Kapitel 11

Arithmetische Schaltkreise

11.1 Addierer

11.2 Subtrahierer
Multiplizierer
ALU

Wiederholung:

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

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

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

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

Addierer

Gegeben: 2 positive Binärzahlen $\langle a \rangle = \langle a_{n-1} \ldots a_0 \rangle$,

$\langle b \rangle = \langle b_{n-1} \ldots b_0 \rangle$,

Eingangsübertrag $c \in \{0,1\}$

Gesucht: Schaltkreis, der Binärdarstellung s von

$\langle a \rangle + \langle b \rangle + c$ berechnet

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

genügen n+1 Ausgänge des Schaltkreises.

Definition 11.1

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

$+_n : \mathbf{B}^{2n+1} \rightarrow \mathbf{B}^{2n+1}$,

$(a_{n-1}, \ldots, a_0, b_{n-1}, \ldots, b_0, c) \rightarrow (s_n, \ldots, s_0)$ mit

$\langle s \rangle = \langle s_n \ldots s_0 \rangle = \langle a_{n-1} \ldots a_0 \rangle + \langle b_{n-1} \ldots b_0 \rangle + c$

Beispiel

Addieren nach der Schulmethode:

```
    1 0 1 1
+   0 1 1 0
+ 1 1 1 0 0
-----------
  1 0 0 0 1
```

Eingangsübertrag

Der Halbaddierer (HA)

Der **Halbaddierer** dient zur Addition zweier 1-Bit-Zahlen ohne Eingangsübertrag.

Er berechnet die Funktion:

$ha : \mathbf{B}^2 \rightarrow \mathbf{B}^2$

mit $ha(a_0, b_0) = (s_1, s_0)$

mit $2s_1 + s_0 = a_0 + b_0$

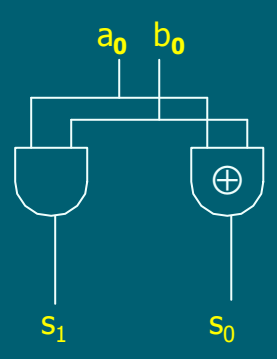
Funktionstabelle des HA

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:

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

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:

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

mit $fa(a_0, b_0, c) = (s_1, s_0)$

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

Funktionstabelle des FA

a_0	b_0	c	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:

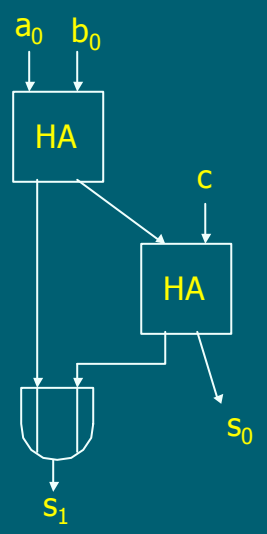
$$fa_0 = a_0 \oplus b_0 \oplus c = ha_0(c, ha_0(a_0, b_0))$$

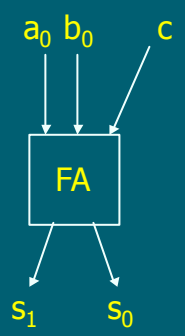
$$fa_1 = a_0 \wedge b_0 \vee c \wedge (a_0 \oplus b_0)$$
$$= ha_1(a_0, b_0) + ha_1(c, ha_0(a_0, b_0))$$

Kosten und Tiefe eines FA:

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

Schaltkreis eines Volladdierers





Realisieren der Schulmethode: Carry Ripple Addierer (CR)

Hierarchisches Vorgehen:

(induktive Definition)

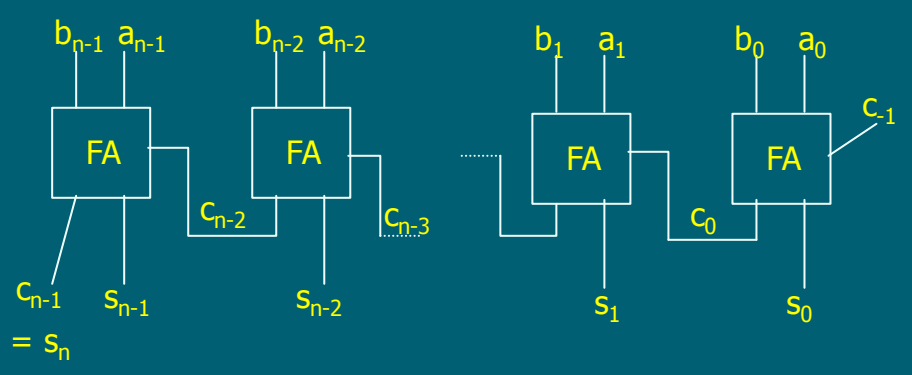
Für n=1: $CR_1 = FA$

Für n>1: Schaltkreis CR_n wie folgt definiert

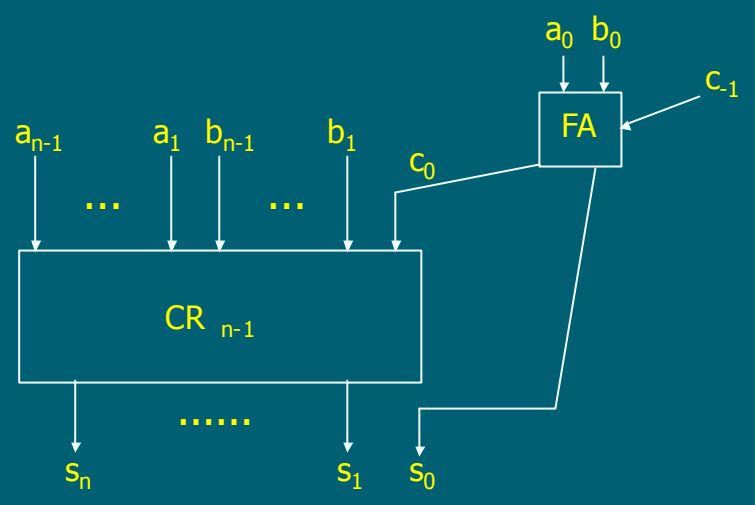
Bezeichnung:

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

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 → n:

  Eingabe an CR_n: $(a_{n-1}, \ldots, a_0, b_{n-1}, \ldots b_0, c_{-1})$

  Zeige für Ausgabe $(s_n, \ldots, s_0)$ von CR_n

  $\langle s \rangle = \langle s_n \ldots s_0 \rangle = \langle a_{n-1} \ldots a_0 \rangle + \langle b_{n-1} \ldots b_0 \rangle + c_{-1}$

Satz 11.1 (ff)

Nach I.V.:

a) $<c_0, s_0> = a_0 + b_0 + c_{-1}$ (FA)

b) Für CR_{n-1}: $<s_n \ldots s_1> = <a_{n-1} \ldots a_1> + <b_{n-1} \ldots b_1> + c_0$

Insgesamt:

$<s_n \ldots s_0> = 2 \cdot <s_n \ldots s_1> + s_0$

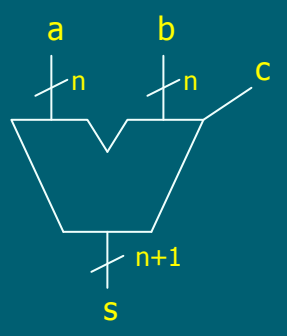
I.V.b) $= 2 \cdot (<a_{n-1} \ldots a_1> + <b_{n-1} \ldots b_1> + c_0) + s_0$

I.V.a) $= 2 \cdot <a_{n-1} \ldots a_1> + a_0 + 2 \cdot <b_{n-1} \ldots b_1> + b_0 + c_{-1}$

$= <a> + <b> + c_{-1}$

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

Definition 11.2

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

$inc_n : \mathbf{B}^{n+1} \rightarrow \mathbf{B}^{n+1}$,

$(a_{n-1}, \ldots, a_0, c) \rightarrow (s_n, \ldots, s_0)$ mit

$\langle s_n \ldots s_0 \rangle = \langle a \rangle + c$

Inkrementer

Ein Inkrementer ist ein Addierer mit

$b_i = 0 \quad \forall i$

➔ Ersetze in CR_n die FA durch HA.

Kosten und Tiefe:

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

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

Definition 11.3

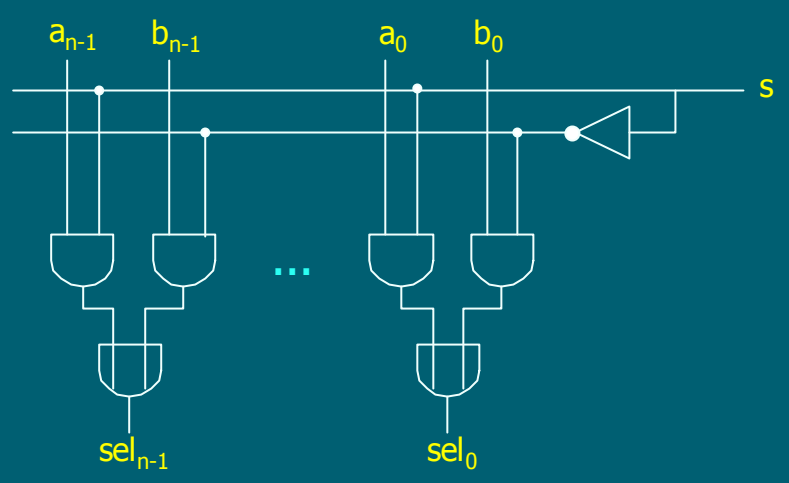
Ein **n-Bit-Multiplexer** (MUX_n) ist ein Schaltkreis, der die folgende Funktion berechnet:

$sel_n : \mathbf{B}^{2n+1} \rightarrow \mathbf{B}^n$ mit

$$sel_n(a_{n-1},\ldots,a_0,b_{n-1},\ldots,b_0,s) = \begin{cases} (a_{n-1}\ldots a_0), \text{falls} & s = 1 \\ (b_{n-1}\ldots b_0), \text{falls} & s = 0 \end{cases}$$

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

Schaltbild zum n-Bit Multiplexer

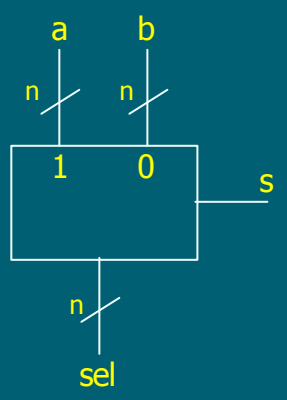


Komplexität eines n-Bit MUX

Kosten und Tiefe:

$C(MUX_n) = 3n + 1$

$depth(MUX_n) = 3$



Rückkehr zum Addierer

Gibt es billigere Addierer als CR_n ?

Untere Schranken:

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

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

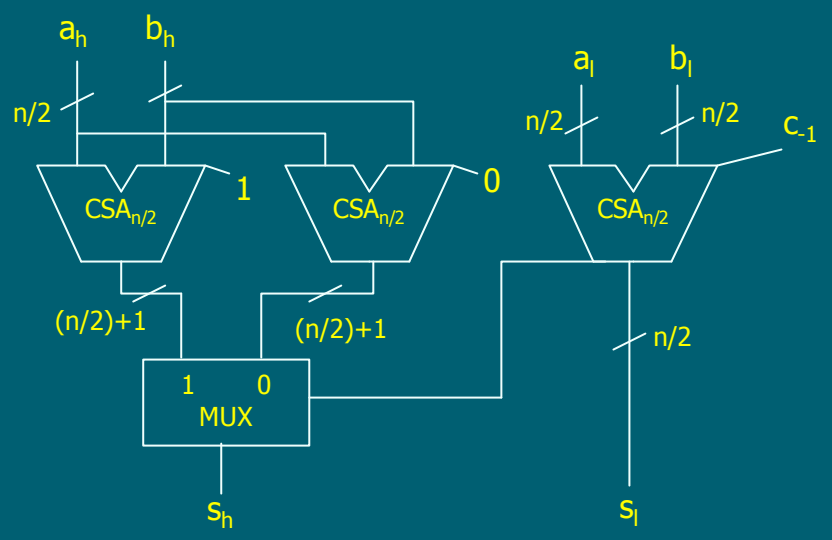
Im folgenden sei $n = 2^k$.

Der Conditional Sum Addierer (CSA)

Idee: Nutze Parallelverarbeitung, um Tiefe zu reduzieren!

$$
\begin{array}{lcccccccc}
 & a_{n-1} & \dots & a_{\frac{n}{2}} & a_{\frac{n}{2}-1} & \dots & a_0 \\
+ & b_{n-1} & \dots & b_{\frac{n}{2}} & b_{\frac{n}{2}-1} & \dots & b_0 \\
+ & & & c_{\frac{n}{2}-1} & & & c_{-1} \\
\hline
s_n & s_{n-1} & \dots & s_{\frac{n}{2}} & s_{\frac{n}{2}-1} & \dots & s_0
\end{array}
$$

Schaltbild des CSA_n



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: $depth(CSA_1) = depth(FA) = 3$

n>1: $depth(CSA_n) \leq depth(CSA_{n/2}) + depth(MUX_{(n/2)+1})$

$\leq depth(CSA_{n/2}) + 3$

$\leq depth(CSA_{n/4}) + 3 + 3$

$\leq depth(CSA_{n/8}) + 3 + 3 + 3$

.....

$\leq depth(CSA_{n/(2^k)}) + k \cdot 3$

$\leq 3 \cdot (k + 1) = 3 \log(n) + 3$

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

$f(n) = a \cdot f(\frac{n}{b}) + g(n)$ und $f(1) = c$

Betrachte dabei nur $n = b^k$

Kosten eines CSA_n (ff)

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

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

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

$$\vdots$$

Kosten eines CSA_n (ff)

$$f(n) = g(n) + a \cdot g\left(\frac{n}{b}\right) + a^2 \cdot g\left(\frac{n}{b^2}\right) + \ldots + a^{k-1} \cdot g\left(\frac{n}{b^{k-1}}\right) + a^k \cdot \underbrace{f\left(\frac{n}{b^k}\right)}_{f(1)=c}$$

$$= \sum_{i=0}^{k-1} a^i \cdot g\left(\frac{n}{b^i}\right) + a^k \cdot c$$

$$\overset{n=b^k}{=} \sum_{i=0}^{\log_b n-1} a^i \cdot g\left(\frac{n}{b^i}\right) + a^{\log_b n} \cdot c$$

Lemma 11.1

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

$$f(1) = c, \quad f(n) = a \cdot f\left(\tfrac{n}{b}\right) + g(n) \quad \forall\, n = b^k$$

Dann gilt für alle $n = b^k$

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

Beweis durch Induktion über k.

Beispiel zu Lemma 11.1

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

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

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

Beispiel (ff)

i) $3^{\log n} = \left(2^{\log 3}\right)^{\log n} = \left(2^{\log n}\right)^{\log 3} = n^{\log 3}$

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

iii) $\frac{3}{2} n \cdot \sum_{i=0}^{\log n - 1} \left(\frac{3}{2}\right)^i = \frac{3}{2} n \cdot \frac{\left(\frac{3}{2}\right)^{\log n} - 1}{\left(\frac{3}{2}\right) - 1} = \frac{3}{2} \frac{n^{\log 3} - n}{\frac{1}{2}} = 3n^{\log 3} - 3n$

Beispiel (ff)

$$\overset{i),ii),iii)}{\Rightarrow} \quad C(CSA_n) = 5n^{\log 3} + 2n^{\log 3} - 2 + 3n^{\log 3} - 3n$$
$$= 10n^{\log 3} - 3n - 2$$

<u>Bemerkung</u>:

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 Å b_i Å c_{i-1}.

Berechnung der c_i durch parallele Präfix-Berechnung!

Parallele Präfix-Berechnung

Sei M eine Menge,

$\circ : M \times M \to M$ eine assoziative Abbildung.

Problem:

Realisiere die parallele Präfix-Funktion $PP_\circ^n$ durch

$PP_\circ^n : M^n \to M^n$, $PP_\circ^n(x_{n-1},\ldots,x_0) = (y_{n-1},\ldots,y_0)$ mit

$y_i = x_i \circ \ldots \circ x_0$ für $0 \le i \le n$.

Parallele Präfix-Berechnung (ff)

Annahme:

- $\circ$ wird durch ein spezielles Gatter berechnet:

Beispiel:

Å - Funktion für $\circ$.

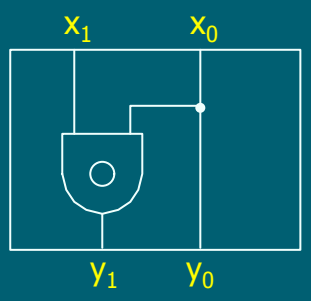
$\circ$ ist assoziativ.

Idee:

Nutze Assoziativität von $\circ$!

Grundgatter für parallele Präfixberechnung (n=2)

Realisierung von $PP_{\circ}^{2}$:



Funktionsweise der parallelen Präfixberechnung

Sei $n = 2^k$.

$$y_{2i+1} = x_{2i+1} \circ x_{2i} \circ \ldots \circ x_1 \circ x_0$$
$$= (x_{2i+1} \circ x_{2i}) \circ \ldots \circ (x_1 \circ x_0) \qquad (\forall i \in \{0,\ldots,\tfrac{n}{2}-1\})$$

$$y_{2i} = x_{2i} \circ x_{2i-1} \circ x_{2i-2} \circ \ldots \circ x_1 \circ x_0$$
$$= x_{2i} \circ (x_{2i-1} \circ x_{2i-2}) \circ \ldots \circ (x_1 \circ x_0) \qquad (\forall i \in \{0,\ldots,\tfrac{n}{2}-1\})$$

Funktionsweise der parallelen Präfixberechnung (ff)

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

$$y_{2i+1} = x'_i \circ \ldots \circ x'_0 := y'_i \qquad (\forall i \in \{0,\ldots,\tfrac{n}{2}-1\})$$
$$y_{2i} = x_{2i} \circ x'_{i-1} \circ \ldots \circ x_0' \qquad (\forall i \in \{0,\ldots,\tfrac{n}{2}-1\})$$
$$= x_{2i} \circ y_{2i-1} = x_{2i} \circ y'_{i-1}$$
$$\vdots$$
$$y_0 = x_0$$

Vereinfachte Beschreibung des Vorgehens

1.Schritt:

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

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

Vereinfachte Beschreibung des Vorgehens (ff)

2.Schritt:

Benutze Schaltkreis $P_{n/2}$ mit Inputs x'_i , $0 \le i \le n/2 - 1$:

$$
\begin{aligned}
y'_0 &= x'_0 &&= x_1 \circ x_0 &&= y_1 \\
y'_1 &= x'_1 \circ x'_0 &&= (x_3 \circ x_2) \circ (x_1 \circ x_0) &&= y_3 \\
y'_2 &= x'_2 \circ x'_1 \circ x'_0 &&= (x_5 \circ x_4) \circ (x_3 \circ x_2) \circ (x_1 \circ x_0) &&= y_5 \\
&\vdots && \vdots && \vdots \\
y'_{\frac{n}{2}-1} &= x'_{\frac{n}{2}-1} \circ \ldots \circ x'_0 &&= (x_{n-1} \circ x_{n-2}) \circ \ldots \circ (x_1 \circ x_0) &&= y_{n-1}
\end{aligned}
$$

Vereinfachte Beschreibung des Vorgehens (ff)

3.Schritt:

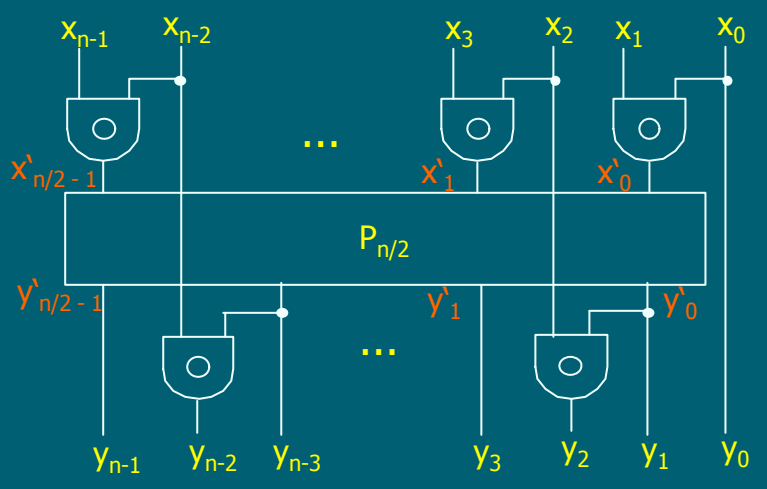
Ergänze die fehlenden y_i mit i gerade:

$$y_{2i} = x_{2i} \circ (x_{2i-1} \circ \ldots \circ x_0) = x_{2i} \circ y_{2i-1} \qquad 1 \le i \le \tfrac{n}{2} - 1$$

$$y_0 = x_0$$

→ Schaltkreis P_n zur Realisierung von $PP_\circ^n$

Schaltkreis P_n



Lemma 11.2

P_n hat Kosten $C(P_n) \leq 2n$ und

Tiefe depth $(P_n) \leq 2 \log(n) - 1$ für $n=2^k$.

Beweis:

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

1) Kosten:

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

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

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

Lemma 11.2 (ff)

Beweis:

2) Tiefe:

k=1: $\text{depth}(P_0^2) = 1$

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

$\leq 2 + (2(k-1) - 1)$

$= 2k - 1$

Konstruktion des Carry Lookahead Addierers

Für $0 \le i < n$: $s_i = a_i \oplus b_i \oplus c_{i-1}$

→ Schnelle Berechnung der c_{i-1} genügt

$$
\begin{array}{rcccc}
 & a_{n-1} \ldots a_{j+1} & a_j \ldots a_i & a_{i-1} \ldots a_0 \\
+ & b_{n-1} \ldots b_{j+1} & b_j \ldots b_i & b_{i-1} \ldots b_0 \\
+ & c_j & c_{i-1} & c
\end{array}
$$

Schnelle Berechnung der c_{i-1}

Betrachte Stellen i bis j, $i \le j$.

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

1) $c_j = 1$ unabhängig von c_{i-1}, d.h. für $c_{i-1} = 0$ <u>und</u> $c_{i-1} = 1$

   <u>Sprechweise</u>:

   Stellen i bis j generieren einen Übertrag.

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

2) $c_j = 1$ Û $c_{i-1} = 1$

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.

Funktionsdefinition

Definiere die Funktion

$g_{j,i}$, $p_{j,i}$: $\{0,1\}^{2n}$ ® $\{0,1\}$ für $0 \le i \le j < n$ mit

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

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

Bemerkung:

$g_{j,i}$ und $p_{j,i}$ hängen nur ab von $a_j, \ldots, a_i, b_j, \ldots, b_i$!

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

1) $p_{i,i} = a_i \text{ Å } b_i$ für $0 \le i < n$

   $g_{i,i} = a_i \wedge b_i$ für $0 \le i < n$

2) Für $i \le k < j$:

   $g_{j,i} = g_{j,k+1} \vee (g_{k,i} \wedge p_{j,k+1})$

   $p_{j,i} = p_{k,i} \wedge p_{j,k+1}$

3) Für $0 \le i < n$:

   $c_i = g_{i,0} + p_{i,0} \cdot c_{-1}$

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

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

$s_i = a_i \oplus b_i \oplus c_{i-1} = p_{i,i} \oplus (g_{i-1,0} + p_{i-1,0} \cdot c_{-1})$ für $1 \le i < n$

$s_n = c_{n-1} = g_{n-1,0} + p_{n-1,0} \cdot c_{-1}$

$s_0 = a_0 \oplus b_0 \oplus c_{-1} = p_{0,0} \oplus c_{-1}$

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

Der Operator $\circ$

Wähle

$M = (\mathbf{B}_{2n})^2, \quad \circ : (\mathbf{B}_{2n})^2 \times (\mathbf{B}_{2n})^2 \rightarrow (\mathbf{B}_{2n})^2$ mit

$(g_2, p_2) \circ (g_1, p_1) = (g_2 \vee (g_1 \wedge p_2), p_1 \wedge p_2)$

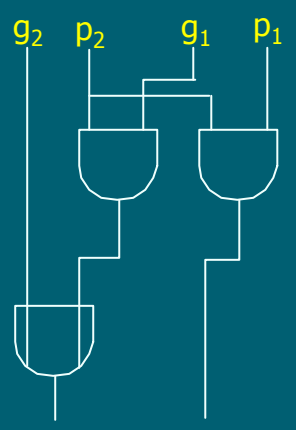
Damit läßt sich 2) schreiben als

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

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

$\text{depth}(\circ) = 2$

Basiszelle der Operation $\circ$



Lemma 11.3

Die Operation $\circ$ ist assoziativ.

Beweis:

Nachrechnen unter Verwendung von Gesetzen der Booleschen Algebra.

Bedeutung für die parallele Präfixberechnung

Aus 2) und der Assoziativität folgt:

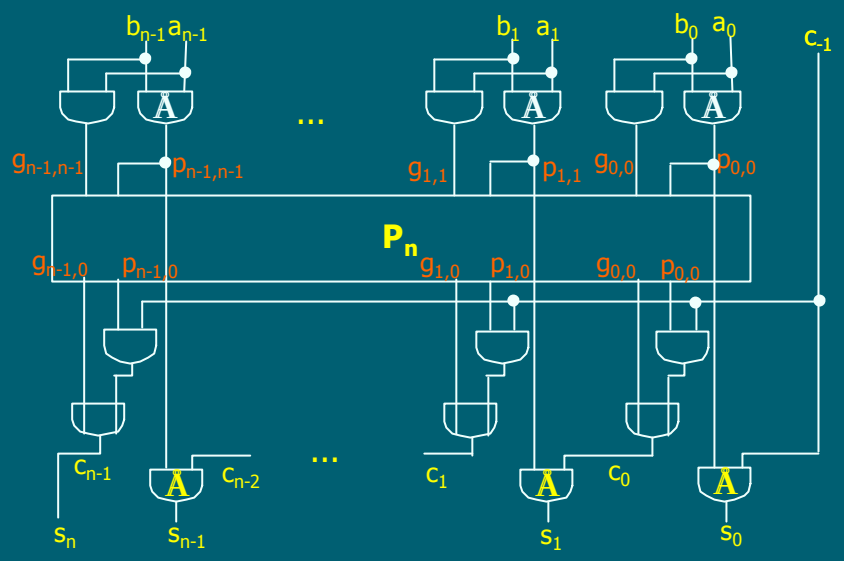
$$(g_{i,0}, p_{i,0}) = (g_{i,i}, p_{i,i}) \circ \ldots \circ (g_{1,1}, p_{1,1}) \circ (g_{0,0}, p_{0,0})$$

Wende nun parallele Präfixberechnung an.

$(g_{i,0}, p_{i,0})$ $(0 \le i < n)$ lassen sich aus $g_{i,i}$, $p_{i,i}$ bestimmen mit Kosten $\le 2n \cdot C(\circ) = 6n$

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

Gesamtschaltkreis



Gesamtkosten

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

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

$= 4 \log(n) + 2$

Addition von Zweierkomplementzahlen

Wiederholung:

Formale Darstellung:

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

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

Behauptung

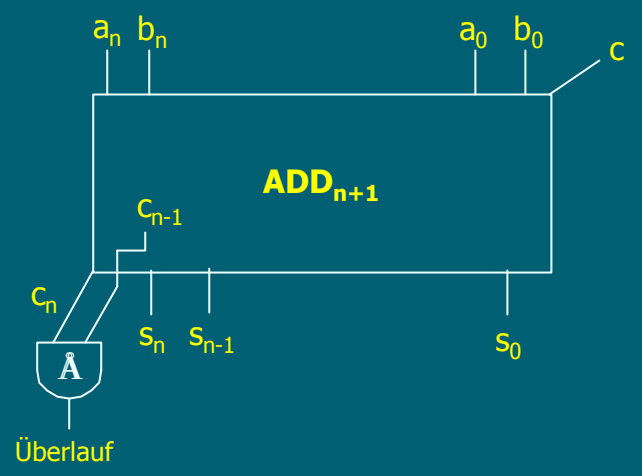
Zur Addition von (n+1)-Bit-Zweierkomplementzahlen kann man (n+1)-Bit-Binäraddierer benutzen.

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

$R_n = \{-2^n, \ldots, 2^n-1\}$ ist,

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

n-Bit Addierer



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 $\langle c_n,s\rangle = \langle a\rangle + \langle b\rangle + c_{-1}$.

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

$$[a]+[b]+c_{-1} \notin R_n \Leftrightarrow a_n = b_n \neq s_n$$

verwenden.