

06 – Digitalschaltungen, Kombinatorik und Sequenzielle Logik

Grundzüge digitaler Systeme

Vortrag von: Jürgen Kogler

- Abbildung Boolescher Algebra auf physikalische Systeme
- Gatter und Gatterschaltungen
- Standardbaugruppen in der kombinatorischen Digitaltechnik (Multiplexer, Addierer, Decoder, ...)
- Sequenzielle Logik (Takt und Register)

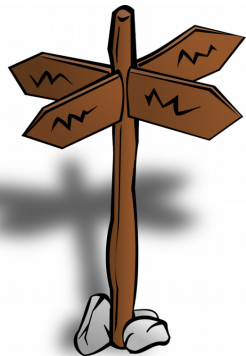


Abbildung Boolescher Algebra auf physikalische Systeme

Digitale Systeme

- Umsetzung der Boole'schen Algebra in einem System
- Diskretisierung einer kontinuierlichen Wertemenge (Kontinuum) → Abbilden von „0“ und „1“ auf den Wertebereich einer **physikalischen Größe**
 - Elektrische Potentialdifferenz, Position eines Stabs, (Luft)Druck, ...
 - **Variable** → **Signal**
- Verknüpfen der physikalischen Signale entsprechend der Boole'schen Funktion durch **Gatter**
 - **Operator** → **Gatter**
- Beispiele:

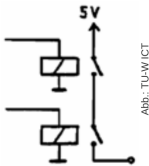
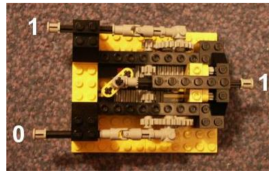


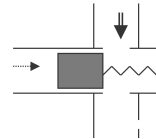
Abb.: TU-W/ICT

AND



<http://www.goldfish.org.uk/logic.html>

OR



NOT

Abbildung analoger Werte auf Boole'sche Zustände

- Zuordnung von „0“ und „1“ zu bestimmten physikalischen Wertebereichen ist ausschließlich Konvention
- HIGH bezeichnet immer das höhere Spannungspegelintervall
- LOW bezeichnet immer das niedrigere Spannungspegelintervall
- Logische Werte („0“/„1“) können beliebig zu HIGH/LOW zugeordnet werden: **Positive** oder **negative** Logik

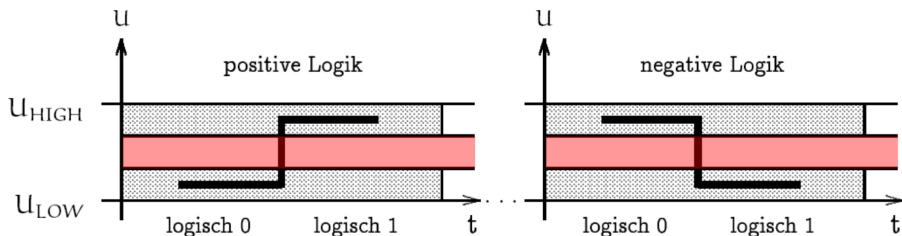
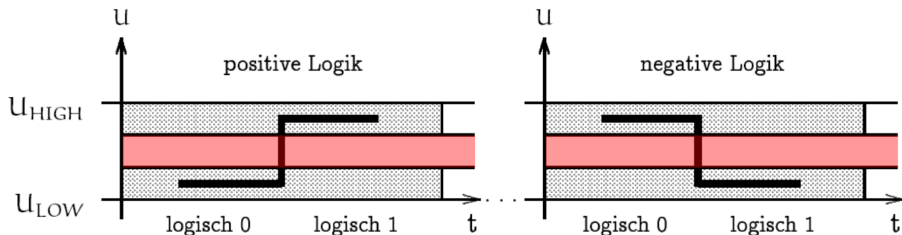


Abbildung analoger Werte auf Boole'sche Zustände

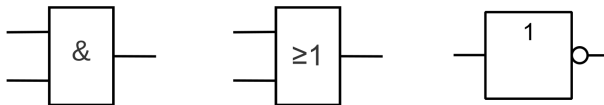
- Zuordnung von „0“ und „1“ zu bestimmten physikalischen Wertebereichen ist ausschließlich Konvention
- HIGH bezeichnet immer das höhere Spannungspegelintervall
- LOW bezeichnet immer das niedrigere Spannungspegelintervall
- Logische Werte („0“/„1“) können beliebig zu HIGH/LOW zugeordnet werden: **Positive** oder **negative** Logik



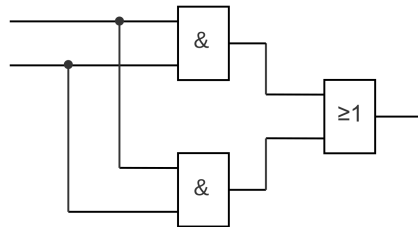
- Deutlicher Abstand zwischen gültigen Bereichen nötig, um Störungen ausgleichen zu können: **Verbotene Zone**

Gatter (gates)

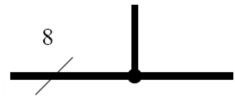
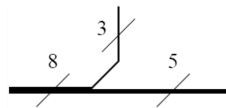
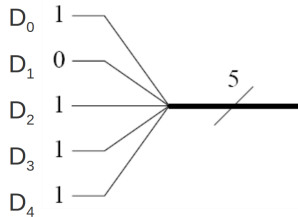
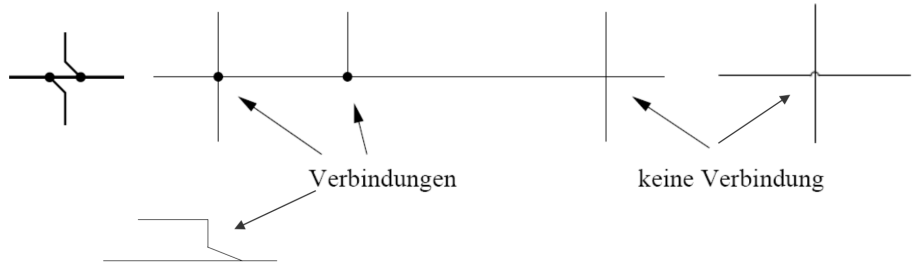
- Elementare Bauelemente der Digitaltechnik
- Implementiert durch elektronische Bauelemente
- Realisieren Boole'sche Operationen
 - Grundoperationen: Und (AND), Oder (OR), Nicht (NOT)



- Erweiterte Operationen: NAND, NOR, ...
- Realisierung von komplexen Boole'schen Funktionen durch Zusammenschaltung von Gattern

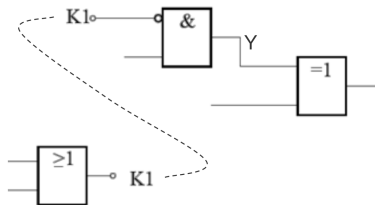


Signalverbindungen, Signalbündel (Busse)

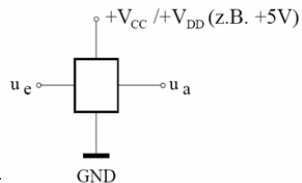


Schaltplan

- Namensgleiche Signale sind implizit verbunden

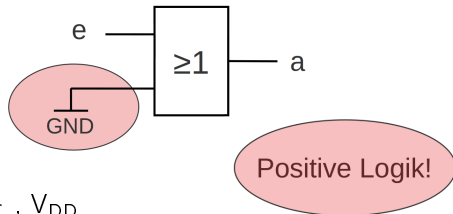
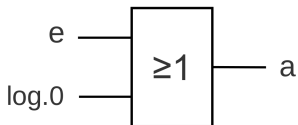


- Gilt auch für Versorgungsanschlüsse
 - V_{CC}/V_{DD} : positive Versorgungsspannung
 - GND: Masse / 0V
 - Werden der Übersichtlichkeit halber meist weggelassen.

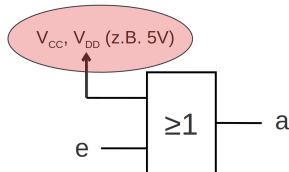
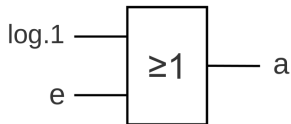


Schaltplan

- Fixe logische Zustände können auch durch Spannungspegel definiert werden
 - Logisch 0 \leftrightarrow Masse, Ground, GND, 0V



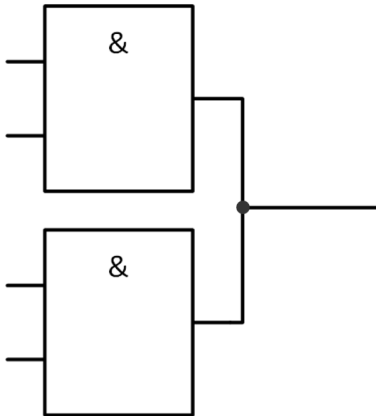
- Logisch 1 \leftrightarrow Versorgungsspannung, V_{CC} , V_{DD}



→ Eingänge bleiben nie offen (immer definiertes Signal)

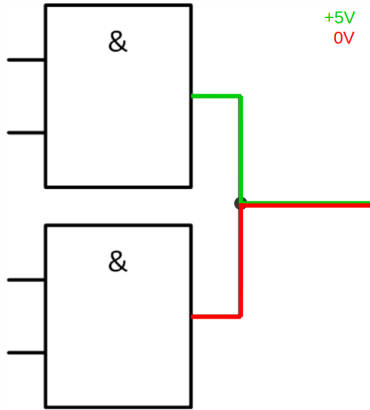
Gemeinsam genutzte Signalwege

- Zusammenschalten mehrerer Ausgänge?



Gemeinsam genutzte Signalwege

- Zusammenschalten mehrerer Ausgänge? Nein!

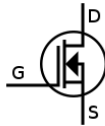


Warum ist das so?

Transistoren

(nicht prüfungsrelevant)

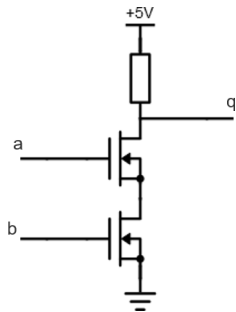
- Widerstand ist elektrisch steuerbar (engl. **transfer resistor**)
- Zum Verstärken und Schalten von Signalen
- Abstrahiert als „elektronischer Schalter“



Boole'sche Funktionen mit Schaltern

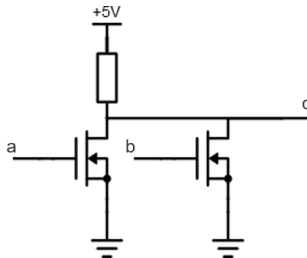
(nicht prüfungsrelevant)

■ NAND: $q = \neg(a \wedge b)$



a	b	$\neg(a \wedge b)$
0	0	1
0	1	1
1	0	1
1	1	0

■ NOR: $q = \neg(a \vee b)$

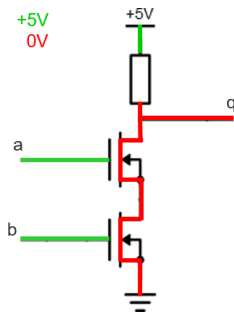


a	b	$\neg(a \vee b)$
0	0	1
0	1	0
1	0	0
1	1	0

Boole'sche Funktionen mit Schaltern

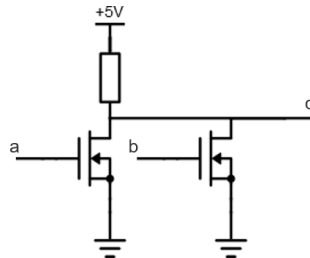
(nicht prüfungsrelevant)

■ NAND: $q = \neg(a \wedge b)$



a	b	$\neg(a \wedge b)$
0	0	1
0	1	1
1	0	1
1	1	0

■ NOR: $q = \neg(a \vee b)$

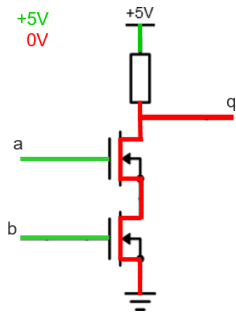


a	b	$\neg(a \vee b)$
0	0	1
0	1	0
1	0	0
1	1	0

Boole'sche Funktionen mit Schaltern

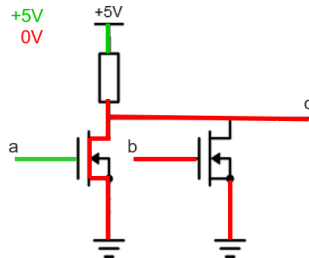
(nicht prüfungsrelevant)

■ NAND: $q = \neg(a \wedge b)$



a	b	$\neg(a \wedge b)$
0	0	1
0	1	1
1	0	1
1	1	0

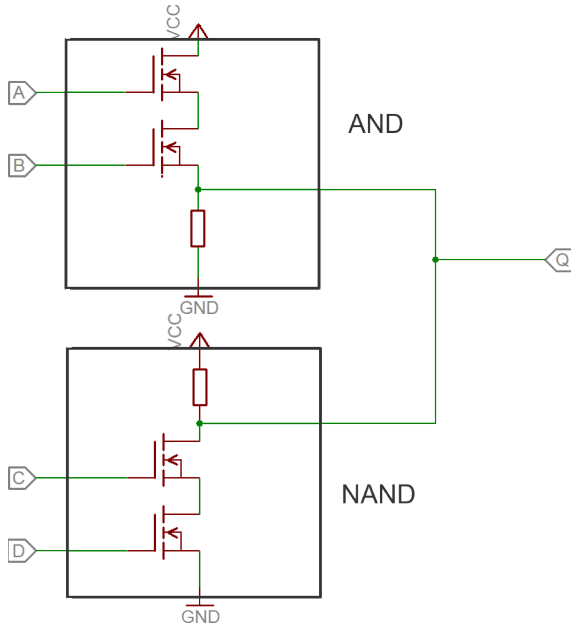
■ NOR: $q = \neg(a \vee b)$



a	b	$\neg(a \vee b)$
0	0	1
0	1	0
1	0	0
1	1	0

Zusammenschalten mehrerer Ausgänge

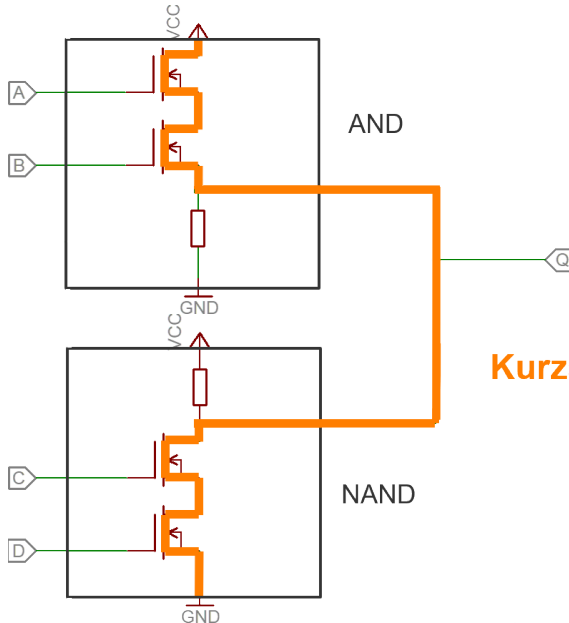
(nicht prüfungsrelevant)



a	b	c	d	q
...
1	1	1	1	-
...

Zusammenschalten mehrerer Ausgänge

(nicht prüfungsrelevant)



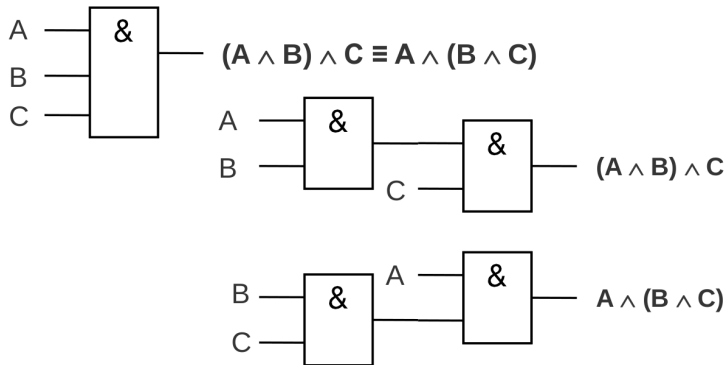
a	b	c	d	q
...
1	1	1	1	-
...

Kurzschluss

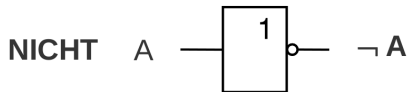
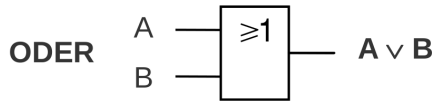
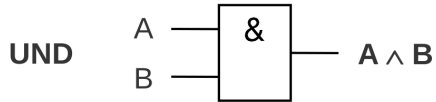
Gatter und Gatterschaltungen

Gatter vs. Boole'sche Operatoren

- Boole'sche Operatoren
 - Einstellig: Negation \neg
 - Zweistellig: Oder \vee , Und \wedge
- Gatter
 - Mehrere Eingänge möglich



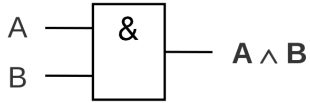
IEC 60617-12



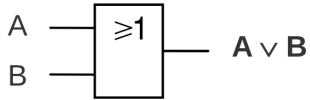
Grundgatter / Schaltzeichen

IEC 60617-12

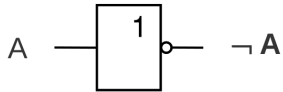
UND



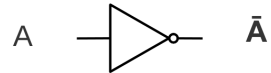
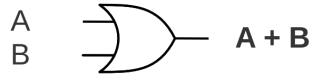
ODER



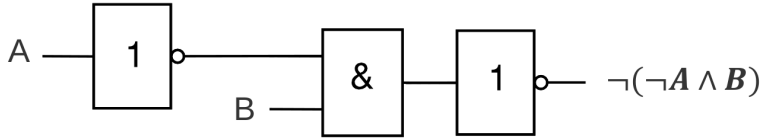
NICHT



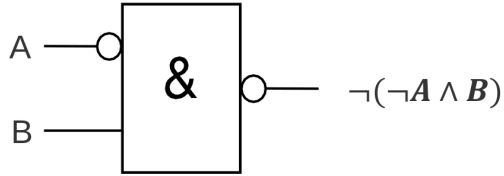
US ANSI 91-1984



Negationsring



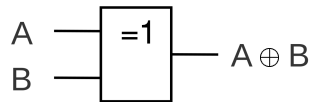
\equiv



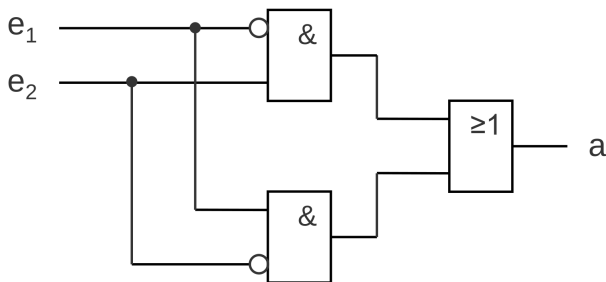
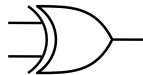
Beispiel / XOR als Gatterschaltung

■ Boolescher Ausdruck:

$$(\neg e_1 \wedge e_2) \vee (e_1 \wedge \neg e_2) = e_1 \oplus e_2 = a$$



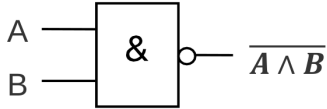
■ Gatterschaltung:



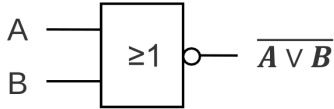
NAND und NOR

IEC 60617-12

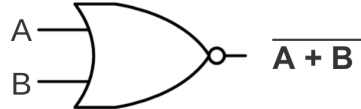
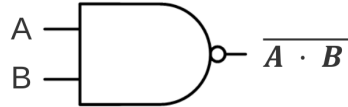
NAND



NOR



US ANSI 91-1984



Warum NAND/NOR?

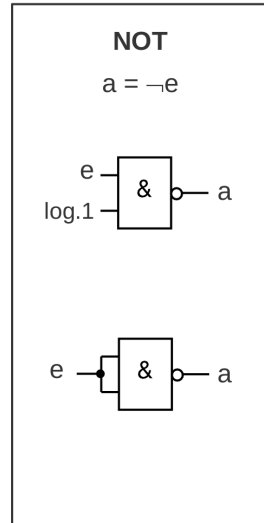
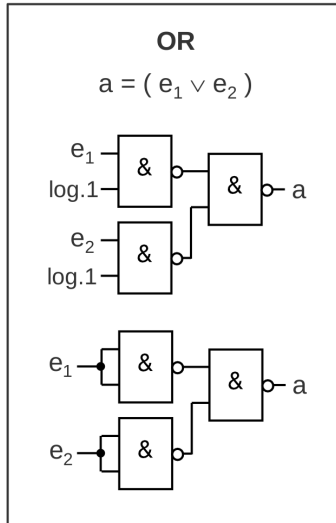
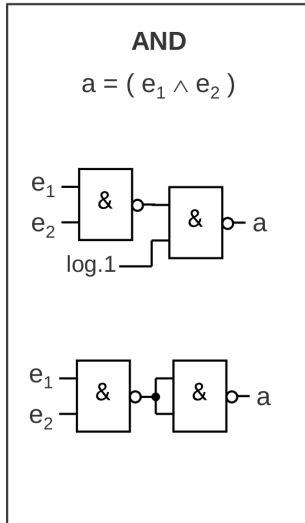
- Weniger Transistoren → Weniger Chipfläche → Weniger Kosten

Gatter	#Eingänge	Transistoren
Inverter	1	2
NAND / NOR	2	4
XOR	2	5
AND	2	6
OR	2	6

- Eine Menge an Boole'schen Operatoren heisst *funktional vollständig*, wenn damit alle anderen Boole'schen Operatoren ausgedrückt werden können.
 - Z.B.: $\{\wedge, \neg\}$, $\{\text{NAND}\}$, $\{\text{NOR}\}$, ...
- NAND / NOR effizient realisierbar mit Transistoren
- **Substitution:** Umformung von Gatterfunktionen mit dem Ziel geringeren Konstruktionsaufwand zu erreichen
 - Heutige Hardware meist aus NANDs / NORs

Substitution / NAND

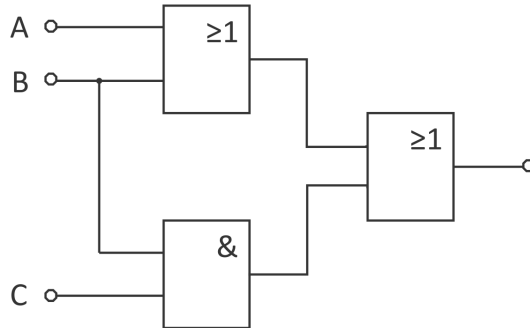
- Realisierung jeder Funktion nur mit NAND (oder NOR) möglich



Beispiel / Transformation

- Die Funktion $f(A, B, C) = (A \vee B) \vee (B \wedge C)$ soll **nur aus NAND-Gattern** aufgebaut werden.

- Gatterschaltung:

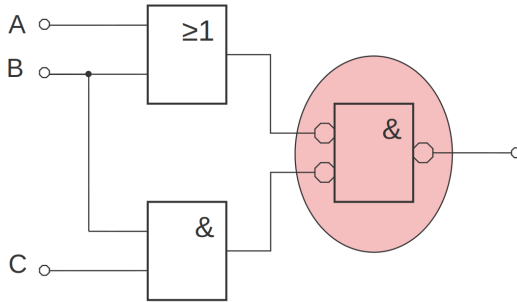


Beispiel / Transformation

- Schrittweises Ersetzen der einzelnen Gatter durch NANDs

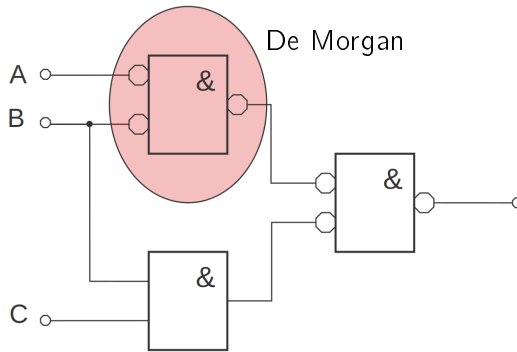
$$\neg\neg(A \vee B) \equiv \neg(\neg A \wedge \neg B)$$

De Morgan'sches Gesetz

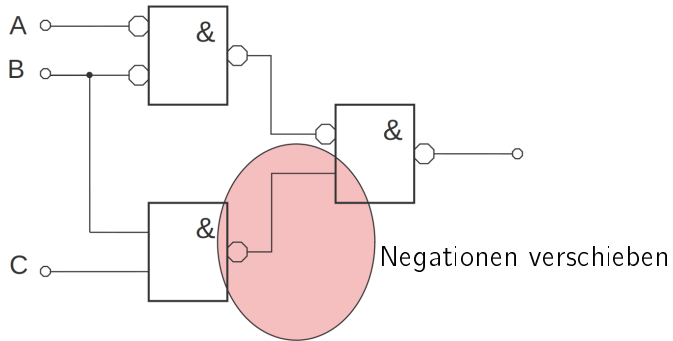


Beispiel / Transformation

- Schrittweises Ersetzen der einzelnen Gatter durch NANDs



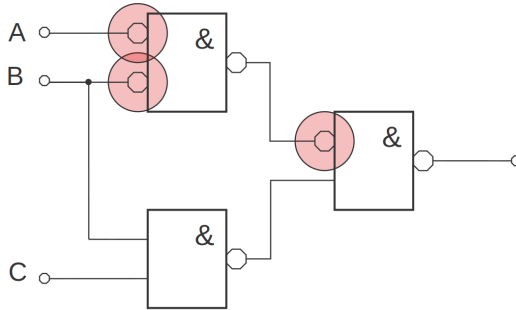
Beispiel / Transformation



Beispiel / Transformation

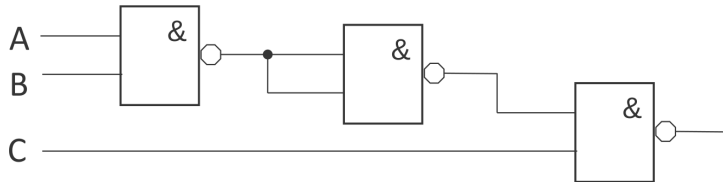
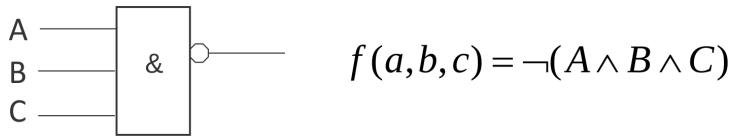
! Wir wollen **ausschließlich** NAND-Gatter in der Schaltung

- Alle drei Gatter sind zwar NAND-Gatter, aber Negationen bei den Eingängen müssen noch ersetzt werden
 - „Negationskreise“ sind verkürzte Darstellung von **Negationsgattern**



Beispiel / NAND

Beispiel: NAND-Gatter (3 Eingänge) \rightarrow NAND-Gatter (2 Eingänge)

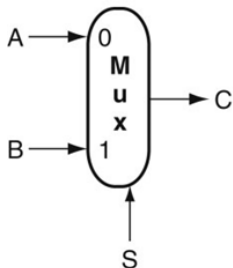


Standardbaugruppen in der kombinatorischen Digitaltechnik

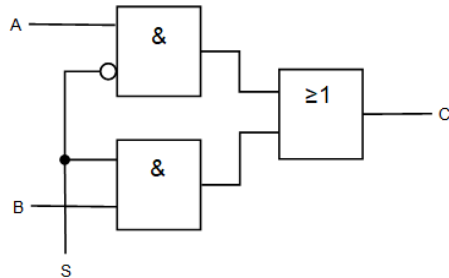
Multiplexer (MUX)

- Durchschalten eines gewählten Eingangs auf den Ausgang
- Steuersignal bestimmt welcher Eingang durchgeschaltet wird

(2 zu 1) - Multiplexer



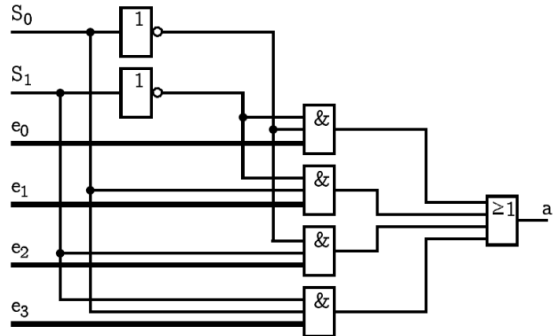
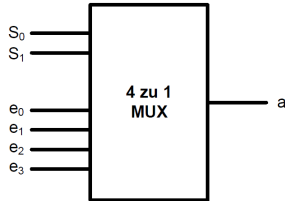
S	C
0	A
1	B



(4 zu 1)-Multiplexer (MUX)

- Durchschalten eines gewählten Eingangs auf den Ausgang
- Steuersignal bestimmt welcher Eingang durchgeschaltet wird

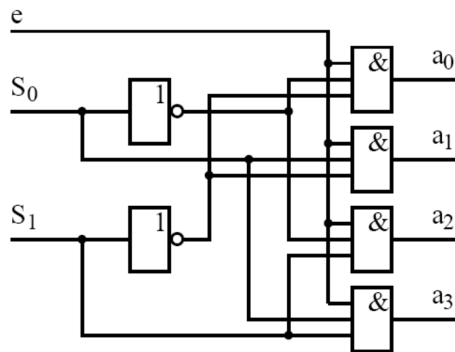
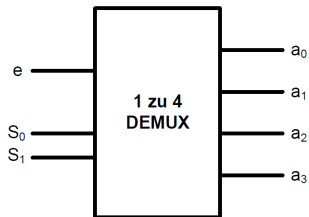
S_1	S_0	a
0	0	e_0
0	1	e_1
1	0	e_2
1	1	e_3



(1 zu 4)-Demultiplexer (DEMUX)

- Durchschalten des Eingangssignals auf einen ausgewählten Ausgang.

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

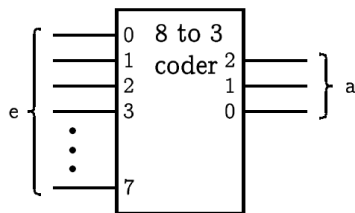


Codierer vs. Decodierer

- Dienen der Umwandlung von Codes
- Encoder (Codierer) verdichtet Information
 - Eingangscodelänge $(n) \geq$ Ausgangscodelänge (m)
 - Typische Notation: $(n \text{ zu } m)$ -Encoder
 - Umwandlung in einen (**dichten/dichteren**) Code
 - Optional zusätzliche Ausgänge zum Anzeigen der Gültigkeit des Wortes
- Decoder (Decodierer)
 - Ausgangscodelänge $(m) \geq$ Eingangscodelänge (n)
 - Typische Notation: $(n \text{ zu } m)$ -Decoder
 - Umwandlung in einen (**redundanten/redundanteren**) Code
 - Optional zusätzliche Eingänge zum Aktivieren des Decoders

Beispiel Binär-Encoder

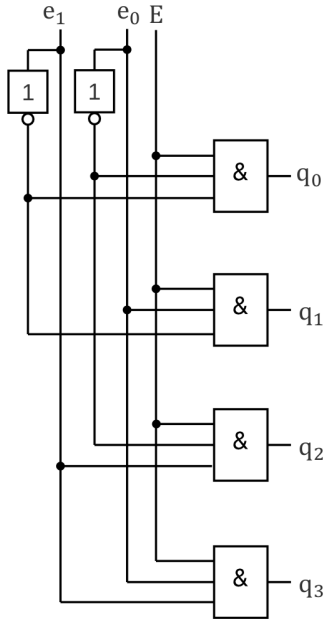
- Ein „1 aus 2^n “ Code wird zu einer n-Bit Binärzahl (Binär-Encoder)
- Bitfolge am Eingang \rightarrow Binärzahl am Ausgang



e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0	a_2	a_1	a_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

- Ausgangscodierung benötigt weniger Bits als Eingangscodierung

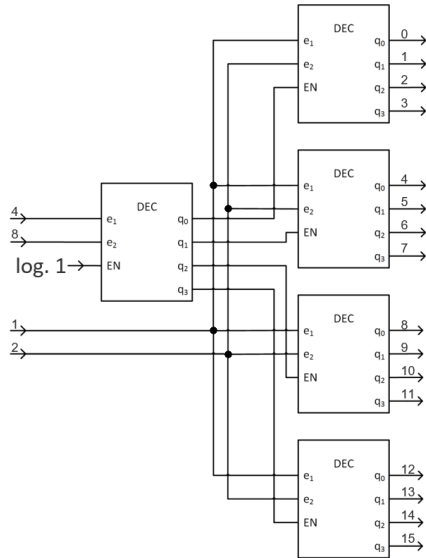
Beispiel (2 zu 4)-Decoder



- E ... Enable-Eingang
- UND-Gatter verarbeiten Eingangskombinationen
- Genau ein Ausgang wird auf log. 1 geschaltet – Auswahl mittels Eingangsbits
- Ausgangscodierung benötigt mehr Bits als Eingangscodierung

e_1	e_0	E	q_0	q_1	q_2	q_3
X	X	0	0	0	0	0
0	0	1	1	0	0	0
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	1

Realisierung eines (4 zu 16)-Decoders durch Kaskadierung

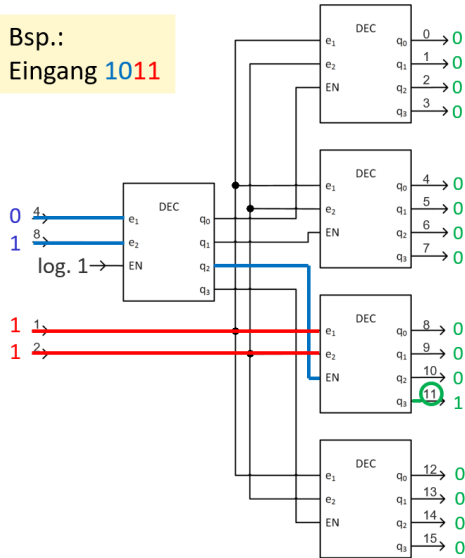


- Aufgebaut aus 5 Stk. (2 zu 4)-Decodern
- Enable-Eingang aktiviert die verschiedenen Decoder

Realisierung eines (4 zu 16)-Decoders durch Kaskadierung

Bsp.:

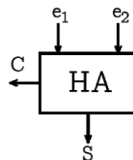
Eingang 1011



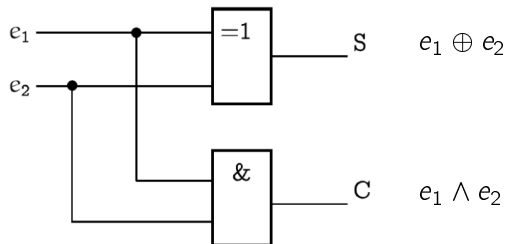
- Aufgebaut aus 5 Stk. (2 zu 4)-Decodern
- Enable-Eingang aktiviert die verschiedenen Decoder

Addierer

- Addiert zwei einstellige Binärzahlen ohne Übertrag (Halbaddierer, Half Adder)



e_1	e_2	$e_1 + e_2$
0	0	00
0	1	01
1	0	01
1	1	10



Volladdierer (Full Adder)

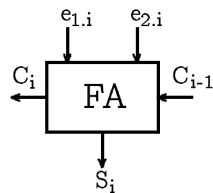
- Stufe i bezieht Übertrag aus der Vorstufe $i - 1$ mit ein

$e_{1,i}$	$e_{2,i}$	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

KDNF:

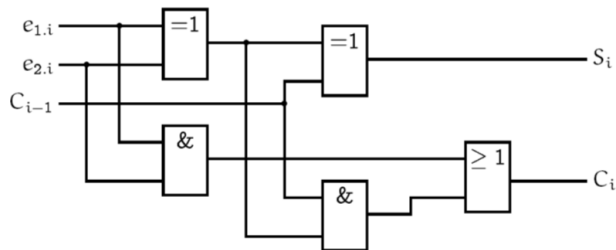
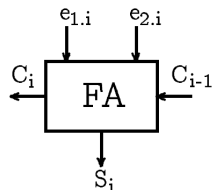
$$S_i = (\neg e_{1,i} \wedge \neg e_{2,i} \wedge C_{i-1}) \vee (\neg e_{1,i} \wedge e_{2,i} \wedge \neg C_{i-1}) \vee \\ \vee (e_{1,i} \wedge \neg e_{2,i} \wedge \neg C_{i-1}) \vee (e_{1,i} \wedge e_{2,i} \wedge C_{i-1})$$

$$C_i = (\neg e_{1,i} \wedge e_{2,i} \wedge C_{i-1}) \vee (e_{1,i} \wedge \neg e_{2,i} \wedge C_{i-1}) \vee \\ \vee (e_{1,i} \wedge e_{2,i} \wedge \neg C_{i-1}) \vee (e_{1,i} \wedge e_{2,i} \wedge C_{i-1}).$$



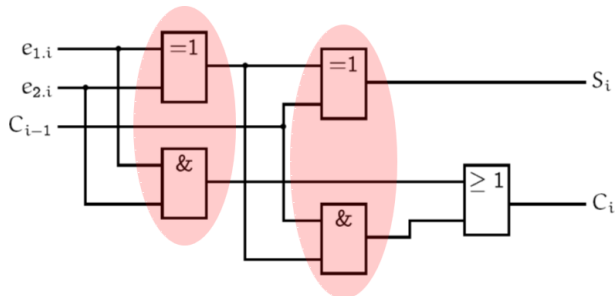
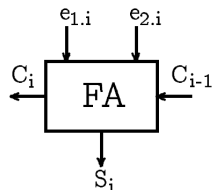
Volladdierer (Full Adder)

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



Volladdierer (Full Adder)

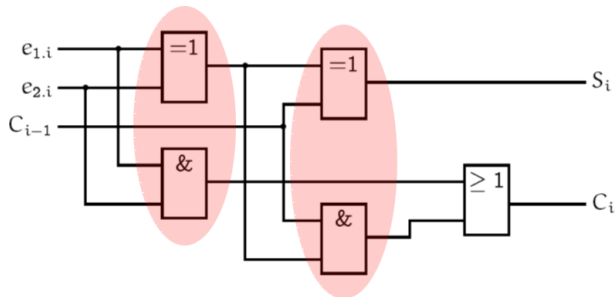
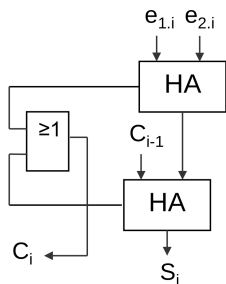
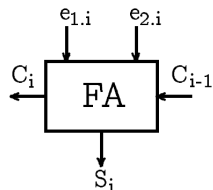
- $S_i = e_{1,i} \oplus e_{2,i} \oplus C_{i-1}$
- $C_i = (e_{1,i} \wedge e_{2,i}) \vee (C_{i-1} \wedge (e_{1,i} \oplus e_{2,i}))$



2 Halbaddierer nacheinander
geschaltet (kaskadiert)

Volladdierer (Full Adder)

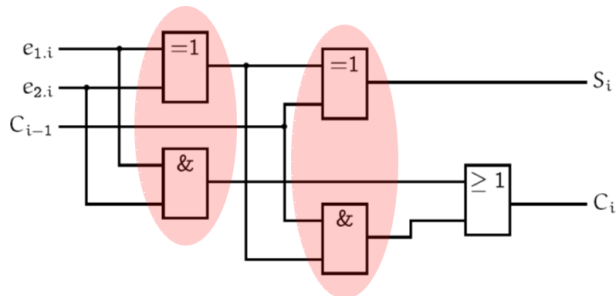
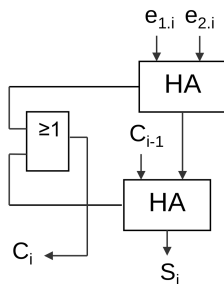
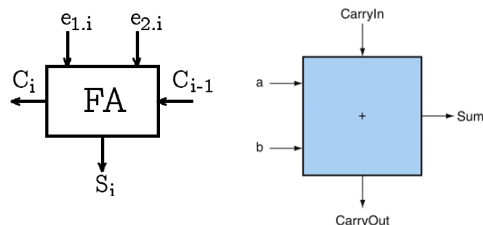
- $S_i = e_{1,i} \oplus e_{2,i} \oplus C_{i-1}$
- $C_i = (e_{1,i} \wedge e_{2,i}) \vee (C_{i-1} \wedge (e_{1,i} \oplus e_{2,i}))$



2 Halbaddierer nacheinander
geschaltet (kaskadiert)

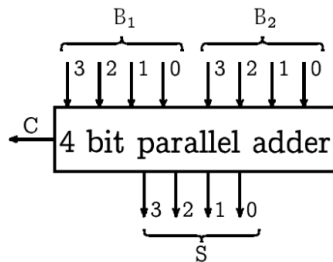
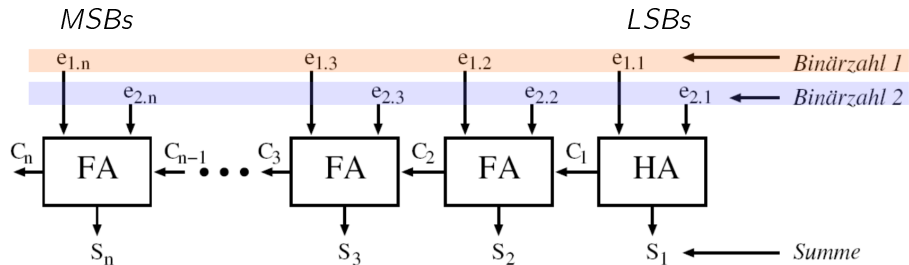
Volladdierer (Full Adder)

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



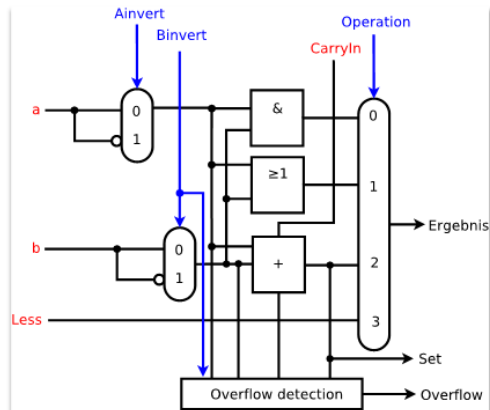
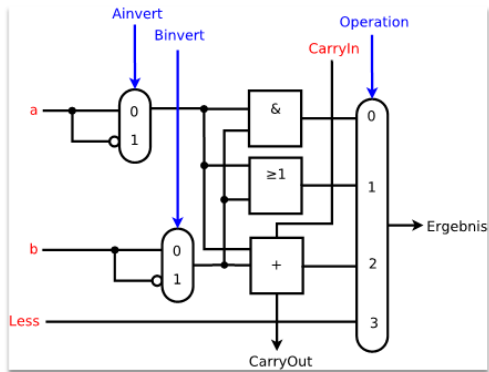
2 Halbaddierer nacheinander
geschaltet (kaskadiert)

Beispiel 4-Bit Paralleladdierer

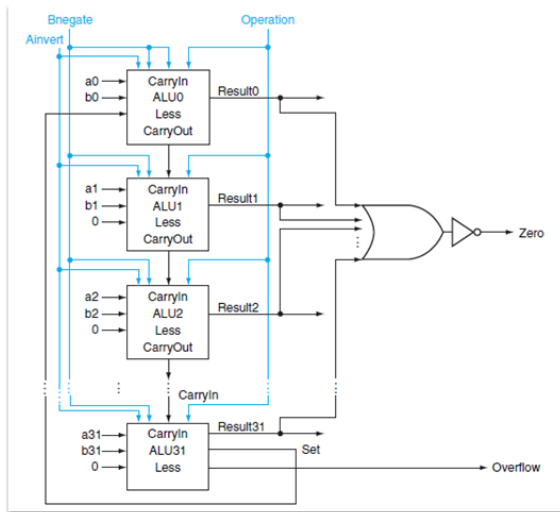


Addierer erweitern zur Arithmetic Logic Unit (ALU)

- 1-Bit ALU unterstützt die Funktion and, or, add und slt (set on less than)
 - Rechts zeigt eine 1-Bit ALU mit Überlauferkennung für das höchstwertige Bit einer n-Bit Zahl

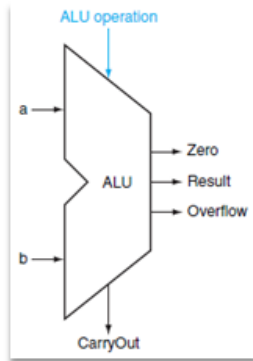
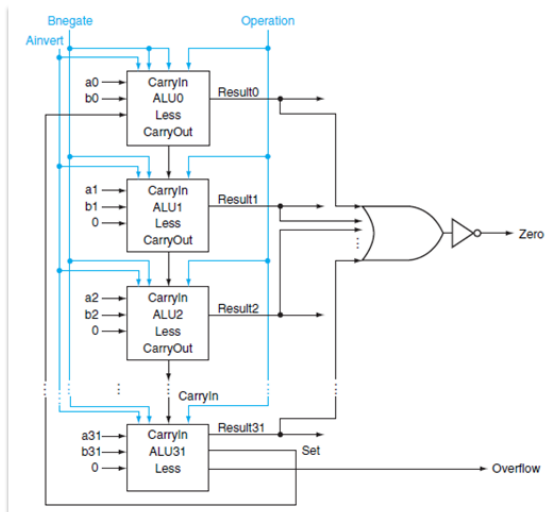


32-Bit ALU nach dem Ripple-Carry-Prinzip



- Subtraktion wird auf eine Addition zurückgeführt
 - Negative Zahl addieren
 - Zweierkomplementbildung
 - Summand invertieren und 1 addieren
 - $Bnegate$ auf 1 setzen
 - $CarryIn$ von ALU0 auf 1 setzen

32-Bit ALU nach dem Ripple-Carry-Prinzip



Sequenzielle Logik

Sequenzielle Logik vs. Kombinatorische Logik

- Bisher:

Kombinatorische Logik: Ausgänge nur vom momentanen Zustand der Eingänge bestimmt

- Keine Speicherung von Information, kein interner Zustand
- Schaltnetz, „combinational logic“

- Nun:

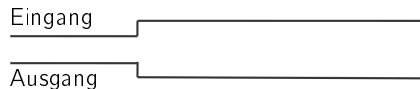
Sequenzielle Logik: Ausgänge vom momentanen Zustand der Eingänge **und** vom bisherigen Verlauf der Eingänge (Historie) bestimmt

- aktueller Zustand wird in Speicherelementen gehalten
- Schaltwerk, „sequential logic“

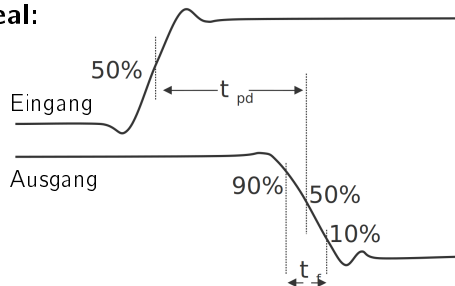
Zeitverhalten

- Z.B.: NOT-Gatter
- Ladungsträger benötigen Zeit zum Fließen
- **Durchlaufzeit** t_{pd}
(propagation delay)
„Zeitspanne zwischen dem Anliegen eines stabilen und gültigen Signals am Eingang bis zum Vorliegen eines stabilen und gültigen Signals am Ausgang.“
- **Flankensteilheit** t_f
 - Anstiegs-/Abfallzeit (rise/fall time)

Idealisiert:

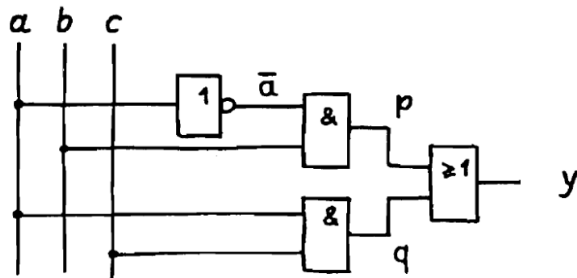


Real:



Zeitverhalten einer Boole'schen Funktion - Hazard

- Temporäre Falschaussage der Boole'schen Funktion $f(a, b, c)$ am Ausgang y



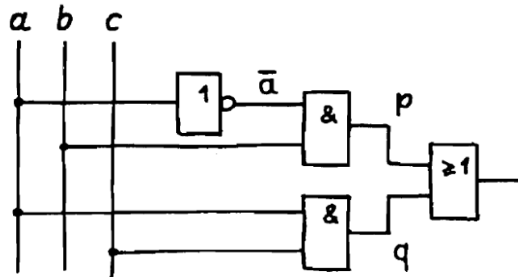
$$f(a, b, c) = (\neg a \wedge b) \vee (a \wedge c)$$

Annahme:

$$a = b = c = 1 \rightarrow f(a, b, c) = 1$$

Zeitverhalten einer Boole'schen Funktion - Hazard

- Temporäre Falschaussage der Boole'schen Funktion $f(a, b, c)$ am Ausgang y

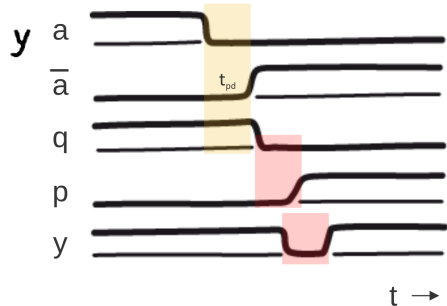


$$f(a, b, c) = (\neg a \wedge b) \vee (a \wedge c)$$

Annahme:

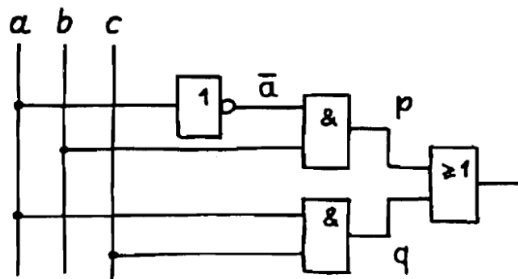
$$a = b = c = 1 \rightarrow f(a, b, c) = 1$$

$a = 1$ geht über auf $a = 0$



Zeitverhalten einer Boole'schen Funktion - Hazard

- Temporäre Falschaussage der Boole'schen Funktion $f(a, b, c)$ am Ausgang y



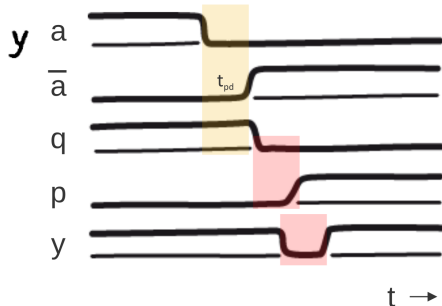
Abhilfe: Signale nur in festgelegten Intervallen betrachten (Takt)

$$f(a, b, c) = (\neg a \wedge b) \vee (a \wedge c)$$

Annahme:

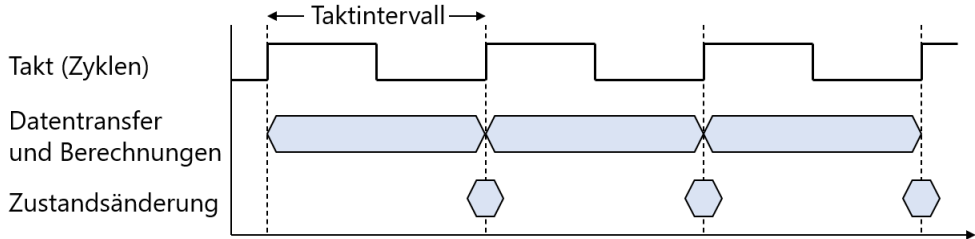
$$a = b = c = 1 \rightarrow f(a, b, c) = 1$$

$a = 1$ geht über auf $a = 0$



Takt und Taktzyklus

- Binäres Rechtecksignal
 - Wechselt in periodischen Abständen zwischen 0 und 1
 - Legt fest, wann Ereignisse innerhalb der Hardware stattfinden
- Taktzyklus oder auch Tick, Takt, Zyklus (clock cycle)
- Taktzykluszeit oder Taktintervall (clock period)
 - Die Länge eines Taktzyklus, z. B. $250\text{ps} = 0.25\text{ns} = 250 * 10^{-12}\text{s}$



Taktrate

- Taktgeschwindigkeit oder Taktrate (clock frequency (rate))
 - Das Inverse des Taktintervalls $\rightarrow \text{Taktrate} = \frac{1}{\text{Taktzykluszeit}}$
 - Zyklen pro Sekunde
 - Die Einheit ist Hertz (Hz)

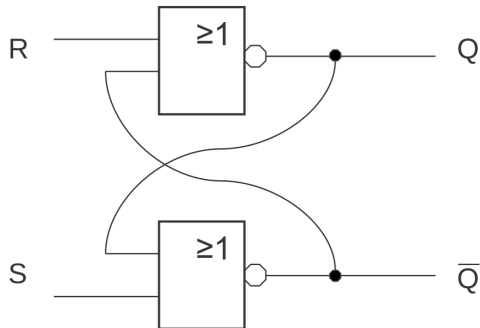
Taktrate	Taktzykluszeit
1 Hz	1 s
1 KHz (10^3 Hz)	1 ms (10^{-3} s)
100 KHz (10^5 Hz)	10 μ s (10^{-5} s)
500 KHz (5×10^5 Hz)	2 μ s
1 MHz (1×10^6 Hz)	1 μ s = 1000 ns
400 MHz (4×10^8 Hz)	2.5 ns
1 GHz (1×10^9 Hz)	1 ns = 1000 ps
2 GHz (2×10^9 Hz)	0.5 ns = 500 ps
4 GHz (4×10^9 Hz)	250 ps

Speicherelemente

- Schaltungen, die 1 Bit über einen gewissen Zeitraum speichern können
- Besitzen 2 stabile Zustände („bistabile Kippstufe“)
 - Set („gesetzt“)
 - Reset („rückgesetzt“)
- Diese Zustände können gespeichert werden
 - Solange Spannung anliegt
- Zustandswechsel durch Signale von außen z.B. Taktsignal

Speicherelemente

- Prinzip der Rückkoppelung (z.B. RS-Latch)



R ... Reset

S ... Set

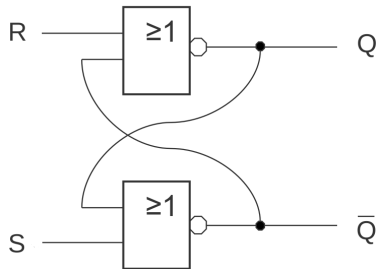
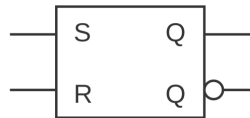
Q ... Ausgang

\bar{Q} ... Neg. Ausgang

RS-Latch

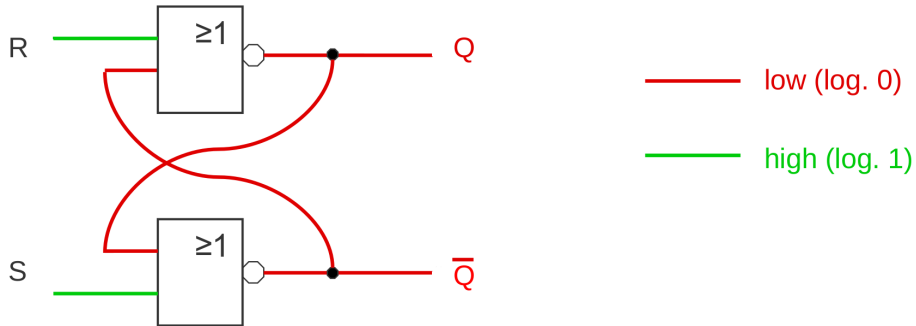
S	R	Q_{next}	
0	0	Q	Hold
0	1	0	Reset
1	0	1	Set
1	1	X	nicht erlaubt

Warum ist $S=R=1$ nicht erlaubt?



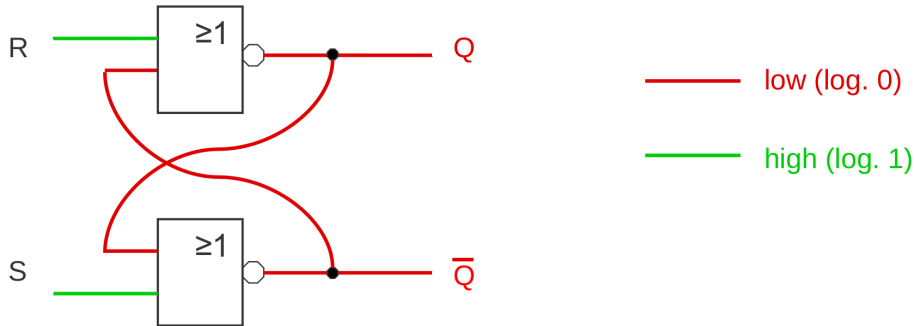
RS-Latch / Metastabilität

- $R = S = 1 \rightarrow Q = \overline{Q} = 0$ (Widerspruch zu $Q = \neg \overline{Q}$)



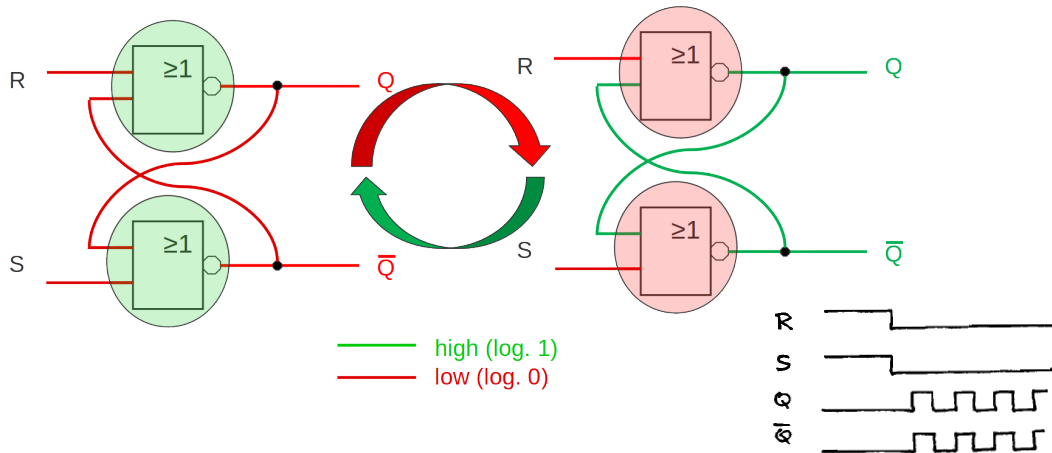
RS-Latch / Metastabilität

- $R = S = 1 \rightarrow Q = \overline{Q} = 0$ (Widerspruch zu $Q = \neg \overline{Q}$)
 - Was passiert nun wenn R und S gleichzeitig auf 0 wechseln?



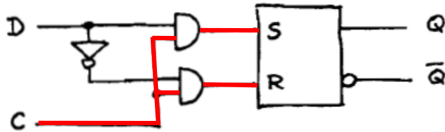
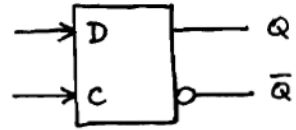
RS-Latch / Metastabilität

- Gleichzeitig von $R = S = 1$ auf $R = S = 0 \rightarrow Q = \overline{Q} = 1 = 0 = 1 = \dots$
- Ausgänge pendeln undefiniert zwischen 0 und 1 (die Ausgänge sind metastabil)



D-Latch

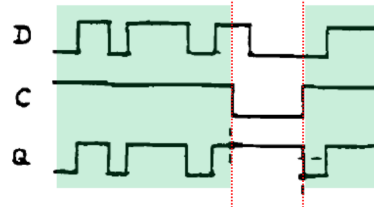
- Nur ein Dateneingang (D... Data)
 - Zu speichernder Wert
- Taktzustandsgesteuert (→ „quasi-synchron“)
 - Zustandsübergang nur in bestimmten Zeitintervallen möglich
- Steuereingang (C... Control) schaltet „transparent“
 - Wirkt wie Enable
 - Verarbeitet das Taktsignal



$C = \text{LOW} \rightarrow S = R = 0$

Ausgänge sind eingefroren
Letzter Wert wird gespeichert

Zeitablaufdiagramm (Impulsdiagramm)



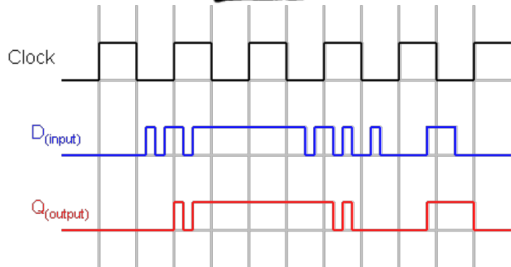
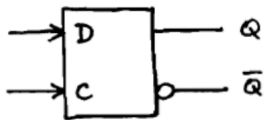
Nur wenn $C = \text{HIGH}$ wird Eingang (D)
an Ausgang (Q) übernommen

Abb.: TU-W/ICT

Taktflankenerkennung

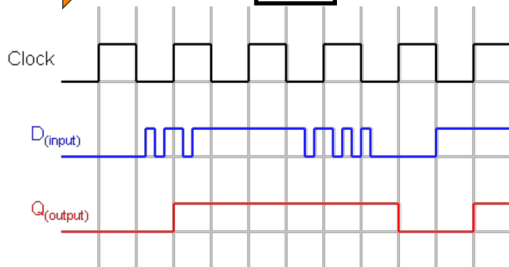
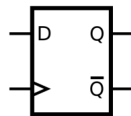
■ D-Latch

■ Taktzustandsgesteuert



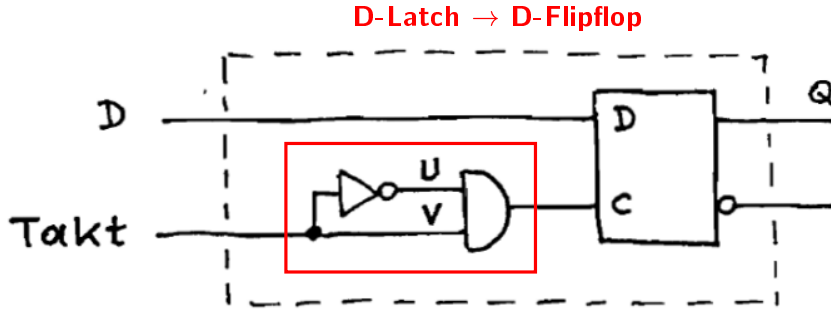
■ D-Flipflop

■ Taktflankengesteuert



Taktflankenerkennung

- Wie kann man eine (positive) Taktflanke erkennen?
- Kurzer Impuls – *monostabile Kippstufe*



- Schaltung erzeugt Hazard/Glitch
- C nimmt für die Durchlaufzeit von Negation HIGH Pegel an

D-Flipflop / Zeitspezifikation

- D-Flipflop: Eingang muss während Taktflanke stabil bleiben
 - Vorbereitungszeit (t_{su} setup time) – vor der Taktflanke
 - Haltezeit (t_h hold time) – nach der Taktflanke
 - Sonst metastabile Zustände
- Durchlaufzeit (t_{pd} propagation delay)
 - Von Taktflanke bis Ausgangsänderung

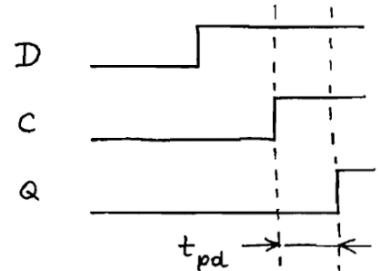
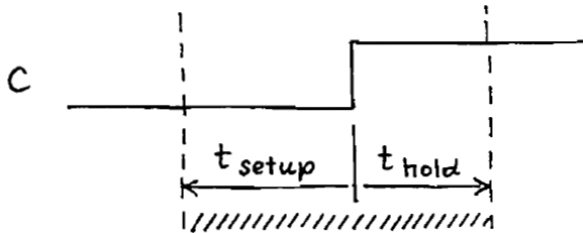
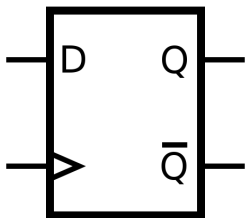


Abb.: TU-W ICT

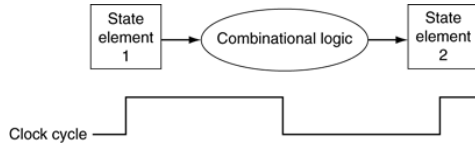
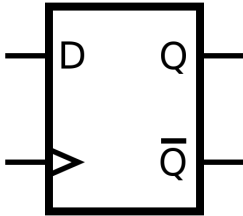
D-Flipflop als Speicherelement

- D-Flipflop als Registerspeicher
 - Schneller Speicher für Bitinformationen
 - Meist mehrere Bits (Wortbreite des Prozessors) gemeinsam abgespeichert
 - Takt bestimmt wann Wert in Register verändert wird
 - Update der Daten, wenn z.B. der Takt von 0 auf 1 wechselt (positive Taktflanke)



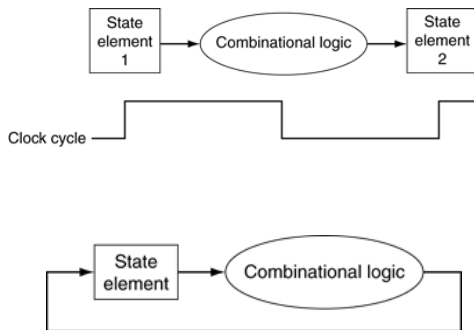
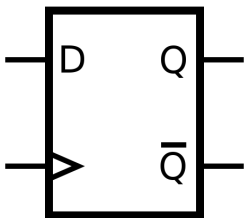
D-Flipflop als Speicherelement

- D-Flipflop als Registerspeicher
 - Schneller Speicher für Bitinformationen
 - Meist mehrere Bits (Wortbreite des Prozessors) gemeinsam abgespeichert
 - Takt bestimmt wann Wert in Register verändert wird
 - Update der Daten, wenn z.B. der Takt von 0 auf 1 wechselt (positive Taktflanke)

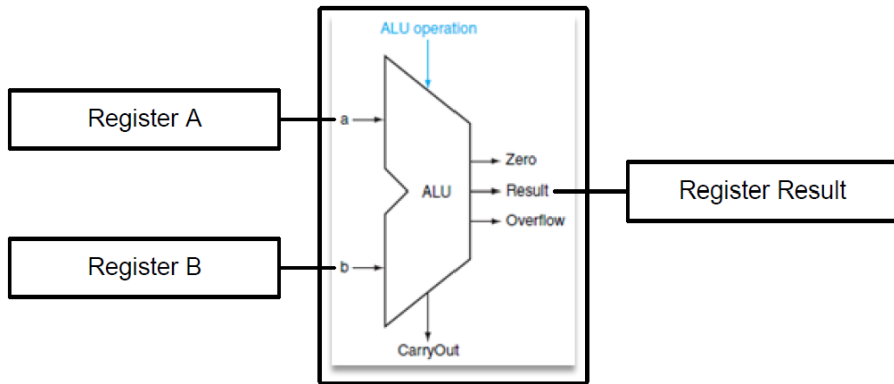


D-Flipflop als Speicherelement

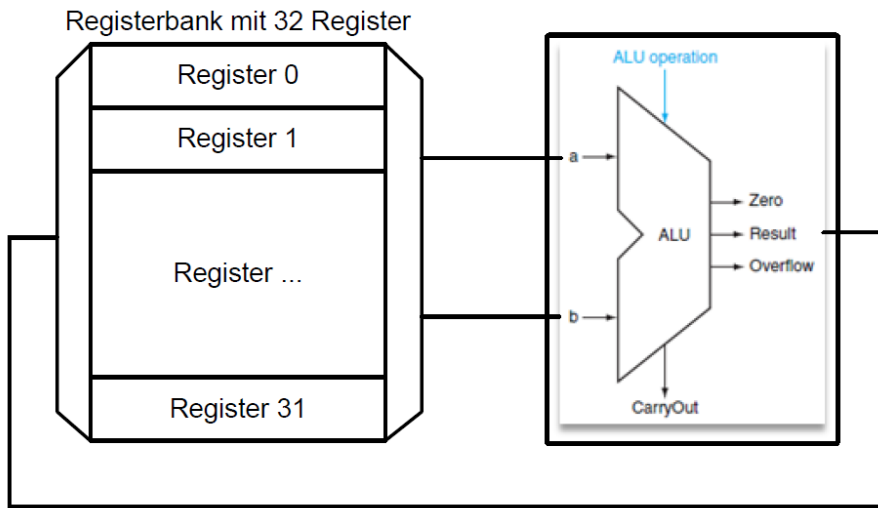
- D-Flipflop als Registerspeicher
 - Schneller Speicher für Bitinformationen
 - Meist mehrere Bits (Wortbreite des Prozessors) gemeinsam abgespeichert
 - Takt bestimmt wann Wert in Register verändert wird
 - Update der Daten, wenn z.B. der Takt von 0 auf 1 wechselt (positive Taktflanke)



Registerspeicher in Verbindung mit ALU



Registerspeicher in Verbindung mit ALU



Datenpfad und Steuereinheit MIPS32

