Exkurs – Multiplizierer

Albert-Ludwigs-Universität Freiburg

Prof. Dr. Christoph Scholl
Institut für Informatik

WS 2015/16

Multiplizierer

$63 \cdot 421$

$$\begin{array}{rl} 63 & \leftarrow 63 \cdot 1 \\ 126 & \leftarrow + 63 \cdot 2 \cdot 10 \\ 252 & \leftarrow + 63 \cdot 4 \cdot 100 \\ \hline 26523 & \end{array}$$

$$= 63 \cdot [\underbrace{1 + 2 \cdot 10 + 4 \cdot 100}_{421}]$$

- **Gesucht**: Schaltkreis zur Multiplikation zweier Binärzahlen $< a_{n-1}, \ldots, a_0 >, < b_{n-1}, \ldots, b_0 >$.

- **Beispiel**:

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

$$\begin{array}{cccccc} & & & 1 & 1 & 0 \\ + & & 0 & 0 & 0 & \\ + & 1 & 1 & 0 & & \\ \hline & 1 & 1 & 1 & 1 & 0 \end{array}$$

$$= 30_{10}$$

Allgemeines zum Multiplizierer

- Wieviele Stellen werden für das Ergebnis benötigt?

$$\begin{aligned} <a> \cdot <b> &\leq (2^n - 1) \cdot (2^n - 1) \\ = 2^{2n} - 2^{n+1} + 1 &\leq 2^{2n} - 1 \end{aligned}$$

für $n \geq 0$

- **Also**:
  $2n$ Stellen zur Multiplikation von Binärzahlen.

Vorgehen bei der Multiplikation

Bei Zahlen aus $\mathbb{Z}$:

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

$\Rightarrow$ Betrachte im Folgenden nur Binärzahlen!

4

n-Bit-Multiplizierer

Definition

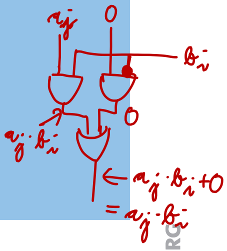
Ein n-Bit-Multiplizierer ist ein Schaltkreis, der die folgende Funktion berechnet:

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

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

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

$$< a > \cdot < b > = < a > \cdot \sum_{i=0}^{n-1} b_i \cdot 2^i = \sum_{i=0}^{n-1} \underbrace{< a > \cdot b_i \cdot 2^i}_{\text{„Partialprodukte“}}$$



$pp_0 = \langle a \rangle \cdot b_0 \cdot 2^0$

$pp_1 = \langle a \rangle \cdot b_1 \cdot 2^1$

$\vdots$

Die Multiplikationsmatrix

$$\begin{pmatrix} pp_0 \\ pp_1 \\ \vdots \\ pp_{n-1} \end{pmatrix} =$$

$\langle a \rangle \cdot b_0 \cdot 2^0$ $\quad \langle a \rangle \cdot b_1 \cdot 2^1$

$$\begin{pmatrix}
0 & 0 & \dots & 0 & 0 & a_{n-1} b_0 & a_{n-2} b_0 & \dots & a_1 b_0 & a_0 b_0 \\
0 & 0 & \dots & 0 & a_{n-1} b_1 & a_{n-2} b_1 & a_{n-3} b_1 & \dots & a_0 b_1 & 0 \\
\vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\
0 & a_{n-1} b_{n-1} & \dots & a_2 b_{n-1} & a_1 b_{n-1} & a_0 b_{n-1} & 0 & \dots & 0 & 0
\end{pmatrix}
\begin{array}{l} pp_0 \\ pp_1 \\ \vdots \\ pp_{n-1} \end{array}$$

$\widehat{=} \langle a \rangle \cdot b_{n-1} \cdot 2^{n-1}$

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

Daraus entstehende Aufgabe:

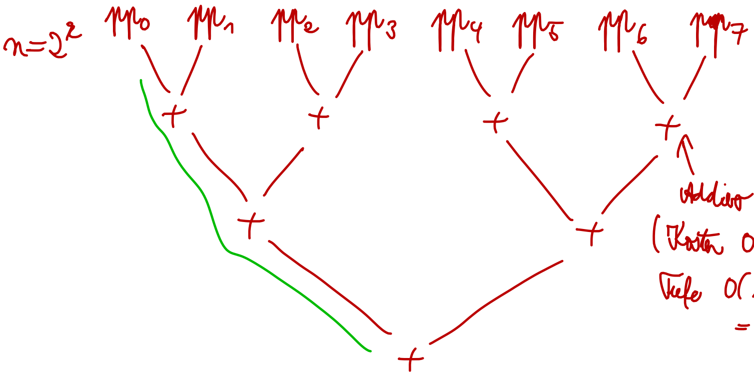
- Schnelle Addition von n Partialprodukten der Länge $2n$.

Carry-Lookahead-Addierer: $C(CLA_n) \in O(n)$, $depth(CLA_n) \in O(\log n)$

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

$n-1$ CLA_{2n} $\Rightarrow$ Kosten $\in O(n^2)$, $n-1$ CLA_{2n} hintereinander $\Rightarrow$ $n-1 \cdot depth(CLA_{2n})$ $\Rightarrow \in O(n \cdot \log n)$

Tiefe $O(\log(2n) \cdot \log(n)) = O(\log^2(n))$ bei *baumartigem Zusammenfassen* der Partialprodukte.



Addierer der Bitbreite $2n$
(Kosten $O(2n) = O(n)$
Tiefe $O(\log 2n) = O(\log n + 1)$
$= O(\log n)$)

$n = 2^k$ Partialprodukte zu addieren $\Longrightarrow$ k Addierer hintereinander

Bei allgemeinem n: $\lceil \log n \rceil$ Addierer in Folge hintereinander.

$\Longrightarrow$ Insgesamt Tiefe: $\lceil \log n \rceil \cdot$ logarithmische Tiefe für Addiererbaum $\rightsquigarrow$ Tiefe $O(\log^2 n)$

$\rightsquigarrow$ Tiefe $O(\log^2 n)$ für kompletten Multiplizierer, da Partialprodukte erzeugen nur Tiefe O

$\Longrightarrow$ Kosten: $O(n^2)$ für Addiererbaum ($n-1$ Addierer mit Kosten $O(n)$),

insgesamt für Multiplizierer: Kosten $O(n^2)$, da Mult.matrix auch $O(n^2)$ braucht

Verbesserung

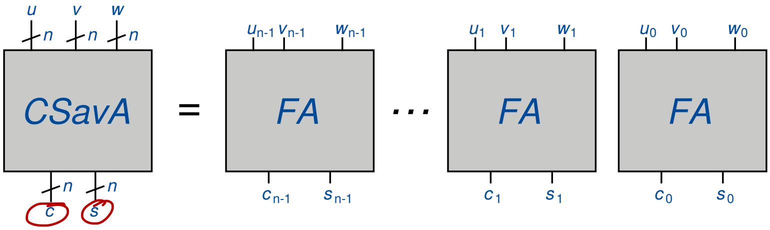
- Verwende Carry-Save-Addierer.
- Reduktion von 3 Eingabeworten u, v, w zu zwei Ausgabeworten s, c mit $<u> + <v> + <w> = <s> + <c>$.

$$
\begin{array}{ccccccc}
 & u_{n-1} & u_{n-2} & \dots & u_2 & u_1 & u_0 \\
 & v_{n-1} & v_{n-2} & \dots & v_2 & v_1 & v_0 \\
 & w_{n-1} & w_{n-2} & \dots & w_2 & w_1 & w_0 \\
\hline
c_{n-1} & c_{n-2} & c_{n-3} & \dots & c_1 & c_0 & 0 \\
0 & s_{n-1} & s_{n-2} & \dots & s_2 & s_1 & s_0
\end{array}
$$

- Gelöst durch Nebeneinandersetzen von Volladdierern (keine Carry-Chain!)

9

Carry-Save-Addierer (CSavA)



Bemerkung zum Aufbau des CSavA

- Speziell bei Partialprodukten:
  Reduziere 3 2n-Bit-Zahlen zu 2 2n-Bit-Zahlen.

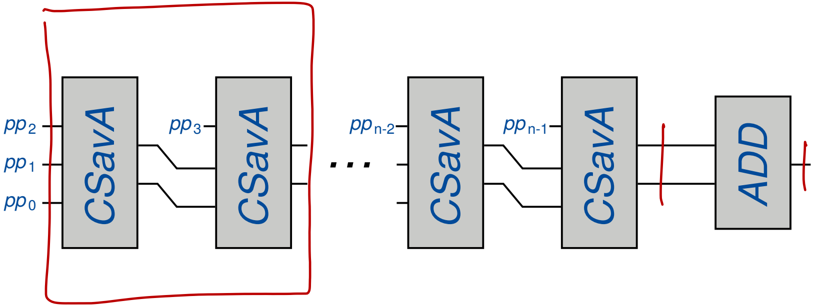
$$
\begin{array}{lcccccl}
 & pp_{0,2n-1} & \cdots & pp_{0,1} & pp_{0,0} & \leftarrow & pp_0 \\
 & pp_{1,2n-1} & \cdots & pp_{1,1} & pp_{1,0} & \leftarrow & pp_1 \\
 & pp_{2,2n-1} & \cdots & pp_{2,1} & pp_{2,0} & \leftarrow & pp_2 \\
\hline
\cancel{c_{2n-1}} & c_{2n-2} & \cdots & \underline{c_0} & \underline{0} & & \\
\cancel{\text{'' } 0} & s_{2n-1} & \cdots & s_1 & \underline{s_0} & &
\end{array}
$$

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

1. Serielle Lösung

- Hintereinanderschalten von n-2 CSav-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 $O(n)$

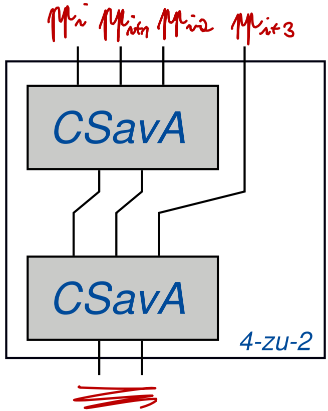
Addierstufe im Multiplizierer



2. Baumartige Lösung

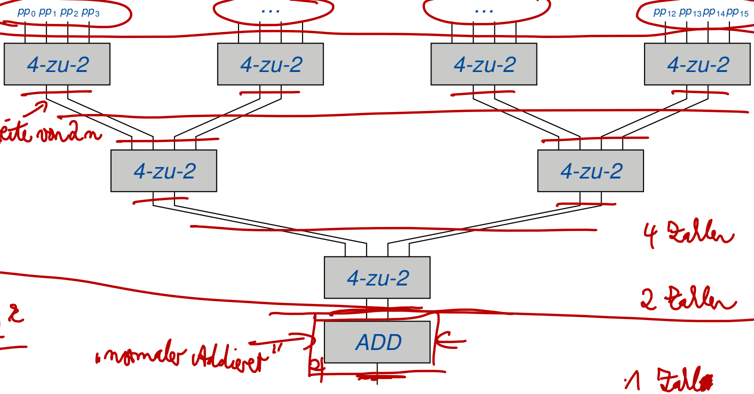
- Neue Grundzelle zur Reduktion von 4 2n-Bit-Eingabeworten zu zwei Ausgabeworten, bestehend aus 2 CSAs (*siehe Abb. zur Reduktionszelle*).
- Baumartiges Zusammenfassen der Partialprodukte mit 4-zu-2-Bausteinen zu 2 2n-Bit Worten.
- Addiere die 2n-Bit-Worte mit CLA.
- *siehe Abb. der Addierstufe mit log.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

4-zu-2 Reduktions-Grundzelle



Addierstufe des log-Zeit-Multiplizierers für 16 Bit

16 Partialprodukte



Bitbreite von 2n

8 Zahlen

4 Zahlen

2 Zahlen

1 Zahl

ℓ

$n = 2^\ell$

„normaler Addierer“

Tiefe des Addiererbaums: 1) $\ell - 1$ „Stufen“ von 4-2-Bausteinen, Tiefe eines 4-zu-2-Bausteins: $2 \cdot depth(FA) = 6$

$\Longrightarrow 6(\ell-1)$ für 4-2-Reduktionen, $6 \cdot (\ell - 1) \in O(\log n)$

2) $depth(ADD) = O(\log n)$

16

UNI FREIBURG

$\Longrightarrow$ insgesamt Tiefe von $O(\log n)$!

Kosten: 1) linear viele 4-zu-2-Zellen | jeweils lineare Kosten $\Rightarrow$ $O(n^2)$ Gatter

2) lineare Kosten für CLA_{2n} ($O(n)$ Gatter)

$\overset{(1)+2)}{\Longrightarrow}$ $\underline{O(n^2) \text{ Gatter}}$ insgesamt