

Wiederholung:

Sei $a = a_{n-1} \dots a_0$ eine Folge von Ziffern, $a_i \in \{0,1\}$

Binärdarstellung: $\langle a \rangle = \sum_{i=0}^{n-1} a_i 2^i$

Zweierkomplement: $[a_n a_{n-1} \dots a_0] = \sum_{i=0}^{n-1} a_i 2^i - a_n 2^n$

Rechenregel: $-[a] = [\overline{a}] + 1$ mit $\overline{a} = \overline{a}_n \overline{a}_{n-1} \dots \overline{a}_0$

Addierer

Gegeben: 2 positive Binärzahlen
$$<$$
a $>$ = $<$ a_{n-1} ... a₀ $>$, $<$ b $>$ = $<$ b_{n-1} ... b₀ $>$, Eingangsübertrag c \in {0,1}

Wegen
$$\langle a \rangle + \langle b \rangle + c \le 2 \cdot (2^n - 1) + 1 = 2^{n+1} - 1$$

genügen n+1 Ausgänge des Schaltkreises.

BB TII 11.1/3

Definition 11.1

Ein **n-Bit Addierer** ist ein Schaltkreis, der die folgende Boolesche Funktion berechnet:

$$\begin{split} +_n \colon \boldsymbol{B}^{2n+1} &\to \boldsymbol{B}^{2n+1} \,, \\ (a_{n-1}, \, ..., \, a_0 \,, \, b_{n-1}, \, ..., \, b_0 \,, \, c) &\to (s_n, \, ..., \, s_0) \qquad \text{mit} \\ ~~&= \, = < a_{n-1} \, ... \, a_0> \, + < b_{n-1} \, ... \, b_0> \, + \, c \end{split}~~$$

Der Halbaddierer (HA)

Der Halbaddierer dient zur Addition zweier

1-Bit-Zahlen ohne Eingangsübertrag.

Er berechnet die Funktion:

ha :
$$\mathbf{B}^2 \to \mathbf{B}^2$$
 mit ha(a₀, b₀) = (s₁, s₀) mit 2s₁ + s₀ = a₀ + b₀

3 TI 11.1

Funktionstabelle des HA

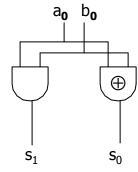
a ₀	b ₀	ha ₁	ha ₀
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Folglich:

$$ha_0 = a_0 \oplus b_0$$
 $ha_1 = a_0 \wedge b_0$

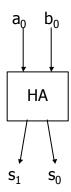
BB TI I 11.1/7

Schaltkreis eines Halbaddierers



dabei:

$$C(HA) = 2$$
, $depth(HA) = 1$



Der Volladdierer (FA)

Der Volladdierer dient zur Addition zweier

1-Bit-Zahlen mit Eingangsübertrag.

Er berechnet die Funktion:

fa :
$$\mathbf{B}^3 \to \mathbf{B}^2$$

mit
$$2s_1 + s_0 = a_0 + b_0 + c$$

BB TII

Funktionstabelle des FA

a_0	b_0	С	fa ₁	fa ₀
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Volladdierer als Funktion von HAs

Aus der Tabelle folgt:

$$fa_0 = a_0 \oplus b_0 \oplus c = ha_0(c, ha_0(a_0, b_0))$$

$$fa_1 = a_0 \wedge b_0 \vee c \wedge (a_0 \oplus b_0)$$

= $ha_1(a_0, b_0) + ha_1(c, ha_0(a_0, b_0))$

Kosten und Tiefe eines FA:

$$C(FA) = 5$$
, $depth(FA) = 3$

BB TII 11.1/11

Realisieren der Schulmethode: Carry Ripple Addierer (CR)

Hierarchisches Vorgehen:

(induktive Definition)

Für n=1: $CR_1 = FA$

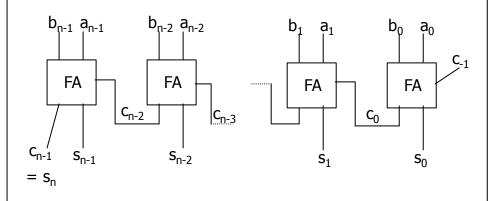
Für n>1: Schaltkreis CR_n wie folgt definiert

Bezeichnung:

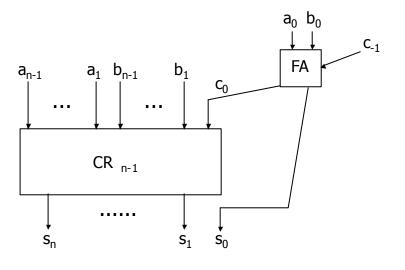
Bezeichne den Eingangsübertrag mit $c_{\text{-}1}$, den Übertrag von Stelle i nach i+1 mit c_{i} .

BB TII 11.1/13

Aufbau eines Carry Ripple Addierers



Schaltbild des n-Carry Ripple Addierers (CR_n)



Satz 11.1

Der Schaltkreis CR_n ist ein n-Bit-Addierer.

Beweis durch Induktion:

- n=1: ✓
- n-1 \rightarrow n: Eingabe an CR_n: (a_{n-1}, ..., a₀, b_{n-1}, ... b₀, c₋₁) Zeige für Ausgabe (s_n, ..., s₀) von CR_n <s> = <s_n ... s₀> = <a_{n-1} ... a₀> + <b_{n-1} ... b₀> +c₋₁

Satz 11.1 (ff)

Nach I.V.:

a)
$$\langle c_0, s_0 \rangle = a_0 + b_0 + c_1$$
 (FA)

b) Für
$$CR_{n-1}$$
: $\langle s_n ... s_1 \rangle = \langle a_{n-1} ... a_1 \rangle + \langle b_{n-1} ... b_1 \rangle + c_0$

Insgesamt:

$$\langle s_n ... s_0 \rangle = 2 \cdot \langle s_n ... s_1 \rangle + s_0$$

I.V.b) =
$$2 \cdot (\langle a_{n-1} ... a_1 \rangle + \langle b_{n-1} ... b_1 \rangle + c_0) + s_0$$

I.V.a) =
$$2 \cdot \langle a_{n-1} \dots a_1 \rangle + a_0 + 2 \cdot \langle b_{n-1} \dots b_1 \rangle + b_0 + c_{-1}$$

= $\langle a \rangle + \langle b \rangle + c_{-1}$

BB TII 11.1/17

Komplexität eines Carry Ripple Addierer

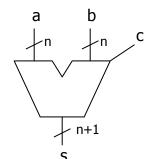
Schaltbild eines n-Bit-Addierers:

Kosten eines CR_n:

$$C(CR_n) = n \cdot C(FA) = 5n$$

Tiefe eines CR_n:

$$depth(CR_n) = 3 + 2(n-1)$$



Einige weitere wichtige Schaltkreise:

- Der n-Bit Inkrementer
- Der n-Bit Multiplexer

BB TI I 11.1/19

Definition 11.2

Ein **n-Bit Inkrementer** berechnet die Funktion:

$$\begin{aligned} &\text{inc}_n \colon \boldsymbol{B}^{n+1} \to \boldsymbol{B}^{n+1} \,, \\ &(a_{n-1}, \, ..., \, a_0 \,, \, c) \to (s_n, \, ..., \, s_0) \qquad \text{mit} \\ &< s_n \, ... \, s_0 > \ = \ < a > \ + \ c \end{aligned}$$

Inkrementer

Ein Inkrementer ist ein Addierer mit

$$b_i = 0 \quad \forall i$$

→ Ersetze in CR_n die FA durch HA.

Kosten und Tiefe:

$$C(INC_n) = n \cdot C(HA) = 2n$$

 $depth(INC_n) = n \cdot depth(HA) = n$

BB TII 11.1/21

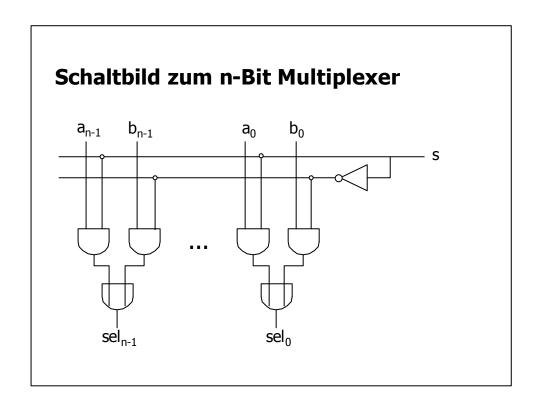
Definition 11.3

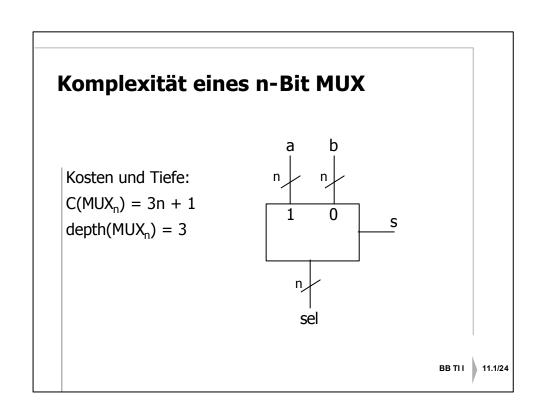
Ein **n-Bit-Multiplexer** (MUX_n) ist ein Schaltkreis, der die folgende Funktion berechnet:

$$\operatorname{sel}_n: \mathbf{B}^{2n+1} \to \mathbf{B}^n \quad \text{ mit}$$

$$sel_{n}(a_{n-1},...,a_{0},b_{n-1},...,b_{0},s) = \begin{cases} (a_{n-1}...a_{0}), \text{falls} & s=1\\ (b_{n-1}...b_{0}), \text{falls} & s=0 \end{cases}$$

$$(sel_n)_i = s \cdot a_i + \overline{s} \cdot b_i$$





Rückkehr zum Addierer

Gibt es billigere Addierer als CR_n ?

Untere Schranken:

$$C(+_n) \ge 2 \cdot n$$
, $depth(+_n) \ge log(n) + 1$

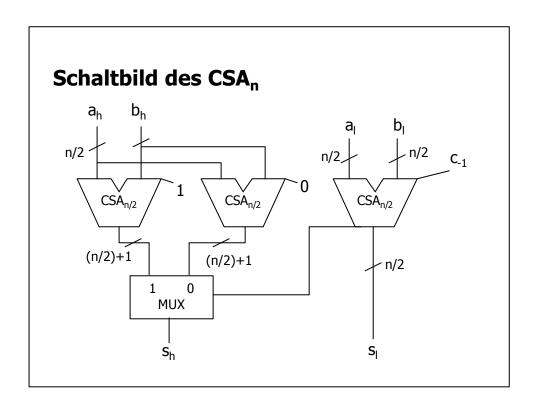
Binäre Bäume mit 2n+1 Blättern haben 2n innere Knoten. Binäre Bäume mit n Blättern haben mindestens Tiefe $\lceil \log n \rceil$.

Im folgenden sei $n = 2^k$.

BB TI I 11.1/25

Der Conditional Sum Addierer (CSA)

<u>Idee</u>: Nutze Parallelverarbeitung, um Tiefe zu reduzieren!



Zur Komplexität des CSA_n

- Ein CSA_n besteht aus 3 CSA_{n/2}
- Dabei steht x_h für die n/2 höchstwertigen Bits, x_l für die n/2 niederwertigen Bits der Eingabe
- Es gilt: $CSA_1 = FA$

Satz 11.2.

Der CSA_n hat die Tiefe O(log n).

Beweis:

```
n=1: depth(CSA<sub>1</sub>) = depth(FA) = 3
n>1: depth(CSA<sub>n</sub>) \leq depth(CSA<sub>n/2</sub>) + depth(MUX<sub>(n/2)+1</sub>)
        \leq depth(CSA<sub>n/2</sub>) + 3
        \leq depth(CSA<sub>n/4</sub>) + 3 + 3
        \leq depth(CSA<sub>n/8</sub>) + 3 + 3 + 3
        \leq depth(CSA<sub>n/(2</sub><sup>k</sup>)) + k · 3
        \leq 3 \cdot (k + 1) = 3 \log(n) + 3
```

BB TI I 11.1/29

Kosten eines CSA_n

$$C(CSA_1) = C(FA) = 5$$

 $C(CSA_n) = 3 \cdot C(CSA_{n/2}) + C(MUX_{(n/2)+1})$
 $= 3 \cdot C(CSA_{n/2}) + 3 \cdot n/2 + 4$

Zu lösen ist also ein Gleichungssystem der Form

$$f\!\left(n\right)\!=a\cdot f\!\left(\tfrac{n}{b}\right)\!+g\!\left(n\right)\quad und\quad f\!\left(1\right)\!=c$$

Betrachte dabei nur $n = b^k$

Kosten eines CSA_n (ff)

$$f(n) = g(n) + a \cdot \underbrace{f\left(\frac{n}{b}\right)}_{g\left(\frac{n}{b}\right) + a \cdot f\left(\frac{n}{b^2}\right)}$$

$$= g(n) + a \cdot g\left(\frac{n}{b}\right) + a^2 \cdot \underbrace{f\left(\frac{n}{b^2}\right)}_{g\left(\frac{n}{b^2}\right) + a \cdot f\left(\frac{n}{b^3}\right)}$$

$$= g(n) + a \cdot g\left(\frac{n}{b}\right) + a^2 \cdot g\left(\frac{n}{b^2}\right) + a^3 \cdot f\left(\frac{n}{b^3}\right)$$

$$\vdots$$

BB TII 11.1/31

Kosten eines CSA_n (ff)

$$\begin{split} f(n) &= g(n) + a \cdot g\big(\tfrac{n}{b}\big) + a^2 \cdot g\Big(\tfrac{n}{b^2}\big) + \ldots + a^{k-1} \cdot g\Big(\tfrac{n}{b^{k-1}}\big) + a^k \cdot \underbrace{f\Big(\tfrac{n}{b^k}\big)}_{f(1) = c} \\ &= \sum_{i=0}^{k-1} a^i \cdot g\Big(\tfrac{n}{b^i}\big) + a^k \cdot c \\ \\ &\stackrel{n=b^k}{=} \sum_{i=0}^{log_b} a^{i-1} \cdot g\Big(\tfrac{n}{b^i}\big) + a^{log_b} \cdot r \cdot c \end{split}$$

Lemma 11.1

Sei $f: \mathbf{N} \to \mathbf{N}$ mit

$$f\big(1\big)=c\text{,}\quad f\big(n\big)=a\cdot f\big(\textstyle\frac{n}{b}\big)+g\big(n\big)\qquad\forall\ n=b^k$$

Dann gilt für alle $n = b^k$

$$f\!\left(n\right)\!=a^{log_b\,n}\cdot c+\sum_{i=0}^{log_b\,n-1}\!\!\!a^i\cdot g\!\!\left(\!\tfrac{n}{b^i}\!\right)$$

Beweis durch Induktion über k.

BB TII 11.1/33

,

Beispiel zu Lemma 11.1

Sei
$$a = 3$$
, $b = 2$, $c = 5$,

$$g(n) = \frac{3}{2} \cdot n + 4$$

$$C\big(CSA_n\big) = 3^{log\,n} \cdot 5 + \sum_{i=0}^{log\,n-1} 3^i \cdot \left(\tfrac{3}{2} \cdot \tfrac{n}{2^i} + 4\right)$$

Beispiel (ff)

$$i) \qquad 3^{logn} = \left(2^{log3}\right)^{logn} = \left(2^{logn}\right)^{log3} = n^{log3}$$

$$ii) \qquad 4 \cdot \sum_{i=0}^{log \, n-1} 3^i = 4 \cdot \frac{3^{log \, n} - 1}{3-1} = 2 \cdot \left(3^{log \, n} - 1\right) \stackrel{i)}{=} \ 2n^{log \, 3} - 2$$

$$iii) \quad \ \ \frac{_3}{^2}n \cdot \sum_{_{i=0}}^{log\, n-1} \left(\!\tfrac{_3}{^2}\!\right)^{\!i} = \tfrac{_3}{^2}\, n \cdot \frac{\left(\!\tfrac{_3}{^2}\!\right)^{\!log\, n} - 1}{\left(\!\tfrac{_3}{^2}\!\right) - 1} = \tfrac{_3}{^2} \frac{n^{log\, 3} - n}{\tfrac{_1}{^2}} = 3n^{log\, 3} - 3n$$

BB TII 11.1/35

Beispiel (ff)

$$\Rightarrow C(CSA_n) = 5n^{\log 3} + 2n^{\log 3} - 2 + 3n^{\log 3} - 3n$$
$$= 10n^{\log 3} - 3n - 2$$

Bemerkung:

Man kann CSA_n in einfacher Weise modifizieren, so dass Tiefe = O(log n) und Kosten = O(n log n).

Der Carry Lookahead Addierer (CLA)

Gibt es Addierer mit linearen Kosten und logarithmischer Tiefe? → Carry Lookahead Addierer

Idee:

Man kann das Problem reduzieren auf die schnelle Berechnung des Übertragbits c_i . Sind c_i bekannt, so ergibt sich s_i durch a_i $\mathbf{\mathring{A}}$ b_i $\mathbf{\mathring{A}}$ c_{i-1} . Berechnung der c_i durch parallele Präfix-Berechnung!

BB TII 11.1/37

Parallele Präfix-Berechnung

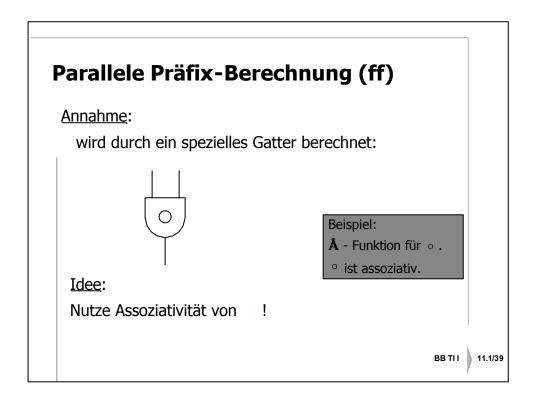
Sei M eine Menge,

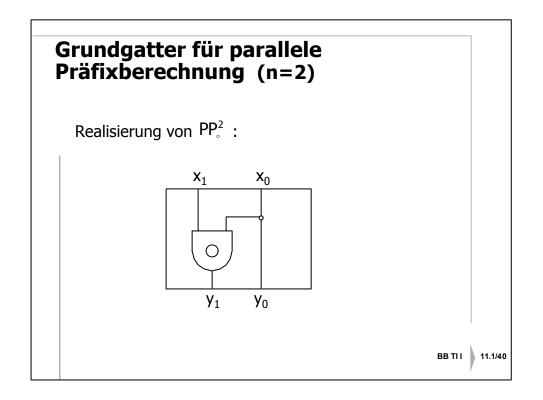
 \circ : M \times M \rightarrow M eine assoziative Abbildung.

Problem:

Realisiere die parallele Präfix-Funktion $\mbox{PP}^n_{\mbox{\tiny o}}$ durch

$$\begin{split} & PP_{\circ}^{n}: M^{n} \rightarrow M^{n}, \quad PP_{\circ}^{n}(x_{n-1},...,x_{0}) = (y_{n-1},...,y_{0}) \quad \text{ mit } \\ & y_{i} = x_{i} \circ ... \circ x_{0} \qquad \text{für } 0 \leq i \leq n. \end{split}$$





Funktionsweise der parallelen **Präfixberechnung**

Sei $n = 2^k$.

$$\begin{vmatrix} \mathbf{y}_{2i+1} = \mathbf{x}_{2i+1} \circ \mathbf{x}_{2i} \circ \dots \circ \mathbf{x}_{1} \circ \mathbf{x}_{0} \\ = \left(\mathbf{x}_{2i+1} \circ \mathbf{x}_{2i} \right) \circ \dots \circ \left(\mathbf{x}_{1} \circ \mathbf{x}_{0} \right) \qquad \left(\forall i \in \left\{ 0, \dots, \frac{n}{2} - 1 \right\} \right)$$

$$\begin{aligned} y_{2i} &= x_{2i} \circ x_{2i-1} \circ x_{2i-2} \circ ... \circ x_{1} \circ x_{0} \\ &= x_{2i} \circ (x_{2i-1} \circ x_{2i-2}) \circ ... \circ (x_{1} \circ x_{0}) \qquad (\forall i \in \{0, ..., \frac{n}{2} - 1\}) \end{aligned}$$

BB TII 11.1/41

Funktionsweise der parallelen Präfixberechnung (ff)

Mit $x_i' := x_{2i+1} \circ x_{2i}$ gilt also:

$$y_{2i+1} = x'_{i} \circ ... \circ x'_{0} := y'_{i} (\forall i \in \{0,...,\frac{n}{2}-1\})$$

$$y_{2i} = x_{2i} \circ x'_{i-1} \circ ... \circ x_0' \qquad \left(\forall i \in \left\{ 0, ..., \frac{n}{2} - 1 \right\} \right)$$

 $= x_{2i} \circ y_{2i-1} = x_{2i} \circ y'_{i-1}$

$$y_0 = x_0$$

Vereinfachte Beschreibung des **Vorgehens**

1.Schritt:

Fasse jeweils benachbarte Paare x_{2i+1} , x_{2i} zusammen:

$$x'_i = x_{2i+1} \circ x_{2i}$$

BB TII 11.1/43

Vereinfachte Beschreibung des Vorgehens (ff)

2.Schritt:

Benutze Schaltkreis $P_{n/2}$ mit Inputs x^{ι}_{i} , $0 \leq i \leq n/2-1$:

Vereinfachte Beschreibung des Vorgehens (ff)

3.Schritt:

Ergänze die fehlenden \mathbf{y}_{i} mit i gerade:

$$\begin{split} \boldsymbol{y}_{2i} &= \, \boldsymbol{x}_{2i} \circ \left(\boldsymbol{x}_{2i-1} \circ \ldots \circ \boldsymbol{x}_{0} \right) = \, \boldsymbol{x}_{2i} \circ \boldsymbol{y}_{2i-1} \qquad 1 \leq i \leq \frac{n}{2} - 1 \\ \boldsymbol{y}_{0} &= \, \boldsymbol{x}_{0} \end{split}$$

 \rightarrow Schaltkreis P_n zur Realisierung von

BB TI I 11.1/45

Lemma 11.2

 P_n hat Kosten $C(P_n) \le 2n$ und Tiefe depth $(P_n) \le 2 \log(n) - 1$ für $n=2^k$.

Beweis:

 $n=2^k$, also $C(P_n) \le 2^{k+1}$, $depth(P_n) \le 2k-1$

1) Kosten:

k=1:
$$C(P_0^2) = 1$$

$$k-1 \to k: \qquad C(P_0^{2^k}) \le C(P_0^{2^{k-1}}) + 2^{k-1} \cdot 2$$

$$\le 2^k + 2^k = 2^{k+1}$$

BB TII 11.1/47

Lemma 11.2 (ff)

Beweis:

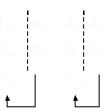
2) Tiefe:

k=1:
$$depth(P_0^2) = 1$$

k-1 \rightarrow k: $depth(P_0^{2^k}) = 2 + depth(P_0^{2^{k-1}})$
 $\leq 2 + (2(k-1)-1)$
 $= 2k-1$

Konstruktion des Carry Lookahead Addierers

Für $0 \le i < n$: $s_i = a_i \oplus b_i \oplus c_{i-1}$ \rightarrow Schnelle Berechnung der c_{i-1} genügt



BB TII 11.1/49

Schnelle Berechnung der c_{i-1}

 $\text{Betrachte Stellen i bis } j, \qquad i \leq j.$

Für Belegungen von $(a_j ... a_i)$ und $(b_j ... b_i)$ können genau 3 Fälle auftreten:

1) $c_j = 1$ unabhängig von c_{i-1} , d.h. für $c_{i-1} = 0$ und $c_{i-1} = 1$ Sprechweise:

Stellen i bis j generieren einen Übertrag.

Schnelle Berechnung der c_{i-1} (ff)

2) $c_j = 1 \hat{U} c_{i-1} = 1$

Sprechweise:

Stellen i bis j propagieren einen Übertrag.

3) $c_j = 0$ unabhängig von c_{i-1} , d.h. für $c_{i-1} = 0$ und $c_{i-1} = 1$

Sprechweise:

Stellen i bis j eliminieren einen Übertrag.

BB TII 11.1/51

Funktionsdefinition

Definiere die Funktion

 $g_{j,i}$, $p_{j,i} \colon \{0,1\}^{2n}$ @ $\{0,1\}$ für $0 \le i \le j < n$ $\quad \mbox{mit}$

 $g_{j,i}(a,b) = \begin{cases} 1: & \text{Stellen i bis j generieren Übertrag} \\ 0: & \text{sonst} \end{cases}$

 $p_{j,i}(a,b) = \begin{cases} 1: & \text{Stellen i bis j propagiere n } \ddot{\text{U}} \text{bertrag} \\ 0: & \text{sonst} \end{cases}$

Bemerkung:

 $g_{j,i}$ und $p_{j,i}$ hängen nur ab von $a_j,\,...,\,a_i,\,b_{j\prime}\,\,...,\,b_i!$

Eigenschaften von $g_{j,i}$, $p_{j,i}$

- $1) \;\; p_{i,i} = a_i \; \text{\ifmmode{A}{A} \end{A}} \; b_i \qquad \; \text{für} \; 0 \leq i < n \label{eq:pi_norm}$ $g_{i,i} \, = \, a_i \, \wedge b_i \qquad \quad \text{für } 0 \leq i < n$
- 2) Für i ≤ k < j: $g_{j,i} = g_{j,k+1} \lor \left(g_{k,i} \land p_{j,k+1}\right)$ $p_{j,i} = p_{k,i} \wedge p_{j,k+1}$
- 3) Für $0 \le i < n$: $\mathsf{C}_{\mathsf{i}} = \mathsf{g}_{\mathsf{i},0} + \mathsf{p}_{\mathsf{i},0} \cdot \mathsf{C}_{-1}$

BB TII 11.1/53

Eigenschaften von $g_{j,i}$, $p_{j,i}$ (ff)

Es genügt also, $g_{i,0}, \; p_{i,0} \; \; (0 \leq i < n)$ zu berechnen.

$$\begin{split} s_{_{i}} &= a_{_{i}} \oplus b_{_{i}} \oplus c_{_{i-1}} = p_{_{i,i}} \oplus \left(g_{_{i-1,0}} + p_{_{i-1,0}} \cdot c_{_{-1}}\right) \quad \text{ für } 1 \leq i < n \\ s_{_{n}} &= c_{_{n-1}} = g_{_{n-1,0}} + p_{_{n-1,0}} \cdot c_{_{-1}} \\ s_{_{0}} &= a_{_{0}} \oplus b_{_{0}} \oplus c_{_{-1}} = p_{_{0,0}} \oplus c_{_{-1}} \end{split}$$

Um parallele Präfixberechnung anwenden zu können, benötigt man einen geeigneten assoziativen Operator

Der Operator o

Wähle

$$\begin{aligned} \mathsf{M} &= \left(\boldsymbol{\mathsf{B}}_{2n}\right)^{\!2}, \quad \circ : \left(\boldsymbol{\mathsf{B}}_{2n}\right)^{\!2} \times \left(\boldsymbol{\mathsf{B}}_{2n}\right)^{\!2} \, \rightarrow \left(\boldsymbol{\mathsf{B}}_{2n}\right)^{\!2} \quad \text{mit} \\ \left(g_{2}, p_{2}\right) \circ \left(g_{1}, p_{1}\right) &= \left(g_{2} \vee \left(g_{1} \wedge p_{2}\right), p_{1} \wedge p_{2}\right) \end{aligned}$$

Damit läßt sich 2) schreiben als

$$\left(g_{j,i},p_{j,i}\right) = \left(g_{j,k+1},p_{j,k+1}\right) \circ \left(g_{k,i},p_{k,i}\right)$$

Es gilt:
$$C(\circ) = 3$$

 $depth(\circ) = 2$

BB TII 11.1/55

Basiszelle der Operation o 92 p2 91 p1 BB TII 11.1/56

Lemma 11.3

Die Operation o ist assoziativ.

Beweis:

Nachrechnen unter Verwendung von Gesetzen der Booleschen Algebra.

BB TII 11.1/57

Bedeutung für die parallele Präfixberechnung

Aus 2) und der Assoziativität folgt:

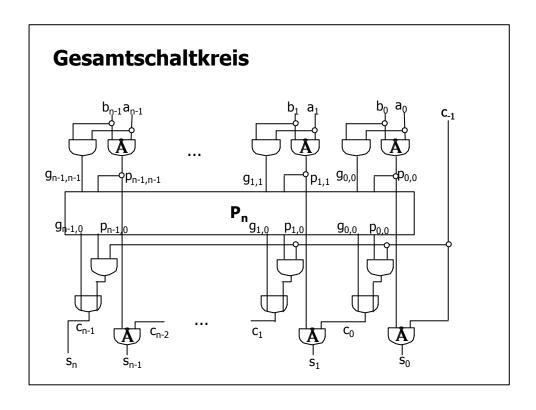
$$\left(g_{i,0},p_{i,0}\right) = \left(g_{i,i},p_{i,i}\right) \circ \dots \circ \left(g_{1,1},p_{1,1}\right) \circ \left(g_{0,0},p_{0,0}\right)$$

Wende nun parallele Präfixberechnung an.

lassen sich aus g_{i,i}, p_{i,i}

bestimmen mit Kosten $\leq 2n \cdot C(\circ) = 6n$

und Tiefe $(2 \log(n) - 1) \cdot depth(\circ) = 4 \log(n) - 2$



Gesamtkosten

Kosten: $C(CLA_n) \le 6n + 2n + 3n = 11n$

Tiefe: depth(CLA_n) \leq 4 log(n) - 2 + 1 + 3 = 4 log(n) + 2

Addition von Zweierkomplementzahlen

Wiederholung:

Formale Darstellung:

$$[a_n a_{n-1} ... a_0] + [b_n b_{n-1} ... b_0] =$$

$$(-a_n 2^n) + (-b_n 2^n) + \sum_{i=0}^{n-1} a_i 2^i + \sum_{i=0}^{n-1} b_i 2^i$$

BB TII 11.1/61

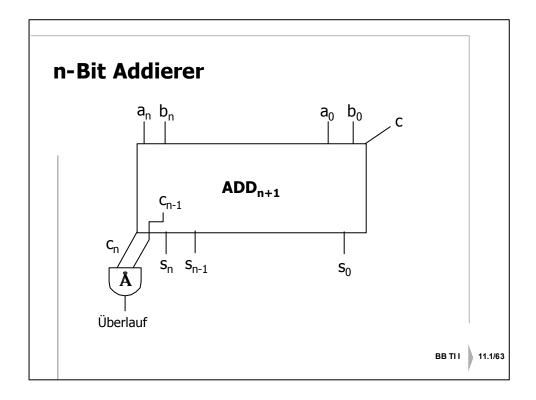
Behauptung

Zur Addition von (n+1)-Bit-Zweierkomplementzahlen kann man (n+1)-Bit-Binäraddierer benutzen.

Der Test, ob das Ergebnis durch eine (n+1)-Bit-Zweierkomplementzahl darstellbar ist, d. h. ob das Ergebnis aus

$$R_n = \{-2^n, ..., 2^{n} \}$$
 ist,

läßt sich zurückführen auf den Test $c_n = c_{n-1}$.



Satz 11.3

Seien $a,b \in \mathbf{B}^{n+1}$, $c_{-1} \in \{0,1\}$ und $s \in \{0,1\}^{n+1}$, so dass $< c_n, s> = < a> + < b> + c_{-1}$.

Dann gilt:

i)
$$[a] + [b] + c_{-1} \in R_n \Leftrightarrow c_n = c_{n-1}$$

ii)
$$[a] + [b] + c_{-1} \in R_n \Rightarrow [a] + [b] + c_{-1} = [s]$$

Beweis durch Fallunterscheidung [a],[b] beide positiv, beide negativ bzw. O.E. [a] negativ, [b] positiv

und Nachrechnen ...

Bemerkung:

Steht c_{n-1} nicht zur Verfügung (z.B. CSA), so kann man den Überlauftest

$$\big[a\big] + \big[b\big] + c_{\scriptscriptstyle{-1}} \not\in \, R_{\scriptscriptstyle{n}} \, \Longleftrightarrow a_{\scriptscriptstyle{n}} \, = b_{\scriptscriptstyle{n}} \, \neq s_{\scriptscriptstyle{n}}$$

verwenden.