

20 Strukturierte Entwürfe

Abschnitt aus Kapitel 20 des Buches: R. Hartenstein: Standort Deutschland: Wozu noch Mikrochips? Einführung in Methoden der Technischen Informatik; ITpress 1964

20.4 Algebraischer Entwurf regelmäßiger Strukturen

Eine dazu geeignete Hardware-Beschreibungs-Sprache, wie beispielsweise KARL-3 [12] kann als Entwurfs-Kalkül für den strukturierten VLSI-Entwurf benutzt werden. Im folgenden wird dies mit Hilfe eines Beispiels veranschaulicht. Neben den topologischen Ausdrucksmitteln (vgl. Abschnitt 20.3) sind sogenannte Verdrahtungs-Funktionen ein wichtiges Werkzeug hierzu. Im folgenden Abschnitt wird hieraus zur Entwicklung eines Multiplizierers eine Schifter-Algebra zusammengestellt. Ein solcher topologisch-funktionaler Kalkül unter Einbezug von Verdrahtungsfunktionen ist auch eine gute Grundlage für die Implementierung von Modul-Generator-Generatoren, wie bei [22] auf der Basis von KARL-3.

20.4.1 Verdrahtungsoperatoren

Die Definition von Verdrahtungs-Operatoren [26] erlaubt die kompakte symbolische Einbindung von Verdrahtungsmustern in funktionale Beschreibungen, wie beispielsweise in einfachen Ausdrücken, in funktional topologischen Ausdrücken oder in anderen Hardware-Beschreibungen. Ein Beispiel die KARL-Verdrahtungsfunktionen gemäß folgender Grammatik [12]:

$$\text{func_name ['&' func_parameter] '(' operand ')}$$

wozu die Schift-Funktionen gehören, die sich wie folgt gliedern lassen:

- **Bit-Schift-Funktionen** (Verschiebung der Reihenfolge von Bits in einem Wort),

In KARL werden diese $x\mathbf{shl}$ genannt (Links-Schift), oder $x\mathbf{shr}$ (Rechts-Schift)

- **Wort-Schift-Funktionen** (Verschiebung der Reihenfolge von Worten in einem Feld),

In KARL werden diese $x\mathbf{push}$ genannt (Links-Schift), oder $x\mathbf{pop}$ (Rechts-Schift). Der oben durch x freigehaltene Anfangsbuchstabe hat die folgende Mnemonische Bedeutung:

Präfix	Mnemonik	ser. Eingabe	z.B.	Präfix	Mnemonik	ser. Eingabe	z.B.
(entfällt)	(starring)	* (don't care)	shr	d	destructive	0	dshr
c	constructive	1	cshr	e	left field <u>e</u> xtension		eshr
ci	circular	(Ring-Schift)	cir	n	non-destructive	from source	nshr

Beispiele mit Verdrahtungsmuster zur Veranschaulichung sind in Bild 19.28 gezeigt. Der Funktions-Parameter, ein '&' gefolgt von einer dezimalen ganzen Zahl (oder einem entsprechenden Ausdruck) gibt die Schrittweite der Funktion an. Beispielsweise ein $\mathbf{shl\&4}$, gibt einen Schift um 4 Bit nach links an. Wenn kein Funktions-Parameter angegeben ist, wird '1' als De-

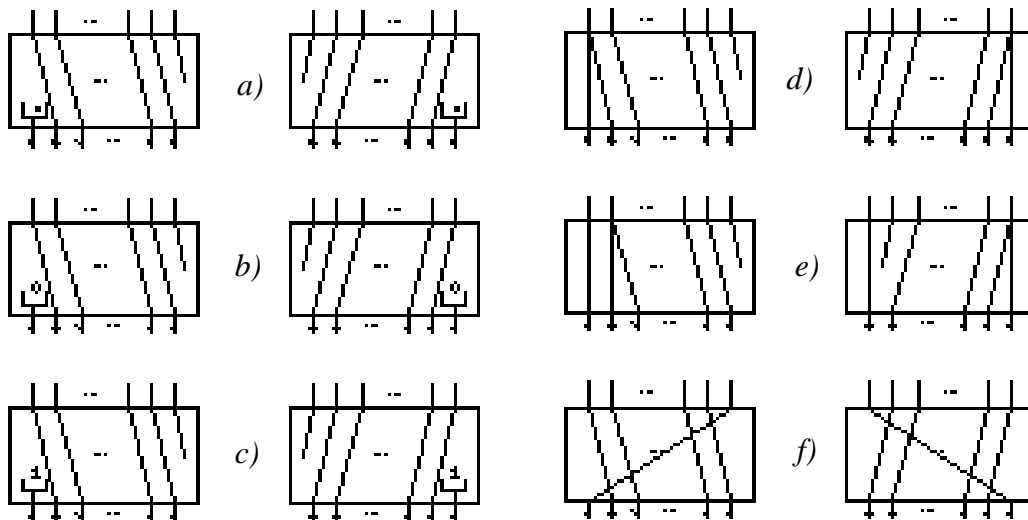


Bild 19.28: KARL-3 Schift-Funktionen (jeweils Links- u. Rechts-Schift): a) starting, b) destruktiv, c) constructive, d) nicht-destruktiv, e) field extension, f) zirkulär.

fault-Wert genommen. Eine systematische Anwendung einer Schifter-Algebra wird in Abschnitt 20.4.2 eingeführt [7][12].

20.4.1.1 Andere Verdrahtungs-Funktionen

Ein *Shuffle*-Operator basiert auf einem Verdrahtungsmuster, wie beispielsweise für mehrstufige Verbindungsnetzwerke für Mehrrechnersysteme [27]. Der Funktionsparameter '&' ist die Zielschrittweite (*destination step width*), ein wichtiger Parameter dieses Operators (s. a. Bild 19.30 a). Wird dieser Parameter weggelassen, so wird ein *default* Wert von '2' verwendet (s. Bild 19.30). Der entsprechende Operator wird mit diesem Parameterwert häufig als *perfect shuffle* bezeichnet. In KARL unterscheiden wir *bit shuffle* (hier genannt **fold**-Operator) und *word shuffle* (hier genannt: **merge**-Operator). Verwandt mit dem Shuffle-Muster ist das *Butterfly*-Interkonnektmuster (vgl. Bild 20.16).

Bild 19.29 zeigt ein Beispiel für das Verdrahtungsmuster, das durch die mirror-Funktion präzise beschrieben, wie in KARL durch die Funktion **reflect** oder **reverse**. Die **reflect**-Funktion kehrt die Bit-Folge in einem Wort um, wohingegen **reverse** die Wortfolge in einem Feld umkehrt. Die KARL-Standardfunktion **msb** (*most significant bit*) selektiert das höchstwertige Bit eines Wortes, die Standardfunktion **msw** (*most significant word*) hingegen das "höchstwertige" (äußerste linke) Wort aus einem Feld. Die KARL-Funktionen **lsb** und **lsw** selektieren ein Bit oder Wort am anderen Ende der Quelle des Pfades: das niedrigstwertige Bit (*least significant bit*), bzw. das geringstwertige Wort (*least significant word*).



Bild 19.2 Funktion

20.4.2 Algebraischer Entwurf eines Multiplizierers

Folgendes Beispiel zeigt die algebraische Gewinnung eines strukturierten Entwurfes. Als Beispiel dient der Entwurf eines Multiplizierers für nicht negative ganze Zahlen. Aus einem Multiplizanden X und einem Multiplikator Y soll das Produkt P gewonnen werden. Zuerst wird ein Satz von Regeln (19.2) bis (19.6) einer Schift-Algebra zusammengestellt.



$$2^i \cdot w = i \text{ shl}(w) \tag{19.2}$$

heißt: die Multiplikation von w mit 2^i entspricht dem Schift von w um i Bitstellen nach links.

$$i \text{ shl}(w) = -i \text{ shr}(w) \tag{19.3}$$

heißt: Linksschift um i Bitstellen entspricht einem Rechtsschift um $-i$ Bitstellen.

$$i \text{ shl}(j \text{ shl}(w)) = (i + j) \text{ shl}(w) \tag{19.4}$$

heißt, ein Schift um i , gefolgt von einem Schift um j , entspricht einem Schift um $i+j$.

$$i \text{ shl}(v) + i \text{ shl}(w) = i \text{ shl}(v + w) \tag{19.5}$$

heißt, daß ein Schift um i , gefolgt von einer Addition äquivalent ist einer Addition, gefolgt vom Schift um i . Von (19.3) und (19.4) können wir ableiten:

$$i \text{ shl}(w) = j \text{ shl}((j - i) \text{ shr}(w)) \tag{19.6}$$

Nun wollen wir diese Regeln für den Entwurf eines strukturierten Multiplizierers für nicht-negative ganze Zahlen verwenden. Wir wählen ein Beispiel mit zwei Operanden einer Wortlänge von 4 Bit: einem Multiplikanden X und einem Multiplikator Y der folgenden Form.

$$X = (x_3, x_2, x_1, x_0) \qquad Y = (y_3, y_2, y_1, y_0) \tag{19.7}$$

Das Produkt P erhalten wir durch die folgende mathematische Formel.

$$P = \sum_{i=0}^3 2^i \cdot y_i \cdot X \tag{19.8}$$

Durch Anwendung der Regel (19.2) konvertieren wir diese in die folgende Form.

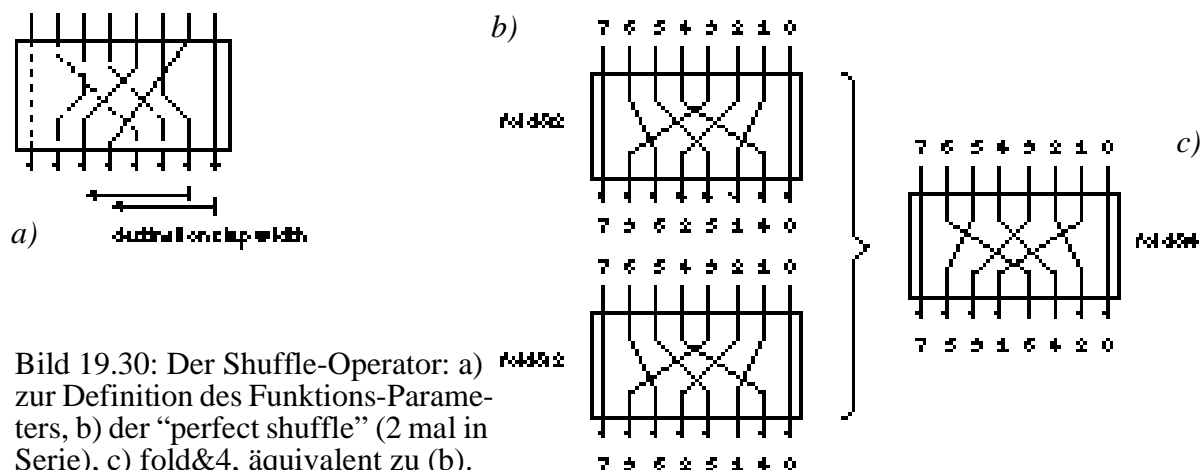
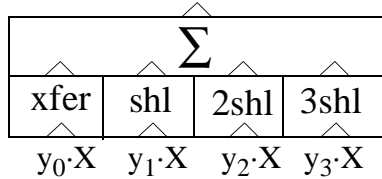


Bild 19.30: Der Shuffle-Operator: a) zur Definition des Funktions-Parameters, b) der "perfect shuffle" (2 mal in Serie), c) fold&4, äquivalent zu (b).



$$P = \sum_{i=0}^3 i \text{ shl}(y_i \cdot X) \tag{19.9}$$

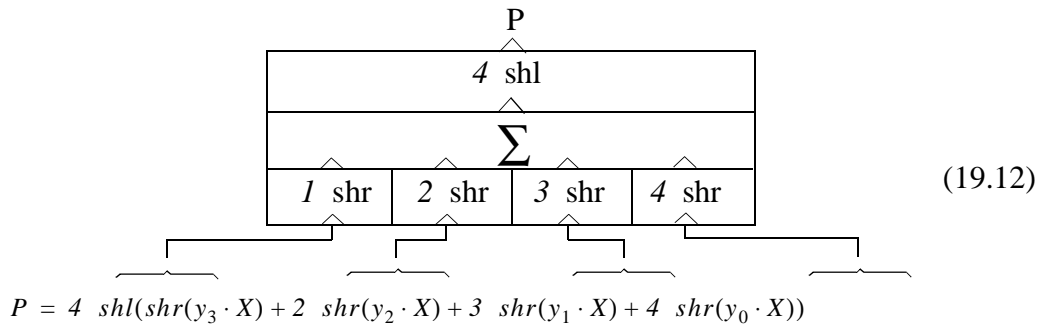
Über Theorem (19.6) erhalten wir die folgende Form.

$$P = 4 \text{ shl} \left(\sum_{i=0}^3 (4-i) \text{ shr}(y_i \cdot X) \right) \tag{19.10}$$

Durch Umkehrung der Akkumulationsfolge von $i = 0, 1, 2, 3$ in $i = 3, 2, 1, 0$ erhalten wir:

$$P = 4 \text{ shl} \left(\sum_{i=3}^0 (4-i) \text{ shr}(y_i \cdot X) \right) \tag{19.11}$$

Folgende detaillierte Form hiervon stellt keinen regelmäßig strukturierten Entwurf dar.



Der Linksschift um 4 Bit am Ausgang, der nicht wirklich significant ist, wird später diskutiert. Durch wiederholte Anwendung der Regel (19.4) erhalten wir die folgende Form.

$$P = 4 \text{ shl}(\text{shr}(y_3 \cdot X + \text{shr}(y_2 \cdot X + \text{shr}(y_1 \cdot X + \text{shr}(y_0 \cdot X)))))) \tag{19.13}$$

Durch Addition von C machen wir keinen Fehler, solange $C = 0$ ist. Dies führt zur Form:

$$P = 4 \text{ shl}(\underbrace{\text{shr}(y_3 \cdot X + \text{shr}(y_2 \cdot X + \text{shr}(y_1 \cdot X + \text{shr}(y_0 \cdot X + C))))}_{\text{ConAddShi}})) \tag{19.14}$$

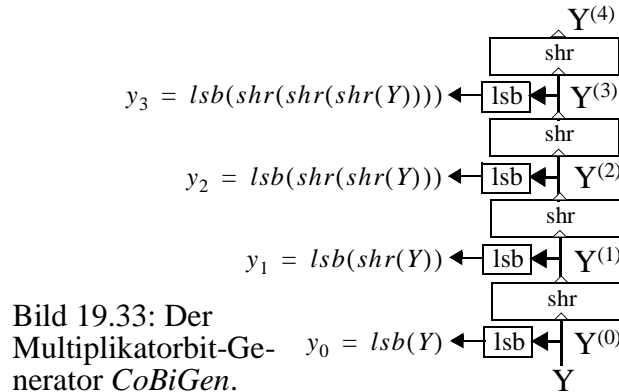
Dies repräsentiert einen regelmäßig strukturierten Design, da wir diesen in vier gleiche Schichten aufteilen können, die wir *ConAddShi* nennen. Da y_i nur ein Einzelbit ist, erhalten wir den folgenden Multiplexer innerhalb von *ConAddShi*:



$$y_i \cdot X = \text{if } y_i \text{ then } X \text{ else } 0 \text{ endif} \tag{19.15}$$

Aus (19.3) erhalten wir Zelle *ConAddShi*, die das ABL-Diagramm in Bild 19.31 a zeigt.

Nun benötigen wir noch eine weitere Schaltung, die alle notwendigen y_i generiert mit $i = 0, \dots, 3$ um alle *ConAddShi* Schichten damit zu speisen. Wir nennen diese Schaltung *Condition Bit Generator*, oder kurz *CoBiGen*. Vom Wort $Y^{(0)}$ (Bild 19.31 c) wird der Selektor y_0 abgeleitet durch Auswahl des geringstwertigen Bit (least significant bit, Bild 19.31 c) durch Anwendung der *lsb* Standardfunktion von KARL. Der Selektor y_1 für die nächste *ConAddShi* Schicht gewinnen wir aus Y , nach rechts geschiftet: von $Y^{(1)} = \text{shr}(Y^{(0)})$ (siehe Bild 19.31 e), wobei $y_1 = \text{lsb}(Y^{(1)})$ der nächsten Schicht ist. Um alle y_i abzuleiten für $i = 0, \dots, 3$ setzen wir eine iterative Folge von Schichten zusammen: der Multiplikatorbit-Generator *CoBiGen* ist in Bild 19.33 gezeigt.



Schließlich setzen wir *CoBiGen* (von rechts) bündig an die 4 Schichten *ConAddShi* (auf der linken Seite). Wir erhalten den vollständigen Multiplizierer aus 4 Schichten *MultiLayer*, wie dieser durch Bild 19.32 a gezeigt wird. Beide *shr* Operatoren in jeder Schicht verbinden wir seriell miteinander von links nach rechts, sodaß der Überlauf des Addierers bei Berechnung der Zwischenprodukte $P^{(1)}, P^{(2)}$, in der Nachbarzelle *CoBiGen* den Platz findet, der frei gemacht wurde durch die bei der Sequenz $Y^{(1)}, Y^{(2)}$, etc. schrumpfende Wortlänge.

Nun können wir den Linksschift um 4 Bitstellen interpretieren der durch (19.11) eingeführt worden war. Durch sukzessives Schiften der partiellen Produkte nach rechts in die Zellen *CoBiGen* haben wir das Produkt nach Stellen geringeren Gewichtes verschoben, was durch Bild 19.33 definiert ist. Dies wird kompensiert durch “4 shl” in Bild (19.12). Bild 19.32 b zeigt

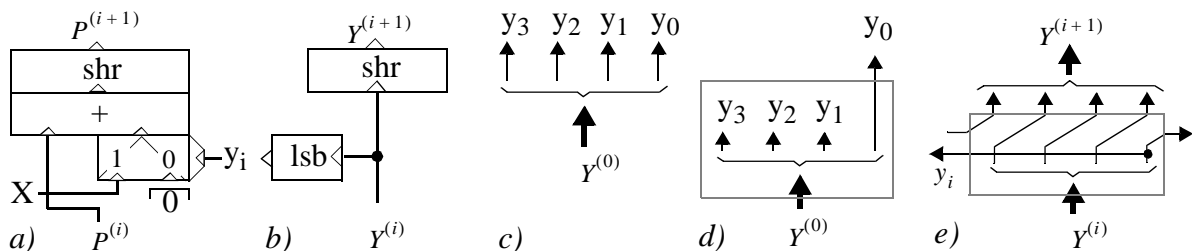


Bild 19.31: Die Slice *MultiLayer*: a) Teil “conditional add-and-shift” *ConAddShi*, b) sein “condition generator” *CoBiGen*, c) Bits des Operanden Y , d) die *lsb*-Funktion, e) *CoBiGen*

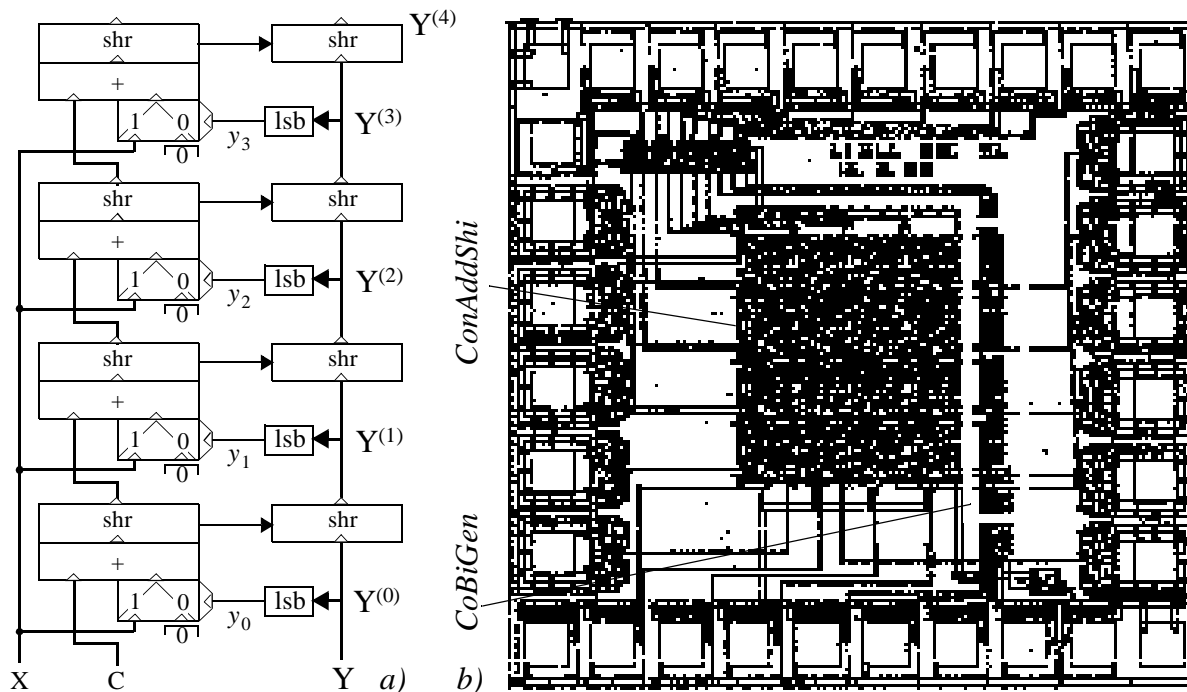


Bild 19.32: Der komplette Multiplizierer: a) Blockdiagramm, b) Layout (Bild vom Scanner).

das nMOS-Layout des Multiplizierers, das 1980 in Kaiserslautern entwickelt wurde und später auf einem Multiprojekt-Chip gefertigt wurde im Rahmen des E.I.S.-Projekt.

20.5 Literatur

- [1] L. Conway: Introduction to VLSI Design; Course Notes, MIT, Cambridge 1979
- [2] M. Davio, J.-P. Deschamps, A. Thayse: Digital Systems with algorithm implementation; John Wiley & Sons, 1983
- [3] D. Fairbairn; VLSI - A New Frontier for System Designers; Computer, January 1982
- [4] L. Glasser, D. Dobberpuhl: The Design and Analysis of VLSI Circuits; Addison-Wesley, 1985
- [5] A. Halaas: VLSI-Implemented Algorithms for Fundamental Searching and Sorting Problems; Report, Fachbereich Informatik, Universität Kaiserslautern 1981
- [6] R. Hartenstein: Microprogramming concepts as a step toward Structured Hardware Design; Proc. 7th Ann. Workshop on Microprogramming, ACM, New York 1974
- [7] R. Hartenstein: Fundamentals of Structured Hardware Design; North Holland, 1977
- [8] R. Hartenstein: VLSI-Bausteine in geringen Stückzahlen für Spezialanwendungen; Elektronische Rechenanlagen 22 (1980), Heft 4
- [9] R. Hartenstein: Die "Neue Mikroelektronik" in der Informatik: Voraussetzungen und auswirkungen; GI-Jahrestagung, 1981 Kaiserslautern, Springer-Verlag, 1981
- [10] R. Hartenstein: Shared Cultures: CIF Library, Starting Frames and Scalable Design Rules; NATO-ASI "Very Large Scale Integration", Louvain-la-Neuve, 1981 Noordhoff & -Stijthoff, Rockville, Maryland, 1981
- [11] R. Hartenstein, P. Liell: KARL-2 language reference manual; Un. Kaiserslautern 1983

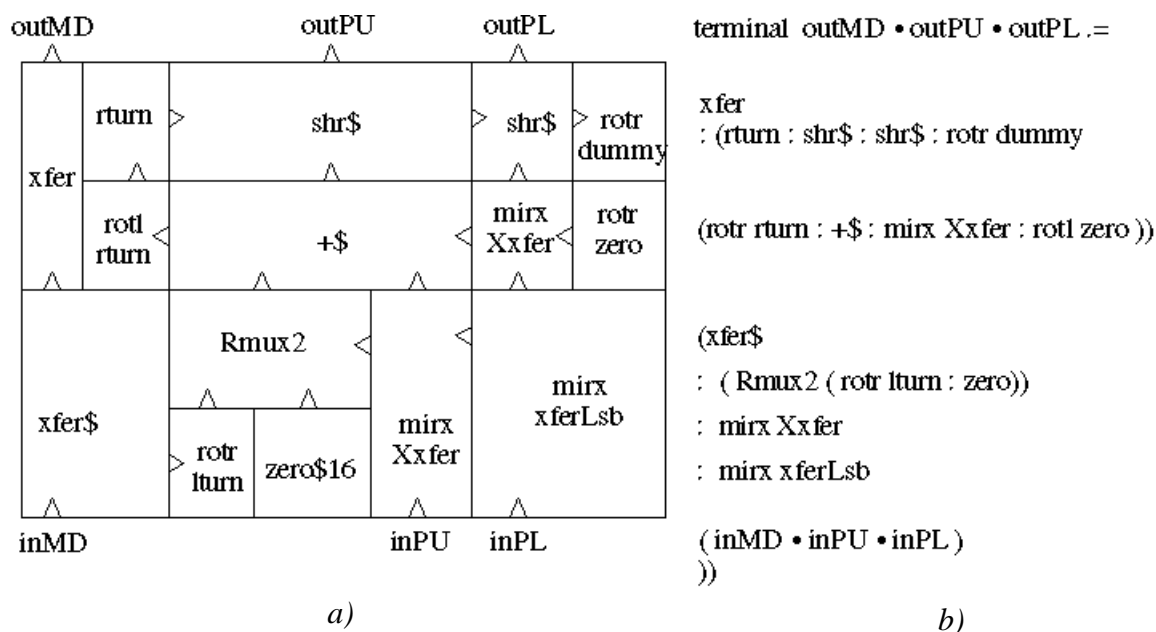


Bild 19.34: Slice-Ausdruck (rechts) zum Multiplizierer in Bild 19.32; mit vom Anwender definierten Verdrahtungszellen (auch topologisch transformiert), die Pfadbreite ist vererbt.

- [12] R. Hartenstein, K. Lemmert: KARL-3 reference manual; Univ. Kaiserslautern, 1984
- [13] R. Hartenstein, K. Bastian, W. Nebel: VLSI-Algorithmen: innovative Schaltungstechnik statt Software; Fachbereich Informatik; Uni Kaiserslautern, April 1985; GME-Tagung Baden-Baden, 1985, VDE-Verlag, Berlin 1985
- [14] R. Hartenstein, U. Welters: Higher Level Simulation and CHDLs; in (eds.: W. Fichtner, M. Morf): Proc. IFIP Summer School on VLSI Design, Beatenberg, Switzerland, 1986; Kluwer Publishing Co., 1986
- [15] R. Hartenstein, R. Hauck: Functional Extraction from Personality Matrixes of MOL (Matrix-Oriented Logic) Circuits; IFIP CHDL'87, North Holland, Amsterdam, 1987
- [16] R. Hartenstein: KARL and ABL; in: (ed. J. P. Mermet) Fundamentals and Standards in Hardware Description Languages; Kluwer Academic Publishers, 1993
- [17] R. Hauck: On Generic Hardware Descriptions; dissertation, Kaiserslautern, 1992
- [18] R. W. Hon, C. H. Sequin: A Guide to LSI Implementation, Xerox Palo Alto Research Center, 1980
- [19] G. Hotz: Eine Algebraisierung des Syntheseproblems für Schaltkreise; EIK, 1965
- [20] R. Kolla, P. Molitor, H. Osthof: Einführung in den VLSI-Entwurf; Teubner, 1989
- [21] C. Mead, L. Conway: Introduction to VLSI Systems; Addison-Wesley, 1980
- [22] V. Moshnyaga, H. Onodera, K. Tamaru, H. Yasuura: A Language for Designing Data-Path Module Generators; IEEE Int'l Design Workshop "Russian Workshop'92", Moscow, Russia; also: IFIP WG 10.5 Workshop on Synthesis, Generation and Portability of Library Blocks for ASIC Design, Grenoble, France, Mar 1992; and: Proc. SASIMI'92 - Synthesis and Simulation Meeting and Int'l Exchange, Kobe, Japan, 1992
- [23] J. Newkirk, R. Mathews: The VLSI Designer's Library; Addison-Wesley, 1983
- [24] D. Patterson, J. Hennessy: Computer Architecture - a Quantitative Approach; Morgan



- Kaufmann Publishers Inc., San Mateo, California, 1990
- [25] M. Sheran: μ FP, an algebraic VLSI design language; Ph. D. dissertation, Oxford University, Technical Monograph INF 116 / 964-39, Oxford, UK, 1983¹
 - [26] M. Sheran, G. Jones: Collecting Butterflies; Technical Monograph, Oxford University, Technical Monograph, Oxford, UK, 1991²
 - [27] H. Siegel: Interconnection Networks for Large Scale Parallel Processing; MacGraw-Hill, 1990
 - [28] Ph. C. Treleaven: VLSI Processor Architectures; Computer, June 1982

1). INF 116 / 964-39
2). INF 116 / 964-91