Rechnerarithmetik

Vorlesung im Sommersemester 2008

Eberhard Zehendner

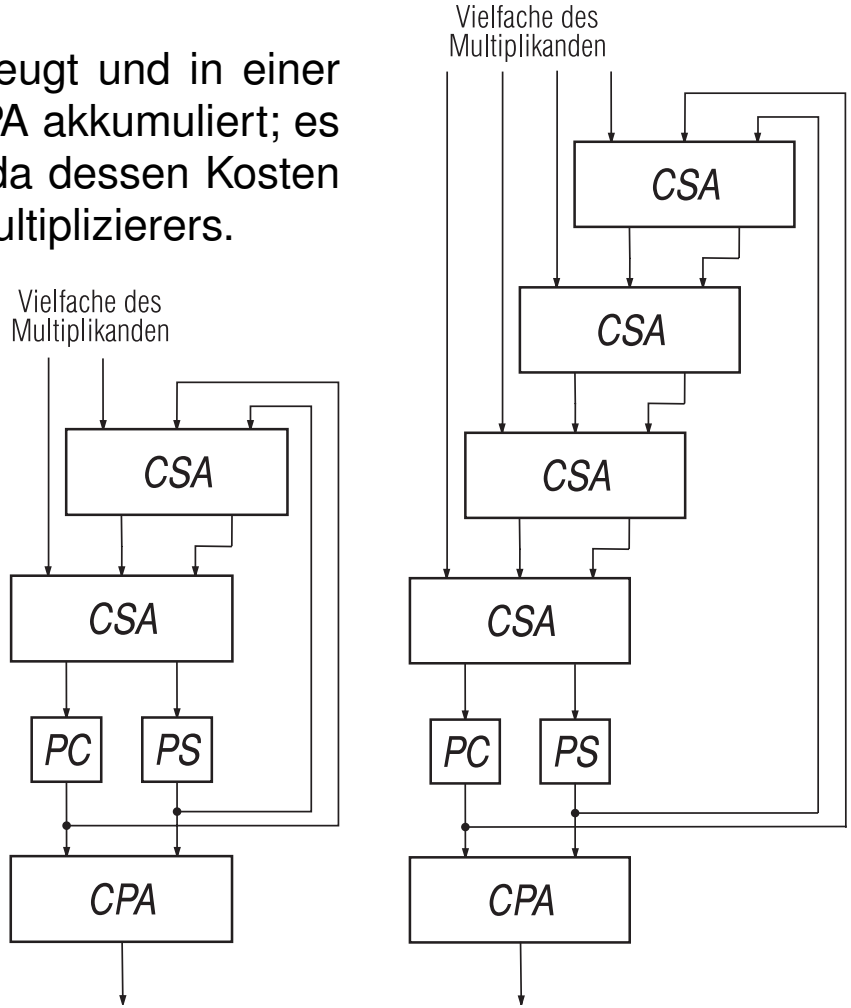
FSU Jena

Thema: Parallele Multiplizierer

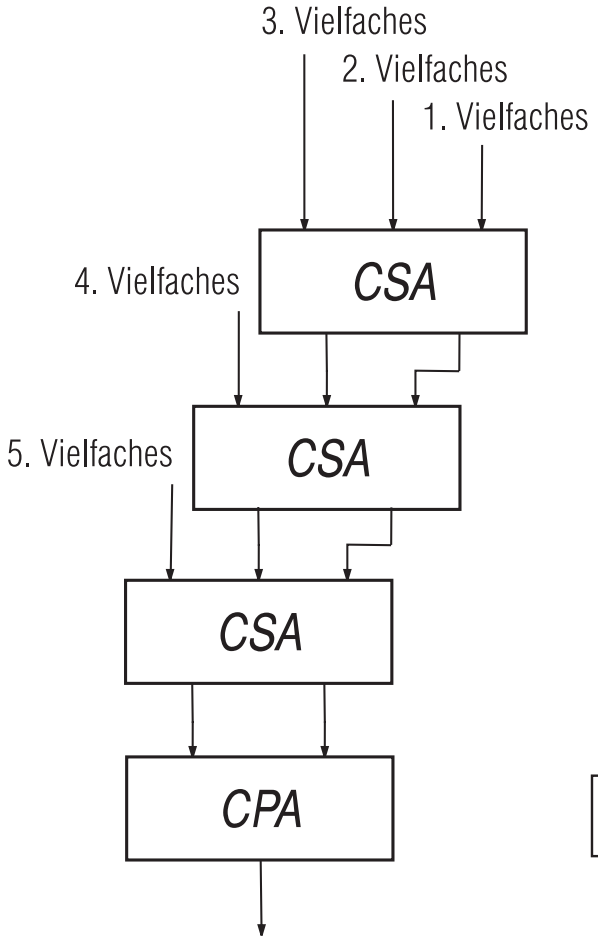
Parallele Multiplizierer mit CSA

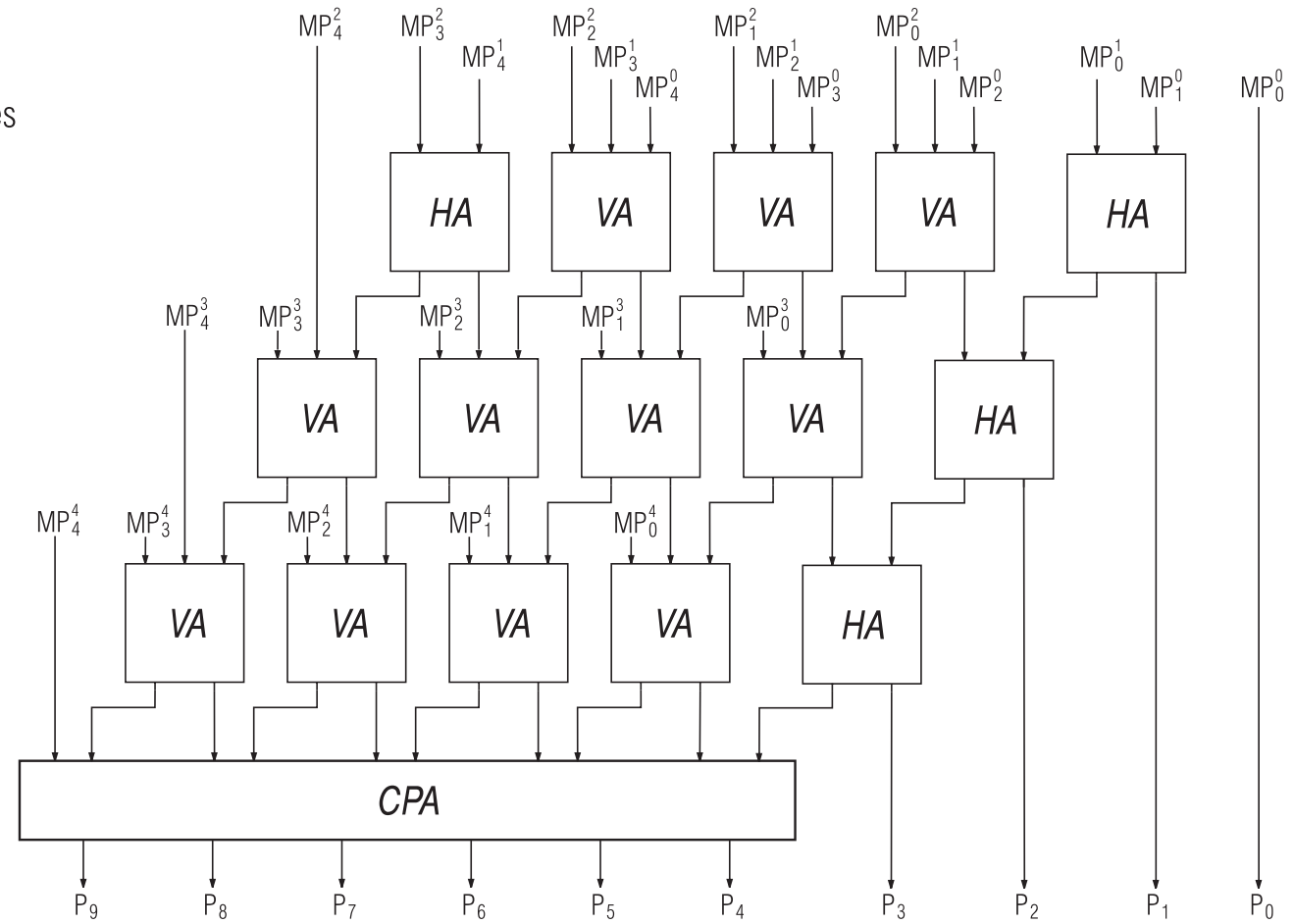
In parallelen Multiplizierern werden Teilprodukte simultan erzeugt und akkumuliert; dies geschieht in der Regel mit Hilfe mehrerer CSA.

Im Extremfall werden alle Teilprodukte simultan erzeugt und in einer nicht-iterativen CSA-Struktur mit abschließendem CPA akkumuliert; es kann dann ein sehr schneller CPA benutzt werden, da dessen Kosten gering sind im Vergleich zum Gesamtaufwand des Multiplizierers.



Akkumulation im Array-Multiplizierer





Vergleich von Array-Multiplizierern

$l = m$	CPA	t	a	$a \times t \times 10^{-4}$
16	RCLA	49	4012	19,66
	MSBCLA	53	3812	20,20
	SRCLA	53	3844	20,37
	2-Level-Carry-Skip	55	3752	20,63
	BCLA	57	3752	21,39
	SBCLA	57	3820	21,77
	1-Level-Carry-Skip	58	3692	21,41
	Carry-Ripple	74	3676	27,20
	Seriell	91	3474	31,61
32	SRCLA	101	15972	161,31
	MSBCLA	101	16012	161,72
	RCLA	109	15768	171,87
	SBCLA	109	15816	172,39
	2-Level-Carry-Skip	111	15640	173,60
	BCLA	113	15668	177,05
	1-Level-Carry-Skip	114	15548	177,24
	Carry-Ripple	154	15516	238,95
	Seriell	187	15090	282,18
64	SRCLA	197	64908	1278,68
	MSBCLA	201	64444	1295,32
	RCLA	205	64276	1317,66
	SBCLA	209	64424	1346,46
	2-Level-Carry-Skip	215	64016	1376,34
	1-Level-Carry-Skip	216	63836	1378,86
	BCLA	217	64196	1393,05
	Carry-Ripple	314	63772	2002,24
	Seriell	379	62898	2383,83

Array-Multiplizierer

In Array-Multiplizierern wird die Erzeugung der Teilprodukte und ihre Akkumulierung simultan vorgenommen; dadurch entfällt der zweifache Aufwand für die Steuerung separater Einheiten.

Jede Zeile des Array-Multiplizierers berechnet ein neues Teilprodukt (oder einen Ausschnitt daraus) und addiert es zur bisherigen Zwischensumme.

Array-Multiplizierer werden in der Regel in Pipelining-Technik ausgeführt:

Der Abschluss des Array-Multiplizierers durch einen CPA begrenzt den Durchsatz.

Der CPA wird deshalb durch weitere Zeilen ohne Propagierung innerhalb einer Zeile ersetzt.

Für die Realisierung der Zellen der zusätzlichen Zeilen genügen Halbaddierer (statt der Volladdierer in den oberen Zeilen).

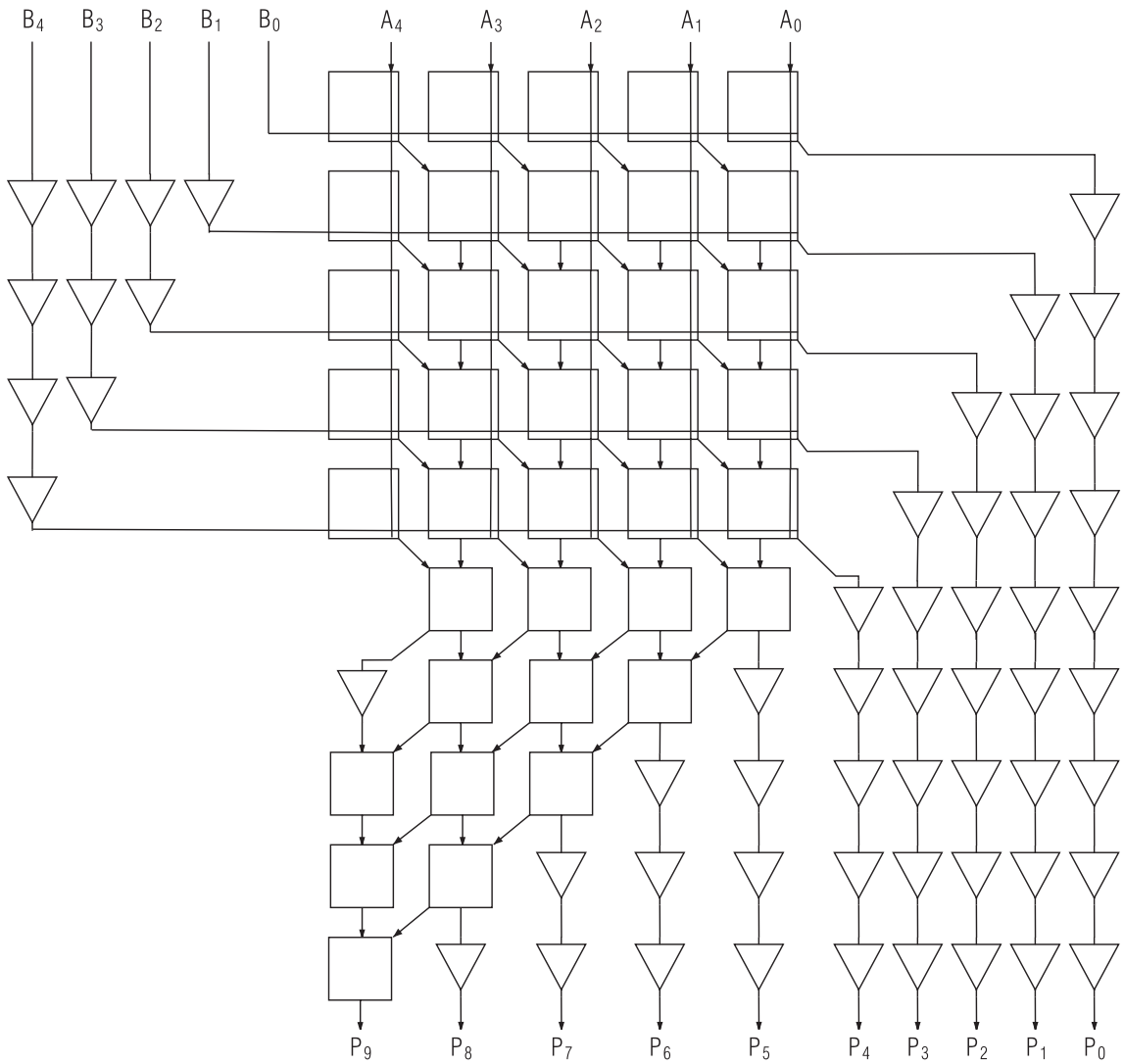
Zusätzlich erfordert das Pipelining ein Array von Puffern für die getaktete Weitergabe der Multiplikatorbits und der Produktbits.

Array-Multiplizierer in Pipelining-Technik

Die oberen und linken Randzellen des rechteckigen Multiplikationsarrays enthalten nur jeweils ein AND-Gatter zur Multiplikation zweier Bits.
Jede andere Zelle der zweiten Zeile beinhaltet zusätzlich einen Halbaddierer.

Jede weitere Zelle des rechteckigen Multiplikationsarrays enthält das AND-Gatter sowie einen Volladdierer.

Jede rechteckige Zelle der linken Spalte des dreieckigen Abschlussarrays enthält ein OR-Gatter zur Addition zweier Bits ohne Übertrag.
Jede weitere rechteckige Zelle des Abschlussarrays enthält einen Halbaddierer.



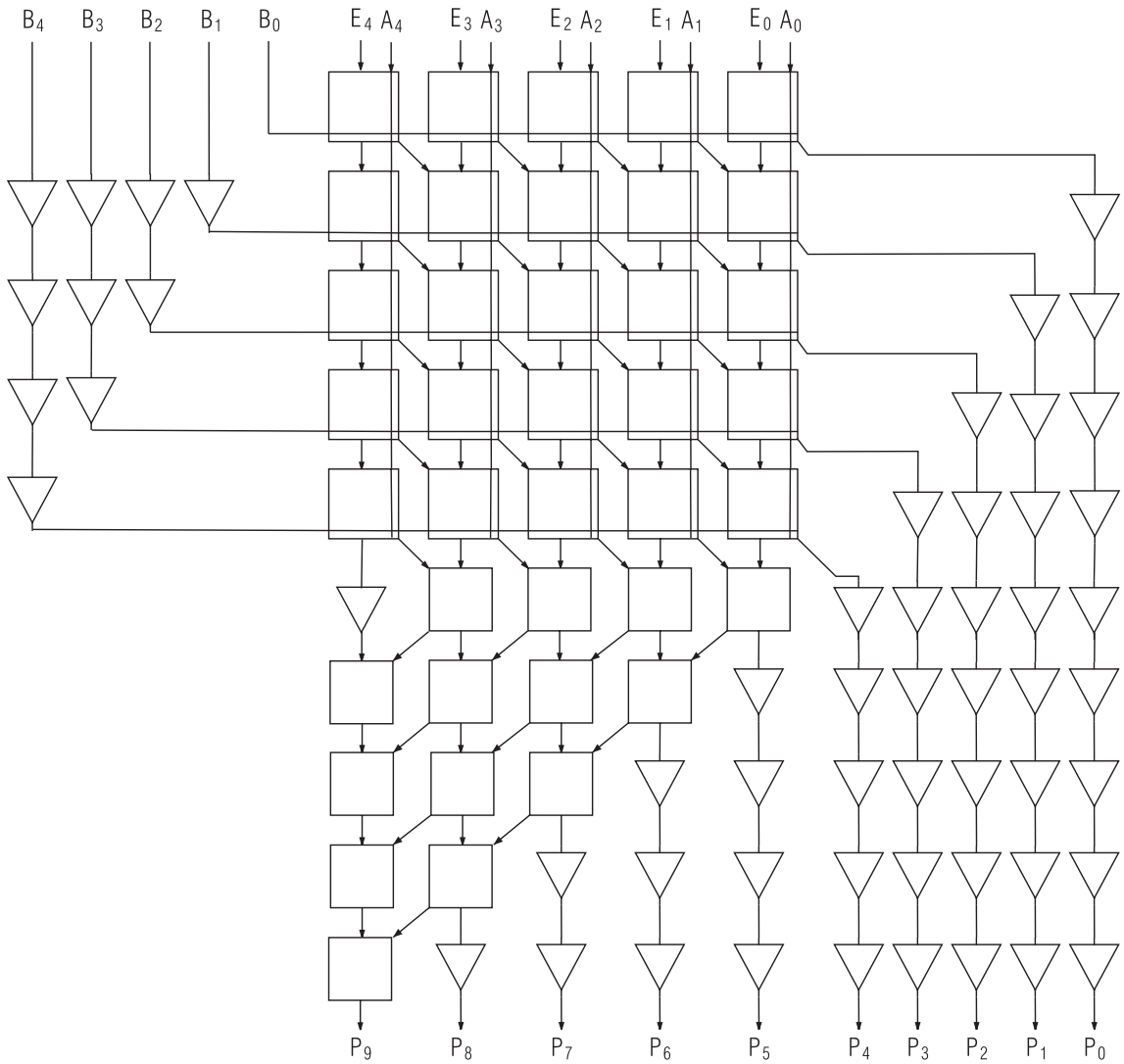
Erweiterung zur Multiply-Add-Einheit

Die oberen und linken Randzellen des rechteckigen Multiplikationsarrays enthalten jeweils ein AND-Gatter zur Multiplikation zweier Bits sowie einen Halbaddierer.

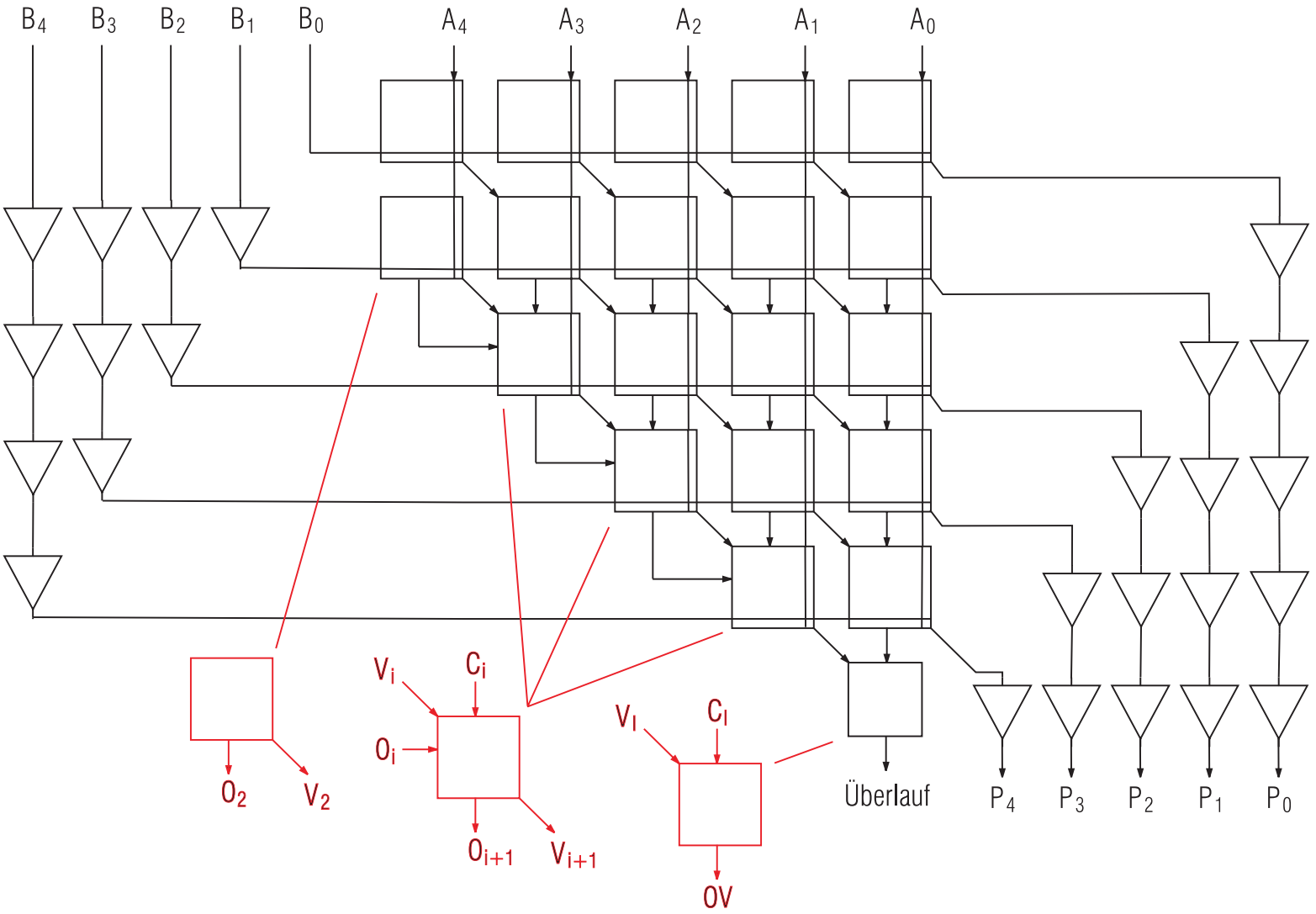
Jede weitere Zelle des rechteckigen Multiplikationsarrays enthält das AND-Gatter sowie einen Volladdierer.

Jede rechteckige Zelle der linken Spalte des dreieckigen Abschlussarrays enthält ein OR-Gatter zur Addition zweier Bits ohne Übertrag.

Jede weitere rechteckige Zelle des Abschlussarrays enthält einen Halbaddierer.



Array-Multiplizierer mit Überlauferkennung

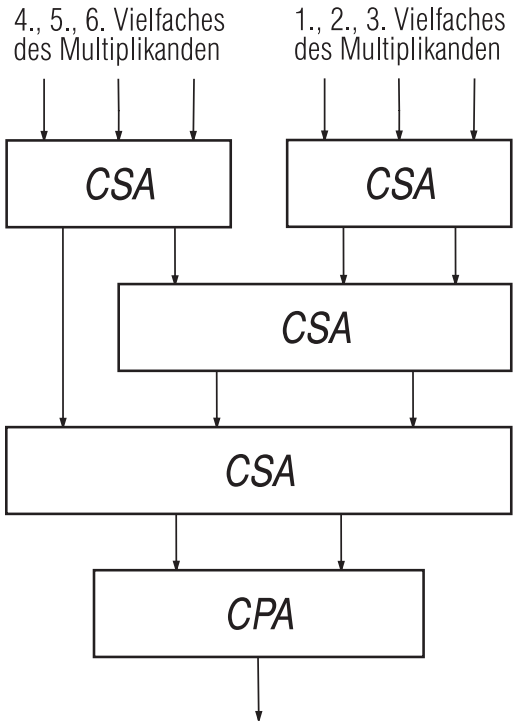


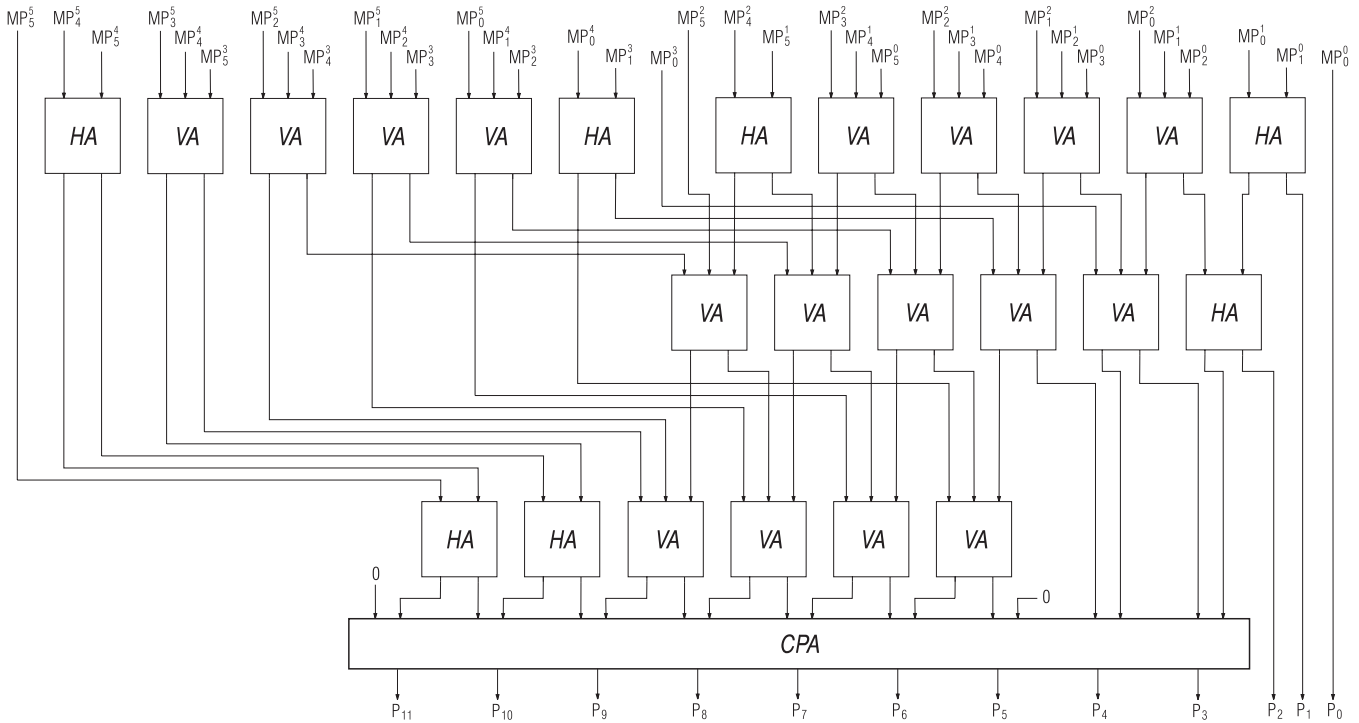
$$O_2 = A_{l-1} \qquad V_2 = A_{l-1} \wedge B_1 \qquad OV = V_l \vee C_l$$
$$O_{i+1} = O_i \vee A_{l-i} \qquad V_{i+1} = V_i \vee C_i \vee O_{i+1} \wedge B_i$$

Baum-Multiplizierer

Statt einer Kaskade von CSA-Addierern kann auch ein CSA-Baum benutzt werden.

Die Latenz eines Baum-Multiplizierers ist kleiner als die eines Array-Multiplizierers;
dies wird durch einen höheren Hardware-Aufwand erkauft
(breiterer CPA, irregulärer Aufbau, komplexere Leitungsführung).



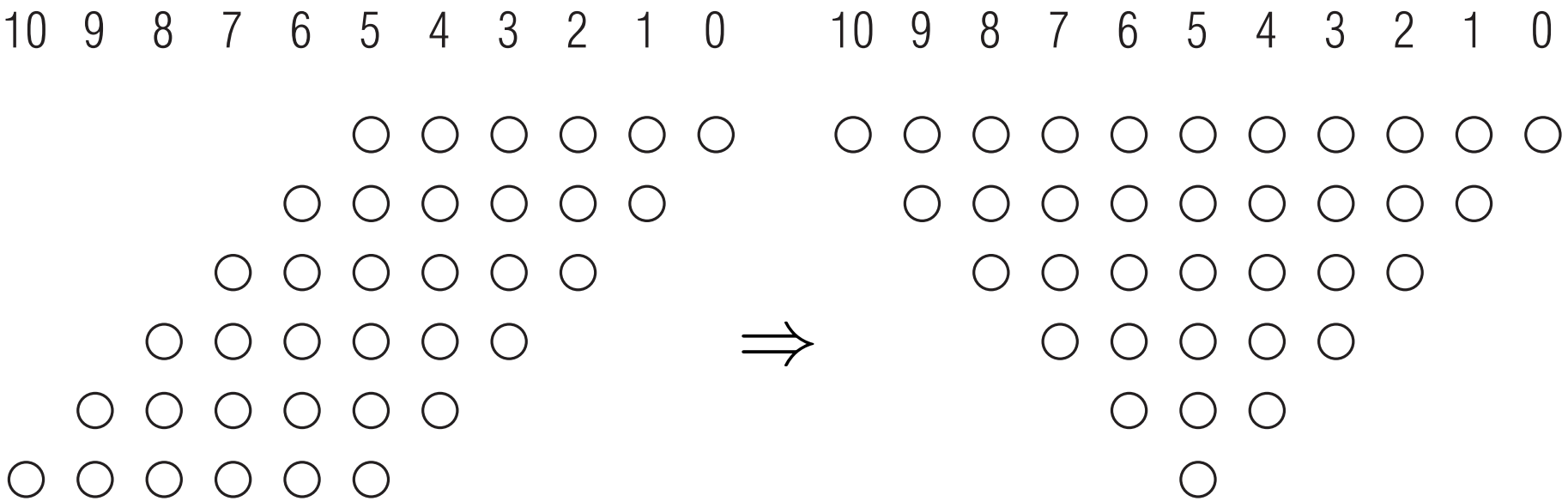


Effiziente Summierung der Teilprodukte

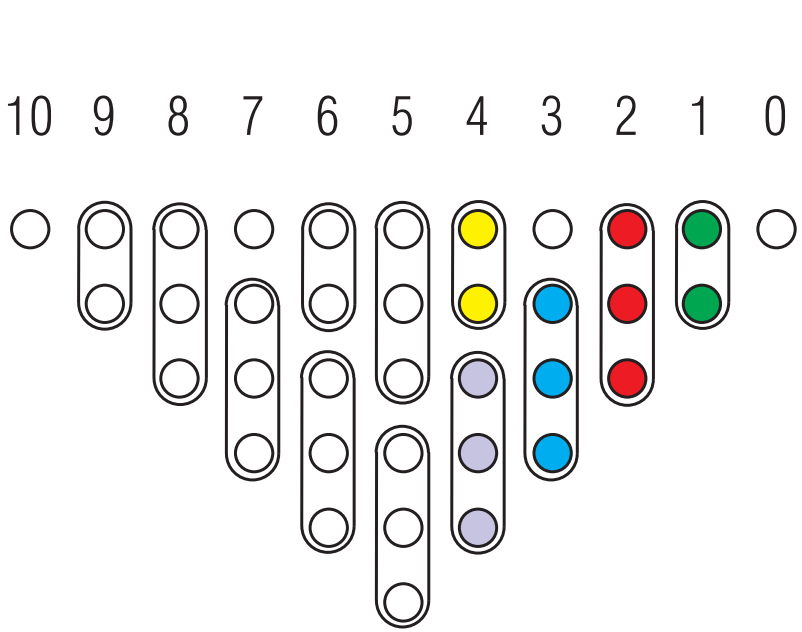
Durch geschickte Beschaltung kann die Anzahl der Volladdierer eines CSA-Baumes zur Summierung der Teilprodukte verringert werden (nicht aber die Anzahl der Stufen).

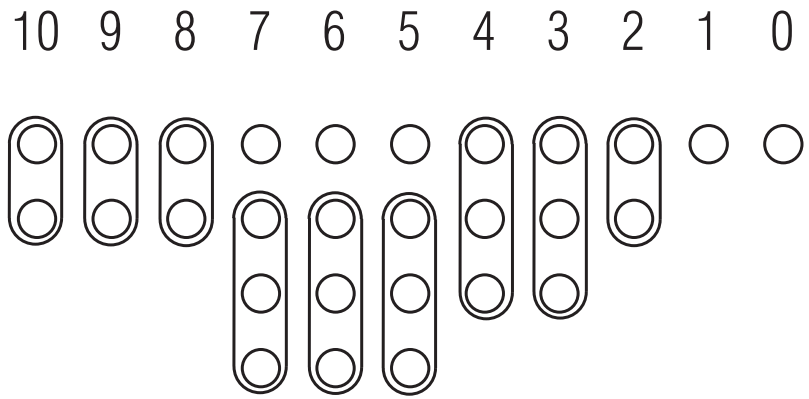
Außerdem können einige Volladdierer durch Halbaddierer ersetzt werden.

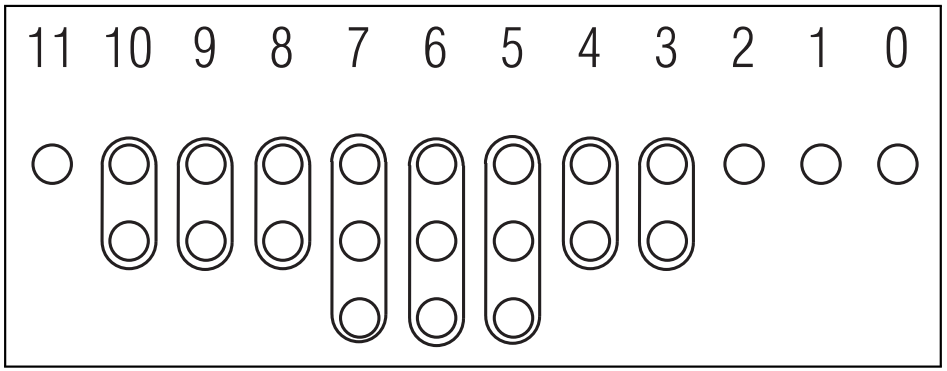
Initiales Bitschema für einen (6×6)-Multiplizierer:

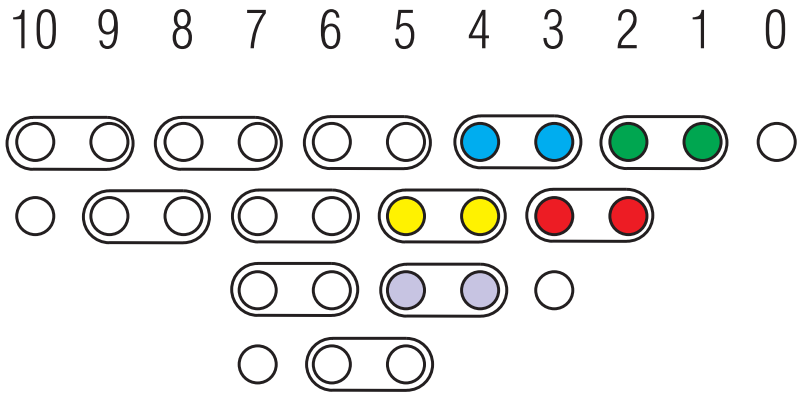


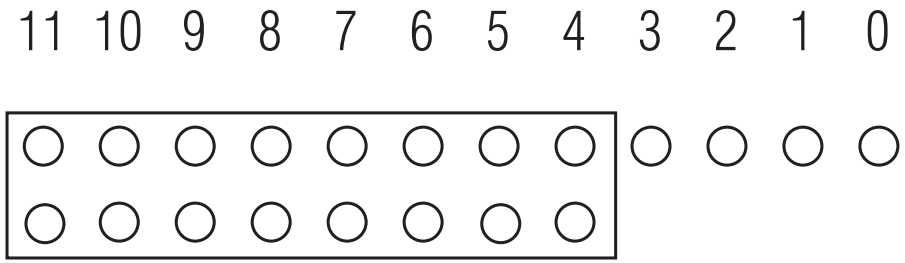
Frühestmögliche Reduktion: Wallace-Multiplizierer











Vergleich von Wallace-Multiplizierern

$l = m$	CPA	t	a	$a \times t \times 10^{-4}$
16	RCLA	25	4642	11,61
	MSBCLA	29	4442	12,88
	SRCLA	29	4472	12,97
	2-Level-Carry-Skip	31	4382	13,58
	BCLA	33	4382	14,46
	SBCLA	33	4450	14,67
	1-Level-Carry-Skip	34	4322	14,69
	Carry-Ripple	50	4306	21,53
	Seriell	67	4104	27,50
32	SRCLA	32	17274	55,28
	MSBCLA	32	17314	55,40
	RCLA	38	17070	64,87
	SBCLA	40	17118	68,47
	2-Level-Carry-Skip	42	16942	71,56
	BCLA	44	16970	74,67
	1-Level-Carry-Skip	45	16850	75,83
	Carry-Ripple	85	16818	142,95
	Seriell	118	16392	193,43
64	SRCLA	35	67554	236,44
	MSBCLA	39	67090	261,65
	RCLA	43	66922	287,76
	SBCLA	47	67070	315,23
	2-Level-Carry-Skip	53	66662	353,30
	1-Level-Carry-Skip	54	66482	359,00
	BCLA	55	66842	367,63
	Carry-Ripple	152	66418	1009,55
	Seriell	217	65544	1422,30

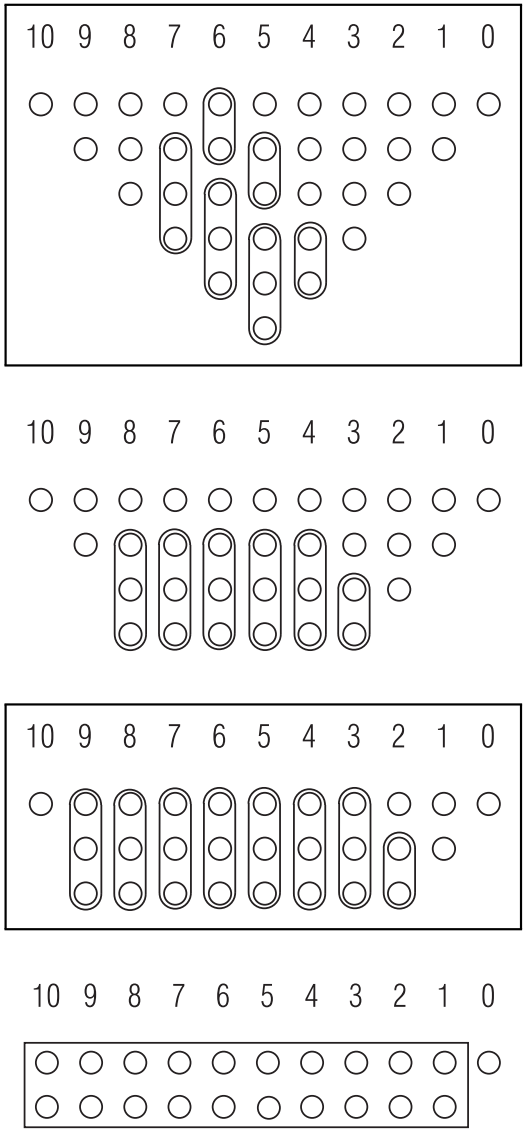
Einsparungen durch verzögerte Reduktion: Dadda-Multiplizierer

Im Dadda-Multiplizierer werden Volladdierer und Halbaddierer so eingesetzt, dass die Anzahl der Ergebnisbits derselben Gewichtung möglichst nahe an den bezüglich der Stufenzahl eines CSA-Baumes optimalen Wert herankommen.

Die Anzahl der Ergebnisbits sollte damit am besten einen der Werte $3, 4, 6, 9, 13, 19, \ldots$ annehmen.

Aufwand (ohne CPA) für parallele (6×6)-Multiplizierer:

Standard-CSA-Baum:	4×12 VA
Zwei niederwertigste Stellen direkt in CPA geleitet:	4×10 VA
Wallace-Multiplizierer (schnellster Multiplizierer):	16 VA, 13 HA
Dadda-Multiplizierer:	15 VA, 5 HA
Optimierter Array-Multiplizierer:	19 VA, 5 HA



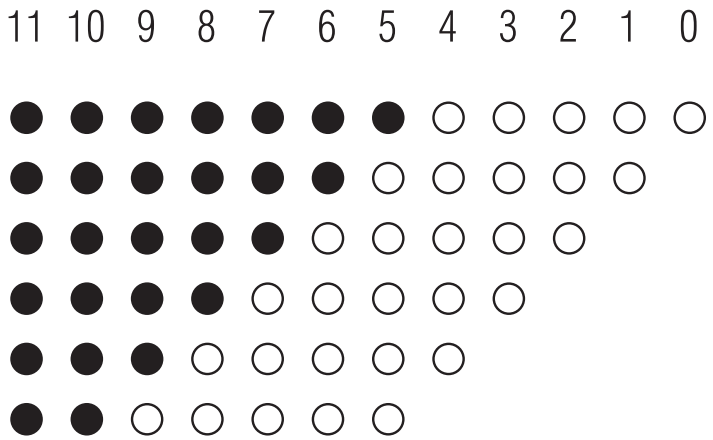
Summierung von Teilprodukten mit Vorzeichen

- In Vorzeichen/Betrag-Darstellung läuft eine $(l \times m)$-Multiplikation praktisch wie eine vorzeichenlose $((l-1) \times (m-1))$-Multiplikation ab.

  Das Vorzeichen wird abgetrennt und separat verarbeitet: $P_{n-1} = A_{l-1} \oplus B_{m-1}$
- In 1- oder 2-Komplement-Darstellung müssen negative Teilprodukte vor ihrer Summierung durch Vervielfachung des Vorzeichens auf die Länge des Ergebnisses erweitert werden.

  Da es wenig Sinn macht, zur Laufzeit zwischen positiven und negativen Teilprodukten zu unterscheiden, erfolgt die Vorzeichenvervielfachung auch für positive Teilprodukte.

  Die Anzahl der zu addierenden Bits steigt hierdurch signifikant an.



Summierung von Teilprodukten in 2-Komplement-Darstellung (Beispiel)

A						0	1	0	1	1	0	Multiplikand 22
B				$\times$		0	0	1	0	1	1	Multiplikator 11
B'						0	1	0	$\bar{1}$	0	$\bar{1}$	umcodierter Multiplikator
1	**1**	**1**	**1**	**1**	**1**	**1**	0	1	0	1	0	
0	**0**	**0**	**0**	**0**	**0**	0	0	0	0	0		
1	**1**	**1**	**1**	**1**	0	1	0	1	0			
0	**0**	**0**	**0**	0	0	0	0	0				
0	**0**	**0**	1	0	1	1	0					
0	**0**	0	0	0	0	0						
0	0	0	0	1	1	1	1	0	0	1	0	Ergebnis 242

Umcodieren der Teilprodukte

Durch Umcodieren der Teilprodukte kann die Anzahl der zusätzlichen Bits stark verringert werden. Beispielsweise gilt mit beliebigem $S \in \{0, 1\}$:

$$SSSSSSSZ_4Z_3Z_2Z_1Z_0 \equiv 000000\bar{S}Z_4Z_3Z_2Z_1Z_0 \pmod{2^{12}}$$

Durch einen Trick lässt sich die Ziffer $\bar{S}$ ohne Verwendung eines ternären Codes darstellen:

- Komplementieren des Vorzeichenbits S liefert die Ziffer $(1 - S)$, die durch Addieren einer zusätzlichen 1 an der Position des Vorzeichenbits in den Wert $(2 - S)$ überführt wird.
- Der Wert $+2$ kann eine Stelle höher als dieselbe zu addierende 1 für das folgende Teilprodukt interpretiert werden.
- Die gewünschten Ziffern $\bar{S}$ ergeben sich also durch Komplementierung des Vorzeichenbits aller Teilprodukte, Addition einer einzigen 1 an der Position des Vorzeichens des am niedrigsten gewichteten Teilprodukts und Komplementieren des Ergebnisvorzeichenbits.

Umcodieren der Teilprodukte (Beispiel)

A						0	1	0	1	1	0	Multiplikand 22
B				$\times$		0	0	1	0	1	1	Multiplikator 11
B'						0	1	0	$\bar{1}$	0	$\bar{1}$	umcodierter Multiplikator
						1						
						0	0	1	0	1	0	
					1	0	0	0	0	0		
				0	0	1	0	1	0			
			1	0	0	0	0	0				
		1	1	0	1	1	0					
	1	0	0	0	0	0						
1	0	0	0	1	1	1	1	0	0	1	0	vorläufiges Ergebnis
0	0	0	0	1	1	1	1	0	0	1	0	Ergebnis 242

Direkte Multiplikation in $\mathrm{Int}_2(l)$ nach Baugh-Wooley

Das Multiplikationsschema für $\mathrm{UInt}_2(l)$ wird zu einem Multiplikationsschema für $\mathrm{Int}_2(l)$ abgeändert, in dem die negativen Gewichte der beiden Vorzeichenbits berücksichtigt sind.

Darauf werden folgende Regeln zur Vereinfachung systematisch angewandt:

- $-A_{l-1}B_j = A_{l-1}(1 - B_j) - A_{l-1} = A_{l-1}\overline{B}_j - A_{l-1}$.
- $-A_{l-1}$ in Spalte k wird ersetzt durch A_{l-1} in Spalte k und $-A_{l-1}$ in Spalte $k+1$.
- $-A_{l-1}$ in Spalte $2l-2$ wird ersetzt durch $\overline{A}_{l-1} - 1$.
- $-A_iB_{l-1} = (1 - A_i)B_{l-1} - B_{l-1} = \overline{A}_iB_{l-1} - B_{l-1}$.
- $-B_{l-1}$ in Spalte k wird ersetzt durch B_{l-1} in Spalte k und $-B_{l-1}$ in Spalte $k+1$.
- $-B_{l-1}$ in Spalte $2l-2$ wird ersetzt durch $\overline{B}_{l-1} - 1$.
- -2 in Spalte $2l-2$ wird ersetzt durch 1 in Spalte $2l-1$.

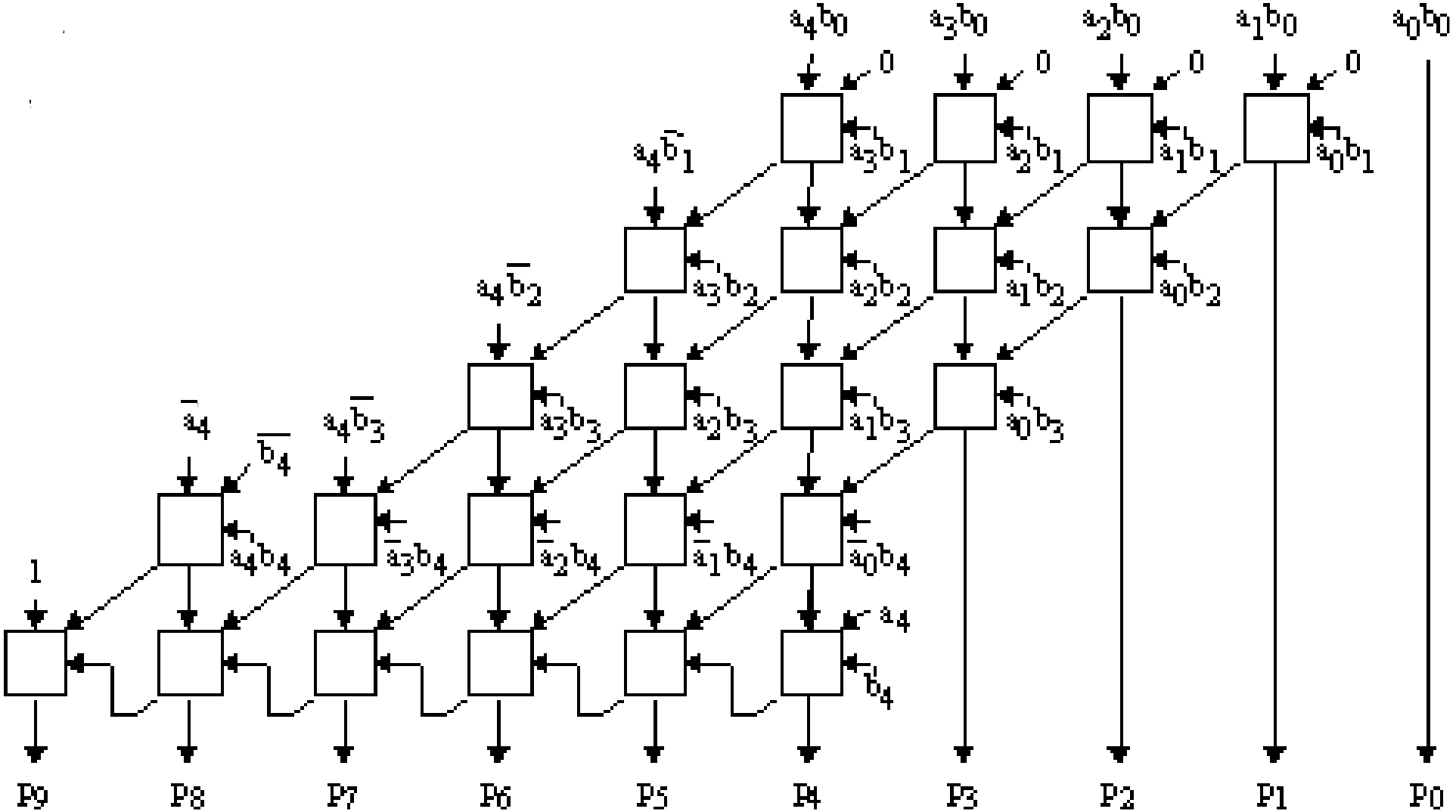
Transformationen im Baugh-Wooley-Multiplizierer

					$-A_4$	A_3	A_2	A_1	A_0
				$\times$	$-B_4$	B_3	B_2	B_1	B_0
					$-A_4B_0$	A_3B_0	A_2B_0	A_1B_0	A_0B_0
				$-A_4B_1$	A_3B_1	A_2B_1	A_1B_1	A_0B_1	
			$-A_4B_2$	A_3B_2	A_2B_2	A_1B_2	A_0B_2		
		$-A_4B_3$	A_3B_3	A_2B_3	A_1B_3	A_0B_3			
	A_4B_4	$-A_3B_4$	$-A_2B_4$	$-A_1B_4$	$-A_0B_4$				
P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

wird transformiert in

					$-A_4$	A_3	A_2	A_1	A_0
				$\times$	$-B_4$	B_3	B_2	B_1	B_0
					$A_4\overline{B}_0$	A_3B_0	A_2B_0	A_1B_0	A_0B_0
				$A_4\overline{B}_1$	A_3B_1	A_2B_1	A_1B_1	A_0B_1	
			$A_4\overline{B}_2$	A_3B_2	A_2B_2	A_1B_2	A_0B_2		
		$A_4\overline{B}_3$	A_3B_3	A_2B_3	A_1B_3	A_0B_3			
	A_4B_4	$\overline{A}_3B_4$	$\overline{A}_2B_4$	$\overline{A}_1B_4$	$\overline{A}_0B_4$				
	$\overline{A}_4$				A_4				
1	$\overline{B}_4$				B_4				
P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

Baugh-Wooley-Multiplizierer



Optimierung der Baugh-Wooley-Multiplikation

In baumartigen Multiplizierern erhöht der Baugh-Wooley-Algorithmus
die Anzahl der Einträge in der kritischen Spalte $l - 1$ und damit evtl. die Latenz.

Mit folgenden alternativen Regeln zur Vereinfachung wird dies vermieden:

- $-S = (1 - S) - 1 = \overline{S} - 1$.
- Alle auftretenden Werte von -1 können zusammengefasst und als Einsen in den Spalten l und $2l - 1$ wiedergegeben werden (in denen sie unschädlich sind).

Transformationen im optimierten Baugh-Wooley-Multiplizierer

					$-A_4$	A_3	A_2	A_1	A_0
				$\times$	$-B_4$	B_3	B_2	B_1	B_0
					$-A_4B_0$	A_3B_0	A_2B_0	A_1B_0	A_0B_0
				$-A_4B_1$	A_3B_1	A_2B_1	A_1B_1	A_0B_1	
			$-A_4B_2$	A_3B_2	A_2B_2	A_1B_2	A_0B_2		
		$-A_4B_3$	A_3B_3	A_2B_3	A_1B_3	A_0B_3			
	A_4B_4	$-A_3B_4$	$-A_2B_4$	$-A_1B_4$	$-A_0B_4$				
P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

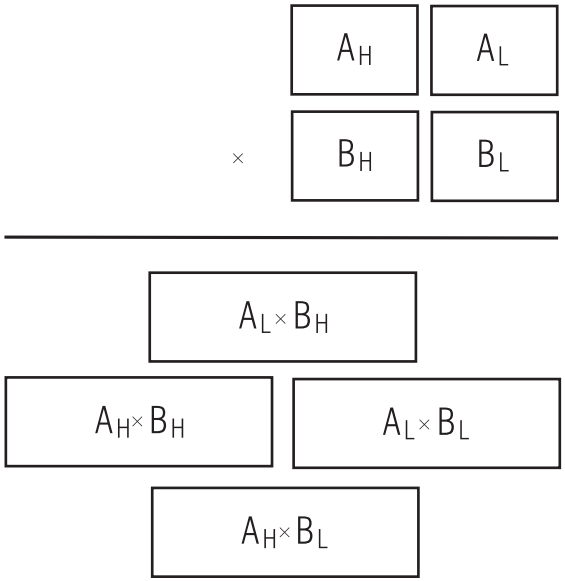
wird transformiert in

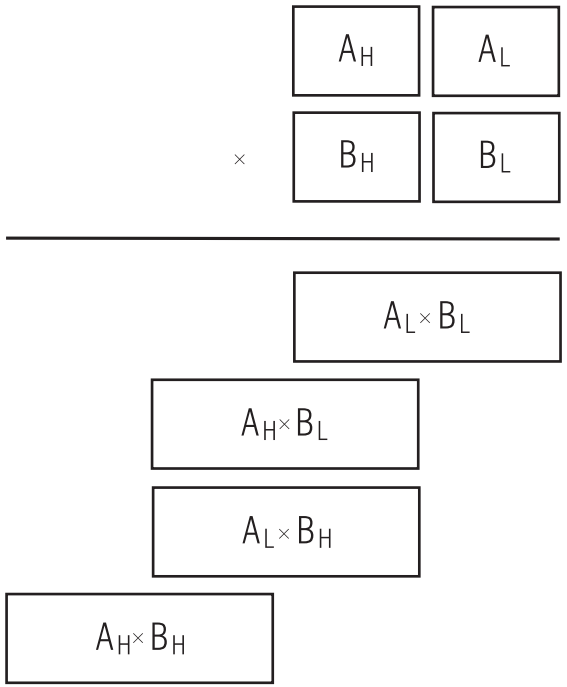
					$-A_4$	A_3	A_2	A_1	A_0
				$\times$	$-B_4$	B_3	B_2	B_1	B_0
1				1	$\overline{A_4B_0}$	A_3B_0	A_2B_0	A_1B_0	A_0B_0
				$\overline{A_4B_1}$	A_3B_1	A_2B_1	A_1B_1	A_0B_1	
			$\overline{A_4B_2}$	A_3B_2	A_2B_2	A_1B_2	A_0B_2		
		$\overline{A_4B_3}$	A_3B_3	A_2B_3	A_1B_3	A_0B_3			
	A_4B_4	$\overline{A_3B_4}$	$\overline{A_2B_4}$	$\overline{A_1B_4}$	$\overline{A_0B_4}$				
P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

Rekursiver Aufbau großer Multiplizierer

Ein $(2n \times 2n)$-Multiplizierer kann aus vier $(n \times n)$-Multiplizierern aufgebaut werden:

$$\begin{aligned} A \times B &= (A_H \times 2^n + A_L) \times (B_H \times 2^n + B_L) \\ &= A_H \times B_H \times 2^{2n} \\ &\quad + (A_H \times B_L + A_L \times B_H) \times 2^n \\ &\quad + A_L \times B_L \end{aligned}$$



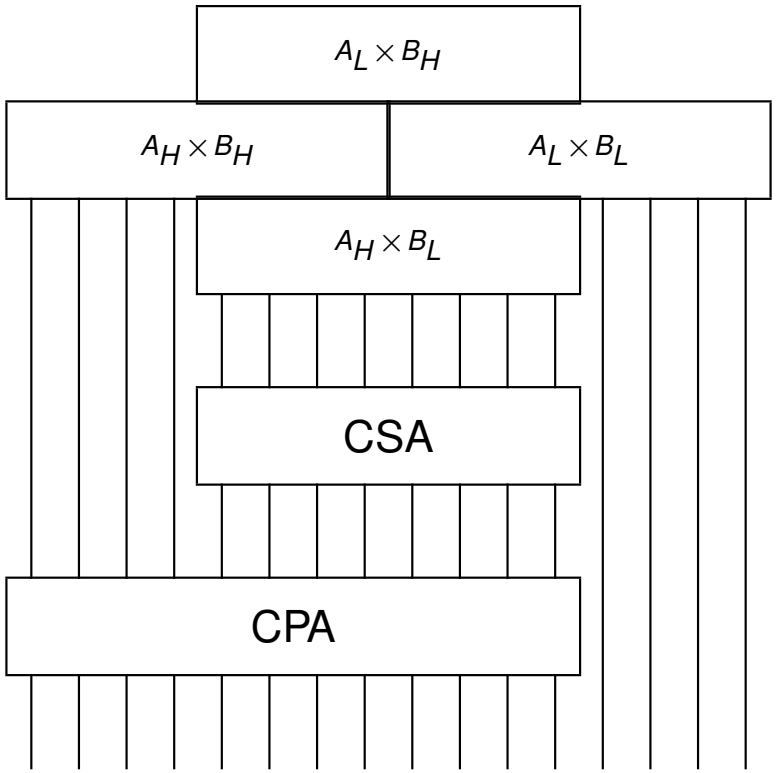


Optimierung durch Umordnung der Teilprodukte:

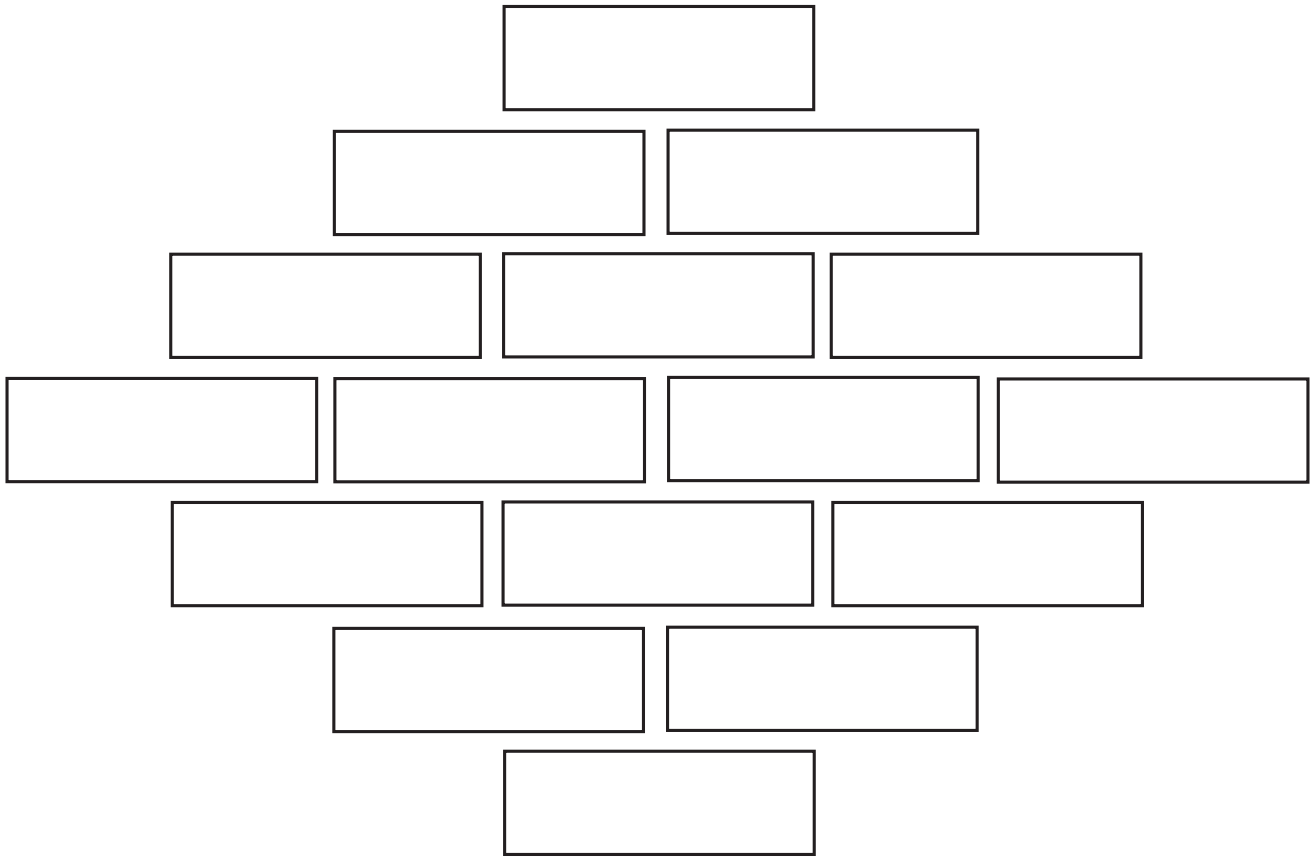
- Minimale Anzahl von Addierern.
- Möglichst kurze Dauer der Summation.

Detailaufbau des $(2n \times 2n)$-Multiplizierers

Benötigt werden vier $(n \times n)$-Multiplizierer, ein $2n$-Bit CSA und ein $3n$-Bit CPA:



Additionsschema für einen $(4n \times 4n)$-Multiplizierer



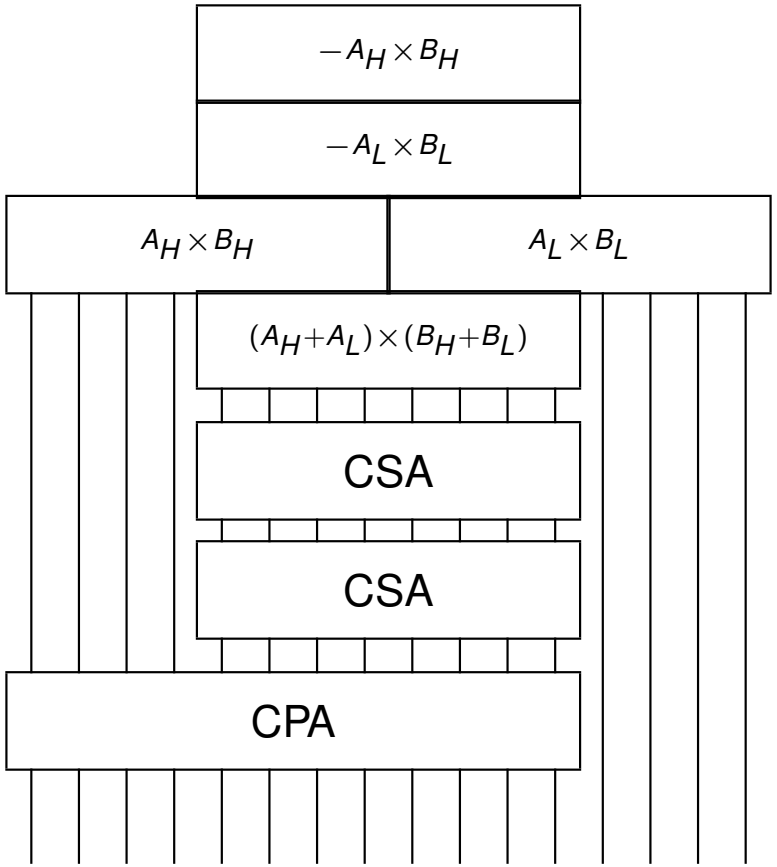
Einsparung von Teilmultiplizierern

Unter Benutzung einer Idee von Karatsuba (1962; beschrieben z. B. in: Aho, Hopcroft, Ullman, The design and analysis of computer algorithms, 1974) kann ein $(2n \times 2n)$-Multiplizierer aus nur drei $(n \times n)$-Multiplizierern aufgebaut werden:

$$\begin{aligned} A \times B &= (A_H \times 2^n + A_L) \times (B_H \times 2^n + B_L) \\ &= A_H \times B_H \times 2^{2n} + (A_H \times B_L + A_L \times B_H) \times 2^n + A_L \times B_L \\ &= A_H \times B_H \times 2^{2n} \\ &\quad + ((A_H + A_L) \times (B_H + B_L) - A_H \times B_H - A_L \times B_L) \times 2^n \\ &\quad + A_L \times B_L \end{aligned}$$

Sparsamer $(2n \times 2n)$-Multiplizierer

Benötigt werden zwei $(n \times n)$-Multiplizierer, ein $((n+1) \times (n+1))$-Multiplizierer, zwei $(2n+2)$-Bit CSA, ein $3n$-Bit CPA und zwei n-Bit CPA:



Quadrierer

Das Quadrieren von A ist im Prinzip durch eine Multiplikation $A \times A$ möglich.
Ein spezieller Quadrierer ist aber schneller (und einfacher) als ein allgemeiner Multiplizierer.

Auf das allgemeine Spaltenschema der Teilprodukte einer Multiplikation können beim Quadrieren folgende Regeln zur Vereinfachung systematisch angewandt werden:

- $A_i A_i = A_i$
- Ein Paar von Termen $A_i A_j$ und $A_j A_i$ in einer Spalte kann ersetzt werden durch den Term $A_i A_j$ (oder $A_j A_i$) in der nächsthöheren Spalte.

Es ist in der Zahlentheorie wohlbekannt, dass $(A^2 \bmod 4) = (A \bmod 2)$. Systematische Reduktion des Spaltenschemas reproduziert unter anderem diesen nützlichen Zusammenhang!

Abhängig von der Länge des Operanden und den spezifischen Optimierungszielen können, obgleich weniger systematisch, weitere Reduktionsschritte erfolgen.

Beispiel: Quadrieren von $A = A_4 A_3 A_2 A_1 A_0 \in \mathrm{UInt}_2(5)$.

Quadrierer (1. Reduktionsschritt, systematisch)

					A_4	A_3	A_2	A_1	A_0
			$\times$		A_4	A_3	A_2	A_1	A_0
					A_4A_0	A_3A_0	A_2A_0	A_1A_0	A_0A_0
				A_4A_1	A_3A_1	A_2A_1	A_1A_1	A_0A_1	
			A_4A_2	A_3A_2	A_2A_2	A_1A_2	A_0A_2		
		A_4A_3	A_3A_3	A_2A_3	A_1A_3	A_0A_3			
	A_4A_4	A_3A_4	A_2A_4	A_1A_4	A_0A_4				
P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

wird reduziert zu

					A_4	A_3	A_2	A_1	A_0
			$\times$		A_4	A_3	A_2	A_1	A_0
	A_4A_3	A_4A_2	A_4A_1	A_4A_0	A_3A_0	A_2A_0	A_1A_0		A_0
	A_4		A_3A_2	A_3A_1	A_2A_1		A_1		
			A_3		A_2				
P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	0	A_0

Quadrierer (2. Reduktionsschritt, sporadisch)

					A_4	A_3	A_2	A_1	A_0
			$\times$		A_4	A_3	A_2	A_1	A_0
	A_4A_3	A_4A_2	A_4A_1	A_4A_0	A_3A_0	A_2A_0	A_1A_0		A_0
	A_4		A_3A_2	A_3A_1	A_2A_1		A_1		
			A_3		A_2				
P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	0	A_0

ergibt mit $A_1A_0 + A_1 = 2A_1A_0 + A_1(1 - A_0) = 2A_1A_0 + A_1\overline{A}_0$

					A_4	A_3	A_2	A_1	A_0
			$\times$		A_4	A_3	A_2	A_1	A_0
	A_4A_3	A_4A_2	A_4A_1	A_4A_0	A_3A_0	A_2A_0	$A_1\overline{A}_0$		A_0
	A_4		A_3A_2	A_3A_1	A_2A_1	A_1A_0			
			A_3		A_2				
P_9	P_8	P_7	P_6	P_5	P_4	P_3	$A_1\overline{A}_0$	0	A_0

Potenzierung

Potenzierung $P = A^k$ mit $k \in \mathbb{N}$ kann als Spezialfall einer Serie von Multiplikationen aufgefasst werden.

Mit $k = \sum_{i=0}^{j-1} k_i \times 2^i$ in kanonischer Binärdarstellung und den Hilfsgrößen $H^{(i)} = A^{2^i}$ für $i = 0, \ldots, j-1$ gilt $A^k = \prod\{H^{(i)} : k_i = 1\}$.

Die $H^{(i)}$ berechnen sich nach den Formeln $H^{(0)} = A, H^{(i+1)} = H^{(i)} \times H^{(i)}$.

Der Iterationsprozess verläuft über $P^{(0)} = 1, P^{(i+1)} = P^{(i)} \times (H^{(i)})^{k_i}, P = P^{(j)}$.

Die $P^{(i)}$ können mit einem Multiplizierer, die $H^{(i)}$ mit einem Quadrierer berechnet werden.

i	k	k_i	$H^{(i)}$	$P^{(i)}$
0	1010111	1	A	1
1	1010111	1	A^2	A
2	1010111	1	A^4	A^3
3	1010111	0	A^8	A^7
4	1010111	1	A^{16}	A^7
5	1010111	0	A^{32}	A^{23}
6	1010111	1	A^{64}	A^{23}
7	1010111		A^{128}	A^{87}

Tabellen-Multiplizierer

Die Schaltfunktion eines Multiplizierers kann auch durch ein ROM realisiert werden.

Dabei wird die Konkatenation der beiden Operanden als Adresse verwendet.

Zur vorzeichenlosen Multiplikation wird ein ROM der Größe $2^{l+m} \times (l + m - 1)$ benötigt (das niederwertigste Bit ergibt sich direkt aus $P_0 = A_0 B_0$).

Für große Operandenlängen scheidet das Verfahren damit als zu aufwendig aus.

Ein rekursiv aufgebauter Multiplizierer kann aber mit kleinen Tabellen-Multiplizierern arbeiten.

Ein ROM zur Quadrierung in $\mathrm{UInt}_2(l)$ hat dagegen nur die Größe $2^l \times (2l - 2)$ und lässt sich damit leichter realisieren (Reduzierung auf $2^l \times (2l - 3)$ Bit möglich).

Realisierung eines Multiplizierers in $\mathrm{UInt}_2(l)$ durch einen Quadrierer

Ausgangspunkt: $4 \times A \times B = (A+B)^2 - (A-B)^2$ nach Vorsortierung $A \geq B$.

Die beiden niederwertigsten Bits der Quadrate sind irrelevant (stimmen überein).

Erforderlich ist ein Quadrierer für Operanden in $\mathrm{UInt}_2(l+1)$.

Bei Auslegung als Tabellenquadrierer sind dazu $2 \times l \times 2^{l+1}$ Bit Speicherplatz nötig.

Bei Parallelberechnung beider Quadrate wird zusätzlich ein Quadrierer in $\mathrm{UInt}_2(l)$ benötigt.

Optimierung:

$$A \times B = \begin{cases} ((A+B)/2)^2 - ((A-B)/2)^2 & \text{für gerades } A+B, \\ ((A-1+B)/2)^2 - ((A-1-B)/2)^2 + B & \text{sonst.} \end{cases}$$

Nun ist nur noch ein Quadrierer in $\mathrm{UInt}_2(l)$ nötig.

Allerdings muss halbiert werden (Rechtsverschiebung),
eine Fallunterscheidung ist nötig (Prüfbit fällt bei der Rechtsverschiebung ab)
und es wird ggf. zusätzlich addiert.

Beispiel (ohne Parallelität): Zur Berechnung von 16-Bit-Produkten aus 8-Bit-Operanden genügt eine Wertetabelle mit 1024 Byte (512 Byte in optimierter Ausführung).