Kapitel 3 – Kombinatorische Logik

1. Kombinatorische Schaltkreise
2. Normalformen, zweistufige Synthese
3. Berechnung eines Minimalpolynoms
4. **Arithmetische Schaltungen**
5. Anwendung: ALU von ReTI

UNI FREIBURG

Albert-Ludwigs-Universität Freiburg

Dr. Tobias Schubert, Dr. Ralf Wimmer

Professur für Rechnerarchitektur

WS 2016/17

Arithmetische Schaltungen

- Addieren nach der Schulmethode: Carry-Ripple-Addierer.
- Effizienteres Addieren: Conditional-Sum-Addierer.
- Addition von Zweierkomplement-Zahlen.
- Subtrahierer.
- Exkurs: Multiplizierer.

Kosten von Schaltkreisen

- Um unterschiedliche Schaltkreise, die eine Funktion (z.B. Addierer) implementieren, miteinander zu vergleichen, benötigt man ein Kostenmaß.

Definition

Die Kosten $C(SK)$ eines Schaltkreises SK sind durch die Anzahl seiner Gatter gegeben.

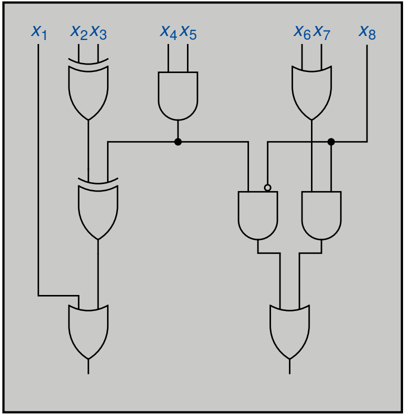
- Deutet auf die Fläche und den Energieverbrauch der resultierenden Hardware-Blöcke hin.

Definition

Die Tiefe $depth(SK)$ eines Schaltkreises ist die maximale Anzahl von Gattern auf einem Pfad von einem beliebigen Eingang zu einem beliebigen Ausgang von SK.

- Deutet auf die Signallaufzeit durch SK und somit die maximal mögliche Taktfrequenz (Geschwindigkeit) des Schaltkreises hin.

Beispiel: Kosten und Tiefe

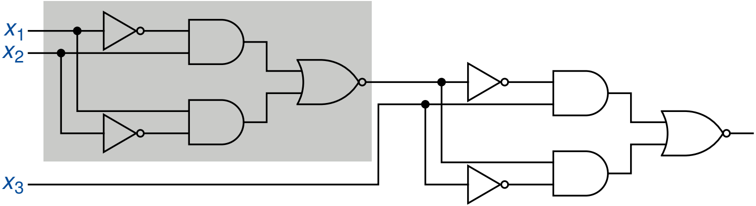


$$C(SK) = 8$$

$$Depth(SK) = 3$$

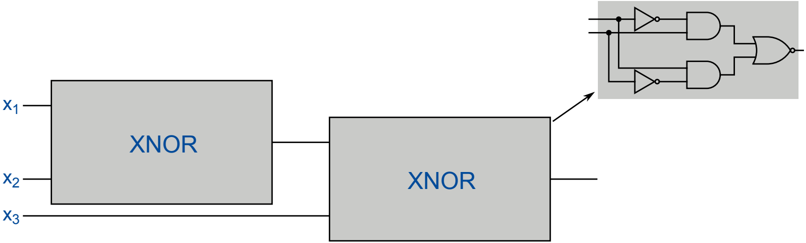
Teilschaltkreise, hierarchischer Entwurf (informell)

- Illustration eines Teilschaltkreises.



Hierarchische Schaltkreise

- In hierarchischen Schaltkreisen sind Teilschaltkreise durch Symbole ersetzt.
- Den zugehörigen („flachen”) Schaltkreis erhält man, indem man die Symbole durch Einsetzen der Teilschaltkreise wieder entfernt.



Wiederholung Zahlendarstellung

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 für nichtnegative Zahlen

- **Gegeben:**
  2 positive Binärzahlen $\langle a \rangle = \langle a_{n-1} \ldots a_0 \rangle, \langle b \rangle = \langle b_{n-1} \ldots b_0 \rangle$ mit 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 $n + 1$ Stellen für die Darstellung von s, d.h. der Schaltkreis hat $n + 1$ Ausgänge.

Formale Definition n-Bit-Addierer

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

$$+_n : \mathbb{B}^{2n+1} \to \mathbb{B}^{n+1},$$
$$+_n : (a_{n-1}, \ldots, a_0, b_{n-1}, \ldots, b_0, c) = (s_n, \ldots, s_0)$$
$$\text{mit } \langle s \rangle = \langle s_n \ldots s_0 \rangle = \langle a_{n-1} \ldots a_0 \rangle + \langle b_{n-1} \ldots b_0 \rangle + c$$

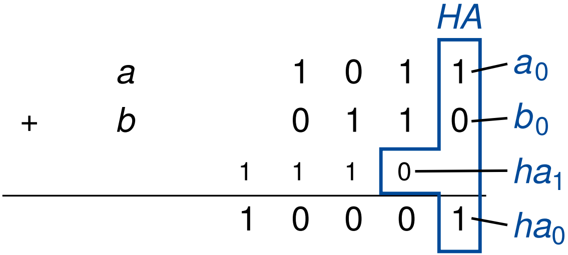
Addieren nach der Schulmethode (1/4)

- Wir werden im Folgenden den einfachsten Addierertypen einführen, der die „Schulmethode" umsetzt.
- Hierzu werden einige Grundschaltungen (Halb- und Volladierer) notwendig sein.
- Beispiel für die Schulmethode:

$$
\begin{array}{cc r}
 & a & 1\,0\,1\,1 \\
+ & b & 0\,1\,1\,0 \\
 & & {\scriptstyle 1\;1\;1\;0\;\;} \\
\hline
 & & 1\,0\,0\,0\,1
\end{array}
$$

UNI FREIBURG

Addieren nach der Schulmethode (2/4)



a_0	b_0	ha_1	ha_0
0	0		
0	1		
1	0		
1	1		

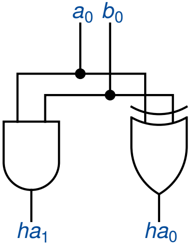
Halbaddierer (Half Adder, *HA*)

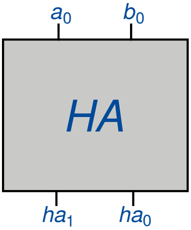
- Ein Halbaddierer dient zur Addition zweier 1-Bit-Zahlen ohne Eingangsübertrag.
- Er berechnet die Funktion $ha : \mathbb{B}^2 \to \mathbb{B}^2$ mit $ha(a_0, b_0) = (ha_1, ha_0)$, wobei $2ha_1 + ha_0 = a_0 + b_0$.

a_0	b_0	ha_1	ha_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$ha_0(a_0, b_0) = a_0 \oplus b_0$$
$$ha_1(a_0, b_0) = a_0 \wedge b_0$$

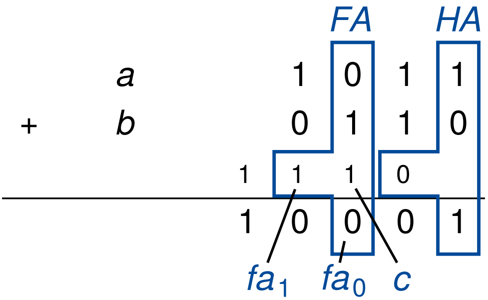
Schaltkreis eines Halbaddierers





dabei gilt:
$C(HA) = 2, \quad depth(HA) = 1$

Addieren nach der Schulmethode (3/4)



Volladierer (Full Adder, *FA*)

- Ein Volladdierer dient zur Addition zweier 1-Bit-Zahlen mit Eingangsübertrag.
- Er berechnet die Funktion $fa : \mathbb{B}^3 \to \mathbb{B}^2$ mit $fa(a_0, b_0, c) = (fa_1, fa_0)$ wobei $2fa_1 + fa_0 = a_0 + b_0 + c$

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 *HA*s

Aus der Tabelle folgt:

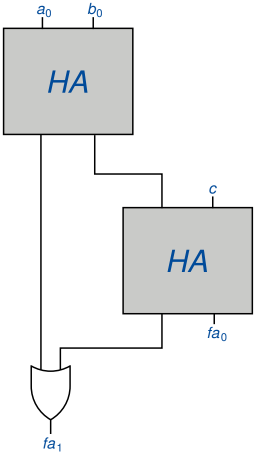
$fa_0 = a_0 \oplus b_0 \oplus c$
$= ha_0(c, ha_0(a_0, b_0))$

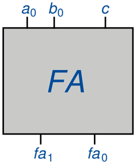
$fa_1 = (a_0 \wedge b_0) \vee (c \wedge (a_0 \oplus b_0))$
$= ha_1(a_0, b_0) + ha_1(c, ha_0(a_0, b_0))$

Kosten und Tiefe eines *FA*:
$C(FA) = 5, \quad depth(FA) = 3$

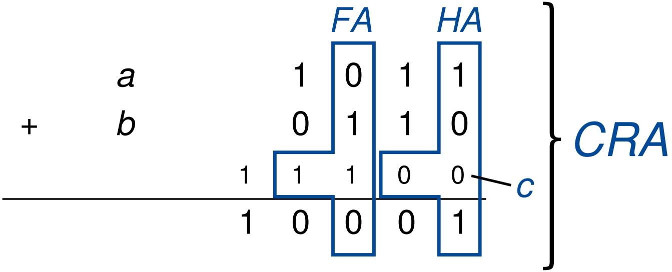
a_0	b_0	c	fa_1	fa_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Schaltkreis eines Volladdierers

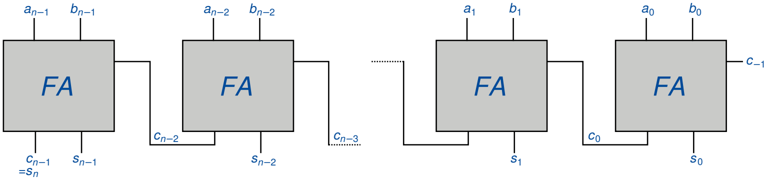




Addieren nach der Schulmethode (4/4)

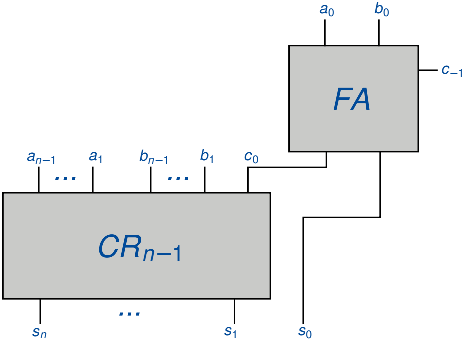


Aufbau eines Carry-Ripple-Addierers



Induktive Definition des Carry-Ripple-Addierers *CR*

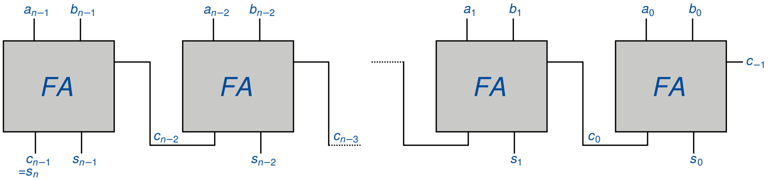
- Für $n = 1$: $CR_1 = FA$
- Für $n > 1$: Folgender Schaltkreis:

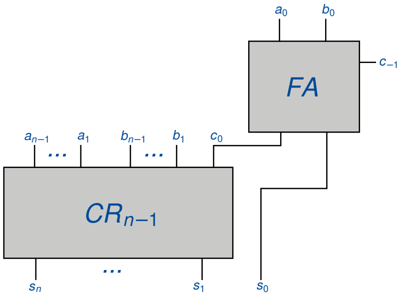


c_{-1}: Eingangsübertrag

c_i: Übertrag von Stelle i nach $i+1$

Zwei (identische) Darstellungen von *CR*





Carry-Ripple-Addierer

Satz

CR_n ist ein n-Bit-Addierer.

Beweis (duch Induktion):

- $n = 1$ ($CR_1 = FA$) ✓
- $n-1 \to n$: 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:
  $\langle s\rangle = \langle s_n \ldots s_0\rangle = \langle a_{n-1}\ldots a_0\rangle + \langle b_{n-1}\ldots b_0\rangle + c_{-1}$.
- Nach Induktionsvoraussetzung gilt für CR_{n-1}:
  $\langle s_n \ldots s_1\rangle = \langle a_{n-1}\ldots a_1\rangle + \langle b_{n-1}\ldots b_1\rangle + c_0$. (a)
  Wegen FA-Eigenschaft gilt $\langle c_0, s_0\rangle = a_0 + b_0 + c_{-1}$. (b)
- Insgesamt: $\langle s_n \ldots s_0\rangle = 2\cdot\langle s_n\ldots s_1\rangle + s_0$
  $\overset{(a)}{=} 2\cdot(\langle a_{n-1}\ldots a_1\rangle + \langle b_{n-1}\ldots b_1\rangle + c_0) + s_0$
  $= 2\cdot(\langle a_{n-1}\ldots a_1\rangle + \langle b_{n-1}\ldots b_1\rangle) + 2\cdot c_0 + s_0$
  $\overset{(b)}{=} 2\cdot\langle a_{n-1}\ldots a_1\rangle + a_0 + 2\cdot\langle b_{n-1}\ldots b_1\rangle + b_0 + c_{-1}$
  $= \langle a\rangle + \langle b\rangle + c_{-1}$

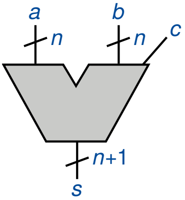
UNI FREIBURG

Schaltbild und Komplexität von *CR*

- $C(CR_n) = n \cdot C(FA) = 5n$.
- $depth(CR_n) = ?$.

Schaltbild und Komplexität von CR

- $C(CR_n) = n \cdot C(FA) = 5n$.
- $depth(CR_n) = 3 + 2(n - 1)$.
- Sowohl die Kosten als auch die Tiefe von CR sind somit linear in n.



- Es gibt (asymptotisch) bessere Addierer. Wir werden hier den Conditional-Sum-Addierer kennen lernen, für den wir wieder eine Hilfsschaltung (Multiplexer) benötigen.
- Eine weitere wichtige Schaltung ist der Inkrementer.

n-Bit-Inkrementer

Definition

Ein n-Bit-Inkrementer INC_n berechnet die Funktion
$inc_n : \mathbb{B}^{n+1} \to \mathbb{B}^{n+1}$
$inc_n(a_{n-1}, \ldots, a_0, c) = (s_n, \ldots, s_0)$ mit $\langle s_n \ldots s_0 \rangle = \langle a \rangle + c$

- Ein Inkrementer ist ein Addierer mit $b_i = 0$ für alle i.
  $\Rightarrow$ Ersetze in CR_n die FA durch HA.
- Kosten und Tiefe:
  - $C(INC_n) = n \cdot C(HA) = 2n$
  - $depth(INC_n) = n \cdot depth(HA) = n$

n-Bit-Multiplexer

Definition

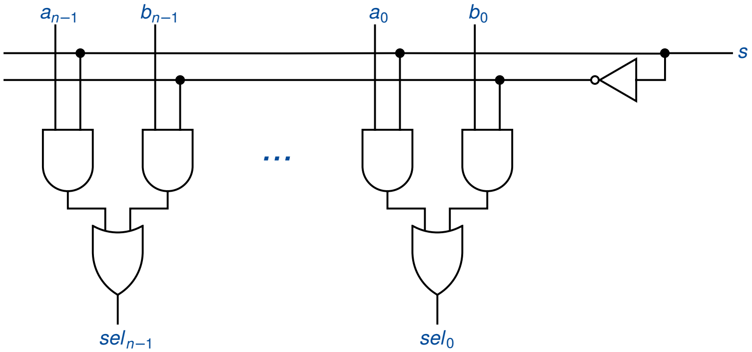
Ein n-Bit-Multiplexer MUX_n berechnet die Funktion

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

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

- Es gilt: $(sel_n)_i = s \cdot a_i + \overline{s} \cdot b_i$

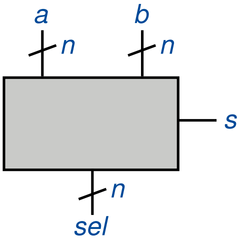
Aufbau von MUX_n



Schaltbild und Kosten MUX_n

Kosten und Tiefe:

$C(MUX_n) = 3n + 1.$
$depth(MUX_n) = 3.$



Rückkehr zum Addierer

Gibt es billigere Addierer als CR_n?

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

Sei $f \in \mathbb{B}_n$. Dann sind $C(f)$ und $depth(f)$ definiert durch
$C(f) = min\{C(SK) | f_{SK} = f\}$ und
$depth(f) = min\{depth(SK) | f_{SK} = f\}$.

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 $n = 2^k$.

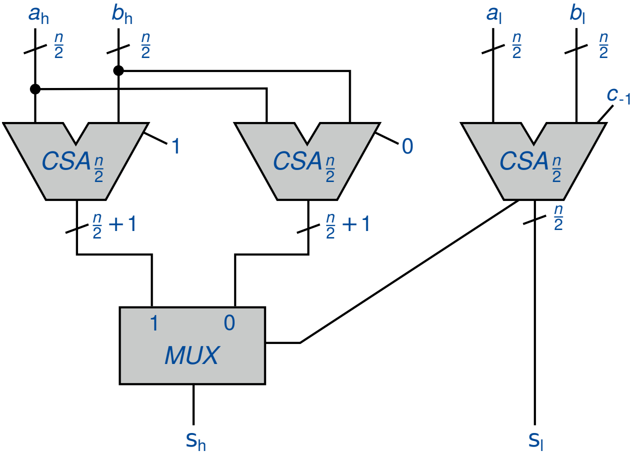
Conditional-Sum-Addierer (*CSA*)

- Idee: Nutze Parallelverarbeitung, um Tiefe zu reduzieren!

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

- $CSA_1 = FA$.
- CSA_n: Siehe nächste Folie.
- Im Folgenden sei $n = 2^k$.

Aufbau von CSA_n



Komplexität von CSA_n: Tiefe

Satz

CSA_n hat Tiefe $\leq 3\log(n)+3$.

Beweis: Setze $n = 2^k$ voraus.

- $n = 1$: $depth(CSA_1) = depth(FA) = 3$.
- $n > 1$: $depth(CSA_n) \leq depth(CSA_{\frac{n}{2}}) + depth(MUX_{\frac{n}{2}+1})$
  $\leq depth(CSA_{\frac{n}{2}}) + 3$
  $\leq depth(CSA_{\frac{n}{4}}) + 3 + 3$
  $\leq depth(CSA_{\frac{n}{8}}) + 3 + 3 + 3$
  $\ldots$
  $\leq depth(CSA_{\frac{n}{2^k}}) + k \cdot 3$
  $= depth(CSA_1) + k \cdot 3$
  $\leq 3 \cdot (k+1) = 3\log(n) + 3$.

Komplexität von CSA_n: Kosten

Satz (ohne Beweis)

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

- Man kann den hier vorgestellten *CSA* in einfacher Weise modifizieren, so dass
  - Tiefe = $O(\log(n))$,
  - Kosten = $O(n \cdot \log(n))$.
- Es gibt auch Addierer mit linearen Kosten und logarithmischer Tiefe.
  - Carry-Lookahead-Addierer (*CLA*).
  - $C(CLA_n) \leq 11n$,
  - $depth(CLA_n) \leq 4 \cdot \log(n) + 2$.

UNI FREIBURG

Addition von Zweierkomplementzahlen

- Auszurechnen ist:

  $$[a_n a_{n-1} \ldots a_0] + [b_n b_{n-1} \ldots b_0] = (-a_n 2^n) + (-b_n 2^n) + \sum_{i=0}^{n-1} a_i 2^i + \sum_{i=0}^{n-1} b_i 2^i$$

- Im Fall von $(n+1)$-Bit-Zweierkomplementzahlen können Ergebnisse im Bereich $R_n = \{-2^n, \ldots, 2^n - 1\}$ dargestellt werden; andernfalls kommt es zu einem Überlauf.

- Der Satz auf der nächsten Folie sagt aus:
  - Kommt es bei der Addition nicht zu einem Überlauf, so kann man den „gewöhnlichen” Binäraddierer zur Addition von Zweierkomplementzahlen benutzen.
  - Ob es zu einem Überlauf kommt, lässt sich anhand von Werten a_n, b_n und s_n im Binäraddierer entscheiden.

Zweierkomplement-Addition formal

Satz

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

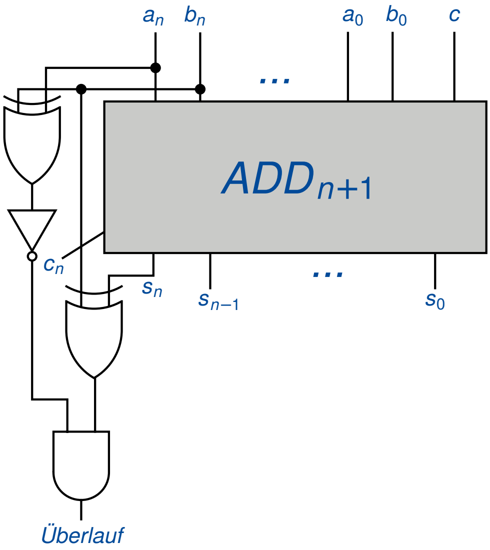
Dann gilt:

- $[a] + [b] + c_{-1} \notin R_n \Leftrightarrow (a_n = b_n) \wedge (b_n \neq s_n)$
- $[a] + [b] + c_{-1} \in R_n \Rightarrow [a] + [b] + c_{-1} = [s]$

- Beweis durch Fallunterscheidung ($[a], [b]$ beide positiv, beide negativ, $[a]$ positiv / $[b]$ negativ) und Nachrechnen (Induktion).
- Man kann einen alternativen Überlauftest zeigen:

$$[a] + [b] + c_{-1} \notin R_n \Leftrightarrow c_n \neq c_{n-1}$$

Addierer für $(n+1)$-Bit-Zweierkomplement-Zahlen



Subtraktion

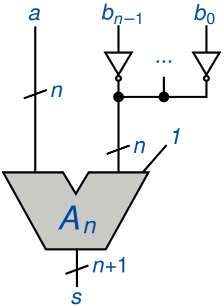
- Wegen $-[b] = [\overline{b}] + 1$ kann $[a] - [b]$ zurückgeführt werden auf $[a] + [\overline{b}] + 1$.
- **Beispiel:**

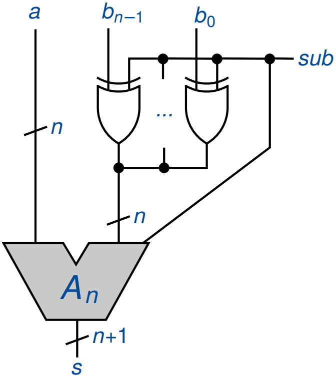
$$[a] = [0110] = 6_{10}, \quad [b] = [0111] = 7_{10}, \quad [\overline{b}] = [1000]$$

$$\begin{array}{rr} & 0\,1\,1\,0 \\ + & 1\,0\,0\,0 \\ + & 1 \\ \hline & 1\,1\,1\,1 \end{array} \qquad 1111 = (-1)_{10}$$

- Den Schaltkreis für Subtraktion gewinnt man aus einem Addierer.
- Kombinierter Addierer/Subtrahierer.

Subtrahierer





$$b_i \oplus 0 = b_i \qquad sub = 0 : [a] + [b] + 0$$
$$b_i \oplus 1 = \overline{b_i} \qquad sub = 1 : [a] + [\overline{b}] + 1 = [a] - [b]$$