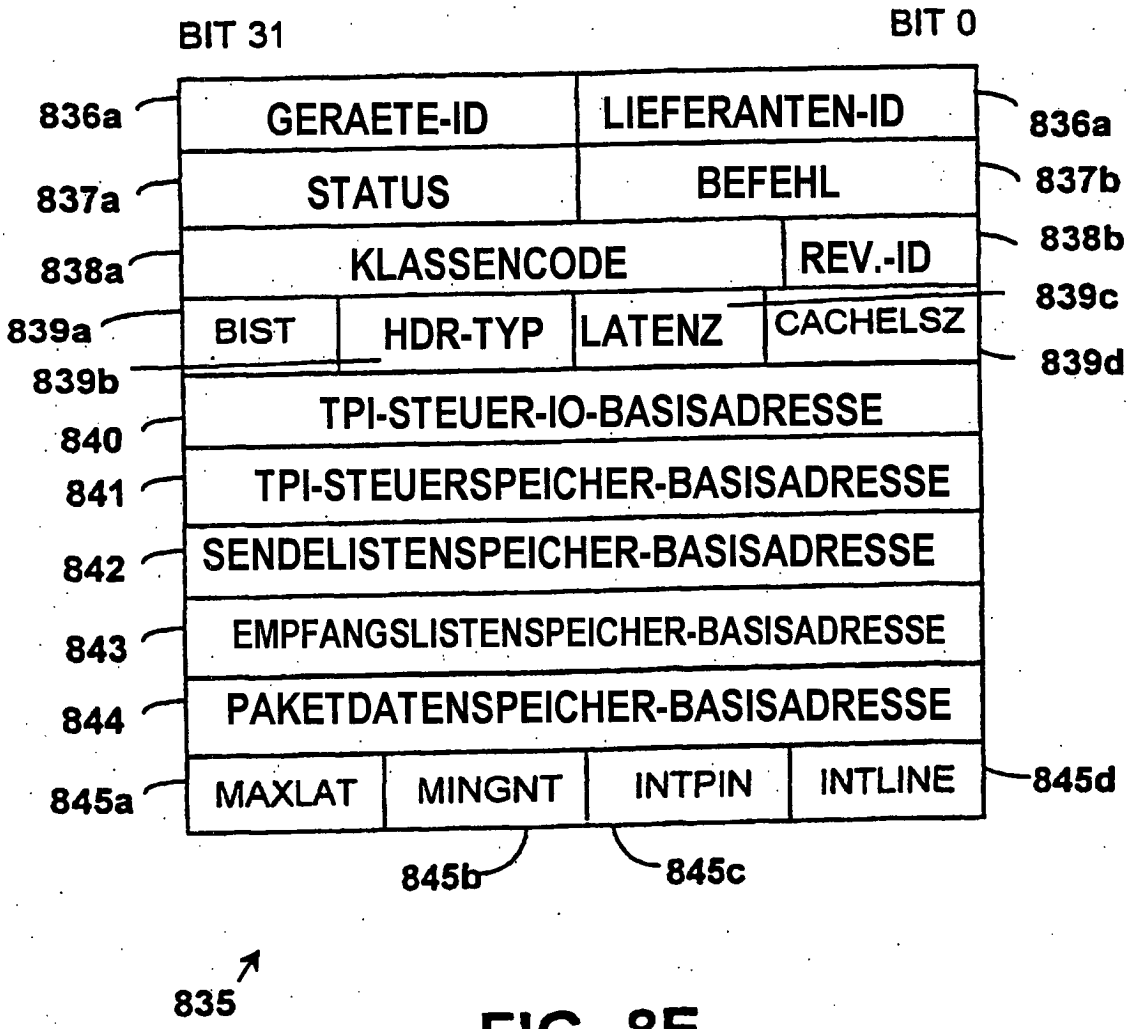

BIT 31
BIT 0
836a
GERAETE-ID
LIEFERANTEN-ID
836a
837a
STATUS
BEFEHL
837b
838a
KLASSENCODE
REV.-ID
838b
839c
839a
BIST
HDR-TYP
LATENZ
CACHELSZ
839d
839b
840
TPI-STEUER-IO-BASISADRESSE
841
TPI-STEUERSPEICHER-BASISADRESSE
842
SENDELISTENSPEICHER-BASISADRESSE
843
EMPFANGSLISTENSPEICHER-BASISADRESSE
844
PAKETDATENSPEICHER-BASISADRESSE
845a
MAXLAT
MINGNT
INTPIN
INTLINE
845d
845b
845c
835

FIG. 8E

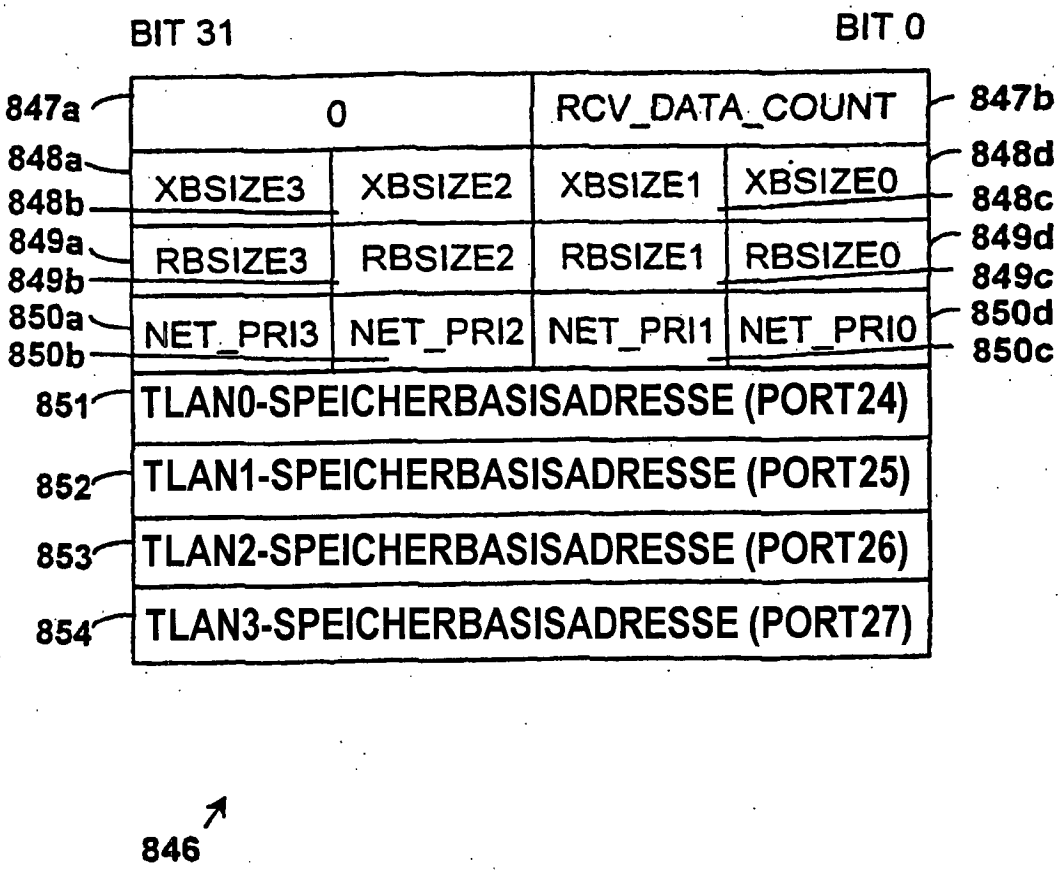

BIT 31
BIT 0
847a
0
RCV_DATA_COUNT
847b
848a
848b
XBSIZE3
XBSIZE2
XBSIZE1
XBSIZE0
848d
848c
849a
849b
RBSIZE3
RBSIZE2
RBSIZE1
RBSIZE0
849d
849c
850a
850b
NET_PRI3
NET_PRI2
NET_PRI1
NET_PRI0
850d
850c
851
TLAN0-SPEICHERBASISADRESSE (PORT24)
852
TLAN1-SPEICHERBASISADRESSE (PORT25)
853
TLAN2-SPEICHERBASISADRESSE (PORT26)
854
TLAN3-SPEICHERBASISADRESSE (PORT27)
846

FIG. 8F

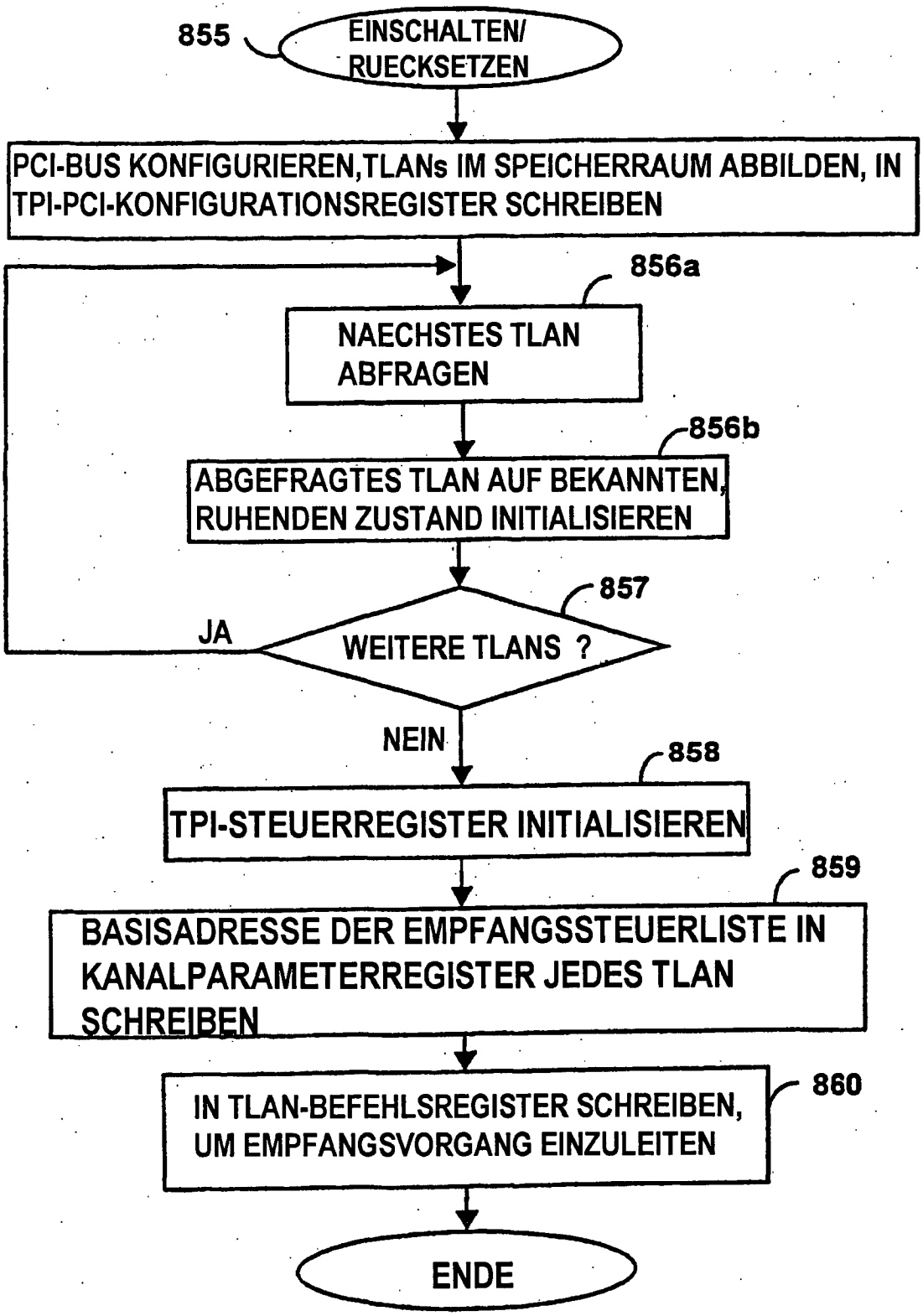

855
EINSCHALTEN/ RUECKSETZEN
PCI-BUS KONFIGURIEREN,TLANs IM SPEICHERRAUM ABBILDEN, IN TPI-PCI-KONFIGURATIONSREGISTER SCHREIBEN
856a
NAECHSTES TLAN ABFRAGEN
856b
ABGEFRAGTES TLAN AUF BEKANNTEN, RUHENDEN ZUSTAND INITIALISIEREN
857
JA
WEITERE TLANS ?
NEIN
858
TPI-STEUERREGISTER INITIALISIEREN
859
BASISADRESSE DER EMPFANGSSTEUERLISTE IN KANALPARAMETERREGISTER JEDES TLAN SCHREIBEN
860
IN TLAN-BEFEHLSREGISTER SCHREIBEN, UM EMPFANGSVORGANG EINZULEITEN
ENDE

FIG. 8G

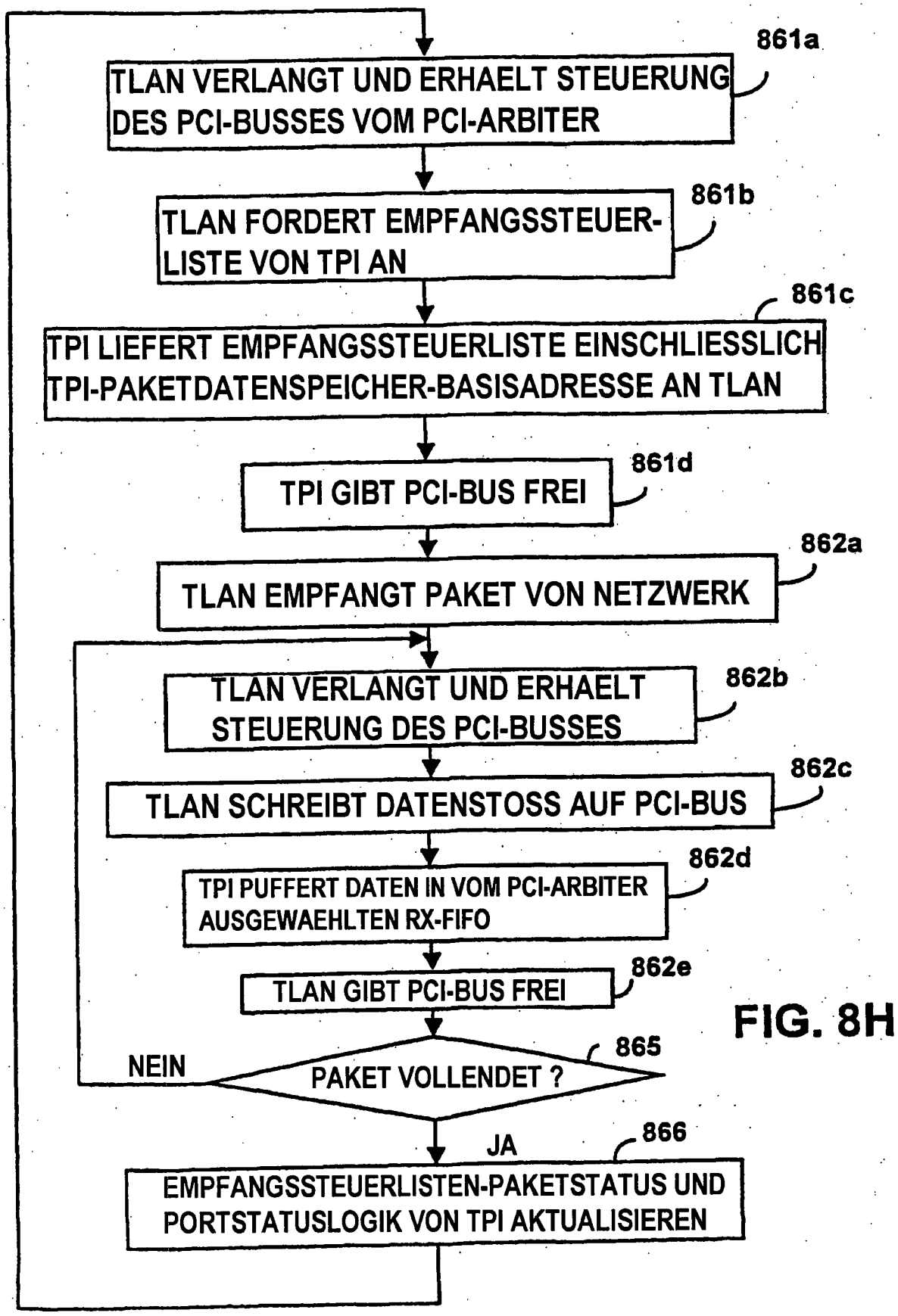

861a
TLAN VERLANGT UND ERHAELT STEUERUNG DES PCI-BUSSES VOM PCI-ARBITER
861b
TLAN FORDERT EMPFANGSSTEUER-LISTE VON TPI AN
861c
TPI LIEFERT EMPFANGSSTEUERLISTE EINSCHLIESSLICH TPI-PAKETDATENSPEICHER-BASISADRESSE AN TLAN
861d
TPI GIBT PCI-BUS FREI
862a
TLAN EMPFANGT PAKET VON NETZWERK
862b
TLAN VERLANGT UND ERHAELT STEUERUNG DES PCI-BUSSES
862c
TLAN SCHREIBT DATENSTOSS AUF PCI-BUS
862d
TPI PUFFERT DATEN IN VOM PCI-ARBITER AUSGEWAEHLTEN RX-FIFO
862e
TLAN GIBT PCI-BUS FREI
865
NEIN
PAKET VOLLENDET ?
JA
866
EMPFANGSSTEUERLISTEN-PAKETSTATUS UND PORTSTATUSLOGIK VON TPI AKTUALISIEREN

FIG. 8H

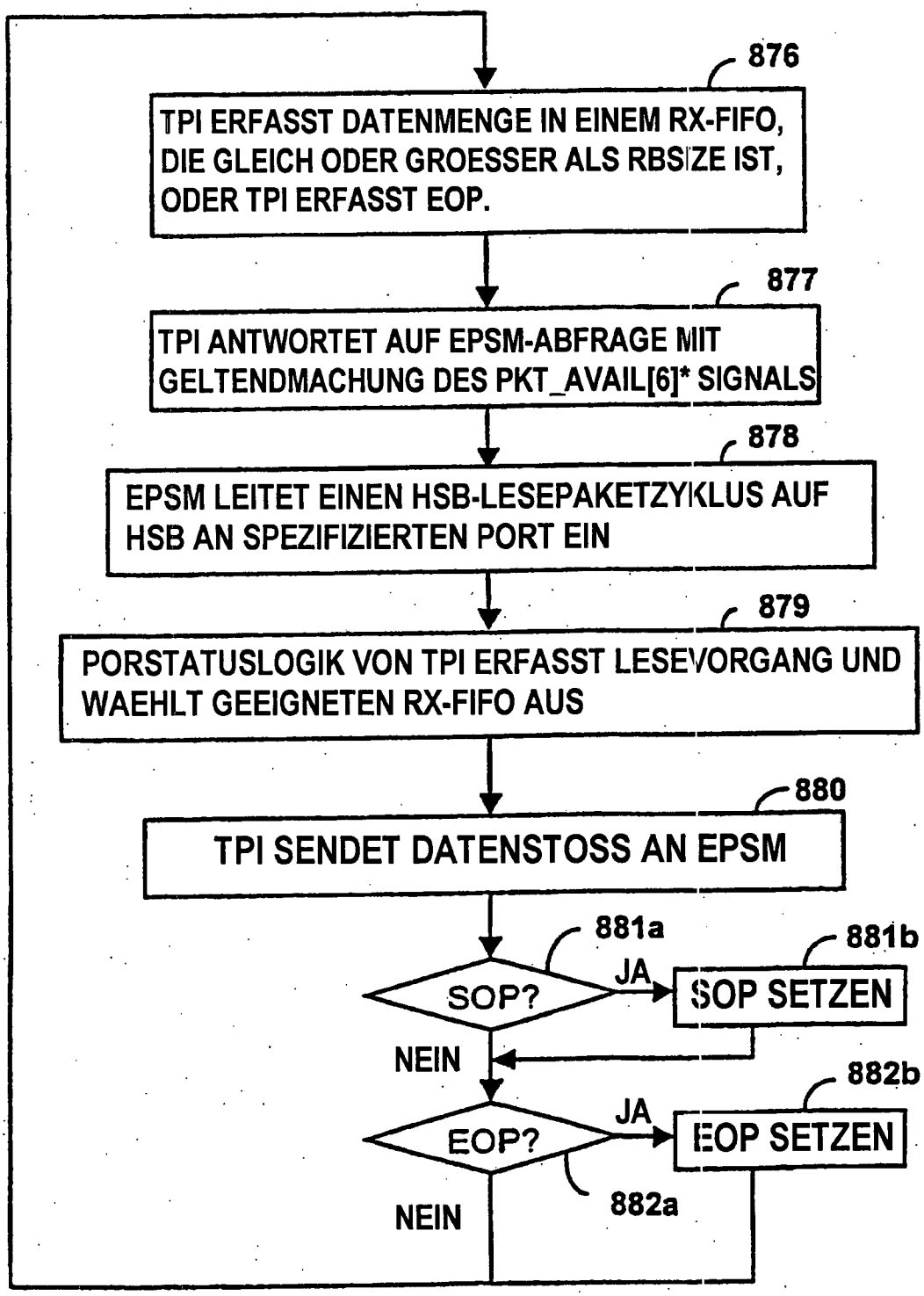

876
TPI ERFASST DATENMENGE IN EINEM RX-FIFO, DIE GLEICH ODER GROESSER ALS RBSIZE IST, ODER TPI ERFASST EOP.
877
TPI ANTWORTET AUF EPSM-ABFRAGE MIT GELTENDMACHUNG DES PKT_AVAIL[6]* SIGNALS
878
EPSM LEITET EINEN HSB-LESEPAKETZYKLUS AUF HSB AN SPEZIFIZIERTEN PORT EIN
879
PORSTATUSLOGIK VON TPI ERFASST LESEVORGANG UND WAEHLT GEEIGNETEN RX-FIFO AUS
880
TPI SENDET DATENSTOSS AN EPSM
881a
SOP?
JA
881b
SOP SETZEN
NEIN
882b
EOP?
JA
EOP SETZEN
882a
NEIN

FIG. 8I

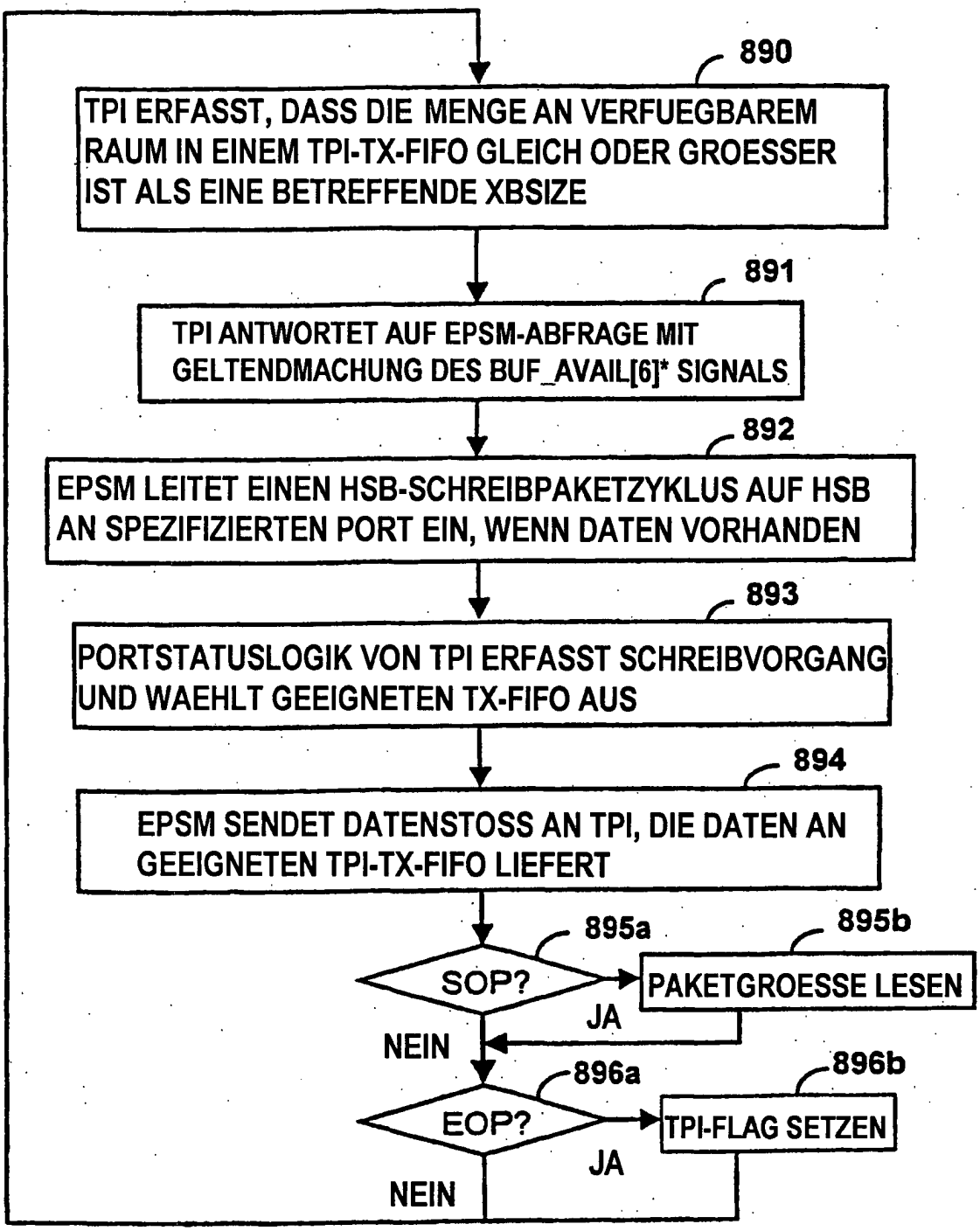

890
TPI ERFASST, DASS DIE MENGE AN VERFUEGBAREM RAUM IN EINEM TPI-TX-FIFO GLEICH ODER GROESSER IST ALS EINE BETREFFENDE XBSIZE
891
TPI ANTWORTET AUF EPSM-ABFRAGE MIT GELTENDMACHUNG DES BUF_AVAIL[6]* SIGNALS
892
EPSM LEITET EINEN HSB-SCHREIBPAKETZYKLUS AUF HSB AN SPEZIFIZIERTEN PORT EIN, WENN DATEN VORHANDEN
893
PORTSTATUSLOGIK VON TPI ERFASST SCHREIBVORGANG UND WAEHLT GEEIGNETEN TX-FIFO AUS
894
EPSM SENDET DATENSTOSS AN TPI, DIE DATEN AN GEEIGNETEN TPI-TX-FIFO LIEFERT
895a
SOP?
JA
895b
PAKETGROESSE LESEN
NEIN
896a
EOP?
JA
896b
TPI-FLAG SETZEN
NEIN

FIG. 8J

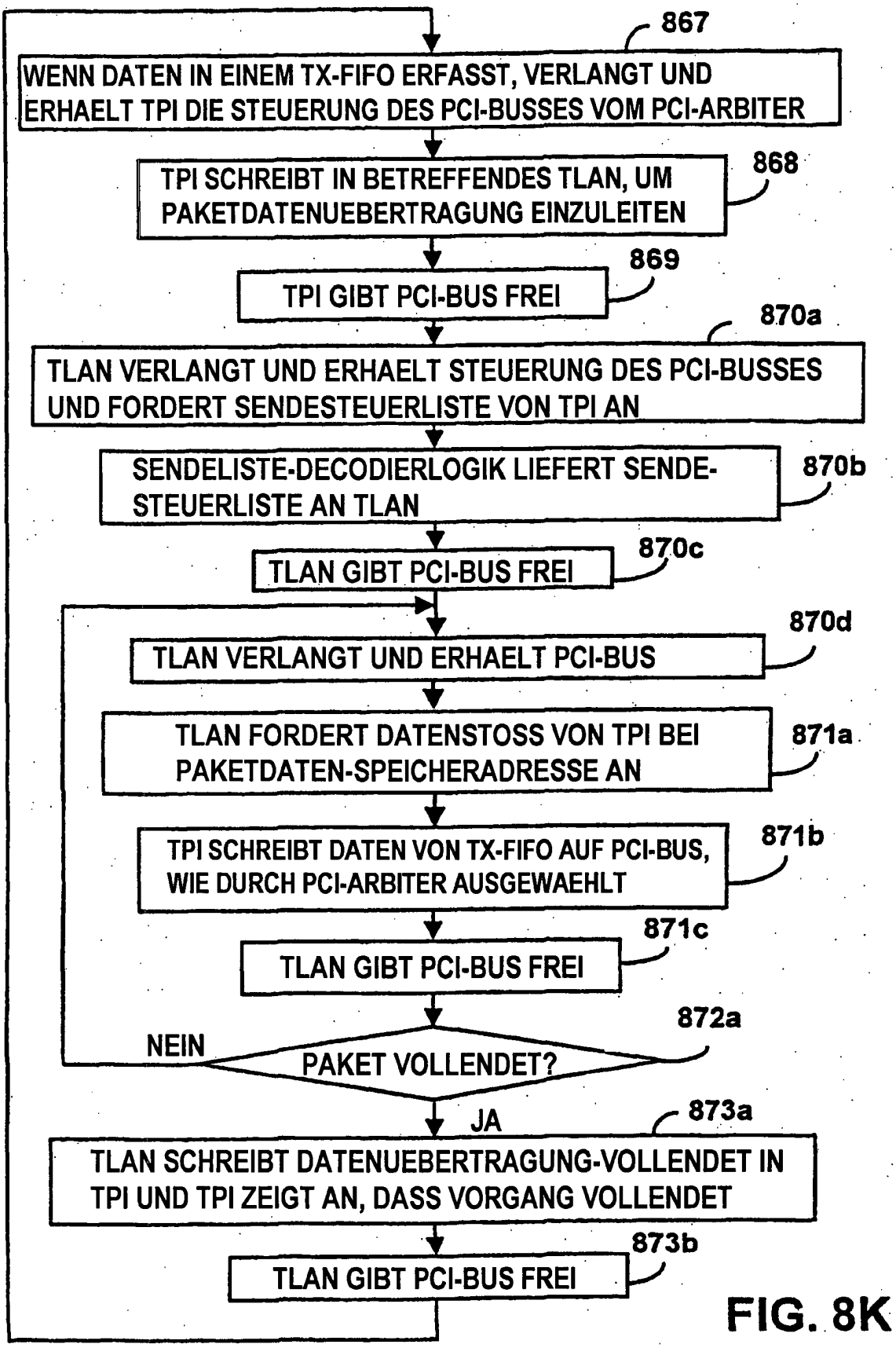

867
WENN DATEN IN EINEM TX-FIFO ERFASST, VERLANGT UND ERHAELT TPI DIE STEUERUNG DES PCI-BUSSES VOM PCI-ARBITER
868
TPI SCHREIBT IN BETREFFENDES TLAN, UM PAKETDATENUEBERTRAGUNG EINZULEITEN
869
TPI GIBT PCI-BUS FREI
870a
TLAN VERLANGT UND ERHAELT STEUERUNG DES PCI-BUSSES UND FORDERT SENDESTEUERLISTE VON TPI AN
870b
SENDELISTE-DECODIERLOGIK LIEFERT SENDE-STEUERLISTE AN TLAN
870c
TLAN GIBT PCI-BUS FREI
870d
TLAN VERLANGT UND ERHAELT PCI-BUS
871a
TLAN FORDERT DATENSTOSS VON TPI BEI PAKETDATEN-SPEICHERADRESSE AN
871b
TPI SCHREIBT DATEN VON TX-FIFO AUF PCI-BUS, WIE DURCH PCI-ARBITER AUSGEWAEHLT
871c
TLAN GIBT PCI-BUS FREI
872a
NEIN
PAKET VOLLENDET?
JA
873a
TLAN SCHREIBT DATENUEBERTRAGUNG-VOLLENDET IN TPI UND TPI ZEIGT AN, DASS VORGANG VOLLENDET
873b
TLAN GIBT PCI-BUS FREI

FIG. 8K

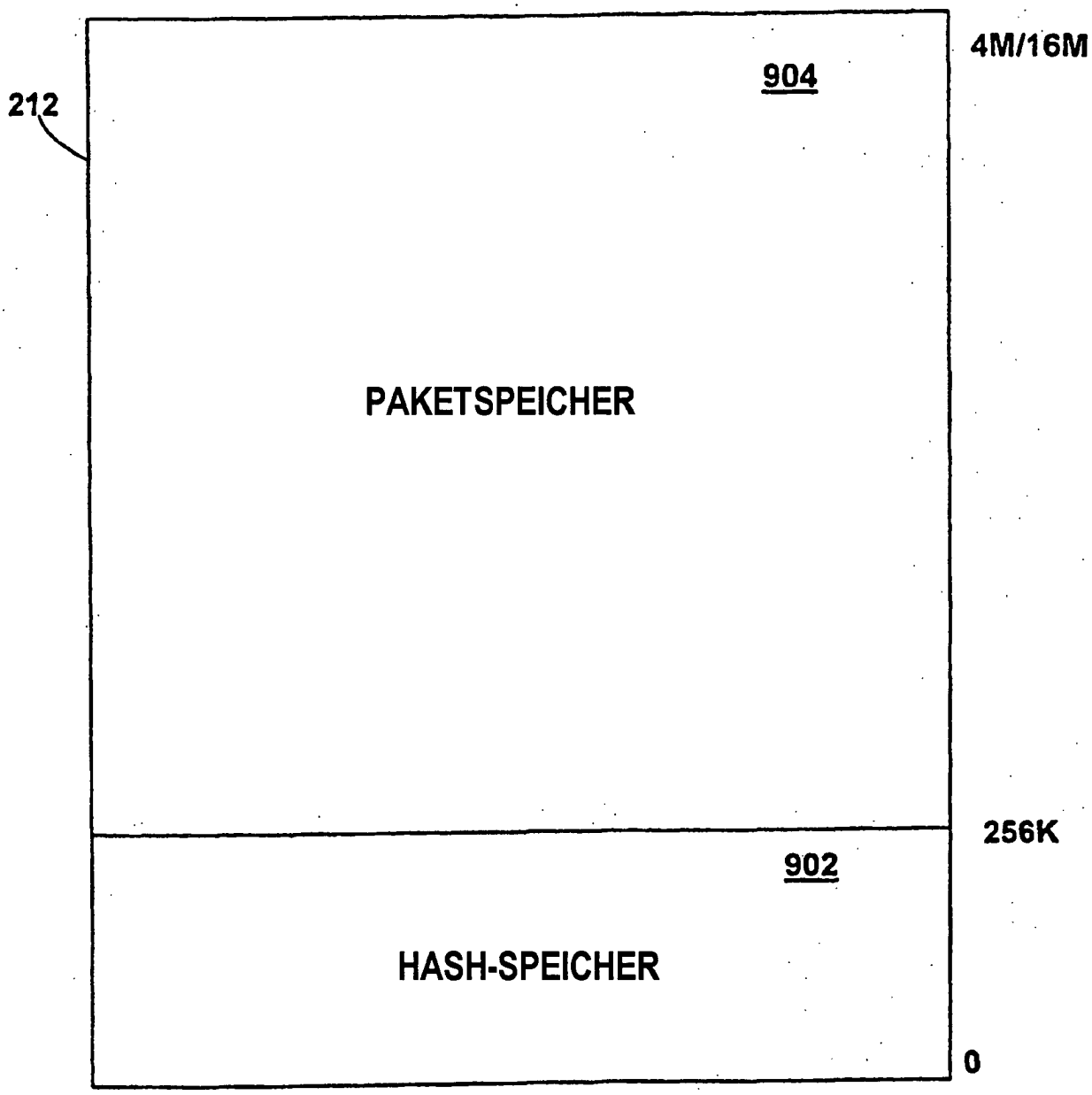


FIG. 9A

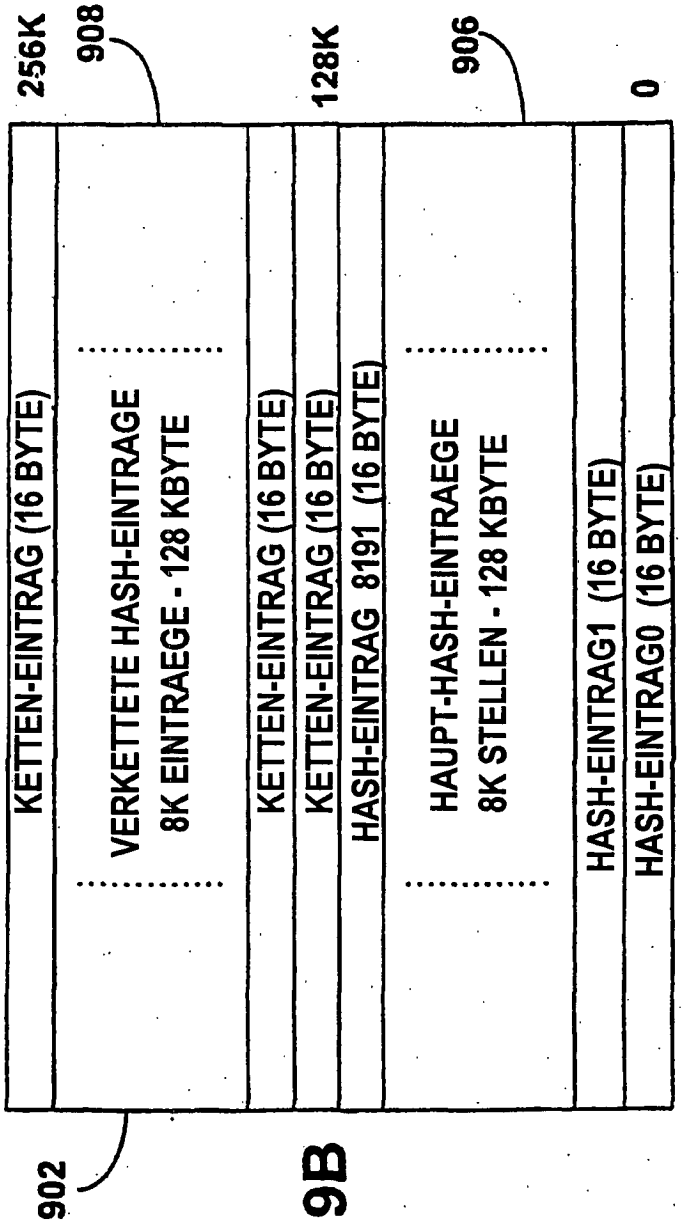

256K
908
128K
906
0
KETTEN-EINTRAG (16 BYTE)
VERKETTETE HASH-EINTRAGE
8K EINTRAEGE - 128 KBYTE
KETTEN-EINTRAG (16 BYTE)
KETTEN-EINTRAG (16 BYTE)
HASH-EINTRAG 8191 (16 BYTE)
HAUPT-HASH-EINTRAEGE
8K STELLEN - 128 KBYTE
HASH-EINTRAG1 (16 BYTE)
HASH-EINTRAG0 (16 BYTE)
902

FIG. 9B

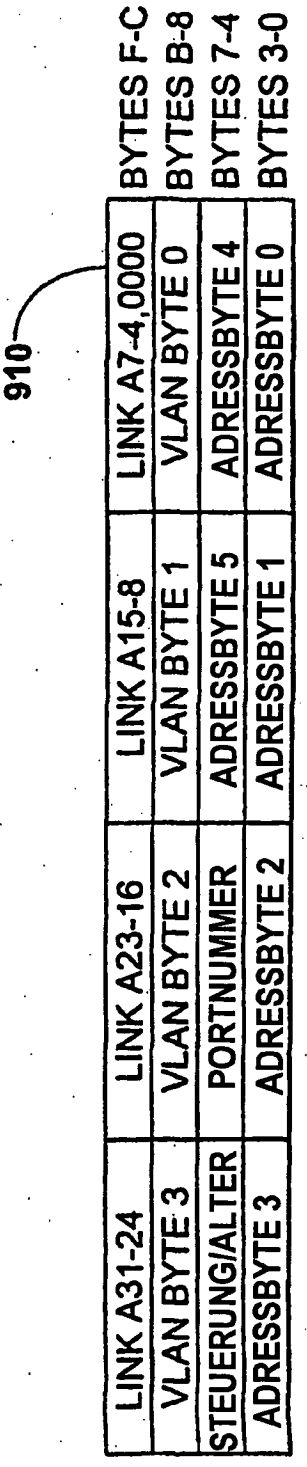

910
LINK A31-24
LINK A23-16
LINK A15-8
LINK A7-4,0000
BYTES F-C
VLAN BYTE 3
VLAN BYTE 2
VLAN BYTE 1
VLAN BYTE 0
BYTES B-8
STEUERUNG/ALTER
PORTNUMMER
ADRESSBYTE 5
ADRESSBYTE 4
BYTES 7-4
ADRESSBYTE 3
ADRESSBYTE 2
ADRESSBYTE 1
ADRESSBYTE 0
BYTES 3-0

FIG. 9C

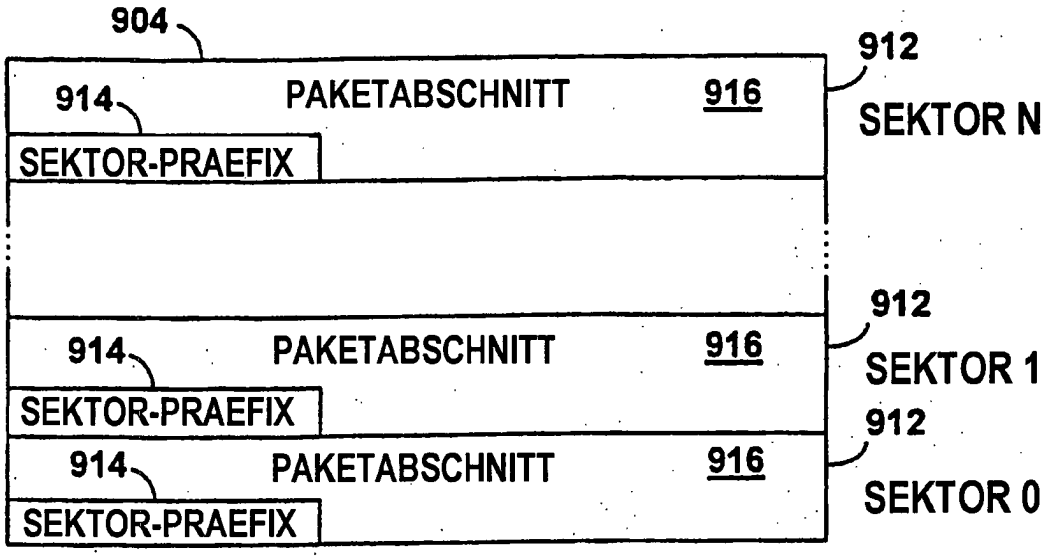


FIG. 9D

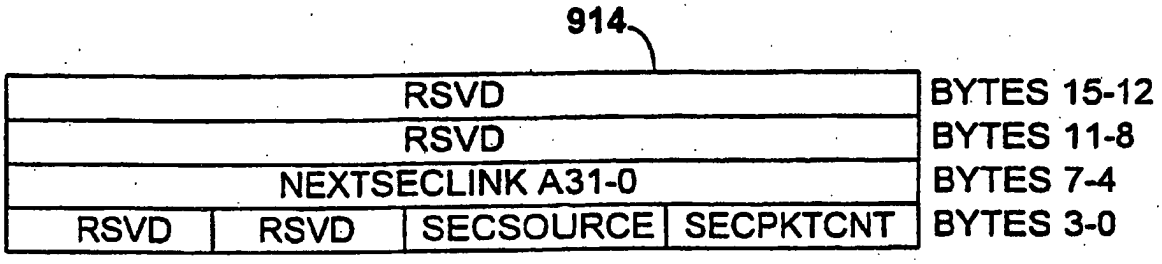


FIG. 9E

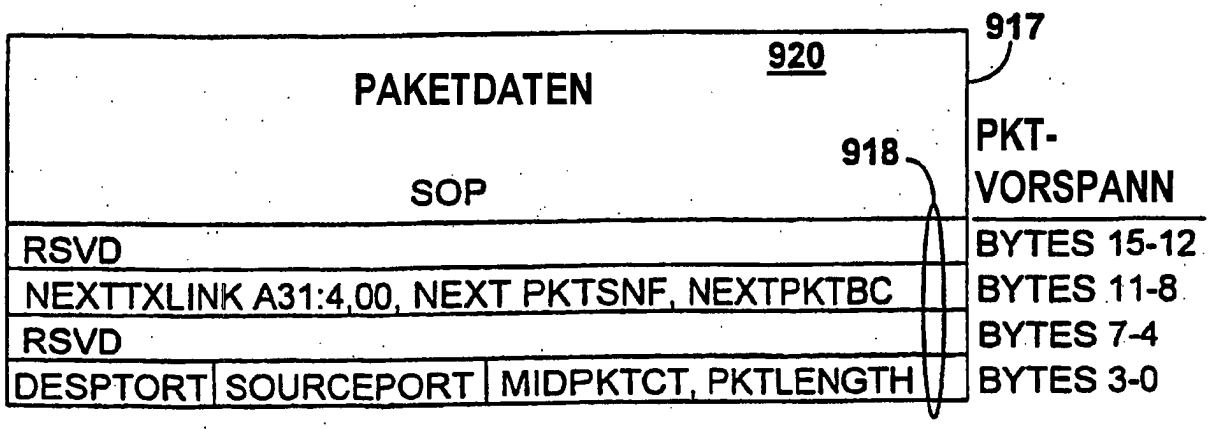


FIG. 9F

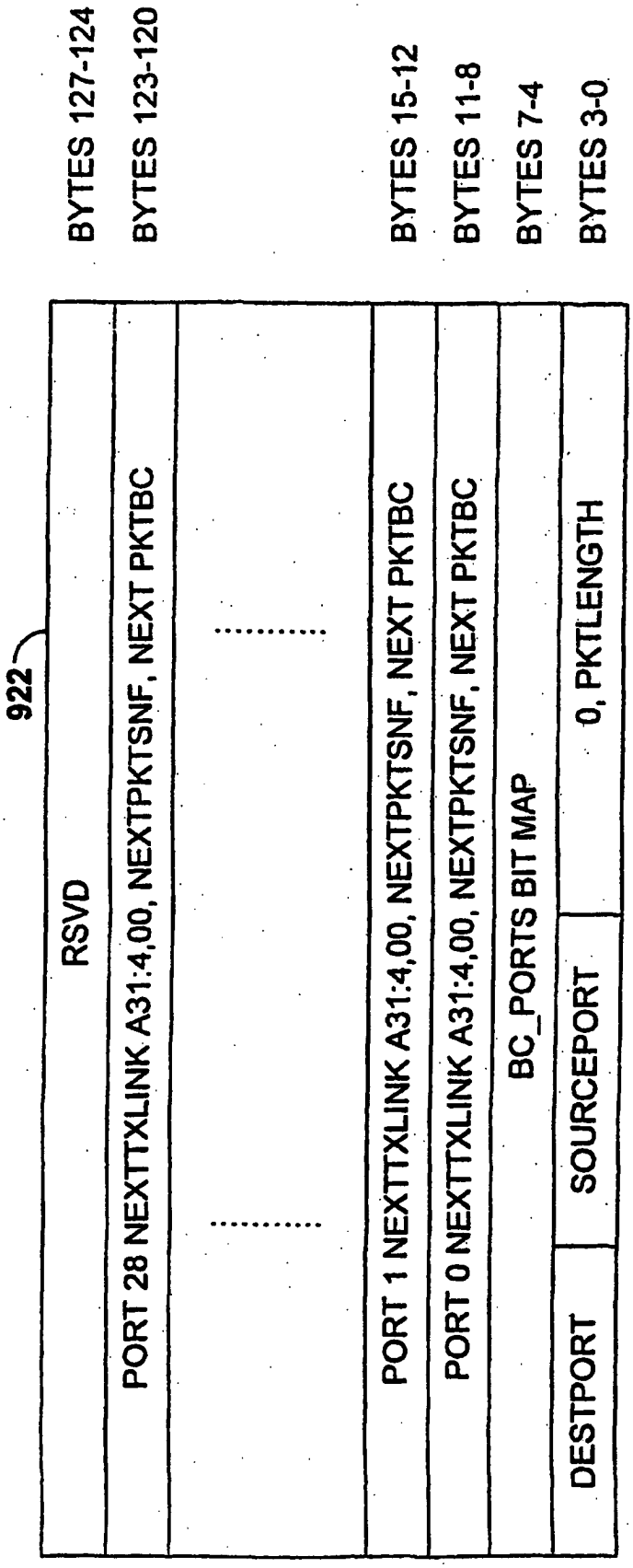

922
RSVD
BYTES 127-124
PORT 28 NEXTTXLINK A31:4,00, NEXTPKTSNF, NEXT PKTBC
BYTES 123-120
PORT 1 NEXTTXLINK A31:4,00, NEXTPKTSNF, NEXT PKTBC
BYTES 15-12
PORT 0 NEXTTXLINK A31:4,00, NEXTPKTSNF, NEXT PKTBC
BYTES 11-8
BC_PORTS BIT MAP
BYTES 7-4
DESTPORT
SOURCEPORT
0, PKTLENGTH
BYTES 3-0

FIG. 9G

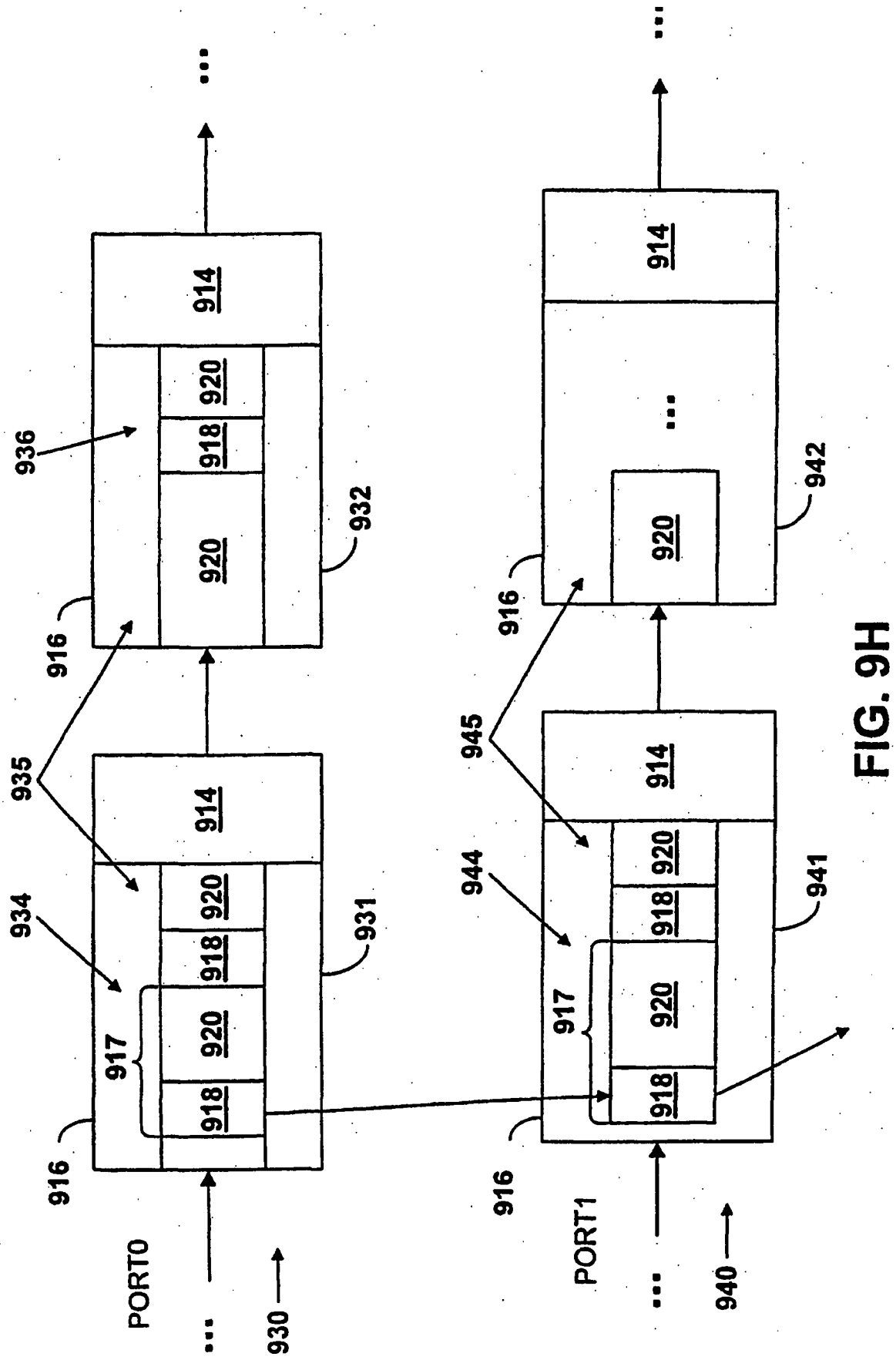

914
920
918
920
936
932
916
935
914
920
918
920
934
931
917
916
PORT0
930
914
920
942
916
945
914
920
918
920
944
941
917
916
PORT1
940

FIG. 9H

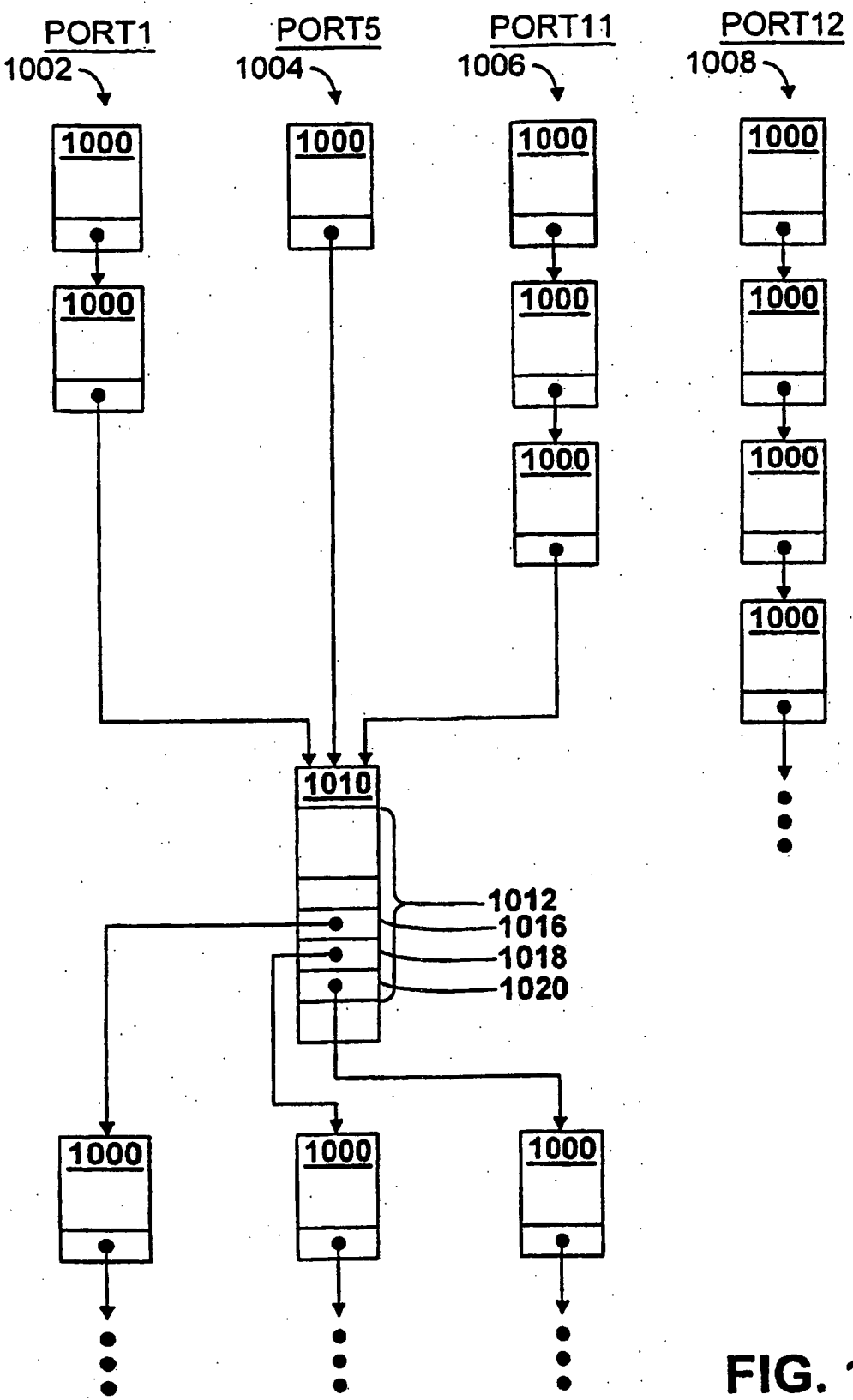

PORT1
1002
PORT5
1004
PORT11
1006
PORT12
1008
1000
1010
1012
1016
1018
1020

FIG. 10

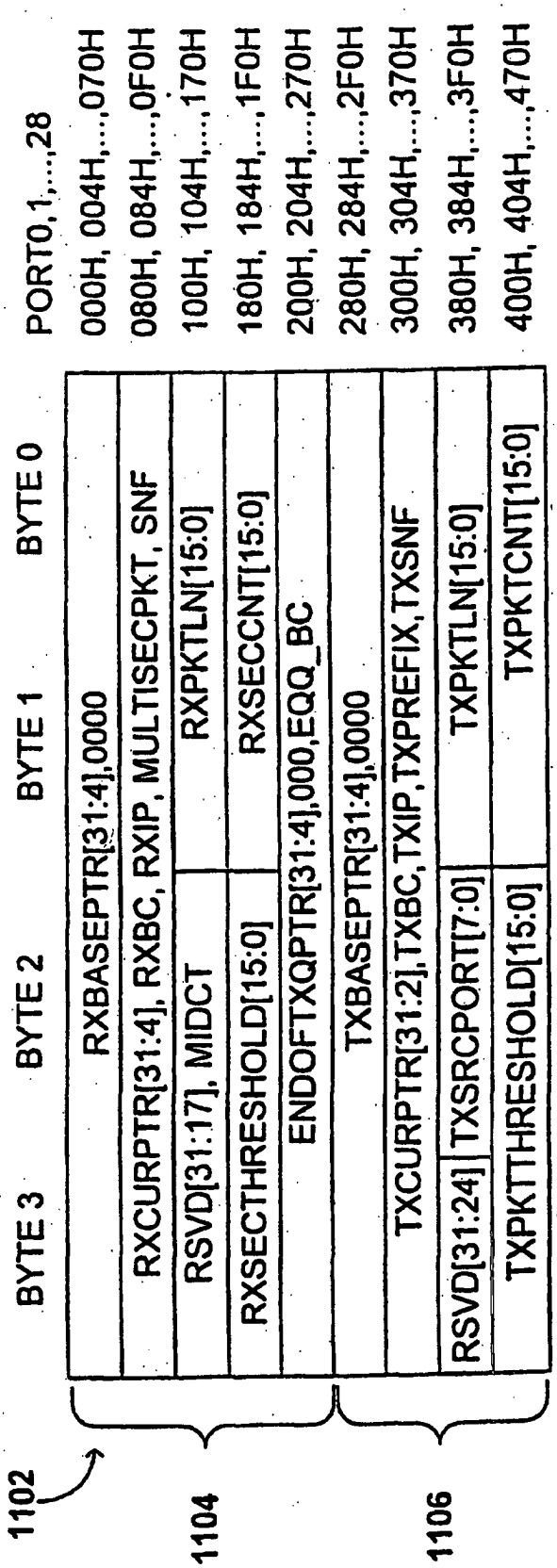


FIG. 11A

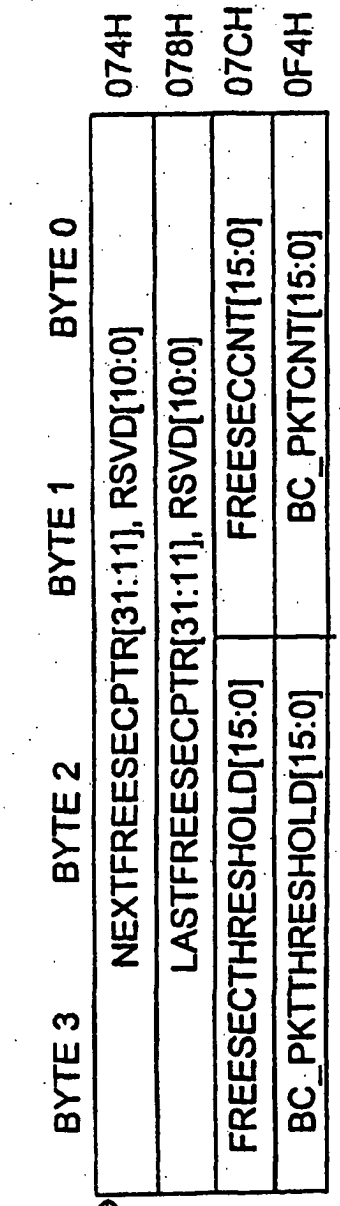


FIG. 11B

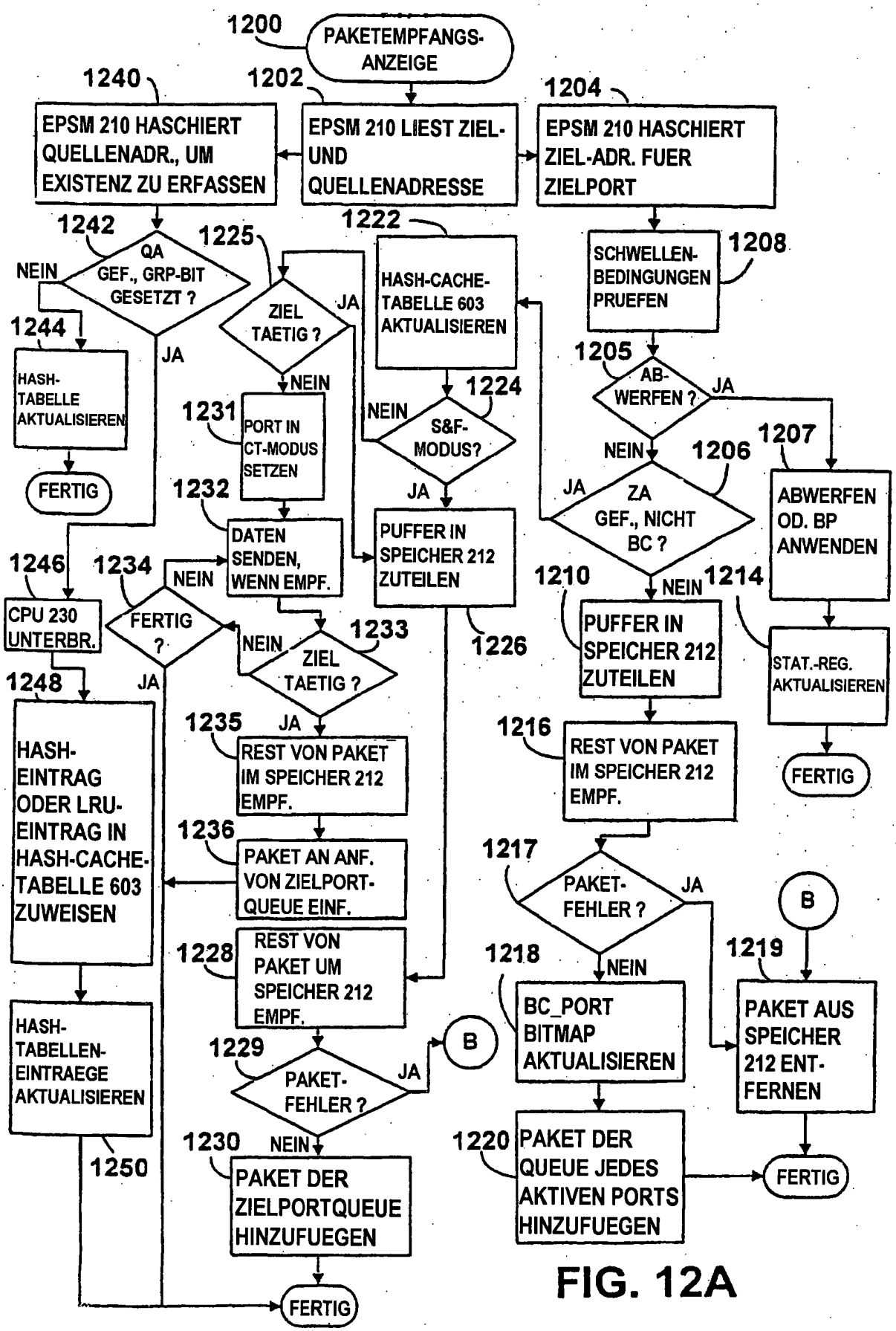
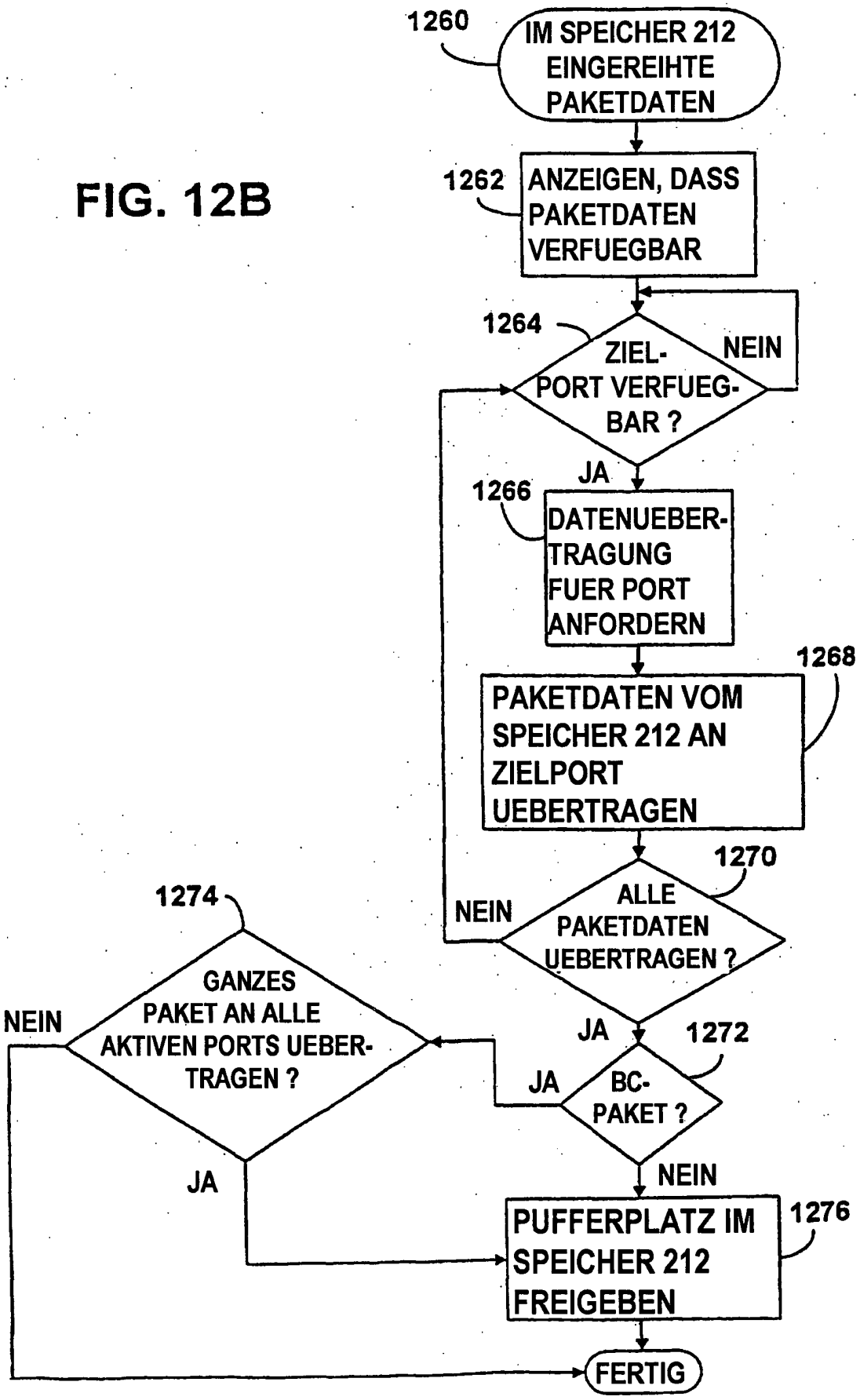

1200
PAKETEMPFANGS-ANZEIGE
1202
EPSM 210 LIEST ZIEL- UND QUELLENADRESSE
1240
EPSM 210 HASCHIERT QUELLENADR., UM EXISTENZ ZU ERFASSEN
1204
EPSM 210 HASCHIERT ZIEL-ADR. FUER ZIELPORT
1242
QA GEF., GRP-BIT GESETZT ?
NEIN
JA
1244
HASH-TABELLE AKTUALISIEREN
FERTIG
1225
ZIEL TAETIG ?
JA
NEIN
1231
PORT IN CT-MODUS SETZEN
1232
DATEN SENDEN, WENN EMPF.
1222
HASH-CACHE-TABELLE 603 AKTUALISIEREN
1224
S&F-MODUS?
NEIN
JA
PUFFER IN SPEICHER 212 ZUTEILEN
1226
1208
SCHWELLEN-BEDINGUNGEN PRUEFEN
1205
AB-WERFEN ?
JA
NEIN
1206
ZA GEF., NICHT BC ?
JA
NEIN
1207
ABWERFEN OD. BP ANWENDEN
1210
PUFFER IN SPEICHER 212 ZUTEILEN
1214
STAT.-REG. AKTUALISIEREN
FERTIG
1216
REST VON PAKET IM SPEICHER 212 EMPF.
1217
PAKET-FEHLER ?
JA
NEIN
B
1219
PAKET AUS SPEICHER 212 ENT-FERNEN
1218
BC_PORT BITMAP AKTUALISIEREN
1220
PAKET DER QUEUE JEDES AKTIVEN PORTS HINZUFUEGEN
FERTIG
1246
CPU 230 UNTERBR.
1234
FERTIG ?
NEIN
JA
1233
ZIEL TAETIG ?
NEIN
JA
1235
REST VON PAKET IM SPEICHER 212 EMPF.
1236
PAKET AN ANF. VON ZIELPORT-QUEUE EINF.
1228
REST VON PAKET UM SPEICHER 212 EMPF.
1229
PAKET-FEHLER ?
JA
NEIN
B
1230
PAKET DER ZIELPORTQUEUE HINZUFUEGEN
1248
HASH-EINTRAG ODER LRU-EINTRAG IN HASH-CACHE-TABELLE 603 ZUWEISEN
HASH-TABELLEN-EINTRAEGE AKTUALISIEREN
1250
FERTIG

FIG. 12A

FIG. 12B


1260
IM SPEICHER 212 EINGEREIHTE PAKETDATEN
1262
ANZEIGEN, DASS PAKETDATEN VERFUEGBAR
1264
ZIEL-PORT VERFUEG-BAR ?
NEIN
JA
1266
DATENUEBER-TRAGUNG FUER PORT ANFORDERN
1268
PAKETDATEN VOM SPEICHER 212 AN ZIELPORT UEBERTRAGEN
1270
ALLE PAKETDATEN UEBERTRAGEN ?
NEIN
JA
1272
BC-PAKET ?
JA
NEIN
1274
GANZES PAKET AN ALLE AKTIVEN PORTS UEBER-TRAGEN ?
NEIN
JA
1276
PUFFERPLATZ IM SPEICHER 212 FREIGEBEN
FERTIG

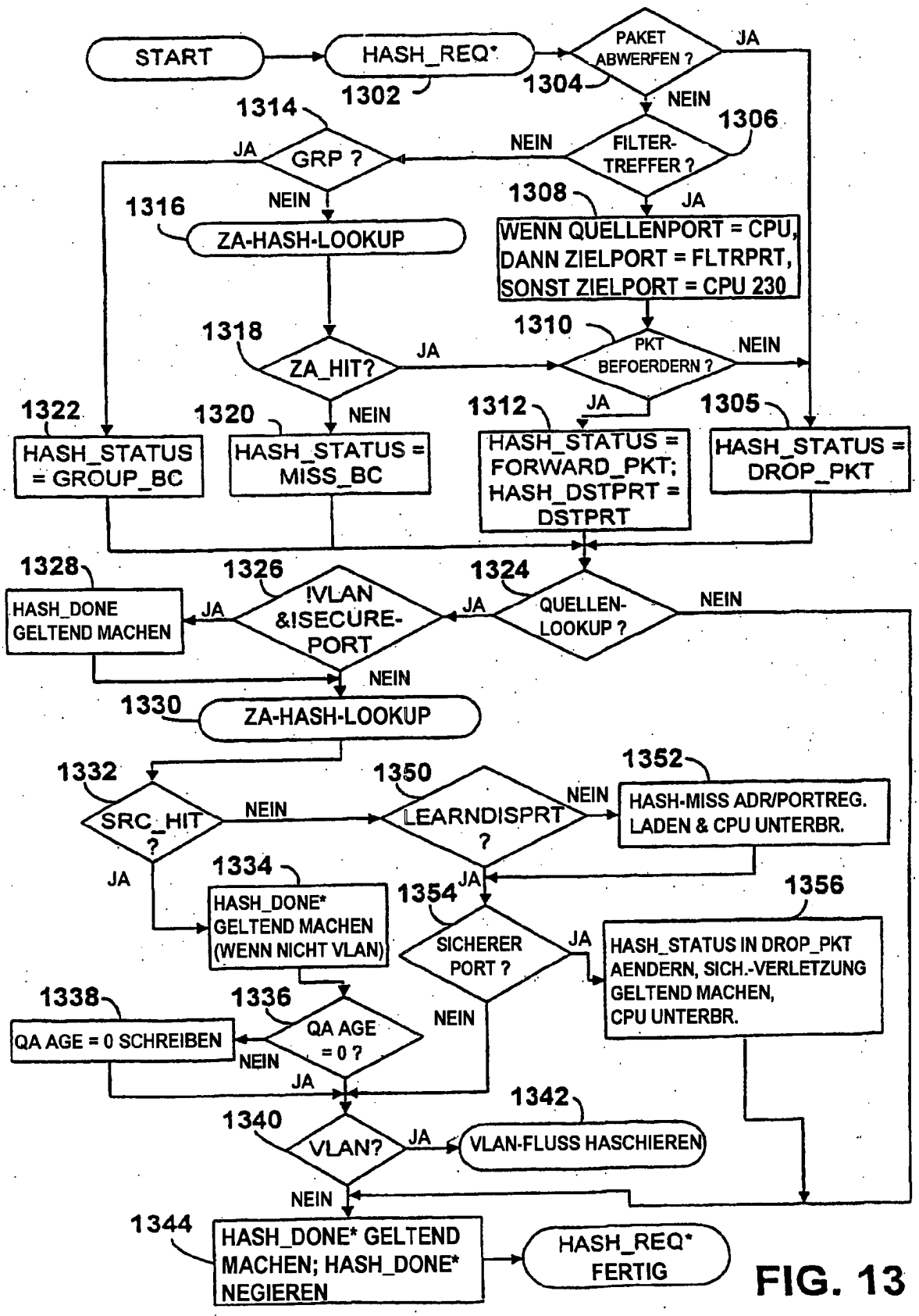

START
HASH_REQ*
1302
PAKET ABWERFEN ?
1304
JA
NEIN
1306
FILTER-TREFFER ?
NEIN
JA
1314
GRP ?
JA
NEIN
1316
ZA-HASH-LOOKUP
1308
WENN QUELLENPORT = CPU, DANN ZIELPORT = FLTRPRT, SONST ZIELPORT = CPU 230
JA
1310
PKT BEFOERDERN ?
NEIN
1318
ZA_HIT?
JA
NEIN
1322
HASH_STATUS = GROUP_BC
1320
HASH_STATUS = MISS_BC
1312
HASH_STATUS = FORWARD_PKT; HASH_DSTPRT = DSTPRT
JA
1305
HASH_STATUS = DROP_PKT
1328
HASH_DONE GELTEND MACHEN
JA
1326
!VLAN &ISECURE-PORT
JA
1324
QUELLEN-LOOKUP ?
NEIN
NEIN
1330
ZA-HASH-LOOKUP
1332
SRC_HIT ?
NEIN
1350
LEARNDISPRT ?
NEIN
1352
HASH-MISS ADR/PORTREG. LADEN & CPU UNTERBR.
JA
1334
HASH_DONE* GELTEND MACHEN (WENN NICHT VLAN)
1354
JA
SICHERER PORT ?
JA
1356
HASH_STATUS IN DROP_PKT AENDERN, SICH.-VERLETZUNG GELTEND MACHEN, CPU UNTERBR.
1338
QA AGE = 0 SCHREIBEN
1336
QA AGE = 0 ?
NEIN
NEIN
JA
1342
1340
VLAN?
JA
VLAN-FLUSS HASCHIEREN
NEIN
1344
HASH_DONE* GELTEND MACHEN; HASH_DONE* NEGIEREN
HASH_REQ* FERTIG

FIG. 13

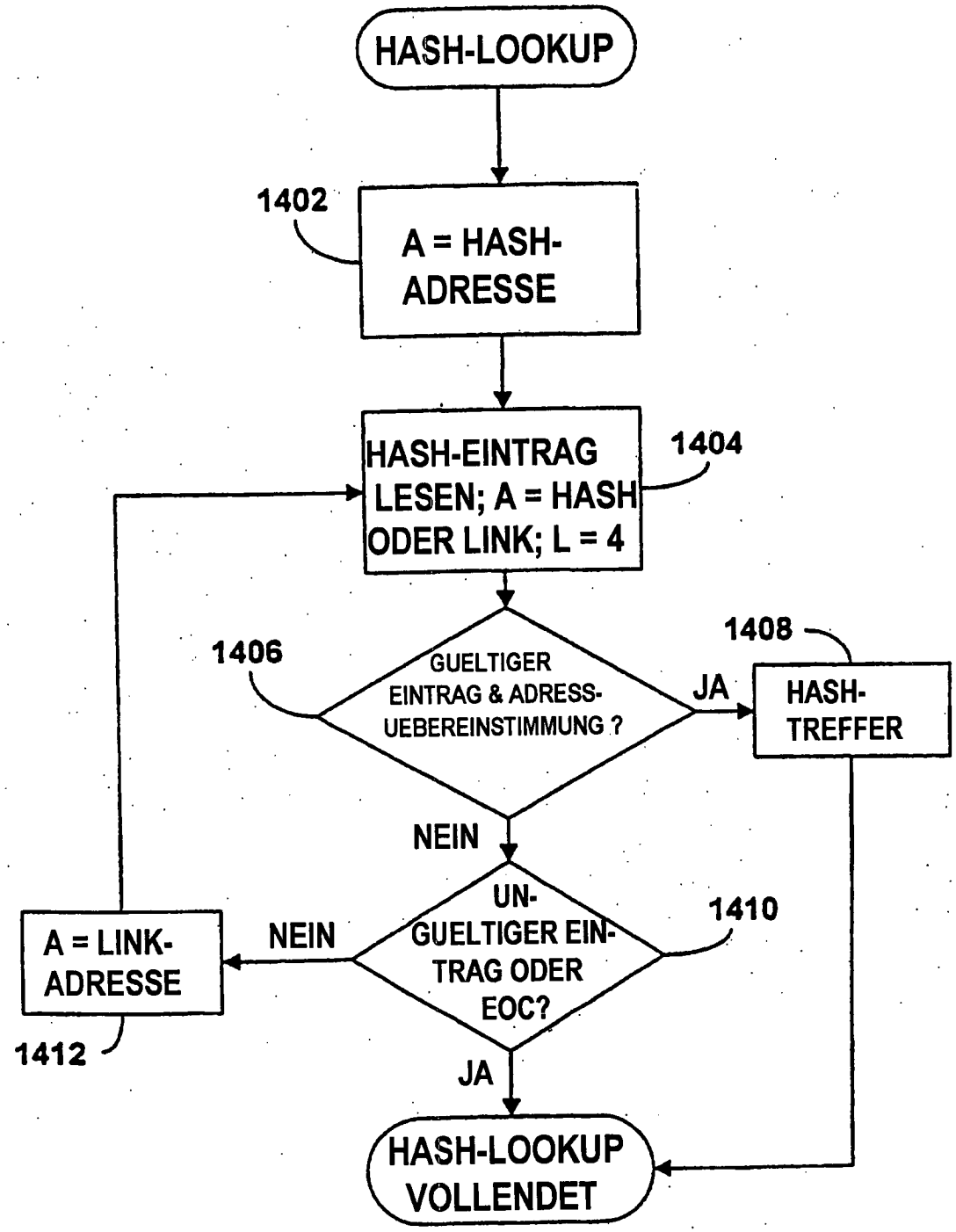

HASH-LOOKUP
1402
A = HASH-ADRESSE
HASH-EINTRAG LESEN; A = HASH ODER LINK; L = 4
1404
1406
GUELTIGER EINTRAG & ADRESS-UEBEREINSTIMMUNG ?
JA
1408
HASH-TREFFER
NEIN
UN-GUELTIGER EIN-TRAG ODER EOC?
1410
NEIN
A = LINK-ADRESSE
1412
JA
HASH-LOOKUP VOLLENDET

FIG. 14