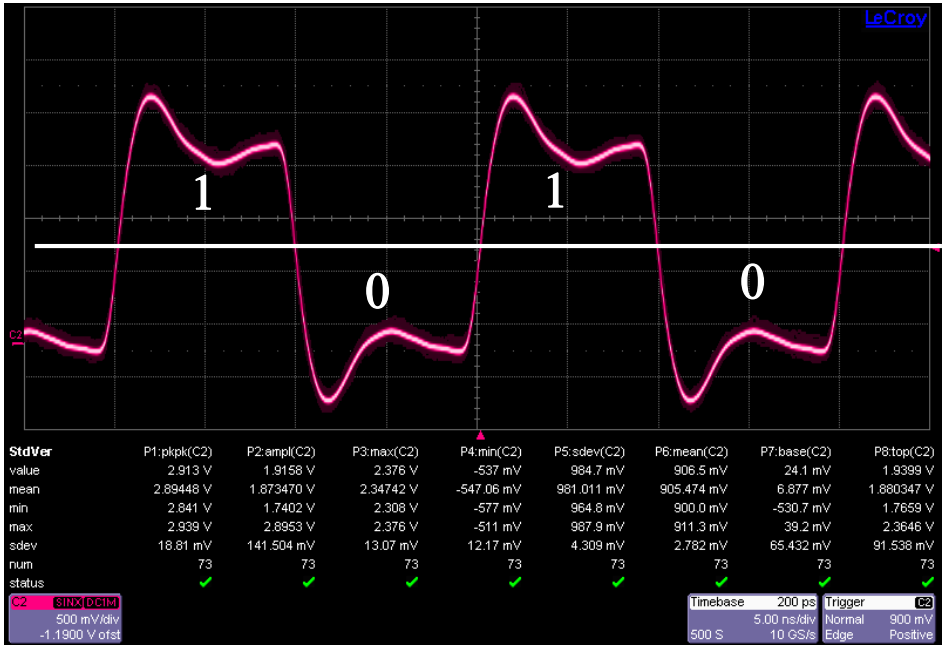
Schaltkreis- und Systementwurf

Teil 2: Digitale Systeme

FAKULTÄT **ELEKTROTECHNIK** UND **INFORMATIONSTECHNIK**

Logikschaltungen

- Zweiwertige Darstellung:
  - 0, LOW, L, FALSE
  - 1, HIGH, H, TRUE
- Elektrische Repräsentation als Spannungssignal:
  - $V_{sig} > V_{th} \rightarrow 1$
  - $V_{sig} < V_{th} \rightarrow 0$



- Verknüpfung von Eingangsgrößen zu Ausgangsgrößen

$$\boxed{(Z_1, Z_2, \cdots) = F(A_1, A_2, \cdots)}$$

- Darstellungsformen:
  - Logikgleichung
  - Wahrheitswerttabelle
  - Gatter-Netzliste

Boolsche Operationen

- Operatoren:
  - Inversion: $Z = \bar{A}$

A	Z
0	1
1	0

  - UND Verknüpfung: $Z = A_1 \cdot A_2$

A_1	A_2	Z
0	0	0
0	1	0
1	0	0
1	1	1

  - ODER Verknüpfung $Z = A_1 + A_2$

A_1	A_2	Z
0	0	0
0	1	1
1	0	1
1	1	1

- Rechenregeln:

	Operation + (ODER)	Operation · (UND)
1)	$(x+y)+z = x+(y+z)$	$(xy)z = x(yz)$
2)	$(x+y) = (y+x)$	$(xy) = (yx)$
3)	$x+xy = x$	$x(x+y) = x$
4)	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$
5)	$x+\bar{x} = 1$	$x\bar{x} = 0$
6)	$x+0 = x$	$x1 = x$
7)	$x+1 = 1$	$x0 = 0$
8)	$x+x = x$	$xx = x$
8)	$\overline{x+y} = \bar{x}\bar{y}$	$\overline{xy} = \bar{x}+\bar{y}$
9)	$\bar{\bar{x}} = x$	

Wahrheitswerttabelle

- Darstellung von Logikfunktionen als Tabelle

Eingänge			Ausgänge	
A_1	A_2	A_3	Z_1	Z_2
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	0	0

Don‘t care Statement

- Reduktion von Wahrheitswerttabellen durch Don‘t care (**X**)
- → Werte die keinen Einfluss auf die Ausgänge haben

Eingänge			Ausgänge	
A_1	A_2	A_3	Z_1	Z_2
X	0	0	0	0
X	0	1	1	1
X	1	0	1	0
X	1	1	0	0

- Verknüpfung der Min- oder Max-Terme der Logikfunktion
- MIN-Term:
  - für genau eine Belegung der Eingänge 1
  - ODER Verknüpfung der Min-Terme einer Ausgangsgröße

Eingänge			Ausgänge	
A_1	A_2	A_3	Z_1	Z_2
X	0	0	0	0
X	0	1	1	1
X	1	0	1	0
X	1	1	0	0

Min-Terme

$$Z_1 = \overline{A_2} \cdot A_3 + A_2 \cdot \overline{A_3}$$

$$Z_2 = \overline{A_2} \cdot A_3$$

- MAX Term:
  - für genau eine Belegung 0
  - UND Verknüpfung der Max Terme einer Ausgangsgröße

Eingänge			Ausgänge	
A_1	A_2	A_3	Z_1	Z_2
X	0	0	0	0
X	0	1	1	1
X	1	0	1	0
X	1	1	0	0

Max-Terme

$$\boldsymbol{Z_1 = (A_2 + A_3) \cdot (\overline{A_2} + \overline{A_3})}$$

$$\boldsymbol{Z_2 = (A_2 + A_3) \cdot (\overline{A_2} + A_3) \cdot (\overline{A_2} + \overline{A_3})}$$

- Verfahren zur Vereinfachung von Logikfunktionen
- Modifizierte Anordnung der Wahrheitswerttabelle
- Karnaugh-Diagramme mit
  - 2 Eingängen:

Z	A_1	$\overline{A_1}$
A_2		
$\overline{A_2}$		

  - 3 Eingängen:

Z	A_1A_2	$\overline{A_1}A_2$	$\overline{A_1A_2}$	$A_1\overline{A_2}$
A_3				
$\overline{A_3}$				

  - 4 Eingängen:

Z	A_1A_2	$\overline{A_1}A_2$	$\overline{A_1A_2}$	$A_1\overline{A_2}$
A_3A_4				
$\overline{A_3}A_4$				
$\overline{A_3A_4}$				
$A_3\overline{A_4}$				

Karnaugh-Verfahren: Min-Terme

- Eintragen der Logiktabelle in das Karnaugh-Diagramm
- Min-Term-Methode:
  - Zusammenfassen von Feldern, die eine 1 enthalten (Min-Terme)
  - Umwandeln dieser Felder in Konjunktionsterme (UND Verknüpfungen)
  - Weglassen von Variablen in negierter und nicht negierter Form
  - ODER Verknüpfung dieser Terme

Z	A_1A_2	$\overline{A_1}A_2$	$\overline{A_1A_2}$	$A_1\overline{A_2}$
A_3A_4	1	1	0	0
$\overline{A_3}A_4$	0	0	1	1
$\overline{A_3A_4}$	0	0	1	1
$A_3\overline{A_4}$	1	1	0	0

$$Z = \overline{A_2} \cdot \overline{A_3} + A_2 \cdot A_3$$

- Max-Term-Methode:
  - Zusammenfassen von Feldern, die eine 0 enthalten (Max-Terme)
  - Umwandeln dieser Felder in Disjunktionsterme (ODER Verknüpfungen), Invertierung der Variablen
  - Weglassen von Variablen in negierter und nicht negierter Form
  - UND Verknüpfung dieser Terme

Z	A_1A_2	$\overline{A_1}A_2$	$\overline{A_1A_2}$	$A_1\overline{A_2}$
A_3A_4	1	1	0	0
$\overline{A_3}A_4$	0	0	1	1
$\overline{A_3A_4}$	0	0	1	1
$A_3\overline{A_4}$	1	1	0	0

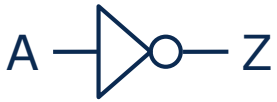
$$\boldsymbol{Z = (\overline{A_2} + \overline{\overline{A_3}}) \cdot (\overline{\overline{A_2}} + \overline{A_3})}$$

$$\boldsymbol{Z = (\overline{A_2} + A_3) \cdot (A_2 + \overline{A_3})}$$

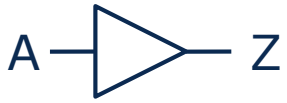
$$\boldsymbol{Z = \overline{A_2} \cdot \overline{A_3} + A_2 \cdot A_3}$$

- Realisierung von logischen Verknüpfungen durch Logikzellen (Gatter, Gates)
- Beispiele:

INV: $Z = \bar{A}$

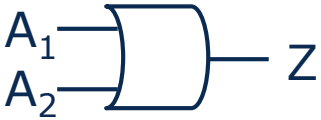


BUF: $Z = A$

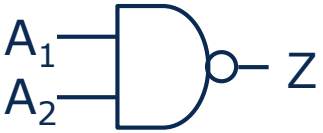


AND2: $Z = A_1 \cdot A_2$

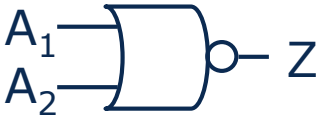
OR2: $Z = A_1 + A_2$



NAND2: $Z = \overline{A_1 \cdot A_2}$

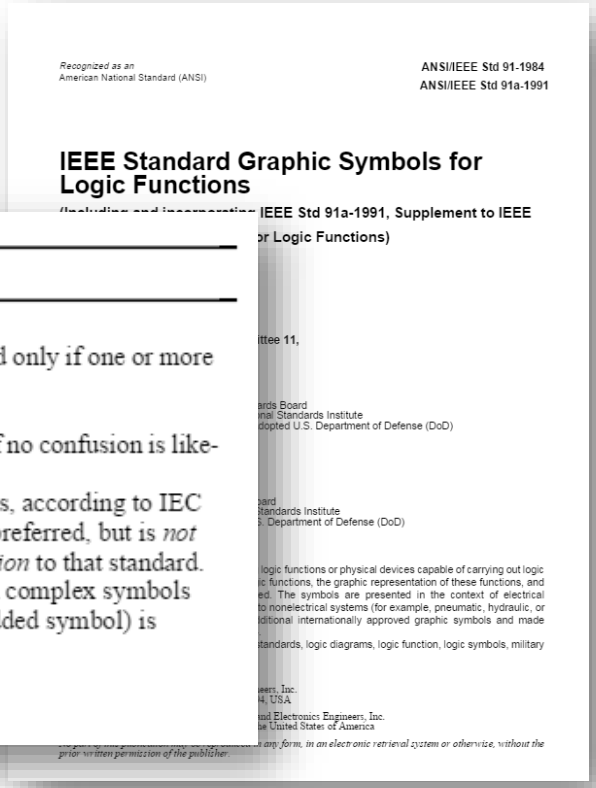


NOR2: $Z = \overline{A_1 + A_2}$



- Logikzellenbibliotheken beinhalten zusätzlich komplexe Gatter mit >2 Eingängen
  - Adder, Multiplexer, AOI, OAI

- Standardisierte Darstellung von Logiksymbolen
  - IEC 60617-12 : 1997
  - ANSI/IEEE Std 91/91a-1991

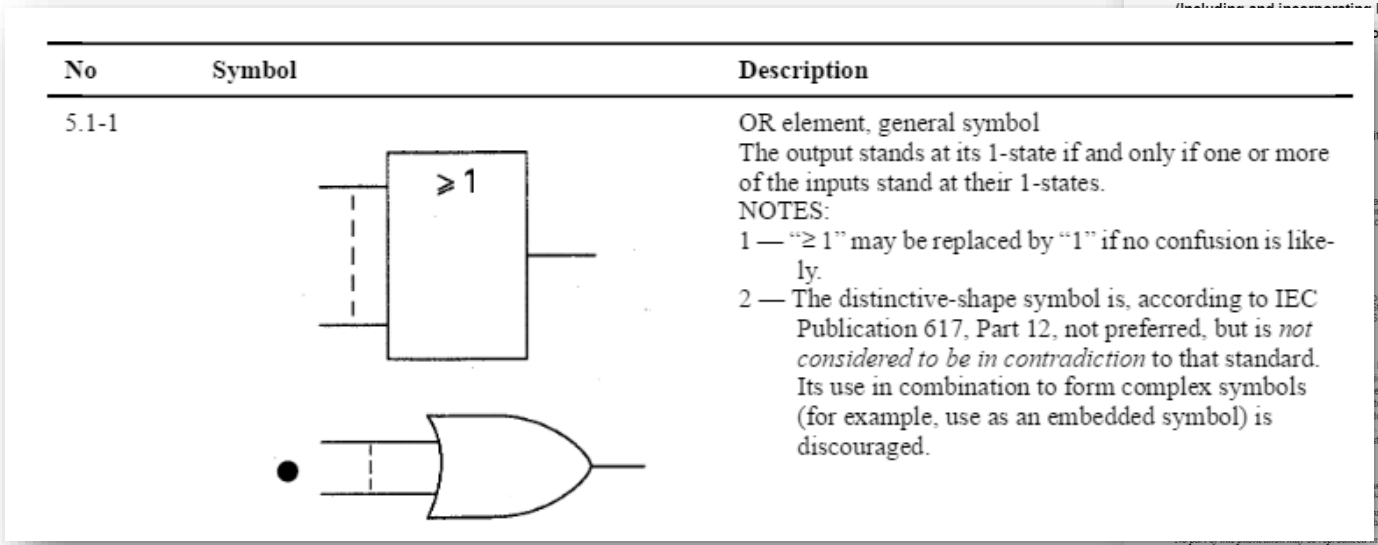


Recognized as an American National Standard (ANSI)

ANSI/IEEE Std 91-1984
ANSI/IEEE Std 91a-1991

IEEE Standard Graphic Symbols for Logic Functions

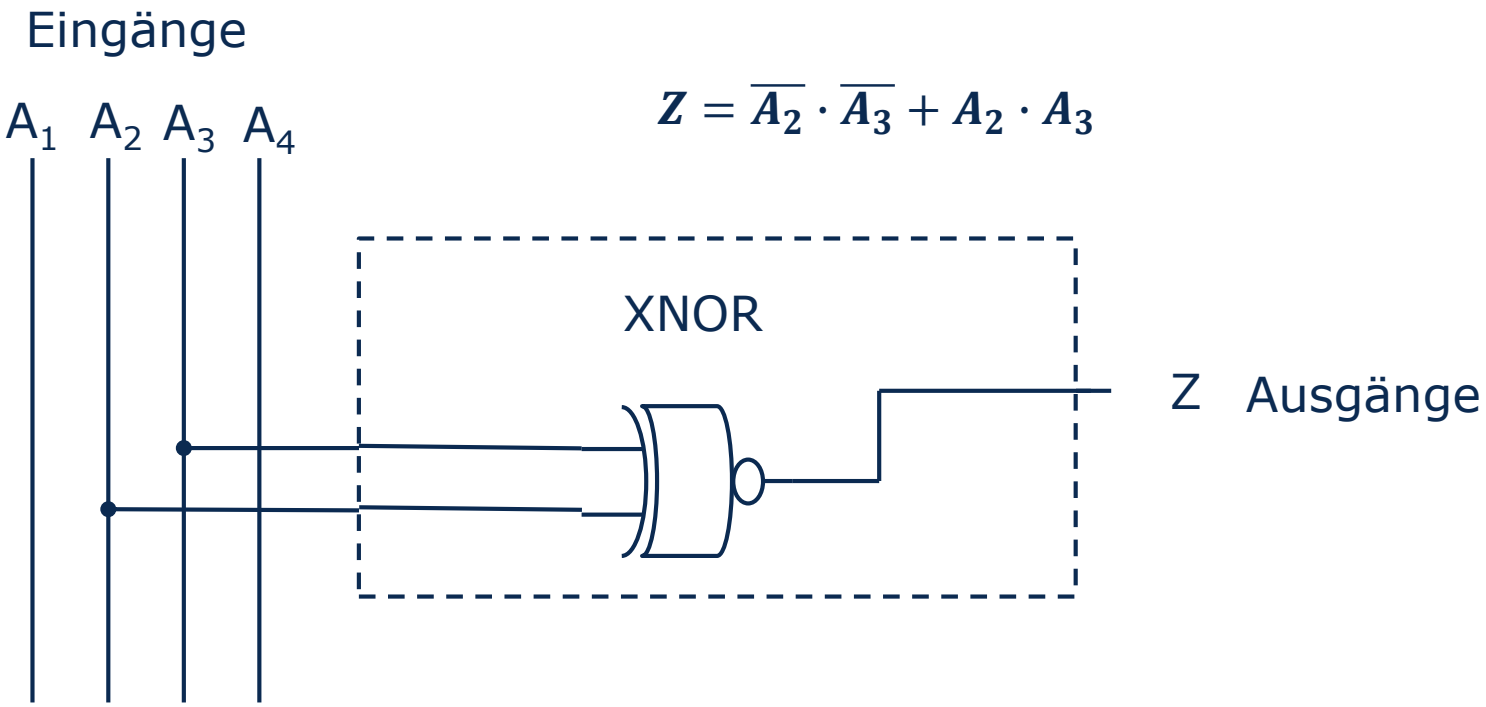
IEEE Std 91a-1991, Supplement to IEEE
r Logic Functions)



No	Symbol	Description
5.1-1	≥1	OR element, general symbol The output stands at its 1-state if and only if one or more of the inputs stand at their 1-states. NOTES: 1 — "≥ 1" may be replaced by "1" if no confusion is likely. 2 — The distinctive-shape symbol is, according to IEC Publication 617, Part 12, not preferred, but is *not considered to be in contradiction* to that standard. Its use in combination to form complex symbols (for example, use as an embedded symbol) is discouraged.

- Weit verbreitete Verwendung der „distinctive-shape symbols" in Handbüchern, Standardzellenbibliotheken, wiss. Publikationen
- *„ … im englischen Sprachraum waren und sind die amerikanischen Symbole (mittlere Spalte) üblich. Die IEC-Symbole sind international auf beschränkte Akzeptanz gestoßen und werden in der amerikanischen Literatur (fast) durchgängig ignoriert. "*
  https://de.wikipedia.org/wiki/Logikgatter (23.10.2015)

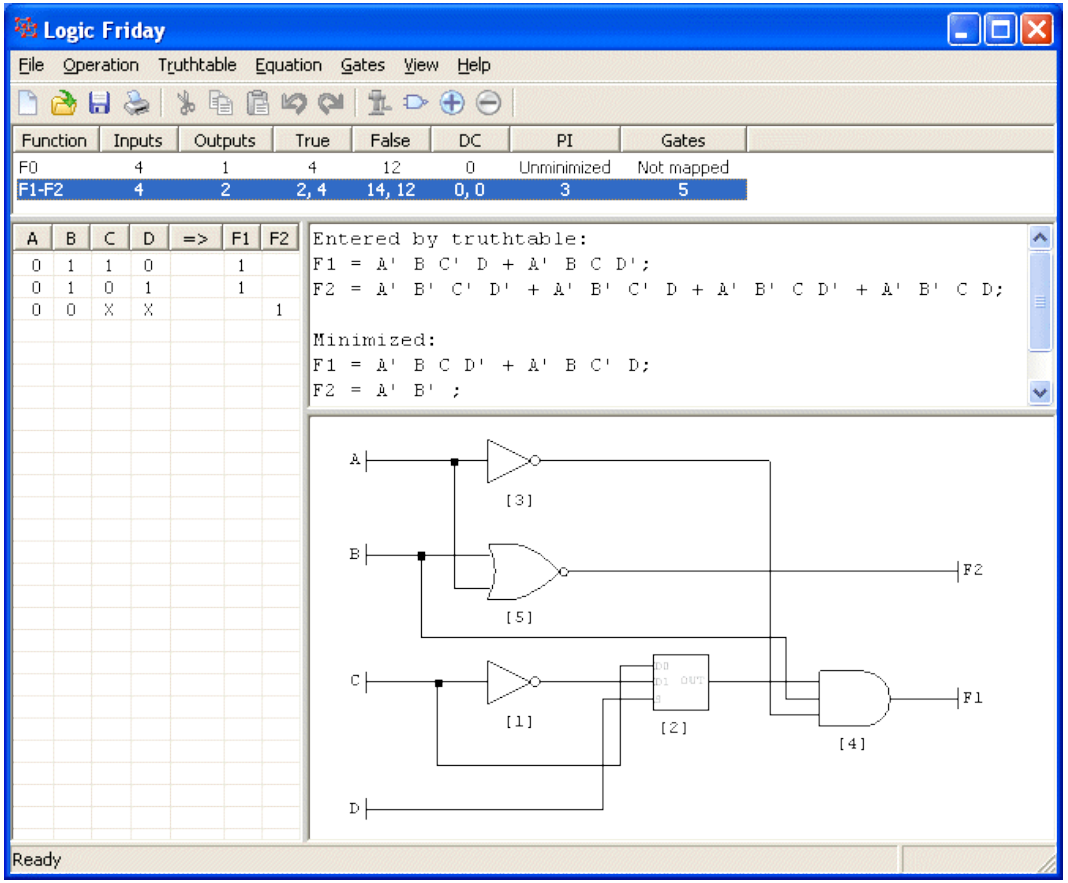
- Darstellung der Logikfunktion durch Gatterschaltung



- Darstellung der Logikfunktion abhängig von verfügbarer Gatterbibliothek
- → Schaltungssynthese, Mapping

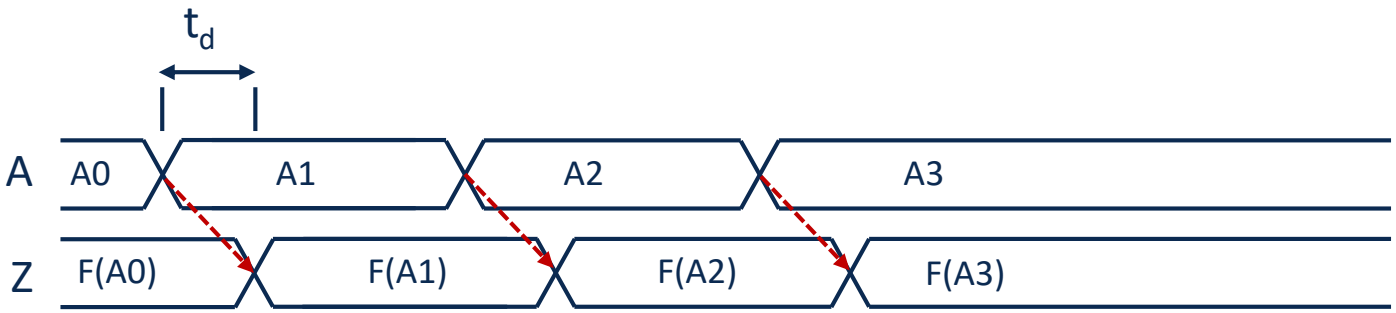
Software: Logic Friday

- Freeware Logic Friday → **www.sontrak.com**
- Darstellung, Vereinfachung und Optimierung von Logikfunktionen (Gleichung, Tabelle, Gatternetzliste)



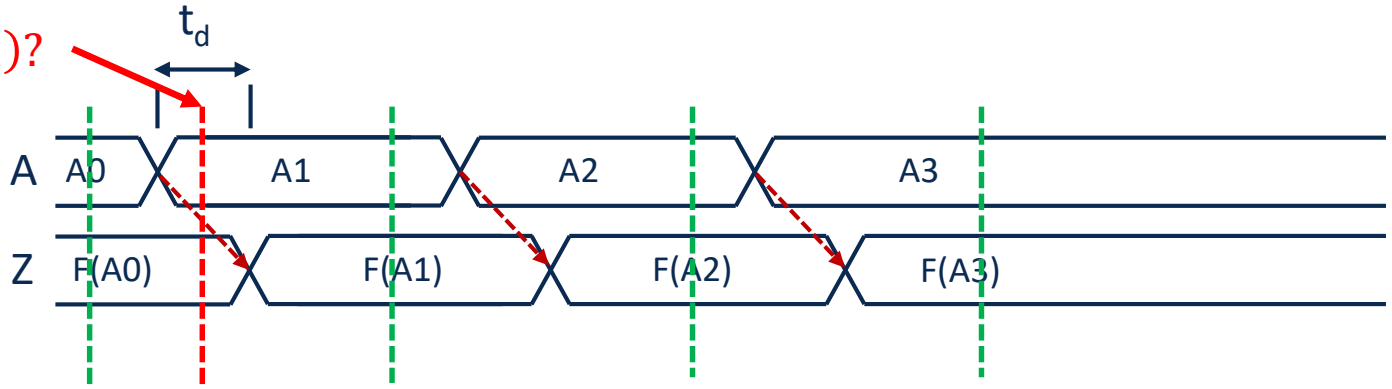
Quelle: www.sontrak.com

- Der Ausgangswert ändert sich **bei Änderung** der Eingangswerte nach einer Verzögerungszeit t_d (Delay)
- Die ideale Delay-Zeit ist $t_d=0$
- Signalpfade in Schaltungen mit mehreren Ein-und Ausgängen können individuell unterschiedliche Delays haben.

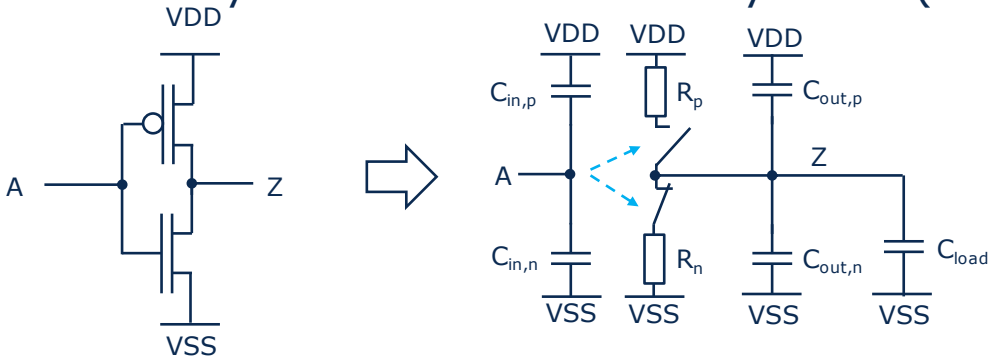


- Kombinatorische Schaltungen beinhalten **keine getakteten Speicher**
- $Z = F(A)$

- $Z = F(A)?$



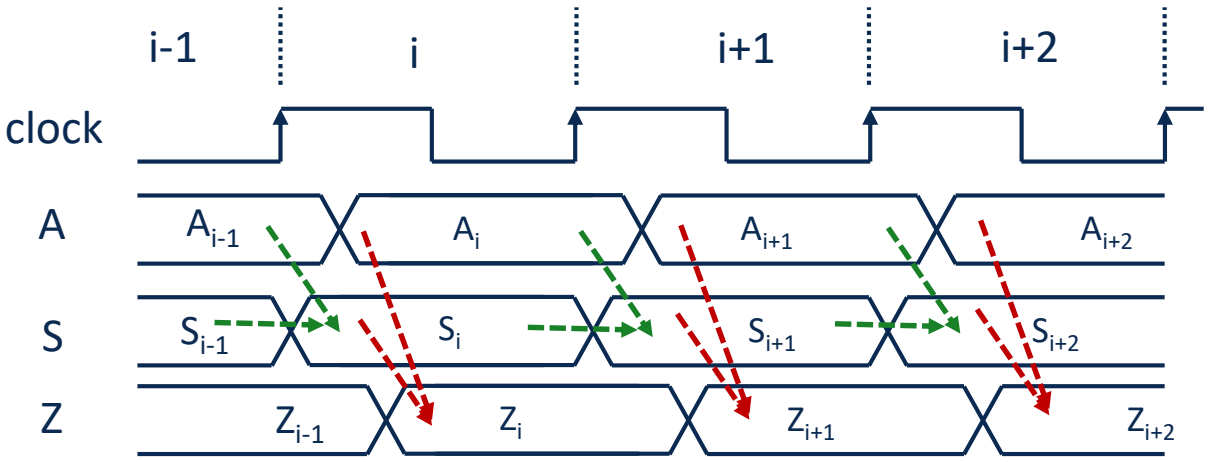
- Kombinatorische Schaltungen beinhalten **keine getakteten Speicher**
- **ABER:** Sie beinhalten dynamische elektrische Systeme (RC) mit Speicher



- → Betrachtung der Signale zu **definierten Zeitpunkten** bei denen $Z = F(A)$ gilt
- → Anwendung der Theorie der kombinatorischen Logik **ohne Speicher**.

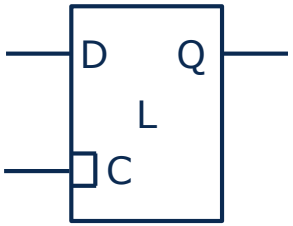
- Sequentielle Logikschaltungen **beinhalten getaktete Speicher**
- Speicher Zustand S
- Betrachtung des Systems zu Zeitpunkten (i,i+1,i+2,…), z.B. realisiert durch ein Taktsignal (clock)

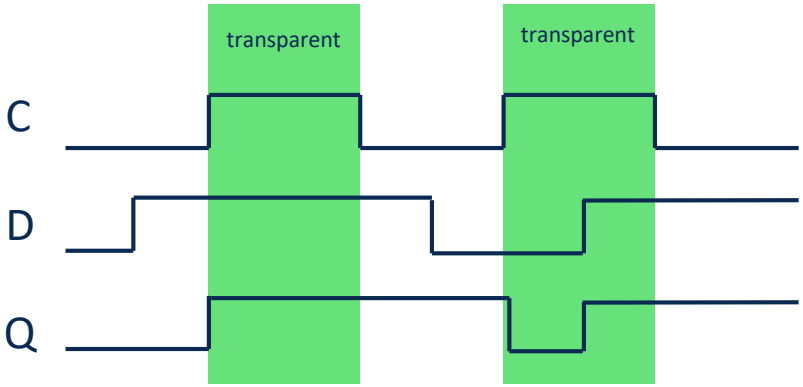
$$S_{i+1} = G(A_i, S_i)$$
$$Z_i = F(A_i, S_i)$$



Zustandsgesteuerter Speicher: D-Latch

- Zustandsgesteuertes Speicherelement
- Eingänge:
  - D: Dateneingang
  - C: Clock (alternativ auch E: Enable)
    - C=1 schaltet das Latch transparent (Speicher wird geschrieben)
- Ausgang:
  - Q: Datenausgang

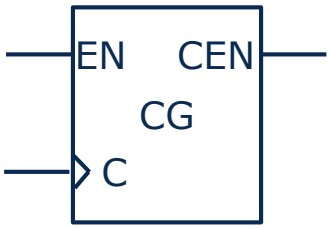




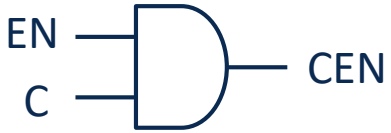
D	C	Q
0	1	0
1	1	1
X	0	Q_{i-1}

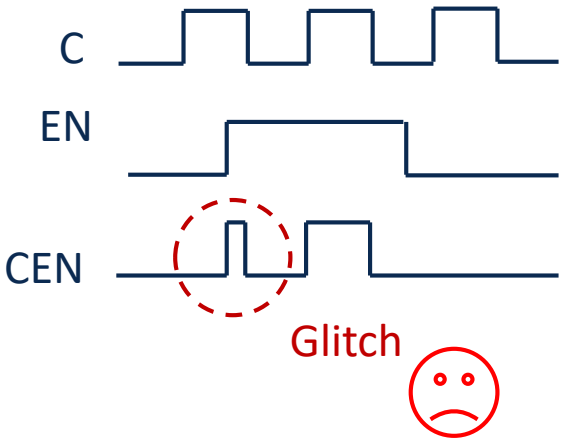
Beispiel: Clock Gate

- Unterbrechen des Taktsignals durch ein Steuersignal EN (enable)
- EN=0 → CEN=0
- Reduktion der Verlustleistung temporär ungenutzter Schaltungsteile

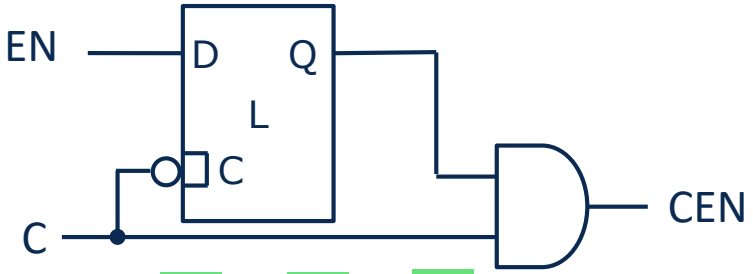


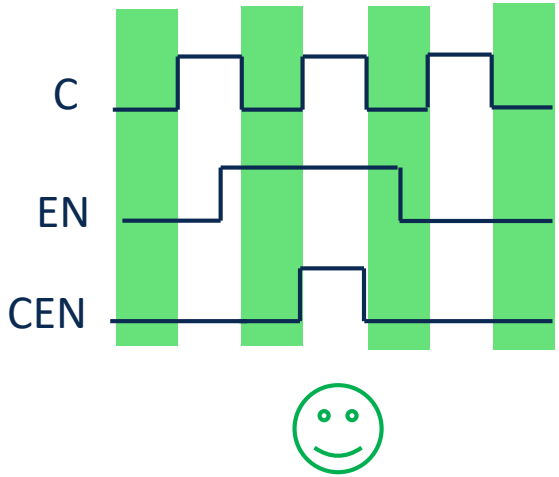
Lösung A



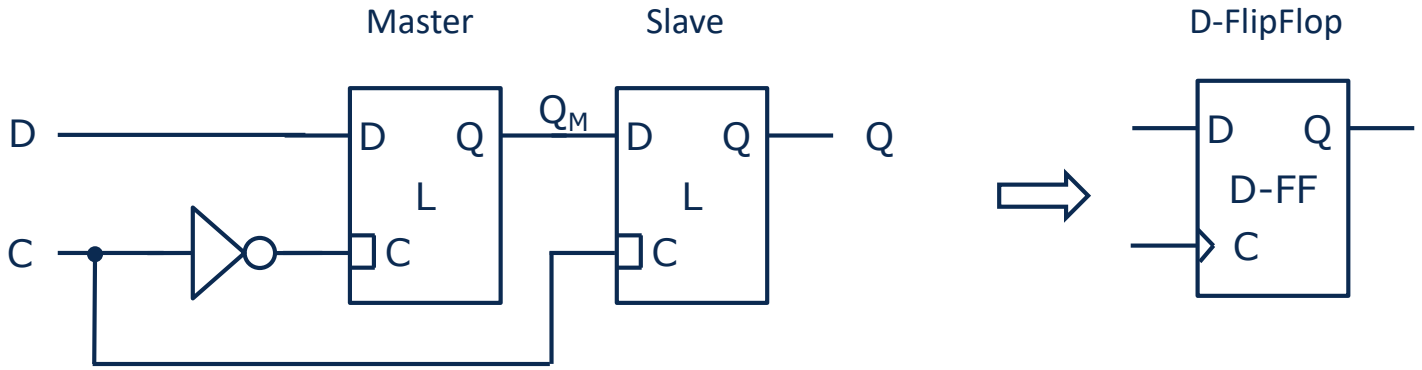


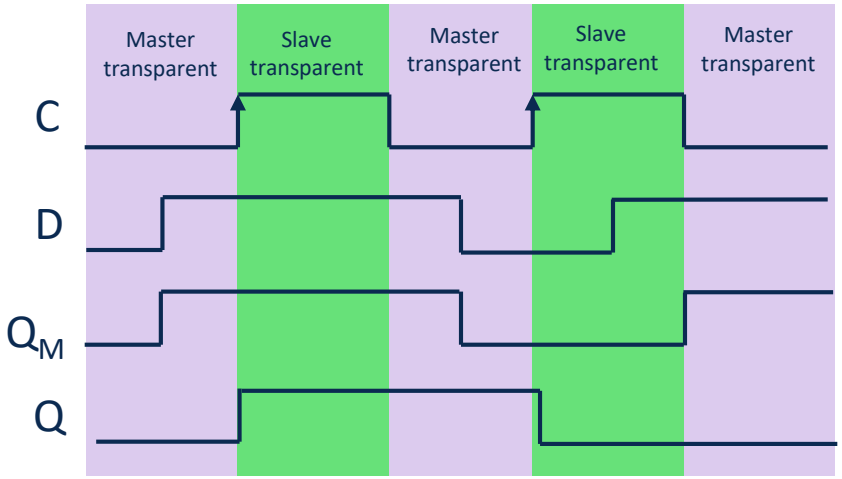
Lösung B





Flankengesteuerter Speicher: Master-Slave FlipFlop

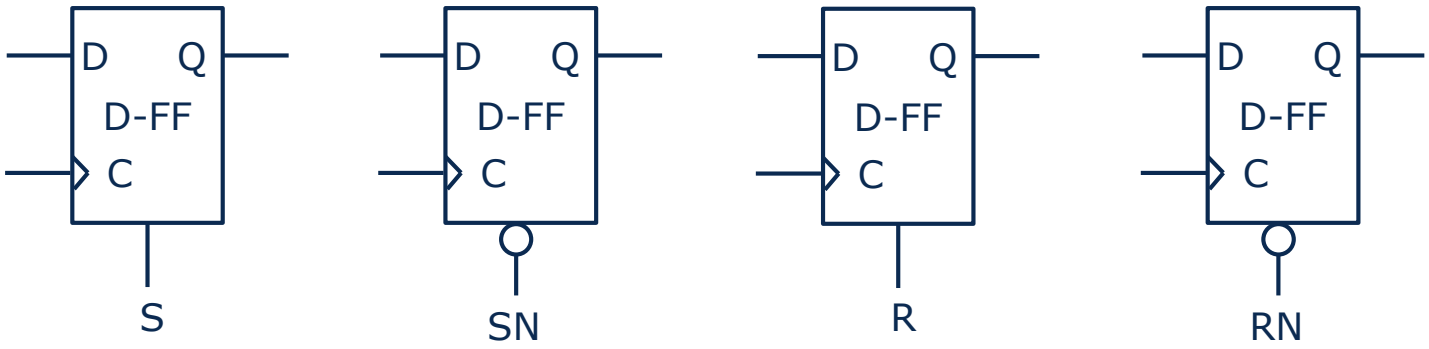




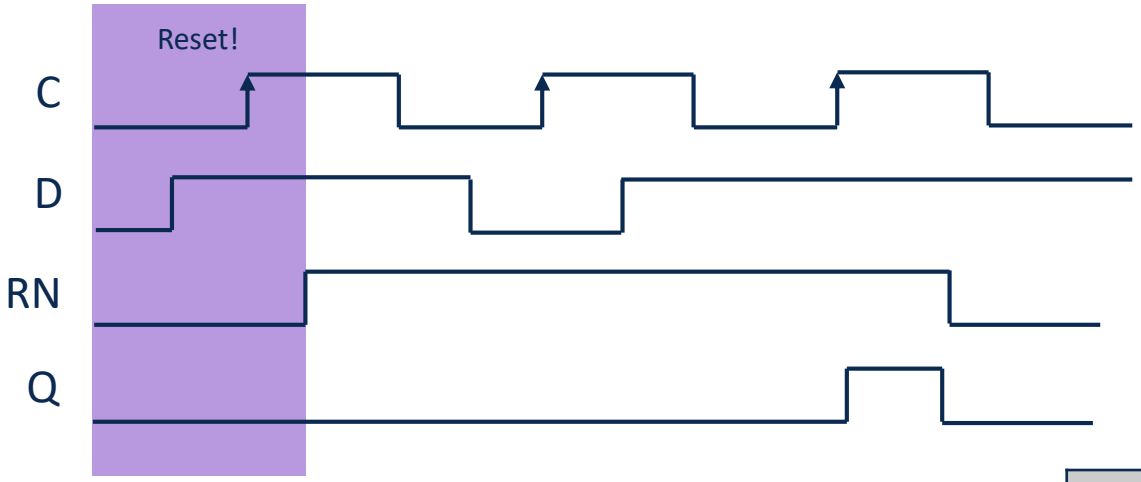
D	C	Q
0	↑	0
1	↑	1
X	↓	Q_{i-1}

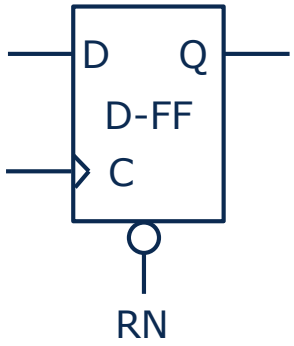
Asynchrones Set und Reset

- Zusätzlicher asynchroner Eingang zum definieren eines Anfangszustandes **unabhängig vom Clock C**
- Möglich bei Latches und FlipFlops
- Aktiv nur auf einen Logikpegel (High-aktiv (1), Low-aktiv (0))
  - High-aktives Set S: aktiv bei S=1 → $Q_0 = 1$
  - Low-aktives Set SN: aktiv bei SN=0 → $Q_0 = 1$
  - High-aktives Reset R: aktiv bei R=1 → $Q_0 = 0$
  - Low-aktives Reset RN: aktiv bei RN=0 → $Q_0 = 0$



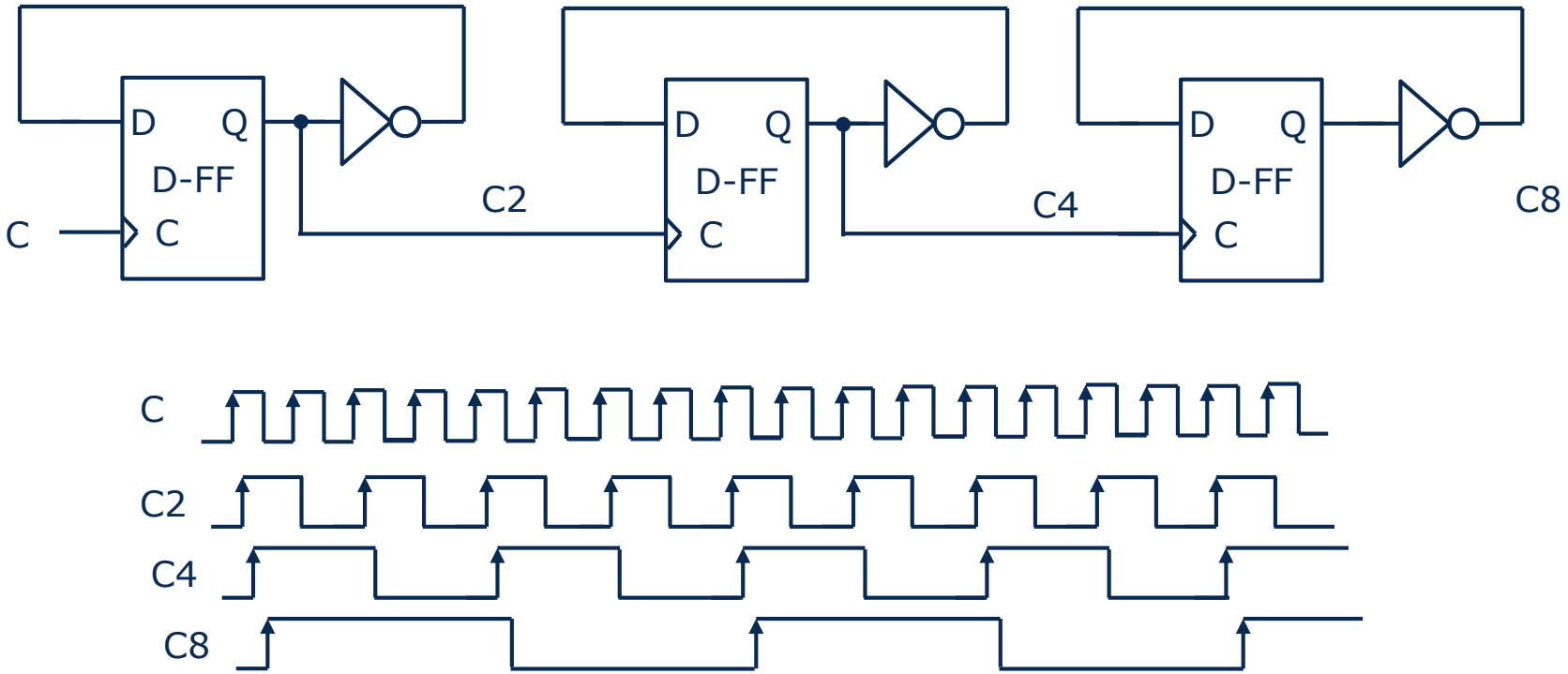
Beispiel: D-FlipFlop mit asynchronem low-aktivem Reset



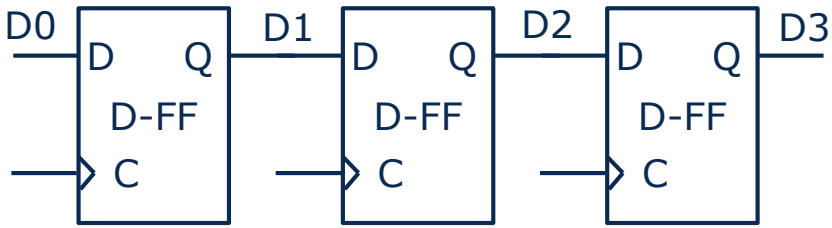


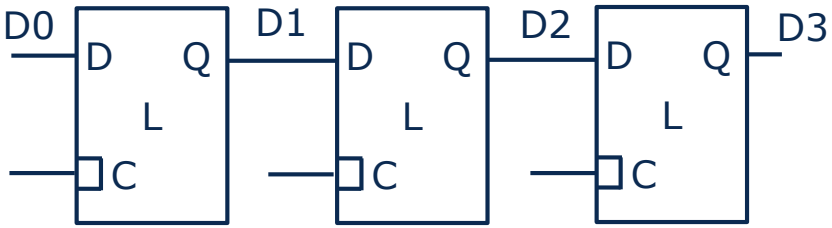
RN	D	C	Q
1	0	↑	0
1	1	↑	1
1	X	↓	Q_{i-1}
0	X	X	0

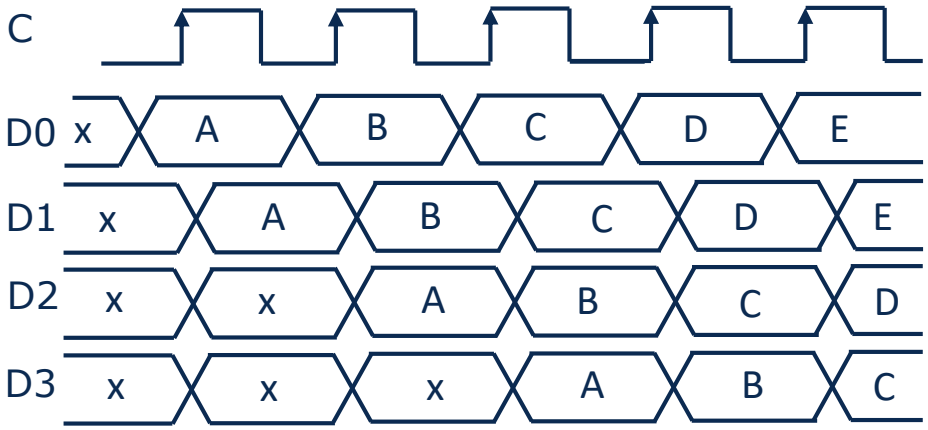
Beispiel: Binärer Frequenzteiler

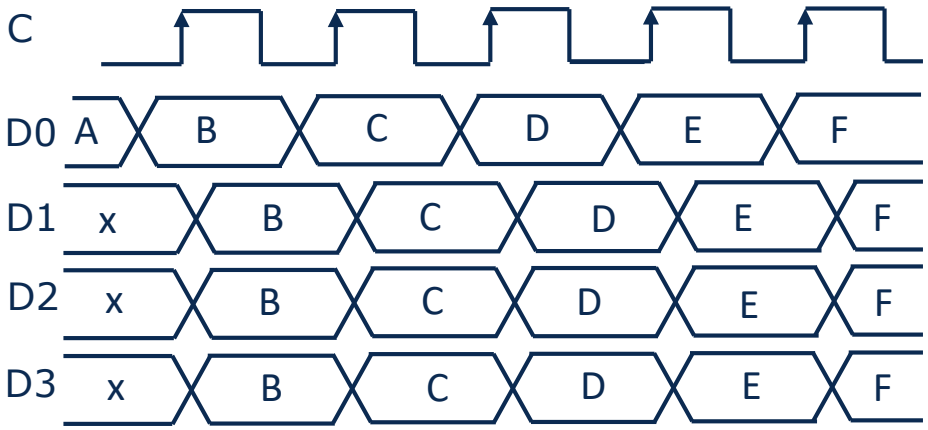


Beispiel: Schieberegister

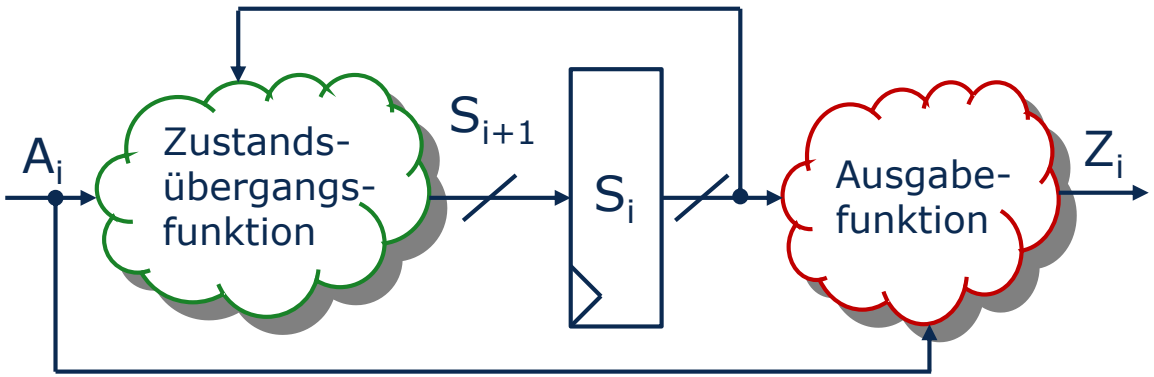




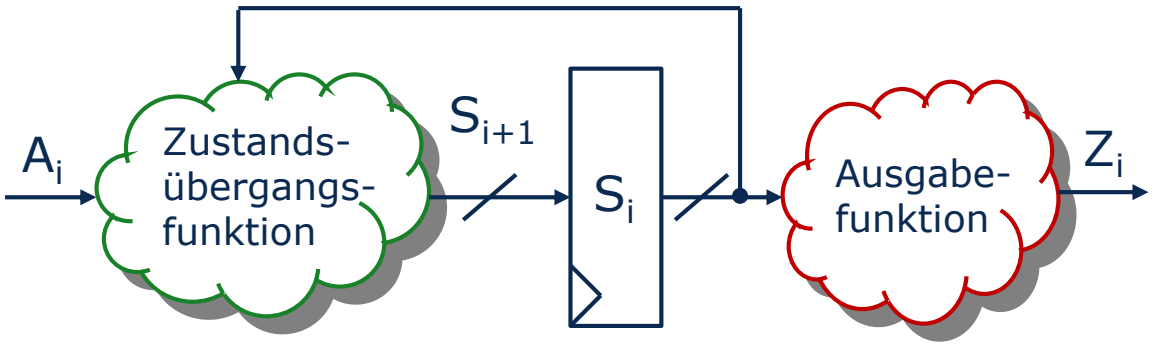




- Zustandsautomaten (Finite-State Machines (FSM)) sind Grundbestandteil digitaler Steuerwerke
- FSMs sind sequentielle digitale Systeme
- **Mealy Automat**
  - $S_{i+1} = G(A_i, S_i)$
  - $Z_i = F(A_i, S_i)$

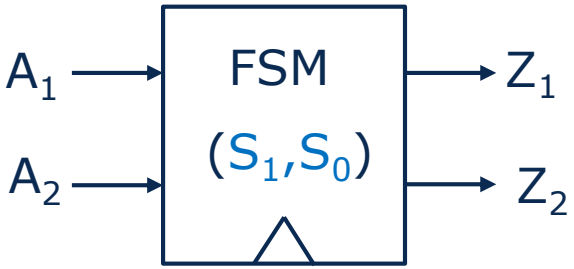


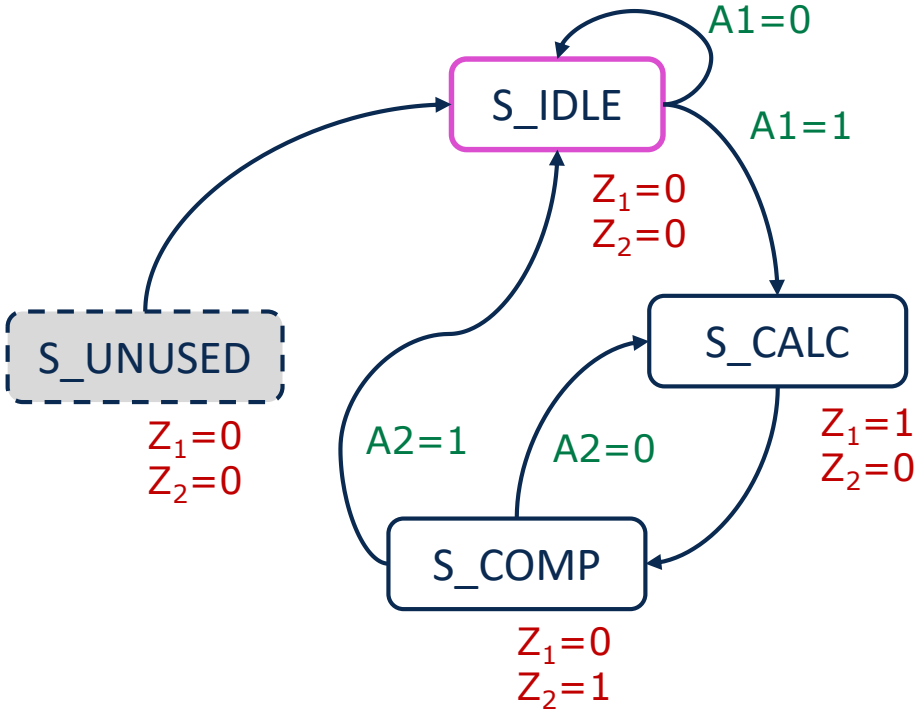
- **Moore Automat**
  - $S_{i+1} = G(A_i, S_i)$
  - $Z_i = F(S_i)$



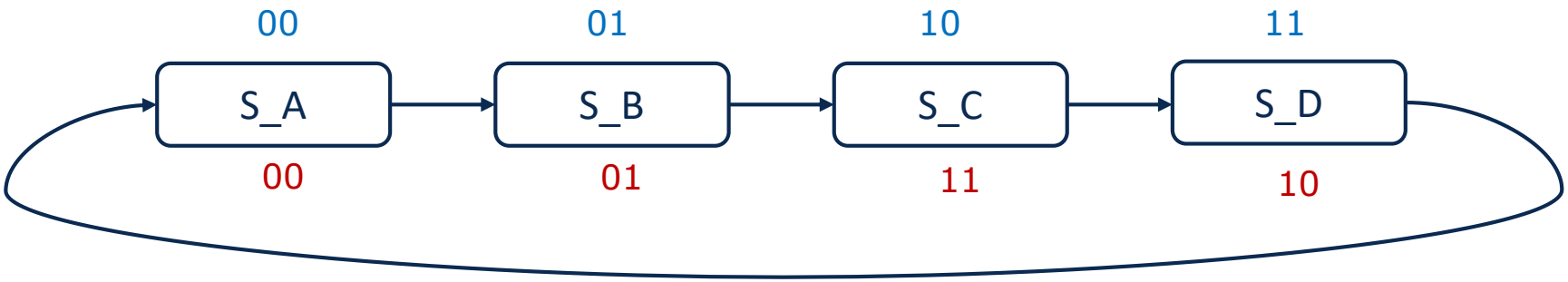
Zustandssübergangsdiagramm

- Darstellung der Zustände und ihrer Übergänge
- Eindeutige Benennung der Zustände
- Kennzeichnung des Reset Zustandes

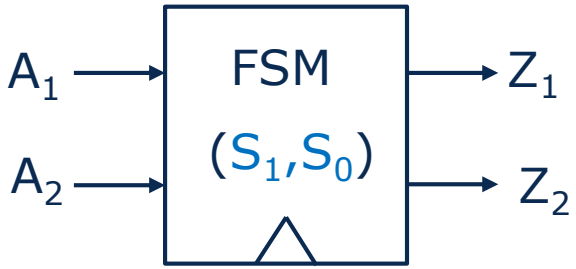


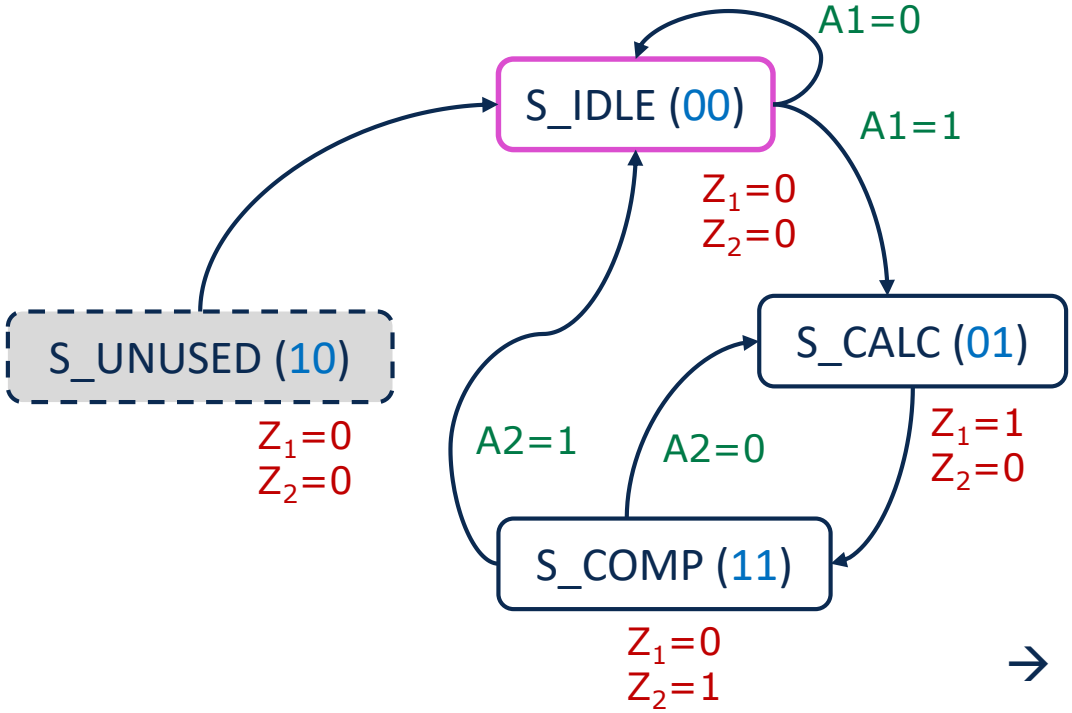


- Zuweisung eines eindeutigen Binärwortes für jeden Zustand
  - Bei m Zuständen → N Zustandsbits mit $2^N \geq m$
  - Kodierung auch der 2^N-m nicht genutzten Zustände!
- Zustandskodierung hat Einfluss auf:
  - Verlustleistungsaufnahme
    - → Reduktion der Signalwechsel im Zustandswort bei den am häufigsten erwarteten Routen durch den Zustandsgraphen
  - Komplexität der Zustandsübergangslogik und Ausgangslogik
- Beispiel:
  - Binär-Code → 6 Toggles in **S** pro Durchlauf
  - Gray-Code → 4 Toggles in **S** pro Durchlauf



Zustandsübergangskodierung





→ 2-Bit Zustandsvariable

Zustandssübergangs- und Ausgangstabellen

$S_{1,i}$	$S_{0,i}$	A_2	A_1	$S_{1,i+1}$	$S_{0,i+1}$
0	0	X	0	0	0
0	0	X	1	0	1
0	1	X	X	1	1
1	0	X	X	0	0
1	1	0	X	0	1
1	1	1	X	0	0

$S_{1,i}$	$S_{0,i}$	$Z_{2,i}$	$Z_{1,i}$
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

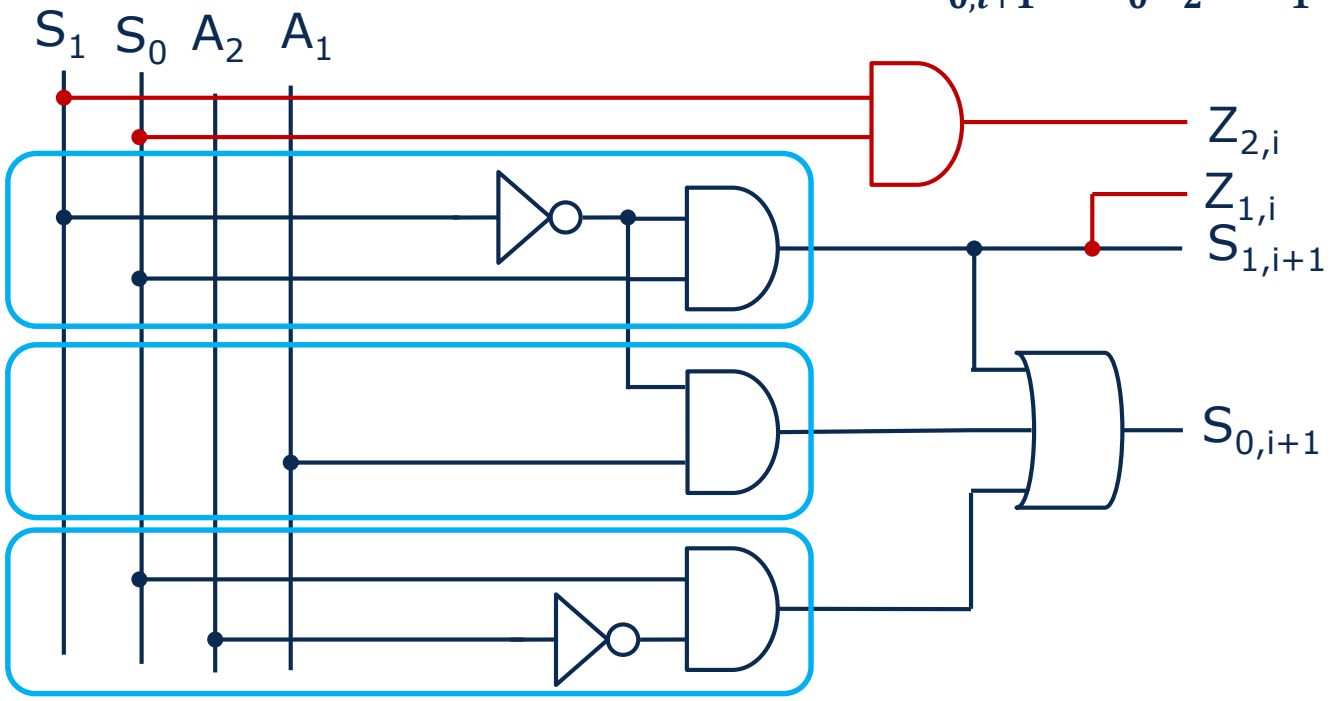
Vereinfachung der Logik und Gatternetzliste

$S_{1,i+1}$	S_1S_0	$\overline{S_1}S_0$	$\overline{S_1}\overline{S_0}$	$S_1\overline{S_0}$
A_1A_2	0	1	0	0
$\overline{A_1}A_2$	0	1	0	0
$\overline{A_1A_2}$	0	1	0	0
$A_1\overline{A_2}$	0	1	0	0

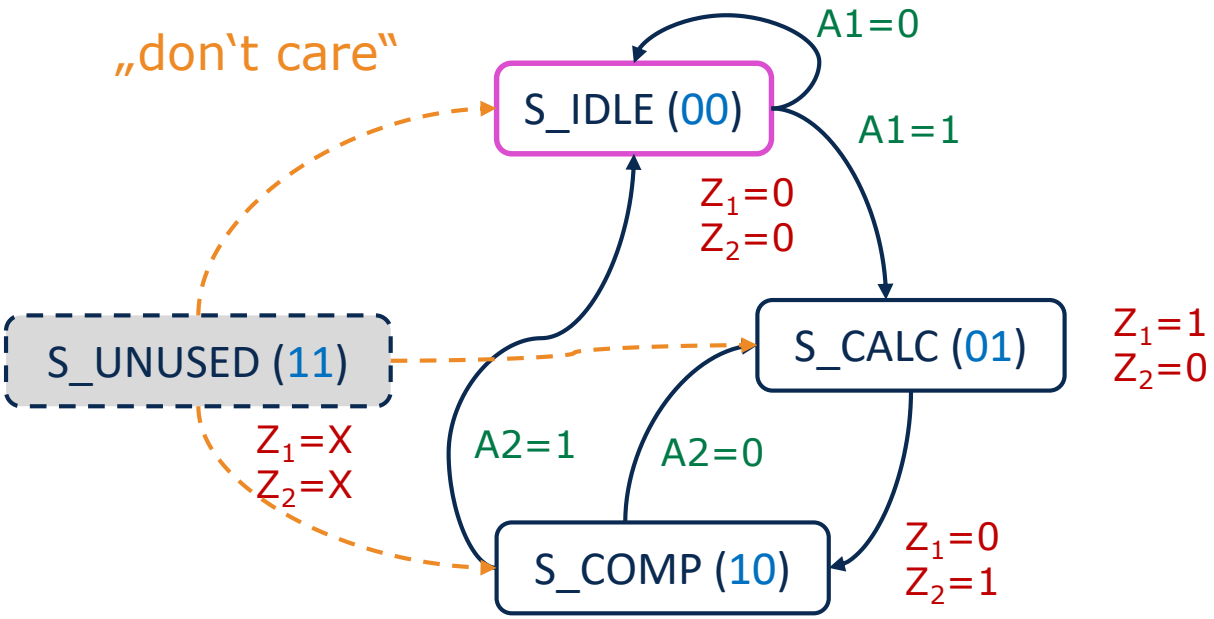
$S_{0,i+1}$	S_1S_0	$\overline{S_1}S_0$	$\overline{S_1}\overline{S_0}$	$S_1\overline{S_0}$
A_1A_2	0	1	1	0
$\overline{A_1}A_2$	0	1	0	0
$\overline{A_1A_2}$	1	1	0	0
$A_1\overline{A_2}$	1	1	1	0

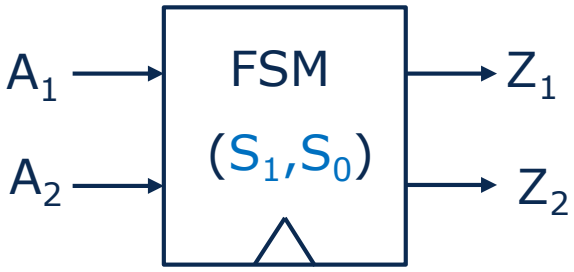
$$S_{1,i+1} = \overline{S_1}S_0$$

$$S_{0,i+1} = S_0\overline{A_2} + \overline{S_1}S_0 + \overline{S_1}A_1$$



Alternative Zustandsübergangskodierung





$S_{1,i}$	$S_{0,i}$	A_2	A_1	$S_{1,i+1}$	$S_{0,i+1}$
0	0	X	0	0	0
0	0	X	1	0	1
0	1	X	X	1	0
1	0	0	X	0	1
1	0	1	X	0	0
1	1	X	X	X	X

$S_{1,i}$	$S_{0,i}$	$Z_{2,i}$	$Z_{1,i}$
0	0	0	0
0	1	0	1
1	0	1	0
1	1	X	X

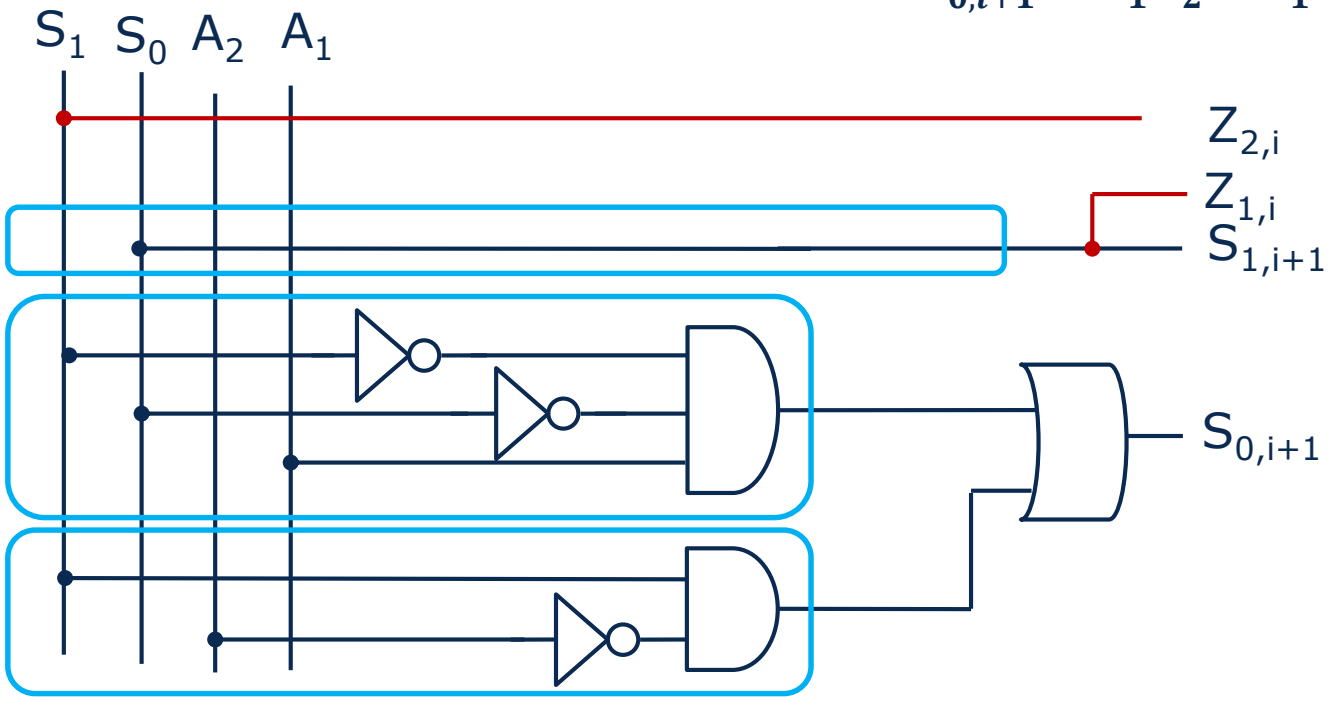
Alternative Vereinfachung der Logik und Gatternetzliste

$S_{1,i+1}$	S_1S_0	$\overline{S_1}S_0$	$\overline{S_1}\overline{S_0}$	$S_1\overline{S_0}$
A_1A_2	x	1	0	0
$\overline{A_1}A_2$	x	1	0	0
$\overline{A_1A_2}$	x	1	0	0
$A_1\overline{A_2}$	x	1	0	0

$$S_{1,i+1} = S_0$$

$S_{0,i+1}$	S_1S_0	$\overline{S_1}S_0$	$\overline{S_1}\overline{S_0}$	$S_1\overline{S_0}$
A_1A_2	x	0	1	0
$\overline{A_1}A_2$	x	0	0	0
$\overline{A_1A_2}$	x	0	0	1
$A_1\overline{A_2}$	x	0	1	1

$$S_{0,i+1} = S_1\overline{A_2} + \overline{S_1S_0}A_1$$

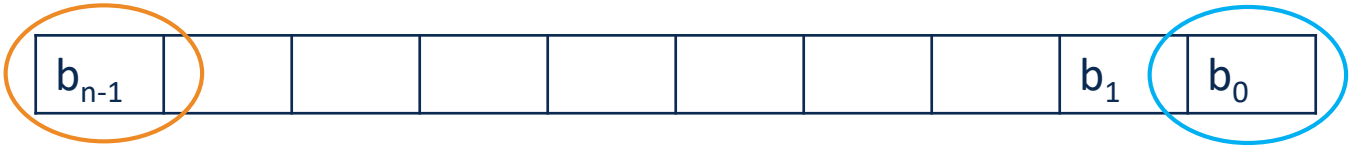


Zusammenfassung: Entwurf einer FSM

1. Zustandsübergangsdiagramm aufstellen
2. Bestimmung der Anzahl der Zustandsbits und Zustandskodierung
3. Zustandsübergangstabelle und Ausgangstabelle
4. Vereinfachung der Logik (Karnaugh, Gleichung)
5. Gatternetzliste erstellen

Arithmetik - Zahlenformate

- Darstellung vorzeichenloser (unsigned) ganzer Zahlen als n-bit Vektor
- $b \in (0; 1)$
- Dezimaler Wert: $D = \sum_{i=0}^{n-1} b_i \cdot 2^i$
- Wertebereich: $0 \leq D \leq 2^n - 1$



„most significant bit“ (MSB)

„least significant bit“ (LSB)

Dezimalwert	Binäre Darstellung	Hexadezimale Darstellung
0	0b0000	0x0
1	0b0001	0x1
2	0b0010	0x2
3	0b0011	0x3
4	0b0100	0x4
5	0b0101	0x5
6	0b0110	0x6
7	0b0111	0x7
8	0b1000	0x8
9	0b1001	0x9
10	0b1010	0xA
11	0b1011	0xB
12	0b1100	0xC
13	0b1101	0xD
14	0b1110	0xE
15	0b1111	0xF

Addition von binären Zahlen

- 1-Bit Addition a+b

a	b	Summe	Übertrag
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Beispiel Addition:

		0	1	0	1 (5)
+		1	1	0	1 (13)
Übertrag	1	1	0	1	
Summe		0	0	1	0 (2)

- Addition von n-Bit Zahlen → n+1 Bit Ergebnis

Multiplikation von binären Zahlen

- 1-Bit Multiplikation a·b

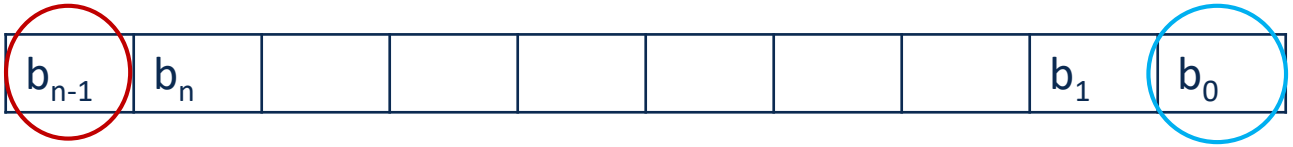
a	b	a·b
0	0	0
0	1	0
1	0	0
1	1	1

- Beispiel: 0b1001 · 0b0101 (9·5)

					1	0	0	1	
					1	0	0	1	1
	+			0	0	0	0		0
	+		1	0	0	1			1
	+	0	0	0	0				0
Produkt	0	0	1	0	1	1	0	1	(45)

- Multiplikation von n-Bit Zahlen → 2n-Bit Ergebnis

- Darstellung vorzeichenbehafteter (signed), ganzer Zahlen als n-bit Vektor
- $b \in (0;1)$
- Zweierkomplement Darstellung
- Dezimaler Wert:
  - positive Zahl: wenn $b_{n-1}=0$: $D = \sum_{i=0}^{n-1} b_i \cdot 2^i$
  - negative Zahl: wenn $b_{n-1}=1$: $D = -1 \cdot (\sum_{i=0}^{n-1} \overline{b_i} \cdot 2^i + 1)$
- Wertebereich: $-2^{n-1} \leq D \leq 2^{n-1} - 1$



„sign bit“

„least significant bit“ (LSB)

Beispiele

Dezimalwert	Binäre Signed Darstellung (Zweierkomplement)
0	0b0000
1	0b0001
2	0b0010
3	0b0011
4	0b0100
5	0b0101
6	0b0110
7	0b0111
-8	0b1000
-7	0b1001
-6	0b1010
-5	0b1011
-4	0b1100
-3	0b1101
-2	0b1110
-1	0b1111

- Berechnung von $-1 \cdot A$
  1. Invertierung aller Bits von A
  2. Addition von +1

- Beispiel:
  - -1·3 =-1·(0b0011)→ 0b1100+0b0001=0b1101 =-3
  - -1·(-3) =-1·(0b1101)→ 0b0010+0b0001=0b0011 = 3

Addition/Subtraktion vorzeichenbehafteter Zahlen

- Beispiel 1:

		0	0	1	0	(2)
	+	1	1	1	0	(−2)
Übertrag	1	1	1	0		
Summe		0	0	0	0	(0)

OK

- Beispiel 2:

		1	1	1	0	(−2)
	+	1	1	1	0	(−2)
Übertrag	1	1	1	0		
Summe		1	1	0	0	(−4)

OK

- Beispiel 3:

		0	1	0	0	(4)
	+	0	1	1	0	(6)
Übertrag	0	1	0	0		
Summe		1	0	1	0	(−6)

Error
Addition von pos. Zahlen muss pos. sein!

- Beispiel 4:

		1	1	0	0	(−4)
	+	1	0	1	0	(−6)
Übertrag	1	0	0	0		
Summe		0	1	1	0	(6)

Error
Addition von neg. Zahlen muss neg. sein!

Carry und Overflow

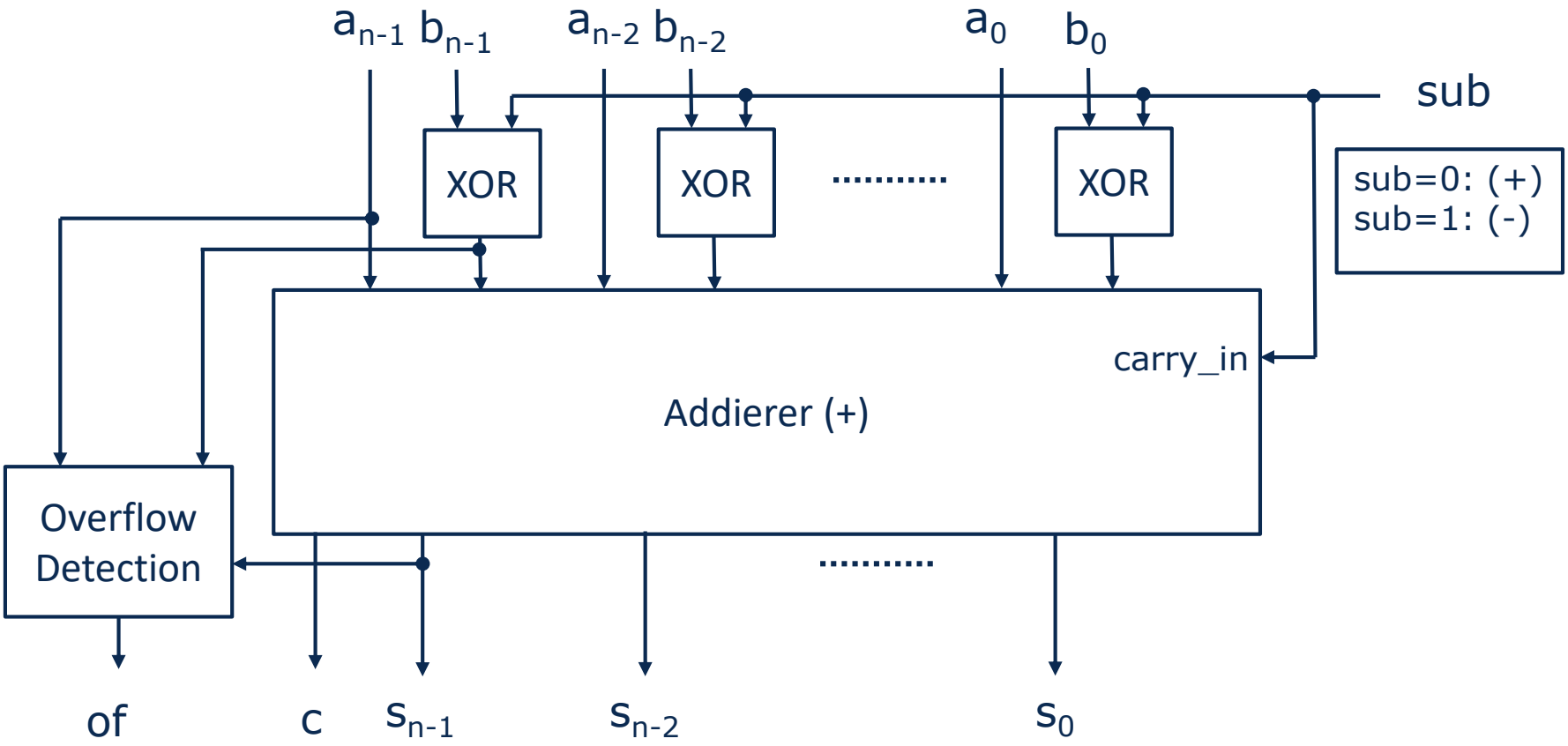
- Carry: Übertrag von der höchsten binärstelle (MSB) im **unsigned** Bereich
- Overflow: Detektion von Überläufen des **signed** Bereich

Sign(A)	Sign(B)	Sign(A+B)	Overflow
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Dezimalwert	Unsigned Darstellung	Signed Darstellung
Carry Bit	**Error**	nicht relevant
Overflow Bit	nicht relevant	**Error**

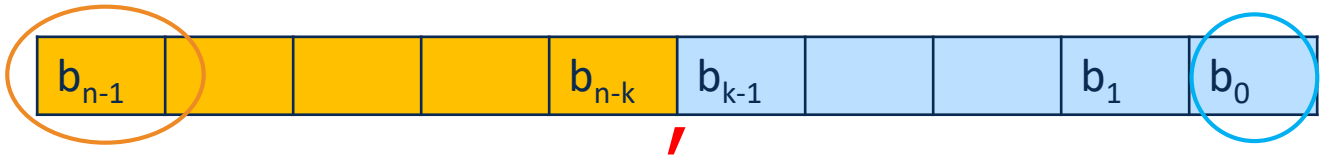
Addition/Subtraktion vorzeichenbehafteter Zahlen

- Addition und Subtraktion kann mit der gleichen Schaltung erfolgen



Festkomma Darstellung (Fixed-Point)

- Darstellung von Zahlen mit Nachkommastellen als n-bit Vektor
- k Nachkommastellen und n-k Vorkommastellen
- Wertigkeit des LBS: 2^{-k}
- Dezimaler Wert: $D = \sum_{i=0}^{n-1} b_i \cdot 2^{i-k}$
- Wertebereich: $0 \leq D \leq (2^n - 1) \cdot 2^{-k}$



„most significant bit" (MSB) „Komma" „least significant bit" (LSB)

Darstellung vorzeichenbehafteter Fixed-Point Zahlen analog zu ganzen Zahlen (Skalierung mit 2^{-k})

- Addition/Subtraktion von n-Bit Fixed-Point Zahlen:
  - $A=a\cdot 2^{-k}$, $B=b\cdot 2^{-k} \rightarrow A\pm B=(a\pm b)\cdot 2^{-k}$
  - (n+1)-Bit Ergebnis

- Beispiel 1:
  - n=4, k=2, $2^{-k}=0{,}25$, unsigned

		0	1	1	0	(1,50)
	+	1	0	0	1	(2,25)
Übertrag	0	0	0	0		
Summe		1	1	1	1	(3,75)

- Beispiel 2:
  - n=4, k=2, $2^{-k}=0{,}25$, signed

		0	1	1	0	(1,50)
	+	1	0	1	1	(−1,25)
Übertrag	1	1	1	0		
Summe		0	0	0	1	(0,25)

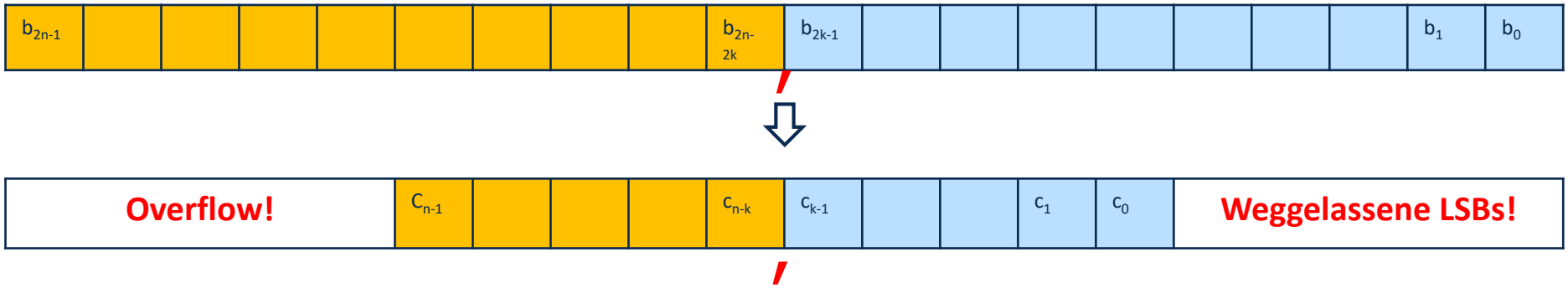
- Multiplikation von n-Bit Fixed-Point Zahlen:
  - $A=a\cdot 2^{-k}$, $B=b\cdot 2^{-k}$ → $A\cdot B=(a\cdot b)\cdot 2^{-2k}$
  - → Verdopplung der Anzahl der Nachkommastellen
  - 2n-Bit Ergebnis mit 2^{-2k} LSB Genauigkeit

- Beispiel:
  - n=4, k=2, $2^{-k}=0{,}25$, unsigned
  - 0b1001 · 0b0101 (2,25 · 1,25)

					1	**0**	**0**	**1**	
					1	0	0	1	1
	+			0	0	0	0		0
	+		1	0	0	1			1
	+	0	0	0	0				0
Produkt	0	0	1	0	1	1	0	1	(2,8125)

Skalierung des Ergebnisses

- Multiplikation von n-Bit Werten → 2n-Bit Ergebnis
- Bei n-Bit Datenbussen ist Skalierung notwendig c=Scale(b)



- Im Praktikum:
  - **SCALER64_to_32**: Ermittlung von Overflows (signed, unsigned)
  - Angabe der Anzahl k LSBs
  - Nur Verwenden wenn Overflow Signale benötigt werden
  - Sonst: Direktes Verdrahten der Bussignale

Fixed-Point Division

- Division von n-Bit Fixed-Point Zahlen:
  - $A=a\cdot 2^{-k}$, $B=b\cdot 2^{-k}$ → $A/B=(a/b)\cdot 2^{-k+k}=a/b$
- Bei n-Bit Operanden:
  - n-Bit ganzzahliges Ergebnis + n-Bit **Rest (Modulo)**
    - Beispiel: 0b1101 / 0b0011 (13/3) = 0b0100 Rest: 0b0001
    - Praktikum: **DIV_FIXED32_signed** und **DIV_FIXED32_unsigned**
  - n-Bit ganzzahliges Ergebnis + unendlich viele Nachkommastellen
    - Beispiel: 0b1101 / 0b0011 (13/3) =0b0100,010101… (4,33333…)
    - Im Praktium: **DIV_FIXED64_signed (**32 Vorkomma, und Nachkommastellen, begrenzte Genauigkeit!)

Fließkommadarstellung (Floating Point)

- Probleme bei Festkommadarstellung:
  - Eingeschränkter Wertebereich
  - Fixed-Point: Kompromiss zwischen Wertebereich und Genauigkeit

- Zahlendarstellung in der Form: $X = S \cdot M \cdot 2^E$
  - Basis: 2
  - S: Vorzeichen (sign) → 1-Bit
  - M: Mantisse → unsigned fixed-point
  - E: Exponent (variabel) → signed integer

- Sehr hohe Präzision bei kleinen Zahlen
- Sehr großer Wertebereich bei großen Zahlen (bei geringerer Präzision)

- Fließkommadarstellungen sind Näherungen!

Beispiel Floating Point

- Beispiel: Zahl 5 als Fließkommadarstellung
  - $5= +1 \cdot 1{,}25 \cdot 2^2$
  - $S=0$
  - $M=0b1{,}01\_0000\_0000\_0000\_0000\_0000$
  - $E=0b00000010$

- Aber auch: $5= +1 \cdot 0{,}625 \cdot 2^3$, $5= +1 \cdot 0{,}3125 \cdot 2^4$, $5= +1 \cdot 0{,}15625 \cdot 2^5$, …
- Die Fließkommadarstellungen ist nicht eindeutig bestimmt

- → Normierung des Wertebereichs der Mantisse
- z.B. $1 \leq M < 2$

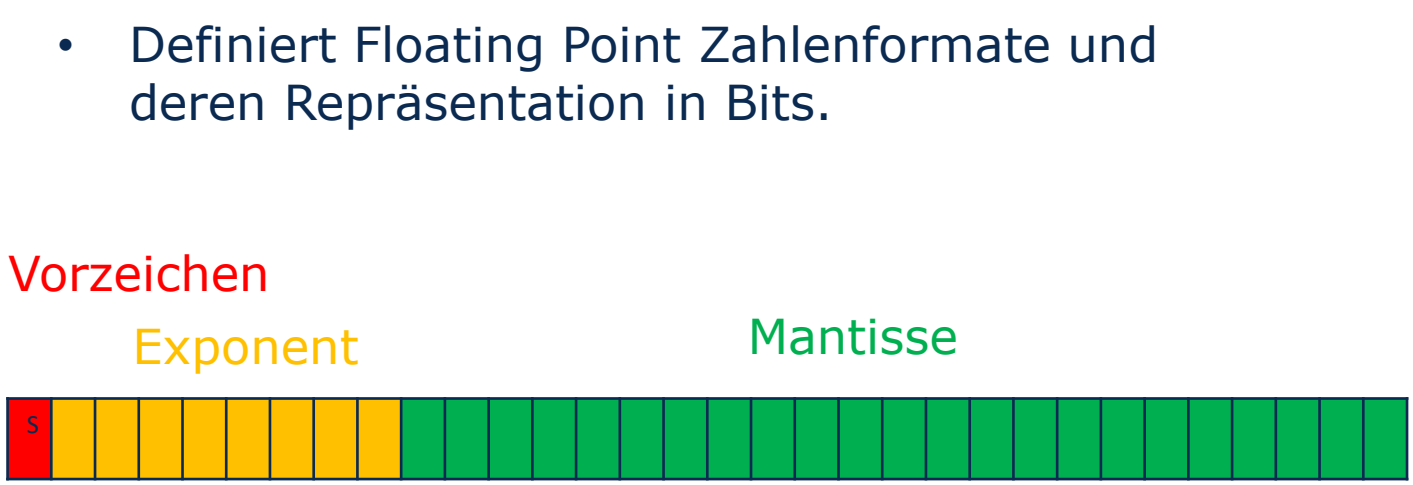
Bias

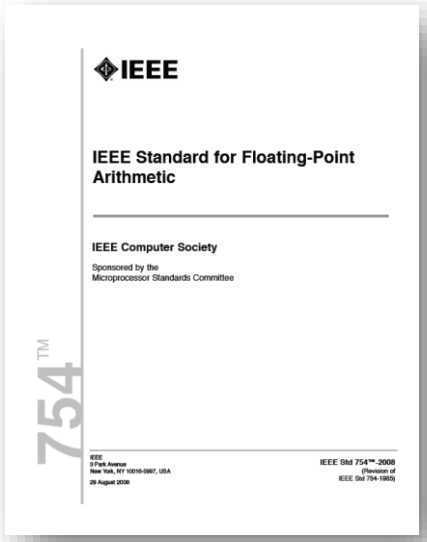
- Der Exponent einer Fließkommazahl ist vorzeichenbehaftet (signed)
- Hardware für signed Operationen notwendig

- Speicherung des Exponenten als E`=E+B als vorzeichenlose (unsigned) Zahl
  - B: Bias
  - Beispiel: 8-Bit Exponent: B=127

- Ermöglicht Nutzung von unsigned Arithmetik Komponenten für Floating Point Units

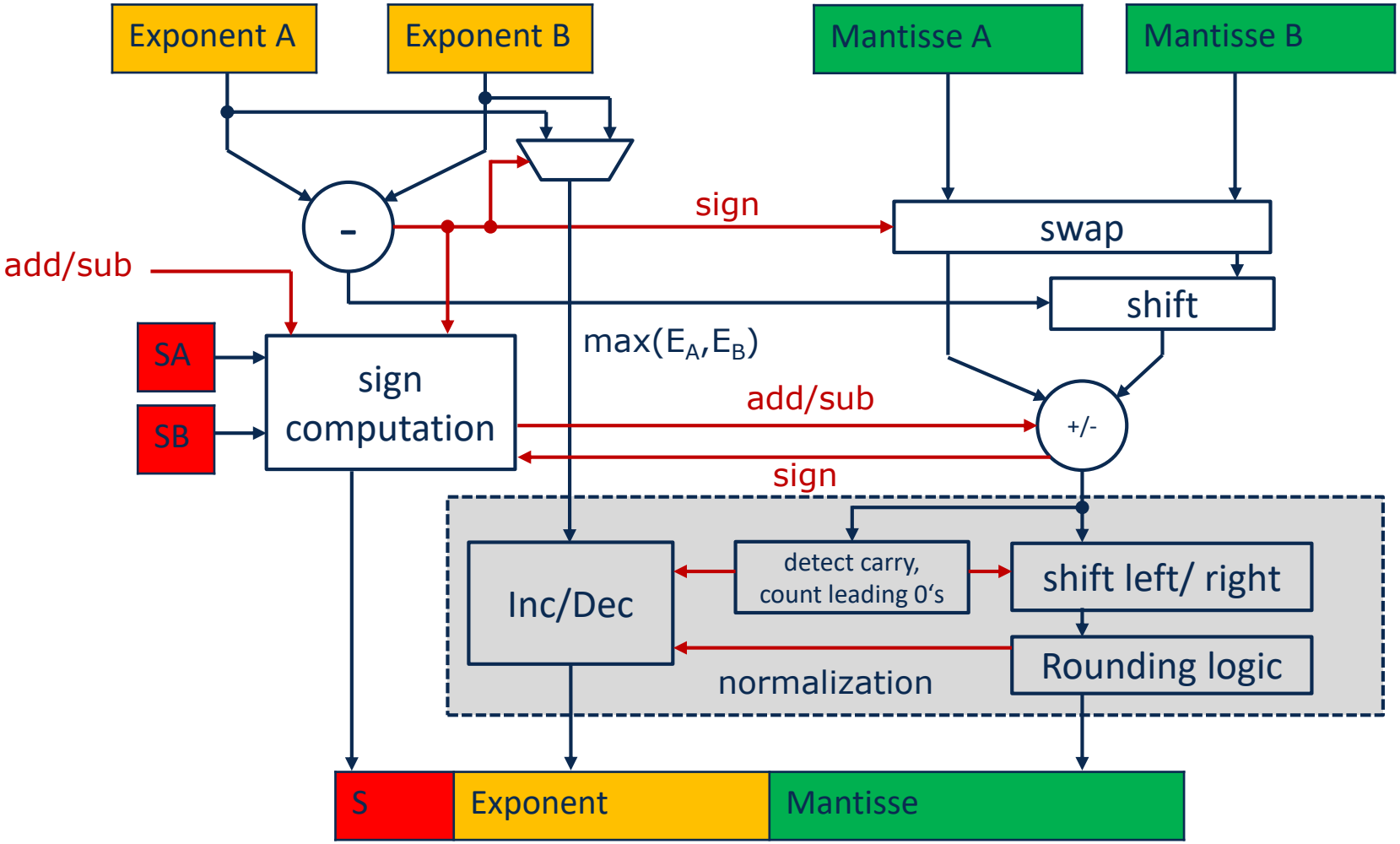
- Definiert Floating Point Zahlenformate und deren Repräsentation in Bits.



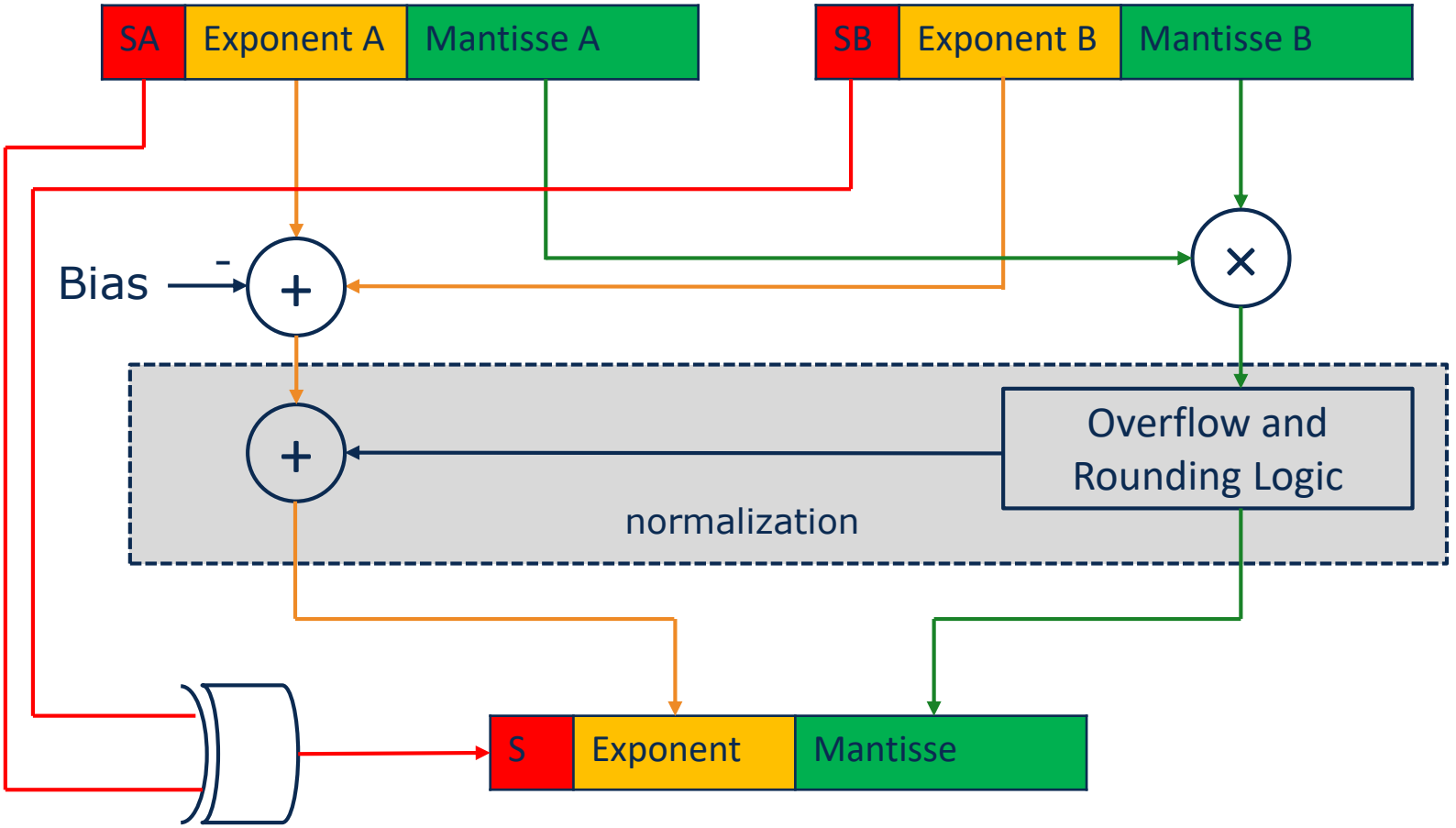


	Bits	Mantisse (Bit)	Exponent (Bit)	Bias	Wertebereich	Genauigkeit (Dezimale Nachkommastellen)
Single (float)	32	23	8	127	10^{-38} bis 10^{38}	≈7
Double	64	52	11	1023	10^{-308} bis 10^{308}	≈16

Floating Point Addierer Prinzipschaltung



Floating Point Multiplizierer Prinzipschaltung



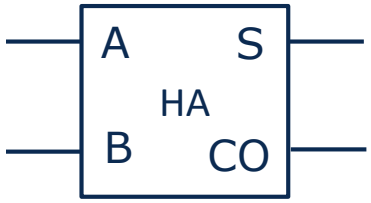
Zusammenfassung

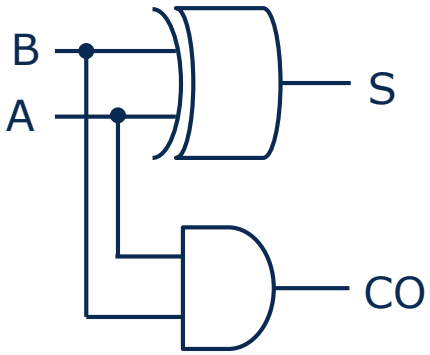
- Zahlenformate Fixed-Point (singed, unsigned) und Floating Point
- Höhere Genauigkeit erfordert mehr Aufwand bezüglich:
  - Höherer Speicherbedarf
  - Größere Chipfläche für Arithmetik
  - Längere Logiklaufzeiten → Häufig Realisierung komplexer Arithmetik Blöcke in mehreren Taktzyklen (Pipelining)
  - Höheren Energieaufwand der Berechnung
- Wahl des Zahlenformates ist entscheidend für die Effizienz der Hardwareimplementierung

Arithmetik - Schaltungen

Halbaddierer (HA)

- Addition von 2-Bit
  - Eingänge A, B
  - Ausgänge S (Sum), CO (Carry)





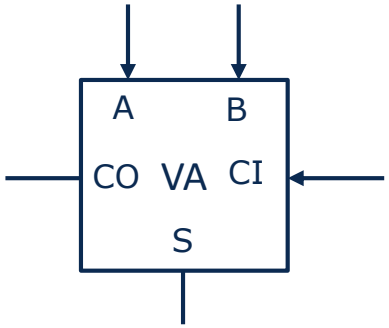
A	B	S	CO
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$\boldsymbol{S = A \oplus B}$$

$$\boldsymbol{CO = A \cdot B}$$

Volladdierer (VA)

- Addition von 2 Bit und Carry
  - Eingänge A, B, CI (Carry In)
  - Ausgänge S (Sum), CO (Carry)



A	B	CI	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

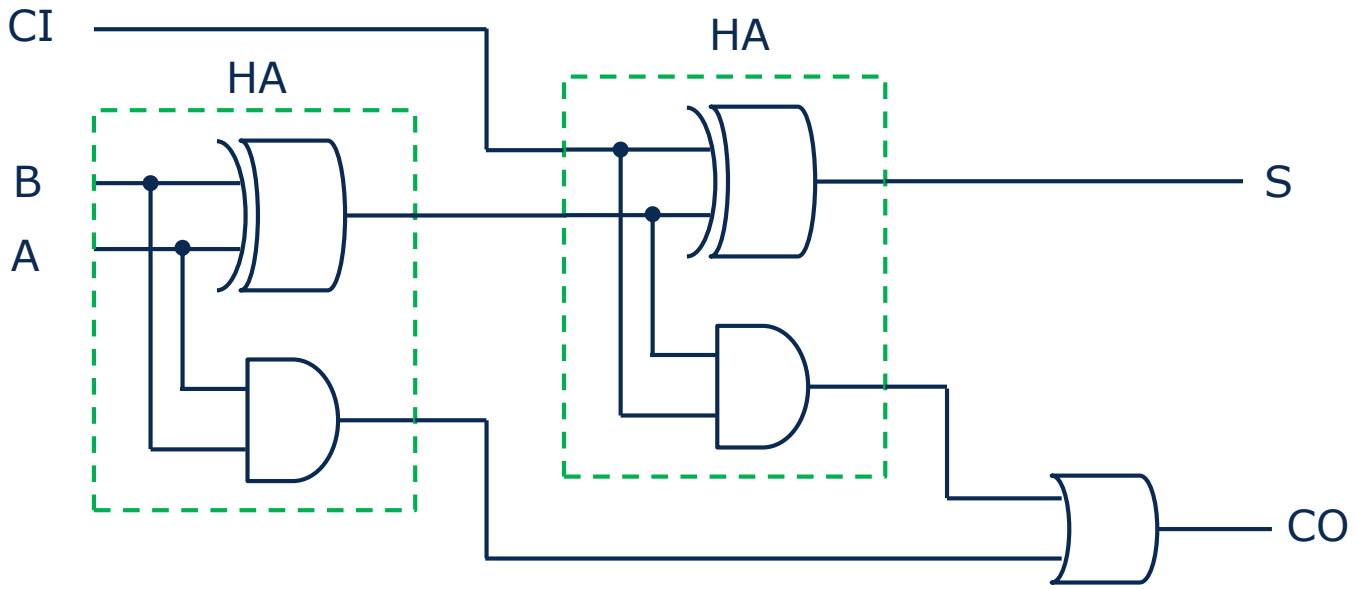
$$\boldsymbol{S = A \oplus B \oplus CI}$$

$$\boldsymbol{CO = A \cdot B + CI \cdot (A \oplus B)}$$

Volladdierer Gatterschaltung

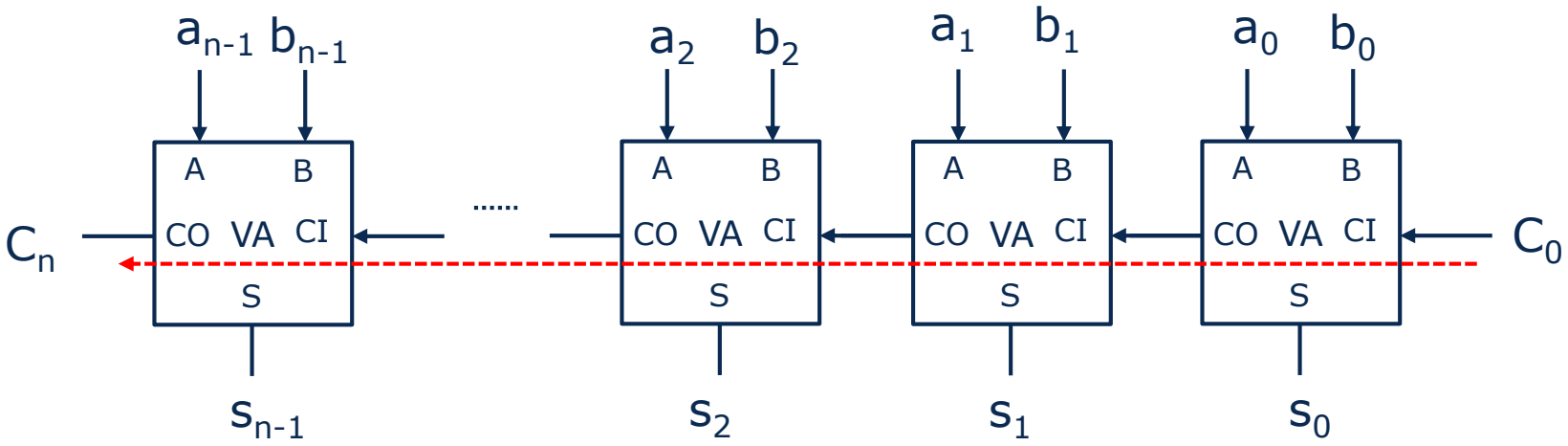
$$S = A \oplus B \oplus CI$$

$$CO = A \cdot B + CI \cdot (A \oplus B)$$



Carry Pfad CI → CO: 2 Gatterstufen

- Verkettung von Volladdierern zur Addition von n-Bit Zahlen
- Wenn kein CI benötigt wird, kann erste Stufe durch HA ersetzt werden



Kritischer Pfad: CI → CO: 2*n Gatterstufen

- Beispiel:
  - 32-Bit Addierer → 64 Gatterverzögerungen
  - 28nm: t_{gate}≈50ps → T=3,2ns → f_{max}≈310MHz
- Stand der Technik: 32-Bit Prozessoren bei >2GHz → Wie geht das?

Carry Generate and Propagate

- Carry Pfade sind Timing kritisch!
- → Reduktion der Logikstufen im Carry Pfad
- Einführen neuer Größen:
  - P: Propagate: Weiterreichen des Carry
  - G: Generate: Erzeugen des Carry

A	B	CI	S	CO	P	G
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	1	0	1	0
0	1	1	0	1	1	0
1	0	0	1	0	1	0
1	0	1	0	1	1	0
1	1	0	0	1	0	1
1	1	1	1	1	0	1

$$S = A \oplus B \oplus CI$$

$$CO = A \cdot B + CI \cdot (A \oplus B)$$

$$P = A \oplus B$$

$$G = A \cdot B$$

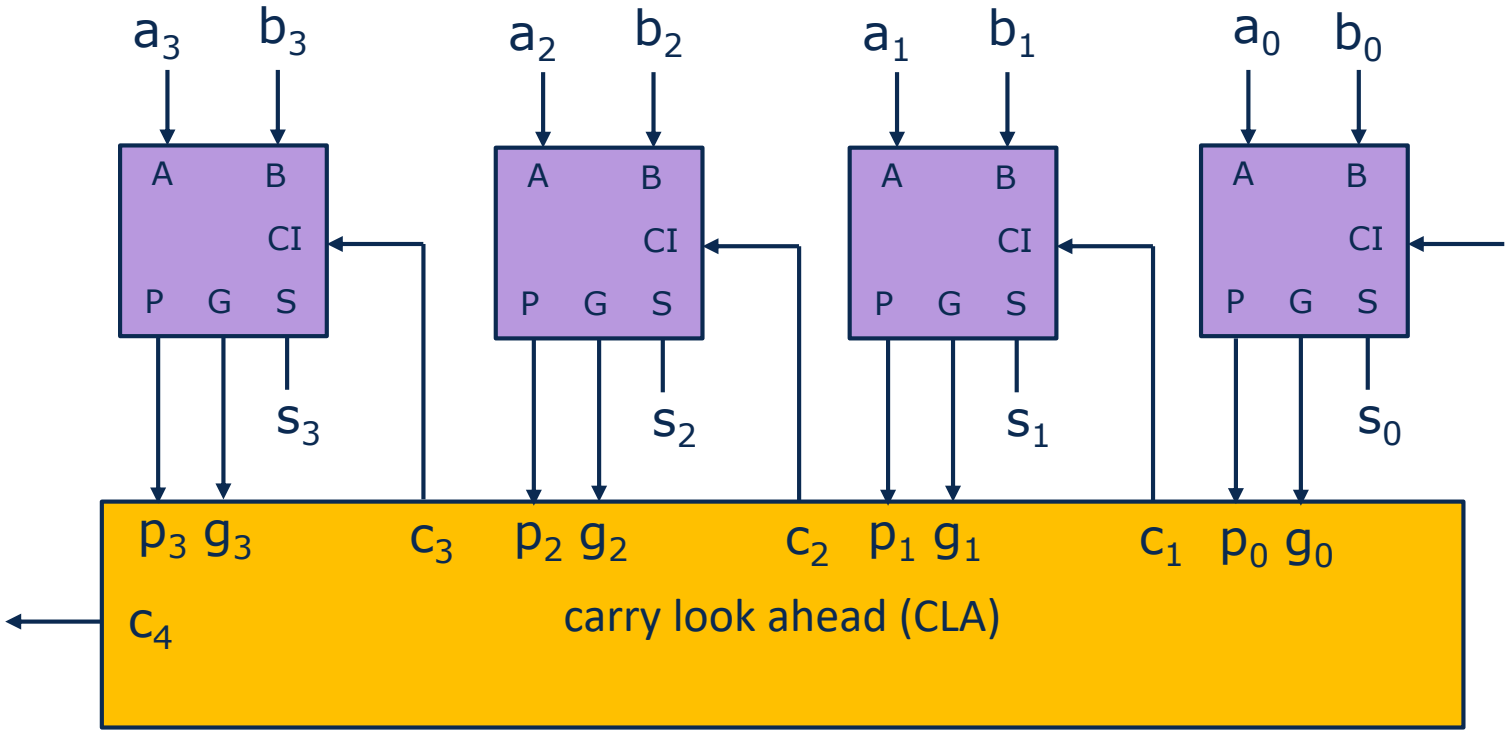
$$S = P \oplus CI$$

$$CO = G + CI \cdot P$$

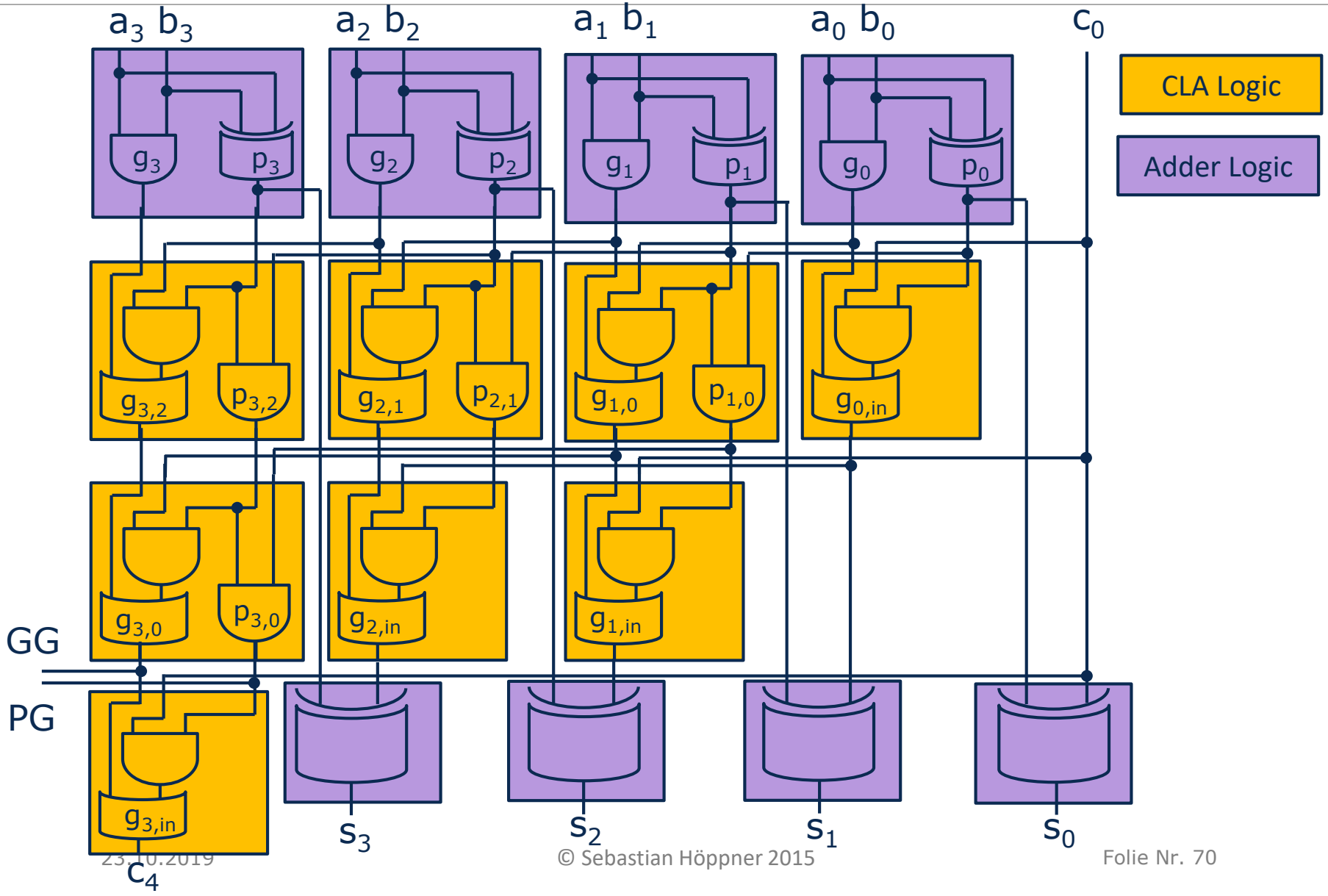
- Vorausberechnung der Carry Bits → Carry Look Ahead
- Beispiel n=4:
  - $C_1=G_0+C_0P_0$
  - $C_2=G_1+C_1P_1=G_1+G_0P_1+C_0P_0P_1$
  - $C_3=G_2+C_2P_2=G_2+G_1P_2+G_0P_1P_2+C_0P_0P_1P_2$
  - $C_4=G_3+C_3P_3=G_3+G_2P_3+G_1P_2P_3+G_0P_1P_2P_3+C_0P_0P_1P_2P_3$

Benötigt nur 5 Logikstufen!

Carry Look Ahead Addierer

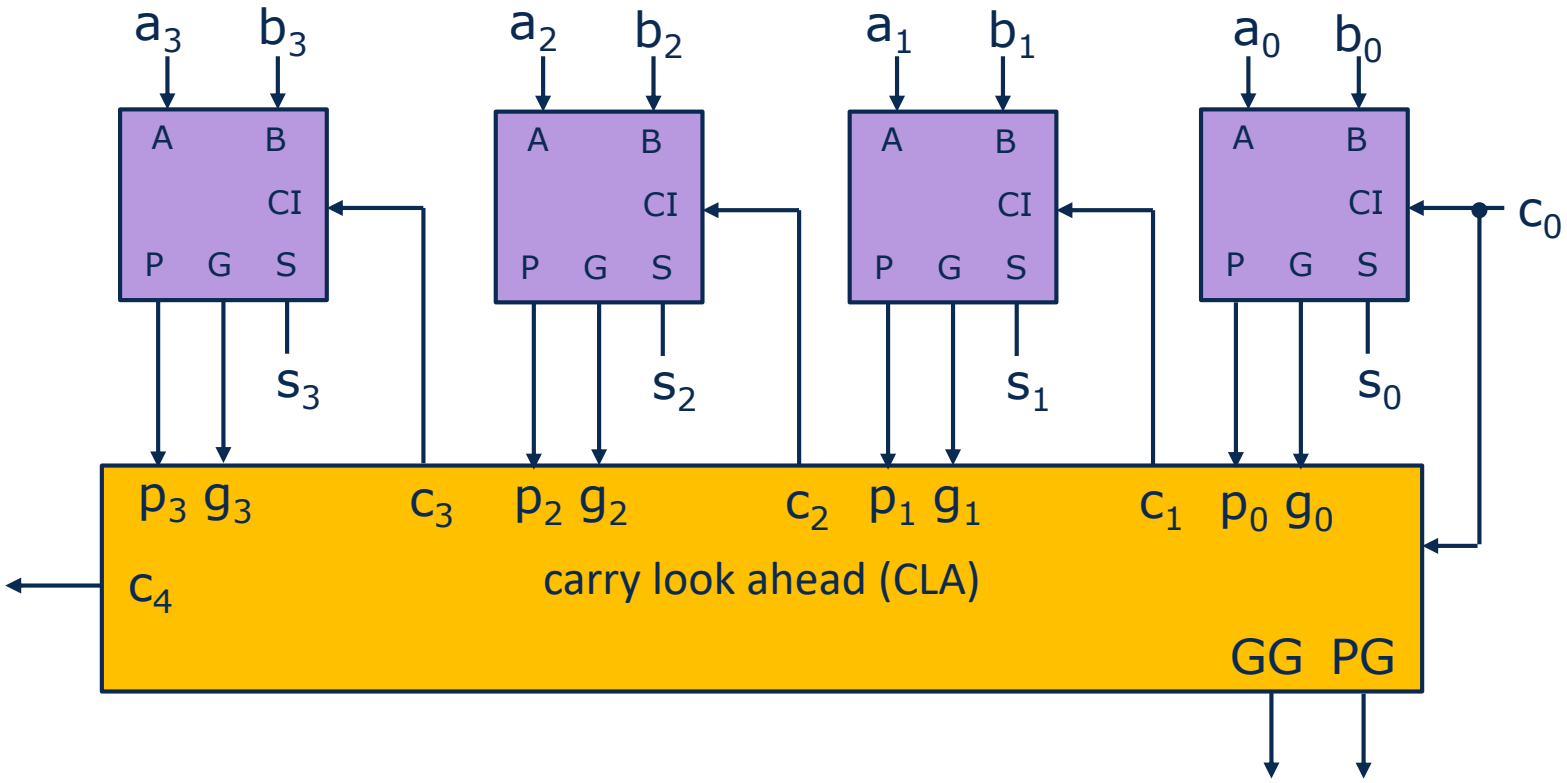


Carry Look Ahead Logik: „Kogge Stone Addierer“



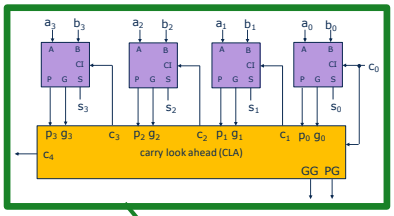
Group Propagate und Group Generate

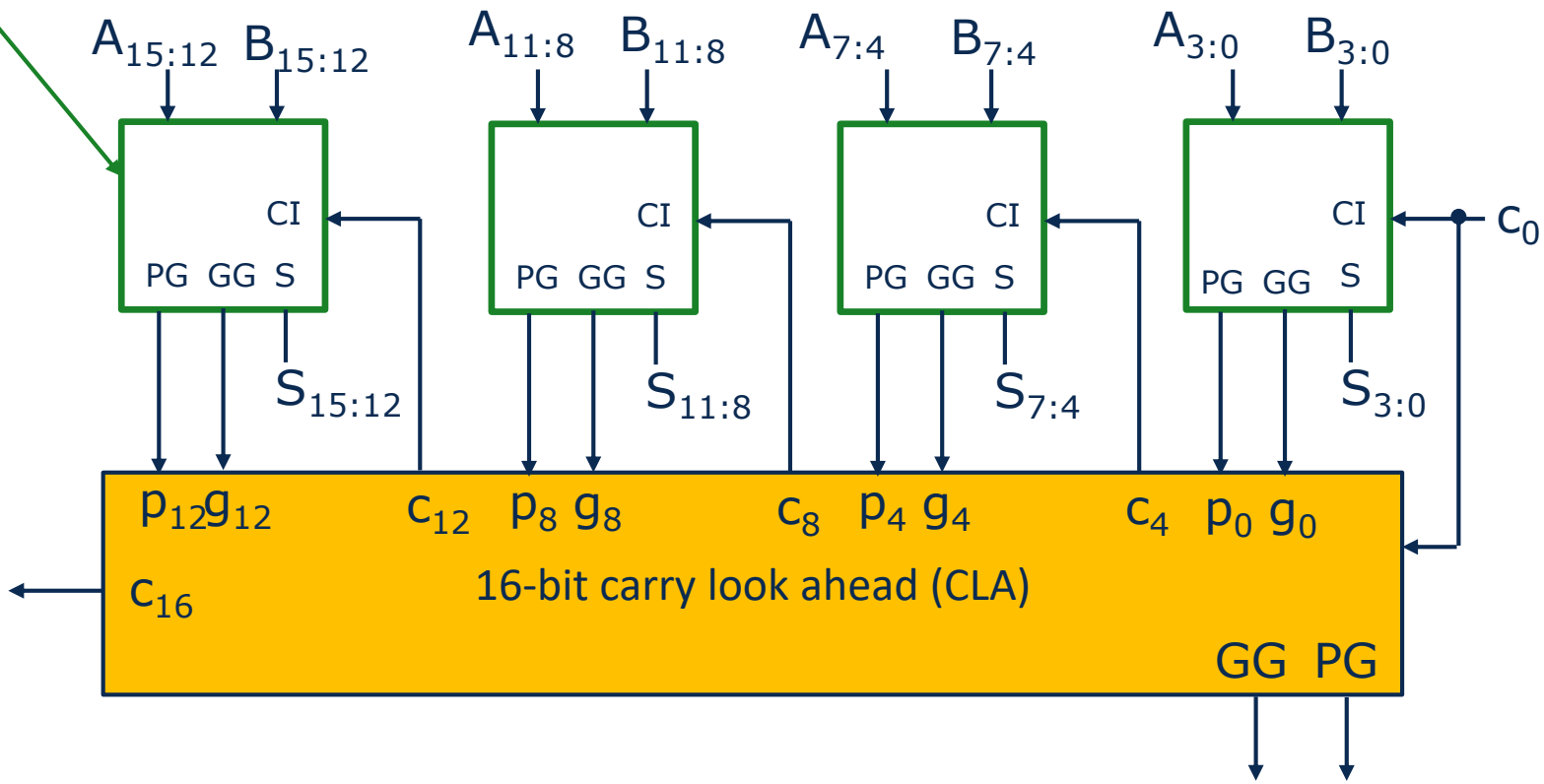
- Vorausberechnung der von Generate und Propagate für eine n-Bit Gruppe

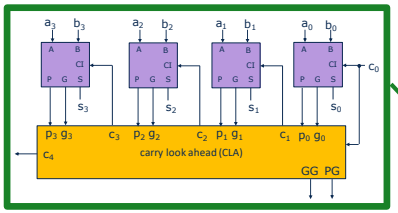


- Group Propagate: $PG=P_3P_2P_1P_0$ → 2+1 Gatterstufen
- Group Generate: $GG=G_3+G_2P_3+G_1P_3P_2+G_0P_3P_2P_1$ → 4+1 Gatterstufen

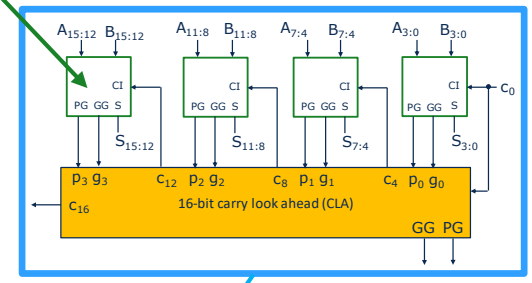
4-Bit CLA Adder





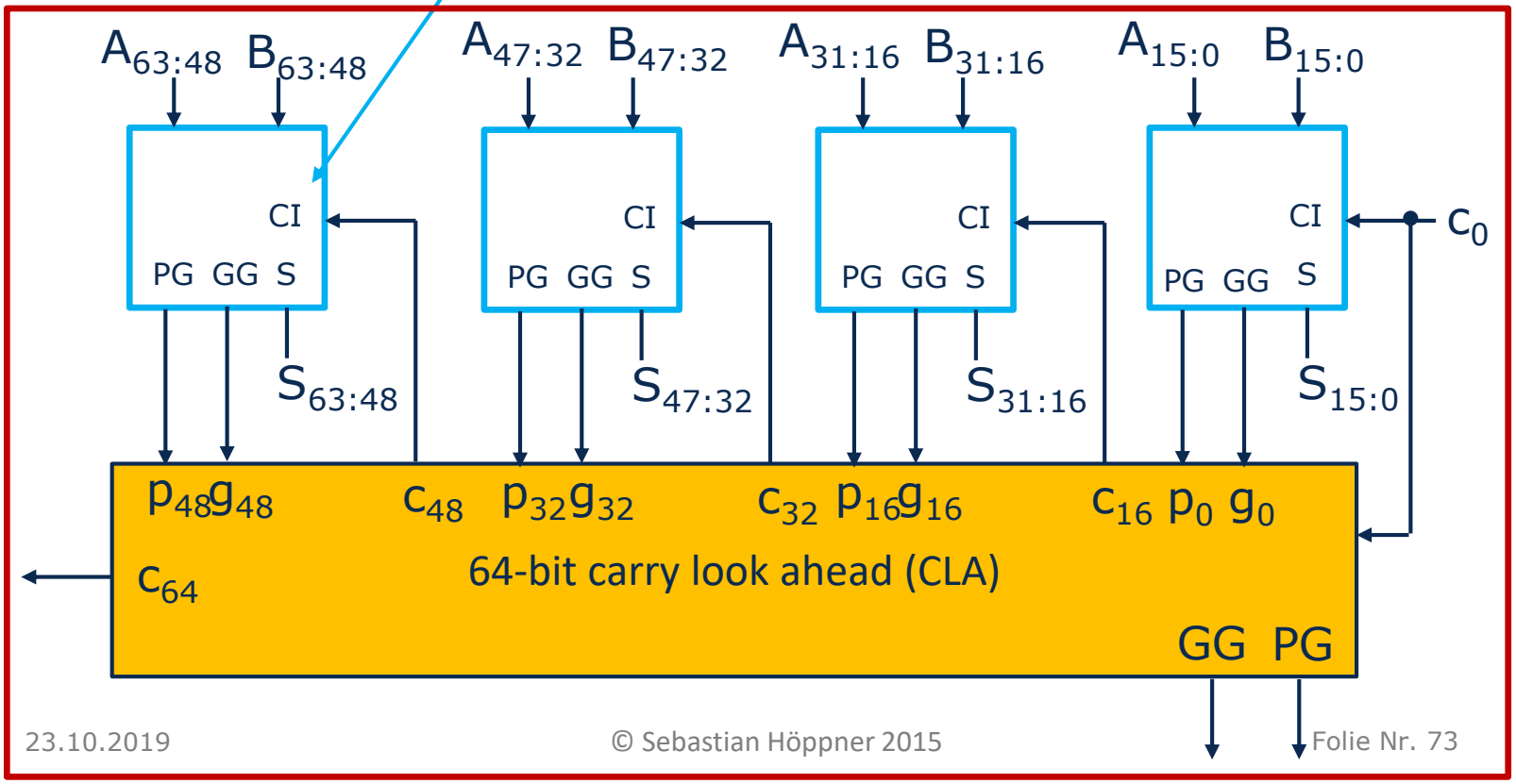


4-Bit CLA Adder

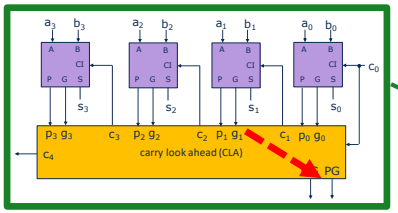


16-Bit CLA Adder

64-Bit CLA Adder

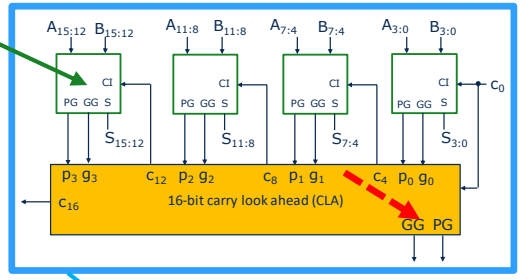


Partitionierung von schnellen Addierern



4-Bit CLA Adder

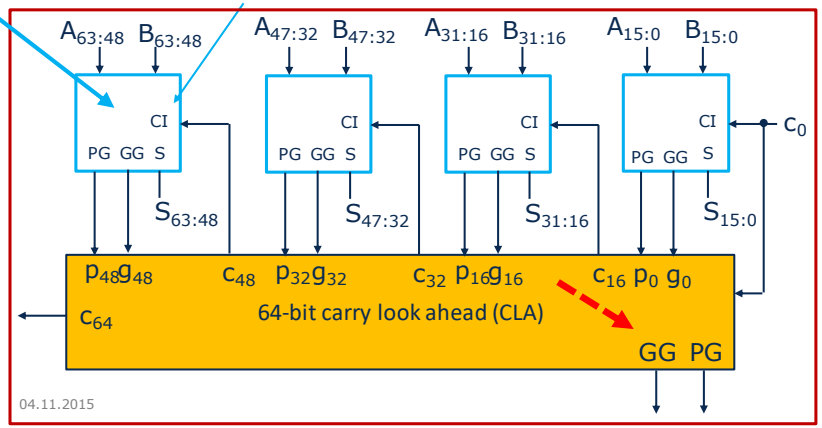
Carry Pfad (G)
4 Gatterstufen



16-Bit CLA Adder

Carry Pfad (G) 4 Gatterstufen

64-Bit CLA Adder



Carry Pfad (G) 4 Gatterstufen

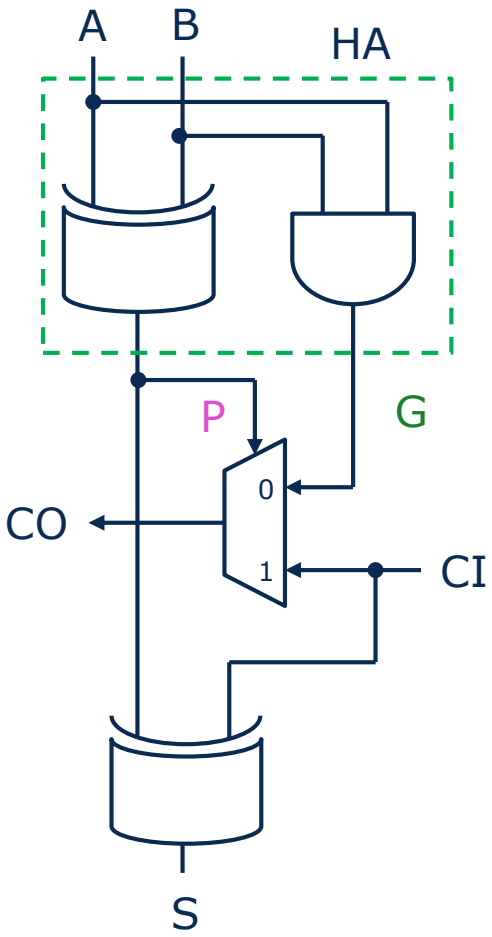
- → 1+3·4+2=15 Gatterstufen für C64
- 64-Bit Ripple Carry Adder hätte ≈128 Gatterstufen

„Manchester Chain“ Volladdierer

- Schnelle Carry Weiterleitung durch Multiplexer
- Homogene Hardware Realisierung von Addierer Ketten

$$S = P \oplus CI$$

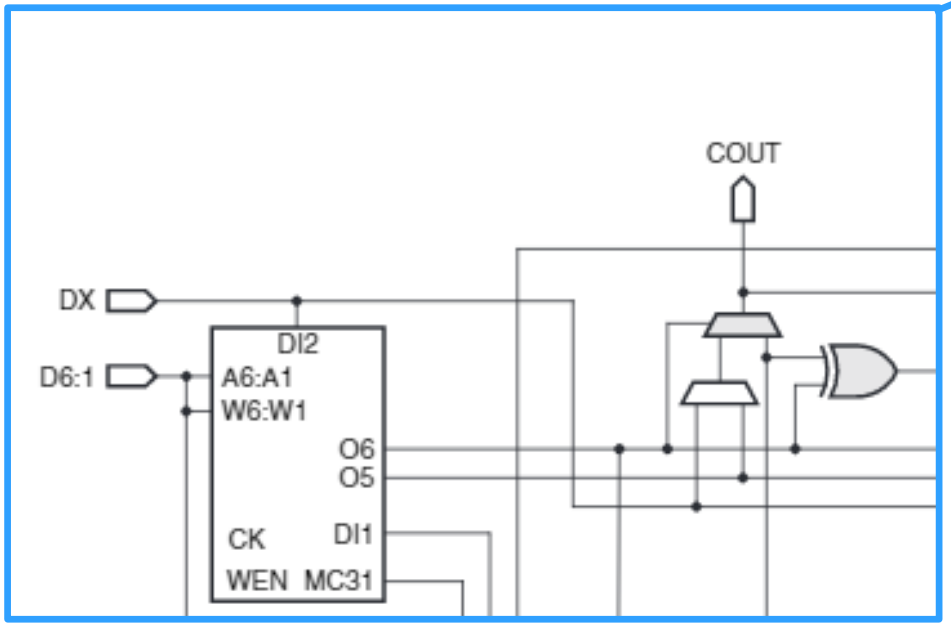
$$CO = G + CI \cdot P$$



Carry Pfad CI → CO: 1 Multiplexer

Anwendungsbeispiel: Carry Chain in FPGA Logik Blöcken

- Schnelle Carry Pfade in Field-Programmable Gate Arrays (FPGAs)
- Realisierung von schnellen Addierern und Subtrahierern mit konfigurierbarer Wortbreite



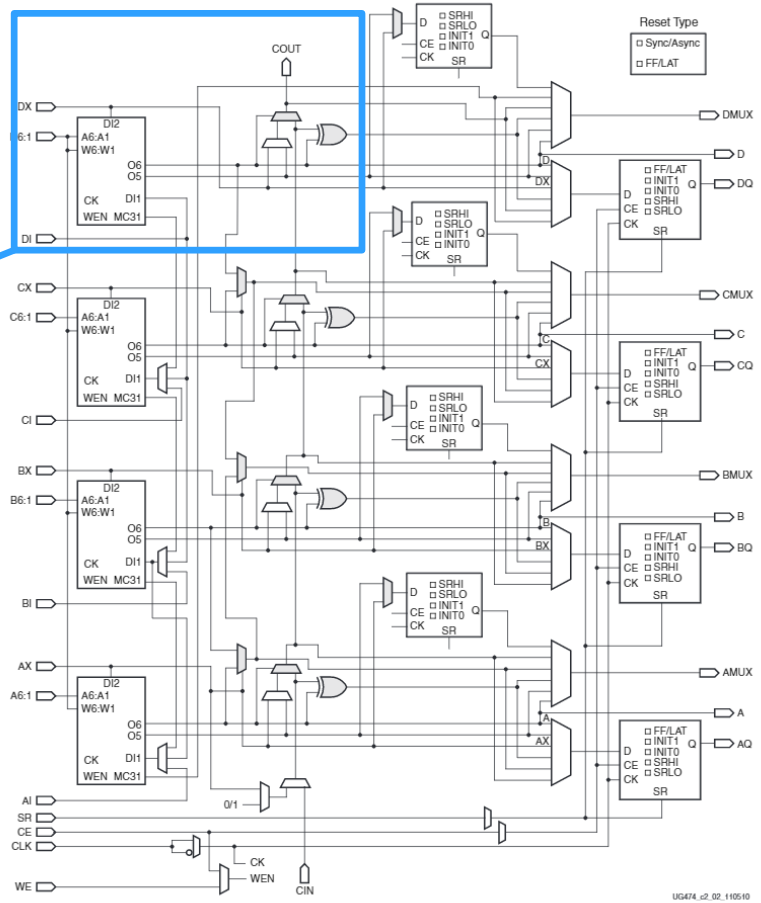


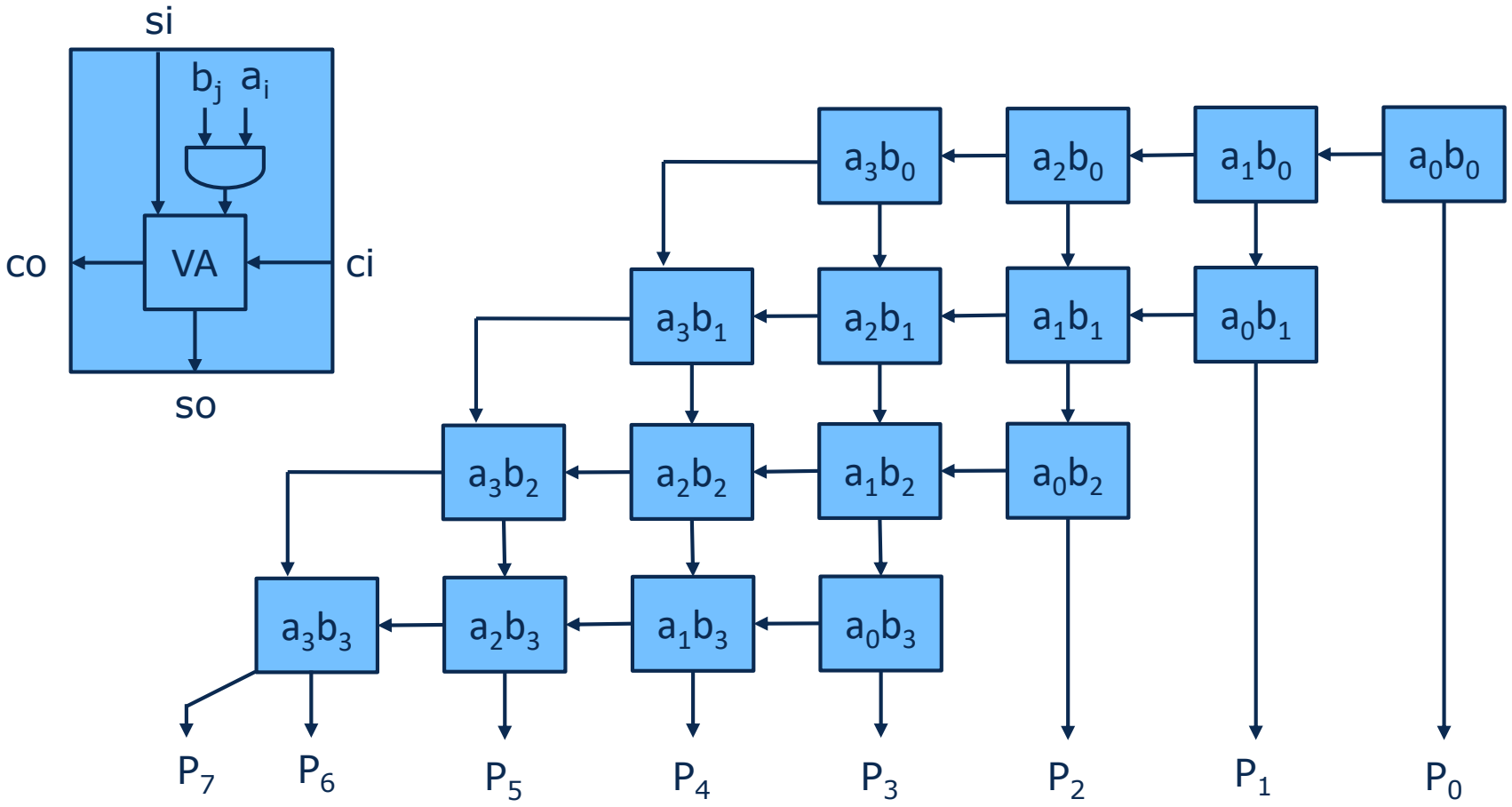
Figure 2-3: **Diagram of SLICEM**

- Quelle: Xilinx 7 Series FPGAs Configurable Logic Block, User GuideUG474 (v1.7) November 17, 2014

- Gewichtete Addition von Partialprodukten $a_i \cdot b_j$

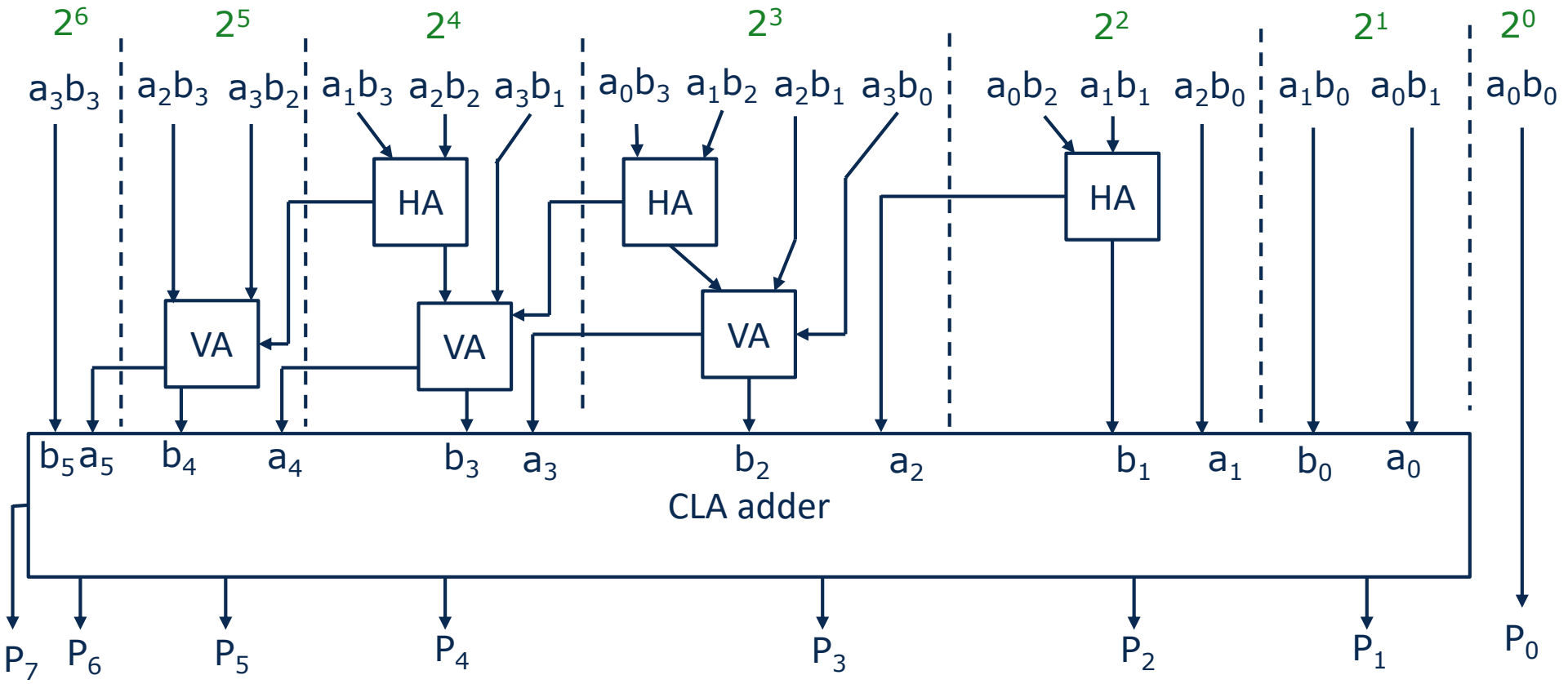
					A3	**A2**	**A1**	**A0**	
					A3B0	A2B0	A1B0	A0B0	**B0**
+				A3B1	A2B1	A1B1	A0B1		**B1**
+			A3B2	A2B2	A1B2	A0B2			**B2**
+		A3B3	A2B3	A1B3	A0B3				**B3**
	P7	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**	

- Addition von Partialprodukten $a_i \cdot b_j$ durch Ripple Carry Ketten



- Addition der **gewichteten** Partialprodukte in einer Baumstruktur Reduktion der Logikstufen speziell bei großen Addierern (Komplexität O(log n))

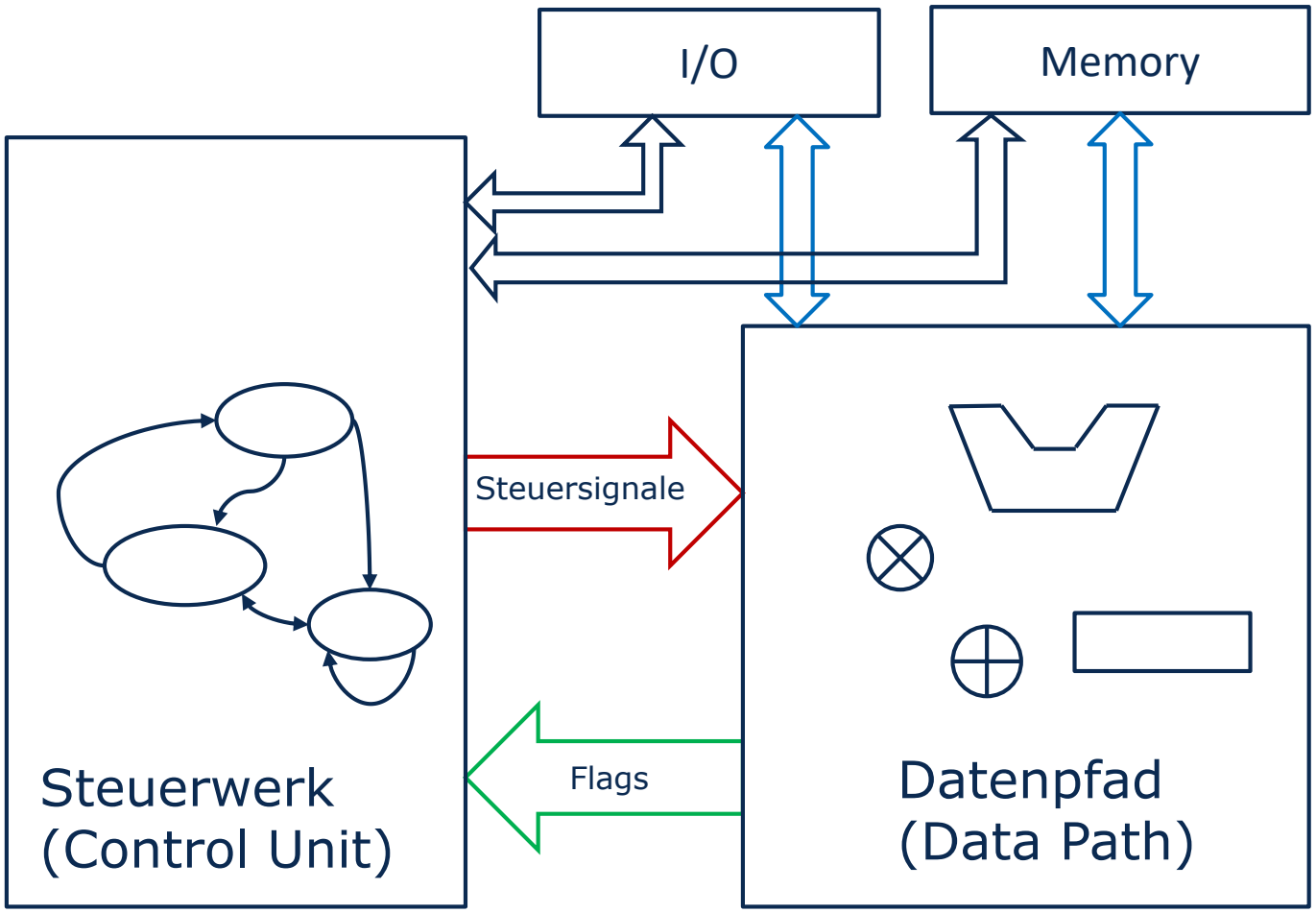
					A3	A2	A1	A0	
					A3B0	A2B0	A1B0	A0B0	B0
+				A3B1	A2B1	A1B1	A0B1		B1
+			A3B2	A2B2	A1B2	A0B2			B2
+		A3B3	A2B3	A1B3	A0B3				B3
	P7	P6	P5	P4	P3	P2	P1	P0	



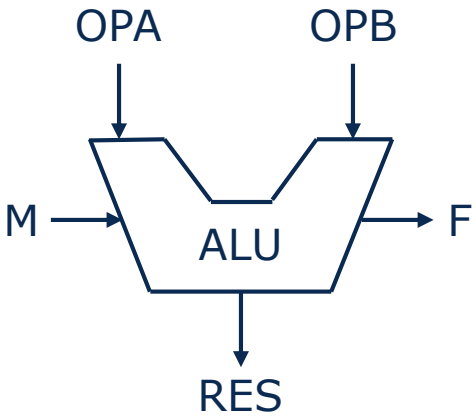
Zusammenfassung Arithmetik Schaltungen

- Vorstellung von Schaltungen zur binären Addition und Multiplikation
- Strategien zur Optimierung der Hardware Implementierung bezüglich Gatterlaufzeiten
- Bei der praktischen Implementierung sind weitere Randbedingungen zu beachten, wie z.B. Chipfläche, Verlustleistung

Datenpfade

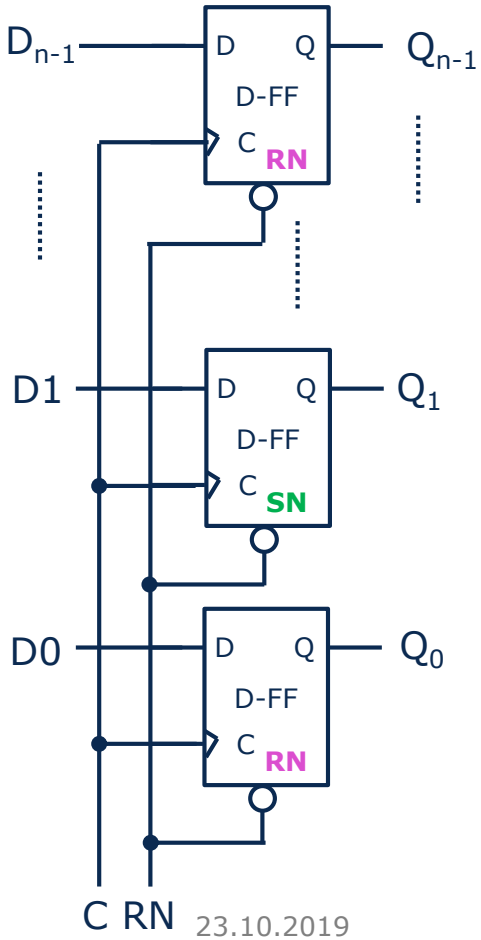


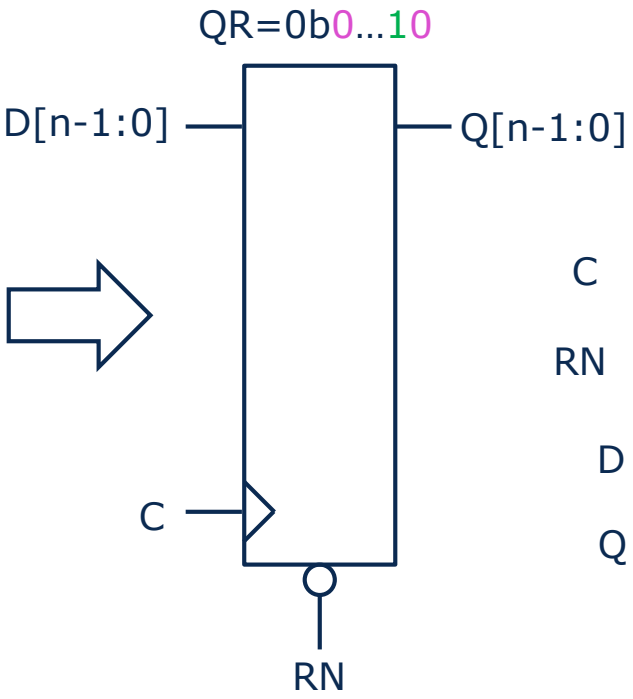
- Arithmetic Logic Unit (ALU) prozessieren numerische und logische Daten
  - Operanden (OPA, OPB, …)
  - Modus (M): Wahl der Operation, z.B.
    - ADD, SUB, MUL, DIV, SHIFT, AND, OR, …)
  - Ergebnis (RES)
  - Status Flags (F): Zusatzinformation zur durchgeführten Operation, z.B.
    - CARRY, OVERFLOW, SIGN, ZERO

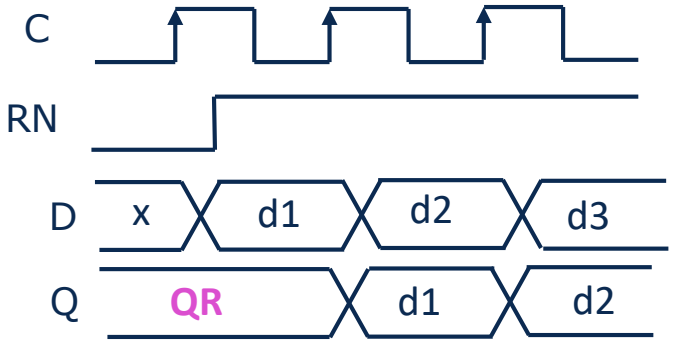


- ALU Datenpfad Baublöcke können kombinatorisch oder sequentiell realisiert sein
- Im Praktikum stehen dedizierte Baublöcke für die Arithmetischen Operationen zu Verfügung
- → Details siehe Anleitung zum Praktikum

- Anordnung aus n-Bit FlipFlops zur Datenspeicherung
- **Reset-Wert** des Registers, festgelegt zur Implementierungszeit durch Auswahl des FlipFlop Typs (**Set**, **Reset**)



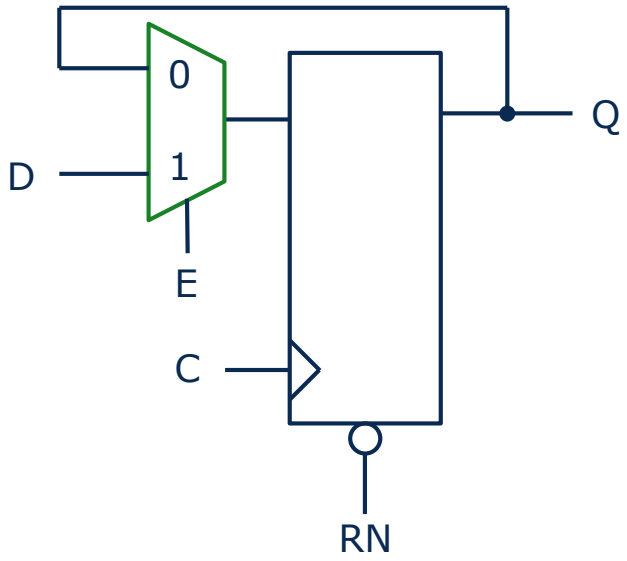


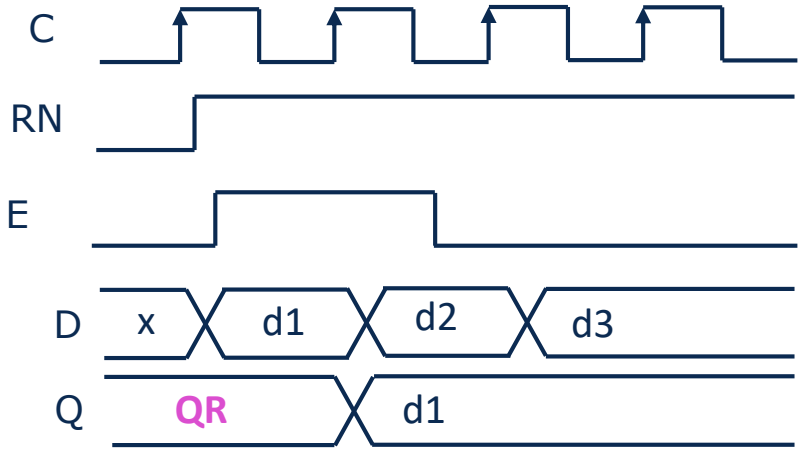


Sehr schneller Zugriff (1 Takt),
sehr kurzes Delay CLK ↑ →Q

Register mit Enable 1

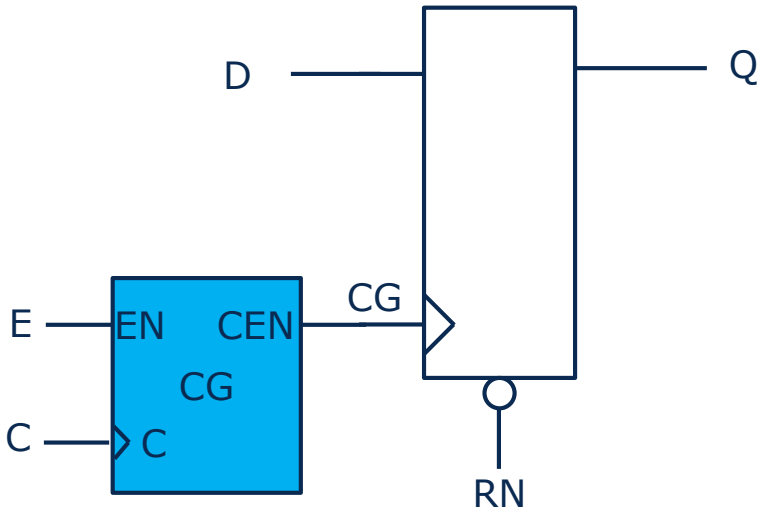
- Bedingtes Schreiben auf das FlipFlop bei E=1
  - $Q_{i+1}=D_i$, wenn E=1
  - $Q_{i+1}=Q_i$, sonst
- Realisierung mit Multiplexer
  - permanentes Schreiben der Daten
  - Auswahl der Daten

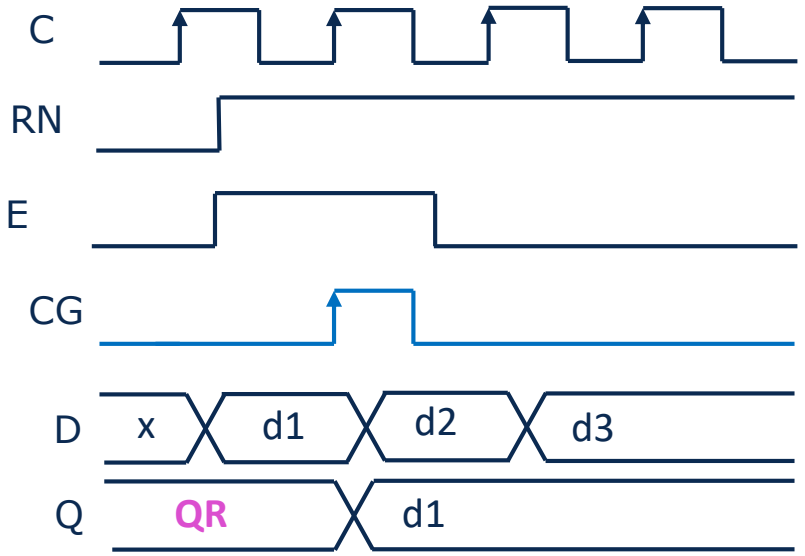




- geringer Schaltungsaufwand
- Kaum Erhöhung der Verlustleistung bei hoher Schreibaktivität

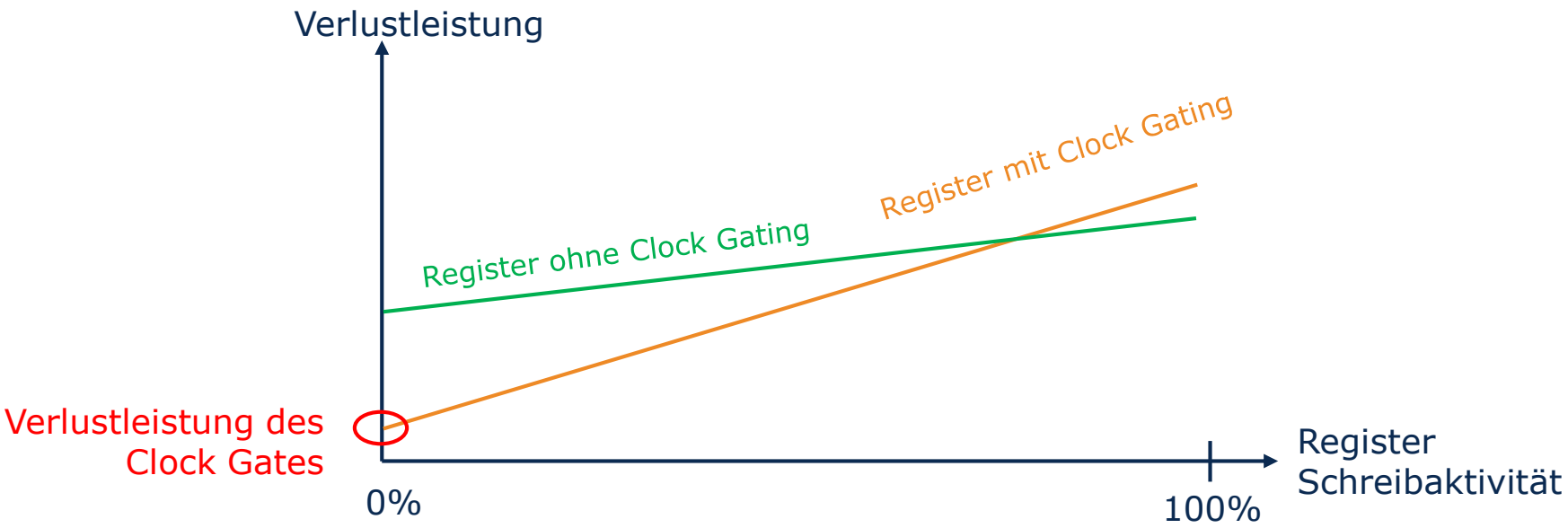
- Realisierung mit Clock Gate
  - Selektives Takten des Registers





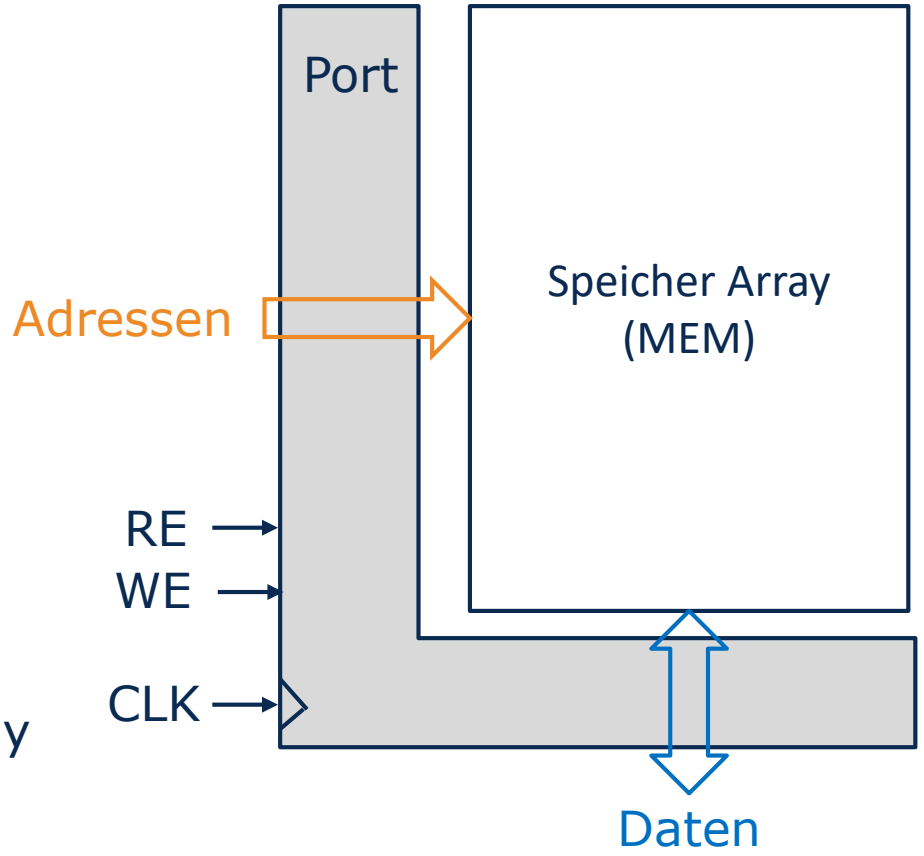
- Höherer Schaltungsaufwand (Fläche)
- Höhere Verlustleistung bei hoher Schreibaktivität
- Geringere Verlustleistung bei nur niedriger Schreibaktivität

Verlustleistungsreduktion durch Clock-Gating



- → Anwendung von Clock Gating abhängig vom Betriebsszenario

- Adressierbare Speicher
- Speicher Array
  - Datenwortbreite: n_D
  - Adressen: n_A
  - Kapazität $n_A \cdot n_D$
- Zugriff über Ports
  - Write: MEM(addr)=D_{IN}
  - Read: Q=MEM(addr)
- Kriterien:
  - Speicherdichte [Bit/µm²]
  - Zugriffzeit (Anzahl Takte, Delay CLK→Q)
  - Verlustleistung
  - Nicht-flüchtiger Speicher (ja/nein)

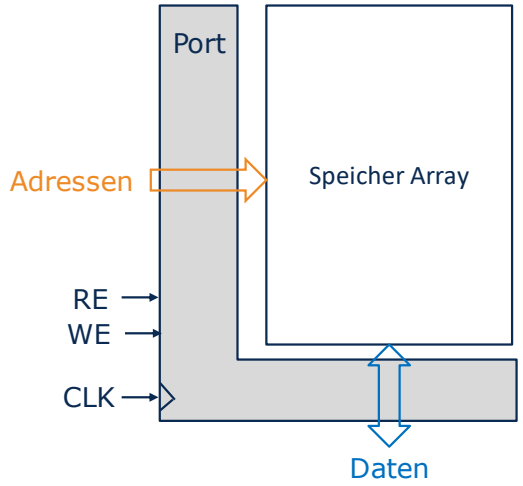


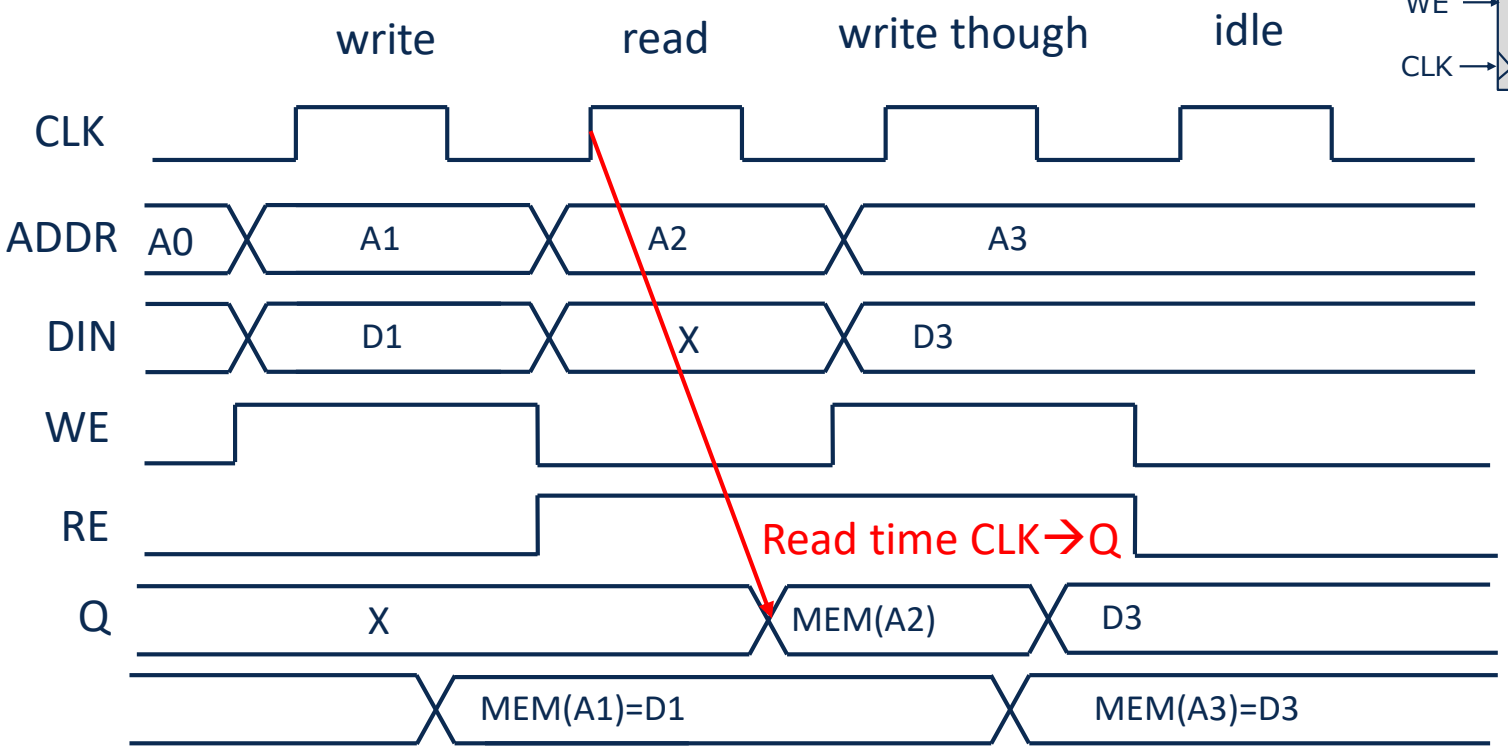
Typen von on-Chip Speicher

Typ	NVM	Bits/µm²	Zugriffzeit	Typische Speichergrößen	Kommentar
Latch	Nein	<1	Einige 10ps	einige Bit	Ansteuerschaltung nötig da Level sensitiv!
FlipFlop	Nein	<0.4	einige 10ps	einige Bit	
SRAM	Nein	<7 (Zelle) <5 (Makro)	einige 100ps	einige kBit bis MBit	Komplexe Ansteuerschaltung (SRAM Makro)
(embedded) DRAM	Nein	<30	einige 100ps	einige kByte bis Mbyte	Dynamischer Speicher, refresh nötig. Ggf. extra Prozessoptionen (Kosten!)
(embedded) Flash	Ja	<30	einige ns	einige kByte bis Mbyte	Extra Prozessoptionen (Kosten!)
ROM	Ja	<15	einige 100ps	einige kBit bis MBit	„Maskenprogrammierung“ bei der Implementierung
OTP	Ja	<10	einige 100ps	einige kBit bis MBit	„Einmalprogrammierung“ beim Test der Chips

Zahlenwerte geschätzt für 28nm Technologieknoten

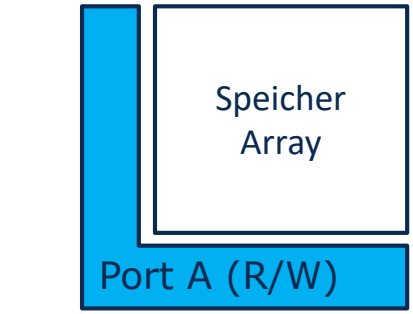
- Beispiel:
  - Synchroner SRAM Port
  - 1 Takt Zugriffszeit
  - Write-Through Funktion

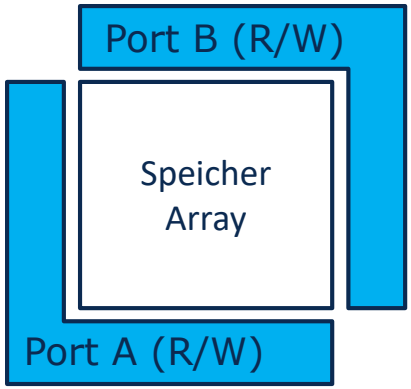


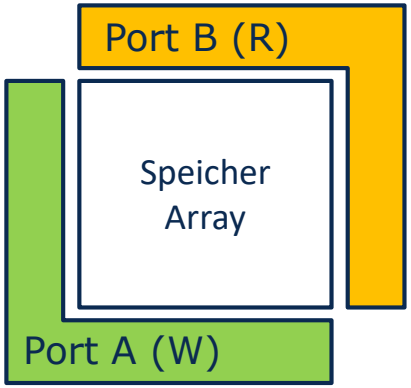


Klassifikation von SRAM Makros

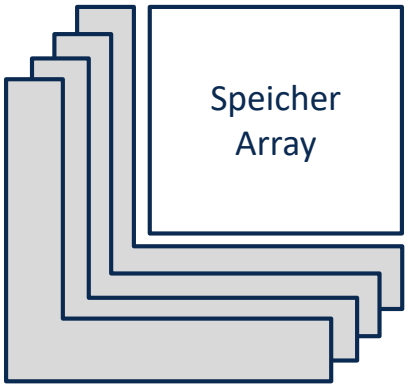
- Single Port (SP)
  - Zugriff von **einem Write/Read** Port
- Dual Port (DP)
  - Zugriff von **2 unabhängigen Write/Read** Ports
  - Synchrone und asynchrone Variante möglich
  - Arbitrierung/Priorisierung von Write Zugriffen auf gleiche Adressen nötig
- Two-Port (TP)
  - Zugriff von **einem Write Port** und **einem separaten Read Port**

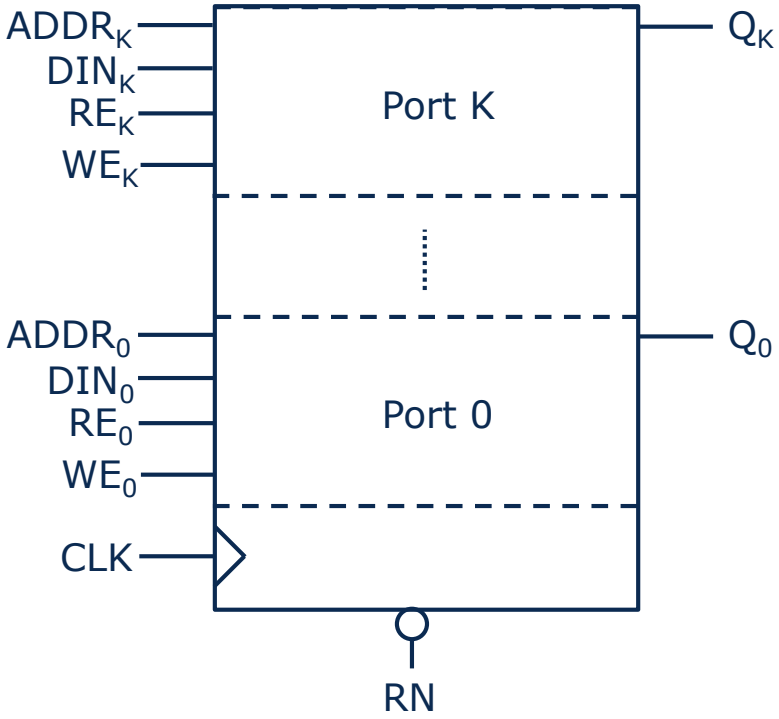






- Arrays aus Registern mit mehren Write- und Read-Ports
- Adressierbarer Zugriff
- Typischer Weise synchrone Realisierung (gleicher Takt für alle Ports)
- Schneller Zugriff (1 Takt), kurzes Delay CLK ↑ →Q



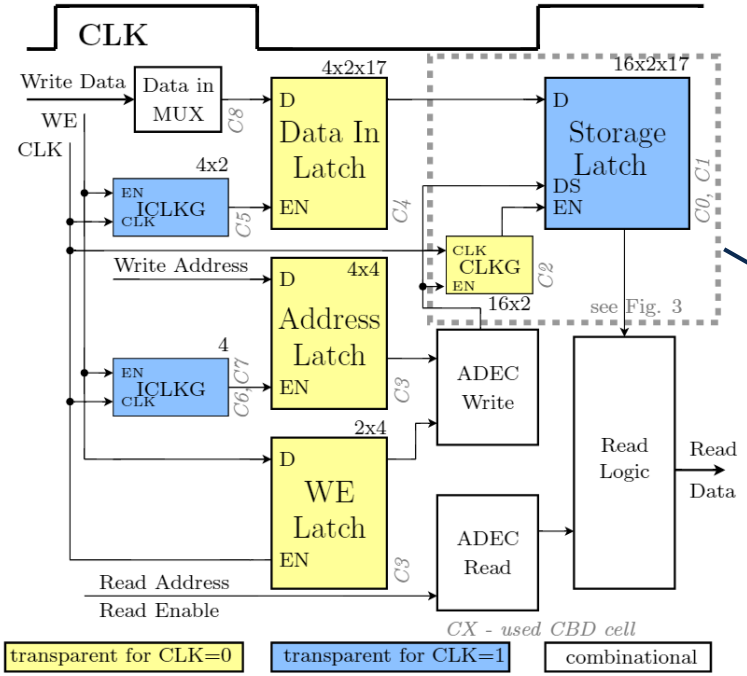


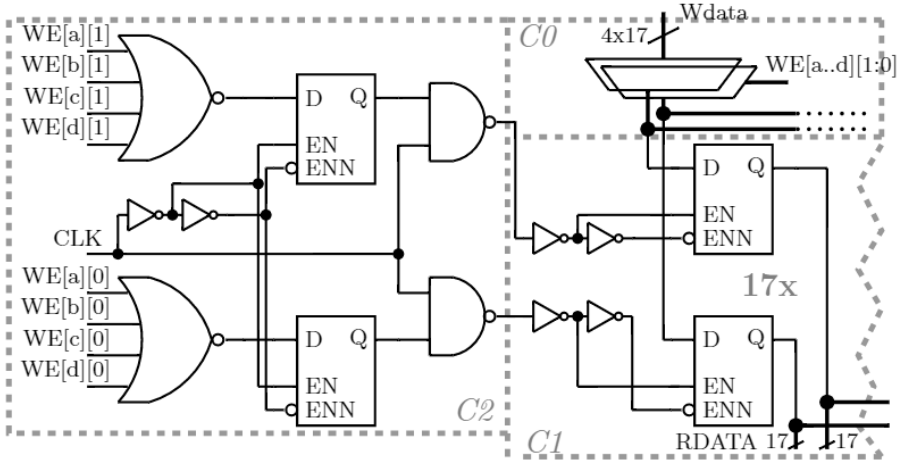
- Anwendung in Systemen mit mehreren Rechenkernen

Beispiel: Register File für SDR Prozessor

- Register File für Infineon X-GOLD SDR™ 20 Prozessor
- Entwickelt an der Stiftungsprofessur HPSN (2008/2009)
- 16-Worte zu je 34 Bit, 4-Write Ports, 6 Read Ports
- Cell-Based Design → Optimierung auf Transistorlevel

Quelle: infineon.com

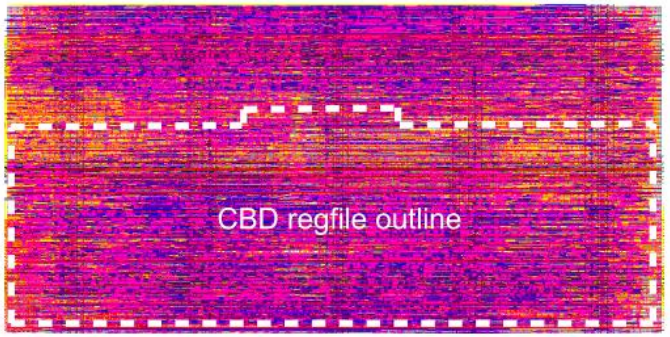


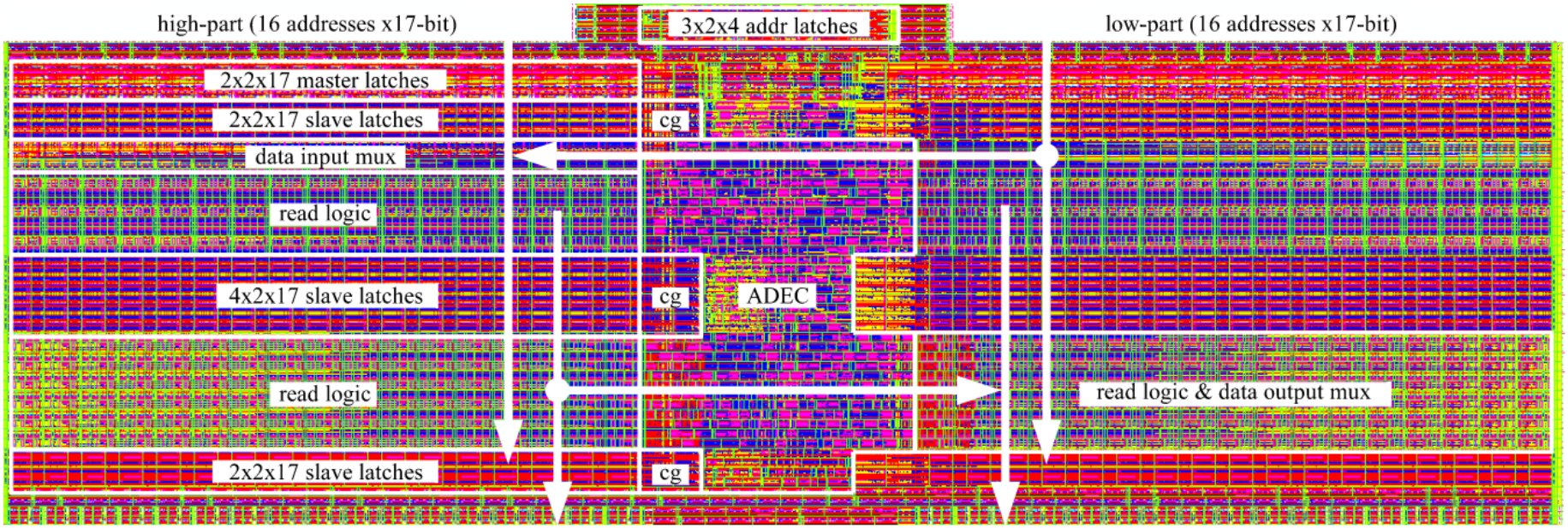


Johannes Uhlig, Sebastian Höppner, Georg Ellguth, and René Schüffny, A Low-Power Cell-Based-Design Multi-Port Register File in 65nm CMOS Technology, IEEE International Symposium on Circuits and Systems ISCAS 2010, 2010, p. 313-316,

Beispiel: Register File für SDR Prozessor

- Implementierung in 65nm CMOS Technologie
- ≈0,026mm² Chipfläche (0,02Bit/µm²)
- 300MHz Taktfrequenz
- Effizienzsteigerung verglichen mit einer Synthese und Place&Route Implementierung:
  - 44% Flächenersparnis und
  - 30% Verlustleistungsreduktion

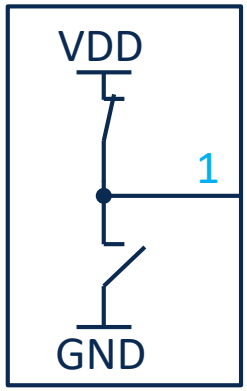


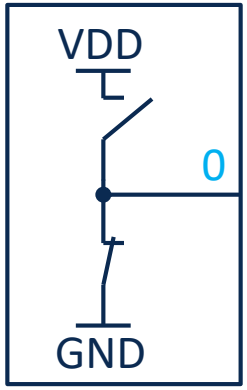


Johannes Uhlig, Sebastian Höppner, Georg Ellguth, and René Schüffny, A Low-Power Cell-Based-Design Multi-Port Register File in 65nm CMOS Technology, IEEE International Symposium on Circuits and Systems ISCAS 2010, 2010, p. 313-316,

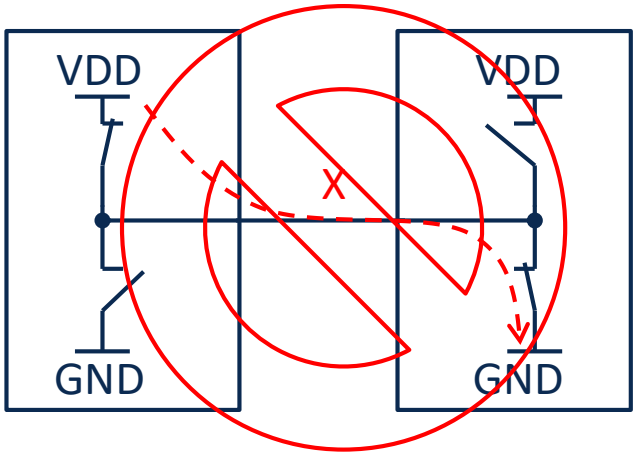
- Die Verschaltung von Datenpfadbaublöcken soll flexibel und re-konfigurierbar erfolgen
- Bussysteme dienen der Vernetzung von Datenpfadelementen
- Vorstellung von 3 Ansätzen:
  - Tri-state Bus
  - Multiplexer Bus
  - Komplexe Bussysteme und Network-on-Chip

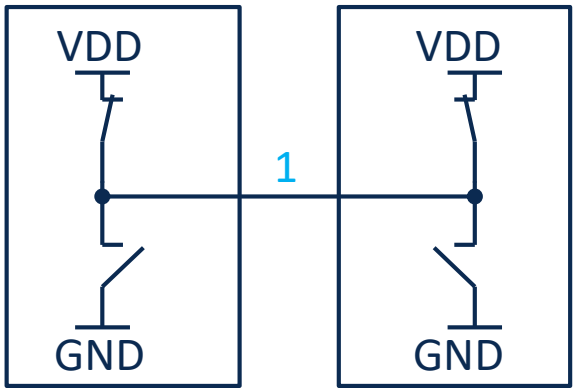
- Gatter**ausgänge** erzeugen 2 Logikpegel (0,1)

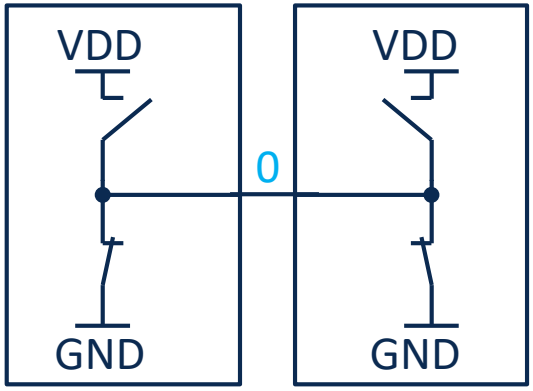




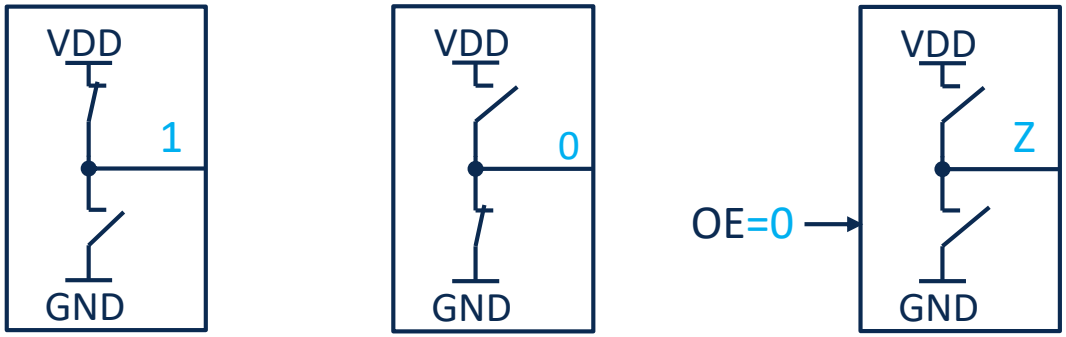
- Das Zusammenschalten mehrerer Gatter**ausgänge** ist nur möglich wenn sie die **gleichen** Pegel treiben



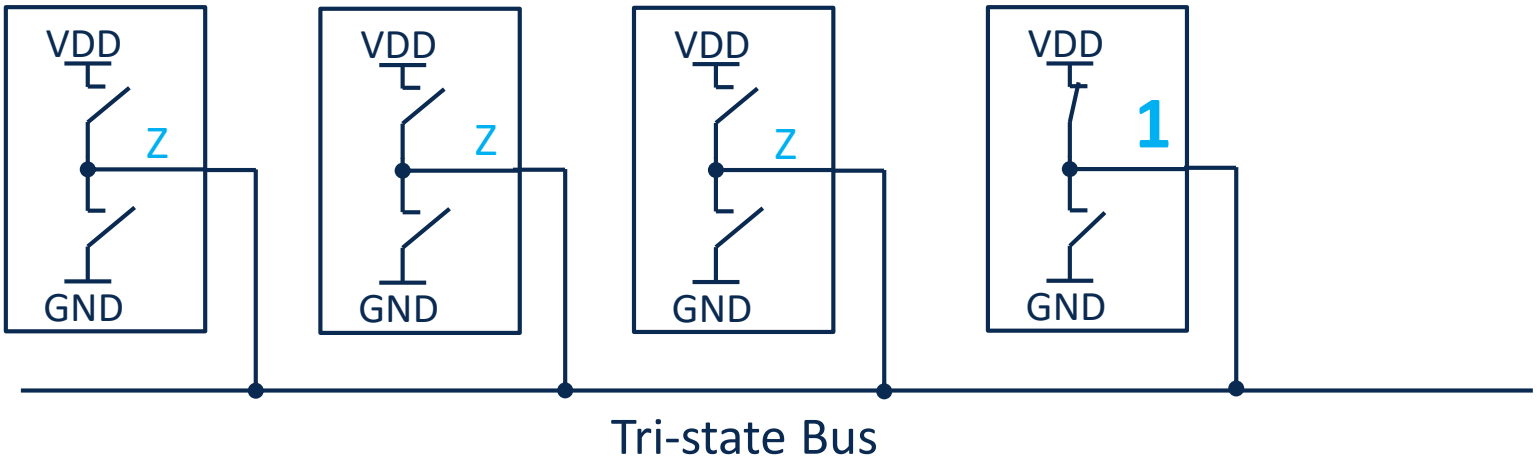




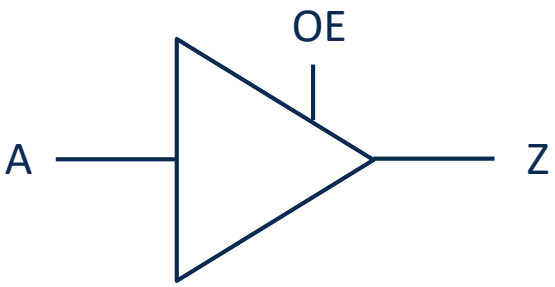
- Einführung eines **Dritten** Ausgangszustandes (Tri-State) **Z**
- Ausgangstreiber wird hochohmig geschalten, durch separates Steuersignal (OE)



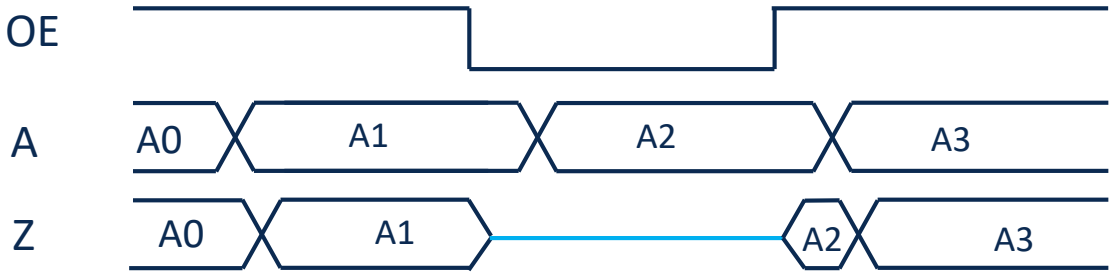
- Das Zusammenschalten mehrerer Gatter**ausgänge** im Tri-state ist möglich.
- Es darf nur ein Treiber aktiv sein!



- Buffer mit Tri-state Ausgangstreiber

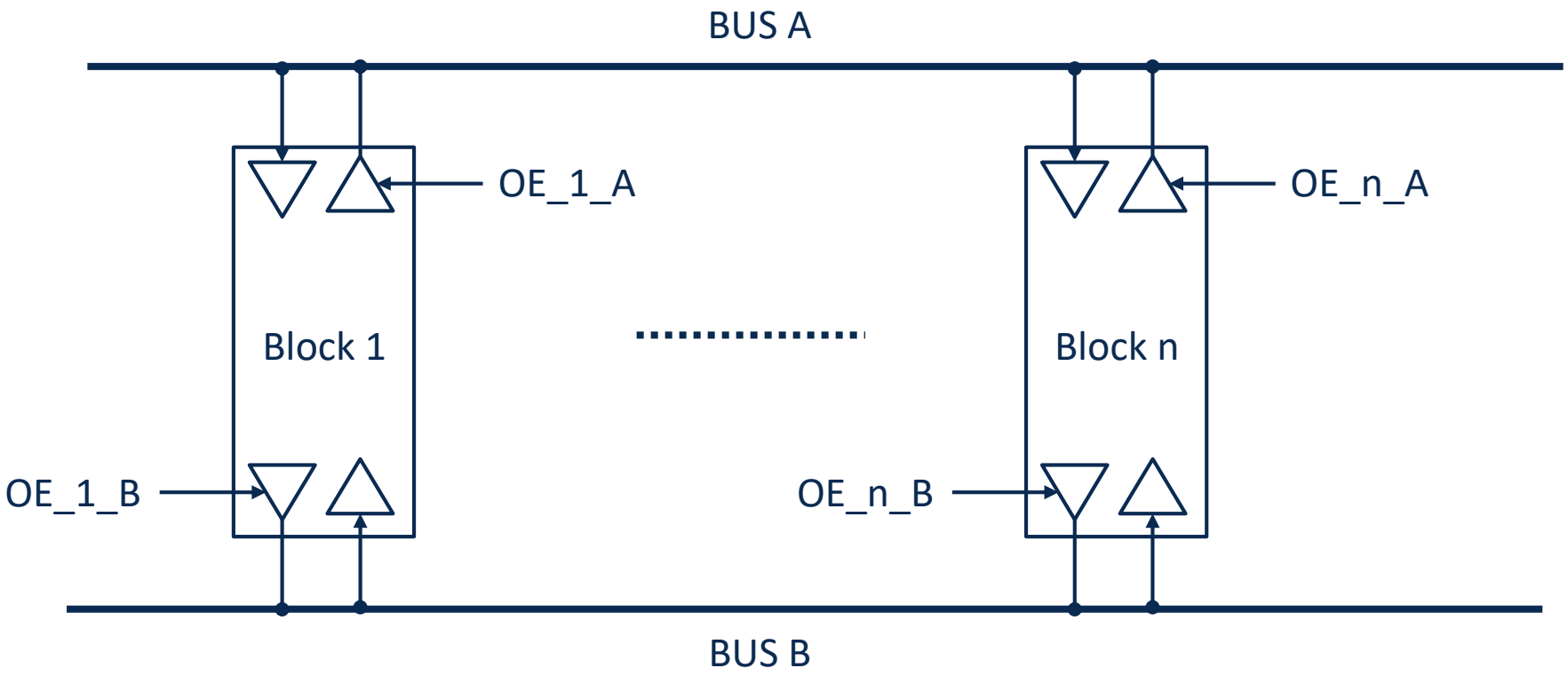


OE	A	Z
0	X	Z
1	0	0
1	1	1



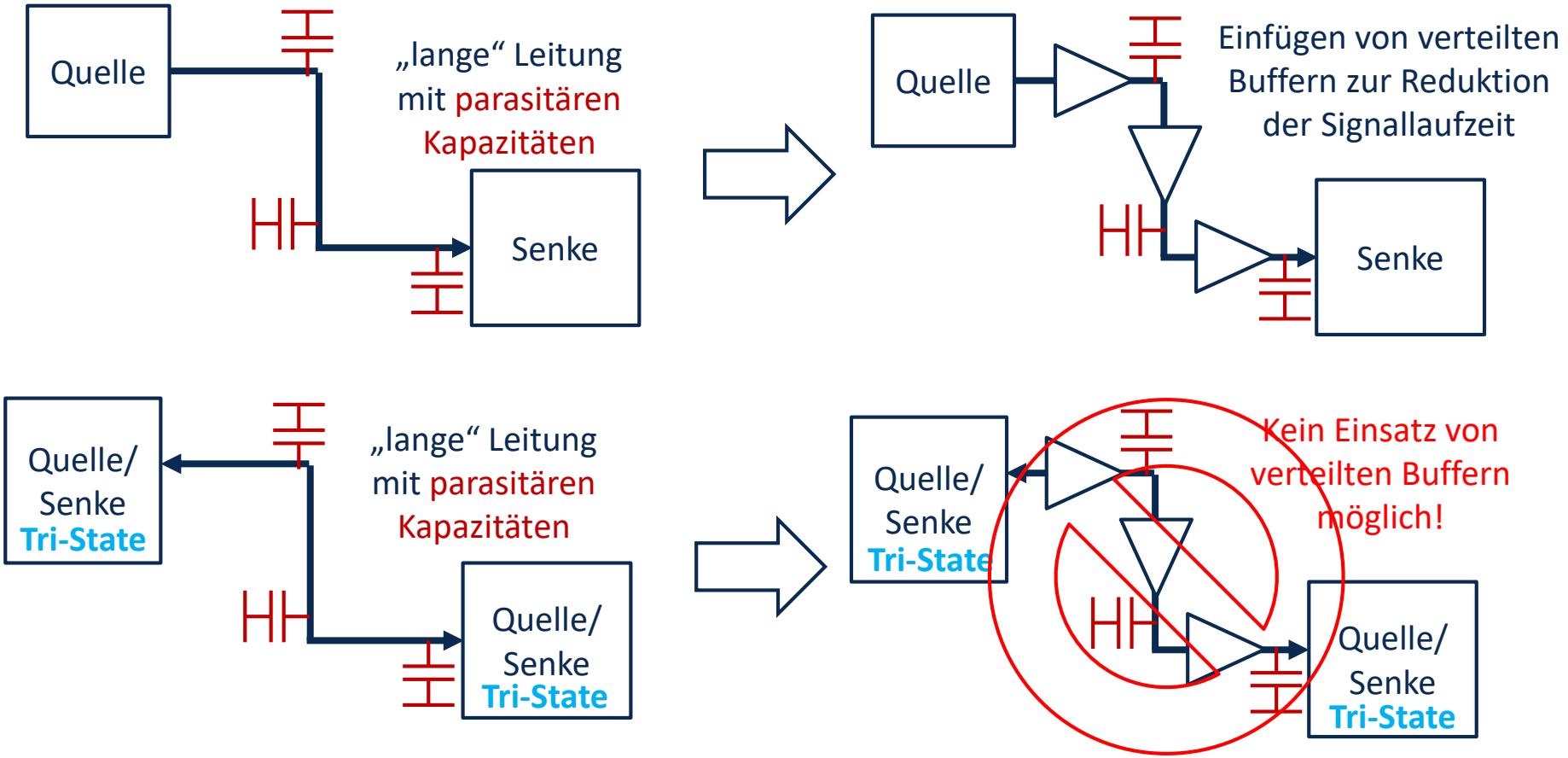
- Kombinatorische Gatter und sequentielle Baublöcke (z.B. Register) können mit Tri-state Ausgängen versehen werden.

Datenpfade mit Tri-State-Bussen



- Mehrere Baublöcke können an **einen** Tri-State Bus angeschlossen werden
- Pro Tri-State Bus darf nur **ein** Treiber aktiv sein → Steuerwerk!
- Mehrere Busse möglich für **parallelen** Datentransfer zwischen Baublöcken

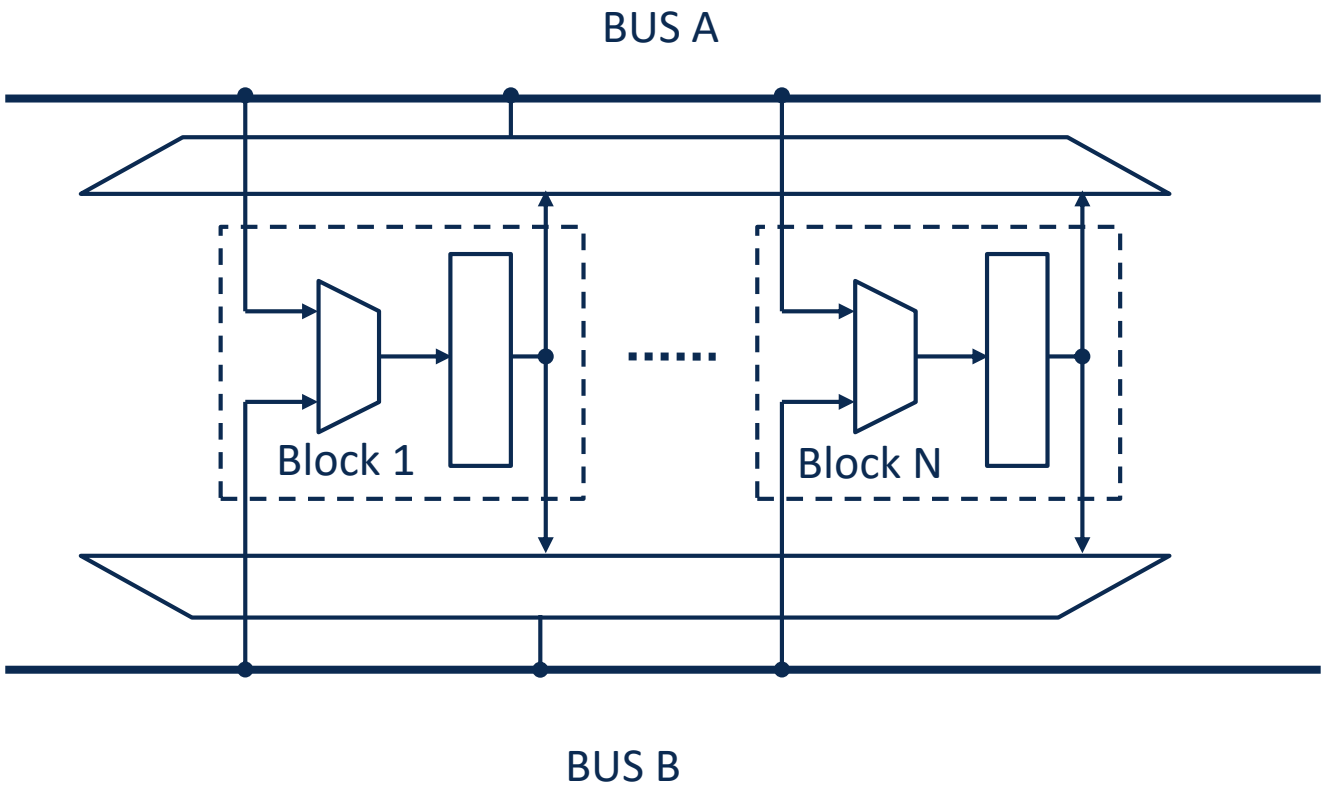
Nachteil von Tri-State-Bussen



- In aktuellen Entwürfen mit automatischem Platzieren und Verdrahten (P&R), werden Tri-State Signale vermieden!

Multiplexer Busse

- Auswahl der Datenquellen für den Bus durch Multiplexer
- Auswahl des Busses für den Eingang des Datenpfad Elements durch Multiplexer
- Setzen der Multiplexer Select Signale durch das Steuerwerk



Multiplexer Bus

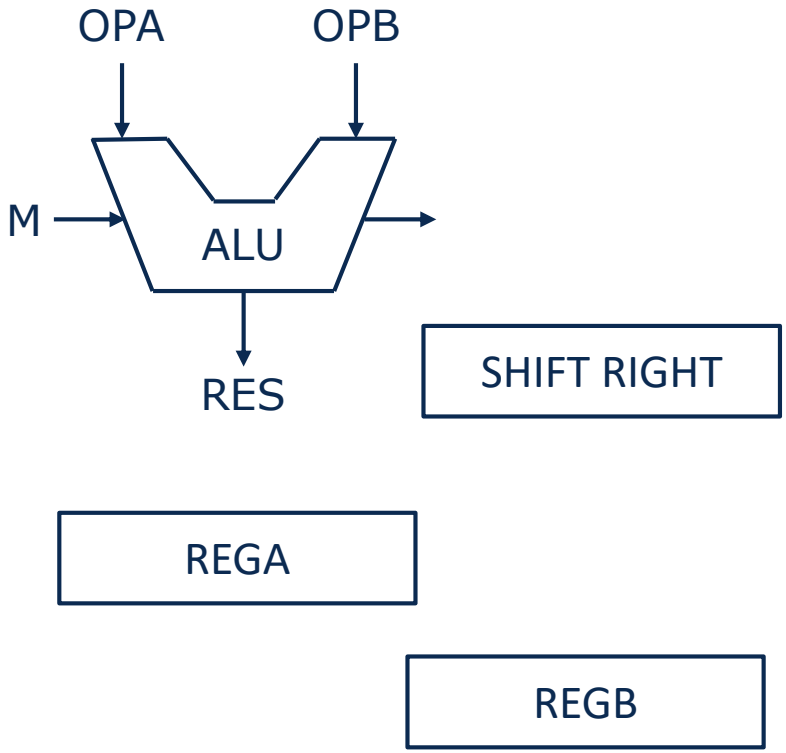
- Vorteil: Keine Tri-State Treiber

- Anzahl der Multiplexer Eingänge abhängig von notwendigen Datenverbindungen
- Ziel: Minimierung der notwendigen Multiplexer Eingänge
- Möglichkeit der hierarchischen Realisierung von Multiplexern
  - Bus Multiplexer: Auswahl der Datenquelle für den Bus
  - Eingangs Multiplexer: Auswahl der Datenquelle für den Eingang des Datenpfadelementes

Beispiel: Bus-Implementierung

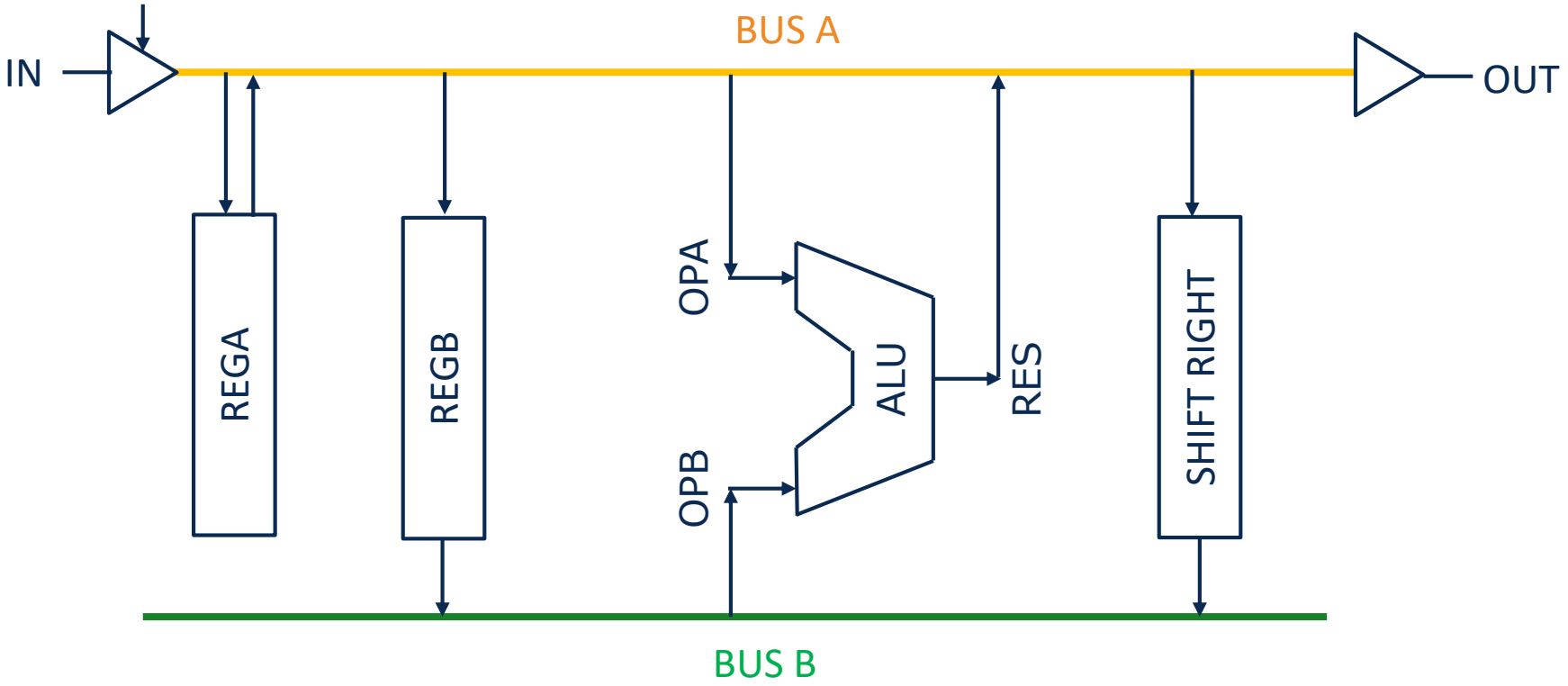
- Beispiel: C=(A+B)+A/2

Zustand	Datenquelle	Anzahl Busse
LOAD A	IN→ REGA	1
LOAD B	IN→ REGB	1
COMP1	REGA → ALU(OPA) REGB → ALU(OPB) REGA → SHIFT	2
COMP2	ALU(RES)→ ALU(OPA) SHIFT→ ALU(OPB)	2
STORE	ALU(RES) → OUT	1

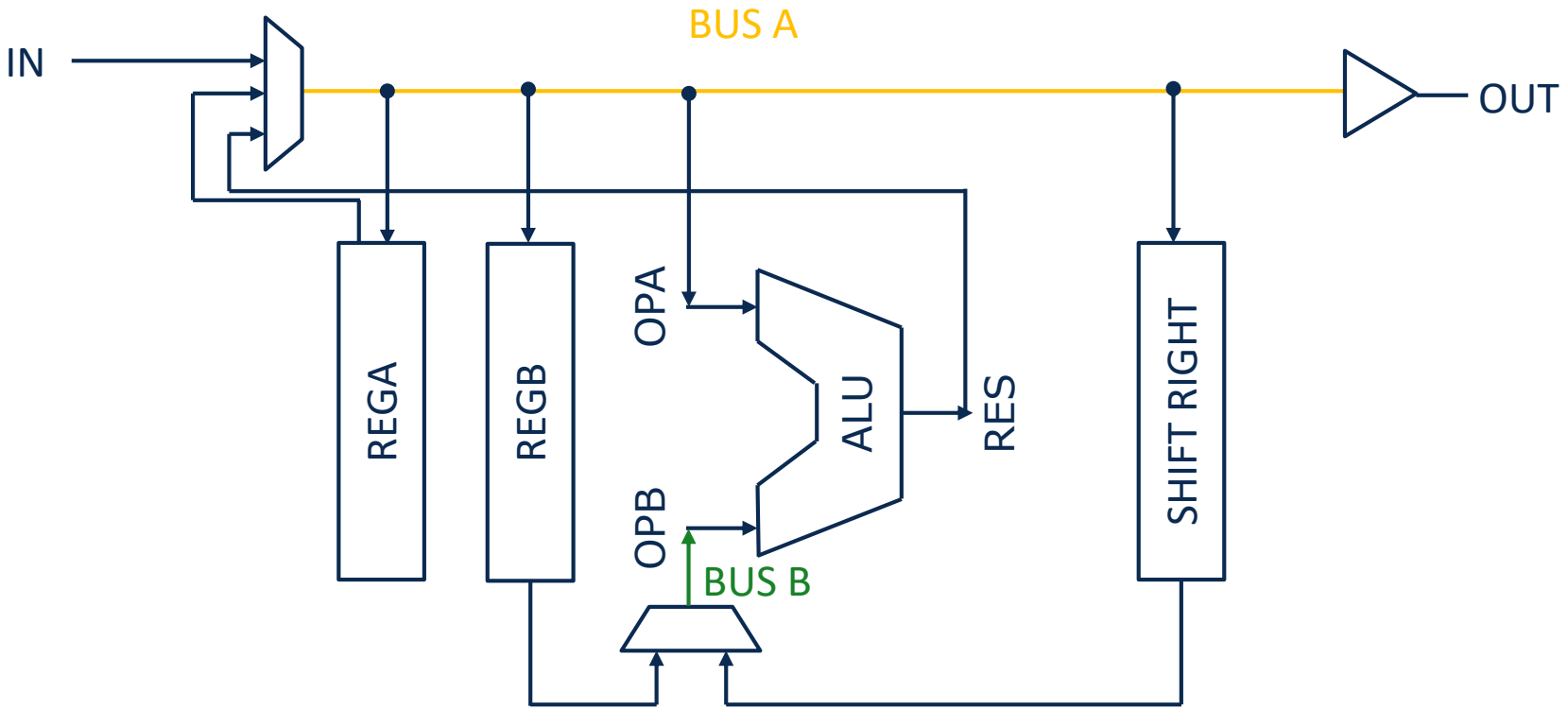


- → 2 Datenbusse nötig (BUS A, BUS B)

Beispiel: Bus-Implementierung Tri-State Bus

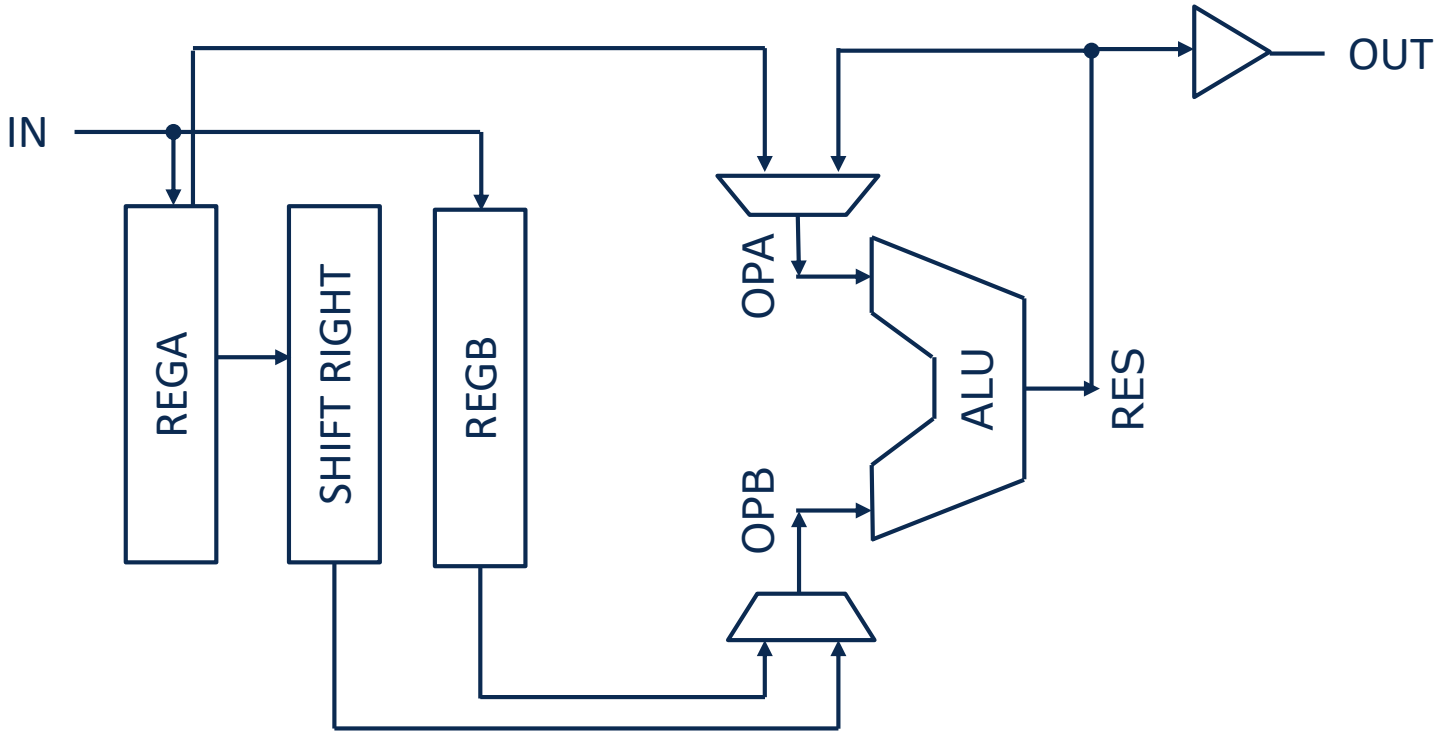


Beispiel: Bus-Implementierung Multiplexer Bus Variante 1

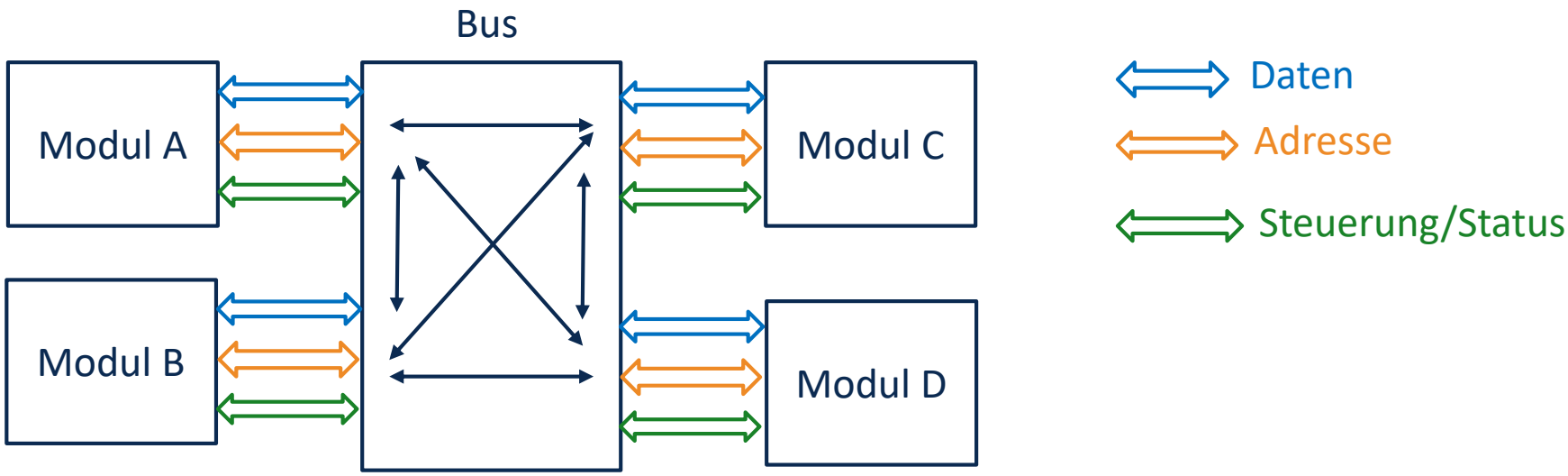


Beispiel: Bus-Implementierung Multiplexer Bus Variante 2

- Reduktion der Multiplexer

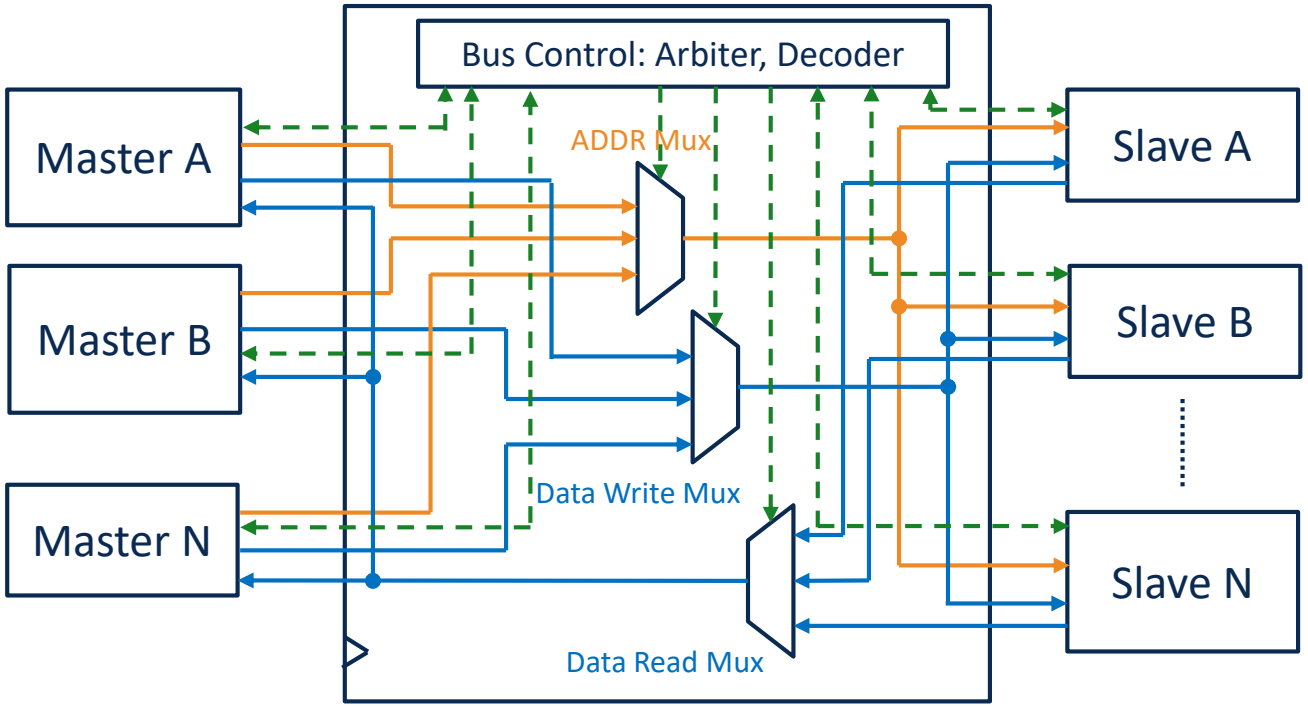


- Die bisher vorgestellten Bus-Systeme erfordern **Anwendungsspezifische** Kontrolle durch ein Steuerwerk
- Sie sind nicht standardisiert → Einbinden vorgefertigter Baublöcke (IP) schwierig.
- Komplexe digitale Systeme erfordern flexible, standardisierte Bussysteme
- Grundkonzept:
  - Standardisierte Busschnittstelle (Daten, Adresse, Steuer- und Statussignale)
  - Adressierung der Busteilnehmer (ID)
  - Zugriff über Ports (ähnlich Memory)
  - Businterne Steuerlogik realisiert den Datentransfer und Überwacht den Zugriff



Komplexe Bussysteme – Crossbar Busse

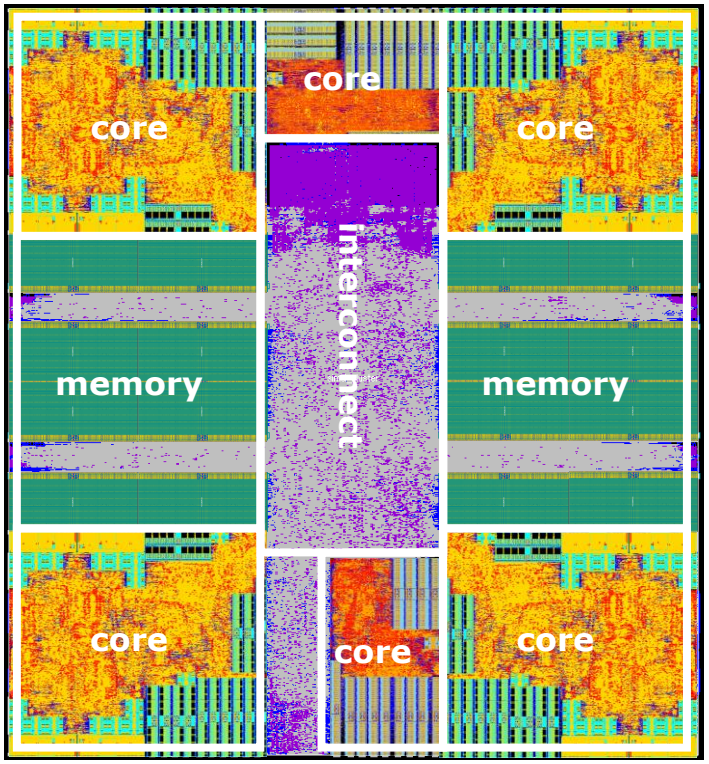
- „Circuit Switched Network" → Multiplexer Bus
- Master und Slave Komponenten mit standardisiertem Bus-Interface
- Arbitrierung und Priorisierung des Zugriffs durch Bus Controller
- Synchrone Realisierung (CLK)
- Sequentieller Zugriff (Adress-Cycle, Data Cycle, Wartezyklen wenn Bus „busy", Burst)
- Beispiele: ARM AHB, WishBone, …



Beispiel – Crossbar Busse

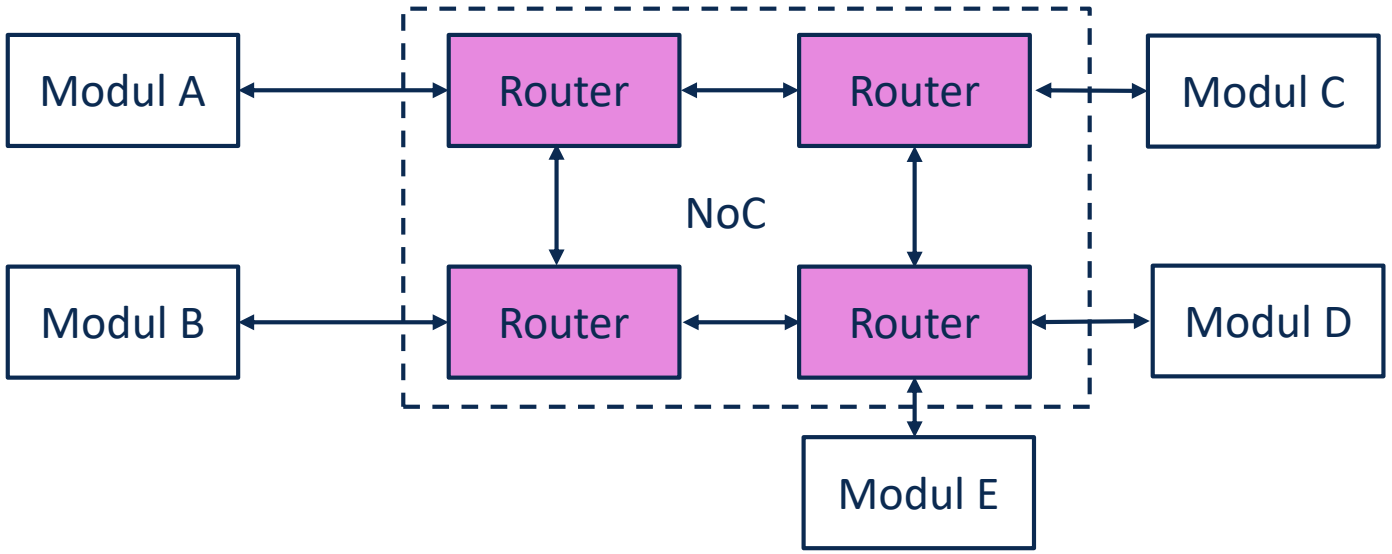
- Vorteile:
  - Standardisierte Busschnittstelle
  - Geringer Latenz (wenige Takte)
  - Einfach zu implementieren (synchrones Design)
- Nachteile:
  - Lokal synchrone Taktung für „globale Verdrahtung“ auf dem Chip
  - Hoher Aufwand an Chipfläche und Verlustleistung
  - Kann die maximale Taktfrequenz limitieren.

- Beispiel: Music 2 SIMD Cluster



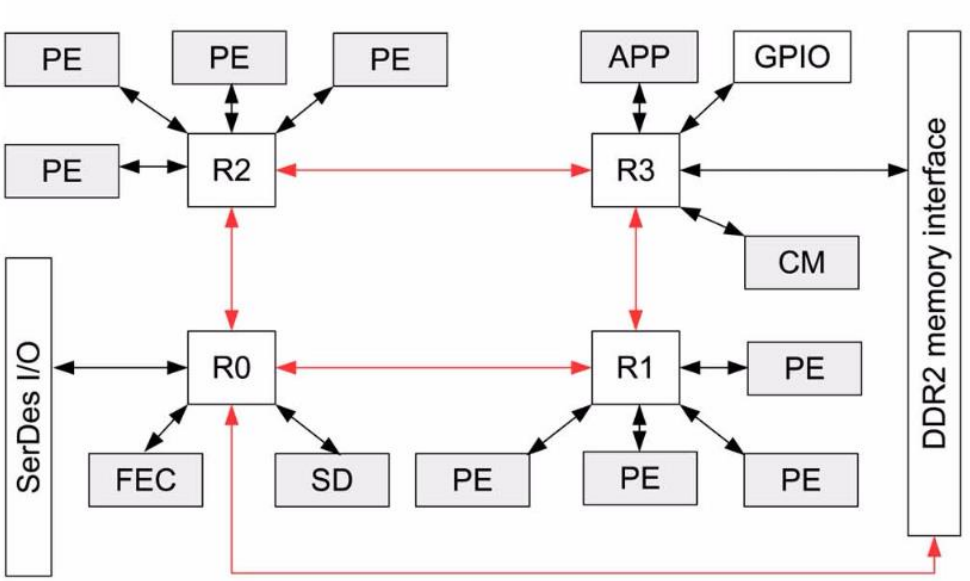
Infineon/IMC MuSiC2 SIMD Cluster (conv. interconnect)

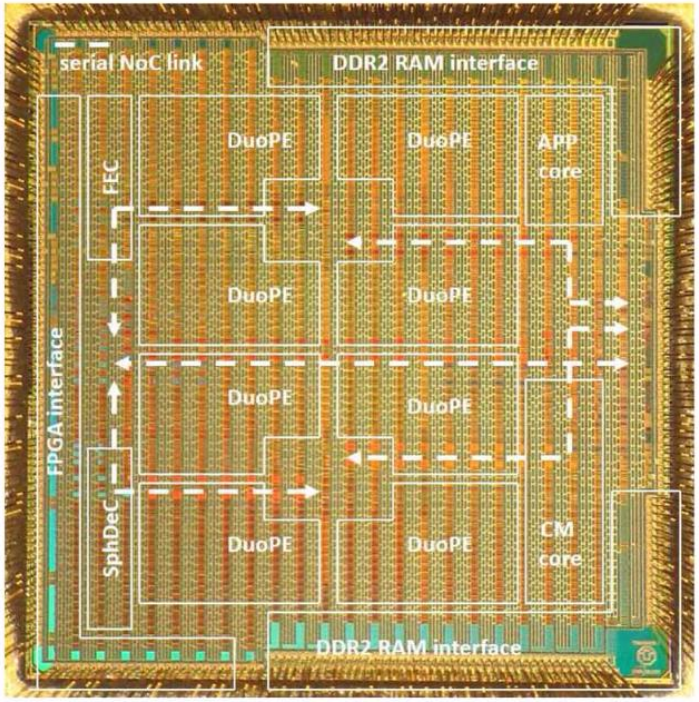
- „Packet Switched Network“ → Routing von Paketen durch das Netzwerk (Router)
- Flexible Topologien möglich
- Standardisiertes Paketformat und Interface der Komponenten
- Hohe Performanz (Datendurchsatz, Latenz) in komplexen Systemen durch Parallelität
- Global asynchrone Realisierung möglich, individuelle Takte der einzelnen Komponenten



Beispiel: Tomahawk 2 NoC

- Tomahawk2: Software-Defined Radio Basisbandprozessor, entwickelt von TUD-MNS und TUD-HPSN
- Vernetzung der Systemkomponenten in einem NoC
- Punkt-zu-Punkt Verbindungen mit schnellen seriellen Links → kompaktes Layout

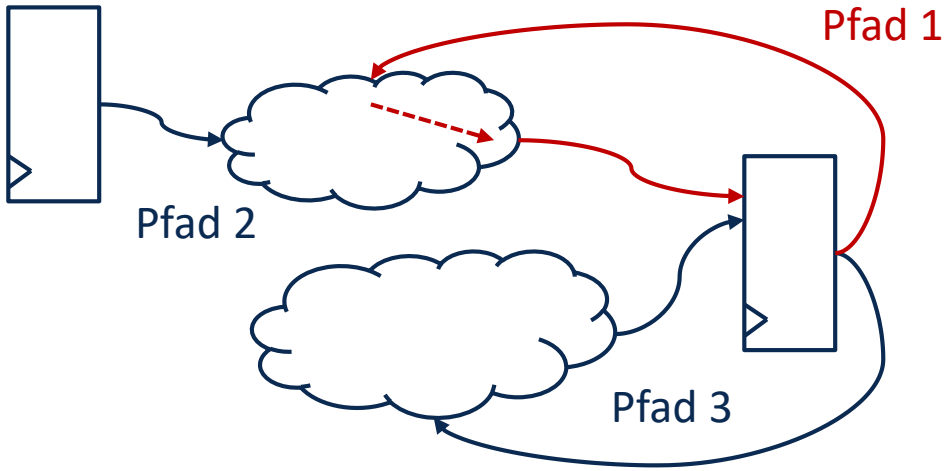




Höppner, Sebastian and Walter, Dennis and Hocker, Thomas and Henker, Stephan and Hänzsche, Stefan and Sausner, Daniel and Ellguth, Georg and Schlüssler, Jens-Uwe and Eisenreich, Holger and Schüffny, René, An Energy Efficient Multi-Gbit/s NoC Transceiver Architecture With Combined AC/DC Drivers and Stoppable Clocking in 65 nm and 28 nm CMOS, IEEE Journal of Solid State Circuits 50 (2015), no. 3, 749-762,

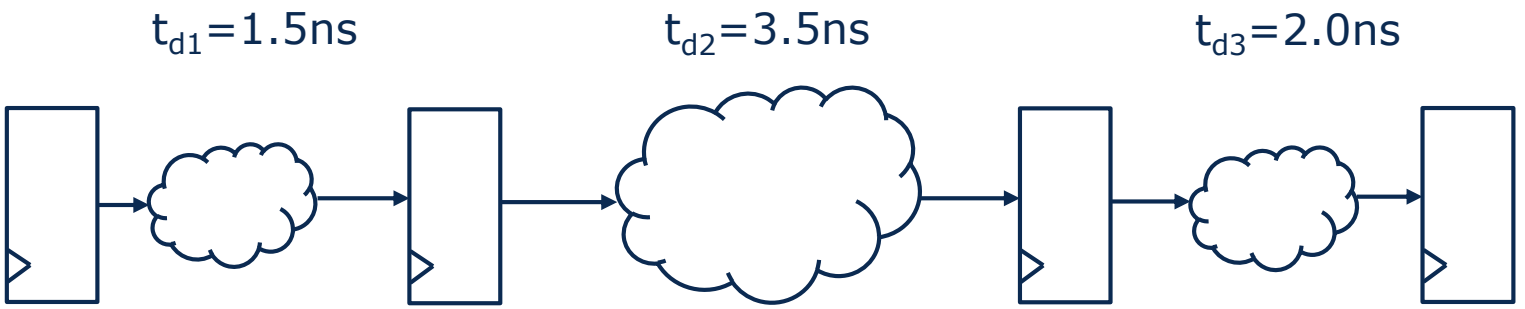
Anforderungen an Datenpfade

- Taktfrequenz und Datendurchsatz, gemessen in
  - [GHz]
  - Operationen pro Sekunde [GOPS]
- Chipfläche, gemessen in
  - [µm²]
  - NAND2 Äquivalenten (technologieunabhängige Normierung)
- Energie pro Rechenschritt, gemessen in
  - [pJ]



- Datenpfade besitzen viele Timing-Pfade mit Verzögerung t_{delay}
  - Startpunkt: Register Ausgang
  - Endpunkt: Register Eingang
- Taktfrequenz limitiert durch den **kritischen Pfad** ($f_{max} \approx 1/max(t_{delay})$)
- Oft sind nur wenige Datenpfadelemente kritisch für das Timing

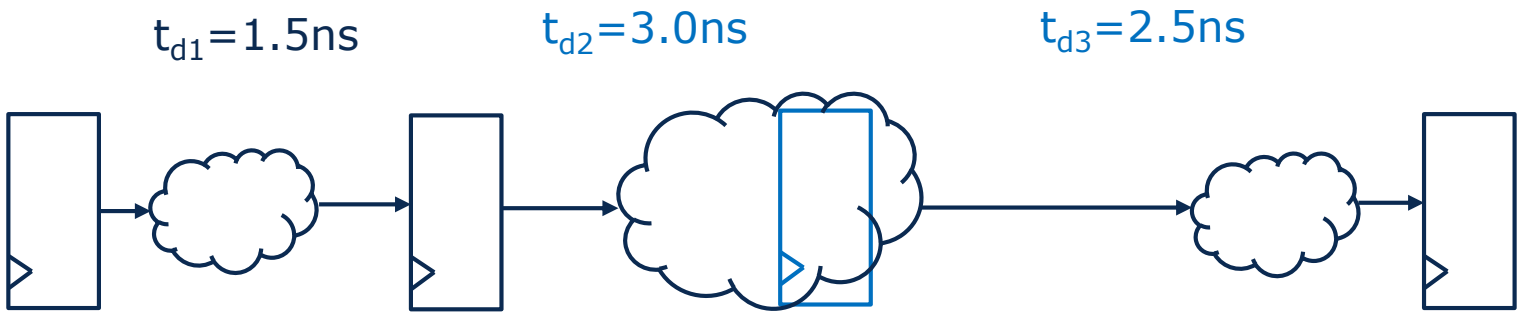
Taktfrequenz und Durchsatz



- $f_{max}=1/max(t_{delay})$ → 285MHz
- Latenz: 4 Taktzyklen
- Datendurchsatz: 285MHz/4=**71 MOP/s** (ohne Pipelining)

Optimierung von Taktfrequenz und Durchsatz

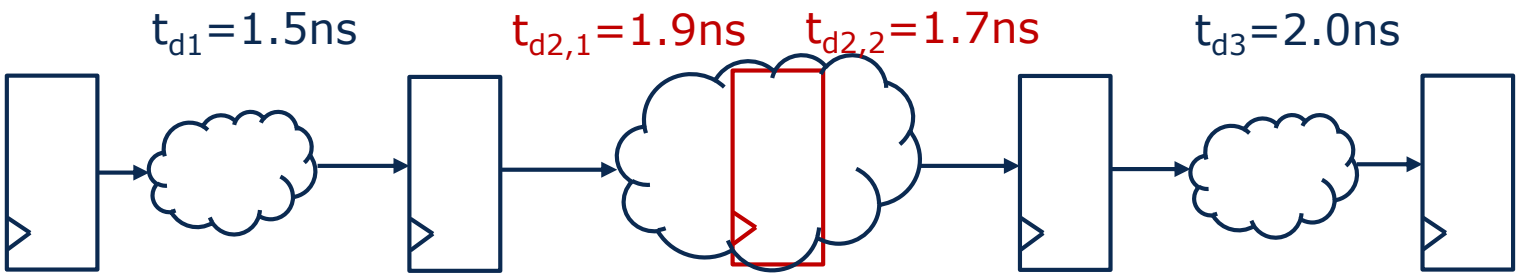
- Logic Re-Timing:



  - $f_{max}=1/max(t_{delay})$ → 333MHz
  - Latenz: 4 Taktzyklen
  - Datendurchsatz: 333MHz/4=**83 MOP/s** (ohne Pipelining)
- Kann automatisch von Synthesetools durchgeführt werden

Optimierung von Taktfrequenz und Durchsatz

- Einfügen von Registerstufen:

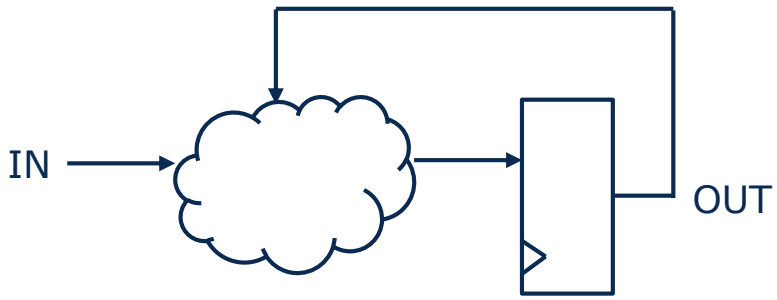


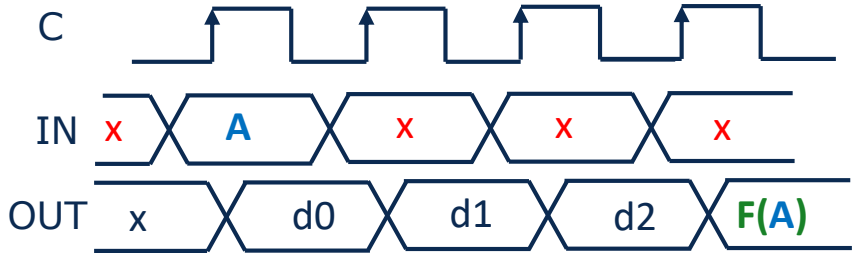
- $f_{max}=1/max(t_{delay})$ → 500MHz
- Latenz: 5 Taktzyklen
- Datendurchsatz: 500MHz/5=**100 MOP/s** (ohne Pipelining)

- Arithmetische Schaltungs-IP (RTL) oft konfigurierbar hinsichtlich der Anzahl benötigter Takte (z.B. Synopsys Design Ware)

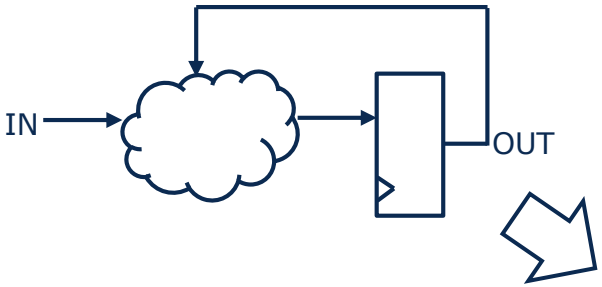
Sequentielle Datenpfad Baublöcke

- Realisierung von sequentiellen Abläufen
- Register Werte in Zeitschritt i sind Grundlage für Berechnung in Schritt i+1
- Vorteile:
  - Kürzere Logiklaufzeiten (Berechnung in mehreren Taktzyklen)
  - Geringer Hardware Aufwand (Wiederverwendung von Baublöcken in unterschiedlichen Taktzyklen)
- Nachteile:
  - Längere Berechnungsdauer

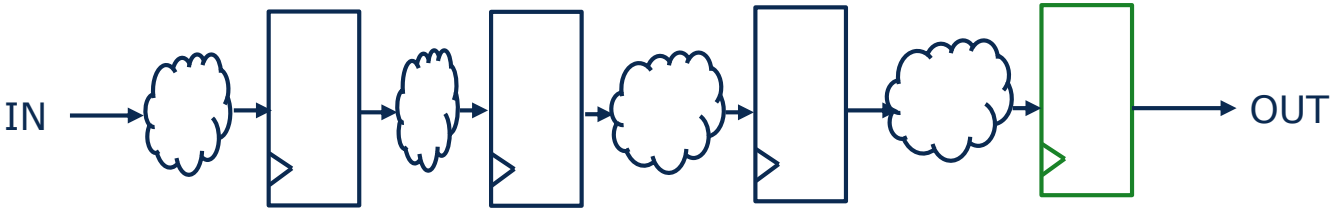


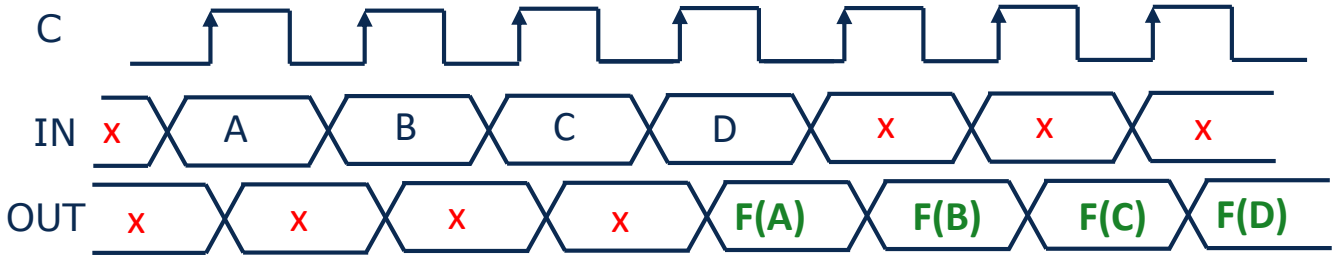


- Latenz N Takte
- Datenrate 1/N Operationen pro Takt



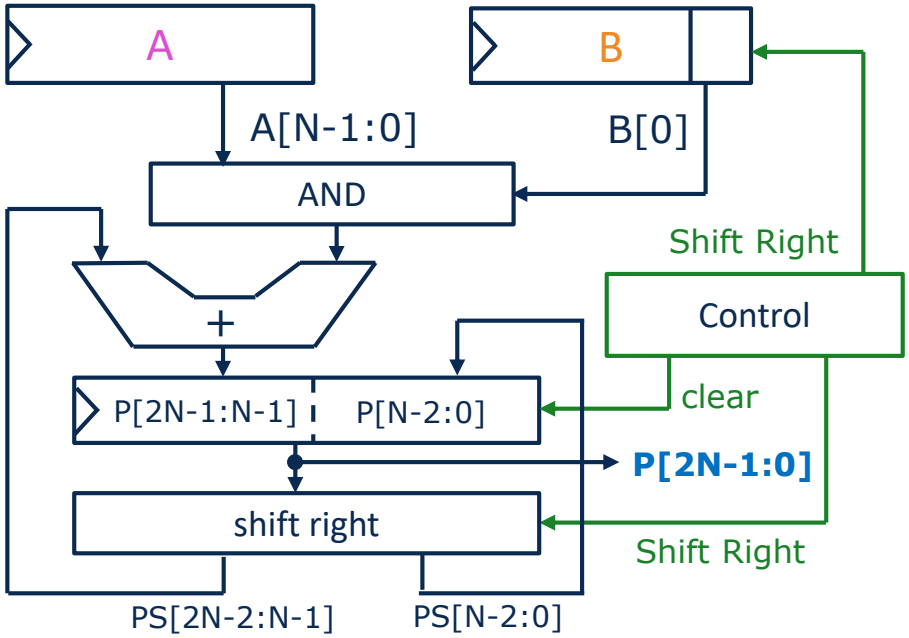
- **Parallele** Abarbeitung von Teilaufgaben (Pipelining), durch:
  - Aufspalten der Logik pro Rechenschritt in **separate** Schaltungen („Loop Unrolling“)
  - Oder: Einfügen von Registerstufen in kombinatorische Arithmetische Baublöcke



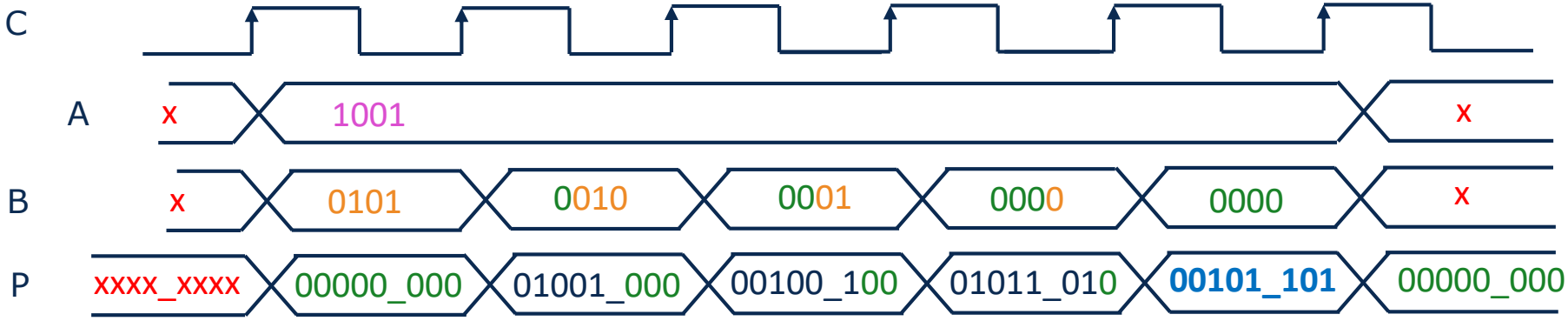


- Latenz N Takte
- Datenrate 1 Operation pro Takt

Beispiel: Sequentieller Binär-Multiplizierer



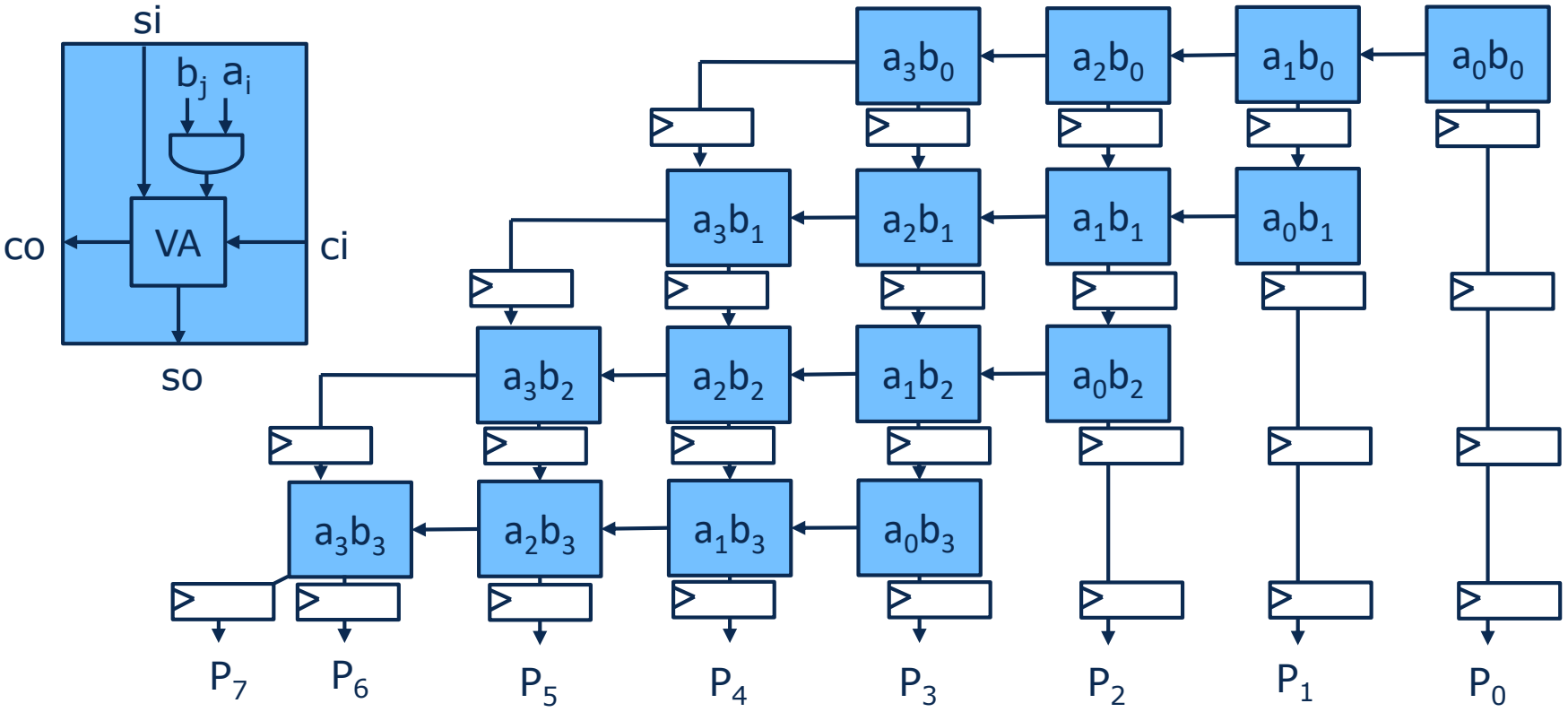
					1	0	0	1	
				1	0	0	1		1
	+			0	0	0	0		0
	+		1	0	0	1			1
	+	0	0	0	0				0
P	0	0	1	0	1	1	0	1	



→ Ergebnis in N+1 Takten

Beispiel: Pipeline Multiplizierer

- Einfügen von Registerstufen in einen binären Multiplizierer
- Latenz: 4 Takte, Datendurchsatz: 1 Ergebnis/Takt



Zusammenfassung Datenpfade

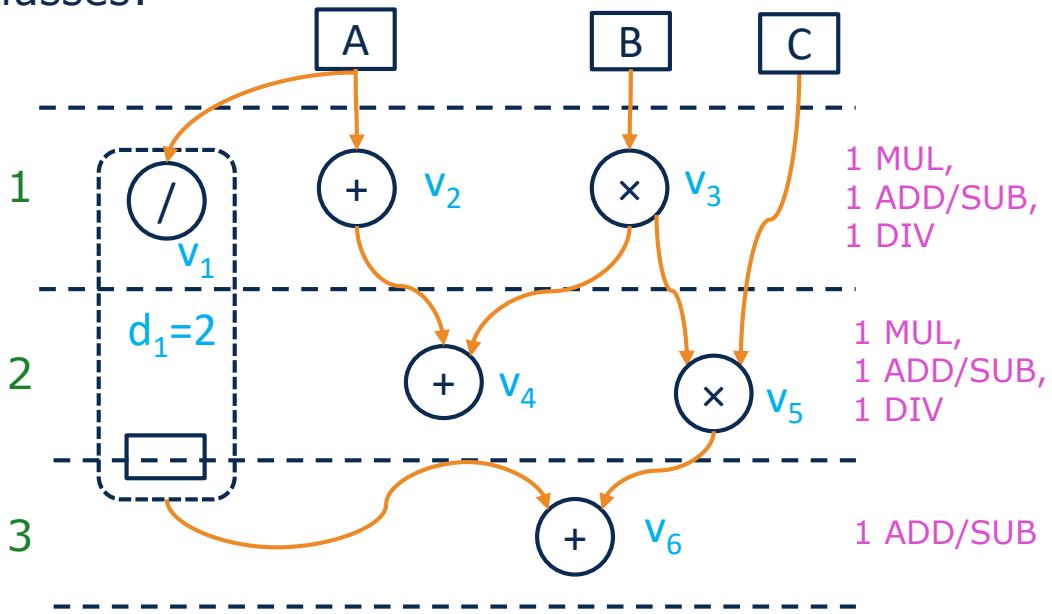
- Vorstellung von
  - Datenpfadbaublöcken
  - Speicherarchitekturen
  - Bussystemen
- Timing Optimierung von sequentiellen Datenpfadbaublöcken
  - Einfügen von Registerstufen
  - Sequential Re-Timing
  - Pipelining

Realisierung von Algorithmen in Hardware

- Algorithmus Beschreibungsformen
  - Textform
  - Gleichungen
  - (Pseudo-) Code
  - Struktogramm
  - …

- Szenario 1:
  - Programmierung einer gegebenen Hardware **(Prozessor)**
    - Gegebener Datenpfad, Speicherarchitektur, Befehlssatz
  - **Abbilden** des Codes auf die Hardware durch Compiler
- Szenario 2:
  - Entwurf einer für den Algorithmus spezifischen Hardware (ASIC)
    - **Entwurf** von Datenpfad, Speicherarchitektur, Control-Flow

- Formale Darstellung des Datenflusses:



- Operationen $V=\{v_0,v_1,v_2,\ldots\}$
- Verzögerung $D=\{d_0,d_1,d_2,\ldots\}$
- Menge an Kanten $E=\{e_{ij},\ldots\}$
- $Pred_{vi}$ alle Vorgänger von v_i
- $Succ_{vi}$ alle Nachfolger von v_i

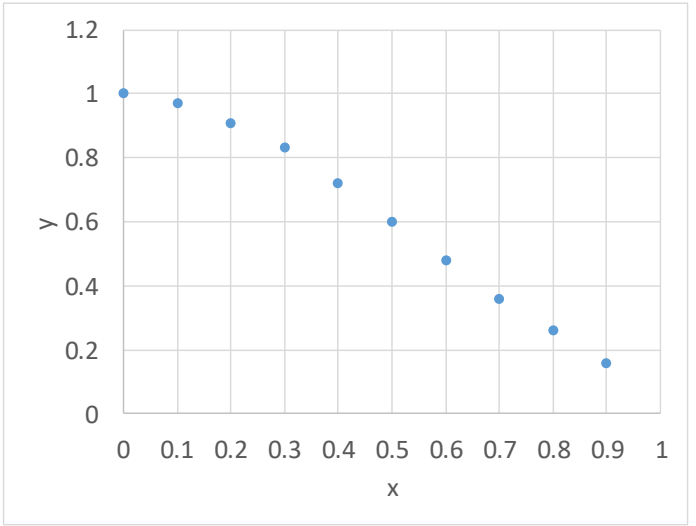
- Die „Laufzeit" einer Operation wird in Takt Zyklen angegeben
- Darstellung der **Startzeit** und der **Verzögerungszeit**
- Angabe der notwendigen **Hardware Ressourcen**

- Festlegen der Abarbeitungsreihenfolge (Schedule) der Operationen
- Berücksichtigung von Randbedingungen (Constraints)
  - Laufzeit (Timing Constraints)
  - Hardwareaufwand (Ressource Constraints)
- Scheduling ist ein Optimierungsproblem
  - Festlegen der Start-Zeit der Operationen
  - Einhalten der gegebenen Randbedingungen

- Ergebnis des Scheduling:
  - Erstellen eines DFG
  - Daraus abgeleitet:
    - Datenpfad
      - Welche und wie viele Datenpfadbaublöcke?
      - Welche und wie viele Register?
      - Wie viele Busse?
    - Register-Transfer Folge zum Entwurf des Steuerwerks

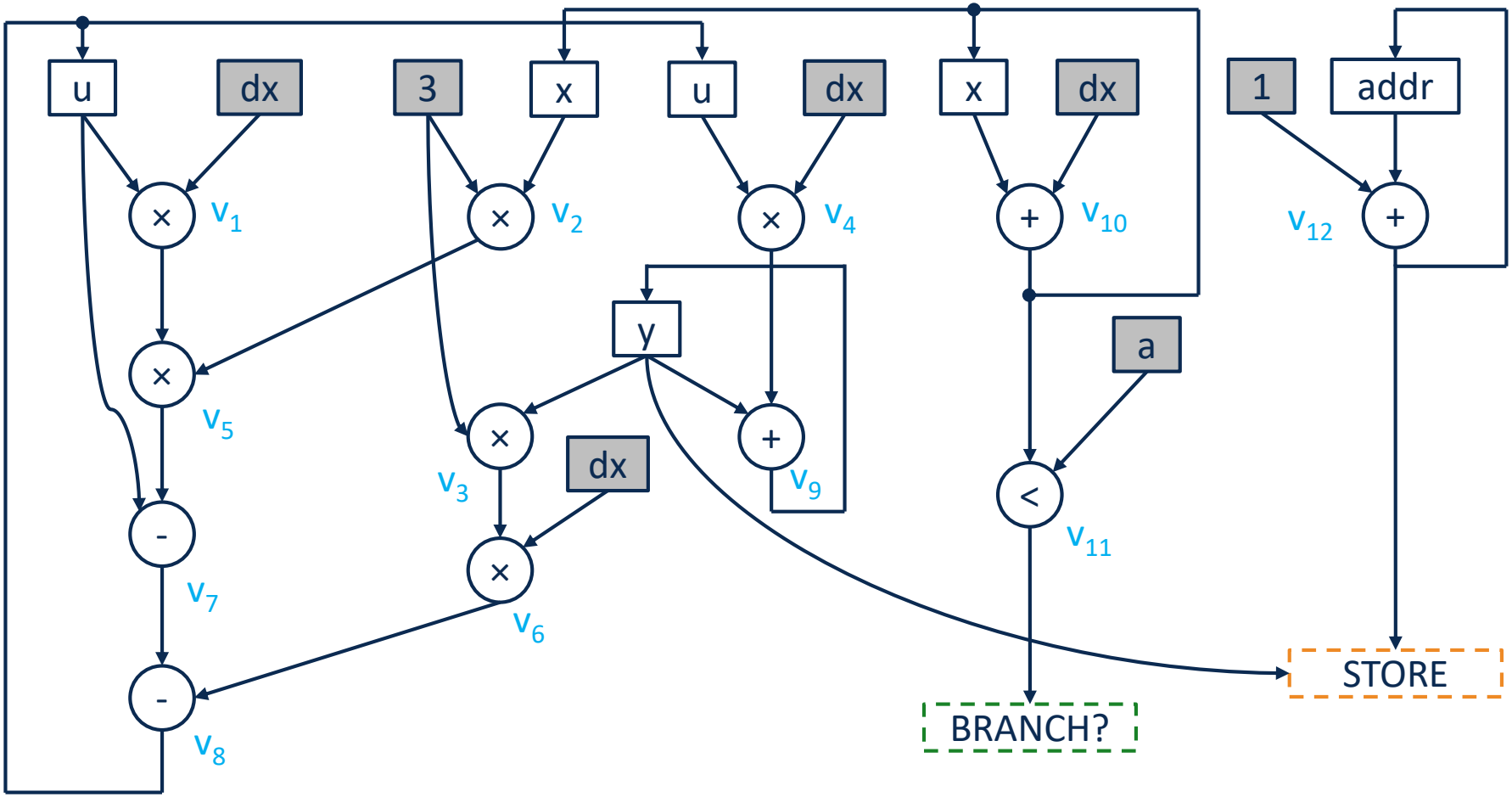
- Numerische DGL Lösung: y‘‘+3xy‘+3y=0
- Euler-Vorwärts-Verfahren
- Speichern des Ergebnis im RAM an ADDR

```
WHILE (x<a) DO
       x1:=x+dx;
       u1:=u-(3·x·u·dx)-(3·y·dx);
       y1:=y+(u·dx);
       x := x1; u := u1; y :=y1;
       addr := addr+1; STORE(y1,addr);
ENDWHILE
```



Quelle: Scheduling Algorithms for High-Level Synthesis, Zoltan Baruch

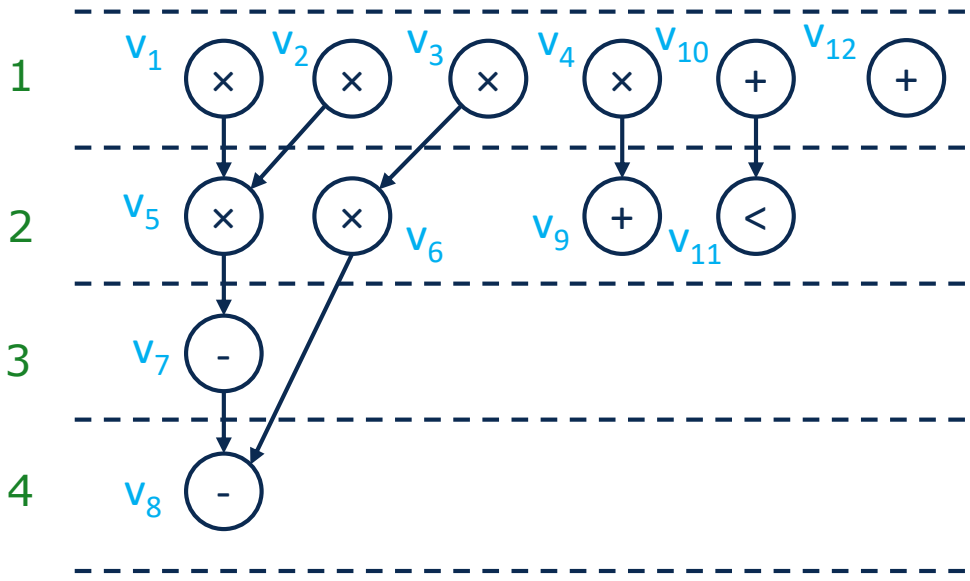
Beispiel Algorithmus Datenfluss



Quelle: Scheduling Algorithms for High-Level Synthesis, Zoltan Baruch

As Soon As Possible (ASAP) Scheduling

- Scheduling Algorithmus:
  - Wiederhole für alle Knoten v_i
    - Auswahl eines v_i dessen **Vorgänger** alle zugewiesen sind
    - Terminiere v_i → $\mathbf{t_i = max\ (t_j + d_j)}$ für alle j aus $Pred_{vi}$ (→ frühester Zeitpunkt)
  - Bis alle v_i zugewiesen sind



4 MUL, 2 ADD/SUB

2 MUL, 2 ADD/SUB

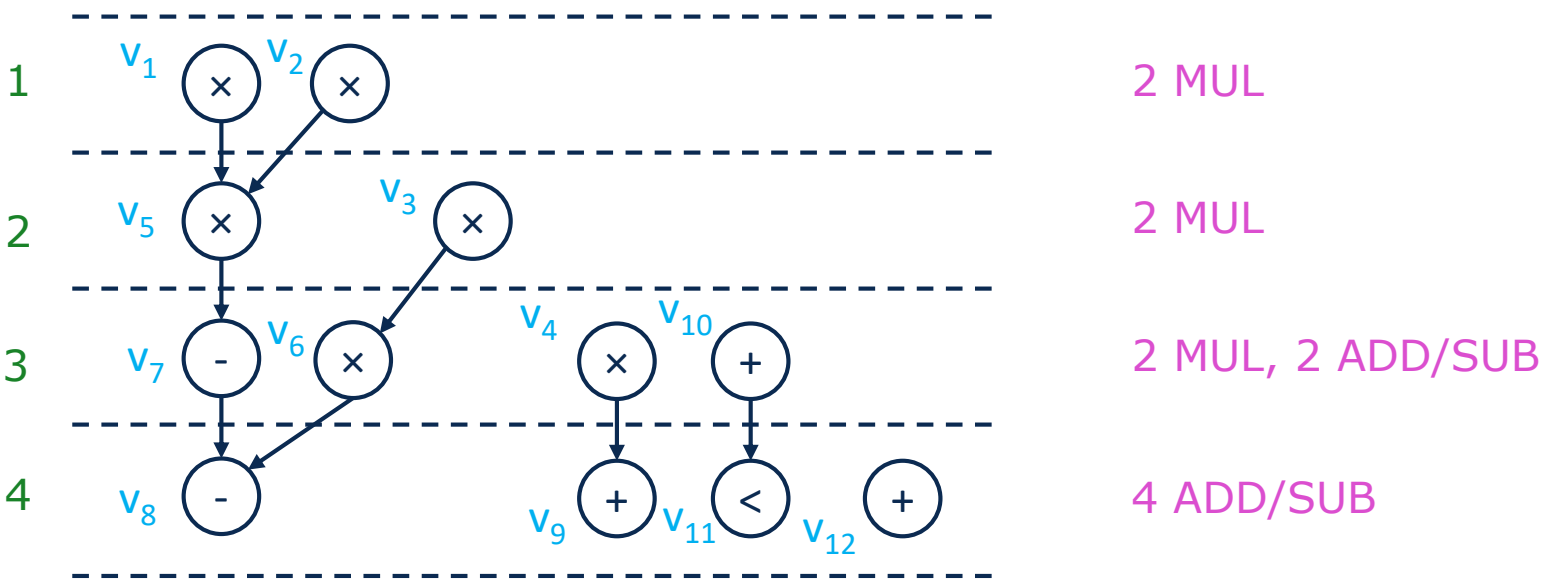
1 ADD/SUB

1 ADD/SUB

→ 4 MUL, 2 ADD/SUB

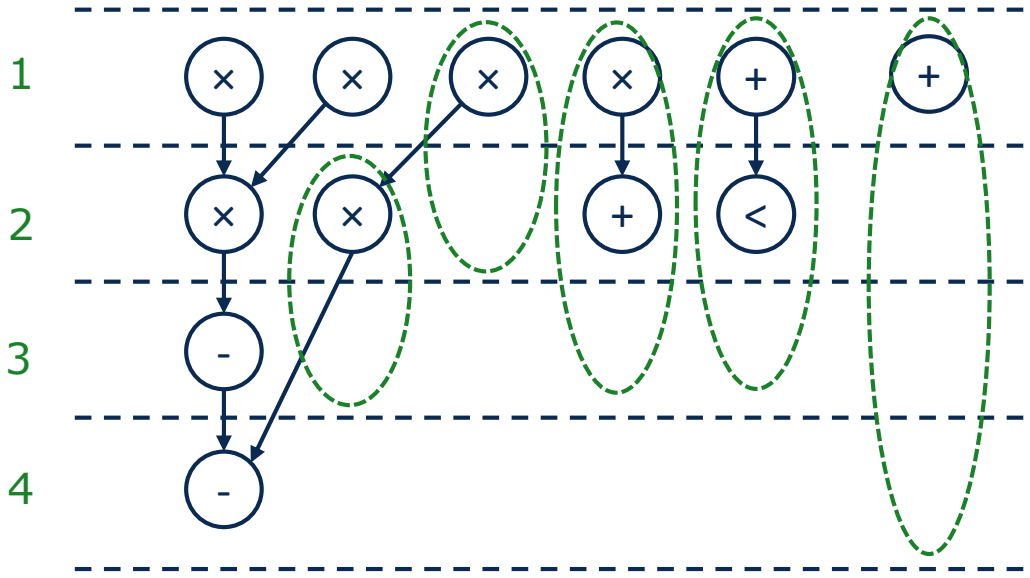
As Late As Possible (ALAP) Scheduling

- Scheduling Algorithmus :
  - Wiederhole für alle Knoten v_i
    - Auswahl eines v_i dessen **Nachfolger** alle zugewiesen sind
    - Terminiere v_i → $\mathbf{t_i = min\ (t_j - d_i)}$ für alle j aus $Succ_{vi}$ (→ spätester Zeitpunkt)
  - Bis alle v_i zugewiesen sind

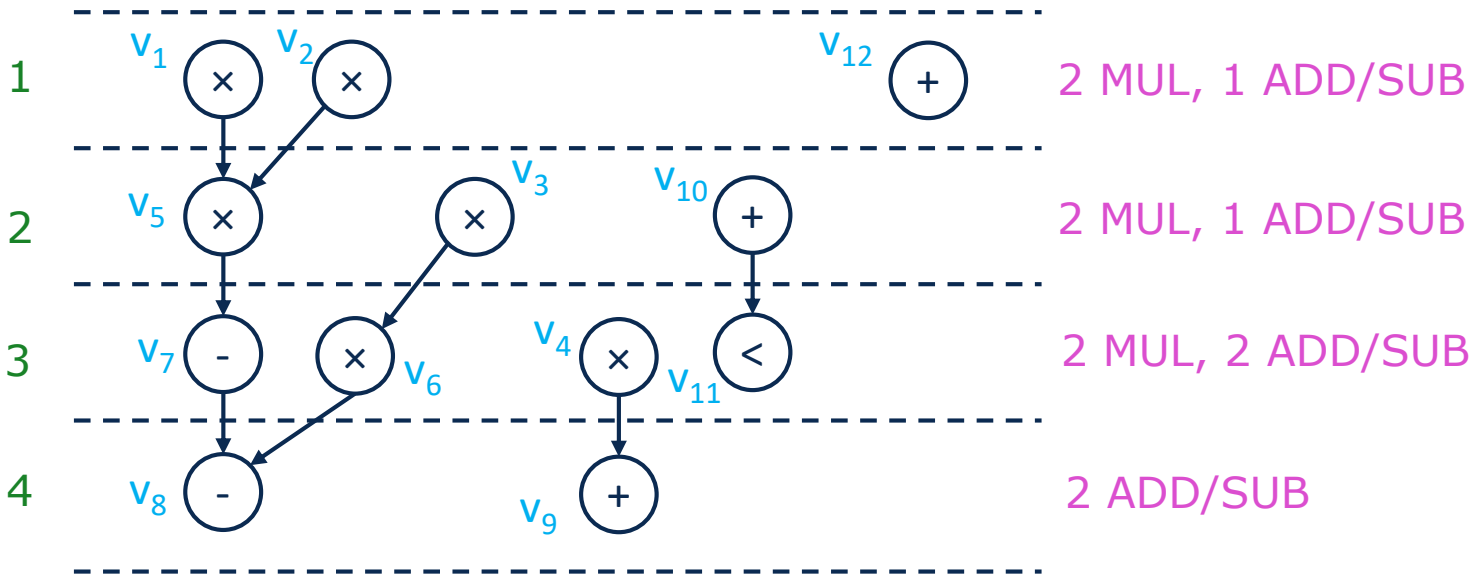


→ 2 MUL, 4 ADD/SUB

- Freiheitsgrad des Startzeitpunktes von Operationen ohne Auswirkung auf die Latenz des DFG



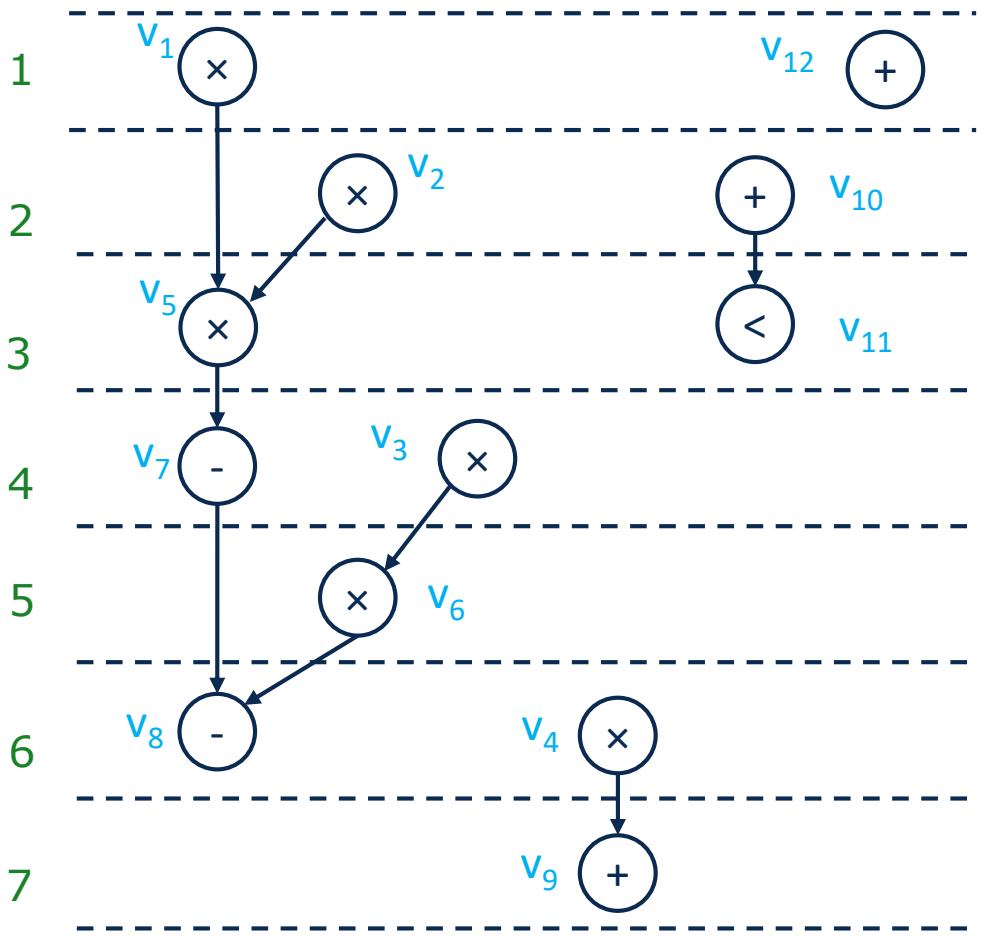
- Scheduling für
  - gegebene Hardware Ressourcen (Chipfläche)
  - Extremfall: minimale Hardware (Module, Busse)
- Nutzung der Mobility (keine Erhöhung der Latenz):



→ 2 MUL, 2 ADD/SUB

Beispiel: Scheduling für minimale Hardware

- Einfügen zusätzlicher Takte → Erhöhung der Latenz



1 MUL, 1 ADD/SUB

1 MUL, 1 ADD/SUB

1 MUL, 1 ADD/SUB

1 MUL, 1 ADD/SUB

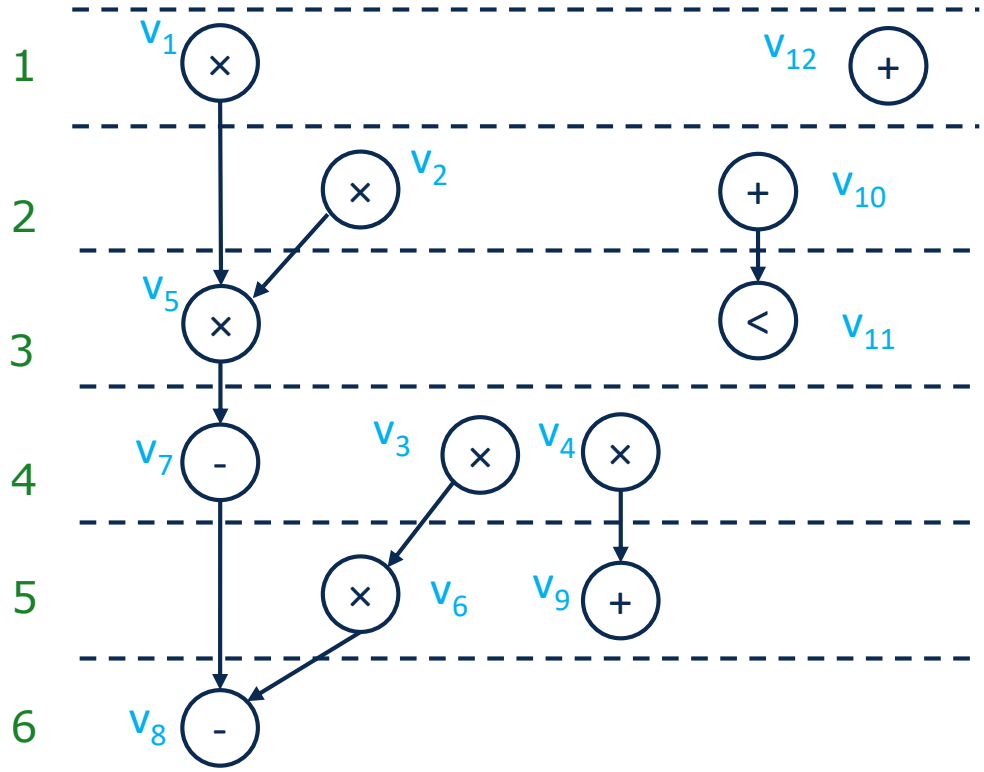
1 MUL

1 MUL, 1 ADD/SUB

1 ADD/SUB

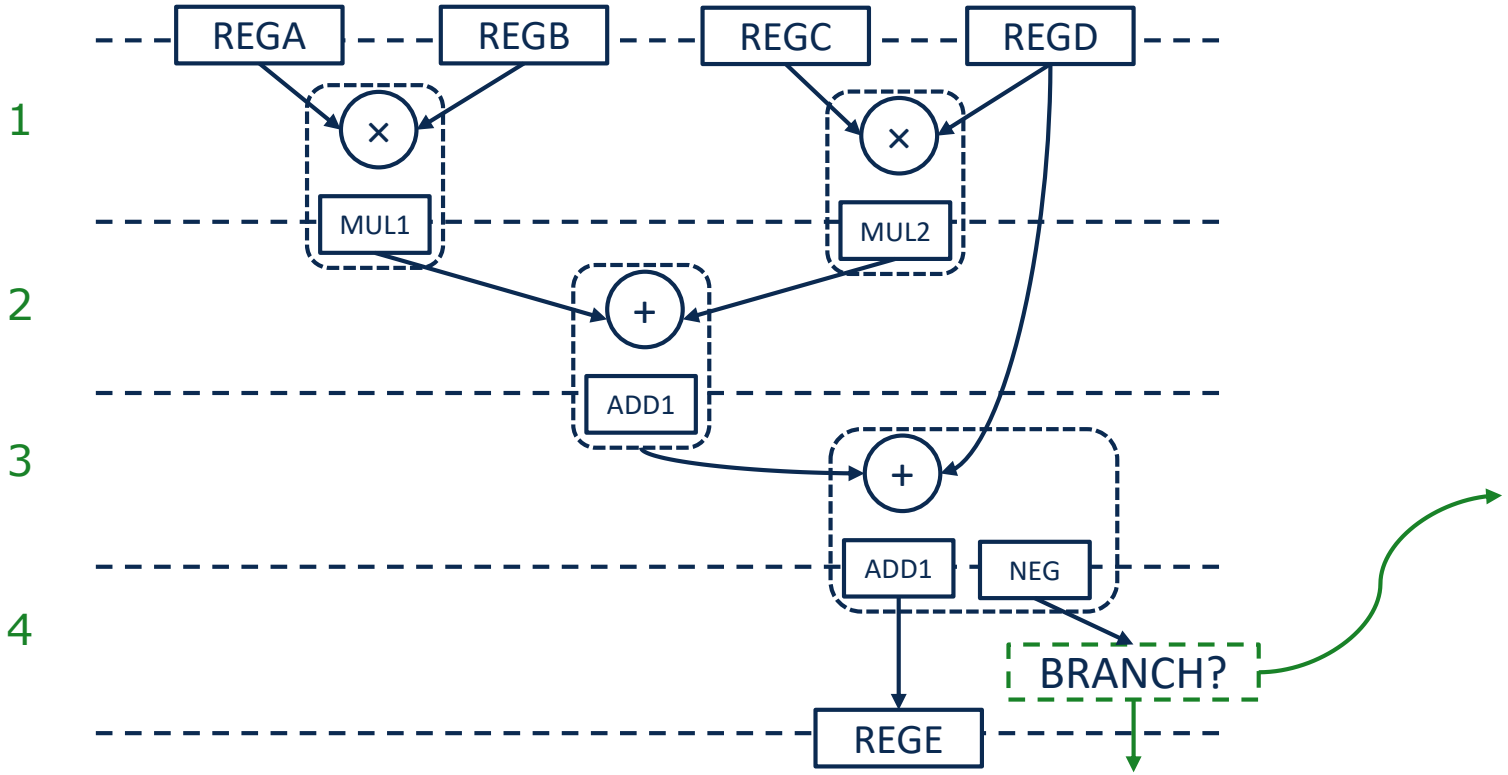
→ 1 MUL, 1 ADD/SUB

- Scheduling für gegebene Abarbeitungszeit
- Anwendung z.B. digitale Signalprozessoren (DSP) mit Echtzeit Anforderung → Ergebnis muss nach gegebener Taktzahl vorliegen
- Beispiel: Minimale Hardware bei 6 Takten Latenz



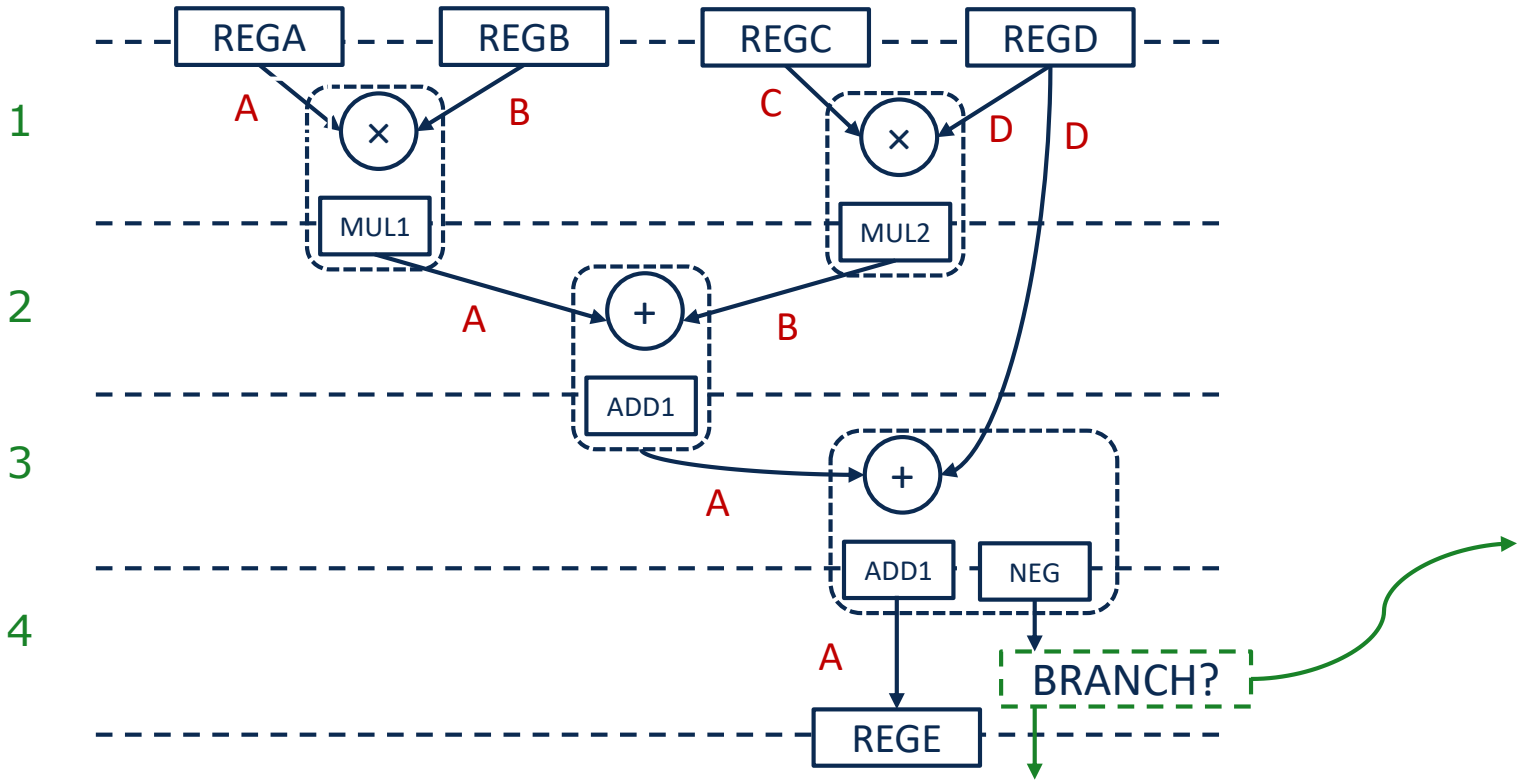
Register und Verzweigungen im DFG

- Detaillierter DFG mit expliziter Darstellen von Registern
  - Visualisierung der Speicherung von Daten → Register Hardware
  - Grundlage des Entwurfs von Pipelines
- Explizite Darstellung von Verzweigungen

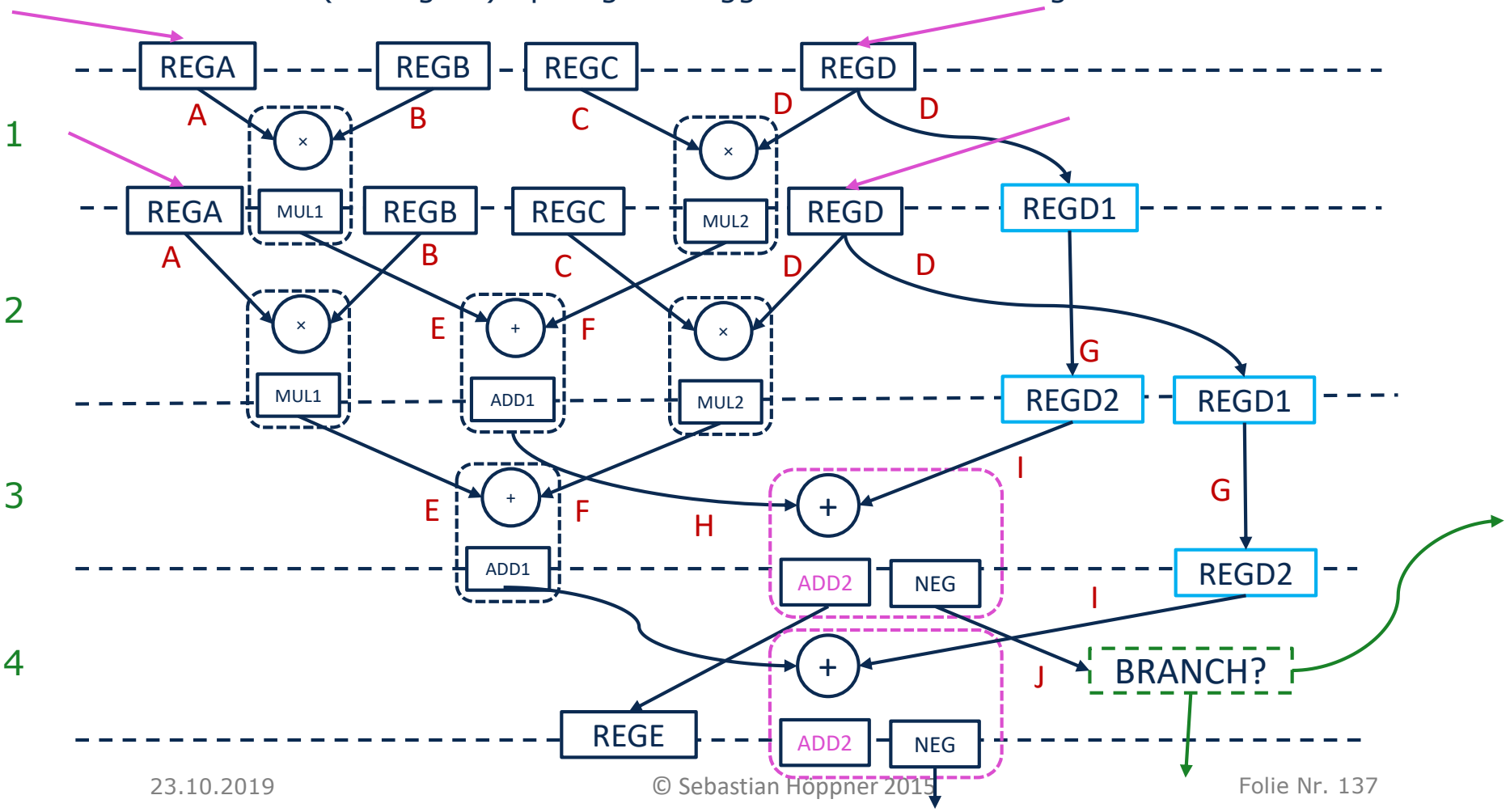


Darstellung von Datenbussen im DFG

- Zuweisung von Datenbussen an Kanten des DFG
- → Bestimmung der Anzahl der notwendigen Datenbusse

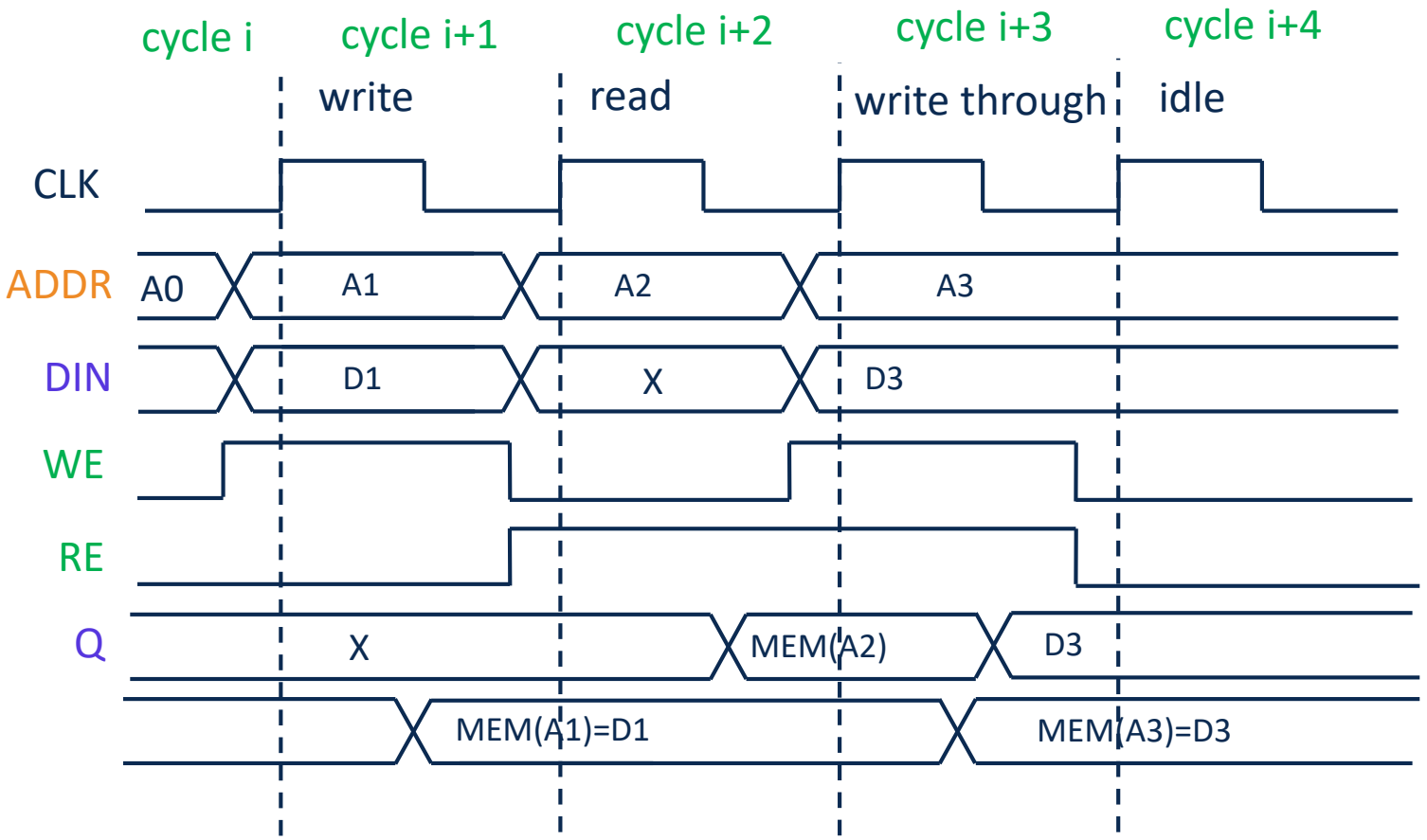


- Aufbau von Pipelines durch bei Bedarf Hinzufügen von:
  - Registern, Datenpfadbaublöcken, Bussen
- Durchsatz von 1 Ergebnis pro Taktzyklus bei N Takten Latenz
- Beachten von (bedingten) Sprüngen! → ggf. Verwerfen von Ergebnissen



Darstellung von Speicherzugriffen im DFG

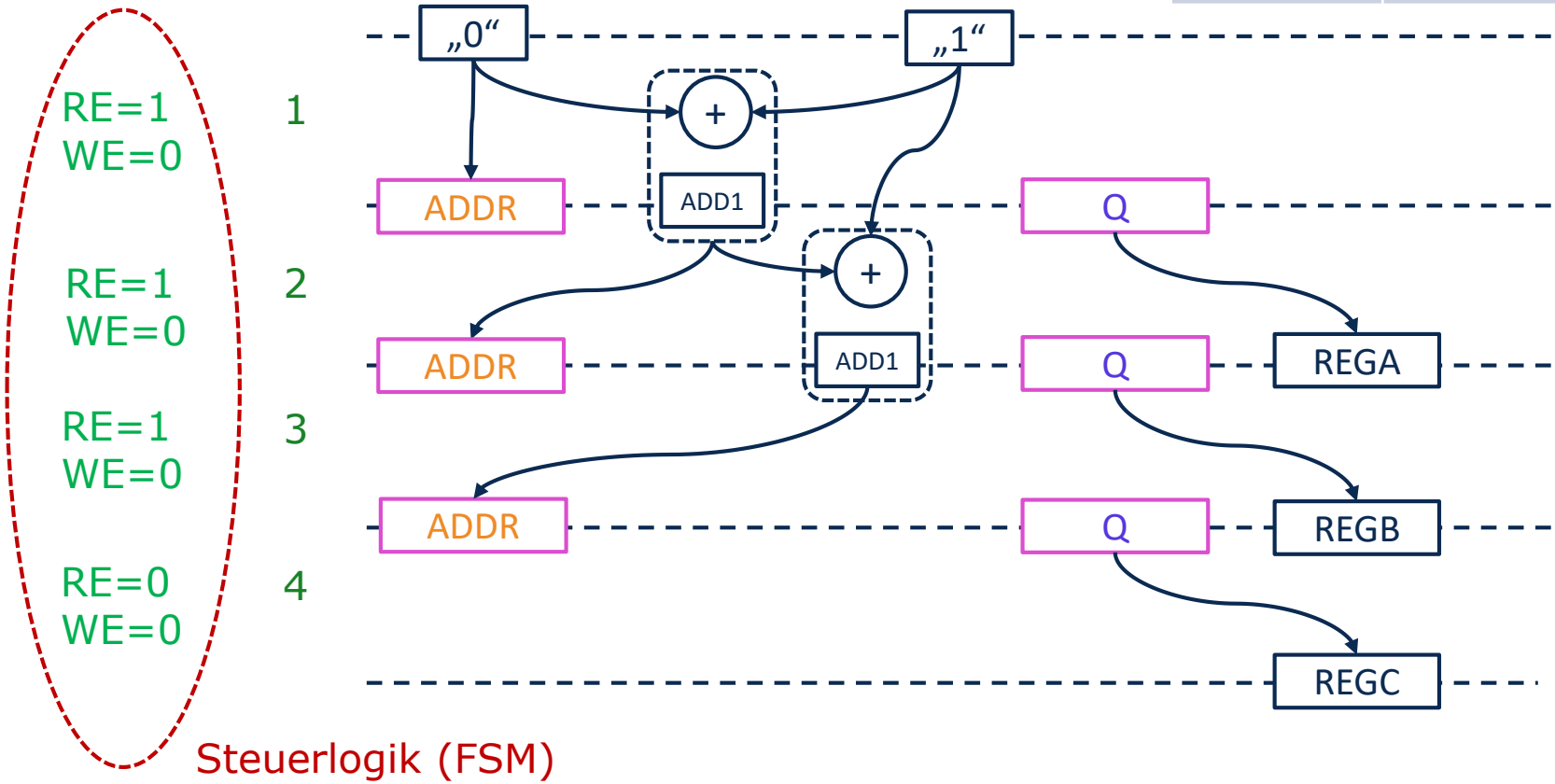
- Sequentieller SRAM Zugriff



Darstellung von Speicherzugriffen- Lesen vom SRAM

- Berechnen der Adresse durch Fixed-Point ALU
- Betrachtung der SRAM Signale wie Register
- Beispiel:

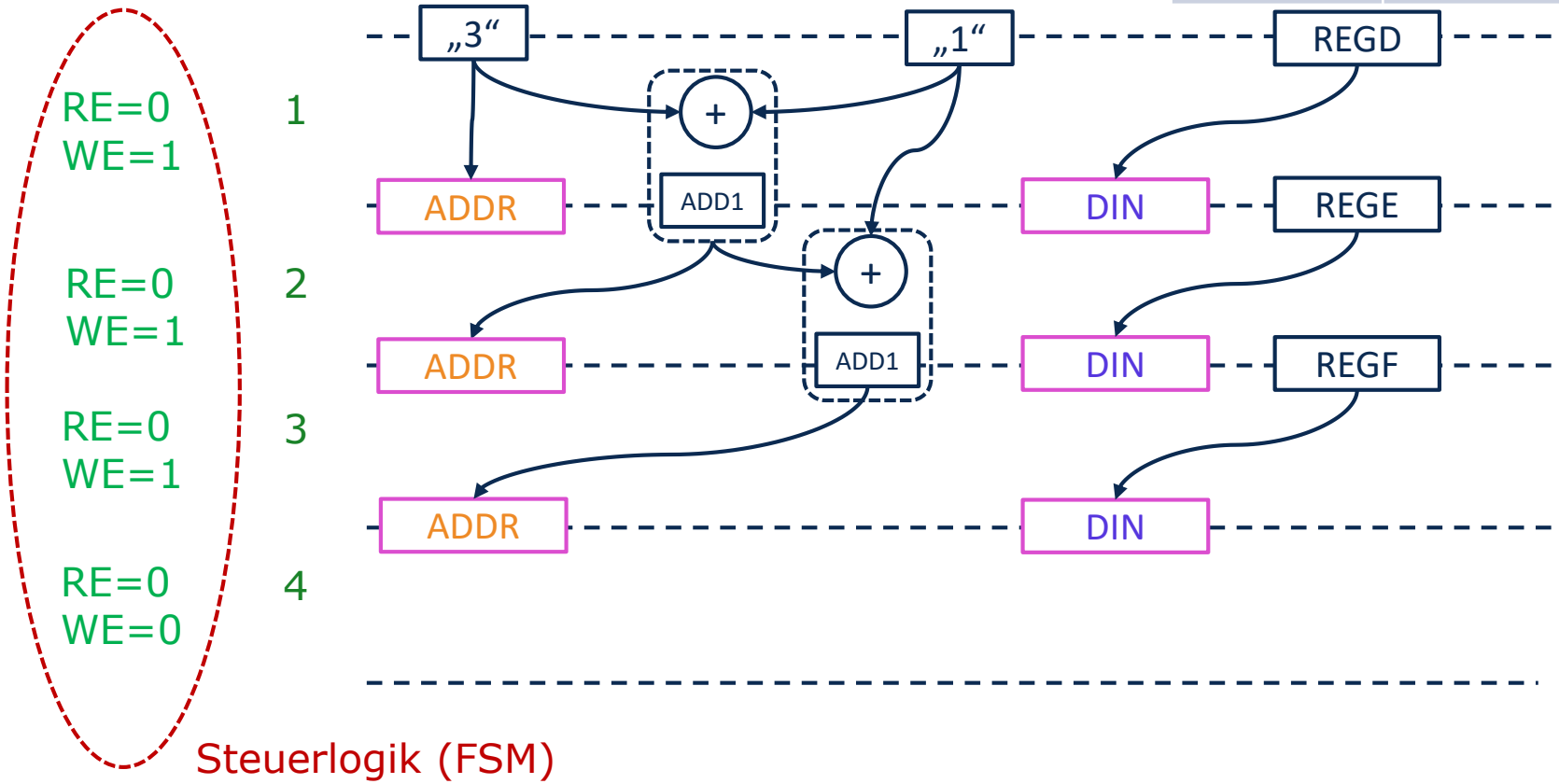
Adresse	Aktion
0	Lese nach REGA
1	Lese nach REGB
2	Lese nach REGC



Darstellung von Speicherzugriffen- Schreiben auf SRAM

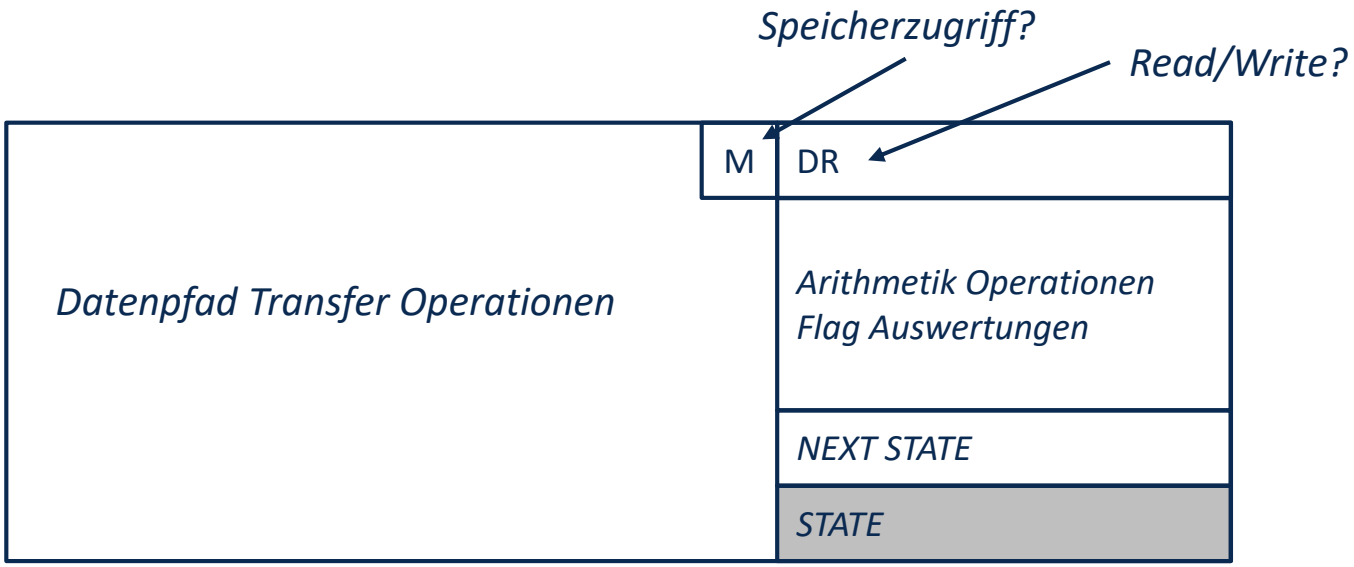
- Berechnen der Adresse durch Fixed-Point ALU
- Betrachtung der SRAM Signale wie Register
- Beispiel:

Adresse	Aktion
3	Schreibe von REGD
4	Schreibe von REGE
5	Schreibe von REGF



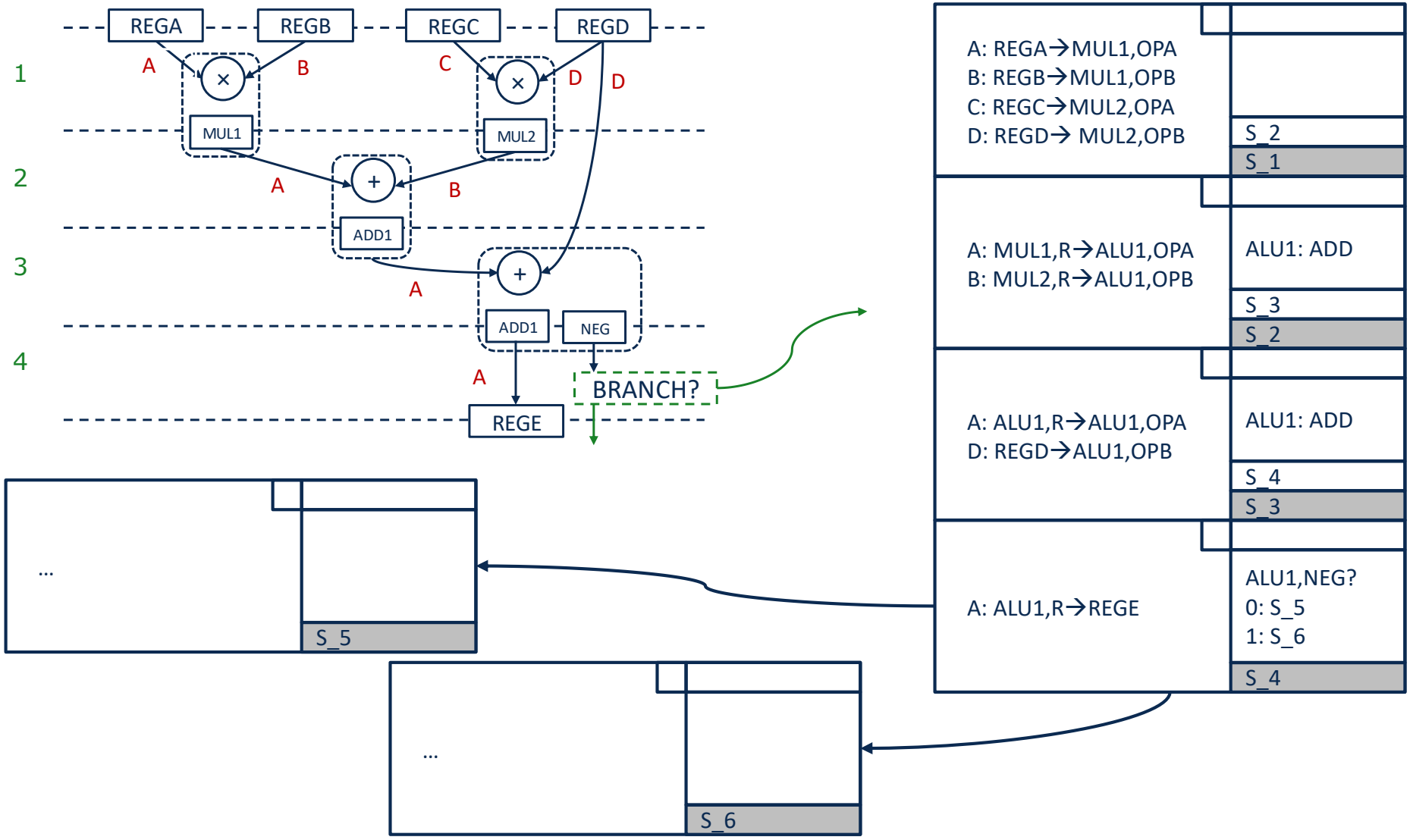
- Konstruktion des Datenpfades aus einem DFG
- Bestimmen der Anzahl von
  - Modulen
  - Registern
  - Busse/Datenverbindungen
- Entwurf des Datenpfades mit Bussystem (z.B.Multiplexer)
  → siehe Abschnitt Datenpfade/Busse

- Darstellung der Abläufe in Register Transfer Notation



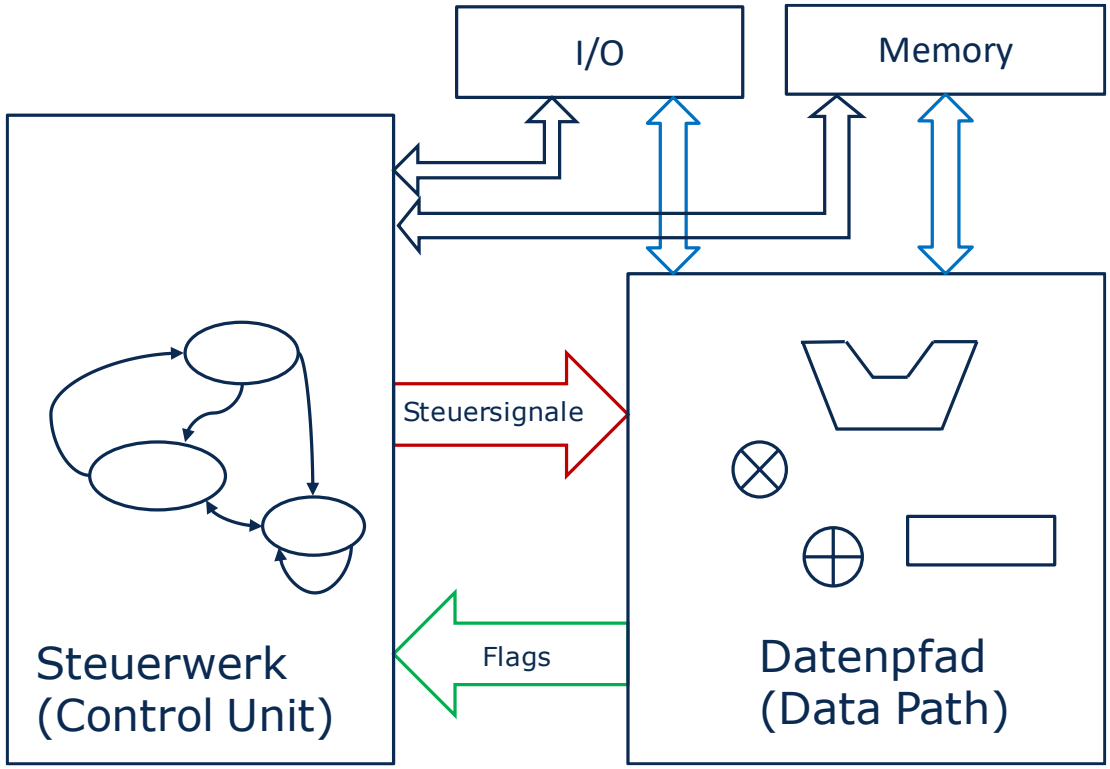
- Ein Register-Transfer Block beschreibt **einen** Taktzyklus

Beispiel RT-Folge



- Die RT-Darstellung legt die Zustände und Zustandsübergänge der FSM fest
  - Bedingungen: Flags
- Die RT-Darstellung legt die Signale → Ausgangslogik der FSM
  - Schalten von Bussen (Tristate-Treiber, Multiplexer)
  - Konfigurieren von Datenpfadbaublöcken (ADD/SUB)
  - …
- → siehe Abschnitt FSM
- → Details und ausführliches Beispiel siehe Praktikumsanleitung

- Vorstellung des Datenflussgraphen, Scheduling und Optimierung
  - ASAP, ALAP, Ressource Constraints und Timing Constraints
  - Pipelining
- Register Transfer Folge → Ableiten von Datenpfad und FSM

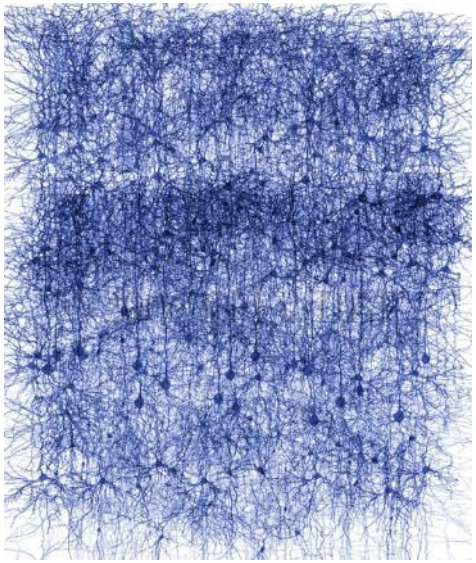


Beispiel: Exponentialfunktion für Neuromorphe Hardware

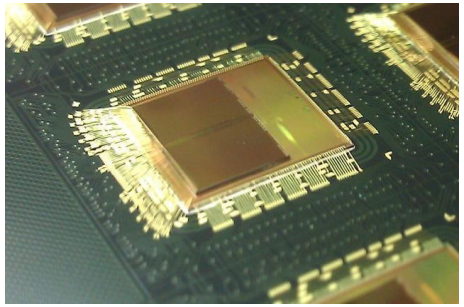
- Neuromorphe Hardware für das Human Brain Project
  - Gehirnsimulation
  - Technische Anwendungen, z.B. Robotik
- Herausforderung:
  - Simulation von einer **großen Anzahl** Neuronen und Synapsen in **biologischer Echtzeit**
- Entwicklung eines Many-Core Computers mit > 4 Millionen ARM Prozessoren (SpiNNaker 2)
  - Architekturentwicklung → University of Manchester
  - Chip-Design → TU Dresden

Quelle: Human Brain Project

Human Brain Project



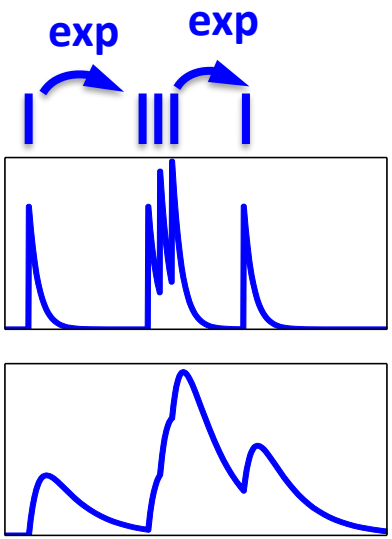
http://bluebrain.epfl.ch

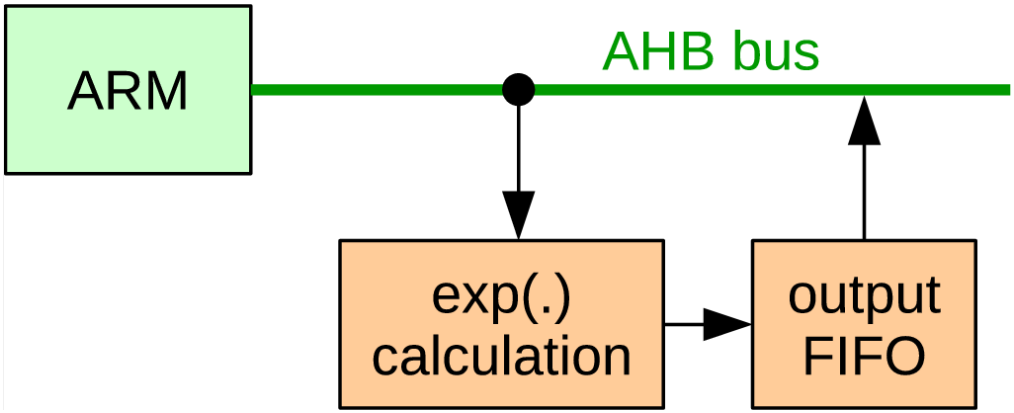


SpiNNaker 1 Chip

Beispiel: Exponentialfunktion für Neuromorphe Hardware

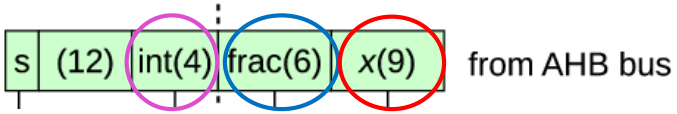
- Berechnung der Dynamiken von Neuronen und Synapsen
- Dabei häufig verwendet: $\mathbf{e^x}$
- Zahlenformat: 32-Bit, s16.15 fixed-point
- Prozessor Kern (ARM Cortex M4)
  - Bisher **exp()** in Software realisiert (≈30 Takte)
  - → Hardware Beschleuniger für **exp()**,
- Anbindung als AHB Slave an den Prozessor

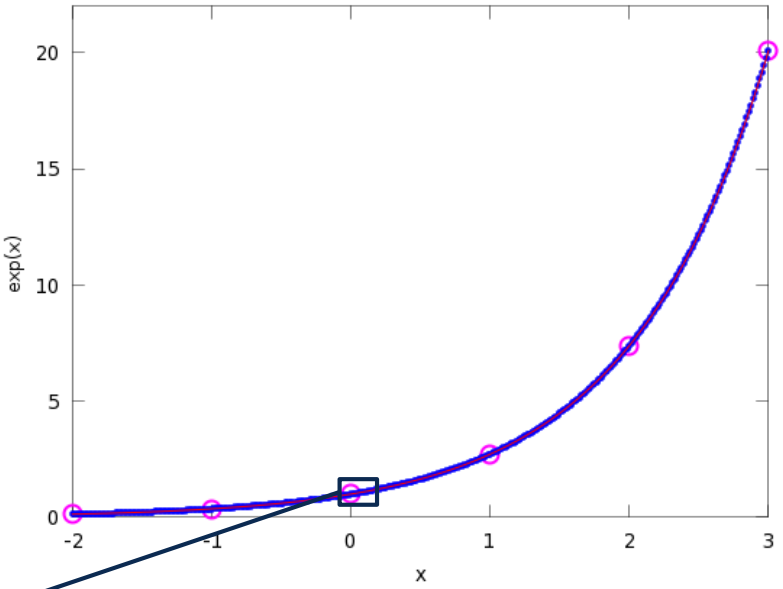


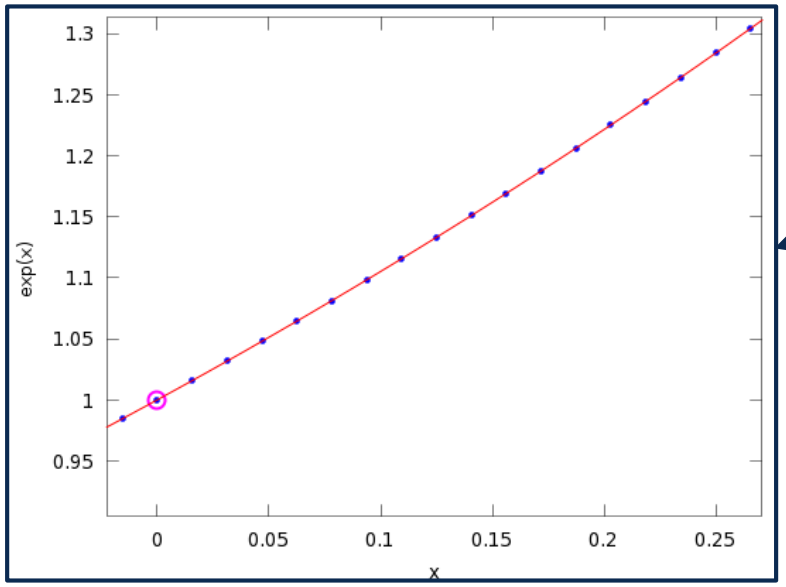


Beispiel: Exponentialfunktion für Neuromorphe Hardware

- Aufteilung des Argumentes der Exponentialfunktion
- $e^x = e^{(xint+xfrac+xlsb)}$
- Lookup-Tabellen für ganzzahligen Teil (int) und fraktionalen Teil (frac)
- Polynomapproximation 4. Grades für LSBs
- Genauigkeit <1LSB für 32-Bit, s16.15 fixed-point Zahlenformat

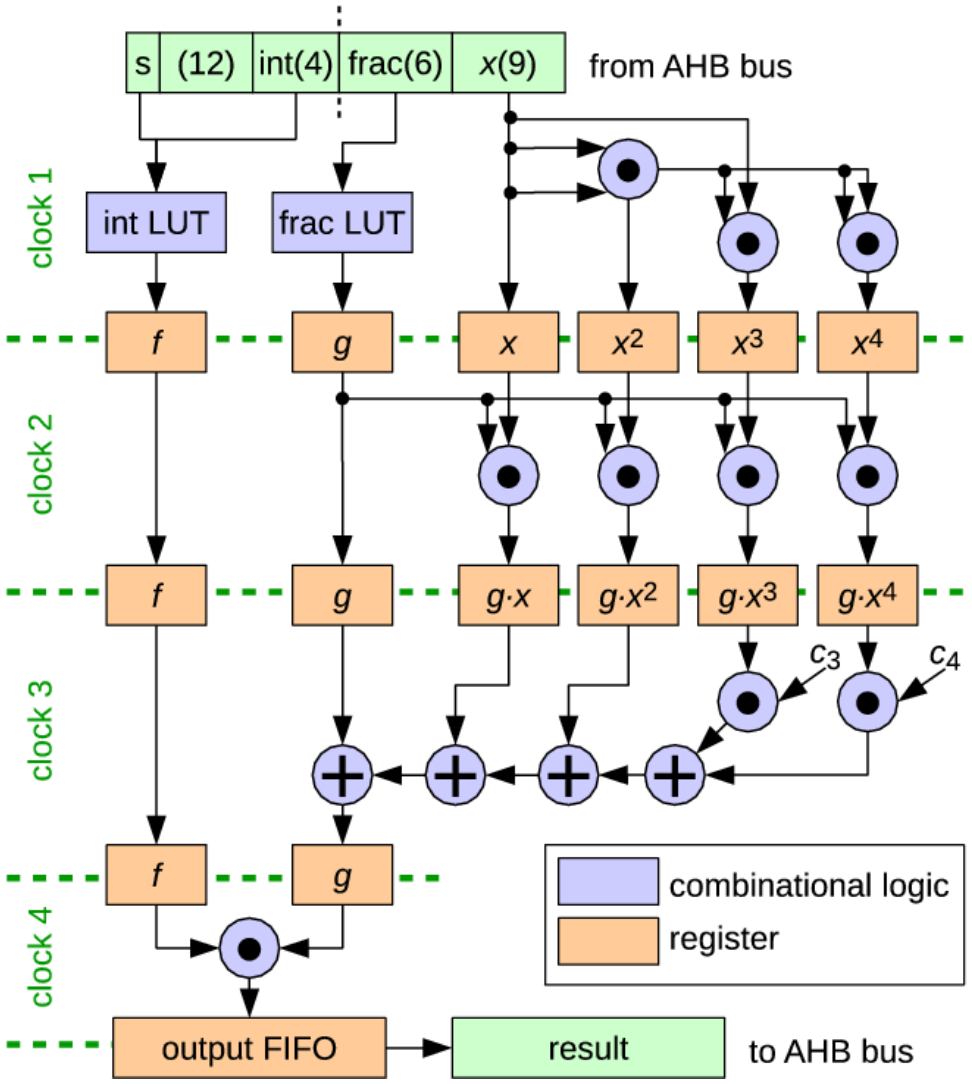






Beispiel: Exponentialfunktion für Neuromorphe Hardware

- Pipeline mit 4 Stufen
- FIFO am Ausgang (32 Werte)
- Realisierung in 28nm CMOS
- >500MHz Taktfrequenz möglich

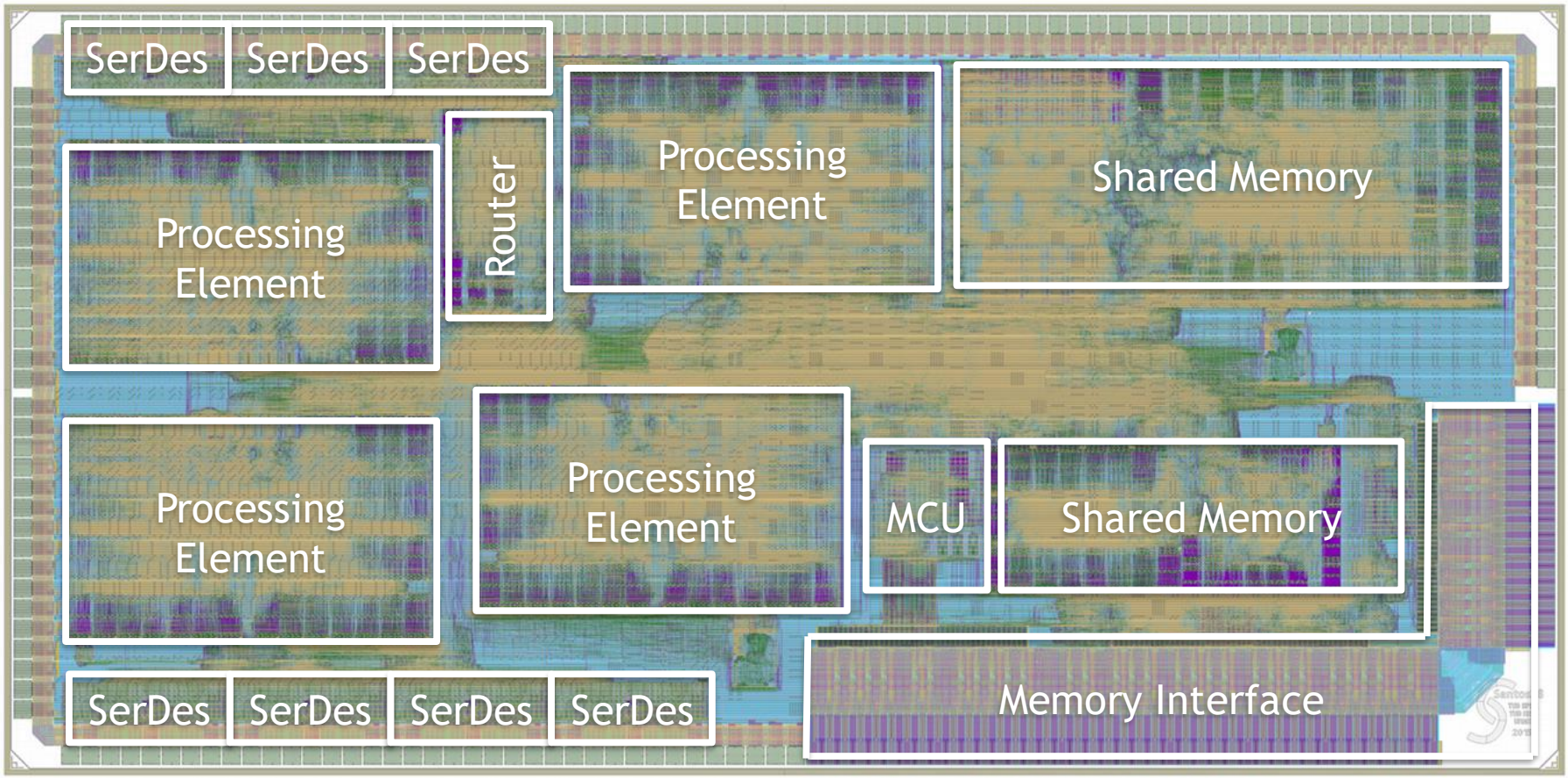


Beispiel: Exponentialfunktion für Neuromorphe Hardware

Measure	exp accelerator	software exp
Throughput @500MHz	250Mexp/s	5.3Mexp/s
Time per exp, pipelined	2clks/exp	95clks/exp
Latency	6clks/exp	95clks/exp
Energy per exp (nominal)	0.44nJ/exp	25nJ/exp
Energy per exp (0.7V/154MHz)	0.21nJ/exp	12nJ/exp
Total area	10800μm^2	-

J. Partzsch et al., "A fixed point exponential function accelerator for a neuromorphic many-core system," 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, 2017, pp. 1-4. doi: 10.1109/ISCAS.2017.8050528

Beispiel: Exponentialfunktion für Neuromorphe Hardware

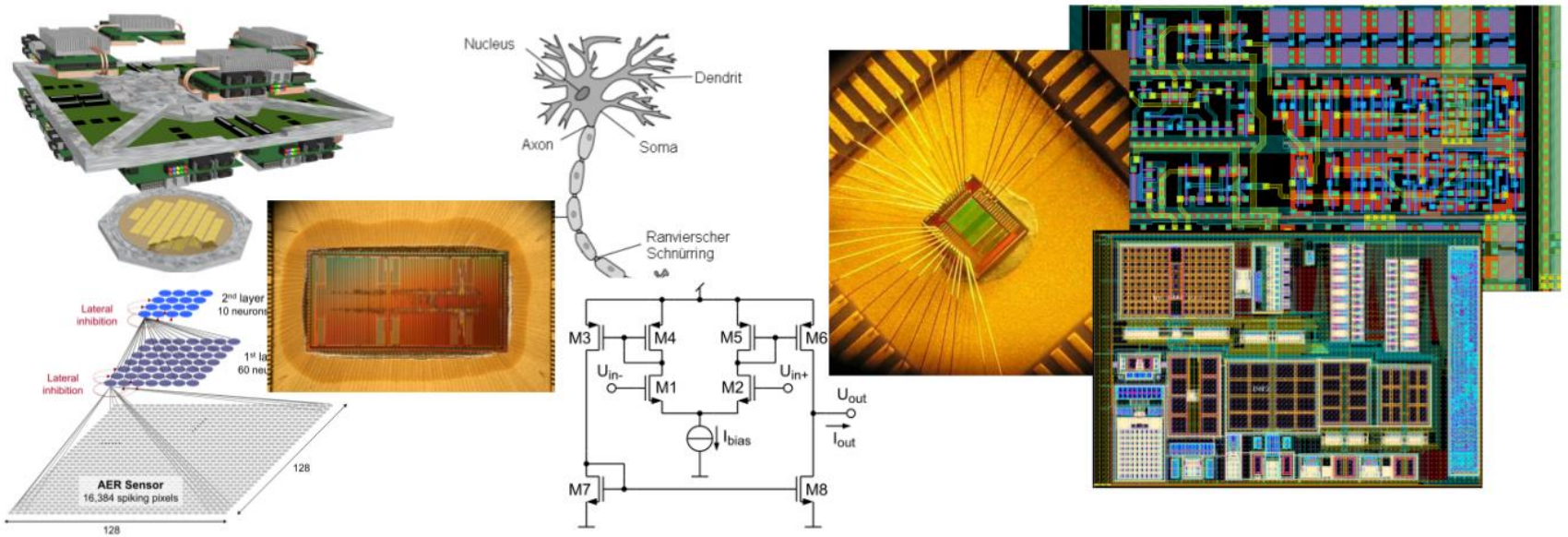


- Testchip: Santos
- 28nm SLP CMOS, 18mm²
- Komponenten Testchip für Neuromorphen Supercomputer
- 4 Processing Elements, Speicherinterface, schnelle Serielle I/Os

Modul Neuromorphic VLSI-Systems

Neuromorphe Systeme

Analoger CMOS-Schaltungsentwurf



- Grundlagen zu neuronalen Netzen und ihrer technischen Realisierung
- Integrierte analoge Schaltungen: Entwurf, Simulation, Verifikation
- Praktischer Schaltungsentwurf und Layout mit Cadence

Human Brain Project