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

(10) **DE 698 27 589 T2** 2005.11.03

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 924 625 B1**
(21) Deutsches Aktenzeichen: **698 27 589.6**
(96) Europäisches Aktenzeichen: **98 309 600.9**
(96) Europäischer Anmeldetag: **24.11.1998**
(97) Erstveröffentlichung durch das EPA: **23.06.1999**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **17.11.2004**
(47) Veröffentlichungstag im Patentblatt: **03.11.2005**

(51) Int Cl.[7]: **G06F 15/78**

(30) Unionspriorität:

97310220	**17.12.1997**	**EP**
9811776	**02.06.1998**	**GB**

(84) Benannte Vertragsstaaten: **DE, FR, GB**

(73) Patentinhaber:
Elixent Ltd., Bristol, GB

(74) Vertreter:
WUESTHOFF & WUESTHOFF Patent- und Rechtsanwälte, 81541 München

(72) Erfinder:
Stansfield, Anthony, Hotwells, Bristol BS8 4YB, GB; Marshall, Alan David, Merchant's Landing, Bristol BS1 4RJ, GB; Vuillemin, Prof., Jean, 75116 Paris, FR

(54) Bezeichnung: **Konfigurierbare Verarbeitungsanordnung und Verfahren zur Benutzung dieser Anordnung, um eine Zentraleinheit aufzubauen**

Beschreibung

[0001] Die vorliegende Erfindung betrifft ein umkonfigurierbares Prozessorbauteil.

[0002] Ein herkömmlicher Prozessor (wie zum Beispiel der Pentium II, hergestellt von Intel Corp. – Pentium ist eine Marke von Intel Corp.) ist ein Mehrzweckbauteil. Er ist nicht für eine spezifische Aufgabe optimiert, sondern ist vielmehr in der Lage, programmiert zu werden, um einen großen Bereich von Funktionen zu erfüllen.

[0003] Die Folge der Mehrzweckarchitektur des herkömmlichen Prozessors ist, dass die Leistung des Prozessors für spezifische Aufgaben viel schlechter sein wird als für Hardware, die dafür ausgelegt ist, die spezifischen Aufgaben zu erfüllen. Dies liegt daran, dass die Architektur des Mehrzweckprozessors nicht der Struktur der Aufgabe folgt, sondern vielmehr auf einer komplexen ALU (arithmetische Logikeinheit; engl.: arithmetic logic unit) beruht, die während der Aufgabe sehr stark benutzt wird und sehr häufige Aufrufe an ihre notwendigerweise großen Speicherressourcen macht. Wenn solche Aufgaben rechenintensiv sind, ist dieser Ansatz besonders ungeeignet.

[0004] Wenn es eine Aufgabe gibt, welche regelmäßig durchgeführt werden muss, ist ein geeigneter Ansatz, Schaltkreise bereitzustellen, die für diese Aufgabe speziell optimiert sind. Ein typischer Ansatz ist, solche Schaltkreise in der Form eines Koprozessors oder ASIC (anwendungsspezifischer integrierter Schaltkreis; engl.: applicaton specific integrated circuit) zusammen mit dem Mehrzweckprozessor bereitzustellen, so dass die Aufgaben, für welche der Koprozessor oder ASIC optimiert ist, dem Koprozessor oder ASIC von dem Mehrzweckprozessor zugeführt werden können.

[0005] Obwohl ein ASIC für eine spezifische Aufgabe optimal sein kann, da er für eine spezifische Aufgabe gebaut worden ist, wird er im Allgemeinen schwach oder vollkommen funktionsuntüchtig für jede andere Rechenaufgabe sein. Eine vorteilhafte Möglichkeit besteht zwischen den zwei Extremen: einerseits ein ASIC mit fester Konfiguration und andererseits ein herkömmlicher Prozessor (für welchen eine "Konfiguration" in silico nur als während eines einzelnen Zyklusses existierend erachtet werden kann). Diese Zwischenmöglichkeit ist ein umkonfigurierbares Bauteil: diese haben eine bestimmte Konfiguration, aber erlauben, falls benötigt, Rekonfiguration auf eine verschiedene bestimmte Konfiguration. Umkonfigurierbare Bauteile bieten daher die Möglichkeit eines Computers, der seine Hardware-Resourcen abändern kann, um seinen gegenwärtigen Rechenanforderungen durch geeignete Rekonfiguration zu dienen.

[0006] Eine kommerziell erfolgreiche Form eines umkonfigurierbaren Bauteils ist die Feldprogramierbare Gate-Anordnung (FPGA, field programmable gate array). Diese Bauteile bestehen aus einer Ansammlung von konfigurierbaren Verarbeitungselementen, die in einem konfigurierbaren Verbindungsnetzwerk eingebettet sind. Konfigurationsspeicher ist vorgesehen, um die Verbindungskonfiguration zu beschreiben – oft wird SRAM benutzt. Diese Vorrichtungen haben eine sehr feinkörnige Struktur: üblicherweise ist jedes Verarbeitungselement eines FPGA ein konfigurierbares Gate. Anstatt in einer zentralen ALU konzentriert zu sein, ist das Verarbeiten daher über das Bauteil verteilt und der Silizium-Bereich des Bauteils wird wirksamer benutzt. Ein Beispiel einer kommerziell erhältlichen FPGA-Serie ist die Xilinx-4000-Serie.

[0007] Solche umkonfigurierbaren Bauteile können prinzipiell für jede Rechenanwendung verwendet werden, für welche ein Prozessor oder ein ASIC verwendet wird. Jedoch ist eine besonders geeignete Verwendung solcher Bauteile als ein Koprozessor, um Aufgaben zu handhaben, die rechenintensiv sind, aber welche nicht so gebräuchlich sind, dass sie eine speziell angefertigte ASIC verdienten. Ein umkonfugierterbarer Koprozessor könnte daher zu verschiedenen Zeiten mit verschiedenen Konfigurationen programmieret werden, wobei jede zum Ausführen einer verschiedenen rechenintensiven Aufgabe angepasst ist, was größere Effizienz als für einen Mehrzweckprozessor alleine ohne eine große Zunahme in den Gesamtkosten bereitstellt. In neuen FPGA-Bauteilen wird ein Rahmen für dynamische Rekonfiguration bereitgestellt, wobei teilweise oder vollständige Rekonfiguration während des Ausführens des Codes vorgesehen ist, so dass Zeit-Multiplexverarbeitung verwendet werden kann, um Konfigurationen bereitzustellen, die für verschiedene Unteraufgaben bei verschiedenen Stufen der Ausführung eines Teils des Codes optimiert sind.

[0008] FPGA-Bauteile sind nicht speziell für bestimme Arten von Rechenaufgaben geeignet. Da die einzelnen Rechenelemente sehr klein sind, sind die Datenwege extrem eng und viele von diesen werden benötigt, so dass eine große Zahl von Vorgängen bei dem Konfigurationsprozess benötigt werden. Obwohl diese Strukturen für Aufgaben relativ wirksam sind, welche auf kleinen Datenelementen arbeiten und von Zyklus zu Zyklus regelmäßig sind, sind sie für unregelmäßige Aufgaben mit großen Datenelementen weniger zufriedenstellend. Solche Aufgaben werden häufig auch nicht gut von einem Mehrzweckprozessor gehandhabt, können jedoch

von beträchtlicher Wichtigkeit (zum Beispiel bei der Bildverarbeitung) sein.

[0009] Alternative umkonfigurierbare Architekturen sind vorgeschlagen worden. Ein Beispiel ist die PADDI-Architektur, die von der University of California in Berkeley entwickelt worden ist, beschrieben in D. Chen und J. Rabaey, "A Reconfigurable Multiprocessor IC for Rapid Prototyping of Real Time Data Paths", ISSCC, Feb. 1992 und A. Yeung und J. Rabaey, "A Data-Driven Architecture for Rapid Prototyping of High Throughput DSP Algorithms", IEEE VLSI Signal Processing Workshop, Oktober 1992. Diese Architektur war auf den Bau eines Prototyps von Hochgeschwindigkeits-Echtzeit-DSP-Systemen ausgerichtet, wobei DSP-Algorithmen ein Beispiel von Berechnungen sind, die weder von herkömmlichen Prozessoren noch FPGAs gut gehandhabt werden. Die Architektur umfasst eine Vielzahl von relativ einfachen Verarbeitungsausführungseinheiten, die durch ein rekonfigurierbares Netzwerk verbunden sind. Jede Ausführungseinheit arbeitet bei 16-Bit Bandweite, hat Registerdateien für die Eingangsoperanden und hat ihren eigenen Anweisungsspeicher. Ein 53-Bit-Anweisungswort ist notwendig, um den Betrieb einer Anweisungseinheit zu spezifizieren.

[0010] Bei PADDI werden Anweisungen sowohl bei Konfiguration als auch bei Laufzeit verteilt. Zum Konfigurationszeitpunkt werden die Speicher, welche als Steuerspeicher wirken, mit einem Satz von Anweisungen geladen. Zur Laufzeit werden die Adressen für alle Steuerspeicher global gesendet und jeder dieser lokalen Anweisungsspeicher ruft seine eigene lokale Anweisung für die Benutzung von der lokalen Ausführungseinheit ab. Beim Betrieb ist die Kommunikation zwischen Verarbeitungselementen datengetrieben und die Verarbeitungselemente wirken auf die Daten gemäß ihren lokalen Anweisungen.

[0011] Eine andere alternative Architektur ist MATRIX, entwickelt am Massachussetts Institute of Technology und beschrieben in Ethan Mirsky und André deHon, "MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources", FCCM '96 – IEEE Symposium on FPGAs for Custom Computing Machines, 17.–19. April 1996, Mapa, Kalifornien, USA und ausführlicher in André deHon, "Reconfigurable Architectures for General-Purpose Computing", Seiten 257 bis 296, Technical Report 1586, MIT Artificial Intelligence Laboratory. MATRIX ist eine grobkörnige Struktur, in welcher eine Anordnung von identischen 8-Bit Funktionseinheiten mit einem konfigurierbaren Netzwerk verbunden ist. Jede Funktionseinheit enthält einen 256 × 8-Bit-Speicher, eine 8-Bit ALU mit adressierbaren Eingangsregistern, einem Ausgaberegister und einem Multiplizierer und Steuerlogik. Diese Architektur ist relativ vielseitig, da sie die Dezentralisierung der Verarbeitung eines FPGA bereitstellt, während sie einen breiteren Datenweg und den Rahmen bereitstellt, um den Anweisungsstrom auf das abstimmt, was für eine gegebene Anwendung benötigt wird.

[0012] Die MATRIX-Struktur hat vorteilhafte Aspekte, aber die grobkörnige Größe bedeutet, dass sie mehr Silizium als eine herkömmliche FPGA-Struktur verbraucht und voraussichtlich weniger wirksam für Aufgaben ist, die von Zyklus zu Zyklus regelmäßig sind. Es wäre daher wünschenswert, umkonfigurierbare Strukturen weiter zu entwickeln, welche so gut wie möglich die Vorteile von MATRIX und von herkömmlichen FPGAs kombinieren.

[0013] Eine Veröffentlichung mit dem Titel "Digit Pipeline Arithmetic on Fine-Grain Array Processors", Journal of VLSI Signal Processing, Ausgabe 9, Nummer 3, 1. April 1995, veröffentlicht von Nagendra C. et al., veröffentlicht ein Architekturmodell umfassend eine Vielzahl von Verarbeitungsbauteilen, eine Verbindungsmatrix, welche eine Verbindung zwischen den Verarbeitungsbauteilen bereitstellt, und Mittel, um die Konfiguration der Verbindungsmatrix zu definieren.

[0014] Dementsprechend stellt die Erfindung ein umkonfigurierbares Bauteil bereit, umfassend: eine Vielzahl von Verarbeitungsbauteilen; eine Verbindungsmatrix, die eine Verbindung zwischen den Verarbeitungsbauteilen bereitstellt, und ein Mittel, um die Konfiguration der Verbindungsmatrix zu definieren; wobei jedes der Verarbeitungsbauteile eine arithmetisch-logische Einheit aufweist, die dazu eingerichtet ist, eine Funktion auf Eingangsoperanden auszuführen und eine Ausgabe zu erzeugen, wobei die Eingangsoperanden als Eingabe an die arithmetisch-logische Einheit von der Verbindung in jedem Zyklus auf dem gleichen Weg bereitgestellt werden, und wobei Mittel bereitgestellt sind, um die Ausgabe eines ersten der Verarbeitungsbauteile an ein zweites der Verarbeitungsbauteile zu leiten, um die durch das zweite der Verarbeitungsbauteile ausgeführte Funktion zu bestimmen.

[0015] Im Gegensatz zu MATRIX umfasst dieser Ansatz kein adressierbares Eingangsregister (und daher keine Eingaberegisterdatei), da Eingangsoperanden von der Verbindung auf dem gleichen Weg in jedem Zyklus bereitgestellt werden. Dies erfordert, dass einzelne Verarbeitungsbauteile als ein Teil einer Verarbeitungsleitung verwendet werden (denkbarerweise kann es Anweisungen zu sich selbst zurücksenden, aber es wird dies durch die Verbindung tun müssen). Ein einzelnes Verarbeitungsbauteil in MATRIX ist daher für einen größeren

Funktionsbereich geeignet als ein einzelnes Verarbeitungsbauteil in dem umkonfigurierbaren Bauteil der Erfindung. Jedoch wird dies mehr als kompensiert durch die erhöhte Anzahl von Verarbeitungsvorrichtungen für einen gegebenen Siliziumbereich.

[0016] Der vorliegende Ansatz bezieht auch nicht den Verlust eines beträchtlichen Siliziumbereichs ein, um den für die PADDI-Architektur benötigten Steuerspeicher zu bilden: dieser Steuerspeicher muss bei PADDI von einer erheblichen Größe sein und die Ausführungseinheiten von PADDI werden von einer viel größeren Größe als jene der vorliegenden Erfindung für eine äquivalente Funktionalität sein. Der Steuerspeicher wird auch häufig in der PADDI-Architektur redundant sein (wenn die Ausführungseinheit nur benötigt wird, um dieselbe Anweisung in jedem Zyklus durchzuführen). Das Erfordernis bei PADDI, dass alle Steuerungsspeicher von einer einzigen globalen Adresse adressiert werden, hindert verschiedene Teile der Maschine daran, in daten-abhängigen Wegen sequenziert zu werden, oder auf verschiedenen Berechnungs-Threads zu operieren: bei der PADDI-Anordnung müssen alle Ausführungseinheiten synchron arbeiten.

[0017] Es sollte erwähnt werden, dass Eingangsregister nicht notwendigerweise bei Architekturen dieses Typs abwesend sind: Eingangsregister, die nicht adressierbar sind, sind mit der Erfindung vereinbar (da Eingangsoperanden weiter auf dem selben Weg in jedem Zyklus empfangen werden und die ALUs in einer Verarbeitungsleitung verwendet werden). Jedoch enthält in einem bevorzugten Ausführungsbeispiel keines der Verarbeitungsbauteile ein eigenes Register irgendeiner Art, so dass Eingangsoperanden unmittelbar von der Verbindung von der arithmetisch-logischen Einheit empfangen werden.

[0018] Die Verarbeitungsbauteile brauchen eine Konfiguration, um geeignete Funktionen durchzuführen, und mindestens irgendeine Maßnahme zur Bereitstellung von dynamischer Anweisung muss bereitgestellt sein. Eine vorteilhafte Lösung ist, dass jedes Verarbeitungsbauteil eine erste Vielzahl von Konfigurationsbits, die von der Ausgabe eines anderen der Verarbeitungsbauteile bestimmt werden kann, und eine zweite Vielzahl von Konfigurationsbits hat, die von der Ausgabe von einem anderen der Verarbeitungsbauteile bestimmt werden kann.

[0019] Bei einem bevorzugten Ausführungsbeispiel hat jedes der Verarbeitungsbauteile einen ersten Operandeneingang, einen zweiten Operandeneingang, eine Funktionsergebnisausgabe, einen Übertrags-Eingang und eine Übertrags-Ausgabe hat, wobei der erste Operandeneingang, der zweite Operandeneingang und die Funktionsergebnisausgabe n-Bit sind, wobei n eine Ganzzahl größer als 1 ist und der Übertrags-Eingang und die Übertrags-Ausgabe 1 Bit sind. Eine besonders gute Konstruktionslösung ist gefunden, wenn n gleich 4 ist.

[0020] Bei einem bevorzugten Ausführungsbeispiel ist die Vorrichtung für dynamische Anweisung, dass jedes der Verarbeitungsbauteile dazu eingerichtet ist, zum Bestimmen seiner Funktion eine n-Bitfunktionseingabe von einem anderen der Verarbeitungsbauteile zu erhalten.

[0021] Eine weitere vorteilhafte Art und Weise, um dynamische Anweisung bereitzustellen, ist durch Bereitstellung von Mitteln, um dem Übertrags-Eingang für eines der Verarbeitungsbauteile zu erlauben, die Funktion der arithmetisch-logischen Einheit dieses Verarbeitungsbauteils zu ändern (zum Beispiel, dem Übertrags-Eingang zu erlauben, die Funktion der arithmetisch-logischen Einheit zu seinem logischen Komplement zu ändern). Jedoch ist es für vielseitigen Betrieb auch vorteilhaft, dass für jedes der Verarbeitungsbauteile Mittel vorgesehen sind, um den Übertrags-Eingang auf einem konstanten Wert zu halten. Ein weiterer vorteilhafter Ansatz für ein erstes der Verarbeitungsbauteile ist, dazu verwendet zu werden, zwischen zwei Werten eines Anweisungseingangs auf ein zweites der Verarbeitungsbauteile entsprechend dem Wert des Übertragseingangs des ersten der Verarbeitungsbauteile zu multiplexen, wahlweise auch so, dass der Übertragseingang des ersten der Verarbeitungsbauteile durch das erste der Verarbeitungsbauteile zu dem Übertragseingang des zweiten der Verarbeitungsbauteile hindurchgeleitet werden kann.

[0022] Es ist auch vorteilhaft, dass jedes der Verarbeitungsbauteile ein Zwischenspeicherbasisausgaberegister für den Funktionsaufgang aufweist. Dies ist nützlich zum Konstruieren einer "tiefen" Leitung, wenn es zum Beispiel notwenig ist, eine Zahl von Vorgängen parallel durchzuführen und die Bereitstellung der Ausgabe von verschiedene ALUs zu synchronisieren.

[0023] Um einem bestimmten Bauteil zu erlauben, dynamische Anweisungen anzunehmen oder abzulehnen, ist es wünschenswert, für jedes der Verarbeitungsbauteile ein dynamisches Freigabe-Gate vorzusehen, um zu bestimmen, ob Anweisungen zum Bestimmen der Funktion der arithmetisch-logischen Einheit dynamisch angenommen werden oder vom Konfigurationsspeicher in dem Verarbeitungsbauteil bereitgestellt werden. Ein weiteres vorteilhaftes Merkmal für jedes der Verarbeitungsbauteile ist eine dynamische Anweisungsmaske,

wobei ein Anwenden der dynamischen Anweisungsmaske auf eine Anweisung, die von dem Verarbeitungsbauteil empfangen wurde, die Anweisung in die Lage versetzt, sowohl einen Anweisungseingang an die arithmetisch-logische Einheit zum Bestimmen der Funktion der arithmetisch-logischen Einheit als auch einen Peripherieschaltungs-Anweisungseingang zum Steuern der Peripherieschaltung in dem Verarbeitungsbauteil bereit zu stellen.

[0024] Spezifische Ausführungsbeispiele der Erfindung sind unten als Beispiel in Bezug auf die begleitenden Zeichnungen beschrieben, in welchen:

[0025] Fig. 1 einen Teil einer Prozessoranordnung zeigt, wobei sechs Schaltbereiche und die Orte von sechs arithmetisch-logischen Einheiten dargestellt sind;

[0026] Fig. 2 eine Darstellung eines Teils der in Fig. 1 gezeigten Anordnung auf einem größeren Maßstab zeigt, wobei einer der Schaltbereiche und einer der Orte der arithmetisch-logischen Einheiten gezeigt sind;

[0027] Fig. 3 einen Teil der in Fig. 1 gezeigten Prozessoranordnung auf einem kleineren Maßstab zeigt, wobei die Orte der arithmetisch-logischen Einheiten und "vertikalen" Busse, die sich über diese erstrecken, dargestellt sind;

[0028] Fig. 4 ähnlich zu Fig. 3 ist, aber "horizontale" Busse darstellt, die sich über die Orte der arithmetisch-logischen Einheiten erstrecken;

[0029] Fig. 5 die Verbindungen zwischen den Bussen der Fig. 2, Fig. 3 und Fig. 4 am Ort einer der arithmetisch-logischen Einheiten zeigt;

[0030] Fig. 6A ausführlich die Schaltung eines Typs von programmierbarem Schalter in den Schaltbereichen zeigt, zum Verbinden eines Paars von 4-Bit Bussen, die einander kreuzen;

[0031] Fig. 6B ausführlich die Schaltung eines anderen Typs von programmierbarem Schalter in den Schaltbereichen zeigt, zum Verbinden eines Paars von 4-Bit Bussen, die einander Ende an Ende treffen;

[0032] Fig. 6C ausführlich die Schaltung eines anderen Typs von programmierbarem Schalter in den Schaltbereichen zeigt, zum Verbinden von Übertrags-Bits Bussen;

[0033] Fig. 7 die Schaltung einer Reihe von NOR-Gates zeigt, die in den programmierbaren Schaltern der Fig. 5 und 6 verwendet werden können;

[0034] Fig. 8 zeigt eine Abänderung der Schaltung aus Fig. 7;

[0035] Fig. 9 einen Puffer und Register zeigt, die in jedem Schaltbereich verwendet werden können;

[0036] Fig. 10 eine schematische Abbildung ist, die veranschaulicht, wie Freigabesignale zu den programmierbaren Schaltern in den Schaltbereichen verteilt werden können;

[0037] Fig. 11 ausführlicher die Schaltung der in Fig. 10 gezeigten Anordnung zeigt;

[0038] Fig. 12a ein Blockdiagramm zeigt, welches eine einzelne arithmetisch-logische Einheit zur Verwendung in der Anordnung aus Fig. 1 zeigt; Fig. 12b schematisch eine Bit-Scheibe dieser einzelnen arithmetisch-logischen Einheit zeigt; Fig. 12c eine technische Ausführung der Bit-Scheibe aus Fig. 12b zeigt, alle entsprechend einem Ausführungsbeispiel der Erfindung;

[0039] Fig. 13 dynamische Anweisungsauswahl zwischen OR und AND in einer arithmetisch-logischen Einheit zeigt;

[0040] Fig. 14 dynamische Ausführungsauswahl zwischen NAND und XOR in einem Paar von arithmetisch-logischen Einheiten zeigt;

[0041] Fig. 15 dynamische Anweisungsauswahl zwischen XOR und NOR in einem Paar von arithmetisch-logischen Einheiten zeigt;

[0042] Fig. 16 eine Struktur zeigt, um Anweisungen dynamisch aus dem Schaltungsnetzwerk in eine arithmetisch-logische Einheit einzuspeisen; und

[0043] Fig. 17 einen Übertrags-Eingang/Ausgabe-Weg zeigt, der Umleitung von Bits mit einer dynamischen Anweisungsmaske wie in Fig. 16 gezeigt verwendet.

[0044] Fig. 18 ein Beispiel einer Bit-Scheibe einer sehr einfachen CPU zeigt.

[0045] In der folgenden Beschreibung sind die Ausdrücke "horizontal", "vertikal", "Nord", "Süd", "Ost" und "West" verwendet worden, um beim Verständnis der relativen Richtungen zu helfen, aber ihre Verwendung ist nicht dazu bestimmt, irgendeine Beschränkung der absoluten Ausrichtung des Ausführungsbeispiels der Erfindung zu unterstellen.

[0046] Die Prozessoranordnung für das Ausführungsbeispiel der Erfindung ist in einem integrierten Schaltkreis bereitgestellt. Auf einem Niveau wird die Prozessoranordnung von einer rechtwinkligen (und vorzugsweise quadratischen) Anordnung von "Kacheln" 10 gebildet, von denen eines in Fig. 1 von einer dicken Linie begrenzt gezeigt ist. Jede geeignete Zahl von Kacheln kann verwendet werden, zum Beispiel in einer 16 × 6, 32 × 32 oder 64 × 64 Anordnung. Jede Kachel **10** ist rechtwinklig und in vier Schaltkreisbereiche unterteilt. Es ist für diese Kacheln bevorzugt, folgerichtig quadratisch zu sein (um Verbindungssymmetrie bereitzustellen), obwohl es von geringerer Wichtigkeit ist, dass sie physikalisch quadratisch sind (dies mag einen Vorteil haben beim Bereitstellen von Symmetrie bei der Zeiteinteilung, aber dies wird im Allgemeinen wahrscheinlich weniger von Bedeutung sein). Zwei der Schaltkreisbereiche **12**, die in der Kachel **10** diagonal entgegengesetzt sind, stellen die Orte für zwei arithmetisch-logische Einheiten ("ALUs") bereit. Die anderen zwei Schaltkreisbereiche, die in der Kachel **10** diagonal gegenüberliegend sind, stellen die Orte für ein Paar von Schaltbereichen **14** bereit.

[0047] Bezugnehmend auf die Fig. 1 und Fig. 2 hat jede ALU ein erstes Paar von 4-Bit-Eingängen a, die unmittelbar innerhalb der ALU verbunden sind, ein zweites Paar von 4-Bit-Eingängen b, die auch unmittelbar mit der ALU verbunden sind, und vier 4-Bit-Ausgaben f, die unmittelbar innerhalb der ALU verbunden sind. Jede ALU hat auch ein unabhängiges Paar von 1-Bit-Übertragseingängen hci, vci und ein Paar von 1-Bit-Übertrags-Ausgaben co, die unmittelbar innerhalb der ALU verbunden sind. Die ALU kann Standardoperationen auf den Eingangssignalen a, b, hci, vci durchführen, um die Ausgabesignale f, co, wie zum Beispiel Addition, Subtraktion, AND, NAND, OR, NOR, XOR, NXOR und Multiplexen zu erzeugen, und kann wahlweise das Ergebnis der Operation aufzeichnen. Der Arbeitsablauf einer einzelnen ALU wird unten ausführlicher diskutiert. Die Anweisungen an die ALUs können von entsprechenden 4-Bit-Speicherzellen bereitgestellt werden, deren Werte über die unten beschriebene "H-Baum"-Struktur gesetzt werden kann, oder können auf dem Bus-System bereitgestellt werden, das unten beschrieben werden wird.

[0048] Auf dem in den Fig. 1 und Fig. 2 gezeigten Niveau hat jeder Schaltbereich **14** acht Busse, die sich horizontal darüber erstrecken, und acht Busse, die sich vertikal darüber erstrecken, womit eine rechtwinklige 8 × 8-Anordnung von 64 Kreuzungspunkten gebildet wird, die in Fig. 2 mit Kartesischen Koordinaten nummeriert worden sind. Alle Busse haben eine Weite von vier Bits, mit der Ausnahme des Übertragsbusses vc bei X = 4 und des Übertragsbusses hc bei Y = 3, die eine Weite von einem Bit haben. Bei vielen der Kreuzungspunkte ist ein programmierbarer 4-Gang-Schalter **16** vorgesehen, der auswählbar die zwei Busse bei dem Kreuzungspunkt verbinden kann. Bei einigen der Kreuzungspunkte ist ein programmierbarer 4-Gang-Schalter **18** vorgesehen, der auswählbar zwei Busse verbinden kann, die sich bei diesem Kreuzungspunkt Ende an Ende treffen, ohne jede Verbindung mit dem Bus in rechten Winkeln dazu. Bei dem Kreuzungspunkt bei (4, 3) ist ein programmierbarerer Schalter **20** (zum Beispiel in Fig. 6C gezeigt) vorgesehen, der auswählbar die Übertragsbusse vc, hc verbinden kann, die sich bei diesem Punkt in rechten Winkeln kreuzen.

[0049] Die horizontalen Busse in dem Schaltbereich **14** werden nun beschrieben werden.

[0050] Bei Y = 0 sind Busse h2s durch programmierbare Schalter **16** an die vertikalen Busse bei X = 0, 1, 2, 5, 6 verbindbar. Die Busse h2s haben eine Länge von zwei Kacheln und sind Ende an Ende in jedem anderen Schaltbereich **14** durch einen programmierbaren Schalter **18** bei (4, 0) verbindbar.

[0051] Bei Y = 1 ist ein Bus, der sich von einem Eingang b der ALU zu dem Westen hin erstreckt, durch Schalter **16** an die vertikalen Busse bei X = 0, 1, 2, 3 verbindbar. Auch ist ein Bus fw, der sich von einer Ausgabe f der ALU zu dem Osten hin erstreckt, durch Schalter **16** mit den vertikalen Bussen bei X = 5, 6, 7 verbindbar. Die Enden der Busse be, fw sind durch einen programmierbaren Schalter **18** bei (4, 1) verbindbar.

[0052] Bei Y = 2 ist ein Bus hregs durch programmierbare Schalter **16** mit den vertikalen Bussen bei X = 1, 2, 3, 5, 6, 7 verbindbar.

[0053] Bei Y = 3 erstreckt sich ein Bus hco von der Übertragsausgabe co der ALU zu dem Westen hin zu einem programmierbaren Schalter **20** bei (4, 3), der den Bus hco (a) mit einem Übertragsbus hci, der sich von dem Übertragseingang hci der ALU zu dem Osten hin erstreckt, oder (b) mit einem Übertragsbus vci, der sich von dem Übertragseingang vci der ALU zu dem Süden hin erstreckt, verbinden kann.

[0054] Bei Y = 4 ist ein Bus hregn durch programmierbare Schalter **16** mit den vertikalen Bussen bei X = 0, 1, 2, 3, 5, 6 verbindbar.

[0055] Bei Y = 5 sind Busse h1 mit den vertikalen Bussen bei X = 0, 1, 2, 3, 5, 6, 7 verbindbar. Die Busse h1 haben eine Länge von einer Kachel und sind Ende an Ende in jedem Schaltbereich **14** durch eine programmierbaren Schalter **18** bei (4, 5) verbindbar.

[0056] Bei Y = 6 ist ein Bus fe, der sich von einer Ausgabe f der ALU zu dem Westen erstreckt, durch Schalter **16** mit den vertikalen Bussen bei X = 0, 1, 2, 3 verbindbar. Auch ist ein Bus aw, der sich von einem Eingang a der ALU zu dem Osten hin erstreckt, durch Schalter **16** mit den vertikalen Bussen bei X = 5, 6, 7 verbindbar. Die Enden der Busse fe, aw sind durch einen programmierbaren Schalter **18** bei (4, 6) verbindbar.

[0057] Bei Y = 7 sind Busse h2n durch programmierbare Schalter **16** mit den vertikalen Bussen bei X = 1, 2, 3, 6, 7 verbindbar. Die Busse h2n haben eine Länge von zwei Kacheln und sind Ende an Ende in jedem anderen Schaltbereich **14** durch einen programmierbaren Schalter **18** bei (4, 7) verbindbar, gestaffelt in Bezug auf die programmierbaren Schalter **18**, die die Busse h2s bei (4, 0) verbinden.

[0058] Die vertikalen Busse in dem Schaltbereich **14** werden nun beschrieben werden.

[0059] Bei X = 0 sind Busse v2w durch programmierbare Schalter **16** mit den horizontalen Bussen bei Y = 0, 1, 4, 5, 6 verbindbar. Die Busse v2w haben eine Länge von zwei Kacheln und sind Ende an Ende in jedem anderen Schaltbereich **14** durch einen programmieren Schalter **18** bei (0, 3) verbindbar.

[0060] Bei X = 1 ist ein Bus fn, der sich von einer Ausgabe f der ALU zu dem Süden hin erstreckt, durch programmierbare Schalter **16** mit den horizontalen Bussen bei Y = 0, 1, 2 verbindbar. Auch ist beispielsweise ein Bus, der sich von einem Eingang b der ALU zu dem Norden hin erstreckt, durch Schalter **16** mit den horizontalen Bussen bei Y = 4, 5, 6, 7 verbindbar. Die Enden der Busse fn, bs sind durch einen programmierbaren Schalter **18** bei (1, 3) verbindbar.

[0061] Bei X = 2 sind Busse v1 mit den horizontalen Bussen bei Y = 0, 1, 2, 4, 5, 6, 7 verbindbar. Die Busse v1 haben eine Länge von einer Kachel und sind Ende an Ende in jedem Schaltbereich **14** durch einen programmierbaren Schalter **18** bei (2, 3) verbindbar.

[0062] Bei Y = 3 ist ein Bus vregw durch programmierbare Schalter **16** mit den horizontalen Bussen bei Y = 1, 2, 4, 5, 6, 7 verbindbar.

[0063] Bei X = 4 erstreckt sich ein Bus vco von der Übertragsausgabe co der ALU zu dem Norden hin zu dem programmierbaren Schalter **16** bei (4, 3), der den Bus vco (a) mit dem Übertragsbus hci, der sich von dem Übertragseingang hci der ALU zu dem Osten hin erstreckt, oder (b) mit dem Übertragsbus vci, der sich von dem Übertragseingang vci der ALU zu dem Süden erstreckt, verbinden kann.

[0064] Bei X = 5 ist ein Bus vrege durch programmierbare Schalter **16** mit den horizontalen Bussen bei Y = 0, 1, 2, 4, 5, 6 verbindbar.

[0065] Bei X = 6 ist in Bus an, der sich von einem Eingang a der ALU zu dem Süden hin erstreckt, durch Schalter **16** mit den horizontalen Bussen bei Y = 0, 1, 2 verbindbar. Auch ist ein Bus fs, der sich von einer Ausgabe f der ALU zu dem Norden hin erstreckt, durch programmierbare Schalter **16** mit den horizontalen Busen bei Y = 4, 5, 6, 7 verbindbar. Die Enden der Busse an, fs sind durch einen programmierbaren Schalter **18** bei (6, 3) verbindbar.

[0066] Bei X = 7 sind Busse v2e durch programmierbare Schalter **16** mit den horizontalen Bussen bei Y = 1, 2, 5, 6, 7 verbindbar. Die Busse v2e haben eine Länge von zwei Kacheln und sind Ende an Ende mit jedem

anderen Schaltbereich **14** durch einen programmierbaren Schalter **18** bei (7, 3) verbindbar, der in Bezug auf die programmierbaren Schalter **18**, welche die Busse v2w (0, 3) verbinden, gestaffelt ist.

[0067] Wie in Fig. 2 gezeigt, sind die Busse bs, vco, fs jeweils mit Eingang b, Ausgabe co und Ausgabe f der ALU mit dem Norden des Schaltbereichs **14** verbindbar. Auch sind die Busse fe, hco, be jeweils mit der Ausgabe f, Ausgabe co und Eingang b der ALU, in dem Westen des Schaltbereichs **14** verbindbar. Ferner sind die Busse aw, hci, fw jeweils mit dem Eingang a, Eingang ci und Ausgabe f der ALU in dem Osten des Schaltbereichs **14** verbunden. Außerdem sind die Busse fn, vci an jeweils mit der Ausgabe f, Eingang ci und Eingang a der ALU in dem Süden des Schaltbereichs **14** verbunden.

[0068] Zusätzlich zu diesen Verbindungen sind die Busse vregw, vrege über entsprechende programmierbare Schalter **18** mit 4-Bit-Verbindungspunkten vtsw, vtse jeweils (durch Kreuze in Fig. 2 dargestellt) mit dem Bereich **12** der ALU in dem Norden des Schaltbereichs **14** verbunden. Auch sind die Busse hregs, hregn über entsprechende programmierbare Schalter **18** mit 4-Bit-Verbindungspunkten htse, htne jeweils mit dem Bereich **12** der ALU in dem Westen des Schaltbereichs **14** verbunden. Ferner sind die Busse hregs, hregn über entsprechende programmierbare Schalter **18** mit 4-Bit-Verbindungspunkten htsw, htnw jeweils in dem Bereich **12** der ALU mit dem Osten des Schaltbereichs **14** verbunden. Außerdem sind die Busse vregw, vrege über entsprechende programmierbare Schalter **18** mit 4-Bit-Verbindungspunkten vtnw, vtne jeweils in dem Bereich **12** der ALU mit dem Süden des Schaltbereichs **14** verbunden. Diese Verbindungspunkte vtnw, vtne, htne, htse, vtse, vtsw, htsw, htnw werden unten ausführlicher in Bezug auf Fig. 3 bis Fig. 5 beschrieben werden.

[0069] Die Busse hregn, vrege, hregs, vregw haben auch, wie in Fig. 2 gezeigt, entsprechende 4-Bit-Verbindungspunkte **22** (durch kleine Quadrate in Fig. 2 gezeigt), die unten ausführlicher in Bezug auf Fig. 9 beschrieben werden.

[0070] Fig. 3 zeigt ein Niveau der Verbindungen zwischen den Orten der arithmetisch-logischen Einheiten, die von Quadraten mit abgerundeten Ecken dargestellt sind. Eine Gruppe von vier 4-Bit-Bussen v8, v4w, v4e, v16 erstrecken sich vertikal über jede Spalte der ALU Orte **12**. Der Bus v8 ganz links in jeder Gruppe ist in Segmenten, von denen jedes eine Länge von im Allgemeinen acht Kacheln hat. Der Bus v4w vor dem ganz links in jeder Gruppe ist in Segmenten, von denen jedes eine Länge von den allgemeinen vier Kacheln hat. Der Bus v2e vor dem ganz rechts in jeder Gruppe ist in Segmenten, von denen jedes wiederum eine Länge von im Allgemeinen vier Kacheln hat, aber um zwei Kacheln von dem Bus v4w vor dem ganz links versetzt ist. Der Bus v16 ganz rechts in jeder Gruppe ist in Segmenten, von denen jedes eine Länge von im Allgemeinen sechzehn Kacheln hat. An der Oberkante der Anordnung, welche oben in Fig. 4 ist, und an der Unterkante können die Längen der Segmente etwas größer als oder kürzer als oben angegeben sein.

[0071] Bezugnehmend auf Fig. 3 und Fig. 5, wo jede Gruppe von vier Bussen v8, v4w, v4e, v16 jeden ALU-Ort **12** kreuzt, werden vier 4-Bit-Verzweigungsverbindungen bei den Verbindungspunkten htnw, htsw, htse, htne gemacht. Die Enden der Bussegmente haben Vorrang, so verbunden zu werden, vor einer Verbindung mit einem Bussegment, welches den ALU-Ort kreuzt.

[0072] In ähnlicher Weise, wie in Fig. 4 und Fig. 5 gezeigt, erstreckt sich eine Gruppe von vier 4-Bit-Bussen h8, h4n, h4s, h16 horizontal über jede Reihe von ALU-Orten **12**. Der oberste Bus h8 in jeder Gruppe ist in Segmenten, von denen jedes eine Länge von im Allgemeinen acht Kacheln hat. Der Bus h4n vor dem obersten in jeder Gruppe ist in Segmenten, von denen jedes eine Länge von im Allgemeinen vier Kacheln hat. Der Bus h4s vor dem untersten in jeder Gruppe ist in Segmenten, von denen jedes wiederum eine Länge von im Allgemeinen vier Kacheln hat, aber um zwei Kacheln von dem Bus h4n vor dem obersten versetzt ist. Der unterste Bus h16 in jeder Gruppe ist in Segmenten, von denen jedes eine Länge von im Allgemeinen sechzehn Kacheln hat. Bei der linken Kante der Anordnung, die auf der linken Seite der Fig. 4 ist, und bei der rechten Kante können die Längen der Segmente leicht größer als oder kürzer als oben spezifiziert sein. Wo jede Gruppe von Bussen h8, h4n, h4s, h16 jeden ALU-Ort **12** kreuzt, sind weitere vier 4-Bit-Verzweigungsverbindungen bei den Verbindungspunkten vtnw, vtsw, vtse, vtne gemacht. Die Enden der Bussegmente haben Vorrang, so verbunden zu werden, vor einer Verbindung mit einem Bussegment, welches den ALU-Ort kreuzt.

[0073] Wie in Fig. 5 gezeigt, sind die Verbindungspunkte htnw, htsw, htne, htse über programmierbare Schalter mit den Bussen hrgen, hregs der Schaltbereiche zu dem Westen und dem Osten des ALU-Orts hin verbunden. Auch sind die Verbindungspunkte vtnw, vtne, vtsw, vtse über programmierbare Schalter mit den Bussen vregw, vrege der Schaltbereiche zu dem Norden und dem Süden des ALU-Orts hin verbunden.

[0074] Die programmierbaren Verbindungen **16** zwischen Paaren von 4-Bit-Bussen, die in rechten Winkeln

kreuzen, werden nun in Bezug auf Fig. 6A beschrieben werden. Die Leiter der horizontalen Busse werden als x0, x1, x2, x3 beschrieben und die Leiter der vertikalen Busse werden als y0, y1, y2, y3 beschrieben. Zwischen jedem Paar von Leitern des selben Bit-Stellenwerts ist ein entsprechender Transistor **160**, **161**, **162**, **163** bereitgestellt. Die Gates der Transistoren **160**, **161**, **162**, **163** werden im Allgemeinen mit der Ausgabe eines NOR-Gates **16g** verbunden, welches an seinen zwei Eingängen ein invertiertes FREIGABE-Signal von einer Einzelbit-Speicherzelle, die von einer Gruppe von den Schaltern geteilt werden kann, und der invertierte Inhalt einer Einzelbit-Speicherzelle **24** empfängt. Dementsprechend werden die Leiter x0, x1, x2, x3, nur wenn das FREIGAGE-Signal hoch ist und der Inhalt der Speicherzelle **24** hoch ist, von den Transistoren **160**, **161**, **162**, **163** jeweils mit den Leitern y0, y1, y2, y3 verbunden.

[0075] Die programmierbaren Verbindungen **18** zwischen Paaren von 4-Bit-Bussen, welche einander Ende an Ende in Reihe treffen, werden nun in Bezug auf Fig. 6B beschrieben werden. Die Lieter eines Busses werden als x10, x11, x12, x13 bezeichnet, und die Leiter des anderen Busses werden als x20, x21, x22, x23 bezeichnet. Zwischen jedem Paar von Leitern der selben Bit-Signifikanz ist ein entsprechender Transistor **180**, **181**, **182**, **183** vorgesehen. Die Gates der Transistoren **180**, **181**, **182**, **183** werden im Allgemeinen mit der Ausgabe eines NOR-Gates **18g** verbunden, welches an seinen zwei Eingängen ein invertiertes FREIGABE-Signal von einer Einzelbitspeicherzelle, welche von einer Gruppe von Schaltern geteilt werden kann, und den invertierten Inhalt einer Einzelbit-Speicherzelle **24** empfängt. Dementsprechend werden, nur wenn das FREIGABE-Signal hoch ist und der Inhalt der Speicherzelle **24** hoch ist, die Leiter x10, x11, x12, x13 von den Transistoren **180**, **181**, **182**, **183** jeweils mit den Leitern x20, x21, x22, x23 verbunden.

[0076] Die programmierbaren Verbindungen **20** zwischen den Übertragsleitern hco, vco, hci, vci werden nun in Bezug auf Fig. 6C beschrieben werden. Der horizontale Übertragsausgabeleiter hco ist jeweils mit dem horizontalen Übertragseingangsleiter hci und dem vertikalen Übertragseingangsleiter vci über Transistoren **20hh**, **20hv** verbunden. Ferner ist der vertikale Übertragsausgabeleiter vco jeweils mit dem vertikalen Übertragseingangsleiter vci und dem horizontalen Übertragseingangsleiter hci über Transistoren **20vv**, **20vh** verbunden. Die Gates der Transistoren **20hh**, **20vv** sind im Allgemeinen mit der Ausgabe eines Inverters **20i** verbunden und die Gates der Transistoren **20hv**, **20vh** und der Eingang zu dem Inverter **20i** sind mit der Ausgabe eines NOR-Gates **20g** verbunden. Das NOR-Gate **20g** empfängt an seinen zwei Eingängen ein invertiertes FREIGABE-Signal von einer Einzelbit-Speicherzelle, die von einer Gruppe von Schaltern geteilt werden kann, und den invertierten Inhalt einer Einzeitspeicherzelle **24**. Dementsprechend, wenn das FREIGABE-Signal hoch ist, sind die Leiter hco, vco jeweils mit den Leitern hci, vci verbunden oder jeweils mit den Leitern vci, hci in Abhängigkeit von dem Inhalt der Speicherzelle **24**.

[0077] Es wird bemerkt werden, dass jede der schaltbaren Verbindungen **16**, **18**, **20**, die in Bezug auf Fig. 6A bis Fig. 6C beschrieben sind, ein NOR-Gate **16g**, **18g**, **20g** einschließen. Wie in Fig. 7 gezeigt, wird ein NOR-Gate **16g** üblicherweise von vier Transistoren **16g1**, **16g2**, **16g3**, **16g4** gebildet, von denen zwei **16g1**, **16g3** auf das invertierte FREIGABE-Singal ansprechend sind und zwei **16g2**, **16g4** auf den invertierten Inhalt der Speicherzelle **24** ansprechend sind. Es ist wünschenswert, dass eine Gruppe von den umschaltbaren Verbindungen **16**, **18**, **20** gemeinsam ausgeschaltet werden kann, ohne jegliches Erfordernis, dass nur ein Teil einer solchen Gruppe ausgeschaltet werden muss. Solch eine Gruppe kann aus allen der umschaltbaren Verbindungen in einem Schaltbereich **14** bestehen, aus allen der umschaltbaren Verbindungen in den zwei Umschaltbereichen **14** in einer bestimmten Kachel oder aus allen der umschaltbaren Verbindungen in einem größeren Bereich der Anordnung bestehen. In diesem Fall kann der Transistor **16g1** für alle der umschaltbaren Verbindungen **16**, **18**, **20** in der Gruppe wie in Fig. 8 gezeigt gemeinsam gemacht werden. Dies ermöglicht eine Einsparung von 25% weniger eins in der Anzahl von für das Gate benötigten Transistoren, aber benötigt einen weiteren Leiter, der das Gate, wie in Fig. 8 gezeigt, verbindet.

[0078] Weitere Vereinfachung ist möglich, obwohl hier nicht gezeigt, indem die Eigenschaften der Speicherzelle **24** benutzt werden. Sowohl der Inhalt als auch das Komplement des Inhalts dieser Speicherzelle **24** sind als Ausgaben leicht erhältlich. Es kann daher gesehen werden, dass die Schaltung der Fig. 8 zum Beispiel in den Verbindungskosten gesenkt werden kann, indem diese Eigenschaft der Speicherzellen **24** verwendet wird, um die Erfordernis auszuschließen, sowohl das tatsächliche als auch das invertierte Freigabe-Signal zu übertragen, da bei der Benutzung von entweder dem tatsächlichen oder dem komplementären Wert der Speicherzellen nur ein Freigabesignal benötigt wird.

[0079] Wie oben in Bezug auf Fig. 1 und Fig. 2 erwähnt, werden bei jedem Schaltbereich **14** die Busse hregn, hregs, vregw, vrege von entsprechenden 4-Bit-Verbindungen **22** mit einem Register oder eine Pufferschaltung verbunden, und diese Schaltung wird nun ausführlicher in Bezug auf Fig. 9 beschrieben werden. Die vier Verbindungen **22** werden jeweils mit entsprechenden Eingängen eines Multiplexers **26** verbunden. Der Multiplexer

26 wählt einen der Eingänge als eine Ausgabe aus, die an ein Register oder einen Puffer **28** gespeist wird. Die Ausgabe der Register oder Puffer **28** wird an vier Puffer **30s**, **30w**, **30n**, **30e** mit drei Ausgaben gespeist, die mit den Verbindungen **22** jeweils zu den Bussen hregs, vregw, hregn, vrege zurück verbunden sind. In dem Fall, wo ein Puffer **28** verwendet wird, wird das 4-Bit-Signal auf einen ausgewählten der Busse hregs, vregw, hregn, vrege verstärkt und zu einem anderen ausgewählten der Busse hregs, vregw, hregn, vrege gespeist. In dem Fall, wo ein Register **28** verwendet wird, wird das 4-Bit-Signal auf einem ausgewählten der Busse hregs, vregw, hregn, vrege verstärkt und an einen ausgewählten der Busse hregs, vregw, hregn, vrege nach der nächsten aktiven Taktkante gespeist.

[0080] Verwendung einer erweiterten Form dieser Struktur aus Fig. 9 macht es möglich, ein 4-Bit-Signal auf einem ausgewählten der Busse hregs, vregw, hregn und vrege für einen verschiedenen Zweck aus dem Interbus-Routing herauszuziehen. Geeignete Konstruktion und Verbindung eines Multiplexers **26** (oder, bei alternativen Anordnungen, des Puffers **28**) erlaubt die Auswahl eines Werts, der von dem Vernetzungsnetzwerk als die Ausgabe des Multiplexers **26** oder Puffers **28** (diese Wahl ist jeweils als **260** und **280** in Fig. 9 angezeigt) empfangen wird, wobei dieser Wert dann dafür verwendet wird, die Anweisung der ALU, die mit der Schaltbox verknüpft ist, auszuwählen. Die Anwendungen dieser Anordnung werden weiter unten diskutiert werden.

[0081] Verwendung des Multiplexers **26** oder Puffers **28** zu diesem Zweck bedeutet, dass der Wert, der zum Bereitstellen von Anweisung für die ALU verwendet wird, auch der Wert ist, der bereitgestellt ist zum Weitergeben durch das Vernetzungsnetzwerk. Ein verschiedener Schaltbereich **14** muss verwendet werden, wenn es gewünscht ist, einen verschiedenen Wert zwischen den Drähten weiter zu geben. Jedoch wird es bei vielen Anordnungen wünschenswert sein, dass der zu der ALU weitergegebene Wert, um seine Anweisung zu bestimmen, auch der Draht ist, der von einem Draht zu einem anderen weitergegeben wird: dies ist geeignet, wenn es gewünscht ist, die selbe Anweisung einer Zahl von ALUs bereitzustellen, was häufig in einer tiefen Verarbeitungsleitung auftreten wird. Ein alternatives, nicht gezeigtes, Ausführungsbeispiel verwendet zwei oder mehr Paare von Multiplexern **26** und Puffern **28**: in diesem Fall kann ein Multiplexer/Puffer-Paar dazu ausgelegt sein, Anweisungseingaben für die verknüpfte ALU bereitzustellen, wobei das andere Paar oder Paare für das Routing verwendet werden können.

[0082] Es wird geschätzt werden, dass die oben beschriebene Anordnung eine größere Flexibilität bei dem Routing von Signalen rund um und über die Anordnung bereitstellt. Mit einer geeigneten Einstellung der Schalter **16**, **18**, **20** unter Verwendung der Speicherzellen **24** und mit einer geeigneten Einstellung der Multiplexer **26** und Register oder Puffer **28** können Signale über große Entfernungen, in erster Linie unter Verwendung der Busse v16, h16, v8, h8, v4e, v4w, h4n, h4s von der Kante der Anordnung zu einem bestimmten ALU, zwischen ALUs und von einem bestimmten ALU zu der Kante der Anordnung gesandt werden. Diese Busse können in Reihe oder in rechten Winkeln von den Schaltbereichen **14** mit Verstärkung durch die Register oder Puffer **28**, um Fortpflanzungsverzögerungen zu verringern, und mit von Registern **28** eingeführten Leistungsstufen zusammengefügt werden. Auch können diese Busse auf halbem Weg entlang ihrer Längen angezapft werden, so dass die Einstellung der ALUs, um einen bestimmen Verarbeitungsvorgang durchzuführen, nicht vollständig von den Längen der Busse vorgeschrieben ist, und so dass Signale zu mehr als einer ALU verteilt werden können. Ferner können die in Bezug auf die Fig. 1 und Fig. 2 beschriebenen Busse kürzerer Länge verwendet werden, um Signale zwischen den Schaltbereichen **14** und den ALUs weiter zu leiten und um Signale in erster Linie über kürzere Entfernungen zu senden, zum Beispiel von einer ALU zu einer benachbarten ALU in der selben Reihe oder Spalte oder diagonal benachbart, auch wenn sich die Busse horizontal oder vertikal erstrecken. Die Register oder Puffer **28** können wiederum verwendet werden, um die Signale zu verstärken oder programmierbare Verzögerungen in diese einzuführen.

[0083] In der oben beschriebenen Anordnung sind die Speicherzellen **24** über die Anordnung in dem selben Ausmaß wie die Schaltbereiche **14** und die ALU-Orte **12** verteilt. Jede Speicherzelle **24** ist benachbart zu dem Schalter oder den Schaltern, Multiplexern, Register oder Puffer, welchen es steuert, angeordnet. Dies ermöglicht das Erreichen einer hohen Schaltkreisdichte.

[0084] Es wird nun eine Beschreibung der Weise gemacht werden, in welcher Daten geschrieben oder von den Speicherzellen **24** gelesen werden, der Art und Weise, in welcher die FREIGABE-Signale für die programmierbaren Schalter **16**, **18**, **20** auf die Speicherzellen geschrieben werden, der Art und Weise, in welcher Anweisungen und möglicherweise Konstanten zu den ALUs verteilt werden, und der Art und Weise, in welcher andere Steuersignale, wie zum Beispiel ein Taktsignal, über die Anordnung übertragen wird. Für alle diese Funktionen kann eine "H-Baum"-Struktur (die an sich bekannt ist) verwendet werden, wie in Fig. 10 gezeigt. Bezugnehmend auf Fig. 10 und Fig. 11 werden, um ein FREIGABE-Signal zu einem der 64 Orte in dem gezeigten Beispiel zu verteilen, das FREIGABE-Signal **30a** und eine 6-Bit-Adresse **32a** dafür einem Decoder **34a**

bereitgestellt. Der Decoder **34a** bestimmt, welcher der vier Äste von ihm zu der Adresse führt, und stellt ein FREIGABE-Signal **30b** einem weiteren Decoder **34b** in diesem Ast zur Verfügung, zusammen mit einer 4-Bit-Adresse **32b** für die Decoder **34b** in allen vier Ästen. Der Decoder **34b**, welcher das FREIGABE-Signal **30b** empfängt, bestimmt, welcher der vier Äste davon zu der benötigten Adresse führt, und stellt ein FREIGABE-Signal **30c** einem weiteren Decoder **34c** in diesem Ast zur Verfügung, zusammen mit einer 4-Bit-Adresse **32c** für die Decoder **34c** in allen vier Ästen. Der Decoder **34c**, der das FREIGABE-Signal **30c** empfängt, stellt dann das FREIGABE-Signal **34d** der benötigten Adresse zur Verfügung, wo es in einer Einzelbitspeicherzelle gespeichert werden kann. Ein Vorteil der H-Baum-Struktur ist, dass die Längen der Signalwege zu allen Zielen ungefähr gleich sind, was in dem Fall des Taktsignals besonders vorteilhaft ist.

[0085] Ein großer Vorteil der oben geschriebenen Anordnung ist, dass Gruppen der Speicherzellen **24** in zum Beispiel einem Schaltbereich **14** oder in den zwei Schaltbereichen in einer Kachel oder in den Schaltbereichen in einer Unter-Anordnung der Kacheln en bloc durch die invertierten FREIGABE-Signale ausgeschaltet werden können, so dass die Inhalte dieser Speicherzellen nicht die verknüpften Schalter beeinträchtigen. Es ist dann möglich, dass diese Speicherzellen **24** als "Benutzer"-Speicher von einer Anwendung benutzt werden können, anstatt zum Konfigurieren der Schaltung der Anordnung verwendet zu werden.

[0086] Die Struktur der bei diesem Ausführungsbeispiel der Erfindung verwendeten ALU wird nun in Bezug auf **Fig.** 12 beschrieben werden. Wie in **Fig. 12a** gezeigt, hat die ALU vier Eingänge, A, B, I und C_{in}, und zwei Ausgaben F und C_{out}. A, B, I und F sind alle vier Bit weit und sind mit der allgemeinen Verbindung durch die benachbarten Schaltblöcke verbunden, wie es für A, B und F oben beschrieben worden ist. Der Eingang für I ist aus dem in **Fig. 9** gezeigten Multiplexer **26** extrahiert. C_{in} und C_{out} sind beide 1 Bit weit und sind mit einer eingeschränkteren Verbindung verbunden, wie auch oben beschrieben. A und B stellen die Operanden für die ALU bereit, und F die Ausgabe. C_{in} und C_{out} stellen die Übertragsfunktionen bereit, aber sind auch bei der Steuerung von Bedeutung. I stellt einen Anweisungseingang bereit, welcher den funktionalen Betrieb der ALU bestimmt: dies ist im Gegensatz zu einer Standard-FPGA, in welcher Funktionseinheiten von einem Satz von Speicherbits gesteuert werden. Die Bedeutung dieses Merkmals und die für Routing-Anweisungseingänge von dem Schaltnetzwerk zu der ALU bereitgestellten Verfahren werden weiter unten diskutiert.

[0087] Die ALU hat vier Hauptkomponenten:
den ALU-Datenweg, der aus vier identischen Bit-Scheiben besteht;
dem Anweisungsdekoder;
die Übertrags/Steuereingangskonditionierungslogik; und
die Schaltblockprogrammierungsschnittstelle (bei anderen Ausführungsbeispielen der Erfindung muss dies nicht in der ALU selbst vorliegend sein, jedoch erlaubt die Anwesenheit dieses Merkmals innerhalb der ALU die Möglichkeit, die ALU in einem Suchtabellenmodus zu benutzen).

[0088] **Fig. 12b** zeigt ein Blockdiagramm einer einzelnen Bit-Scheibe der ALU.

[0089] Die zwei "Eingangspuffer" **202** und **203** sind nicht mehr als ein Mittel zum Bereitstellen einer elektrischen Verbindung zu dem Routing-Netzwerk. Es gibt kein adressierbares Eingangsregister (und daher keine Registerdatei) in dieser Architektur: die Operanden werden der Funktionseinheit **201** der ALU von dem selben Ort (dem Vernetzungsnetzwerk) in jedem Zyklus bereitgestellt.

[0090] Die Funktionseinheit **201** betreibt eine Suchtabelle (LUT; engl.: look up table), die eine boolesche Funktion U der zwei Eingänge A und B erzeugt. Die genaue Funktion wird von vier Steuersignalen (L_3, L_2, L_1, L_0) gesetzt und erzeugt die in Tabelle 1 gezeigte Karnaugh-Karte:

U =

B \ A	0	1
0	L_0	L_1
1	L_2	L_3

Tabelle 1: Karnaugh-Karte für ALU-Bit-Schreibe

[0091] Die Erzeugung der Steuersignale L_i wird weiter unten diskutiert.

[0092] Das Erzeugen der Summe **204** stellt eine Summenausgabe bereit, die von einem XOR der U und C_{in} abgeleitet ist:
Summe = U XOR C_{in}

[0093] C_{out} wird von der Erzeugung des Übertrags **205** gemäß den folgenden Booleschen Gleichungen erzeugt:
P = U OR L_4
G = A OR L_5
C_{out} = wenn P dann C_{in} sonst G
wobei P als eine Fortpflanzungsfunktion und G als eine Erzeugungsfunktion betrachtet werden kann. Die Signale L_i werden wiederum in einer weiter unten beschriebenen Art und Weise erzeugt.

[0094] Das Ausgaberegister **206** verriegelt wahlweise die Summenausgabe, wobei diese Option unter der Steuerung des ALU-Programmierspeichers wählbar ist. Wahlweise kann eine ähnliche Verriegelungsanordnung für die Übertragsausgabe bereitgestellt werden. Diese Merkmale sind für die Benutzung in tiefen Leitungen vorteilhaft, wo der selbe Vorgang synchron oder in einer zeitgesteuerten Art und Weise in mehreren ALUs durchgeführt werden muss.

[0095] Eine große Vielfalt von verschiedenen möglichen Bit-Scheiben kann verwendet werden. Die Wahl des Bit-Scheiben-Typs, die in einer gegebenen Architektur ausgewählt wird, kann eine Funktion des Typs der Anweisung sein, welche von der Architektur als ein Ganzes vorgesehen ist, am wirksamsten zu verarbeiten. Es ist offensichtlich wünschenswert, die Verwendung einer Vielzahl von Funktionen zu ermöglichen, die als nützliche Bausteine für komplexere Operationen wirken können. Andere Merkmale sind auch wünschenswert. Ein wünschenswertes Merkmal ist die Eigenschaft, einige Bits aus ihrer normalen Funktion "umzuleiten", um die Steuerung über andere Schaltungselemente zu erlauben. Ein anderes wünschenswertes Merkmal ist die Fähigkeit, eine feste Anweisung für irgendwelche ALUs zu speichern, welche in einer bestimmten Konfiguration kein dynamisches Anweisungsschalten benötigen. Es ist auch wünschenswert, dass es einen geeigneten Vorgabezustand gibt, um der ALU zu erlauben, als ein Lese/Schreib-Anschluss für die Schaltzelle (oder Suchtabelle) verwendet zu werden.

[0096] Fig. 12c zeigt ein Blockdiagramm einer physikalischen Umsetzung einer einzelnen Bit-Scheibe der ALU. Die Eingänge und Ausgaben sind in Bezug auf Fig. 12b oben beschrieben worden. Die Bit-Scheibe erfordert, dass sechs Steuersignale erzeugt werden: der Mechanismus dafür wird weiter unten diskutiert. Ein nützlicher Satz von Logikfunktionen, der von dieser Bit-Scheibe erzeugt wird, ist unten in Tabelle 2 gezeigt:

L_0	L_1	L_2	L_3	L_4	L_5	Übertragswert	
						0	1
1	1	1	0	1	1	A NAND B	A AND B
0	1	1	1	1	1	A OR B	A NOR B
0	1	1	0	1	1	A XOR B	A NXOR B
0	1	0	0	1	1	$\overline{A}$ AND B	$\overline{A}$ OR B
0	0	1	0	1	0	A AND $\overline{B}$	A OR $\overline{B}$
1	1	0	0	1	1	NOT B	B
0	0	1	1	1	0	B	NOT B
1	0	1	0	1	0	NOT A	A
0	1	0	1	1	1	A	NOT A
0	1	1	0	0	0	ADD	
1	0	0	1	0	0	SUB	
1	0	0	1	0	1	A EQUALS B	
1	1	1	0	0	0	MATCH1	
1	1	1	0	0	0	MATCH0	

Tabelle 2: Funktionen für Bit-Scheiben mit verknüpften Steuereingängen

[0097] Die Eingänge fallen in die folgenden Gruppen: arithmetische Anweisungen (ADD, SUB), Zweieingangsbitweise-Anweisungen (AND, OR, NOR, XOR, NXOR), Anweisungen mit einem Eingang (A, B, NOT A, NOT B) und Vergleichs- und Testanweisungen (EQUALS, MATCH1, MATCH0). Die Ausgaben dieser Funktionen sind in Tabelle 3 unten zusammengefasst.

Name	Summenfunktion	C_{aus} Funktion
ADD	A plus B	Arithmetischer Übertrag
SUBA	A minus B	Arithmetischer Übertrag
A AND B	$= A_i$ AND B_i	$C_{out} = C_{in}$
A OR B	$= A_i$ OR B_i	$C_{out} = C_{in}$
A NOR B	= NOT (A_i OR B_i)	$C_{out} = C_{in}$
A XOR B	$= A_i$ XOR B_i	$C_{out} = C_{in}$
A NXOR B	= NOT (A_i XOR B_i)	$C_{out} = C_{in}$
A AND $\bar{B}$	= A_i AND (NOT B_i)	$C_{out} = C_{in}$
B AND $\bar{A}$	= (NOT A_i) AND B_i	$C_{out} = C_{in}$
$\bar{A}$ OR B	= (NOT A_i) OR B_i	$C_{out} = C_{in}$
$\bar{B}$ OR A	= A_i OR (NOT B_i)	$C_{out} = C_{in}$
A	$= A_i$	$C_{out} = C_{in}$
B	$= B_i$	$C_{out} = C_{in}$
NOT A	= NOT A_i	$C_{out} = C_{in}$
NOT B	= NOT B_i	$C_{out} = C_{in}$
A EQUALS B	Nicht anwendbar	wenn A = = B dann 0, sonst 1
MATCH1	Nicht anwendbar	bitweises AND von A und B, gefolgt von OR über die Weite des Worts
MATCH0	Nicht anwendbar	bitweises OR von A und B, gefolgt von einem AND über die Weite des Worts

Tabelle 3: Ausgaben für Anweisungen

[0098] 2s-Komplement-Arithmetik wird verwendet und der arithmetische Übertrag wird so bereitgestellt, dass er mit dieser Arithmetik im Einklang ist. Die MATCH-Funktionen werden so genannt, da für MATCH1 der Wert von 1 nur zurückgegeben wird, wenn es mindestens eine Position gibt, die sowohl in A als auch in B 1 ist, während für MATCHO der Wert von 0 nur dann zurückgegeben wird, wenn es mindestens eine Position gibt, die sowohl in A als auch in B 0 ist.

[0099] Sechs Steuersignale L_i werden benötigt, um die Ausgaben der Bit-Scheibe zu bestimmen. Jedoch ist es wünschenswert, dass jede dynamische Anweisung, um die Funktion der ALU zu bestimmen, nicht mehr als 4 Bits enthält, so dass sie auf die 4-Bit-Wege der Verbindung passt und so dass eine ALU-Anweisung als die Ausgabe einer andere ALU bereitgestellt werden kann. Ein Schema zum Ableiten von Steuersignalen L_i von vier Anweisungsbits J_i ist wie folgt:

$L_0 = J_3$

$L_1 = (J_1 \text{ OR } \overline{J}_0) \text{ AND } (J_3 \text{ NAND } J_2)$

$L_2 = (\overline{J}_1 \text{ OR } J_0) \text{ AND } (J_3 \text{ NAND } J_2)$

$L_3 = J_2$

$L_4 = (\overline{J}_1 \text{ OR } \overline{J}_0) \text{ AND } (J_3 \text{ NAND } J_2)$

$L_5 = \overline{J}_0$

[0100] Jedoch hat dies einen Nachteil darin, dass es keinen Multiplexer bereitstellt, der nur von C_{in} gesteuert wird. Um dies zu erreichen, können die Definitionen von L_0 und L_3 geändert werden, um einige Abhängigkeit von C_{in} zu geben. Die anderen Definitionen sind unverändert.

$L_0 = \text{IF } (C_{in} \text{ AND } L_4) \text{ THEN } J_2 \text{ ELSE } J_3$

$L_3 = \text{IF } (C_{in} \text{ AND } L_4) \text{ THEN } J_3 \text{ ELSE } J_2$

[0101] Die entstehende Anweisungstabelle ist unten als Tabelle 4 gezeigt.

J_3	J_2	J_1	J_0	L_0	L_1	L_2	L_3	L_4	L_5	Übertragswert	
										0	1
0	0	0	0	0	1	1	0	1	1	XOR	NXOR
0	0	0	1	0	0	1	0	1	0	$\overline{A}$ AND B	A OR $\overline{B}$
0	0	1	0	0	1	0	0	1	1	A AND $\overline{B}$	$\overline{A}$ OR B
0	0	1	1	0	1	1	0	0	0	ADD	
0	1	0	0	C_{in}	1	1	$\overline{C}_{in}$	1	1	A OR B	A AND B
0	1	0	1		0	1		1	0	B	A
0	1	1	0		1	0		1	1	A	B
0	1	1	1	0	1	1	1	0	0	MATCH0	
1	0	0	0	$\overline{C}_{in}$	1	1	C_{in}	1	1	A NAND B	A NOR B
1	0	0	1		0	1		1	0	NOT A	NOT B
1	0	1	0		1	0		1	1	NOT B	NOT A
1	0	1	1	1	1	1	0	0	0	MATCH1	
1	1	0	0	1	0	0	1	0	1		
1	1	0	1	1	0	0	1	0	0		
1	1	1	0	1	0	0	1	0	1	A EQUALS B	
1	1	1	1	1	0	0	1	0	0	SUB	

Tabelle 4: Anweisungsbits und entsprechende Funktionen

[0102] Die Herkunft der Anweisungsbits für die ALU wird nun diskutiert werden. Ein Element der vorliegenden Erfindung, zumindest bei einem Aspekt davon, welcher mit der MATRIX-Architektur geteilt wird, ist die Fähig-

keit, eine Anweisung für eine Funktionseinheit als die Ausgabe einer anderen Funktionseinheit zu erzeugen. In der Matrix-Architektur wird dies erreicht durch eine Struktur mit relativ grobkörnigen Funktionseinheiten, wobei jede eine 8-Bit-ALU und Eingangsregister mit ihren eigenen Registerdateien umfasst. Bei dem vorliegenden Ausführungsbeispiel werden sehr viel feinkörnigere Funktionseinheiten umfassend 4-Bit-ALUs und ohne adressierbare Eingangsregister verwendet. Der Mangel an Eingangsregistern erfordert, dass die Funktionseinheiten in einer Verarbeitungsleitung operieren. Eine Schaltung, die ermöglicht, Anweisungen an die ALU aus dem Vernetzungsnetzwerk in solch ein Leitungsrechenmodell einzugeben, ist in Fig. 16 gezeigt.

[0103] Eingangssignale umfassend die dynamische Anweisungen I (4-Bit-Anweisungen, die von einem anderen ALU in der Anordnung erzeugt werden oder wahlweise von einem dem Vernetzungsnetzwerk zugänglichen Speicher erhalten werden) werden von Verbindungen **301** zu dem Vernetzungsnetzwerk empfangen: Diese können durch den Multiplexer **26** (siehe Fig. 9) wie oben angezeigt erhalten werden. Wenn es gewünscht ist, dass mehrere Wahlmöglichkeiten erhältlich sind, kann dies durch Benutzung eines oder mehrerer zusätzlicher ALUs in Multiplexer-Konfigurationen erreicht werden.

[0104] Bei anderen Ausführungsbeispielen der Architektur können alle Anweisungen dynamisch bereitgestellt werden, in welchem Fall das Signal unmittelbar in die ALU fließen kann. Jedoch gibt es bei dem gezeigten Ausführungsbeispiel eine Option darüber, ob Anweisungen dynamisch oder lokal bereitgestellt werden. Die von dem Vernetzungsnetzwerk eingegebenen Signale werden durch ein dynamisches Anweisungsfreigabegate **304** geleitet. Die Funktion dieses Gates ist, entweder dynamische Anweisungsbits I, die von dieser ALU zu verwenden sind, zu ermöglichen oder diese daran zu hindern, benutzt zu werden, in welchem Fall gespeicherte Anweisungsbits stattdessen verwendet werden müssen. Dies wird durch ein Einzelbit **303** des Konfigurations-RAM für diese ALU unterstützt. Wenn dynamische Anweisungsbits zu verwenden sind, leitet das Gate **304** die Werte des I-Eingangs von dem Vernetzungsnetzwerk weiter. Wenn nicht, wird die Ausgabe des Gates **304** Null sein und der Anweisungseingang an die ALU wird der sein, der bereits als gespeicherte Anweisungsbits in einem 4-Bit-Steuerregister **313** gespeichert worden ist. Solche gespeicherten Anweisungsbits könnten zum Beispiel früher durch das H-Baum-Netzwerk geladen worden sein.

[0105] Vorteile können erhalten werden, wenn der Mechanismus zum Bereitstellen von Eingang an die ALU als dynamische Anweisungsbits I von dem Vernetzungsnetzwerk **301** oder als gespeicherte Anweisungsbits vom Register **313** in der Form einer dynamischen Anweisungsmaske **305** bereitgestellt wird. Dies schließt zwei Gates ein, ein OR-Gate **311** und ein AND-Gate **312**. Die Eingänge an jedem Gate sind die selben – die Ausgabe von dynamischer Anweisung schaltet die Gates **304** und das Register **313** frei.

[0106] Die Ausgabe des OR-Gates **311** ist das relevante Anweisungsbit J_i. Die Ausgabe des AND-Gates **312** ist als Ausgaben K_i zum Bereitstellen für andere Teile der ALU erhältlich, aus Gründen, die weiter unten diskutiert werden. Wenn das Eingangsfreigabebit **303** niedrig ist, sind alle Ausgaben K_i niedrig und die Anweisungsbits J_i für die Bit-Scheibe folgen dem 4-Bit-Steuerregister **313**. Wenn das Eingangsfreigabebit **303** hoch ist, und ein Steuerregister **313** Bit niedrig ist, wird der externe Eingang I_i zu der verknüpften Ausgabe für das Anweisungsbit J_i weitergeleitet und das verknüpfte K_i wird auf niedrig gezwungen. Wenn das Eingangsfreigabebit **303** hoch ist und ein Steuerregister **313** Bit hoch ist, dann wird der externe Eingang I_i zu dem verknüpften K_i weitergeleitet und die verknüpfte Ausgabe für das Anweisungsbit J_i wird auf hoch gezwungen.

[0107] Der Vorteil des Bereitstellens der dynamischen Anweisungsmaske **305** ist, dass, um wirksamere Verwendung dynamischer Anweisungen zu machen, es häufig wünschenswert ist, andere Schaltungen synchron mit der ALU zu steuern. Zum Beispiel muss in bestimmten Fällen, wenn eine ALU zwischen Additions- und Subtraktionsoperationen geschaltet werden muss, die Konstante in das am Wenigstens signifikante Bit der Übertragungsleitung auf 1 für die Subtraktion und auf 0 für die Addition gesetzt werden. Das Bereitstellen der dynamischen Anweisungsmaske **305** vermeidet die Erfordernis zusätzlicher Steuereingänge, um diese periphere Schaltung zu steuern, wie es in Fig. 17 gezeigt ist. Die Maskenschaltung erlaubt bestimmten Bits des dynamischen Anweisungseingangs, in die ALU gespeist zu werden, und anderen Bits des dynamischen Anweisungseingangs, in die periphere Schaltung gespeist zu werden.

[0108] In dem in Fig. 17 gezeigten Fall hat das Steuerregister **313** den Wert 0011. Die Auswirkung davon ist, dass I_3 und I_2 sich jeweils mit J_3 und J_2 verbinden, aber I_1 und I_0 sich mit K_1 und K_0 verbinden. J_1 und J_0 sind beide auf einen Wert von 1 festgelegt. Dies gibt einen erweiterten Satz von ADD- und SUB-Anweisungscodes, welche ADD_LSB und SUB_LSB Codes einschließen, die angepasst sind, die am Wenigsten signifikanten Bits zu handhaben. Die sich ergebenden Codes für I, um diesen Satz von Funktionscodes zu erreichen, sind:

I Eingang ADD Anweisungscode	0000
I Eingang ADD_LSB Anweisungscode	0001
I Eingang SUB Anweisungscode	1100
I Eingang SUB_LSB Anweisungscode	1111

[0109] Die ALU-Anweisungscodes J werden die selben (0011) sowohl für ADD und ADD_LSB sein, aber für ADD wird C_{in} einfach weitergeleitet, um für die Bitscheibe C_{in} zu sein, während für ADD_LSB der Wert von C_{in} für die Bit-Scheibe immer 0 ist. Der Ort für die SUB-Anweisungen ist ähnlich: für SUB_LSB ist der Wert von C_{in} für die Bit-Scheibe immer 1.

[0110] Die Kombination von ALU-Anweisungen, die zur selben Zeit, wenn die periphere Schaltung von dem dynamischen Anweisungseingang gesteuert wird, wird daher eingeschränkt. Dies führt jedoch zu keiner praktischen Schwierigkeit, da nur eine beschränke Zahl von Fällen behandelt werden muss. Zum Beispiel benötigt der in Fig. 17 veranschaulichte Fall, dass es zwei gemeinsame Bits zwischen den ADD- und SUB-Anweisungen gibt: in diesem Fall Bits J_1 und J_0, welche beide einen Wert von 1 haben.

[0111] Die 4-Bit-Ausgabe eines ALU kann daher als ein dynamischer Anweisungseingang I für eine andere ALU verwendet werden. Die Übertragsausgabe einer ALU kann auch verwendet werden als der Übertragseingang für eine andere ALU und dies kann bei der Bereitstellung von dynamischen Anweisungen ausgenutzt werden. Es gib drei grundlegende Weisen, in welchen die Operation einer ALU dynamisch variiert werden kann:

1. C_{in} kann verwenden werden, um zwischen zwei Versionen einer Funktion zu multiplexen, wobei die Anweisungsbits I konstant bleiben. Ein Beispiel ist in Fig. 13 gezeigt, welche Multiplexen zwischen OR und AND zeigt. Dies erlaubt Multiplexen zwischen Funktionen, die in den zwei rechten Spalten in Tabelle 4 benachbart gezeigt sind.
2. Die Anweisungsbits I können geändert werden, während C_{in} dieselben bleiben. Dies erlaubt das Tauschen von Funktionen in der selben Spalte der Anweisungstabelle, welche die selben Erfordernisse für C_{in} haben: zum Beispiel zwischen NAND und XOR, die Anweisungseingänge 1000 und 0000 jeweils mit Übertragseingang C_{in} = 0 haben. In der Praxis wird dies wie in Fig. 14 gezeigt am Leichtesten erreicht werden, mit einer zweiten ALU, die verwendet wird, um zwischen zwei Anweisungen zu multiplexen. Die zwei I-Werte werden als A- und B-Eingänge für die erste ALU verwendet, die mit der Multiplex-Funktion 0110 programmiert ist, und die Ausgabe wird als I-Eingang für die zweite ALU bereitgestellt.
3. Sowohl die Anweisung als auch der Wert von C_{in} kann verändert werden. Dies erlaubt das Tauschen zwischen irgendwelchen zwei Einträgen in den zwei rechten Spalten der Anweisungstabelle. Fig. 15 zeigt diese Anordnung, die ähnlich der aus Fig. 14 ist, in welcher sie eine erste ALU als einen Multiplexer und eine zweite ALU verwendet, welcher die Anweisung gegeben wird, die bei dem Multiplexer als ihr I-Eingang gewählt ist. In diesem Fall sind die Funktionen XOR mit Anweisungscode 0000 und NOR mit Anweisungscode 1000: die benötigten Werte von C_{in} sind jeweils 0 und 1. Das als C_{in} verwendete Signal der Multiplex-ALU, um zwischen den alternativen Anweisungseingängen zu wählen, wird hier als C_{out} der Multiplex-ALU weitergeleitet und als C_{in} der zweiten ALU verwendet, und die benötigte Kombination von I und C_{in} ergibt sich.

[0112] Wie vorher angezeigt, ist die Nachschlagtabellenoperation möglich, da die ALU die Speicherschnittstellenlogik für den benachbarten Schaltblock enthält. Eine ALU und Schaltblockpaar kann daher in eine 4-Eingang-, 4-Ausgabe-Nachschlagtabelle (LUT; engl.: look-up table) umgewandelt werden. Die LUT-Adresse wird von dem A-Eingang genommen, so dass die Summenausgabe jede beliebige Boolesche Funktion von A sein kann. Dies kann für Anweisungen nützlich sein, die nicht wirksam mit dem ALU-Anweisungssatz implementiert werden können (mögliche Beispiele wären die Gleichheitserzeugung, Bit-Rotation und der Abgleich von komplexen Mustern in einer CASE-Anweisung). Die LUT-Operation ist durch ein Modus-Bit innerhalb der ALU wählbar und kann bei diesem Anführungsbeispiel von einem I-Eingang nicht ausgewählt werden.

[0113] Während der LUT-Operation einer ALU ist der I-Eingang ausgeschaltet. Wahlweise können Schreibvorgänge zu dem Speicher während der LUT-Operation erlaubt werden, in welchem Fall B als der Dateneingang und C_{in} als Schreib-Freigabe verwendet werden. Die LUT-Ausgabe kann verwendet werden, um die Ausgabe der ALU in der normalen Art und Weise anzusteuern.

[0114] Der Grundmodus eines Prozessors in dieser Prozessoranordnung ist, eine aus einem vorher bestimmten Satz von logischen Operationen auf zwei oder drei Eingängen durchzuführen. In dem einfachsten Fall ist die von einem Prozessor durchgeführte Anweisung statisch in vier Bits vom Konfigurationsspeicher program-

miert und die Anweisung ändert sich nicht zwischen nachfolgenden Taktzyklen. Es ist jedoch, wie oben gezeigt, möglich, dass dynamische Anweisungen auch bereitgestellt werden: die Anweisung, welche die Operation jedes Prozessors bestimmt, wird dann als eine logische Kombination der 4 Bits des Konfigurationsspeichers zusammen mit einem 4-Bit-Eingang, der von dem allgemeinen Vernetzungsnetzwerk genommen wird, gebildet. Die Funktion des relevanten Prozessors (oder Teils der Anordnung) kann dann Zyklus nach Zyklus geändert werden, was die Kosten von vollständigen Rekonfigurationen erspart, die Zahl von Anweisungen, die bereitgestellt werden können, erhöht und die Kosten von Daten-abhängiger Operation verringert.

[0115] Da dynamische Anweisungen einem Prozessorelement bereitgestellt werden können und da Benutzerebenenspeicher erhältlich ist (in dem Fall dieser Anordnung durch Umwandlung von Konfigurationsspeicher in Benutzerebenenspeicher), ist herausgefunden worden, dass es möglich ist, eine CPU innerhalb der Prozessoranordnung zu bilden. Ein Beispiel einer Bit-Scheibe einer sehr einfachen CPU ist in Fig. 18 gezeigt. Es sollte erwähnt werden, dass viel komplexere Bit-Scheiben auch möglich sind, wobei die Bit-Scheibe der Fig. 18 die einfachste Baubare ist. Sie umfasst einen Speicher wie zum Beispiel RAM **401** (der zum Beispiel ein zu einem Benutzerebenenspeicher umgewandelter Konfigurationsspeicher sein kann), die arithmetisch-logische Einheit (ALU) **402** eines Prozessorelements, das Ausgaberegister **403** des Prozessorelements und ein Codespeicher **404**, der im Allgemeinen auch RAM sein wird (und wiederum ein oder mehrere Prozessorelemente, die zu Benutzerebenenspeicher umgewandelt wurden, sein könnte). RAM **401** hat Ausmaße von 16 Wörtern von 4 Bits. Diese CPU ist ausgebildet, auf einer 4-Bit breiten Scheibe von Daten zu arbeiten, da dies die Bit-Scheibe ist, die von dem Prozessorelement gehandhabt werden kann. Für eine benötigte Datenwegweite von größer als 4 Bits werden ein oder mehrere zusätzliche Prozessorelemente benötigt.

[0116] RAM **401** wirkt als eine Registerdatei für die CPU und hält Daten für die CPU. Anweisungen für die CPU werden von dem Codespeicher **404** empfangen. Diese Anweisungen sind von zwei Arten: Anweisungen für die ALU **402** selbst und Adress- und Lese/Schreib-Steuerung für den RAM **401** (Adressinformation, die von einem Adressanschluss zufließt, als ADDR gezeigt, durch vier zugeordnete Drähte, und Lese/Schreib-Steuerung durch einen als R/W gezeigten separaten Anschluss). Vorteilhafterweise wird diese Anweisungsinformation in einer verkleinerten Form im Codespeicher **404** gehalten werden, in welchem Fall ein zusätzlicher Anweisungsdekoder zwischen dem Codespeicher **404** und der CPU benötigt werden wird. Die von der ALU **402** durchgeführte Anweisung hat ein Ergebnis, welches in dem Ausgaberegister **403** gespeichert wird. Dieses Ergebnis kann in den nächsten Zyklus in die ALU **402** zurückgespeist werden oder kann im RAM **401** gespeichert werden: in einem einzelnen Zyklus kann es entweder ein Lesen von oder ein Schreiben auf den RAM **401** geben.

[0117] Ein einfacher Anweisungssatz, der für diese CPU geeignet ist, ist der Folgende (wobei REG das Register **403** ist, der RAM RAM **401** ist und addr die Adresse für RAM **401** ist):

REG := RAM(addr)	– lade Wortzahl (addr) von RAM **401** in REG
REG := NOT RAM(addr)	– lade das logische Inverse von RAM(addr) in das REG
REG := REG NAND RAM(addr)	– führe ein bitweises logisches NAND der Werte in REG und RAM(addr) durch, speichere das Ergebnis in REG
REG := REG AND NOT RAM(addr)	– führe eine bitweises logisches AND des Werts in REG und des logischen Inversen des Werts in RAM(addr) durch, speichere das Ergebnis in REG
REG := REG + RAM(addr)	– führe eine arithmetische Addition der Werte in REG und RAM(addr) durch, speichere das Ergebnis in REG
REG := REG OR RAM(addr)	– führe ein bitweises logisches OR der Werte in REG und RAM(addr) durch, speichere das Ergebnis in REG
RAM(addr) := REG	– erlaube, dass die Registerdatei beschrieben wird.

[0118] Es ist einfach, eine CPU mit einer komplexeren Datenweg-Scheibe, die für bestimmte Operationstypen optimiert ist, mit den erhältlichen Funktionseinheiten zu entwerfen. Die Zahl der benötigten ALUs hängt von dem für die CPU benötigen Anweisungssatz ab – es wird im Allgemeinen wünschenswert sein, dass die Minimalanzahl von ALUs, die notwenig ist, um den benötigten Anweisungssatz, der zu verwenden ist, zu implementieren, von Zeitbeschränkungen abhängt. Wenn zusätzliche Register benötigt werden, ist es einfach, die-

se aus einem oder mehreren 16-Wort mal 4-Bit-RAMs (ähnlich zu dem RAM **401**) zu konstruieren. Wenn es eine Zahl von ALUs mit einer Datenweg-Scheibe gibt, ist es notwendig, dass jede mit Anweisungswerten auf ihren Anweisungseingängen angetrieben wird, die sie veranlasst, die benötigte Anweisung gemeinsam zu berechnen.

[0119] Es kann mehr als eine Datenweg-Scheibe ergeben. In diesem Fall ist es vorteilhaft, dass die entsprechenden ALUs in jeder Scheibe Anweisungseingänge von einem gemeinsamen Codespeicher teilen: wenn Anweisungen in dem gemeinsamen Codespeicher komprimiert sind, können dekomprimierte Anweisungen jeder ALU durch einen gemeinsamen Anweisungsdekoder bereitgestellt werden. Überträge werden von der weniger signifikanten zu der mehr signifikanten Datenweg-Scheibe für jede einzelne ALU aneinander gekettet, welche die Anweisungen unter Verwendung der Übertragskette implementiert.

[0120] Es gibt eine Zahl von Weisen, die vorhanden sind, um den Anweisungsstrom für eine CPU dieser Art zu erzeugen. Die Einfachste ist in **Fig. 18** angezeigt: der Strom von Anweisungen wird von einem Speicher eingelesen. Jedoch ist es auch möglich, dass Anweisungen von Datenwerten abgeleitet werden, mit nachfolgender datenabhängiger Ausführung. Die flexibelste Anordnung wird eine Kombination von diesen beiden Methoden verwenden.

[0121] Das Aufrührungsbeispiel der Erfindung ist bloß mittels eines Beispiels beschrieben worden und viele Änderungen und Entwicklungen können gemacht werden, wobei an der vorliegenden Erfindung festgehalten wird. Zum Beispiel verwendet das Ausführungsbeispiel 4-Bit-ALUs als die Verarbeitungseinheiten, aber andere Formen von ALU oder andere Verarbeitungseinheiten können zusätzlich oder alternativ verwendet werden.

[0122] Ferner ist das Ausführungsbeispiel beschrieben worden, als ob die ganze Anordnung von ALUs und Schaltbereichen bedeckt ist. Jedoch können andere Arten von Bereichen in die Anordnung eingeschlossen werden. Zum Beispiel kann eine Unter-Anordnung aus einer 4 × 4 Anordnung von Kacheln von ALUs und Schaltbereichen wie oben beschrieben zusammengesetzt sein und die Anordnung kann aus solchen Unter-Anordnungen und Speicher in einer 4 × 4 Anordnung oder solchen Unter-Anordnungen und RISC-CPUs in einer 4 × 4 Anordnung zusammengesetzt sein.

[0123] Bei dem oben beschriebenen Ausführungsbeispiel ist jeder ALU-Ort quadratisch und jeder Schaltbereich ist quadratisch und von der selben Größe wie ALU-Orte, aber es sollte erwähnt werden, dass die steuerbaren Schalter **18** in den Registerbussen vregw, vrege, hregn, hregs übergreifen auf den quadratischen Umfang der ALU-Orte. Die ALU-Orte müssen nicht von der selben Größe wie die Schaltbereiche sein und insbesondere können sie kleiner sein, was einem oder mehreren Bussen erlaubt, unmittelbar von einem Schaltbereich **14** horizontal oder vertikal zu einem diagonal benachbarten Schaltbereich **14** zu reichen, der zum Beispiel zwischen den Bussen h2s, h2n oder zwischen den Bussen v2e, v2w verläuft.

[0124] Im oben beschriebenen Ausführungsbeispiel hat jede ALU zwei unabhängige Übertragseingänge vci, hci und ein verbundenes Paar von Übertragsausgaben co. Falls benötigt, können die ALUs angeordnet werden, um mit zwei Arten von Übertrag umzugehen: einem schnellen Übertrag zwischen den benachbarten ALUs, der von einer besonderen Verwendung für Multi-Bit-Addieroperationen sein kann; und einem langsamen Übertrag, der flexibler weitergeleitet werden kann und von besonderer Verwendung für digitale serielle Arithmetik sein kann. Der schnelle Übertrag kann in einer ähnlichen Art und Weise wie die oben mit Bezug auf die Zeichnungen beschriebene angeordnet werden, wobei der langsame Übertrag programmierbare Schalter in den Schaltbereichen **14** zwischen dem Übertragsleiter und bestimmten Bits der 4-Bit-Busse tragen kann.

[0125] Bei dem oben beschriebenen Ausführungsbeispiel sind bestimmte Bitweiten, Größen von Schaltbereichen und Größen von Anordnungen erwähnt worden, aber es sollte erwähnt werden, dass all diese Werte soweit erforderlich geändert werden können. Auch sind die programmierbaren Schalter **16**, **18**, **20** so beschrieben worden, dass sie bei bestimmten Orten in jedem Schaltbereich **14** angeordnet sind, aber andere Orte können wie benötigt und gewünscht verwendet werden.

[0126] Bei dem oben beschriebenen Ausführungsbeispiel ist die Anordnung zweidimensional, aber die Prinzipien der Erfindung sind auch anwendbar auf dreidimensionale Anordnungen, zum Beispiel durch Bereitstellen eines Stacks von Anordnungen wie oben beschrieben, mit den Schaltbereichen in benachbarten Schichten, die in Bezug auf einander gestaffelt sind. Der Stack kann nur zwei Schichten umfassen, aber umfasst bevorzugt mindesten drei Schichten, und die Zahl von Schichten ist bevorzugt eine Potenz von Zwei.

[0127] Bei dem oben beschriebenen Ausführungsbeispiel können die Speicherzellen **24** durch die Gates **16g**,

18g, **20g** von den Schaltern isoliert werden, welche sie so steuern, dass die Speicherzellen für andere Zwecke verwendet werden können, das heißt in die "Benutzerebene" gesetzt werden. Die FREIGABE-Signal-Speicherzellen können jedoch nicht in die Benutzerebene überführt werden. Bei einem alternativen Ausführungsbeispiel können die Schalter in einem bestimmten Schaltbereich **14** von dem Rest der Anordnung durch weitere Schalter in den Bussen an der Grenze dieses Schaltbereichs **14** abtrennbar sein, wobei die weiteren Schalter von einer weiteren Speicherzelle gesteuert werden, die nicht in die Benutzerebene überführt werden kann.

[0128] Viele andere Änderungen und Entwicklungen können auch gemacht werden.

Patentansprüche

1. Ein umkonfigurierbares Bauteil, mit:
einer Vielzahl von Verarbeitungsbauteile (**10**);
einer Verbindungsmatrix (**14**), die eine Verbindung zwischen den Verarbeitungsbauteilen (**10**) bereitstellt; und
einem Mittel (**16**, **18**, **20**, **30**, **32**, **34**) um die Konfiguration der Verbindungsmatrix zu definieren;
dadurch gekennzeichnet, daß jedes der Verarbeitungsbauteile (**10**) eine arithmetisch-logische Einheit (ALU) aufweist, die dazu eingerichtet ist, eine Funktion an Eingangsoperanden auszuführen und eine Ausgabe zu erzeugen, wobei die Eingangsoperanden als Eingaben an die arithmetisch-logische Einheit (ALU) von der Verbindung in jedem Zyklus auf dem gleichen Weg bereitgestellt werden, und wobei Mittel bereitgestellt sind, um die Ausgabe eines ersten der Verarbeitungsbauteile an ein zweites der Verarbeitungsbauteile zu leiten, um die durch das zweite der Verarbeitungsbauteile auszuführende Funktion zu bestimmen, und wobei wenigstens eines oder alle der Verarbeitungsbauteile (**10**) kein adressierbares Eingangsregister aufweist, wobei Eingangsoperanden von der arithmetisch-logischen Einheit (ALU) direkt von der Verbindung empfangen werden.

2. Ein umkonfigurierbares Bauteil wie beansprucht in Anspruch 1, wobei jedes der Verarbeitungsbauteile eine erste Vielzahl von Konfigurationsbits hat, die bestimmt werden kann durch die Ausgabe eines anderen der Verarbeitungsbauteile, und eine zweite Vielzahl von Konfigurationsbits, die nicht durch die Ausgabe eines anderen der Verarbeitungsbauteile bestimmt werden kann.

3. Ein umkonfigurierbares Bauteil wie beansprucht in einem der Ansprüche 1 bis 2, bei dem jedes der Verarbeitungsbauteile einen ersten Operandeneingang, einen zweiten Operandeneingang, eine Funktionsergebnisausgabe, einen Übertrags-Eingang und eine Übertrags-Ausgabe hat, wobei der erste Operandeneingang, der zweite Operandeneingang und die Funktionsergebnisausgabe n-bit sind, wobei n eine Ganzzahl größer als 1 ist, und der Übertrags-Eingang und die Übertrags-Ausgabe 1-bit sind.

4. Ein umkonfigurierbares Bauteil wie beansprucht in Anspruch 3, wobei n gleich 4 ist.

5. Ein umkonfigurierbares Bauteil wie beansprucht in Anspruch 3 oder 4, wobei jedes der Verarbeitungsbauteile dazu eingerichtet ist, zum Bestimmen seiner Funktion eine n-bit Funktionseingabe von einem anderen der Verarbeitungsbauteile zu erhalten.

6. Ein umkonfigurierbares Bauteil wie beansprucht in einem der Ansprüche 3 bis 5, wobei ein Mittel bereitgestellt ist, das es erlaubt, dass der Übertrags-Eingang an einem der Verarbeitungsbauteile die Funktion der arithmetisch-logischen Einheit dieses Verarbeitungsbauteils verändert.

7. Ein umkonfigurierbares Bauteil wie beansprucht in Anspruch 6, bei dem dieses Mittel es erlaubt, dass der Übertrags-Eingang die Funktion der arithmetisch-logischen Einheit in deren logisches Komplement verändert.

8. Ein umkonfigurierbares Bauteil wie beansprucht in einem der Ansprüche 3 bis 7, bei dem für jedes der Verarbeitungsbauteile Mittel bereitgestellt sind um es zu ermöglichen, den Übertragseingang auf einem konstanten Wert zu halten.

9. Ein umkonfigurierbares Bauteil wie beansprucht in einem der Ansprüche 3 bis 6, bei dem ein erstes der Verarbeitungsbauteile dazu verwendbar ist, zwischen zwei Werten eines Befehlseingangs auf ein zweites der Verarbeitungsbauteile entsprechend dem Wert des Übertragseingangs des ersten der Verarbeitungsbauteile zu multiplexen.

10. Ein umkonfigurierbares Bauteil wie beansprucht in Anspruch 9, bei dem der Übertrags-Eingang des

ersten der Verarbeitungsbauteile durch das erste der Verarbeitungsbauteile hindurchgeleitet werden kann zu dem Übertrags-Eingang des zweiten der Verarbeitungsbauteile.

11. Ein umkonfigurierbares Bauteil wie beansprucht in einem der vorhergehenden Ansprüche, bei dem jedes der Verarbeitungsbauteile ein zwischenspeicherbares Ausgangsregister für den Funktionsausgang aufweist.

12. Ein umkonfigurierbares Bauteil wie beansprucht in einem der vorhergehenden Ansprüche, bei dem jedes der Verarbeitungsbauteile ein dynamisches Freigabegatter aufweist um zu bestimmen, ob Befehle zum Bestimmen der Funktion der arithmetisch-logischen Einheit dynamisch von der Verbindung angenommen werden oder vom Konfigurationsspeicher in dem Verarbeitungsbauteil bereitgestellt zu werden.

13. Ein umkonfigurierbares Bauteil wie beansprucht in einem der vorhergehenden Ansprüche, bei dem jedes der Verarbeitungsbauteile eine dynamische Befehlsmaske aufweist, wobei ein Anwenden der dynamischen Befehlsmaske auf einen Befehl, der von dem Verarbeitungsbauteil empfangen wurde, den Befehl in die Lage versetzt, sowohl einen Befehlseingang an die arithmetisch-logische Einheit zum Bestimmen der Funktion der arithmetisch-logischen Einheit als auch einen Peripherieschaltungs-Befehlseingang zum Steuern der Peripherieschaltung in dem Verarbeitungsbauteil bereitzustellen.

14. Ein Verfahren zum Konstruieren einer zentralen Verarbeitungseinheit aus einem umkonfigurierbaren Bauteil wie beansprucht in einem der Ansprüche 1 bis 13, umfassend:
Bereitstellen eines oder mehrerer Verarbeitungsbauteile um eine arithmetisch-logische Einheit (**402**) der zentralen Verarbeitungseinheit zu bilden;
Bereitstellen eines ersten Speichers (**401**) als Registerreihe für die zentrale Verarbeitungseinheit;
und Bereitstellen eines zweiten Speichers (**404**) als Codespeicher um Befehle für die zentrale Verarbeitungseinheit bereitzustellen;
wobei Befehlseingänge für die arithmetisch-logische Einheit der zentralen Verarbeitungseinheit von dem zweiten Speicher bereitgestellt werden.

15. Ein Verfahren wie beansprucht in Anspruch 14, bei dem eines oder beide der ersten und zweiten Speicher bereitgestellt werden durch Umkonfigurieren eines oder mehrerer Teile des umkonfigurierbaren Bauteils in Benutzerebenenspeicher.

16. Ein Verfahren wie beansprucht in Anspruch 15, bei dem eines oder mehrere Teile des umkonfigurierbaren Bauteils Teile des Mittels zum Definieren der Konfiguration der Konfigurationsmatrix enthalten.

17. Ein Verfahren wie beansprucht in Anspruch 15 oder in Anspruch 16, bei dem eines oder mehrere Teile des umkonfigurierbaren Bauteils eines oder mehrere Verarbeitungsbauteile umfassen.

18. Ein Verfahren wie beansprucht in einem der Ansprüche 14 bis 17, bei dem die zentrale Verarbeitungseinheit vielfache arithmetisch-logische Einheiten hat, und bei der Befehle für die vielfachen arithmetisch-logischen Einheiten von dem zweiten Speicher bereitgestellt werden.

19. Ein Verfahren wie beansprucht in einem der Ansprüche 14 bis 18, bei dem eine Befehlsdecodierschaltung zwischen dem zweiten Speicher und sowohl dem ersten Speicher als auch der arithmetisch-logischen Einheit der zentralen Verarbeitungseinheit bereitgestellt sind, so dass Befehle in komprimierter Form in dem zweiten Speicher gespeichert werden können und durch die Befehlsdecodierschaltung vor dem Ausführen in der zentralen Verarbeitungseinheit decodiert werden können.

20. Ein Verfahren wie beansprucht in Anspruch 19, soweit abhängig von Anspruch 18, bei dem eine gemeinsame Befehlsdecodierschaltung für die vielfachen arithmetisch-logischen Einheiten bereitgestellt ist.

Es folgen 13 Blatt Zeichnungen

Anhängende Zeichnungen

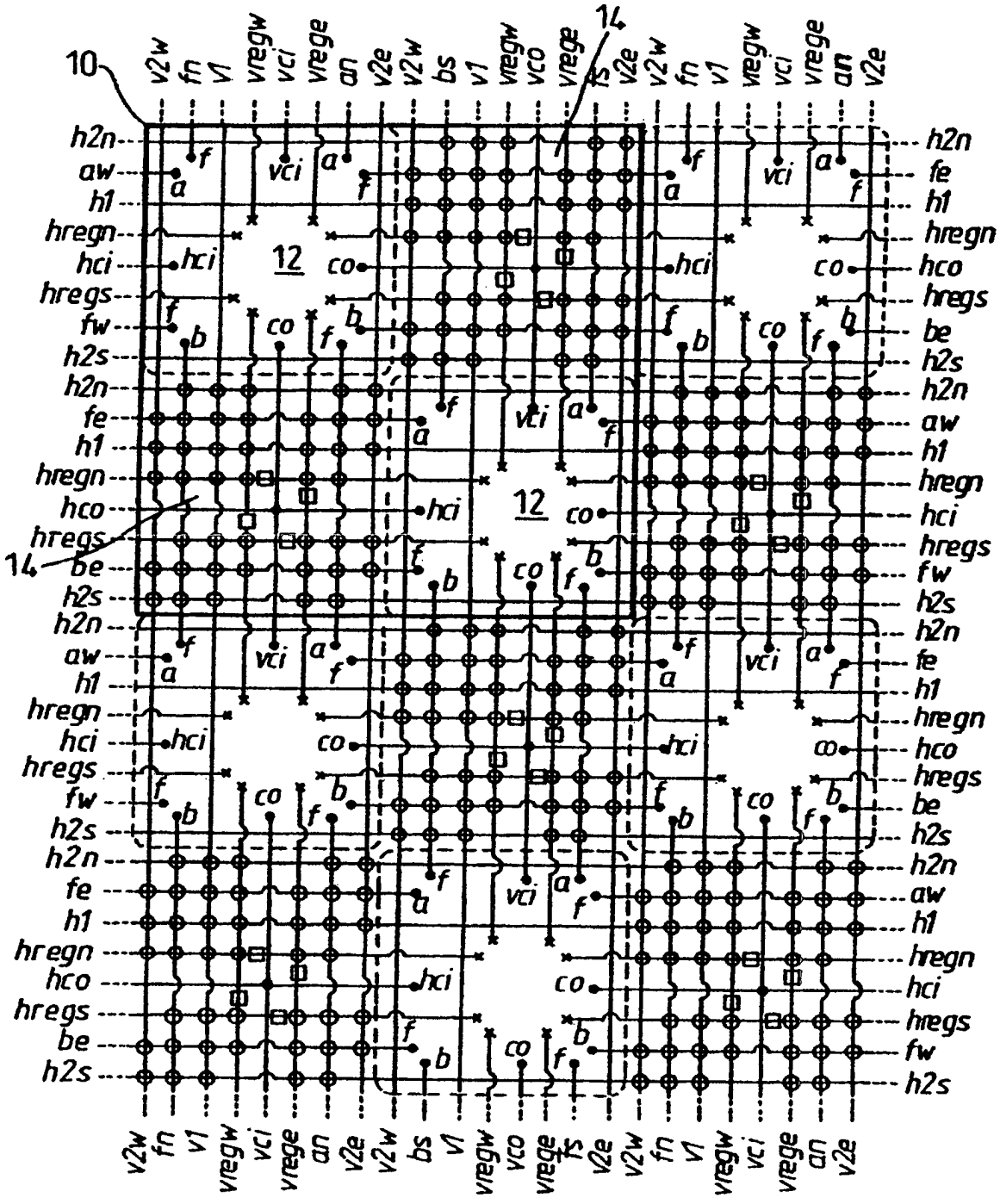

Fig. 1

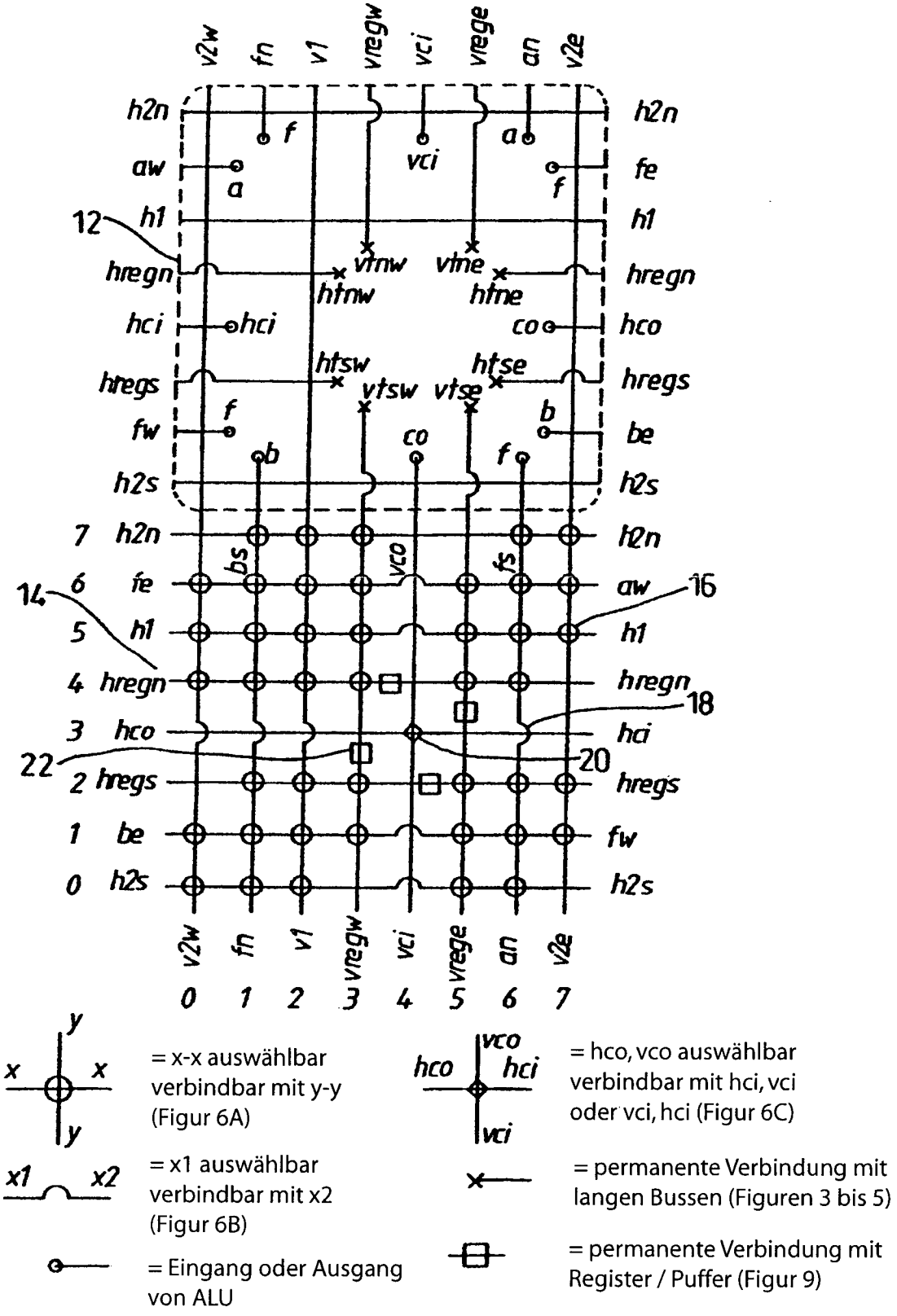
v2w
fn
v1
vregw
vci
vrege
an
v2e
h2n
aw
h1
hregn
hci
hregs
fw
h2s
fe
be
hco
12
14
16
18
20
22
htnw
vtnw
htne
vtne
htsw
vtsw
htse
vtse
bs
ts
vco
0
1
2
3
4
5
6
7
= x-x auswählbar verbindbar mit y-y (Figur 6A)
= hco, vco auswählbar verbindbar mit hci, vci oder vci, hci (Figur 6C)
= x1 auswählbar verbindbar mit x2 (Figur 6B)
= permanente Verbindung mit langen Bussen (Figuren 3 bis 5)
= Eingang oder Ausgang von ALU
= permanente Verbindung mit Register / Puffer (Figur 9)

Fig. 2

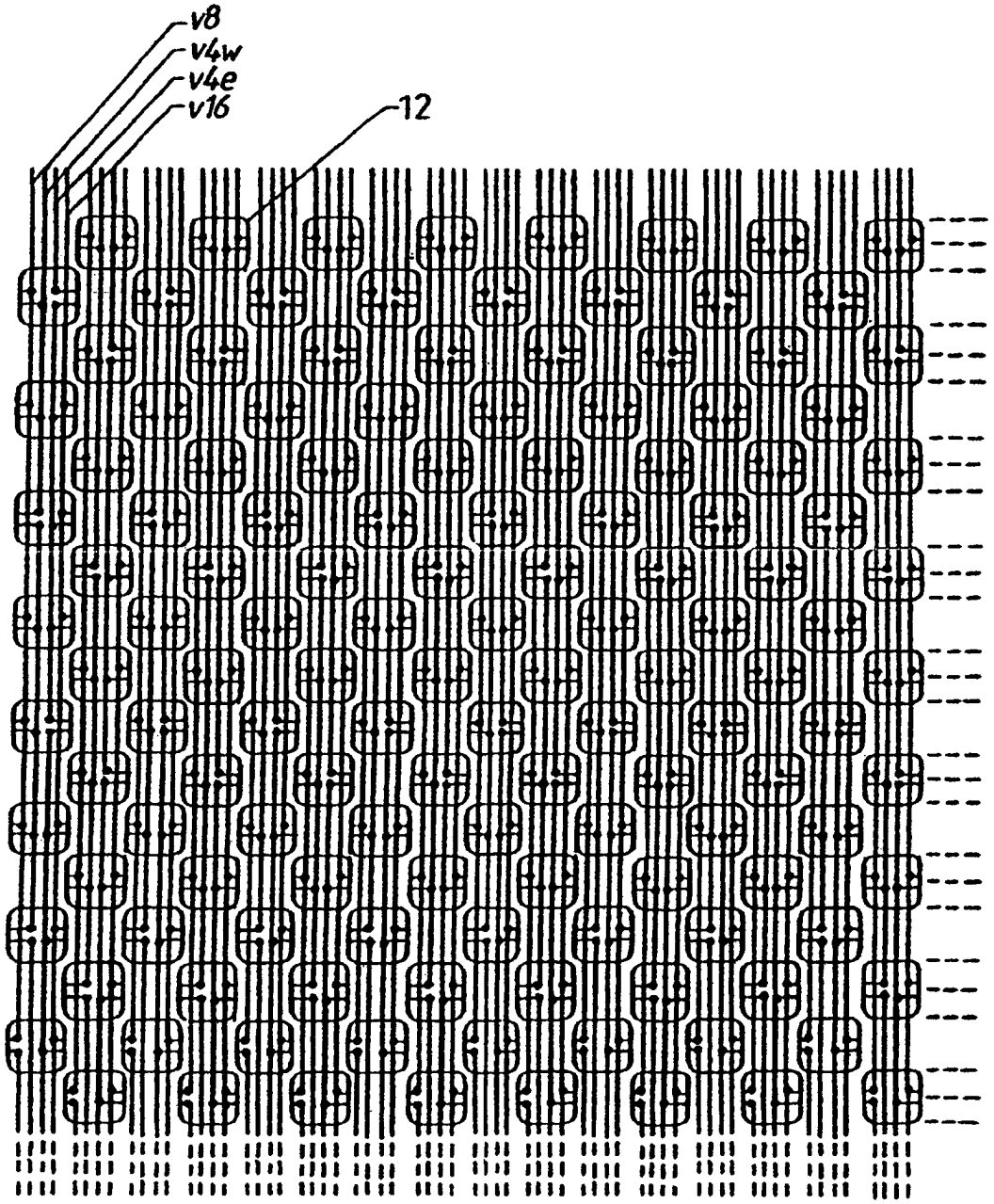

Fig. 3

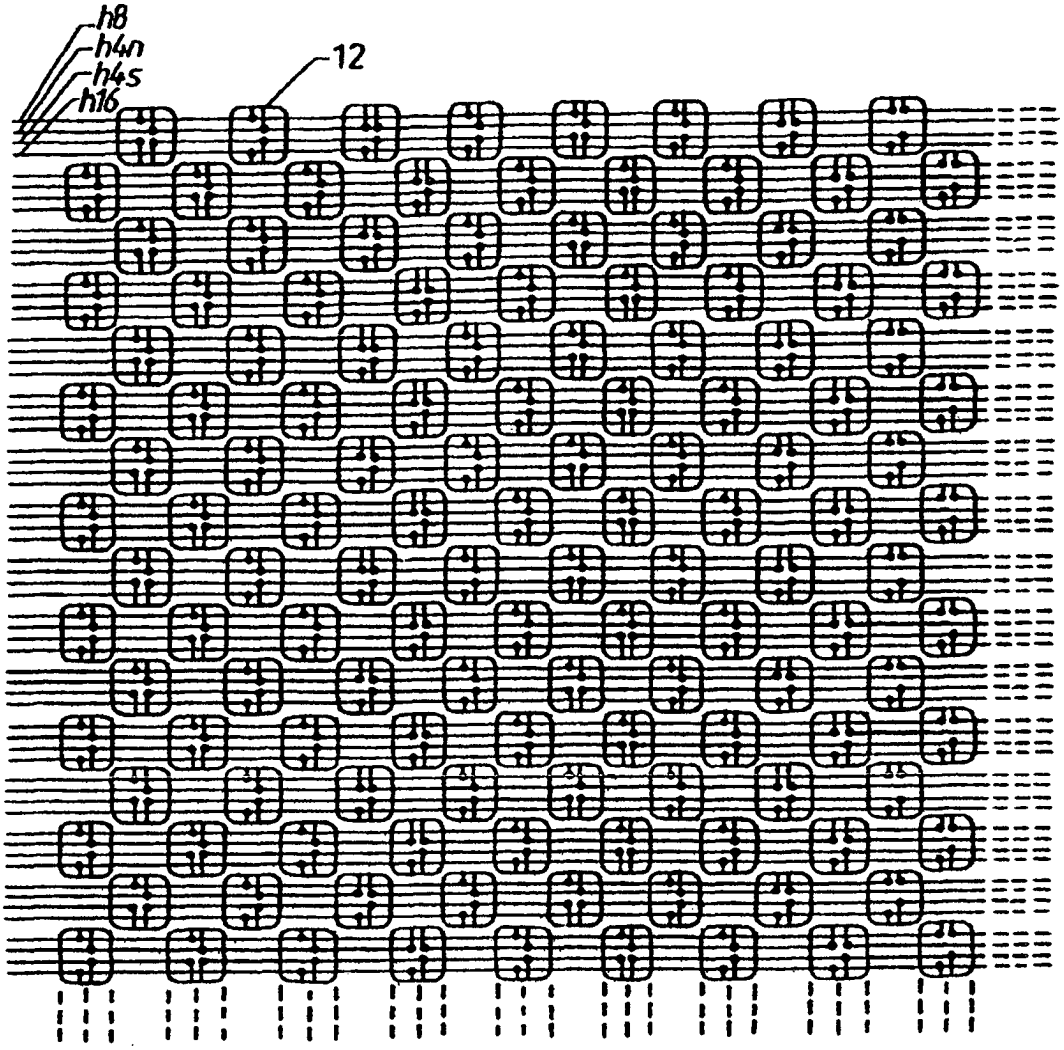
h8
h4n
h4s
h16
12

Fig. 4

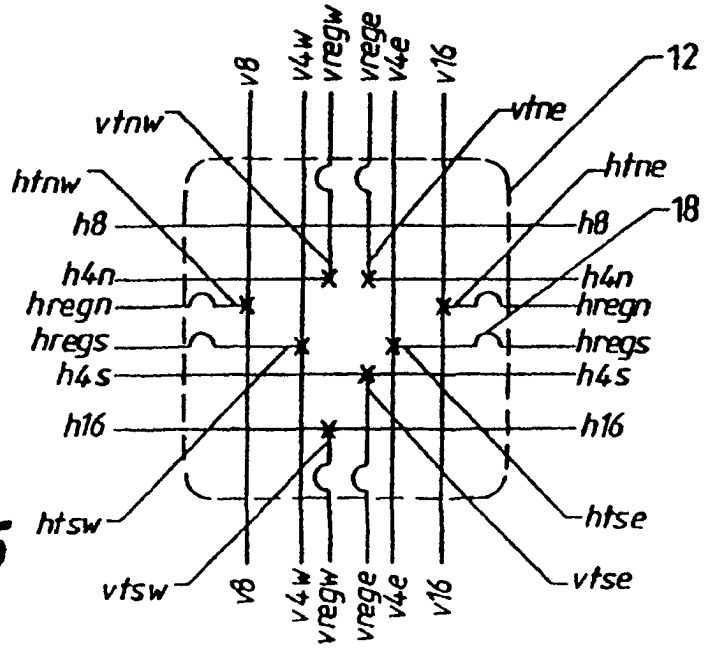
v8
v4w
vregw
vrege
v4e
v16
12
vtnw
vtne
htnw
htne
h8
h8
18
h4n
h4n
hregn
hregn
hregs
hregs
h4s
h4s
h16
h16
htsw
htse
vtsw
vtse
v8
v4w
vregw
vrege
v4e
v16

Fig. 5

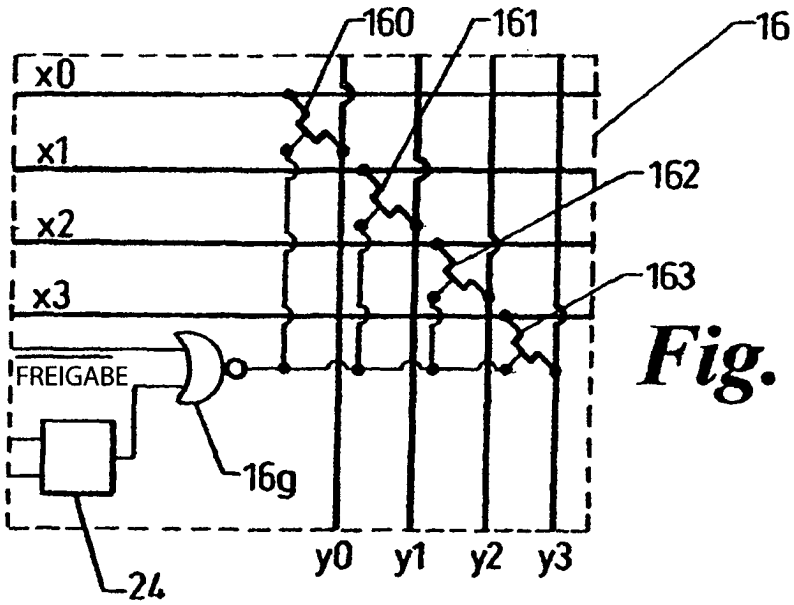

Fig. 6A

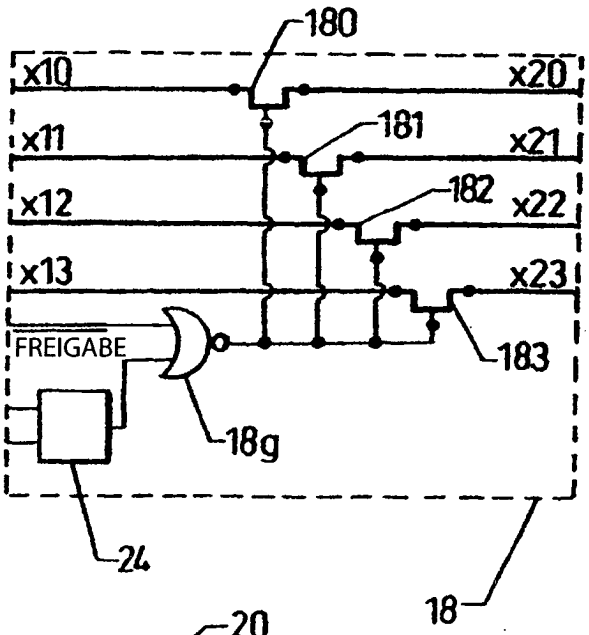

Fig. 6B

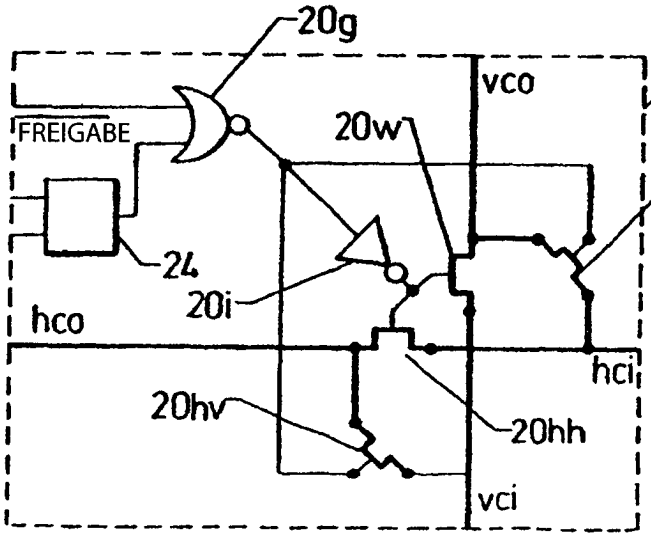

Fig. 6C

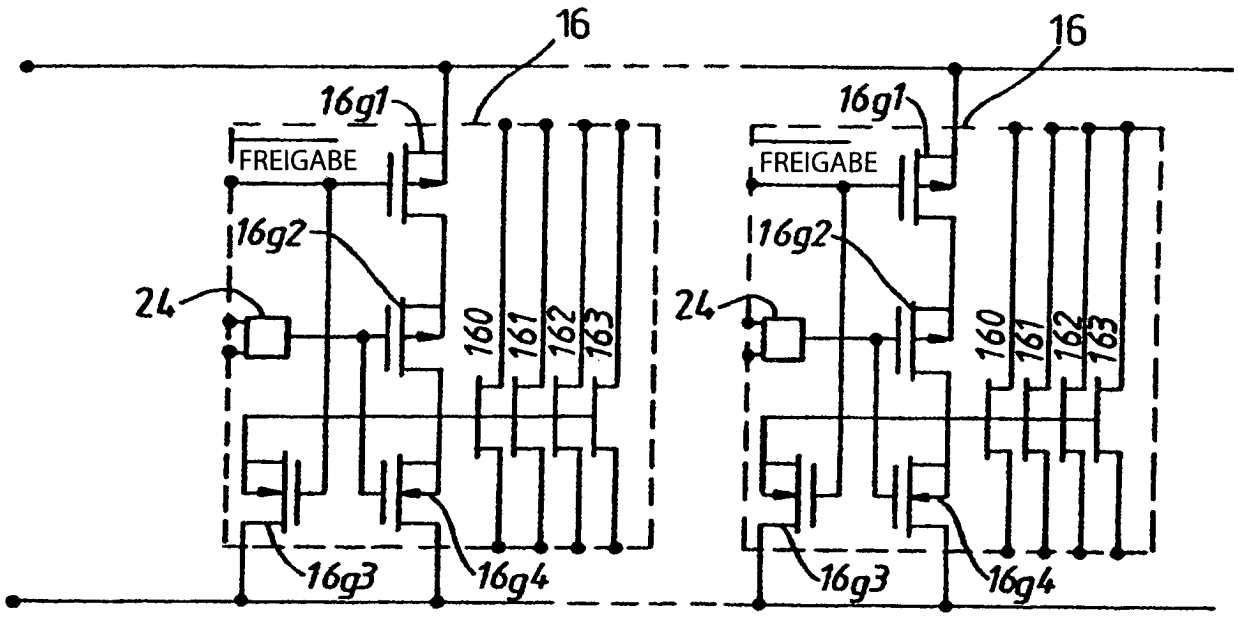
16
16g1
FREIGABE
16g2
24
160
161
162
163
16g3
16g4
16
16g1
FREIGABE
16g2
24
160
161
162
163
16g3
16g4

Fig. 7

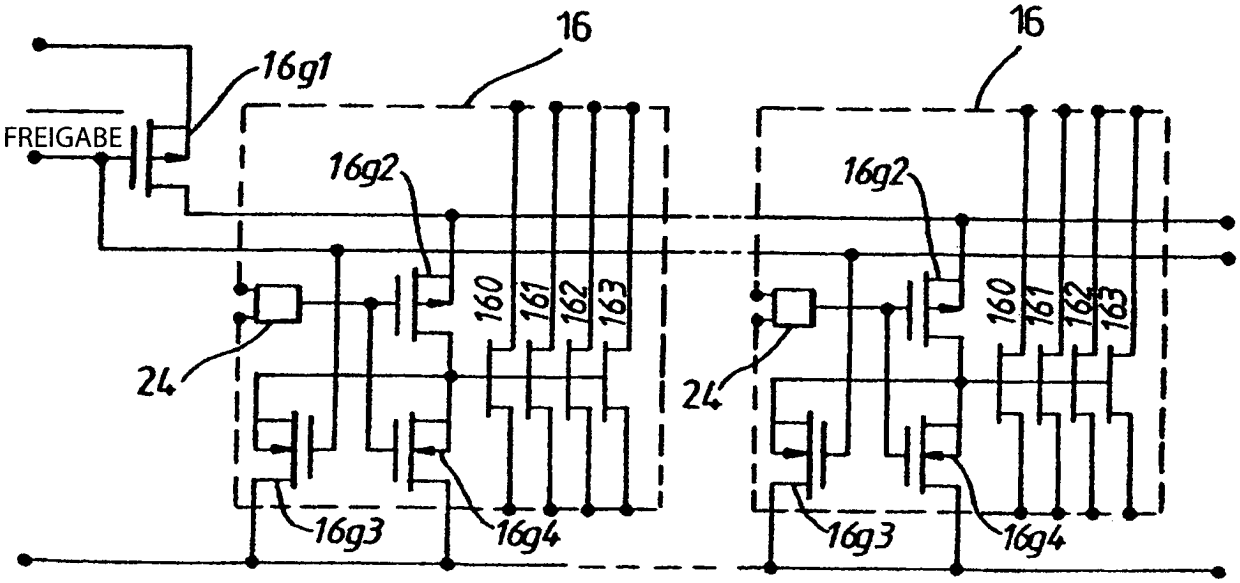
16
16g1
FREIGABE
16g2
24
160
161
162
163
16g3
16g4
16
16g2
24
160
161
162
163
16g3
16g4

Fig. 8

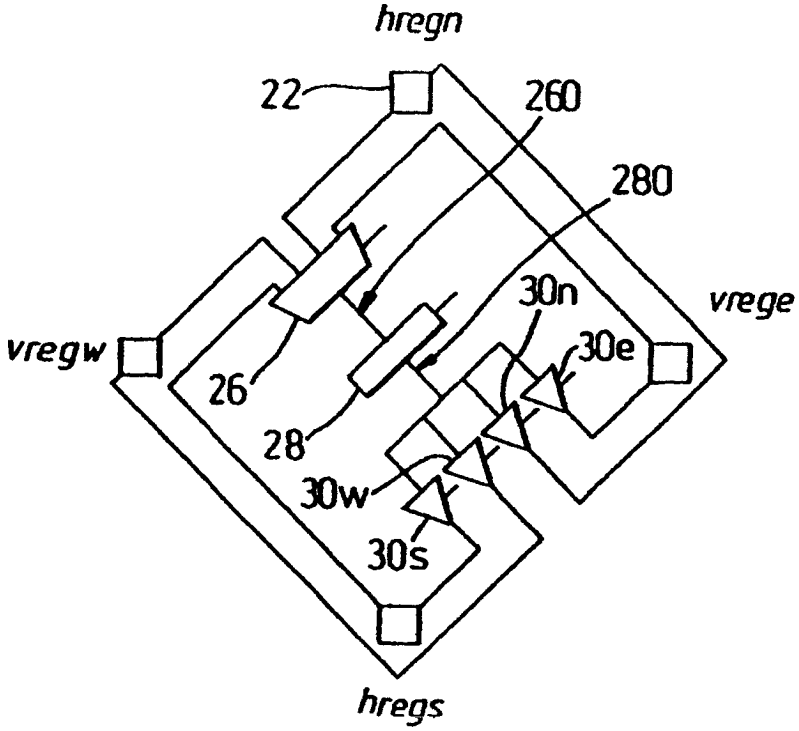
hregn
22
260
280
30n
vrege
vregw
30e
26
28
30w
30s
hregs

Fig. 9

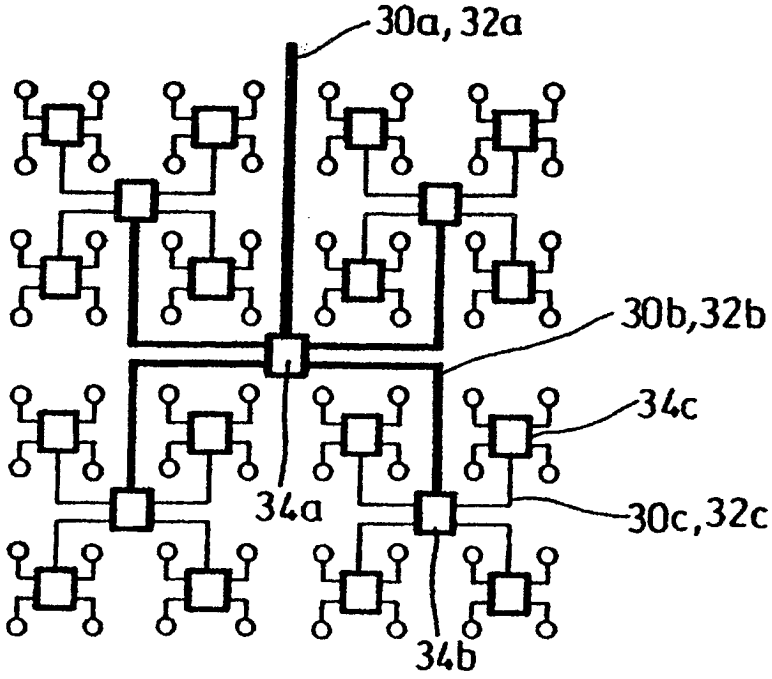
30a, 32a
30b,32b
34c
34a
30c,32c
34b

Fig. 10

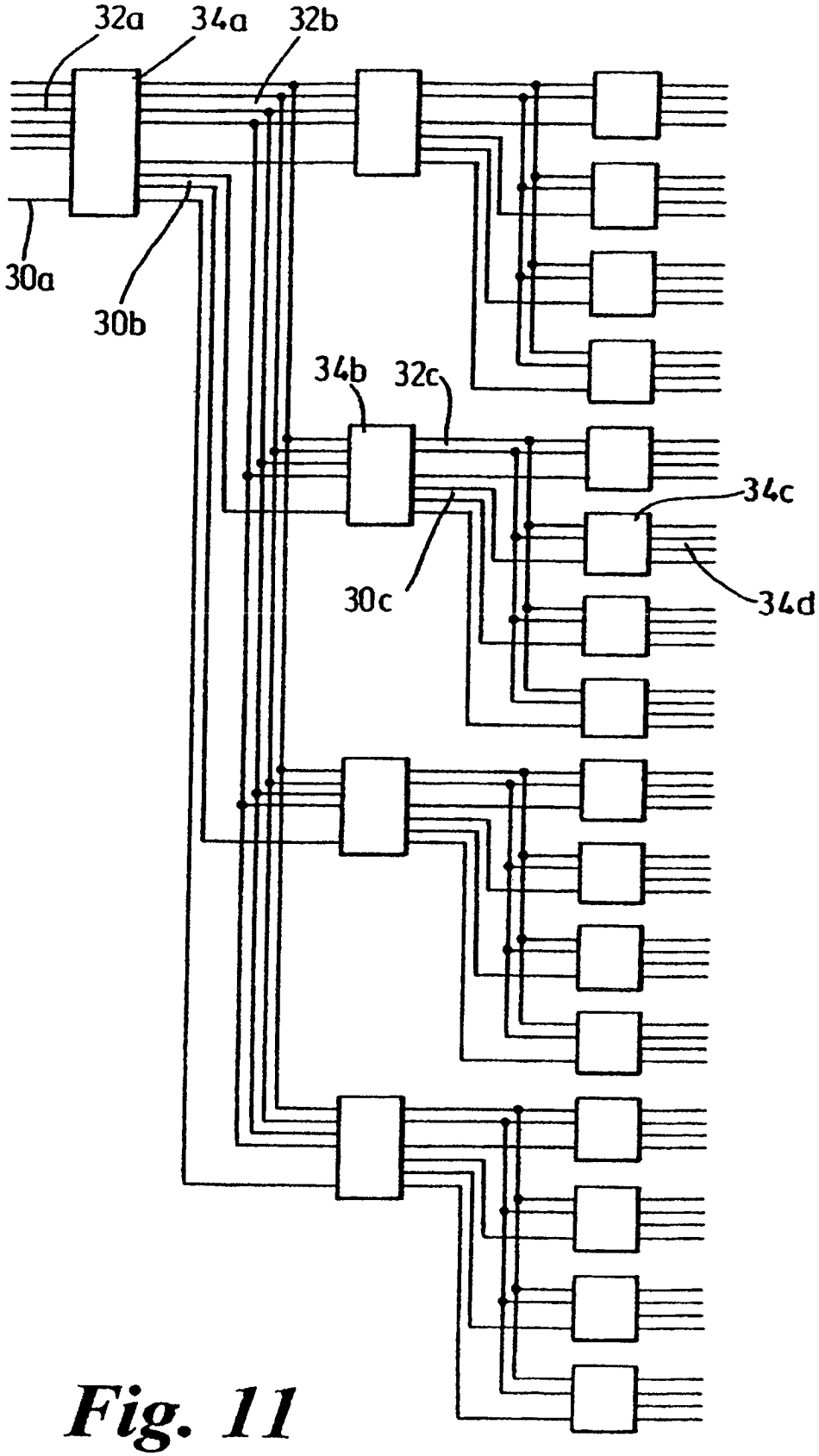

Fig. 11

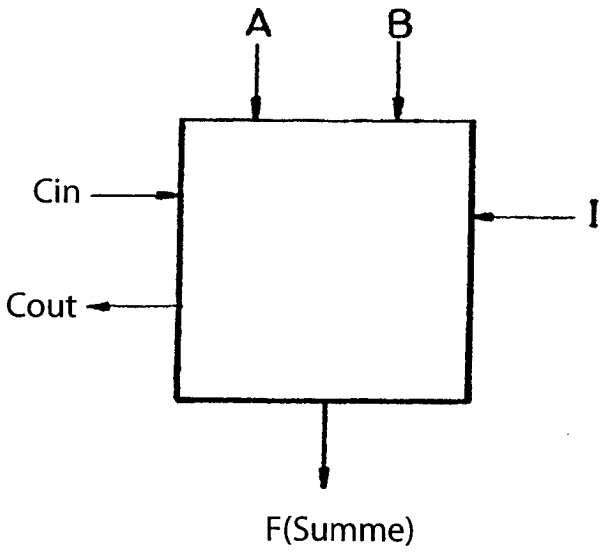

Fig. 12A

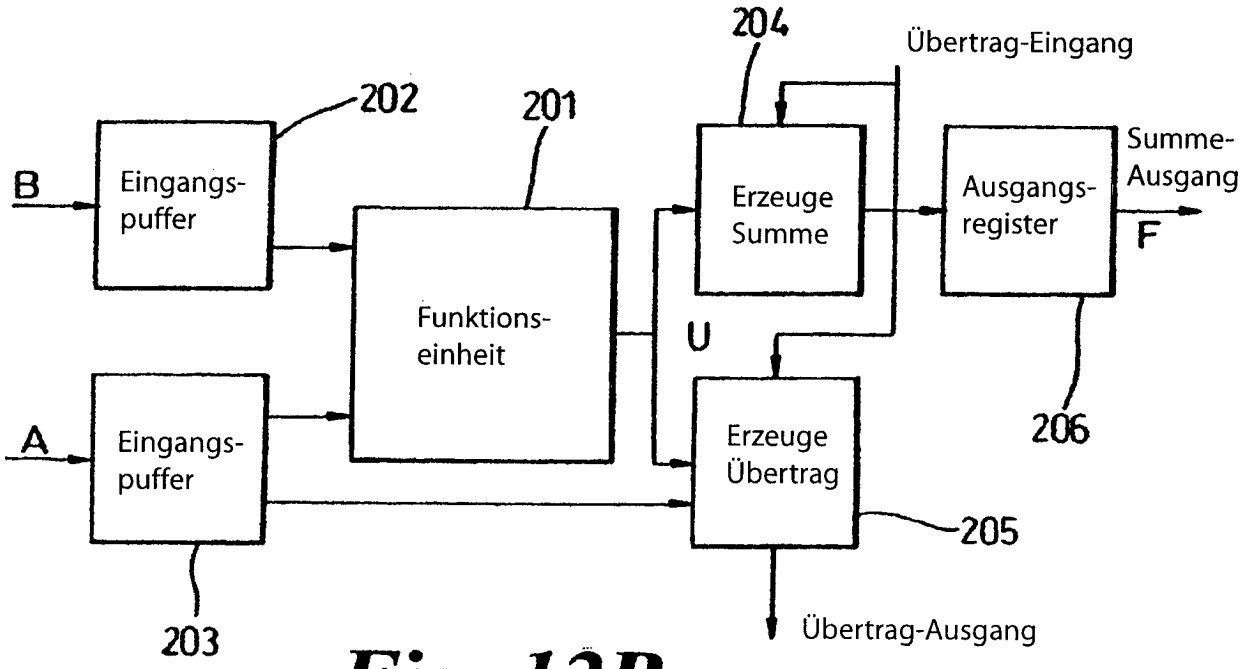

Fig. 12B

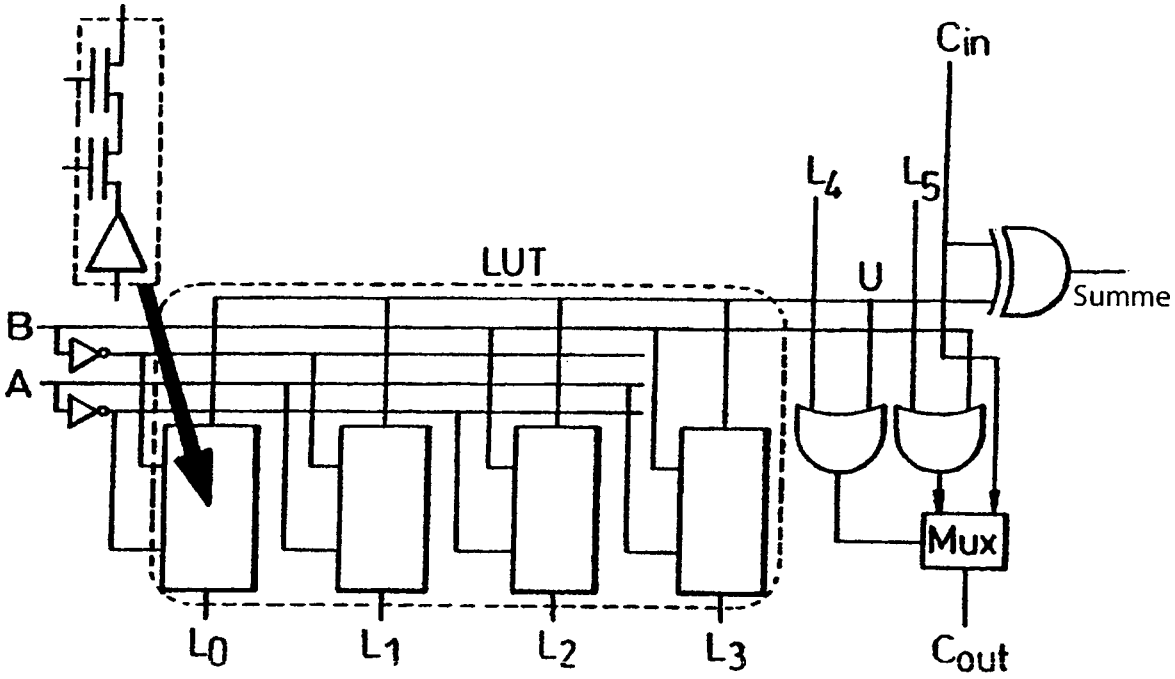

Fig. 12C

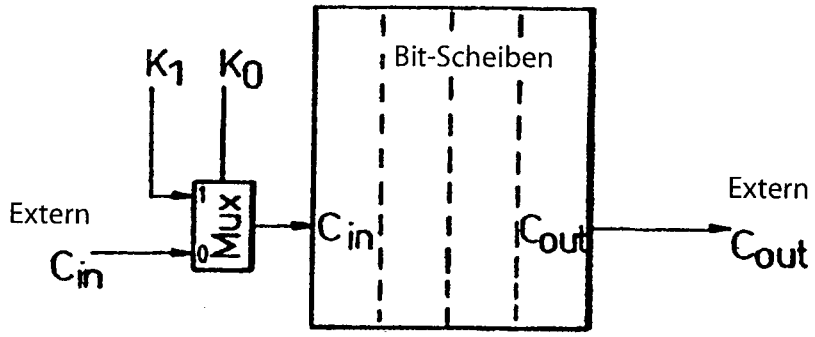

Fig. 17

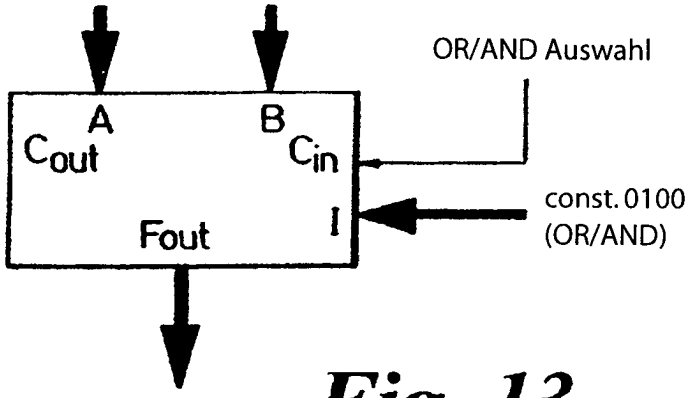

Fig. 13

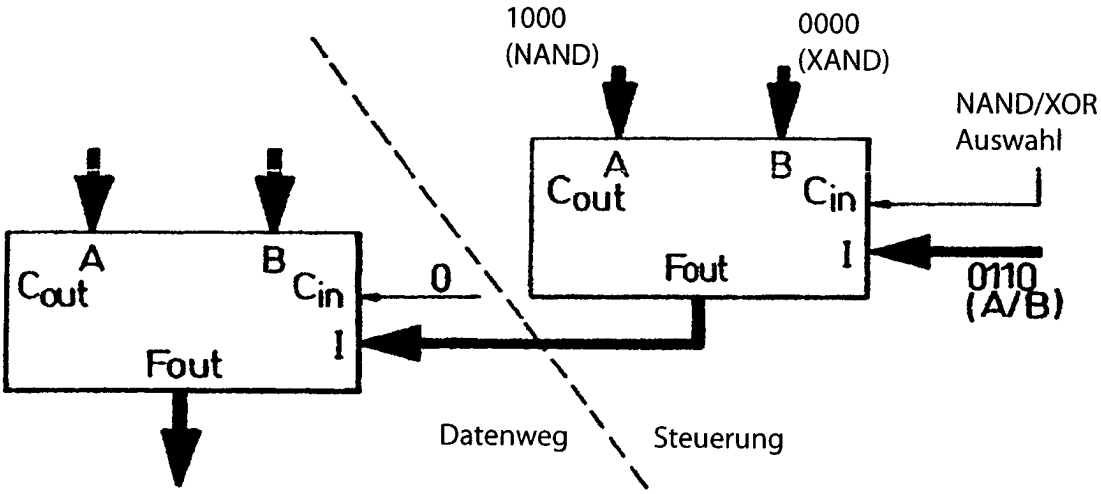

Fig. 14

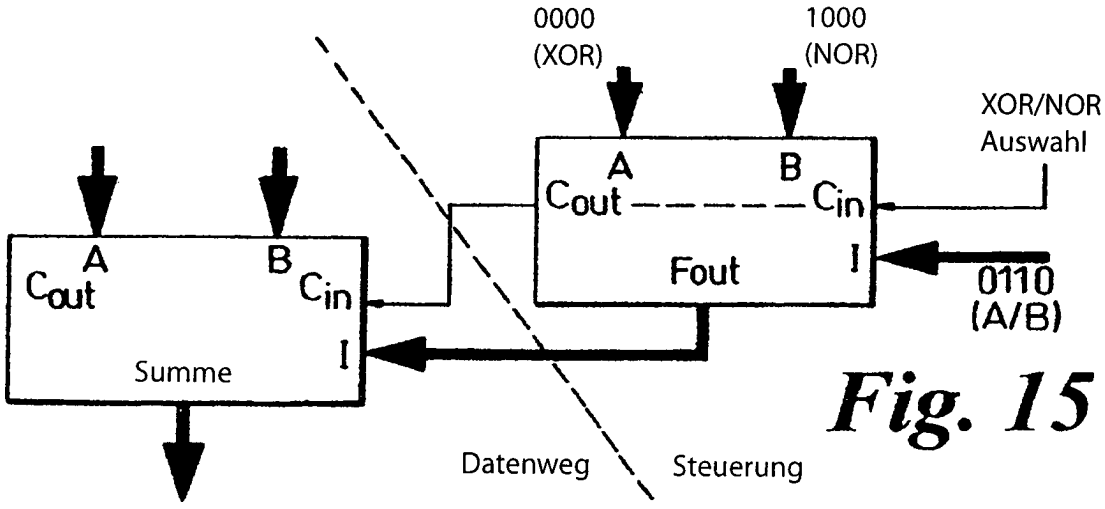

Fig. 15

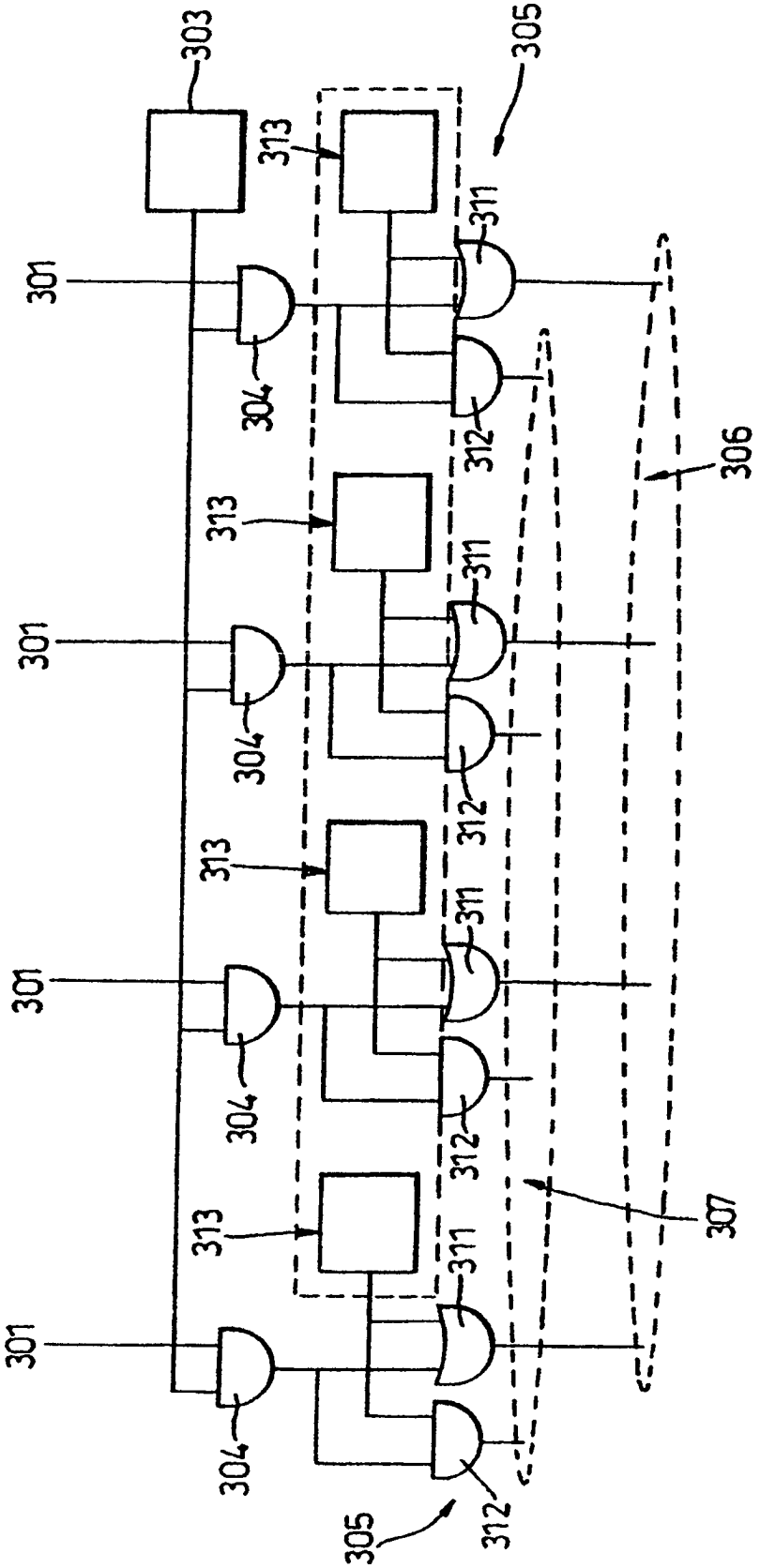

Fig. 16

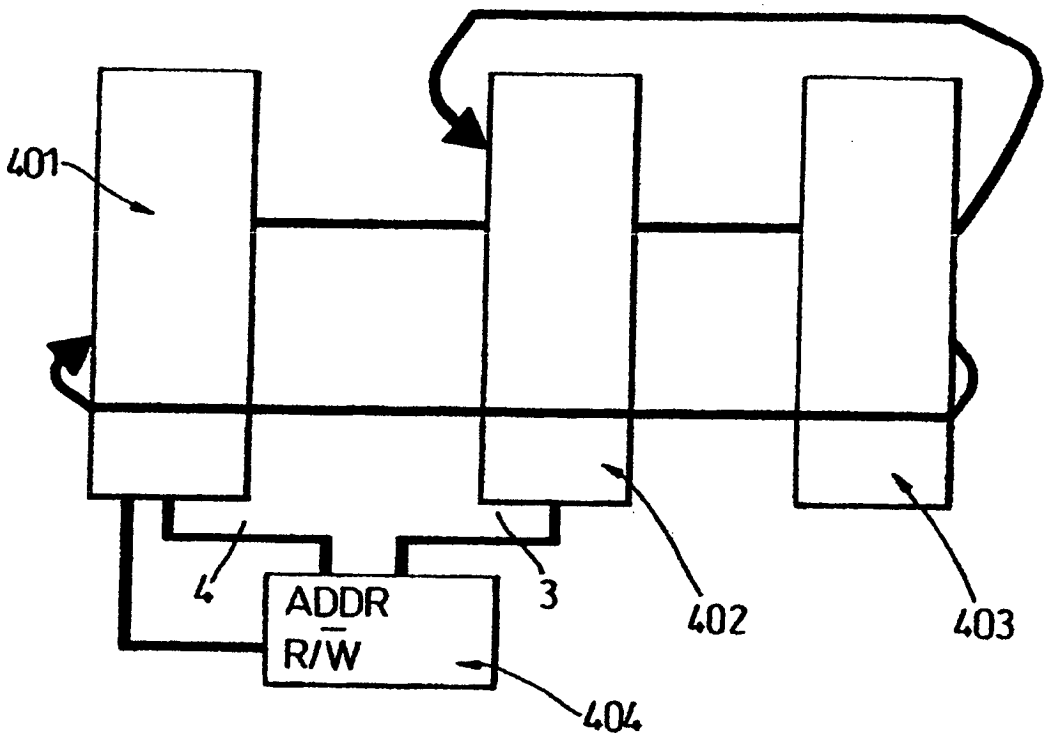

Fig. 18