21 Systolische Arrays

Zusammen mit einigen Algorithmen als Beispiele für Entwurfsprobleme werden in diesem Kapitel systolische Arrays (SA) auf anschauliche Weise eingeführt. SAs sind regelmäßige Arrays von PEs (processing elements) auf folgende Weise:

- Außer dem Taktgeber wird keinerlei Steuerung benötigt
- PEs kommunizieren nur mit ihren nächsten Nachbarn (VLSI-gerecht: wiring by abutment!)
- Ein- und Ausgabe des Array erfolgt nur durch die PEs am äußersten Rand des Array
- "Datenströme" (zeitlich / räumlich geordnet) werden durch den Array hindurch getaktet

Dieses Kapitel berührt die in Bild 21.2 gezeigten Abstraktions-Ebenen.

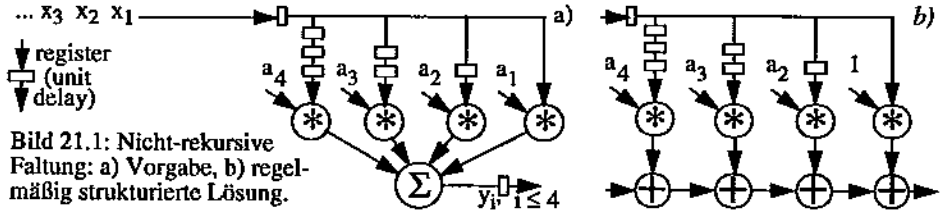


Bild 21.1: Nicht-rekursive Faltung: a) Vorgabe, b) regelmäßig strukturierte Lösung.

21.1 Die Faltung als einführendes Beispiel

SAs haben viele Anwendungen, wie z.B.[10]. Die nicht-rekursive Faltung soll in diesem Abschnitt die Anwendung eines SAs veranschaulichen. Dies ist ein einfaches Beispiel mit einem nur eindimensionalen Datenstrom. Es handelt sich um die Faltung eines eindimensionalen Datenstromes unter Verwendung eines Fensters mit einer konstanten Länge k [4]. Gegeben sei eine Folge $x_1, x_2, x_3, \ldots$, von Daten. Für jede ganze Zahl $i \leq k$ wird folgender Wert berechnet:

$$y_i = a_1 \times x_i + a_2 \times x_{i-1} + a_3 \times x_{i-2} + \ldots + a_k \times x_{i-k+1} \qquad (21.1)$$

wobei die Gewichte $a_1, a_2, \ldots a_k$ feste Koeffizienten sind. Dies läßt sich wie folgt zusammenfassen:

$$y_i = \sum_{j=1}^{k} a_j x_{i-j+1} \qquad (21.2)$$

Bild 21.1 zeigt eine bekannte Lösung der numerischen Filtertheorie mit k = 4.

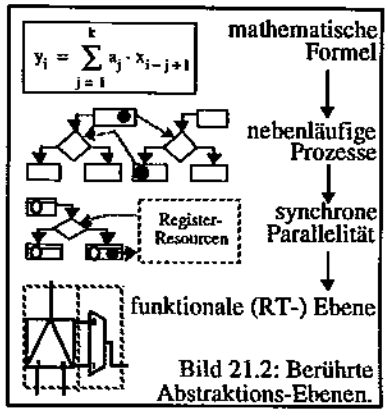


Bild 21.2: Berührte Abstraktions-Ebenen.

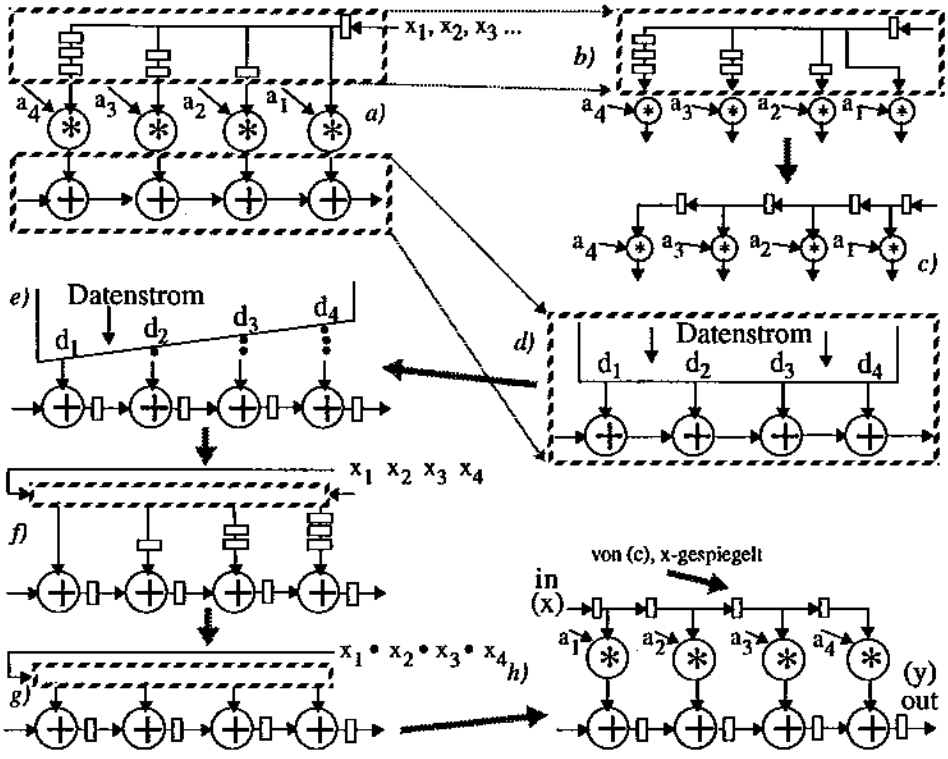


Bild 21.3: Umwandlung des Filters in Bild 21.1 (Bild a) in eine systolisierte Version; c-d) Umwandlung der Verzögerungsharfe (b) aus Bild 21.1 in eine Verzögerungs-Pipeline (c), d-h) Umwandlung der Addiererkette (a) in eine Addierer-Pipeline (g), f) Verzögerung der Eingänge zwecks Äquivalenz m.(e), g) Beseitigung der Verzögerungsharfe in (f) durch Einfügung v. Leerworten in die Eingabefolge (Folge wird 2-slow[9]) h) vollständig umgewandeltes Filter.

Das Funktionsprinzip ist einfach. Die Rechtecke im Bild 21.1 a und b symbolisieren getaktete Register. Zu jedem Taktzeitpunkt wird ein neues x_i in das System (Bild 21.1 a) von außen eingegeben. Die k Multiplikationen werden parallel durchgeführt. Der akkumulierende Summa-

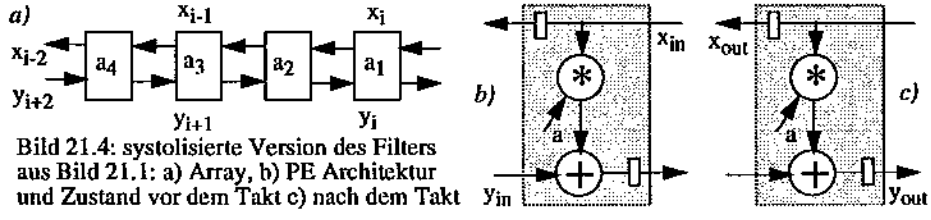


Bild 21.4: systolisierte Version des Filters aus Bild 21.1: a) Array, b) PE Architektur und Zustand vor dem Takt c) nach dem Takt

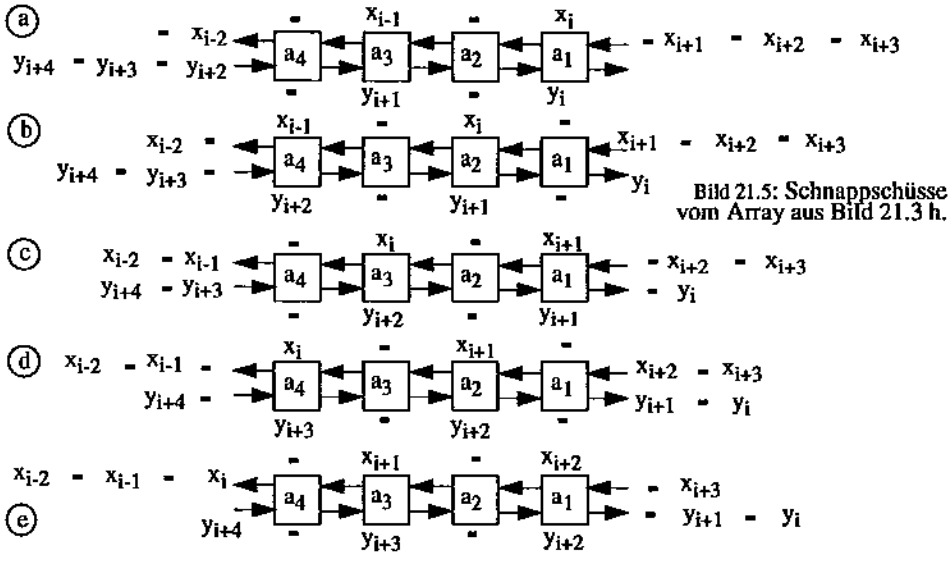


Bild 21.5: Schnappschüsse vom Array aus Bild 21.3 h.

tions-Operator Σ (Bild 21.1 a) kann durch eine Kette parallel arbeitender dyadischer Addierer (Bild 21.1 b) realisiert werden (der erste Addierer addiert den Wert Null, da eigentlich nur drei Addierer benötigt würden.). Für jeden Taktzyklus r_k wird ein y_i berechnet.

Bild 21.3 zeigt die Umwandlung des Filters aus Bild 21.1 in eine systolische Version. Bild 21.3 zeigt das Ergebnis dieser Umwandlung in der abstrakteren Darstellungsweise der Szene für systolische Arrays: partitioniert in ein Feld (a) einfacher PEs (b, c) (PE steht für *processing element*, ein einfacher Prozessor, in der Regel ohne eigenes Steuerwerk). Bild b und c zeigen Schnappschüsse zur Veranschaulichung des Hindurchtaktens der Daten durch solche PEs: Bild b zeigt den Zustand vor dem Taktungsereignis, Bild c nach dem Takt. Bild 21.5 veranschaulicht das Hindurchtakten der Daten durch den gesamten Array durch einen Ausschnitt aus der Folge von Schnappschüssen.

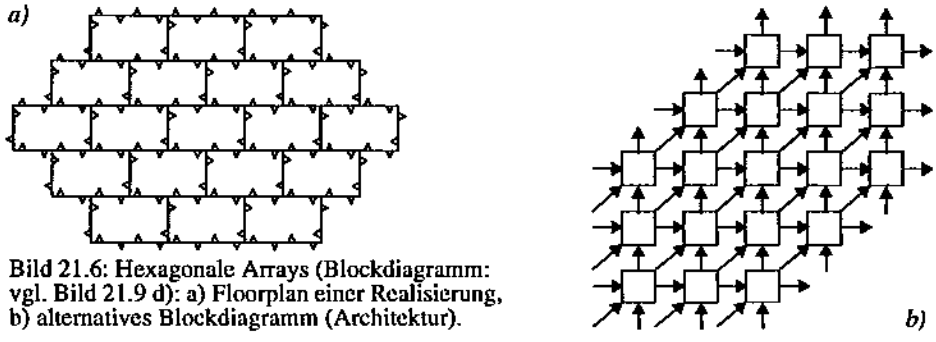


Bild 21.6: Hexagonale Arrays (Blockdiagramm: vgl. Bild 21.9 d): a) Floorplan einer Realisierung, b) alternatives Blockdiagramm (Architektur).

21.2 Formen-Vielfalt systolischer Arrays

Eine Vielfalt von Strukturen systolischer Arrays ist realisierbar. Aus der Einschränkung, daß nur Nachbar-PEs miteinander kommunizieren können, ergeben sich beispielsweise Strukturen, wie sie in Bild Bild 21.9 gezeigt sind. Array nach Bild d ist gemäßt Bild 21.6 realisierbar.

Datenströme. Meist werden parallele Datenfolgen in einen SA vom Rand her eingegeben, wie Bild 21.7 zeigt. Solche in den SA zu taktenden parallelen Daten-Queues (Daten-Warteschlangen) werden ***Datenströme*** genannt. Bild 21.8 führt hierzu das als graphische Notation übliche **Datenstrom-Diagramm** ein (s. a. Bild 21.10), eine anschauliche Darstellung für *schedules* (s. S. 516), die externe räumliche Bewegung der Daten zu den PE-Eingängen modellierend.

21.2.1 Matrix-Vektor-Multiplikation

In diesem Abschnitt sollen mehrere alternative systolische Arrays für die Matrix-Vektor-Multiplikation veranschaulicht werden [9]. Der Einfachheit halber werden diese Beispiele mit einer (3, 3)-Koeffizientenmatrix gewählt nach dem folgenden Schema:

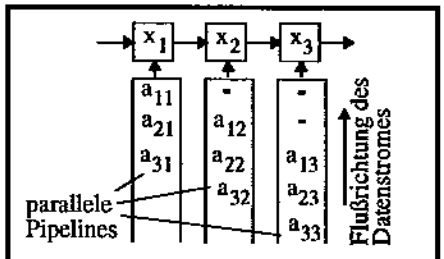


Bild 21.7: Pipeline-Modell zur Eingabe von Datenströmen am Rande von SAs.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \qquad (21.3)$$

Diese Matrix-Vektor-Operation ist eine Notation für die folgenden -Berechnungen:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= y_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= y_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= y_3 \end{aligned} \qquad (21.4)$$

Insgesamt sind also $3 \cdot 3 = 9$ Multiplikationen durchzuführen. Je Gleichung sind 3 Multiplikationen Teil einer MAC-Operation (MAC steht für multiply / accumulate), wobei die folgende Sequenzen von Zwischenwerten auftreten:

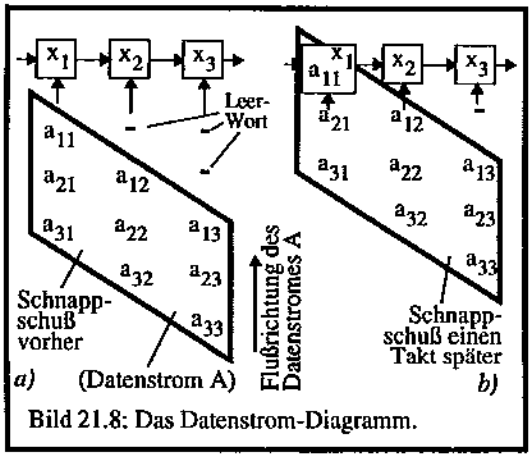


Bild 21.8: Das Datenstrom-Diagramm.

$$\begin{aligned} y_1^{(0)} &= 0 \\ y_1^{(1)} &= a_{11}x_1 + y_1^{(0)} \\ y_1^{(2)} &= a_{12}x_2 + y_1^{(1)} \\ y_1^{(3)} &= a_{13}x_3 + y_1^{(2)} \end{aligned} \qquad (21.5)$$

Die MAC-Sequenz darf auch die folgende umgekehrte Reihenfolge haben:

$$\begin{aligned} y_1^{(0)} &= 0 \\ y_1^{(1)} &= a_{13}x_3 + y_1^{(0)} \\ y_1^{(2)} &= a_{12}x_2 + y_1^{(1)} \\ y_1^{(3)} &= a_{11}x_1 + y_1^{(2)} \end{aligned} \qquad (21.6)$$

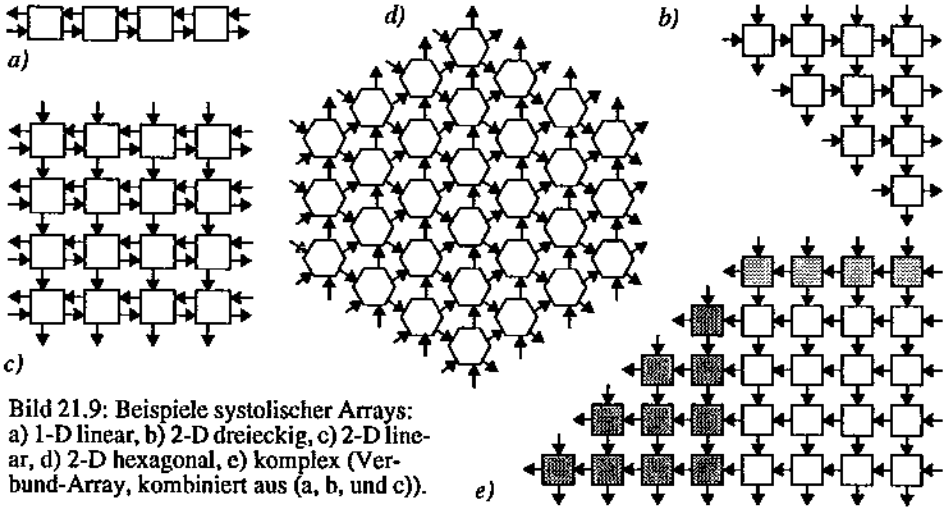


Bild 21.9: Beispiele systolischer Arrays: a) 1-D linear, b) 2-D dreieckig, c) 2-D linear, d) 2-D hexagonal, e) komplex (Verbund-Array, kombiniert aus (a, b, und c)).

Bild 21.10 zeigt vier alternative systolische Arrays für die Matrix-Vektor-Multiplikation. Beim ersten Array (Bild 21.10 e) gehen drei Datenströme mit x_i, y_i und a_{ij} durch den Array hindurch mit den (im Gegensatz zum systolischen Array für die Faltung, durch welchen nur zwei Datenströme mit x_i und y_i hindurchgehen). Der dritte dieser Datenströme besteht aus den Koeffizienten a_{ij} (im Gegensatz zur Faltung, wo die Koeffizienten a_i im PE fest verdrahtet sind. Bild 21.11 zeigt die zu diesem Array gehörige Schnappschußfolge, wobei zu flächensparender Darstellung die x_i- und y_i-Folgen oft nur teilweise gezeigt sind. Bild 21.10 f bis Bild 21.10 h zeigen Arrays, bei denen jeweils eines der drei Datenfelder im Array gespeichert ist. Bild 21.10 f zeigt den Fall, daß die Koeffizientenmatrix im Array gespeichert ist. Bild 21.10 g bzw. h zeigen Lösungen bei denen der Vektor $y^{(0)}$ bzw. der Eingabevektor $x = \{x_i\}$ im Array gespeichert ist.

Für die Matrizen-Multiplikation (Matrix-Matrix-Multiplikation) gibt es ein ganzes Sortiment verschiedener systolischer Arrays [1] [7] [9] (die sich durch unterschiedliche Größe und Effizienz unterscheiden), wie hexagonale systolische Arrays [2] [5] [6], oder orthogonale systolische Arrays nach Hwang und Cheng, oder nach Jover, Kailath und Schreiber [3], oder aber eindimensionale und dreidimensionale Arrays [8]. Eine typische Anwendung hierfür ist die lineare Transformation, eine direkte Berechnung der Koordinaten eines Punktes beim Übergang von einem ersten Koordinatensystem in ein zweites Koordinatensystem und von diesem dann in ein drittes Koordinatensystem [1]. Bei der Steuerung eines Roboterarmes sind beispielsweise bis zu sieben solcher Transformationen für jeden Punkt auszuführen. Zu den vielen Beispielen gehören die Matrix-Matrix-Multiplikation [1] und die Gauß'sche Elimination [9], nebst LU-Zerlegung, Rückwärts-Elimination, Matrix-Invertierung, und Transformation in eine Dreiecksmatrix [1].

21.3 Synthese und Optimierung mit Retiming

Dieser Abschnitt führt die Anwendung von Retiming ein, sowohl zur Synthese systolischer Arrays, als auch allgemein zur Optimierung synchroner Systeme, beispielsweise zur Optimierung

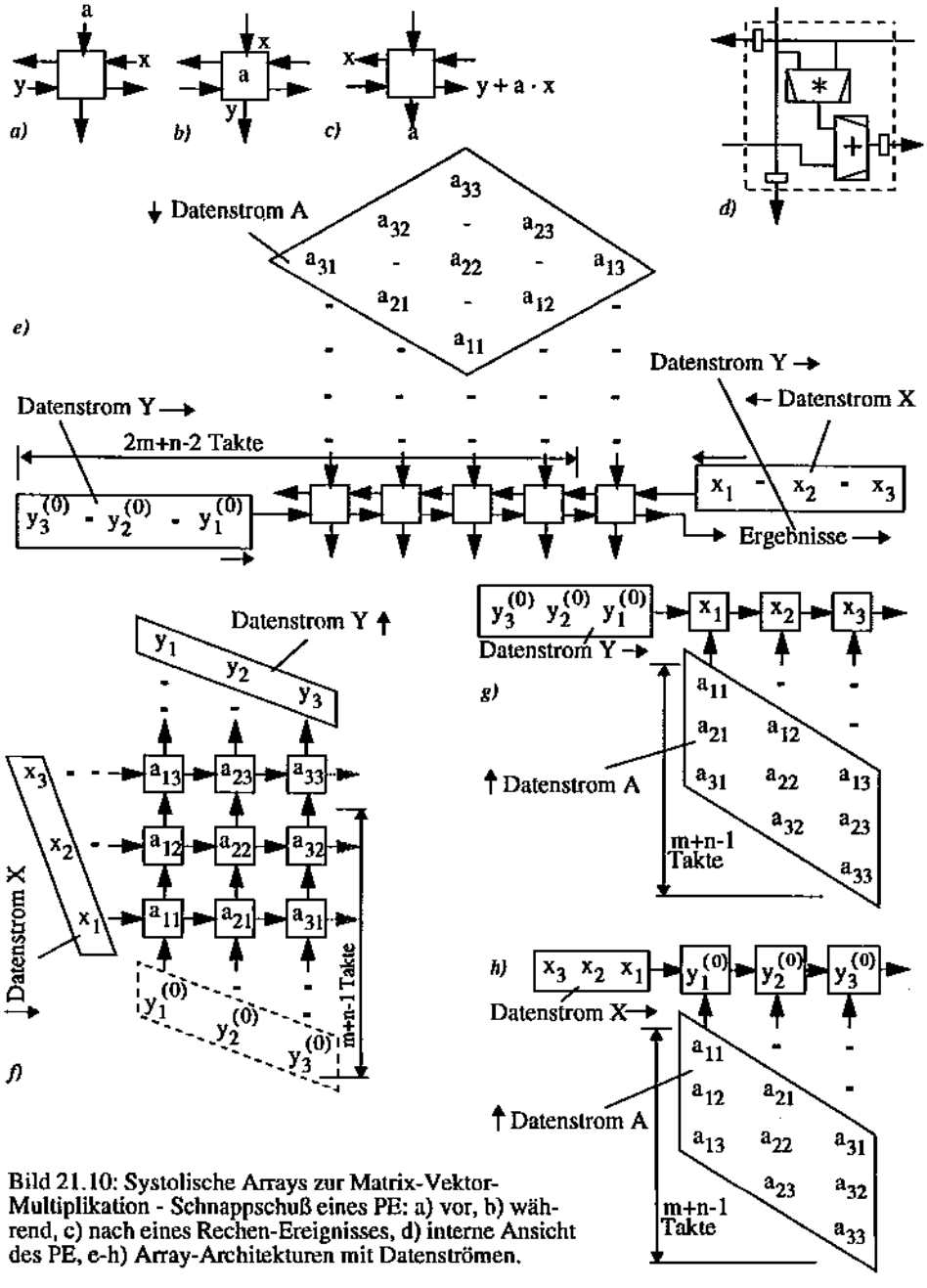


Bild 21.10: Systolische Arrays zur Matrix-Vektor-Multiplikation - Schnappschuß eines PE: a) vor, b) während, c) nach eines Rechen-Ereignisses, d) interne Ansicht des PE, e-h) Array-Architekturen mit Datenströmen.

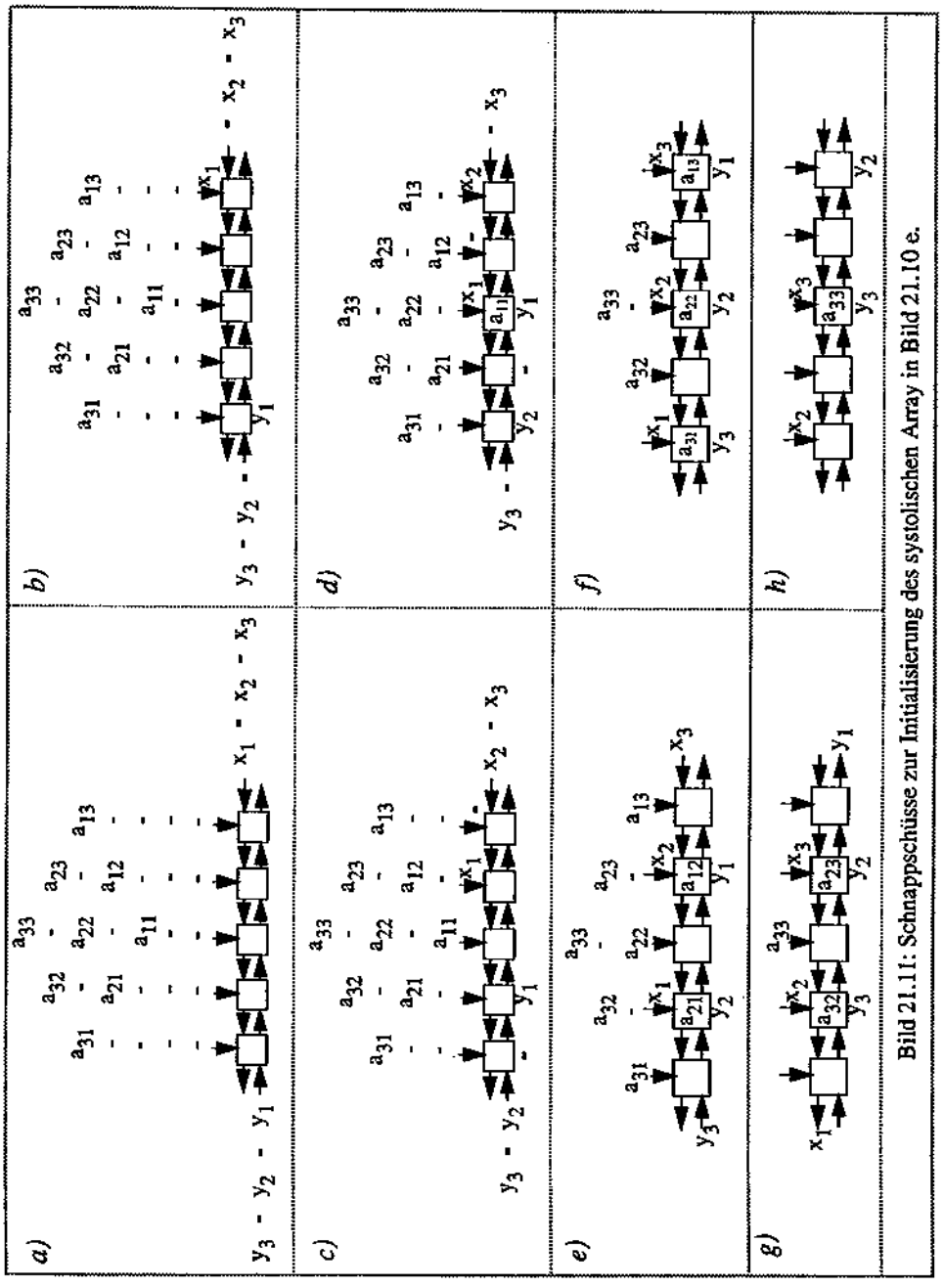


Bild 21.11: Schnappschüsse zur Initialisierung des systolischen Array in Bild 21.10 e.

des Durchsatzes. Wir beginnen mit der Synthese. Die mehr für diese Methode spezifischen Teile der Prozedur bestehen aus folgenden Schritten (mehr Details in Kapitel 22):

1. Aufstellen und Lokalisierung des DDG (Abschn. 22.2.2)
2. Abbildung des DDG in einen SFG
3. Überführung des SFG in ein systolisches Array.

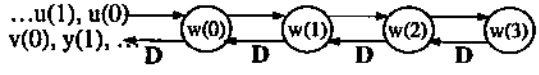


Bild 21.12: SFG eines Arrays für die Faltung.

Bisher galt:
$$\vec{s}^T \cdot \vec{e} \geq 0;\ \vec{s}^T \cdot \vec{d} > 0. \tag{21.7}$$

SFG steht für *Signalflußgraph.* Für systolische Arrays gilt ganz grob:
$$\vec{s}^T \cdot \vec{e} > 0;\ \vec{s}^T \cdot \vec{d} > 0. \tag{21.8}$$

DDG steht für *Datenabhängigkeits-Graph (data deoencency graph).* Oft müssen systolische Designs auf Erfüllung weiterer Bedingungen überprüft werden: es muß geprüft und sichergestellt werden, daß wirklich jede Kante des resultierenden SFG mindestens eine Verzögerung hat. Denn manche SFGs sind wirklich lokalisiert und noch nicht zeitlich voll serialisiert (Gegenbeispiel: der SFG für die Faltung gem. Bild 21.12).

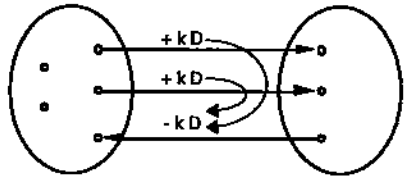
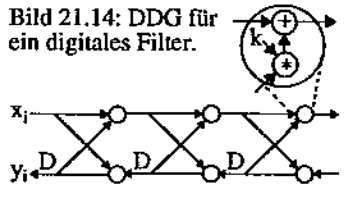


Bild 21.13: Cut-Set-Veranschaulichung.

Der Datenstrom u wird vom linken PE zu den rechten PEs in 0 (Null) Zeitschritten per Broadcasting propagiert. Dies ist nicht wünschenswert für ein VLSI-gerechtes systolisches Design. Der SFG nach Bild 21.12 ist jedoch der VLSI-gerechten Lösung relativ nahe. Er muß nur noch durch Retiming lokalisiert werden. Deshalb gilt hier die folgende Kurzfassung einer Methode: Systolisches Array = SFG-Array + Pipeline-Retiming. Eine Möglichkeit besteht in der Anwendung des Cut-Set Retiming [7].

Bild 21.14: DDG für ein digitales Filter.



21.3.1 Retiming synchroner Systeme

Unter Retiming wird eine Folge von Äquivalenz-Transformationen verstanden, die ein synchrones System in ein anderes solches mit gleicher Gesamt-Datenoperation aber unterschiedlichem Timing umwandelt. Retiming erfolgt durch die Verschiebung von Verzögerungen bzw. Registern an andere Stellen innerhalb des Operatoren-Netzwerkes. Dies wird im Folgenden durch Beispiele veranschaulicht. Im Wesentlichen kann Retiming durch sukzessive lokale oder auch globale Anwendung folgender Regeln durchgeführt werden. Eine solche lokale Transformation kann jeweils an einer Schnittmenge (engl.: *cut set*) erfolgen (Definition auf Seite 441 und 442), die ein System in 2 Subnetze zerteilt (veranschaulicht in Bild 21.13).

Regel 1 (Zeit-Skalierung): Alle Delays D (Verzögerungsglieder) dürfen mit demselben Faktor a (a ist natürliche Zahl) skaliert werden: D → a D'.

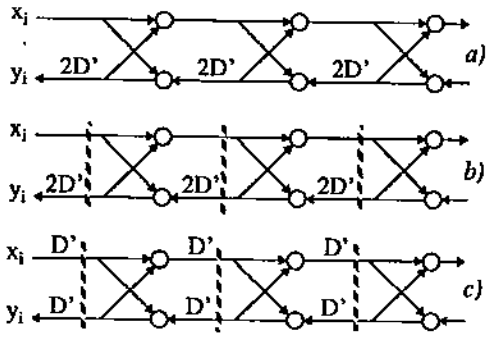


Bild 21.15: Retiming: Filter nach Bild 21.14.

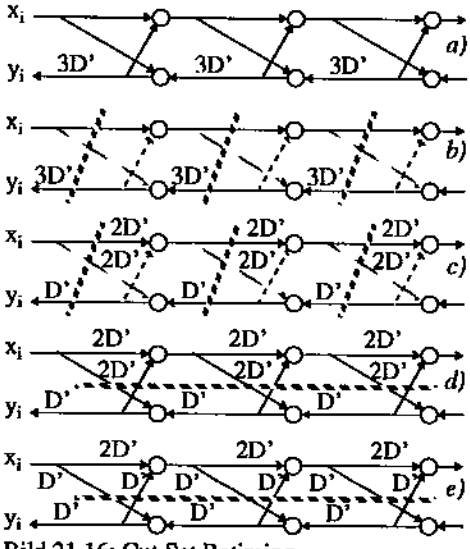


Bild 21.16: Cut Set Retiming am Beispiel eines Sortierers.

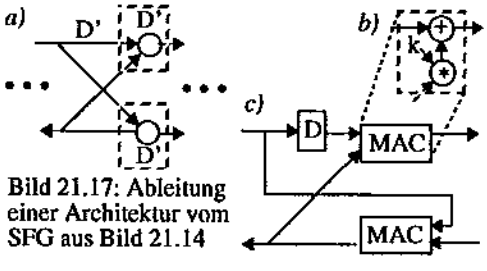


Bild 21.17: Ableitung einer Architektur vom SFG aus Bild 21.14

Regel 2 (Delay-Transfer): Die Gesamtheit aller Delays, die in Kanten gleicher Richtung eines *Cut Set* liegen, darf nach der Gesamtheit aller Kanten der Gegenrichtung dieses Cut Set verschoben werden.

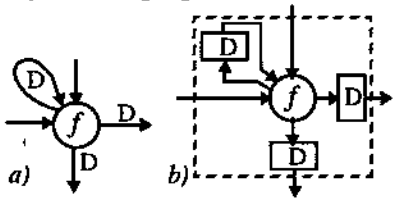
Ein Cut Set ist eine Menge von Kanten, die alles SFG in zwei Teile teilt (vgl. Bild 21.13). Wichtig ist dabei: Die Ein/Ausgabe-Beziehungen bleiben gleich, wenn die Eingabe im gleichen Teil des DDG stattfindet, wie die Ausgabe. Sonst ist eine Anpassung der Ein- bzw. Ausgabe durch Hinzufügen von Delays (Scherung der Eingabe) nötig. Für den Fall, daß mehr als ein Cut-Set betroffen ist, akkumulieren sich die Delays.

Die Anwendung von Regel 2 sollte in gleicher Weise für alle Elemente des DDG erfolgen. Zur Veranschaulichung folgendes Beispiel: ein Digitales Filter (Bild 21.14, mit Architektur nach Bild 21.17; Bild 21.18 veranschaulicht das Prinzip der SFG-zu-Architektur-Umwandlung.). Wir zeigen zwei verschiedene Möglichkeiten des Retiming. Die erste Möglichkeit des Retiming eines digitalen Filters (Typ A) wird in Bild 21.15 gezeigt. Eine zweite Möglichkeit des Retiming eines digitalen Filters (vom Typ B) wird durch Bild 21.16 veranschaulicht.

21.3.2 Optimierung systolischer Arrays

Retiming ist für alle getakteten Systeme anwendbar, also auch für systolische Arrays. Eine Rückkopplungskante (Selbstschleife: Datenabhängigkeit, die zu demselben PE zurückführt) im Cut Set sollte im Allgemeinen nicht geschnitten werden. Man kann sich solche Selbstschleifen in die PEs integriert vorstellen, d.h. zu den Operatoren ge-

Bild 21.18: Prinzip der SFG-zu-Array-Wandlung (vgl. Bild 21.17).



rechnet (Bild 21.18 a). Nach der Optimierung wird der SFG wieder in die Architektur zurückgewandelt, was in Bild 21.14, Bild 21.17 und Bild 21.18 gezeigt wird.

Finden eines guten Cut-Set (Schnittmenge)

Ein Cut-Set ist eine minimale Zahl von Kanten, die den Graph in zwei Teile teilt. Zu einem guten Cut-Set gehören nur: die zu bearbeitende Kante (Ziel-Kante), Kanten und Verzögerungen (delays) in jede beliebige Richtung, und Kanten ohne Verzögerungen in der gleichen Richtung wie die Ziel-Kante, sowie: letztlich keine negativen Delays. Retiming ist eine einleuchtende Methode, sollte aber stärker formalisiert werden. Dies gilt speziell für die Behandlung von Ein- / Ausgabe: z.B. Skew (s. S. 516) von Matrizen.

Bild 21.19: Cut-Sets bei der Matrix-Matrix-Multiplikation

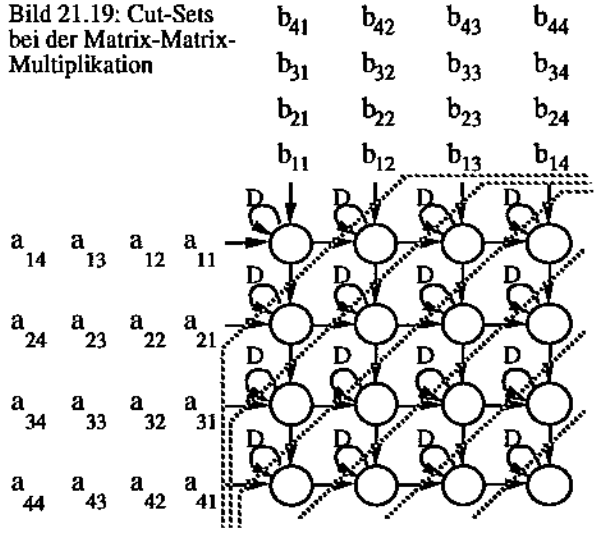


Bild 21.19 zeigt Beispiele von Schnittmengen in einem Graphen für Matrix-Matrix-Multiplikation. Mitunter sind auch negative Verzögerungen sinnvoll: nämlich beispielsweise als Zwischenlösungen bei einer aus mehreren Retiming-Schritten bestehenden Optimierungsprozedur. Im Endergebnis dürfen allerdings keine negativen Verzögerungen mehr vorhanden sein.

21.4 Effizienz-Analyse systolischer Arrays

Als mögliche Zielfunktion existieren mehrere Optimierungskriterien. Deren sinnvolle Auswahl ist von der Anwendung abhängig. Optimierungskriterien können sein: Flächenbedarf des implementierten Arrays, die Anzahl der Prozessoren, die Leistungsaufnahme u. a. m. Ein solches Optimierungskriterium ist auch die Berechnungszeit (letzterer Begriff und andere sind auf Sete 443 definiert).

Bild 21.21 veranschaulicht einführend die Effizienz-Analyse einer Version mit k=5 des Arrays aus Bild 21.3 bzw. Bild 21.5. Ein PE ist immer nur dann aktiv, wenn es gleichzeitig Dreieck und Quadrat enthält. Der Aktivitätsfaktor gibt für eine jeweilige Taktperiode an, wieviele PEs

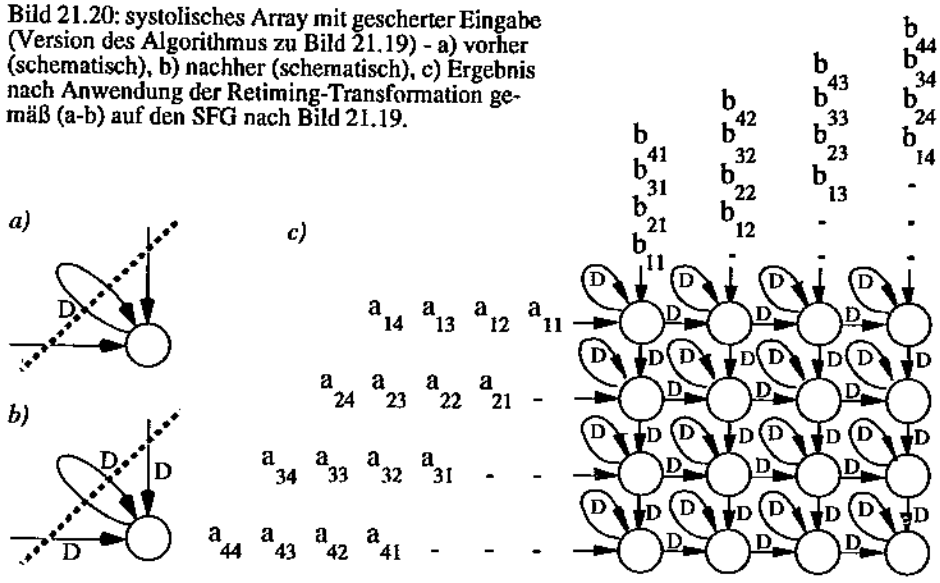


Bild 21.20: systolisches Array mit gescherter Eingabe (Version des Algorithmus zu Bild 21.19) - a) vorher (schematisch), b) nachher (schematisch), c) Ergebnis nach Anwendung der Retiming-Transformation gemäß (a-b) auf den SFG nach Bild 21.19.

aktiv sind. Man sieht, daß bei diesem Array höchstens jedes zweite PE innerhalb der Reihe aktiv sein kann. Petkov nennt einen solchen Array *2-slow* [9].

Interessant ist auch ein Vergleich der vier verschiedenen systolischen Arrays für die Matrix-Vektor-Multiplikation in Bild 21.10 e-h, wo jeweils die Latenzzeiten eingetragen sind. Die Latenzzeit der Arrays nach Bild 21.10 f-h sind mit m+n-1 gleich groß (während diejenige des Array nach Bild 21.10 e mit 2m+n-2 größer ist). Trotz gleicher Latenzzeiten ist mit dem Array nach Bild 21.10 f durch höhere Parallelität wegen der viel größeren Anzahl von PEs ein höherer Durchsatz zu erzielen als mit den Arrays nach Bild 21.10 g-h, sofern ein kontinuierlicher Strom von Aufgaben hindurchgetaktet wird. Der Klarstellung dienen folgende Definitionen:

Initialisierungszeit. Die Initialisierungszeit ist die Zeit vom Eintreten des ersten Wertes in den Array bis zur Ausgabe des ersten Resultatwertes (bis zur vollständigen Füllung aller Pipes).

Durchsatz (engl.: *throughput*). Der Durchsatz ist die durchschnittliche Anzahl der pro Takt austretenden Resultatwerte im stationären Betriebszustand.

Die Latenzzeit eines Resultat-Datenstroms spezifiziert die Zeit, vom Eintritt des ersten Datenelementes in den Array bis zum Austritt des allerletzten Datenelementes am Ausgang.

Berechnungszeit (engl.: *computation time* T): Die Zeit, die der Array vom Beginn der ersten Berechnung bis zum Abschluß der letzten Berechnung für ein gegebenes Problem benötigt.

Pipeline-Periode. Die Pipeline-Periode, das zeitliche Intervall zwischen zwei aufeinanderfolgenden Berechnungen in einem Prozessor-Element wird mit "α" bezeichnet:

$$\text{Pipelining Rate} \equiv \frac{1}{\alpha} \tag{21.9}$$

Die Block-Pipelining-Periode b ist die Zeit, die vergeht zwischen zwei aufeinanderfolgenden Berechnungen einer Instanz eines Problems.

Die Feld-Größe (Array-Größe) ist die Zahl der Prozessoren im Array.

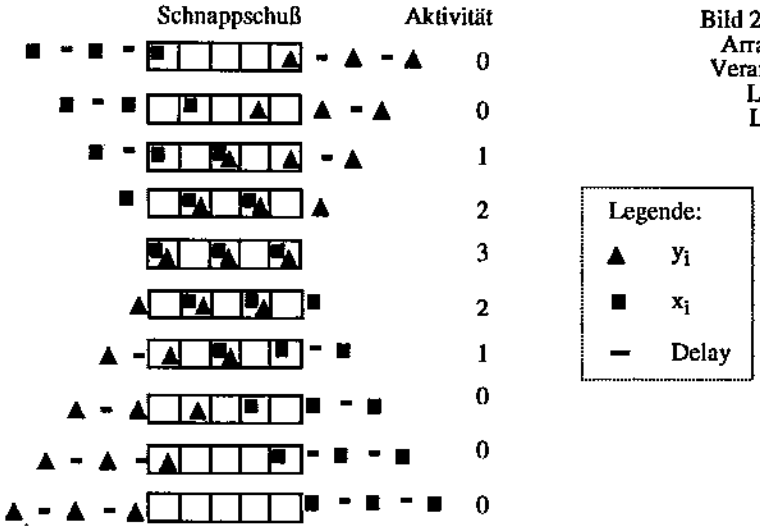


Bild 21.21: Systolischer Array für die Faltung: Veranschaulichung der Leistungs-Analyse. Latenzzeit: L = 10.

Die Zahl der I/O - Kanäle ist die Zahl der Verbindungen zu dem Host-Computer; Auswirkungen auf die Zahl der Pins eines Chips etc.

Die Behandlung der systolischen Arrays, insbesondere der Synthese durch formale Ableitung aus der mathematischen Formel wird an Hand von Beispielen in Kapitel 22 fortgesetzt.

21.5 Literatur

[1] R. Hartenstein: Einführung in den VLSI-Entwurf, Band 1; IT Press Verlag, (in Vorbereitung für 1995)

[2] K. Hwang, F. A. Briggs: Computer Architecture and Parallel Processing, McGraw-Hill, New York 1984

[3] J.M.Jover, T.Kailath: Design Framework for Systolic-type Arrays; Proc. ICASSP 1984

[4] H. T. Kung: The Structure of Parallel Algorithms; Advances in Computers 19, 65-112, 1980

[5] H. T. Kung, C. E. Leiserson: Systolic Arrays for VLSI; Sparse Matrix Proc. 1978, Society for Industrial and Applied Mathematics, 1979, p. 256 - 282

[6] H. T. Kung, C. E. Leiserson: Systolic Array Apparatuses for Matrix Computations; US Patent, 4,493,048/8.J.1985

[7] C. E. Leiserson, F. M. Rose, J. B. Saxe: Optimizing Synchronous Circuitry by Retiming; Proc. 3rd Caltech Conf. on VLSI, Computer Science Press, Rockville, MD, 1983

[8] R. W. Lindermann, W. H. Ku: A Three-dimensional Array Architecture for fast Matrix Multiplication; ICASSP 1984

[9] N. Petkov: Systolische Algorithmen und Arrays; Akademie-Verlag, Berlin 1989

[10] R. Roncella, R. Saletti, P. Terreni: 70-MHz 2-µm CMOS Bit-Level Systolic Array Median Filter; IEEE J. of Solid-State Circuits, SSC-28,5 (May 1993)