Einführung in die Technische Informatik

Prof. Jian-Jia Chen | Dr.-Ing. Christian Hakert | Dr. rer. nat. Marcus Völker

RWTH Aachen University

Kapitel 4: Rechnerarithmetik

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Einleitung

Rechnen zu können ist für Computer zentral

- Grundlegende Rechenoperationen sollten durch die Hardware unterstützt werden.

Betrachtete Rechenoperationen

- Addition
- Subtraktion
- Multiplikation
- **Keine Division**

Betrachtete Datentypen

- Natürliche Zahlen
- Ganze Zahlen
- Rationale Zahlen (IEEE 754-1985)

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Abschnitt 4.1

Addition natürlicher Zahlen

- Halbaddierer und Volladdierer
- Ripple-Carry-Addierer
- Carry-Look-Ahead-Addierer
- Fan-In

Addition natürlicher Zahlen

Schulalgorithmus für die Addition von Dezimalzahlen

1. Summand		9	3	8	9	9	8	9
2. Summand			9	7	9	8	9	8
Übertrag	1	1	1	1	1	1	1	
Summe	1	0	3	6	9	8	8	7

Übertragung auf Binärzahlen

1. Summand		1	1	0	0	1	1	1
2. Summand		1	0	0	1	0	1	1
Übertrag	1			1	1	1	1	
Summe	1	0	1	1	0	0	1	0

Addition natürlicher Zahlen

Beobachtung zu den Überträgen

- $1 + 1$ erzeugt einen Übertrag
- $0 + 0$ eliminiert einen vorhandenen Übertrag
- $0 + 1$ und $1 + 0$ reichen einen vorhandenen Übertrag weiter
- Übertrag ist höchstens 1

Kann man Addition als Boolesche Funktion ausdrücken?

- **Summenbit** s und **Übertrag** c (*engl. carry*)
- $f_{HA} : B^2 \to B^2$
- f_{HA} realisiert die Forderungen zum Teil

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

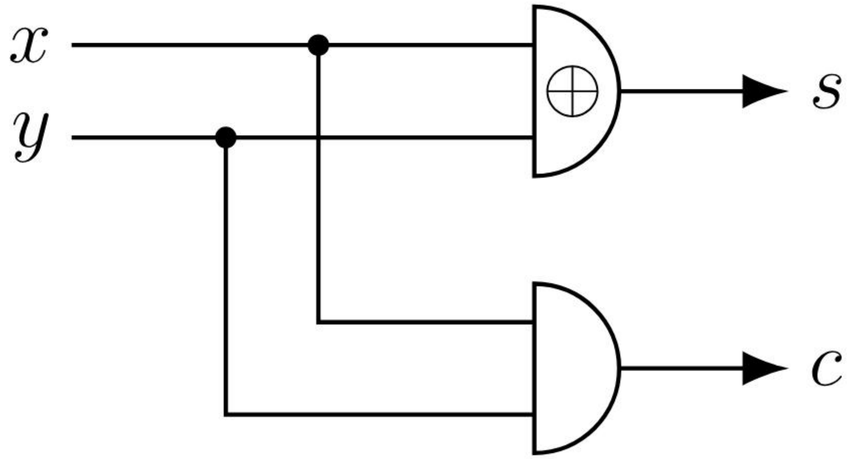
Halbaddierer

f_{HA} realisiert einen sog. **Halbaddierer**

$f_{HA} : B^2 \to B^2$

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Schaltung für den Halbaddierer: $c = x \wedge y$ $s = x \oplus y$



Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Volladdierer

Halbaddierer

- Berechnet Übertrag aus der Addition
- Berücksichtigt keinen Übertrag, der schon vorher entstanden ist

Volladdierer

- $f_{VA} : B^3 \to B^2$
- Berechnet **Übertrag** c aus der Addition
- Berücksichtigt **Übertrag** c_{alt}, der schon vorher entstanden ist

c_{alt}	x	y	c	s
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

Beobachtung für s

- $s = 1 \Leftrightarrow$ Anzahl der Einsen ungerade
- $s = c_{alt} \oplus x \oplus y$

Beobachtung für c

- $c = 1 \Leftrightarrow$ Anzahl Einsen ≥ 2
- DNF: $c = \overline{c_{alt}}xy \vee c_{alt}\overline{x}y \vee c_{alt}x\overline{y} \vee c_{alt}xy$
- Resolution:
  - $\overline{c_{alt}}xy \vee c_{alt}xy = xy$
  - $c_{alt}xy \vee c_{alt}\overline{x}y = c_{alt}y$
  - $c_{alt}xy \vee c_{alt}x\overline{y} = c_{alt}x$
- $\boldsymbol{c = xy \vee c_{alt}y \vee c_{alt}x}$

c_{alt}	x	y	c	s
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

- Verbesserte Realisierung für c
- Durch Wiederverwenden von Teilergebnissen

Es gilt bisher: $c = xy \vee c_{alt}y \vee c_{alt}x$

Wir müssen die Belegungen für $(c_{alt}, x, y) \in \{(011), (101), (110), (111)\}$ realisieren

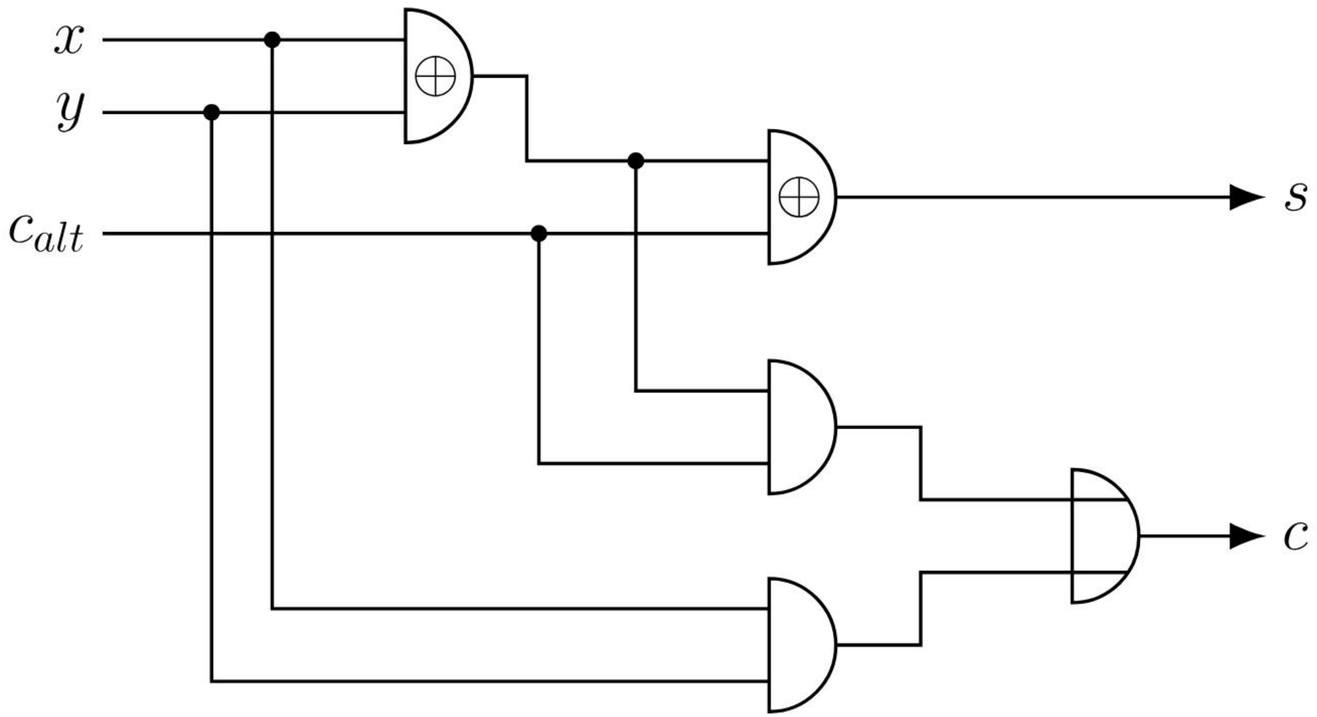
- $x \oplus y = 1 \Leftrightarrow$ genau eine 1 in x, y
- Also $c_{alt}(x \oplus y)$ realisiert c für $\{(101), (110)\}$
- Fehlen noch die Belegungen $\{(011), (111)\}$, realisiert durch xy

$\Rightarrow c = c_{alt}(x \oplus y) \vee xy$

c_{alt}	x	y	c	s
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

Schaltnetz Volladdierer

- $s = c_{alt} \oplus x \oplus y$
- $c = c_{alt}(x \oplus y) \vee xy$
- Größe 5
- Tiefe 3

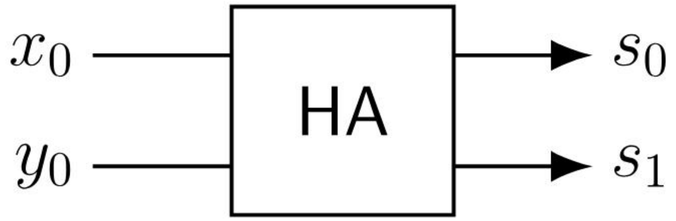


c_{alt}	x	y	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

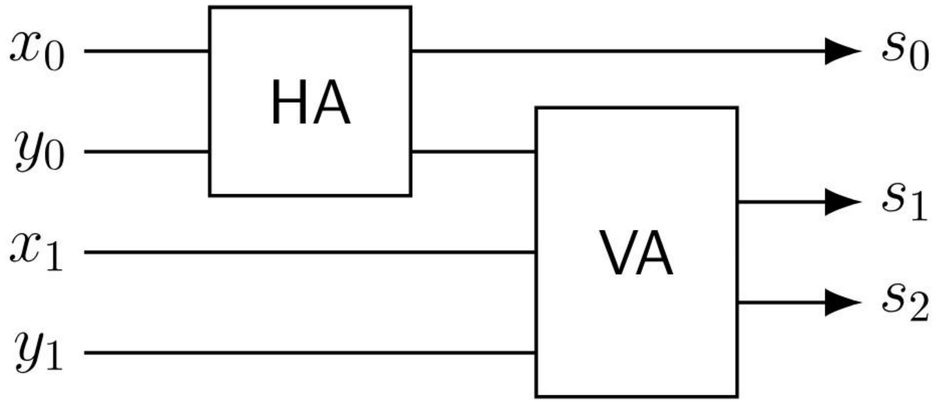
Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Vollständiger Addierer

$$\begin{array}{cccc} & & & x_0 \\ + & & & y_0 \\ \hline & & s_1 & s_0 \end{array}$$



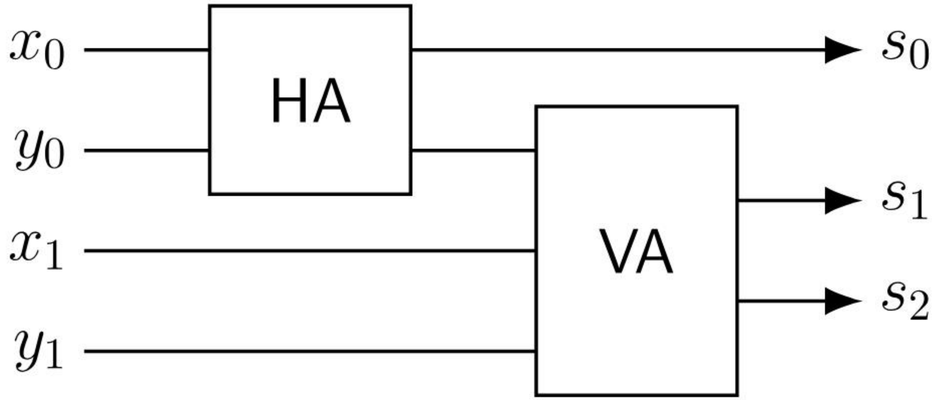
$$\begin{array}{cccc} & & x_1 & x_0 \\ + & & y_1 & y_0 \\ \hline & s_2 & s_1 & s_0 \end{array}$$



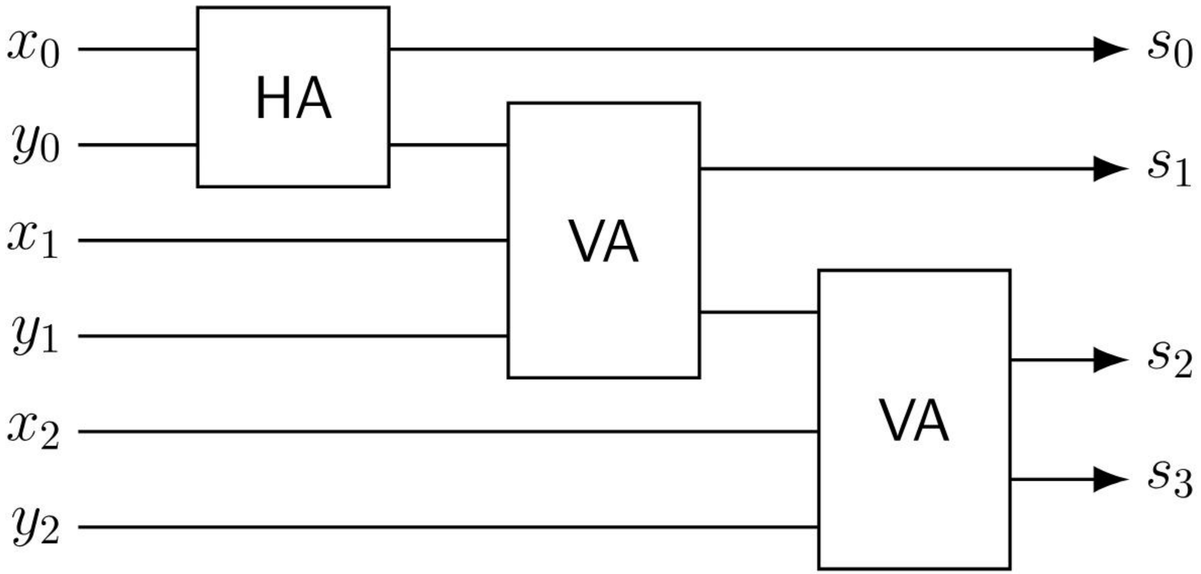
Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Vollständiger Addierer

$$
\begin{array}{cccc}
 & & x_1 & x_0 \\
+ & & y_1 & y_0 \\
\hline
 & s_2 & s_1 & s_0
\end{array}
$$

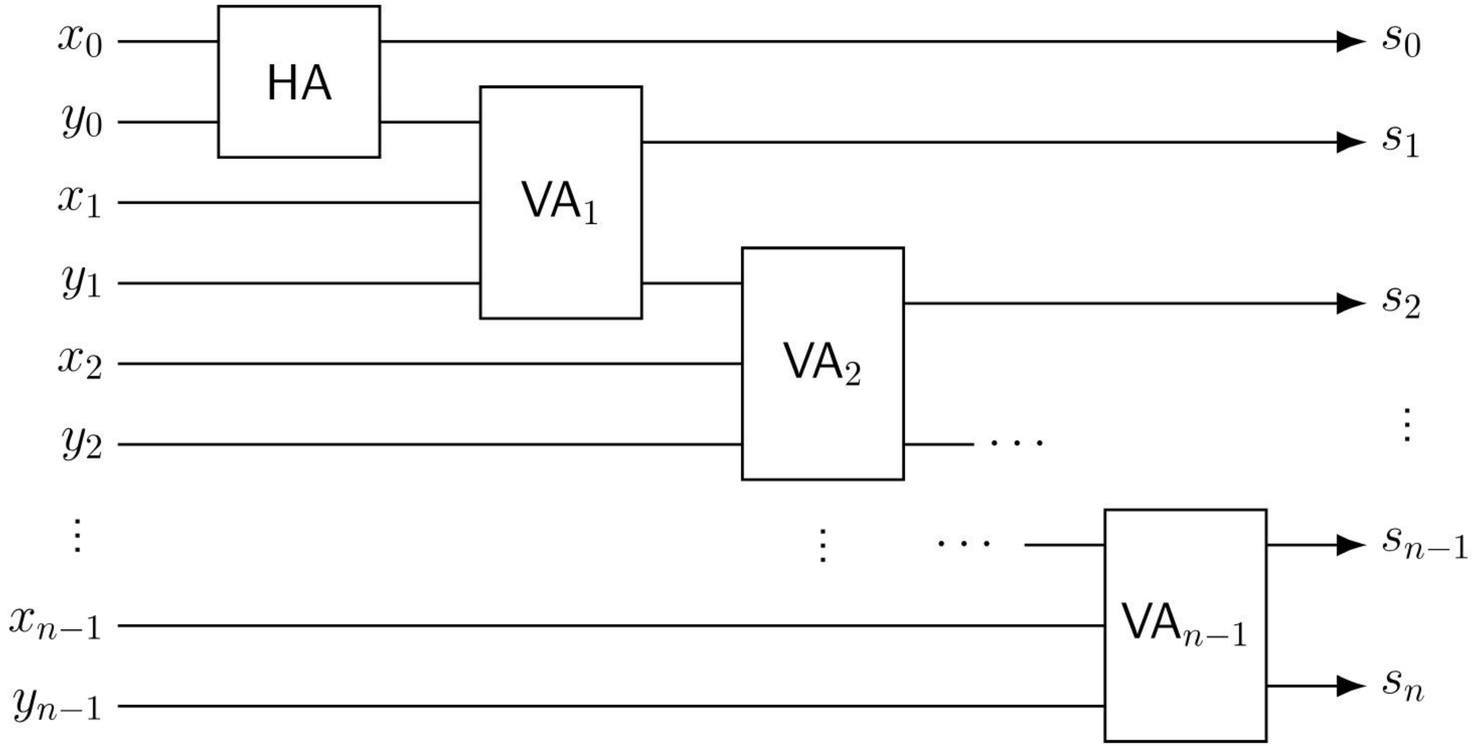


$$
\begin{array}{ccccc}
 & & x_2 & x_1 & x_0 \\
+ & & y_2 & y_1 & y_0 \\
\hline
 & s_3 & s_2 & s_1 & s_0
\end{array}
$$



Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Natürliche Zahlen addieren



- **Größe:** $2 + (n-1) \cdot 5 = 5n - 3$
- **Tiefe:** $1 + (n-1) \cdot 3 = 3n - 2$

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Ergebnis: Ripple-Carry-Addierer

Realisierung „Addition von natürlichen Zahlen“

- Sehr gut strukturiert
- Größe $5n - 3$ $\rightarrow$ **sehr klein**
- Tiefe $3n - 2$ $\rightarrow$ **viel zu tief**

Warum ist unser Schaltnetz so tief?

- Offensichtlich gilt: Überträge brauchen sehr lange
- Verbesserungsidee: **Überträge früh berechnen**
- Wie? Struktureinsicht

Struktureinsicht

- $(x_i, y_i) = (1, 1)$ **generiert** Übertrag
- $(x_i, y_i) = (0, 0)$ **eliminiert** Übertrag
- $(x_i, y_i) \in \{(0, 1), (1, 0)\}$ **reicht** Übertrag **weiter**

Dieses Verhalten ist sehr gut durch einen **Halbaddierer** realisierbar

- $u_i = 1$ **generiert** Übertrag
- $v_i = 1$ **reicht** Übertrag **weiter**

Zentrale Beobachtung

- Jeder HA ist in **Tiefe 1** parallel realisierbar und berechnet u_i und v_i

x_i	y_i	u_i	v_i	**Übertrag**
0	0	0	0	eliminieren
0	1	0	1	weiterreichen
1	0	0	1	weiterreichen
1	1	1	0	generieren

Carry-Look-Ahead-Addierer

Notation

- $f_{HA} : B^2 \to B^2$, $f_{HA}(x_i, y_i) = (u_i, v_i)$
- $u_i = 1$ **generiert** Übertrag, $v_i = 1$ **reicht** Übertrag **weiter**

Vorausberechnung der Überträge

$$c_i = u_{i-1} \vee c_{i-1} v_{i-1}$$

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Carry-Look-Ahead-Addierer

Notation

- $f_{HA} : B^2 \to B^2$, $f_{HA}(x_i, y_i) = (u_i, v_i)$
- $u_i = 1$ **generiert** Übertrag, $v_i = 1$ **reicht** Übertrag **weiter**

Vorausberechnung der Überträge

$$
\begin{aligned}
c_1 &= u_0 \\
c_2 &= u_1 \vee u_0 v_1 \\
c_3 &= u_2 \vee u_1 v_2 \vee u_0 v_1 v_2 \\
c_4 &= u_3 \vee u_2 v_3 \vee u_1 v_2 v_3 \vee u_0 v_1 v_2 v_3 \\
c_i &= u_{i-1} \vee u_{i-2} v_{i-1} \vee u_{i-3} v_{i-2} v_{i-1} \vee \ldots \vee u_0 v_1 v_2 \cdots v_{i-1}
\end{aligned}
$$

$$
c_i = \bigvee_{j=0}^{i-1} \left(u_j \wedge \bigwedge_{k=j+1}^{i-1} v_k \right)
$$

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Carry-Look-Ahead-Addierer

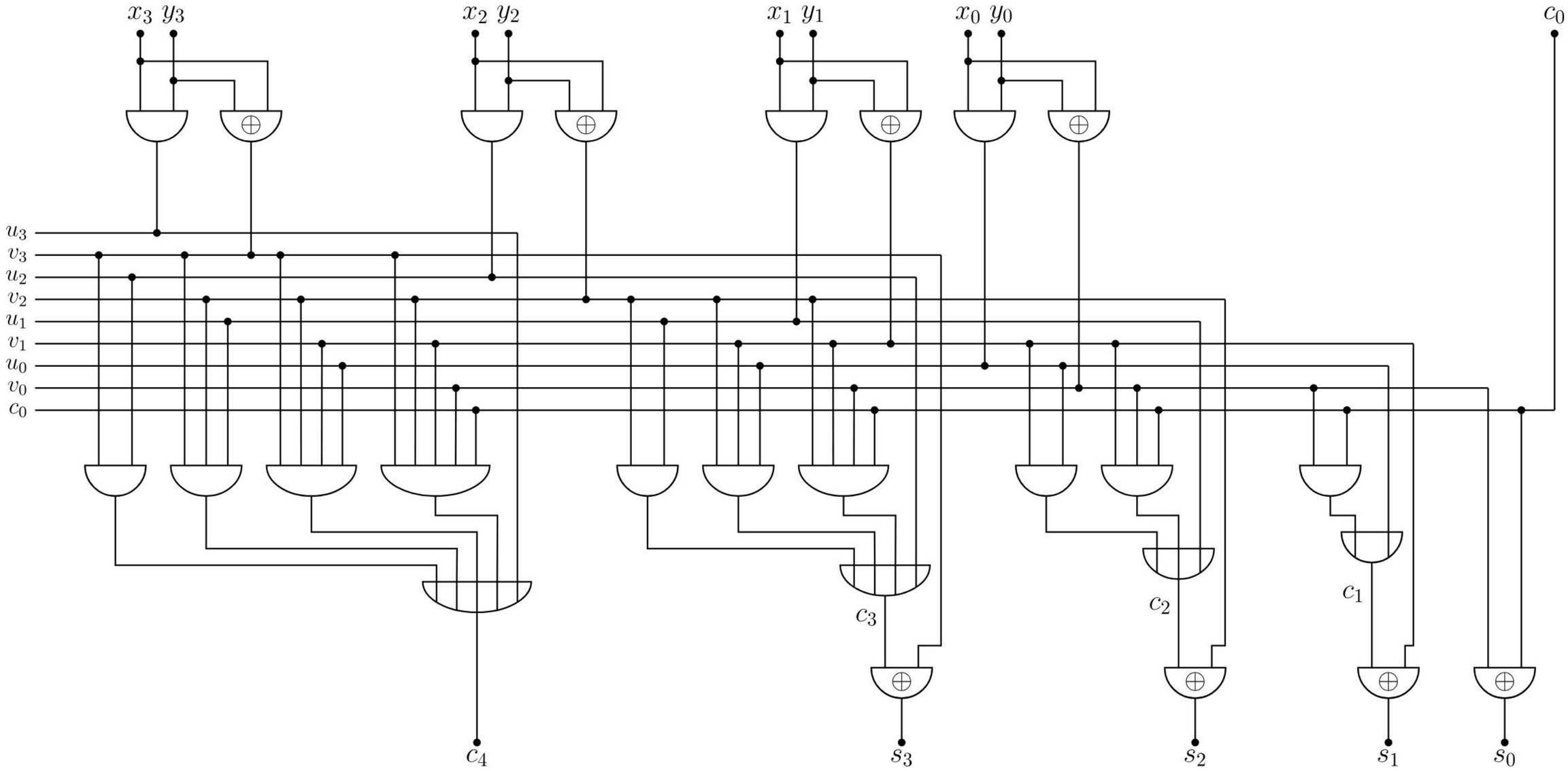
$$c_i = \bigvee_{j=0}^{i-1} \left(u_j \wedge \bigwedge_{k=j+1}^{i-1} v_k \right)$$

Gesamttiefe des CLA: 4

Baustein	Tiefe
Alle Halbaddierer (u_i, v_i)	1
Alle Und-Gatter $\left(u_j \wedge \bigwedge_{k=j+1}^{i-1} v_k \right)$	1
Großes Oder-Gatter	1
$n \times \oplus$-Gatter für korrekte Summenbits	1

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

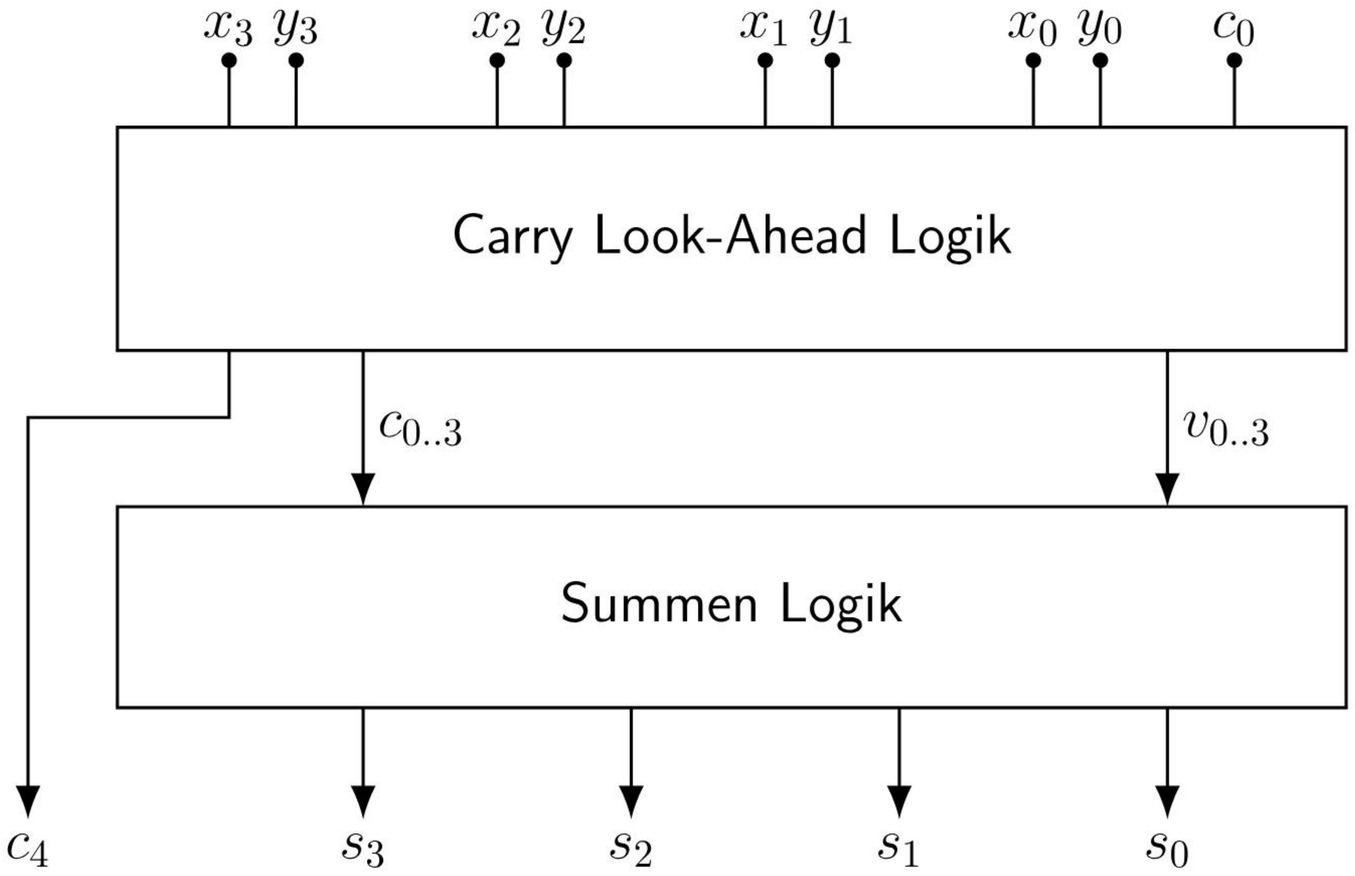
Carry-Look-Ahead-Addierer



Quelle: Denis Pitzschel

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Carry-Look-Ahead-Addierer



Quelle: Denis Pitzschel

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Carry-Look-Ahead-Addierer

Waren wir wirklich fair bei der Bestimmung der Ebenen?

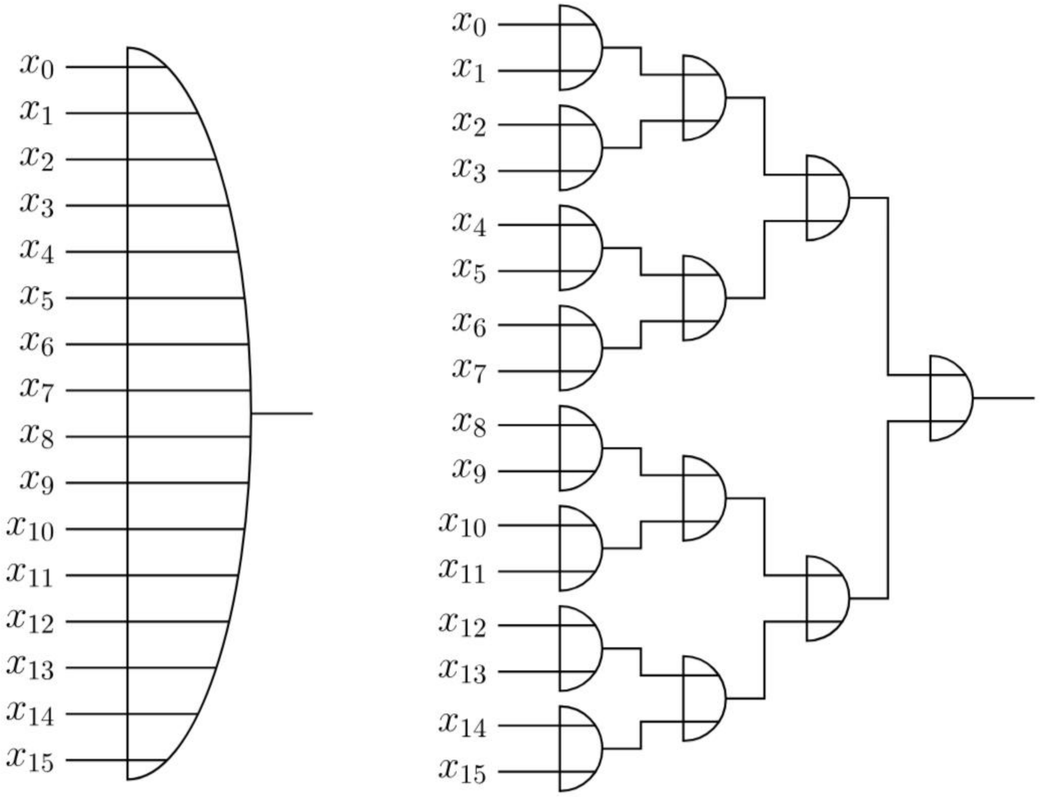
- Beliebig lange Zahlen in Tiefe 4 addieren …
- Anzahl Eingänge eines Gatters heißt **Fan-In**

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Vermeidung von großem Fan-In

Großes Fan-In **systematisch** verkleinern

Beispiel: ODER, Fan-In 16



Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Carry-Look-Ahead-Addierer

Waren wir wirklich fair bei der Bestimmung der Ebenen?

- Beliebig lange Zahlen in Tiefe 4 addieren …
- Anzahl Eingänge eines Gatters heißt **Fan-In**
- Wir vergleichen ein Schaltnetz mit
  - max. Fan-In 2 **(Ripple-Carry-Addierer)**
  - mit einem Schaltnetz mit max. Fan-In n **(Carry-Look-Ahead-Addierer)**
- Große Fan-Ins sind technologisch schwierig zu realisieren

Ziel: Vergleichbarkeit herstellen

Vermeidung von großem Fan-In

Am Beispiel

- Aus Fan-In 16 bei Tiefe 1 (Größe 1)
- wird Fan-In 2 bei Tiefe 4 (Größe 15)

Wie lässt sich das verallgemeinern?

- **Größe**
  - Bei jeder Stufe halb so viele Gatter wie in der Stufe davor
  - $\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots + 1 \approx n$ (*geometrische Reihe*)
- **Tiefe**
  - So oft halbieren, bis man bei einem Gatter ist
  - $\approx \log_2(n)$

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Carry-Look-Ahead-Addierer (CLA)

- Fokus auf AND-Gatter:

$$\sum_{i=1}^{n} \sum_{j=1}^{i} (j-1)$$

- Größe $\approx \frac{n^3}{6}$
- Tiefe $\approx 2\log_2(n)$

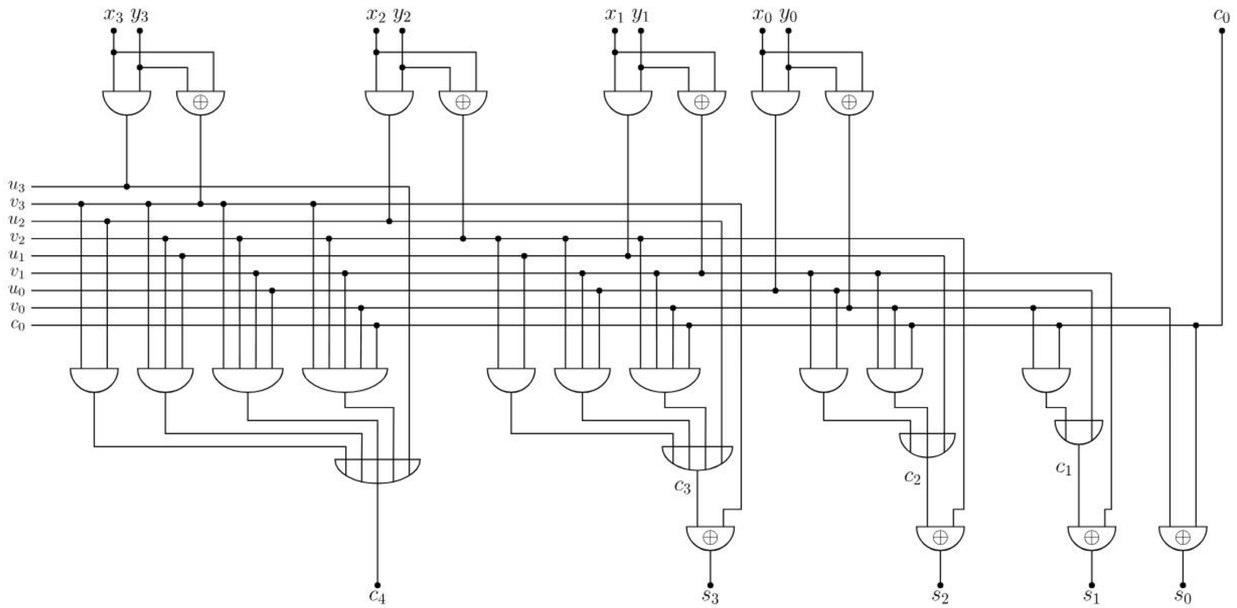
Wolfram Alpha

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Carry-Look-Ahead-Addierer (CLA)

Realisierung der Addition von natürlichen Zahlen

- Gut strukturiert
- Größe $\approx \frac{n^3}{6}$ $\leftarrow$ ziemlich groß
- Tiefe $\approx 2\log_2(n)$ $\leftarrow$ ziemlich flach



Vergleich zum Ripple-Carry-Addierer

n	Ripple-Carry Größe	Ripple-Carry Tiefe	Carry-Look-Ahead Größe	Carry-Look-Ahead Tiefe
8	37	22	170	6
16	77	46	1365	8
32	157	94	10922	10
64	317	190	87381	12

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Quiz: Addierer

Welcher Addierer ist besser?

A: ☐ Ripple-Carry-Addierer

B: ☐ Carry-Look-Ahead-Addierer

Addition natürlicher Zahlen

Darstellung der Tiefe / Größe

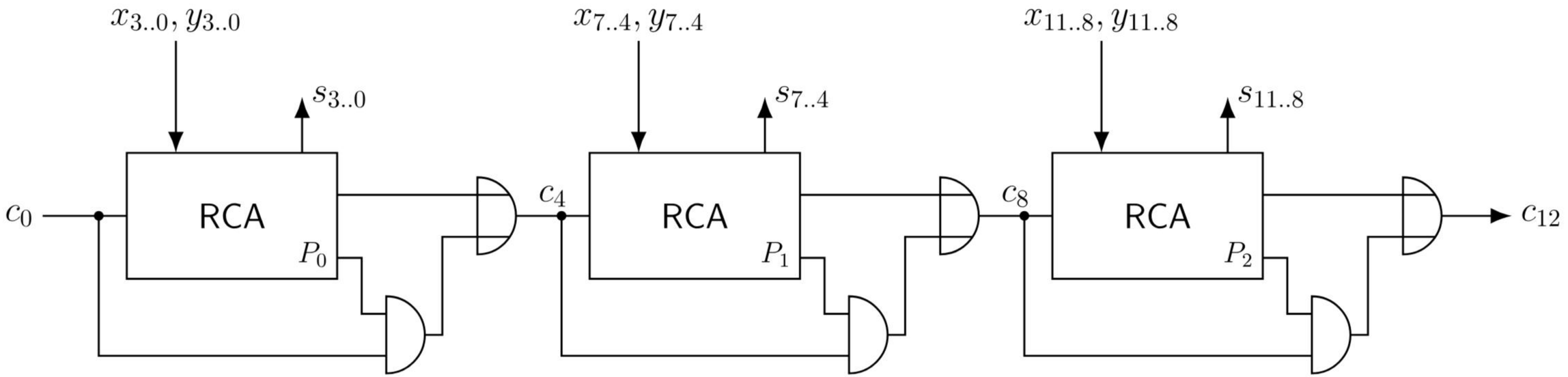
Carry-Skip-Addierer: Motivation

- CLA ist flach, aber **groß** ($\approx \frac{n^3}{6}$) und komplex
- **Frage:** Können wir den Ripple-Carry-Addierer nur **etwas** verbessern, sodass er zumindest **manchmal** schneller ist?

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Carry-Skip-Addierer

Idee: mehrere kleine RCAs, jeder mit eigenem Propagate-Check

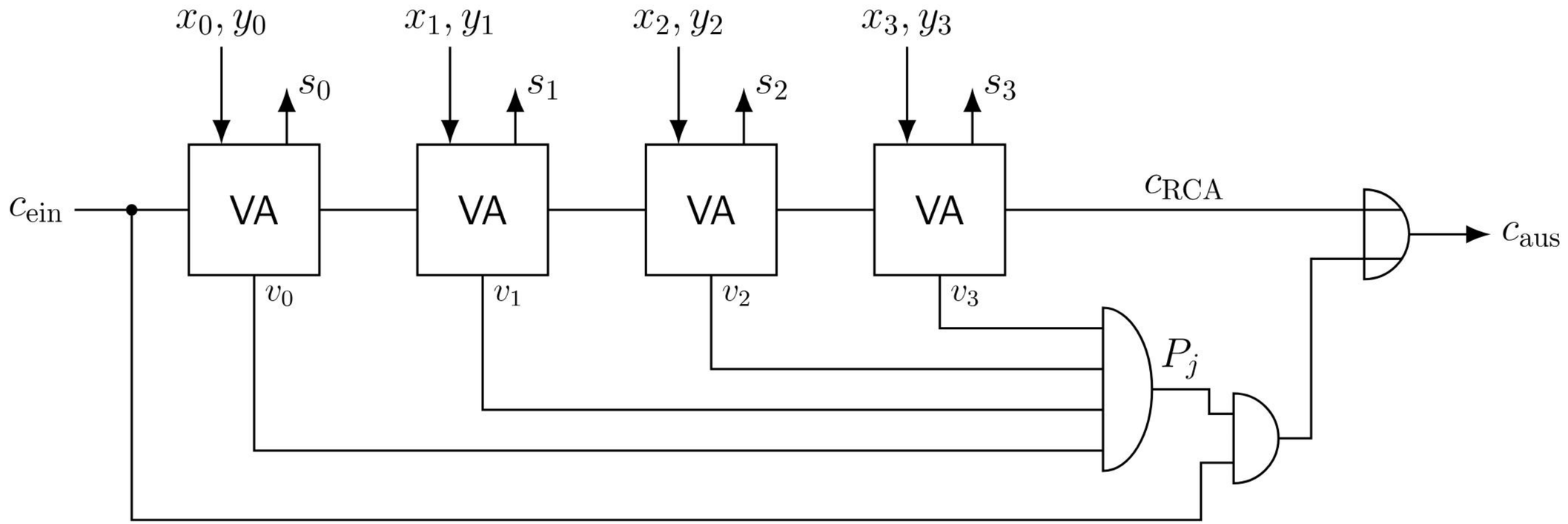


- P_j: alle Stellen des Blocks reichen weiter $\Rightarrow$ Übertrag **überspringt** den Block
- Sonst entsteht der Übertrag im Block selbst: normaler RCA-Übertrag

$$c_{\text{aus}} = c_{\text{RCA}} \vee \left(P_j \wedge c_{\text{ein}}\right)$$

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Carry-Skip-Addierer: ein Block im Detail



- $v_i = x_i \oplus y_i$ berechnet jeder Volladdierer ohnehin (erstes XOR) $\Rightarrow$ nur herausführen
- $P_j = v_0 \wedge v_1 \wedge v_2 \wedge v_3$: ein Und-Gatter, parallel zur Übertragskette

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Carry-Skip-Addierer: Bewertung

- **Größe:** RCA plus je Block ein Und-Gatter für P_j, ein Und und ein Oder $\Rightarrow$ nur wenig größer als RCA
- **Ungünstigster Übertrag:** entsteht im ersten Block, läuft durch ihn hindurch, **überspringt** alle mittleren Blöcke und läuft nur noch durch den letzten Block
- **Aber:** Der lange Pfad durch alle RCAs existiert im Schaltnetz weiterhin
  - Die **Tiefe** des Schaltnetzes bleibt wie beim RCA
  - Er wird nur nie gebraucht: Muss ein Übertrag lange weitergereicht werden, nimmt er die Abkürzung
- **Fazit:** Kompromiss zwischen RCA (klein, langsam) und CLA (groß, flach)

Abschnitt 4.2

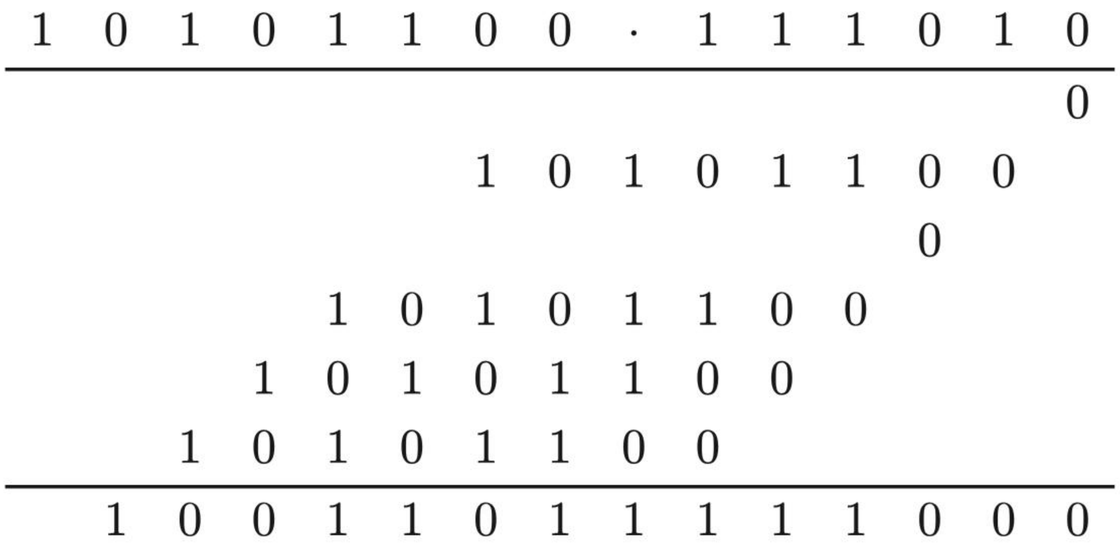
Multiplikation natürlicher Zahlen

- Addition vieler Zahlen
- Carry-Save-Addierer
- Wallace-Tree

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Multiplikation

Direkt mit Binärzahlen …



Multiplizieren heißt

- Nullen passend schreiben
- Zahlen passend verschoben kopieren
- Viele Zahlen addieren

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Multiplikation als Schaltnetz

Multiplikation ist

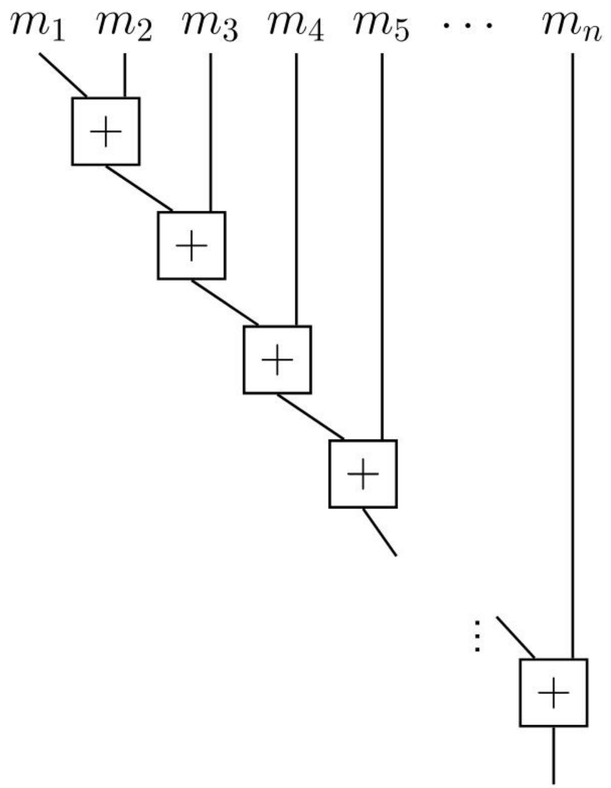
- Nullen passend schreiben
- Zahlen passend verschoben kopieren
- Viele Zahlen addieren

Beobachtung

- Nullen schreiben: **einfach** und **kostenlos**
- Zahlen verschieben und kopieren: **einfach** und **kostenlos**
- **Viele** Zahlen addieren: nicht ganz so einfach

Multiplikation: Addition vieler Zahlen

- Wir haben Addierer für die Addition zweier Zahlen
- Wie addieren wir damit n Zahlen?



Erster Ansatz: Einfach nacheinander

- Schema hat die Tiefe $(n-1)$
- In jeder Ebene noch die Tiefe des Addierwerks
- Tiefe $= (n-1) \cdot$ Tiefe(Addierer) $\Rightarrow$ ist zu tief!

Idee

- In Schaltnetzen **niemals** alles nacheinander tun!
- Was geht gleichzeitig?

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Multiplikation: Addition vieler Zahlen

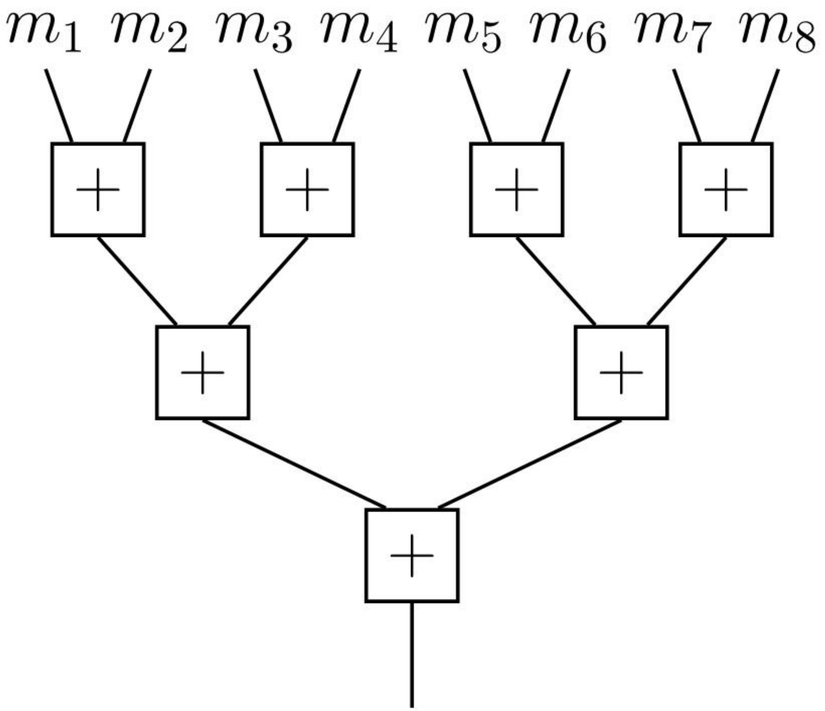
Besserer Ansatz: Paarweise addieren

Bewertung

- Anzahl Addierer $= \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots + 1 \approx n$
- Gesamtgröße $\approx n \cdot$ Größe(Addierer)
- Tiefe $\approx \log_2(n)$ Ebenen

Gesamttiefe

$\approx \log_2(n) \cdot 2\log_2(n) = 2(\log_2(n))^2$



(Anzahl der zu addierenden Bitvektoren ist gleich der Anzahl Bits)

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Multiplikation: Addition vieler Zahlen

Gibt es einen noch besseren Ansatz?

Beobachtung

- Addition ersetzt **zwei** Zahlen
- durch **eine** Zahl gleicher Summe

Zentral für uns

- Gleiche Summe $\rightarrow$ Korrektheit
- Weniger Zahlen $\rightarrow$ Fortschritt

(Verrückte?) Idee: Vielleicht ist es einfacher, **drei** Zahlen zu ersetzen durch **zwei** Zahlen gleicher Summe?

Multiplikation: Addition vieler Zahlen

Gibt es einen noch besseren Ansatz?

(Verrückte?) Idee: Vielleicht ist es einfacher, **drei** Zahlen zu ersetzen durch **zwei** Zahlen gleicher Summe?

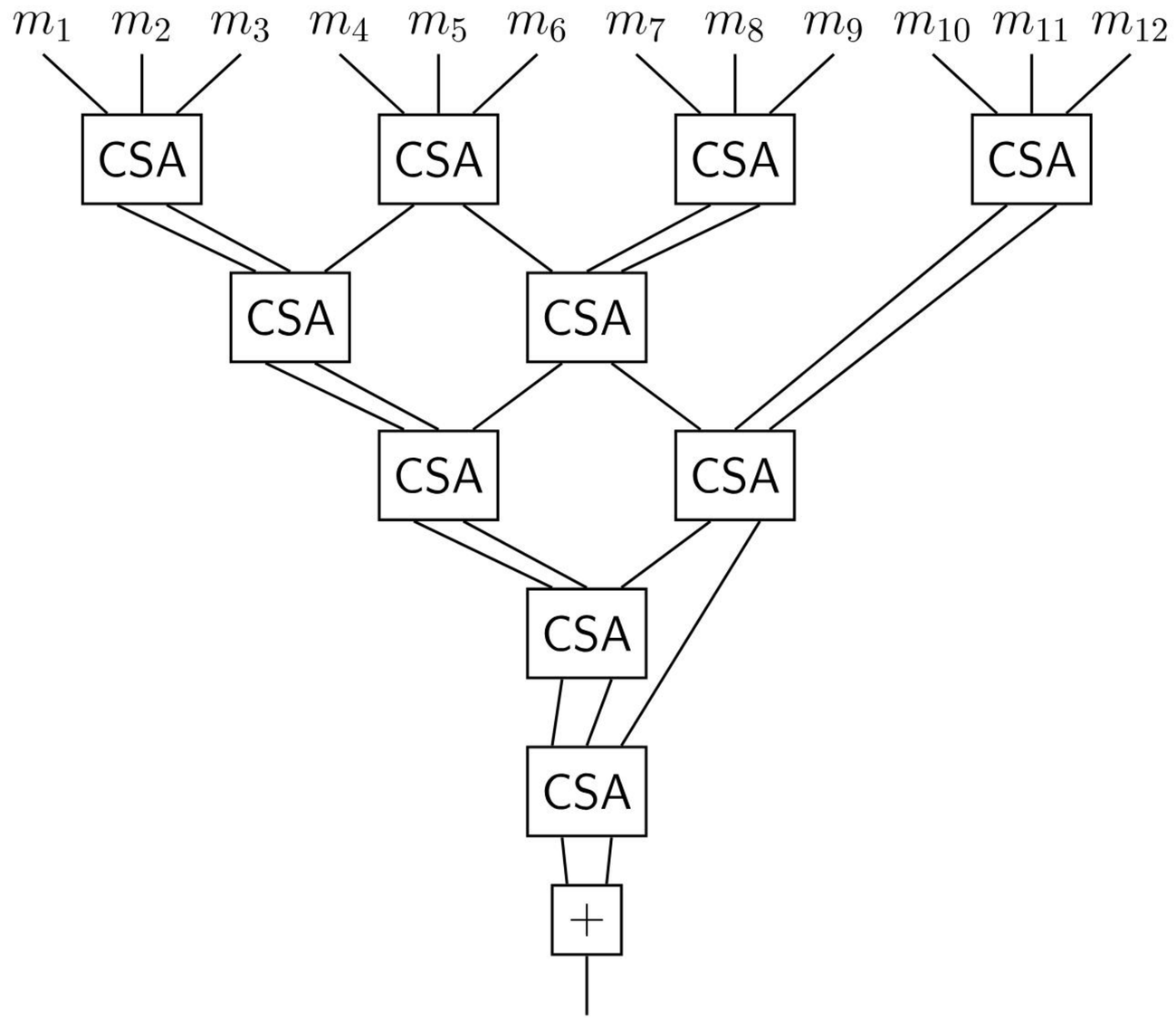
Beobachtung

- **Es gilt immer noch:**
  Gleiche Summe $\rightarrow$ Korrektheit
- **Weniger Fortschritt:**
  $3 \rightarrow 2$ statt $2 \rightarrow 1$

Multiplikation: Addition vieler Zahlen

Wallace-Tree

- Carry-Save-Adder (CSA)
- m_i sind n-Bit-Zahlen
- Sinnvoll, wenn CSA flacher ist als CLA-Addierer



Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Multiplikation: Carry-Save-Adder

Gesucht

- Addierer mit drei Eingängen und zwei Ausgängen, für den gilt:

$$x + y + z = a + b$$

Gefunden

- **Volladdierer**, der für $x, y, z \in \{0, 1\}$ folgendes Resultat liefert:

$$x + y + z = 2 \cdot c + s$$

- s ist das Summenbit
- $2 \cdot c$ ist das Übertragsbit, bezogen auf seine Stelle eine Position weiter links

Multiplikation: Carry-Save-Adder

- Parallelschaltung von n Volladdierern
- Liefern jeweils die **Summenbits**
- und die **Übertragsbits**

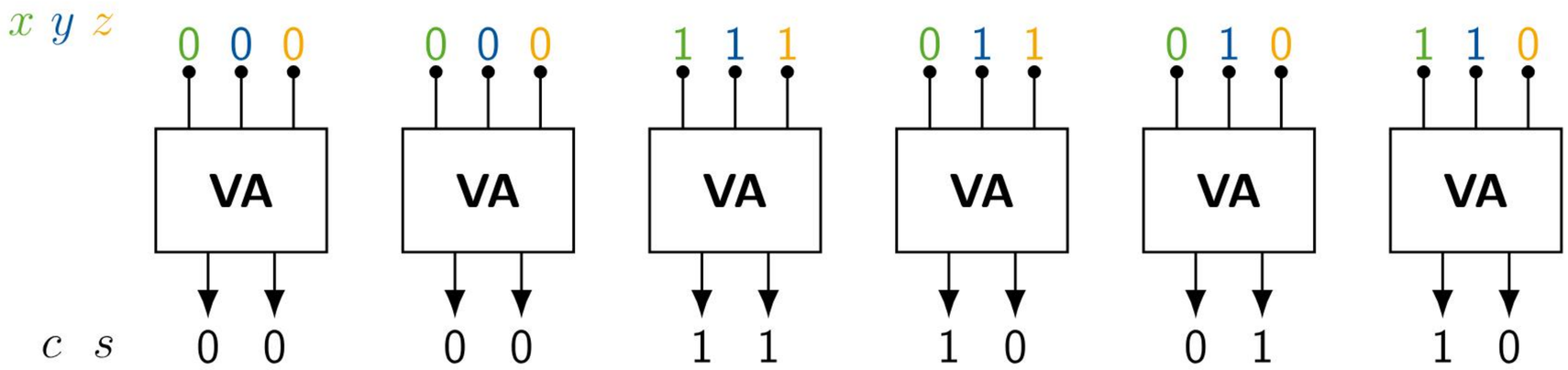
Diese Volladdierer sind nicht verkettet

- Jeder liefert ein s-Bit und ein c-Bit
- Aus diesen Bits werden
  - das Summen-Wort gebildet, das an der höchstwertigen Stelle (Anfang) mit 0 erweitert wird,
  - und das Carry-Wort, das an der niederwertigsten Stelle (Ende) mit 0 erweitert wird
- **Summe** aus Summen-Wort und Carry-Wort **bildet das Resultat**

Beispiel: Carry-Save-Adder

Wir addieren

- $x = 001001$
- $y = 001111$
- $z = 001100$



Summen-Wort $= (0)\ 0\ 0\ 1\ 0\ 1\ 0 = 0001010$

Carry-Wort $= 0\ 0\ 1\ 1\ 0\ 1\ (0) = 0011010$

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Beispiel: Carry-Save-Adder

Summand x			x_5	x_4	x_3	x_2	x_1	x_0
Summand y	$+$		y_5	y_4	y_3	y_2	y_1	y_0
Summand z	$+$		z_5	z_4	z_3	z_2	z_1	z_0
Resultat			$2c_5 + s_5$	$2c_4 + s_4$	$2c_3 + s_3$	$2c_2 + s_2$	$2c_1 + s_1$	$2c_0 + s_0$
Carry-Wort		c_5	c_4	c_3	c_2	c_1	c_0	(0)
Summen-Wort	$+$	(0)	s_5	s_4	s_3	s_2	s_1	s_0

Summand x			0	0	1	0	0	1
Summand y	$+$		0	0	1	1	1	1
Summand z	$+$		0	0	1	1	0	0
Carry-Wort		0	0	1	1	0	1	(0)
Summen-Wort	$+$	(0)	0	0	1	0	1	0
Summe		0	1	0	0	1	0	0

Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Quiz: Carry-Save-Adder

Wie viele Schritte braucht es, c und s zu berechnen?

A: ☐ 1 **B:** ☐ 2 **C:** ☐ 3 **D:** ☐ 4 **E:** ☐ 5

Multiplikation: Carry-Save-Adder

Berechnung von drei n-Bit-Zahlen

- Zum Einsatz kommen n Volladdierer
- Größe: $5n$
- Tiefe: 3

Beobachtung

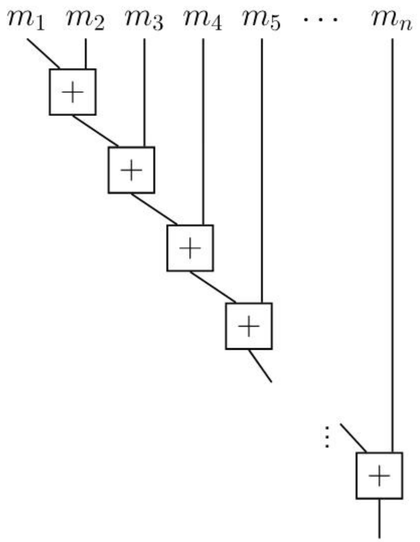
- Der (große) Carry-Look-Ahead-Addierer wird in einem Wallace-Tree erst als letzte Operation benötigt
- Der Wallace-Tree hat somit für die Addition von n Zahlen folgende Eigenschaften:
  - Größe: $\approx \frac{n^3}{3}$ (proportional, bedingt durch den CLA)
  - Tiefe: $\approx 3\log_{3/2}(n) + 2\log_2(n) \approx 7{,}13\log_2(n)$

Fazit: Multiplikation ist nicht **wesentlich teurer oder langsamer** als die Addition

Multiplikation

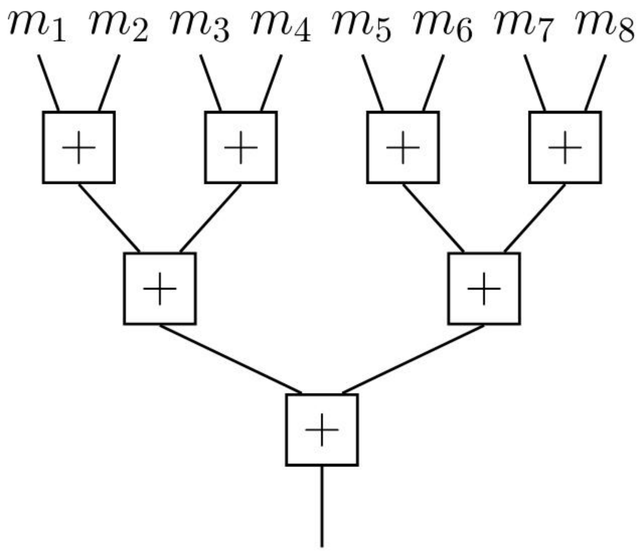
Serielle Addierer

- Tiefe $= (n-1) \cdot$ Tiefe(Addierer)
- Tiefe $= (n-1) \cdot 2\log_2(n)$



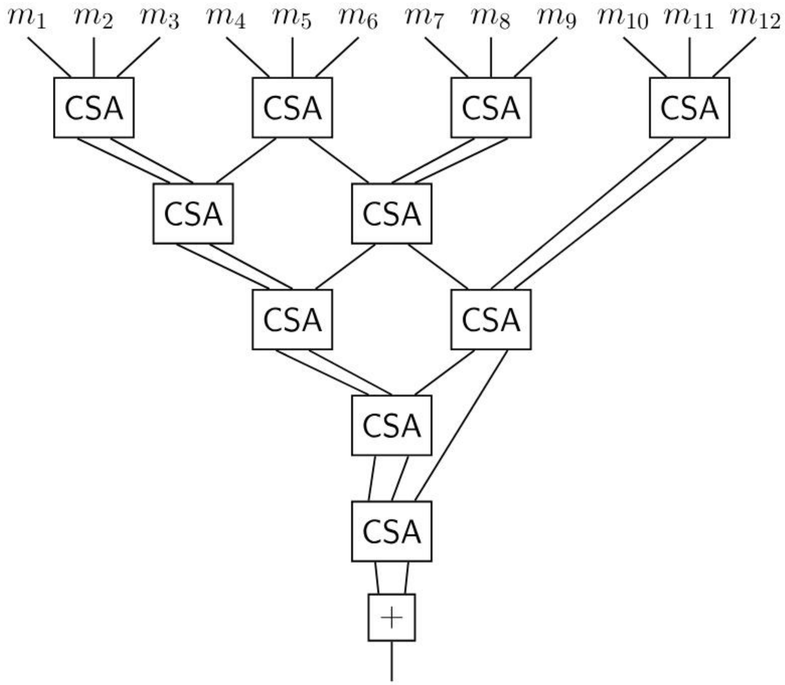
Paarweise Addierer

- Tiefe $= \log_2(n) \cdot$ Tiefe(Addierer)
- Tiefe $= 2(\log_2(n))^2$



Wallace-Tree mit Carry-Save-Addierern

- Tiefe $\approx 3\log_{3/2}(n) + 2\log_2(n)$
  $\approx 7{,}13\log_2(n)$



Chair of Cyber-Physical Systems
RWTH AACHEN UNIVERSITY

Multiplikation

Quiz: Wallace-Tree

Welche Aussage ist korrekt, wenn n n-Bit-Vektoren mit einem Wallace-Tree addiert werden?

A: ☐ Der Wallace-Tree benötigt nur einen Carry-Look-Ahead-Addierer (CLA) mit der Tiefe $2\log_2(n)$

B: ☐ Jeder Carry-Save-Adder (CSA) in dem Wallace-Tree ist ein n-Bit-Addierer, der die Summe von 3 n-Bit-Vektoren berechnet

C: ☐ Wallace-Trees zeigen, dass die Multiplikation deutlich langsamer als die Addition ist

D: ☐ Paarweise Addition ist schneller als ein Wallace-Tree, wenn n groß genug gewählt wird