11.2 Subtrahierer Multiplizierer ALU

Subtrahierer

Wegen $-[b] = [\overline{b}] + 1$ kann $[a] - [b]$ zurückgeführt

werden auf $[a] + [\overline{b}] + 1$.

→ Schaltkreis für Subtrahierer aus Addiererschaltkreis

→ kombinierter Addierer/Subtrahierer

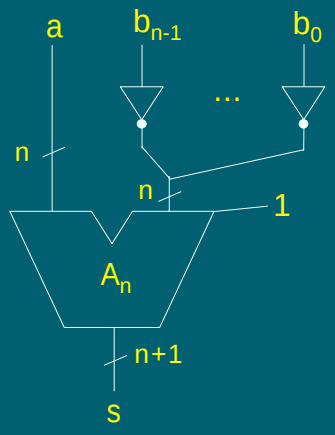
Beispiel

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

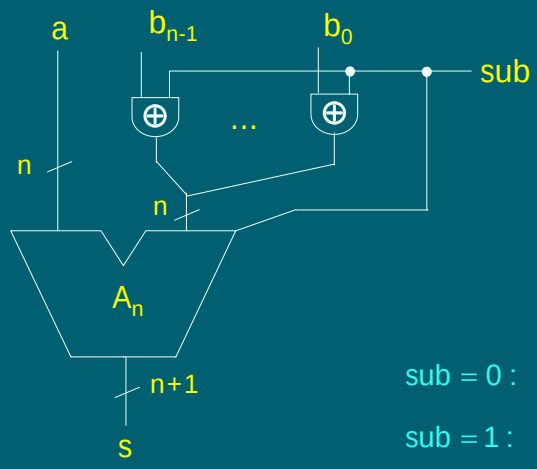
$$\begin{array}{cccc} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ & & & 1 \\ \hline 1 & 1 & 1 & 1 \end{array}$$

$= (-1)_{10}$

Schaltbild eines Subtrahierers



Schaltbild für einen Addierer/Subtrahierer



$b_i \oplus 0 = b_i$

$b_i \oplus 1 = \overline{b_i}$

sub = 0 : $[a] + [b] + 0$

sub = 1 : $[a] + [\overline{b}] + 1 = [a] - [b]$

Multiplizierer

Gesucht: Schaltkreis zur Multiplikation zweier Binärzahlen

$<a_{n-1}, \ldots, a_0>$, $<b_{n-1}, \ldots, b_0>$

Beispiel:

$(\underbrace{110}_{6_{10}}) \cdot (\underbrace{101}_{5_{10}})$

			1	1	0
		0	0	0	
	1	1	0		
1	1	1	1	0	

$= 30_{10}$

Allgemeines zum Multiplizierer

Wieviele Stellen werden für das Ergebnis benötigt?

$< a > \cdot < b > \leq$

$$(2^n - 1) \cdot (2^n - 1) = 2^{2n} - 2^{n+1} + 1 \leq 2^{2n} - 1$$

Also: 2n Stellen zur Multiplikation von Zweierkomplementzahlen

Vorgehen bei der Multiplikation

- Multipliziere die Beträge der Zahlen
- Bestimme das Vorzeichen des Produkts
- Setze das Endergebnis zusammen

Definition 2.4.:

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

mul_n: $\{0,1\}^{2n} \rightarrow \{0,1\}^{2n}$ mit

$mul_n(a_{n-1}, \ldots, a_0, b_{n-1}, \ldots, b_0) = (p_{2n-1}, \ldots, p_0)$ mit

$<p_{2n-1}, \ldots, p_0> = <a> \cdot <b>$

$$<a> \cdot <b> = <a> \cdot \sum_{i=0}^{n-1} b_i \cdot 2^i = \sum_{i=0}^{n-1} <a> \cdot b_i \cdot 2_i$$

Die Multiplikationsmatrix

$$\begin{pmatrix} pp_0 \\ pp_1 \\ \vdots \\ pp_{n-1} \end{pmatrix} = \begin{pmatrix} 0 & 0 & \ldots & 0 & 0 & a_{n-1}b_0 & a_{n-2}b_0 & \ldots & a_1b_0 & a_0b_0 \\ 0 & 0 & \ldots & 0 & a_{n-1}b_1 & a_{n-2}b_1 & a_{n-3}b_1 & \ldots & a_0b_1 & 0 \\ \vdots & & & & \vdots & \vdots & \vdots & & & \vdots \\ 0 & a_{n-1}b_{n-1} & \ldots & a_2b_{n-1} & a_1b_{n-1} & a_0b_{n-1} & 0 & \ldots & 0 & 0 \end{pmatrix}$$

Realisierung der Multiplikationsmatrix mit n^2 AND-Gattern (und n^2 Konstanten 0).

Daraus entstehende Aufgabe:

Schnelle Addition von n Partialprodukten der Länge 2n.

Mit CLAs lösbar mit Kosten $O(n^2)$, Tiefe $O(n \log(n))$ bei *linearem Aufsummieren* der Partialprodukte $((((pp_0+pp_1)+pp_2)+\ldots)+pp_{n-1})$,
$O(\log^2 n)$ bei *baumartigem Zusammenfassen* der Partialprodukte.

Verbesserung

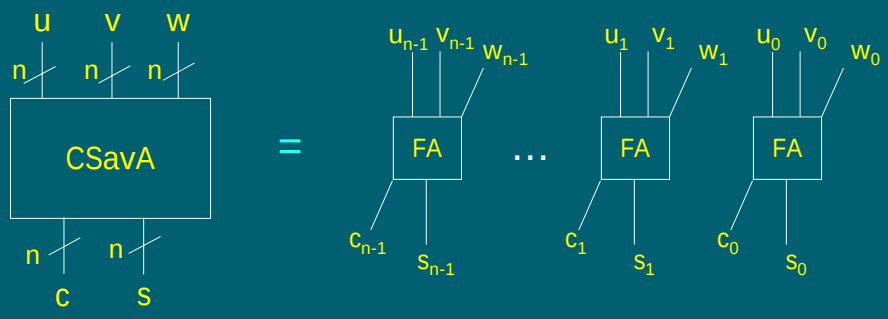
Verwende Carry-Save-Addierer.

Reduktion von 3 Eingabewerten u,v,w zu zwei Ausgabewerten s,c mit <u> + <v> + <w> = <s> + <c>

	u_{n-1}	u_{n-2}	…	u_2	u_1	u_0
	v_{n-1}	v_{n-2}		v_2	v_1	v_0
	w_{n-1}	w_{n-2}		w_2	w_1	w_0
c_{n-1}	c_{n-2}	c_{n-3}	…	c_1	c_0	0
0	s_{n-1}	s_{n-2}		s_2	s_1	s_0

Gelöst durch Nebeneinandersetzen von Volladdierern (kein Carry-Chain!).

Carry-Save Addierer



Bemerkung zum Aufbau des CSavA

Speziell bei Partialprodukten:

Reduziere 3 2n-Bit-Zahlen zu 2 2n-Bit-Zahlen

$pp_{0,2n-1}$	...	$pp_{0,1}$	$pp_{0,0}$
$pp_{1,2n-1}$	...	$pp_{1,1}$	$pp_{1,0}$
$pp_{2,2n-1}$	...	$pp_{2,1}$	$pp_{2,0}$
c_{2n-2}	...	c_0	0
s_{2n-1}	...	s_1	s_0

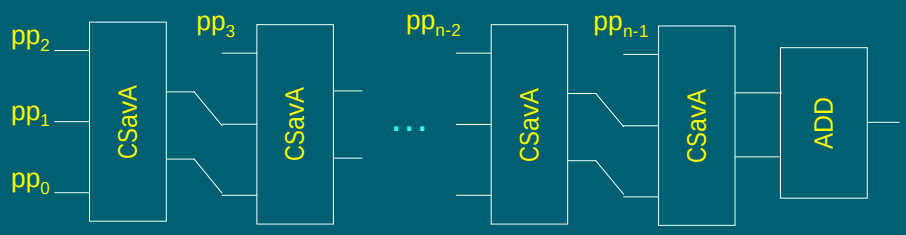
($c_{2n-1} = 0$ → Carry-Ausgang des letzten FA nicht verwendet)

1. Serielle Lösung:

- Hintereinanderschalten von n-2 CSA-Addierern der Länge 2n

  → Fasse n Partialprodukte zu 2 2n-Bit-Worten zusammen
- Addiere die 2n-Bit-Worte mit CLA
- *siehe Abb. einer Addierstufe*
- Kosten $O(n^2)$, Tiefe $O(n)$

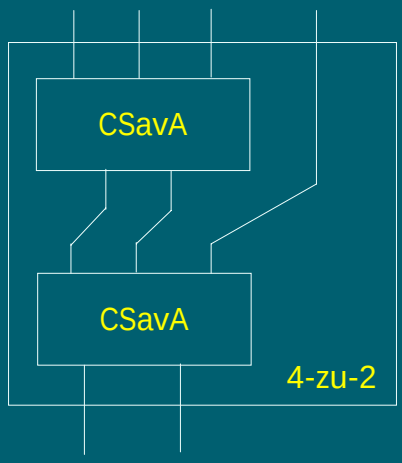
Addierstufe im Multiplizierer



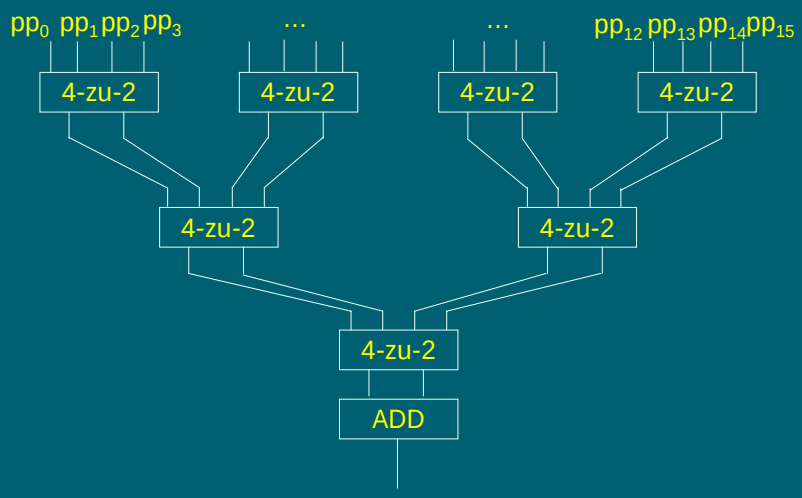
2. Baumartige Lösung:

- Neue Grundzelle zur Reduktion von 4 2n-Bit Eingabeworten zu zwei Ausgabeworten, bestehend aus 2 CSAs (*siehe Abb. zur Reduktionszelle*)
- Baumartiges Zusammenfassen der Partialprodukte mit 4-zu-2-Bausteinen zu 2 2n-Bit-Worten
- Addiere die 2n-Bit-Worte mit CLA
- *siehe Abb. der Addierstufe mit log. Zeit*
- Kosten $O(n^2)$, Tiefe $O(\log n)$

4-zu-2 Reduktions-Grundzelle



Addierstufe des log-Zeit-Multiplizierers für 16 Bit



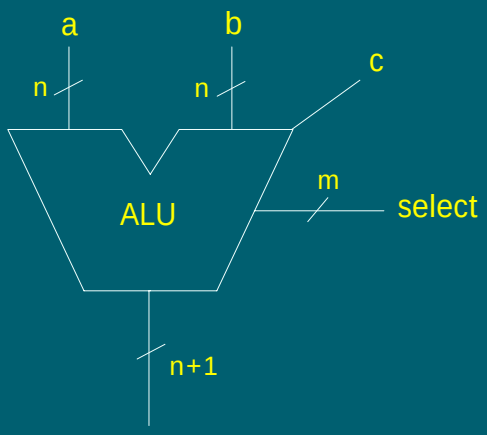
Aufbau einer ALU

ALU = Arithmetic Logic Unit zur Berechnung von arithmetischen und logischen Basisoperationen

n-Bit-ALU mit:

- 2 n-Bit-Operanden a, b, Eingangscarry c
- m-Bit select-Eingang, der auswählt, welche Funktion ausgeführt wird
- (n+1)-Bit-Ausgang

Schaltzeichen einer n-Bit-ALU



Beispiel zum select-Eingang

Hier: 8 Funktionen, d.h. 3-Bit select-Eingang

Funktionsnummer			ALU-Funktion
s_2	s_1	s_0	
0	0	0	0 ... 0
0	0	1	[b] – [a]
0	1	0	[a] – [b]
0	1	1	[a] + [b] + c
1	0	0	$a \oplus b = (a_{n-1} \oplus b_{n-1}, \ldots, a_0 \oplus b_0)$
1	0	1	$a \vee b = (a_{n-1} \vee b_{n-1}, \ldots, a_0 \vee b_0)$
1	1	0	$a \wedge b = (a_{n-1} \wedge b_{n-1}, \ldots, a_0 \wedge b_0)$
1	1	1	1 ... 1

Mögliche Realisierungen einer ALU

1. Möglichkeit:

   Realisiere Fkt. $f_0, \ldots, f_{2^m-1}$ getrennt durch SK_{f_i} für f_i, dann Auswahl durch verallgemeinerten Multiplexer

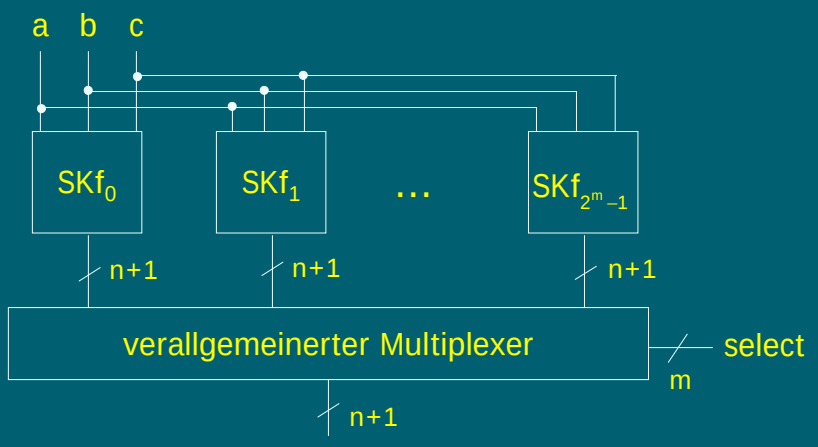
   (*siehe Abb. der möglichen Realisierung*)

2. Möglichkeit:

   gemeinsame Behandlung ähnlicher Funktionen

   (*siehe Abb. zur Schaltungsrealisierung*)

Mögliche Realisierung der n-Bit ALU



Schaltungsrealisierung der n-Bit ALU