

VO 050041:  
Technische Grundlagen der Informatik  
Begleitende Folien zur Vorlesung  
Wintersemester 2016/17

Vortragende: Peter Reichl, Andreas Janecek

Zuletzt aktualisiert: 21. Oktober 2016

# Teil 4: Digitale Logik

## Outline: Teil 4

### ① Digitale Schaltungen

- Logische Zustände

- Transistor

- Gatter

### ② Gatterschaltungen

- Einleitung

- Addierer

- Multiplexer

- Decoder

- Arithmetic Logic Unit ALU

- Integrierte Schaltungen (ICs)

### ③ Speicher

- Einleitung

- Read Only Memory (ROM)

- Random Access Memory (RAM)

# Literatur

- Mikroprozessortechnik (Wüst): Kapitel 3, 4, 5, 7
- Informatik (Blieberger, Springer-Verlag): Kapitel 5



# Überblick

## ① Digitale Schaltungen

Logische Zustände

Transistor

Gatter

## ② Gatterschaltungen

## ③ Speicher

# Analog vs. Digital

## Digitalschaltungen

- ...kennen nur die Zustände '0' und '1', z.B. per Spannungsniveau
- ...bestehen aus simplen Bausteinen: **Gatter**, **Speicher**, ...
- ...ermöglichen Aufbau komplexer Schaltungen (CPUs, etc.)
- NB: fundamentale Digitalkomponenten bestehen ihrerseits aus analogen Schalteilen.

## Gatter vs. Speicher

- 1 **Gatter implementieren logische Funktion und verändern Daten**
- 2 **Speicher speichern Daten**

# Physikalische Realisierung

## Wie realisiert man logische Zustände?

- unterschiedliche Spannungsniveaus
- Strom fließt/fließt nicht
- Kondensator geladen/ungeladen
- Position eines (elektromagnetischen) Relais'
- ...
- Also: viele Arten der Realisierung möglich  
→ manche praktisch, manche weniger...

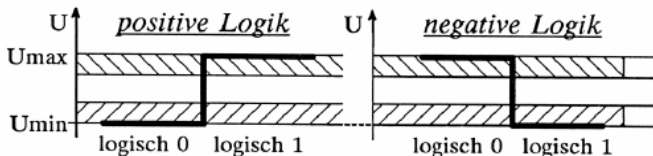
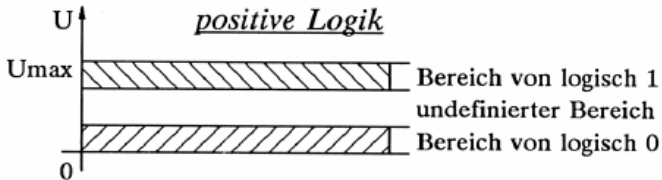
# Potential und Spannung

## Spannung

- Physik: Spannung = Potentialdifferenz
  - Formelzeichen:  $U$
  - Einheit: Volt ( $V$ )
  - $1 V = 1 J / C$  (Energie pro Ladung!)
- 
- Wo Ladung ist gibt es auch ein Potential.
  - **Zwischen zwei unterschiedlichen Potentialen liegt Spannung.**
  - **Es muss kein Strom fließen, damit Spannung da ist.**
  - Je größer die Spannung, desto mehr Energie trägt jedes Coulomb der Ladungen.

# Logische Zustände

## Zwei Spannungsniveaus - positive vs negative Logik:



# Halbleiter

## Was ist ein Halbleiter?

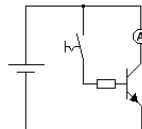
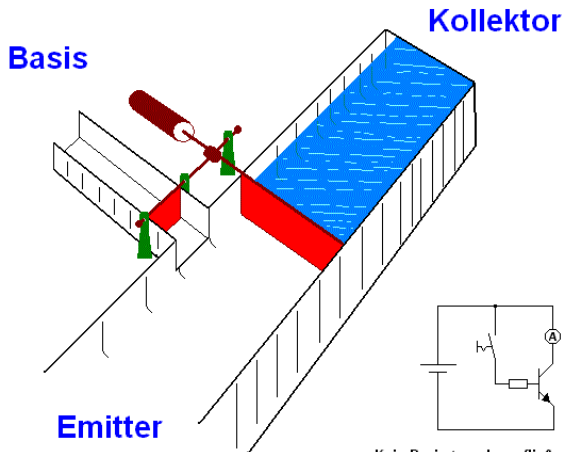
- elektrischer Leiter: Kristallgitter + freie Elektronen
  - Ladungstransport (Beispiel: Silber, Kupfer, ...)
- Isolator: Elektronen fest an Atome gebunden
- Halbleiter: Kristallgitter ohne freie Elektronen
  - leitet besser als Isolator, aber schlechter als Leiter
  - Beispiel: Germanium, Silizium, ...
- Idee: Dotierung = gezieltes Einschleusen von Fremdatomen in die Kristallstruktur
- Zwei Möglichkeiten:
  - *Donatoren* setzen ein Elektron frei
    - bewegliche negative Ladungen (*n-leitend*)
  - *Akzeptoren* binden jeweils ein Elektron
    - Löcher = “positive Ladungen” (*p-leitend*)

# Halbleiter

## Anwendung: Diode

- Besonders interessant: p-n-Übergang
- Diffusion: Elektronen wandern von n-Zone zur p-Zone  
→ elektrisches Feld, bis Gleichgewicht erreicht ist  
→ Entstehung einer Sperrschicht = Verarmungszone
- Was passiert beim Anlegen einer äusseren Spannung?
  - Durchlassrichtung: Minus an n-Zone, Plus an p-Zone
  - Sperrrichtung: Plus an n-Zone, Minus an p-Zone
- Wichtige Anwendungen:
  - Gleichrichterdiode
  - Bipolartransistor (npn- bzw. pnp-Transistor)

# Prinzip des Transistors: Offen



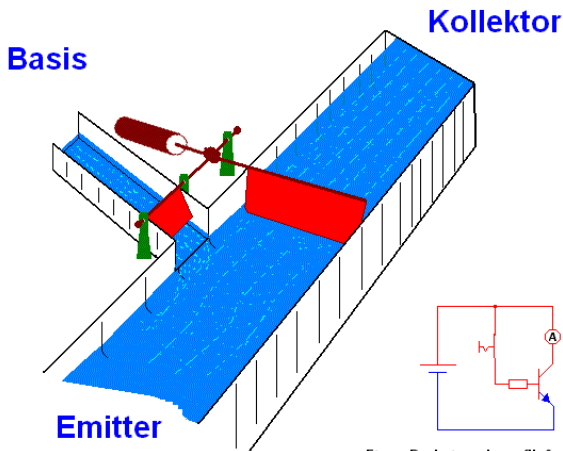
Kein Basisstrom kann fließen;  
Der Transistor ist gesperrt.

1

<sup>1</sup> [http://de.academic.ru/pictures/dewiki/116/transistor\\_animation.gif](http://de.academic.ru/pictures/dewiki/116/transistor_animation.gif)



# Prinzip des Transistors: Geschalten



Etwas Basisstrom kann fließen;  
Der Transistor schaltet durch.

# Feldeffekt-Transistor

## MOSFET - Metal Oxide Semiconductor Field Effect Transistor

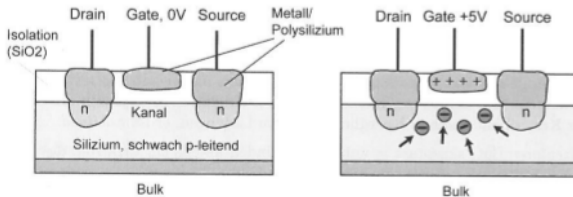
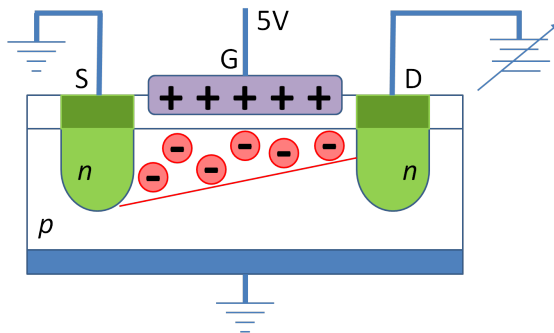


Abbildung 3.1: Selbstsperrender n-Kanal-MOSFET. Ein leitender Kanal bildet sich erst aus, wenn positive Gate-Spannung anliegt (rechts).

2

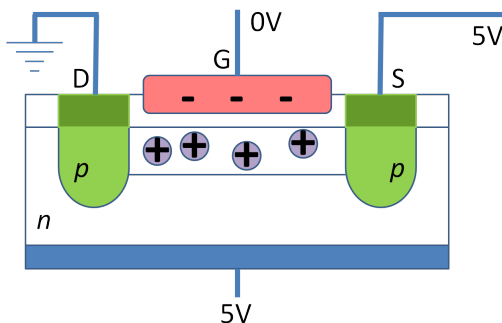
- Vier Anschlüsse: Source - Drain - Gate - Bulk
- Grundsätzliche Varianten: n-Kanal-MOSFET (NMOS) vs. p-Kanal-MOSFET (PMOS)

# Feldeffekt-Transistor: NMOS



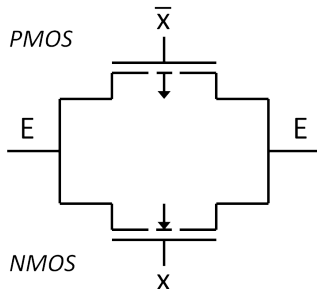
- Gate nicht geladen: keine Ladungen im Kanal, kein Strom von Source nach Drain
- Gate geladen: Kanal leitet
- Aber: zusätzlich p-n-Übergang am Drain → Sperrichtung!
- Also: NMOS leitet niedriges  $U_{DS}$  falls hohes  $U_G$

# Feldeffekt-Transistor: PMOS



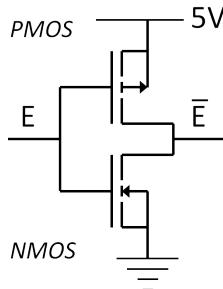
- Im Vergleich zum NMOS alles umgedreht!
- Niedriges Potential am Gate:  
Kanal enthält Löcher, keine vollständige Rekombination
- Also: PMOS leitet hohes  $U_{DS}$  falls niedriges  $U_G$

# Feldeffekt-Transistor



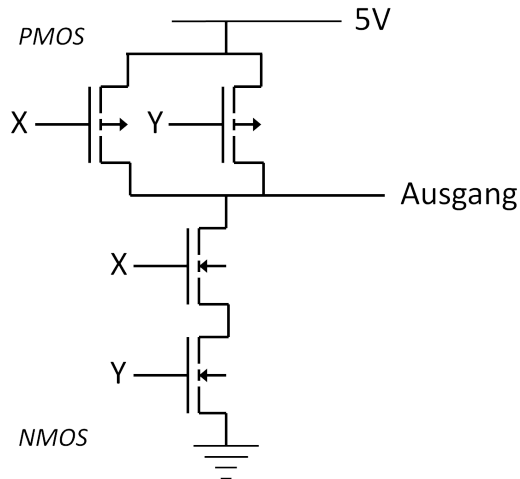
- MOSFET allein noch kein Schalter  
(NMOS leitet nur niedriges  $U_{DS}$ , PMOS nur hohes  $U_{DS}$ )
- Idee: kombiniere NMOS + PMOS in einer Schaltung  
→ Transfer Gate (= Schalter, Transmission Gate)

# CMOS: Complementary MOS

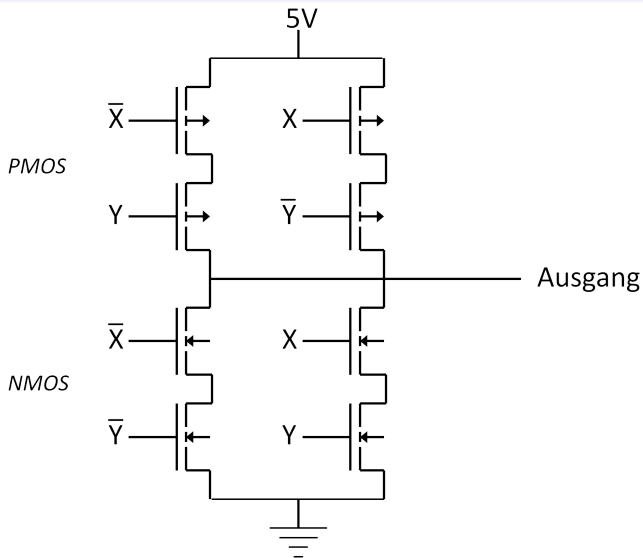


- Kombination von NMOS- (Pull-down-Pfad) und PMOS-Technologie (Pull-up-Pfad) auf gemeinsamem Substrat
- Einfachste CMOS-Schaltung: Inverter
- Viele komplexe Logikschaltungen benötigen Inverter an Ein- bzw. Ausgängen (z.B. XOR)

# CMOS: NAND



# CMOS: XOR





# Gatter

## Gatter

- Haben einen/mehrere Eingänge und einen Ausgang.
- Wert des Ausgangs hängt von den Werten der Eingänge und des Gattertyps ab.

## Gatter mit einem Eingang

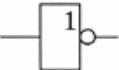











- **NOT**: liefert '1' wenn der Eingang '0' ist
- **Buffer**: liefert '1' wenn der Eingang '1' ist

# Gatter

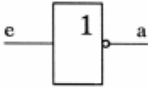
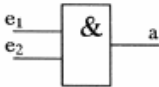
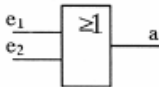
## Gatter mit mehreren Eingängen

- **AND**: liefert '1' wenn alle Eingänge '1' sind
- **NAND**: liefert '0' wenn alle Eingänge '1' sind
- **OR**: liefert '1' wenn mindestens ein Eingang '1' ist
- **NOR**: liefert '0' wenn mindestens ein Eingang '1' ist
- **XOR ("Antivalenz")**: liefert '1' wenn eine ungerade Anzahl von Eingängen '1' ist

# Gattersymbole

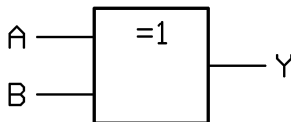
<i>Funktion</i>	<i>Aktuelle Darstellung</i>	<i>DIN alt</i>	<i>USA alt</i>
<i>NOT</i>			
<i>AND</i>			
<i>OR</i>			
<i>Antivalenz</i>			

# Grundgatter: NOT, AND, OR

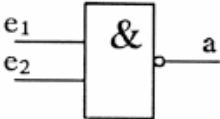
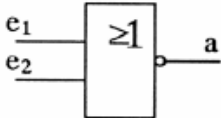
<i>NOT</i>	<i>AND</i>	<i>OR</i>
$a = \neg e$	$a = e_1 \wedge e_2$	$a = e_1 \vee e_2$
		

# Grundgatter: XOR

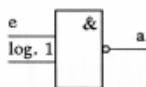
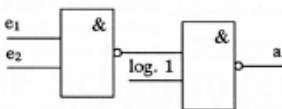
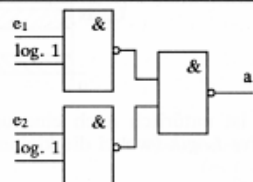
- $Y = A \underline{\vee} B$  (alternativ:  $Y = A \oplus B$ )
- Ausgang ist genau dann logisch '1', wenn an einer ungeraden Anzahl von Eingängen '1' anliegt und an den restlichen '0'



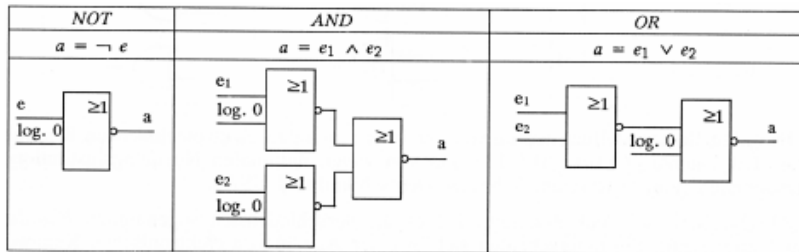
# Grundgatter: NAND, NOR

<i>NAND</i>	<i>NOR</i>
$a = \neg (e_1 \wedge e_2)$	$a = \neg (e_1 \vee e_2)$
	

# NAND: Logisch vollständig

<i>NOT</i>	<i>AND</i>	<i>OR</i>
$a = \neg e$	$a = e_1 \wedge e_2$	$a = e_1 \vee e_2$
		

# NOR: Logisch vollständig





# Überblick

## 1 Digitale Schaltungen

## 2 Gatterschaltungen

Einleitung

Addierer

Multiplexer

Decoder

Arithmetic Logic Unit ALU

Integrierte Schaltungen (ICs)

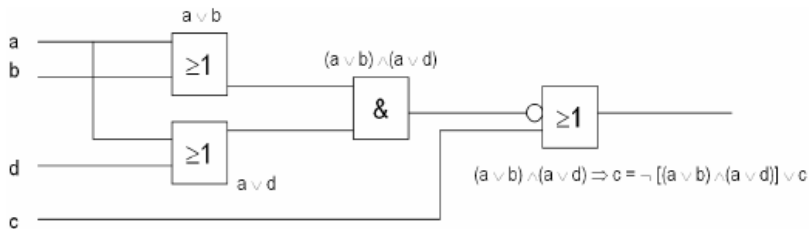
## 3 Speicher

# Gatterschaltung: Beispiel 1

## Logische Funktion

- $f = [(a \vee b) \wedge (a \vee d)] \Rightarrow c$
- Nur NOT, AND, OR verwenden, daher umformen
- $f = \neg[(a \vee b) \wedge (a \vee d)] \vee c$

# Gatterschaltung: Beispiel 1



## Gatterschaltung: Beispiel 2

Ausgang true, wenn binärer Wert der Eingänge kleiner 3

Dezimal	Binär	$e_1$	$e_2$	$e_3$	$a$
0	000	0	0	0	1
1	001	0	0	1	1
2	010	0	1	0	1
3	011	0	1	1	0
4	100	1	0	0	0
5	101	1	0	1	0
6	110	1	1	0	0
7	111	1	1	1	0

### DNF

$$a = f(e_1, e_2, e_3) =$$

$$(\neg e_1 \wedge \neg e_2 \wedge \neg e_3) \vee (\neg e_1 \wedge \neg e_2 \wedge e_3) \vee (\neg e_1 \wedge e_2 \wedge \neg e_3)$$

## Gatterschaltung: Beispiel 2

	$e_1$	$e_1$	$\neg e_1$	$\neg e_1$
$e_3$	0	0	1	0
$\neg e_3$	0	0	1	1
	$e_2$	$\neg e_2$	$\neg e_2$	$e_2$

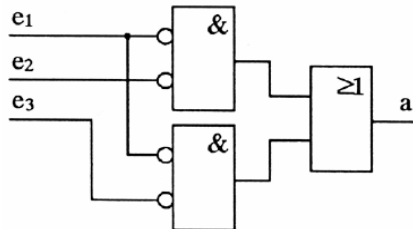
Minimiert

$$a = f(e_1, e_2, e_3) = (\neg e_1 \wedge \neg e_2) \vee (\neg e_1 \wedge \neg e_3)$$

## Gatterschaltung: Beispiel 2

### Minimiert

$$a = f(e_1, e_2, e_3) = (\neg e_1 \wedge \neg e_2) \vee (\neg e_1 \wedge \neg e_3)$$



# Arten von Gatterschaltungen

## Kombinationsschaltungen

- Multiplexer
- Dekodierer
- Komparatoren
- Programmable Logic Arrays (PLA)

## Arithmetische Schaltungen

- Schieber (Shifter)
- Addierer
- Arithmetische Logikeinheiten (ALUs)

Taktgeber/Clocks, ...

# Halbaddierer

## Halbaddierer

- Addiert zwei **einstellige** Binärzahlen  $\Rightarrow$  2 Eingänge
- Zwei Ausgänge: Summe (S), Carry (C)

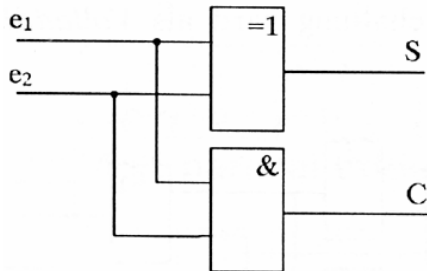
## Wahrheitstabelle

$e_1$	$e_2$	<b>C</b>	<b>S</b>
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- **S** entspricht XOR
- **C** entspricht AND



# Halbaddierer: Schaltung

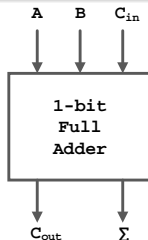


- geht auch nur mit AND und OR
- einfache Rechenzeitabschätzung für diesen Fall:  
Annahme  $10psec$  pro Gatter  $\rightarrow$  gesamter HA:  $30psec$   
(kritisch: XOR-Gatter, braucht  $30psec$ )

# Volladdierer

## 1-bit Volladdierer

- EIN 1-bit Volladdierer addiert **drei einstellige** Binärzahlen  
⇒ 3 Eingänge
- drei Eingänge:  $A$ ,  $B$ , und Carry in  $C_{in}$
- $C_{in}$  entspricht dem “Eingangsübertrag”
- zwei Ausgänge: Summe ( $S$  bzw.  $\Sigma$ ), Carry out ( $C_{out}$ )



# Volladdierer: Wahrheitstabelle

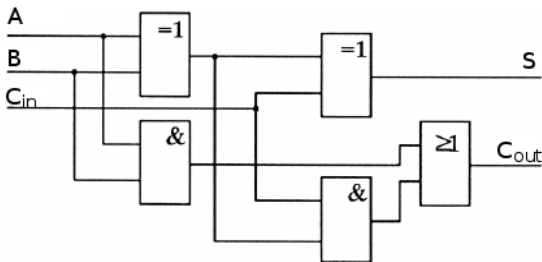
## Wahrheitstabelle eines 1-bit Volladdierers

$A$	$B$	$C_{in}$	$C_{out}$	$S$
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

# Volladdierer: Schaltung

## Minimiert

- $S = A \oplus B \oplus C_{in}$
- $C_{out} = (A \wedge B) \vee (C_{in} \wedge (A \oplus B))$

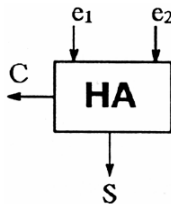


- Abschätzung: zwei HA + OR-Gatter → 70psec gesamt
- etwas genauer: zwei XOR-Gatter für S → 60psec gesamt

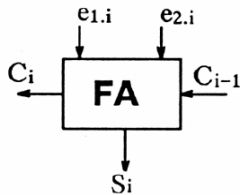
# Blockschaltbilder: Halb- und Volladdierer

## Blockschaltbilder

- Eingänge, Ausgänge, Funktion
- nur Funktionalität, nicht innere Struktur (Blackbox)
- enthalten Module, die wieder kombiniert werden können

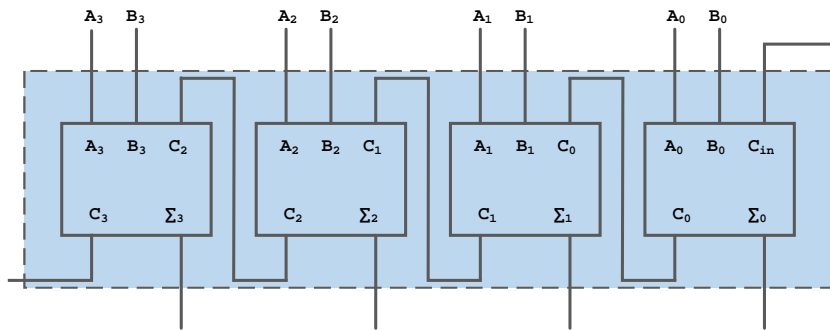


Halbaddierer



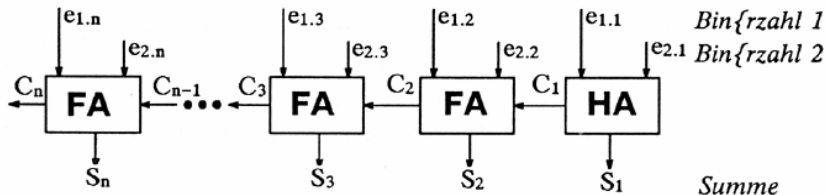
Volladdierer

# Carry-Ripple-Addierer



Der Übertrag ("Carry") der nächst-niedrigeren Stelle läuft durch ("to ripple" = plätschern) auf den Addierer der aktuellen Stelle

# Carry-Ripple-Addierer



Paralleladdierer

- Ist als Blockschaltbild wieder als neue Einheit auffassbar!
- Bei eingehendem Übertrag muss der Halbaddierer (rechts) durch Volladdierer ersetzt werden!

# Carry-Ripple-Addierer

## Carry-Ripple-Addierer

- **n-stellige** Binärzahlen addieren
- Stellen der ersten Zahl:  $e_{1i}, i = 1 \dots n$
- Stellen der zweiten Zahl:  $e_{2i}, i = 1 \dots n$
- $e_{xn}$ : msb
- $e_{x1}$ : lsb
- **Übertrag der jeweils niederwertigeren Stelle muss berücksichtigt werden!**
- $C_{i-1}$ : Übertrag der bei Addition an der Stelle  $i - 1$  auftrat und an Stelle  $i$  berücksichtigt werden muss



# Carry-Ripple-Addierer: Wahrheitstabelle

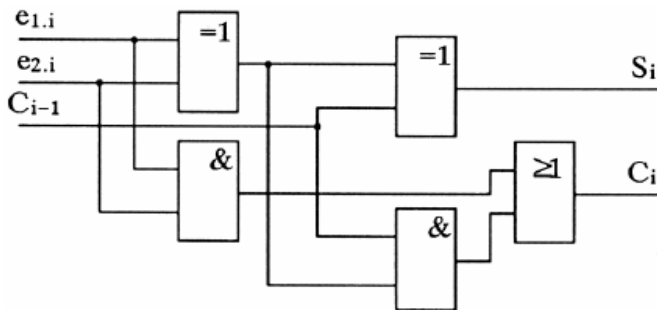
## Wahrheitstabelle eines Carry-Ripple-Addierers

$e_{1i}$	$e_{2i}$	$C_{i-1}$	$C_i$	$S_i$
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

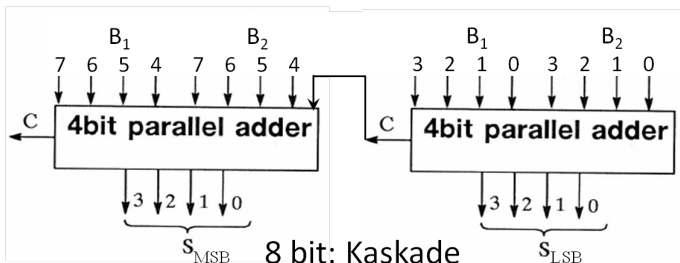
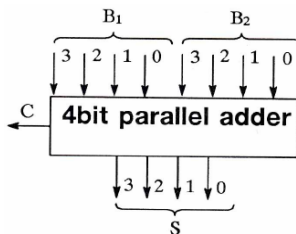
# Carry-Ripple-Addierer: Schaltung

## Minimiert

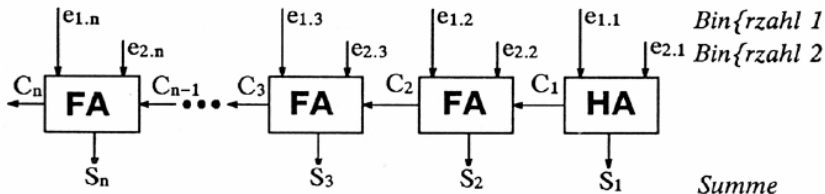
- $S_i = e_{1i} \oplus e_{2i} \oplus C_{i-1}$
- $C_i = (e_{1i} \wedge e_{2i}) \vee (C_{i-1} \wedge (e_{1i} \oplus e_{2i}))$



## Blockschaltbilder: Paralleladdierer (4bit)



# Carry-Ripple-Addierer



Paralleladdierer

- Jeder Addierer braucht für Berechnung Carry-Information von voriger Einheit.
- **Worst case: Carry-Bit muss die ganze Schaltung durchlaufen!** (*carry propagation*)
- Abschätzung: 3 FA + HA:  $3 \cdot 60\text{psec} + 30\text{psec} = 210\text{psec}$

# Optimierungen

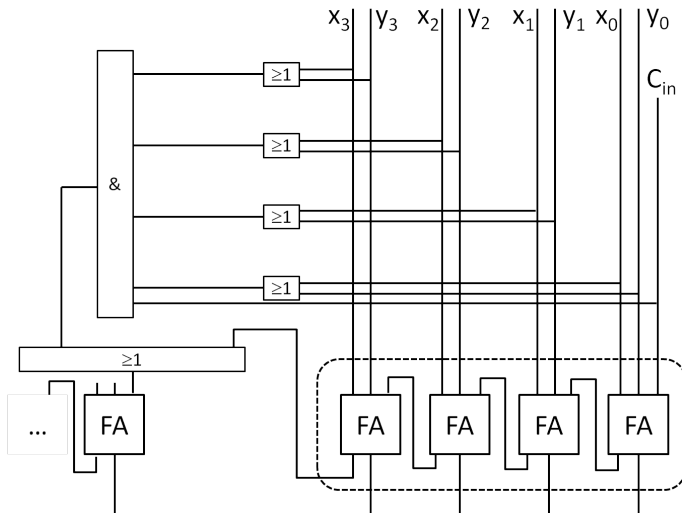
## Carry-Skip-Addierer

- Volladdierer werden gruppiert
- Zusatzlogik untersucht, ob sich ein Übertrag durch gesamte Gruppe propagiert
- Wenn ja, wird dies der nächsten Gruppe gemeldet, damit diese ebenfalls mit der Berechnung anfangen kann.

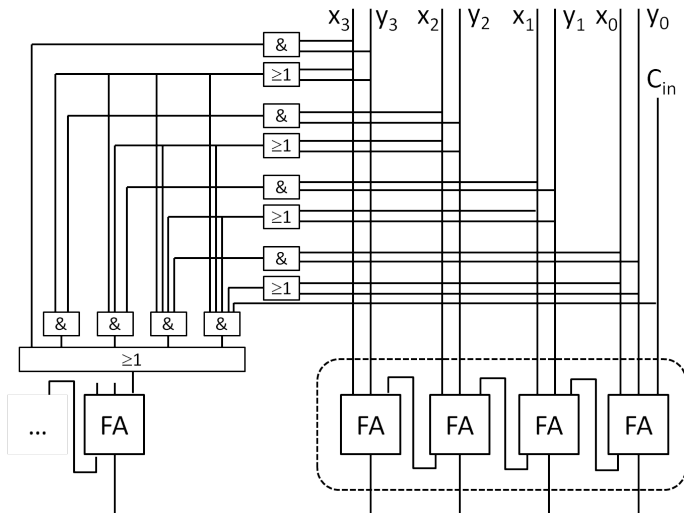
## Carry-Look-Ahead-Addierer

- Überträge werden bereits im ersten Additionsschritt ermittelt
- Größerer Schaltungsaufwand nötig
- Nur für kleine Wortbreiten effektiv

# Carry-Skip-Addierer



# Carry-Look-Ahead-Addierer



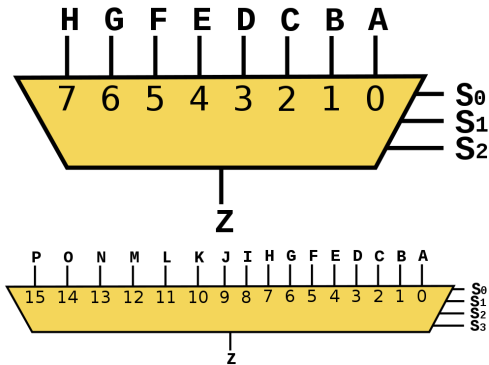
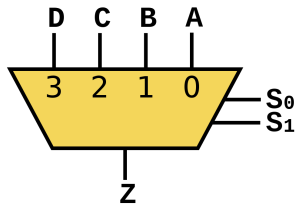
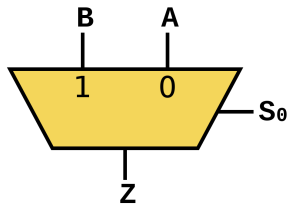
# Multiplexer (MUX)

## Multiplexer

- $2^n = m$  **Dateneingänge**
- $n = \log_2(m) = \text{ld}(m)$  **Steuereingänge**
- **Ein Datenausgang – Multiple input, single output!**
- **Der über die Steuereingänge gewählte Dateneingang wird unverändert zum Datenausgang geleitet**
- Umkehrung: **Demultiplexer**



# Multiplexer

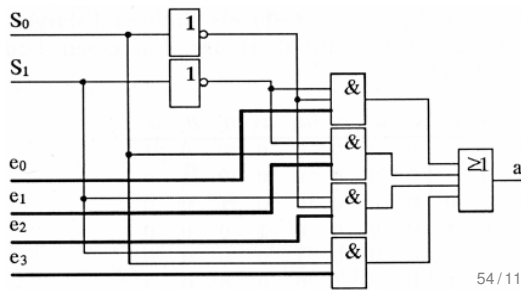


# Multiplexer: Beispiel

## Bsp: 4 to 1 MUX

- Angenommen:  $(S_1 S_0) = (10)$
- Nur das AND-Gatter mit  $e_2$  kann geschaltet sein
- $\Rightarrow e_2$  wird auf  $a$  durchgeschaltet

$S_1$	$S_0$	$a$
0	0	$e_0$
0	1	$e_1$
1	0	$e_2$
1	1	$e_3$



# Multiplexer: Anwendungen

## Allgemeines Prinzip

- **Parallelen Datenstrom in seriellen Datenstrom umwandeln**
- Per Demultiplexer wieder rückwandeln
- Anstatt vielen Leitungen zwischen zwei Einheiten
- D.h.: Multiplexed + Steuerleitungen < Leitungen

## I/O

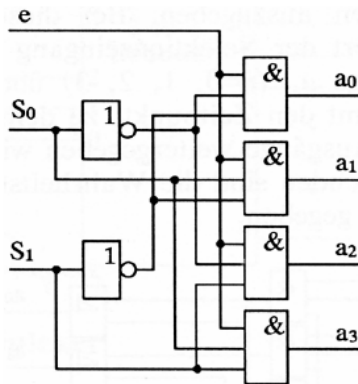
Z.B. Tastatur: Jede Taste wird per 7-bzw. 8-Bit codiert, bei einem Anschlag werden die Bits aber nicht parallel, sondern **seriell (hintereinander) über eine einzige Leitung übertragen.**

# Demultiplexer

## Prinzip

**Steuereingänge**  $S_i$   
bestimmen, auf  
welchen der  
**Ausgänge**  $a_i$  der  
**Eingang**  $e$   
durchgeschaltet wird

$S_1$	$S_0$	$a_0$	$a_1$	$a_2$	$a_3$
0	0	$e$	0	0	0
0	1	0	$e$	0	0
1	0	0	0	$e$	0
1	1	0	0	0	$e$



# Multiplexer: Anwendungen

## Adressmultiplexing

- Z.B.: 256MiB-Chip in 64Mx4 Organisation
- $64\text{M} = 2^{26}$ : 26 Adressleitungen, 4 Datenleitungen etc. nötig
- Mindestens 30 Anschlüsse nötig, Gehäuse unnötig groß
- $\Rightarrow$  **Adressmultiplexing**:
  - Zeilenadresse und Spaltenadresse werden nacheinander über **dieselben** Leitungen an den Chip gegeben
  - An zwei speziellen Leitungen wird signalisiert, **was** gerade übertragen wird
  - **RAS**, **Row** Address Strobe: signalisiert Zeilenadresse
  - **CAS**, **Column** Address Strobe: signalisiert Spaltenadresse

# Decoder

## Decoder

- **Multiple input, multiple output**
- Wandelt kodierten Input in kodierten Output um, wobei Input-Code und Output-Code unterschiedlich sind!
- $n$ -zu- $2^n$  Decoder:  $n$  Eingänge,  $2^n$  Ausgänge
- Zu jeder Zeit ist nur ein Ausgang aktiv
- Umkehrung: Encoder (Kodierer)

# Decoder: Wahrheitstabelle

$e_2$	$e_1$	$e_0$	$a_7$	$a_6$	$a_5$	$a_4$	$a_3$	$a_2$	$a_1$	$a_0$
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Abbildung: 3-zu-8 Decoder

# Decoder mit Enable-Eingang

## Enable-Eingang

- Nur wenn enabled, kann ein Ausgang überhaupt aktiv sein
- Bsp: 2-zu-4 Decodierer mit Enable

E	e <sub>1</sub>	e <sub>0</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



# Decoder

## Wichtige Anwendung: Random-access memory

- Adresse, die über Adress-Bus hereinkommt, wird in Zeilen- und Spalten-Adressen umgewandelt

## Simplex Beispiel

- Speicher<sup>a</sup> aus 8 Chips mit je 1MiByte<sup>b</sup>, Chip 0 = Adressen 0-1MiByte, Chip 1 = Adressen 1-2 MiByte, etc.
- 3 höchstwertigen Bits der Speicheradresse geben zu wählenden Chip an
- 3-zu-8 Logik Decoder

---

<sup>a</sup><http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/40-memories/40-ram/ram-decoder24.html>

<sup>b</sup>Präfix „Mi,, (sprich: mebi)  $2^{20} = 1048576$ ; IEEE 1541 Binärpräfixe

# Decoder

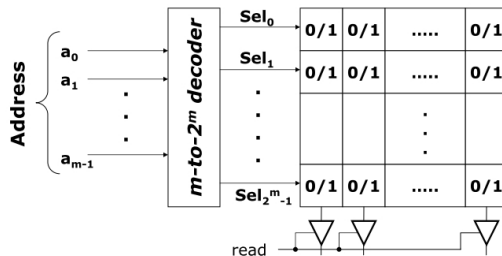


Abbildung: Dekodierung von Speicher Adressen

# Decoder

## Wichtige Anwendung: Instruction Decoder (CPU)

- Instruction Decoder (CPU)
- **Wandelt die Bits des Instruktionsregisters in Kontrollsignale um, die andere Teile der CPU steuern**
- Bei Microcode-basierenden CPUs wird die Mikroinstruktion umgewandelt (dekodiert)

## Simplex Beispiel

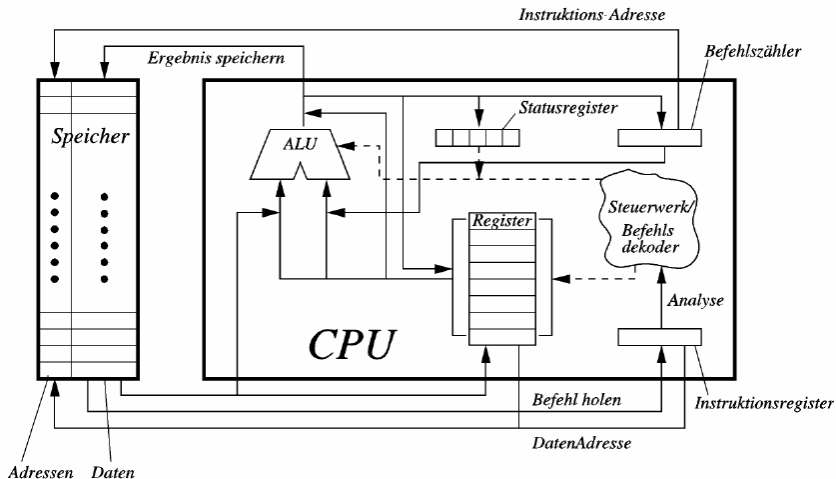
- CPU mit 8 Registern
- 3-zu-8 Logik Decoder
- 2 Quellregister als Input für die ALU
- 1 Zielregister als Output für die ALU

# ALU

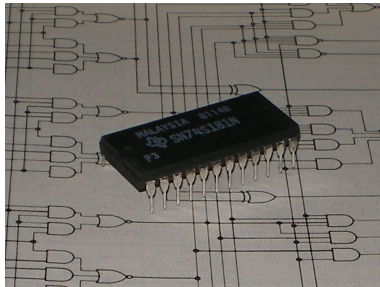
## ALU – Rechen-/Operationswerk

- Führt **logische und arithmetische** Operationen aus
- Wird vom Steuerwerk nach Dekodierung einer entspr. Instruktion angesprochen
- Verfügbare Operationen unterschiedlich, z.B.:
  - $a \wedge b$
  - $\neg b$
  - $a + b$
  - $a + 1$
  - $a - b$
  - bit shift left/right
  - ...
- Zum Ausprobieren: <http://tams-www.informatik.uni-hamburg.de/research/software/applets/hades/webdemos/20-arithmetic/50-74181/SN74181.html>

# CPU: Aufbau



## Ansteuerung: Beispiel 74181



- erste komplette ALU auf einem einzigen Chip (Ende 60er/Anfang 70er Jahre)
- 4-Bit ALU, 24 Pin DIP, 75 logische Gatter
- keine Multiplikation
- Rechengeschwindigkeit: ca.  $22ns$  pro Operation

# Ansteuerung: Beispiel 74181

Steuersignale				Ergebnis am Ausgang der ALU		
				M=H logische Operationen	M=L arithmetische Operationen	
$S_3$	$S_2$	$S_1$	$S_0$		$C_n = H$ (ohne Carry)	$C_n = L$ (mit Carry)
L	L	L	L	$\overline{A}$	$A$	$A + 1$
L	L	L	H	$\overline{A \vee B}$	$A \vee B$	$(A \vee B) + 1$
L	L	H	L	$\overline{A \wedge B}$	$A \vee \overline{B}$	$(A \vee \overline{B}) + 1$
L	L	H	H	0	-1	0
L	H	L	L	$\overline{A \wedge B}$	$A + A \wedge \overline{B}$	$A + A \wedge \overline{B} + 1$
L	H	L	H	$\overline{B}$	$(A \vee B) + A \wedge \overline{B}$	$(A \vee B) + A \wedge \overline{B} + 1$
L	H	H	L	$A \neq B$	$A - B - 1$	$A - B$
L	H	H	H	$A \wedge \overline{B}$	$A \wedge \overline{B} - 1$	$A \wedge \overline{B}$
H	L	L	L	$\overline{A \vee B}$	$A + A \wedge B$	$A + A \wedge B + 1$
H	L	L	H	$\overline{A \neq B}$	$A + B$	$A + B + 1$
H	L	H	L	$B$	$(A \vee \overline{B}) + A \wedge B$	$(A \vee \overline{B}) + A \wedge B + 1$
H	L	H	H	$A \wedge B$	$A \wedge B - 1$	$A \wedge B$
H	H	L	L	1	$A + (A \text{ shl } 1)$	$A + A + 1$
H	H	L	H	$A \vee \overline{B}$	$(A \vee B) + A$	$(A \vee B) + A + 1$
H	H	H	L	$A \vee B$	$A \vee \overline{B} + A$	$A \vee \overline{B} + A + 1$
H	H	H	H	$A$	$A - 1$	$A$

## Beispiel

### Register soll inkrementiert werden

- Steuerwerk bringt Registerinhalt an A-Eingang
- **ALU erhält folgende Steuersignale:**  $M = L$ ,  $\overline{C_n} = L$ ,  
 $S_3 = L$ ,  $S_2 = L$ ,  $S_1 = L$ ,  $S_0 = L$

### Register soll mit zweiten bitweise AND-verknüpft werden

- ...und Ergebnis wieder in Register gespeichert werden
- Steuerwerk bringt Inhalt beider Register an A- und B-Eingang
- **ALU erhält folgende Steuersignale:**
- $M = H$ ,  $S_3 = H$ ,  $S_2 = L$ ,  $S_1 = H$ ,  $S_0 = H$
- Ergebnis wird über internen Bus ins erste der beiden Register geschrieben

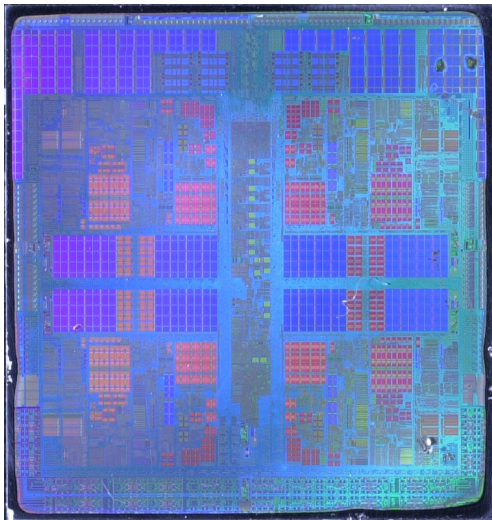


# ICs

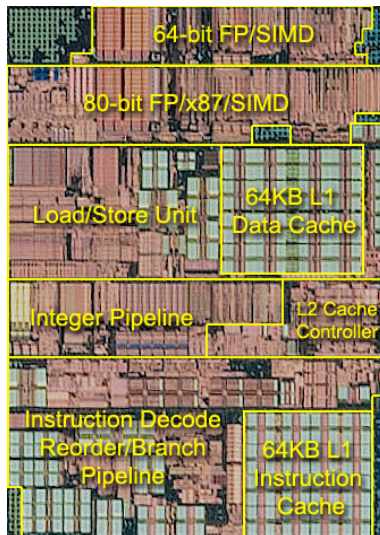
## ICs (Integrated Circuits, Chips)

- Gewisse Anzahl von Transistoren/Gattern auf einem Stück Silizium
- Pins stellen über Socket Verbindung nach außen her
- IC Skalen
  - SSI (Small Scale Integration): einige Transistoren
  - MSI (Medium Scale): hunderte Transistoren
  - LSI (Large Scale): Zehntausende Transistoren
  - VLSI (Very Large Scale): Hunderttausende Transistoren
  - ULSI (Ultra), SLSI (Super), ELSI (Extra), GLSI (Giant), ...
  - **Größenordnung heute: Milliarden von Transistoren!**

## AMD K10h (Quadcore) - “Barcelona” (2006)



## AMD K10h (Single core shot)



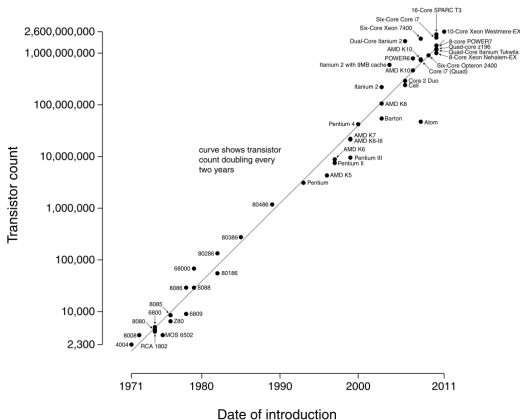
# Moore's Law

## Das Moore'sche Gesetz

- Gordon Moore (Gründer von Intel) 1965 (vor 50 Jahren!)
- verschiedene Formulierungen
- am bekanntesten: "Anzahl Schaltkreiskomponenten pro Chip verdoppelt sich ca. alle zwei Jahre"
- Resultat: exponentielles Wachstum (jährliche Wachstumsrate ca. 58%)
- heute z.T. auch selbsterfüllende Prophezeiung
- zum Vergleich: die jährliche Wachstumsrate "Anzahl Transistoren pro Designermonat" beträgt ca. 21% (ebenfalls exponentiell)

# Moore's Law

Microprocessor Transistor Counts 1971-2011 & Moore's Law



Quelle: [http://upload.wikimedia.org/wikipedia/commons/0/00/Transistor\\_Count\\_and\\_Moore%27s\\_Law\\_-\\_2011.svg](http://upload.wikimedia.org/wikipedia/commons/0/00/Transistor_Count_and_Moore%27s_Law_-_2011.svg)

# Probleme der Komplexität

## Probleme der Komplexität

- Gatter verhalten sich nicht instantan, sondern haben einen **gate delay**:
  - **Schaltzeit**
  - **Signalausbreitung**
- Erhöht sich mit höherer Betriebstemperatur
- Erhöht sich mit höherem Fan-out (Anzahl der angeschlossenen Gatter)
- **Thermische Probleme**: sehr viel Abwärme pro Fläche
- **Elektrische/quantenphysikalische Probleme**: Trennschichten oft nur mehr wenige Atome dick
- ...

# Überblick

1 Digitale Schaltungen

2 Gatterschaltungen

3 Speicher

Einleitung

Read Only Memory (ROM)

Random Access Memory (RAM)

# Einleitung

## Speicher

- **Halbleiterspeicher**
  - Festwertspeicher, Schreib-/Lese-Speicher
- Magnetische Speicher



# Halbleiter

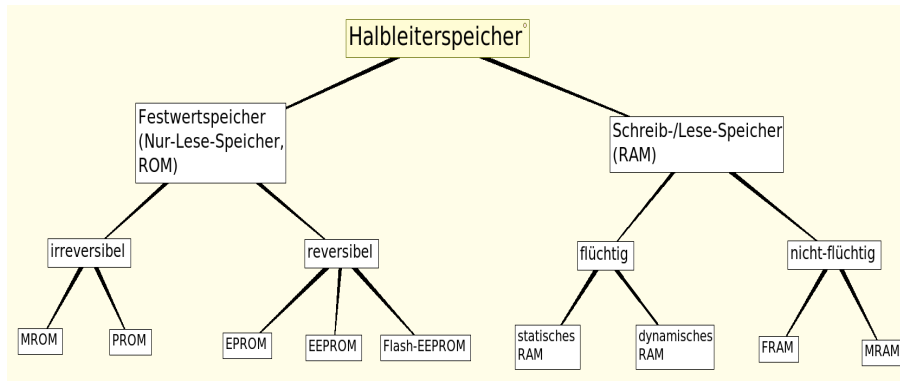
## Festwertspeicher, Nur-Lese-Speicher

- Read Only Memory (ROM)
- Kann im normalen Betrieb nur gelesen werden
- Nicht flüchtig, Dateninhalt bleibt auch ohne Versorgungsstrom erhalten

## Schreib-/Lese-Speicher

- Random Access Memory (RAM)
- Kann im normalen Betrieb beliebig beschrieben und gelesen werden
- Üblicherweise flüchtig, Dateninhalt geht nach Abschalten des Versorgungsstromes verloren

# Halbleiter – Speichertypen

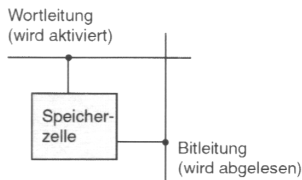


# Aufbau von Speicherbausteinen

## Aufbau

- **Gitterartiger Aufbau** (Matrix)
- **Waagrechte Wortleitungen:** Aktivieren alle Speicherzellen einer Zeile (→ Gates)
- **Senkrechte Bitleitungen:** Zum Lesen/Schreiben einer Speicherzelle (→ Drains)
- An Kreuzungspunkten sitzen Speicherzellen

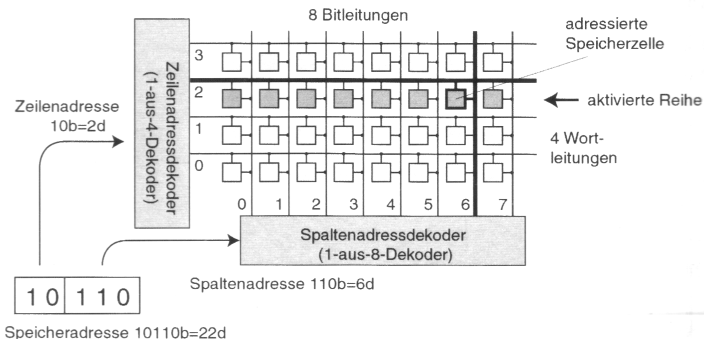
⇒ **1 Bit pro Speicherzelle**



# Aufbau von Speicherbausteinen

## Beispiel

- 32-Bit-Speicherbaustein (5 Adress-Bits)
- Speicherzelle mit Adresse 22 wird selektiert



# Zugriff

## Ablauf

- Speicheradresse wird geteilt (Zeilen-/Spalten-Adresse)
- Erster Teil geht an Zeilenadressdekoder (Wortleitung)
- ⇒ Alle Speicherzeilen dieser Wortleitung werden aktiviert
- Spaltenadressdekoder wählt **eine** Bitleitung aus
- ⇒ Nur Signal dieser Bitleitung wird gelesen/geschrieben

## Lese vs. Schreibzugriff

- Mittels zusätzlichem Signal “READ/WRITE” wird gesteuert, ob gelesen oder geschrieben werden soll
- ⇒ Typische Bezeichnung:  $R/\overline{W}$ ; HIGH:lesen, LOW:schreiben

# Schreib-Zugriff

## Beispiel: Einzelner Schreibzyklus

- CS - Eingang zur Bausteinaktivierung (Chip Select)

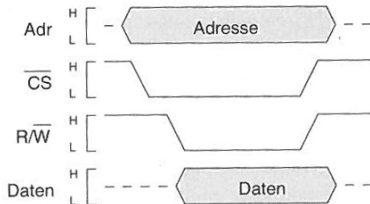


Abbildung 4.4: Typischer Ablauf eines Schreibzyklus. Die Daten werden dem Speicherbaustein schon früh im Zyklus zur Verfügung gestellt. Da es viele Adress- und Datenleitungen gibt, ist bei diesen immer durch eine Aufspreizung angedeutet, wann gültige Werte anliegen.  $\overline{CS}$  = Chip Select,  $R/\overline{W}$  = Read/Write.

# Schreib-Zugriff

## Beispiel: Zwei aufeinanderfolgende Lesezyklen

- Zugriffszeit: Zeitdifferenz zwischen Beginn des ersten Lesezyklus und der Bereitstellung der Daten
- Zykluszeit: Zeitdifferenz zwischen zwei Lesezugriffen

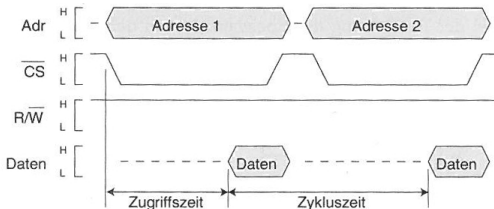


Abbildung 4.5: Typischer Ablauf zweier aufeinanderfolgender Lesezyklen. Der Speicher stellt die Daten gegen Ende der Zyklen zur Verfügung. Die Zykluszeit ist hier deutlich größer als die Zugriffszeit.

# Organisation von Speicherbausteinen

## Organisation

- Bezeichnet Anzahl der Speicheradressen die unter jeder Speicheradresse gespeichert wird (*auch Speichertiefe genannt*)

## Beispiel: 4kx1

- Baustein mit 4096 Speicherzellen mit je 1 Bit
- 12 Adressleitungen ( $2^{12}$ ), eine Datenleitung, Gesamtkapazität 4KiBit



# Organisation von Speicherbausteinen

## 8kx4

- Zellenmatrix von 8192 Speicherzellen vierfach vorhanden
- 13 Adressleitungen ( $2^{13}$ ), 4 Datenleitungen, Gesamtkapazität 32KiBit ( $2^{13} * 4 = 2^{15}$ )

## Speicher von Mikroprozessorsystemen

- Meist Byte- oder Wort-strukturiert  $\Rightarrow$  Kein einzelnes Bit ansprechbar, sondern immer nur Gruppen (8, 16, 32 Bit)
  - Um diese Strukturierung zu erreichen, müssen je nach Organisation mehrere Speicherbausteine parallel geschaltet werden
- $\Rightarrow$  z.B. für Byte-Struktur: ein Nx8 Speicherbaustein, oder zwei Nx4 Speicherbausteine in parallel

# Read Only Memory (ROM)

## Festwertspeicher, Nur-Lese-Speicher

- Kann im normalen Betrieb nur gelesen werden
- **Nicht-flüchtig** (non-volatile), Inhalt bleibt erhalten
- Kommen zum Einsatz um **Daten dauerhaft und unabänderlich zu speichern**
- BIOS, Messgeräte, Unterhaltungselektronik, Steuerungen

## Arten von ROMs

- MROM: Masken-ROM
- PROM: Programmable ROM
- EPROM: Erasable PROM
- EEPROM (Electrical EPROM) bzw. Flash-Speicher

└ Speicher

└ Read Only Memory (ROM)

# MROM

## Masken ROM

- **Schon bei Herstellung des Speicherchips** werden an Kreuzungspunkten zw. Wort-/Bit-Leitung Brücken erstellt
- ⇒ **Brücke vorhanden: 1; Brücke fehlt: 0**
- Brücke: Diode oder Transistor
  - Dateninhalt muss schon **vor** der Herstellung feststehen und wird in Belichtungsmaske eingearbeitet
  - Belichtungsmaske **teuer**, lohnt sich nur in großen Stückzahlen



Diode



Bipolartransistor



Feldeffekttransistor

Abbildung 4.6: In den Speicherzellen von MROMs werden verschiedene Arten von Brücken zwischen Wortleitung und Bitleitung verwendet.

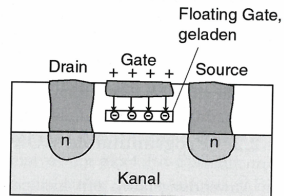
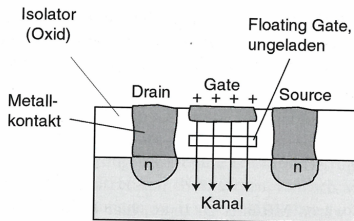
# PROM

## Programmable ROM

- Speicherzellen so gebaut, dass mit einem Programmierimpuls die Brückenfunktion **dauerhaft** hergestellt/unterbrochen wird
  - Programmierimpuls: Einmaliger, kräftiger Stromstoß
  - Im Originalzustand alle Verbindungen aufrecht
  - Gezieltes Zerstören der Verbindungen  $\Rightarrow$  **0/1**
  - **Lässt sich nicht mehr rückgängig machen!**
- $\Rightarrow$  "One Time Programmable ROM" (OTPROM)
- Bei kleinen Stückzahlen, billiger als MROMs

# PROM - Floating Gate

## Floating Gate



K. Wüst, Mikroprozessortechnik, 4. Auflage

# EPROM

## Erasable Programmable ROM

- **Unpraktisch jedes Mal PROM zu opfern, z.B. Testphase**
- **EPROM: Per Bestrahlung mit UV-Licht löscher**
- Elektronen in Transistoren<sup>a</sup> werden durch Licht-Quanten in ursprüngliche Position gebracht
- Löschen dauert ca. 20min (100 bis 200 Mal, danach kaputt)
- Quarzfenster (teuer)
- Auch Tageslicht enthält UV-Anteile, gute Abdeckung nötig
- **Etwas umständliche Löscherprozedur ⇒ EEPROM**

---

<sup>a</sup>*Floating Gate*

# EEPROM

## Electrical Erasable Programmable ROM

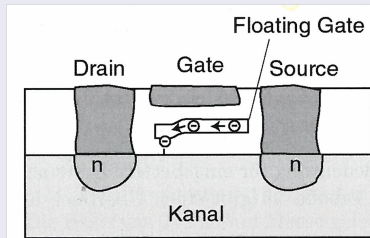
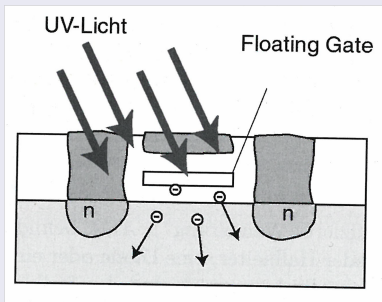
- Floating Gate des Transistors kann **elektrisch** entladen (und damit neu programmiert) werden
- Anzahl der Schreibzyklen ist begrenzt: ca.  $10^4$  bis  $10^6$  Zyklen

## NVRAM (non-volatile RAM)

- Alle Speicherzellen doppelt vorhanden: Als RAM- und EEPROM-Zelle
- RAM für normalen Betrieb
- Vor Abschalten der Betriebsspannung in EEPROM sichern
- $\Rightarrow$  Nicht-flüchtiges RAM

# EPROM - EEPROM

## Entladungsmechanismen EPROM vs EEPROM





# Flash-Speicher

## Flash-Speicher

- **Spezielle EEPROM-Variante**
  - Dünneres Tunneloxid
  - Löschen/Beschreiben der Zellen erfordert geringere Spannungen/Ströme
- ⇒ **Nur ganze Blöcke von Zellen löschar/beschreibbar**
- Arbeitet ähnlich wie RAM-Baustein, aber **nicht-flüchtig**
  - CF, MMC, SD, SSD, USB-Memory-Stick, ...

# Eigenschaften

## Eigenschaften von ROM Bausteinen

Typ	Dauer Schreibvorgang	max. # Schreibvorg.
MROM	Monate	1
(OT)PROM	Minuten	1
EPROM	Minuten	bis zu 100
EEPROM	Millisekunden	$10^4 - 10^6$
Flash	$10 \mu s$ ( $1 \mu s = 10^{-6}$ Sek)	$10^4 - 10^6$

# Random Access Memory

## Schreib-/Lese-Speicher

- Für Register, Akkumulatoren, Caches, Hauptspeicher, ...
- **Random Access:** Es kann in beliebiger Reihenfolge zugegriffen werden, in theoretisch gleicher Zeit (vgl. Bandlaufwerk oder Disk)
- **Sind aus einzelnen Speicherzellen aufgebaut, die gruppiert werden**
- Verschiedene Arten der Organisation möglich

## Arten von flüchtigen RAMs

- SRAM: Statisches RAM
- DRAM: Dynamisches RAM

# Statisches RAM

## Statisches RAM

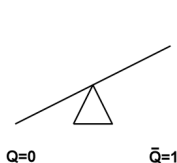
- **Schnelle Lesezugriffe und Umschaltzeiten**
- **Kein Refresh nötig, dennoch flüchtig**
- **Teurer, größer als DRAM**
- Für Register, Akkumulatoren und Caches verwendet
- SRAM-Zelle besteht meist aus 6 Transistoren
- Funktionsprinzip wie **taktgesteuerte D-FlipFlops**

# Bistabiler Schaltkreis

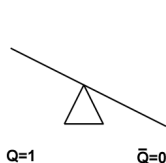
## Bistabiler Schaltkreis - Bistabile Kippstufe

- **Zwei mögliche stabile Zustände**
  - In stabilen Zustand gebracht, verbleibt Schaltkreis darin
- ⇒ **Schaltkreis hat „Gedächtnis“**
- Realisierung: z.B. mit zwei Inverter oder zwei NANDs/NORs, deren Ausgänge an den Input des jeweils anderen rückgekoppelt ist

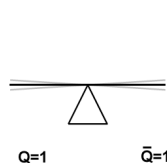
Stabiler Zustand 1



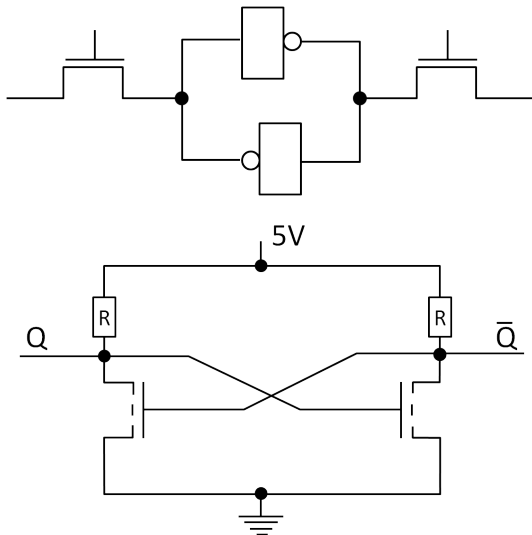
Stabiler Zustand 2



Metastabiler Zustand



## Bistabiler Schaltkreis - Realisierung



# Statisches RAM

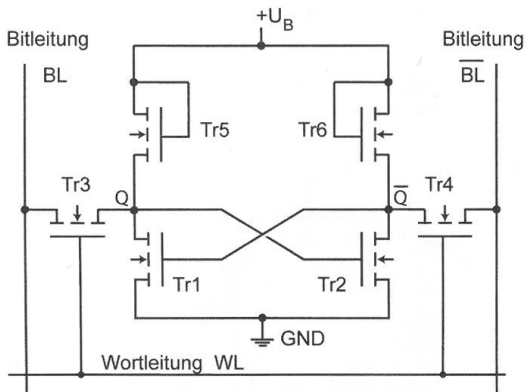


Abbildung 4.13: Das Flipflop aus zwei NMOS-Transistoren ist der Kern der SRAM-Speicherzelle. Die Wortleitung schaltet über zwei weitere Transistoren die Ein-/Ausgänge auf die Bitleitungen  $BL$  und  $\overline{BL}$  durch. Über diese erfolgt das Schreiben und Lesen von Daten.

# SR-Latch (**S**et / **R**eset)

## Unterschiedliche Bezeichnungen

- Auch RS-Latch oder RS-Flipflop bezeichnet
- ⇒ Vorsicht: Flipflop eigentlich taktflankengesteuert
- Stellt die einfachste Art eines Flipflops dar
- **Pegel- bzw. zustandsgesteuert, und nicht taktgesteuert** (da kein Taktsignal vorhanden)
- ⇒ Taktsteuerung mittels Zusatzschaltungen möglich

## Aufbau

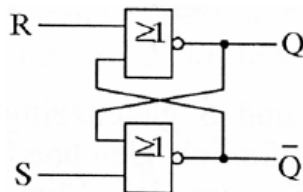
- Üblicherweise aus 2 NOR Gattern aufgebaut
- Kann auch aus zwei NAND Gattern und (üblicherweise) negierten Eingängen  $\overline{S}$ ,  $\overline{R}$  aufgebaut werden!



# SR-Latch (**S**et / **R**eset) aus NOR Gattern

## Grundsaltung (kein Strom)

- $R = 0, S = 0$
- Ausgang ( $Q$ ) und negierter Ausgang ( $\bar{Q}$ )



<http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/16-flipflops/10-srff/srff.html>

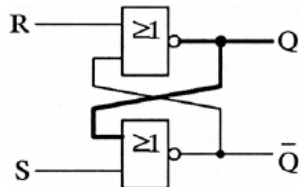
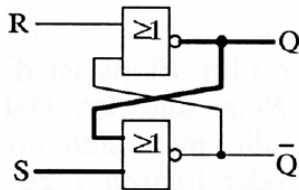
## SR-Latch (**S**et / **R**eset) aus NOR Gattern

“Set” –  $R = 0, S = 1$

- Setze S auf 1
- ⇒ Dicke Linien: Logisch 1
- Ausgänge ändern sich
- ⇒ Q auf 1 gesetzt

$R = 0, S = 0$

- Liegt nachher an beiden Eingängen 0 an, bleibt der Zustand unverändert
- ⇒ Stabiler Zustand



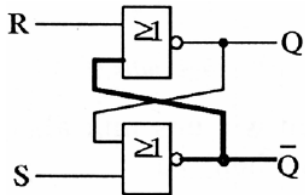
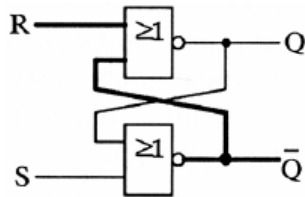
## SR-Latch (**S**et / **R**eset) aus NOR Gattern

“Reset” –  $R = 1, S = 0$

- Setze R auf 1
  - Ausgänge ändern sich
- ⇒ Q auf 0 gesetzt

$R = 0, S = 0$

- Liegt nachher an beiden Eingängen 0 an, bleibt der Zustand unverändert
- ⇒ Stabiler Zustand



# SR-Latch (**S**et / **R**eset) aus NOR Gattern

## Wahrheitstabelle

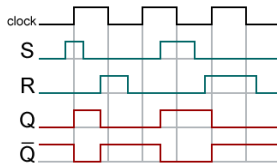
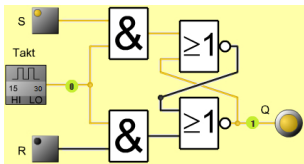
$S$	$R$	$Q$	$\overline{Q}$	Bemerkung
0	0	$Q$	$\overline{Q}$	Speichern des Zustandes
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	potentielle race condition wenn $R=S=0$ folgt <sup>a</sup>

<sup>a</sup>Dieser Eingangszustand wird in der Literatur oft als instabil bezeichnet (was eigentlich nicht stimmt), bzw. als “verbotener Zustand”, da er einen logischen Widerspruch ergibt ( $Q = \overline{Q}$ )

# Getaktetes ("clocked") SR-Latch

## Getaktetes SR-Latch

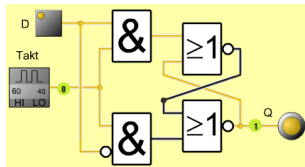
- Latch kann Zustand nur wechseln wenn Takt/Clock-Eingang aktiv (logisch 1)
  - Kann mehrmals während aktiver Taktphase den Zustand wechseln!
- ⇒ **Latch ist taktzustands-(taktpegel-)gesteuert**
- Zustand  $R = S = 1$ : weiterhin potentielle race condition



# D-Latch

## Data(Delay)-Latch

- R-Eingang wird durch das invertierte S-Signal definiert
- ⇒  $R = S = 1$  nicht mehr möglich
- Der Wert des Daten-Eingangs wird an den Datenausgang durchgeschleift so lange **Takt 1** ist. (**Taktzustand!**)
  - Geht **Takt auf 0**, wird letzter Wert des Eingangs “eingesperrt” (*latch*)



# D-FlipFlop

## Date(Delay)-FlipFlop

- Hat wie das D-Latch einen Daten-Eingang und einen Clock/Takt-Eingang
- Übernimmt den Eingangswert NUR zu dem Zeitpunkt, zu dem Takteingang von Low auf High wechselt

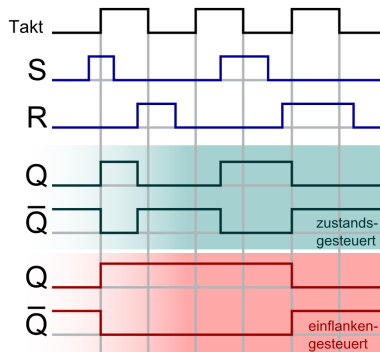
⇒ **FlipFlop ist taktflankengesteuert**

- Wird über Kaskade von zwei D-Latches realisiert

## Notation

- In der (deutschen) Literatur wird das D-Latch manchmal als **taktzustandsgesteuertes** D-Flipflop bezeichnet

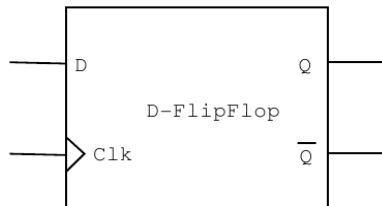
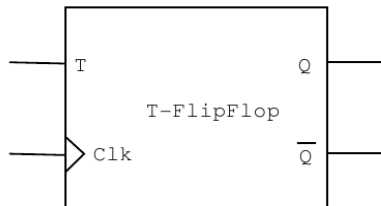
# Taktzustands- vs. taktflankengesteuert



<http://de.wikipedia.org/wiki/Flipflop>



# Speichersymbole



# DRAM

## Dynamic RAM

- Kleiner und billiger als SRAM  $\Rightarrow$  Hauptspeicher
- DRAM-Zelle besteht aus Kondensator und Transistor
- Kondensator = Ladungsspeicher = 0/1
- (Tor-)Transistor zum Ansteuern
- **Regelmäßiger Refresh nötig, sonst Datenverlust**
- Typische Refreshzeiten: alle 32ms oder 64ms (d.h. hält nur für Sekundenbruchteile!)
- In Array angeordnet - immer ganze Zeilen aktiviert!

# DRAM

## DRAM-Zelle

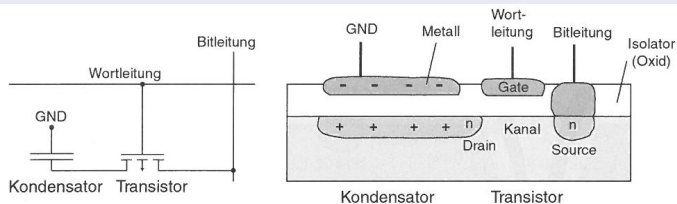
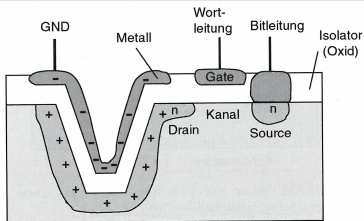


Abbildung 4.14: Die Zelle eines DRAM besteht aus einem Kondensator und einem Tortransistor. Links Schaltbild, rechts Schichtenaufbau.

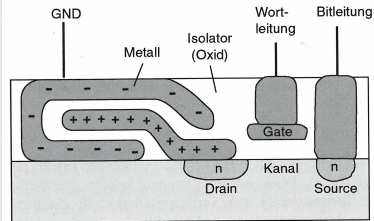
# DRAM

## Graben- vs Stapelkondensator



Grabenkondensator

Transistor



Kondensator

Transistor