Rekursives Schaltungsdesign

Addierer

Wir werden fünf verschiedene Addiererkonstruktionen kennen lernen:

- Ripple Carry Adder (RCA)
- Conditional Sum Adder (CSA)
- Two Sum Adder (TSA)
- Block Carry Adder (BCA)
- Carry Lookahead Adder (CLA)

Halbaddierer

Ein Halbaddierer realisiert eine 1-stellige Addition zweier Binärzahlen.

Schaltfunktion $HA : \{0,1\}^2 \longrightarrow \{0,1\}^2$ mit

$$HA(a_0, b_0) = (c_1, c_0)$$

wobei gilt:

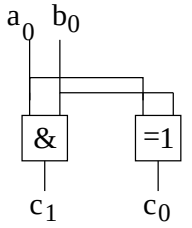
$$\langle a_0 \rangle_2 + \langle b_0 \rangle_2 = \langle c_1 c_0 \rangle_2$$

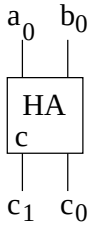
Der Ausgang c_0 wird auch als Summenbit bezeichnet, c_1 als Übertragsbit oder auch 'carry bit'.

Wertetabelle:

a_0	b_0	c_1	c_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Die Schaltfunktionen können direkt mit jeweils einem Gatter (also Kosten 2 und Tiefe 1) realisiert werden.





Volladdierer

Ein Volladdierer realisiert eine 1-stellige Addition zweier Binärzahlen mit Eingangsübertrag.

Schaltfunktion $FA : \{0,1\}^3 \longrightarrow \{0,1\}^2$ mit

$$FA(a_0, b_0, c_0) = (s_1, s_0)$$

wobei gilt:

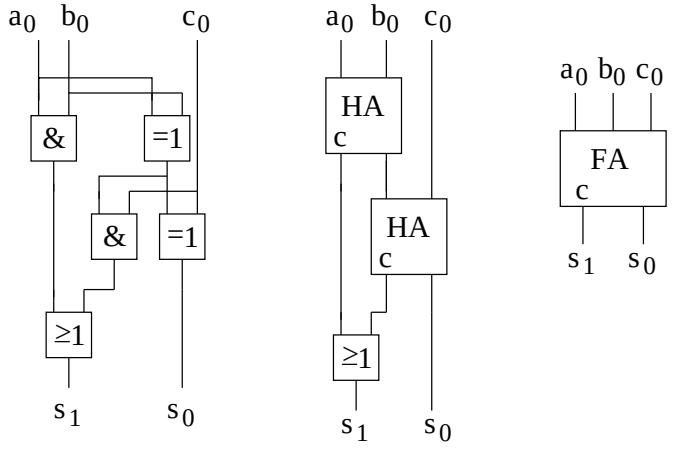
$$\langle a_0 \rangle_2 + \langle b_0 \rangle_2 + \langle c_0 \rangle_2 = \langle s_1 s_0 \rangle_2$$

Analog zum Halbaddierer bezeichnet man s_0 als Summenbit und s_1 als Übertrags– bzw. 'carry'-Bit.

d.h. wir müssen folgende Wertetabelle realisieren:

a_0	b_0	c_0	s_1	s_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

Dies ist mit einfachen Ausdrücken möglich, was folgende Schaltung mit Kosten 5 und Tiefe 3 zeigt:



n-Bit Addierer

Wir haben gesehen, dass es verschiedene Möglichkeiten gibt, einen Schaltkreis aufzubauen. Man kann versuchen, die Kosten oder die Tiefe des Schaltkreises zu minimieren. Es ergibt sich häufig ein trade-off zwischen Kosten und Tiefe und man muss sich je nach Gegebenheit für eine Variante entscheiden. Wir wollen im Folgenden einige Addierer kennen lernen, die sich bezüglich Kosten und Tiefe zum Teil erheblich unterscheiden.

Zuerst die formale Definition eines Addierers. Ein *n*-Bit Addierer realisiert eine *n*-stellige Addition zweier Binärzahlen mit Eingangsübertrag.

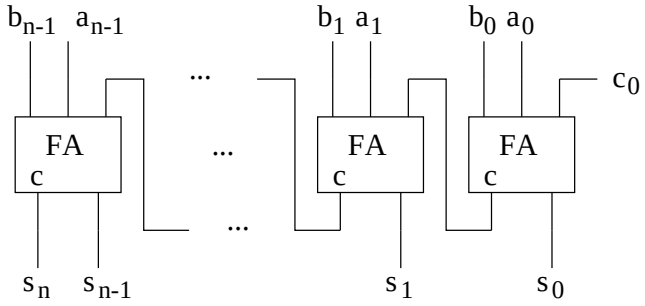
> Schaltfunktion $ADD_n : \{0,1\}^{2n+1} \longrightarrow \{0,1\}^{n+1}$ mit
>
> $ADD_n(a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}, c_0) = (s_0, \ldots, s_{n-1}, s_n)$
>
> wobei gilt:
>
> $\langle a_{n-1} \ldots a_0 \rangle_2 + \langle b_{n-1} \ldots b_0 \rangle_2 + \langle c_0 \rangle_2 = \langle s_n s_{n-1} \ldots s_0 \rangle_2$
>
> Analog zu den 1-Bit Addierern bezeichnet man s_{n-1} bis s_0 als Summenbits und s_n als Übertrags– bzw. 'carry'-Bit.

Insbesondere gibt es je Stelle nur ein Übertragsbit, welches sich gerade auf die nachfolgende Stelle auswirkt (d.h. eins im Sinn hat seinen Sinn).

Wir wollen im Folgenden auf formale Beweise der Korrektheit der Addierer verzichten. Der Leser sei jedoch gewarnt, man muss diese Beweise führen.

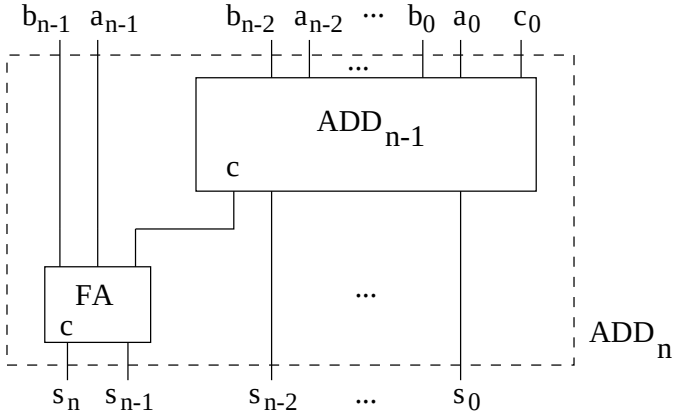
Ripple Carry Addierer (RCA)

Der erste Addierer arbeitet nach der Schulmethode.



Von rechts nach links werden die entsprechenden Bits und der Übertrag der vorhergehenden Stellen addiert. (Das Bild verdeutlicht auch, weshalb der Addierer Ripple Carry Addierer heißt: ripples sind kleine regelmäßige Wellen, auf deutsch müsste man Kräuseladdierer sagen.) Der Ripple Carry Addierer wird auch häufig Carry Chain Addierer genannt. Den Eingangsübertrag der gesamten Schaltung belegt man mit 0. Ist das oberste Bit der Summe gleich 1, so ist die Summe nicht mehr als *n*-Bit Zahl darstellbar.

Wir können den RCA auch mit folgender Konstruktion rekursiv realisieren:



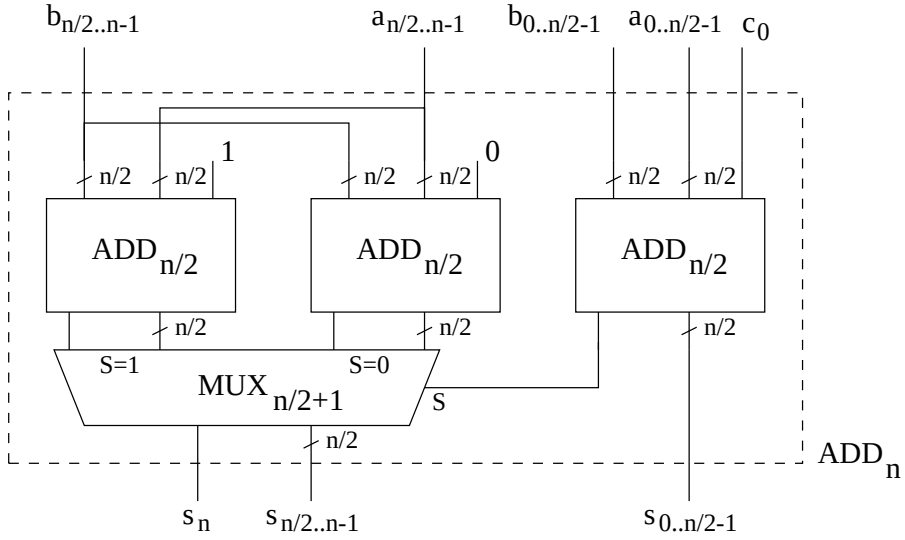
Ohne jetzt explizit die Formel der linearen Rekursion anzuwenden, erkennen wir, dass die Kosten und die Tiefe eines *n*-Bit RCAs sich zu folgenden Werten berechnen:

$$\begin{aligned} C(RCA_n) &= 5n \\ T(RCA_n) &= 3n \end{aligned}$$

Lassen wir wieder nur Zweierpotenzen für die Länge der Eingabezahlen zu, so können wir einen wesentlich schneller addierenden Schaltkreis entwerfen.

Conditional Sum Addierer (CSA)

Sei *n* im Folgenden eine Zweierpotenz.



Die Kosten des CSA berechnen sich gemäß der Formel

$$
\begin{aligned}
C(CSA_1) &= C(FA) = 5 \\
C(CSA_n) &= 3 \cdot C(CSA_{n/2}) + C(MUX_{n/2+1}) \\
&= 3 \cdot C(CSA_{n/2}) + 3(n/2+1) + 1 \\
&= 3 \cdot C(CSA_{n/2}) + \frac{3}{2}n + 4
\end{aligned}
$$

Was zu den folgenden Parametern für die Kosten und die Tiefe führt:

	C	T
a	3	1
b	2	2
c	5	3
$g(n)$	$\frac{3}{2}\ n + 4$	3

Damit erhalten wir (unter Verwendung der geometrischen Reihe und einiger Potenzrechenregeln):

$$
\begin{aligned}
C(CSA_n) &= a^{\log_b n} \cdot c + \sum_{i=0}^{\log_b n - 1} a^i \cdot g\left(\frac{n}{b^i}\right) \\
&= 3^{\log n} \cdot 5 + \sum_{i=0}^{\log n - 1} 3^i \cdot \left(\frac{3}{2} \cdot \frac{n}{2^i} + 4\right) \\
&= 3^{\log n} \cdot 5 + \sum_{i=0}^{\log n - 1} 3^i \cdot \frac{3}{2} \cdot \frac{n}{2^i} + \sum_{i=0}^{\log n - 1} 3^i \cdot 4 \\
&= 3^{\log n} \cdot 5 + \frac{3}{2}\, n \cdot \sum_{i=0}^{\log n - 1} \left(\frac{3}{2}\right)^i + 4 \cdot \sum_{i=0}^{\log n - 1} 3^i \\
&= 3^{\log n} \cdot 5 + \frac{3}{2}\, n \cdot \frac{\left(\frac{3}{2}\right)^{\log n} - 1}{\frac{3}{2} - 1} + 4 \cdot \frac{3^{\log n} - 1}{3 - 1} \\
&= 3^{\log n} \cdot 5 + 3n \cdot \left(\frac{3^{\log n}}{2^{\log n}} - 1\right) + 2 \cdot 3^{\log n} - 2 \\
&= 5 \cdot 3^{\log n} + 3 \cdot 3^{\log n} - 3n + 2 \cdot 3^{\log n} - 2 \\
&= 10 \cdot 3^{\log n} - 3n - 2 \\
&= 10 n^{\log 3} - 3n - 2
\end{aligned}
$$

$$
\begin{aligned}
T(CSA_n) &= a^{\log_b n} \cdot c + \sum_{i=0}^{\log_b n - 1} a^i \cdot g\left(\frac{n}{b^i}\right) \\
&= 1^{\log n} \cdot 3 + \sum_{i=0}^{\log n - 1} 1^i \cdot 3 \\
&= 3 + 3 \log n
\end{aligned}
$$

Er ist also um einiges teurer aber wesentlich schneller als der RCA. Wir können den CSA jedoch noch erheblich verbessern, wie der folgende Abschnitt erläutert.

Two Sum Addierer (TSA)

Wir haben beim CSA gesehen, dass es sich ausgezahlt hat, beide Möglichkeiten - also *ein* Übertrag von rechts bzw. *kein* Übertrag von rechts - vorzuberechnen.

Bauen wir mit dieser Methode gleich einen ganzen Addierer, der generell stets beide Summen berechnet. Dies scheint ein Mehraufwand zu bedeuten, wir werden aber sehen, dass der Addierer nicht zu teuer, aber sehr schnell ist.

Beginnen wir wieder mit einer 1-Bit Version:

$$\text{Schaltfunktion } TSA' : \{0,1\}^2 \longrightarrow \{0,1\}^4 \text{ mit}$$

$$TSA(a_0, b_0) = (s_0, s_1, t_0, t_1)$$

wobei gilt

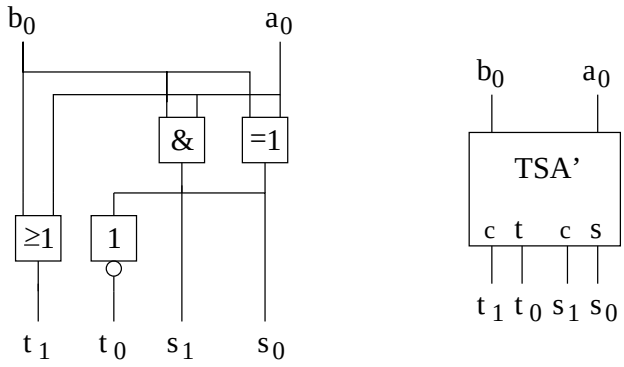
$$\begin{aligned} \langle a_0 \rangle + \langle b_0 \rangle + 0 &= \langle s_1 s_0 \rangle \\ \langle a_0 \rangle + \langle b_0 \rangle + 1 &= \langle t_1 t_0 \rangle \end{aligned}$$

Wertetabelle

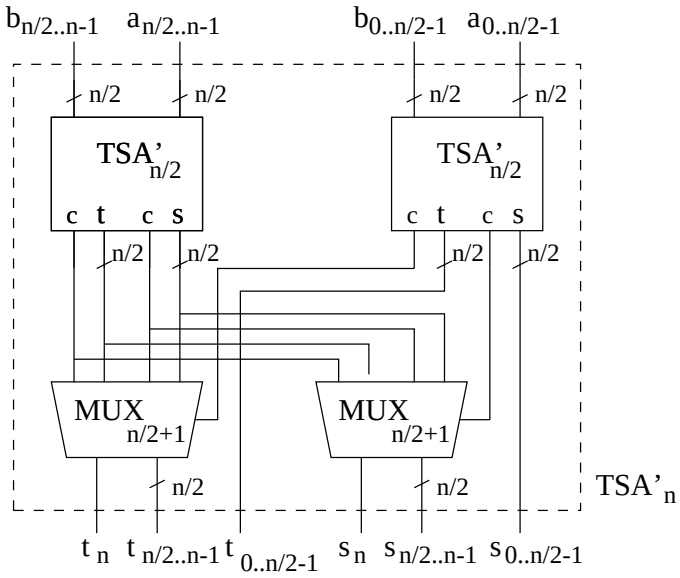
a_0	b_0	s_1	s_0	t_1	t_0
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	0	1	1

Und wir sehen sofort

$$
\begin{aligned}
s_1 &= a_0 b_0 \\
s_0 &= a_0 \oplus b_0 (= a_0 \overline{b_0} + \overline{a_0} b_0) \\
t_1 &= a_0 + b_0 \\
t_0 &= \overline{s_0}
\end{aligned}
$$



Die Kosten betragen also 4 und die Tiefe beträgt 2. Nun konstruieren wir rekursiv einen *n*-stelligen TSA'. Nehmen wir an, wir hätten zwei *n/2* TSA', die die beiden Summen für die obere und die untere Hälfte der Eingabezahlen berechnen. Die untere Hälfte der Ausgabe wird nur durch den unteren TSA' bestimmt. Die obere Hälfte der Ausgabe wählen wir entsprechend des entstehenden Übertrags an der unteren Hälfte aus.



Bestimmen wir wieder wie schon gewohnt die Kosten und die Tiefe der TSA' Addierer mithilfe der Rekursionsformeln. Die Konstruktion liefert folgende Formel für die Kosten:

$$\begin{aligned} C(TSA'_1) &= C(TSA') = 4 \\ C(TSA'_n) &= 2 \cdot C(TSA'_{n/2}) + 2 \cdot C(MUX_{n/2+1}) \\ &= 2 \cdot C(TSA'_{n/2}) + 2 \cdot (3 \cdot (n/2+1) + 1) \\ &= 2 \cdot C(TSA'_{n/2}) + 3n + 8 \end{aligned}$$

Die Parameter für die Rekursionsformeln sind also (bis auf die Konstante *c* ergeben sich die gleichen Parameter für die Tiefe wie oben beim CSA):

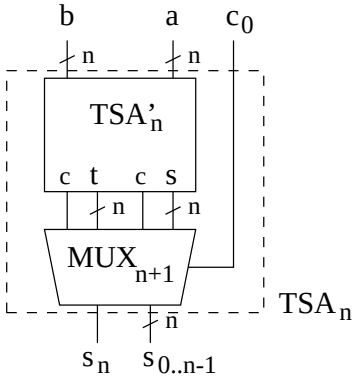
	C
a	2
b	2
c	4
$g(n)$	$3n+8$

Und damit

$$\begin{aligned} C(TSA'_n) &= a^{\log_b n} \cdot c + \sum_{i=0}^{\log_b n - 1} a^i \cdot g\left(\frac{n}{b^i}\right) \\ &= 2^{\log n} \cdot 4 + \sum_{i=0}^{\log n - 1} 2^i \cdot \left(3 \cdot \frac{n}{2^i} + 8\right) \\ &= 4n + 3n \log n + 8n - 8 \\ &= 3n \log n + 12n - 8 \end{aligned}$$

$$T(TSA'_n) = 3 \log n + 2$$

Da wir am Ende noch mithilfe des Eingangsübertrags die eigentlich gewollte Summe selektieren müssen, ergibt sich der gewünschte Addierer TSA aus einem TSA' durch Anbauen eines weiteren Multiplexers.

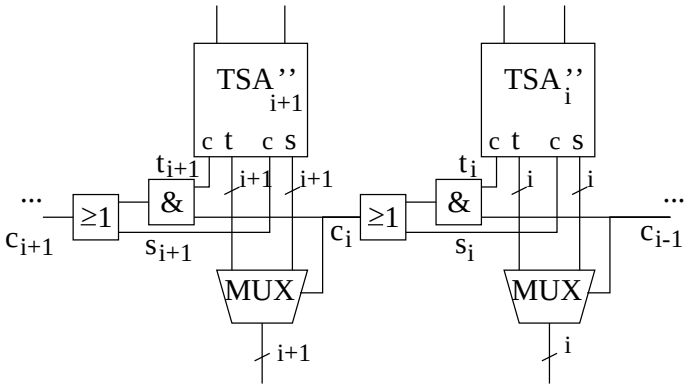


Und wir erhalten für die Kosten und die Tiefe letztendlich:

$$
\begin{aligned}
C(TSA_n) &= C(TSA'_n) + C(MUX_{n+1}) \\
&= 12n + 3n\log n - 8 + 3 \cdot (n+1) + 1 \\
&= 3n\log n + 15n - 4 \\
T(TSA_n) &= T(TSA'_n) + T(MUX_{n+1}) \\
&= 3\log n + 5
\end{aligned}
$$

Block Carry Addierer (BCA)

Wir wollen einen weiteren recht billigen Addierer kennen lernen, der nicht ganz so schnell wie obige CSA bzw. RCA ist, allerdings wesentlich billiger. Er arbeitet ebenfalls nach dem Zwei-Summen-Prinzip und nutzt den Trick, die zu addierenden Zahlen so in Blöcke einzuteilen, dass die Berechnung des Übertrags der vorhergehenden Blöcke gerade ungefähr so lange dauert wie die Berechnung der Summe des aktuellen Blocks. Man betrachte den folgenden Ausschnitt aus einem Schaltkreis.

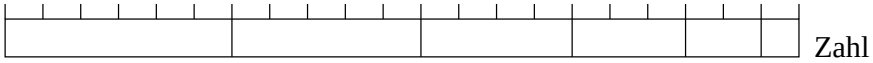


Das Selektsignal c_i für den Block $i+1$ berechnet sich als

$$c_i = s_i + t_i c_{i-1}$$

Warum? Nun, wir müssen für Block $i+1$ die Summe mit Übertrag t auswählen, falls bereits der Block i auf jeden Fall einen Übertrag generiert, d.h. $s_i = 1$, *oder* wir müssen für Block $i+1$ die Summe mit Übertrag t auswählen, falls der Block i einen Übertrag propagiert, d.h. $t_i = 1$ und es wird ein Übertrag vom Block $i-1$ geliefert, d.h. $c_{i-1} = 1$.

Wählen wir die Blockgröße so, dass deren Länge stets um 1 wächst, so erhalten wir folgende Einteilung der Zahlen. Die Tabelle gibt die Blockgröße, die gleich der Blocknummer ist, und die maximale Länge n der Zahl an.



Blockeinteilung

Blöcke	1	2	3	4	5	6	7	8	9	10	11
n	1	3	6	10	15	21	28	36	45	55	66

Für eine Stellenzahl n berechnet sich die Anzahl k der Blöcke zu:

$$k = \lceil \sqrt{2n + 0.25} - 0.5 \rceil \leq \lceil \sqrt{2n} \rceil$$

Dies sieht man wie folgt ein. Die Anzahl der Stellen ist gerade die Summe der Blockgrößen:

$$n = \sum_{i=1}^{k} i = \frac{k \cdot (k+1)}{2}$$

Um nun die Anzahl der Blöcke zu berechnen, lösen wir die quadratische Gleichung $n = k \cdot (k+1)/2$ und erhalten die obige (positive und aufgerundete) Lösung für k. Die rechte Abschätzung überprüft man leicht!

Wir wollen den BCA geschickt aufbauen, so dass die Tiefe in einem Block *i*, die den Selekteingang der Stufe *i+1* bestimmt, gerade ungefähr gleich der Tiefe des dortigen TSA'' ist. Es soll also gelten

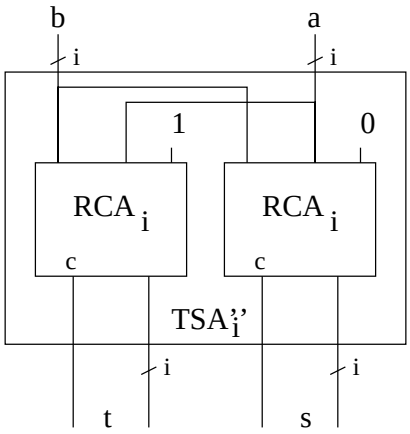
$$T(c_i) \leq T(TSA''_{i+1})$$

Nun gilt für die Tiefe dieses Selekteingangs

$$\begin{aligned} T(c_i) &= 2 + \max(T(TSA''_i), T(c_{i-1})) \\ &= 2 + T(TSA''_i) \end{aligned}$$

d.h. wenn wir als spezielle Two Sum Addierer TSA'' gerade einfache RCAs - einen für jede zu berechnende Summe - nehmen, haben wir die Bedingung erfüllt und es gilt:

$$\begin{aligned} T(c_i) &= T(TSA''_i) + 2 \\ &= T(RCA_i) + 2 \\ &= 3i + 2 \\ &< 3(i+1) \\ &= T(RCA_{i+1}) \\ &= T(TSA''_{i+1}) \end{aligned}$$



Damit ist aber die Tiefe eines BCA durch die Tiefe seines größten Blocks und des nachfolgenden Multiplexers bestimmt und wir erhalten:

$$\begin{aligned} T(BCA_n) &= T(TSA''_k) + T(MUX_k) \\ &= T(RCA_k) + T(MUX_k) \\ &= 3k + 3 \end{aligned}$$

Berechnen wir die Kosten.

Wir benötigen für jeden Block zwei RCA, einen Multiplexer, der die korrekte Summe auswählt und - außer für den letzten Block - die beiden Gatter für die Übertragsweitergabe. Es ergibt sich also

$$\begin{aligned} C(BCA_n) &= \sum_{i=1}^{k} (2 \cdot C(RCA_i) + C(MUX_i)) + 2 \cdot (k-1) \\ &= \sum_{i=1}^{k} (2 \cdot 5 \cdot i + 3 \cdot i + 1) + 2 \cdot (k-1) \\ &= \sum_{i=1}^{k} (13i + 1) + 2 \cdot (k-1) \\ &= 13k \cdot (k+1)/2 + k + 2 \cdot (k-1) \\ &= 0.5 \cdot (13k^2 + 19k) - 2 \end{aligned}$$

Mit obiger Abschätzung für *k* ergibt sich also abschließend

$$\begin{aligned} C(BCA_n) &\leq 13n + 9.5 \cdot \lceil \sqrt{2n} \rceil - 2 \\ T(BCA_n) &\leq 3 \cdot (\lceil \sqrt{2n} \rceil + 1) \end{aligned}$$

Carry Lookahead Addierer (CLA)

Der CLA ist theoretisch und auch in vielen Fällen praktisch der beste Addierer, insbesondere, wenn große Zahlen addiert werden sollen und Verdrahtung nicht dominierend wirkt.

Die Idee dieses Addierers besteht darin, parallel für alle Stellen des Addierers gleichzeitig zu berechnen, ob eine Stelle einen Übertrag generiert oder propagiert. Dies haben wir bereits beim BCA kennen gelernt. Man beobachtet nun, dass die Berechnung der beiden Signale carry-generate und carry-propagate für jede Stelle als assoziative Verknüpfung aufgefasst werden kann. Mithilfe eines parallel-prefix Schaltkreises können dann alle diese Signale mit Kosten linear in Anzahl der Stellen und Tiefe logarithmisch in Anzahl der Stellen berechnet werden.

Wir möchten an dieser Stelle jedoch nicht genau auf diesen wichtigen Addierer und das dahinterstehende Konzept eingehen, sondern lediglich Kosten und Tiefe mit angeben.

Zusammenfassung der Addierer

Folgende Tabelle fasst die Kosten und die Tiefe der verschiedenen Addierer einer Breite *n* zusammen:

Name	Abk.	Kosten	Tiefe
'ripple carry'	RCA	$5n$	$3n$
'conditional sum'	CSA	$\approx 10n^{1.585} - 3n - 2$	$3\log n + 3$
'two sum'	TSA	$3n\log n + 15n - 4$	$3\log n + 5$
'block carry'	BCA	$\approx 13n + 9.5\sqrt{2n} - 2$	$\approx 3\sqrt{2n} + 3$
'carry look ahead'	CLA	$9n + 3$	$4\log n + 2$

Damit ergibt sich für 32-Bit Addierer:

Addierer	Kosten	Tiefe
RCA	160	96
CSA	2342	18
TSA	956	20
BCA	490	27
CLA	291	22

Man beachte, dass diese Formeln so nur für das angegebene Kostenmaß (nämlich pro verwendetem Gatter 1) gelten.

Die verschiedenen Konzepte der Addierer können je nach Anforderungen der zu realisierenden Schaltung auch kombiniert angewendet werden. So kann man sich z.B. vorstellen aus 4-Bit Addierern, die schon als sogenannte Macros auf einem Chip vom Hersteller vorliegen, einen 32-Bit CSA zusammenzubauen, der dabei jedoch nur 15 der Macros verwendet und den Rest nach dem Zwei-Summen-Prinzip über Multiplexer verdrahtet.

Subtrahierer

Als erstes sei bemerkt: wir können einen Subtrahierer mit der gleichen Methodik wie einen Addierer aufbauen: Realisierung eines Halbsubtrahierers, Realisierung eines Vollsubtrahierers, der den borrow (den, den man im Sinn hat) berücksichtigt; und dann z.B. Realisierung eines Ripple Borrow
Subtrahieriers, der genau nach der Schulmethode arbeitet.

Versuchen Sie dies zu tun! Beachten Sie, dass wir noch keine besondere Kennzeichnung der negativen Zahlen festgelegt haben. Bis jetzt argumentierten wir einfach mit Betrag und Vorzeichen!

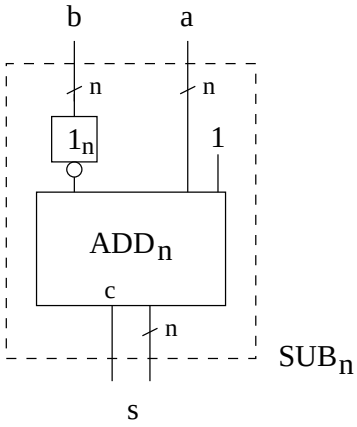
Wir könnten also Zahlen in Darstellung mit Betrag und Vorzeichen addieren bzw.\ subtrahieren, was aber zwei Einheiten und eine spezielle Auswahlschaltung erforderlich machen würde.

Es gibt aber auch eine geschicktere Methode unter Verwendung der Darstellung im Zweierkomplement.

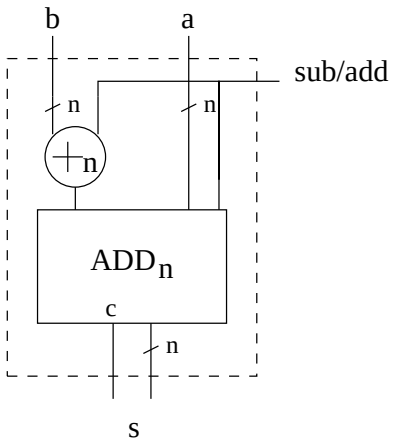
Seien a und b zwei Zahlen im Bereich $\{-2^{n-1}, \ldots, 2^{n-1}-1\}$, d.h. sie sind im Zweierkomplement als $[a]$ und $[b]$ mit n Bits darstellbar.
Wir wissen, dass gilt: $a - b = a + (-b)$ und von oben: $-[b] = [\overline{b}] + 1$. Wir können also die Subtraktion auf eine Addition zurückführen:

$$\begin{aligned} [a] - [b] &= [a] + (-[b]) \\ &= [a] + ([\overline{b}] + 1) \\ &= [a] + [\overline{b}] + 1 \end{aligned}$$

Mit anderen Worten, um zu subtrahieren, invertieren wir *b* bitweise, addieren *a* wie gewohnt und setzen dabei den Eingangsübertrag auf 1. Nun sehen wir auch ein, weshalb wir diesen Eingangsübertrag bei obigen Addierern stets berücksichtigt haben.



Ersetzen wir die NOT-Gatter durch XOR-Gatter, so erhalten wir einen Schaltkreis, der im Zweierkomplement sowohl addieren als auch subtrahieren kann.



Weitere kurz angesprochene Themen waren:

- Ringzähler unter Ausnutzung der Modulo-Darstellung
- Integration des Zero-Testers in die Addierschaltung

CVS: Id

formella@cs.uni-sb.de

http://www-wjp.cs.uni-sb.de/~formella/izfp.html
