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

(10) **DE 698 34 942 T2** 2007.06.06

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 038 216 B1**
(21) Deutsches Aktenzeichen: **698 34 942.3**
(86) PCT-Aktenzeichen: **PCT/GB98/03786**
(96) Europäisches Aktenzeichen: **98 960 041.6**
(87) PCT-Veröffentlichungs-Nr.: **WO 1999/031574**
(86) PCT-Anmeldetag: **16.12.1998**
(87) Veröffentlichungstag der PCT-Anmeldung: **24.06.1999**
(97) Erstveröffentlichung durch das EPA: **27.09.2000**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **14.06.2006**
(47) Veröffentlichungstag im Patentblatt: **06.06.2007**

(51) Int Cl.[8]: ***G06F 7/52*** (2006.01)

(30) Unionspriorität:

97310220	**17.12.1997**	**EP**
9812362	**09.06.1998**	**GB**

(84) Benannte Vertragsstaaten:
DE, FR, GB

(73) Patentinhaber:
Panasonic Europe Ltd., Uxbridge, Middlesex, GB

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

(72) Erfinder:
MARSHALL, Alan David, Merchant's Landing, Bristol BS1 4RJ, GB; STANSFIELD, Anthony, Hotwells, Bristol BS8 47B, GB; VUILLEMIN, Jean, F-75116 Paris, FR

(54) Bezeichnung: **Vorrichtung zum Multiplizieren**

Beschreibung

[0001] Die vorliegende Erfindung betrifft eine Implementierung von Multiplizierern in programmierbaren Arrays, insbesondere in einer rekonfigurierbaren Prozessorvorrichtung.

[0002] Eine kommerziell erfolgreiche Form einer rekonfigurierbaren Vorrichtung ist das feldprogrammierbare Gate-Array (FPGA). Diese Vorrichtungen bestehen auf einer Ansammlung von konfigurierbaren Verarbeitungselementen, die in ein konfigurierbares verknüpftes Netzwerk eingebettet sind. Ein Konfigurationsspeicher ist vorgesehen, um die Verknüpfungskonfiguration zu beschreiben – oft wird ein SRAM verwendet. Diese Vorrichtungen haben eine sehr Feinkörnige Struktur: Typischerweise ist jedes Verarbeitungselement eines FPGA ein konfigurierbares Gate. Anstatt in einer zentralen ALU konzentriert zu sein, ist die Verarbeitung so auf die Vorrichtung verteilt, und die Siliziumfläche der Vorrichtung wird effektiver ausgenutzt. Ein Beispiel einer kommerziell verfügbaren FPGA Serie ist die Xilinx 4000 Serie.

[0003] Solche rekonfigurierbaren Vorrichtungen können im Prinzip für jegliche Rechenanwendung verwendet werden, für die ein Prozessor oder ein ASIC verwendet wird. Eine besonders geeignete Verwendung von solchen Vorrichtungen besteht jedoch in der Verwendung als Koprozessor, um Aufgaben zu bewältigen, die zwar rechenintensiv, aber nicht so häufig sind, als dass sie einen für diesen Zweck gebauten ASIC rechtfertigen würden. Ein rekonfigurierbarer Koprozessor könnte folglich zu verschiedenen Zeiten mit verschiedenen Konfigurationen programmiert werden, wobei jede Konfiguration für die Ausführung einer anderen rechenintensiven Aufgabe angepaßt ist, was für größere Effizienz sorgt als im Falle eines Universalprozessors alleine, und ohne starke Erhöhung der Gesamtkosten. In neuesten FPGA-Vorrichtungen ist eine dynamische Rekonfiguration möglich, wobei eine teilweise oder vollständige Rekonfiguration während der Ausführung von Code möglich ist, so dass Zeit-Multiplexing verwendet werden kann, um Konfigurationen zur Verfügung zu stellen, die für unterschiedliche Teilaufgaben zu verschiedenen Phasen der Ausführung eines Codestückes optimiert sind.

[0004] FPGA-Vorrichtungen sind nicht speziell geeignet für bestimmte Arten von Rechenaufgaben. Da die einzelnen Rechenelemente sehr klein sind, sind die Datenwege extrem schmal und es wird eine Vielzahl von ihnen benötigt, so dass eine große Zahl von Operationen beim Konfigurationsvorgang nötig sind. Obwohl diese Strukturen relativ effizient für Aufgaben sind, bei denen kleine Datenelemente bearbeitet werden und von Zyklus zu Zyklus regulär sind, so sind sie weniger befriedigend für unregelmäßige Aufgaben mit großen Datenelementen. Solche Aufgaben werden auch von einem Universalprozessor nicht gut bewältigt, können jedoch von erheblicher Bedeutung sein (z. B. bei der Bildbearbeitung).

[0005] Alternative rekonfigurierbare Architekturen sind vorgeschlagen worden. Ein Beispiel ist die PADDI-Architektur, entwickelt von der University of California in Berkeley und beschrieben in der Arbeit von D. Chen und J. Rabacy "A Reconfigurable Multiprocessor IC for Rapid Prototyping of Real Time Data Paths", ISSCC, Februar 1992 und A. Yeung und J. Rabacy "A Data-Driven Architecture for Rapid Prototyping of High Throughput DSP Algorithms", IEEE VLSI Signal Processing Workshop, Oktober 1992. Eine weitere alternative Architektur ist MATRIX, entwickelt am Massachusetts Institute of Technology und beschrieben von Ethan Mirsky und André deHon in "MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources", FCCM '96 – IEEE Symposium on FPGAs for Custom Computing Machines, April 17–19, 1996, Napa, Kalifornien, USA und detaillierter von Andre deHon in "Reconfigurable Architectures for General-Purpose Computing", Seiten 257 bis 296, Technical Report 1586, MTT Artificial Intelligence Laboratory. Die MATRIX-Struktur hat vorteilhafte Gesichtspunkte, aber die grobe Korngröße bedeutet, dass sie mehr Silizium verbraucht als eine konventionelle FPGA-Struktur und sie womöglich weniger effizient für Aufgaben ist, die von Zyklus zu Zyklus gleichförmig sind. Es wäre daher wünschenswert, weitere konfigurierbare Strukturen zu entwickeln, welche auf bestmögliche Weise die Vorteile der MATRIX mit denen herkömmlicher FPGAs kombinieren.

[0006] Die US 5,291,431 sieht einen Array-Multiplizierer vor, der eine modifizierte Zellen-Kodierung von Eingangssignalen des Multiplizierers verwendet, und der auf der Oberfläche eines monolithischen integrierten Schaltkreises unter Verwendung von einem Computer erzeugter Masken entsprechend einem Siliziumcompilerprogramm gebildet ist, indem ein Array aus Standardzellen, die aus einer Sammlung standardmäßiger Zelldesigns ausgewählt sind, bei einem Mosaikvorgang angeordnet werden.

[0007] Eine weitere Entwicklung der Anmelder der vorliegenden Erfindung, beschrieben in der internationalen Patentveröffentlichung WO-A-98/33276, mit einer als "CHESS" bezeichneten Gesamtarchitektur, beschreibt eine rekonfigurierbare Vorrichtung, die umfasst: eine Vielzahl von Verarbeitungsvorrichtungen; eine Verknüpfungsmatrix, die eine Verknüpfung zwischen den Verarbeitungsvorrichtungen bereitstellt; und ein Mittel zum

Definieren der Konfiguration der Verknüpfungsmatrix; wobei jede der Verarbeitungsvorrichtungen eine arithmetische Logikeinheit umfasst, die angepasst ist, über Eingabeoperanden eine Funktion auszuführen und eine Ausgabe zu erzeugen, wobei die Eingabeoperanden als Eingaben der arithmetischen Logikeinheit von der Verknüpfung in jedem Zyklus auf dem gleichen Weg zur Verfügung gestellt werden, und wobei Mittel zur Verfügung gestellt werden, um die Ausgabe von einer ersten Verarbeitungsvorrichtung zu einer zweiten Verarbeitungsvorrichtung weiterzuleiten, um die Funktion zu bestimmen, die von der zweiten Verarbeitungsvorrichtung ausgeführt wird.

[0008] In einer bevorzugten Ausgestaltung von CHESS hat jede der Verarbeitungsvorrichtungen eine erste Operandeneingabe, eine zweite Operandeneingabe, eine Funktionsergebnisausgabe, eine Übertragseingabe und eine Übertragsausgabe, wobei die erste Operandeneingabe, die zweite Operandeneingabe und die Funktionsergebnisausgabe n-Bit sind, wobei n eine ganze Zahl größer als 1 ist, und die Übertragseingabe und die Übertragsausgabe 1-Bit sind. Eine besonderes gute Designlösung findet man, wenn n gleich 4 ist. Der für eine dynamische Instruktion verwendete Mechanismus besteht darin, dass jede der Verarbeitungsvorrichtungen dazu angepasst ist, zum Bestimmen ihrer Funktion eine n-Bit Instruktionseingabe von einer anderen Verarbeitungsvorrichtung zu empfangen.

[0009] Es ist außerdem vorteilhaft, dass jede der Verarbeitungsvorrichtungen ein verriegelbares Ausgaberegister für die Funktionsausgabe umfasst. Dies ist nützlich zur Konstruktion einer "tiefen" Pipeline, wenn es z. B. notwendig ist, eine Anzahl von Operationen parallel auszuführen und die Bereitstellung der Ausgabe von verschiedenen ALUs zu synchronisieren.

[0010] Eine besonders wichtige Frage für alle Architekturen, die oben beschrieben sind, ist die Implementierung eines Multiplizierers. Multiplizierer sind Schlüsselelemente für viele Berechnungen, und viele Anwendungen, die am geeignetsten für den Gebrauch eines ASIC oder Koprozessors sind, enthalten eine große Zahl von Multiplikationsoperationen. Ein konventioneller Ansatz zur Implementierung eines Multiplizierers wird nun beschrieben.

[0011] Ein kombinatorischer Multiplizierer ist üblicherweise als ein repetitierendes Array aus Kernzellen gebaut, bei dem jede Zelle einige Bits (z. B. M Bits) des Multiplikanden A mit einigen Bits (z. B. N Bits) des Multiplikators B multipliziert, um ein (M + N)-Bit-Partialprodukt zu erzeugen. Um den Bau eines vollständigen Multiplizierers zu erlauben, muss jede Kernzelle in der Lage sein, zwei zusätzliche Eingaben zu dem Partialprodukt zu addieren, d. h. die Funktion ((A·B) + C + D) zu berechnen. Die D-Eingabe wird verwendet, um alle Partialprodukte gleicher Signifikanz zu summieren, und die C-Eingabe wird verwendet, um Überträge von weniger signifikanten Partialprodukten zu addieren. Das (M + N)-Bitergebnis von jeder Kernzelle wird in zwei Teile unterteilt:

1. Die am wenigsten signifikanten M Bits werden der D-Eingabe der angrenzenden Kernzelle zugeführt, die ein Ergebnis der gleichen arithmetischen Signifikanz erzeugt
2. Die am meisten signifikanten M Bits werden der C-Eingabe der benachbarten Kernzelle zugeführt, die ein M-Bit signifikanteres Ergebnis erzeugt.

[0012] Die Kernzelle eines 1-Bit·1 Bit-Multiplizierers kann auf eine von drei Arten implementiert werden:

1. Als zwei 4-Eingaben-Nachschlagtabellen (lookup-tables, LUTs), wobei jede A, B, C und D als Eingaben aufweist und eines der beiden Ausgabebits als Ausgabe erzeugt.
2. Als Zwei-Eingabe-UND-Gatter zum Berechnen (A·B) zum Füttern eines Volladdierers, der das Ergebnis zu C und D addiert. Dies setzt einen 2-Eingaben-LUT und zwei 3-Eingaben-LUTs voraus.
3. Als Volladdierer zum Berechnen von (A + C + D), zum Füttern eines Multiplexers, der entweder dieses Ergebnis oder D der Ausgabe zuführt, abhängig von B.

[0013] Jede dieser Lösungen kostet mehr Ressourcen als nötig wäre, um einfach die Volladdition durchzuführen. Multiplizierer sind folglich teuer (in Bezug auf die Siliziumfläche, und folglich in Bezug auf die tatsächlichen Kosten) in FPGA-Strukturen. Jeglicher Ansatz in Vorrichtungen dieses allgemeinen Typs, der die Dichte der Multiplizierer in einem Verarbeitungsarray erhöhen kann, kann in Bezug auf die Reduktion der Kosten äußerst vorteilhaft sein. Die Offenbarung der EP-A-0 833 244 stellt eine feldprogrammierbare Vorrichtung dar, die eine Multiplikation mit 1 × 1 multiplizierenden Zellen ausführt.

[0014] Entsprechend stellt die Erfindung ein Gerät und ein Verfahren zum Multiplizieren einer ersten Zahl mit einer zweiten Zahl gemäß den unabhängigen Ansprüchen 1 und 10 zur Verfügung, unter Verwendung eines Arrays von Verarbeitungsvorrichtungen. Jede der Verarbeitungsvorrichtungen weist eine Vielzahl von Dateneingaben, eine Vielzahl von Datenausgaben und eine Instruktionseingabe zum Steuern der Funktion der Ver-

arbeitungsvorrichtung auf, wobei die Verarbeitungsvorrichtungen und eine Eingabe für die erste Zahl und eine Eingabe für die zweite Zahl durch eine frei konfigurierbare Verknüpfung verknüpft sind, und wobei jede Verarbeitungsvorrichtung ein Partialprodukt berechnet, zum Multiplizieren eines oder mehrerer Bits der ersten Zahl mit einem oder mehreren Bits der zweiten Zahl, und bei jeder Verarbeitungsvorrichtung: der Wert, der bei der Instruktionseingabe empfangen wird, durch ein oder mehrere Bits der ersten Zahl bestimmt ist; die Dateneingaben durch m Bits der zweiten Zahl, eine Summationseingabe zum Summieren aller Partialprodukte gleicher Signifikanz, und, wenn angebracht, eine Übertragseingabe zum Addieren eines Übertrags eines weniger signifikanten Partialprodukts zur Verfügung gestellt werden; und Datenausgaben als Summationsausgaben vorgesehen sind, die die am Wenigsten signifikanten m Bits des Partialprodukts und eine Übertragsausgabe umfassen, die jegliche signifikanteren Bits des Partialprodukts enthält.

[0015] Die vorliegende Erfindung betrifft Vorrichtungen und Architekturen, welche eine frei verknüpfbare Zusammenschaltung umfassen, wobei es im allgemeinen möglich ist (außer möglicherweise für bestimmte spezielle Fälle), jegliche Eingabe und Ausgabe miteinander zu verbinden. Die üblichste Architektur dieses allgemeinen Typs ist ein feldprogrammierbares Gate-Array (FPGA). Architekturen, bei denen bestimmte begrenzte Auswahlmöglichkeiten von Verknüpfungen zwischen Eingaben und Ausgaben, oder bestimmten Eingaben und Ausgaben, möglich sind, sind hier unbeachtlich und fallen nicht in den Rahmen des Begriffs "frei verknüpfbar".

[0016] Es ist den Erfindern der vorliegenden Erfindung bekannt, dass bei einem konventionellen Multiplizierer-Design das erforderliche Ergebnis von jeder Kernzelle entweder (A + C + D) oder D ist, abhängig von dem Wert von B. (Wenn B Null ist, dann ist (A·B) Null, und auch C wird Null, weil B auch Null ist in der Kernzelle, die C erzeugt). Mit anderen Worten, ein Volladdierer wäre ausreichend, um das benötigte Ergebnis zu berechnen, wenn seine Funktion in Abhängigkeit des Wertes von B gesteuert werden könnte. In einem konventionellen FPGA wird die Funktion jeder Logikeinheit zum Zeitpunkt der Konfiguration festgelegt, und eine solche datenabhängige Steuerung der Funktionalität ist nicht möglich. Mit der Struktur, die oben erläutert ist, ist eine solche Steuerung möglich und ein dichterer Multiplizierer wird erreicht.

[0017] Spezielle Ausführungsformen der Erfindung sind unten beispielhaft mit Bezug auf die beigefügten Zeichnungen beschrieben:

[0018] Fig. 1 zeigt einen Teil eines Prozessor-Arrays, bei welchem Ausführungsformen der Erfindung verwendet werden können, wobei sechs Schaltabschnitte und die Orte von sechs arithmetischen Logikeinheiten illustriert sind;

[0019] Fig. 2 ist ein Diagramm eines Teils der in Fig. 1 gezeigten Anordnung in größerem Maßstab, wobei einer der Schaltabschnitte und eine der lokalen arithmetischen Logikeinheiten illustriert werden;

[0020] Fig. 3 zeigt einen Puffer und ein Register, die in jedem Schaltabschnitt verwendet werden können;

[0021] Fig. 4a zeigt ein Blockdiagramm, das eine einzelne arithmetische Logikeinheit für die Verwendung in dem Array von Fig. 1 illustriert; und Fig. 12b zeigt schematisch eine Bitscheibe dieser einzelnen arithmetischen Logikeinheit;

[0022] Fig. 5a und Fig. 5b zeigen die grundlegende Struktur eines kombinatorischen Multiplizierers;

[0023] Fig. 6a und Fig. 6b zeigen konventionelle Ansätze in Bezug auf die Implementierung eines Multiplizierers in einem Verarbeitungselement eines Verarbeitungsarrays;

[0024] Fig. 7 zeigt eine Implementierung eines Multiplizierers in einem Verarbeitungselement eines Verarbeitungsarrays;

[0025] Fig. 8 zeigt eine Implementierung eines Multiplizierers in einem Verarbeitungselement eines Verarbeitungsarrays entsprechend einer Ausführungsform der Erfindung;

[0026] Fig. 9a zeigt einen Multiplizierer, wie in Fig. 7 oder Fig. 8 gezeigt, mit einer diagrammmäßigen Darstellung der zusätzlichen Bits, die benötigt werden, um jedes der Partialprodukte zur vollen Länge des Ergebnisses zu erweitern;

[0027] Fig. 9b zeigt vier Multiplizierer, wie in Fig. 7 oder Fig. 8 gezeigt, in einer Anordnung, die zur Vorzeichenerweiterung angepasst ist;

[0028] Fig. 10a und Fig. 10b zeigen Arraymultiplizierer mit Multiplizierzellen, wie in den Fig. 7 oder Fig. 8 gezeigt, die jeweils für den Gebrauch mit vorzeichenbehafteten und vorzeichenlosen Multiplikanden angepasst sind;

[0029] Fig. 11a und Fig. 11b zeigen Linearmultiplizierer, die Multiplizierzellen, wie in Fig. 7 oder Fig. 8 gezeigt, verwenden für einen seriellen Multiplikator und einen parallelen Multiplikanden bzw. einen parallelen Multiplikator und einen seriellen Multiplikanden; und

[0030] Fig. 12a und Fig. 12b zeigen jeweils die Folge von Operationen und die schematische Struktur für einen Seriell-Seriell-Multiplizierer, der die Linearmultiplizierer aus **Fig.** 11 und Fig. 11b verwendet.

[0031] Ausführungsformen der Erfindung werden im Kontext der CHESS-Architektur beschrieben, die in der internationalen Patentveröffentlichung WO-A-98/33276 beschrieben ist. Eine kurze Beschreibung der relevanten Aspekte der Architektur und der Mechanismen zum Weitergeben der Instruktionen zu den Verarbeitungselementen ist wiedergegeben. Die konventionelle Herangehensweise an die Konstruktion eines kombinatorischen Multiplizierers wird dann beschrieben, zusammen mit der Anwendung dieser konventionellen Herangehensweise an ein CHESS-artiges Array. Ausführungsformen, die die ersten und zweiten Aspekte der Erfindung in einem CHESS-artigen Array verwenden, werden nachfolgend beschrieben.

[0032] In der folgenden Beschreibung werden die Begriffe "horizontal", "vertikal", "Nord", "Süd", "Ost" und "West" verwendet, um das Verständnis der relativen Richtungen zu erleichtern, aber ihre Verwendung impliziert keinerlei Einschränkungen der absoluten Orientierung der Ausführungsform der Erfindung.

[0033] Das Prozessorarray für die Ausführungsform der Erfindung ist in einem integrierten Schaltkreis vorgesehen. Auf einer Ebene wird das Prozessorarray durch ein rechteckiges (und bevorzugt quadratisches) Array von "Fliesen" **10** gebildet, von denen eine durch eine dicke Linie umrandet in Fig. 1 dargestellt ist.

[0034] Jede geeignete Zahl von Fliesen kann verwendet werden, z. B. in einem 16 × 16-, 32 × 32- oder 64 × 64-Array. Jede Fliese **10** ist rechteckig und in vier Schaltkreisbereiche unterteilt. Diese Fliesen sind bevorzugt logisch quadratisch (um eine Symmetrie der Verbindungen bereitzustellen), obwohl es weniger signifikant ist, dass sie auch physikalisch quadratisch sind (dies könnte vorteilhaft sein, um eine Symmetrie im Zeitablauf bereitzustellen, aber dies ist im allgemeinen wahrscheinlich von geringer Bedeutung). Zwei der Schaltkreisbereiche **12**, die sich auf der Fliese diagonal gegenüber liegen, sind die Orte für zwei arithmetische Logikeinheiten ("ALUs"). Die anderen zwei Schaltkreisbereiche, welche sich auf der Fliese **10** diagonal gegenüber liegen, sind die Orte für ein Paar von Schaltabschnitten **14**.

[0035] Bezugnehmend auf Fig. 1 und Fig. 2 hat jede ALU ein erstes Paar von 4-Bit-Eingaben a, welche direkt mit der ALU verbunden sind, ein zweites Paar von 4-Bit-Eingaben b, welche auch direkt mit der ALU verbunden sind, und vier 4-Bit-Ausgaben f, welche direkt innerhalb der ALU verknüpft sind. Jede ALU hat ein unabhängiges Paar von 1-Bit-Übertragseingaben hci, vci und ein Paar von 1-Bit-Übertragsausgaben co, welche direkt inerhalb der ALU verknüpft sind. Die ALU kann Standardoperationen über die Eingabesignale a, b, hci, vci ausführen, um die Ausgabesignale f, co zu erzeugen, wie z. B. Addition, Subtraktion, UND, NUND, ODER, NODER, XODER, NXODER und Multiplexen, und sie kann optional das Ergebnis der Operation speichern. Die Arbeitsweise einer einzelnen ALU wird unten detaillierter besprochen. Die Anweisungen an die ALUs können jeweils von 4-Bit-Speicherzellen zugeführt werden, deren Werte extern festgelegt werden können, oder sie können über ein Bus-System zugeführt werden.

[0036] Auf den Ebenen, die in Fig. 1 und Fig. 2 gezeigt sind, hat jeder Schaltabschnitt **14** acht Busse, die sich horizontal in ihm ausdehnen, und acht Busse, die sich vertikal in ihm ausdehnen, so dass ein rechteckiges 8 × 8-Array mit 24 Schnittpunkten entsteht, welche in Fig. 2 mit kartesischen Koordinaten nummeriert sind. Alle diese Busse haben eine Breite von vier Bits, mit der Ausnahme des Übertragsbusses vc bei X = 4 und dem Übertragsbus hc bei Y = 3, welche eine Breite von 1 Bit aufweisen. An vielen der Schnittpunkte wird ein programmierbarer 4-Bandschalter **18** bereitgestellt, der selektiv zwei Busse verbinden kann, welche sich Ende an Ende an diesem Kreuzungspunkt treffen, ohne rechtwinklige Verbindung zu dem Bus. An dem Kreuzungspunkt (**4**, **3**) ist ein programmierbarer Schalter **20** vorgesehen, der selektiv die Übertragsbusse vc, vh, die sich an diesem Punkt rechtwinklig schneiden, verbindet.

[0037] Wie in Fig. 2 gezeigt, sind die Busse bs, vco, fs jeweils mit Eingabe b, Ausgabe co und Ausgabe f der ALU nördlich des Schaltabschnitts **14** verknüpft. Außerdem sind die Busse fe, hco, be jeweils mit der Ausgabe f, Ausgabe co und Eingabe b der ALU westlich des Schaltabschnitts **14** verknüpft. Außerdem sind die Busse

aw, hci, fw jeweils mit der Eingabe a, Eingabe ci und Ausgabe f der ALU östlich des Schaltabschnitts **14** verknüpft. Außerdem sind die Busse fn, vci, an jeweils mit der Ausgabe f, Eingabe ci und Eingabe a der ALU südlich des Schaltabschnittes **14** verknüpft.

[0038] Zusätzlich zu diesen Verbindungen sind die Busse vregw, vrege jeweils über programmierbare Schalter **18** mit 4-Bit-Verbindungspunkten vtsw, vtse (durch Kreuze in Fig. 2 gezeigt) im Bereich **12** der ALU nördlich des Schaltabschnittes **14** verbunden. Auch sind die Busse hregs, hregn jeweils durch programmierbare Schalter **18** mit 4-Bit-Verbindungspunkten htse, htne im Gebiet **12** der ALU westlich des Schaltabschnitts **14** verbunden. Außerdem sind die Busse hregs, hregn durch programmierbare Schalter **18** jeweils mit 4-Bit-Verbindungspunkten htsw, htnw im Gebiet **12** der ALU östlich des Schaltabschnitts **14** verbunden. Außerdem sind die Busse vregw, vrege durch programmierbare Schalter **18** jeweils mit 4-Bit-Verbindungspunkten vtnw, vtne im Gebiet **12** der ALU südlich des Schaltabschnitts **14** verbunden.

[0039] Wie außerdem in Fig. 2 dargestellt, haben die Busse hregn, vrege, hregs, vregw jeweils 4-Bit-Verbindungspunkte **22** (dargestellt durch kleine Quadrate in Fig. 2), welche unten in größerem Detail mit Bezugnahme auf Fig. 3 beschrieben werden.

[0040] Wie oben in Bezugnahme auf Fig. 1 und Fig. 2 erwähnt wurde, sind an jedem Schaltabschnitt **14** die Busse hregn, hregs, vregw, vrege jeweils mit 4-Bit-Verbindungen **22** mit einem Register oder Pufferschaltkreis verbunden, und dieser Schaltkreis wird nun genauer mit Bezugnahme auf Fig. 3 beschrieben. Die vier Verbindungen **22** sind jeweils mit entsprechenden Eingaben eines Multiplexers **26** verbunden. Der Multiplexer **26** wählt einen der Eingaben als eine Ausgabe, die einem Register oder Puffer **28** bereitgestellt wird. Die Ausgabe des Registers oder Puffers **28** wird vier Drei-Zustands-Puffern **30s**, **30w**, **30n**, **30e** zugeführt, welche durch die Verbindungen **22** jeweils zu den Bussen hregs, vregw, hregn, vrege zurück verbunden sind. In dem Fall, in dem ein Puffer **28** benutzt wird, wird das 4-Bit-Signal auf einem ausgewählten Bus aus der Gruppe der Busse hregs, vregw, hregn, vrege verstärkt und einem anderen ausgewählten Bus aus der Gruppe der Busse hregs, vregw, hregn, vrege zugeführt. In dem Fall, in dem ein Register **28** verwendet wird, wird das 4-Bit-Signal auf einem ausgewählten Bus der Gruppe der Busse hregs, vregw, hregn, vrege verstärkt und einem ausgewählten Bus aus der Gruppe der Busse hregs, vregw, hregn, vrege nach der nächsten aktiven Taktflanke zugeführt.

[0041] Eine verbesserte Ausführung der Struktur von Fig. 3 ermöglicht es, ein 4-Bit-Signal auf einem ausgewählten Bus aus der Gruppe der Busse hregs, vregw, hregn und vrege für einen anderen Zweck von dem Interbus-Routing zu extrahieren. Eine geeignete Konstruktion und Verbindung von Multiplexer **26** (oder in einer alternativen Anordnung von Puffer **28**) erlaubt das Auswählen eines Wertes, der von dem Verdrahtungsnetzwerk als Ausgabe des Multiplexers **26** oder Puffers **28** (diese Wahlmöglichkeiten sind jeweils als **260** und **280** in Fig. 3 bezeichnet) empfangen wurde, wobei dieser Wert dann verwendet wird, um die Anweisung der ALU, die mit dieser Schaltbox verbunden ist, zu bestimmen. Die Anwendungen dieser Anordnung werden unten weiter diskutiert.

[0042] Die Verwendung des Multiplexers **26** oder Puffers **28** für diesen Zweck bedeutet, dass der Wert, der verwendet wird, um eine Instruktion an die ALU bereitzustellen, auch der Wert ist, der zum Weiterreichen durch das Verdrahtungsnetzwerk bereitgestellt wird. Ein anderer Schaltabschnitt **14** muss verwendet werden, wenn es erwünscht ist, einen anderen Wert zwischen den Verdrahtungen zu übertragen. In vielen Anordnungen wird es jedoch wünschenswert sein, dass der Wert, der zu der ALU weitergeleitet wird, um ihre Instruktion festzulegen, auch der Wert ist, der von einer Verdrahtung zur anderen weitergereicht wird: dies ist angemessen, wenn es gewünscht wird, die gleiche Instruktion einer Anzahl von ALUs zur Verfügung zu stellen, was oft in einer tiefen Verarbeitungs-Pipeline auftritt. Eine alternative Ausführung, die nicht gezeigt ist, verwendet zwei oder mehr Paare von Multiplexern **26** und Puffern **28**: in diesem Fall kann ein Paar von Multiplexern/Puffern einer Instruktionseingabe der assoziierten ALU zugeordnet werden, während das andere Paar oder die Paare für das Routing verwendet werden können.

[0043] Es soll darauf hingewiesen werden, dass, obwohl Bitbreiten, Größen von Schaltabschnitten und Größen von Arrays erwähnt worden sind, diese Werte, wo angezeigt, verändert werden können. Obwohl die programmierbaren Schalter **16**, **18**, **20** als an bestimmten Orten des jeweiligen Schaltabschnitts **14** angeordnet beschrieben wurden, können, wenn erwünscht und notwendig, auch andere Orte verwendet werden. Die Prinzipien der CHESS-Architektur sind auf dreidimensionale Arrays anwendbar, z. B. durch Bereitstellen eines Stapels der oben beschriebenen Arrays, bei dem die Schaltabschnitte in benachbarter Ebenen zueinander gestaffelt sind. Es ist möglich, dass jeder Stapel nur zwei Ebenen, bevorzugt jedoch wenigstens drei Ebenen enthält, und die Zahl der Ebenen ist bevorzugt ein Vielfaches von zwei.

[0044] Die Struktur der ALU, die in dieser Ausführungsform der Erfindung verwendet wird, wird im folgenden mit Verweise auf **Fig. 4a** und **Fig. 4b** beschrieben. Wie in **Fig. 4a** dargestellt, hat die ALU vier Eingaben, A, B, I und C_{in}, und zwei Ausgaben, F und C_{out}. A, B, I und F sind alle vier Bit breit und mit der allgemeinen Verknüpfung durch benachbarte Schaltblöcke verbunden, wie oben beschrieben für a, b und f. Die Eingabe für I ist aus dem Multiplexer **26**, der in **Fig. 3** dargestellt ist, extrahiert. C_{in} und C_{out} sind beide 1 Bit breit und mit einer stärker eingeschränkten Verknüpfung verbunden, wie ebenfalls oben beschrieben. A und B stellen die Operanden für die ALU und F die Ausgaben zur Verfügung. C_{in} und C_{out} stellen die Übertragsfunktion zur Verfügung, sind aber auch für die Steuerung wichtig. I stellt eine Instruktionseingabe zur Verfügung, welche die funktionale Operation der ALU bestimmt: dies steht im Gegensatz zu einem Standard-FPGA, bei welchem die funktionalen Einheiten durch einen Satz von Speicherbits gesteuert wird. Die Bedeutung dieses Merkmals und die Mechanismen, die zur Verfügung gestellt werden, um Instruktionseingaben von dem Verdrahtungsnetzwerk zu der ALU zu leiten, werden unten besprochen.

[0045] Die ALU hat vier Hauptkomponenten:
den ALU-Datenpfad, welcher aus vier identischen Bitscheiben besteht;
den Instruktionsdecoder;
die Übertrags-/Steuerungseingabe-Aufbereitungslogik; und
die Schaltblock-Programmierschnittstelle (in anderen Ausführungsformen der Erfindung muss diese nicht in der ALU selbst vorliegen, jedoch erlaubt die Anwesenheit dieses Merkmals in der ALU, dass die ALU in einem Nachschlagtabellen-Betrieb verwendet wird).

[0046] **Fig.** 4 zeigt ein Blockdiagramm einer einzelnen Bit-Scheibe der ALU.

[0047] Die zwei "Eingabepuffer" **202** und **203** sind nichts weiter als ein Mittel, eine elektrische Verbindung zum Routing-Netzwerk bereitzustellen. In dieser Architektur gibt es kein adressierbares Eingaberegister (und entsprechend keine Registerdatei): die Operanden werden der Funktionseinheit **201** der ALU in jedem Zyklus vom selben Ort (dem Verdrahtungsnetzwerk) zur Verfügung gestellt.

[0048] Die Funktionseinheit **201** arbeitet als eine Nachschlagtabelle (LUT), welche eine Boolesche Funktion, U, der beiden Eingaben A und B erzeugt. Die exakte Funktion wird durch die vier Steuersignale (L_3, L_2, L_1, L_0) bestimmt und erzeugt die Karnaugh-Tafel, die in Tabelle 1 gezeigt ist:

U =

	A	0	1
B			
0		L_0	L_1
1		L_2	L_3

Tabelle 1: Karnaugh-Tafel für ALU-Bitscheibe

[0049] Die Erzeugung der Steuersignale L_i wird unten weiter besprochen.

[0050] "Summenerzeugung" **204** stellt eine Summenausgabe zur Verfügung, die durch ein XODER von U und C_{in} abgeleitet ist:

Summe = U XODER C_{in}

[0051] C_{out} wird durch Erzeugen eines Übertrags **204** gemäß der folgenden Booleschen Gleichung erzeugt:

P = U ODER L_4
G = A ODER L_5
C_{out} = WENN P DANN C_{in} SONST G

wobei P als fortpflanzende Funktion und G als erzeugende Funktion betrachtet werden kann. Die Signale L_i werden wiederum auf eine Weise erzeugt, die im Anschluß beschrieben wird.

[0052] Das Ausgaberegister **206** verriegelt wahlweise die Summenausgabe, wobei diese Option unter der Steuerung des ALU-Programmierspeichers auswählbar ist. Wahlweise kann eine ähnliche Verriegelungsanordnung für die Übertragsausgabe zur Verfügung gestellt werden. Diese Merkmale sind für den Gebrauch in tiefen Pipelines vorteilhaft, bei denen die gleiche Operation möglicherweise synchron in mehreren ALUs in zeitgesteuerter Weise ausgeführt werden muss.

[0053] Eine breite Vielfalt von unterschiedlichen möglichen Bit-Scheiben können eingesetzt werden. Die Wahl des ausgewählten Bit-Scheiben-Typs in einer gegebenen Architektur kann eine Funktion des Typs von Instruktion sein, für die die Architektur insgesamt als am effizientesten arbeitend vorgesehen ist. Es ist klarerweise erstrebenswert, die Verwendung einer Vielzahl von Funktionen, die als nützliche Bausteine für komplexere Operationen dienen können, zu ermöglichen. Andere Merkmale sind ebenfalls wünschenswert. Ein wünschenswertes Merkmal ist die Fähigkeit, einige Bits von ihrer normalen Funktion abzuzweigen, um die Steuerung über andere Schaltkreiselemente zu ermöglichen. Ein weiteres wünschenswertes Merkmal ist die Fähigkeit, eine festgelegte Instruktion für irgendeine der ALUs zu speichern, die in einer bestimmten Konfiguration keine dynamische Instruktionsschaltung benötigen. Es ist ebenfalls wünschenswert, dass ein geeigneter Anfangszustand vorhanden ist, um die ALU als einen Lese/Schreibanschluss für die Schaltbox (oder die Nachschlagtabelle) nutzbar zu machen. Für diese Anwendung werden keine spezifischen Bit-Scheiben beschrieben: ein Beispiel für geeignete Bit-Scheiben ist in der internationalen Patentveröffentlichung WO-A-98/33276 dargelegt. Zum Zwecke der vorliegenden Patentanmeldung ist es nur notwendig, dass die ALU in der Lage ist, die Funktionen zu unterstützen, die in den verschiedenen unten beschriebenen Multiplizierer Implementationen beschrieben sind.

[0054] Die Herkunft der Instruktions-Bits für die ALU wird jedoch diskutiert. Ein Element der CHESS-Architektur, die hier beschrieben ist, ist die Fähigkeit, eine Instruktion für eine Funktionseinheit als Ausgabe einer anderen Funktionseinheit zu erzeugen.

[0055] Eingabesignale, die dynamische Instruktionen I (4-Bit-Instruktionen, die durch eine andere ALU im Array erzeugt worden sind, oder optional von einem Speicher, der dem Verdrahtungsnetzwerk zugängig ist, erhalten wurde) enthalten, werden von Verknüpfungen zum Verdrahtungsnetzwerk empfangen: diese können durch Multiplexer **26** (siehe **Fig. 3**) wie oben erläutert gewonnen werden. Es ist erwünscht, dass mehrere Wahlmöglichkeiten zur Verfügung stehen, was durch die Verwendung eines oder mehrerer zusätzlicher ALUs in der Multiplexer-Konfiguration erreicht werden kann.

[0056] Die 4-Bit-Ausgabe einer ALU kann folglich als dynamische Instruktionseingabe I für eine weitere ALU verwendet werden. Der Übertragsausgang einer ALU kann auch als Übertragungseingang für eine andere ALU verwendet werden, und dies kann beim Bereitstellen dynamischer Instruktionen ausgenutzt werden. Es gibt drei grundsätzliche Möglichkeiten, wie die Operation einer ALU dynamisch variiert werden kann:

1. C_{in} kann verwendet werden, um zwischen zwei Versionen einer Funktion zu multiplexen, wobei die Instruktions-Bits I konstant bleiben.
2. Die Instruktions-Bits I können verändert werden, während C_{in} gleich bleibt.
3. Sowohl die Instruktion als auch der Wert C_{in} können verändert werden.

[0057] Die Anwendung dieses dynamischen Instruktions-Bereitstellungsmerkmals bei einer Implementation des Multiplizierers wird unten beschrieben.

[0058] Wie oben angedeutet, ist ein konventioneller kombinatorischer Multiplizierer üblicherweise als repetitives Array von Kernzellen aufgebaut, wobei jede Zelle einige Bits (z. B. M Bits) des Multiplikanden A mit einigen Bits (z. B. N Bits) des Multiplikators B multipliziert, um ein (M + N)-Bit-Partialprodukt zu erzeugen. Jede Kernzelle muss auch in der Lage sein, die Funktion ((A·B) + C + D) zu errechnen, wobei die D-Eingabe verwendet wird, um alle Partialprodukte zu summieren, die die gleiche Signifikanz haben, und die C-Eingabe verwendet wird, um Überträge von weniger signifikanten Partialprodukten zu addieren. Das (M + N)-Bit-Ergebnis von jeder Kernzelle ist in zwei Teile aufgeteilt: die am wenigsten signifikanten M Bits werden zur D-Eingabe der benachbarten Kernzelle geleitet, die ein Resultat der gleichen arithmetischen Signifikanz erzeugt; und die signifikantesten N Bits werden zur C-Eingabe der benachbarten Kernzelle geleitet, die ein M Bit signifikanteres Ergebnis erzeugt. Wie vorher angedeutet, ist das benötigte Ergebnis von jeder Kernzelle entweder (A + C + D) oder D, abhängig von dem Wert von B.

[0059] Die zugrunde liegende Struktur eines Multiplizierers, die ausgelegt ist wie in einer Langschriftmultiplikation, ist in **Fig. 5a** gezeigt. Der Multiplikand X ist in mehrere Sätze von Bits x_m aufgeteilt, und der Multiplikator Y ist in mehrere Sätze von Bits y_n aufgeteilt. Der Multiplizierer ist aus einem Array von Grundzellen aufgebaut,

von denen jede $((x_m \cdot y_n) + c + d)$ für irgendwelche m und n berechnet. Das Resultat wird in zwei Teile unterteilt, lo und hi, welche jeweils die mehr oder weniger signifikanten Teile des Ergebnisses darstellen. Das lo-Ergebnis wird zu der d-Eingabe der darunterliegenden Grundzelle geleitet, und das hi-Ergebnis (welches größere arithmetische Signifikanz hat) wird zur c-Eingabe der Grundzelle auf der linken Seite geleitet.

[0060] Die gleiche Multipliziererstruktur ist um 45 Grade geneigt in Fig. 5b dargestellt. Die geneigte Anordnung von Fig. 5b zeigt, wie der Multiplizierer auf ein rechteckiges Array passt, wobei der Multiplikator Y von links zugeführt wird. Dieses Layout ist für eine effektive physikalische Implementierung geeignet.

[0061] Alternative Methoden zum Implementieren solch konventioneller Herangehensweisen in einem feldprogrammierbaren Array, welche vom Typ eines CHESS oder eines konventionellen FPGA sein können, werden nun beschrieben.

[0062] Fig. 6a zeigt eine Implementation einer Multiplizierergrundzelle in einer Struktur vom Typ CHESS, wobei die konventionelle Herangehensweise an die Konstruktion von Multiplizierern angewandt wird. Die Multiplikation der Bits x_m und y_n wird durch ein UND-Gatter **501** ausgeführt. Ein Volladdierer **502** summiert das Resultat dieser partiellen Multiplikation mit den Eingaben c und d, um das lokale lo- und hi-Resultat zu erzeugen. Eine alternative Implementierung ist in Fig. 6b gezeigt. Hier wird die Addition durch den Addierer **503** durchgeführt, so als sei das Multiplikator-Bit y_m gleich 1. Wenn y_m gleich 0 wäre, so würde der Multiplexer **504** verwendet werden, um dem lo-Ergebnis den Wert des hereinkommenden d-Signals aufzuzwingen, was in diesem Fall das korrekte Ergebnis wäre. Es muss beachtet werden, dass, obwohl es keinen Übertrag C in dem Fall, dass y_m gleich 0 ist, geben sollte, ein Verarbeitungselement einen scheinbaren Übertrag an der hi-Ausgabe fortpflanzt. Weil jedoch y_m gleich 0 für das Verarbeitungselement, das den Übertrag empfangen wird, ist (die nächste Stufe der Signifikanz für das gleiche Multiplikator-Bit), wird dieses Signal keinen Effekt auf das Gesamtergebnis haben.

[0063] Beide Multiplizierzellen, die in den Fig. 6a und Fig. 6b gezeigt sind, erfordern einen etwas größeren Aufwand zur Implementierung als die Addierer alleine. Typischerweise kostet in einem feldprogrammierbaren Array der Addierer eine Verarbeitungszelle, während das UND-Gatter (von Fig. 6a) oder der Multiplexer (von Fig. 6b) eine weitere Verarbeitungszelle kostet.

[0064] Von dem Erfinder der vorliegenden Erfindung wurde ermittelt, dass im Falle von CHESS und vergleichbaren Architekturen ein Multiplizierer in einer einzigen Verarbeitungszelle implementiert werden kann. Dies kann erzielt werden, indem man die Funktion einer Funktionseinheit auf datenabhängige Weise bestimmen läßt. Die folgende Herangehensweise verwendet eine Variante der "Multiplexer"-Option, die in Fig. 6b gezeigt ist. Die Multiplizierzelle, die in Fig. 7 gezeigt ist, und diese Zelle können so gelegt werden, dass sie einen vollständigen Multiplizierer bilden. Die funktionale Einheit wird angewiesen, entweder das Resultat der Addition zu erzeugen, oder den Wert der d-Eingabe weiterzureichen, abhängig vom Wert von y_m. Dies erfordert, dass die Instruktionseingabe I (bezeichnet mit **510**) an die ALU in gewisser Weise von y_m unabhängig ist: dies kann z. B. erreicht werden, indem die zusätzliche Logik **505** verwendet wird, um y_m für die Instruktionseingaben zu decodieren, die benötigt werden, um die Funktion zu bestimmen, die für I-Eingabe, die auf dem Verarbeitungselement **506** gezeigt ist, angedeutet ist. Diese zusätzliche Logik **505** muß nicht ausführlich sein: für einen großen Arraymultiplizierer wird folglich jeglicher Overhead, der von der zusätzlichen Logik vorgesehen wird, klein sein im Vergleich zu dem Faktor zwei in der Reduktion der Fläche der Multiplizierzelle.

[0065] Fig. 8 zeigt, wie die Fähigkeit, die Funktion einer funktionalen Einheit auf datenabhängige Weise zu verändern, verwendet werden kann, um die Multiplizierdichte weiter zu vergrößern. In diesem Fall wird wiederum ein einzelnes Prozessorelement (mit der funktionalen Einheit **507**) verwendet, aber es werden in der Operation zwei Bits, anstatt von einem alleine, des Multiplikators für ein einziges Verarbeitungselement verwendet.

[0066] Zwei Bits des Multiplikators werden der Decodierlogik **505** zugeführt, welche Instruktionseingaben mit den vier folgenden Werten produziert: $(d + x_m + c)$; (d); $(d - x_m - c)$ und $(d - 2x_m - c)$. Jede dieser Instruktionen addiert oder subtrahiert ein anderes Vielfaches des Multiplikanden, wobei die Auswahl der Instruktionen entsprechend der Werte der zwei Bits y_n, y_{n+1} auswählbar ist.

[0067] Zu beachten ist, dass in dem oben dargestellten Beispiel die Instruktionen, die übergeben worden sind, Resultate von –2, –1, 0 und +1 ergeben, und nicht etwa das 0-, 1-, 2- und 3-fache des Multiplikatorbits. Dies kann mit einem Vorverarbeitungsschritt korrigiert werden, so dass das richtige Resultat an den Schaltkreisausgaben erscheint.

[0068] Eine beispielhafte Vorverarbeitungsprozedur, um dies zu erreichen, ist unten zusammengefasst.

1. Der Multiplikand ist zu einer Basis-(–2)-Darstellung konvertiert, bei denen die Bits die Signifikanz haben:

... – 32 + 16 – 8 + 4 – 2 + 1

[0069] Betrachtet man jedes Paar von Bits, haben die vier Kombinationen die folgenden arithmetischen Werte in der Basis (–2):

Bit-Werte	arithmetische Werte
00	0
01	+1
10	–2
11	–1

[0070] Diese vier Vielfachen des Multiplikanden sind die vier Vielfachen, die das Array von Fig. 7 der Anwendung zu der Partialsumme in jeder Stufe hinzuaddiert.

[0071] 2. Um den Basis-2-Multiplikanden zur Basis-(–2) zu konvertieren, erzeugen wir ein konstantes Bit-Muster M, das das Bit-Muster (**10**) in jedem Paar von Bits enthält. Folglich ist das am wenigsten signifikante Bit von M gleich 0 und die alternierenden signifikanteren Bits sind abwechseln 1 und 0. Dann ist:

Basis-(–2)-Multiplikand = ((Basis 2-Multiplikand) + M exoder M

[0072] Die Operationen sind hierbei:

+	arithmetische Addition
exoder	Bit-für-Bit exklusiv ODER

[0073] Der Effekt des exoder ist eine Inversion alternierender Bits des Ergebnisses der Addition, wobei das am wenigsten signifikante Bit nicht invertiert wird.

3. Um diesen Vorgang zu verstehen, betrachte man die zwei möglichen Werte eines der Bits eines Multiplikanden, für welchen M gleich 1 ist. Dies sind die Multiplikanden-Bits, die eine negative Signifikanz in der Basis-(–2) haben.

Bit = 0. In diesem Fall ist die Null in der Basis (–2) genau der gleiche Wert wie die Null in der Basis 2, so dass keine Änderung notwendig ist. Ein Addieren des 1 Bit von M macht aus dem Summen-Bit eine 1, und das exoder würde dies dann invertieren, um ein Null-Ergebnis zu erzeugen – d. h. es ist der Originalwert, wie benötigt.

Bit = 1. Man betrachte ein Bit mit dem Wert 2 in der Basis 2, und (–2) in der Basis (–2) – die anderen alternierenden Bits verhalten sich auf gleiche Weise. Wenn das Bit 1 ist, subtrahieren wir durch das Interpretieren des Bit-Wertes (–2) statt (+2) im Ergebnis 4 von dem Wert des Multiplikanden. Wir gleichen dies aus, indem wir 4 in 2 Schritten zurückaddieren:

– Wir addieren M zu dem Multiplikanden. Das 1-Bit in dieser Bit-Position ist eine zusätzliche 2 wert, welches einen Übertrag zu dem Bit im Wert von (+4) erzwingt, und setzt den Bit-Wert (–2) auf 0.

– Wir führen die Operation exoder mit M durch. Dies hat keinen Einfluss auf den Bitwert (+4), aber invertiert den Bitwert (–2) wieder zu 1- seinem ursprünglichen Wert.

Der Nettoeffekt besteht darin, dass wir (+4) zu dem Wert des Multiplikanden hinzuaddiert haben, um die Neuinterpretation des (+2)-Bits als (–2) auszugleichen. Das resultierende Bitmuster ist die Repräsentation des Multiplikanden in der Basis (–2), wie es für den Multiplikator benötigt wird.

[0074] Folglich können Instruktionen zur Verfügung gestellt werden, welche das richtige Resultat bei den jeweiligen Ausgaben für jede dieser Optionen zur Verfügung stellen, in dem Sinne, dass die Instruktionen den Effekt eines Weiterreichens der Datenausgabewerte haben, die das 0-, 1-, 2- und 3-fache der m-Bits des Multiplikanden repräsentieren, die in dem Verarbeitungselement gemäß der zwei Bits des Multiplikators gehandhabt werden. Die Decodierlogik **505** muss in diesem Fall ein wenig stärker ausgedehnt sein als im Falle der Fig. 7, aber da außerdem um den Faktor zwei weniger Verarbeitungsgrundelemente benötigt werden, um den Arraymultiplizierer bereitzustellen, sind die zusätzlichen Einsparungen für einen Multiplizierer jeder Größe äußerst bedeutend.

[0075] Die Anordnung, die oben mit Bezug auf Fig. 8 beschrieben worden ist, kann weiter verbessert werden durch das Verwenden zusätzlicher Eingabebits. Insbesondere, wenn eine fünfte Instruktion verwendet wird, die zu einem weiteren anderen 4-fachen des Multiplikanden zugeordnet ist (insbesondere $d + 2x_m + c$), wird es möglich, den Decoder ohne Abhängigkeiten zwischen den unterschiedlichen Multipliziererstufen zu implementieren. Solche Abhängigkeiten führen zu unerwünschten sich langsam ausbreitenden Übertragsverzögerungseffekten. Letzteres kann erreicht werden, indem drei Bits eines Multiplikanden, anstelle von zwei, betrachtet werden, und ein Radix-4 modifizierter Booth-Algorithmus implementiert wird (z. B. beschrieben von Koren, I., in "Computer Arithmetic Algorithms", 1993, Prentice-Hall Inc., Englewood Cliffs, Seiten 99–103). Das y_n- und y_{n+1}-Bitpaar wird durch y_{n-1} als weitere Eingabe ergänzt, und es werden Instruktionen gemäß der Tabelle 2 unten erzeugt.

y_{n+1}	y_n	y_{n-1}	Instruction
0	0	0	(d)
0	0	1	$(d + x_m + c)$
0	1	0	$(d + x_m + c)$
0	1	1	$(d + 2x_m + c)$
1	0	0	$(d - 2x_m - c)$
1	0	1	$(d - x_m - c)$
1	1	0	$(d - x_m - c)$
1	1	1	(d)

Tabelle 2: Instruktionen des Radix-4 modifizierten Booth-Algorithmus für Multiplizierer vierfacher Dichte

[0076] Tabelle 2 kann wie folgt verstanden werden. Wenn das signifikanteste Bit eines Bitpaars (z. B. y_{n+1}) 1 ist, dann trägt es –2 zu seinem eigenen Bitpaar und +1 zu dem nächsten signifikantesten Bitpaar bei (mit dem Wert +4 in Bezug auf sein eigenes Bitpaar). Wenn folglich y_{n-1} des nächsten weniger signifikanten Bitpaars 1 ist, trägt es +1 zu dem gegenwärtigen Bitpaar bei. Für jede Reihe in der Tabelle ergibt $(-2y_{n+1} + y_n + y_{n-1})$ das Vielfache des Multiplikanden, das an der gegenwärtigen Bit-Position hinzugefügt werden muss, und man kann erkennen, dass die Instruktionen dieser Gleichung entsprechen.

[0077] Eine Nachschlagtabelle für jedes Paar von Multiplikatorbits kann folglich eine Multiplikatorrecodierung implementieren, mit der Eingabe von diesen Bits und dem benachbarten weniger signifikanten Bit y_{n-1} (oder 0 in dem Fall des am wenigstens signifikanten Bitpaars). In der CHESS-Architektur, beschrieben mit Hinweis auf Fig. 1 bis 4, kann y_{n-1} in einem anderen Halbbit als y_n und y_{n+1} sein, in welchem Fall der Multiplikator um ein Bit verschoben werden muss, so dass alle drei Bits der Nachschlagtabelle zugänglich sind. Dies kann erreicht werden, indem eine ALU verwendet wird, die als ein Addierer für jede Stelle im Multiplikator y wirkt, um den Wert (y + y) über den Rand des Arrays zu berechnen, von welchem der Multiplikator y bereitgestellt wird.

[0078] Man beachte, dass, wenn y_{n+1} gleich 1 ist, ein Übertrag zum nächsten signifikanteren Bitpaar fortgepflanzt wird. Wenn das vorliegende Bitpaar das signifikanteste Bitpaar wäre, würde der Übertrag verloren gehen, was zu einem inkorrekten Resultat führen würde. Um diesem vorzubeugen, muss ein Multiplikator ohne Vorzeichen erweitert werden, um sicherzustellen, dass er wenigstens ein am meisten signifikantes Nullbit aufweist – wie unten diskutiert wird, wird diesem Ansatz für einen Multiplikator mit Vorzeichen nicht gefolgt.

[0079] Das Übertragsbit für jede ALU wird addiert oder subtrahiert, abhängig davon, ob die Instruktion eine Addition oder eine Subtraktion ist. Das impliziert, dass jeder Übertrag in zwei ALUs erzeugt und aufgenommen werden muss, die entweder beide Additionen oder beide Subtraktionen ausführen. Da die Wahl zwischen Addition und Subtraktion durch die Instruktion bestimmt wird, impliziert dies, dass die Überträge durch das Array in die gleiche Richtung transportiert werden müssen wie die Instruktionen. Wenn die Signifikanz der Überträge nicht instruktionsabhängig wäre, könnten die Überträge entweder horizontal oder vertikal durch das Array transportiert werden, und die Wahl könnte so gemacht werden, dass sie die parallele Verarbeitung des Arrays vereinfacht.

[0080] Dieses Schema setzt voraus, dass der Multiplikand mit 2 vormultipliziert ist, um durch das Array trans-

portiert zu werden. Dieses erfordert einen Addierer/Schieber pro Ziffernposition über den Rand des Arrays. Es setzt außerdem voraus, dass das Array in der Breite des Multiplikanden ausgeweitet wird, um die Breite des vormultiplizierten Multiplikanden aufzunehmen.

[0081] Für die Implementation der Architektur, die in Fig. 1 bis 4 beschrieben ist, in welcher die gesamte Verdrahtung entlang der 4-Bit-Breite ausgerichtet ist, erfordert dieses Schema, dass der Multiplikand mit 2, 4 und 8 vormultipliziert ist, um durch das Array transportiert zu werden. Dies setzt drei Addierer/Schieber pro Ziffernstelle über den Rand des Arrays voraus. Es ist auch notwendig, dass das Array durch eine 4-Bit-Ziffer in der Multiplikandenbreite ausgedehnt wird, um die Breite der vormultiplizierten Multiplikanden aufnehmen zu können. Wenn der Multiplikand vorzeichenbehaftet ist, dann wird eine Vorzeichenerweiterung notwendig sein, um die zusätzliche Breite auszufüllen (Vorzeichenerweiterung wird weiter unten erläutert).

[0082] Multiplikatoren, wie oben beschrieben, können für die Verwendung mit vorzeichenbehafteten Zahlen angepasst werden. Die Gesamtstruktur des Multipliziererarrays wird von vorzeichenbehafteten Multiplikatoren und vorzeichenbehafteten Multiplikanden unterschiedlich beeinflusst, wie unten erläutert wird. Für den Zweck der folgenden Diskussion wird vorausgesetzt, dass die behandelte Zahl als Komplement von zwei dargestellt ist: in der normalen Form dieser Darstellung hat eine positive Zahl Y einen unveränderten Wert (von Y), während eine negative Zahl -Y den Wert R -Y zugeteilt bekommt, wobei R 2X + 1 ist (wobei X der maximale zulässige Wert von Y ist).

[0083] Das signifikanteste Bit einer Zahl ist daher -2^n "Wert", und nicht 2^n, wie bei einer nicht vorzeichenbehafteten Zahl. Dies bedeutet, dass in einem Multipliziererarray, das vorzeichenbehaftete Multiplikatoren verarbeitet, das Partialprodukt, das vom signifikantesten Bit des Multiplizierers erzeugt wird, eine negative arithmetische Signifikanz aufweist, und daher von dem Gesamtresultat abgezogen anstatt addiert werden muss, wie im Fall ohne Vorzeichen. Jedoch haben die fünf Instruktionen des oben beschriebenen Booth-Recodierverfahrens bereits die gewünschten Eigenschaften, und entsprechend handhaben sie einen vorzeichenbehafteten Multiplikator automatisch. Im Gegensatz zu dem Fall ohne Vorzeichen muss der Multiplikator nicht mit dem signifikantesten Bit gleich Null erweitert werden – stattdessen ist das signifikanteste Bit ein Vorzeichenbit.

[0084] Für einen vorzeichenbehafteten Multiplikanden wird ein zusätzlicher Schaltkreis benötigt, wie in Fig. 9a gezeigt ist. Fig. 9a zeigt eine vollständige Multiplikation, die in einer Struktur ausgelegt ist, die eine Langschriftrechnung wiederspiegelt, mit einer charakteristischen Zelle und mit seinen Eingaben und Ausgaben. Wenn der Multiplikand vorzeichenbehaftet ist, dann sind die Partialprodukte auch vorzeichenbehaftet und müssen, wie auf der linken Seite der Struktur gezeigt ist, um ein korrektes Resultat zu ergeben, auf die volle Breite des vollständigen Produkts erweitert werden. Eine direkte Implementierung einer Vorzeichenerweiterung würde einen Overhead von näherungsweise 50% für eine Multiplikatorzelle des Typs bedeuten, der in Fig. 8 gezeigt ist. Jedoch gibt es Mögllichkeiten für eine größere Effizienz bei der Summation dieser Vorzeichenerweiterung, da es sich wiederholende 1en und 0en in jedem Partialprodukt gibt. Um das Vorzeichen zu erweitern, verwenden wir eine grundlegende Eigenschaft der Arithmetik von zweier Komplementen, nämlich das

$$(-S)\ S\ S\ S\ S\ S\ Z_4\ Z_3\ Z_1\ Z_0 = 0\ 0\ 0\ 0\ 0\ (-S)\ Z_4\ Z_3\ Z_2\ Z_1\ Z_0$$

[0085] Dies erlaubt uns, das Vorzeichen im notwendigen Umfang zu erweitern.

[0086] Arbeitet man sich von der am wenigsten signifikanten zu der am meisten signifikanten Partialsumme vor, so ist die Summe der bisher kennengelernten Vorzeichenerweiterung das Ergebnis des signifikantesten Summenbits auf der linken Seite des trapezförmigen Arrays, das in Fig. 9a gezeigt ist. Folglich können wir den Effekt einer Summation der Vorzeichenerweiterung erreichen, ohne einen derart großen Overhead zu benötigen, wenn wir das signifikanteste Summenbit in jeder Partialsumme entlang der linken Seite extrahieren, dieses Bit zu einer vollständigen Stelle erweitern, und das Resultat zur nächsten Partialsumme addieren.

[0087] Fig. 9b zeigt, wie dies effektiv in der Anordnung, die in Fig. 9a gezeigt ist, implementiert werden kann, indem z. B. Multiplizierzellen verwendet werden, wie sie in Fig. 8 gezeigt sind. In Fig. 9b ist ein Teil des linken Randes des Arrays für eine Folge von Multiplikatorbitpaaren **521**, **522**; **523**, **524** gezeigt. Die Vorzeichenerweiterung wird von der lo-Ausgabe eines dieser Multiplikatorbitpaare genommen und zu der Vorzeichenerweiterungseinheit weitergeleitet, welche als Nachschlagtabelle ausgeführt werden kann. Wenn das signifikanteste Bit der lo-Ausgabe 0 ist, erzeugt die Vorzeichenerweiterungseinheit eine Zahl mit dem Binärwert 0000. Wenn das signifikanteste Bit der lo-Ausgabe 1 ist, erzeugt die Zeichenerweiterungseinheit eine Zahl mit dem Binärwert **1111**. Mit dieser Anordnung wird eine Zeichenerweiterung zur Verfügung gestellt, die äquivalent ist zu dem

vollständigen Erweitern des Vorzeichens, wie es in Fig. 9a gezeigt ist, aber ohne wesentliche zusätzliche Kosten.

[0088] Alternativ könnte das Multiplikatorarray so weit wie nötig in der Breite erweitert werden, so dass jede Partialsumme garantiert eine am meisten signifikante Stelle besitzt, die ausschließlich aus Vorzeichenbits besteht. In diesem Fall wäre ein separater Zeichenerweiterungsschaltkreis nicht notwendig, weil die lo-Ausgabe eines der Multiplikatorbitpaare entweder aus lauter Nullen oder Einsen bestünde, und diese lo-Stelle einfach kopiert werden könnte, um eine neue Zeichenerweiterungsstelle zu erzeugen, die in die nächste Reihe des Arrays eingegeben werden könnte. Die Kosten dieses Ansatzes wären jedoch bedeutend.

[0089] Die obigen Abhandlungen beschreiben einen Arraymultiplizierer, welcher rein kombinatorisch funktioniert, ohne Parallelverarbeitungsregister. Parallelverarbeitung kann insbesondere bei großen Arraymultiplizierern wichtig sein, um eine hohe Geschwindigkeit zu erzielen – daher ist es hilfreich, den Multiplizierer vorsichtig zu parallelisieren. Die folgenden Faktoren sind wichtig, um die beste Parallelisierung zu wählen:

1. Eine Parallelverarbeitung quer über den Multiplikanden zieht bei dieser Architektur nicht nur eine Parallelverarbeitung des Multiplikanden selbst, sondern auch seine x2-, x4- und x8-Ableitungen mit sich. Dies ist teuer und sollte daher nur dann durchgeführt werden, wenn preiswertere Alternativen ausgeschöpft worden sind.
2. Mit einem vorzeichenbehafteten Multiplikanden erzeugt der Vorzeichenerweiterungsmechanismus Abhängigkeiten, die von dem am wenigsten signifikanten zu dem am meisten signifikanten Ende des Multiplikators verlaufen, was die Partialprodukte dazu zwingt, in dieser Reihenfolge gebildet zu werden. Mit einem vorzeichenfreien Multiplikanden ist dies nicht der Fall, und die Partialprodukte können in beliebiger Reihenfolge gebildet werden.
3. Weil die arithmetische Signifikanz eines Übertrags davon abhängig ist, ob die Operation, die ausgeführt wird, eine Addition oder eine Subtraktion ist, muss der Übertrag nach links fortgepflanzt werden, wie dies in Fig. 9a für den Multiplizierer vierfacher Dichte aus Fig. 8 dargestellt ist.
4. Für einen vorzeichenbehaften Multiplikanden pflanzen sich die Überträge nach links fort, und wegen der Abhängigkeiten im Vorzeichenerweiterungsmechanismus müssen die Partialprodukte von oben nach unten gebildet werden. Dies zwingt die Partialsummen, sich nach rechts fortzupflanzen. Dies bedeutet, dass vertikale Zeitabschnitte durch das Array nicht möglich sind, weil die Überträge die Zeitabschnitte in eine Richtung und die Partialsummen die Zeitabschnitte in die entgegengesetzte Richtung überqueren würden. Folglich sind wir in diesem Fall gezwungen, entweder horizontale oder diagonale (von oben links nach unten rechts) Zeitabschnitte einzuführen. Die diagonalen Zeitabschnitte schneiden nicht die Pfade d bis lo, was eine große kombinatorische Verzögerung quer durch das Array anhäufen würde. Die beste Herangehensweise ist daher, horizontale Zeitabschnitte einzuführen, wo es sich um vorzeichenbehaftete Multiplikanden handelt, und den Overhead der Parallelisierung des Multiplikanden und der drei Ableitungen in Kauf zu nehmen. Fig. 10a zeigt den Datenfluss quer durch das vollständig Multipliziererarray für einen vorzeichenbehafteten Multiplikanden und fasst die verschiedenen Arrayoverheads für diesen Fall zusammen.
   Diese Zeitabschnitte implizieren, dass der Multiplikand im Multiplizierer in paralleler Form und der Multiplikator in einem bitversetzten Format präsentiert werden sollten, bei dem weniger signifikante Stellen vor den signifikanteren Stellen stehen sollten. Die exakte Taktsteuerung hängt von den Positionen, die für die Zeitabschnitte gewählt worden sind, ab. Die weniger signifikante Hälfte des Ergebnisses wird in einem bitversetzten Format erzeugt, und die signifikantere Hälfte liegt in einem Parallelformat vor. Zusätzlich kann die signifikanteste Hälfte des Ergebnisses neu getaktet werden, um sie in gleicher Weise bitzuversetzen, wie die am wenigsten signifikante Hälfte des Ergebnisses. Wird dieser Weg gewählt, führt diese zusätzliche Neutaktung zu einem zusätzlichen Overhead am unteren Ende des Arrays, wie dargestellt.
5. Für einen Multiplikanden ohne Vorzeichen existiert kein Vorzeichenerweiterungsmechanismus, und folglich können wir die Partialprodukte von dem signifikantesten Ende des Multiplikators zum am wenigsten signifikanten Ende bilden – in der umgekehrten Richtung zu der in Fig. 10a gezeigten Richtung. In diesem Fall überqueren die lo- und hi-Ausgaben beide die vertikalen Zeitabschnitte in der gleichen Richtung (nach links), so dass vertikale Zeitabschnitte zulässig sind und auch bevorzugt sind, weil sie weniger Aufwand erfordern. Die Fig. 10b illustriert das Multipliziererarray für diesen Fall.

[0090] Zeitabschnitte dieser Form implizieren, dass der Multiplikator dem Array in paralleler Form präsentiert werden sollte, und dass der Multiplikand dem Array im bitversetzten Format präsentiert werden sollte, wobei die am wenigsten signifikanten Stellen zuerst kommen. Dies ist gerade umgekehrt zu der Datentaktung, die für die vorzeichenbehafteten Multiplikatoren mit horizontalen Zeitabschnitten gefordert ist. Die exakte Taktung hängt von den Positionen ab, die für die Zeitabschnitte gewählt sind.

[0091] Wie im Falle der vorzeichenbehafteten Multiplikatoren wird die weniger signifikante Hälfte des Ergeb-

nisses in einem bitversetzten Format und die signifikantere Hälfte in einem Parallelformat zur Verfügung gestellt. Zusätzliche Neutaktung könnte verwendet werden, um die signifikantere Hälfte des Ergebnisses in der gleichen Weise wie die am wenigsten signifikante Hälfte des Ergebnisses bitzuversetzen. Wird dieser Weg gewählt, würde die zusätzliche Neutaktung in diesem Fall am oberen Rand des Arrays einen zusätzlichen Overhead verursachen, wie dargestellt.

[0092] Der Ansatz einer vierfachen Dichte, wie er in Fig. 8 gezeigt ist, ergibt ein exzellentes Array, aber benötigt für Arrays brauchbarer Größe eine Parallelverarbeitung. Solch eine Parallelverarbeitung setzt jedoch voraus, dass einige der Eingaben und Ausgaben in einem bitversetzten Format präsentiert werden, was bei manchen Anwendungen teuer einzurichten sein könnte. Die bitversetzte Taktung ist dadurch erzwungen, dass die Partialprodukte in einer linearen Kette gebildet werden, was wiederum darin seine Ursache hat, dass die Grundzelle nur ein Partialprodukt bilden und nur eine weitere Eingabe (und eine Übertragseingabe) akkumulieren kann. Wird ein Ansatz gewählt, bei dem die Bildung der Partialprodukte von deren Akkumulation getrennt ist, dann ist die Bildung jeden Partialprodukts eine unabhängige Operation, und sie können alle parallel ausgeführt werden. Dies führt zu Lösungen, bei denen einige Zellen ungenügend ausgelastet sind und nur verwendet werden, um die Partialprodukte zu bilden (wobei eine Eingabe ungenutzt bleibt), und andere Zellen werden verwendet, um Paare von Partialprodukten und Partialsummen zu bilden. Bei diesem Ansatz können die Zellen, die verwendet werden, um die Partialprodukte zu bilden, immer noch eine Booth-Codierung des Multiplikators verwenden, um die Instruktionseingaben anzusteuern, was ihnen erlaubt, das Partialprodukt, das zwei Bits des Multiplikators entspricht, in jeder Zelle zu akkumulieren. Die Zellen, die Paare von Partialprodukten akkumulieren, sind statisch als Addierer konfiguriert und verwenden keine Instruktionseingaben.

[0093] Der Vorteil dieses Ansatzes im Vergleich zu dem Ansatz der linearen Akkumulation besteht darin, dass die Eingaben und Ausgaben parallel zur Verfügung stehen, und dass der Baum von Addierern eine geringere Wartezeit aufweist als die lineare Kette von Addierern. Die Nachteile sind, dass er mehr Zellen benötigt als der Ansatz der linearen Akkumulation und die Zellverknüpfungen so sind, dass einige lange Verbindungen über das Array benötigt werden, die die Bearbeitungsgeschwindigkeit verringern könnten, und dass ein solcher Ansatz unvereinbar ist mit der unten beschriebenen effizienten Vorzeichenerweiterung des Multiplikanden, weil diese effiziente Form der Vorzeichenerweiterung voraussetzt, dass die Partialprodukte mit dem am wenigsten signifikanten Teil zuerst verarbeitet werden, was zu einer zusätzlichen Abhängigkeit führt, die eine lineare Akkumulation der Partialprodukte zwingend macht.

[0094] Alternativen zu den parallelisierten Arraymultiplizierern aus Fig. 10a und Fig. 10b können bei analogen Seriell-Parallel-Multiplizierern zur Verfügung gestellt werden. Bei diesen sind die einander folgenden "Zeitscheiben" der vollständigen Kalkulation auf denselben linearen Hardware-Streifen in aufeinander folgenden Taktzyklen abgebildet, anstatt im Raum verteilt zu sein, wie dies bei Arraymultiplizierer der Fall ist. Die zur Verfügung stehenden Optionen sind die gleichen wie bei den Arraymultiplizierern, aber bei den Arraymultiplizierern beeinflussen die Kosten der Parallelisierung des Multiplikanden die Auswahl der Parallelisierung, wohingegen dies kein wesentliches Thema im Fall eines Seriell-Parallel-Multiplizierers ist. Auf der anderen Seite werden jegliche Arrayoverheads, die entlang der Länge des linearen Multiplizierers zur Verfügung gestellt werden müssen, proportional teurer sein als im Fall eines Arraymultiplizierers.

[0095] Fig. 11a zeigt einen seriellen Multiplikator mit einem parallelen Multiplikanden. Dies ist im Ergebnis äquivalent zu der Projektion des Arraymultiplizierers von Fig. 10a auf ein lineares Array, wobei das lineare Array die Arbeit aufeinander folgender horizontaler Streifen des Arrays in aufeinander folgenden Taktschritten erfüllt. Der Multiplikator ist inkrementell in serieller Weise eingegeben, während der Multiplikand dem Verarbeitungsarray **111** parallel zur Verfügung steht. Im ersten Schritt wird das erste (am wenigsten signifikante Bit) des Multiplikators mit dem folgenden Multiplikanden multipliziert. Das Ergebnis wird im Register **113** ausgelesen, mit Ausnahme für das am wenigsten signifikante Bit, welches separat nach außen geleitet wird. Jeder Zeitschritt erzeugt in dieser Weise eine weitere Stelle des am wenigsten signifikanten Teils des Ergebnisses, und mit dem letzten Zeitschritt ist der Rest des Ergebnisses (der signifikanteste Teil) in paralleler Form vom Register **113** zugänglich. Um dem Multiplikator zu erlauben, sofort für eine weitere Berechnung wiederverwendet zu werden, wird das Ergebnis des signifikantesten Teils, welches parallel zum Schieberegister **112** übertragen wird, in digitalem seriellem Format von einem separaten Bus abgerufen.

[0096] Vorteile dieser Form eines Parallel-Seriell-Multiplizierers sind, dass der Booth-Codieroverhead für nur eine Stelle des Multiplikators anfällt, und der Multiplikand und seine drei Vielfachen nicht parallel verarbeitet werden müssen (sie werden konstant gehalten für die Dauer jeder Multiplikation), und dass nur ein Schieberegister notwendig ist, um die signifikanteste Hälfte des Ergebnisses festzuhalten, weil die Überträge bereits fortgepflanzt wurden, und das Ergebnis in normaler binärer Darstellung (ohne Vorzeichen oder als Komple-

ment von zwei) vorliegt. Nachteile dieses Parallel-Seriell-Multiplizierers sind, dass drei Vielfache des Multiplikanden aufwendig zu berechnen sind, und dass das Fehlen der Parallelisierung beim Übertragspfad die Geschwindigkeit bei großen Multiplikanden begrenzen könnte.

[0097] Fig. 11b zeigt, ähnlich dazu, einen Parallel-Multiplizierer mit seriellen Multiplikanden: dieses Mal als Resultat einer Projektion des Arraymultiplizierers von Fig. 10b auf ein lineares Array. Für diesen Fall sind nun zwei Schieberegister **114** und **115** für die lo- und die c-Komponenten vorgesehen. Jeder Taktschritt erzeugt erneut eine Stelle des am wenigsten signifikanten Teils des Ergebnisses, wobei der am meisten signifikante Teil des Ergebnisses in paralleler Form mit dem letzten Taktschritt verfügbar wird, obwohl die lo- und c-Komponenten des signifikantesten Teils des Ergebnisses immer noch zusammenaddiert werden müssen, um die Darstellung des Komplements von zwei zu erhalten. Für die sofortige Wiederverwendung des Multiplikators ist es hier notwendig, dass diese beiden Teile des Resultats parallel zu den beiden Schieberegistern **114** und **115** übertragen werden, für den Zugang in einem digitalen-seriellen Format, Summation und der Weiterleitung zu einem separaten Bus. Eine besonders effektive Weise, dies durchzuführen ist das Auslesen der Schieberegister **114** und **115** Stelle für Stelle, und das Hinzuaddieren der Stellen in der Reihenfolge, in der sie durch den Addierer **116** erscheinen (dies erlaubt, dass der Übertrag effektiv gehandhabt werden kann).

[0098] Die Vorteile dieser Form eines Addierers sind eine Parallelverarbeitung aller Pfade, um jegliche Fortpflanzungs-Verzögerungsprobleme zu vermeiden, und das Vorliegen nur einer Overheadstelle für die Erzeugung x2-, x4- und x8-Versionen des Multiplikanden. Nachteile ziehen den Overhead der Booth-Codierung für die volle Länge des Multiplikators mit sich, und einen doppelten Overhead von Schieberegistern, um die signifikantesten Teile des Ergebnisses festzuhalten (weil Überträge nicht vollständig fortgepflanzt werden, bevor der Übertrag zu den Registern stattfindet).

[0099] Die Wahl des Seriell-Parallel-Multiplizierers wird durch die Anforderungen der speziellen Funktion und der speziellen Architektur bestimmt: als generelles (nicht universelles) Prinzip benötigt der Multiplizierer mit parallelem Multiplikanden und seriellem Multiplikator weniger Hardware, während der Multiplizierer mit seriellem Multiplikanden und parallelem Multiplikator weniger Einschränkungen bei der Geschwindigkeit aufweist.

[0100] Beide Eingaben werden dem Seriell-Seriell-Multiplizierer seriell zugeführt, und zwar mit der am wenigsten signifikanten Stelle zuerst. Natürlich hat der Multiplizierer, nachdem die am wenigsten signifikanten D-Stellen jedes Operanden geliefert worden sind, genug Information, um die am wenigsten signifikanten D-Stellen des Ergebnisses zu bilden. Bezugnehmend auf Fig. 12a kann zum Taktschritt D das Produkt der am wenigsten signifikanten D-Stellen der zwei Operanden gebildet werden. Dieses wird im allgemeinen 2D-Stellen umfassen, von denen wenigstens D signifikante Stellen sich nicht noch einmal ändern werden und von dem Schaltkreis ausgegeben werden können. Die signifikantesten Stellen werden zurückgehalten, und im nächsten Zeitschritt werden weitere Terme zu ihnen addiert, um das nächste Partialprodukt zu bilden.

[0101] Die zusätzlichen Terme, die zu jedem Zeitschritt addiert werden, sind in Fig. 12a durch Paare von Rechtecken markiert, welche die Nummer des Taktschrittes tragen. Zum Taktschritt 5, z. B., zeigt das vertikale Rechteck (schattiert) mit der Nummer 5 das Produkt von Stelle 5 von X und den Stellen 1 bis 5 einschließlich von Y. Das horizontale Rechteck mit der Ziffer 5 zeigt das Produkt von Stelle 5 von Y und Stellen 1 bis 4 einschließlich von X. Beide Produkte müssen zu dem Partialprodukt addiert werden, das zum Taktschritt 4 erzeugt wurde, um das Partialprodukt zum Taktschritt 5 zu bilden.

[0102] Die horizontal und vertikal schattierten Rechtecke in Fig. 12a korrespondieren mit den Seriell-Parallel-Multiplikationen, wie sie unter Bezugnahme auf die Fig. 11a und Fig. 11b beschrieben wurden. Der Fall der Fig. 12a unterscheidet sich jedoch darin, dass die parallelen Komponenten nicht durch die gesamte Berechnung hindurch konstant sind, sondern zu jedem Taktschritt geändert werden. Die Berechnung kann daher, wie in Fig. 12a gezeigt, durch zwei Schieberegister **121**, **122** ausgeführt werden, wobei jedes einen jeweiligen Seriell-Parallel-Multiplizierer versorgt, wobei die Ausgabe der zwei Multiplizierer **123**, **124** durch einen seriellen Addierer **125** addiert wird. Jeder der Seriell-Parallel-Multiplizierer **123**, **124** kann auf jede der Weisen, die in Fig. 11a und Fig. 11b gezeigt sind, gebaut werden.

[0103] Der Fachmann erkennt, dass bei der Anwendung der beanspruchten Erfindung eine Vielzahl von Veränderungen und Modifikationen möglich ist.

Patentansprüche

1. Eine integrierte Schaltung für die Multiplikation einer ersten Zahl mit einer zweiten Zahl, die integrierte

Schaltung umfasst:
eine erste Zahleneingabe zum Empfangen der ersten Zahl, eine zweite Zahleneingabe zum Empfangen der zweiten Zahl, und eine Ausgabe zum Bereitstellen des Ergebnisses der Multiplikation;
eine Verschlüsselung (**505**) für Booth Verschlüsselungsgruppen von wenigstens zwei Bits der ersten Zahl; ein Array (**111**) von Verarbeitungseinrichtungen (**12**; **507**; **521**–**524**), wobei jede Verarbeitungseinrichtung eine Vielzahl von Dateneingaben, eine Übertragseingabe, eine Übertragsausgabe, und eine Vielzahl von Datenausgaben besitzt, und jede Verarbeitungseinrichtung in der Lage ist, ein Partialprodukt für die Multiplikation von wenigsten zwei Bits der ersten Zahl mit m Bits der zweiten Zahl auszuführen, wobei m größer oder gleich eins ist; und
Verknüpfungsvorrichtungen (**14**) zur Verknüpfung der Eingaben und Ausgaben derart, dass für jede Verarbeitungseinrichtung:
die Dateneingaben von m Bits der zweiten Zahl zur Verfügung gestellt werden; und eine Summationseingabe für eine Summe von Partialprodukten der gleichen Signifikanz;
eine Übertragseingabe wird bereitgestellt, wenn geeignet, um einen Übertrag von einem weniger signifikanten Partialprodukt zu addieren;
die Datenausgaben werden als Summationsausgabe zur Verfügung gestellt, die die am wenigsten signifikanten m Bits des Partialproduktes beinhaltet; und
eine Übertragsausgabe wird bereitgestellt, die jedwede signifikanten Bits des Partialproduktes enthält; wobei die integrierte Schaltung ein programmierbares Array umfasst wobei jede Verarbeitungseinrichtung (**12**; **507**; **521**–**524**) in dem Array eine Instruktionseingabe (INST) zur Steuerung der Funktion umfasst, die in der Verarbeitungseinrichtung ausgeführt wird und wobei die Verschlüsselungseinrichtung (**505**) eine Instruktion für eine Vielzahl von Verarbeitungseinrichtungen basierend auf einer Gruppe von Booth verschlüsselten Bits der ersten Zahl generiert; und
die integrierte Schaltung eine Vielzahl von Feldern (**10**) umfasst, von denen jedes Feld eine Vielzahl von Zeilen und Spalten umfasst, jede Zeile und jede Spalte eine alternierende Sequenz einer Verarbeitungseinrichtung (**12**) umfasst und einen Schaltungsbereich (**14**), jeder der Schaltungsbereiche beinhaltet eine Vielzahl von Schaltern für eine konfigurierbare Verbindung der Eingaben und Ausgaben, mit der Verbindungsvorrichtung, die von den Schalterbereichen in der Vielzahl von Feldern zur Verfügung gestellt wird.

2. Eine integrierte Schaltung gemäß Anspruch 1, wobei der Wert, der an der Instruktionseingabe empfangen wird, durch zwei Bits der ersten Zahl bestimmt wird, so dass die bereitgestellten Instruktionen jeweils verursachen, dass Werte, die das 0-, 1-, 2 und 3-fache der m Bits der die zweite Zahl repräsentierenden Werte zu den Datenausgaben weitergegeben werden, entsprechend der Werte der zwei Bits der ersten Zahl.

3. Eine integrierte Schaltung gemäß Anspruch 1, wobei der Wert, der an der Instruktionseingabe empfangen wird, durch drei Bits der ersten Zahl bestimmt wird, in der Weise, dass die bereitgestellten Instruktionen jeweils verursachen, dass zu den Datenausgaben Werte weitergereicht werden, die das 0, 1, 2, 3 und 4-fache der m Bits der zweiten Zahl repräsentieren, entsprechend der Werte der drei Bits der ersten Zahl.

4. Eine integrierte Schaltung gemäß Anspruch 3, wobei das Verhältnis zwischen den drei Bits der ersten Zahl und den Werten, die zu den Datenausgaben weitergereicht werden, mit dem Radix-4 modifizierten Booth Algorithmus übereinstimmen.

5. Eine integrierte Schaltung nach einem der vorhergehenden Ansprüche, wobei die zweite Zahl vorzeichenbehaftet ist, wobei Mittel zur Verfügung gestellt werden, um das Vorzeichen durch fortschreitende Addition der am meisten signifikanten Summenbits zu jeder Partialsumme abzuleiten, unter Verwendung des am meisten signifikanten Bits der zweiten Zahl.

6. Eine integrierte Schaltung nach einem der vorhergehenden Ansprüche, die das zur Verfügungstellen entweder der ersten Zahl oder der zweiten Zahl in einer Parallelverarbeitung umfasst, wobei eine der ersten Zahl und der zweiten Zahl im Array von Verarbeitungseinrichtungen parallel und wobei die andere der ersten Zahl und der zweiten Zahl dem Array von Verarbeitungseinrichtung fortschreitend von am wenigsten signifikanten zum signifikantesten Bit zu Verfügung gestellt wird.

7. Eine integrierte Schaltung gemäß Anspruch 6, wobei das Array von Verarbeitungseinrichtungen weiter eine Vielzahl von Hilfsverarbeitungseinheiten umfasst, die zum Summieren der Partialprodukte vorgesehen sind, wobei die Bildung der Partialprodukte durch eine Verarbeitungseinrichtung ohne große Abhängigkeit von der Ausgabe einer anderen Verarbeitungseinrichtung durchgeführt werden kann.

8. Ein Verfahren zum Multiplizieren einer ersten Zahl mit einer zweiten Zahl unter Verwendung einer inte-

grierten Schaltung, die eine erste Zahleneingabe zum Empfangen der ersten Zahl, eine zweite Zahleneingabe zum Empfangen der zweiten Zahl, eine Ausgabe zum Bereitstellen des Ergebnisses der Multiplikation, ein Array (**111**) von Verarbeitungseinrichtungen (**12**, **507**; **521**–**524**) umfasst, jede Verarbeitungseinrichtung weist eine Vielzahl von Dateneingaben, eine Übertragseingabe, eine Übertragsausgabe, eine Vielzahl von Datenausgaben auf, und Vorrichtungen (**14**), die die Eingaben und Ausgaben verbinden,
die integrierte Schaltung umfasst ferner eine Vielzahl von Feldern (**10**), jedes Feld umfasst eine Vielzahl von Reihen und Spalten, jede Reihe und jede Spalte umfasst eine Wechselfolge einer Verarbeitungseinrichtung (**12**) und eines Schalterbereiches (**14**), die Schalterbereiche, von denen jeder eine Vielzahl von Schaltern für eine konfigurierbare Verbindung der Eingaben und Ausgaben umfasst, mit der Verbindungsvorrichtung, die durch die Schalterbereiche in der Vielzahl von Feldern zur Verfügung gestellt wird,
umfasst das Verfahren die Multiplikationsschritte von:
zur Verfügung stellen von m Bits der zweiten Zahl an den Dateneingängen jeder Verarbeitungseinrichtung, wobei m größer oder gleich eins ist, und eine Summationseingabe für die Summe von Partialprodukten der gleichen Signifikanz;
zur Verfügung stellen einer Übertragseingabe, wenn angebracht, um einen Übertrag eines weniger signifikanten Partialprodukts zu addieren;
Berechnen eines Partialprodukts mit jeder Verarbeitungseinrichtung;
Bereitstellen einer Summationsausgabe an den Datenausgaben, die die am wenigsten signifikanten m Bits des Partialproduktes enthält; und
Bereitstellen einer Übertragsausgabe, die jedwede signifikanten Bits des Partialprodukts enthält;
und wobei die integrierte Schaltung ein programmierbares Array umfasst, bei dem jede Verarbeitungseinrichtung in dem Array eine Instruktionseingabe (INST) zur Steuerung der Funktion besitzt, die in der Verarbeitungseinrichtung ausgeführt werden soll
und das Verfahren umfasst ferner Booth Verschlüsselungsgruppen von wenigstens zwei Bits der ersten Zahl und generiert eine Instruktion für eine Vielzahl von Verarbeitungseinrichtungen basierend auf einer Gruppe von Booth Verschlüsselungsbits der ersten Zahl und stellt die Instruktion zu einer Vielzahl von Verarbeitungseinrichtungen bereit.

9. Ein Verfahren zum Multiplizieren gemäß Anspruch 8, wobei der Wert, der an der Instruktionseingabe empfangen wird, durch zwei Bits der ersten Zahl bestimmt wird, in der Weise, dass die bereitgestellten Instruktionen jeweils verursachen, dass Werte, die das 0, 1, 2 und 3-fache der m Bits der die zweite Zahl repräsentierenden Werte zu den Datenausgaben weitergegeben werden, entsprechend der Werte der zwei Bits der ersten Zahl.

10. Verfahren zum Multiplizieren gemäß Anspruch 8, wobei der Wert, der am Instruktionseingang empfangen wird, durch drei Bits der ersten Zahl bestimmt wird, in der Weise, dass die bereitgestellten Instruktionen jeweils verursachen, dass zu den Datenausgaben Werte weitergereicht werden, die das 0, 1, 2, 3 und 4-fache der m Bits der zweiten Zahl repräsentieren, entsprechend der Werte der drei Bits der ersten Zahl.

11. Verfahren zum Multiplizieren gemäß Anspruch 10, wobei das Verhältnis zwischen den drei Bits der ersten Zahl und den Werten, die zu den Datenausgaben weitergereicht werden, mit dem Radix-4 multiplizierten Booth-Algorithmus übereinstimmen.

12. Verfahren zum Multiplizieren gemäß einem der Ansprüche 8 bis 11, wobei wenigstens eine der ersten Zahl und der zweiten Zahl vorzeichenbehaftet ist.

13. Verfahren zum Multiplizieren nach Anspruch 12, wobei die zweite Zahl vorzeichenbehaftet ist, wobei das Vorzeichen durch fortschreitende Addition der am meisten signifikanten Summenbits zu jeder Partialsumme abgeleitet ist, unter Verwendung des am meisten signifikanten Bits der zweiten Zahl.

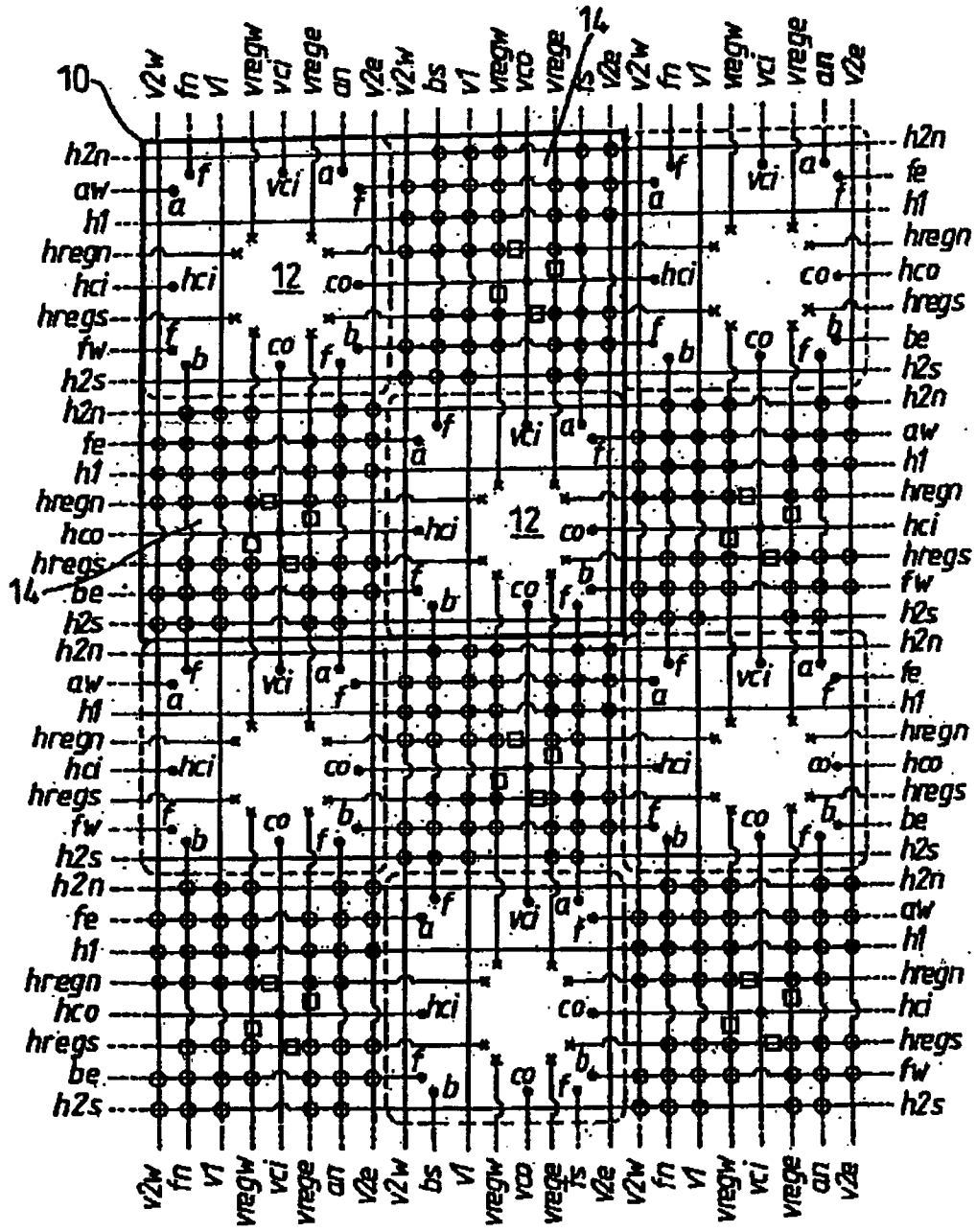
14. Verfahren zum Multiplizieren nach einem der Ansprüche 8 bis 13, welches das zur Verfügung stellen entweder der ersten Zahl oder der zweiten Zahl in einer Parallelverarbeitung umfasst, wobei eine der ersten Zahl und der zweiten Zahl im Array von Verarbeitungseinrichtungen parallel und die andere der ersten Zahl und der zweiten Zahl dem Array von Verarbeitungseinrichtungen fortschreitend vom am wenigsten signifikanten zum signifikantesten Bit zur Verfügung gestellt wird.

15. Verfahren zum Multiplizieren nach Anspruch 14, wobei das Array von Verarbeitungseinrichtungen weiter eine Vielzahl von Hilfsverarbeitungseinheiten umfasst, die zum Summieren der Partialprodukte vorgesehen sind, wobei die Bildung der Partialprodukte durch eine Verarbeitungseinrichtung unabhängig von der Ausgabe

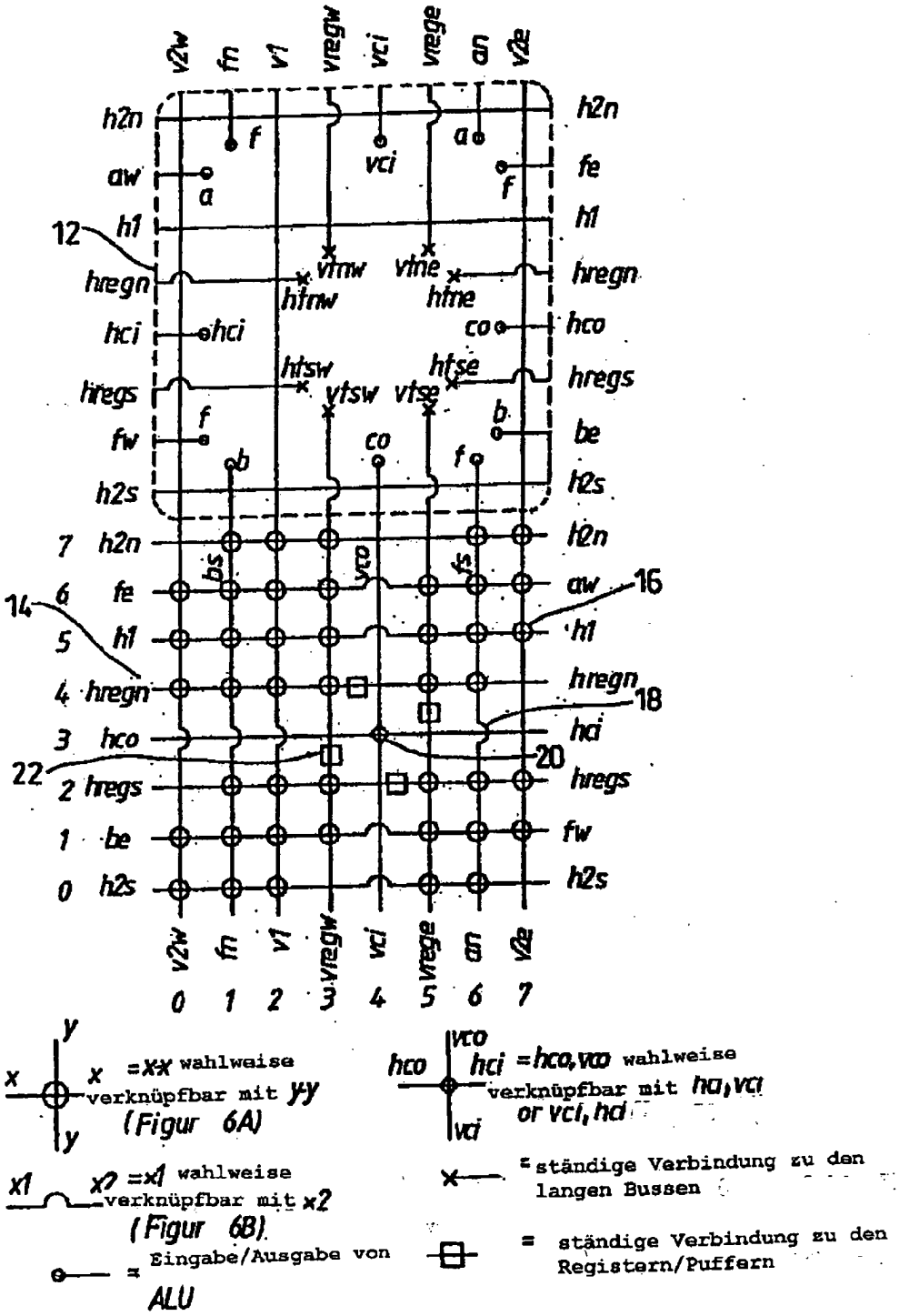
einer anderen Verarbeitungseinrichtung ausgeführt werden kann.

Es folgen 15 Blatt Zeichnungen

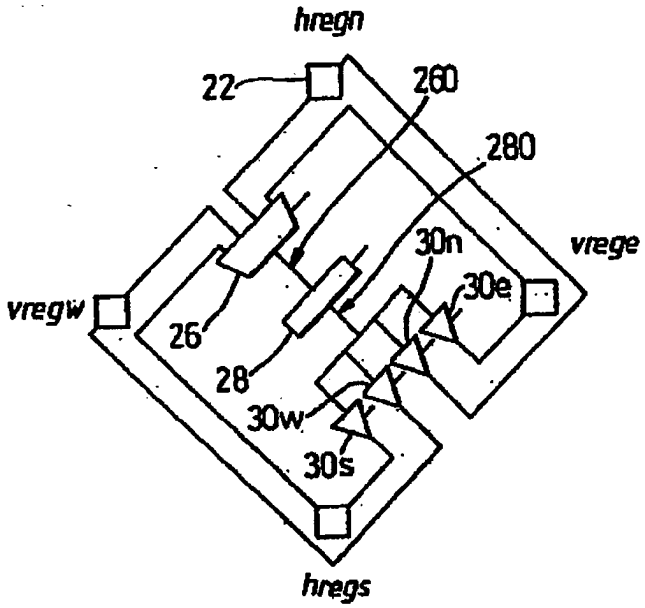
Anhängende Zeichnungen



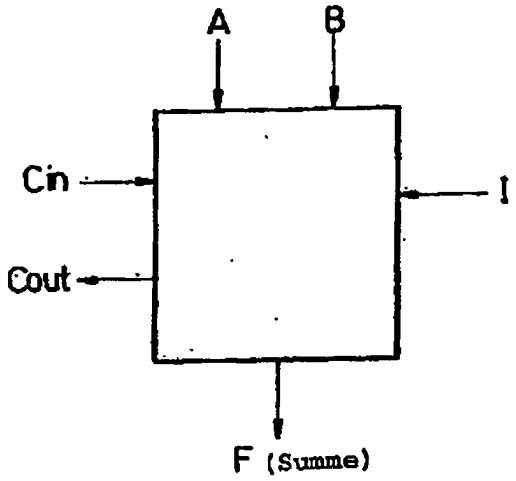
Figur 1


v2w
fm
v1
vregw
vci
vrege
an
v2e
h2n
aw
h1
hregn
hci
hregs
fw
h2s
fe
be
hco
12
14
16
18
20
22
vfnw
vfne
hfnw
hfne
vfsw
vfse
hfsw
hfse
bs
vco
fs
0 1 2 3 4 5 6 7
=xx wahlweise verknüpfbar mit y-y
(Figur 6A)
=x1 wahlweise verknüpfbar mit x2
(Figur 6B)
= Eingabe/Ausgabe von ALU
=hco,vco wahlweise verknüpfbar mit hci,vci or vci,hci
= ständige Verbindung zu den langen Bussen
= ständige Verbindung zu den Registern/Puffern

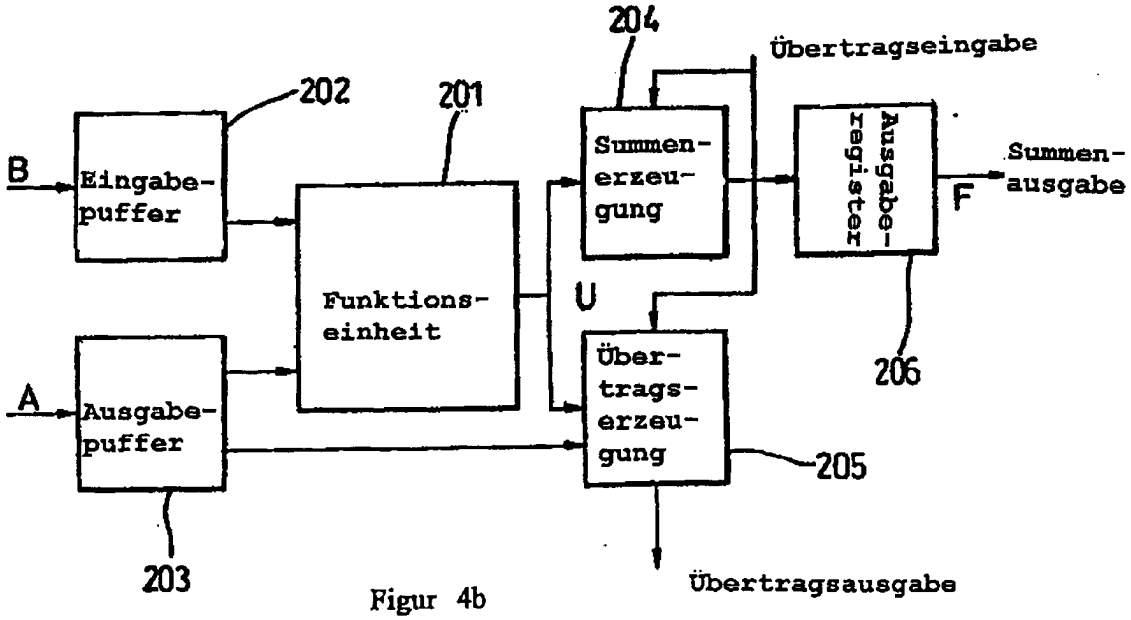
Figur 2



Figur 3



Figur 4a



Figur 4b

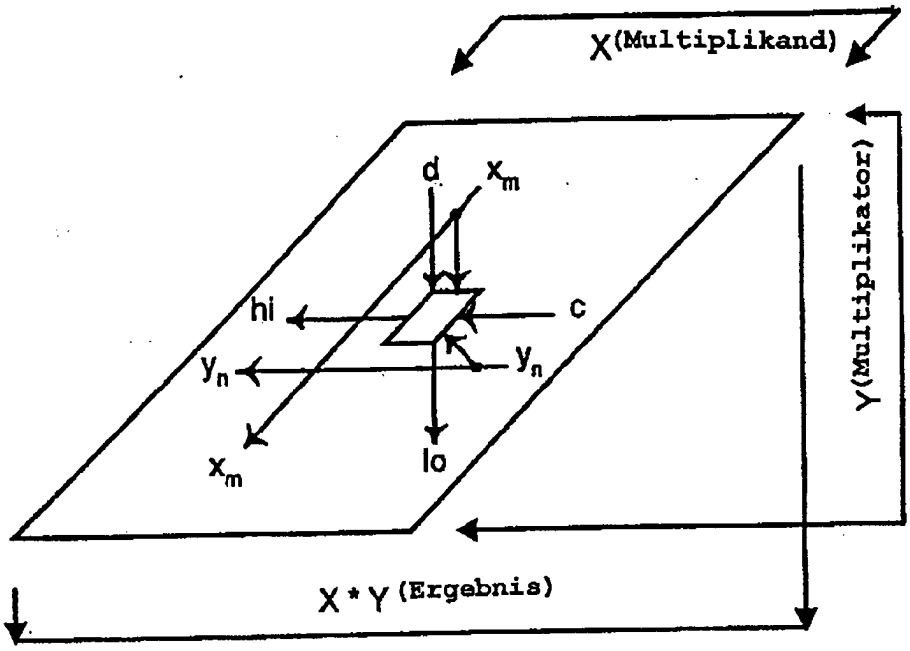

X(Multiplikand)
Y(Multiplikator)
d
x_m
hi
c
y_n
y_n
x_m
lo
X * Y (Ergebnis)

FIG. 5a

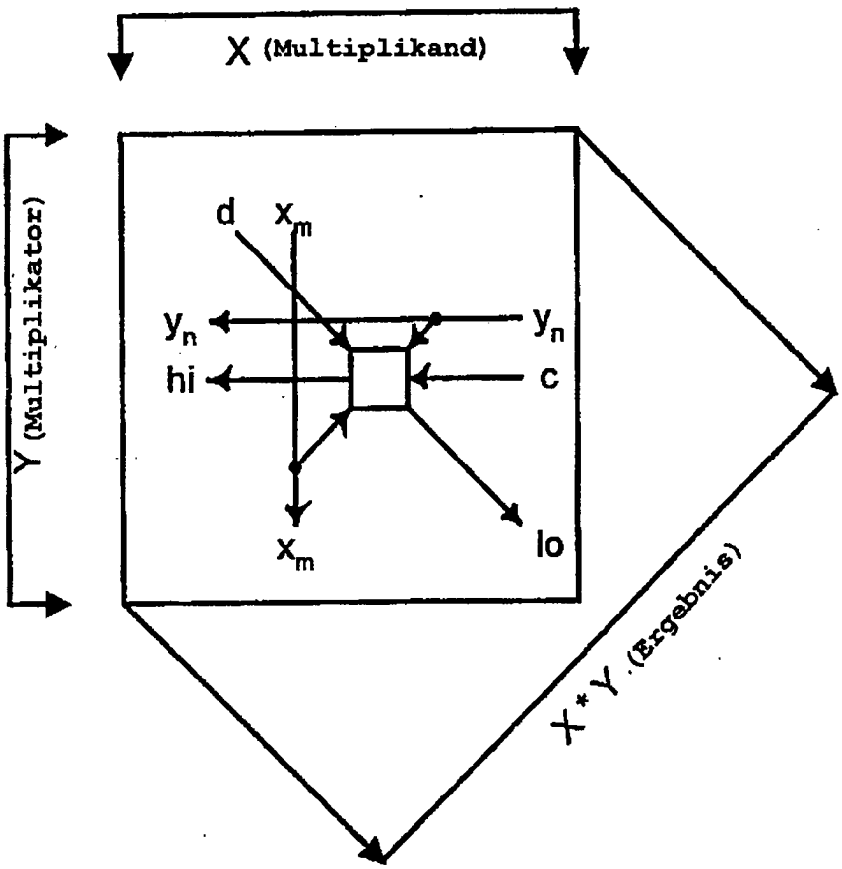

X (Multiplikand)
Y (Multiplikator)
d
x_m
y_n
hi
y_n
c
x_m
lo
X * Y (Ergebnis)

FIG. 5b

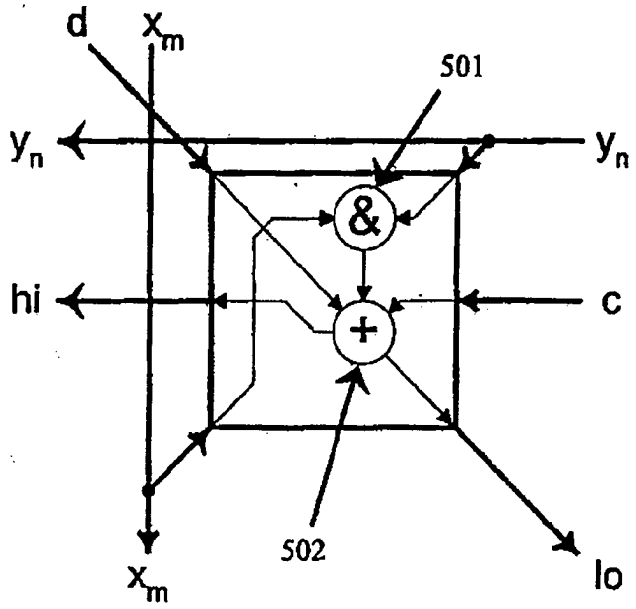


FIG. 6a

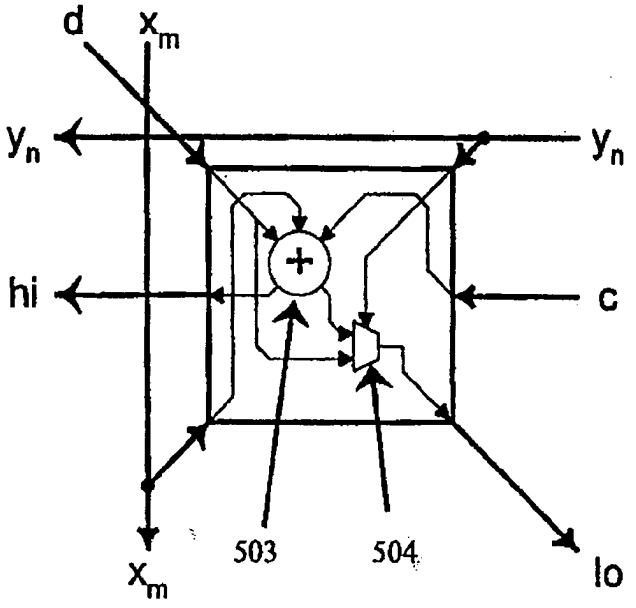


FIG. 6b

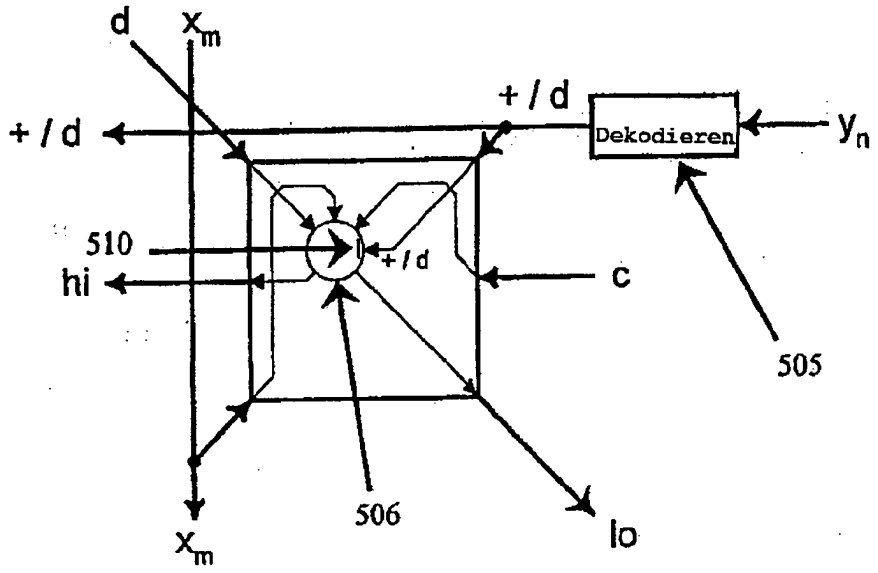


FIG. 7

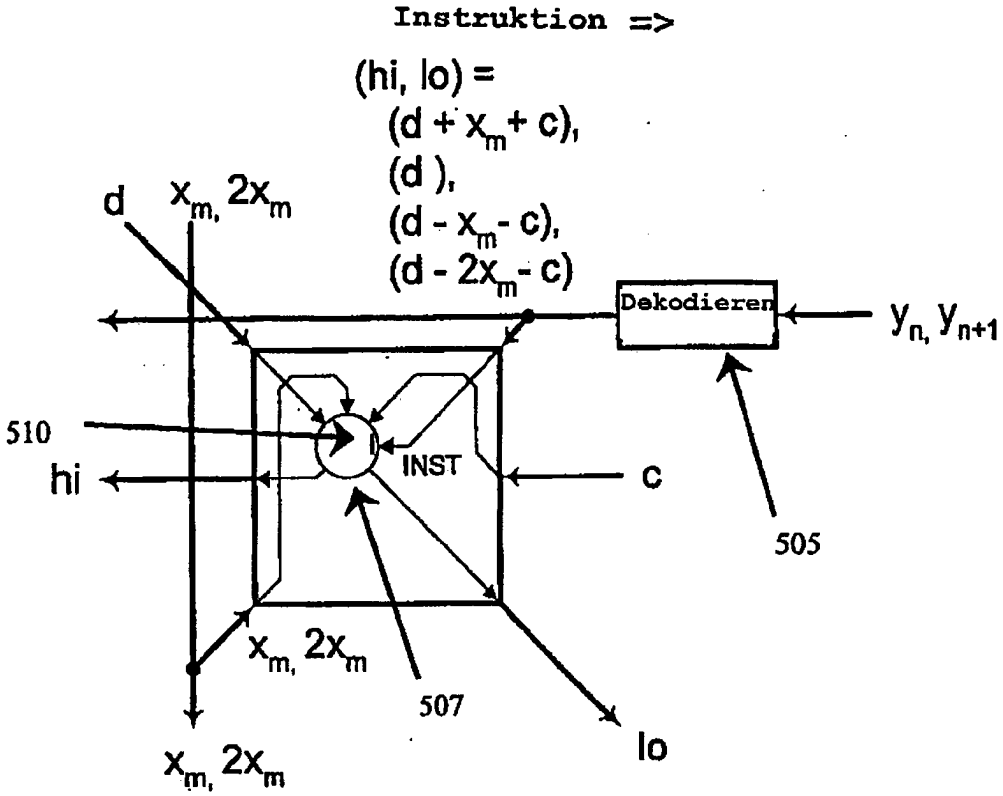
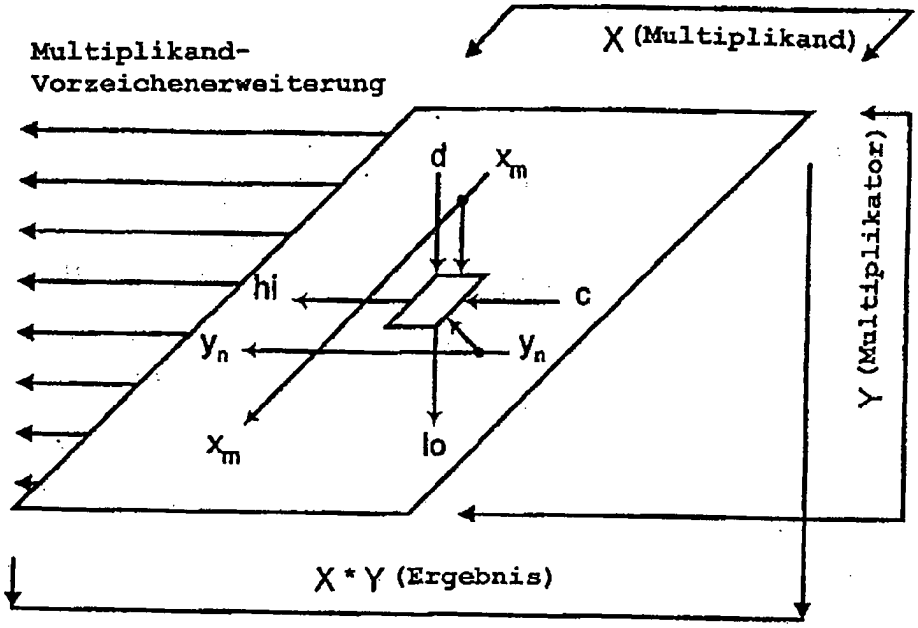
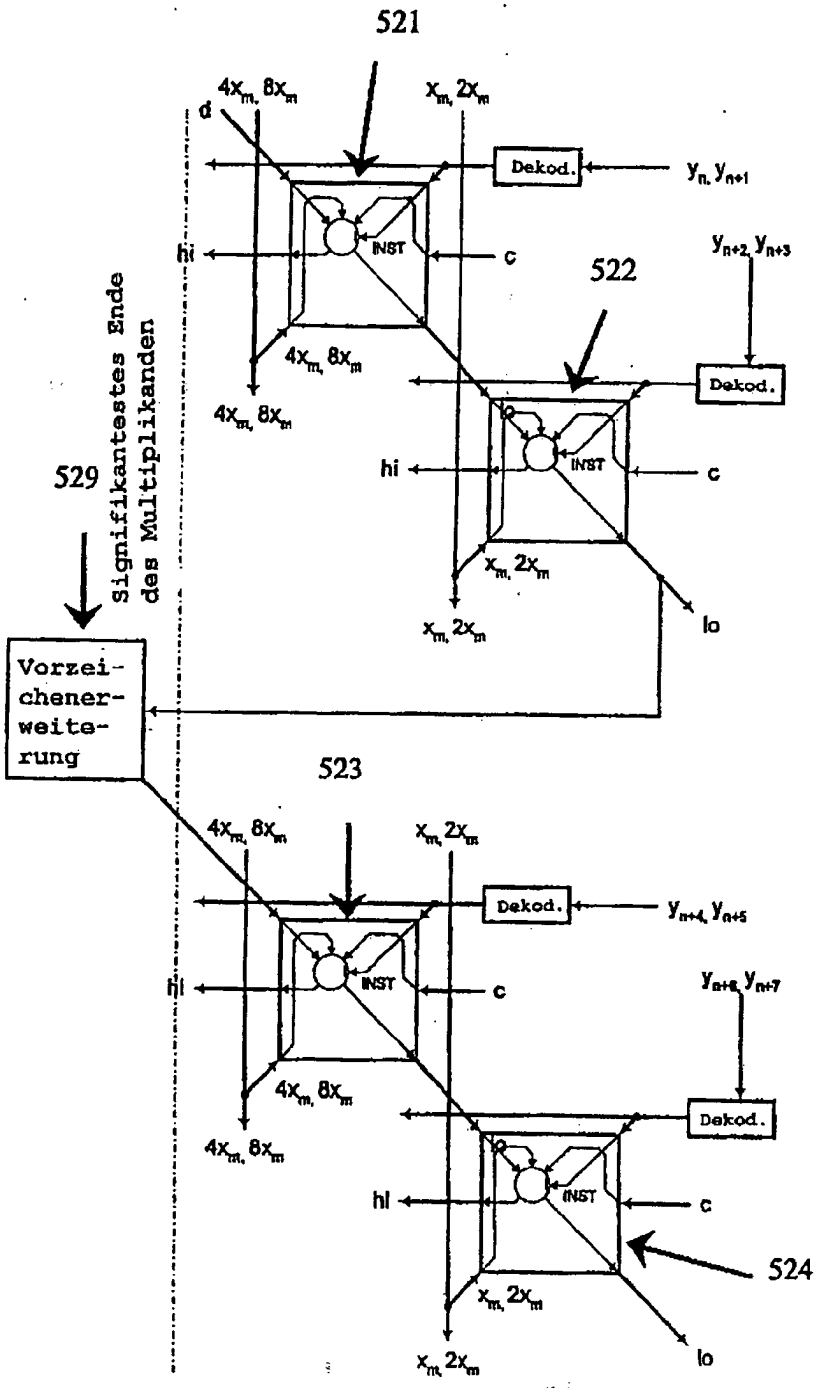

Instruktion =>
(hi, lo) =
(d + x_m + c),
(d),
(d - x_m - c),
(d - $2x_m$ - c)
d
x_m, $2x_m$
Dekodieren
y_n, y_{n+1}
510
hi
INST
c
505
x_m, $2x_m$
507
lo
x_m, $2x_m$

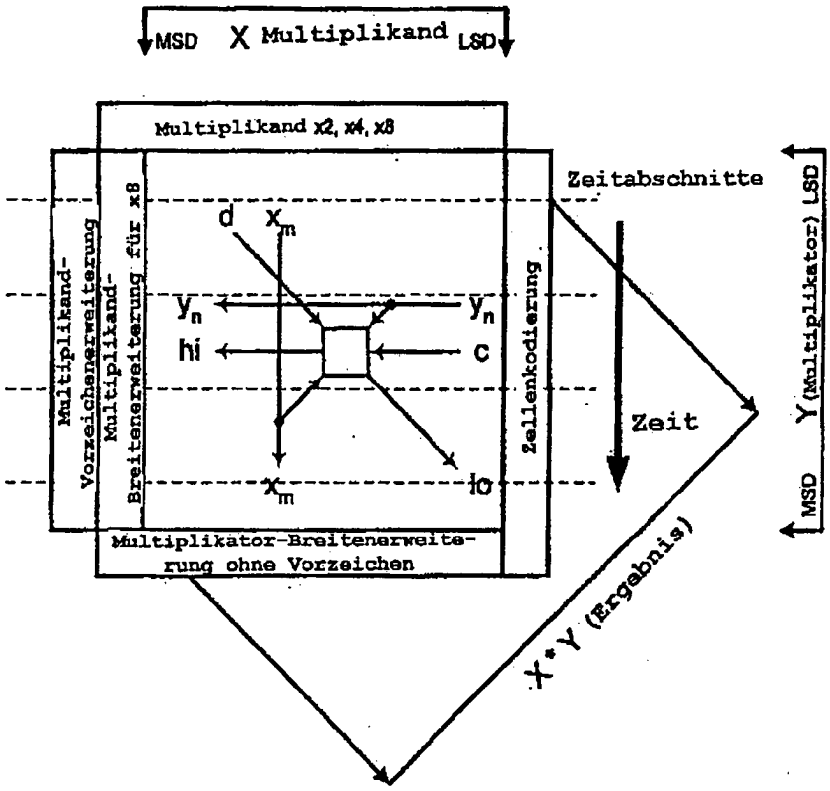
FIG. 8


Multiplikand-
Vorzeichenerweiterung
X (Multiplikand)
d
x_m
hi
c
y_n
y_n
x_m
lo
Y (Multiplikator)
X * Y (Ergebnis)

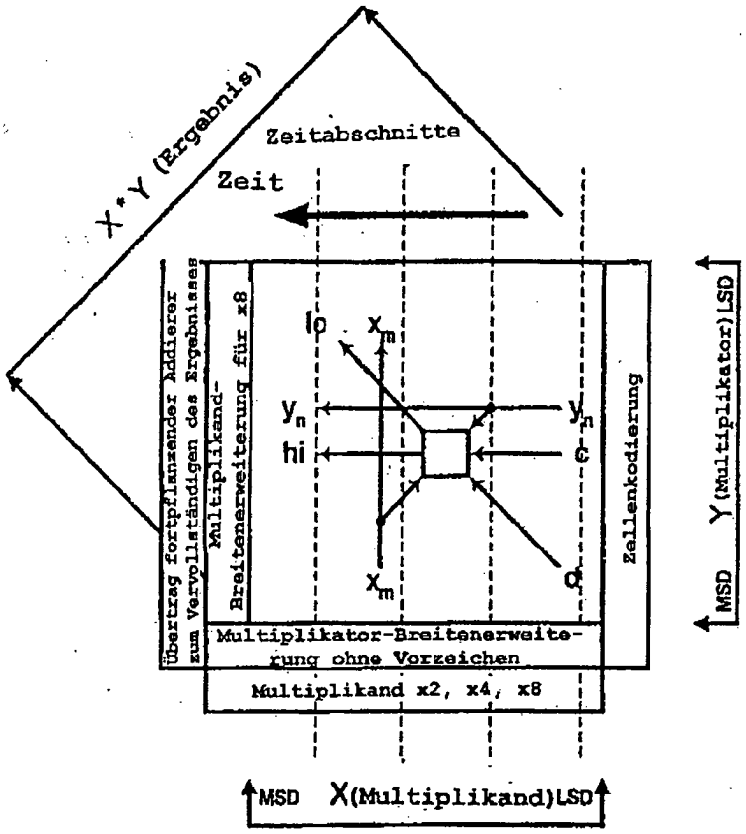
Figur 9a


521
522
523
524
529
Signifikantestes Ende des Multiplikanden
Vorzeichenerweiterung
Dekod.
INST
hi
lo
c

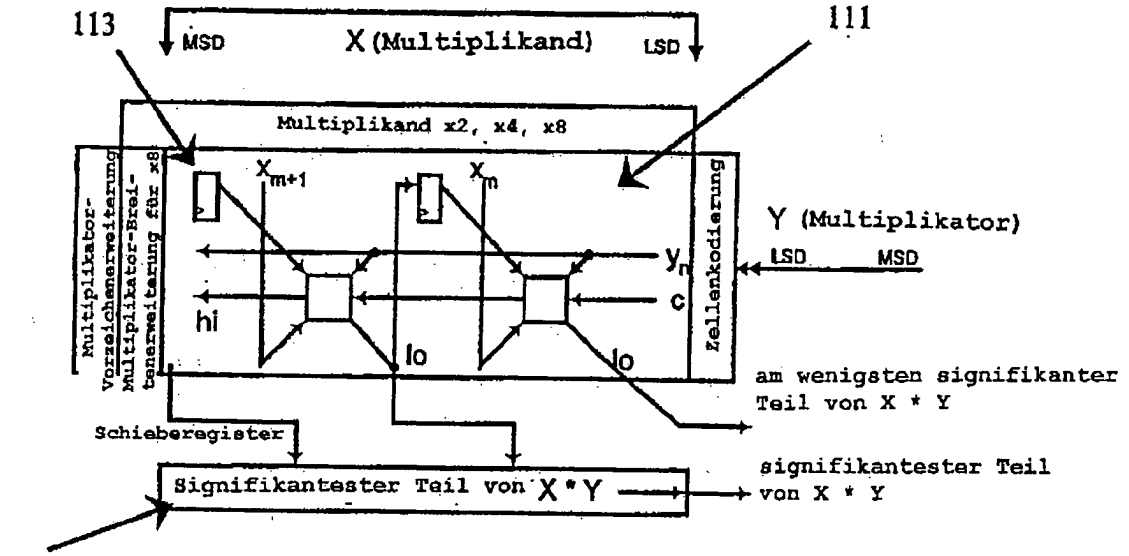
Figur 9b


MSD X Multiplikand LSD
Multiplikand x2, x4, x8
Multiplikand-Vorzeichenerweiterung
Multiplikand-Breitenerweiterung für x8
Zeitabschnitte
Zeit
Zellenkodierung
Y (Multiplikator) LSD
MSD
Multiplikator-Breitenerweiterung ohne Vorzeichen
X * Y (Ergebnis)
d
x_m
y_n
y_n
hi
c
x_m
lo

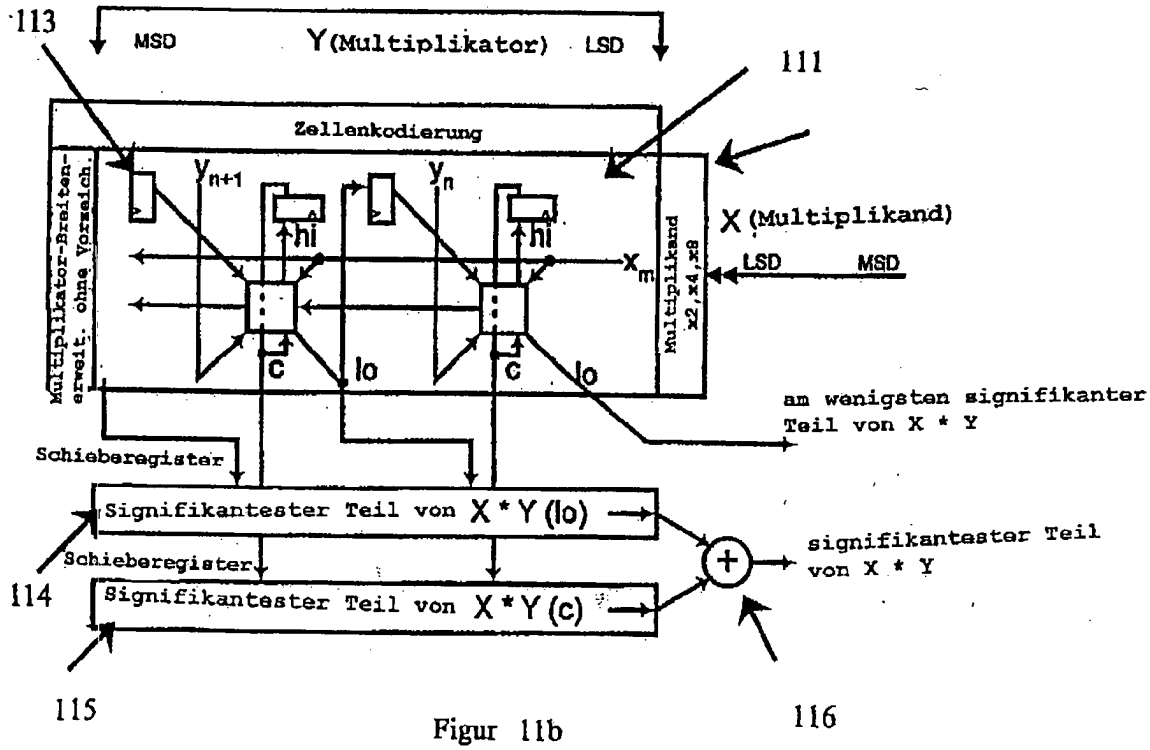
Figur 10a


X*Y (Ergebnis)
Zeitabschnitte
Zeit
Übertrag fortpflanzender Addierer zum Vervollständigen des Ergebnisses
Multiplikand-Breitenerweiterung für x8
lo
x_m
y_n
hi
y_n
x_m
o
Zellenkodierung
MSD Y(Multiplikator)LSD
Multiplikator-Breitenerweiterung ohne Vorzeichen
Multiplikand x2, x4, x8
MSD X(Multiplikand)LSD

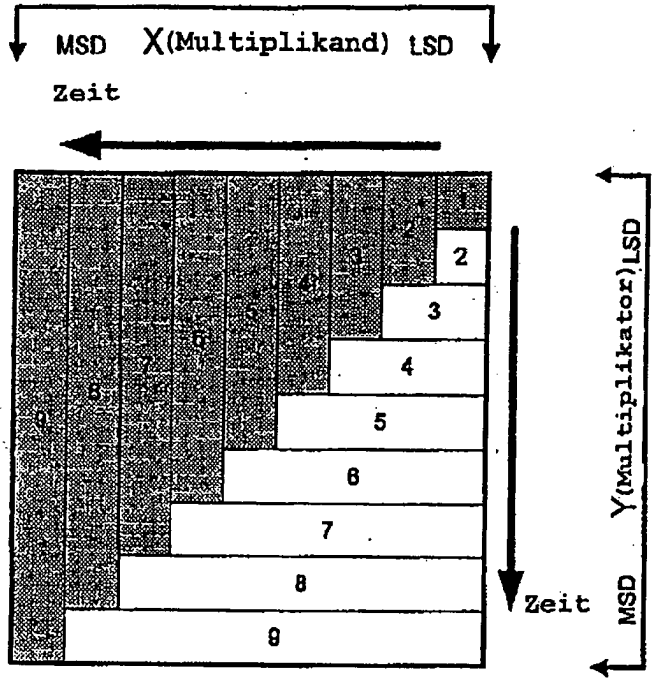
Figur 10b



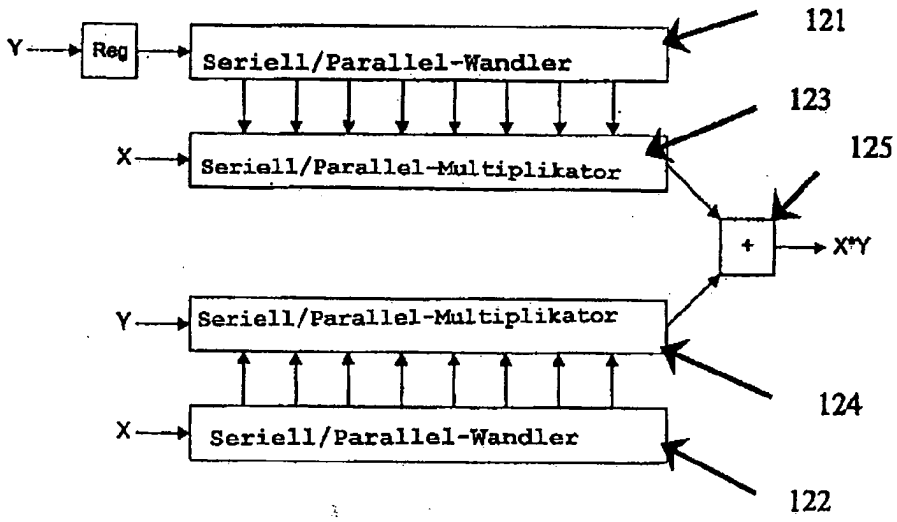
Figur 11a



Figur 11b



Figur 12a



Figur 12b