Digitaltechnik – Kapitel 5

TECHNISCHE UNIVERSITÄT DARMSTADT

Prof. Sarah Harris, Ph.D.
Fachgebiet Eingebettete Systeme und ihre Anwendungen (ESA)
Fachbereich Informatik

WS 15/16

Kapitel 5 : Themenübersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Einleitung**
- **Arithmetische Schaltungen**
- **Zahlendarstellungen**
- **Sequentielle Grundelemente**
- **Speicherblöcke**
- **Programmierbare Logikfelder und -schaltungen**

Einleitung

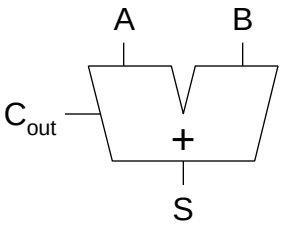
TECHNISCHE UNIVERSITÄT DARMSTADT

- Grundelemente digitaler Schaltungen:
  - Gatter, Multiplexer, Decoder, Register, Arithmetische Schaltungen, Zähler, Speicher, programmierbare Logikfelder
- Grundelemente veranschaulichen
  - Hierarchie: Zusammensetzen aus einfacheren Elementen
  - Modularität: Wohldefinierte Schnittstellen und Funktionen
  - Regularität: Strukturen leicht auf verschiedene Größen anpassbar
- Grundelemente werden verwendet zum Aufbau eines eigenen Mikroprozessors
  - Kapitel 7

1-Bit Addierer

TECHNISCHE UNIVERSITÄT DARMSTADT

Halb-addierer

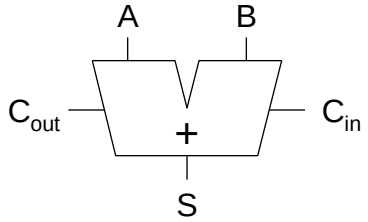


A	B	C_{out}	S
0	0		
0	1		
1	0		
1	1		

S =

C_{out} =

Voll-addierer



C_{in}	A	B	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

S =

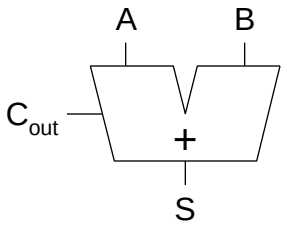
C_{out} =

Embedded Systems & Applications

1-Bit Addierer

TECHNISCHE UNIVERSITÄT DARMSTADT

Halb-addierer

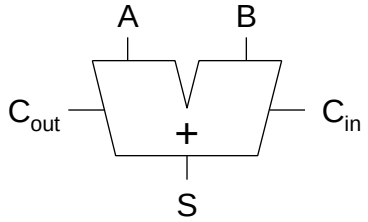


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

S =

C_{out} =

Voll-addierer



C_{in}	A	B	C_{out}	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

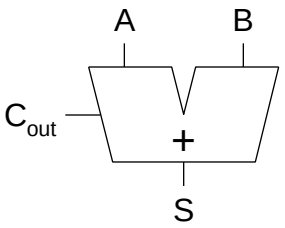
S =

C_{out} =

Embedded Systems & Applications

1-Bit Addierer

Halb-addierer

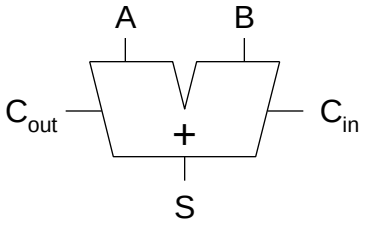


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = A \oplus B$

$C_{out} = AB$

Voll-addierer



C_{in}	A	B	C_{out}	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

$S = A \oplus B \oplus C_{in}$

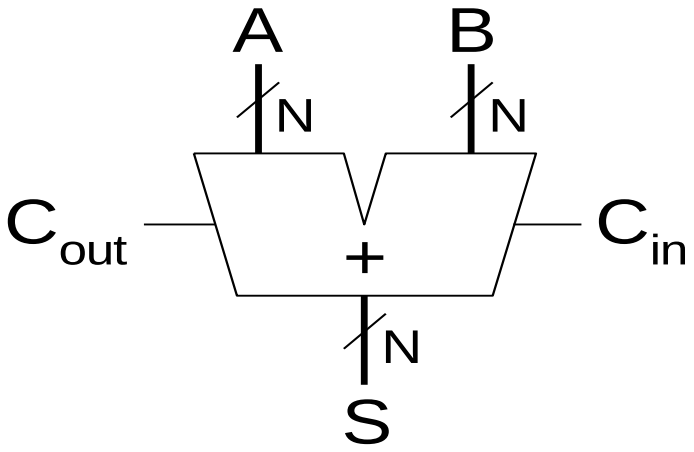
$C_{out} = AB + AC_{in} + BC_{in}$

Mehrbit-Addierer mit Weitergabe von Überträgen

TECHNISCHE UNIVERSITÄT DARMSTADT

Carry-propagate adder (CPA)

Schaltsymbol



Embedded Systems & Applications

Mehrbit-Addierer mit Weitergabe von Überträgen

TECHNISCHE UNIVERSITÄT DARMSTADT

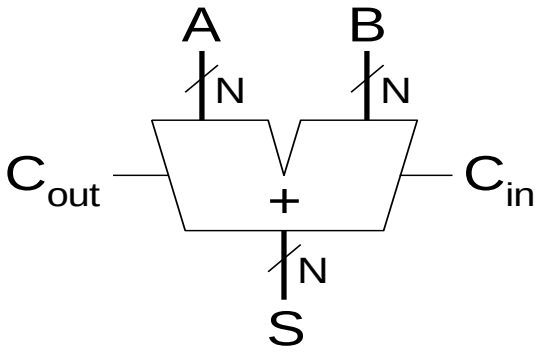
Carry-propagate adder (CPA)

- Verschiedene Typen
  - Ripple-carry-Addierer (langsam)
  - Carry-Lookahead Addierer (schnell)
  - Prefix-Addierer (noch schneller)

Carry-Lookahead und Prefix-Addierer sind schneller bei breiteren Datenworten

- Benötigen aber auch mehr Fläche

Schaltsymbol



Embedded Systems & Applications

Mehrbit-Addierer mit Weitergabe von Überträgen

TECHNISCHE UNIVERSITÄT DARMSTADT

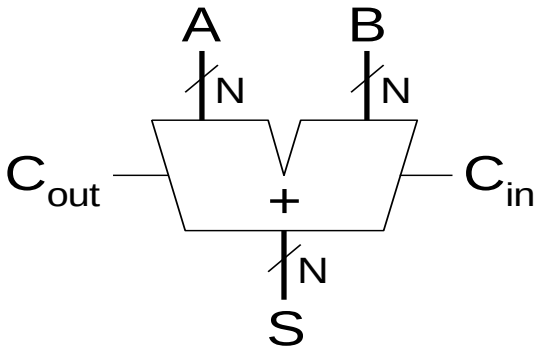
Carry-propagate adder (CPA)

- Verschiedene Typen
  - **Ripple-carry-Addierer** **(langsam)**
  - Carry-Lookahead Addierer (schnell)
  - Prefix-Addierer (noch schneller)

Carry-Lookahead und Prefix-Addierer sind schneller bei breiteren Datenworten

- Benötigen aber auch mehr Fläche

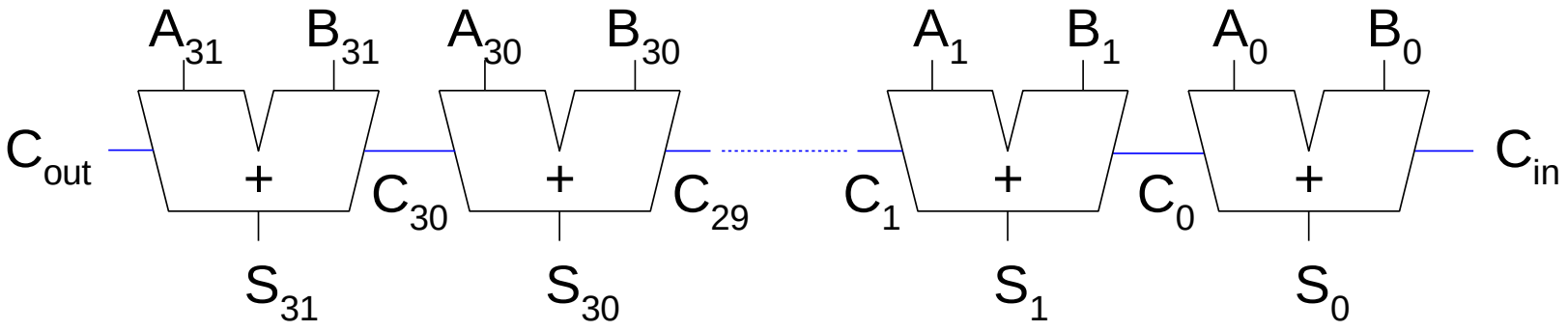
Schaltsymbol



Embedded Systems & Applications

Ripple-Carry-Addierer

- Kette von 1-bit Addierern
- Überträge werden von niedrigen zu hohen Bits weitergegeben
  - Rippeln sich durch die Schaltung
- Nachteil: Langsam



Verzögerung durch Ripple-Carry-Addierer

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verzögerung durch einen *N*-bit Ripple-Carry-Addierer ist

$$t_{ripple} = N\, t_{FA}$$

t_{FA} ist die Verzögerung durch einen Volladdierer

Mehrbit-Addierer mit Weitergabe von Überträgen

TECHNISCHE UNIVERSITÄT DARMSTADT

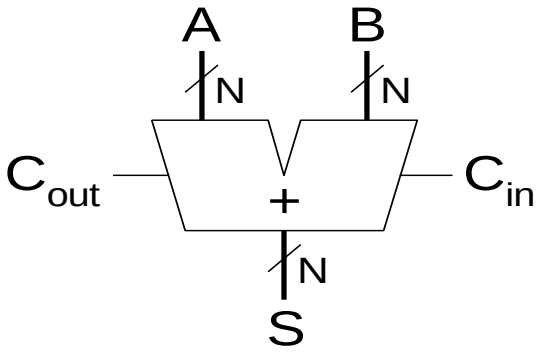
Carry-propagate adder (CPA)

- Verschiedene Typen
  - Ripple-carry-Addierer (langsam)
  - **Carry-Lookahead Addierer** **(schnell)**
  - Prefix-Addierer (noch schneller)

Carry-Lookahead und Prefix-Addierer sind schneller bei breiteren Datenworten

- Benötigen aber auch mehr Fläche

Schaltsymbol



Embedded Systems & Applications

Carry-Lookahead-Addierer (CLA)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Überträge nicht mehr von Bit-zu-Bit
- Stattdessen: Berechne Übertrag C_{out} aus Block von k Bits

Carry-Lookahead-Addierer (CLA)

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Überträge nicht mehr von Bit-zu-Bit**
- **Stattdessen: Berechne Übertrag C_{out} aus Block von *k* Bits**
  - Nun zwei Signale
    - ***Generate*** (erzeuge neuen Übertrag)
    - ***Propagate*** (leite eventuellen Übertrag weiter)

Carry-Lookahead-Addierer (CLA)

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Überträge nicht mehr von Bit-zu-Bit**
- **Stattdessen: Berechne Übertrag C_{out} aus Block von *k* Bits**
  - Nun zwei Signale
    - ***Generate*** (erzeuge neuen Übertrag)
    - ***Propagate*** (leite eventuellen Übertrag weiter)
- **Bits werden in Spalten organisiert**
  - Haben wir eben beim Ripple-Carry-Addierer auch schon gemacht
  - War aber nicht spannend: Es gab nur eine Zeile
  - … ändert sich jetzt

Carry-Lookahead-Addierer: Definitionen

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Eine Spalte *i* erzeugt** (**g**enerates) einen Übertrag falls A_i und B_i beide 1 sind.

$$G_i = A_i B_i$$

- **Eine Spalte leitet einen eingehenden Übertrag weiter** falls A_i oder B_i 1 ist (weitergeleitet = **p**ropagate)

$$P_i = A_i + B_i$$

Embedded Systems & Applications

Carry-Lookahead-Addierer: Definitionen

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Eine Spalte *i* erzeugt** (**g**enerates) einen Übertrag falls A_i und B_i beide 1 sind.

$$G_i = A_i B_i$$

- **Eine Spalte leitet einen eingehenden Übertrag weiter** falls A_i oder B_i 1 ist (weitergeleitet = **p**ropagate)

$$P_i = A_i + B_i$$

- **Eine Spalte (Bit *i*) produziert einen Übertrag an ihrem Ausgang C_i**
  - Wenn sie den Übertrag selbst erzeugt (*Generate,* G_i) oder
  - Wenn sie einen von C_{i-1} eingehenden Übertrag weiterleitet (*Propagate,* P_i)

Embedded Systems & Applications

Carry-Lookahead-Addierer: Definitionen

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Eine Spalte *i* erzeugt** (**g**enerates) einen Übertrag falls A_i und B_i beide 1 sind.

$$G_i = A_i B_i$$

- **Eine Spalte leitet einen eingehenden Übertrag weiter** falls A_i oder B_i 1 ist (weitergeleitet = **p**ropagate)

$$P_i = A_i + B_i$$

- **Eine Spalte (Bit *i*) produziert einen Übertrag an ihrem Ausgang C_i**
  - Wenn sie den Übertrag selbst erzeugt (*Generate,* G_i) oder
  - Wenn sie einen von C_{i-1} eingehenden Übertrag weiterleitet (*Propagate,* P_i)

- Damit ist der Übertrag C_i aus der Spalte *i* heraus

$$C_i = A_i B_i + (A_i + B_i)C_{i-1} = G_i + P_i C_{i-1}$$

Embedded Systems & Applications

Addition im Carry-Lookahead-Verfahren

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Schritt 1:** Berechne G und P-Signale für einzelne Spalten (Einzelbits)
- **Schritt 2:** Berechne G und P Signale für Gruppen von *k* Spalten (*k* Bits)
- **Schritt 3:** Leite C_{in} nun **nicht** einzelbitweise, sondern in *k*-Bit Sprüngen weiter
  - Jeweils durch einen *k*-bit Propagate/Generate-Block

Addition im Carry-Lookahead-Verfahren

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Schritt 1:** Berechne G und P-Signale für einzelne Spalten (Einzelbits)

$$G_i = A_i B_i$$

$$P_i = A_i + B_i$$

Addition im Carry-Lookahead-Verfahren

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schritt 1: Berechne G und P-Signale für einzelne Spalten (Einzelbits)
- **Schritt 2: Berechne G und P Signale für Gruppen von *k* Spalten (*k* Bits)**
- Schritt 3: Leite C_{in} nun nicht einzelbitweise, sondern in *k*-Bit Sprüngen weiter
  - Jeweils durch einen *k*-bit Propagate/Generate-Block

Beispiel: Carry-Lookahead Addierer

TECHNISCHE UNIVERSITÄT DARMSTADT

Schritt 2: Berechne G und P Signale für Gruppen von *k* Spalten (*k* Bits)

- Bestimme $P_{3:0}$ und $G_{3:0}$ Signale für einen 4b Block
- Überlegung: Der 4b Block leitet einen Übertrag direkt weiter
  - … wenn alle Spalten den Übertrag weiterleiten

$$P_{3:0} = P_3 P_2 P_1 P_0$$

- Überlegung: 4b Block erzeugt Übertrag wenn
  - … Spalte 3 einen Übertrag erzeugt (G_3=1) oder
  - … Spalte 3 einen Übertrag weiterleitet (P_3=1), der vorher erzeugt wurde

$$G_{3:0} = G_3 + P_3 [G_2 + P_2 (G_1 + P_1 G_0)]$$

- Damit ist der Übertrag durch einen *i:j* Bit breiten Block C_i

$$C_i = G_{i:j} + P_{i:j} C_{j-1}$$

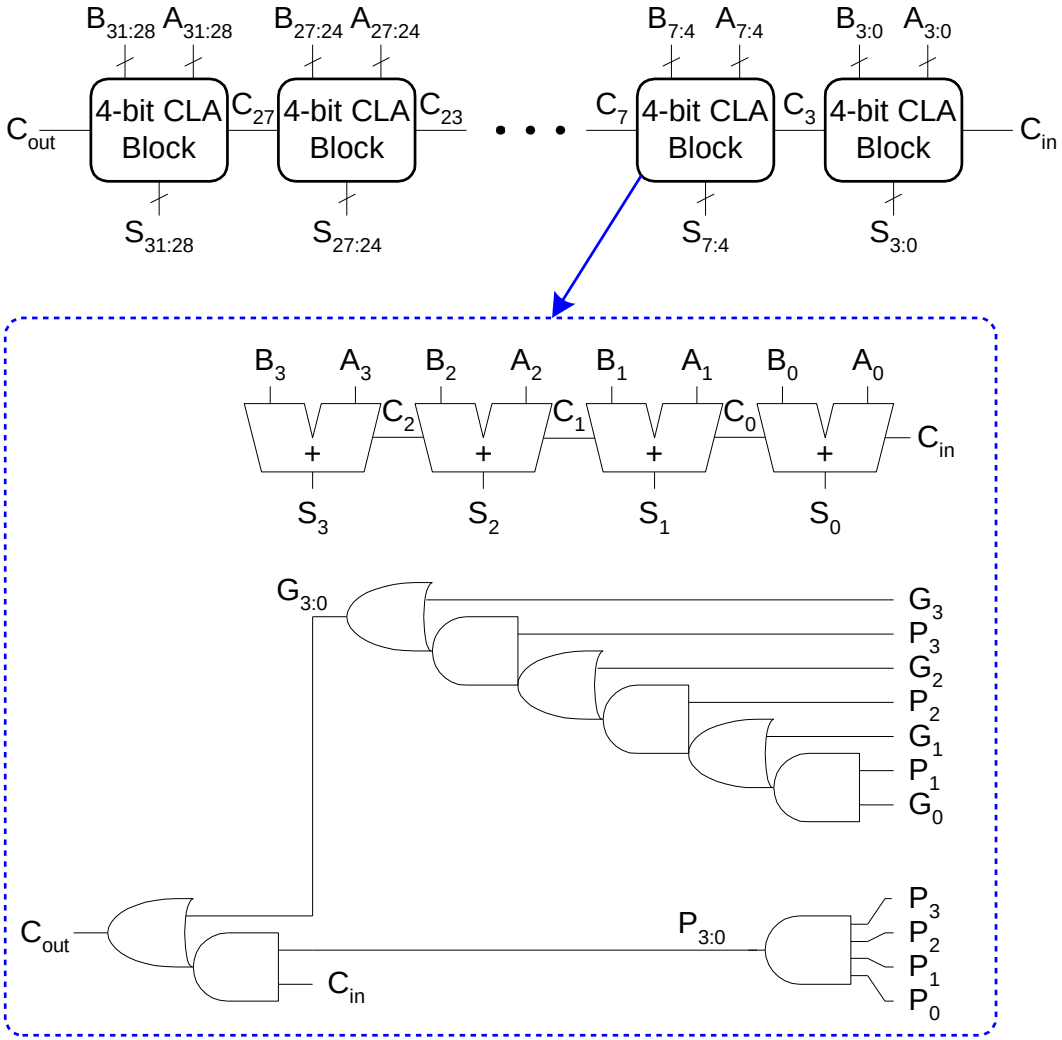
Embedded Systems & Applications

Addition im Carry-Lookahead-Verfahren

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schritt 1: Berechne G und P-Signale für einzelne Spalten (Einzelbits)
- Schritt 2: Berechne G und P Signale für Gruppen von *k* Spalten (*k* Bits)
- **Schritt 3: Leite C_{in} nun nicht einzelbitweise, sondern in *k*-Bit Sprüngen weiter**
  - Jeweils durch einen *k*-bit Propagate/Generate-Block

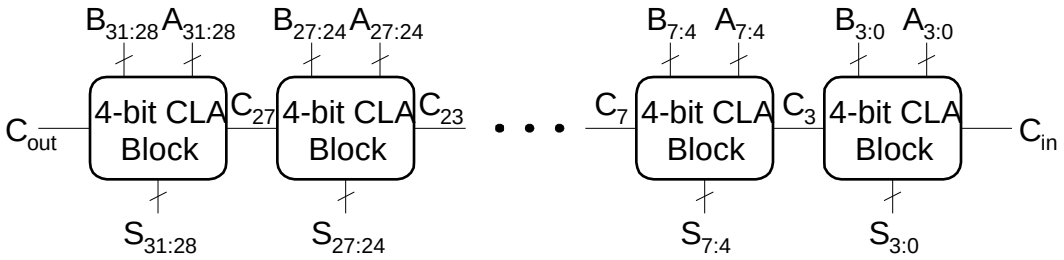
32-bit CLA mit 4b Blöcken

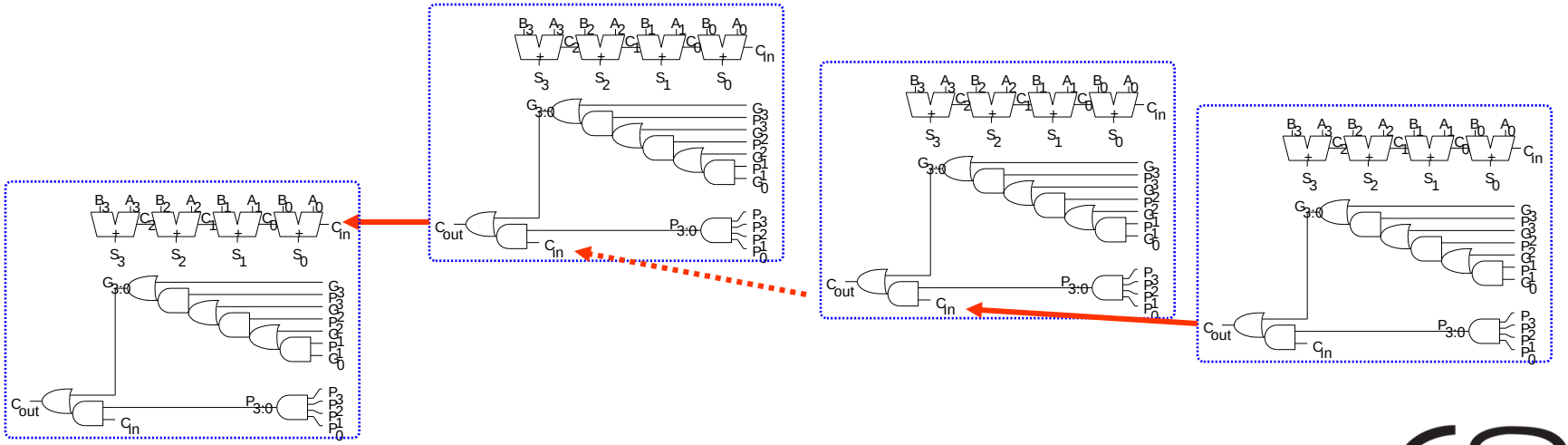


$$G_i = A_i B_i$$

$$P_i = A_i + B_i$$

32-bit CLA mit 4b Blöcken





Carry-Lookahead Addierer

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verzögerung durch *N*-bit carry-lookahead Addierer mit *k*-Bit Blöcken

$$t_{CLA} = t_{pg} + t_{pg\_block} + (N / k - 1)\, t_{AND\_OR} + k\, t_{FA}$$

wobei

- t_{pg} : Verzögerung P, G Berechnung für eine Spalte (ganz rechts)
- t_{pg_block}: Verzögerung P, G Berechnung für einen Block (rechts)
- t_{AND_OR}: Verzögerung durch AND/OR je *k*-Bit CLA Block (“Weiche”)
- $k\, t_{FA}$: Verzögerung zur Berechnung der k höchstwertigen Summenbits

- Für N > 16 ist ein CLA oftmals schneller als ein Ripple-Carry-Addierer
- Aber: Verzögerung hängt immer noch von N ab
  - Im wesentlichen linear

Embedded Systems & Applications

Präfix-Addierer

TECHNISCHE UNIVERSITÄT DARMSTADT

- Führt Ideen des CLA weiter
- Berechnet den Übertrag C_{i-1} in jede Spalte i so schnell wie möglich
- Bestimmt damit die Summe jeder Spalte

$$S_i = (A_i \oplus B_i) \oplus C_{i-1}$$

- Vorgehen zur schnellen Berechnung aller C_i
  - Berechne P und G für größer werdende Blöcke
    - 1b, 2b, 4b, 8b, …
    - Bis die Eingangsüberträge für alle Spalten bereitstehen

Präfix-Addierer

TECHNISCHE UNIVERSITÄT DARMSTADT

- Nun nicht mehr N / k Stufen
- Sondern $\log_2 N$ Stufen
  - Breite der Operanden geht also nur noch logarithmisch in Verzögerung ein
- Allerdings: Sehr viel Hardware erforderlich!

Präfix-Addierer

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ein Übertrag wird entweder
  - … in einer Spalte *i* generiert
  - … oder aus einer Vorgängerspalte *i*-1 propagiert
- Definition: Eingangsübertrag C_{in} in den ganzen Addierer kommt aus Spalte -1

$$G_{-1} = C_{in}, P_{-1} = 1$$

- Eingangsübertrag in eine Spalte *i* ist Ausgangsübertrag C_{i-1} der Spalte *i*-1

$$C_{i-1} = G_{i-1:-1}$$

$G_{i-1:-1}$ ist das Generate-Signal von Spalte -1 bis Spalte *i-1*

Interpretation: Ein Ausgangsübertrag aus Spalte *i*-1 entsteht

- … wenn der Block *i*-1:-1 selbst Übertrag in *i*-1 generiert oder aus *i*-2,3, … weiterleitet

Präfix-Addierer

TECHNISCHE UNIVERSITÄT DARMSTADT

- Damit Summenformel für Spalte *i* umschreibbar zu

$$S_i = (A_i \oplus B_i) \oplus G_{i\text{-}1:\text{-}1}$$

- **Deshalb nun Ziel der Hardware-Realisierung:**
  - Bestimme so schnell wie möglich:

$$G_{0:-1},\ G_{1:-1},\ G_{2:-1},\ G_{3:-1},\ G_{4:-1},\ G_{5:-1},\ \ldots$$

  - Sogenannte Präfixe

Präfix-Addierer

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Berechnung von P und G für variabel großen Block**
  - Höchstwertiges Bit: *i*
  - Niederwertiges Bit: *j*
  - Unterteilt in zwei Teilblöcke (*i*:*k*) und (*k*-1:*j*)

- Für einen Block *i*:*j*

$$G_{i:j} = G_{i:k} + P_{i:k}\, G_{k-1:j}$$

$$P_{i:j} = P_{i:k} P_{k-1:j}$$

Embedded Systems & Applications

Präfix-Addierer

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Berechnung von P und G für variabel großen Block**
  - Höchstwertiges Bit: *i*
  - Niederwertiges Bit: *j*
  - Unterteilt in zwei Teilblöcke (*i*:*k*) und (*k*-1:*j*)

- Für einen Block *i*:*j*

$$G_{i:j} = G_{i:k} + P_{i:k}\, G_{k-1:j}$$

$$P_{i:j} = P_{i:k} P_{k-1:j}$$

- **Bedeutung**
  - Ein Block erzeugt einen Ausgabeübertrag, falls
    - … in seinem oberen Teil (*i*:*k*) ein Übertrag erzeugt wird oder
    - … der obere Teil einen Übertrag weiterleitet, der im unteren Teil (*k*-1:*j*) erzeugt wurde
  - Ein Block leitet einen Eingabeübertrag als Ausgabeübertrag weiter, falls
    - Sowohl der untere als auch der obere Teil den Übertrag weiterleiten

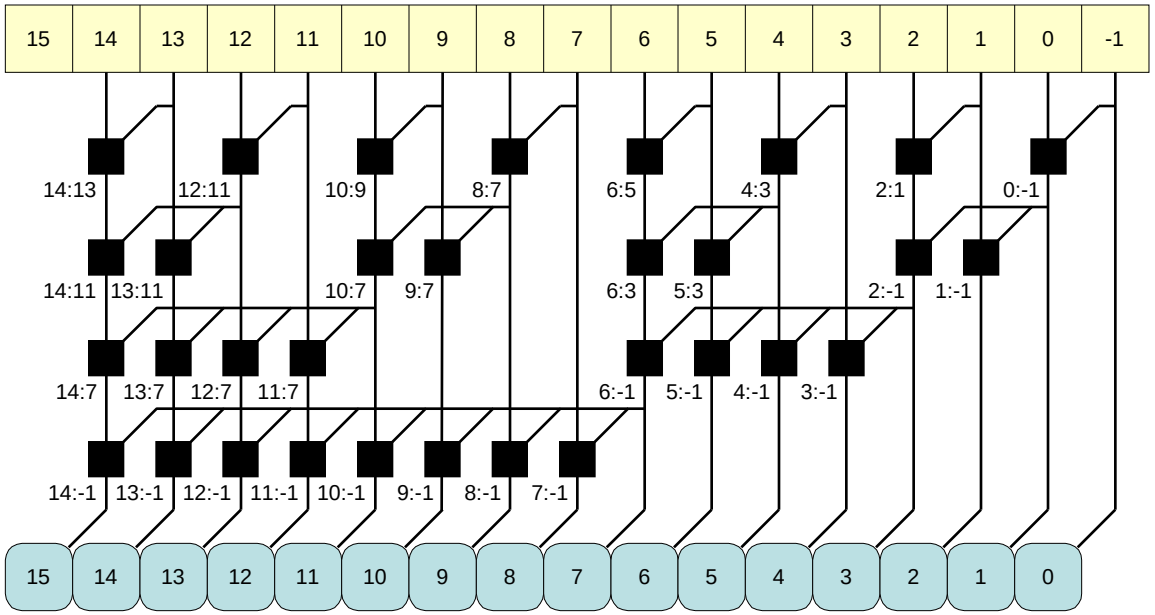
Embedded Systems & Applications

Aufbau eines Präfix-Addierers

TECHNISCHE UNIVERSITÄT DARMSTADT

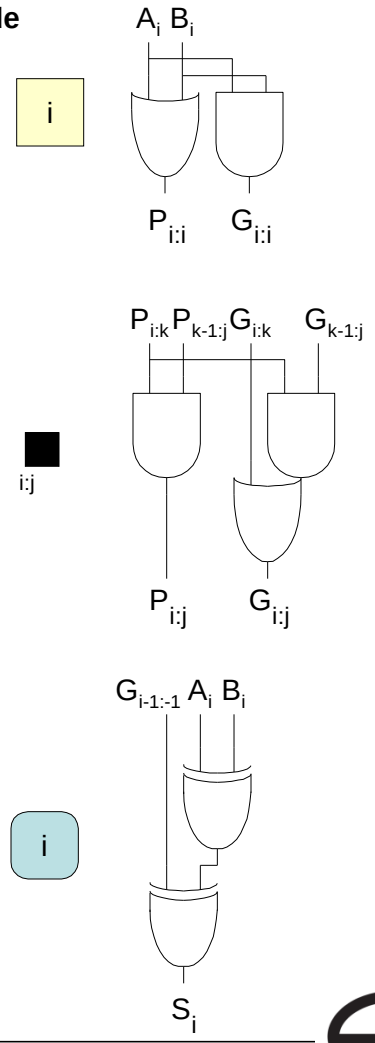
$$G_{i:j} = G_{i:k} + P_{i:k}\, G_{k-1:j}$$

$$P_{i:j} = P_{i:k} P_{k-1:j}$$



$$S_i = (A_i \oplus B_i) \oplus G_{i-1:-1}$$

Legende



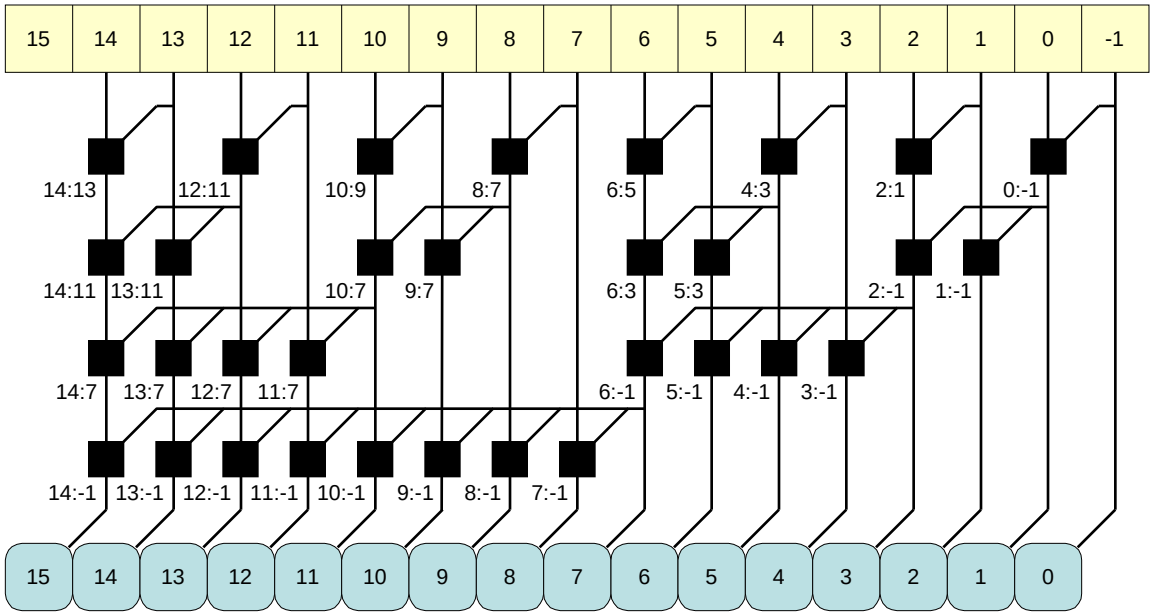
Embedded Systems & Applications

Aufbau eines Präfix-Addierers

TECHNISCHE UNIVERSITÄT DARMSTADT

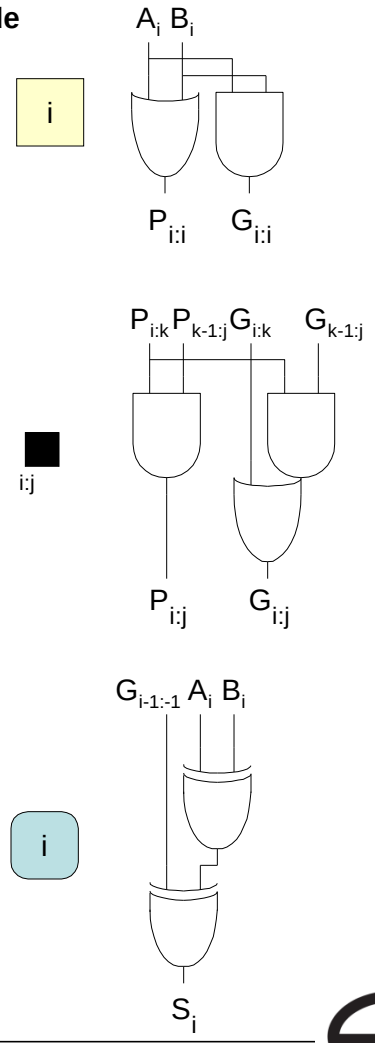
$$G_{i:j} = G_{i:k} + P_{i:k}\, G_{k-1:j}$$

$$P_{i:j} = P_{i:k} P_{k-1:j}$$



$$S_i = (A_i \oplus B_i) \oplus G_{i-1:-1}$$

Legende



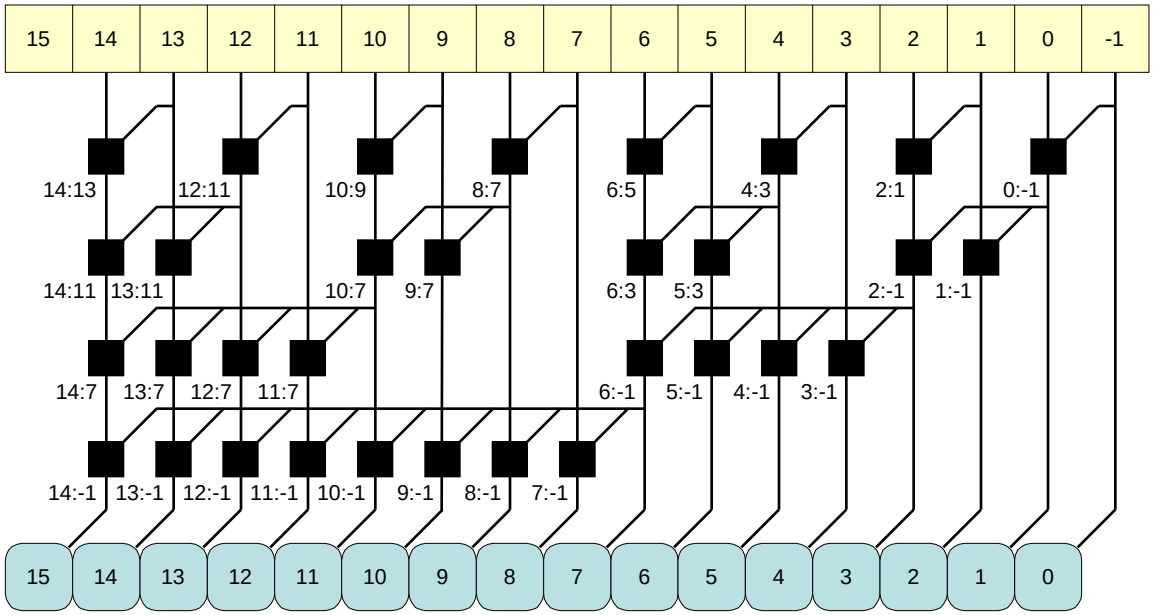
Embedded Systems & Applications

Aufbau eines Präfix-Addierers

TECHNISCHE UNIVERSITÄT DARMSTADT

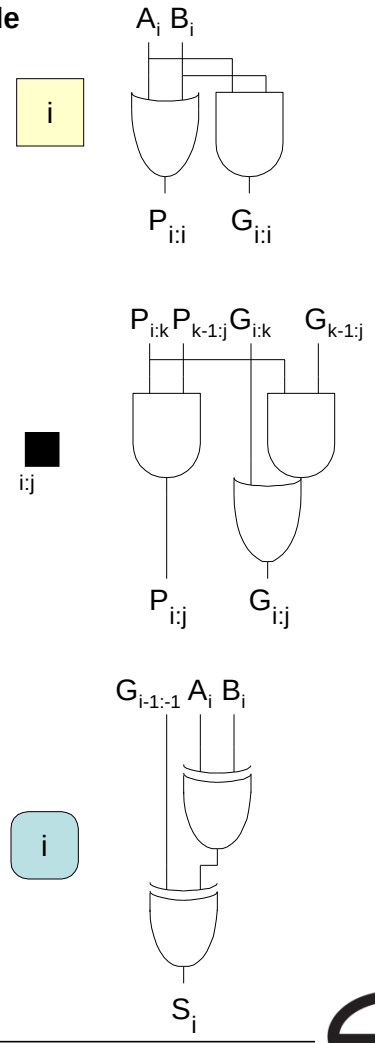
$$G_{i:j} = G_{i:k} + P_{i:k}\, G_{k-1:j}$$

$$P_{i:j} = P_{i:k} P_{k-1:j}$$



$$S_i = (A_i \oplus B_i) \oplus G_{i-1:-1}$$

Legende



Embedded Systems & Applications

Verzögerung durch Präfix-Addierer

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Verzögerung durch einen *N*-bit Präfix-Addierer**

$$t_{PA} = t_{pg} + (\log_2 N)\, t_{pg\_prefix} + t_{XOR}$$

wobei

- t_{pg}: Verzögerung durch P, G-Berechnung für Spalte *i* (ein AND bzw. OR-Gatter)
- t_{pg_prefix}: Verzögerung durch eine Präfix-Stufe (AND-OR Gatter)
- t_{XOR} : Verzögerung durch letztes XOR der Summenberechnung

Vergleich von Addiererverzögerungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Szenario: 32b Addition mit, Ripple-Carry, Carry-Lookahead (4-bit Blöcke), Präfix-Addierer
- Verzögerungen von Komponenten
  - Volladdierer t_{FA} = 300ps
  - Zwei-Eingangs Gatter $t_{AND} = t_{OR} = t_{XOR}$ = 100ps

$$t_{ripple} = N\, t_{FA} = 32\ (300\ ps)$$
$$=$$

$$t_{CLA} = t_{pg} + t_{pg\_block} + (N / k - 1)\, t_{AND\_OR} + k\, t_{FA}$$
$$=$$

$$t_{PA} = t_{pg} + (\log_2 N)\, t_{pg\_prefix} + t_{XOR}$$
$$=$$

Embedded Systems & Applications

Vergleich von Addiererverzögerungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Szenario: 32b Addition mit, Ripple-Carry, Carry-Lookahead (4-bit Blöcke), Präfix-Addierer
- Verzögerungen von Komponenten
  - Volladdierer t_{FA} = 300ps
  - Zwei-Eingangs Gatter $t_{AND} = t_{OR} = t_{XOR}$ = 100ps

$$t_{ripple} = N\, t_{FA} = 32\ (300\ ps)$$
$$= 9{,}6\ ns$$

$$t_{CLA} = t_{pg} + t_{pg\_block} + (N / k - 1)\, t_{AND\_OR} + k\, t_{FA}$$
$$= [100 + 600 + (7)\ 200 + 4\ (300)]\ ps$$
$$= 3{,}3\ ns$$

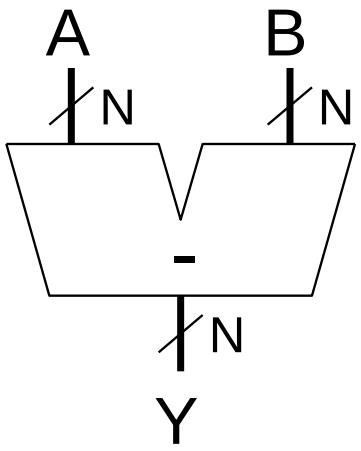
$$t_{PA} = t_{pg} + (\log_2 N)\ t_{pg\_prefix} + t_{XOR}$$
$$= [100 + (\log_2 32)\ 200 + 100]\ ps$$
$$= 1{,}2\ ns$$

Embedded Systems & Applications

Subtrahierer

TECHNISCHE UNIVERSITÄT DARMSTADT

Symbol



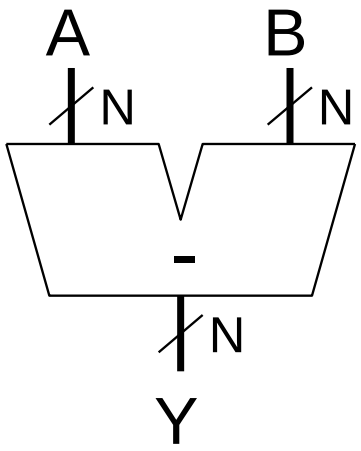
Implementierung

Embedded Systems & Applications

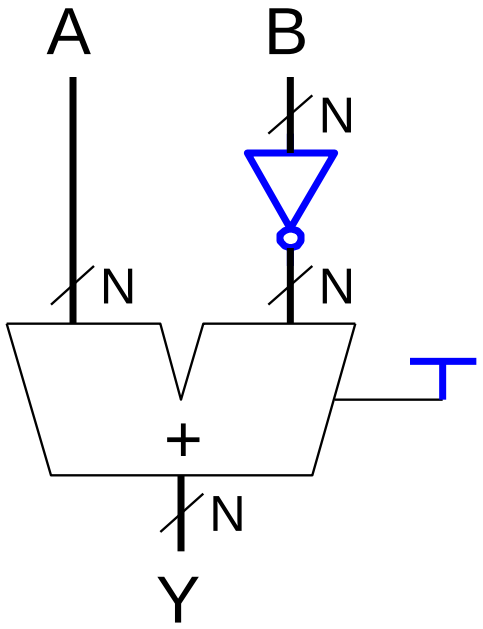
Subtrahierer

TECHNISCHE UNIVERSITÄT DARMSTADT

Symbol



Implementierung

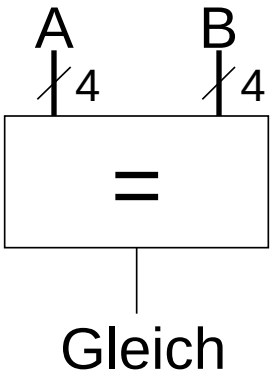


Embedded Systems & Applications

Vergleicher: Gleichheit

TECHNISCHE UNIVERSITÄT DARMSTADT

Symbol



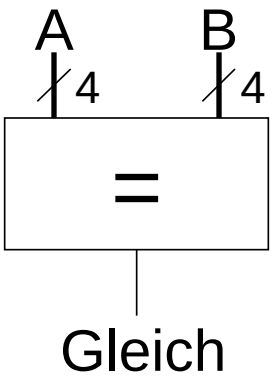
Implementierung

Embedded Systems & Applications

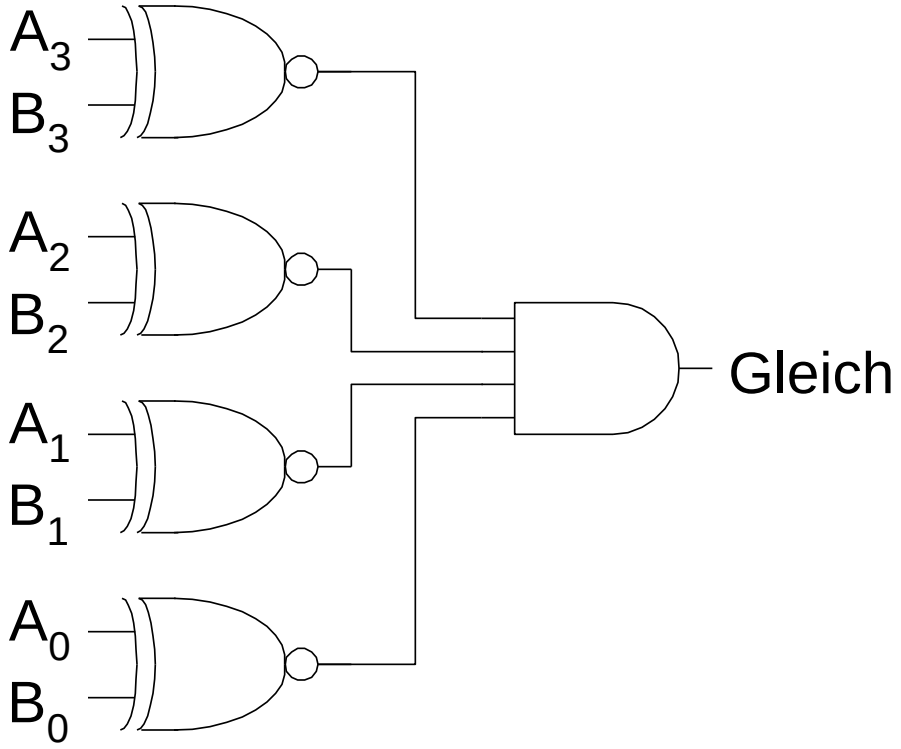
Vergleicher: Gleichheit

TECHNISCHE UNIVERSITÄT DARMSTADT

Symbol



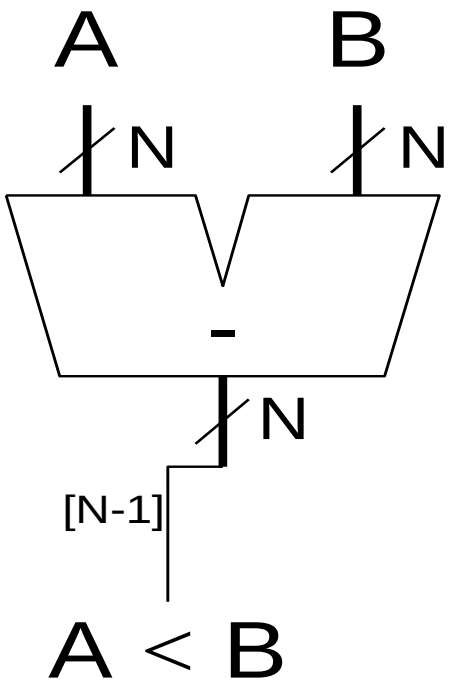
Implementierung



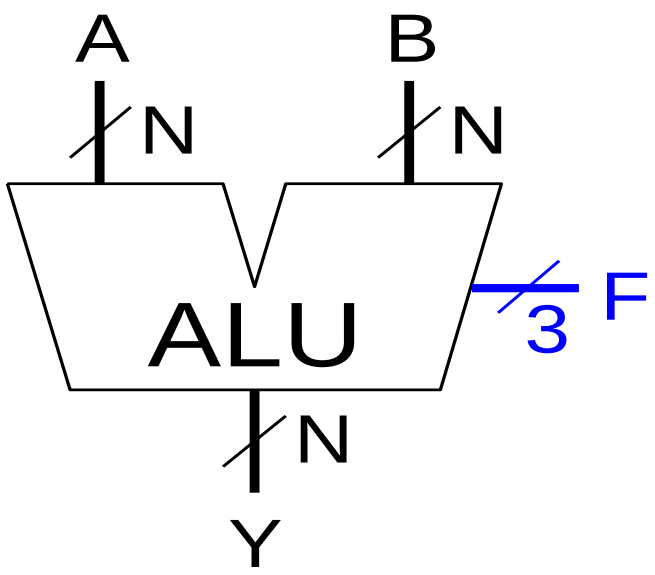
Embedded Systems & Applications

Vergleicher: Kleiner-Als

- Für N-Bit Zweierkomplementzahlen

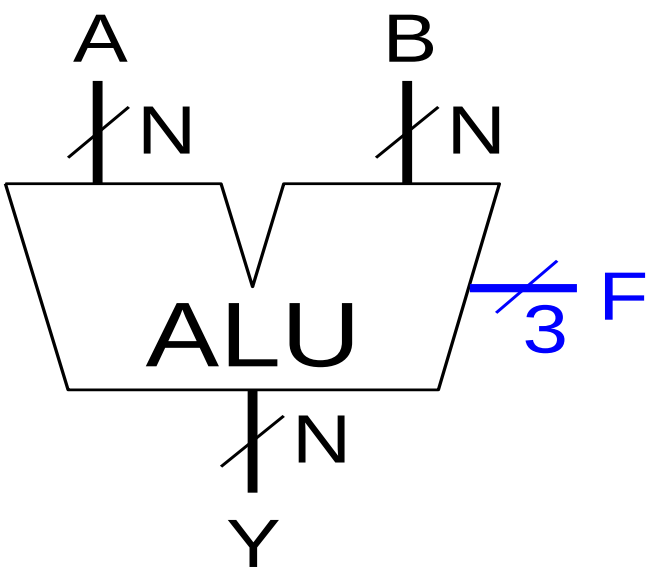


Arithmetisch-logische Einheit (*arithmetic logic unit, ALU*)



- **Funktionen:**
  - UND
  - ODER
  - Addition/Subtraktion

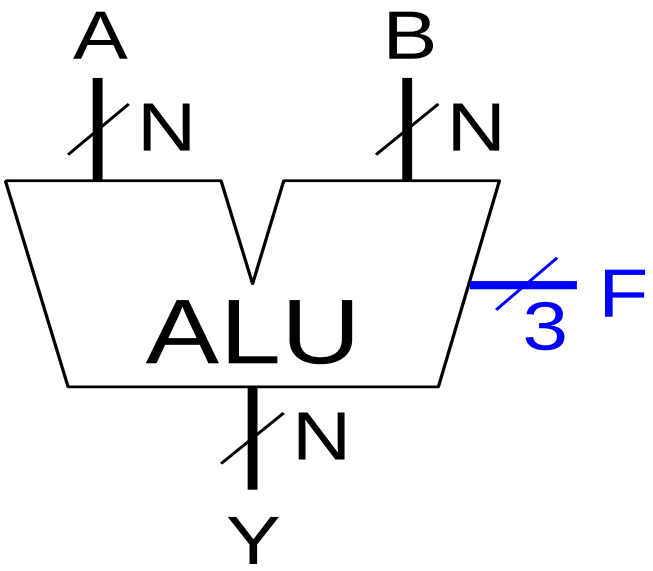
Arithmetisch-logische Einheit (*arithmetic logic unit, ALU*)



- **Funktionen:**
  - UND
  - ODER
  - Addition/Subtraktion

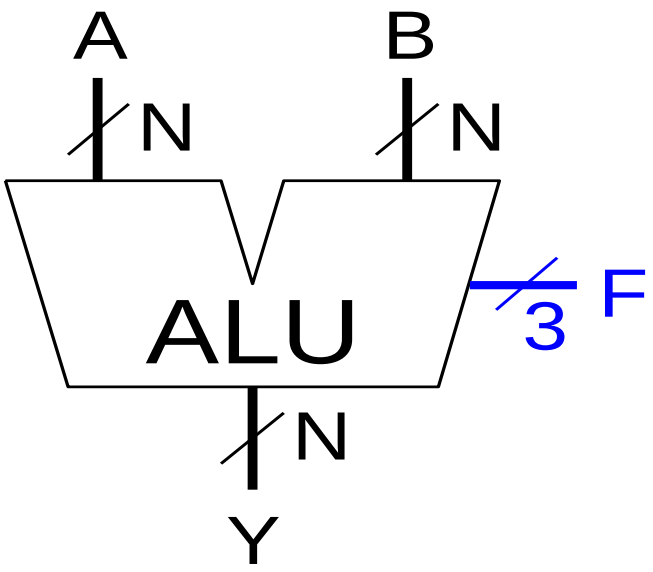
$F_{2:0}$	Funktion
000	A & B
001	A | B
010	A + B

Arithmetisch-logische Einheit (*arithmetic logic unit, ALU*)



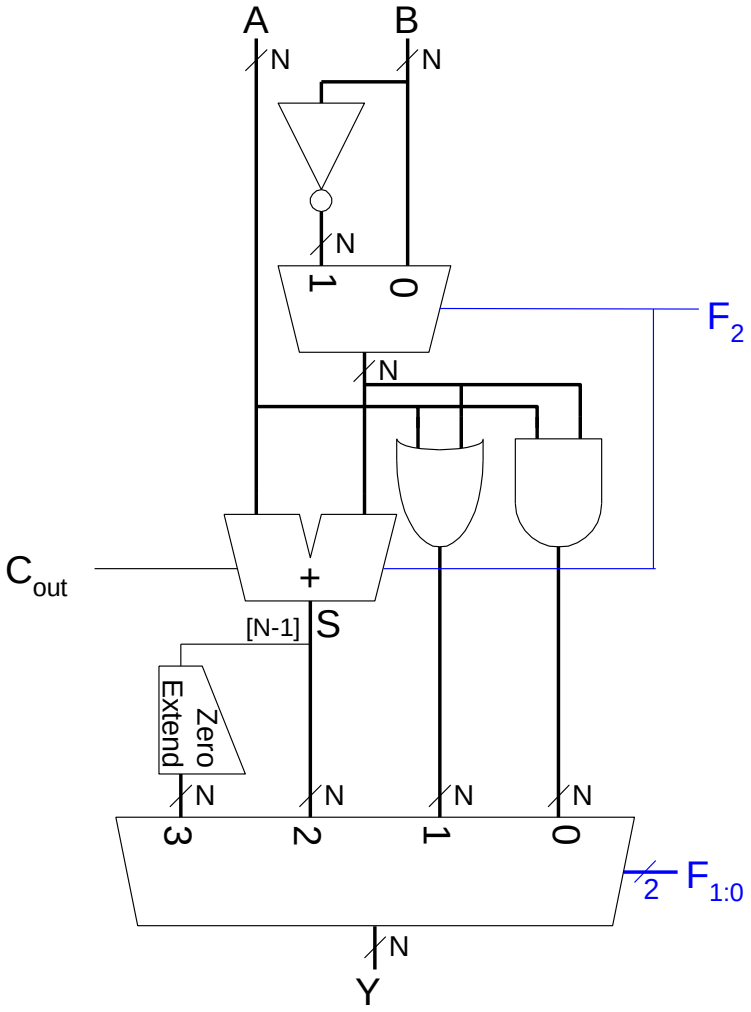
$F_{2:0}$	Funktion
000	A & B
001	A | B
010	A + B

Arithmetisch-logische Einheit (*arithmetic logic unit, ALU*)



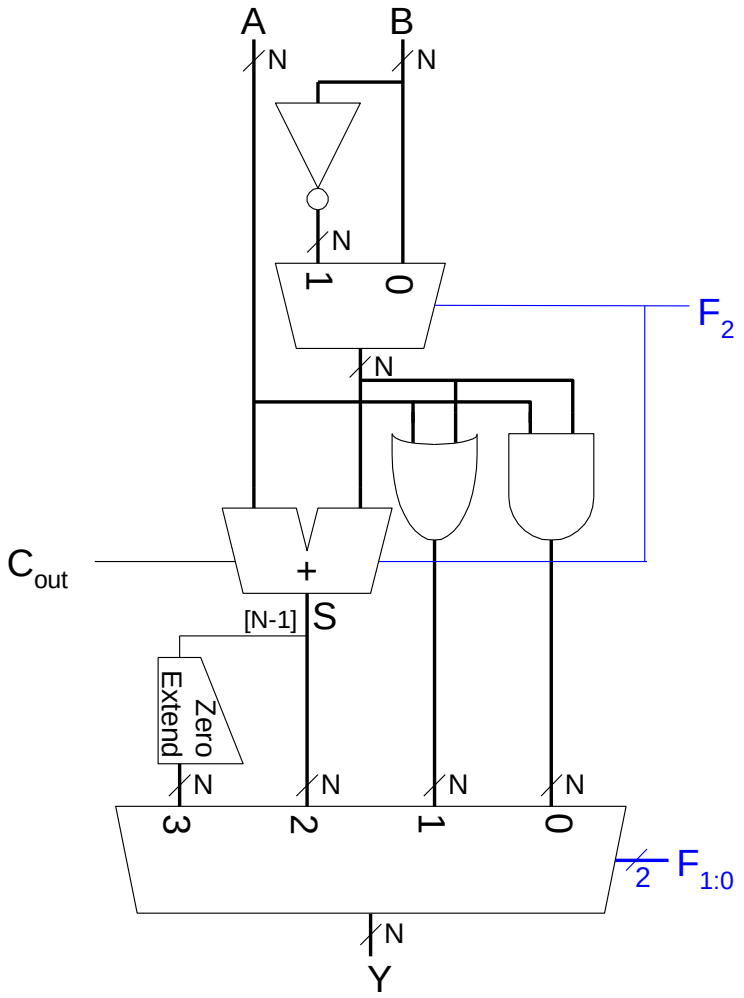
$F_{2:0}$	Funktion
000	A & B
001	A | B
010	A + B
011	Nicht verwendet
100	A & ~B
101	A | ~B
110	A - B
111	SLT

Entwurf einer ALU



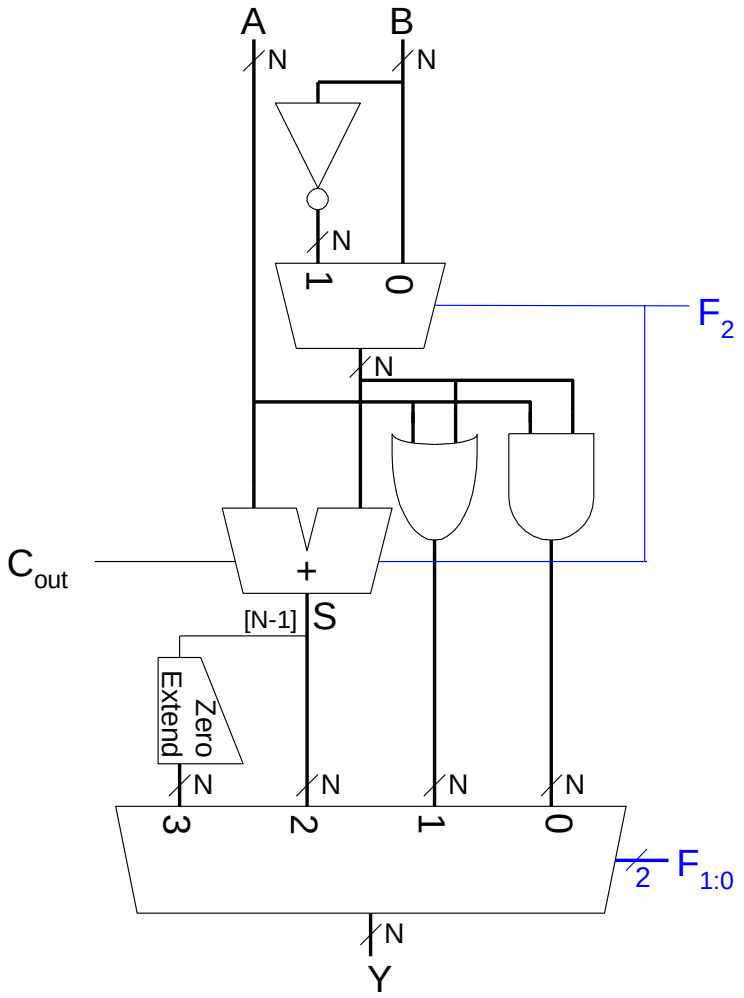
$F_{2:0}$	Funktion
000	A & B
001	A | B
010	A + B
011	Nicht verwendet
100	A & ~B
101	A | ~B
110	A - B
111	SLT

Beispiel: AND



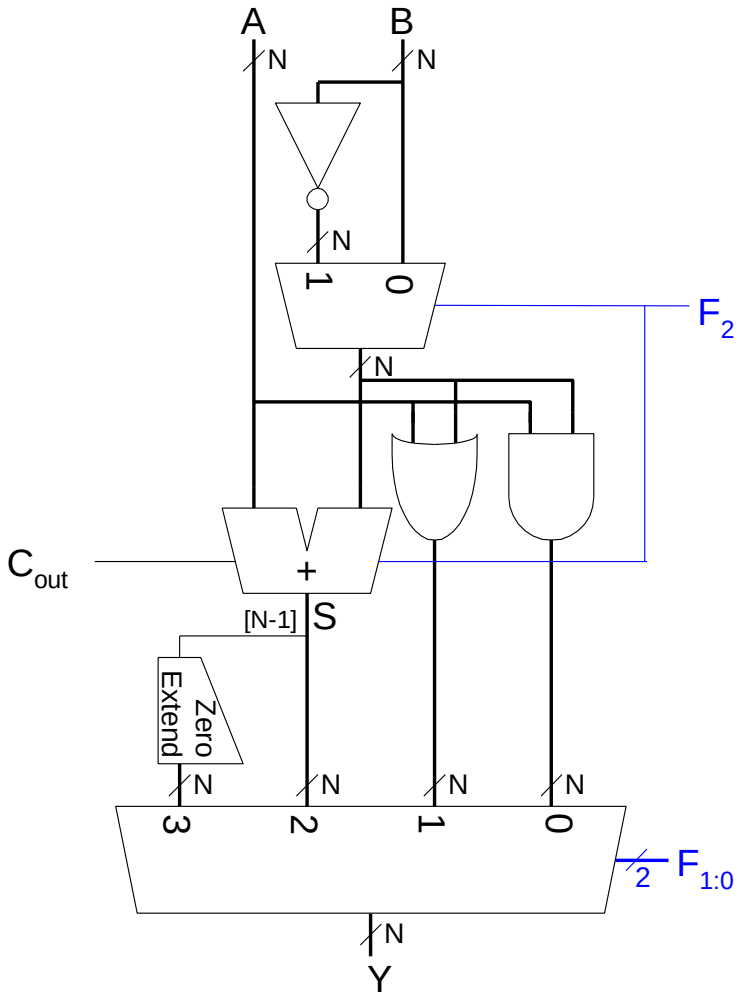
- **Konfiguriere 32b ALU für AND-Berechnung**
  - Annahme: *A* = 32‘b11001, *B* = 32‘b1101

Beispiel: AND



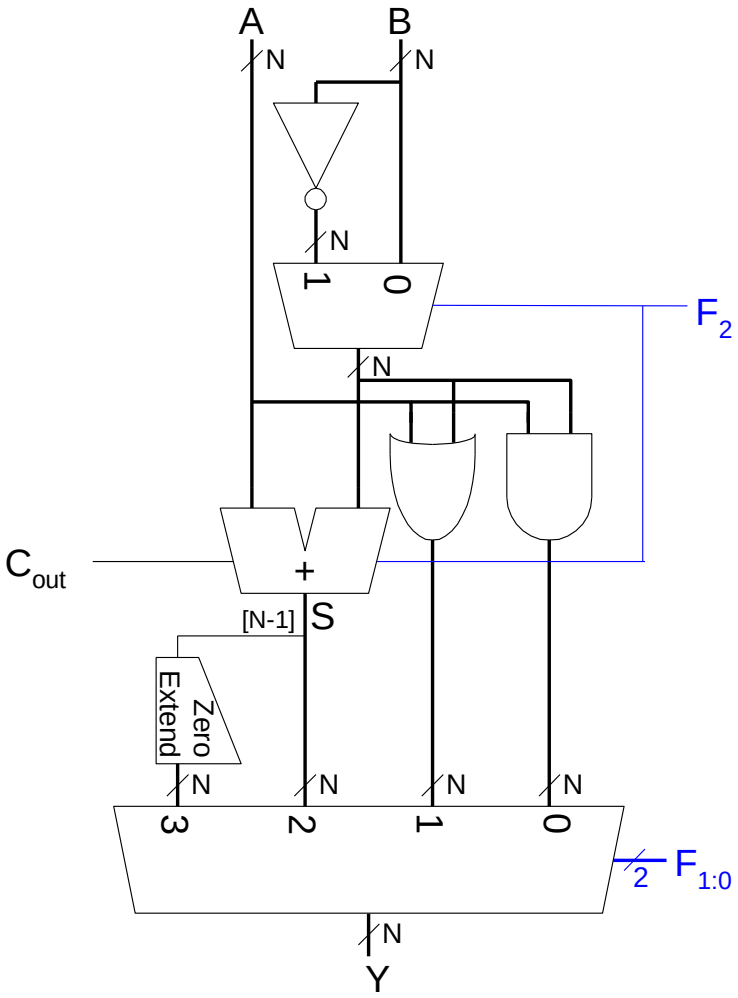
- **Konfiguriere 32b ALU für AND-Berechnung**
  - Annahme: *A* = 32‘b11001, *B* = 32‘b1101
  - **Erwartete Ausgabe**
    - Y = A & B = 32’b1001
  - **Steuereingang** für AND:
    - $F_{2:0}$ = 3’b000
  - ***F*$_{1:0}$ = 2’b00** wählt *Y* = *UND Gatter Ausgänge*

Beispiel: Set Less Than (SLT)



- Konfiguriere 32b ALU für SLT-Berechnung
  - Annahme: *A* = 25, *B* = 32

Beispiel: Set Less Than (SLT)



- Konfiguriere 32b ALU für SLT-Berechnung
  - **Annahme: *A* = 25, *B* = 32**
  - **Erwartete Ausgabe:**

    A < B, also Y = 32'b1
  - **Steuereingang für SLT:** $F_{2:0}$ = 3'b111
  - F_2 = 1'b1 konfiguriert Addierer als **Subtrahierer**
    - S = 25 – 32 = -7
    - Im Zweierkomplement

      -7 = 32'h0xfffffff9 ➔msb S_{31} = 1
  - **$F_{1:0}$ = 2'b11** wählt $Y = S_{31}$ als Ausgabe
  - $Y = S_{31}$ (zero extended) = 32'h00000001

Schiebeoperationen (*shifter*)

TECHNISCHE UNIVERSITÄT DARMSTADT

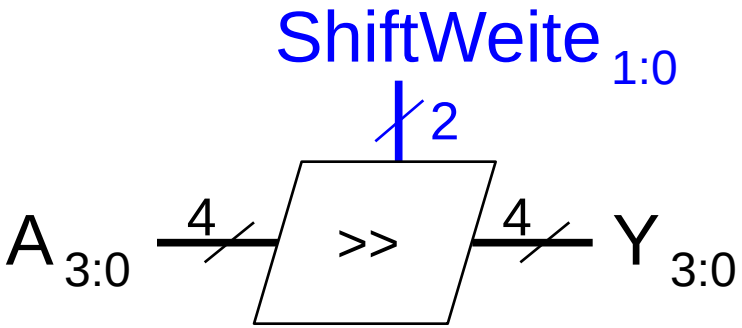
- **Logisches Schieben:** leere Stellen mit 0 aufgefüllt
  - Beispiel: 11001 >> 2 =
  - Beispiel: 11001 << 2 =
- **Arithmetisches Schieben:** wie logisches Schieben. Verwende aber beim Rechtsschieben alten Wert des msb zum Auffüllen leerer Stellen
  - Beispiel: 11001 >>> 2 =
  - Beispiel: 11001 <<< 2 =
- **Rotierer:** rotiert Bits im Kreis, herausgeschobene Bits tauchen am anderen Ende wieder auf
  - Beispiel : 11001 ROR 2 =
  - Beispiel : 11001 ROL 2 =

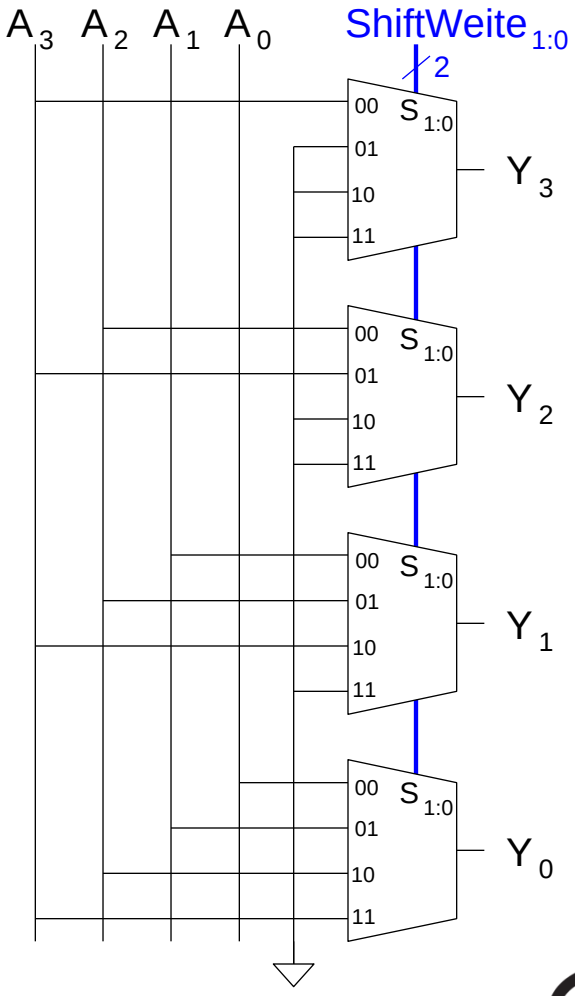
Schiebeoperationen (*shifter*)

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Logisches Schieben:** leere Stellen mit 0 aufgefüllt
  - Beispiel: 11001 >> 2 = 00110
  - Beispiel: 11001 << 2 = 00100
- **Arithmetisches Schieben:** wie logisches Schieben. Verwende aber beim Rechtsschieben alten Wert des msb zum Auffüllen leerer Stellen
  - Beispiel: 11001 >>> 2 = **11**110
  - Beispiel: 11001 <<< 2 = 00100
- **Rotierer:** rotiert Bits im Kreis, herausgeschobene Bits tauchen am anderen Ende wieder auf
  - Beispiel : 11001 ROR 2 = 01110
  - Beispiel : 11001 ROL 2 = 00111

Aufbau von Shiftern





Shifter als Multiplizierer und Dividierer

TECHNISCHE UNIVERSITÄT DARMSTADT

- Logisches Schieben um *N* Stellen nach links **multipliziert** den Zahlenwert mit $\mathbf{2^N}$
  - Beispiel : 00001 << **3** = 01**000** ($1 \times 2^3 = 8$)
  - Beispiel : 11101 << **2** = 101**00** ($-3 \times 2^2 = -12$)

- Arithmetisches Schieben um *N* Stellen nach rechts **dividiert** den Zahlenwert durch $\mathbf{2^N}$
  - Beispiel : 010000 >>> **4** = **0000**01 ($16 \div 2^4 = 1$)
  - Beispiel : 100000 >>> **2** = **11**1000 ($-32 \div 2^2 = -8$)

Embedded Systems & Applications

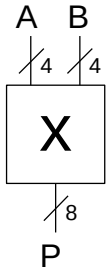
Multiplizierer

- Schrittweise Multiplikation in Dezimal- und Binärdarstellung:
  - Multiplizieren des Multiplikanden mit einzelner Stelle des Multiplikators
    - Berechnet ein Teilprodukt (auch partielles Produkt genannt)
  - Entsprechend der Wertigkeit der aktuellen Multiplikatorstelle nach links verschobene partielle Produkte werden aufaddiert

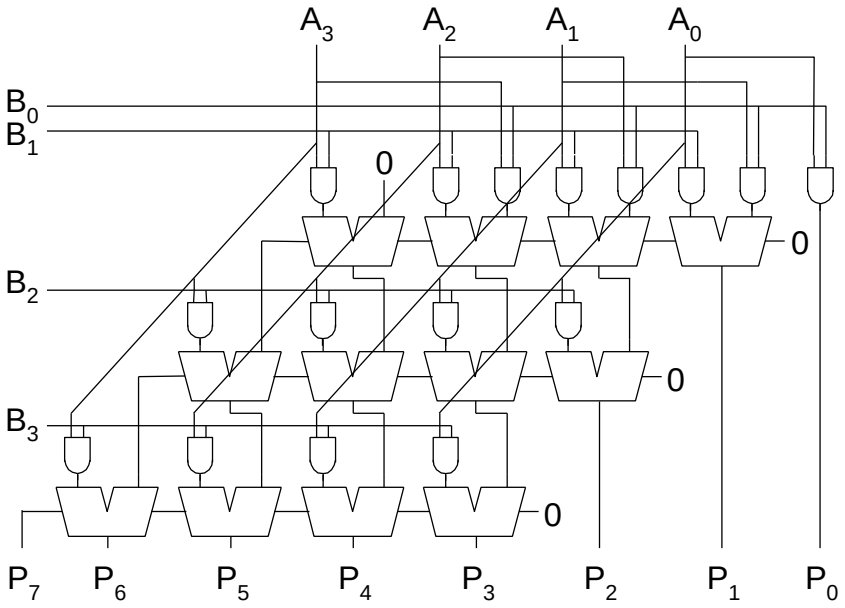
Dezimal		**Binär**
230	Multiplikand	0101
x 42	Multiplikator	x 0111
460	partielle Produkte	0101
+ 920		0101
		0101
		+ 0000
9660	Ergebnis	0100011

230 x 42 = 9660 5 x 7 = 35

4 x 4 Multiplizierer



$$
\begin{array}{rcccccccc}
 & & & A_3 & A_2 & A_1 & A_0 & & \\
\times & & & B_3 & B_2 & B_1 & B_0 & & \\
\hline
 & & & A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 & & \\
 & & A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 & & & \\
 & A_3B_2 & A_2B_2 & A_1B_2 & A_0B_2 & & & & \\
+ & A_3B_3 & A_2B_3 & A_1B_3 & A_0B_3 & & & & \\
\hline
P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}
$$



Multiplikation von *k*-bit Zahlen hat 2*k*-bit breites Produkt

Division

TECHNISCHE UNIVERSITÄT DARMSTADT

- Leidlich einfach, dann aber sehr langsam
- Sehr kompliziert, dann wenigstens etwas schneller
  - Aber immer noch deutlich langsamer als z.B. Multiplikation
- Für Einführungsveranstaltung eher ungeeignet
  - Beschreibung im Buch auch ziemlich schlecht …
- Hier nur aus dem Orbit gestreift
  - Auszug aus

Behrooz Parhami
Computer Arithmetic: Algorithms and Hardware Designs
Oxford U. Press, 2nd ed., 2010, ISBN 978-0-19-532848-6

Dividierer Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

$$A/B = Q + R/B$$

Dezimal Beispiel: $2584/15 = 172\ R4$

Embedded Systems & Applications

Dividierer Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

A/B = Q + R/B

Dezimal Beispiel: 2584/15 = 172 R4

Langschrift:

```
   172 R4
15|2584
  -15
   108
  -105
     34
    -30
      4
```


Embedded Systems & Applications

Dividierer Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

A/B = Q + R/B

Dezimal Beispiel: 2584/15 = 172 R4

Langschrift:

```
   172  R4
15|2584
  -15
   108
  -105
     34
    -30
      4
```

Andersherum:

```
0002
- 15
 -13
```
$0_{3}\ \__{2}\ \__{1}\ \__{0}$

```
0025
- 15
  10
```
$0_{3}\ 1_{2}\ \__{1}\ \__{0}$

```
0108
- 105
    3
```
$0_{3}\ 1_{2}\ 7_{1}\ \__{0}$

```
0034
- 30
   4
```
$0_{3}\ 1_{2}\ 7_{1}\ 2_{0}$

Embedded Systems & Applications

Dividierer Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

Dezimal: 2584/15=172 R4

```
0002      0
- 15      3 2 1 0
 -13
```

```
0025      0 1
- 15      3 2 1 0
  10
```

```
0108      0 1 7
-105      3 2 1 0
   3
```

```
0034      0 1 7 2
- 30      3 2 1 0
   4
```

Binär: 1101/10 = 0110 R1

```
0001      0
-0010     3 2 1 0
 1111
```

```
0011      0 1
-0010     3 2 1 0
 0001
```

```
0010      0 1 1
-0010     3 2 1 0
 0000
```

```
0001      0 1 1 0   R1
-0010     3 2 1 0
 1111
```


Embedded Systems & Applications

Dividierer Algorithmus

TECHNISCHE UNIVERSITÄT DARMSTADT

A/B = Q + R/B

R′ = 0

for *i* = N-1 to 0

  R = {R′ << 1, A_i}

  D = R - B

  if D < 0, Q_i= 0; R′ = R

  else Q_i= 1; R′ = D

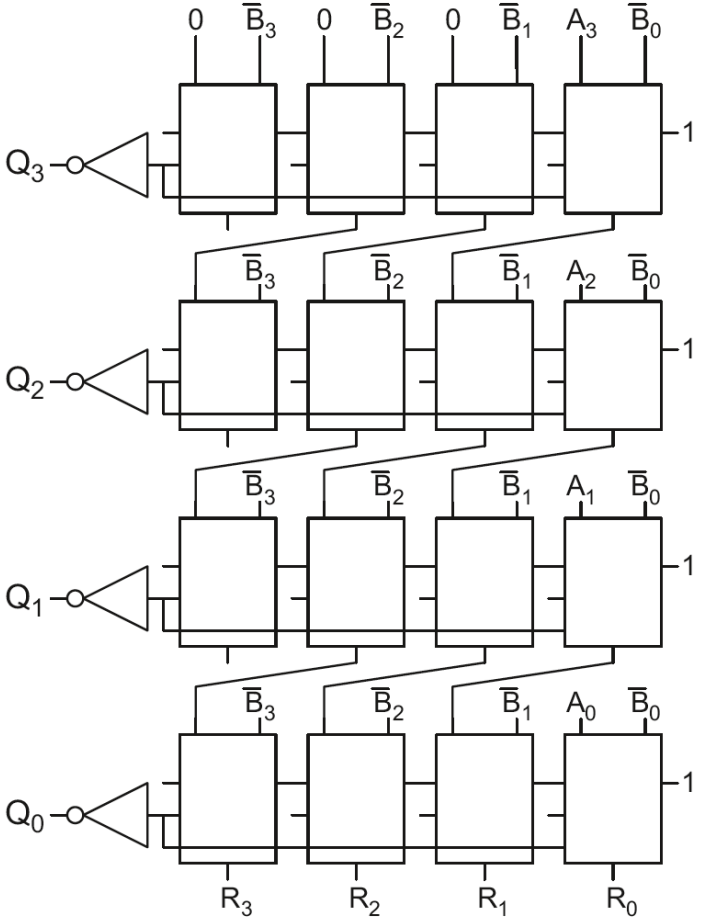
R=R′

Binär: 1101/10 = 0110 R1

```
 0001
-0010      0 _ _ _
 1111      3 2 1 0

 0011
-0010      0 1 _ _
 0001      3 2 1 0

 0010
-0010      0 1 1 _
 0000      3 2 1 0

 0001
-0010      0 1 1 0   R1
 1111      3 2 1 0
```


Embedded Systems & Applications

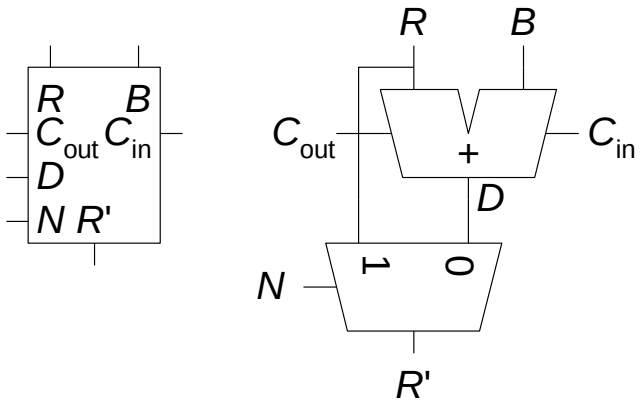
Kombinatorischer 4 x 4 Array-Dividierer: A / B



Jede Zeile rechnet eine Iteration des Algorithmus

Verzögerung proportional zu N^2

Legend



Division: $A/B = Q + R/B$

$R' = 0$

for i = N-1 to 0

  $R = \{R' << 1, A_i\}$

  $D = R - B$

  if $D < 0$, $Q_i=0$, $R'=R$

  else $Q_i=1$, $R'=D$

$R=R'$

Idee: Quotient ziffernweise bestimmen

TECHNISCHE UNIVERSITÄT DARMSTADT

- 2487 / 10 =

- Von höher- zur niederwertigen Stellen
  - 1000er, 100er, 10er, 1er

Notation für Erklärung der Division

z	Dividend	$z_{2k-1}z_{2k-2} \ldots z_3z_2z_1z_0$
d	Divisor	$d_{k-1}d_{k-2} \ldots d_1d_0$
q	Quotient	$q_{k-1}q_{k-2} \ldots q_1q_0$
s	Rest, $z - (d\ q)$	$s_{k-1}s_{k-2} \ldots s_1s_0$

- Dividiere vorzeichenlose Zahlen

	Dividend	***z***	(2k Bit breit)
durch	**Divisor**	***d***	(k Bit breit)

- Ergebnis

Quotient	***q***	(k Bit breit)
Rest	***s***	(k Bit breit)

- Es gilt

$$z = q\ d + s$$

Auftreten von Überläufen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Beispiel k=8:

- Problem: Damit nicht alle Ergebniswerte repräsentierbar
  - Operanden: z = , d =
  - Ergebnis: q =
    s =

Auftreten von Überläufen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Beispiel k=8: 16b Dividend (z), 8b Divisor (d), 8b Quotient (q), 8b Rest (s)

- Problem: Damit nicht alle Ergebniswerte repräsentierbar
  - Operanden: z = 65534, d = 2
  - Ergebnis: q = 32767 **nicht mehr in 8b darstellbar, Überlauf!**,
    s = 0

- Vorgehensweise: Vorher auf darstellbares Ergebnis prüfen
  - Vermeidet Überlauf
  - Fängt auch Division durch Null ab

Abfangen von Überläufen

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Maximalwert für Dividenden**

  $z = q\,d + s$

- **Maximalwert für *q* (*k* Bit breit):**
- **Maximalwert für s (*k* Bit breit):**

Abfangen von Überläufen

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Maximalwert für Dividenden**

  $z = q\,d + s$

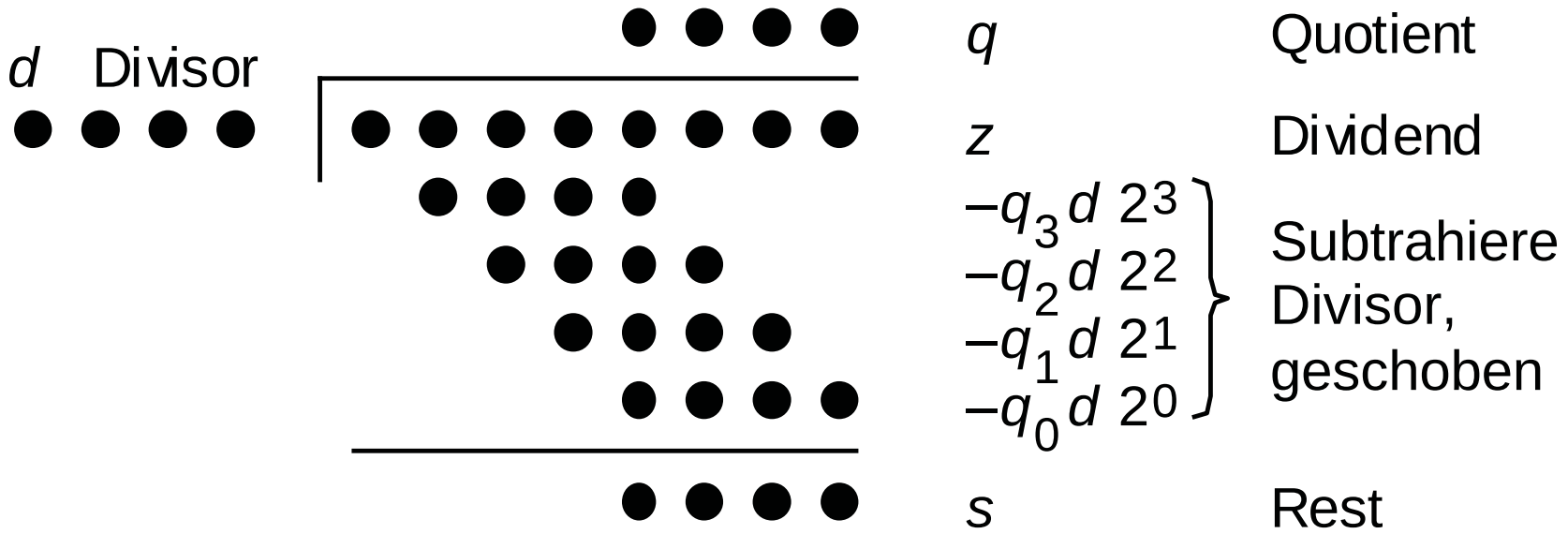
- **Maximalwert für *q* (*k* Bit breit): $2^k - 1$**
- **Maximalwert für s (*k* Bit breit): *d* - 1**
  - $z_{max} = (2^k\text{-}1)\,d + d - 1 = 2^k\,d - 1$
  - $z < z_{max} + 1$ ➔ Wenn $z < 2^k$ d, dann kein Überlauf

Embedded Systems & Applications

Generelle Vorgehensweise für Division

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Hier: Einfaches, aber langsames Verfahren**

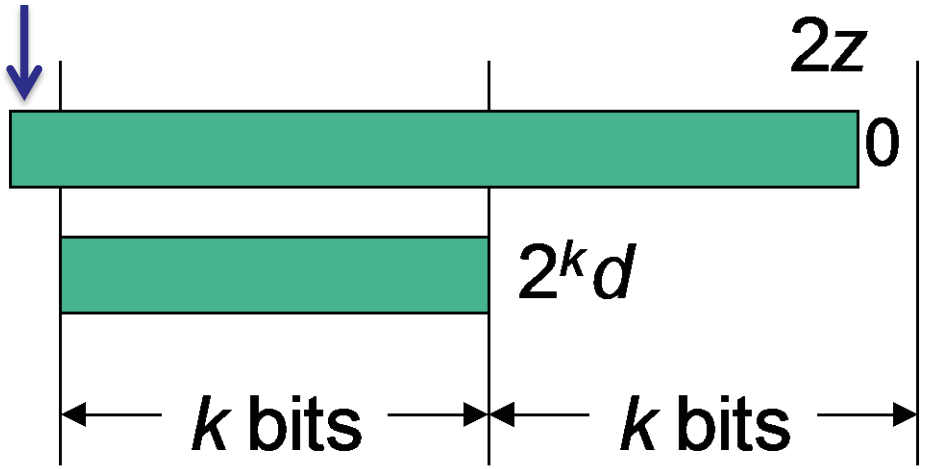


Embedded Systems & Applications

Optimierung

- Schiebe nicht Divisor nach rechts
- … sondern partiellen Rest nach links

msb gesondert behandeln!



Schrittweise nach links geschoben

Bleibt gleich

Beispiel: 117 / 10

Beispiel 8b Dividend, 4b Divisor: 117 / 10

TECHNISCHE UNIVERSITÄT DARMSTADT

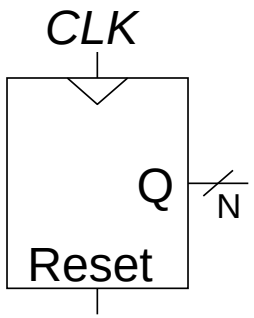
	z	0 1 1 1	0 1 0 1
	$2^4 d$	1 0 1 0	
	==========		
Schritt 0	$s^{(0)}$	0 1 1 1	0 1 0 1
	$2s^{(0)}$	0 1 1 1 0	1 0 1
Schritt 1	$-q_3 2^4 d$	1 0 1 0	**$\{q_3 = 1\}$**
	$s^{(1)}$	0 1 0 0	1 0 1
	$2s^{(1)}$	0 1 0 0 1	0 1
Schritt 2	$-q_2 2^4 d$	0 0 0 0	**$\{q_2 = 0\}$**
	$s^{(2)}$	1 0 0 1	0 1
	$2s^{(2)}$	1 0 0 1 0	1
Schritt 3	$-q_1 2^4 d$	1 0 1 0	**$\{q_1 = 1\}$**
	$s^{(3)}$	1 0 0 0	1
	$2s^{(3)}$	1 0 0 0 1	
Schritt 4	$-q_0 2^4 d$	1 0 1 0	**$\{q_0 = 1\}$**
	$s^{(4)}$	0 1 1 1	
	s		0 1 1 1
	q		1 0 1 1
	==========		

Embedded Systems & Applications

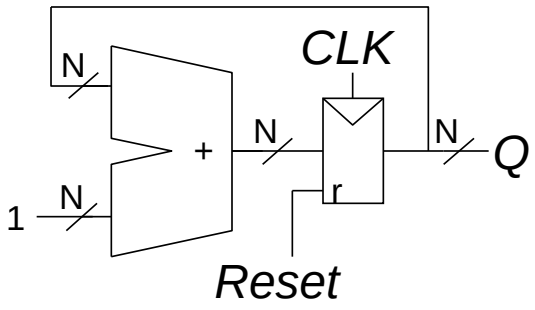
Zähler

- Einfachster Fall: Inkrementieren zu jeder positiven Taktflanke
- Zählen durch einen Zyklus von Werten, Beispiel für 3b Breite
  - 000, 001, 010, 011, 100, 101, 110, 111, 000, 001…
- Beispielanwendungen
  - Digitaluhren
  - Programmzähler: Zeigt auf nächste auszuführende Instruktion

Symbol



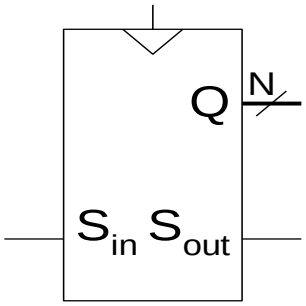
Aufbau



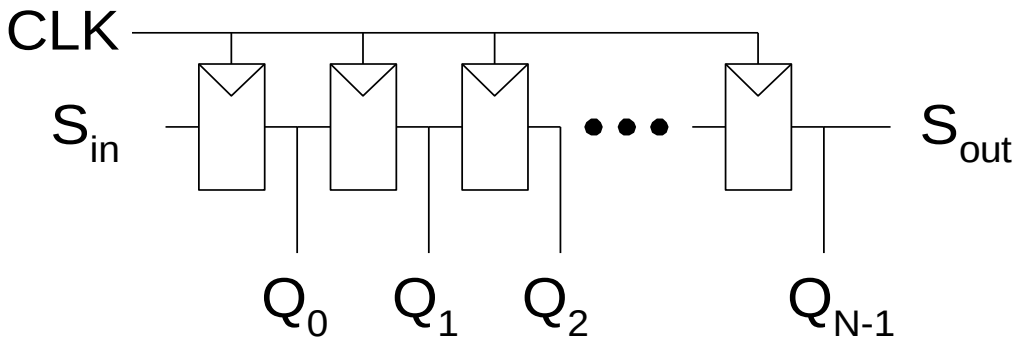
Schieberegister

- Auch: FIFO (*first-in first-out*)
- Schiebe einen neuen Wert jeden Takt ein
- Schiebe einen alten Wert jeden Takt aus
- Kann auch agieren als Seriell-nach-Parallel-Konverter
  - Konvertiert serielle Eingabe (S_{in}) in parallele Ausgabe ($Q_{0:N-1}$)

Symbol:

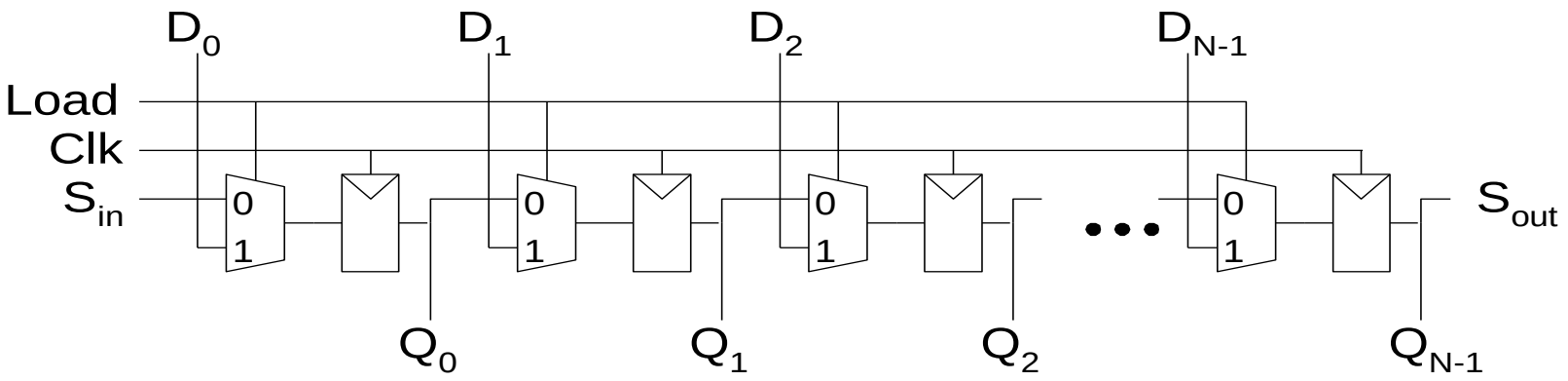


Aufbau:



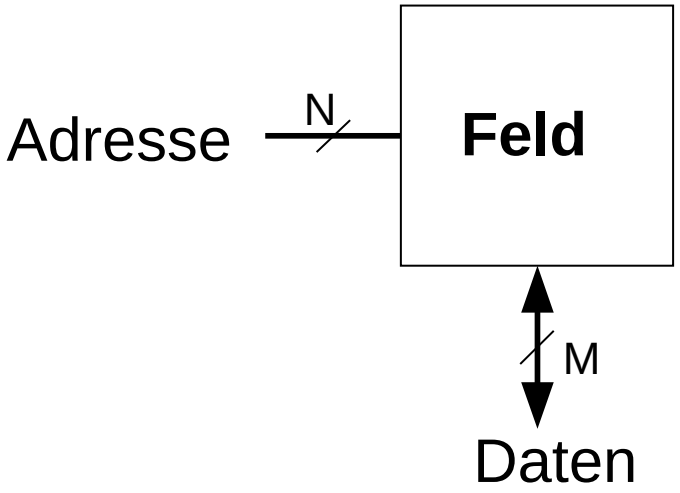
Schieberegister mit parallelem Laden

- Bei *Load* = 1: Agiert als normales *N*-bit Register
- Bei *Load* = 0: Agiert als Schieberegister
- Verwendbar als
  - Seriell-nach-Parallelkonverter (S_{in} nach $Q_{0:N-1}$)
  - Parallel-nach-Seriellkonverter ($D_{0:N-1}$ nach S_{out})



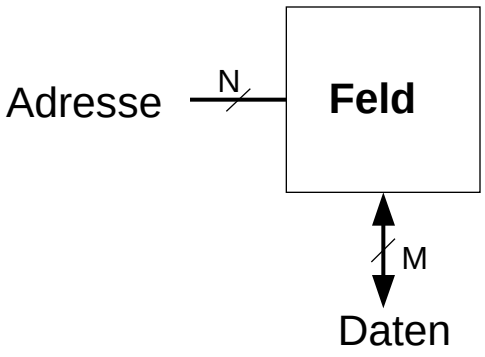
Speicherfelder

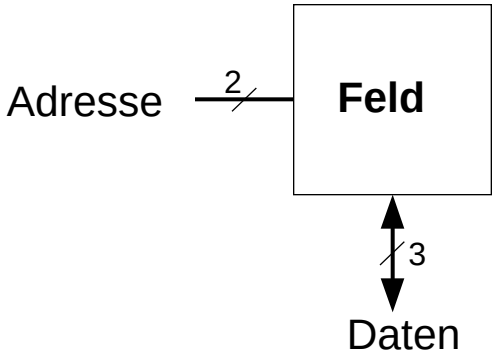
- Können effizient größere Datenmengen speichern
- Drei weitverbreitete Typen:
  - Dynamischer Speicher mit wahlfreiem Zugriff
    - (*Dynamic random access memory,* DRAM)
  - Statischer Speicher mit wahlfreiem Zugriff
    - (*Static random access memory,* SRAM)
  - Nur-Lesespeicher (*Read only memory*, ROM)
- An jede *N*-bit Adresse kann ein *M*-bit breites Datum geschrieben werden

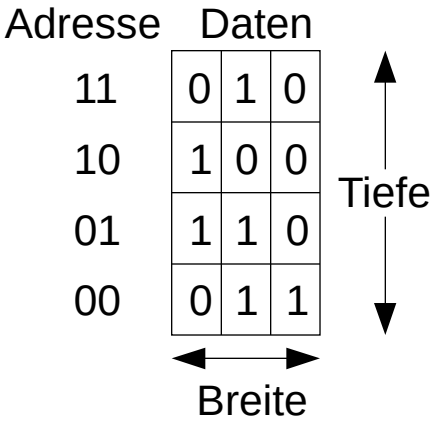


Speicherfelder

- Zweidimensionales Feld von Bit-Zellen
- Jede Bit-Zelle speichert ein Bit
- Feld mit *N* Adressbits und *M* Datenbits:
  - 2^N Zeilen und *M* Spalten
  - **Tiefe:** Anzahl von Zeilen (Anzahl von Worten)
  - **Breite:** Anzahl von Spalten (Bitbreite eines Wortes)
  - **Feldgröße:** Tiefe × Breite = $2^N \times M$



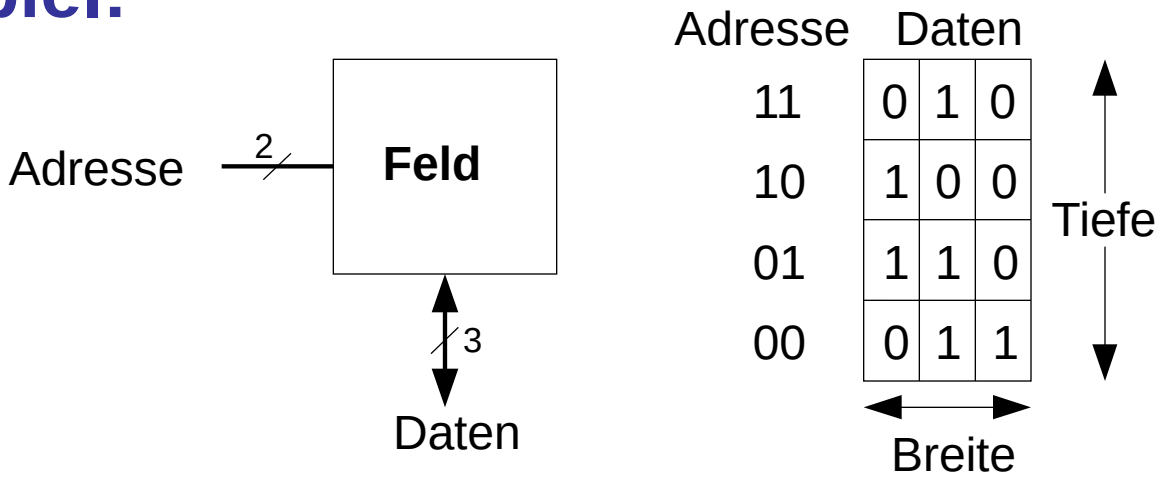




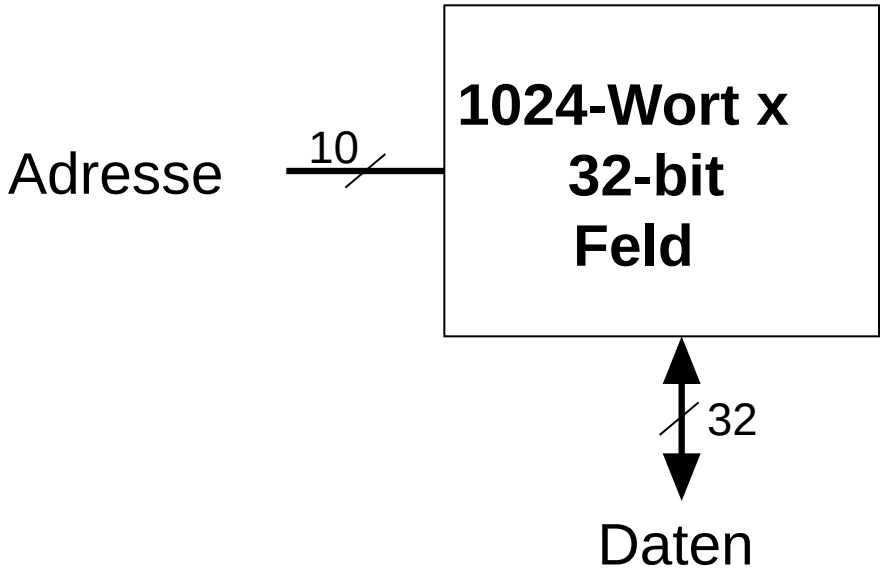
Beispiel: Speicherfeld

- $2^2 \times$ 3-Bit Feld
- Anzahl Worte: 4
- Wortbreite: 3-Bit
- Beispiel: 3-Bit gespeichert an Adresse 2'b10 ist 3'b100

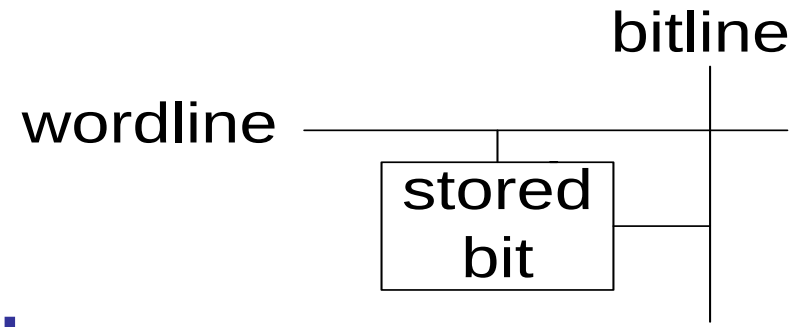
Beispiel:



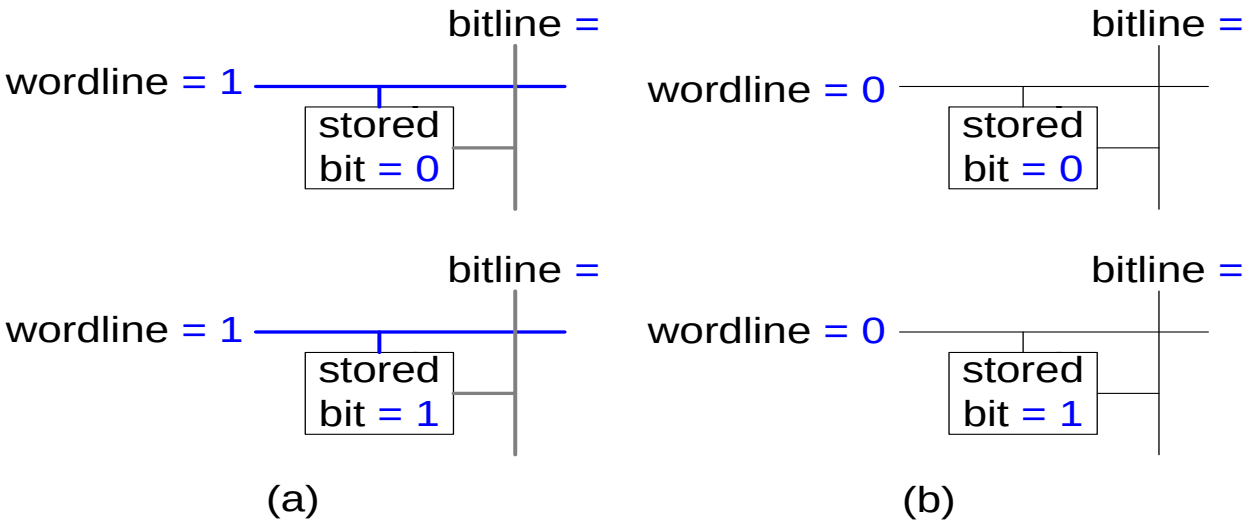
Speicherfelder



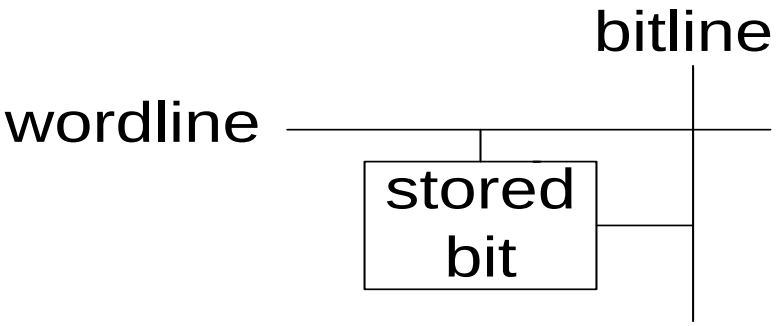
Bit-Zellen für Speicherfelder



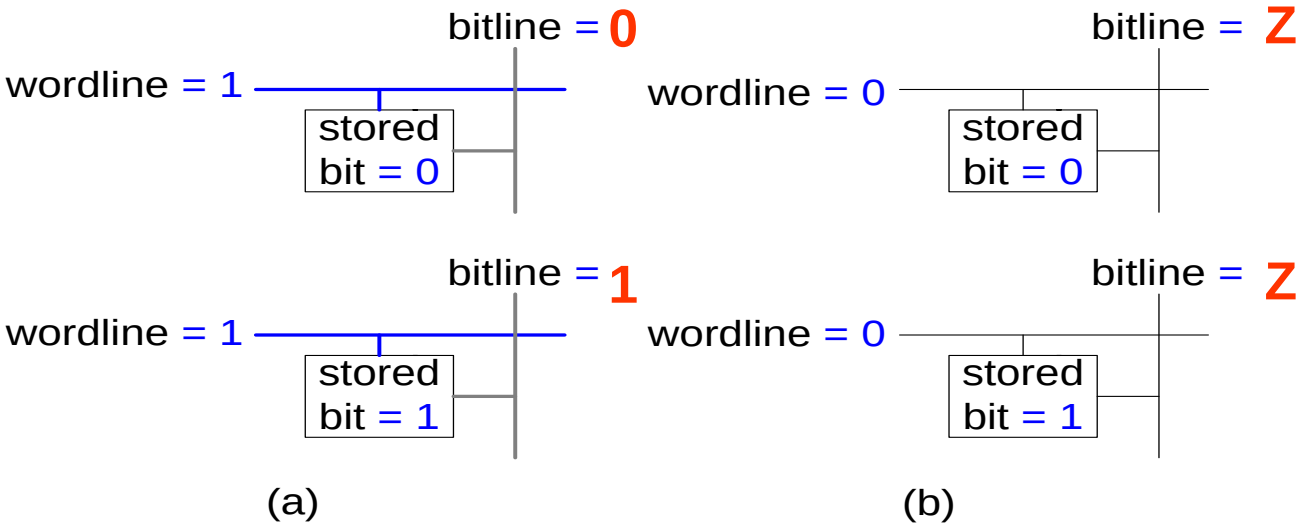
Beispiel:



Aufbau von Speicherfeldern aus Bit-Zellen



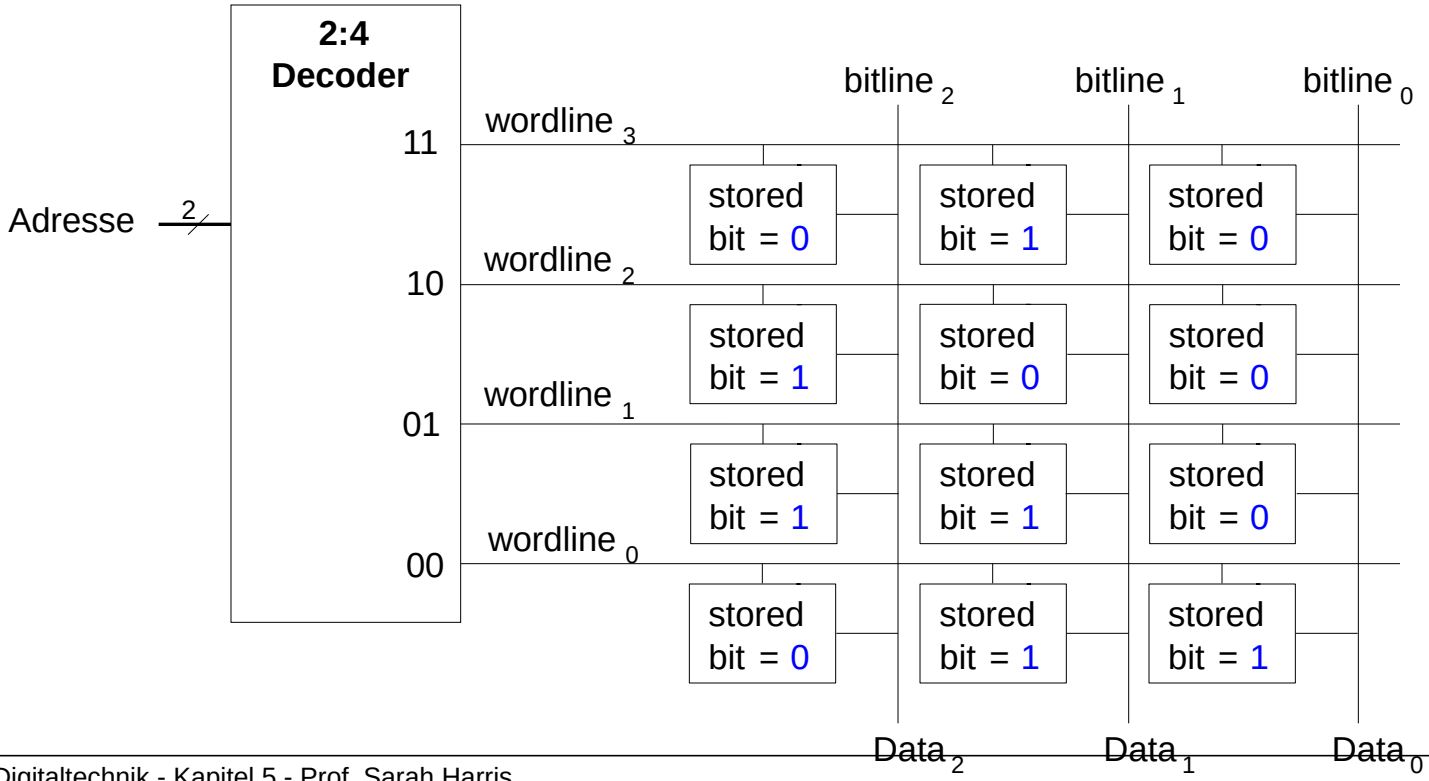
Beispiel:



Aufbau von Speicherfeldern

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Wordline:**
  - Vergleichbar mit Enable-Signal
  - Erlaubt Zugriff auf eine Zeile des Speichers zum Lesen oder Schreiben
  - Entspricht genau einer eindeutigen Adresse
  - Maximal eine Wordline ist zu jedem Zeitpunkt HIGH



Embedded Systems & Applications

Arten von Speicher: Historische Sicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- Speicher mit wahlfreiem Zugriff (RAM)
- Nur-Lese Speicher (ROM)

RAM: Random-Access Memory

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Flüchtig:** Speicherinhalte gehen bei Verlust der Betriebsspannung verloren
- Kann i.d.R. gleich schnell gelesen und geschrieben werden
- Zugriff auf beliebige Adressen mit ähnlicher Verzögerung möglich
- Hauptspeicher moderner Computer ist dynamisches RAM (DRAM)
  - Aktuell & genauer: DDR3-SDRAM
  - *Double Data Rate 3 - Synchronous Dynamic Random Access Memory*

- Name „RAM“ ist historisch gewachsen
  - Früher unterschiedliche Zugriffszeiten auf unterschiedliche Adressen
    - Bandspeicher, Trommelspeicher, Ultraschall-Laufzeitspeicher, …

ROM: Read-Only Memory

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Nicht-flüchtig:** Erhält Speicherinhalt auch ohne Betriebsspannung
- Schnell lesbar
- Schreibbar nur sehr langsam (wenn überhaupt)
- Flash-Speicher ist in diesem Sinne ein ROM
  - Kameras
  - Handys
  - MP3-Player

- Auch hier Nomenklatur „ROM“ historisch
  - Auch aus ROMs kann von beliebigen Adressen gelesen werden
  - Es gibt auch schreibbare Arten von ROMs
    - PROMs, EPROMs, EEPROMs, Flash

Arten von RAM

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zwei wesentliche Typen:
  - Dynamisches RAM (DRAM)
  - Statisches RAM (SRAM)
- Verwenden unterschiedliche Speichertechniken in den Bit-Zellen:
  - DRAM: Kondensator
  - SRAM: Kreuzgekoppelte Inverter

Robert Dennard, 1932 -

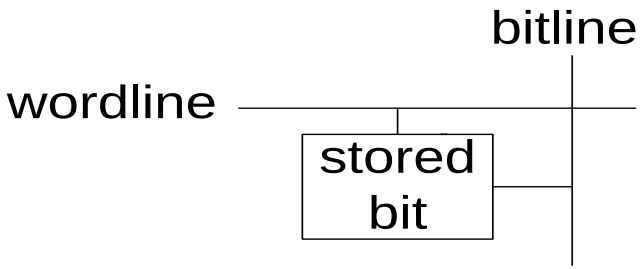
TECHNISCHE UNIVERSITÄT DARMSTADT

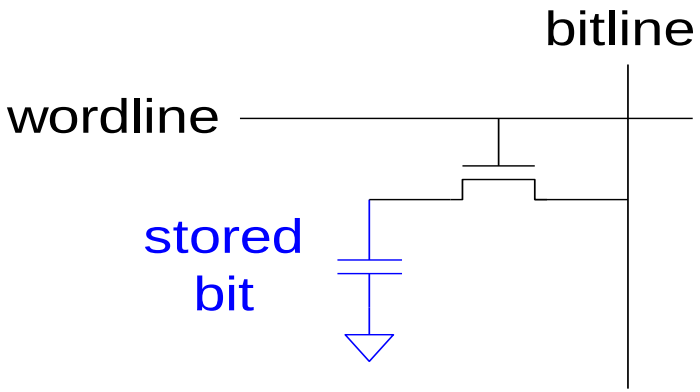
- Erfand 1966 bei IBM das DRAM
- Anfangs große Skepsis, ob Technik praktikabel
- Seit Mitte der 1970er Jahre ist DRAM die am weitesten verbreitete Speichertechnik in Computern

Embedded Systems & Applications

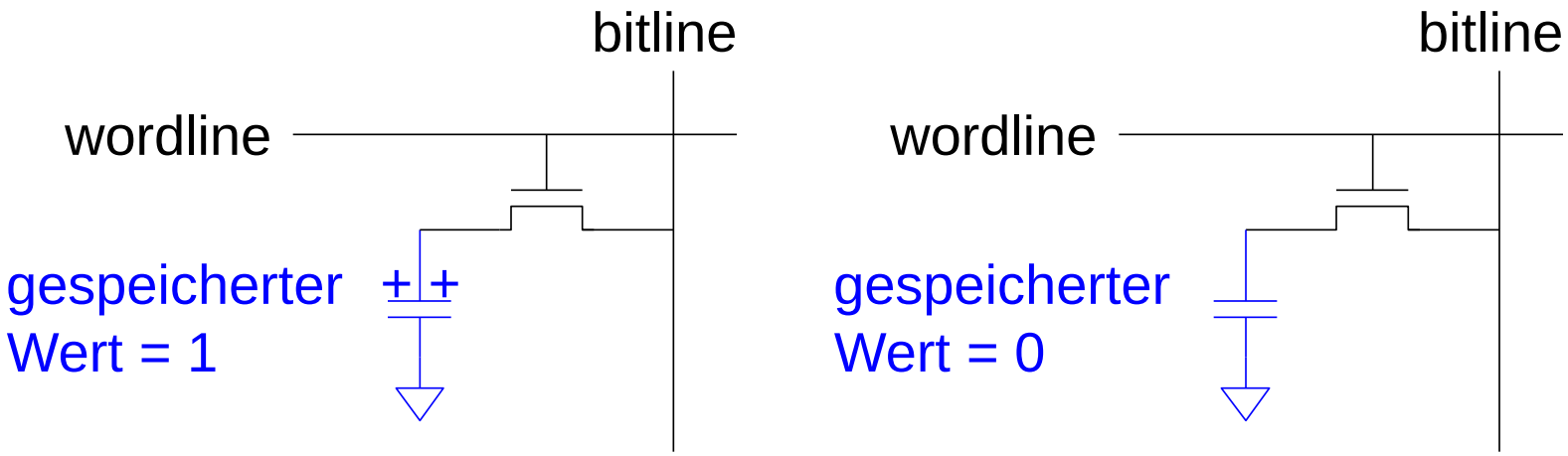
DRAM Bit-Zelle

- Datenbit wird als Ladezustand eines Kondensators gespeichert
- Dynamisch: Der Speicherwert muss periodisch neu geschrieben werden
  - Auffrischung alle paar Millisekunden erforderlich (üblich: 64ms)
  - Kondensator verliert Ladung durch Leckströme
  - … und beim Auslesen



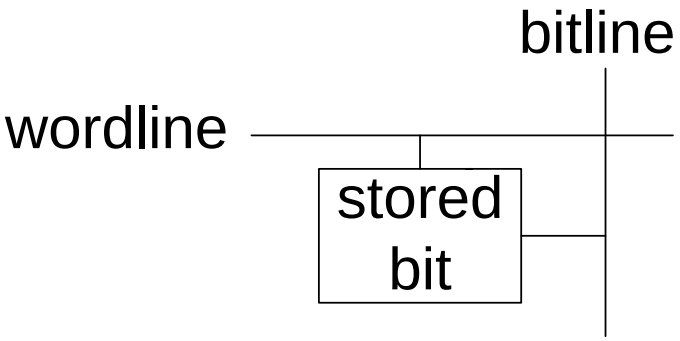


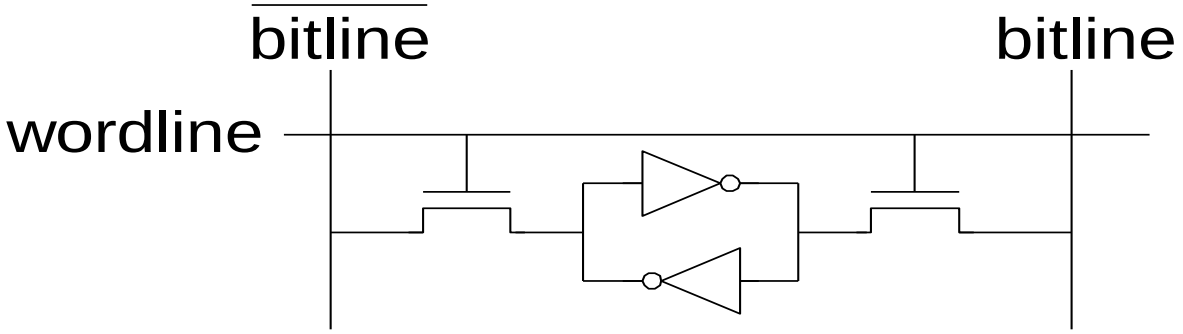
DRAM Bit-Zelle



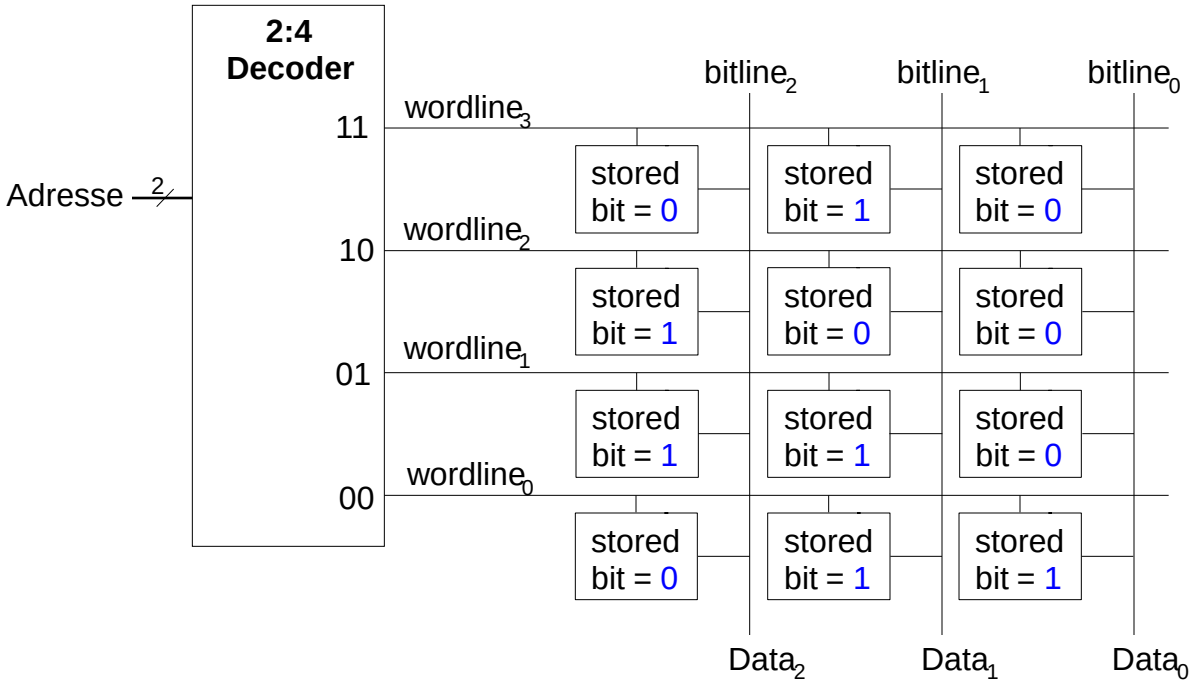
SRAM Bit-Zelle

- Datenbit wird als Zustand von rückgekoppelten Invertern gespeichert
- Statisch: Keine Auffrischung erforderlich
  - Inverter treiben Werte auf gültige Logikpegel

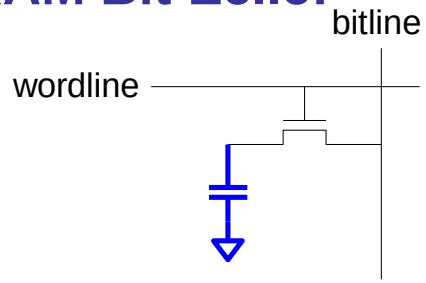




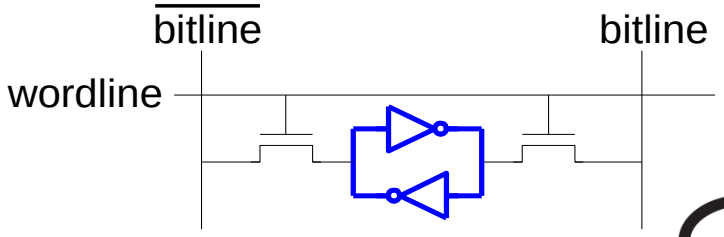
Speicherfelder



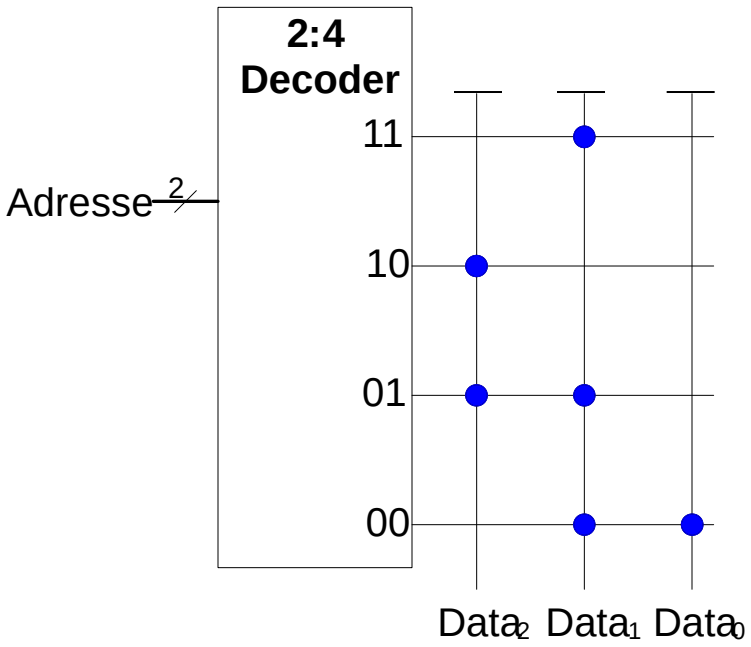
DRAM Bit-Zelle:

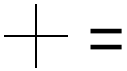


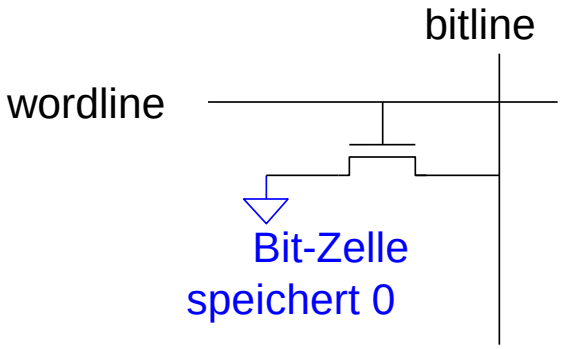
SRAM Bit-Zelle:

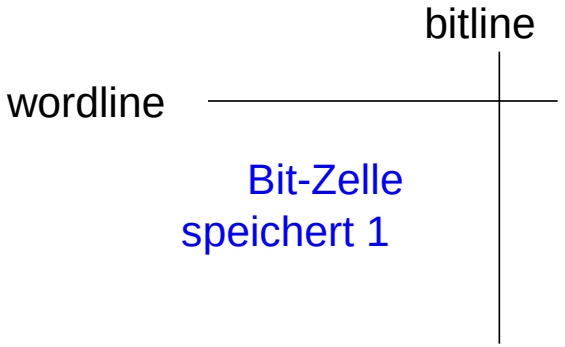


ROMs: Aufbau der Bit-Zellen









Bitlines sind schwach auf HIGH getrieben

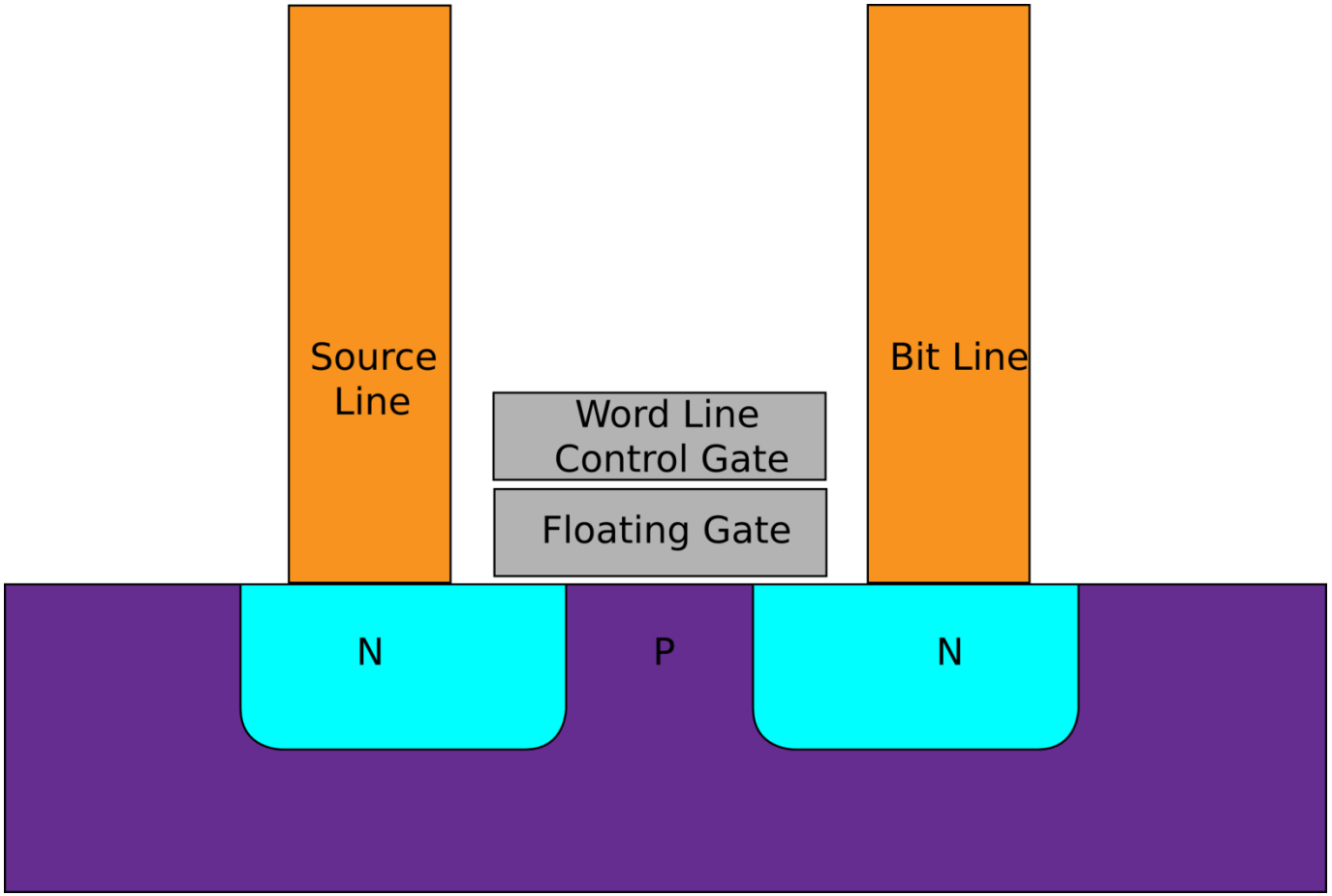
Fujio Masuoka, 1944-

TECHNISCHE UNIVERSITÄT DARMSTADT

- Entwickelte Speicher und schnelle Schaltungen bei Toshiba von 1971-1994
- Erfand Flash-Speicher als eigenes ungenehmigtes Projekt in den späten 1970ern
  - An Wochenenden und abends
- Löschvorgang erinnerte ihn an Kamerablitz
  - Deshalb Flash-Speicher
- Toshiba kommerzialisierte Technik nur zögerlich
- Erste kommerzielle Chips von Intel in 1988
- Flash-Produkte haben großen Erfolg
  - Derzeit USD 25 Milliarden Umsatz / Jahr

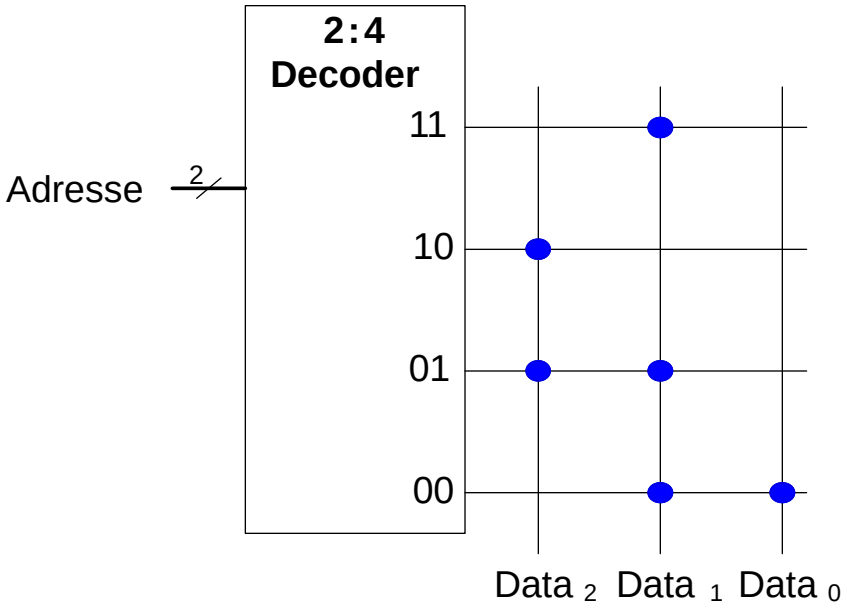
Embedded Systems & Applications

Flash-Speicher: Bit-Zelle



Quelle:Wikipedia

ROMs als Datenspeicher

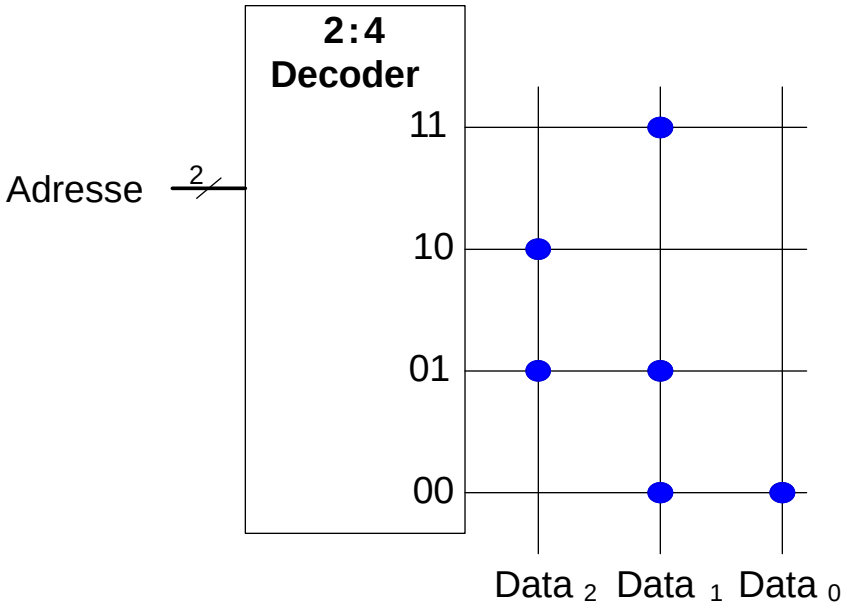


Adresse	Daten		
11	0	1	0
10	1	0	0
01	1	1	0
00	0	1	1

Tiefe

Breite

ROMs als Wertetabellen für boolesche Logik



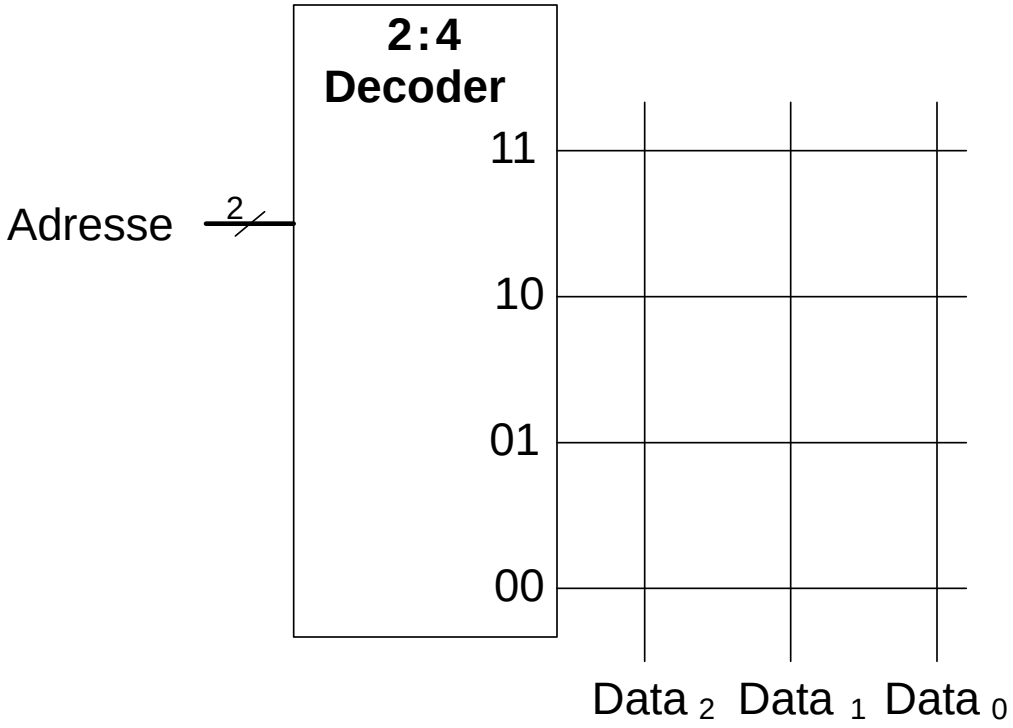
$$Data_2 = A_1 \oplus A_0$$

$$Data_1 = \overline{A_1} + A_0$$

$$Data_0 = \overline{A_1}\,\overline{A_0}$$

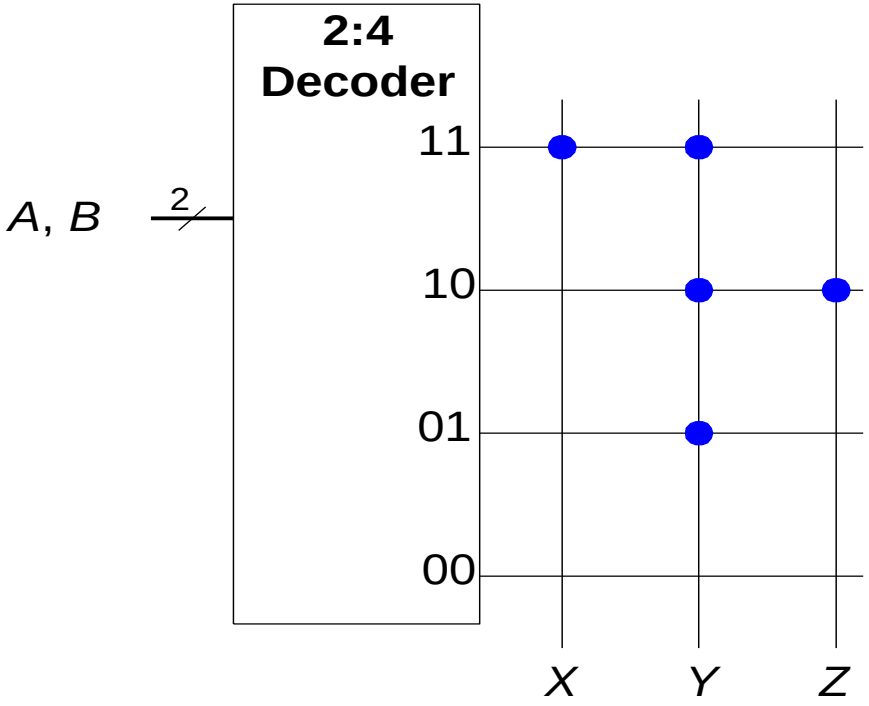
Beispiel: Logik aus ROMs

- Implementierung der folgenden logischen Funktionen durch $2^2 \times 3$-bit ROM:
  - $X = AB$
  - $Y = A + B$
  - $Z = A\overline{B}$

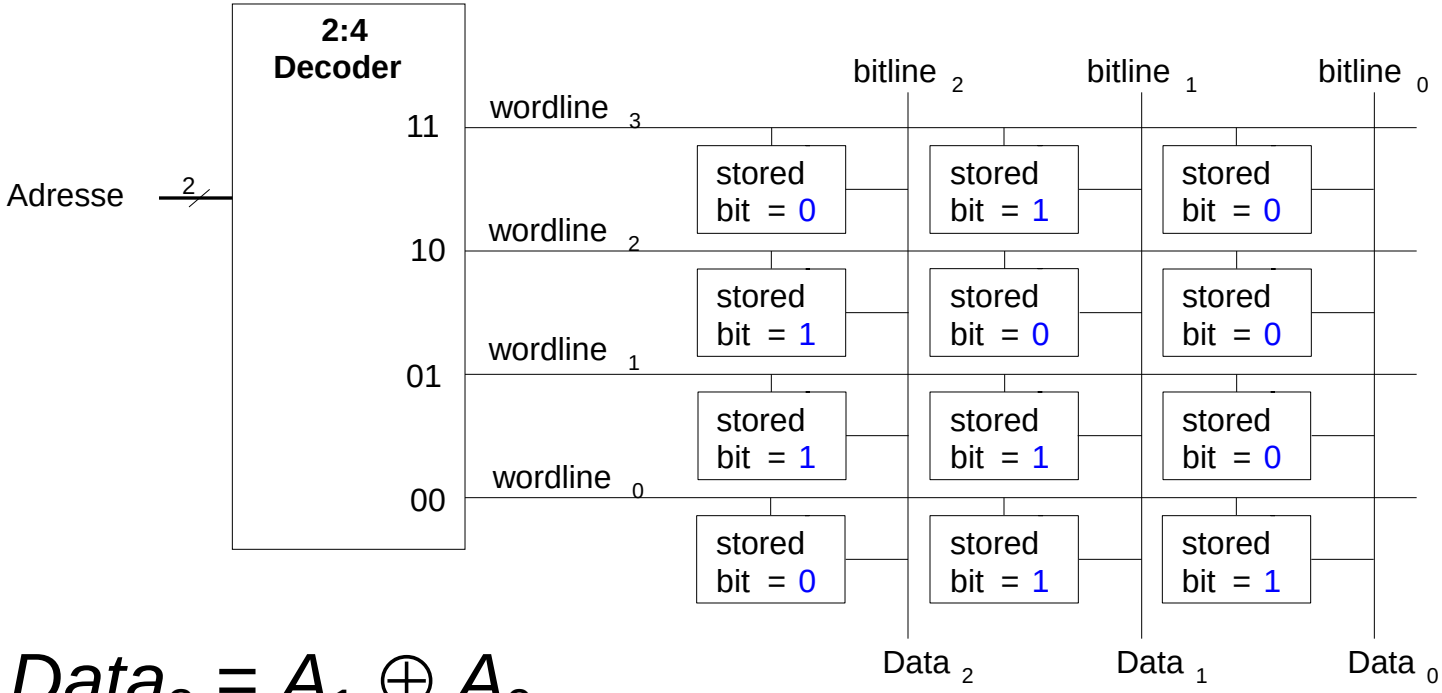


Beispiel: Logik aus ROMs

- Implementierung der folgenden logischen Funktionen durch $2^2 \times 3$-bit ROM:
  - $X = AB$
  - $Y = A + B$
  - $Z = A\overline{B}$



Logik aus beliebigem Speicherfeld



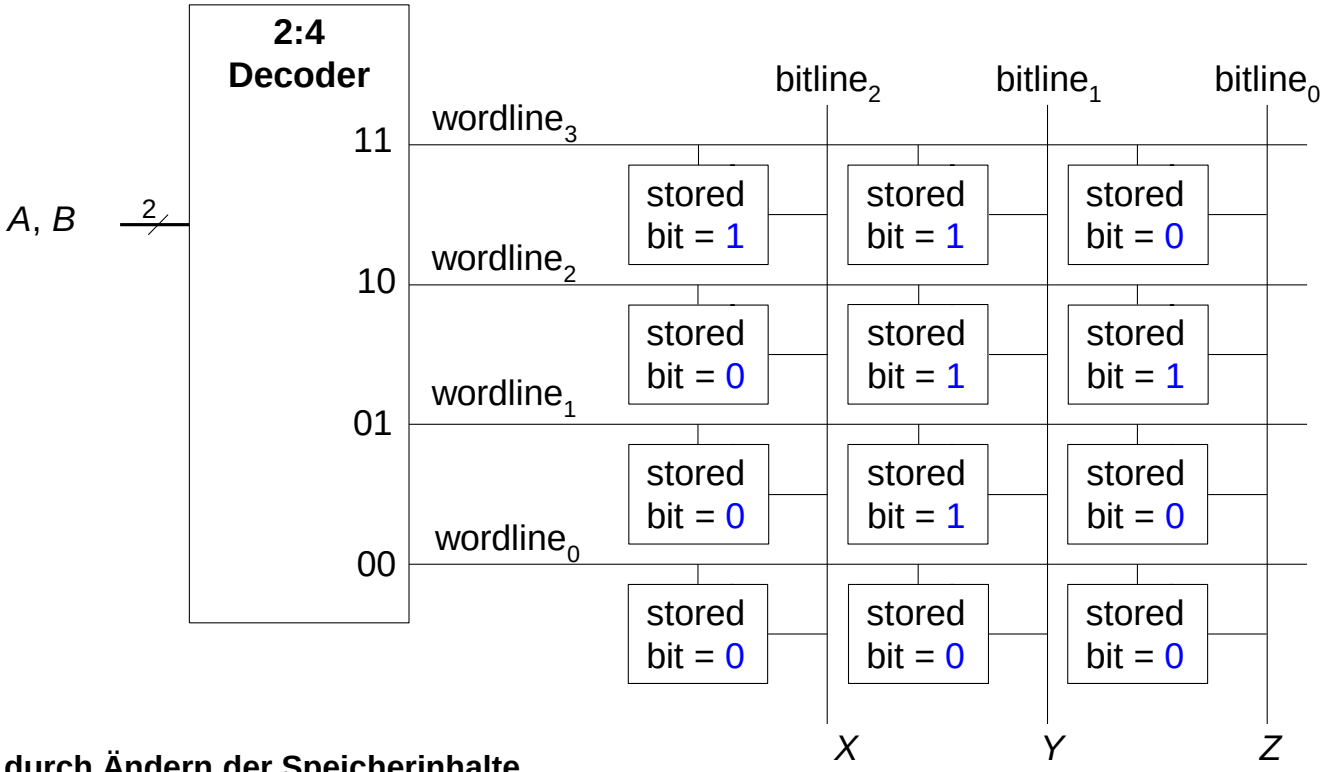
$$Data_2 = A_1 \oplus A_0$$

$$Data_1 = \overline{A_1} + A_0$$

$$Data_0 = \overline{A_1}\,\overline{A_0}$$

Logik aus beliebigem Speicherfeld

- Implementierung der folgenden logischen Funktionen durch $2^2 \times 3$-bit RAM:
  - $X = AB$
  - $Y = A + B$
  - $Z = A\overline{B}$



Andere Funktion nur durch Ändern der Speicherinhalte

Logik aus beliebigen Speicherfeldern

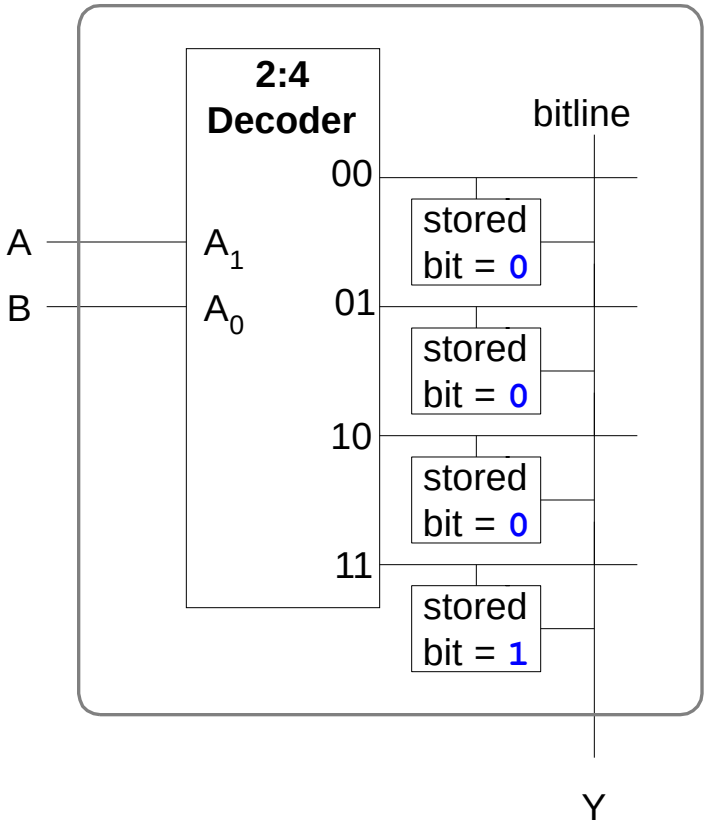
TECHNISCHE UNIVERSITÄT DARMSTADT

- Speicherfelder speichern Wertetabellen
  - Lookup-Tables (LUTs)
- Wort aus Eingangsvariablen bildet Adresse
- Für jede Kombination von Eingangsvariablen ist Funktionsergebnis abgespeichert

Werte-tabelle

A	B	Y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

4-Wort x 1-bit Speicherfeld

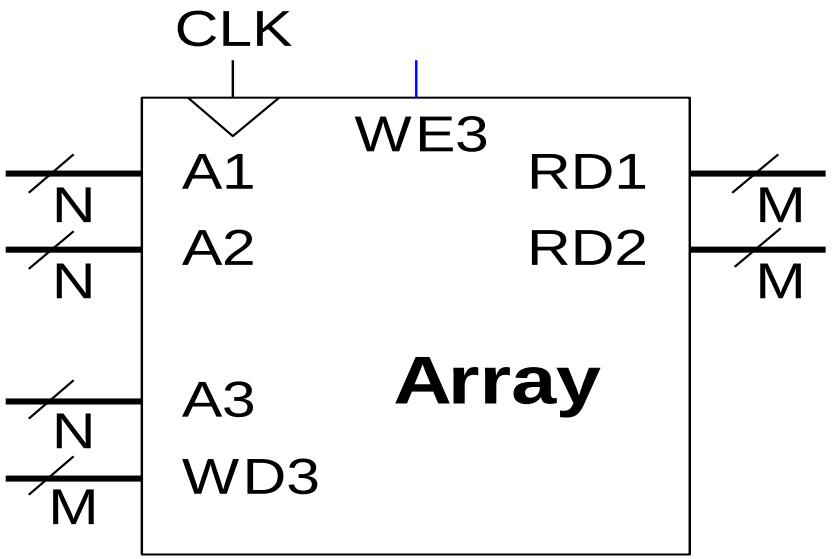


Embedded Systems & Applications

Multi-Port-Speicher

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Port:** Zusammengehörige Anschlüsse für Adresse und Datum
- Drei-Port Speicher
  - 2 Lese-Ports (A1/RD1, A2/RD2)
  - 1 Schreib-Port (A3/WD3, Signal WE3 löst Schreiben aus)
- Kleine Multi-Port-Speicher werden als Registerfelder bezeichnet
  - Werden z.B. in Prozessoren eingesetzt



Embedded Systems & Applications

Speicherfeld in SystemVerilog

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// 256 x 3b Speicher mit einem Schreib/Lese-Port
module dmem(input  logic       clk, we,
            input  logic [7:0] a,
            input  logic [2:0] wd,
            output logic [2:0] rd);

  logic  [2:0] RAM[255:0];

  assign rd = RAM[a];

  always @(posedge clk)
    if (we)
      RAM[a] <= wd;
endmodule
```


Embedded Systems & Applications

Logikfelder (*logic arrays*)

TECHNISCHE UNIVERSITÄT DARMSTADT

Programmable Logic Arrays (PLAs)

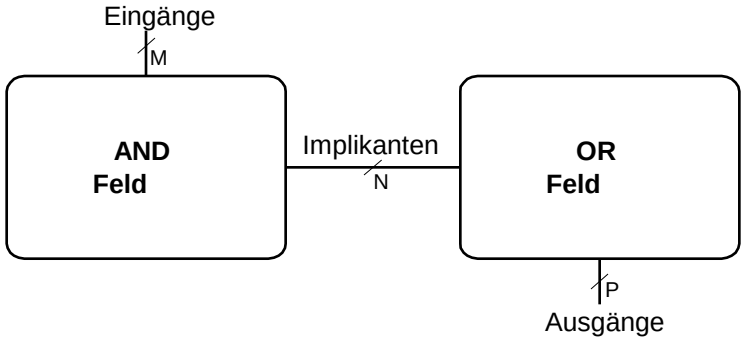
- AND Feld gefolgt von OR Feld
- Kann nur kombinatorische Logik realisieren
- Feste interne Verbindungen, spezialisiert für DNF (SoP-Form)

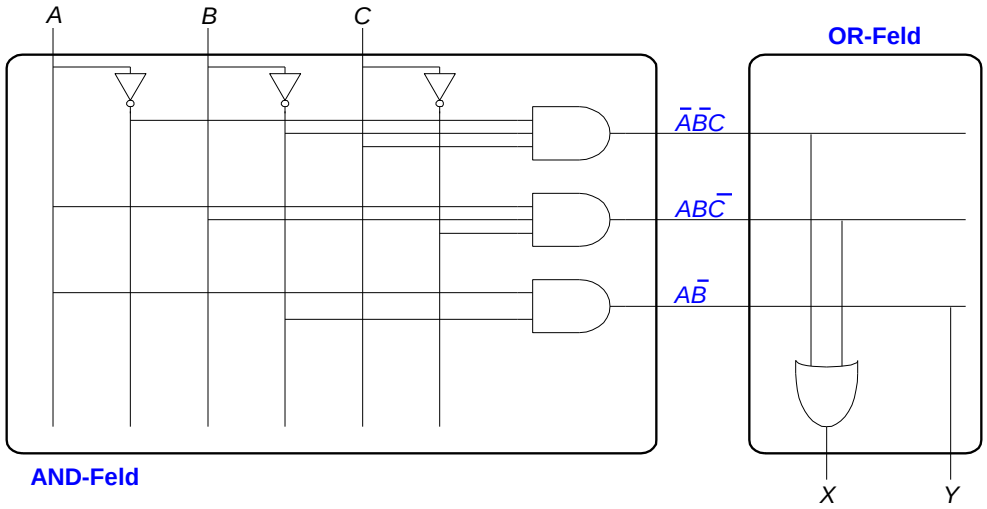
Field Programmable Gate Arrays (FPGAs)

- Feld von konfigurierbaren Logikblöcken (CLBs)
- Können kombinatorische und sequentielle Logik realisieren
- Programmierbare Verbindungsknoten zwischen Schaltungselementen

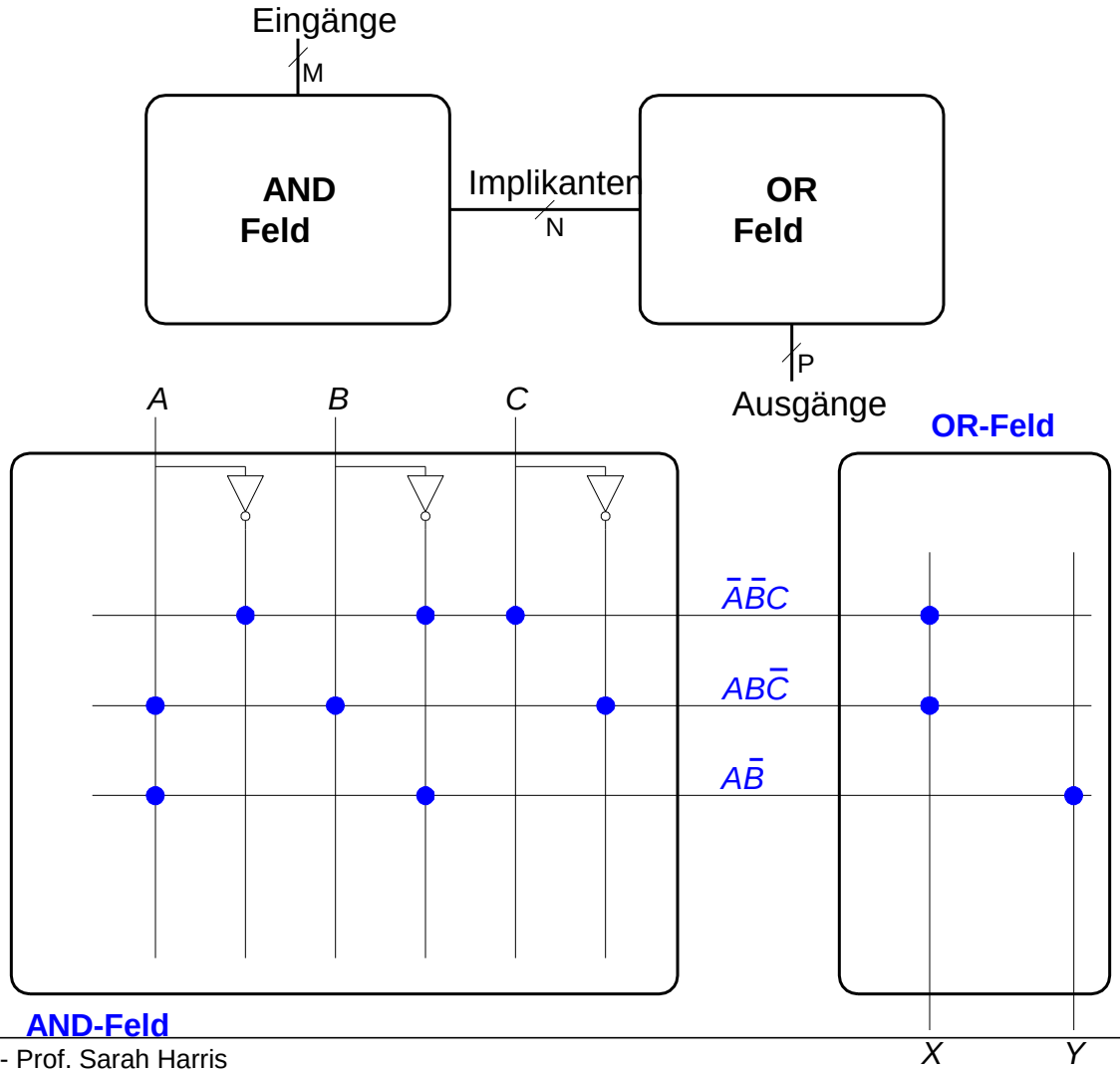
Boole‘sche Funktionen mit PLAs: Idee

- $X = \overline{A}\,\overline{B}C + AB\overline{C}$
- $Y = A\overline{B}$





PLAs: Vereinfachte Schreibweise



FPGAs: Field Programmable Gate Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

Bestehen grundsätzlich aus:

- **CLBs (Configurable Logic Blocks):** Realisieren kombinatorische und sequentielle Logik
  - Konfigurierbare Logikblöcke
- **IOBs (Input/Output Blocks):** Schnittstelle vom Chip zur Außenwelt
  - Ein-/Ausgabeblöcke
- **Programmierbares Verbindungsnetz:** verbindet CLBs und IOBs
  - Kann flexibel Verbindungen je nach Bedarf der aktuellen Schaltung herstellen

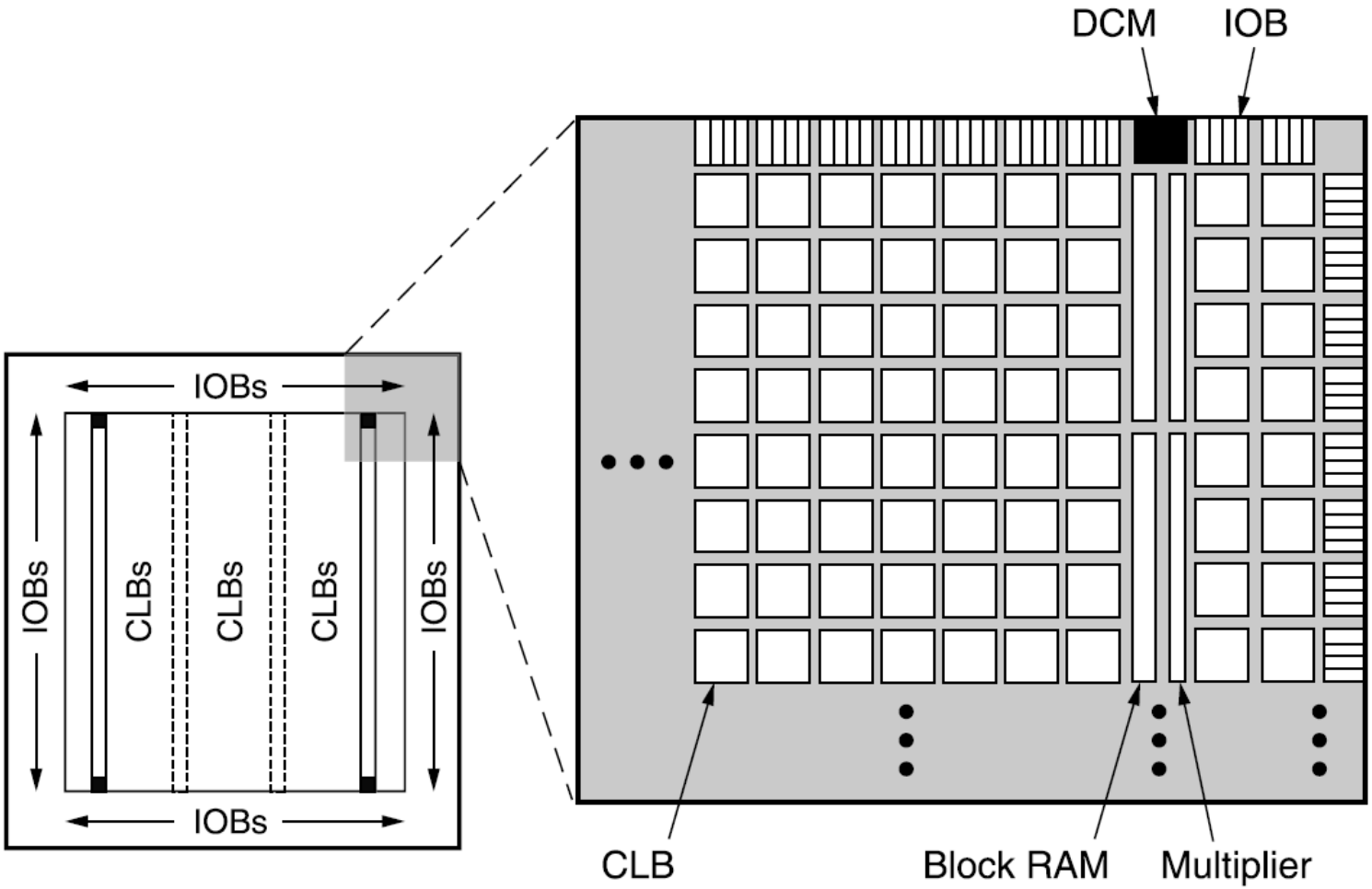
FPGAs: Field Programmable Gate Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

Reale FPGAs enthalten oftmals noch weitere Arten von Blöcken

- RAM
- Multiplizierer
- Manipulation von Taktsignalen (DCM)
- Sehr schnelle serielle Verbindungen (11 Gb/s)
- Komplette Mikroprozessoren
- …

Struktur eines Xilinx Spartan 3 FPGA



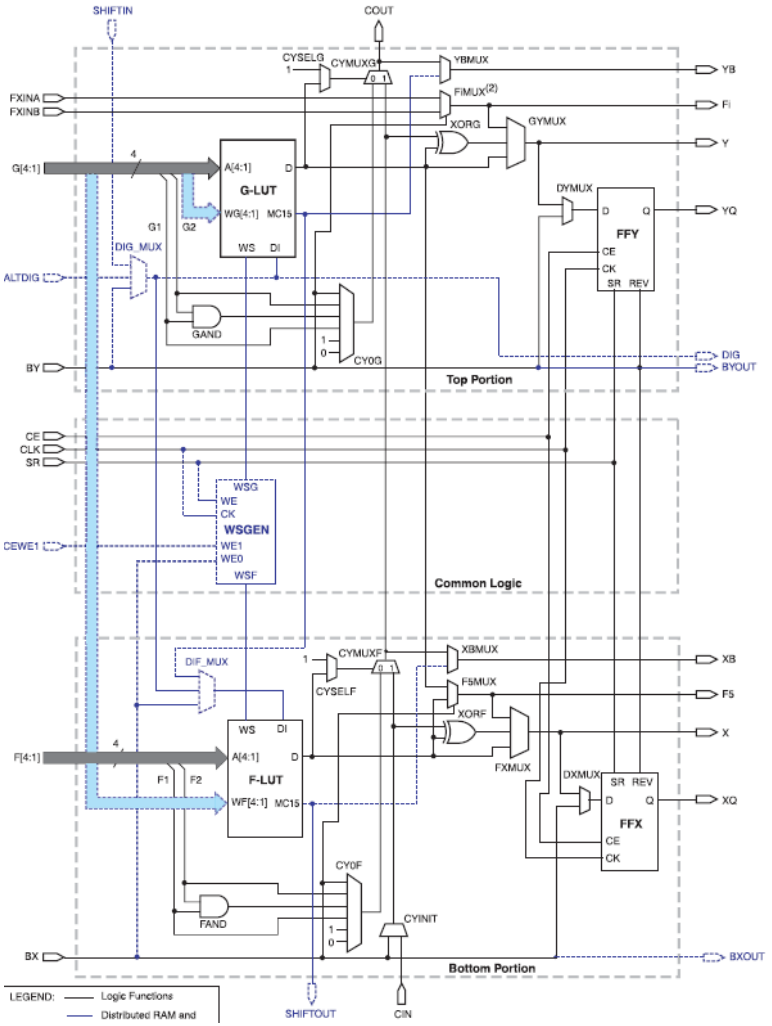
Konfigurierbare Logikblöcke (CLBs)

TECHNISCHE UNIVERSITÄT DARMSTADT

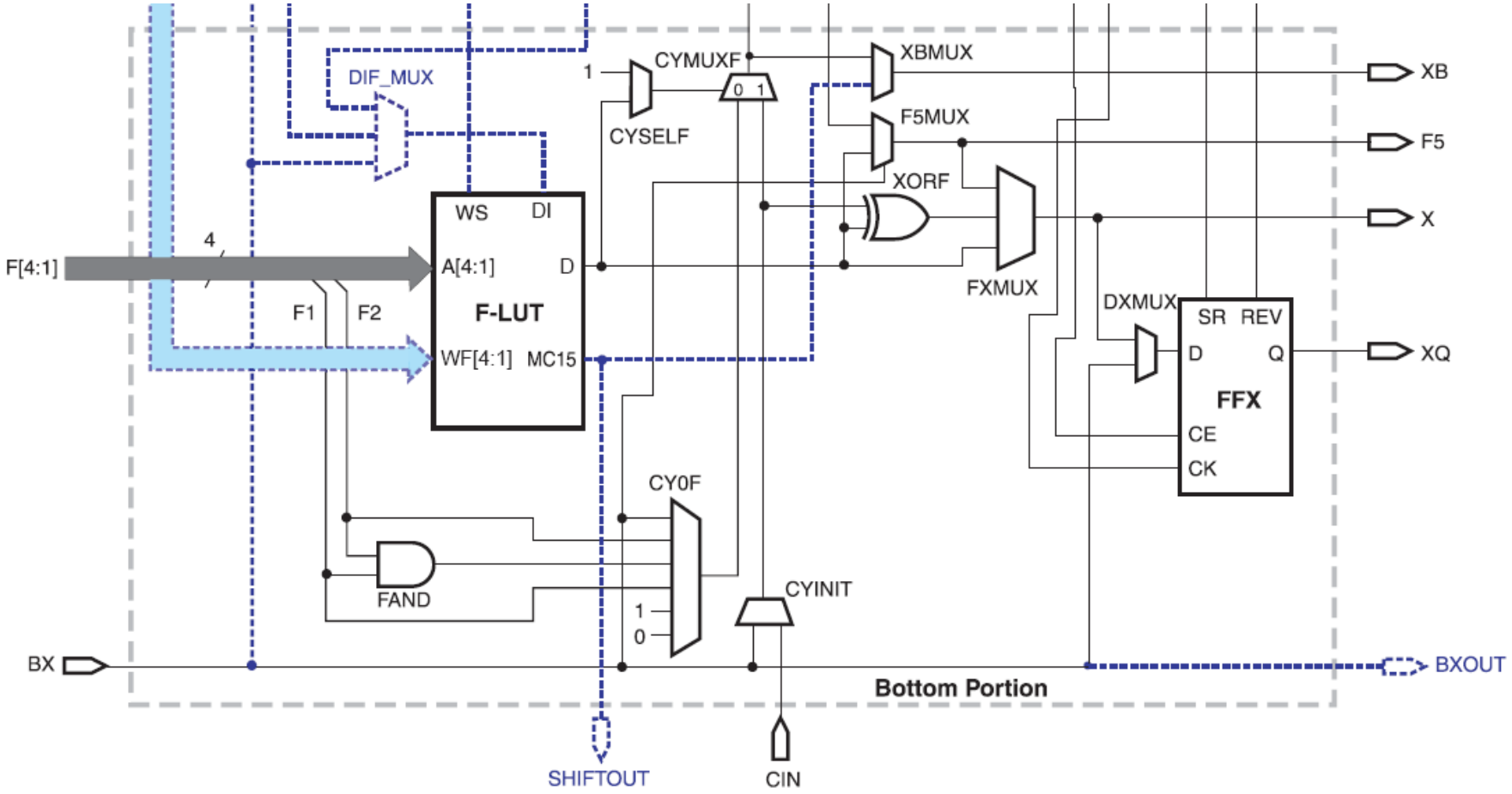
Bestehen im wesentlichen aus:

- **LUTs** (lookup tables): realisieren kombinatorische Funktionen
- **Flip-Flops**: realisieren sequentielle Funktionen
- **Multiplexern**: Verbinden LUTs und Flip-Flops

Xilinx Spartan 3 CLB



Xilinx Spartan CLB



Xilinx Spartan 3 CLB

TECHNISCHE UNIVERSITÄT DARMSTADT

Ein Spartan 3 CLB enthält:

- **2 LUTs:**
  - F-LUT (2^4 x 1-bit LUT)
  - G-LUT (2^4 x 1-bit LUT)
- **2 sequentielle Ausgänge:**
  - *XQ*
  - *YQ*
- **2 kombinatorische Ausgänge:**
  - *X*
  - *Y*

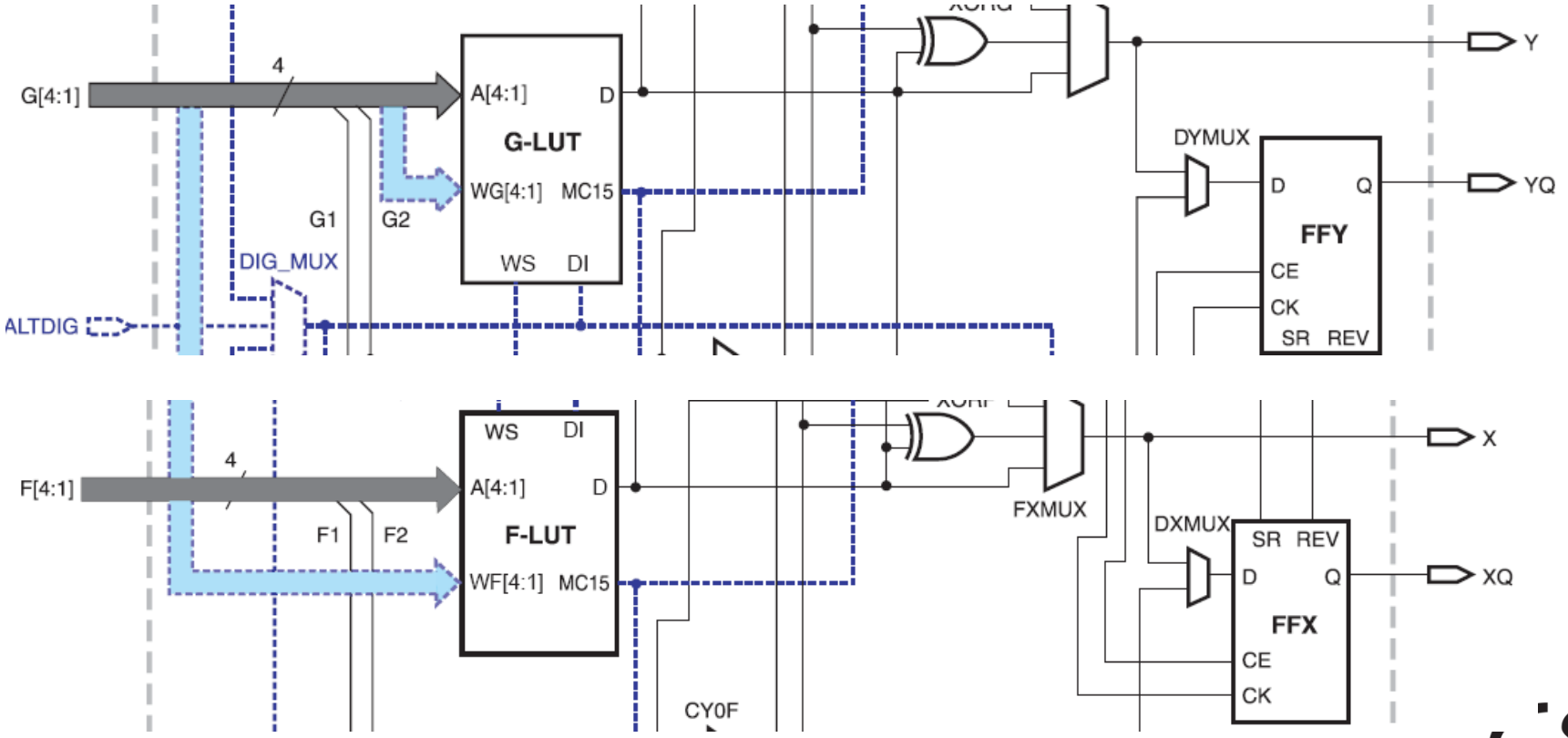
Embedded Systems & Applications

Beispiel: Kombinatorische Logik mit CLBs

TECHNISCHE UNIVERSITÄT DARMSTADT

Berechnung der folgenden Funktionen mit dem Spartan 3 CLB

- $X = A\overline{B}C + \overline{A}BC$
- $Y = AB$



Embedded Systems & Applications

Beispiel: Kombinatorische Logik mit CLBs

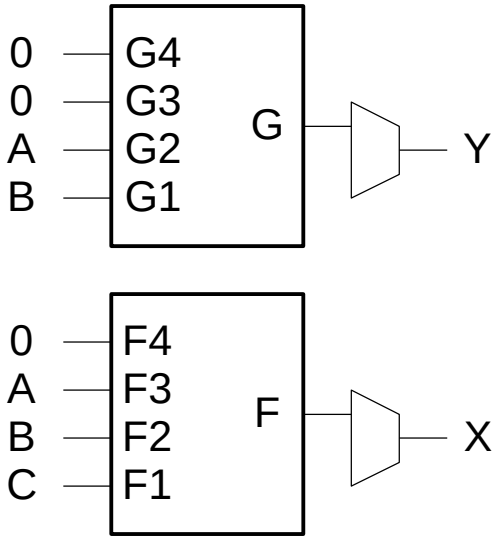
TECHNISCHE UNIVERSITÄT DARMSTADT

Berechnung der folgenden Funktionen mit dem Spartan 3 CLB

- $X = A\overline{B}C + \overline{A}\overline{B}C$
- $Y = AB$

	(A)	(B)	(C)	(X)
F4	F3	F2	F1	F
X	0	0	0	0
X	0	0	1	1
X	0	1	0	0
X	0	1	1	0
X	1	0	0	0
X	1	0	1	0
X	1	1	0	1
x	1	1	1	0

		(A)	(B)	(Y)
G4	G3	G2	G1	G
X	X	0	0	0
X	X	0	1	0
X	X	1	0	1
X	X	1	1	0



Embedded Systems & Applications

Entwurfsfluß für FPGAs

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wird in der Regel durch **Entwurfswerkzeuge** unterstützt
  - Beispiel: Vivado (LiveDemo an 10. Dezember 2015)
- Ist in der Regel ein **iterativer Prozess**
  - Planen
  - Implementieren (SystemVerilog)
  - Simulieren
  - Wiederhole …

Entwurfsfluß für FPGAs

TECHNISCHE UNIVERSITÄT DARMSTADT

Ist in der Regel ein iterativer Prozess

- Entwickler:
  - denkt nach
  - gibt Entwurf als Schaltplan oder HDL-Beschreibung ein
  - wertet Simulationergebnisse aus
- Wenn Simulation zufriedenstellend: Synthetisiere Entwurf in Netzliste
- Bilde Netzliste auf FPGA-Konfiguration ab (CLBs, IOBs, Verbindungsnetz)
- Lade Konfigurationsdaten (*bitstream*) auf FPGA
- Teste Schaltung nun in realer Hardware