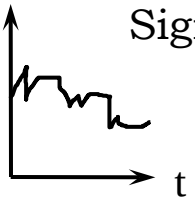
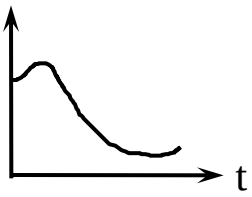
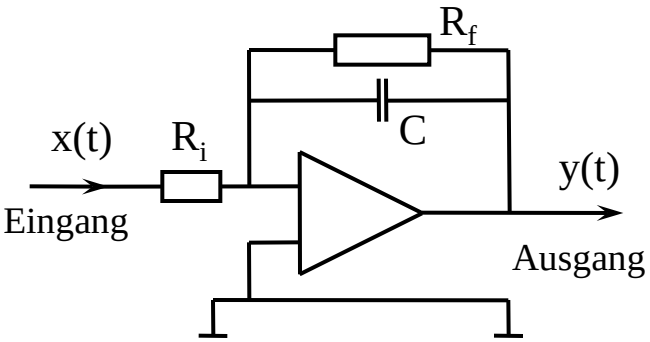
Signalfilterung

Eingangssignal x(t), z.B. von einem Sensor



Ausgangssignal y(t), z.B. für einen Aktuator





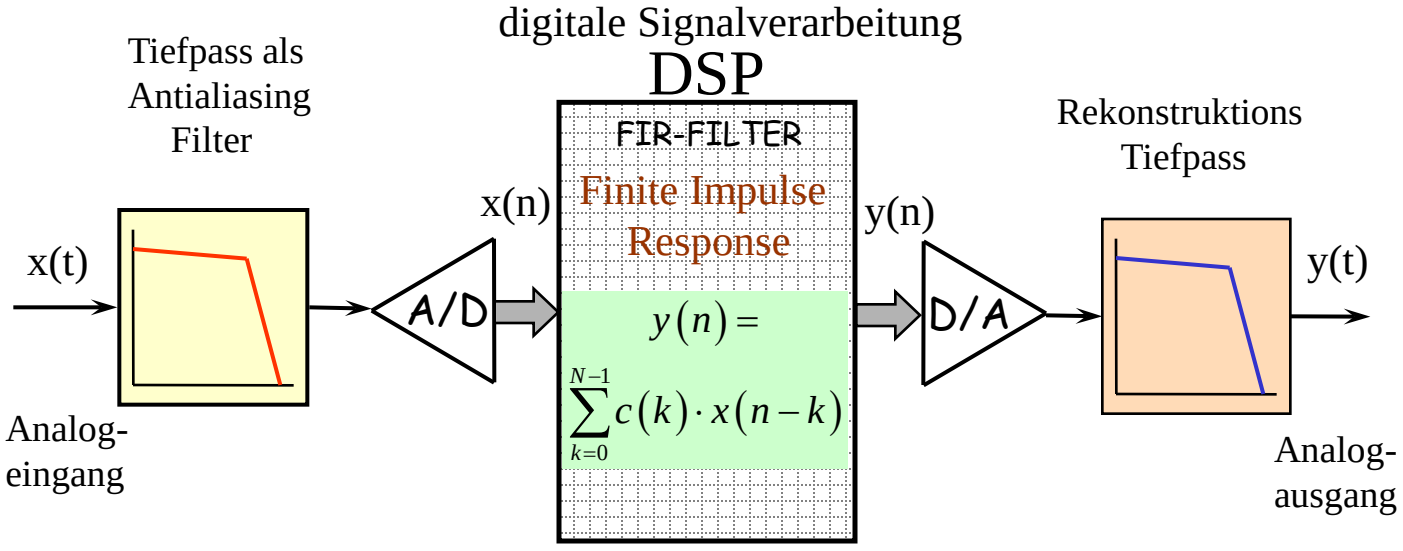
Analoge Filterschaltung (Tiefpass)

Mathematischer Zusammenhang für die analoge Signalverarbeitung

Ansatz mit komplexen Amplituden:

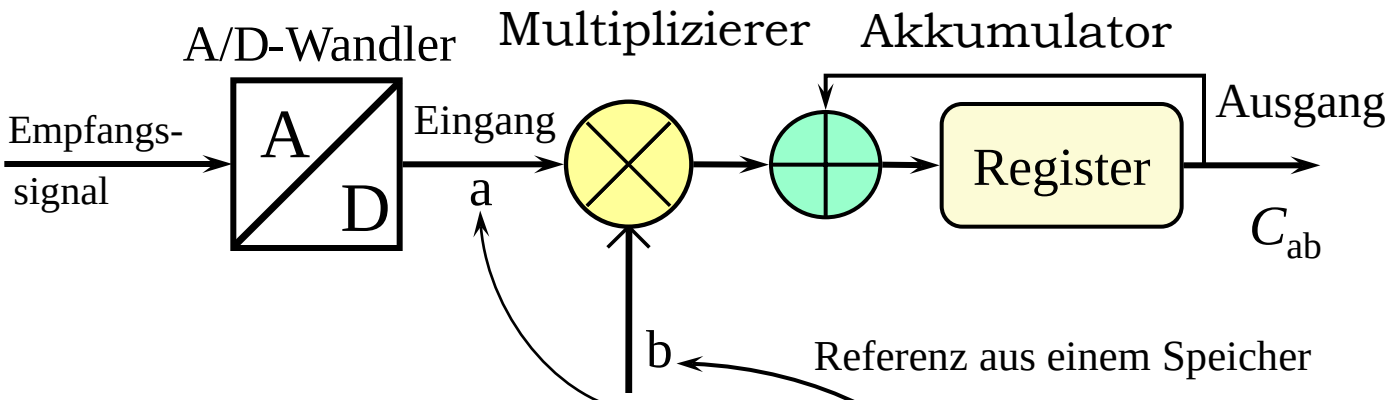
$$\frac{\underline{Y}}{\underline{X}} = -\frac{R_f}{R_i} \cdot \left[\frac{1}{1 + j\omega R_f \cdot C} \right], \text{ mit } y(t) = \mathrm{Re}\left\{ \underline{Y} \cdot e^{j\omega t} \right\}, x(t) = \mathrm{Re}\left\{ \underline{X} \cdot e^{j\omega t} \right\}$$

Gesamtaufbau bei digitaler Signalverarbeitung



Bedeutung der MAC-Operation (I)

Korrelationsempfänger



Korrelation allgemein: $C_{ab}(k) = \sum_{i=0}^{N-1} a(i) \cdot b(k+i)$

Autokorrelation: $C_{aa}(0) = \sum_{i=0}^{N-1} a(i) \cdot a(i) = \sum_{i=0}^{N-1} \left[a(i)\right]^2$

Multiplikation von ganzen Zweierkomplementzahlen s. Gl. (1.9)

$$A \cdot B = -2^{2N-1} + 2^N + a_{N-1} \cdot b_{N-1} \cdot 2^{2N-2} + \sum_{j=0}^{N-2} \sum_{i=0}^{N-2} a_i \cdot b_j \cdot 2^{i+j} +$$

$$+ \sum_{i=0}^{N-2} \overline{a_{N-1} b_i} \cdot 2^{i+N-1} + \sum_{i=0}^{N-2} \overline{b_{N-1} a_i} \cdot 2^{i+N-1}$$

Multiplikation mit Akkumulation (MAC)

$$C'_{ab}(k) = a(\nu) \cdot b(\nu+k) + \sum_{i=0}^{\nu-1} a(i) \cdot b(i+k)$$

$$S_\nu = a \cdot b + S_{\nu-1}$$

Beispiel für Akkulänge N=32

$$S_\nu = -2^{31} + 2^8 + \sum_{i=15}^{30} 2^i + a_7 b_7 \cdot 2^{14} + \sum_{i=0}^{6}\sum_{j=0}^{6} a_i b_j \cdot 2^{i+j}$$

$$+\sum_{i=0}^{6} \overline{a_7 b_i} \cdot 2^{7+i} + \sum_{i=0}^{6} \overline{b_7 a_i} \cdot 2^{7+i} - s_{(\nu-1)31} \cdot 2^{31} + \sum_{i=0}^{30} s_{(\nu-1)i} \cdot 2^i$$

Vorzeichenerweiterung

Summe im Akku

Zwischenschritte der ZK-Multiplikation (I)

$$A \cdot B = \left(-a_{N-1} \cdot 2^{N-1} + \sum_{i=0}^{N-2} a_i 2^i \right) \cdot \left(-b_{N-1} \cdot 2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i \right) =$$

$$= a_{N-1} \cdot b_{N-1} \cdot 2^{2N-2} + \sum_{j=0}^{N-2}\sum_{i=0}^{N-2} a_i \cdot b_j \cdot 2^{i+j} - a_{N-1} \cdot 2^{N-1} \cdot \sum_{i=0}^{N-2} b_i 2^i - b_{N-1} \cdot 2^{N-1} \cdot \sum_{i=0}^{N-2} a_i 2^i$$

Ergebnis ist wieder eine Zweierkomplementzahl

$$A \cdot B = -2^{2N-1} + 2^N + a_{N-1} \cdot b_{N-1} \cdot 2^{2N-2} + \sum_{j=0}^{N-2}\sum_{i=0}^{N-2} a_i \cdot b_j \cdot 2^{i+j} +$$

$$+ \sum_{i=0}^{N-2} \overline{a_{N-1} b_i} \cdot 2^{i+N-1} + \sum_{i=0}^{N-2} \overline{b_{N-1} a_i} \cdot 2^{i+N-1}$$

Zwischenschritte der ZK-Multiplikation (II)

$$-a_{N-1}\cdot 2^{N-1}\cdot\sum_{i=0}^{N-2}b_i 2^i = -2^{N-1}\cdot\sum_{i=0}^{N-2}a_{N-1}b_i\cdot 2^i = 2^{N-1}\cdot\left[-\sum_{i=0}^{N-2}a_{N-1}b_i\cdot 2^i\right.$$

Definition des ZK

$$\mathrm{A}+\overline{\mathrm{A}}=2^{\eta}-1 \quad\Rightarrow\quad -\mathrm{A}=-2^{\eta}+\overline{\mathrm{A}}+1$$

hier: $\eta = \mathrm{N}-1$

$$2^{N-1}\left[-2^{N-1}+\sum_{i=0}^{N-2}\overline{a_{N-1}b_i}\cdot 2^i+1\right] = -2^{2N-2}+\sum_{i=0}^{N-2}\overline{a_{N-1}b_i}\cdot 2^{i+N-1}+2^{N-1}$$

ebenso: $-b_{N-1}\cdot 2^{N-1}\cdot\sum_{i=0}^{N-2}a_i 2^i = -2^{2N-2}+\sum_{i=0}^{N-2}\overline{b_{N-1}a_i}\cdot 2^{i+N-1}+2^{N-1}$

$$-2^{2N-1}+2^{N}+\sum_{i=0}^{N-2}\overline{a_{N-1}b_i}\cdot 2^{i+N-1}+\sum_{i=0}^{N-2}\overline{b_{N-1}a_i}\cdot 2^{i+N-1}$$

Bedeutung der MAC-Operation (II)

Die diskrete Fouriertransformation (DFT) und ihre heutigen Anwendungen:

- Internetzugang mit DSL (digital subscriber line)
- digital video broadcasting (terrestrial) – DVB-T
- digital audio broadcasting - DAB
- digital radio mondiale – DRM
- neuer Mobilfunk – LTE (Long Term Evolution)

Spektrum Abtastwerte der Zeitfunktion

DFT:
$$G\left(\frac{n}{NT}\right)=\sum_{k=0}^{N-1} g\left(kT\right)\cdot \mathrm{e}^{-\mathrm{j}2\pi\frac{nk}{N}},\quad n=0,1...N-1$$

Zerlegung des komplexen „Drehfaktors $e^{-j\ldots}$“ in Real- und Imaginärteil

$$\mathrm{e}^{-\mathrm{j}2\pi\frac{nk}{N}}=\cos\left(2\pi\frac{nk}{N}\right)-\mathrm{j}\sin\left(2\pi\frac{nk}{N}\right),\quad k,n\in\left\{0,1,...N-1\right\}$$

Betrachtungen zum Rechenaufwand für DFT und FFT

auch bei reellen Zeitfunktionen ist die DFT in der Regel komplex

- zweifache **NxN**-Matrixmultiplikation $\Rightarrow$ $2N^2$ Multiplikationen

Fast Fourier Transform (FFT) Algorithmen nutzen Redundanzen aus, um die Zahl der Multiplikationen auf **N·ld(N)** zu reduzieren

- in der Regel ist N eine Zweierpotenz

Beispiel: Bei $N=2^{10}=1024$ bringt die FFT eine Reduktion von $2^{21}=2.097.152$ Multiplikationen auf $2^{10}\cdot 10=10.240$

Weiteres zur Bedeutung der schnellen Multiplikation

Neben der Multiplikation wird in der DSV das Quadrieren benötigt z.B. bei der „geometrischen“ Addition gemäß:

$$z = \sqrt{x^2 + y^2}$$

Auf das Berechnen der Wurzel kann häufig verzichtet werden

Mit $$A = -a_n \cdot 2^{N-1} + \sum_{i=0}^{N-2} a_i \cdot 2^i$$

und (1.9)

$$A \cdot B = -2^{2N-1} + 2^N + a_{N-1} \cdot b_{N-1} \cdot 2^{2N-2} + \sum_{j=0}^{N-2} \sum_{i=0}^{N-2} a_i \cdot b_j \cdot 2^{i+j} + \sum_{i=0}^{N-2} \overline{a_{N-1} b_i} \cdot 2^{i+N-1} + \sum_{i=0}^{N-2} \overline{b_{N-1} a_i} \cdot 2^{i+N-1}$$

erhält man wegen $a_i = b_i$ und $\overline{a_i} \cdot a_i = a_i$ sofort - s. (1.12):

$$A^2 = -2^{2N-1} + 2^N + a_{N-1} \cdot 2^{2N-2} + \sum_{i=0}^{N-2} \overline{a_{N-1} \cdot a_i}\, 2^{i+N} + \sum_{i=0}^{N-2} \sum_{j=0}^{N-2} a_i \cdot a_j \cdot 2^{i+j}$$

Bei der Hardwarerealisierung der obigen Doppelsumme kann des weiteren ausgenutzt werden, daß

$a_i \cdot a_j = a_j \cdot a_i$ ist.

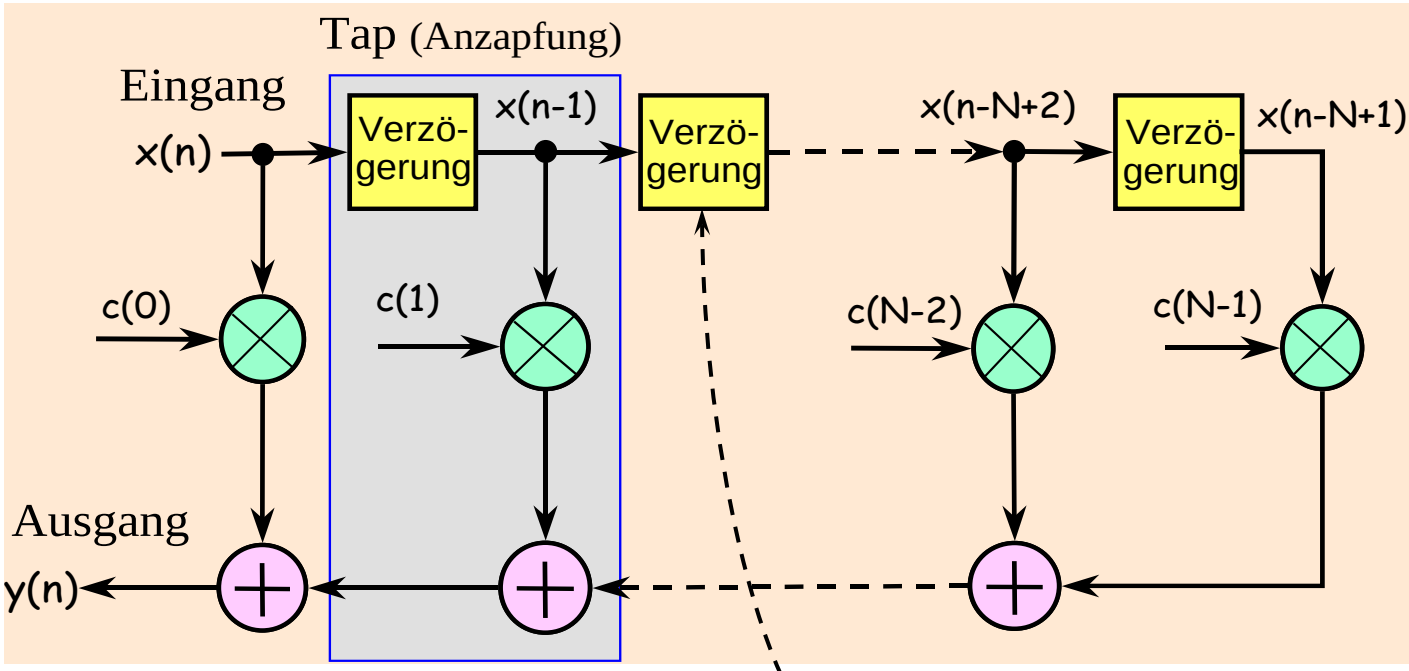
Dadurch kann die Addition dieser Elementarprodukte entfallen, d.h. man schreibt $a_i \cdot a_j$ jeweils in eine Spalte der nächsthöheren Potenz i+j+1 und läßt $a_j \cdot a_i$ weg.

Beispiel:

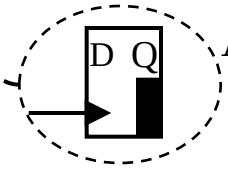
$a_4 a_5$ hat die Wertigkeit 2^9, kommt aber in die Spalte für $2^{10} \Rightarrow a_5 a_4$ braucht dann nicht mehr berücksichtigt zu werden

Bedeutung der MAC-Operation (III)

Digitales Filter mit „endlicher Impulsantwort“
FIR Filter ≡ finite impulse response filter



$$y(n) = \sum_{k=0}^{N-1} c(k) \cdot x(n-k)$$



Anzahl entspr. der Wortbreite in bit

Obwohl die obige Gleichung sehr ähnlich wie die der Korrelation aussieht, gibt es einen beträchtlichen Unterschied:

- Beim Korrelator ist die MAC-Geschwindigkeit gleich der Abtastrate des A/D-Wandlers
- Beim FIR-Filter hingegen erfordert jeder neue Abtastwert x(n) die komplette Berechnung der obigen Summe, d.h. ***N*** Multiplikationen und ***N*** Akkumulationen

Ein typisches Kennzeichen digitaler Filter ist eine hohe Anzahl von Taps, z.B. 100 und mehr.

- Bei einem DSP, der nur eine MAC-Operation pro Maschinenzyklus ausführt, werden bei einer Abtastrate von 1MHz für N=100 $100 \cdot 10^6$ Instruktionen pro Sekunde, also 100 MIPS benötigt.

- heutige preisgünstige DSPs bieten etwa 5000 “echte“ MIPS teurere liefern über 10.000 MIPS, nehmen aber “viel“ Leistung auf

Die Mikrorechnerkonzepte CISC, RISC und ARM

CISC ≡ **C**omplex **I**nstruction **S**et **C**omputer

- große integrierte Mikroprogramme
- „mächtige“ Befehle zur Erleichterung der Programmierarbeit

RISC ≡ **R**educed **I**nstruction **S**et **C**omputer

- Alle Befehle laufen in einem Taktschritt ab
- Der OP-Code hat, zusammen mit allen benötigten Operanden, eine feste Länge, die gleich der Datenbusbitbreite ist
- aufgrund großer Busbreite sind keine zusätzlichen Zugriffe für den Transfer von Operanden nötig
- Speicherzugriffe nur bei Laden und Abspeichern direkte Bearbeitung von Daten in Registern
- das RISC-Konzept benötigt eine große Anzahl von Registern
- Befehlsausführung erfolgt „festverdrahtet“ in einem Taktschritt, d.h. es wird kein Mikroprogramm implementiert

ARM ≡ **A**dvanced **R**ISC **M**achine

- auf breiter Front höchst erfolgreich

Der Parallelmultiplizierer

- ist das Kernstück der ALU moderner Mikrorechner aller Art -

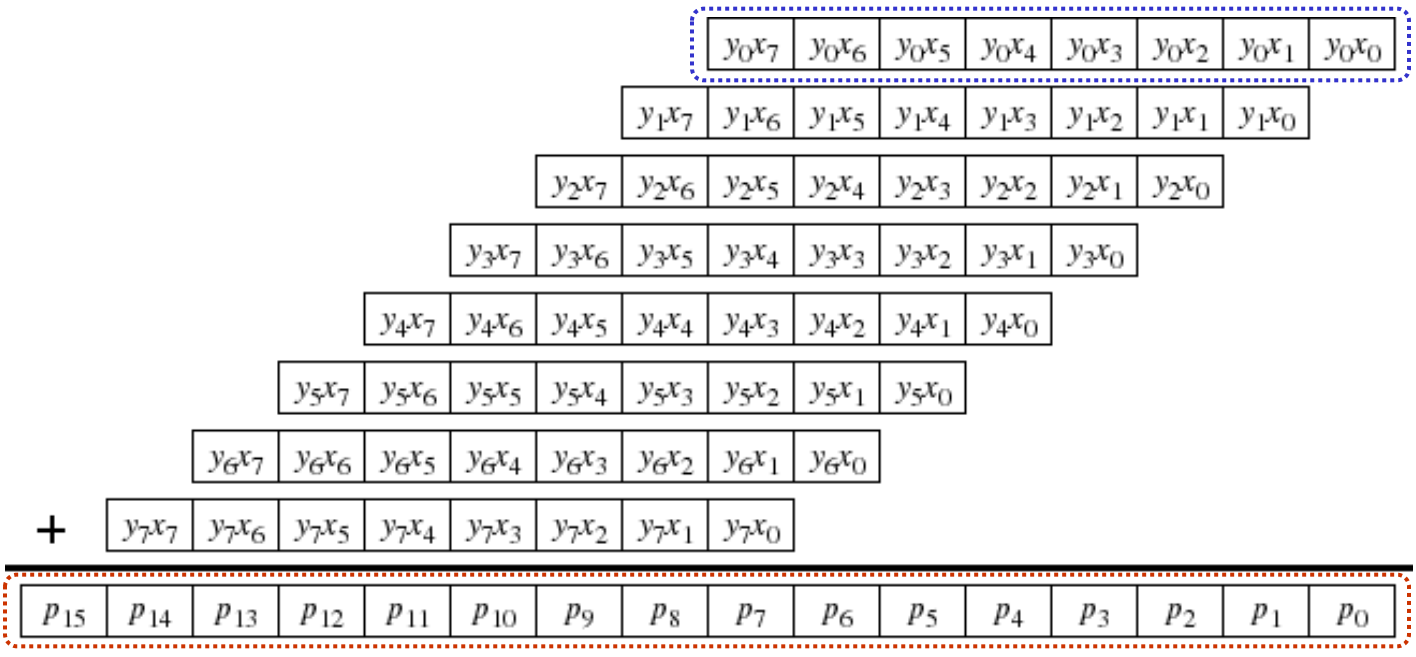
zunächst werden vorzeichenlose Ganzzahlen betrachtet

$$X \cdot Y = \left(\sum_{j=0}^{N-1} x_j 2^j \right) \cdot \left(\sum_{i=0}^{N-1} y_i 2^i \right) = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} x_i \cdot y_j \cdot 2^{i+j}$$

für ein Beispiel mit N=8 ergibt sich

$$X \cdot Y = \sum_{j=0}^{7} \sum_{i=0}^{7} x_i \cdot y_j \cdot 2^{i+j}$$

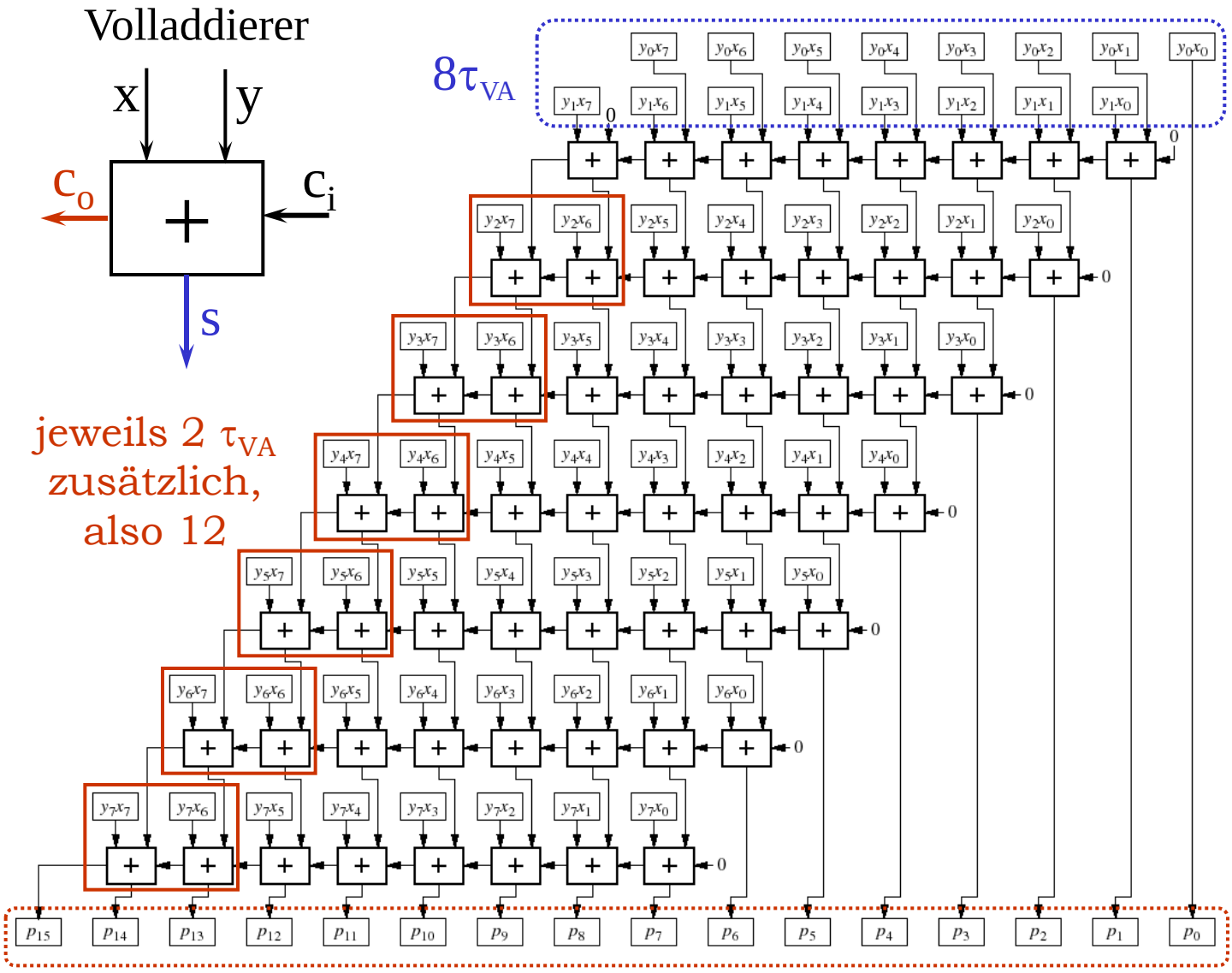
elementare Binärprodukte aus UND-Verknüpfung



fertiges Produkt

Realisierung eines 8-bit-Multiplizierers für vorzeichenlose Ganzzahlen mittels „Ripple-Carry-Addition“ unter Verwendung von 56 Volladdierern

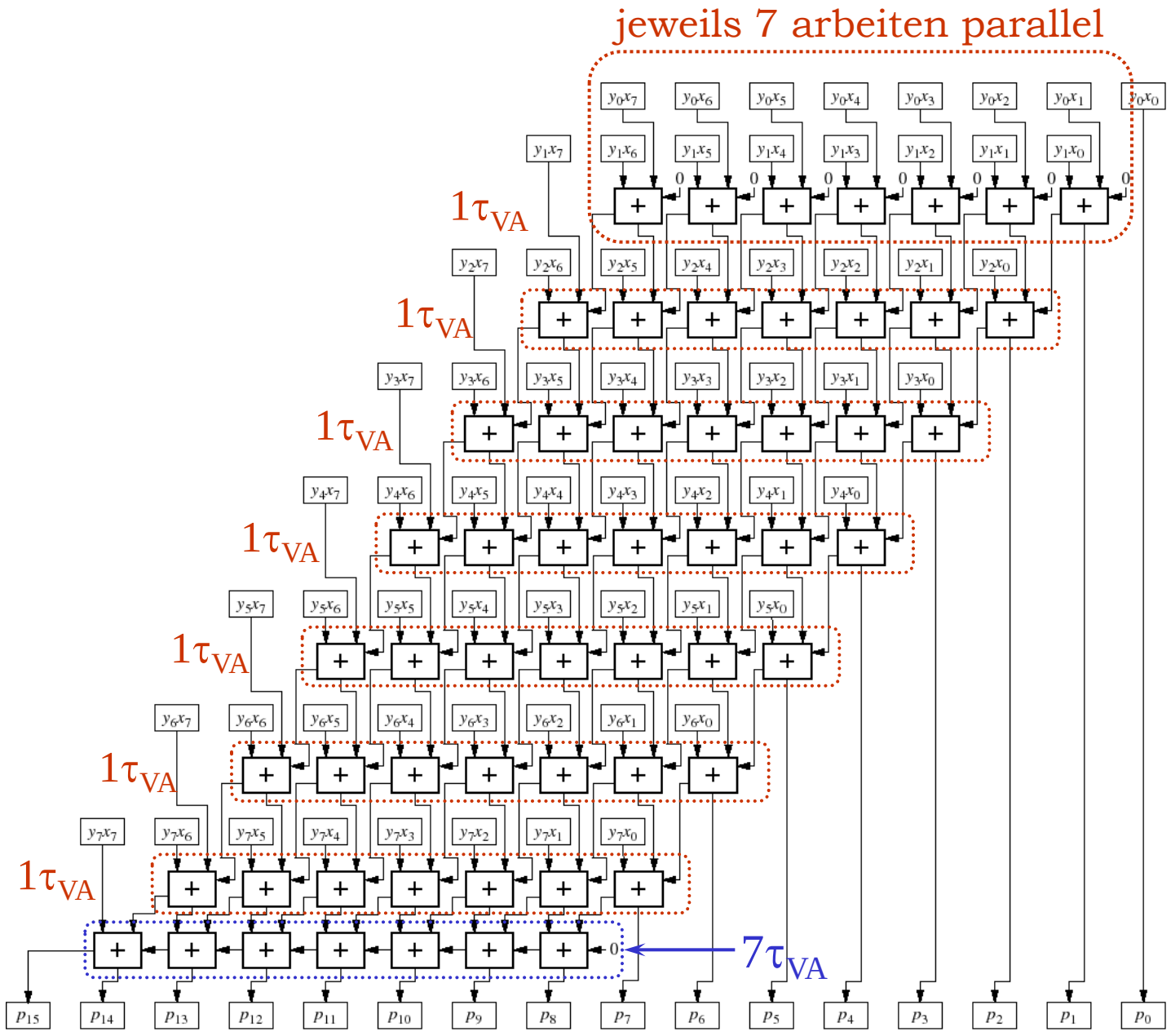
elementare Binärprodukte aus UND-Verknüpfung



fertiges Produkt

gesamte Rechenzeit: $1 \cdot \tau_{UND} + 20 \cdot \tau_{VA}$

Beschleunigung durch „Aufbewahren“ der Carry-Bits und Verrechnung in der Folgestufe (Carry-Save-Addition mit 56 Volladdierern)



gesamte Rechenzeit: $1 \cdot \tau_{UND} + 14 \cdot \tau_{VA}$

Parallelmultiplikation vorzeichenbehafteter Zweierkomplementzahlen

$$A \cdot B = \left(-a_{N-1} \cdot 2^{N-1} + \sum_{i=0}^{N-2} a_i 2^i \right) \cdot \left(-b_{N-1} \cdot 2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i \right)$$

Gleichung (1.9)

$$A \cdot B = -2^{2N-1} + 2^N + a_{N-1} \cdot b_{N-1} \cdot 2^{2N-2} + \sum_{j=0}^{N-2} \sum_{i=0}^{N-2} a_i \cdot b_j \cdot 2^{i+j} +$$

$$+ \sum_{i=0}^{N-2} \overline{a_{N-1} b_i} \cdot 2^{i+N-1} + \sum_{i=0}^{N-2} \overline{b_{N-1} a_i} \cdot 2^{i+N-1}$$

NAND

AND

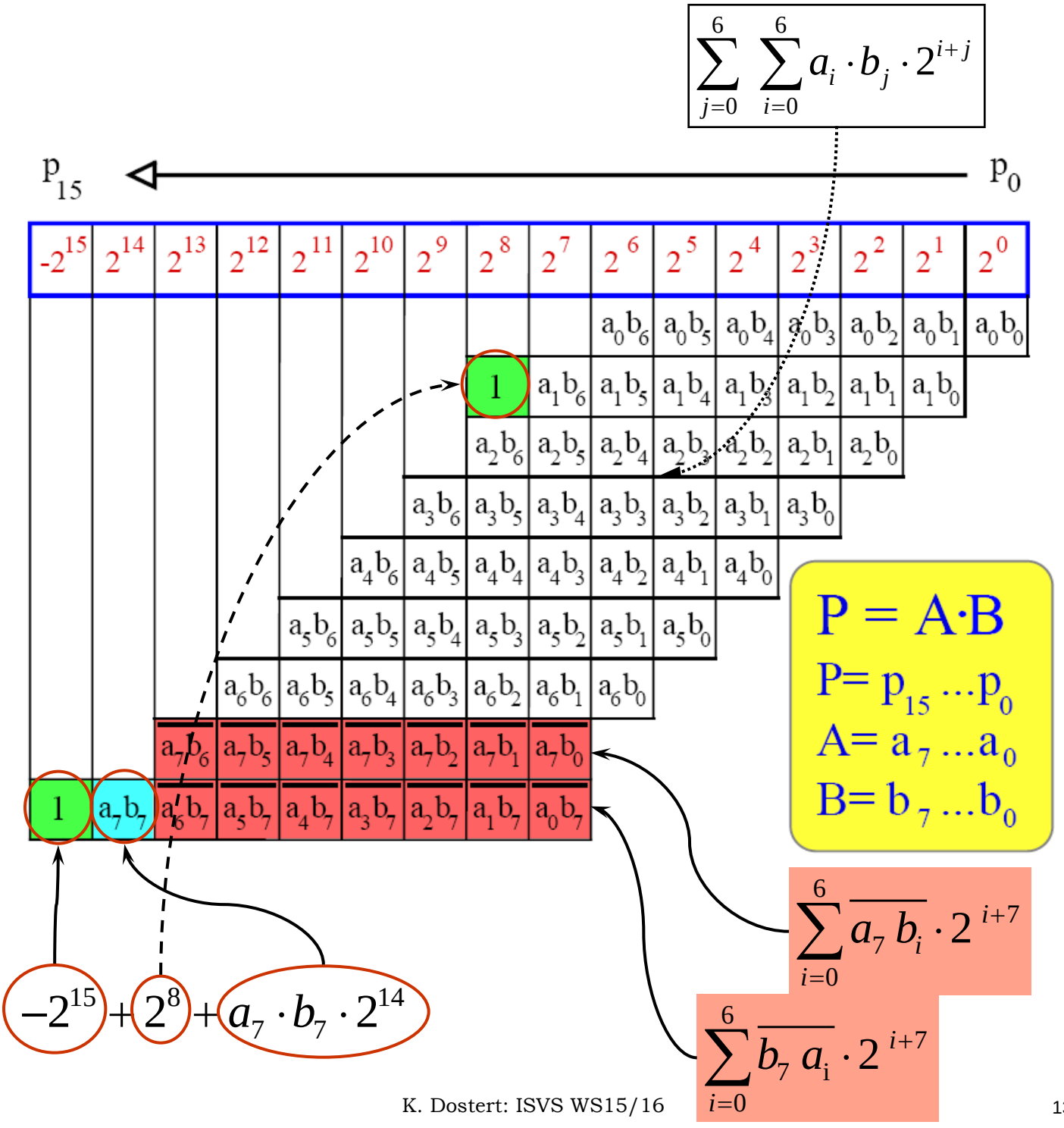
Beispiel für N=8

$$P = A \cdot B = -2^{15} + 2^8 + a_7 \cdot b_7 \cdot 2^{14} + \sum_{j=0}^{6} \sum_{i=0}^{6} a_i \cdot b_j \cdot 2^{i+j}$$

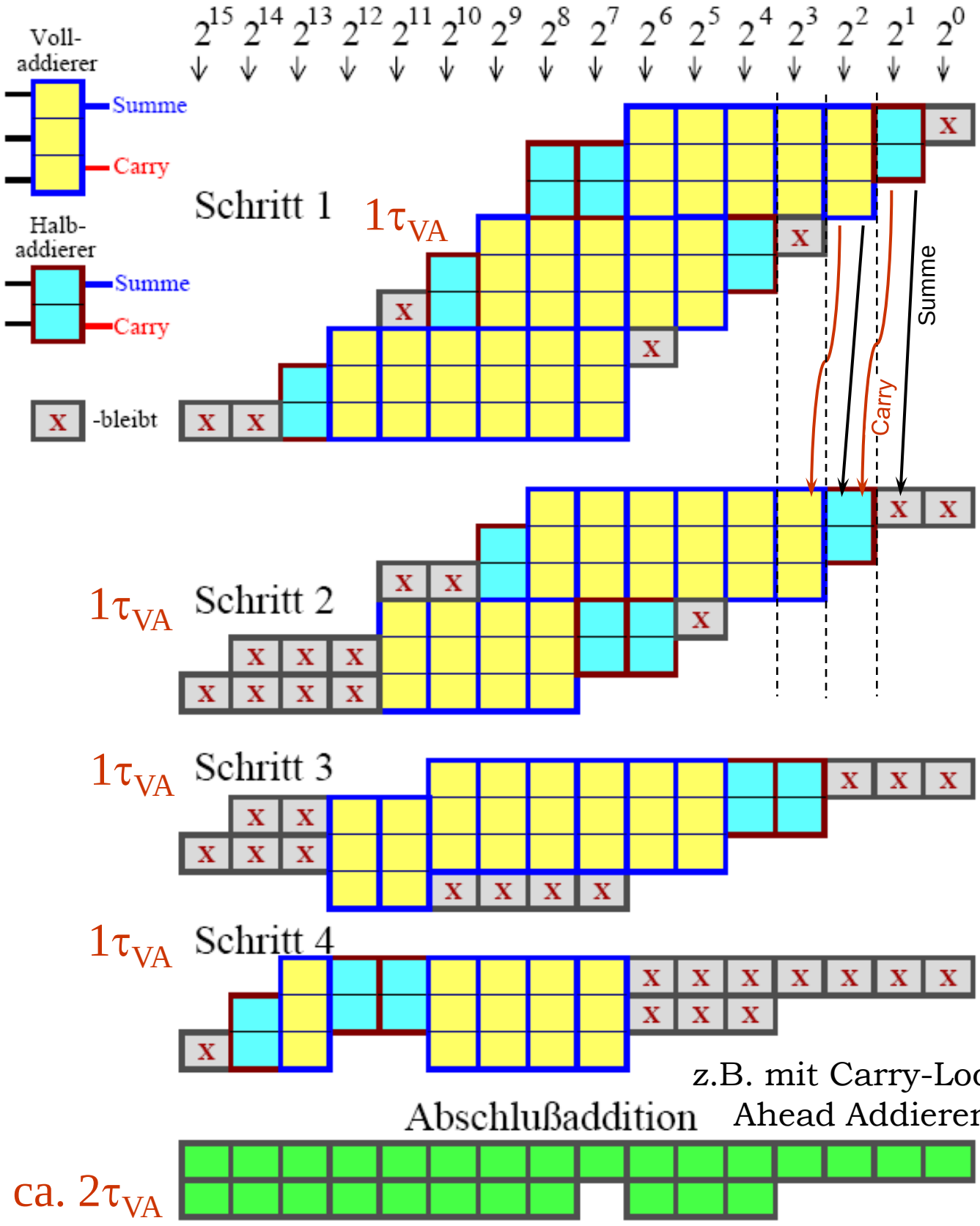
$$+ \sum_{i=0}^{6} \overline{a_7 b_i} \cdot 2^{i+7} + \sum_{i=0}^{6} \overline{b_7 a_i} \cdot 2^{i+7}$$

$$-2^{15} + 2^8 + a_7 \cdot b_7 \cdot 2^{14} + \sum_{j=0}^{6} \sum_{i=0}^{6} a_i \cdot b_j \cdot 2^{i+j}$$

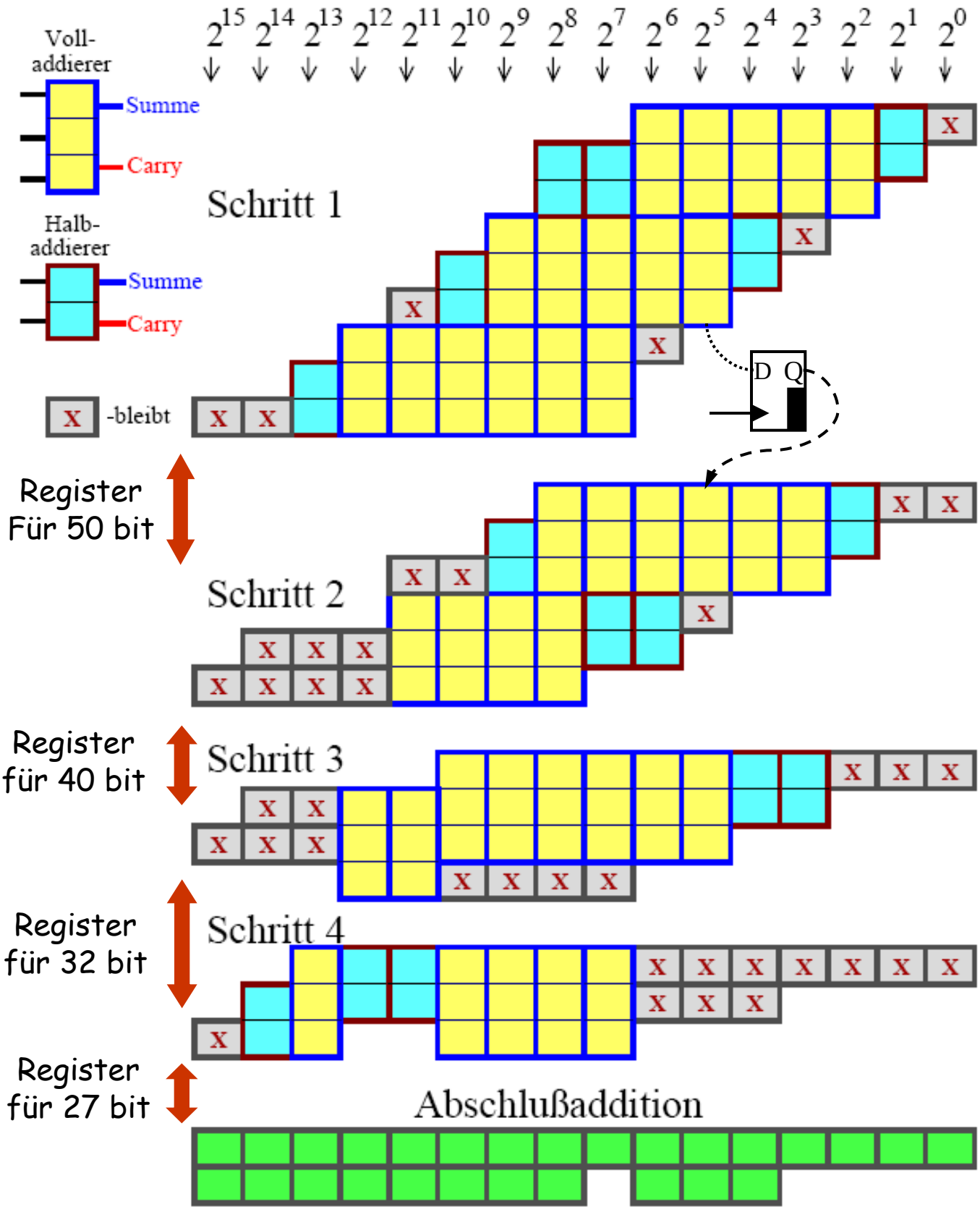
$$+\sum_{i=0}^{6} \overline{a_7 b_i} \cdot 2^{i+7} + \sum_{i=0}^{6} \overline{b_7 a_i} \cdot 2^{i+7}$$



Abarbeitung der Matrix aus mit Hilfe von 15 Halb- und 39 Volladdierern in 4 Schritten



Beschleunigung auf ca. $2\tau_{VA}$ mittels Pipelining



nur die Stufe mit der maximalen Verzögerung ist maßgebend

Schnelle Carryberechnung:
Carry-Look-Ahead-Addierer

Wann entsteht bei einem *N*-bit-Binäraddierer mit den Eingangsoperanden $A=a_0 \ldots a_{N-1}$ und $B=b_0 \ldots b_{N-1}$ ein Übertrag c_{i+1} in der Stufe i, unabhängig von den Werten aller Binärvariablen der Vorstufen?

falls $a_i \cdot b_i = 1$
dieser Fall wird als Carry-Erzeugung (Carry-Generate) bezeichnet, $g_i = a_i \cdot b_i$

Wann wird bei diesem Addierer ein Übertrag c_{i+1}, über die Addierstufe i weitergegeben, der nicht in dieser Stufe entstanden ist. Dazu muss ein Eingangsübertrag $c_i=1$ vorliegen.

falls $a_i + b_i = 1$
dieser Fall wird als Carry-Weiterleitung

(Propagate) bezeichnet, $p_i = a_i + b_i$

Schreibweise für Carry-Generate

$$g_i = a_i \cdot b_i$$

und für Carry-Propagate

$$p_i = a_i + b_i$$

Der Übertrag in die jeweils nächsthöhere Zweierpotenz ist damit:

$$c_{i+1} = g_i + c_i \cdot p_i$$

Damit lassen sich die booleschen Gleichungen für die einzelnen Überträge angeben:

$$c_1 = g_0 + p_0 \cdot c_0$$

$$c_2 = g_1 + p_1 \cdot c_1 = g_1 \oplus p_1 \odot (g_0 + p_0 \cdot c_0)$$

$$c_3 = g_2 + p_2 \cdot c_2 = g_2 \oplus p_2 \odot [g_1 + p_1 \cdot (g_0 + p_0 \cdot c_0)]$$

$$c_4 = g_3 + p_3 \cdot c_3 = g_3 \oplus p_3 \odot [g_2 + p_2 \cdot [g_1 + p_1 \cdot (g_0 + p_0 \cdot c_0)]]$$

$$c_5 = g_4 + p_4 \cdot c_4 = ...$$

Nachteil: Die Verzögerung wächst mit der Ordnungszahl des Übertrags jeweils um 2 Gatterlaufzeiten

=> bessere Lösung suchen!

Alternative Teilsummenberechnung mit EXOR

$$a_i \oplus b_i = a_i \cdot \overline{b_i} + b_i \cdot \overline{a_i} + a_i \cdot \overline{a_i} + b_i \cdot \overline{b_i} =$$
$$= (a_i + b_i) \cdot (\overline{a_i} + \overline{b_i}) = (a_i + b_i) \cdot \overline{a_i \cdot b_i}$$
$$= p_i \cdot \overline{g_i}$$

$=0$

des weiteren kann die Carry-Berechnung

auch in der Form $c_{i+1} = g_i + c_i \cdot p_i$

geschrieben werden, da $c_{i+1} = g_i \cdot p_i + c_i \cdot p_i = p_i \cdot (g_i + c_i)$

$$p_i \cdot g_i = (a_i + b_i) \cdot a_i \cdot b_i = a_i \cdot b_i + a_i \cdot b_i = a_i \cdot b_i \equiv g_i$$

$$c_1 = p_0 \cdot (g_0 + c_0)$$

distributives Gesetz: A+BCD…=(A+B)(A+C)(A+D)(A+…)

$$c_2 = p_1 \cdot (g_1 + c_1) = p_1 \cdot \left[\underbrace{g_1}_{A} + \underbrace{p_0}_{B} \cdot \left(\underbrace{g_0 + c_0}_{C} \right) \right] = p_1 \cdot \left[\underbrace{(g_1 + p_0)}_{A+B} \cdot \underbrace{(g_1 + g_0 + c_0)}_{A+C} \right]$$

$$c_3 = p_2 (g_2 + c_2) = p_2 \cdot \left[\underbrace{g_2}_{A} + \underbrace{p_1}_{B} \cdot \left(\underbrace{g_1 + p_0}_{C} \right) \cdot \left(\underbrace{g_1 + g_0 + c_0}_{D} \right) \right]$$

$$= p_2 \odot \underbrace{(g_2 \oplus p_1)}_{A+B} \odot \underbrace{(g_2 \oplus g_1 \oplus p_0)}_{A+C} \odot \underbrace{(g_2 \oplus g_1 \oplus g_0 \oplus c_0)}_{A+D}$$

$\odot$ 3-fach UND

$\oplus$ 3 par. ODER

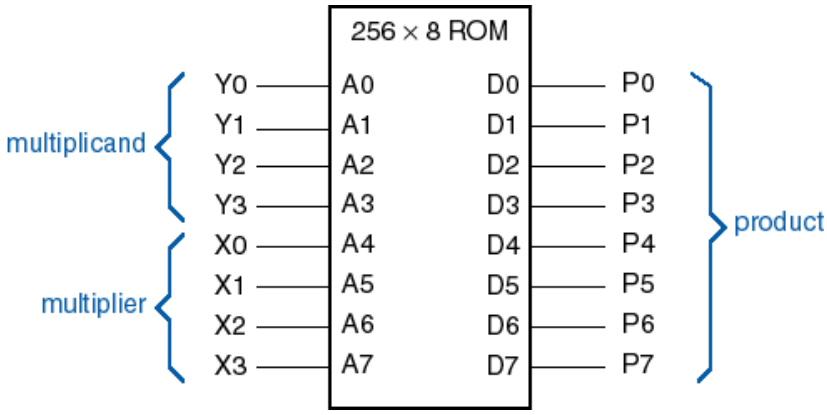
Jetzt nur noch 3 Gatterlaufzeiten für **jedes** Carrybit!

$$c_4 = p_3 (g_3 + c_3) =$$
$$p_3 \cdot (g_3 + p_2) \cdot (g_3 + g_2 + p_1) \cdot (g_3 + g_2 + g_1 + p_0) \cdot (g_3 + g_2 + g_1 + g_0 + c_0)$$

Summenbitberechnung:

mit $a_i \oplus b_i = p_i \cdot \overline{g_i}$ wird $s_i = a_i \oplus b_i \oplus c_i = p_i \cdot \overline{g_i} \oplus c_i$

Schnelle speicherbasierte Multiplikation



Mit einem Festwertspeicher kann in einfacher Weise ein sehr schneller Multiplizierer realisiert werden. Der Speicher (ROM) hat eine 8bit-Adresse und speichert 8bit breite Daten. Wenn man die unteren 4 Adreßbits (A0...A3) als Multiplikand (*Y*) und die oberen (A4...A7) als Multiplikator (*X*) auffaßt, dann sind 256 Kombinationen für das Produkt möglich.

00:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
10:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
20:	00	02	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
30:	00	03	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
40:	00	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
50:	00	05	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
60:	00	06	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
70:	00	07	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
80:	00	08	10	18	20	28	30	38	40	48	50	58	60	68	70	78
90:	00	09	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A0:	00	0A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B0:	00	0B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C0:	00	0C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D0:	00	0D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E0:	00	0E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F0:	00	0F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Alle vorkommenden Produkte sind im Speicher abgelegt, und die Rechenzeit ist die maximale Zugriffzeit auf eine Speicherstelle. Mit dieser Methode kann man leider nur Multiplizierer bauen, die auf kleine Wortbreiten beschränkt sind. Wollte man so einen 32x32bit Multiplizierer realisieren, wäre ein Baustein mit 64 Adreßbits und einer Kapazität von 2^{64} (mehr als $18 \cdot 10^{18}$) Worten von je 64 bit Länge erforderlich.

Beispiel zur Reverse-Carry-Arithmetik

- Das Register M_n wird auf \$0000 gesetzt.

- R_n-Inhalt wird „reversiert“, d.h. das MSB wird zum LSB und entsprechend wird Bit für Bit im Platz vertauscht.

- N_n-Inhalt wird ebenfalls „reversiert“;
  bei $N_n=2^{k-1}$ hat man dann immer den Wert ...01.

- Es wird normal binär addiert

- Das Ergebnis wird „reversiert“
  $\Rightarrow$ jetzt hat man den endgültigen Adresszeiger

Es wird eine Basis-2-FFT betrachtet, d.h. das Offset-Register N_n enthält eine Zweierpotenz

es sei $N=8 \Rightarrow k=3$;

N_n hat den Inhalt $2^{k-1} = 100$

R_n enthalte den Startwert 000

Basis-2-FFT mit N=8

2^2	2^1	2^0	Operation	Adresspointer
0	0	0	R_n	X(0) ≙ X(0)
0	0	1	N_n reversiert	
0	0	1	$R_n + N_n$	
1	0	0	Summe reversiert	X(4) ≙ X(1)
0	0	1	R_n	
0	0	1	N_n reversiert	
0	1	0	$R_n + N_n$	
0	1	0	Summe reversiert	X(2) ≙ X(2)
0	1	0	R_n	
0	0	1	N_n reversiert	
0	1	1	$R_n + N_n$	
1	1	0	Summe reversiert	X(6) ≙ X(3)
0	1	1	R_n	
0	0	1	N_n reversiert	
1	0	0	$R_n + N_n$	
0	0	1	Summe reversiert	X(1) ≙ X(4)
1	0	0	R_n	
0	0	1	N_n reversiert	
1	0	1	$R_n + N_n$	
1	0	1	Summe reversiert	X(5) ≙ X(5)
1	0	1	R_n	
0	0	1	N_n reversiert	
1	1	0	$R_n + N_n$	
0	1	1	Summe reversiert	X(3) ≙ X(6)
1	1	0	R_n	
0	0	1	N_n reversiert	
1	1	1	$R_n + N_n$	
1	1	1	Summe reversiert	X(7) ≙ X(7)