Technische Informatik 1

Prof. Dr. Rolf Drechsler

Christina Plump

Überblick

Teil 1: Der Rechneraufbau (Kapitel 2-5)

- Rechner im Überblick
- Pipelining
- Speicher
- Parallelverarbeitung

Teil 2: Der Funktionalitätsaufbau (Kapitel 6-12)

- Kodierung
- Grundbegriffe, Boolesche Funktionen
- Darstellungsmöglichkeiten
- **Schaltkreise, Synthese, spezielle Schaltkreise**
  - **Addierer, Inkrementer, Multiplexer**
  - Subtrahierer, Multiplizierer, ALU

Kapitel 11: Arithmetische Schaltkreise

Addierer, Multiplexer und Inkrementer

Subtrahierer, Multiplizierer und ALU

Lernziele

- Unterschiedliche Schaltkreise für die Addition kennen und verstehen
  - Halbaddierer, Volladdierer
  - Carry-Ripple Addierer
  - Conditional-Sum Addierer
  - Carry-Lookahead Addierer
- Vor- und Nachteile insbesondere in Bezug auf Kosten und Laufzeit kennen und verstehen
- Schaltkreise für Multiplexer sowie Inkrementer kennen und verstehen
- Konzept der Schaltkreiskonstruktion kennen, verstehen und anwenden können

Wiederholung

Definition (vorzeichenbehaftete Festkommazahl):
Eine Festkommazahl $d = d_n d_{n-1} \dots d_0 \dots d_{-k}$ wird vorzeichenbehaftet wie folgt interpretiert:

- Betrag und Vorzeichen: $[d]_{BV} = [d_n d_{n-1} \dots d_0 \dots d_{-k}]_{BV} := (-1)^{d_n} \sum_{i=-k}^{n-1} d_i \cdot 2^i$
- Einerkomplement-Darstellung: $[d]_1 = [d_n d_{n-1} \dots d_0 \dots d_{-k}]_1 := \sum_{i=-k}^{n-1} d_i \cdot 2^i - d_n(2^n - 2^{-k})$
- **Zweierkomplement-Darstellung:** $[\boldsymbol{d}]_2 = [\boldsymbol{d_n d_{n-1} \dots d_0 \dots d_{-k}}]_2 := \sum_{i=-k}^{n-1} \boldsymbol{d_i \cdot 2^i - d_n 2^n}$

Ist $d_n = 1$, ergibt sich $[d]$ zu einem negativen Wert; ist $d_n = 0$, ergibt sich $[d]$ zu $< d >$ und entspricht der vorzeichenlosen Interpretation einer Festkommazahl.

Lemma:
Sei $\overline{a}$ die Festkommazahl die aus a durch Komplementieren aller Bits hervorgeht. Dann gilt $[\overline{a}]_2 + 2^{-k} = -[a]_2$

Addierer im Allgemeinen

Gegeben:

- Zwei positive Binärzahlen $a = a_{n-1} \dots a_0$, $b = b_{n-1} \dots b_0$
- ein Eingangsübertrag $c \in \{0,1\}$

Gesucht:

Schaltkreis, der Binärdarstellung s mit $< s > = < a > + < b > + c$ berechnet.

Bemerkung:

Wegen $< a > + < b > + c \leq 2 \cdot (2^n - 1) + 1 = 2^{n+1} - 1$, genügen $n + 1$ Bits für die Darstellung von s, d.h. als Ausgang des Schaltkreises.

Addierer im Allgemeinen (2)

Definition (n-Bit Addierer):
Ein n-Bit Addierer ist ein Schaltkreis, der die folgende Boolesche Funktion $+_n$ berechnet:

$$+_n : \mathbb{B}^{2n+1} \to \mathbb{B}^{n+1}$$
$$(a_{n-1}, \dots, a_0, b_{n-1}, \dots, b_0, c) \mapsto (s_n, \dots, s_0),$$

sodass $< s >=< a > + < b > + c$ gilt.

Beispiel:
$a = 1011, b = 0110, c = 0$

$$\begin{array}{r} 1011 \\ +\ 0110 \\ \hline \end{array} \qquad \begin{array}{r} 11 \\ +\ 6 \\ \hline \end{array}$$

Addierer im Allgemeinen (2)

Definition (n-Bit Addierer):
Ein n-Bit Addierer ist ein Schaltkreis, der die folgende Boolesche Funktion $+_n$ berechnet:

$$+_n : \mathbb{B}^{2n+1} \rightarrow \mathbb{B}^{n+1}$$
$$(a_{n-1}, \ldots, a_0, b_{n-1}, \ldots, b_0, c) \mapsto (s_n, \ldots, s_0),$$

sodass $< s >=< a > + < b > + c$ gilt.

Beispiel:
$a = 1011, b = 0110, c = 0$

$$\begin{array}{r} 1011 \\ +\ 0110 \\ \hline 1 \end{array} \qquad \begin{array}{r} {\scriptstyle 0\,0} \\ 11 \\ +\ 6 \\ \hline \end{array}$$

Addierer im Allgemeinen (2)

Definition (n-Bit Addierer):
Ein n-Bit Addierer ist ein Schaltkreis, der die folgende Boolesche Funktion $+_n$ berechnet:

$$+_n : \mathbb{B}^{2n+1} \rightarrow \mathbb{B}^{n+1}$$
$$(a_{n-1}, \ldots, a_0, b_{n-1}, \ldots, b_0, c) \mapsto (s_n, \ldots, s_0),$$

sodass $< s >=< a > + < b > + c$ gilt.

Beispiel:
$a = 1011, b = 0110, c = 0$

$$\begin{array}{r} 1011 \\ +\ 0110 \\ \hline 01 \end{array} \qquad \begin{array}{r} 11 \\ +\ 6 \\ \hline \end{array}$$

Addierer im Allgemeinen (2)

Definition (n-Bit Addierer):
Ein n-Bit Addierer ist ein Schaltkreis, der die folgende Boolesche Funktion $+_n$ berechnet:

$$+_n : \mathbb{B}^{2n+1} \rightarrow \mathbb{B}^{n+1}$$
$$(a_{n-1}, \ldots, a_0, b_{n-1}, \ldots, b_0, c) \mapsto (s_n, \ldots, s_0),$$

sodass $< s >=< a > + < b > + c$ gilt.

Beispiel:
$a = 1011, b = 0110, c = 0$

$$\begin{array}{r} 1011 \\ +\ 0110 \\ \hline 001 \end{array} \qquad \begin{array}{r} 11 \\ +\ 6 \\ \hline \end{array}$$

Addierer im Allgemeinen (2)

Definition (n-Bit Addierer):
Ein n-Bit Addierer ist ein Schaltkreis, der die folgende Boolesche Funktion $+_n$ berechnet:

$$+_n : \mathbb{B}^{2n+1} \rightarrow \mathbb{B}^{n+1}$$
$$(a_{n-1}, \ldots, a_0, b_{n-1}, \ldots, b_0, c) \mapsto (s_n, \ldots, s_0),$$

sodass $< s >=< a > + < b > + c$ gilt.

Beispiel:
$a = 1011, b = 0110, c = 0$

$$\begin{array}{r} 1011 \\ +\ 0110 \\ \hline 0001 \end{array} \qquad \begin{array}{r} 11 \\ +\ 6 \\ \hline \end{array}$$

$$\begin{array}{r} 1\,1\,1\,0\,0 \end{array}$$

Addierer im Allgemeinen (2)

Definition (n-Bit Addierer):
Ein n-Bit Addierer ist ein Schaltkreis, der die folgende Boolesche Funktion $+_n$ berechnet:

$$+_n : \mathbb{B}^{2n+1} \to \mathbb{B}^{n+1}$$
$$(a_{n-1}, \dots, a_0, b_{n-1}, \dots, b_0, c) \mapsto (s_n, \dots, s_0),$$

sodass $< s >=< a > + < b > + c$ gilt.

Beispiel:
$a = 1011, b = 0110, c = 0$

$$\begin{array}{r} 1011 \\ +\ 0110 \\ \hline 10001 \end{array} \qquad \begin{array}{r} 11 \\ +\ 6 \\ \hline 17 \end{array}$$

Der Halbaddierer (*HA*)

Definition (Halbaddierer):
Der Halbaddierer HA führt die Addition zweier 1-Bit-Zahlen ohne Eingangsübertrag durch. Die zugehörige Boolesche Funktion ist also beschrieben durch:

$$ha : \mathbb{B}^2 \to \mathbb{B}^2$$
$$(a_0, b_0) \mapsto (s_1, s_0)$$

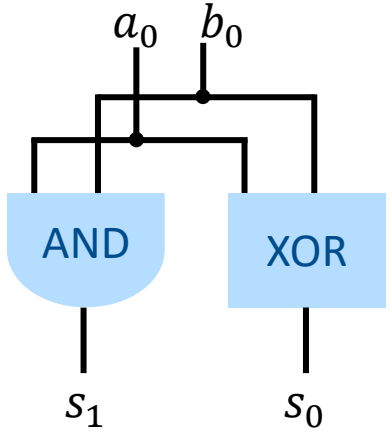
sodass $< s > = 2s_1 + s_0 = a_0 + b_0 = < a > + < b >$ gilt.

Funktionstabelle:

$< a >$	$< b >$	a_0	b_0	s_1	s_0	$< s >$
0	0	0	0	0	0	0
0	1	0	1	0	1	1
1	0	1	0	0	1	1
1	1	1	1	1	0	2

$$s_1 = a_0 \cdot b_0$$

$$s_0 = a_0 \oplus b_0$$



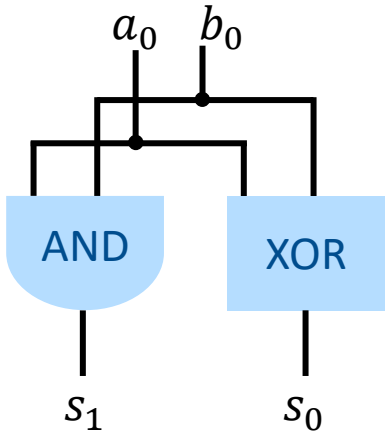
Der Halbaddierer – Kosten und Tiefe

Lemma (Kosten & Tiefe eines Halbaddierers):
Die Kosten eines Halbaddieres betragen $C(HA) = 2$, während die Tiefe $depth(HA) = 1$ entspricht.

Beweis:

Kosten: Der Schaltkreis enthält zwei Modulknoten: AND, XOR

Tiefe: Der längste Weg vom Eingangs- zum Ausgangssignal passiert einen Modulknoten (entweder AND oder XOR).



Der Volladdierer (*FA*)

Definition (Volladdierer):
Der Volladdierer FA führt die Addition zweier 1-Bit-Zahlen mit Eingangsübertrag durch. Die zugehörige Boolesche Funktion ist also beschrieben durch:

$$fa : \mathbb{B}^3 \to \mathbb{B}^2$$
$$(a_0, b_0, c) \mapsto (s_1, s_0)$$

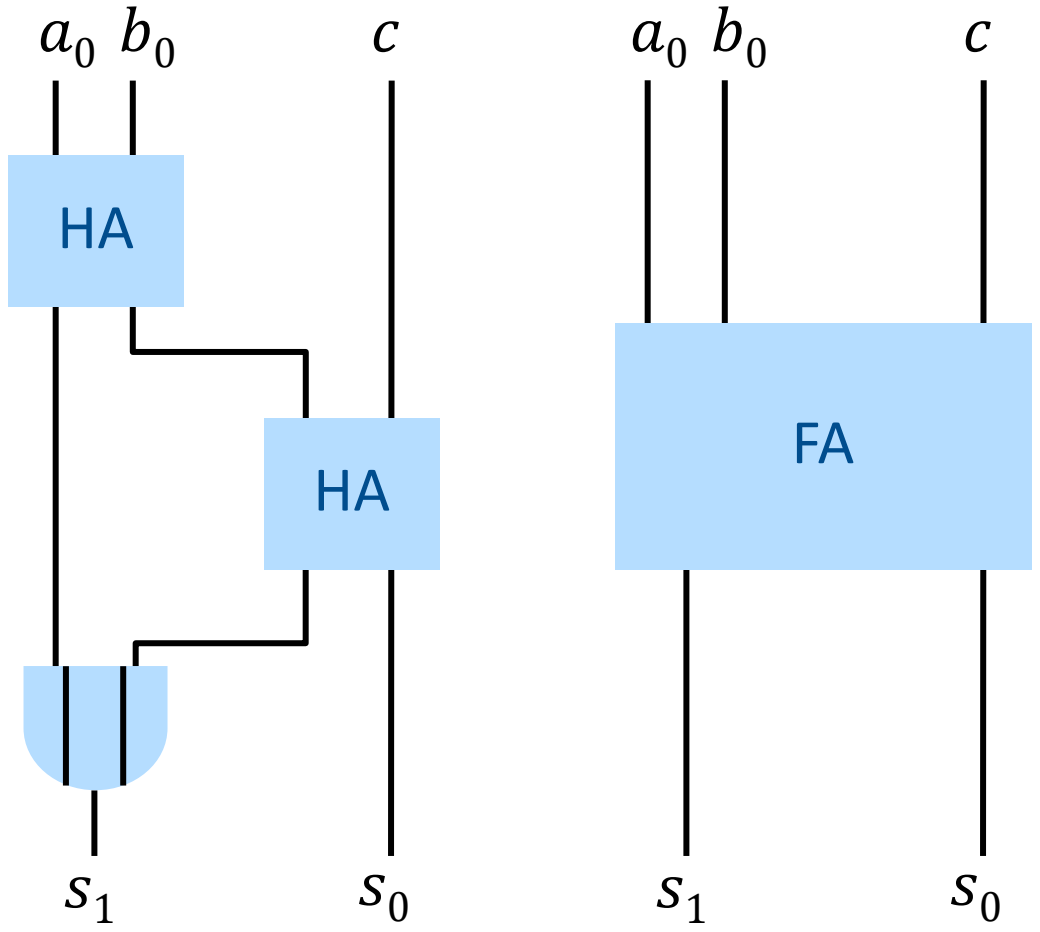
sodass $< s > = 2s_1 + s_0 = a_0 + b_0 + c = < a > \ + < b > \ + c$ gilt.

Der Volladdierer (FA) (2)

$< a >$	$< b >$	c	a_0	b_0	s_1	s_0	$< s >$	ha_1	ha_0
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	1	0	0
0	1	0	0	1	0	1	1	0	1
0	1	1	0	1	1	0	2	0	1
1	0	0	1	0	0	1	1	0	1
1	0	1	1	0	1	0	2	0	1
1	1	0	1	1	1	0	2	1	0
1	1	1	1	1	1	1	3	1	0

$$s_0 = a_0 \oplus b_0 \oplus c = ha_0(c, ha_0(\mathrm{a_0}, \mathrm{b_0}))$$

$$s_1 = a_0 \cdot b_0 + c \cdot (a_0 \oplus b_0)$$
$$= ha_1(a_0, b_0) + ha_1\big(c, ha_0(a_0, b_0)\big)$$



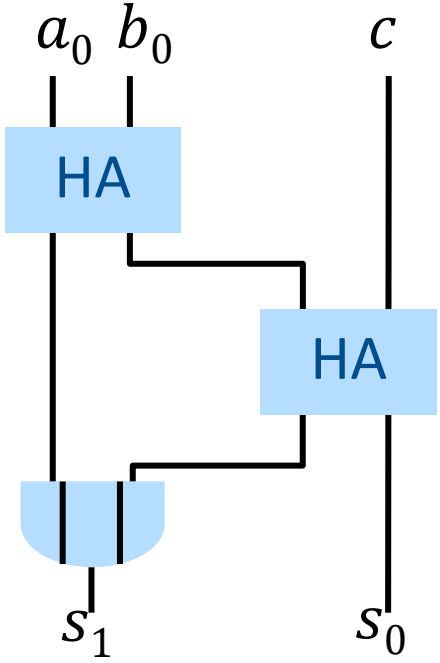
Der Volladdierer – Kosten und Tiefe

Lemma (Kosten & Tiefe eines Volladdierers):
Die Kosten eines Volladdieres betragen $C(FA) = 5$, während die Tiefe $depth(FA) = 3$ entspricht.

Beweis:

Kosten: Der Schaltkreis enthält fünf Modulknoten: $C(FA) = 2 \cdot C(HA) + 1 = 5$

Tiefe: Der längste Weg vom Eingangs- zum Ausgangssignal passiert drei Modulknoten (zwei HA, ein OR).



Bemerkung: Obiges Lemma gilt nur für die dargestellte Realisierung eines Volladdierers.

Carry-Ripple Addierer (CRA)

Idee: Umsetzung der Schulmethode

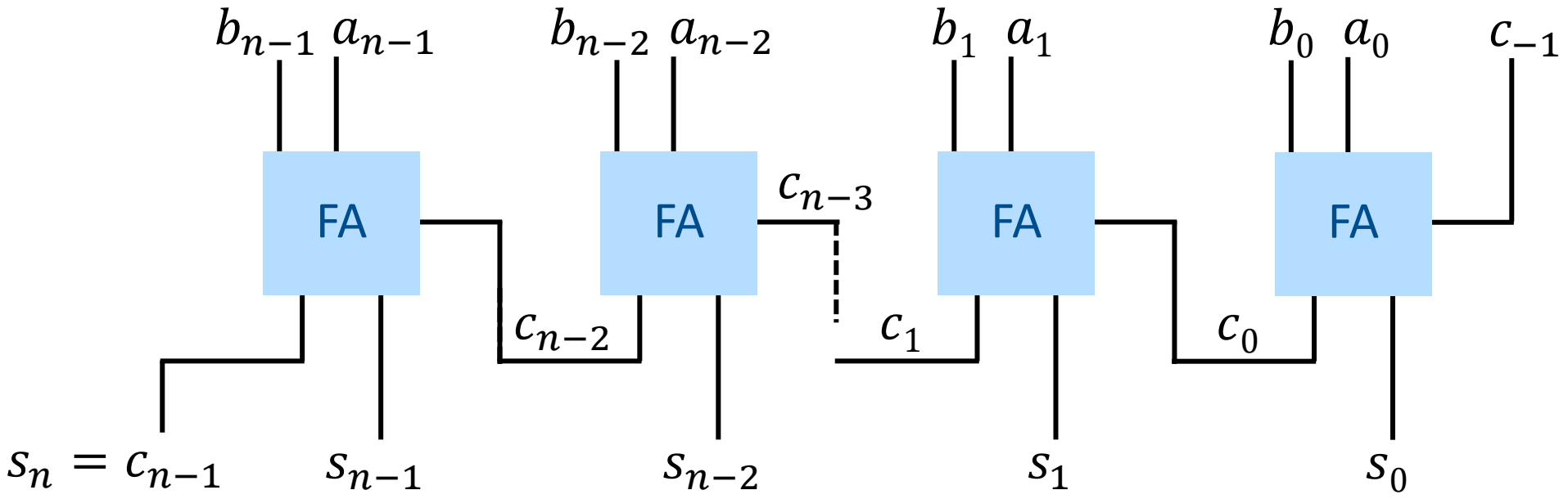
Betrachte klassische schriftliche Addition:

- $CR_1 = FA\,(a, b, c_{-1})$
- $CR_n = CR_{n-1}(a_{n-1} \dots a_1, b_{n-1} \dots b_1, FA(a_0, b_0, c_{-1}))$

$$\begin{array}{rr} & a_{n-1} \dots \dots a_0 \\ + & b_{n-1} \dots \dots b_0 \\ & \qquad\qquad c_{-1} \\ \hline & s_{n-1} \dots \dots s_0 \end{array}$$

Notation:
Der Eingangsübertrag wird mit c_{-1} bezeichnet, der Übertrag von Stelle i nach Stelle $i+1$ mit c_i.

Aufbau eines Carry Ripple Addierers



Addierereigenschaft des Carry-Ripple Addierers

Satz (Addierereigenschaft des CR):
Der Schaltkreis CR_n ist ein n-Bit Addierer.

Beweis:

Durch Induktion über n:

- Induktionsanfang ($n = 1$): $CR_1 = FA$ – erfüllt.
- Induktionsvoraussetzung: Es gibt ein $n \in \mathbb{N}$, sodass der Schaltkreis CR_n ein n-Bit Addierer ist.
- Induktionsschluss: Wenn CR_{n-1} ein $(n-1)$-Bit Addierer ist, ist CR_n ein n-Bit Addierer.

  Beweis: Dafür zeigen wir, dass $CR_n(a_{n-1}, \dots, a_0, b_{n-1}, \dots, b_0, c_{-1}) = (s_n, \dots, s_0)$ mit

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

Fortführung des Beweises

Wir notieren: $s = (s_n, \ldots, s_0), s_{|n-1} = (s_n, \ldots, s_1)$, und analog $a, a_{|n-1}, b, b_{|n-1}$

Wir können nutzen:

- $2 \cdot s_1 + s_0 = a_0 + b_0 + c_{-1}$ für $a, b \in \mathbb{B}^1, s \in \mathbb{B}^2$ (wegen IA), und $c_0 = s_1$ (wegen CR-Konstruktion) (*)
- $< s_{|n-1} > = < a_{|n-1} > + < b_{|n-1} > + c_0$ (wegen der Definition von CR_{n-1}) (**)

Daraus ergibt sich

$$< s > = 2 \cdot < s_{|n-1} > + s_0$$ (Binärdarstellung) (**)

$$\stackrel{\text{def}}{=} 2 \cdot (< a_{|n-1} > + < b_{|n-1} > + c_0) + s_0$$

$$= 2 \cdot < a_{|n-1} > + 2 \cdot < b_{|n-1} > + 2 \cdot c_0 + s_0$$ (*)

$$\stackrel{\text{def}}{=} 2 \cdot < a_{|n-1} > + 2 \cdot < b_{|n-1} > + a_0 + b_0 + c_{-1}$$

$$= 2 \cdot < a_{|n-1} > + a_0 + 2 \cdot < b_{|n-1} > + b_0 + c_{-1}$$ (Binärdarstellung)

$$\stackrel{\text{def}}{=} < a > + < b > + c_{-1}$$

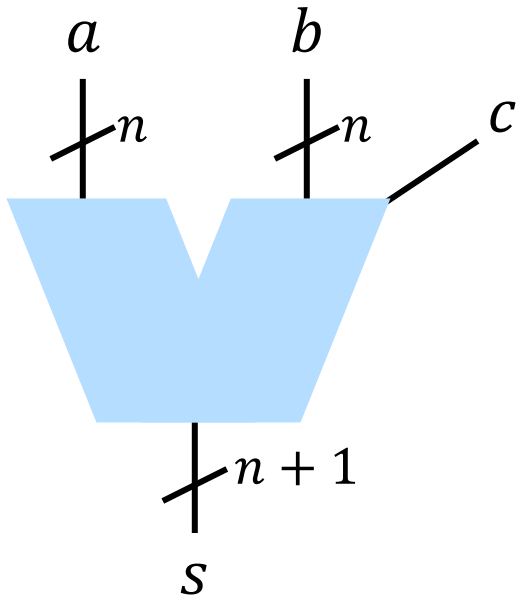
Der Carry-Ripple Addierer – Kosten und Tiefe

Lemma (Kosten & Tiefe eines Carry-Ripple Addierers):
Die Kosten eines Carry-Ripple Addierers betragen $C(CR_n) = 5n$, während die Tiefe $depth(CR_n) = 3 + 2(n-1)$ entspricht.

Beweis:

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

Tiefe: $depth(CR_n) = (n-1) \cdot (depth(FA) - 1) + depth(FA) = 2(n-1) + 3$



Einige weitere wichtige Schaltkreise:

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

Inkrementer

Definition (n-Bit Inkrementer):
Der n-Bit Inkrementer inc_n erhöht eine binäre Zahl a um einen Wert $c \in \{0,1\}$. Die zugehörige Boolesche Funktion ist also beschrieben durch:

$$inc_n: \mathbb{B}^{n+1} \rightarrow \mathbb{B}^{n+1}$$
$$(a_{n-1}, \dots, a_0, c) \mapsto (s_n, \dots, s_0)$$

sodass $< s > = \sum_{i=0}^{n} s_i 2^i = \sum_{i=0}^{n-1} a_i 2^i + c =< a > +c$ gilt.

Umsetzung:

- n-Bit Inkrementer entspricht einem Addierer mit $b_i = 0 \; \forall i \in \{0, \dots, n-1\}$
- Ersetze Volladdierer in CR_n durch Halbaddierer

Inkrementer – Kosten und Tiefe

Lemma (Kosten & Tiefe eines n-Bit Inkrementers):
Die Kosten eines n-Bit Imkrementers betragen $C(inc_n) = 2n$, während die Tiefe $depth(inc_n) = n$ entspricht.

Beweis:

Betrachte Konstruktion durch CR_n mit Halbaddierern:

- Kosten: $C(inc_n) = n \cdot C(HA) = n \cdot 2 = 2n$
- Tiefe: $depth(inc_n) = n \cdot depth\big(C(HA)\big) = n \cdot 1 = n$

Multiplexer

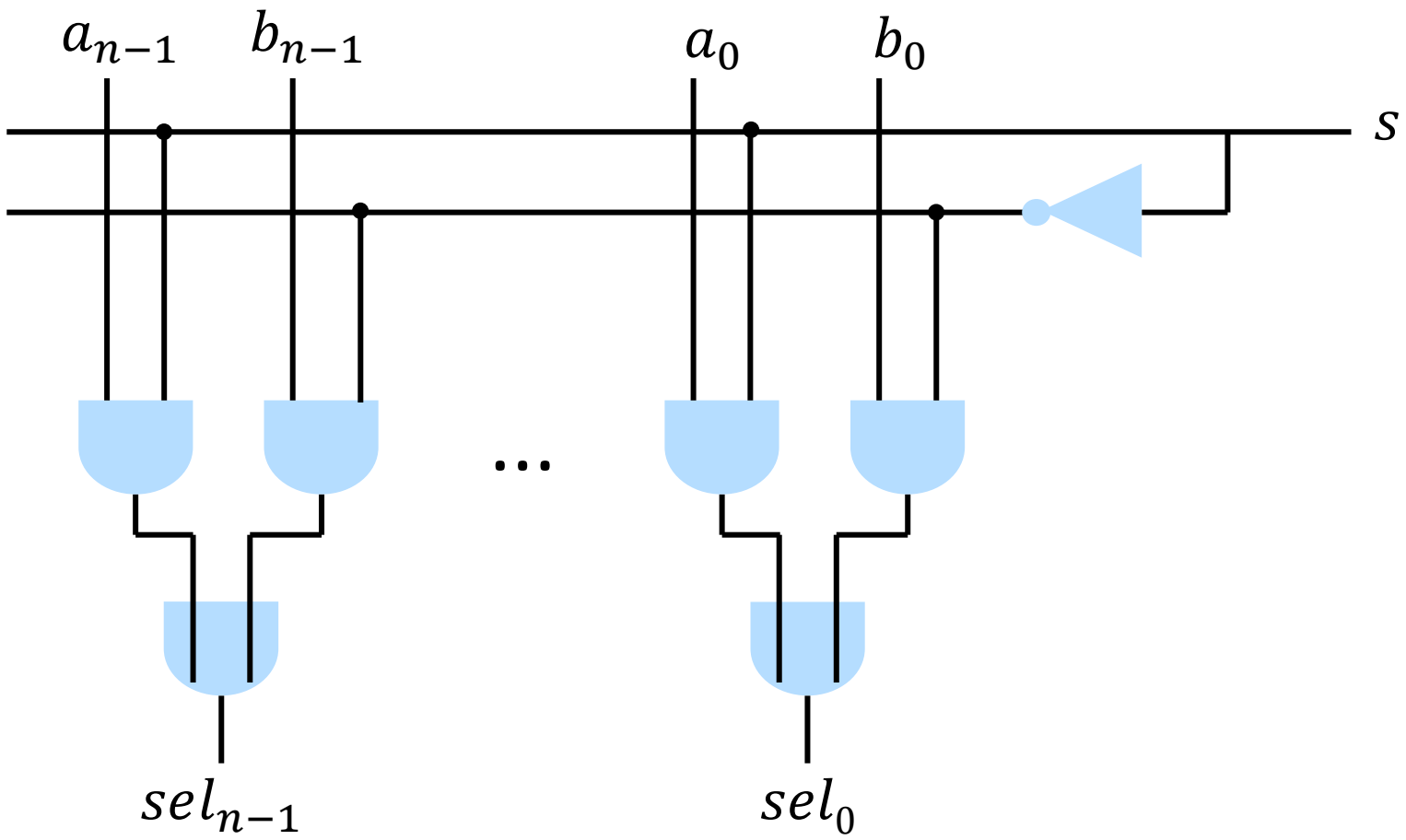
Definition (n-Bit Multiplexer):
Der n-Bit Multiplexer MUX_n wählt entsprechend eines select-Bits $s \in \{0,1\}$ aus zwei n-bit großen Eingängen a, b einen aus. Die zugehörige Boolesche Funktion ist also beschrieben durch:

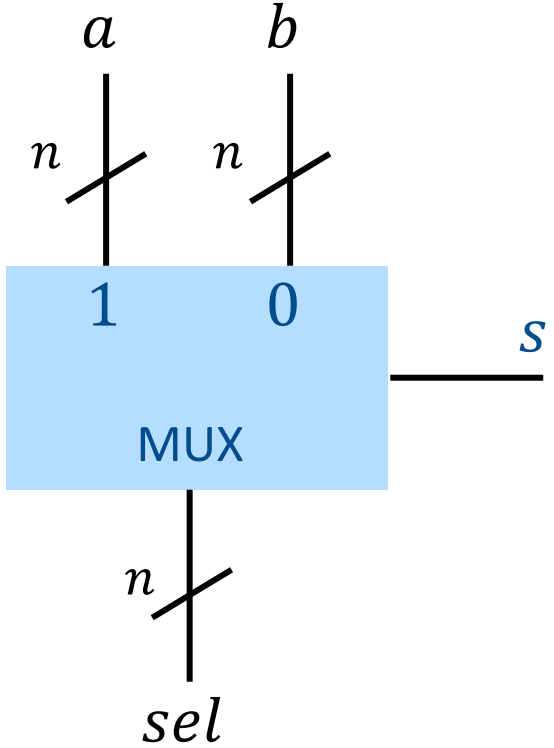
$$sel_n: \mathbb{B}^{2n+1} \to \mathbb{B}^n$$

$$(a_{n-1}, \dots, a_0, b_{n-1}, \dots, b_0, s) \mapsto \begin{cases} (a_{n-1}, \dots, a_0), falls\ s = 1 \\ (b_{n-1}, \dots, b_0), falls\ s = 0 \end{cases}$$

Das heißt, es gilt: $\big(sel_n(a, b, s)\big)_i = s \cdot a_i + \overline{s} \cdot b_i, \forall i \in \{0, \dots, n-1\}$.

n-Bit Multiplexer - Umsetzung





Multiplexer – Kosten und Tiefe

Lemma (Kosten & Tiefe eines n-Bit Multiplexers):
Die Kosten eines n-Bit Multiplexers betragen $C(MUX_n) = 3n + 1$, während die Tiefe $depth(MUX_n) = 3$ entspricht.

Beweis:

Betrachte Konstruktion durch n-fache Wiederholung eines MUX_1:

- Einfacher MUX:
  - Kosten: $C(MUX_1) = 2 + 1 + 1 = 4$ (zwei AND, ein OR, ein NOT)
  - Tiefe: $depth(MUX_1) = 3$ (über NOT, AND, OR)
- Kosten:$C(MUX_n) = 3 \cdot n + 1$ (n Berechnungen von sel_1, ein NOT für s)
- Tiefe:$depth(MUX_1) = 3$ (unabhängig von n)

Rückkehr zum Addierer – geht das auch kostengünstiger?

Lemma (Untere Schranken von Addierern):
Die Kosten eines n-Bit Addierers betragen mindestens die doppelte Bitanzahl, d.h. $C(+_n) \geq 2n$. Die Tiefe eines n-Bit Addierers beträgt mindestens den Logarithmus, d.h. $depth(+_n) \geq \log n + 1$.

Beweis:

n-Bit Addierer können als Binäre Bäume mit $2n + 1$ Blättern gesehen werden.

- Kosten: Binäre Bäume mit $2n + 1$ Blättern haben $2n$ innere Knoten (siehe Foliensatz 10).
- Tiefe: Binäre Bäume mit $2n + 1$ Blättern haben eine Tiefe von $\lceil \log 2n + 1 \rceil \geq \log n + 1$

Ab jetzt: $n = 2^k$

Conditional Sum Addierer (CSA)

Idee: Nutze Parallelverarbeitung, um Tiefe zu reduzieren!

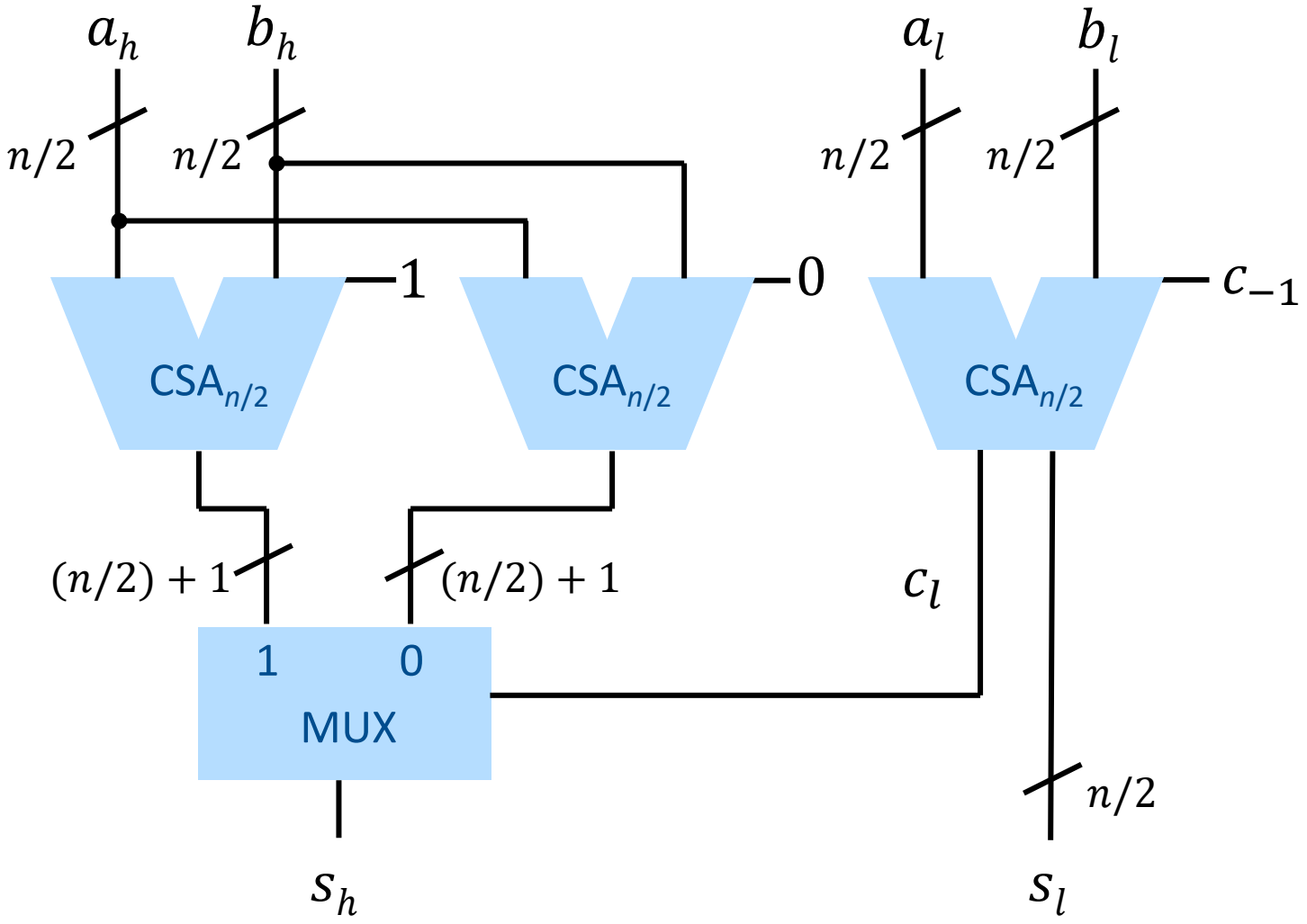
Betrachte klassische schriftliche Addition:

- Aufteilung „in der Mitte“
  - Untere Hälfte für c_{-1} berechnen
  - Obere Hälfte für $c_{n/2-1} = 0$ und $c_{n/2-1} = 1$ berechnen
- Rekursiv wiederholen

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

Schaltbild des CSA_n

- Ein CSA_n enthält drei $CSA_{n/2}$
- a_h, b_h sind die $n/2$ höchstwertigen Bits
- a_l, b_l sind die $n/2$ niederwertigen Bits
- $CSA_1 = FA$



Conditional-Sum Addierer – Kosten und Tiefe

Lemma (Kosten & Tiefe eines Conditional-Sum Addierers):
Die Kosten eines Conditional-Sum Addierers betragen $C(CSA_n) = 10 \cdot n^{\log 3} - 3n - 2 \in \mathcal{O}(n^{\log 3})$, während die Tiefe $depth(CSA_n) = 3 \cdot \log n + 3 \in \mathcal{O}(\log n)$ entspricht.

Beweis:

Beweise folgen auf den nächsten Folien, aufgeteilt auf mehrere Lemmata.

Bemerkung:
Man kann die Idee des CSA fortführen, indem stets beide Summen berechnet werden (Two-Sum Addierer). Dann bleibt die Tiefe in $\mathcal{O}(log\ n)$, während die Kosten sich auf $\mathcal{O}(n\ log\ n)$ reduzieren.

Conditional-Sum Addierer - Tiefe

Lemma (Tiefe eines Conditional-Sum Addierers):
Die Tiefe eines CSA_n liegt in $\mathcal{O}(\log n)$.

Beweis:

- $n = 1$: $depth(CSA_1) = depth(FA) = 3$
- $n \geq 1$:

$$\begin{aligned} depth(CSA_n) &= depth\left(CSA_{\frac{n}{2}}\right) + depth\left(MUX_{\frac{n}{2}+1}\right) \\ &= depth\left(CSA_{\frac{n}{2}}\right) + 3 \\ &= depth\left(CSA_{\frac{n}{4}}\right) + 3 + 3 \\ &= depth\left(CSA_{\frac{n}{8}}\right) + 3 + 3 + 3 \\ &= \cdots \\ &= \text{depth}\left(\text{CSA}_{\frac{\text{n}}{2^{\text{k}}}}\right) + 3 \cdot k \\ &= 3 + 3 \cdot k = 3 \log n + 3 \in \mathcal{O}(\log n) \end{aligned}$$

Conditional-Sum Addierer - Kosten

Lemma (Kosten eines CSA-Addierers):
Die Kosten eines CSA_n liegen bei $C(CSA_n) = 10 \cdot n^{\log 3} - 3n - 2$.

Beweis:

- $n = 1$: $C(CSA_1) = C\ (FA) = 5$
- $n > 1$: $C(CSA_n) = 3 \cdot C\left(CSA_{\frac{n}{2}}\right) + C\left(MUX_{\frac{n}{2}+1}\right)$

$$= 3 \cdot C\left(CSA_{\frac{n}{2}}\right) + 3 \cdot \frac{n}{2} + 4$$

- Es entsteht ein Gleichungssystem der Form: $f(n) = a \cdot f\left(\frac{n}{b}\right) + g(n), f(1) = c$
- Betrachten nur $n = b^k$, für die Lösung des Gleichungssystems

$C(MUX_n) = 3n + 1 \rightarrow$
$C\left(MUX_{\frac{n}{2}+1}\right) = 3\left(\frac{n}{2} + 1\right) + 1$
$= 3 \cdot \frac{n}{2} + 4$

Lösung des Gleichungssystem - Herleitung

Lemma (Gleichungsystem):
Sei $f: \mathbb{N} \to \mathbb{N}$ mit $f(1) = c, f(n) = g(n) + a \cdot f\left(\frac{n}{b}\right)$. Dann gilt für alle $n = b^k$: $f(n) = \sum_{i=0}^{\log_k n - 1} a^i \cdot g\left(\frac{n}{b^i}\right) + a^{\log_k n} \cdot c$

$$f(n) = g(n) + a \cdot f\left(\frac{n}{b}\right) \quad (1)$$
$$= g(n) + a \cdot g\left(\frac{n}{b}\right) + a^2 \cdot f\left(\frac{n}{b^2}\right)$$
$$= g(n) + a \cdot g\left(\frac{n}{b}\right) + a^2 \cdot g\left(\frac{n}{b^2}\right) + a^3 \cdot f\left(\frac{n}{b^3}\right)$$
$$= \cdots$$
$$= g(n) + a \cdot g\left(\frac{n}{b}\right) + a^2 \cdot g\left(\frac{n}{b^2}\right) + \cdots + a^{k-1} \cdot g\left(\frac{n}{b^{k-1}}\right) + a^k \cdot f\left(\frac{n}{b^k}\right) \quad (2)$$
$$= \sum_{i=0}^{k-1} a^i \cdot g\left(\frac{n}{b^i}\right) + a^k \cdot c \quad (3)$$
$$= \sum_{i=0}^{\log_b n - 1} a^i \cdot g\left(\frac{n}{b^i}\right) + a^{\log_b n} \cdot c$$

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

(2) $f\left(\frac{n}{b^k}\right) = f(1) = c, n = b^k$

(3) $n = b^k$

Conditional-Sum Addierer - Kosten

Lemma (Kosten eines CSA-Addierers):
Die Kosten eines CSA_n liegen bei $C(CSA_n) = 10 \cdot n^{\log 3} - 3n - 2$.

Beweis (Fortsetzung) :

- $C(CSA_1) = 5; C(CSA_n) = 3 \cdot C\left(CSA_{\frac{n}{2}}\right) + 3 \cdot \frac{n}{2} + 4$
- Gleichungssystem: $f(n) = a \cdot f\left(\frac{n}{b}\right) + g(n), f(1) = c$
- Für $n = b^k, a = 3, b = 2, c = 5, g(n) = \frac{3}{2}n + 4$:

$$f(n) = \sum_{i=0}^{\log_2 n - 1} a^i \cdot g\left(\frac{n}{b^i}\right) + a^{\log_2 n} \cdot c$$

$$= \sum_{i=0}^{\log_2 n - 1} 3^i \cdot \left(\frac{3}{2} \cdot \frac{n}{2^i} + 4\right) + 3^{\log_2 n} \cdot 5$$

$$= 5 \cdot n^{\log 3} .$$

$$= 10 \cdot n^{\log 3} - 3n - 2$$

$$3^{\log n} = \left(2^{\log_2 3}\right)^{\log_2 n} = \left(2^{\log_2 n}\right)^{\log_2 3} = n^{\log 3}$$

$$4 \cdot \sum_{i=0}^{\log n - 1} 3^i = 4 \cdot \frac{3^{\log n} - 1}{3 - 1} = 2n^{\log 3} - 2$$

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

Carry-Lookahead Addierer (CLA)

Ziel: Näher an untere Schranken – lineare Kosten und logarithmische Tiefe

Idee: Nutze parallele Präfixberechnung für Überträge

Betrachte klassische schriftliche Addition:

- Problemreduktion auf schnelle Berechnung der Überträge
- Sind $c_n, \dots, c_0$ bekannt, so ergibt sich:

$$s_i = a_i \oplus b_i \oplus c_{i-1} \; \forall i \in \{0, \dots, n-1\}$$

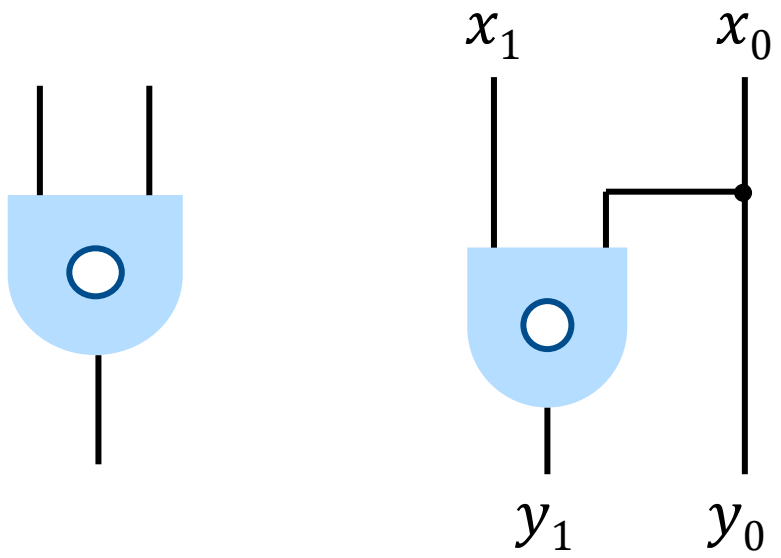
$$\begin{array}{rc} & a_{n-1} \dots \dots a_0 \\ + & b_{n-1} \dots \dots b_0 \\ & c_n \quad \dots \dots \quad c_{-1} \\ \hline & s_{n-1} \dots \dots s_0 \end{array}$$

Parallele Präfix-Berechnung - Definition

Definition (Parallele Präfixfunktion):
Sei M eine Menge, $\circ: M \times M \to M$ eine assoziative Abbildung. Dann heißt die Funktion $PP_{\circ}^{n}: M^n \to M^n, y_i = (x_i \circ \cdots \circ x_0)\ \forall i \in \{0, \ldots, n\}$ parallele Präfixfunktion.

Beispiel:

- Sei die assoziative Abbildung durch ein besonderes Schaltkreiselement abgebildet
- Für $n = 2$, Realisierung folgendermaßen
- Man mache sich die Assoziativität von $\circ$ zunutze – siehe nächste Folie



Funktionsweise der parallelen Präfixberechnung (1)

- Sei im Folgenden: $n = 2^k, i \in \{0, \dots, \left(\frac{n}{2}\right) - 1\}$
- Ungerader Index: Gerade Anzahl an Präfixen

$$y_{2i+1} = x_{2i+1} \circ x_{2i} \circ \cdots \circ x_1 \circ x_0$$
$$= (x_{2i+1} \circ x_{2i}) \circ \cdots \circ (x_1 \circ x_0)$$

- Gerader Index: Ungerade Anzahl an Präfixen

$$y_{2i} = x_{2i} \circ \cdots \circ x_1 \circ x_0$$
$$= x_{2i} \circ (x_{2i-1} \circ x_{2i-2}) \circ \cdots \circ (x_1 \circ x_0)$$

- Zusammenfassung benachbarter Paare: $x'_i = x_{2i+1} \circ x_{2i}$

$$y_{2i+1} = x'_i \circ \cdots \circ x'_0 = y'_i$$
$$y_{2i} = x_{2i} \circ x'_{i-1} \circ \cdots \circ x'_0 = x_{2i} \circ y_{2i-1} = x_{2i} \circ y'_{i-1}$$
$$\dots$$
$$y_0 = x_0$$

Vereinfachte Beschreibung des Vorgehens zur Realisierung von P_n

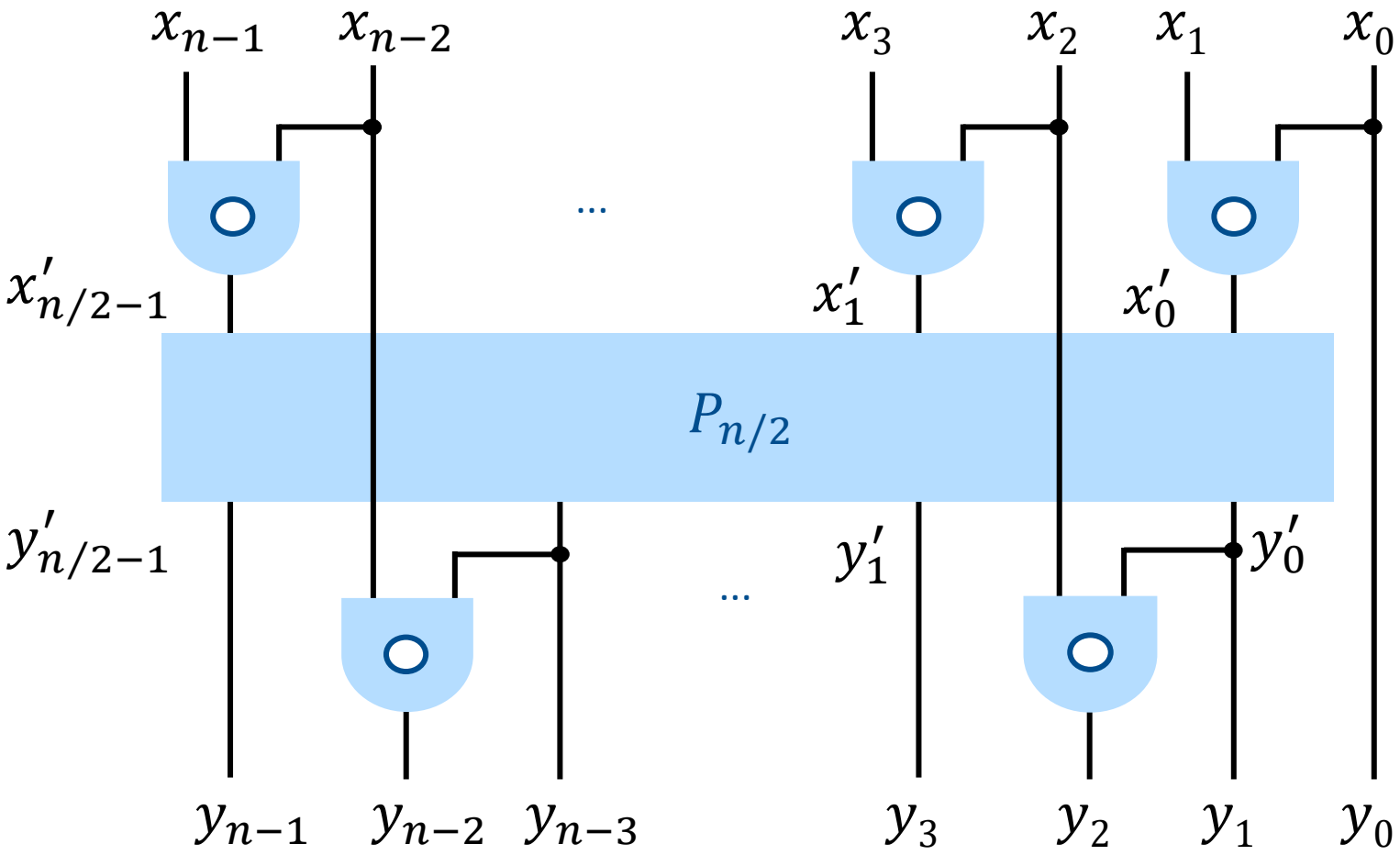
1. Fasse jeweils benachbarte Paare x_{2i+1}, x_{2i} zusammen: $x'_i = x_{2i+1} \circ x_{2i}$
2. Benutze Schaltkreis $P_{\frac{n}{2}}$ mit Eingaben x'_i:

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

3. Ergänze die fehlenden y_{2i}:

$$y_{2i} = x_{2i} \circ (x_{2i-1} \circ \cdots \circ x_0) = x_{2i} \circ y_{2i-1}$$
$$y_0 = x_0$$

Schaltkreis P_n



Parallele Präfixberechnung – Kosten und Tiefe

Lemma (Kosten und Tiefe von P_n):
Sei P_n der parallele Präfixberechner für n-bit Eingaben zu einer assoziativen Verknüpfung $\circ$, und $n = 2^k$. Dann gilt $C(P_n) \leq 2n$ und $depth(P_n) \leq 2 \log n - 1$.

Beweis:

$n = 2^k$, d.h. $C(P_n) \leq 2^{k+1}, depth(P_n) \leq 2k - 1$

- Kosten:
  - Induktionsanfang ($k = 1$): $C(P_0^2) = 1$ (siehe Beispiel)
  - Induktionsschluss ($k - 1 \to k$): $C\left(P_0^{2^k}\right) \leq C\left(P_0^{2^{k-1}}\right) + 2^{k-1} \cdot 2 \leq 2^k + 2^k = 2^{k+1}$
- Tiefe:
  - Induktionsanfang ($k = 1$): $depth(P_0^2) = 1$ (siehe Beispiel)
  - Induktionsschluss ($k - 1 \to k$): $depth\left(P_0^{2^k}\right) \leq 2 + depth\left(P_0^{2^{k-1}}\right) \leq 2 + 2(k-1) - 1 = 2k - 1$

Carry-Lookahead Addierer (CLA)

Ziel: Näher an untere Schranken – lineare Kosten und logarithmische Tiefe

Idee: Nutze parallele Präfixberechnung für Überträge

Betrachte klassische schriftliche Addition:

- Problemreduktion auf schnelle Berechnung der Überträge
- Sind $c_n, \dots, c_0$ bekannt, so ergibt sich:

  $s_i = a_i \oplus b_i \oplus c_{i-1} \; \forall i \in \{0, \dots, n-1\}$
- Betrachte Gruppen von Bits (i bis j)

$$\begin{array}{r} a_{n-1} \dots a_{j+1} a_j \dots a_i a_{i-1} \dots a_0 \\ b_{n-1} \dots b_{j+1} b_j \dots b_i b_{i-1} \dots b_0 \\ \quad\quad\quad\quad c_j \dots \dots c_{i-1} \\ \hline s_{n-1} \dots s_{j+1} s_j \dots s_i s_{i-1} \dots s_0 \end{array}$$

Schnelle Berechnung der c_{i-1}

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

Für Belegungen von $(a_j \ \ldots \ ai)$ und $(b_j \ \ldots \ bi)$ können genau drei Fälle auftreten:

- $c_j = 1$ unabhängig von c_{i-1} (also für $c_{i-1} = 0$ und $c_{i-1} = 1$)

  Sprechweise: Stellen i bis j **generieren** einen Übertrag
- $c_j = 1 \Leftrightarrow c_{i-1} = 1$

  Sprechweise: Stellen i bis j **propagieren** einen Übertrag
- $c_j = 0$ unabhängig von c_{i-1} (also für $c_{i-1} = 0$ und $c_{i-1} = 1$)

  Sprechweise: Stellen i bis j **eliminieren** einen Übertrag

Propagations- und Generationsfunktionen

Definition (Übertragsberechnung):
Seien $n, i, j \in \mathbb{N}, i \leq j < n$. Dann seien die Funktionen

$g_{j,i}: \mathbb{B}^{2n} \to \mathbb{B}$ mit $g_{j,i}(a,b) = \begin{cases} 1, Stellen\ i\ bis\ j\ generieren\ Übertrag \\ 0,\ sonst \end{cases}$ und

$p_{j,i}: \mathbb{B}^{2n} \to \mathbb{B}$ mit $p_{j,i}(a,b) = \begin{cases} 1, Stellen\ i\ bis\ j\ propagieren\ Übertrag \\ 0,\ sonst \end{cases}$ definiert.

Bemerkung:
Die Funktionswerte hängen nur von den Stellen $(a_j \ \dots \ a_i)$ und $(b_j \ \dots \ b_i)$ ab.
Es gelten folgende Eigenschaften:

- $p_{i,i}(a,b) = a_i \oplus b_i$ und $g_{i,i}(a,b) = a_i \cdot b_i$
- Für $i \leq k < j$:
  - $p_{j,i}\ (a,b) = p_{k,i}(a,b) \cdot p_{j,k+1}(a,b)$,
  - $g_{j,i}(a,b) = g_{j,k+1}(a,b) + (g_{k,i}(a,b) \cdot p_{j,k+1}(a,b))$
- $c_i = g_{i,0} + p_{i,0} \cdot c_{-1}$

Anwendung auf Additionsproblem

Lemma:
Seien $g_{j,i}, p_{j,i}$ die Generations- und Propagationsfunktion, seien weiter $a, b \in \mathbb{B}^n, c_{-1} \in \mathbb{B}$. Dann ist für die Berechnung von $s \in \mathbb{B}^{n+1}$ mit $< s > = < a > + < b > + c_{-1}$ nur die Berechnung von $g_{i,0}, p_{i,0}$ notwendig.

Beweis:

- $s_i = a_i \oplus b_i \oplus c_{i-1} = p_{i,i} \oplus \left(g_{i-1,0} + p_{i-1,0} \cdot c_{-1}\right)$
- $s_n = c_{n-1} = g_{n-1,0} + p_{n-1,0} \cdot c_{-1}$
- $s_0 = a_0 \oplus b_0 \oplus c_{-1} = p_{0,0} \oplus c_{-1}$

Aber: Um die parallele Präfixberechnung anwenden zu können, braucht man noch einen assoziativen Operator $\circ$!

Der Operator ∘

Definition (∘-Operator):
Sei $M = (\mathcal{B}_{2n})^2$. Dann ist $\circ: (\mathcal{B}_{2n})^2 \times (\mathcal{B}_{2n})^2 \to (\mathcal{B}_{2n})^2$ mit $(g_2, p_2) \circ (g_1, p_1) \coloneqq (g_2 + (g_1 \cdot p_2), p_1 \cdot p_2)$ ein assoziativer Operator.

Beweis:

Nachrechnen unter Verwendung der Gesetze der Booleschen Algebra.

Eigenschaften:

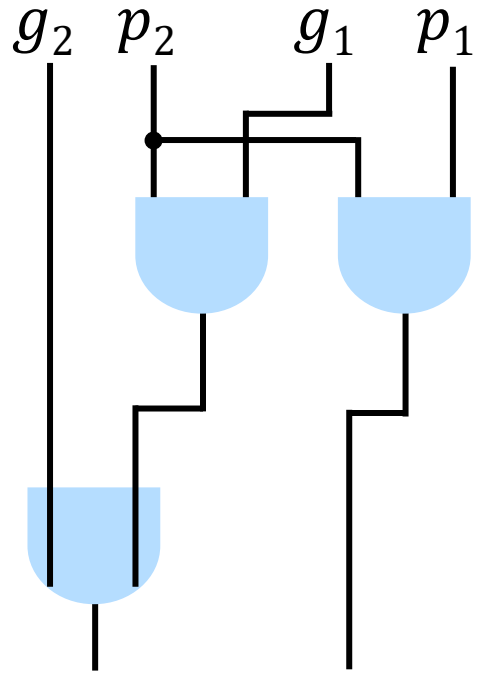
- Die zweite Eigenschaft der Generierungs- und Propagationsfunktionen lässt sich jetzt umschreiben zu: $(g_{j,i}, p_{j,i}) = (g_{j,k+1}, p_{j,k+1}) \circ (g_{k,i}, p_{k,i})$
- Zusammen mit der Assoziativität ergibt sich: $(g_{i,0}, p_{i,0}) = (g_{i,i}, p_{i,i}) \circ \cdots \circ (g_{1,1}, p_{1,1}) \circ (g_{0,0}, p_{0,0})$

Assoziativer Operator: Kosten und Tiefe

Lemma (Kosten und Tiefe des ∘- Operators):
Die Kosten des ∘-Operators betragen $C(\circ) = 3$, die Tiefe beträgt $depth(\circ) = 2$.

Beweis:

(siehe Schaltbild der Basiszelle)



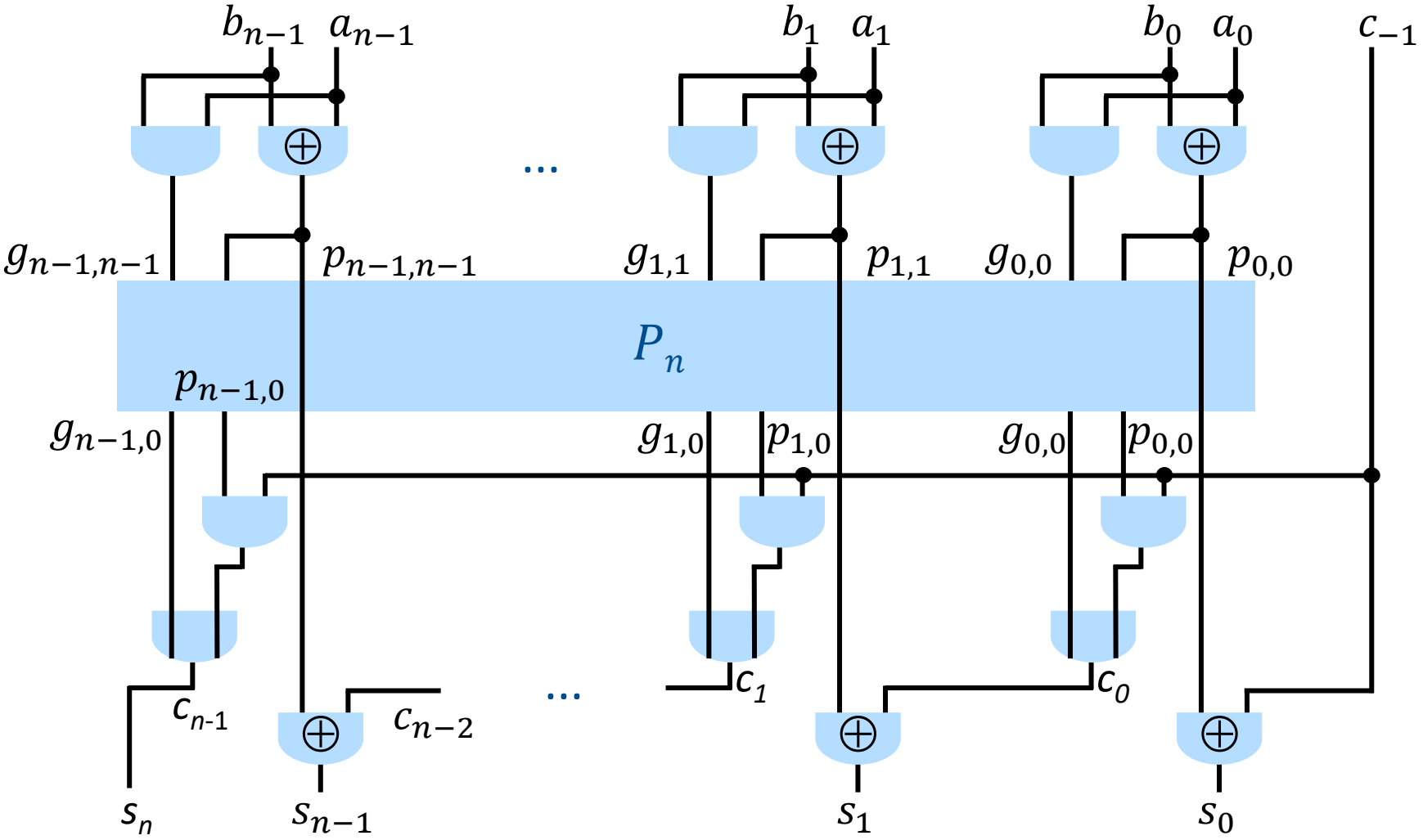
Parallele Präfixberechnung auf assoziativem Operator

Lemma (Kosten und Tiefe von P_n über $\circ$):
Die Kosten der parallelen Präfixberechnung über dem assoziativen Operator betragen $C\big(P_n(\circ)\big) \leq 2n \cdot C(\circ) = 6n$. Die Tiefe beträgt $depth\big(P_n(\circ)\big) = (2 \cdot \log n - 1) \cdot C(\circ) = 4 \cdot \log n \ - 2.$

Beweis:

Kombination der Lemmata zu Kosten und Tiefe von P_n und $\circ$.

Gesamtschaltkreis



Carry-Lookahead Addierer – Kosten und Tiefe

Lemma (Kosten und Tiefe des Carry-Lookahead Addierers):
Die Kosten des Carry-Lookahead Addierers betragen $C(CLA_n) \leq 6n + 2n + 3n = 11n$. Die Tiefe des Carry-Lookahead Addierers beträgt $depth(CLA_n) \leq 4\log n\ - 2 + 1 + 3 = 4\log n + 2$.

Beweis aus Herleitung

Addition von Zweierkomplementzahlen

Definition (vorzeichenbehaftete Festkommazahl):
Eine Festkommazahl $d = d_n d_{n-1} \dots d_0 \dots d_{-k}$ wird vorzeichenbehaftet wie folgt interpretiert:

- Betrag und Vorzeichen: $[d]_{BV} = [d_n d_{n-1} \dots d_0 \dots d_{-k}]_{BV} := (-1)^{d_n} \sum_{i=-k}^{n-1} d_i \cdot 2^i$
- Einerkomplement-Darstellung: $[d]_1 = [d_n d_{n-1} \dots d_0 \dots d_{-k}]_1 := \sum_{i=-k}^{n-1} d_i \cdot 2^i - d_n(2^n - 2^{-k})$
- **Zweierkomplement-Darstellung:** $[\boldsymbol{d}]_2 = [\boldsymbol{d_n d_{n-1} \dots d_0 \dots d_{-k}}]_2 := \sum_{i=-k}^{n-1} \boldsymbol{d_i} \cdot \boldsymbol{2^i} - \boldsymbol{d_n 2^n}$

Ist $d_n = 1$, ergibt sich $[d]$ zu einem negativen Wert; ist $d_n = 0$, ergibt sich $[d]$ zu $< d >$ und entspricht der vorzeichenlosen Interpretation einer Festkommazahl.

Addition erfolgt also wie folgt:

$$[a]_2 + [b]_2 = (-a_n 2^n) + (-b_n 2^n) + \sum_{i=0}^{n-1} a_i 2^i + \sum_{i=0}^{n-1} b_i 2^i = (-(a_n + b_n) \cdot 2^n) + \sum_{i=0}^{n-1} (a_i + b_i) 2^i = [a + b]_2$$

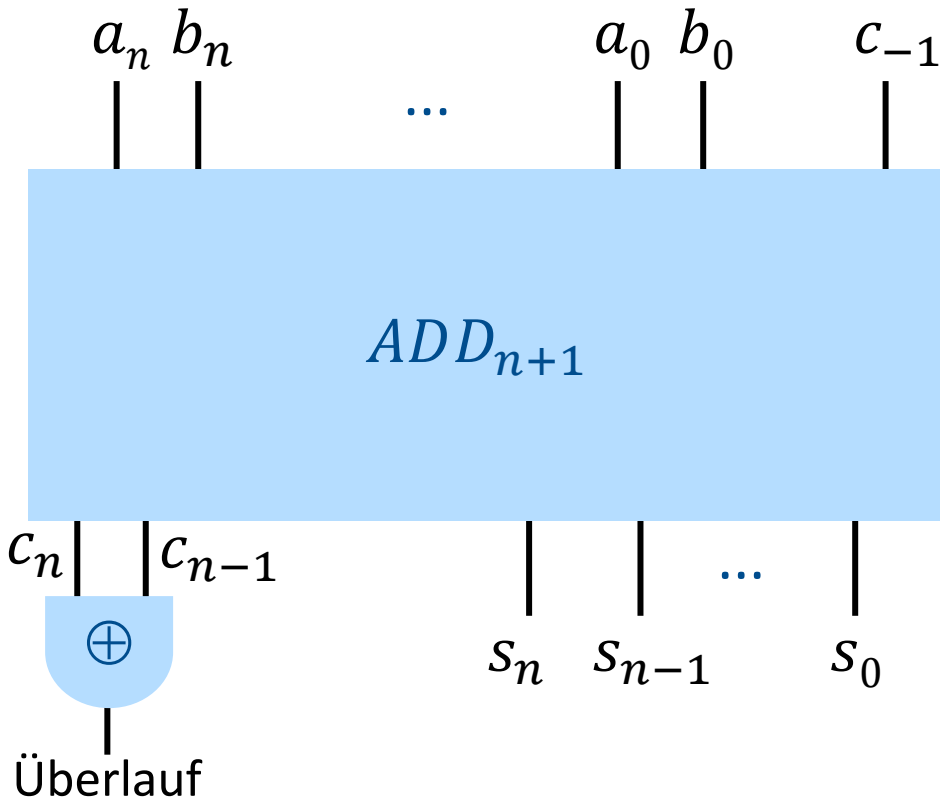
Nutzen von $(n+1)$-Bit Binäraddierern

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

Erinnerung: Addition ist nicht abgeschlossen – Überlauf!

Behauptung: Der Test, ob das Ergebnis durch eine $(n+1)$-Bit Zweierkomplementzahl darstellbar ist, d. h. ob das Ergebnis aus $R_n = \{-2^n, \dots, 2^n - 1\}$ ist, lässt sich zurückführen auf den Test $c_n = c_{n-1}$.

(*n*+1)-Bit Addierer



Überlaufstest bei Addierern

Satz (Überlauftest bei Addierern):
Seien $a, b \in \mathbb{B}^{n+1}, c_{-1} \in \{0,1\}$ und $s \in \{0,1\}^{n+1}$, sodass $< (c_n, s) > = < a > + < b > + c_{-1}$. Dann gilt:

- $[a]_2 + [b]_2 + c_{-1} \in R_n \Leftrightarrow c_n = c_{n-1}$
- $[a]_2 + [b]_2 + c_{-1} \in R_n \Rightarrow [a]_2 + [b]_2 + c_{-1} = [s]$

(ohne Beweis)

Steht c_{n-1} nicht zur Verfügung (z.B. CSA), so kann man den Überlauftest $[a]_2 + [b]_2 + c_{-1} \notin R_n \Leftrightarrow a_n = b_n \neq s_n$ verwenden.