

Grundlagen der Digitaltechnik

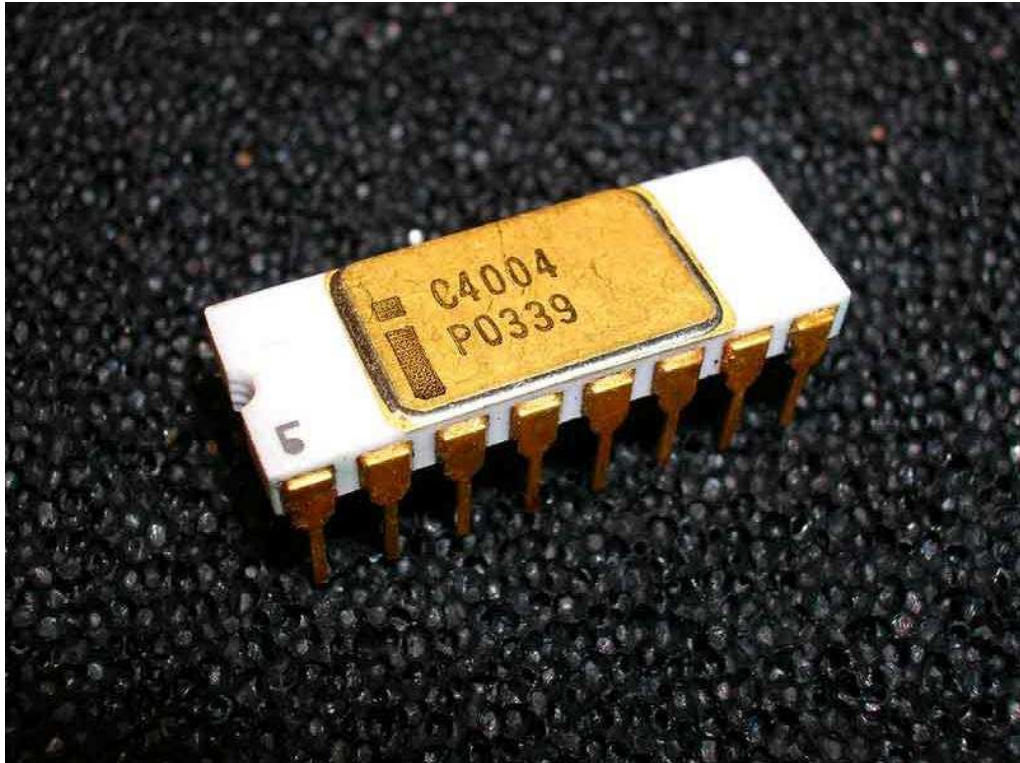
Dipl-Inform. Dr. Merten Joost
Universität Koblenz
Institut für integrierte
Naturwissenschaften
Abteilung Physik
email: merten.joost@uni-koblenz.de

Grundlagen der Digitaltechnik

Vom einfachen Stromkreis
zum Prozessor

<http://www.uni-koblenz.de/~physik/informatik>

Technische Informatik



- ▶ 'Die technische Informatik behandelt die Bestandteile, den Aufbau und die Zusammenarbeit von Computern, also die "Hardware". Die ihr zugrunde liegenden Fortschritte der Halbleitertechnik, Optoelektronik und elektrischen Nachrichtentechnik sind es, denen wir hauptsächlich das Gebäude der Informatik zu verdanken haben. Ohne den realen, aus Schaltkreisen bestehenden Computer hätte sich die Kenntnis von den Algorithmen und Datenstrukturen nicht entwickelt....

Welche technische Errungenschaft man auch nimmt – immer ist die Hardware-Entwicklung der Impuls für Neuerungen in den anderen Teilgebieten der Informatik gewesen. Die technische Informatik ist damit die Ursache fast aller Fortschritte in der Informatik, der Motor, der die Informatik antreibt.'

Informatik Handbuch (Hanser Verlag)

Praktische Informatik

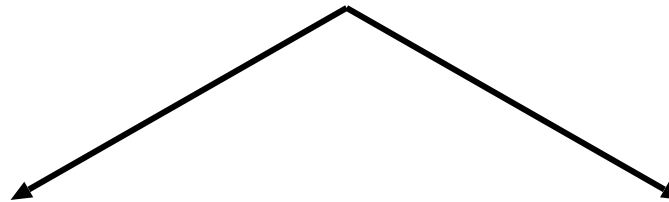
Technische Informatik

Theoretische Informatik



(Technische Informatik A)
Grundlagen der Digitaltechnik

- Vorlesung
- Uebung
- von der Elektrik bis zum Prozessor



(Technische Informatik C)
Mikrocontroller und Robotik

- Vorlesung
- Hardwarepraktikum
- Assembler, digitale/analoge Schnittstellen
- Mikrocontroller, Steuerung

(Technische Informatik B)
Rechnerstrukturen

- Vorlesung
- Rechnerstrukturen

Technische Informatik

Lehrangebot

	Montag	Dienstag	Mittwoch	Donerstag	Freitag
8-10					
10-12		Uebung			
12-14					Vorlesung
14-16	HWP	HWP		Uebung	
16-18			HWP		
18-20					

Literatur

1. Schiffmann, Schmitz
Technische Informatik
Band 1: Grundlagen der digitalen Elektronik
Band 2: Grundlagen der Computertechnik
Übungsbuch zur technischen Informatik
Springer Verlag

2. Beuth
Elektronik 1-4
Band 3: Grundsaltungen
Band 4: Digitaltechnik
Vogel-Buchverlag Würzburg

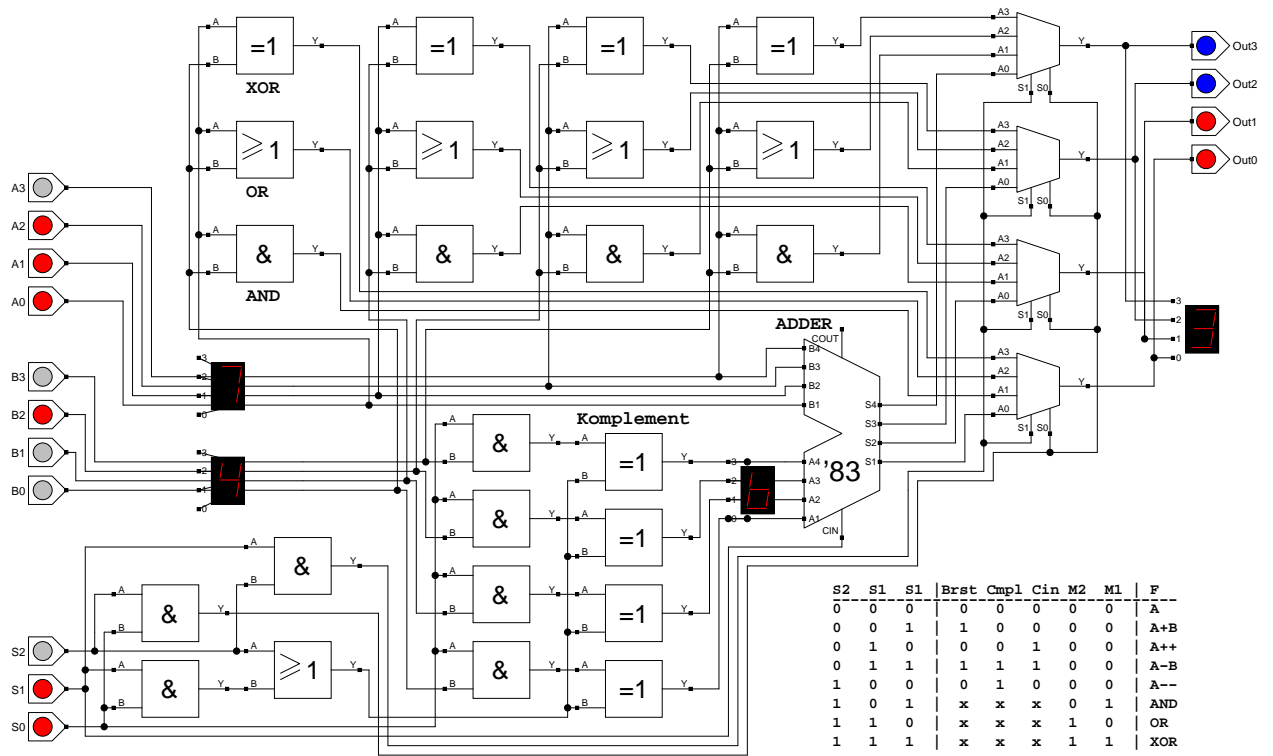
3. Urbanski, Weitowitz
Digitaltechnik
Springer Verlag

4. Borgmeyer
Grundlagen der Digitaltechnik
Hanser Verlag

Grundlagen der Digitaltechnik

- Grundlagen der Elektrik
- boole'sche Algebra
 - Schaltalgebra
 - Schaltfunktionen
- Schaltnetze
 - Analyse, Synthese
 - Codierer, Addierer, Komparatoren
 - Multiplexer
 - Arithmetisch-Logische Einheit
- Speicherglieder
 - Basis FlipFlop
 - RS-FlipFlop
 - D-FlipFlop
 - JK-FlipFlop
 - Master–Slave
 - Zustands/Flankensteuerung
- Schaltwerke
 - Automaten
 - Analyse, Synthese
 - Realisierung
- programmierbare Bausteine
 - Schaltnetze
 - Schaltwerke
- Entwicklung eines Mini–Prozessor

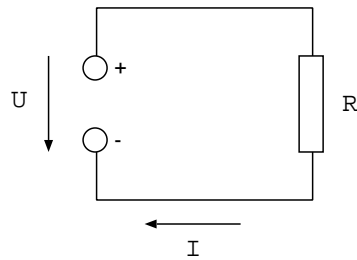
Beispiel: Schaltnetz



Kirchhofschen Gesetze

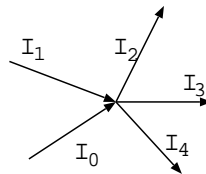
► einfacher Stromkreis

- Spannungsquelle, Verbraucher



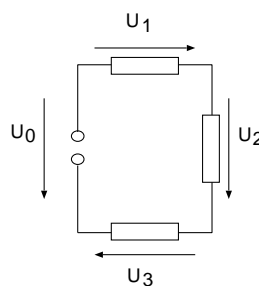
- $U = R * I$ (Ohm'sche Gesetz)

► Knotenregel



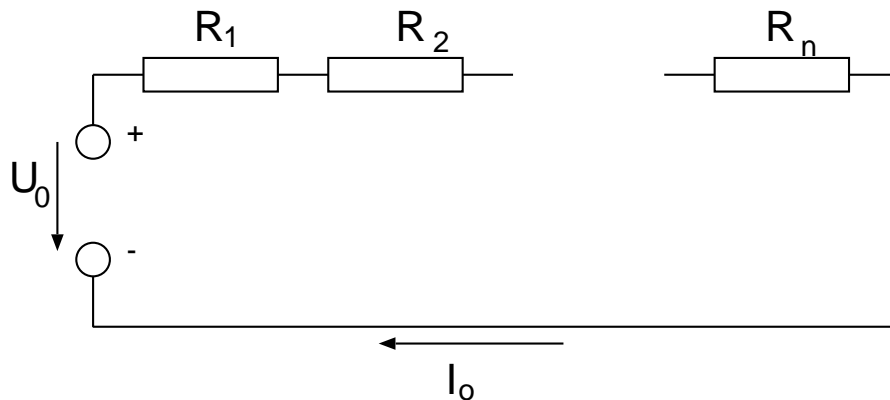
- Summe einlaufender Ströme = Summe auslaufender Ströme
- $\sum_{i=0}^n I_i = 0$

► Maschenregel



- Summe aller Spannungen = 0
- $\sum_{i=0}^n U_i = 0$

Reihenschaltung



- ▶ Ohm'sches Gesetz: $U_0 = R_g \cdot I_0$, $U_i = R_i \cdot I_i$
- ▶ Knotenregel: $I_0 = I_i$
- ▶ Maschenregel: $U_0 = U_1 + U_2 + \dots + U_n = \sum_{i=1}^n U_i$
- ▶ Berechnung der Teilspannungen

$$U_i = R_i \cdot I_i$$

$$U_i = R_i \cdot I_0$$

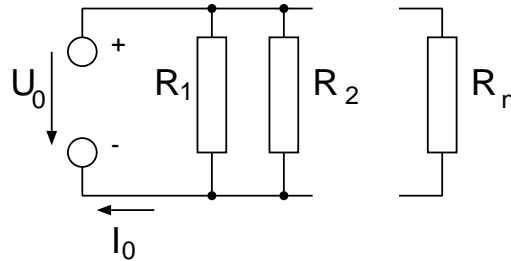
$$U_i \sim R_i$$

- ▶ Gesamtwiderstand

$$\begin{aligned} R_g &= \frac{U_0}{I_0} \\ &= \frac{\sum_{i=1}^n U_i}{I_0} \\ &= \sum_{i=1}^n \frac{U_i}{I_0} \\ R_g &= \sum_{i=1}^n R_i \end{aligned}$$

→ Gesamtwiderstand größer als alle Einzelwiderstände

Parallelschaltung



- ▶ Ohm'sches Gesetz: $U_0 = R_g \cdot I_0$, $U_i = R_i \cdot I_i$
- ▶ Knotenregel: $I_0 = I_1 + I_2 + \dots + I_n = \sum_{i=1}^n I_i$
- ▶ Maschenregel: $U_0 = U_i$
- ▶ Berechnung der Teilströme

$$I_i = \frac{U_i}{R_i}$$

$$I_i = \frac{U_0}{R_i}$$

$$I_i \sim \frac{1}{R_i}$$

- ▶ Gesamtwiderstand

$$R_g = \frac{U_0}{I_0}$$

$$= \frac{U_0}{\sum_{i=1}^n I_i}$$

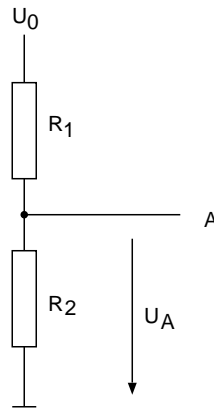
$$= \frac{U_0}{\sum_{i=1}^n \frac{U_0}{R_i}}$$

$$= \frac{U_0}{U_0 \sum_{i=1}^n \frac{1}{R_i}}$$

$$R_g = \frac{1}{\sum_{i=1}^n \frac{1}{R_i}}$$

→ Gesamtwiderstand kleiner als alle Einzelwiderstände

Spannungsteiler



- Einstellung der Ausgangsspannung U_A

$$\begin{aligned} U_A &= U_{R_2} \\ &= U_0 \frac{R_2}{R_1 + R_2} \end{aligned}$$

- R_1, R_2 für $U_A = \frac{1}{2}U_0$?

$$\begin{aligned} \frac{U_0}{2} &= U_0 \frac{R_2}{R_1 + R_2} \\ \frac{1}{2} &= \frac{R_2}{R_1 + R_2} \\ R_1 + R_2 &= 2R_2 \\ R_1 &= R_2 \end{aligned}$$

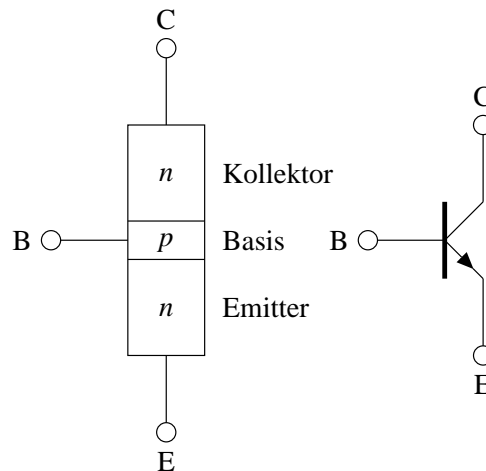
- U_A bei $R_2 \gg R_1$?

$$\begin{aligned} U_A &= U_0 \frac{R_2}{R_1 + R_2} \\ &\approx U_0 \frac{R_2}{R_2} \\ &\approx U_0 \end{aligned}$$

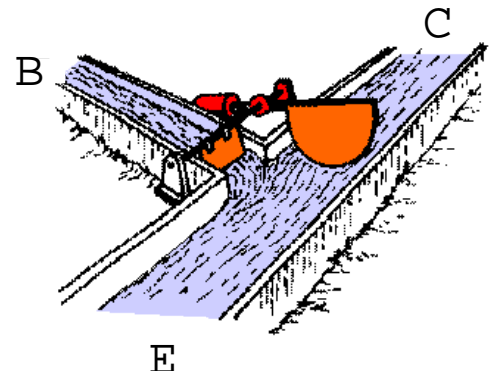
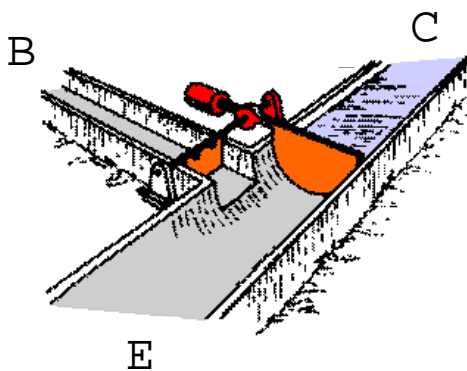
- U_A bei $R_2 \ll R_1$?

$$\begin{aligned} U_A &= U_0 \frac{R_2}{R_1 + R_2} \\ &\approx U_0 \frac{R_2}{R_1} \\ &\approx 0 \end{aligned}$$

Transistor

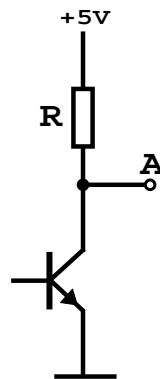


- ▶ Anwendungen: Verstärker und Schalter
- ▶ Wirkungsweise: regelbarer Widerstand
- ▶ Basis–Emitterstrom regelt Kollektor–Emitterstrom
- ▶ Anwendung in der Digitalelektronik: Schalter
 - geringer Basistrom → hoher Widerstand (Transistor sperrt)
 - hoher Basistrom → geringer Widerstand (Transistor leitet)



Quelle: <http://leifi.physik.uni-muenchen.de>

Spannungsteiler mit Transistor

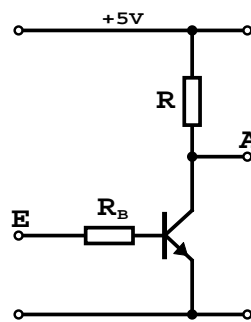


$$R_{T-sperr} \gg R \gg R_{T-leit}$$

- ▶ erzeugt zwei Spannungen am Ausgang

T	A
sperrt	High-Pegel
leitet	Low-Pegel

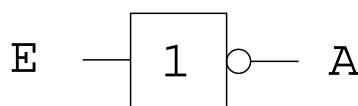
- ▶ einfache Inverterschaltung



E	A	bzw.	E	A
Low	High		0	1
High	Low		1	0

- ▶ Logische NOT-Funktion

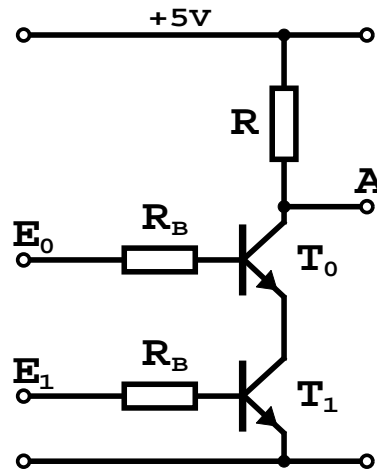
$$A = \overline{E}$$



Schaltungsanalyse

► Reihenschaltung von Transistoren

- Zwei Eingänge, ein Ausgang $A = f(E_0, E_1)$



$$R_{T\text{-sperr}} \gg R \gg R_{T\text{-leit}}$$

$$R_T = R_{T_0} + R_{T_1}$$

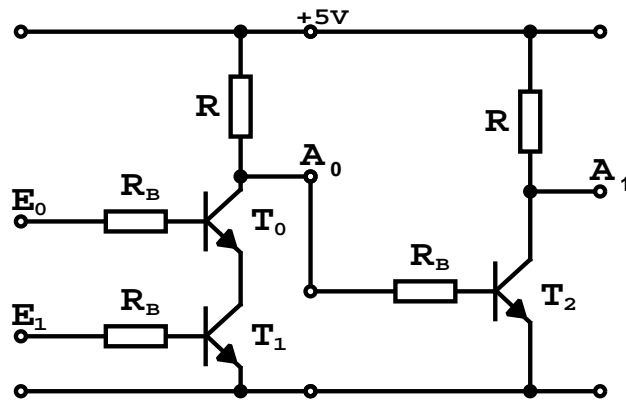
► Analyse

E_0	E_1	T_0	T_1	R_T	A
0	0	sperrt	sperrt	$R_T \gg R$	1
0	1	sperrt	leitet	$R_T \gg R$	1
1	0	leitet	sperrt	$R_T \gg R$	1
1	1	leitet	leitet	$R_T \ll R$	0

Nur wenn $E_0 = 1$ und $E_1 = 1$ ist, wird $A = 0$

Schaltungsanalyse

- ▶ mit nachgeschaltetem Inverter



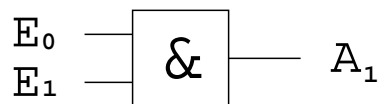
- ▶ Analyse

E_0	E_1	T_0	T_1	A_0	T_2	A_1
0	0	sperrt	sperrt	1	leitet	0
0	1	sperrt	leitet	1	leitet	0
1	0	leitet	sperrt	1	leitet	0
1	1	leitet	leitet	0	sperrt	1

- ▶ Nur wenn $E_0 = 1$ und $E_1 = 1$ ist, wird $A_1 = 1$

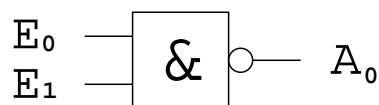
- ▶ Logische AND-Funktion

$$A_1 = E_0 \wedge E_1$$



- ▶ Entsprechend ohne Inverter: NAND-Funktion

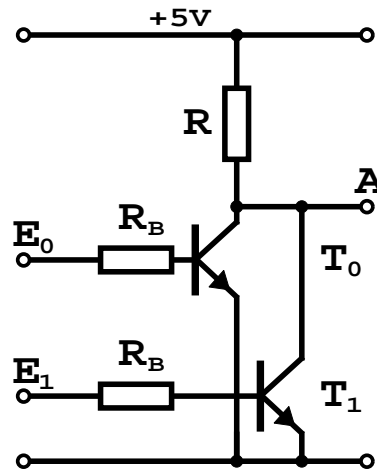
$$A_0 = \overline{E_0 \wedge E_1}$$



Schaltungsanalyse

► Parallelschaltung von Transistoren

- Zwei Eingänge, ein Ausgang $A = g(E_0, E_1)$



$$R_{T\text{-sperr}} \gg R \gg R_{T\text{-leit}}$$

$$R_T = R_{T_0} || R_{T_1}$$

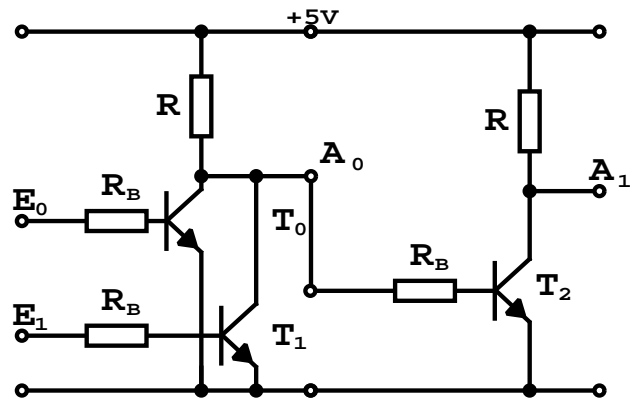
► Analyse

E_0	E_1	T_0	T_1	R_T	A
0	0	sperrt	sperrt	$R_T \gg R$	1
0	1	sperrt	leitet	$R_T \ll R$	0
1	0	leitet	sperrt	$R_T \ll R$	0
1	1	leitet	leitet	$R_T \ll R$	0

- Wenn $E_0 = 1$ oder $E_1 = 1$ ist, wird $A = 0$

Schaltungsanalyse

- ▶ mit nachgeschaltetem Inverter



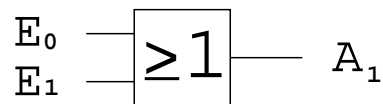
- ▶ Analyse

E_0	E_1	T_0	T_1	A_0	T_2	A_1
0	0	sperrt	sperrt	1	leitet	0
0	1	sperrt	leitet	0	sperrt	1
1	0	leitet	sperrt	0	sperrt	1
1	1	leitet	leitet	0	sperrt	1

- ▶ Wenn $E_0 = 1$ oder $E_1 = 1$ ist, wird $A_1 = 1$

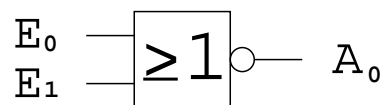
- ▶ Logische OR-Funktion

$$A_1 = E_0 \vee E_1$$

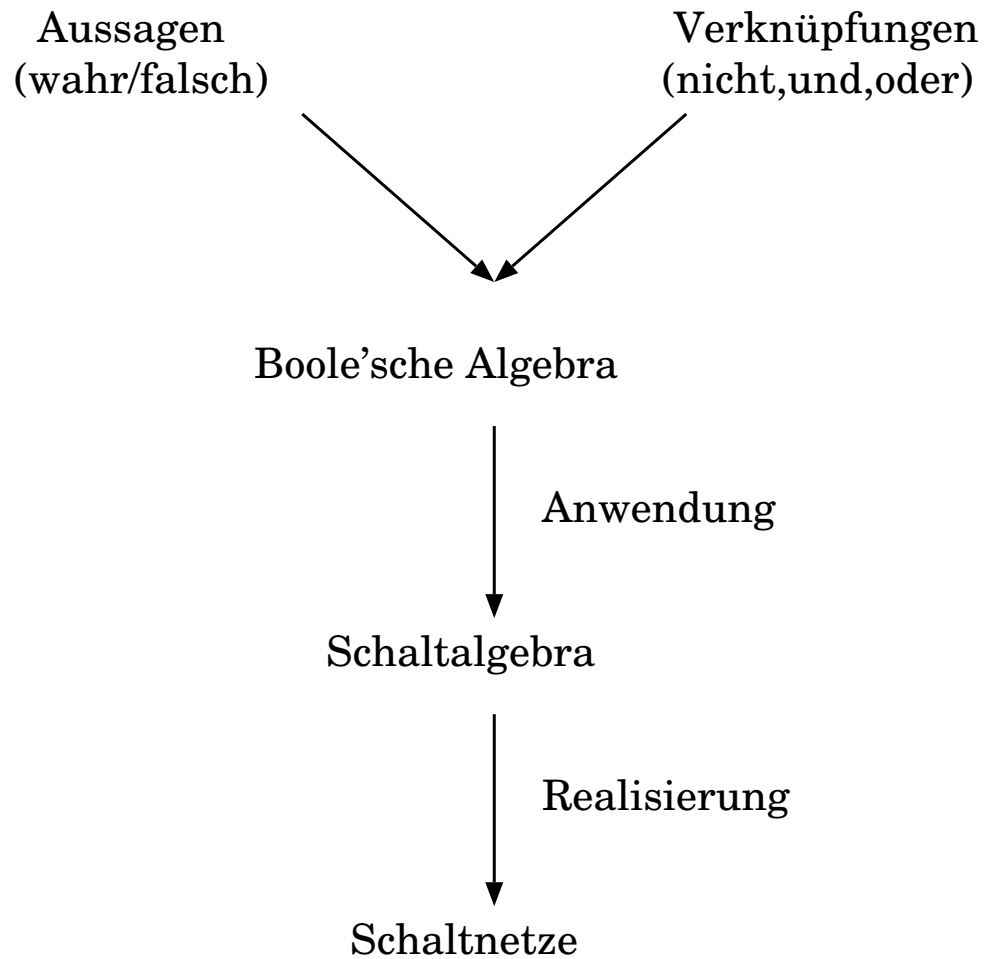


- ▶ Entsprechend ohne Inverter: NOR-Funktion

$$A_0 = \overline{E_0 \vee E_1}$$



Begriffe



Bool'sche Algebra

- ▶ Menge $B = \{a, b, c, \dots\}$
- ▶ Operatoren \neg , \wedge und \vee (Junktoren) mit:
 - $\neg : B \rightarrow B$
 - $\wedge : B \times B \rightarrow B$
 - $\vee : B \times B \rightarrow B$
 - Abgeschlossenheit: Jede boole'sche Operation auf boole'schen Werten liefert ein boole'sches Resultat.
- ▶ Huntington'sche Axiome
 - Kommutativgesetz

$$a \wedge b = b \wedge a$$

$$a \vee b = b \vee a$$
 - Distributivgesetz

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$
 - neutrale Elemente

$$a \wedge e = a$$

$$a \vee n = a$$
 - komplementäre Elemente

$$a \wedge \bar{a} = n$$

$$a \vee \bar{a} = e$$

Schaltalgebra

► Beispiele für boole'sche Algebra

Boole'sche Algebra	Aussagenalgebra	Schaltalgebra	Mengenalgebra	Erläuterung
B	$\{w, f\}$	$\{0, 1\}$	$P(T)$	
\vee	\vee	\vee	\cup	ODER
\wedge	\wedge	\wedge	\cap	UND
n	f	0	$\{\}$	Null-Element
e	w	1	T	Eins-Element
$\neg a(\bar{a})$	$\neg a(\bar{a})$	$\neg a(\bar{a})$	$\neg A$	Negation

► weitere Gesetze der boole'schen Algebra

- Assoziativgesetz

$$(a \wedge b) \wedge c = a \wedge (b \wedge c)$$

$$(a \vee b) \vee c = a \vee (b \vee c)$$

- Idempotenzgesetz

$$(a \wedge a) = a = (a \vee a)$$

- Absorptionsgesetz

$$a \wedge (a \vee b) = a$$

$$a \vee (a \wedge b) = a$$

- De Morgansche Regeln

$$\overline{a \wedge b} = \bar{a} \vee \bar{b}$$

$$\overline{a \vee b} = \bar{a} \wedge \bar{b}$$

Schaltfunktion

- ▶ **Schaltfunktion:**
Gleichung der Schaltalgebra, die die Abhängigkeit einer Schaltvariablen y (Ausgangsvariable) von einer oder mehreren unabhängigen Schaltvariablen $x_0, (x_1, x_2, \dots)$ (Eingangsvariablen) beschreibt.
- ▶ **Schaltvariable:**
ist ein Symbol für die Elemente der Schaltalgebra. Ist $B = \{0, 1\}$, dann bedeutet $x \in B$: Die Variable x kann nur die Werte 0 oder 1 annehmen.
- ▶ Ist $x_1 \in B$ und $x_2 \in B$ so hat die Produktmenge vier Elemente (Wertekombinationen):

x_1	x_2
0	0
0	1
1	0
1	1

- ▶ Mit n Variablen können 2^n Wertekombinationen gebildet werden.
- ▶ Eine Schaltfunktion ist eine eindeutige Zuordnungsvorschrift, die jeder Wertekombination von Schaltvariablen einen Wert zuordnet.

$$f(x_1, x_2, \dots, x_n) \in \{0, 1\}$$

oder

$$y = f(x_1, x_2, \dots, x_n)$$

Schaltfunktion

- ▶ Mit n Variablen können 2^n Wertekombinationen gebildet werden.
- ▶ Für jede Wertekombination kann die Ausgangsvariable die Werte 0,1 annehmen
- ▶ Mit n Variablen können $2^{(2^n)}$ Funktionen gebildet werden.
- ▶ Beispiel: $y = f(x)$

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	0	1	1
1	0	1	0	1

$$f_1(x) = 0$$

$$f_2(x) = x$$

$$f_3(x) = \bar{x}$$

$$f_4(x) = 1$$

- ▶ Beispiel: $y = f(x_1, x_2)$

	Funktionswert	Schreibweise mit den Zeichen	Bemerkung
Benennung der Verknüpfung	$y = f(x_1, x_2)$	$\wedge \vee -$	
Null	$y_0 = 0\ 0\ 0\ 0$	0	Null
UND-Verknüpfung	$y_1 = 0\ 0\ 0\ 1$	$x_1 \wedge x_2$	x_1 UND x_2
Inhibition	$y_2 = 0\ 0\ 1\ 0$	$\bar{x}_1 \wedge x_2$	
Transfer	$y_3 = 0\ 0\ 1\ 1$	x_2	
Inhibition	$y_4 = 0\ 1\ 0\ 0$	$x_1 \wedge \bar{x}_2$	
Transfer	$y_5 = 0\ 1\ 0\ 1$	x_1	
Antivalenz	$y_6 = 0\ 1\ 1\ 0$	$(x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$	Exklusiv-ODER
ODER-Verknüpfung	$y_7 = 0\ 1\ 1\ 1$	$x_1 \vee x_2$	x_1 ODER x_2
NOR-Verknüpfung	$y_8 = 1\ 0\ 0\ 0$	$\bar{x}_1 \vee \bar{x}_2$	NICHT-ODER
Äquivalenz	$y_9 = 1\ 0\ 0\ 1$	$(x_1 \wedge x_2) \vee (\bar{x}_1 \wedge \bar{x}_2)$	
Komplement	$y_{10} = 1\ 0\ 1\ 0$	\bar{x}_1	
Implikation	$y_{11} = 1\ 0\ 1\ 1$	$\bar{x}_1 \vee x_2$	
Komplement	$y_{12} = 1\ 1\ 0\ 0$	\bar{x}_2	
Implikation	$y_{13} = 1\ 1\ 0\ 1$	$x_1 \vee \bar{x}_2$	
NAND-Verknüpfung	$y_{14} = 1\ 1\ 1\ 0$	$\overline{x_1 \wedge x_2}$	NICHT-UND
Eins	$y_{15} = 1\ 1\ 1\ 1$	1	Eins

relevante Verknüpfungen

► AND

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	0
1	0	0
1	1	1

$$f(x_1, x_2) = x_1 \wedge x_2$$

► OR

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	1

$$\begin{aligned}
 f(x_1, x_2) &= (\overline{x_1} \wedge x_2) \vee (x_1 \wedge \overline{x_2}) \vee (x_1 \wedge x_2) && \text{Distributivgesetz} \\
 &= (\overline{x_1} \wedge x_2) \vee [x_1 \wedge (\overline{x_2} \vee x_2)] && \text{komplementäres Element} \\
 &= (\overline{x_1} \wedge x_2) \vee (x_1 \wedge e) && \text{neutrales Element} \\
 &= (\overline{x_1} \wedge x_2) \vee x_1 && \text{Distributivgesetz} \\
 &= (\overline{x_1} \vee x_1) \wedge (x_2 \vee x_1) && \text{komplementäres Element} \\
 &= e \wedge (x_2 \vee x_1) && \text{neutrale Element} \\
 &= x_2 \vee x_1 && \text{Kommutativgesetz} \\
 &= x_1 \vee x_2
 \end{aligned}$$

relevante Verknüpfungen

► NAND

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	1
1	1	0

$$\begin{aligned}
 f(x_1, x_2) &= (\overline{x_1} \wedge \overline{x_2}) \vee (\overline{x_1} \wedge x_2) \vee (x_1 \wedge \overline{x_2}) && \text{Distributivgesetz} \\
 &= [\overline{x_1} \wedge (\overline{x_2} \vee x_2)] \vee (x_1 \wedge \overline{x_2}) && \text{komplem., neutrales Element} \\
 &= \overline{x_1} \vee (x_1 \wedge \overline{x_2}) && \text{Distributivgesetz} \\
 &= (\overline{x_1} \vee x_1) \wedge (\overline{x_1} \vee \overline{x_2}) && \text{komplem., neutrales Element} \\
 &= \overline{x_1} \vee \overline{x_2} && \text{de Morgan} \\
 &= \overline{x_1 \wedge x_2}
 \end{aligned}$$

► NOR

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	0
1	0	0
1	1	0

$$\begin{aligned}
 f(x_1, x_2) &= \overline{x_1} \wedge \overline{x_2} && \text{de Morgan} \\
 &= \overline{x_1 \vee x_2}
 \end{aligned}$$

relevante Verknüpfungen

► Antivalenz (XOR)

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

$$\begin{aligned}
 f(x_1, x_2) &= (\overline{x_1} \wedge x_2) \vee (x_1 \wedge \overline{x_2}) \\
 &= x_1 \neq x_2
 \end{aligned}$$

► Äquivalenz (XNOR)

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	0
1	0	0
1	1	1

$$\begin{aligned}
 f(x_1, x_2) &= (\overline{x_1} \wedge \overline{x_2}) \vee (x_1 \wedge x_2) \\
 &= x_1 \equiv x_2
 \end{aligned}$$

► mit

$$\overline{\overline{x_1 \equiv x_2}} = x_1 \neq x_2$$

NAND und NOR

- ▶ NAND und NOR sind universell

- ▶ alle Schaltfunktionen lassen sich ausschließlich mit NAND-Verknüpfungen realisieren

- NOT

$$\bar{x} = \bar{x} \vee \bar{x} = \overline{x \wedge x}$$

- OR

$$x_1 \vee x_2 = \overline{\overline{x_1 \vee x_2}} = \overline{\overline{x_1} \wedge \overline{x_2}}$$

- AND

$$x_1 \wedge x_2 = \overline{\overline{x_1 \wedge x_2}} = \overline{\overline{x_1} \vee \overline{x_2}}$$

- ▶ alle Schaltfunktionen lassen sich ausschließlich mit NOR-Verknüpfungen realisieren

- NOT

$$\bar{x} = \bar{x} \wedge \bar{x} = \overline{x \vee x}$$

- OR

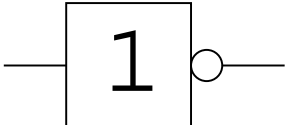
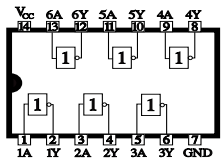
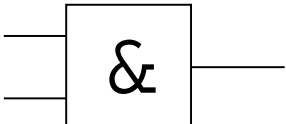
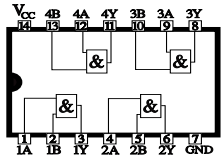
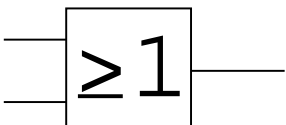
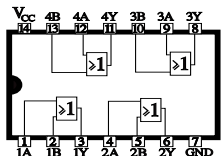
$$x_1 \vee x_2 = \overline{\overline{x_1 \vee x_2}} = \overline{\overline{x_1} \wedge \overline{x_2}}$$

- AND

$$x_1 \wedge x_2 = \overline{\overline{x_1 \wedge x_2}} = \overline{\overline{x_1} \vee \overline{x_2}}$$

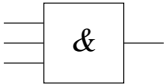
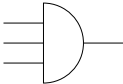

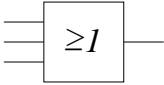
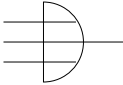
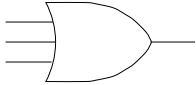
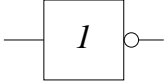
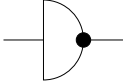
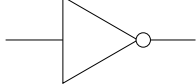
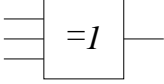
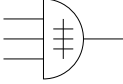
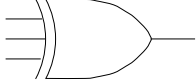
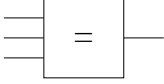
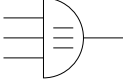
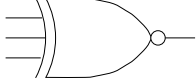
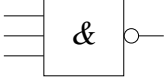
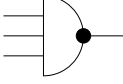
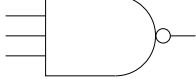
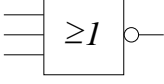
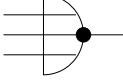







Schaltzeichen

- ▶ Beschreibung einer Verknüpfung \wedge, \vee, \neg
 - Wertetabelle
 - Angabe der Funktion
- ▶ weitere Möglichkeit
 - Schaltzeichen (graphische Darstellung)
 - Schaltnetze (physikalische Realisierung)

Operator	Schaltzeichen	Benennung	Beispiel-IC
—		NOT	7404 
\wedge		AND	7408 
\vee		OR	7432 

Schaltzeichen

► relevante Verküpfungen

DIN 40700 (ab 1976)	Schaltzeichen		Benennung
	Früher	in USA	
			UND - Glied (AND)
			ODER - Glied (OR)
			NICHT - Glied (NOT)
			Exklusiv-Oder - Glied (Exclusive-OR, XOR)
			Äquivalenz - Glied (Logic identity)
			UND - Glied mit negier- tem Ausgang (NAND)
			ODER - Glied mit negier- tem Ausgang (NOR)
			Negation eines Eingangs
			Negation eines Ausgangs

Beschreibung einer Schaltfunktion

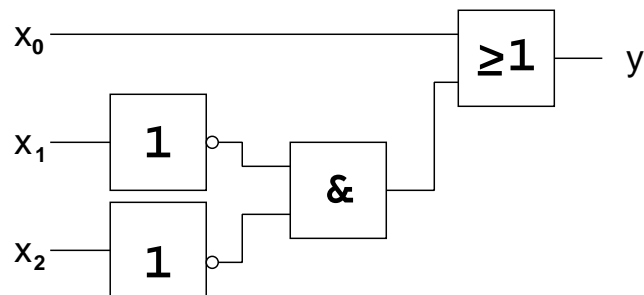
► Wertetabelle

x_0	x_1	x_2	$y = f(x_0, x_1, x_2)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

► Funktionsgleichung

$$y = x_0 \vee (\bar{x}_1 \wedge \bar{x}_2)$$

► Schaltnetz



Min/Max Terme

► Minterme

- sind UND-Verknüpfungen, die alle Eingangsvariablen genau einmal enthalten
- Eingangsvariablen dürfen negiert oder nicht negiert vorkommen
- bei n Variablen gibt es 2^n Wertekombinationen
- somit gibt es auch 2^n Minterme
- jeder Minterm hat nur bei einer Wertekombination den Wert 1

x_0	x_1	m_0 $(\overline{x_0} \wedge \overline{x_1})$	m_1 $(\overline{x_0} \wedge x_1)$	m_2 $(x_0 \wedge \overline{x_1})$	m_3 $(x_0 \wedge x_1)$
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

► Maxterme

- sind ODER-Verknüpfungen, die alle Eingangsvariablen genau einmal enthalten
- Eingangsvariablen dürfen negiert oder nicht negiert vorkommen
- bei n Variablen gibt es 2^n Maxterme
- jeder Maxterm hat nur bei einer Wertekombination den Wert 0

x_0	x_1	M_0 $(x_0 \vee x_1)$	M_1 $(x_0 \vee \overline{x_1})$	M_2 $(\overline{x_0} \vee x_1)$	M_3 $(\overline{x_0} \vee \overline{x_1})$
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

- es gilt: $\overline{m_i} = M_i$

Min/Max Terme

- ▶ Herleitung der Funktionsgleichung aus Mintermen
 - welche Minterme werden bei $f(x_0, x_1) = 1$ ebenfalls 1
 - Beispiel: XOR

x_0	x_1	$f(x_0, x_1)$	m_1	m_2	$m_1 \vee m_2$
0	0	0	0	0	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

$$\begin{aligned}
 f(x_0, x_1) &= m_1 \vee m_2 \\
 &= (\overline{x_0} \wedge x_1) \vee (x_0 \wedge \overline{x_1})
 \end{aligned}$$

- ▶ Herleitung der Funktionsgleichung aus Maxtermen
 - welche Maxterme werden bei $f(x_0, x_1) = 0$ ebenfalls 0
 - Beispiel: XOR

x_0	x_1	$f(x_0, x_1)$	M_0	M_3	$M_0 \wedge M_3$
0	0	0	0	1	0
0	1	1	1	1	1
1	0	1	1	1	1
1	1	0	1	0	0

$$\begin{aligned}
 f(x_0, x_1) &= M_0 \wedge M_3 \\
 &= (x_0 \vee x_1) \wedge (\overline{x_0} \vee \overline{x_1})
 \end{aligned}$$

- Alternativ: über inverse Funktion $\overline{f(x_0, x_1)}$

$$\begin{aligned}
 \overline{f(x_0, x_1)} &= m_0 \vee m_3 \\
 f(x_0, x_1) &= \overline{m_0 \vee m_3} \\
 &= \overline{(x_0 \wedge x_1) \wedge (\overline{x_0} \wedge \overline{x_1})} \\
 &= (\overline{x_0} \vee \overline{x_1}) \wedge (x_0 \vee x_1) \\
 &= M_0 \wedge M_3
 \end{aligned}$$

Normalformen

- Normalformen beschreiben eine Schaltfunktion ausgehend von einer Wertetabelle in Gleichungsform

x_2	x_1	x_0	$y = f(x_0, x_1, x_2)$	Minterme	m_i	Maxterme	M_i
0	0	0	1	$\bar{x}_0 \wedge \bar{x}_1 \wedge \bar{x}_2$	m_0	$x_0 \vee x_1 \vee x_2$	M_0
0	0	1	0	$x_0 \wedge \bar{x}_1 \wedge \bar{x}_2$	m_1	$\bar{x}_0 \vee x_1 \vee x_2$	M_1
0	1	0	0	$\bar{x}_0 \wedge x_1 \wedge \bar{x}_2$	m_2	$x_0 \vee \bar{x}_1 \vee x_2$	M_2
0	1	1	1	$x_0 \wedge x_1 \wedge \bar{x}_2$	m_3	$\bar{x}_0 \vee \bar{x}_1 \vee x_2$	M_3
1	0	0	1	$\bar{x}_0 \wedge \bar{x}_1 \wedge x_2$	m_4	$x_0 \vee x_1 \vee \bar{x}_2$	M_4
1	0	1	1	$x_0 \wedge \bar{x}_1 \wedge x_2$	m_5	$\bar{x}_0 \vee x_1 \vee \bar{x}_2$	M_5
1	1	0	0	$\bar{x}_0 \wedge x_1 \wedge x_2$	m_6	$x_0 \vee \bar{x}_1 \vee \bar{x}_2$	M_6
1	1	1	0	$x_0 \wedge x_1 \wedge x_2$	m_7	$\bar{x}_0 \vee \bar{x}_1 \vee \bar{x}_2$	M_7

- DNF

- disjunktive Verknüpfung der Minterme mit $f(x_0, \dots, x_n) = 1$

$$f(x_0, x_1, x_2) = m_0 \vee m_3 \vee m_4 \vee m_5$$

$$f(x_0, x_1, x_2) = (\bar{x}_0 \wedge \bar{x}_1 \wedge \bar{x}_2) \vee (x_0 \wedge x_1 \wedge \bar{x}_2) \vee (\bar{x}_0 \wedge \bar{x}_1 \wedge x_2) \vee (x_0 \wedge \bar{x}_1 \wedge x_2)$$

- KNF

- konjunktive Verknüpfung der Maxterme mit $f(x_0, \dots, x_n) = 0$

$$f(x_0, x_1, x_2) = M_1 \wedge M_2 \wedge M_6 \wedge M_7$$

$$f(x_0, x_1, x_2) = (\bar{x}_0 \vee x_1 \vee x_2) \wedge (x_0 \vee \bar{x}_1 \vee x_2) \wedge (x_0 \vee \bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_0 \vee \bar{x}_1 \vee \bar{x}_2)$$

- jede Schaltfunktion kann in *disjunktiver* und *konjunktiver Normalform* dargestellt werden
- Beide Darstellungen sind äquivalent und ineinander überführbar.

Minimierung von Schaltfunktionen

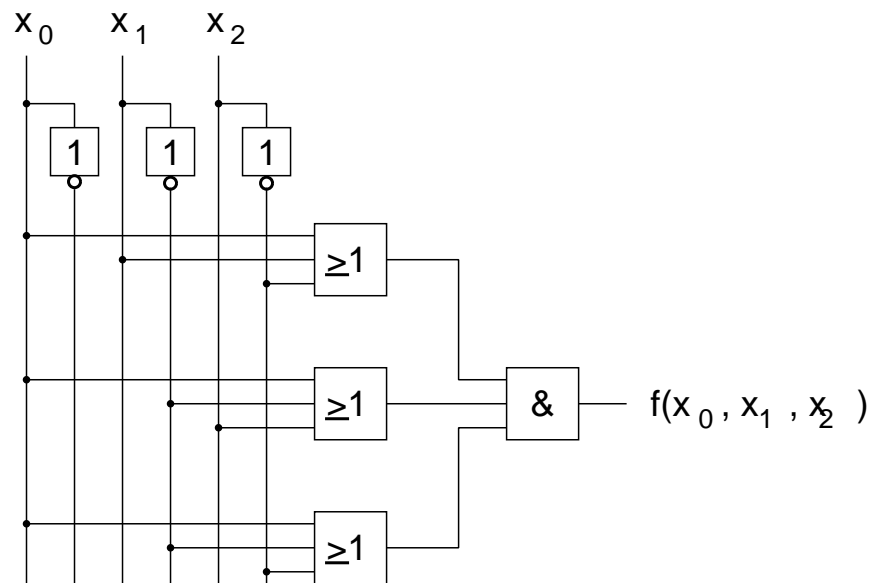
- ▶ Normalformen enthalten häufig redundante Terme
- ▶ Ziel:
 - Vereinfachte Darstellung
 - ↳ möglichst wenig Variablen
 - ↳ möglichst wenig Verknüpfungen
 - geringerer Hardwareaufwand bei der Realisierung als Schaltnetz
- ▶ Mit Hilfe der Regeln der Schaltalgebra
 - Beispiele: siehe Herleitung der OR und NAND Funktion
 - weiteres Beispiel:

x_0	x_1	x_2	$f(x_0, x_1, x_2)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

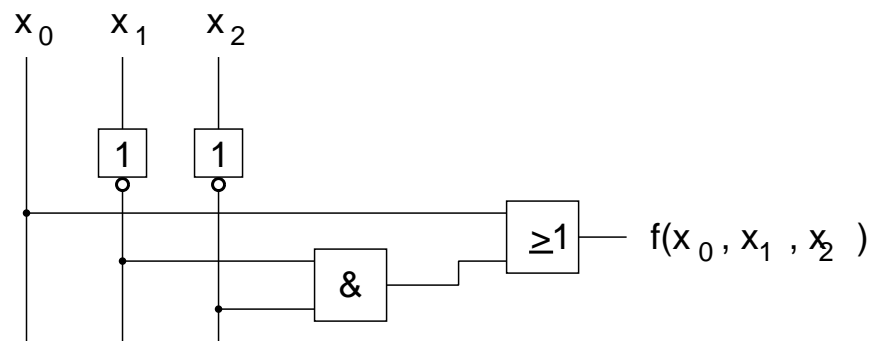
$$\begin{aligned}
 f(x) &= M_1 \wedge M_2 \wedge M_3 \\
 &= (x_0 \vee x_1 \vee \overline{x_2}) \wedge (x_0 \vee \overline{x_1} \vee x_2) \wedge (x_0 \vee \overline{x_1} \vee \overline{x_2}) \\
 &= x_0 \vee [(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2})] \\
 &= x_0 \vee [(x_1 \vee \overline{x_2}) \wedge [\overline{x_1} \vee (x_2 \wedge \overline{x_2})]] \\
 &= x_0 \vee [(x_1 \vee \overline{x_2}) \wedge \overline{x_1}] \\
 &= x_0 \vee (x_1 \overline{x_1} \vee \overline{x_1} \overline{x_2}) \\
 &= x_0 \vee \overline{x_1} \overline{x_2}
 \end{aligned}$$

Schaltnetze

► KNF-Schaltnetz



► minimiertes Schaltnetz



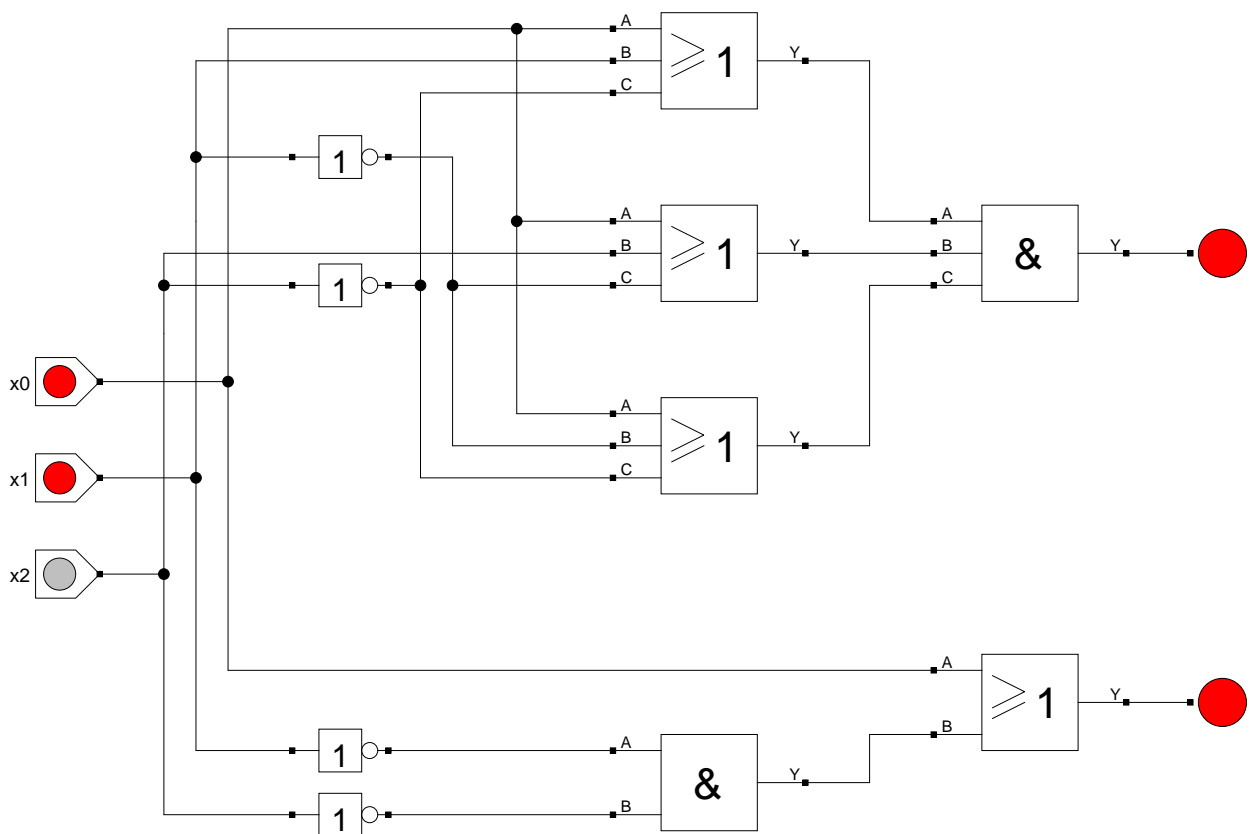
Hades

► Hamburg Design System

► <http://tams-www.informatik.uni-hamburg.de/applets/hades/html/hades.html>

- Freeware
- JAVA-basiert

► knf_vs_min.hds



Karnaugh-Veitch (KV) Diagramm

- ▶ grafisches Minimierungsverfahren
- ▶ stellt die Funktion in DNF oder KNF tabellarisch dar
- ▶ Achsen: Belegung der Eingangsvariablen
- ▶ Tabelle: Funktionswerte
- ▶ Beispiel: NAND

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

- KV Tafeln

	A	\bar{A}	A
B		0	1
\bar{B}	0	1	1
B	1	1	0

$$Q = \bar{A} \vee \bar{B} = \overline{A \wedge B}$$

	A	\bar{A}
B		0
\bar{B}	0	1
B	1	1
\bar{B}	1	0

$$Q = \bar{A} \vee \bar{B} = \overline{A \wedge B}$$

KV-Regeln

- ▶ Tabellarische Darstellung einer Schaltfunktion
- ▶ n Eingangsvariablen $\rightarrow 2^n$ Felder
- ▶ jede Variable in negierter und nicht negierter Form
- ▶ horizontal und vertikal benachbarte Felder unterscheiden sich in genau einer Eingangsvariablen
 - horizontale und vertikale Zyklizität
- ▶ Funktionswerte:
 - Minterme der Funktion: 1
 - Maxterme der Funktion: 0
 - undefiniert (don't care): *
- ▶ Blockbildung:
 - möglichst grosse Rechtecke gleicher Belegung
 - nur zweier Potenzen
 - don't care beachten
 - Zyklizität beachten
 - mehrfache Zuordnung der Felder in Blöcke möglich
- ▶ Auswertung
 - Blöcke bilden vereinfachte DF (KF) Terme
 - Variablen mit beiden Belegungen entfallen
 - Bei KF Variablen invertieren (s. MAX Terme)

KV mit 3 Eingangsvariablen

x_0	x_1	x_2	$f(x_0, x_1, x_2)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

► KV in DNF

		$x_1 x_2$			
		$\overline{x_1} \overline{x_2}$	$\overline{x_1} x_2$	$x_1 \overline{x_2}$	$x_1 x_2$
x_0		00	01	11	10
$\overline{x_0}$	0	1	0	0	0
x_0	1	1	1	1	1

$$y = x_0 \vee \overline{x_1} \overline{x_2}$$

► KV in KNF

		$x_1 x_2$			
		$x_1 x_2$	$\overline{x_1} \overline{x_2}$	$\overline{x_1} x_2$	$x_1 \overline{x_2}$
x_0		00	01	11	10
x_0	0	1	0	0	0
$\overline{x_0}$	1	1	1	1	1

$$y = (x_0 \vee \overline{x_2}) \wedge (x_0 \vee \overline{x_1}) = x_0 \vee \overline{x_1} \overline{x_2}$$

KV mit 4 Eingangsvariablen

► Wahrheitstabelle

x_3	x_2	x_1	x_0	y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0

x_3	x_2	x_1	x_0	y
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

► KV-Diagramme

		$x_1 x_0$			
		$\bar{x}_1 \bar{x}_0$	$\bar{x}_1 x_0$	$x_1 \bar{x}_0$	$x_1 x_0$
x_3	\bar{x}_3	00	01	11	10
	x_3	00	01	11	10
	\bar{x}_3	01	01	11	10
	x_3	11	01	11	10
	\bar{x}_3	10	01	11	10
		1	0	1	1
		0	1	0	0
		0	1	0	0
		1	0	1	1

		$x_1 x_0$			
		$\bar{x}_1 \bar{x}_0$	$\bar{x}_1 x_0$	$x_1 \bar{x}_0$	$x_1 x_0$
x_3	\bar{x}_3	00	01	11	10
	x_3	00	01	11	10
	\bar{x}_3	01	01	11	10
	x_3	11	01	11	10
	\bar{x}_3	10	01	11	10
		1	0	1	1
		0	1	0	0
		0	1	0	0
		1	0	1	1

► minimierte Funktionen

- DF: $y = (x_2 \wedge \bar{x}_1 \wedge x_0) \vee (\bar{x}_2 \wedge \bar{x}_0) \vee (\bar{x}_2 \wedge x_1)$
- KF: $y = (x_2 \vee x_1 \vee \bar{x}_0) \wedge (\bar{x}_2 \vee x_0) \wedge (\bar{x}_2 \vee \bar{x}_1)$

Beschreibungsmöglichkeiten Schaltnetze

► Beispiel NAND-Funktion

- Wahrheitstabelle

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

- Funktionsgleichung

$$Q = \overline{A \wedge B}$$

- KV Tafeln

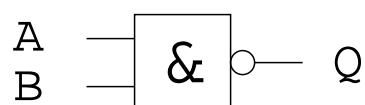
	A	\overline{A}	A
\overline{B}	0	1	1
B	1	1	0

$$Q = \overline{A} \vee \overline{B} = \overline{A \wedge B}$$

	A	A	\overline{A}
\overline{B}	0	1	1
B	1	1	0

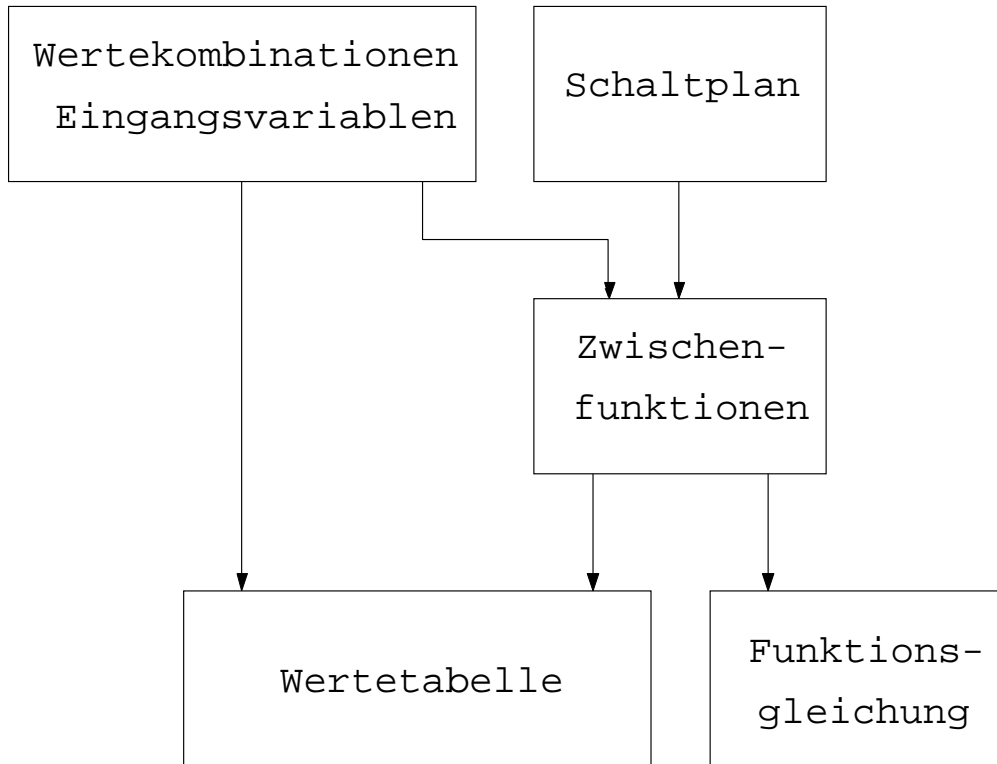
$$Q = \overline{A} \vee \overline{B} = \overline{A \wedge B}$$

- Schaltplan



Analyse

► Analyse von Schaltnetzen



► Ermittlung Funktionsgleichung

- Zwischenfunktionen kennzeichnen
- Funktionsgleichung mittels Zwischenfunktionen ermitteln
- Zwischenfunktionen ersetzen

► Ermittlung Wertetabelle

- Wertekombinationen (Eingangsvariablen) in Tabelle eintragen
- Zwischenfunktionen kennzeichnen
- Funktionswerte der Zwischenfunktionen ermitteln und in Tabelle eintragen
- Aus Zwischenfunktionswerten Wert für Ausgangsfunktion ermitteln

Analyse von SN

► Analyse von Schaltnetzen

• Beispiel zur Schaltnetzanalyse

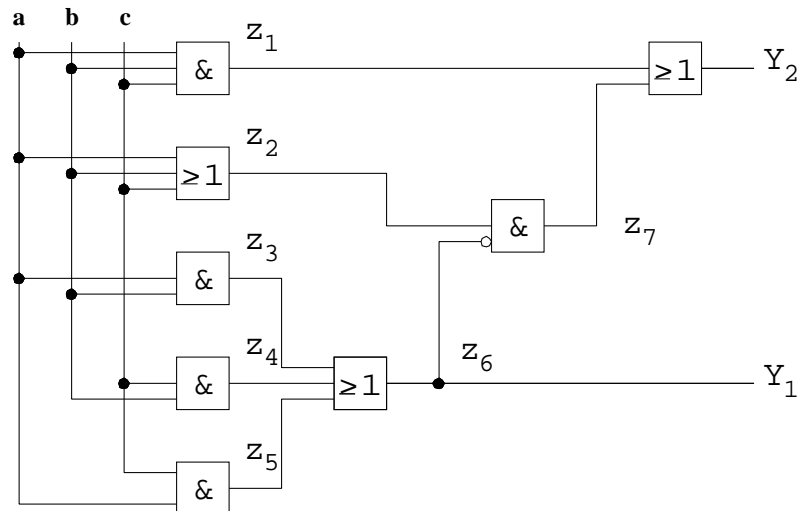


Tabelle der Zwischenfunktionen zur Analyse des Schaltnetzes

Eingangsvar.			Z_1	Z_2	Z_3	Z_4	Z_5	$Z_6 = Y_1$	$\overline{Z_6}$	Z_7	Y_2
c	b	a	abc	$a \vee b \vee c$	ab	bc	ac	$Z_3 \vee Z_4 \vee Z_5$		$Z_2 \wedge \overline{Z_6}$	$Z_1 \vee Z_7$
0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	0	1	0	0	0	0	1	1	1
0	1	0	0	1	0	0	0	0	1	1	1
0	1	1	0	1	1	0	0	1	0	0	0
1	0	0	0	1	0	0	0	0	1	1	1
1	0	1	0	1	0	0	1	1	0	0	0
1	1	0	0	1	0	1	0	1	0	0	0
1	1	1	1	1	1	1	1	1	0	0	1

Analyse von SN

$$\begin{aligned}
 y_2 &= z_1 \vee z_7 \\
 &= z_1 \vee (z_2 \overline{z_6}) \\
 &= (abc) \vee (a \vee b \vee c) \overline{z_6} \\
 &= (abc) \vee a \overline{z_6} \vee b \overline{z_6} \vee c \overline{z_6}
 \end{aligned}$$

$$\begin{aligned}
 \overline{z_6} &= \overline{z_3 \vee z_4 \vee z_5} \\
 &= \overline{z_3} \wedge \overline{z_4} \wedge \overline{z_5} \\
 &= \overline{ab} \wedge \overline{bc} \wedge \overline{ac} \\
 &= (\overline{a} \vee \overline{b}) \wedge (\overline{b} \vee \overline{c}) \wedge (\overline{a} \vee \overline{c})
 \end{aligned}$$

$$\begin{aligned}
 a \overline{z_6} &= a \wedge (\overline{a} \vee \overline{b}) \wedge (\overline{b} \vee \overline{c}) \wedge (\overline{a} \vee \overline{c}) \\
 &= (a \overline{a} \vee a \overline{b}) \wedge (\overline{b} \vee \overline{c}) \wedge (\overline{a} \vee \overline{c}) \\
 &= a \overline{b} \wedge (\overline{b} \vee \overline{c}) \wedge (\overline{a} \vee \overline{c}) \\
 &= (\overline{b} \vee \overline{c}) \wedge (a \overline{a} \overline{b} \vee a \overline{b} \overline{c}) \\
 &= (\overline{b} \vee \overline{c}) \wedge a \overline{b} \overline{c} \\
 &= a \overline{b} \overline{c} \vee a \overline{b} \overline{c} \\
 &= a \overline{b} \overline{c}
 \end{aligned}$$

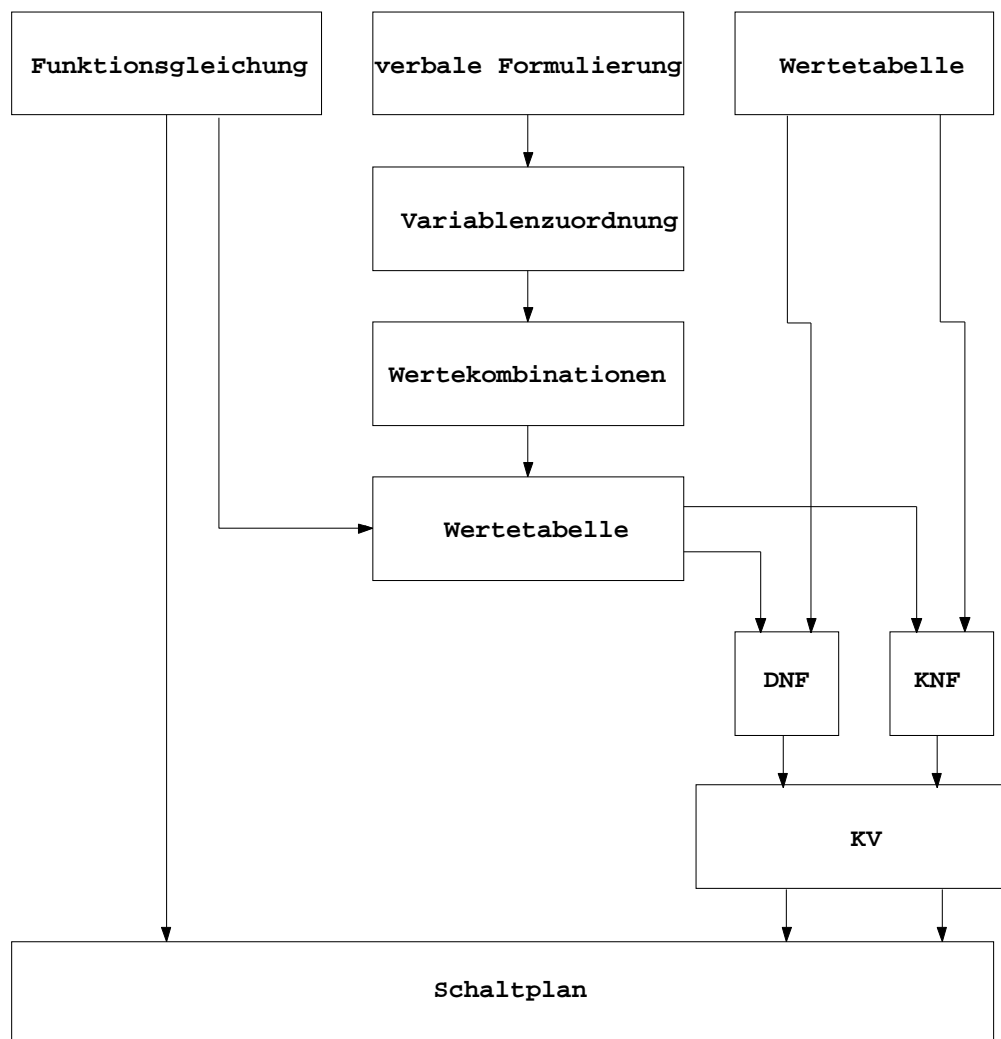
$$\begin{aligned}
 b \overline{z_6} &= b \wedge (\overline{a} \vee \overline{b}) \wedge (\overline{b} \vee \overline{c}) \wedge (\overline{a} \vee \overline{c}) \\
 &= (\overline{a} b \vee b \overline{b}) \wedge (\overline{b} \vee \overline{c}) \wedge (\overline{a} \vee \overline{c}) \\
 &= \overline{a} b \wedge (\overline{b} \vee \overline{c}) \wedge (\overline{a} \vee \overline{c}) \\
 &= (\overline{a} b \overline{b} \vee \overline{a} b \overline{c}) \wedge (\overline{a} \vee \overline{c}) \\
 &= \overline{a} b \overline{c} \wedge (\overline{a} \vee \overline{c}) \\
 &= \overline{a} \overline{a} b \overline{c} \vee \overline{a} b \overline{c} \\
 &= \overline{a} b \overline{c}
 \end{aligned}$$

$$\begin{aligned}
 c \overline{z_6} &= c \wedge (\overline{a} \vee \overline{b}) \wedge (\overline{b} \vee \overline{c}) \wedge (\overline{a} \vee \overline{c}) \\
 &= (\overline{a} \vee \overline{b}) \wedge (\overline{b} c \vee \overline{c} c) \wedge (\overline{a} \vee \overline{c}) \\
 &= (\overline{a} \vee \overline{b}) \wedge \overline{b} c \wedge (\overline{a} \vee \overline{c}) \\
 &= (\overline{a} \vee \overline{b}) \wedge (\overline{a} \overline{b} c \vee \overline{b} c \overline{c}) \\
 &= (\overline{a} \vee \overline{b}) \wedge \overline{a} \overline{b} c \\
 &= \overline{a} \overline{a} \overline{b} c \vee \overline{a} \overline{b} \overline{b} c \\
 &= \overline{a} \overline{b} c
 \end{aligned}$$

$$y_2 = abc \vee a \overline{b} \overline{c} \vee \overline{a} b \overline{c} \vee \overline{a} \overline{b} c$$

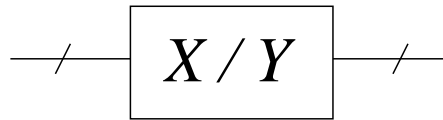
Synthese

► Synthese von Schaltnetzen

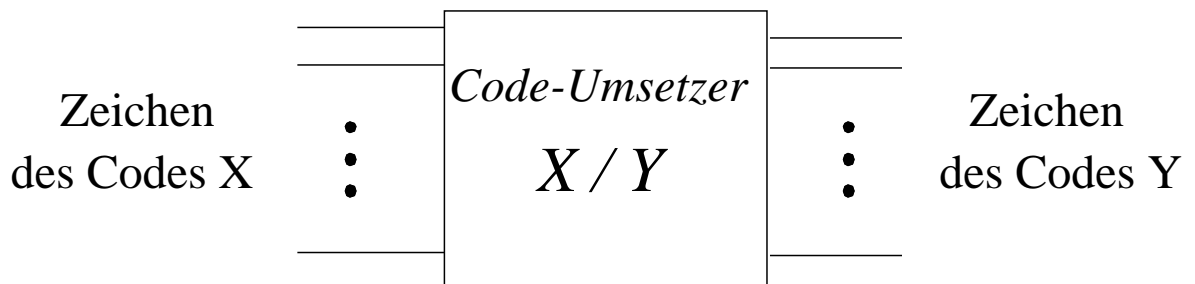


Code Umsetzer

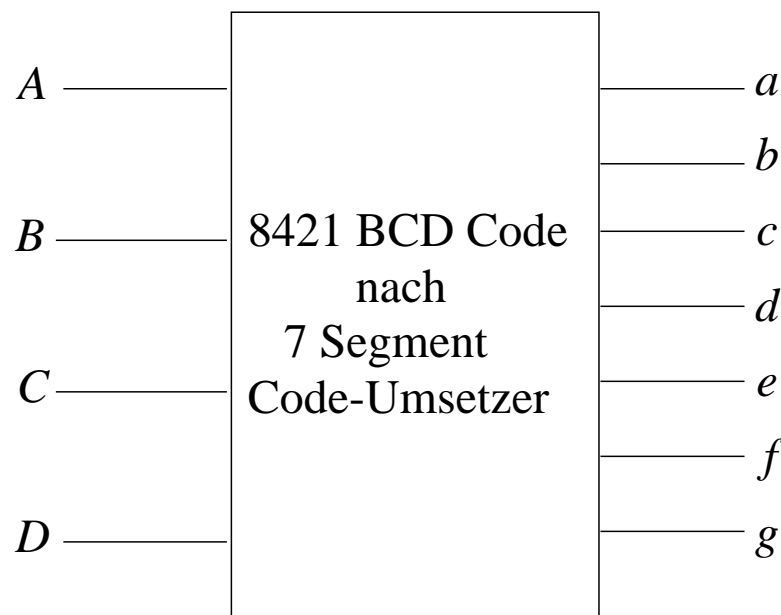
- ▶ Grundschriftzeichen (DIN 40700)



- ▶ Blockschaltbild eines Code-Umsetzer

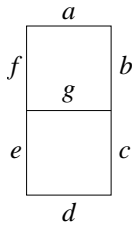
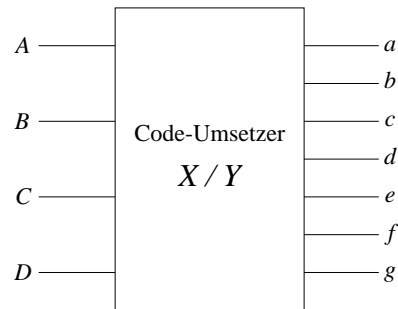


- ▶ Blockschaltbild eines 8421 BCD-Code zu 7-Segment-Code Umsetzer



Sieben Segment Synthese

► Beispiel: 8421 BCD Code → Sieben Segment Code



1 2 3 4 5 6 7 8 9 0

Dezimal Ziffer	8421-BCD				7-Segment-Code						
	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	x	x	x	x	x	x	x
11	1	0	1	1	x	x	x	x	x	x	x
12	1	1	0	0	x	x	x	x	x	x	x
13	1	1	0	1	x	x	x	x	x	x	x
14	1	1	1	0	x	x	x	x	x	x	x
15	1	1	1	1	x	x	x	x	x	x	x

Sieben Segment Synthese

- KV-Diagramm 7-Segment Anzeige, Segment a, DNF

BA \ DC	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	x	x	x	x
10	1	1	x	x

$$a = D \vee B \vee (A \wedge C) \vee (\bar{A} \wedge \bar{C})$$

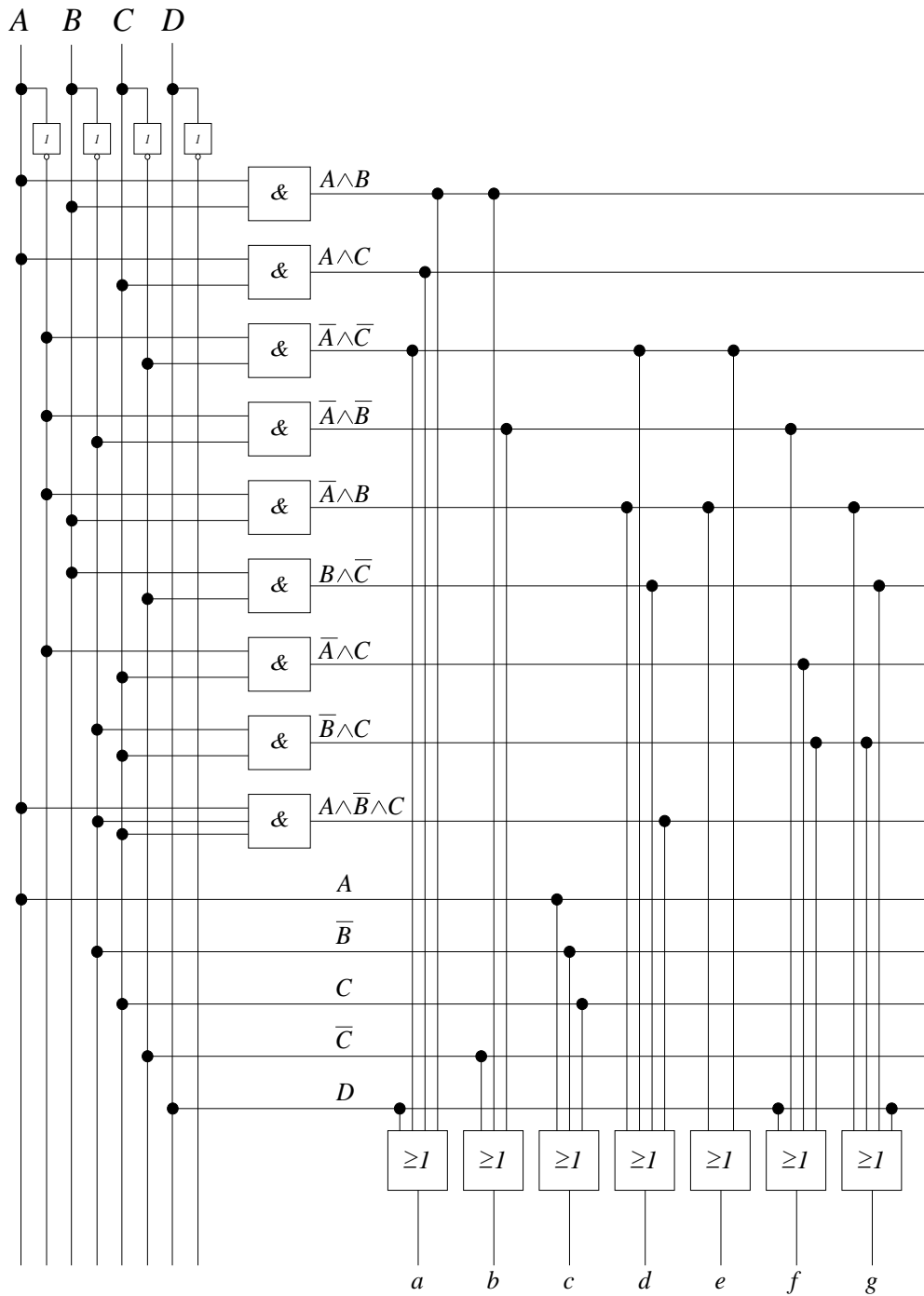
- KV-Diagramm 7-Segment Anzeige, Segment a, KNF

BA \ DC	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	x	x	x	x
10	1	1	x	x

$$a = (\bar{A} \vee B \vee C \vee D) \wedge (A \vee B \vee \bar{C})$$

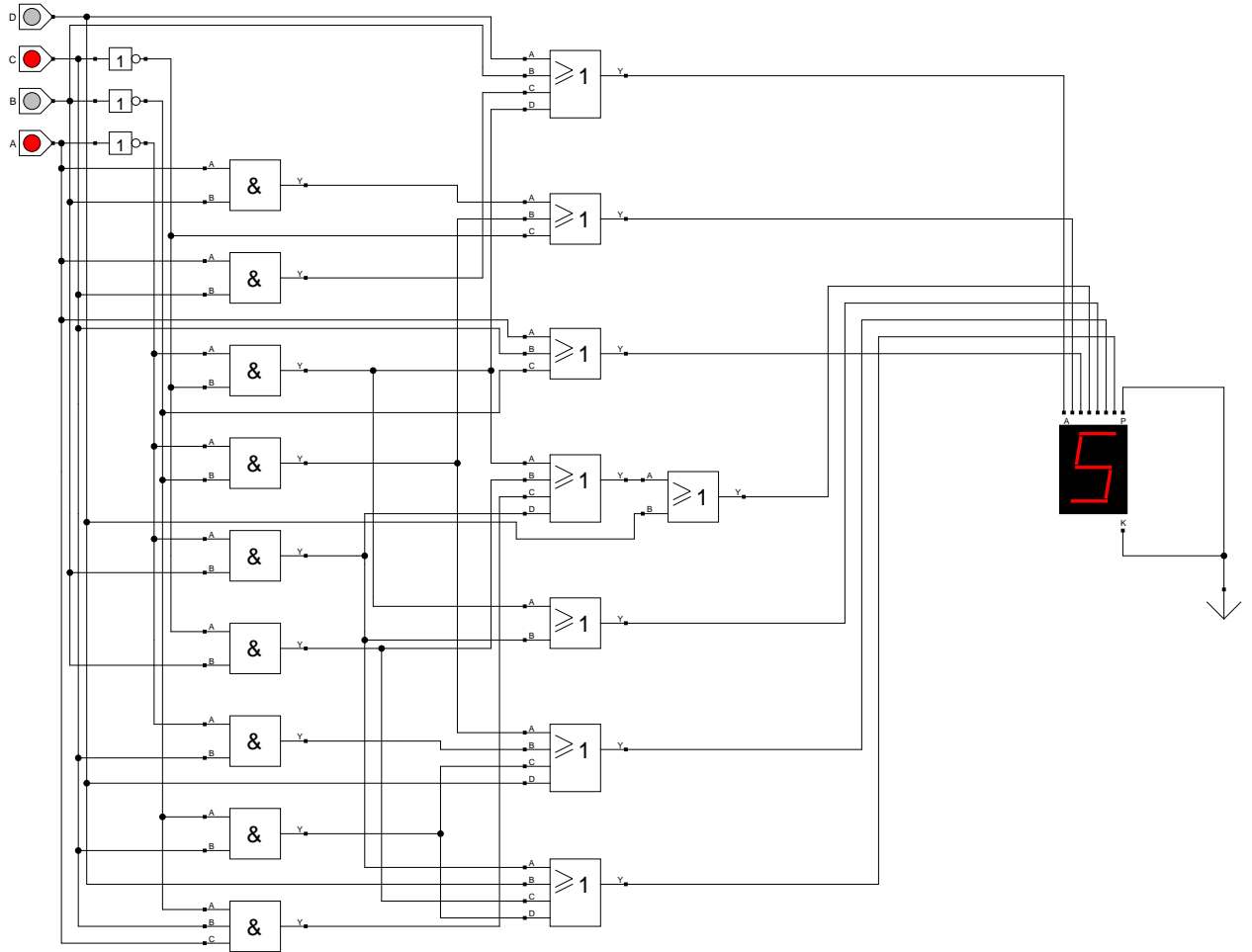
Sieben Segment Synthese

- Schaltnetz



Sieben Segment Synthese

► Schaltnetz (Hades: 7seg_sn.hds)



Gray-Code

► Beispiel: 8421 Dual Code - Gray Code

Dezimal Ziffer	8421-Dual-Code				Gray-Code			
	d_3	d_2	d_1	d_0	g_3	g_2	g_1	g_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

► KV-Diagramm g_0

$d_3 d_2$ \ $d_1 d_0$	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$$g_0 = d_0 \bar{d}_1 \vee \bar{d}_0 d_1 = d_0 \oplus d_1$$

Gray-Code

► KV-Diagramm g_1

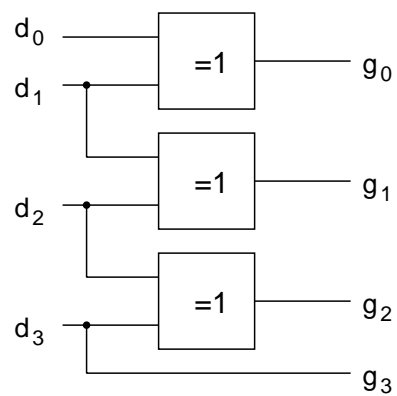
$d_1 d_0$ \ $d_3 d_2$	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

$$g_1 = d_1 \bar{d}_2 \vee \bar{d}_1 d_2 = d_1 \neq d_2$$

$$g_2 = d_2 \bar{d}_3 \vee \bar{d}_2 d_3 = d_2 \neq d_3$$

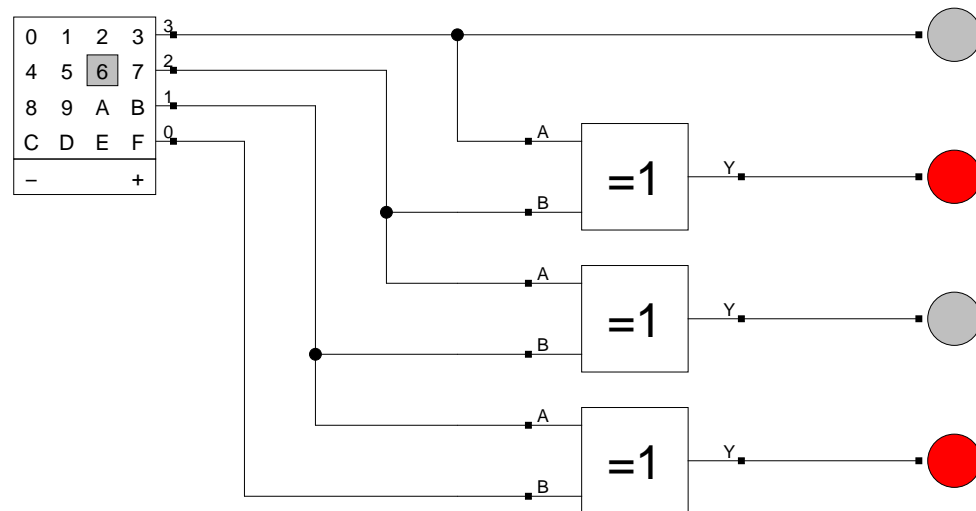
$$g_3 = d_3$$

► Schaltnetz Code-Umsetzer 8421-BCD zu Gray-Code



Gray-Code

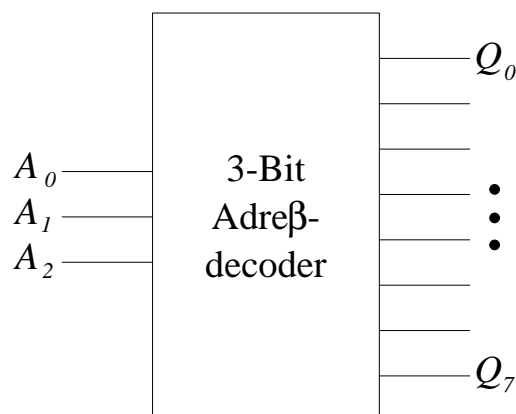
- Schaltnetz (Hades: graycode.hds)



Adressdecodierer

- ▶ Codierer (DIN 44 300):
 - Code-Umsetzer mit mehreren Ein- und Ausgängen, bei dem immer nur ein Eingang mit 1 belegt wird, und damit eine eindeutige Kombination von Ausgangssignalen erzeugt.
- ▶ Decodierer (DIN 44 300):
 - Code-Umsetzer mit mehreren Ein- und Ausgängen, bei dem eine eindeutige Kombination von Eingangssignalen immer nur an einem Ausgang eine 1 erzeugt.

A_2	A_1	A_0	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Adressdecodierer

► Minterme

$$Q_0 = \bar{A}_0 \wedge \bar{A}_1 \wedge \bar{A}_2$$

$$Q_1 = A_0 \wedge \bar{A}_1 \wedge \bar{A}_2$$

$$Q_2 = \bar{A}_0 \wedge A_1 \wedge \bar{A}_2$$

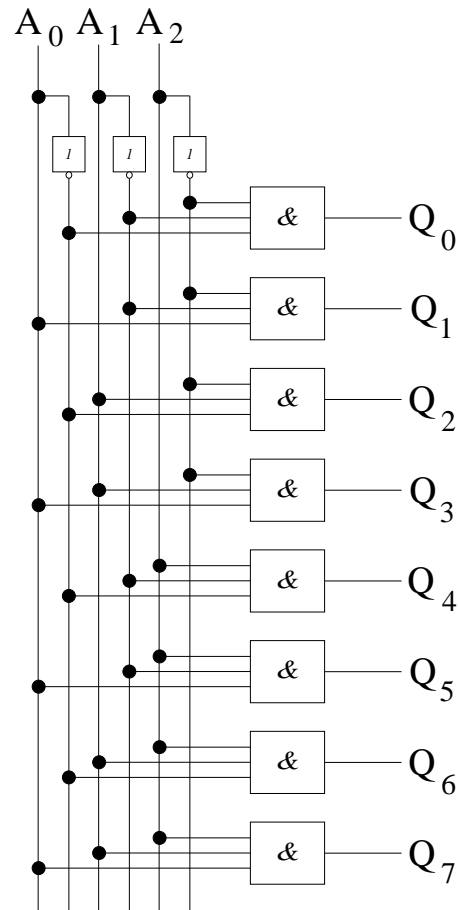
$$Q_3 = A_0 \wedge A_1 \wedge \bar{A}_2$$

$$Q_4 = \bar{A}_0 \wedge \bar{A}_1 \wedge A_2$$

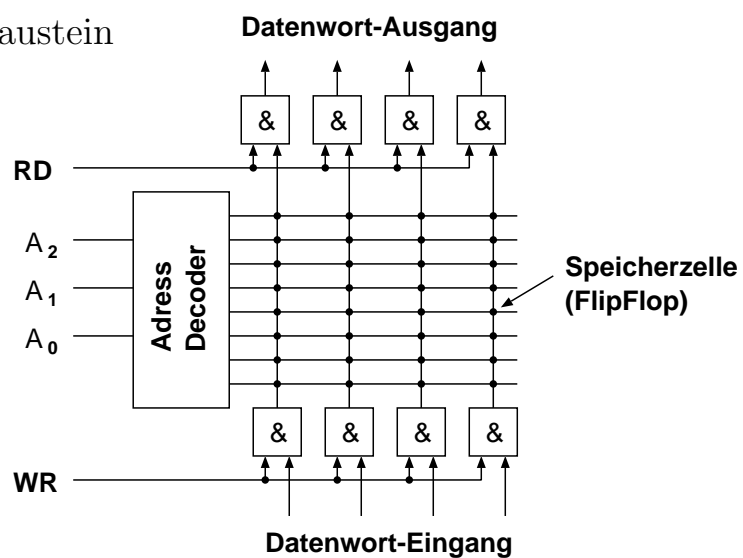
$$Q_5 = A_0 \wedge \bar{A}_1 \wedge A_2$$

$$Q_6 = \bar{A}_0 \wedge A_1 \wedge A_2$$

$$Q_7 = A_0 \wedge A_1 \wedge A_2$$

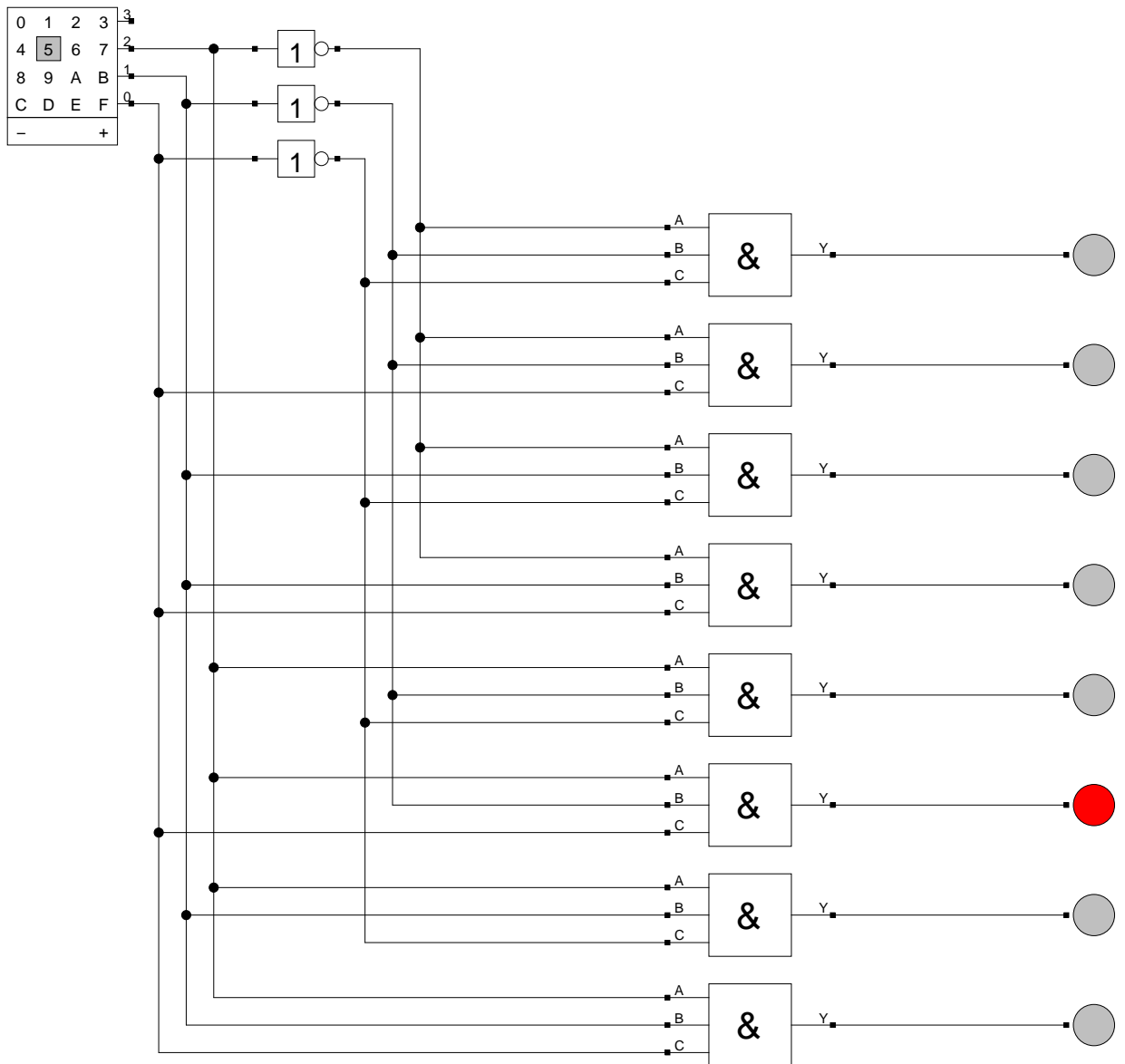


► Speicherbaustein



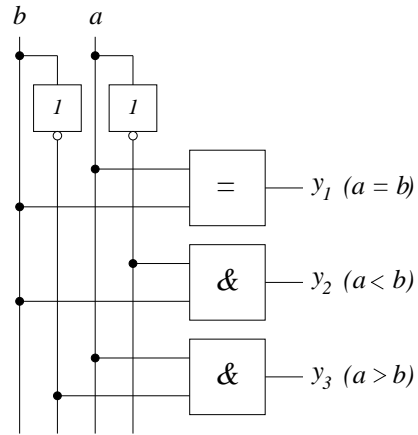
Adressdecodierer

► Schaltnetz (Hades: adressdecoder.hds)



Komparator

- 1Bit Komparator



- 2-Bit Komparator

B		A		Y_1	Y_2	Y_3
b_1	b_0	a_1	a_0	$A = B$	$A < B$	$A > B$
0	0	0	0	1	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	0	0	1
1	1	0	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	0	1	0
1	1	1	1	1	0	0

2-Bit Komparator

► $Y1 = (A = B)$

$a_1 a_0$	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

$$Y_1 = (a_0 \equiv b_0) \wedge (a_1 \equiv b_1)$$

► $Y2 = (A < B)$

$a_1 a_0$	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$$Y_2 = \bar{a}_0 \bar{a}_1 b_0 \vee \bar{a}_0 b_0 b_1 \vee \bar{a}_1 b_1$$

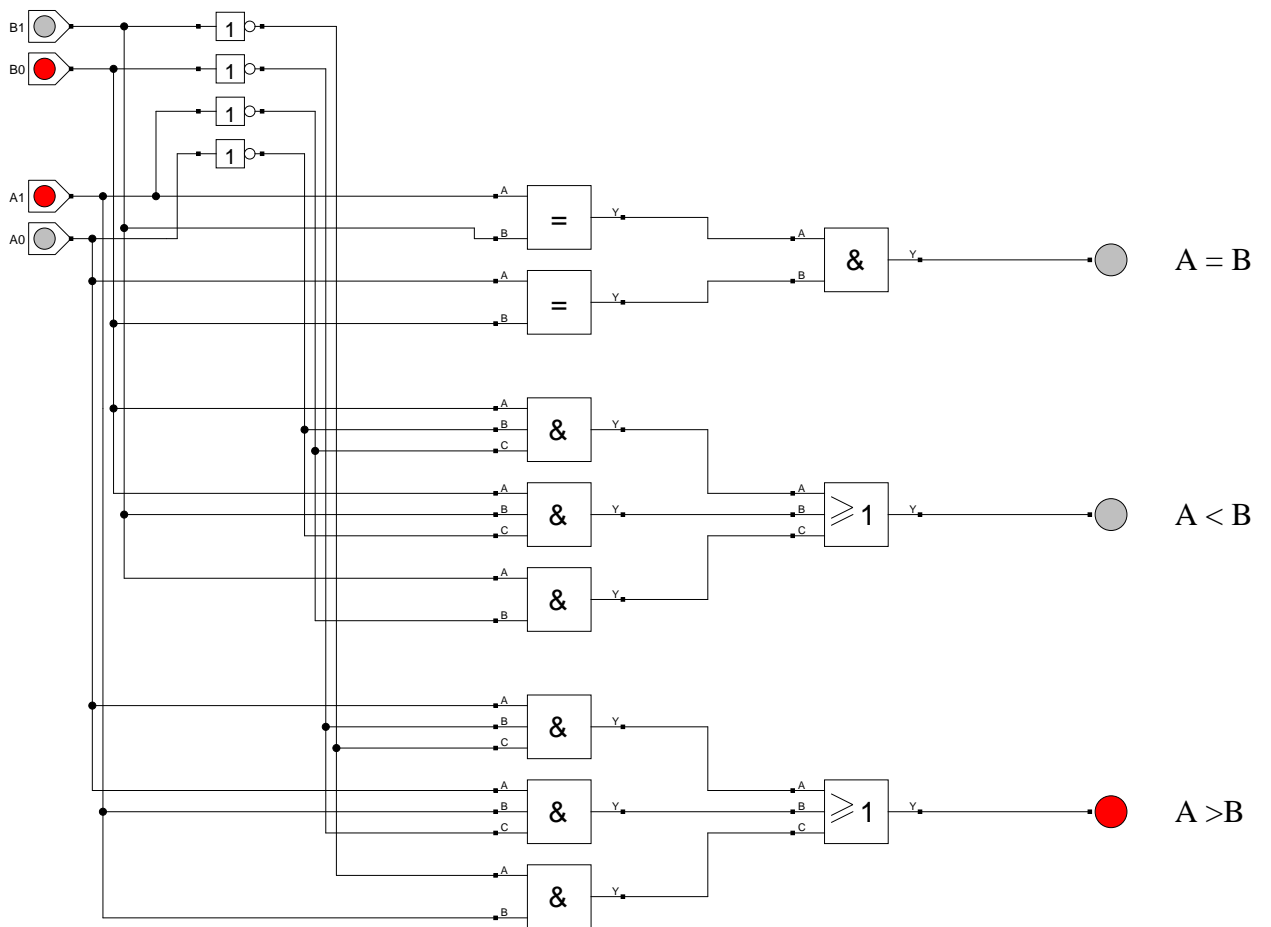
► $Y3 = (A > B)$

$a_1 a_0$	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

$$Y_3 = a_0 \bar{b}_0 \bar{b}_1 \vee a_0 a_1 \bar{b}_0 \vee a_1 \bar{b}_1$$

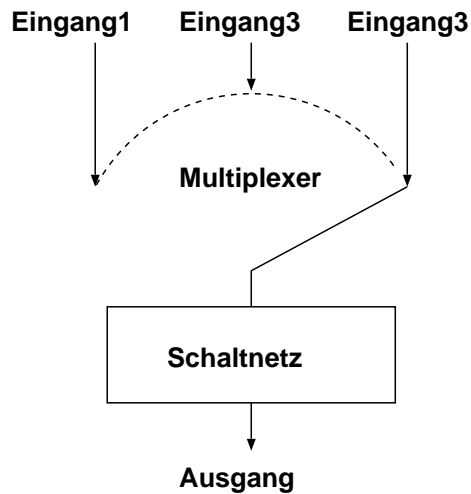
2-Bit Komparator

► Schaltnetz (Hades: 2bitkomp.hds)

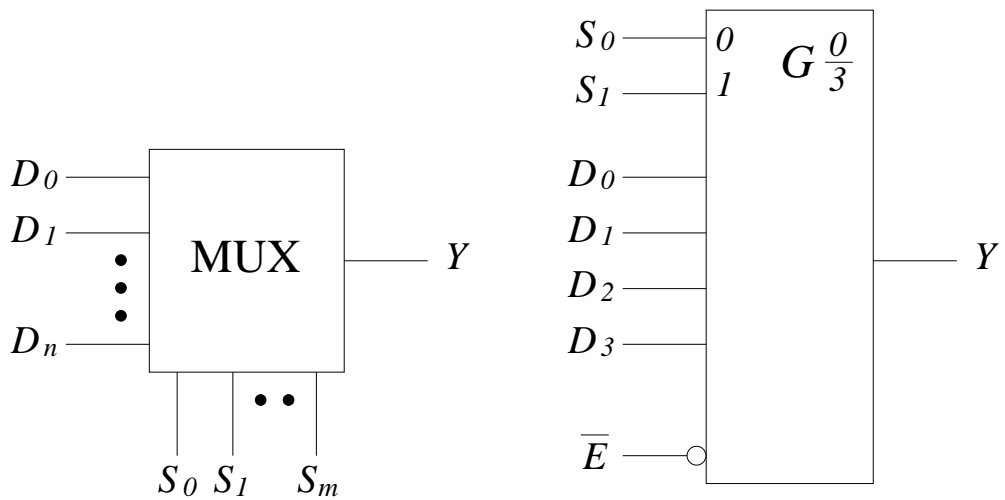


Multiplexer

- auswählendes Schaltnetz



- Blockschaltbild und Schaltzeichen

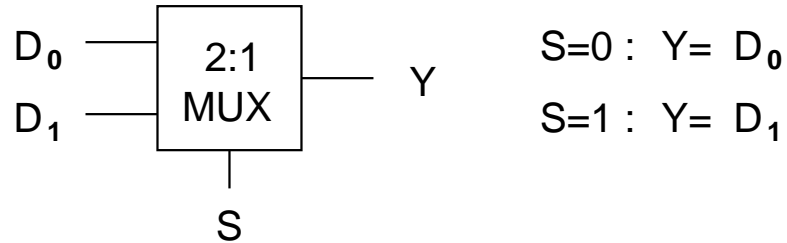


Blockschaltbild

Schaltzeichen

Synthese Multiplexer 2:1

- Beispiel: 2 zu 1 MUX



- Tabelle

S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

- KV-Tafel

$D_1 D_0$	00	01	11	10
s				
0	0	1	1	0
1	0	0	1	1

- Gleichung: $Y = \bar{S}D_0 \vee SD_1$

S	Y
0	D_0
1	D_1

Synthese Multiplexer 4:1

- Beispiel: 4 zu 1 MUX

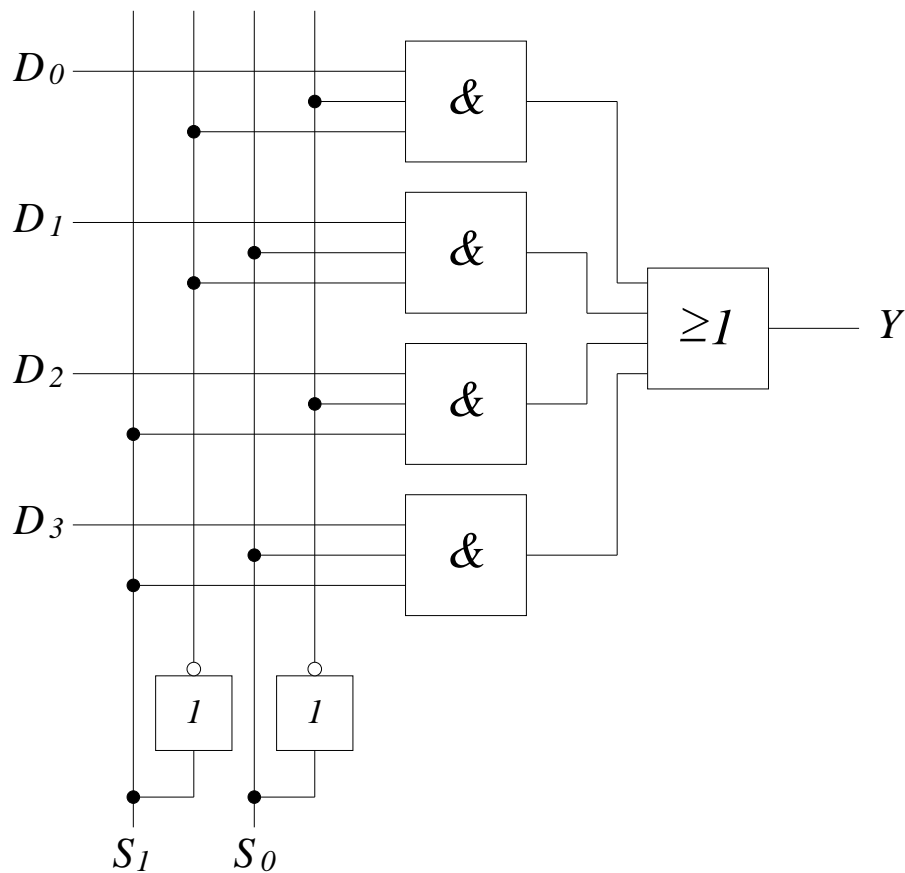
S_1	S_0	$Y =$
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

- Funktionsgleichung

$$y = D_0 m_0 \vee D_1 m_1 \vee D_2 m_2 \vee D_3 m_3$$

$$y = D_0 \bar{S}_1 \bar{S}_0 \vee D_1 \bar{S}_1 S_0 \vee D_2 S_1 \bar{S}_0 \vee D_3 S_1 S_0$$

- Schaltnetz 4 zu 1 MUX



Schaltungen mit Multiplexer

- ▶ Mit Multiplexer lassen sich alle Schaltfunktionen realisieren
 - Jede Kombination der Steuervariablen wird mit *einem* Dateneingang UND-verknüpft
 - Belegung der Steuereingänge mit (beliebigen) Eingangsvariablen der Funktion
 - Funktion mit den Wertekombinationen der Steuereingangsvariablen ermitteln
 - Ermittelte Funktion an den entsprechenden Dateneingang des Multiplexers anlegen

- ▶ Beispiel: $A < B$ (2Bit-Komparator)

- Steuervariable: a_0, b_0

a_0	b_0	$Y_2 = \bar{a}_0\bar{a}_1b_0 \vee \bar{a}_0b_0b_1 \vee \bar{a}_1b_1$	
0	0	$1\bar{a}_10 \vee 10b_1 \vee \bar{a}_1b_1 =$	$\bar{a}_1 \wedge b_1$
0	1	$1\bar{a}_11 \vee 11b_1 \vee \bar{a}_1b_1 = \bar{a}_1 \vee b_1 \vee \bar{a}_1b_1 =$	$\bar{a}_1 \vee b_1$
1	0	$0\bar{a}_10 \vee 00b_1 \vee \bar{a}_1b_1 =$	$\bar{a}_1 \wedge b_1$
1	1	$0\bar{a}_11 \vee 01b_1 \vee \bar{a}_1b_1 =$	$\bar{a}_1 \wedge b_1$

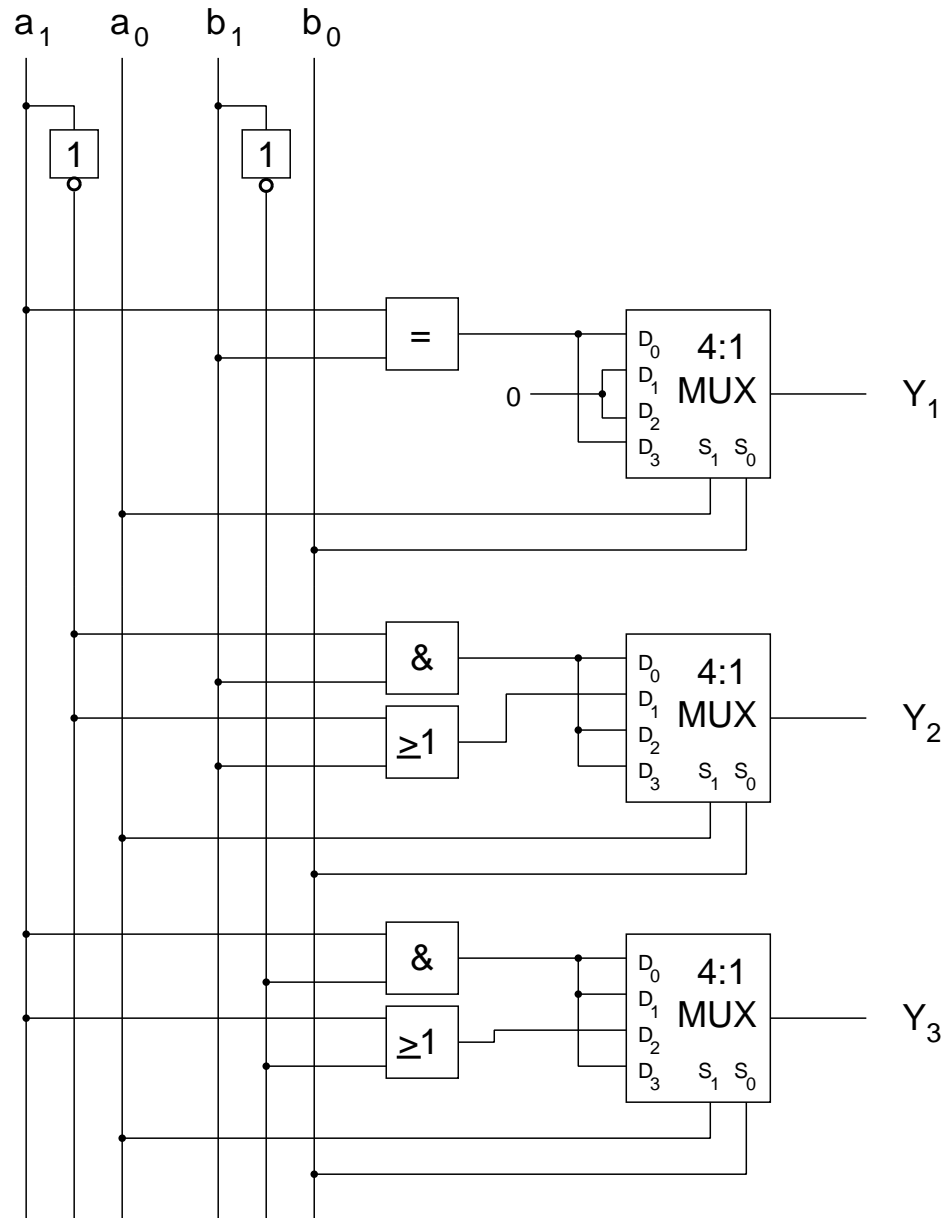
- ▶ Beispiel: $A = B$ (2Bit-Komparator)

- Steuervariable: a_0, b_0

a_0	b_0	$Y_1 = (a_0b_0 \vee \bar{a}_0\bar{b}_0) \wedge (a_1b_1 \vee \bar{a}_1\bar{b}_1)$	
0	0	$(00 \vee 11) \wedge (a_1b_1 \vee \bar{a}_1\bar{b}_1) =$	$a_1 \equiv b_1$
0	1	$(01 \vee 10) \wedge (a_1b_1 \vee \bar{a}_1\bar{b}_1) =$	0
1	0	$(10 \vee 01) \wedge (a_1b_1 \vee \bar{a}_1\bar{b}_1) =$	0
1	1	$(11 \vee 00) \wedge (a_1b_1 \vee \bar{a}_1\bar{b}_1) =$	$a_1 \equiv b_1$

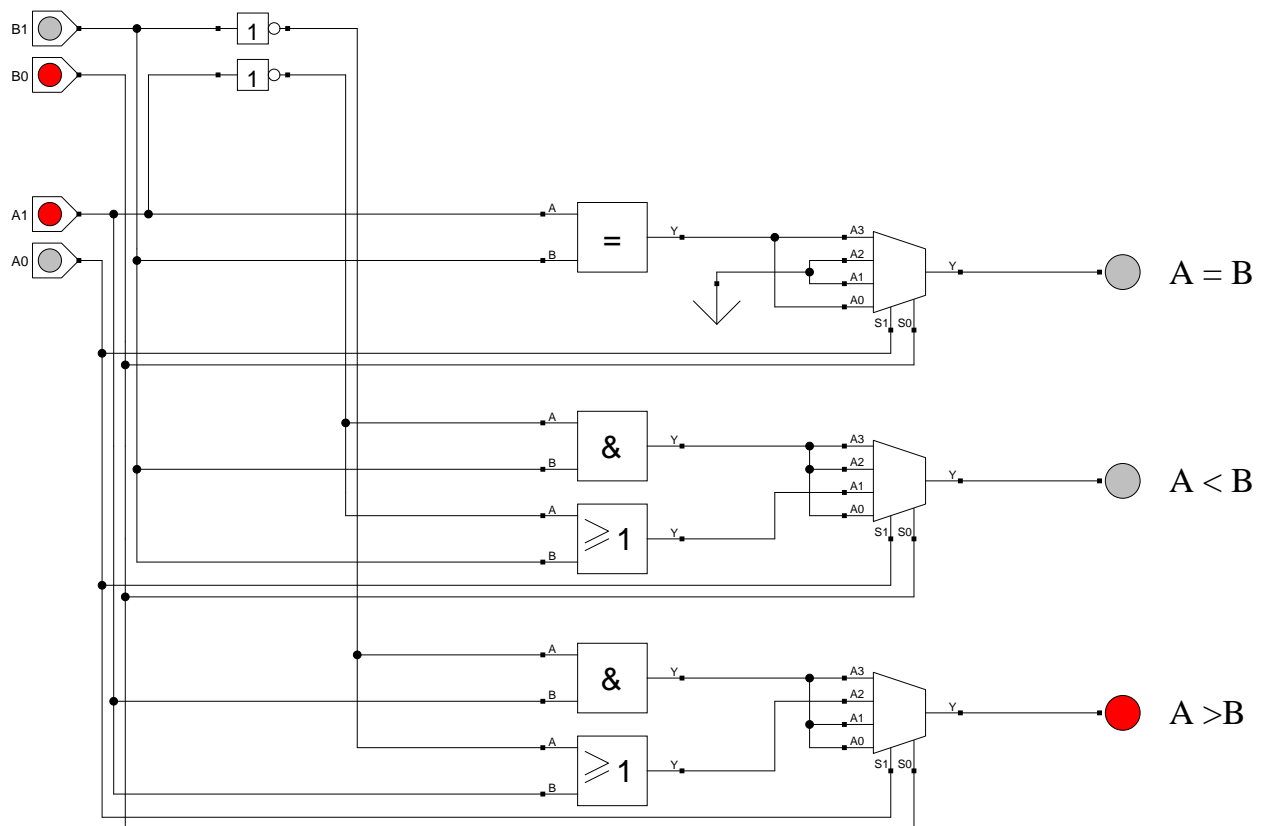
Schaltungen mit Multiplexer

► Schaltnetz



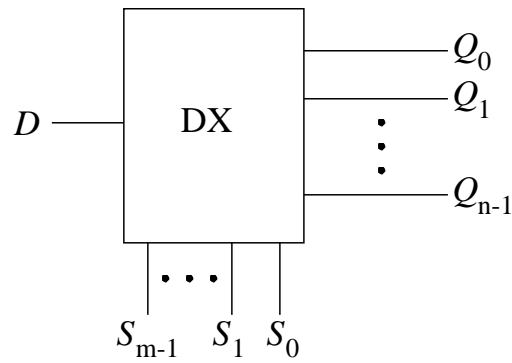
2-Bit Komparator mit Multiplexer

- Schaltnetz (Hades: muxkomp.hds)



Demultiplexer

- 1 zu 4 DEMUX

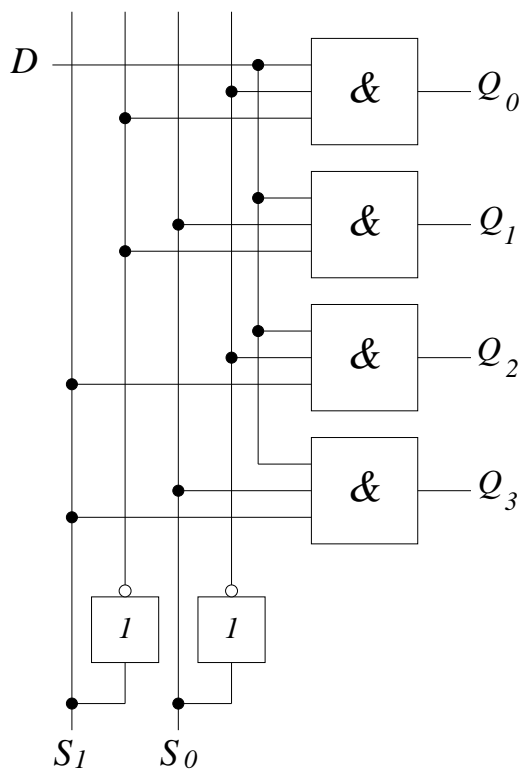


a) Blockschaltbild

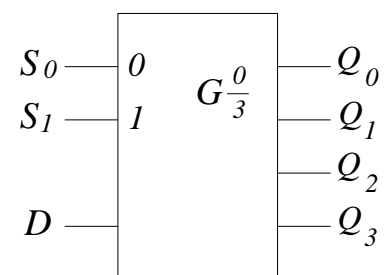
S_1	S_0	Q_0	Q_1	Q_2	Q_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

b) Wertetabelle

- Schaltnetz 1 zu 4 DEMUX



Schaltnetz



Schaltzeichen

Halbaddierer

► Tabelle

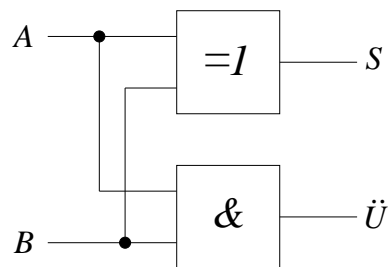
<i>A</i> plus <i>B</i>	<i>S</i>	<i>Ü</i>
0 plus 0	0	0
0 plus 1	1	0
1 plus 0	1	0
1 plus 1	0	1

► Schaltfunktionen

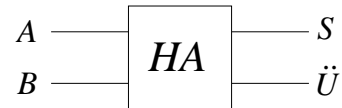
$$S = (A \wedge \bar{B}) \vee (\bar{A} \wedge B) = A \neq B$$

$$\ddot{U} = A \wedge B$$

► Schaltnetz



Schaltnetz



Schaltzeichen

► für mehrstellige Addition Addierer mit Übertragseingang nötig

Volladdierer

► Tabelle

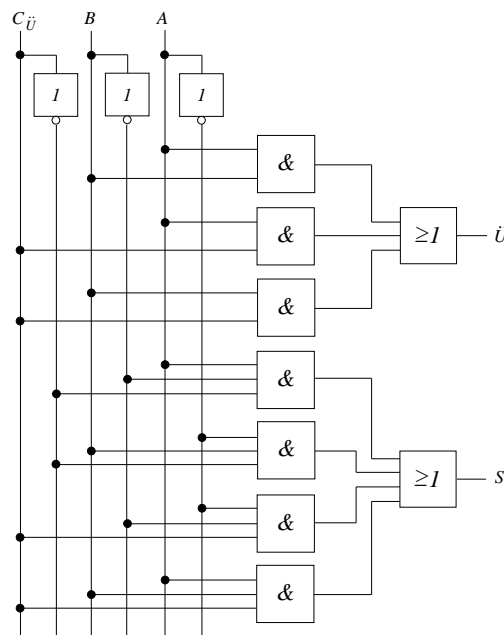
$C_{\ddot{u}}$	B	A	S	\ddot{U}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

► Schaltfunktionen

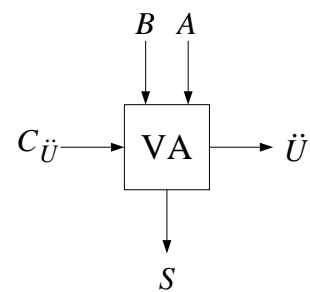
$$\begin{aligned}
 S &= (A \wedge \bar{B} \wedge \bar{C}_{\ddot{u}}) \vee (\bar{A} \wedge B \wedge \bar{C}_{\ddot{u}}) \vee (\bar{A} \wedge \bar{B} \wedge C_{\ddot{u}}) \vee (A \wedge B \wedge C_{\ddot{u}}) \\
 &= A \neq B \neq C_{\ddot{u}}
 \end{aligned}$$

$$\begin{aligned}
 \ddot{U} &= (A \wedge B) \vee (A \wedge C_{\ddot{u}}) \vee (B \wedge C_{\ddot{u}}) \\
 &= (A \wedge B) \vee [(A \vee B) \wedge C_{\ddot{u}}]
 \end{aligned}$$

► Schaltnetz



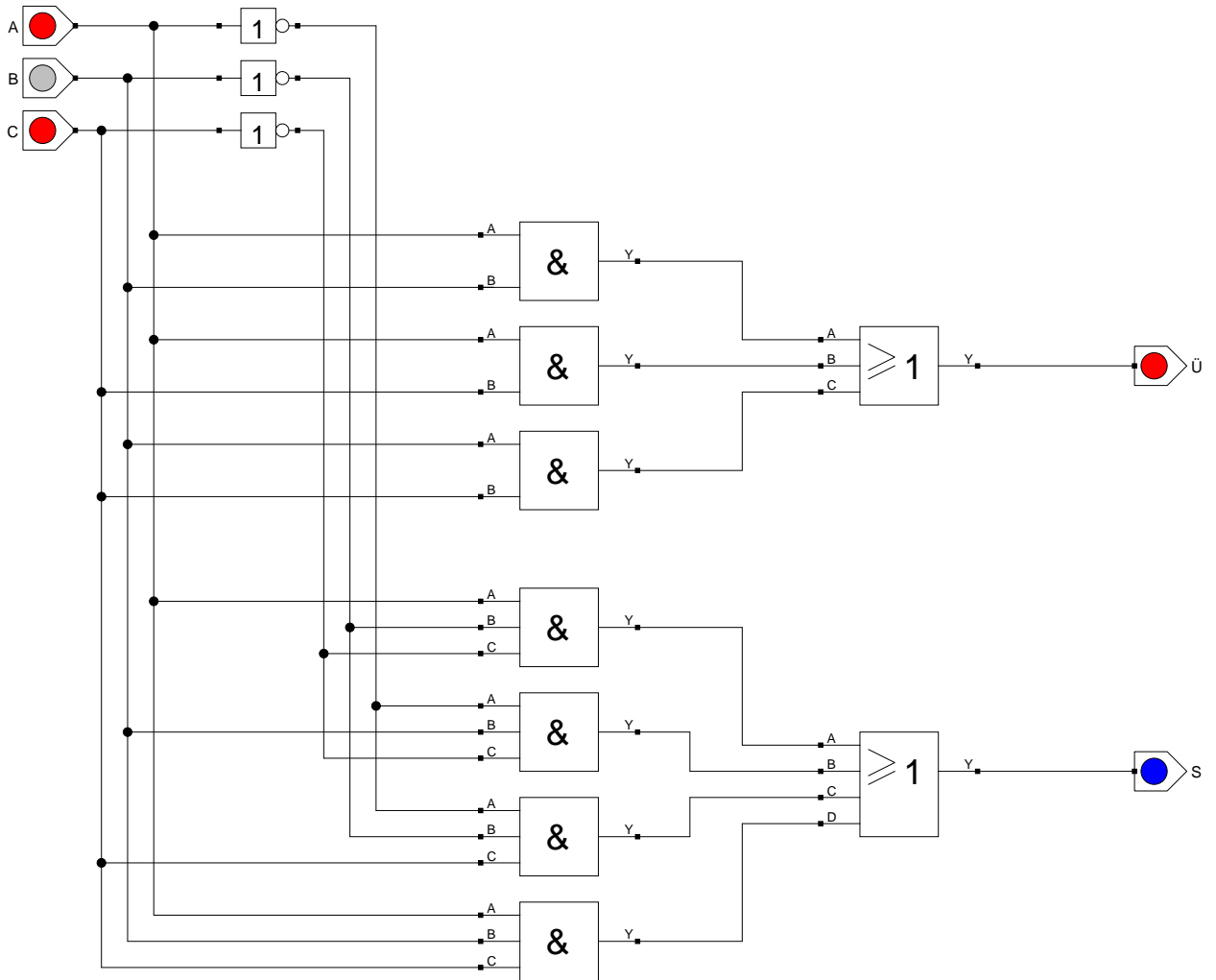
Schaltnetz



Schaltzeichen

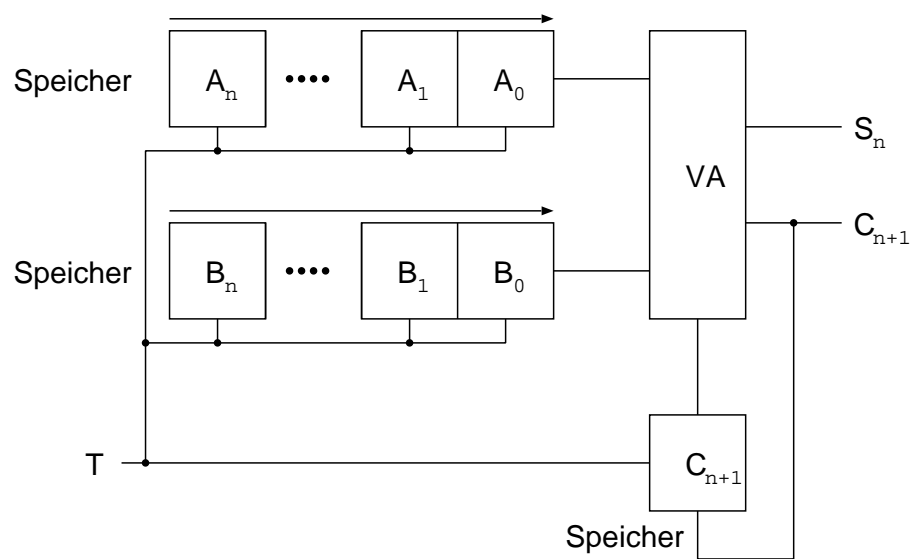
Volladdierer

► va.hds



Serienaddierer

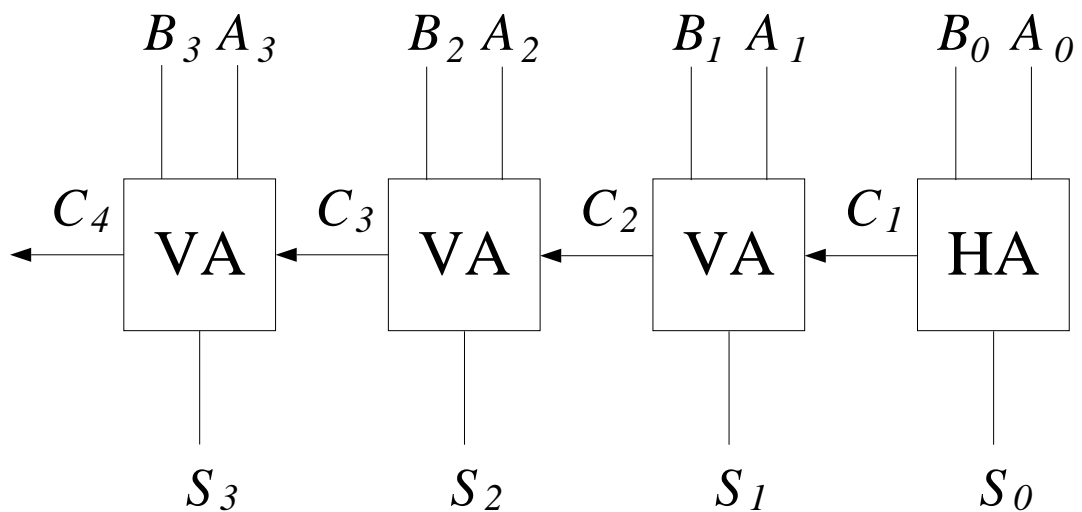
- ▶ ein VA
- ▶ Speicherbausteine zur Aufnahme der Summanden
- ▶ Bitweises addieren der Summanden
- ▶ Speicherbaustein zur Aufnahme des Übertrags



- ▶ minimaler Hardwareaufwand
- ▶ Taktsignal
- ▶ maximale Ausführungszeit

Paralleladdierer

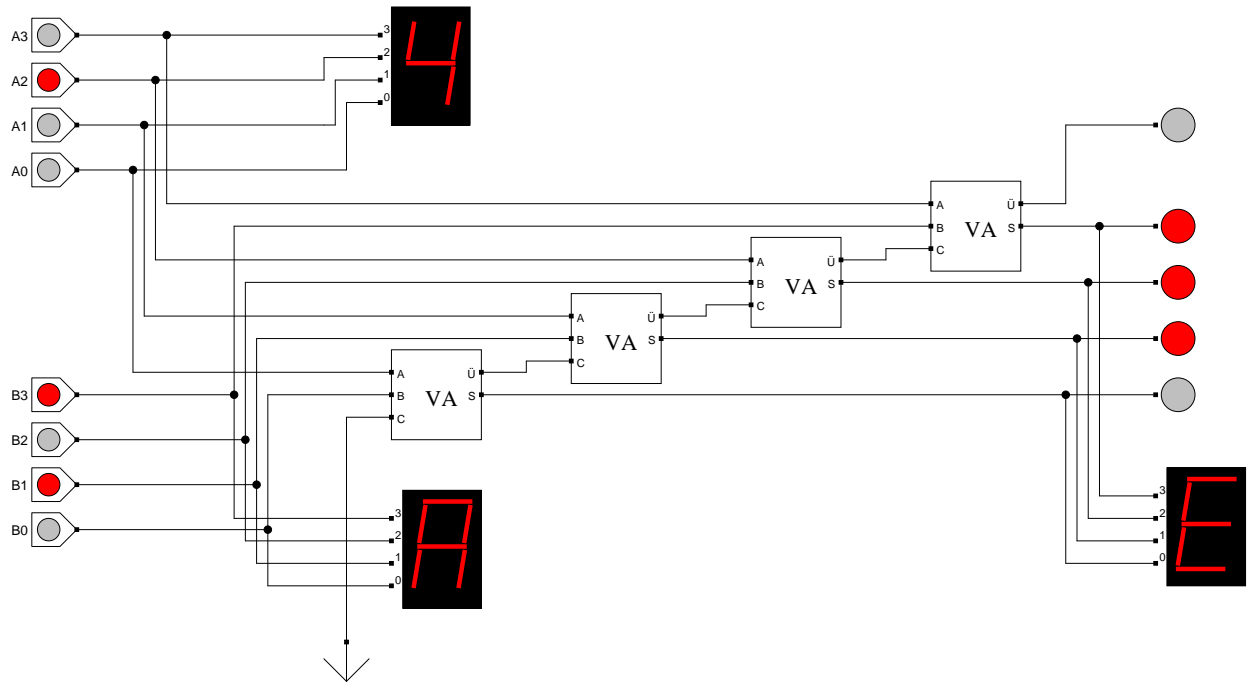
- ▶ Für jede Stelle ein addierendes Schaltnetz (paralleles Berechnen)
- ▶ Ausführungen
 - Ripple-Carry Adder
 - Normalform Paralleladdierer
 - Carry-Look-Ahead Adder
- ▶ Ripple-Carry Adder
 - für jede Stelle ein VA
 - Jede Stelle berechnet Summe und Übertrag aus den Stellenbits und dem Übertrag des vorgeschalteten VA
 - jeder VA muss auf die Gültigkeit des eingehenden Übertrags warten



- geringer Hardwareaufwand
- lange Ausführungszeit

Ripple-Carry Adder

► 4bitadder.hds



Paralleladdierer

► Normalform–Paralleladdierer

- Dreistufiges Schaltnetz aus NICHT, ODER, und UND Gattern
- Beispiel 2 Bit (5 Eingangsvariablen A_1, A_0, B_1, B_0, C_0)

$$\begin{aligned}
 S_0 &= A_0 \oplus B_0 \oplus C_0 \\
 &= (A_0 \bar{B}_0 \bar{C}_0) \vee (\bar{A}_0 B_0 \bar{C}_0) \vee (\bar{A}_0 \bar{B}_0 C_0) \vee (A_0 B_0 C_0) \\
 C_1 &= (A_0 B_0) \vee (A_0 C_0) \vee (B_0 C_0) \\
 S_1 &= A_1 \oplus B_1 \oplus C_1 \\
 &= A_1 \oplus B_1 \oplus [(A_0 B_0) \vee (A_0 C_0) \vee (B_0 C_0)] \\
 &= [(A_1 \bar{B}_1) \vee (\bar{A}_1 B_1)] \oplus [(A_0 B_0) \vee (A_0 C_0) \vee (B_0 C_0)] \\
 &= \vdots \\
 &= (\bar{A}_1 B_1 \bar{B}_0 \bar{C}_0) \vee (\bar{A}_1 \bar{A}_0 B_1 \bar{B}_0) \vee (\bar{A}_1 \bar{A}_0 B_1 \bar{C}_0) \vee (\bar{A}_1 \bar{B}_1 B_0 C_0) \vee \\
 &\quad (\bar{A}_1 A_0 \bar{B}_1 B_0) \vee (\bar{A}_1 A_0 \bar{B}_1 C_0) \vee (A_1 \bar{B}_1 \bar{B}_0 \bar{C}_0) \vee (A_1 \bar{A}_0 \bar{B}_1 \bar{B}_0) \vee \\
 &\quad (A_1 \bar{A}_0 \bar{B}_1 \bar{C}_0) \vee (A_1 A_0 B_1 C_0) \vee (A_1 A_0 B_1 B_0) \vee (A_1 B_1 B_0 C_0) \\
 C_2 &= (A_1 B_1) \vee (A_1 C_1) \vee (B_1 C_1) \\
 &= (A_1 B_1) \vee (A_1 [(A_0 B_0) \vee (A_0 C_0) \vee (B_0 C_0)]) \vee \\
 &\quad (B_1 [(A_0 B_0) \vee (A_0 C_0) \vee (B_0 C_0)]) \\
 &= (A_1 B_1) \vee (A_1 A_0 B_0) \vee (A_1 A_0 C_0) \vee (A_1 B_0 C_0) \vee \\
 &\quad (A_0 B_0 B_1) \vee (A_0 B_1 C_0) \vee (B_0 B_1 C_0)
 \end{aligned}$$

- n -Stellen
 - $2n + 1$ Eingänge
 - $n + 1$ Ausgänge
 - Anzahl Gatter $O(2^n)$
- maximaler Hardwareaufwand (nicht realisierbar)
- minimaler Ausführungszeit

Paralleladdierer

► Carry–Look–Ahead Adder

- Kompromiss

- Für jeden VA wird der eingehende Übertrag in einem eigenen SN berechnet
- Kein Warten auf vorgeschaltete VAs

- 5-stufiges Schaltnetz

- 2 Stufen für den eingehenden Übertrag C_i
- 3 Stufen für die Summe S_i

- Beispiel 2 Bit (5 Eingangsvariablen A_1, A_0, B_1, B_0, C_0)

$$\begin{aligned}
 S_0 &= A_0 \oplus B_0 \oplus C_0 \\
 &= (A_0 \bar{B}_0 \bar{C}_0) \vee (\bar{A}_0 B_0 \bar{C}_0) \vee (\bar{A}_0 \bar{B}_0 C_0) \vee (A_1 B_1 C_0) \\
 C_1 &= (A_0 B_0) \vee (A_0 C_0) \vee (B_0 C_0) \\
 S_1 &= A_1 \oplus B_1 \oplus C_1 \\
 &= (A_1 \bar{B}_1 \bar{C}_1) \vee (\bar{A}_1 B_1 \bar{C}_1) \vee (\bar{A}_1 \bar{B}_1 C_1) \vee (A_1 B_1 C_1) \\
 C_2 &= (A_1 B_1) \vee (A_1 A_0 B_0) \vee (A_1 A_0 C_0) \vee (A_1 B_0 C_0) \vee \\
 &\quad (A_0 B_0 B_1) \vee (A_0 B_1 C_0) \vee (B_0 B_1 C_0)
 \end{aligned}$$

- moderater Hardwareaufwand
- kurze Ausführungszeit

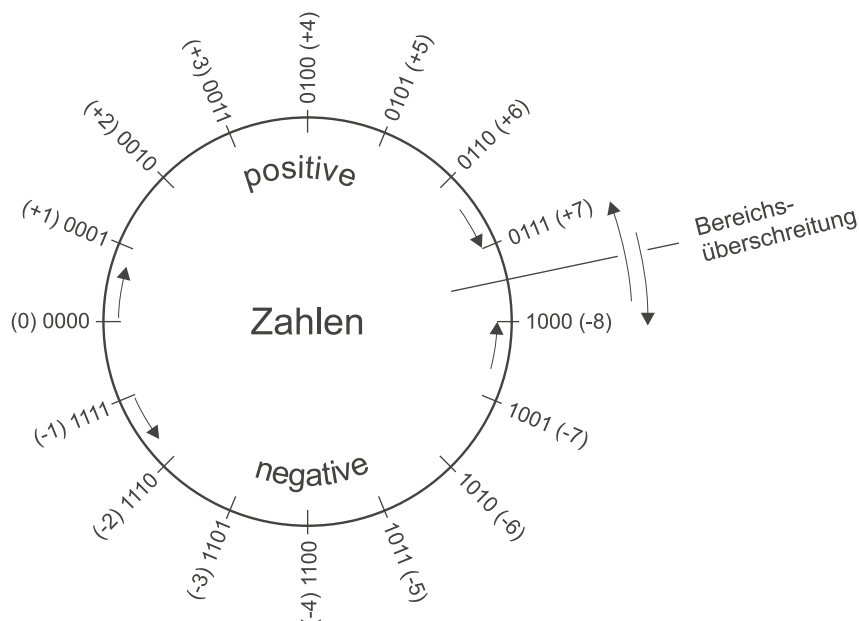
Zahendarstellung

- ▶ Voraussetzung: n -stellige Verarbeitungseinheit (zB. n -Bit VA)
- ▶ Addition einer n -stelligen Zahl A mit ihrem Komplement \bar{A}
 - Ergebnis: n -stellige Zahl der Form: 1111...11
- ▶ Durch Addition einer 1 erhält man eine $n + 1$ stellige Zahl mit
 - 1 im höchstwertigen ($n+1$) Bit
 - alle anderen Stellen sind 0
- ▶ da die $n + 1$ Stelle nicht dargestellt werden kann, ist das Ergebnis 0

$$A + \bar{A} + 1 = 0$$

$$-A = \bar{A} + 1$$

- ▶ $\bar{A} + 1$ ist somit eine Darstellung für $-A$
- ▶ Definition
 - \bar{A} (Einer)Komplement
 - $\bar{A} + 1$ Zweierkomplement
- ▶ Eigenschaften
 - höchstwertige Bit als Vorzeichen, 0 positiv, 1 negativ
 - restlichen Bits stellen den Zahlenwert dar



Subtraktion

- ▶ mit Hilfe des Zweierkomplements:

$$A - B = A + \bar{B} + 1$$

- ▶ Beispiel '5-3' mit einem 4-Bit VA

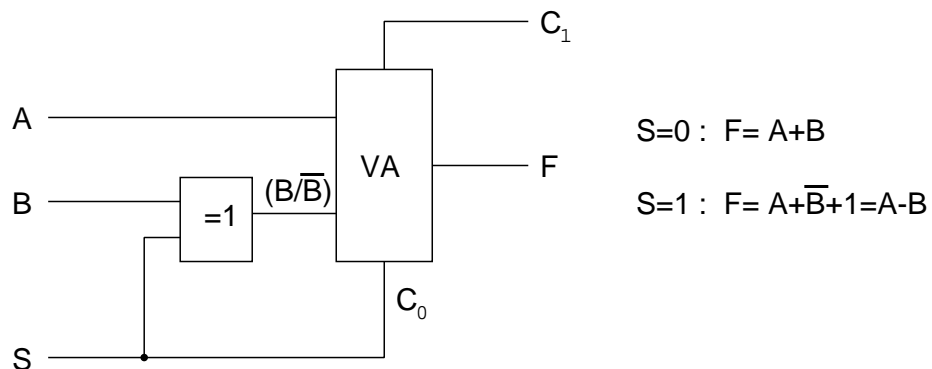
$$\begin{aligned} 5 &= 0101 \\ 3 &= 0011 \\ \bar{3} &= 1100 \\ 5 - 3 &= 0101 + 1100 + 1 \\ &= 1|0010 \\ 0010 &= 2 \end{aligned}$$

- ▶ SN zur (Einer)Komplementbildung

S	B	B/\bar{B}
0	0	0
0	1	1
1	0	1
1	1	0

$$B/\bar{B} = B \neq S$$

- ▶ Realisierung: 1 Bit Addierer/Subtrahierer



Bereichsüberschreitung

- ▶ Aufgrund des beschränkten Zahlenbereichs kann es zu einer Bereichsüberschreitung (*Overflow*) kommen
 - Zahlen werden zu gross oder zu klein
 - Overflow muss erkannt werden
 - kann nur auftreten wenn zwei positive (negative) Zahlen addiert werden ($A_{n-1} \equiv B_{n-1}$)

▶ Beispiele:

- Addition positiver Zahlen

$$\begin{array}{r|l}
 \textit{kein Overflow} & \\
 5 & 0101 \\
 2 & 0010 \\
 C\ 0 & 0000 \\
 \hline
 0 & 0111 = 7
 \end{array}$$

$$\begin{array}{r|l}
 \textit{Overflow} & \\
 5 & 0101 \\
 4 & 0100 \\
 C\ 0 & 1000 \\
 \hline
 0 & 1001 = -7
 \end{array}$$

- Addition negativer Zahlen

$$\begin{array}{r|l}
 \textit{kein Overflow} & \\
 -2 & 1110 \\
 -3 & 1101 \\
 C\ 1 & 1000 \\
 \hline
 1 & 1011 = -5
 \end{array}$$

$$\begin{array}{r|l}
 \textit{Overflow} & \\
 -5 & 1011 \\
 -6 & 1010 \\
 C\ 1 & 0100 \\
 \hline
 1 & 0101 = 5
 \end{array}$$

▶ Overflow $V = C_{n-1} \neq C_n$

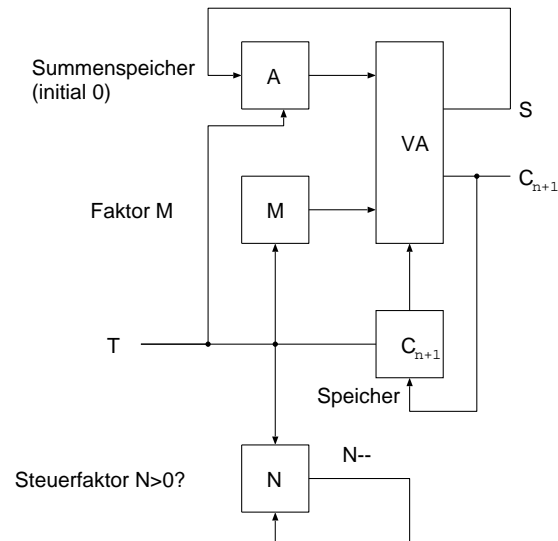
- Zur Herleitung des Overflow siehe Band II, von Neumann Rechner

▶ Alternativ: $V = (A_{n-1} \equiv B_{n-1}) \wedge (S_{n-1} \neq C_n)$

Multiplikation

► Serienmultiplizierer

- $m \cdot n$: addiere n mal die Zahl m



- Dauer der Operation vom Parameter abhängig

► Parallelmultiplizierer

- Normalform Multiplizierer
- s. Normalform Paralleladdierer

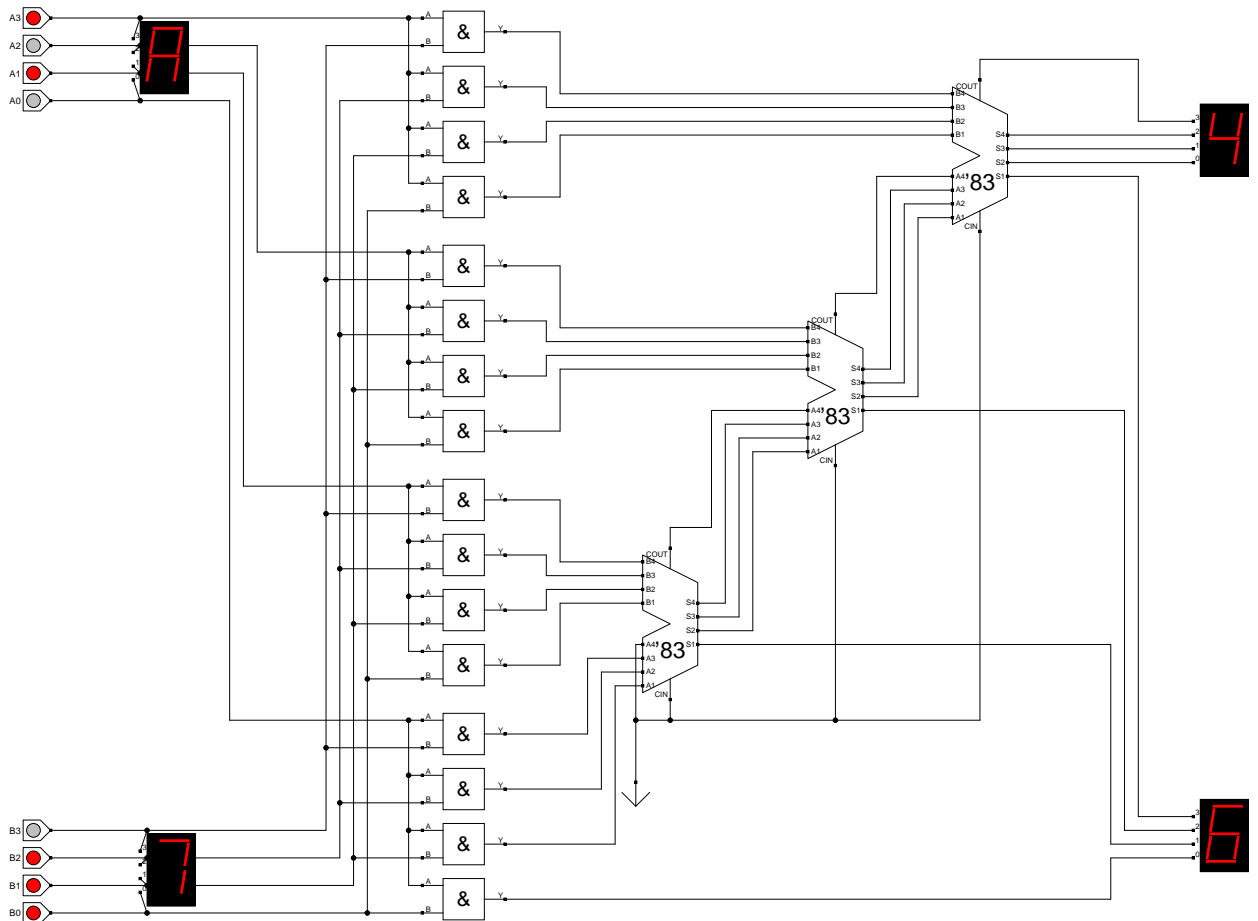
► Kompromiss:

- wie bei der schriftlichen Multiplikation
- Bsp: $6 \cdot 5 = 30$

$$\begin{array}{r}
 1 \ 1 \ 0 \ * \ 1 \ 0 \ 1 \\
 \hline
 1 \ 1 \ 0 \\
 0 \ 0 \ 0 \\
 1 \ 1 \ 0 \\
 \hline
 1 \ 1 \ 1 \ 1 \ 0
 \end{array}$$

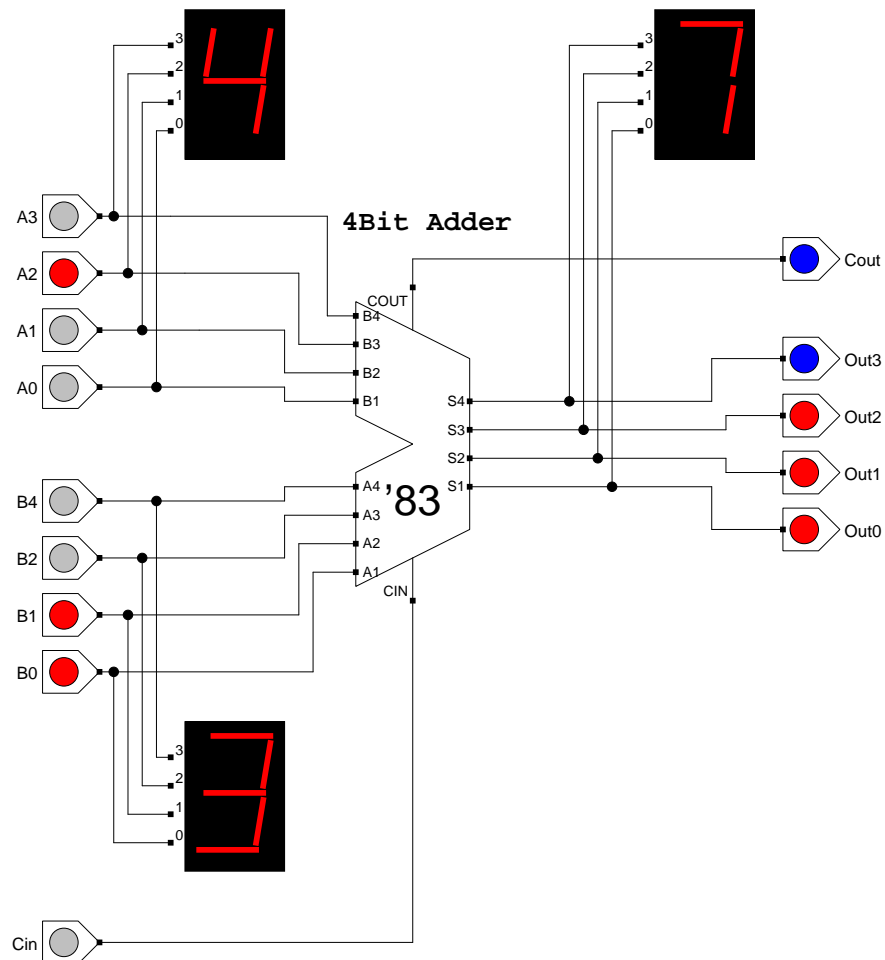
4 Bit Hardware Multiplizierer

- ▶ Verfahren der effizienten Multiplikationn
- ▶ Kompromiss aus Hardwareaufwand und Laufzeit



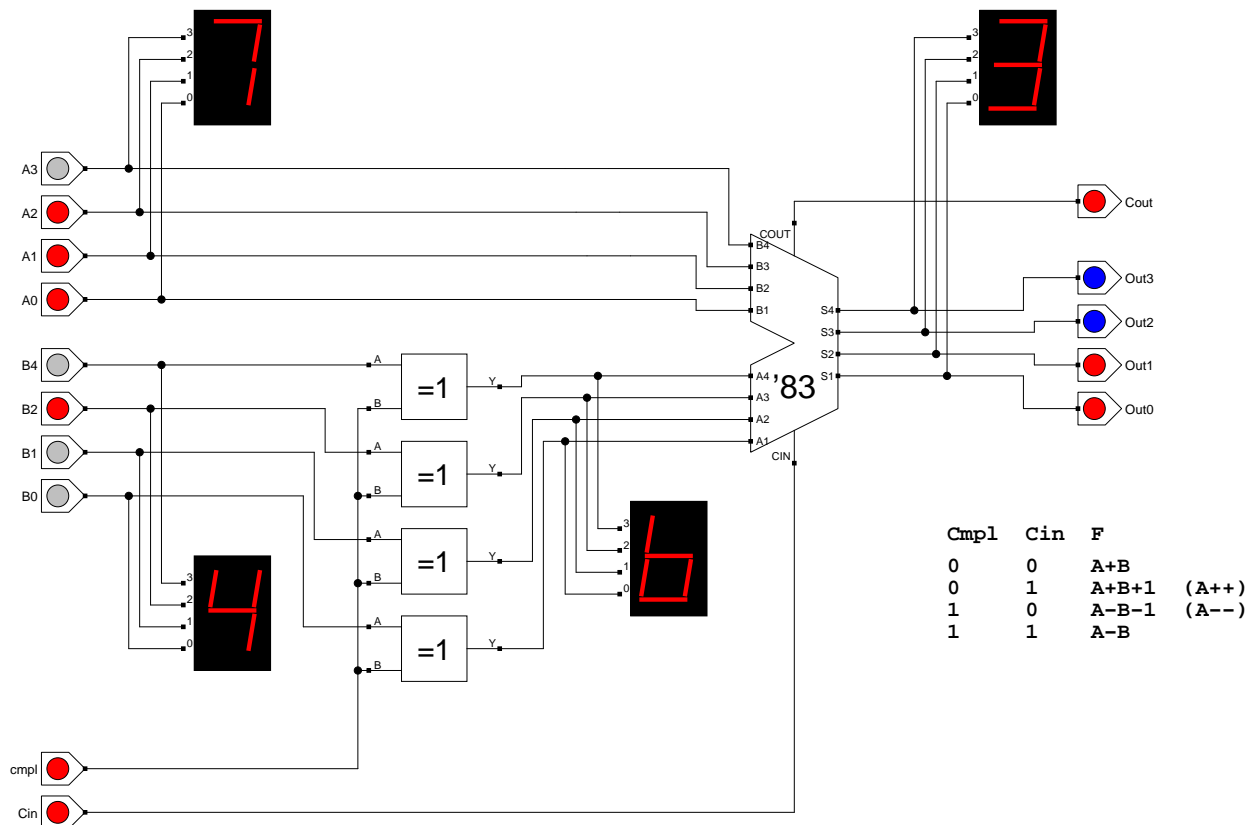
Entwicklung einer Arithmetisch–Logischen Einheit (ALU)

- ▶ 4Bit Carry–Look–Ahead Adder
- ▶ zB: SN 7483
- ▶ Hades: adder.hds



Arithmetische Einheit

- ▶ Zweierkomplementbildung
- ▶ Funktionen
 - A plus B
 - A minus B
- ▶ 4Bit XOR: SN 7486
- ▶ Hades: addsub.hds



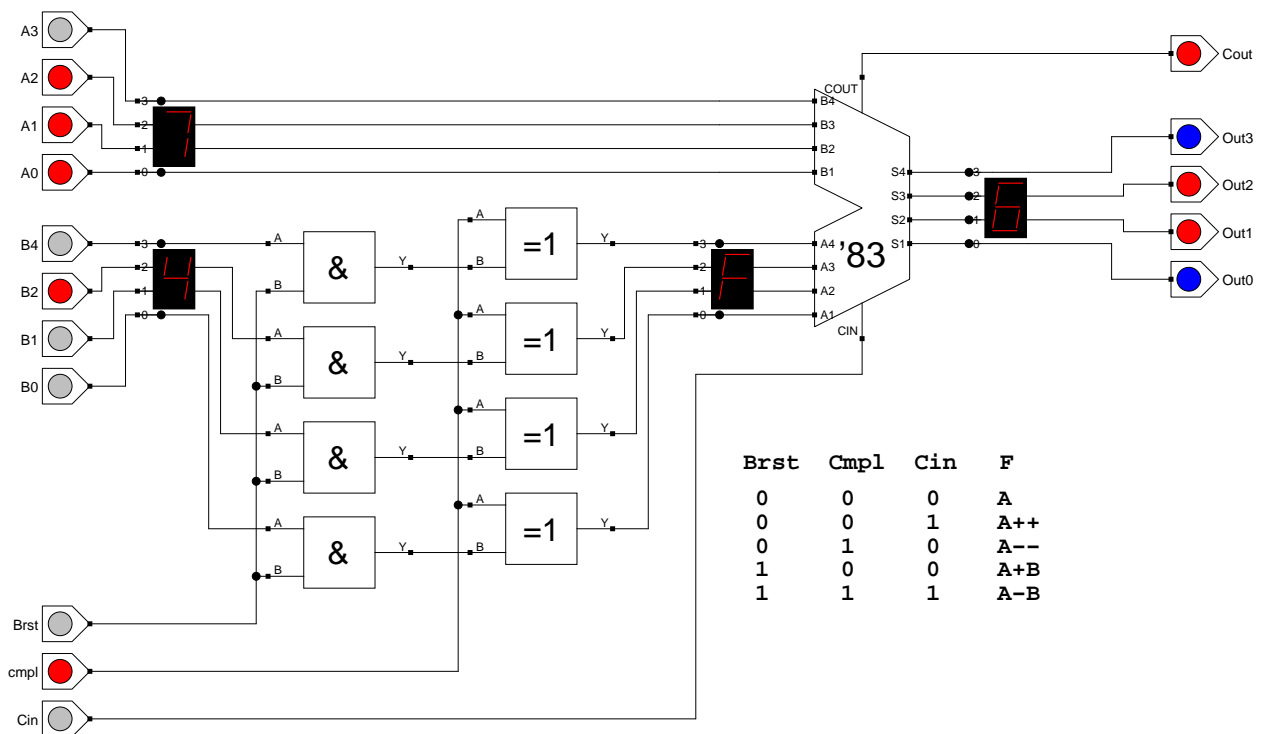
Arithmetische Einheit

- ▶ Zweierkomplementbildung und Rücksetzen des Eingangs B
- ▶ Funktionen

Brst	Cmpl	Cin	F
0	0	0	A
0	0	1	$A++$
0	1	0	$A--$
1	0	0	$A+B$
1	1	1	$A-B$

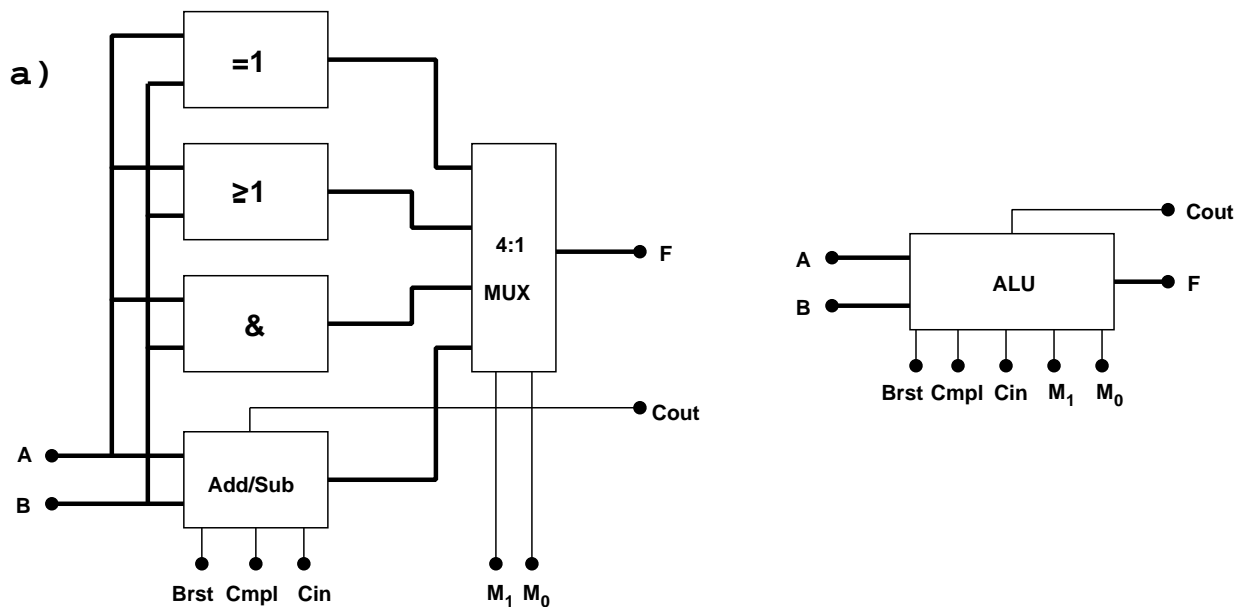
- ▶ 4Bit AND: TTL 7408

- ▶ Hades: 3_aritunit/au.hds



Arithmetisch-logische Einheit

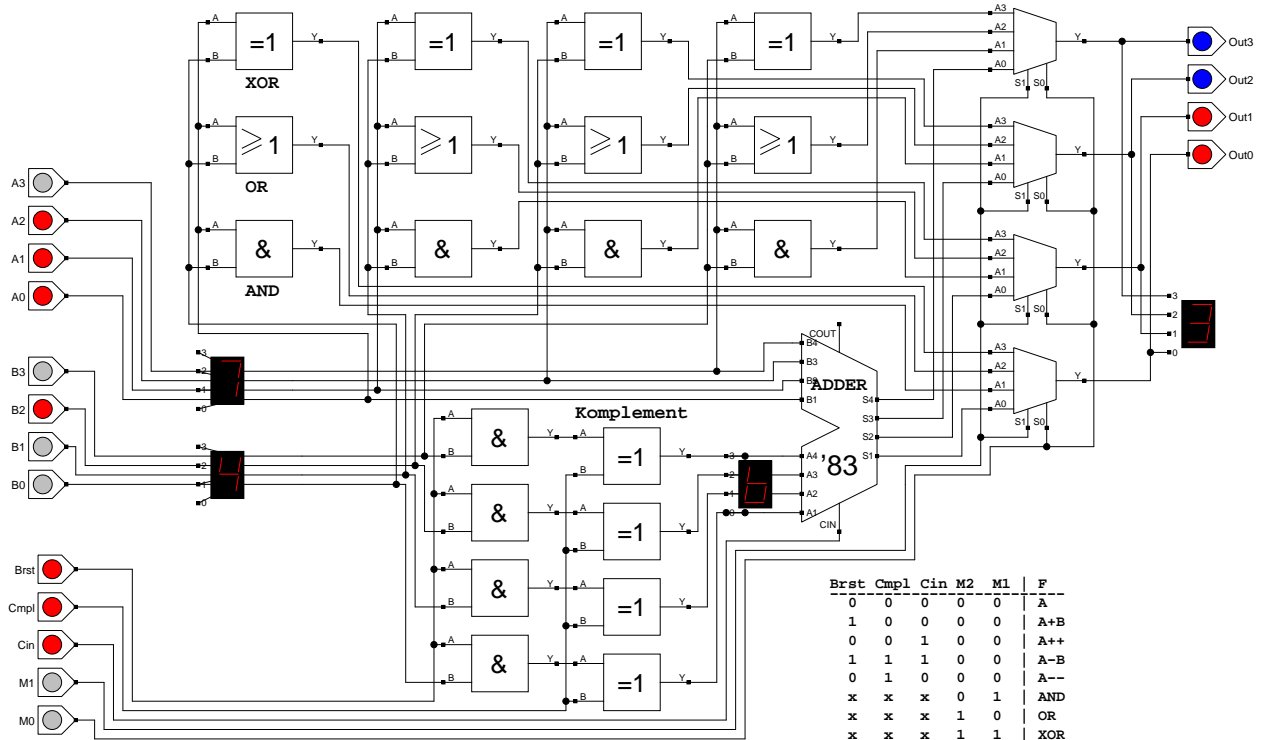
- ▶ Erweiterung der Arithmetischen Einheit zur ALU
- ▶ zusätzliche logische Funktionen
 - AND, OR, XOR Verknüpfungen
- ▶ Funktion im Ausgang durch Multiplexer wählbar
 - zusätzliche Steuerleitungen M_1 und M_0



Brst	Cmpl	Cin	M_1	M_0	F
0	0	0	0	0	A
1	0	0	0	0	A+B
0	0	1	0	0	A+1
1	1	1	0	0	A-B
0	1	0	0	0	A-1
x	x	x	0	1	$A \wedge B$
x	x	x	1	0	$A \vee B$
x	x	x	1	1	$A \oplus B$

Arithmetisch-logische Einheit

- ▶ AND, OR, XOR Verknüpfungen
- ▶ Funktion im Ausgang durch Multiplexer wählbar
- ▶ TTL Gatter:
 - 4Bit AND: TTL 7408
 - 4Bit OR: TTL 7432
 - 4Bit XOR: TTL 7486
 - 2x4:1 MUX: TTL 74153
- ▶ Hades: 4_alu/alu.hds



Arithmetisch-logische Einheit mit Steuerlogik

- ▶ 8 Funktionen
- ▶ 5 Steuereingänge
- ▶ Redundanz: 3 Steuereingänge genügen

S2	S1	S0	Brst	Cmpl	Cin	M1	M0	F
0	0	0	0	0	0	0	0	A
0	0	1	1	0	0	0	0	$A + B$
0	1	0	0	0	1	0	0	$A ++$
0	1	1	1	1	1	0	0	$A - B$
1	0	0	0	1	0	0	0	$A --$
1	0	1	x	x	x	0	1	$A \wedge B$
1	1	0	x	x	x	1	0	$A \vee B$
1	1	1	x	x	x	1	1	$A \neq B$

$$Brst = S0$$

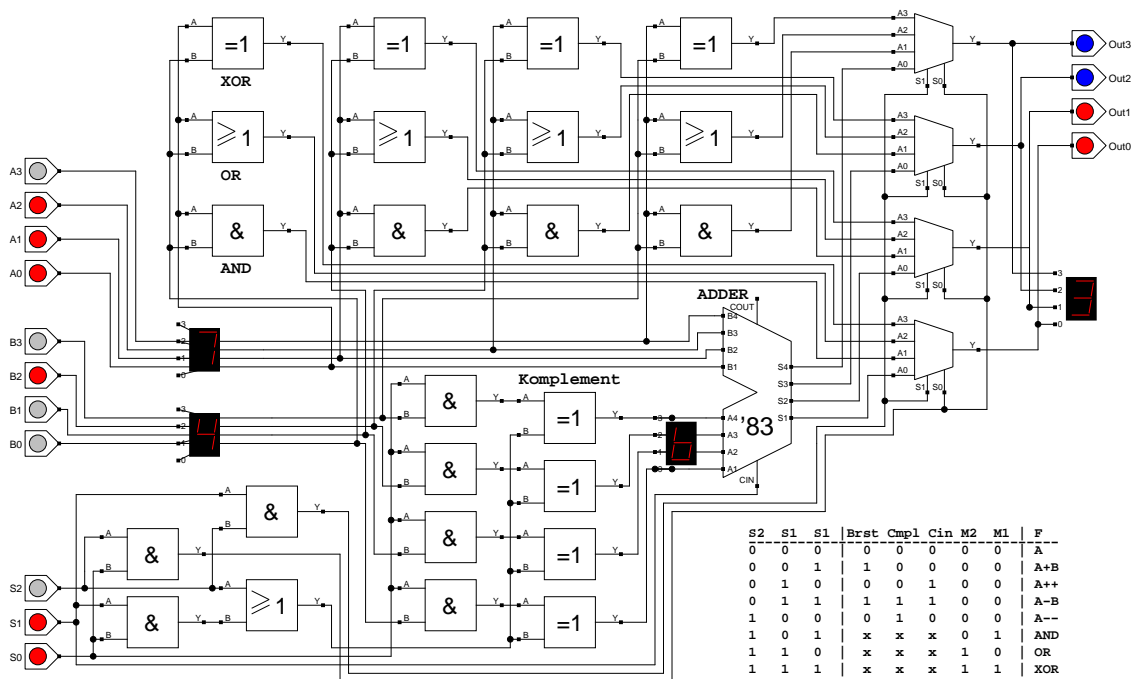
$$Cmpl = S2 \vee (S1 \wedge S0)$$

$$Cin = S1$$

$$M1 = S2 \wedge S1$$

$$M0 = S2 \wedge S0$$

- ▶ Hades: 5_alusteuer/alu.hds



Arithmetisch-logische Einheit mit Statusausgabe

- ▶ zusätzliche Ergebnisanzeige
- ▶ gängige Flags:
 - Carry (C)
 - Zero (Z)
 - Sign (S)
 - Overflow (V)

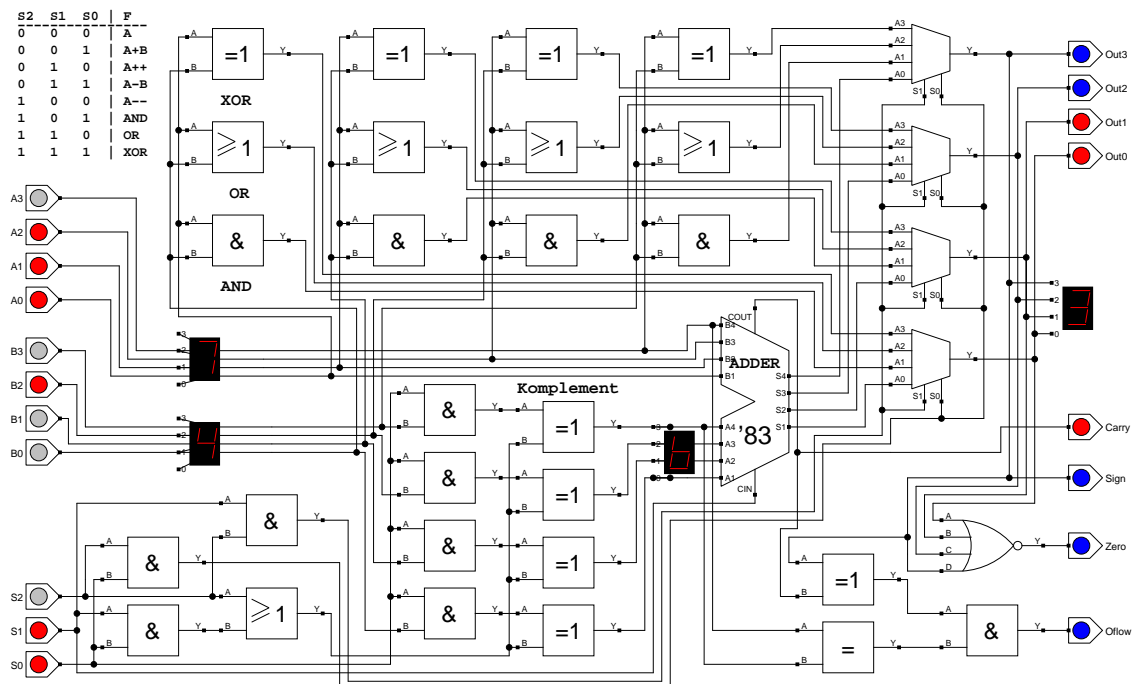
$$C = C_{out}$$

$$Z = \overline{Out_3 \vee Out_2 \vee Out_1 \vee Out_0}$$

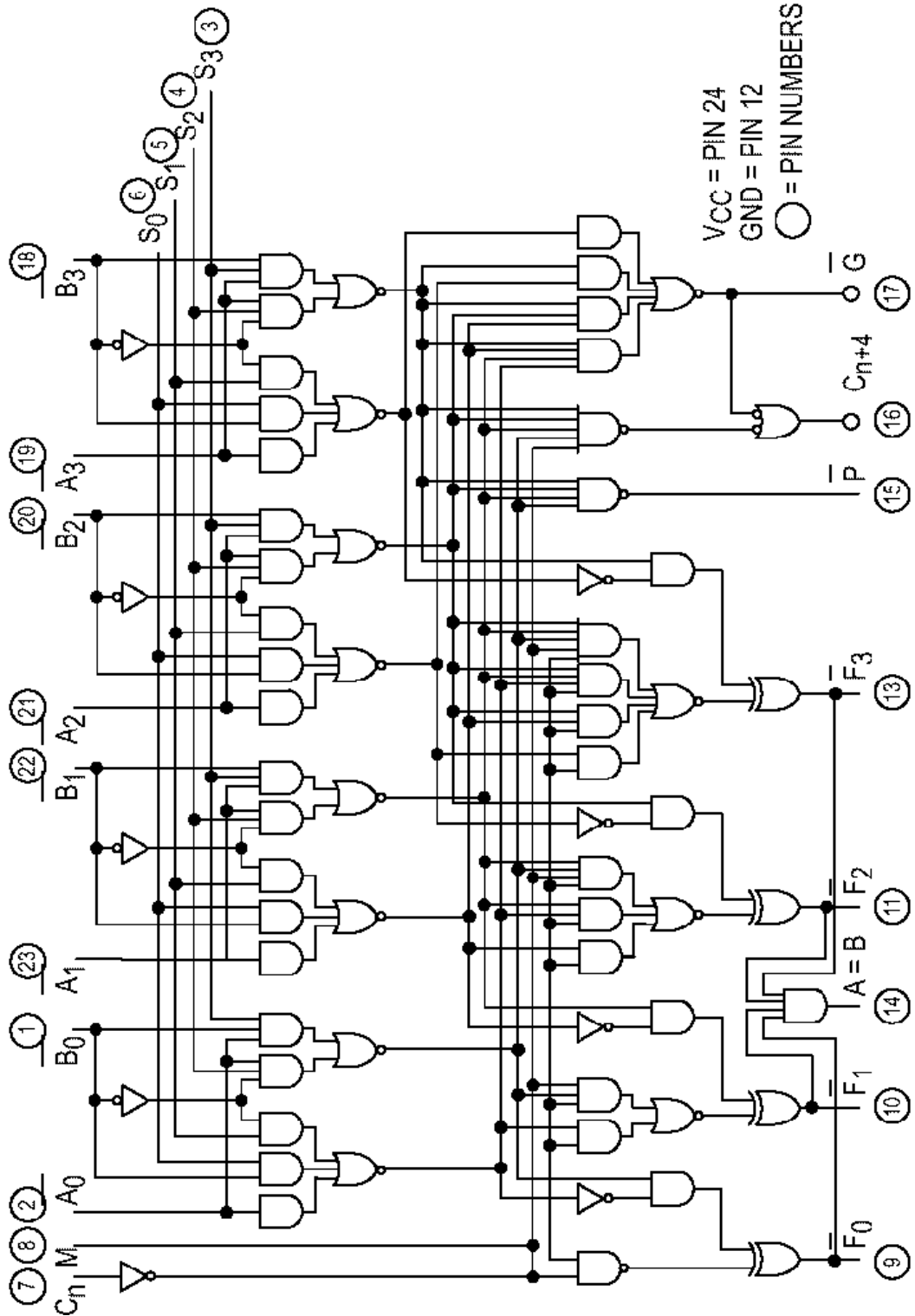
$$S = Out_3$$

$$V = (A_3 \equiv B_3) \wedge (Out_3 \neq C_{out})$$

- ▶ Hades: 6_alustatus/alu.hds



Arithmetisch-logische Einheit SN 74181



Arithmetisch-logische Einheit

SN 74181

- ▶ 4 Bit
- ▶ 4 Steuereingänge
- ▶ 32 Funktionen
 - 16 arithmetische Funktionen
 - 16 logische Funktionen

SN54/74LS181

FUNCTION TABLE

MODE SELECT INPUTS				ACTIVE LOW INPUTS & OUTPUTS		ACTIVE HIGH INPUTS & OUTPUTS	
S ₃	S ₂	S ₁	S ₀	LOGIC (M = H)	ARITHMETIC** (M = L) (C _n = L)	LOGIC (M = H)	ARITHMETIC** (M = L) (C _n = H)
L	L	L	L	\overline{A}	A minus 1	\overline{A}	A
L	L	L	H	\overline{AB}	\overline{AB} minus 1	$\overline{A + B}$	A + \overline{B}
L	L	H	L	A + B	AB minus 1	AB	A + B
L	L	H	H	Logical 1	minus 1	Logical 0	minus 1
L	H	L	L	$\overline{A + B}$	A plus $\overline{(A + B)}$	\overline{AB}	A plus \overline{AB}
L	H	L	H	\overline{B}	AB plus (A + B)	B	(A + B) plus AB
L	H	H	L	$A \oplus \overline{B}$	A minus B minus 1	$A \oplus B$	A minus B minus 1
L	H	H	H	$\overline{A + B}$	A + B	\overline{AB}	AB minus 1
H	L	L	L	AB	A plus (A + B)	$\overline{A + B}$	A plus AB
H	L	L	H	$A \oplus B$	A plus B	$A \oplus B$	A plus B
H	L	H	L	B	AB plus (A + B)	B	(A + B) plus AB
H	L	H	H	A + B	A + B	AB	AB minus 1
H	H	L	L	Logical 0	A plus A*	Logical 1	A plus A*
H	H	L	H	AB	\overline{AB} plus A	A + B	(A + \overline{B}) plus A
H	H	H	L	AB	AB plus A	A + B	(A + B) Plus A
H	H	H	H	A	A	A	A minus 1

L = LOW Voltage Level

H = HIGH Voltage Level

*Each bit is shifted to the next more significant position

**Arithmetic operations expressed in 2s complement notation

Entwicklung Speicherbaustein

- ▶ Entwicklung eines 1-Bit Speicher-Schaltnetzes
- ▶ Forderung
 - Setzen
 - Rücksetzen
 - Speichern
- ▶ zwei Eingänge
 - S (et): Ausgang setzen $Q^+ = 1$
 - R (eset): Ausgang rücksetzen $Q^+ = 0$
 - $S = 0, R = 0$: Speichern $Q^+ = Q$

S	R	Q^+
0	0	Q
0	1	0
1	0	1
1	1	x

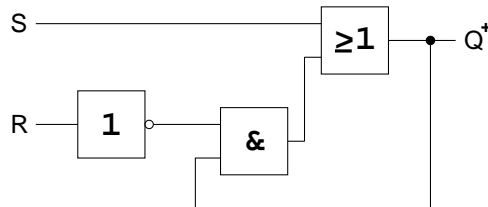
S	R	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

	RQ	00	01	11	10
S		0	1	0	0
0		0	1	0	0
1		1	1	x	x

$$Q^+ = S \vee \bar{R}Q$$

Entwicklung Speicherbaustein

- Übergangsfunktions: $Q^+ = S \vee \bar{R}Q$

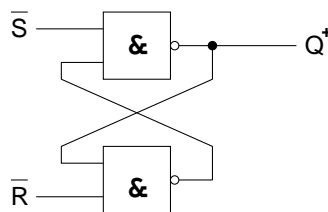


- Nur mit NAND-Gattern

$$Q^+ = S \vee \bar{R}Q$$

$$Q^+ = \overline{\overline{S \vee \bar{R}Q}}$$

$$Q^+ = \overline{\bar{S} \wedge \overline{\bar{R}Q}}$$

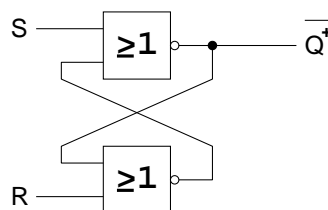


- Nur mit NOR-Gattern

$$\overline{Q^+} = \overline{S \vee \bar{R}Q}$$

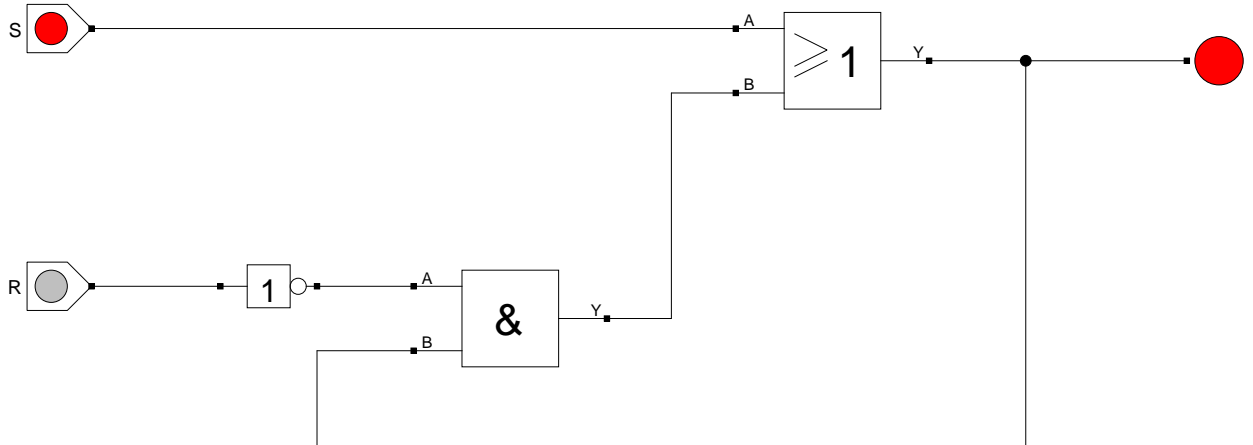
$$\overline{Q^+} = \overline{\bar{S} \wedge (R \vee \bar{Q})}$$

$$\overline{Q^+} = S \vee R \vee \bar{Q}$$



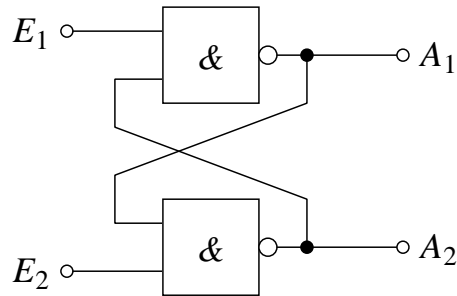
Entwicklung Speicherbaustein

► Hades: ff_df.hds



Basis FlipFlops

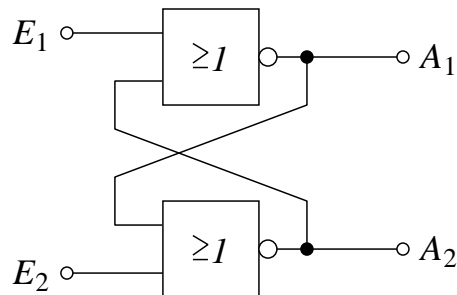
► NAND



E_1	E_2	A_1^+	A_2^+	Funktion
0	0	1	1	(verboten)
0	1	1	0	setzen
1	0	0	1	rücksetzen
1	1	A_1	A_2	speichern

$$\longrightarrow A_1 = \overline{A_2}$$

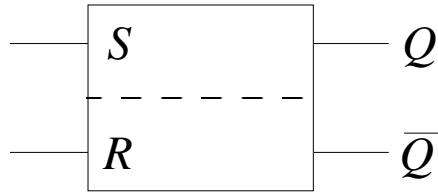
► NOR



E_1	E_2	A_1^+	A_2^+	Funktion
0	0	A_1	A_2	speichern
0	1	1	0	setzen
1	0	0	1	rücksetzen
1	1	0	0	(verboten)

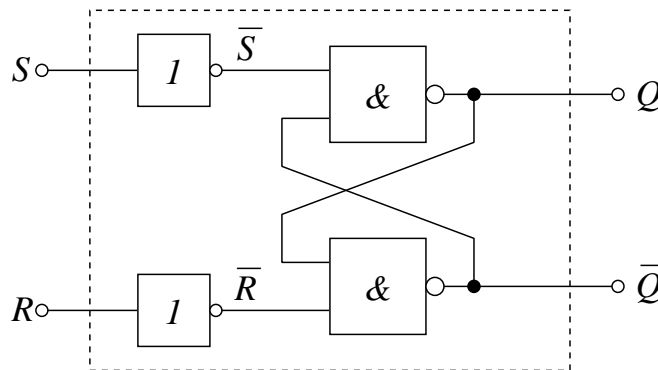
$$\longrightarrow A_1 = \overline{A_2}$$

RS - FlipFlops

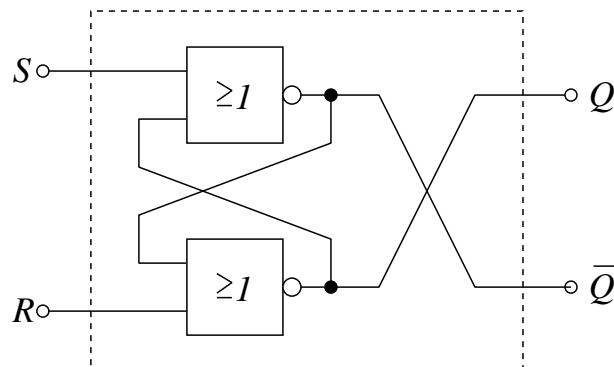


S	R	Q	\bar{Q}	Funktion
0	0	Q	\bar{Q}	speichern
0	1	0	1	rücksetzen
1	0	1	0	setzen
1	1	x	x	(verboten)

► NAND

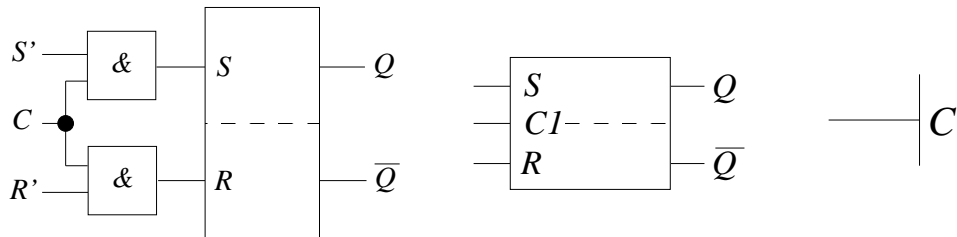


► NOR



RS-FlipFlop mit Zustandssteuerung

- Wirksamwerden des Eingangs abhängig von einer Steuervariablen
 - Steuer- oder Taktsignal



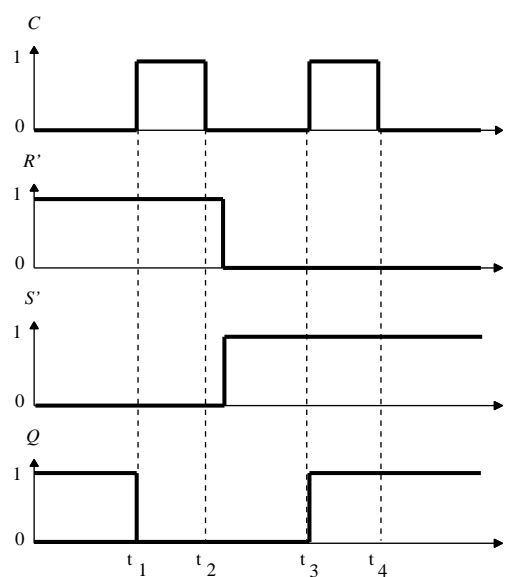
a) Schaltung

b) Schaltzeichen

c) Schaltzeichen für C-Eingang

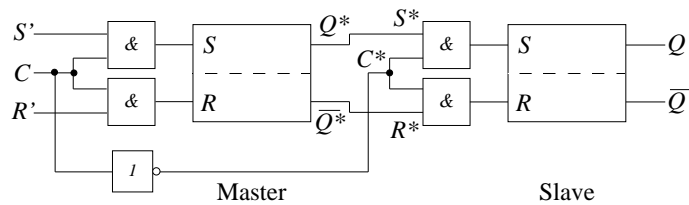
C	S'	R'	Q_{n+1}	
0	0	0	Q_n	
0	0	1	Q_n	keine Änderung
0	1	0	Q_n	des Ausgangszustandes,
0	1	1	Q_n	d.h. Speichern
.....				
1	0	0	Q_n	
1	0	1	0	Rücksetzen
1	1	0	1	Setzen
1	1	1	–	unzulässig.

- Bsp.: Impulsdiagramm

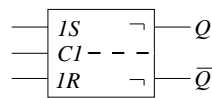


RS-MS-FlipFlop

- ▶ Problem: Schieberegister
 - bei Taktzustandssteuerung rutschen Information durch, $C=1$
- ▶ Zwei gekoppelte RS-FF Master-Slave
- ▶ Slave-Takt invertiert

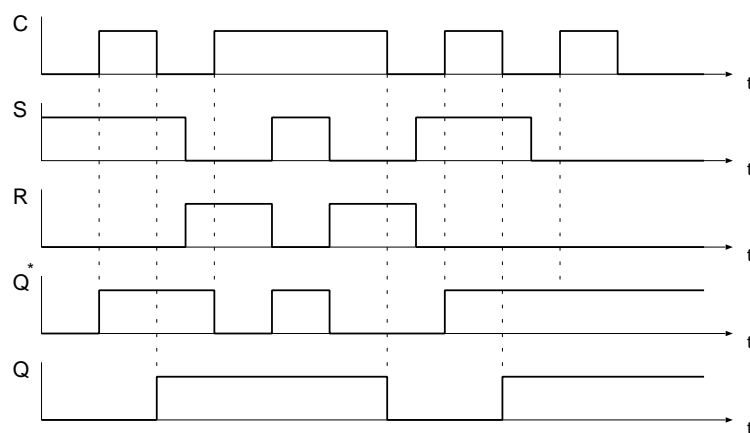


a) Schaltung mit Master-Slave Prinzip



b) Schaltzeichen

- ▶ Beispiel Impulsdiagramm



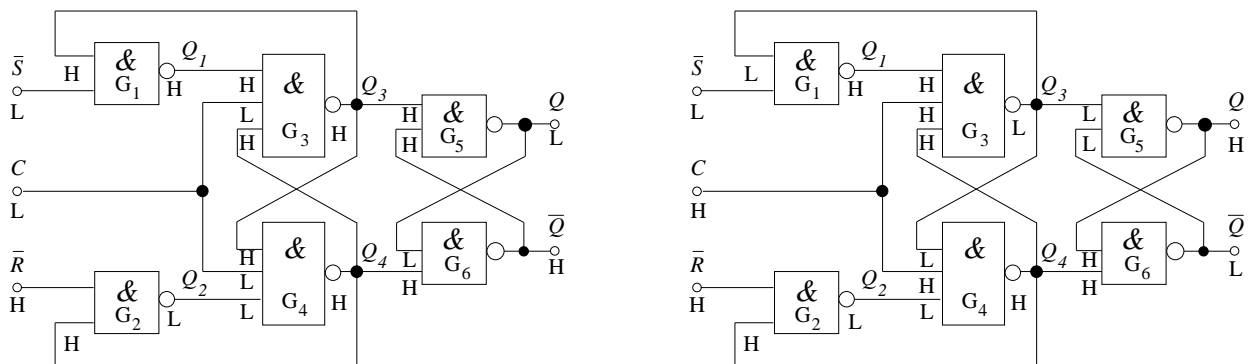
RS-FlipFlop mit Taktflankensteuerung

► Nachteil der Zustandssteuerung

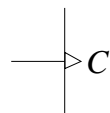
- Solange $C = 1$ ist, wirken sich Änderungen am Eingang noch auf den Folgezustand aus.
- zB. Störimpulse

► Besser:

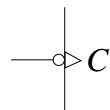
- Eingangsübernahme zu einem *definierten* Zeitpunkt
- zB. bei steigender Taktflanke



► C-Eingang



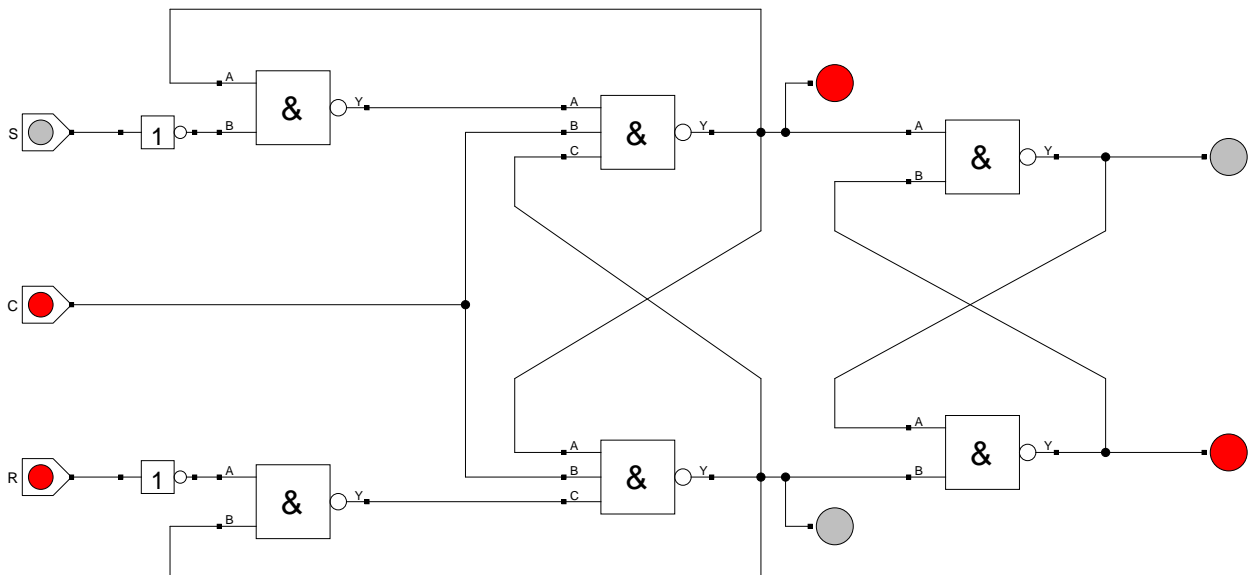
Die Variablen an den Eingängen, die von C abhängen, werden nur beim 0-1-Übergang von C wirksam.



Die Variablen an den Eingängen, die von C abhängen, werden nur beim 1-0-Übergang von C wirksam.

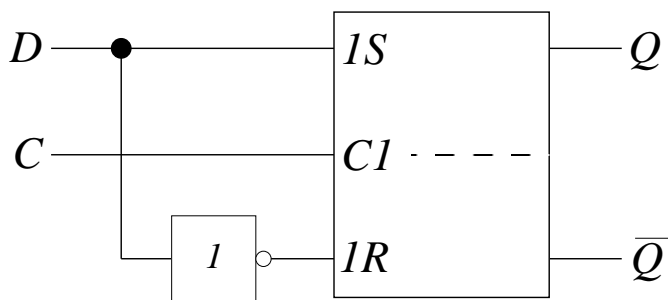
RS-FlipFlop mit Taktflankensteuerung

► rs.flanke.hds

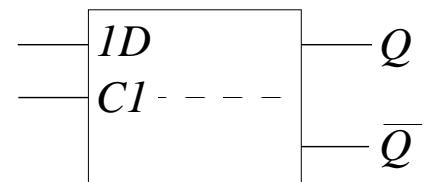


D-FlipFlop

- ▶ Vermeidung der unzulässigen Eingangskombination $R = S = 1$
- ▶ R wird zu \bar{S}
- ▶ Realisierung mit einem RS-FlipFlop:



a) Schaltung



b) Schaltzeichen

- ▶ Zustandstabelle:

C	D	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Speichern

Rücksetzen

Setzen

oder :

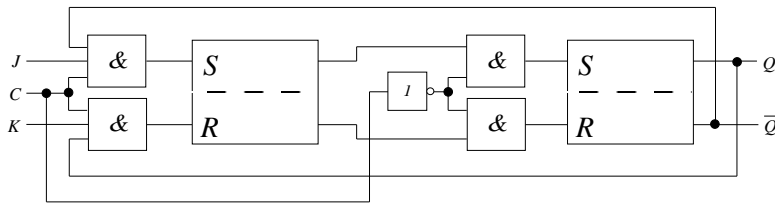
C	D	Q^+
0	0	Q
0	1	Q
1	0	0
1	1	1

- ▶ Übergangsfunktion:

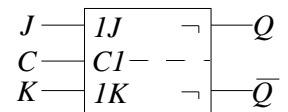
$$Q^+ = CD \vee \bar{C}Q$$

JK-MS-FlipFlop

- ▶ Eingangskombination $S = 1, R = 1$ ungenutzt
- ▶ Forderung bei $S = 1, R = 1$ soll $Q^+ = \bar{Q}$ werden



a) Schaltung



b) Schaltzeichen

J	K	Q	Q^+	
0	0	0	0	Speichern
0	0	1	1	
0	1	0	0	Rücksetzen
0	1	1	0	
1	0	0	1	Setzen
1	0	1	1	
1	1	0	1	Kippen
1	1	1	0	

oder :

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	\bar{Q}_n

- ▶ KV-Tafel

	K Q	00	01	11	10
J	0	0	1	0	0
1	1	1	1	0	1

$$Q^+ = (J \wedge \bar{Q}) \vee (\bar{K} \wedge Q)$$

JK-MS-FlipFlop

- ▶ Alternative Berechnung
- ▶ Ersetze in der RS Übergangsfunktion R und S durch:
 - $S = J \wedge \bar{Q}$
 - $R = K \wedge Q$
- ▶ dann folgt:

$$\begin{aligned}
 Q^+ &= S \vee (\bar{R} \wedge Q) \\
 &= (J \wedge \bar{Q}) \vee [(\overline{K \wedge Q}) \wedge Q] \\
 &= (J \wedge \bar{Q}) \vee [(\bar{K} \vee \bar{Q}) \wedge Q] \\
 &= (J \wedge \bar{Q}) \vee [(\bar{K} \wedge Q) \vee (\bar{Q} \wedge Q)] \\
 Q^+ &= (J \wedge \bar{Q}) \vee (\bar{K} \wedge Q)
 \end{aligned}$$

- ▶ Frage: Wie müssen J und K belegt sein, um von einem Zustand in einen anderen zu gelangen?

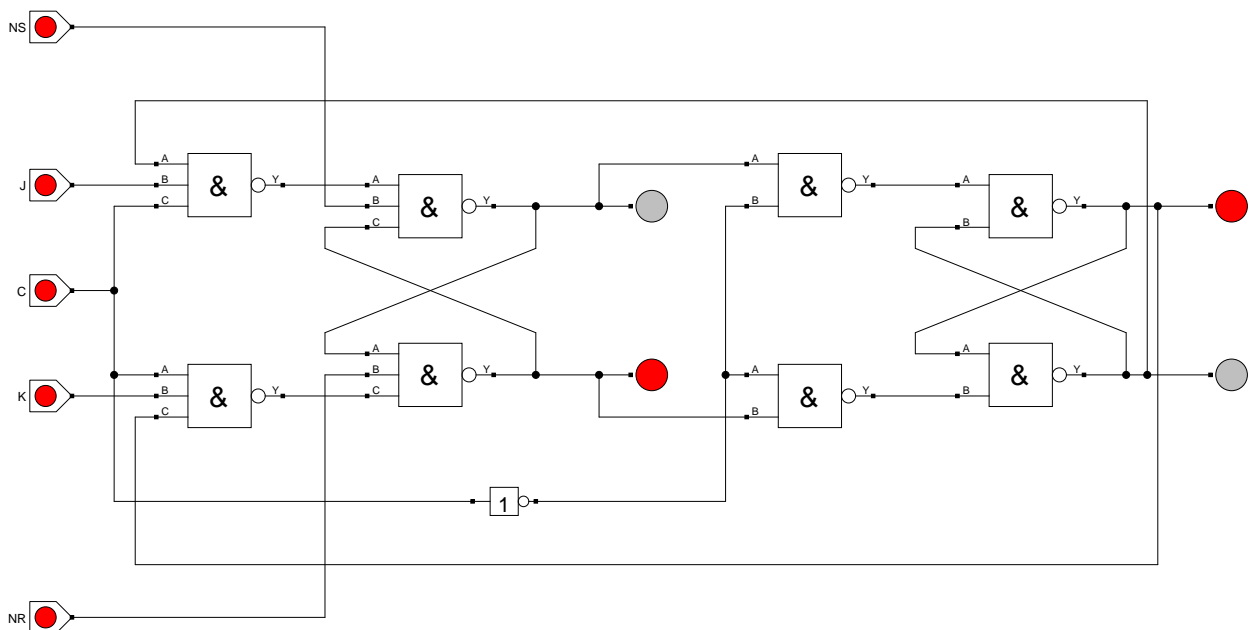
Q ist	Q^+ soll werden	dann muß J/K sein		
		J	K	
0	0	0	×	Rücksetzen oder Speichern
0	1	1	×	Setzen oder Kippen
1	0	×	1	Rücksetzen oder Kippen
1	1	×	0	Setzen oder Speichern

JK-MS-FlipFlop

- ▶ jk-ms-ff.hds
- ▶ mit zusätzlichen asynchronen Set- und Reset-Eingängen
 - Low Aktiv

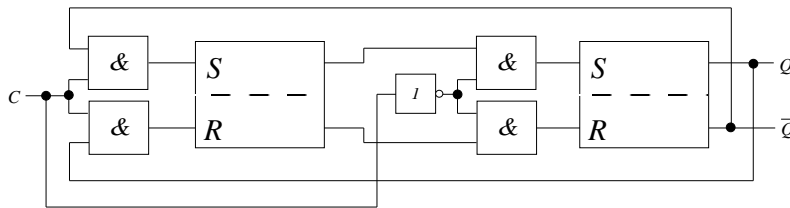
→ Set: \overline{NS}

→ Reset: \overline{NR}

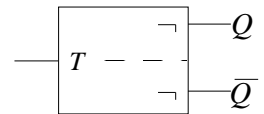


Master–Slave T–FlipFlop

- ▶ Variante des JK-MS-FlipFlops
- ▶ Keine Eingänge J, K
- ▶ Q^+ somit nur von C und Q abhängig
- ▶ Schaltung



a) Schaltung

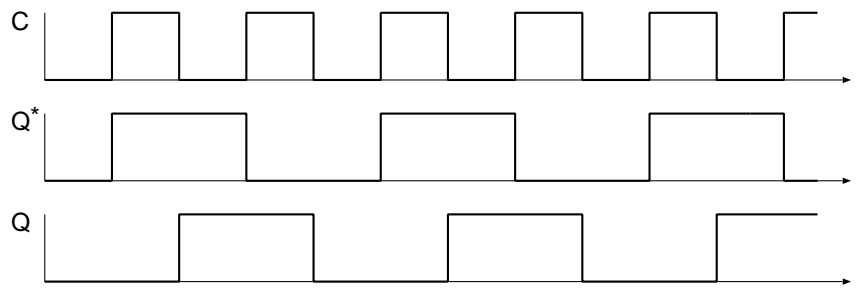


b) Schaltzeichen

- ▶ Somit gilt:
 - $S = \bar{Q}$
 - $R = Q$
- ▶ Eingesetzt in RS–Übergangsfunktion

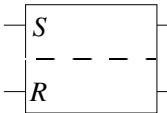
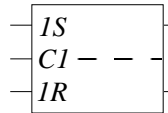
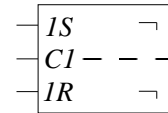
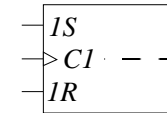
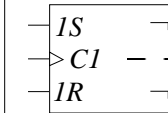
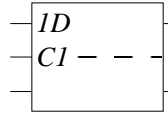
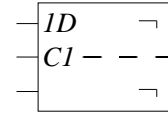
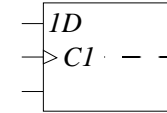
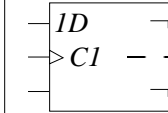
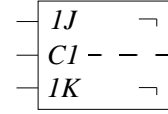
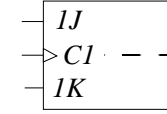
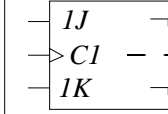
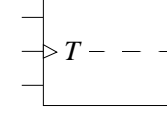
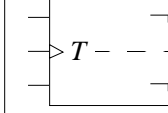
$$\begin{aligned}
 Q^+ &= S \vee (\bar{R} \wedge Q) \\
 &= \bar{Q} \vee (\bar{Q} \wedge Q) \\
 Q^+ &= \bar{Q}
 \end{aligned}$$

- ▶ Toggle–Flipflop
 - wechselt bei jedem Takt den Zustand
 - Frequenzteiler 2:1



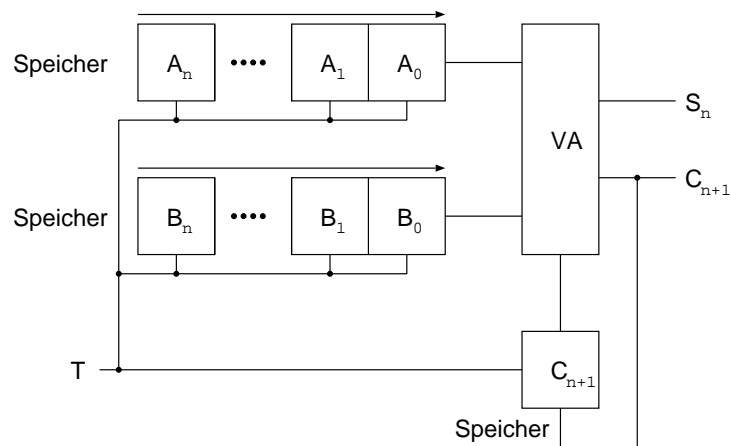
Zusammenfassung

► Übersicht der FlipFlop-Typen

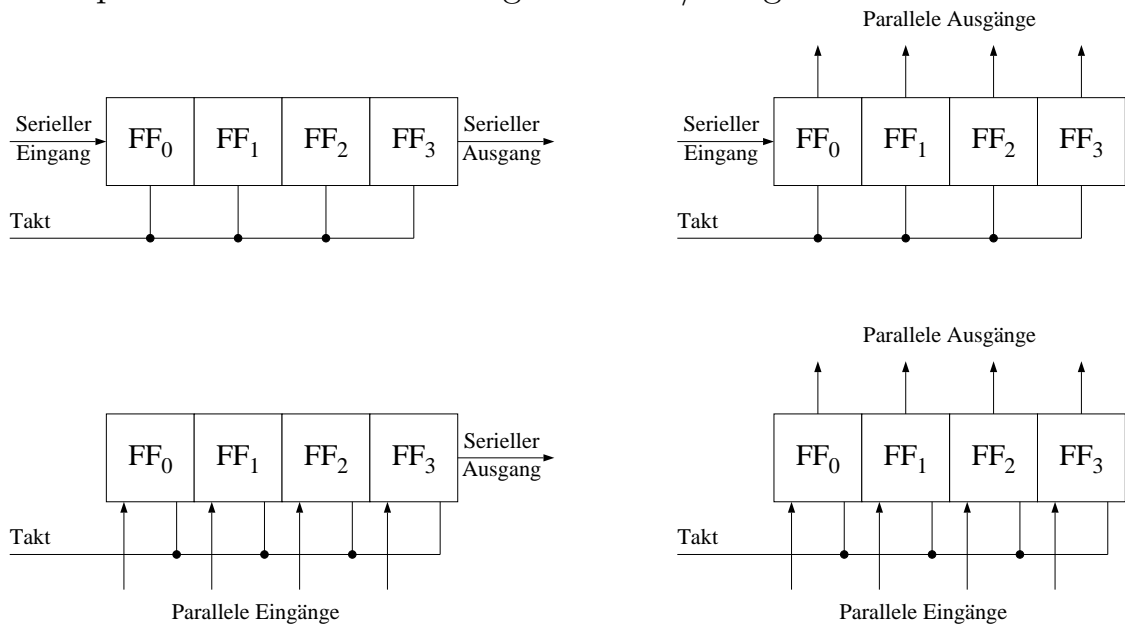
	ohne Takt- steuerung	Zustands- steuerung	Zwei-Zustands- steuerung	Einflanken- steuerung	Zweiflanken- steuerung
RS - FF					
D - FF					
JK - FF					
T - FF					

Register

- ▶ Aufnahme, Speichern und Weitergabe mehrstelliger Binärworte
- ▶ Aufnahme und Weitergabe
 - seriell oder parallel
- ▶ Schieben bzw. Rotieren
 - left/right shift/rotate
- ▶ bekanntes Beispiel: serieller Addierer



- ▶ Beispiele für verschiedene Register Ein-/Ausgaben

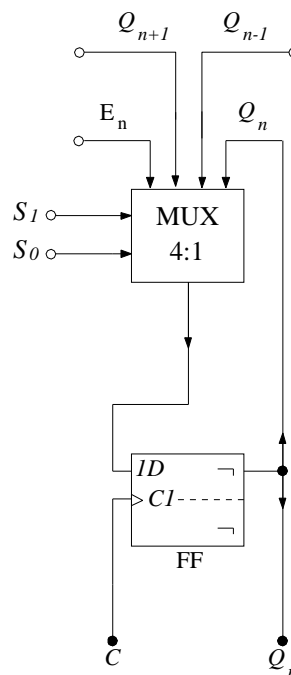


Register

- ▶ Entwicklung eines multifunktionellen Registers
- ▶ Forderungen:
 - parallele Eingabe
 - rechts schieben (Serielle Ein- und Ausgabe)
 - links schieben
 - Speichern
- ▶ Vier Funktionen, steuerbar über 2 Steuerleitungen S_1, S_0

S_1	S_0	Q_n^+	Funktion
0	0	E_n	parallel einlesen
0	1	Q_{n+1}	rechts schieben
1	0	Q_{n-1}	links schieben
1	1	Q_n	Speichern

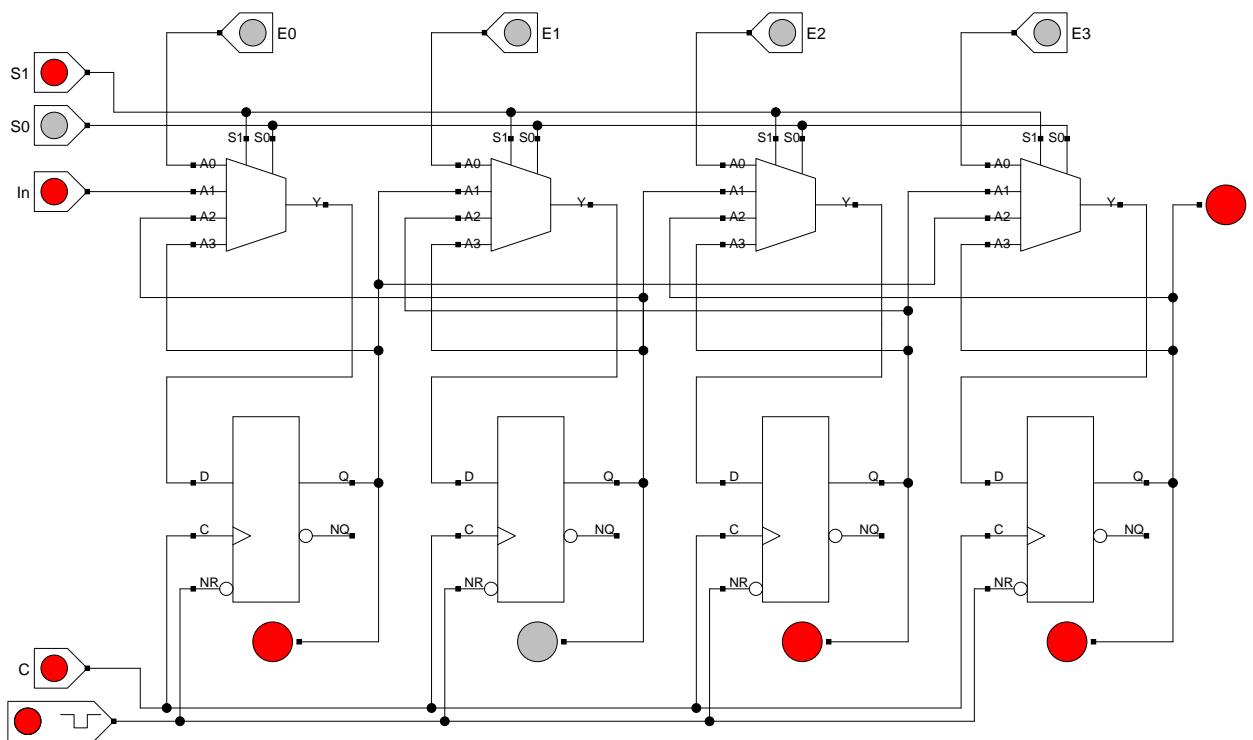
- ▶ Realisierung mit einem Multiplexer



Schieberegister mit MUX

► 4bit_register.hds

- serielle Ein-/Ausgabe
- parallel Ein-/Ausgabe
- rechts schieben
- links rotieren
- speichern



Automaten

► endlicher Automat:

- Eingabe- und Ausgabealphabet sind endlich

► formale Beschreibung:

$$M = \{X, Y, Z, z_0, F, f(), g()\}$$

$$X = x_0, x_1, \dots, x_n \quad (\text{Eingabealphabet})$$

$$Y = y_1, y_2, \dots, y_m \quad (\text{Ausgabealphabet})$$

$$Z = z_1, z_2, \dots, z_l \quad (\text{Zustandsmenge})$$

$$z_0 \in Z \quad (\text{Anfangszustand})$$

$$F \subseteq Z \quad (\text{Endzustände})$$

$$g : (x_i, z_j) \rightarrow z_k \quad (\text{Übergangsfunktion})$$

$$f : (x_i, z_j) \rightarrow y_r \quad (\text{Ausgangsfunktion})$$

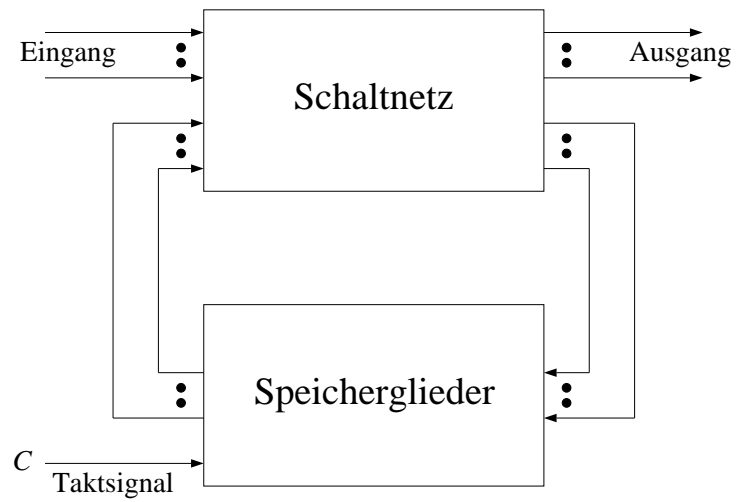
- Alle endliche Automaten lassen sich mit Schaltwerken realisieren
- Schaltwerk (DIN 40300):

Eine Funktionseinheit zum Verarbeiten von Schaltvariablen, wobei der Wert am Ausgang zu einem bestimmten Zeitpunkt abhängt von den Werten am Eingang zu diesem und endlich vielen vorangegangenen Zeitpunkten.

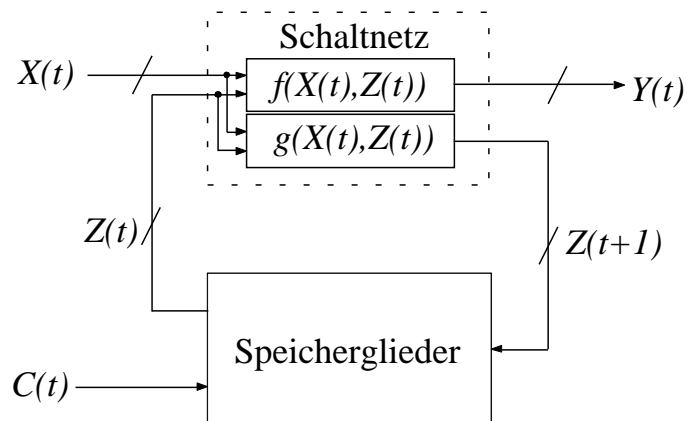
- vgl. Schaltnetz (DIN 40300):

Eine Funktionseinheit zum Verarbeiten von Schaltvariablen, dessen Wert am Ausgang zu einem beliebigen Zeitpunkt nur vom Wert am Eingang abhängt.

Automaten



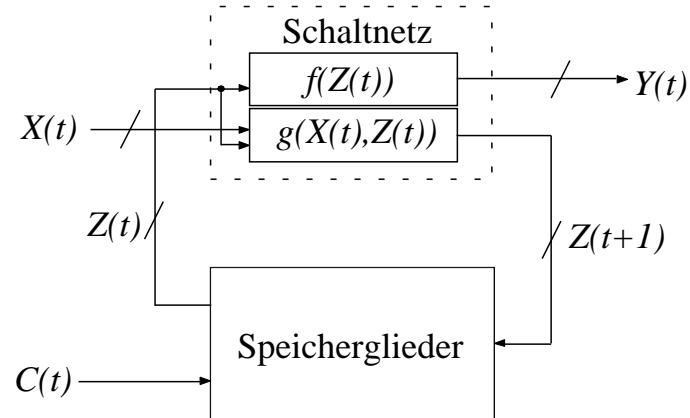
► Mealy Automat



- Übergangsfunktion $Z(t + 1) = g(X(t), Z(t))$
- Ausgangsfunktion $Y(t) = f(X(t), Z(t))$

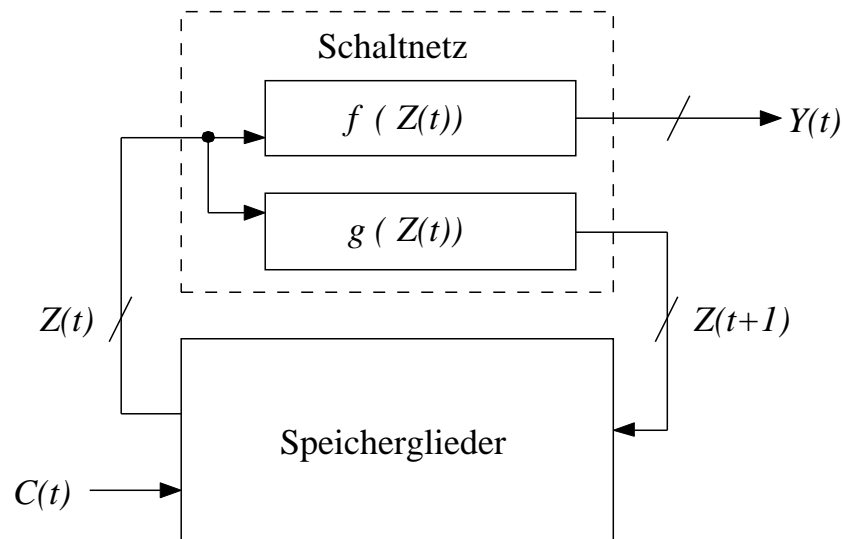
Automaten

► Moore Automat



- Übergangsfunktion $Z(t + 1) = g(X(t), Z(t))$
- Ausgangsfunktion $Y(t) = f(Z(t))$
- Änderung des Ausgangs immer taktsynchron

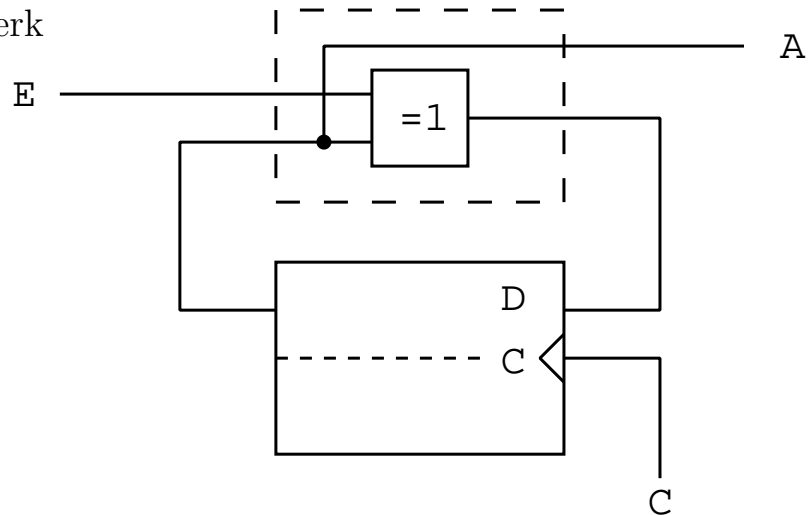
► Autonomer Automat



- Übergangsfunktion $Z(t + 1) = g(Z(t))$
- Ausgangsfunktion $Y(t) = f(Z(t))$

Analyse von Schaltwerken

► Schaltwerk



► Schaltfunktionen (Übergangsfunktion, Ausgangsfunktion)

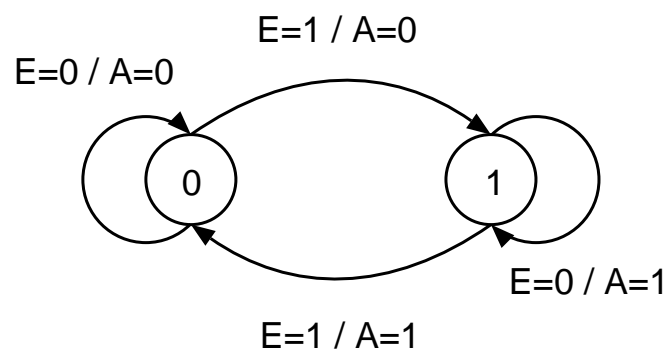
$$Q^+ = E \neq Q$$

$$A = Q$$

► Zustandsfolgetabelle

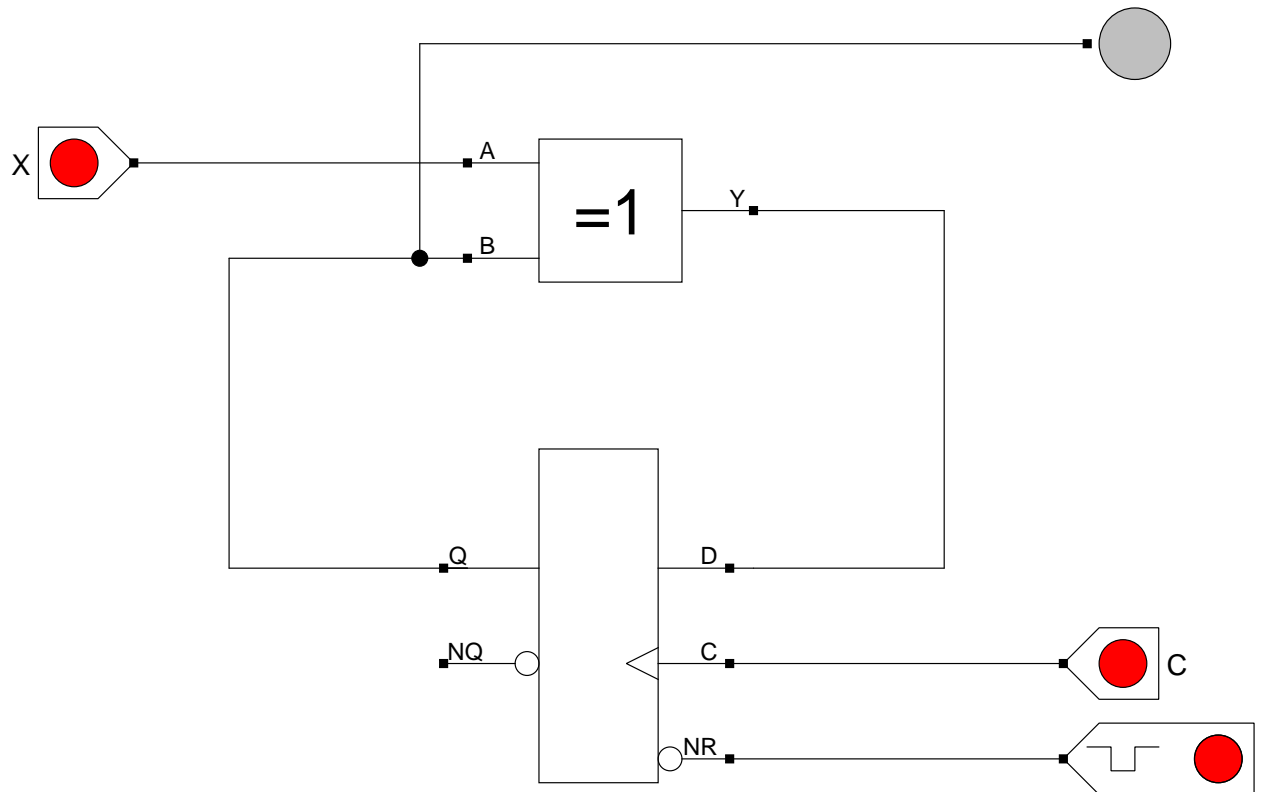
E	Q	Q^+	A
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

► Zustandsgraph



Analyse von Schaltwerken

► simple_sw.hds



Beschreibungsmöglichkeiten von Schaltwerken

- Zustandsfolgetabelle:

E	Q	Q^+	A
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

- KV Tafeln:

$Q \backslash E$	0	1
0	0	1
1	1	0

$$Q^+ = \bar{E}Q \vee E\bar{Q}$$

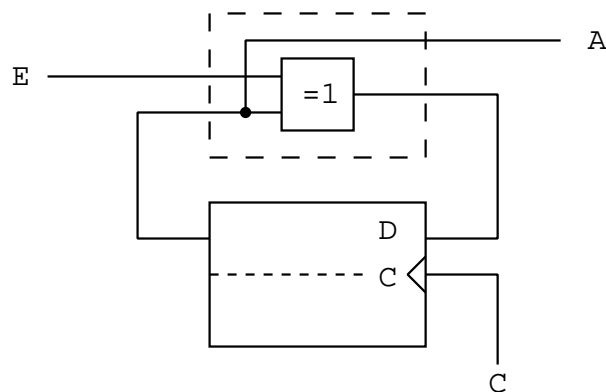
$$Q^+ = E \oplus Q$$

- Schaltfunktionen (Übergangsfunktion, Ausgangsfunktion):

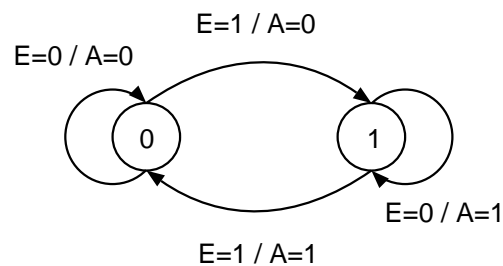
$$Q^+ = E \oplus Q$$

$$A = Q$$

- Schaltplan:

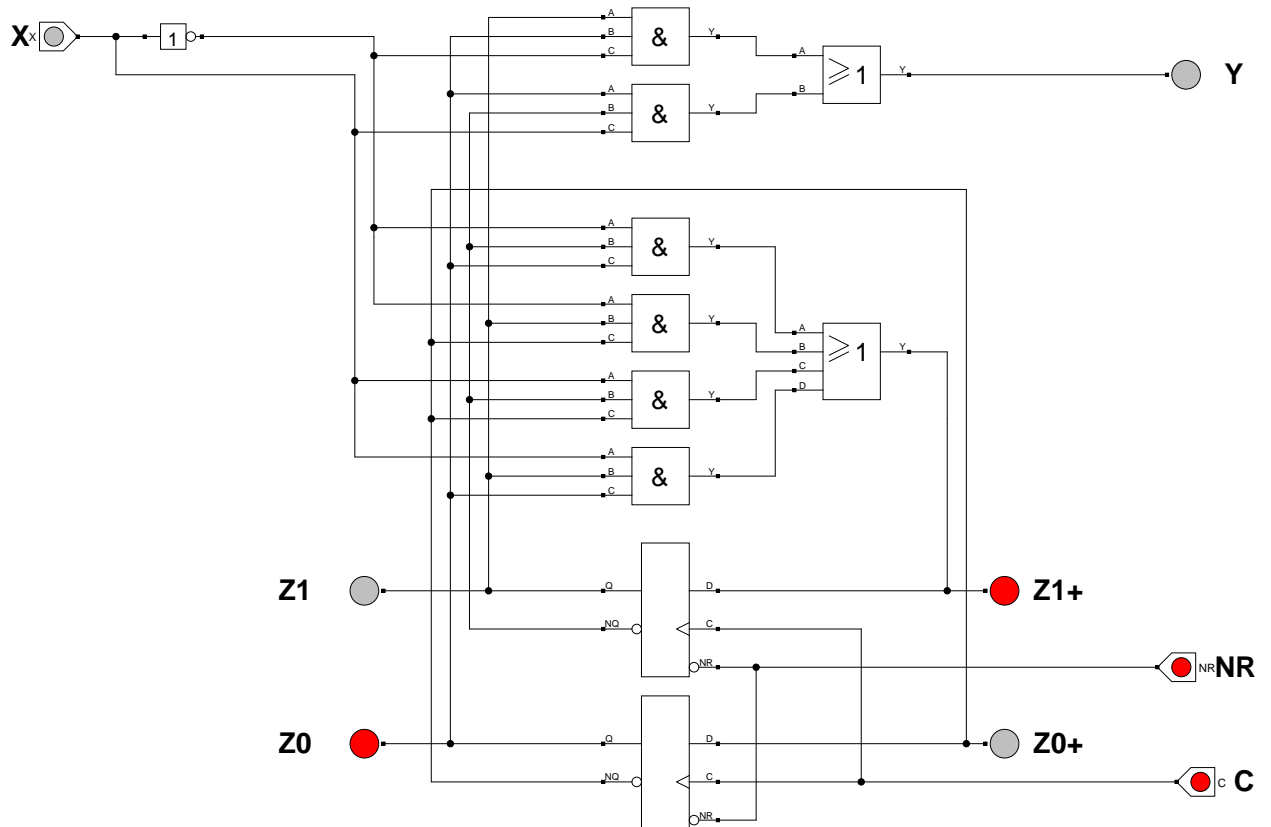


- Zustandsgraph:



Analyse von Schaltwerken (D-FF)

► dff_analyse.hds

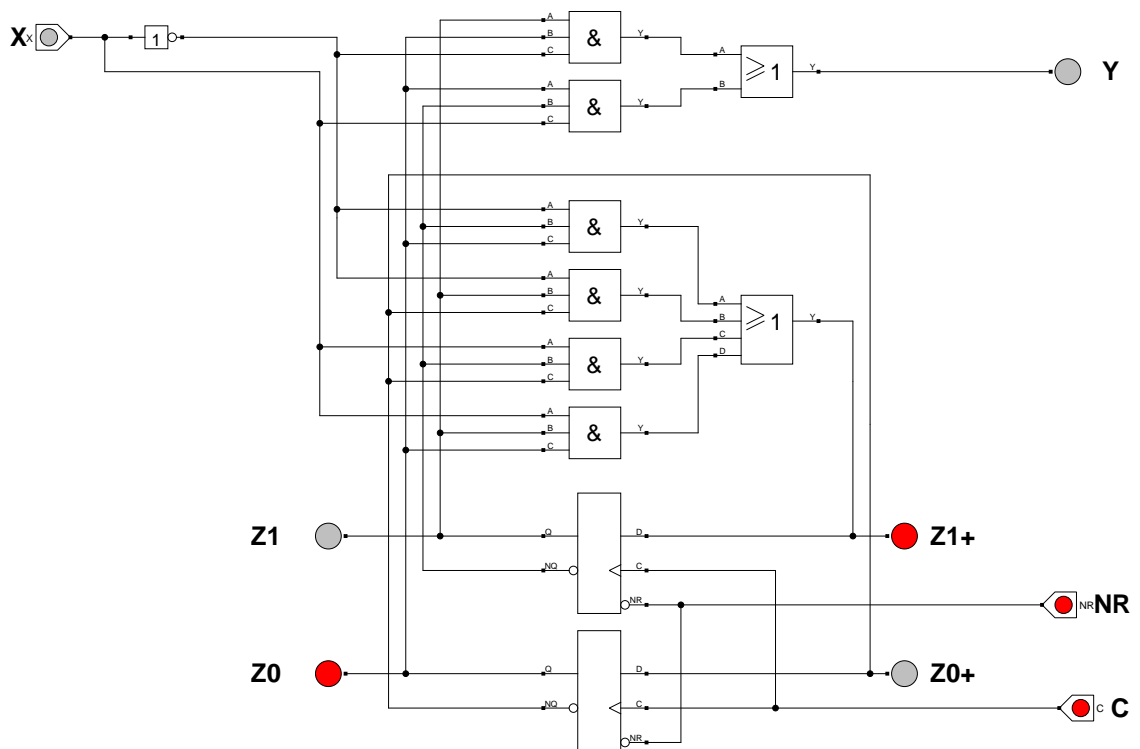


Analyse von Schaltwerken (D-FF)

► Erstellen von:

- Schaltfunktionen (Ausgangs- und Übergangsfunktion)
- Zustandsfolgetabelle
- Zustandsgraphen

► Beispiel: Schaltwerk mit D-FlipFlops:



► Schaltfunktionen

- Übergangsfunktionen

$$z_0^+ = \bar{z}_0$$

$$z_1^+ = \bar{x}\bar{z}_1z_0 \vee \bar{x}z_1\bar{z}_0 \vee x\bar{z}_1\bar{z}_0 \vee xz_1z_0$$

- Ausgangsfunktion

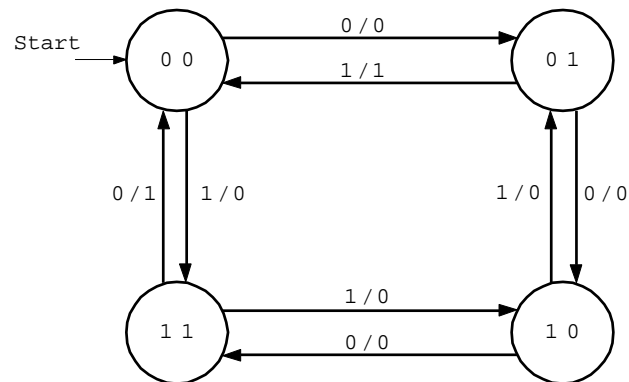
$$y = \bar{x}z_1z_0 \vee x\bar{z}_1z_0$$

Analyse von Schaltwerken (D-FF)

► Zustandsfolgetabelle:

x	z_1	z_0	z_1^+	z_0^+	y
0	0	0	0	1	0
0	0	1	1	0	0
0	1	0	1	1	0
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	0	0	1
1	1	0	0	1	0
1	1	1	1	0	0

► Zustandgraph

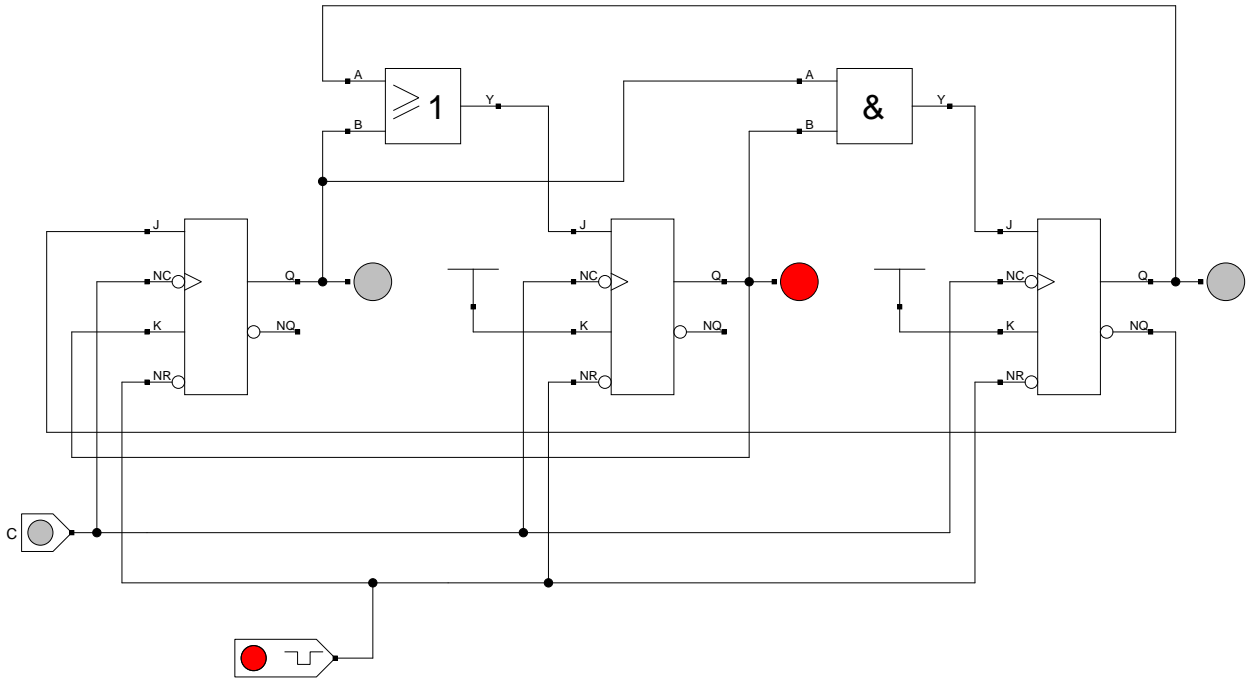


► Beschreibung

- umschaltbarer 2 Bit vorwärts- rückwärts- Zähler
- Zählfolge für $x = 0$ ist: 00, 01, 10, 11, 00, ...
- Zählfolge für $x = 1$ ist: 00, 11, 10, 01, 00, ...
- Beim Übergang zu 00 wird $y = 1$

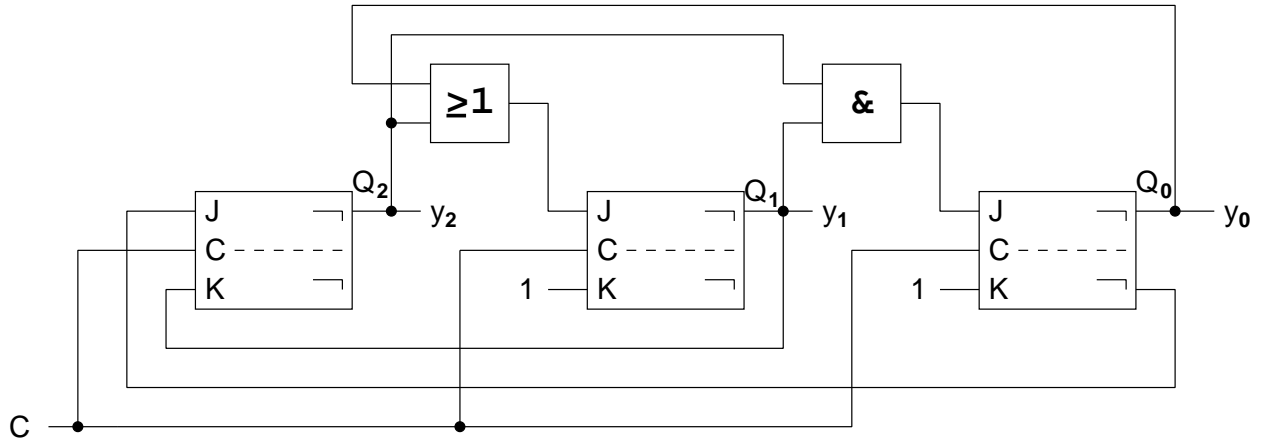
Analyse von Schaltwerken (JK-FF)

► jk_analyse.hds



Analyse von Schaltwerken (JK-FF)

- Beispiel: Schaltwerk mit JK-MS-FlipFlops:



- Schaltfunktionen

- Übergangsfunktion

$$J_2 = \bar{Q}_0$$

$$K_2 = Q_1$$

$$J_1 = Q_2 \vee Q_0$$

$$K_1 = 1$$

$$J_0 = Q_1 \wedge Q_2$$

$$K_0 = 1$$

- Ausgangsfunktion

$$y_2 = Q_2$$

$$y_1 = Q_1$$

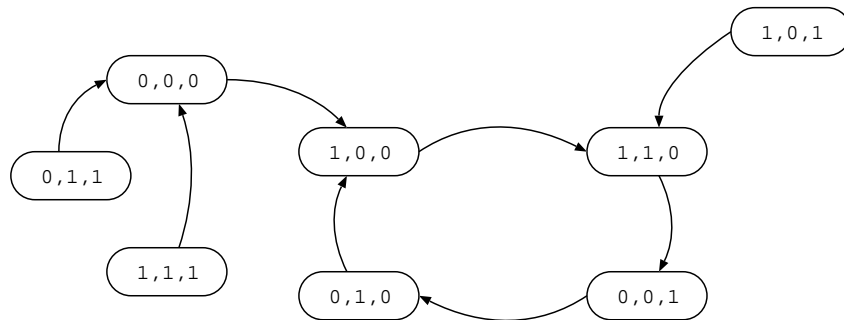
$$y_0 = Q_0$$

Analyse von Schaltwerken (JK-FF)

► Zustandsfolgetabelle:

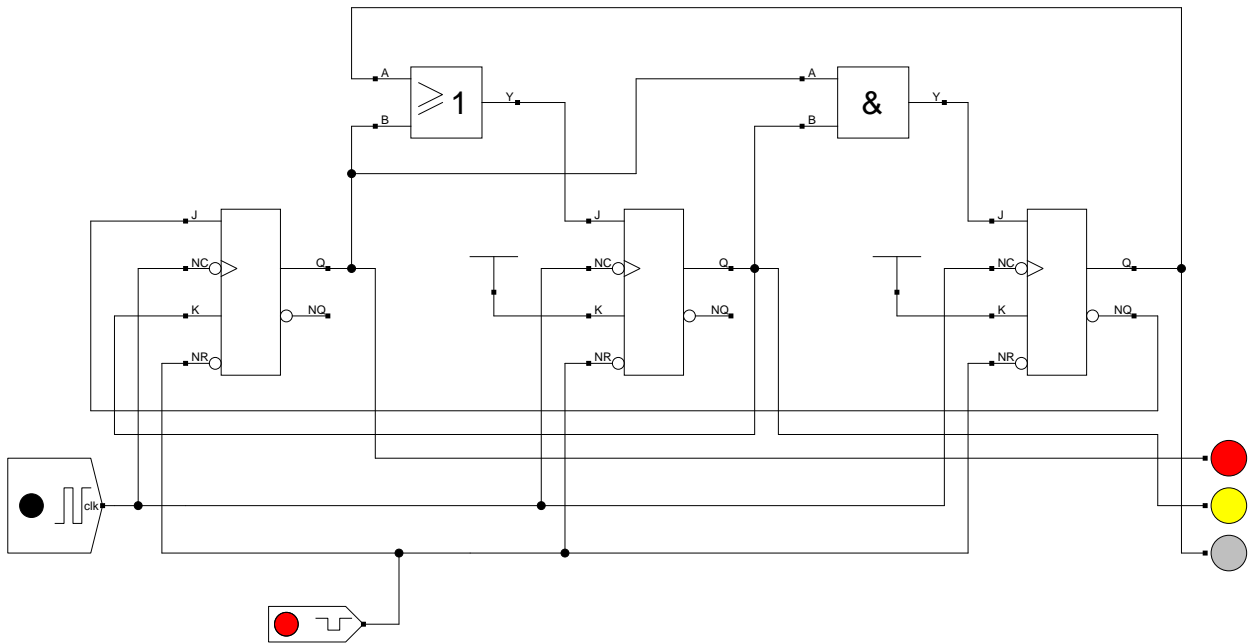
Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	Q_2^+	Q_1^+	Q_0^+
0	0	0	1	0	0	1	0	1	1	0	0
0	0	1	0	0	1	1	0	1	0	1	0
0	1	0	1	1	0	1	0	1	1	0	0
0	1	1	0	1	1	1	0	1	0	0	0
1	0	0	1	0	1	1	0	1	1	1	0
1	0	1	0	0	1	1	0	1	1	1	0
1	1	0	1	1	1	1	1	1	0	0	1
1	1	1	0	1	1	1	1	1	0	0	0

► Zustandgraph



Analyse von Schaltwerken (JK-FF)

► jk_analyse2.hds



► Beschreibung

- einfache Ampelsteuerung

Synthese von Schaltwerken

- ▶ Anzahl der Zustände ermitteln
 - bestimmt die Anzahl der benötigten FF
- ▶ Definition der Eingangs- und Ausgangsvariablen
- ▶ Anfangszustand festlegen
- ▶ zeitliche Zustandsfolge bestimmen
 - Zustandsgraphen
 - Zustandsfolgetabelle
- ▶ KV-Tafel erstellen
- ▶ Übergangsfunktion und Ausgangsfunktion ermitteln
- ▶ Schaltwerk erstellen
 - entwerfen
 - realisieren

Synthese von Schaltwerken

► einfaches Beispiel

- Entwerfen Sie ein Schaltwerk mit D-FF, das bei einer Eingabe von 0 mit jedem Taktimpuls den Zustand (0,1) ändert, und bei einer Eingabe von 1 den aktuellen Zustand hält. Der Zustand ist auszugeben.

1. Anzahl Zustände: 2 (0,1) \rightarrow 1 FF

2. Definition der Eingangs- und Ausgangsvariablen

Eingangsvariable x

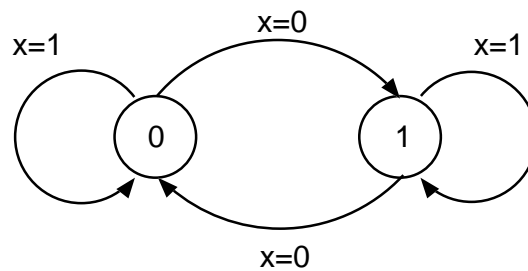
Ausgangsvariable y

Zustandsvariable Q mit $y = Q$

3. Anfangszustand 0

4. zeitliche Zustandsfolge

(a) Zustandsgraph



(b) Zustandsfolgetabelle

x	Q	Q^+	y
0	0	1	0
0	1	0	1
1	0	0	0
1	1	1	1

Synthese von Schaltwerken

► einfaches Beispiel

5. KV

	x		
		0	1
Q			
0		1	0
1		0	1

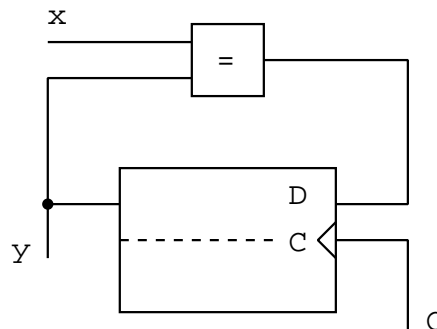
$$Q^+ = \bar{x}\bar{Q} \vee xQ$$

$$Q^+ = x \equiv Q$$

6. Funktionen

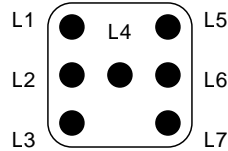
- Übergangsfunktion: $Q^+ = (x \equiv Q)$
- Ausgangsfunktion: $y = Q$

7. Schaltwerk



Schaltwerkssynthese mit D-FF

► Elektronischer Würfel:



- Übergangsfunktion: 1 ... 6 Dualzähler (3 FF)
- Ausgangsfunktion: Dual nach Würfel Codeumsetzer

► Zustandsfolgetabelle für Zähler

Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	0	×	×	×
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	×	×	×

► KV-Tafeln

		D_2				D_1				D_0							
Q_1	Q_0	00	01	11	10	Q_1	Q_0	00	01	11	10	Q_1	Q_0	00	01	11	10
Q_2																	
0	x	0		1	0	0	x	1		0	1	0	x	1	0	0	1
1	1	1	1	x	0	1	0	1		x	0	1	0	0	x		1

► Übergangsfunktionen:

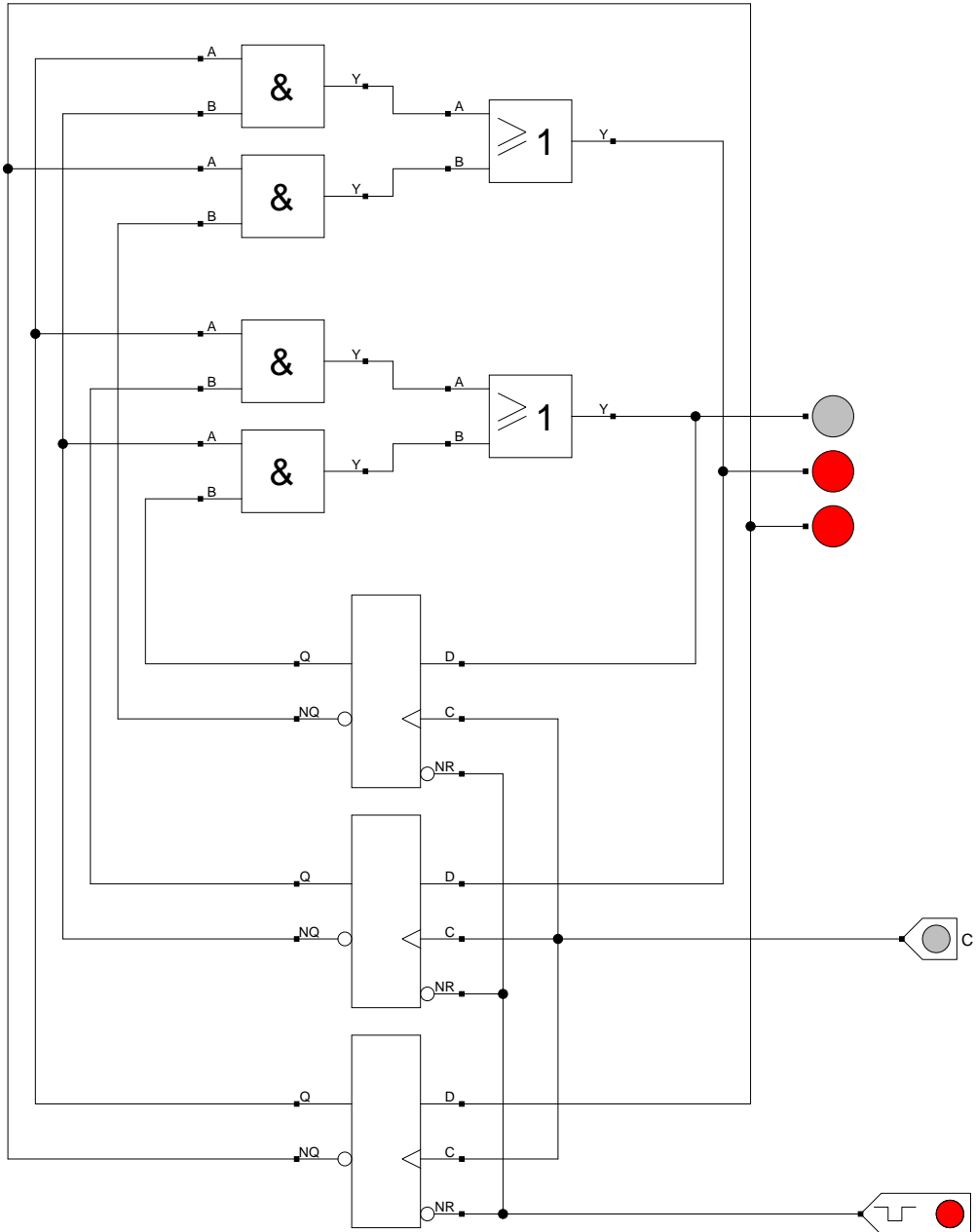
$$D_2 = Q_2 \bar{Q}_1 \vee Q_1 Q_0$$

$$D_1 = \bar{Q}_2 \bar{Q}_0 \vee \bar{Q}_1 Q_0$$

$$D_0 = \bar{Q}_0$$

Schaltwerkssynthese mit D-FF

- ▶ Schaltwerk des Zählers
- ▶ Hades: wuerfel_zaeehler.hds



Schaltwerkssynthese mit D-FF

► Ausgangsfunktionstabelle:

Q_2	Q_1	Q_0	L_1	L_2	L_3	L_4	L_5	L_6	L_7
0	0	0	×	×	×	×	×	×	×
0	0	1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	1	0	0
0	1	1	0	0	1	1	1	0	0
1	0	0	1	0	1	0	1	0	1
1	0	1	1	0	1	1	1	0	1
1	1	0	1	1	1	0	1	1	1
1	1	1	×	×	×	×	×	×	×

► KV-Tafeln:

L₁

$Q_1 Q_0$	00	01	11	10
Q_2				
0	x	0	0	0
1	1	1	x	1

L₂

$Q_1 Q_0$	00	01	11	10
Q_2				
0	x	0	0	0
1	0	0	x	1

L₃

$Q_1 Q_0$	00	01	11	10
Q_2				
0	x	0	1	1
1	1	1	x	1

L₄

$Q_1 Q_0$	00	01	11	10
Q_2				
0	x	1	1	0
1	0	1	x	0

► Ausgangsfunktionen:

$$L_1 = L_7 = Q_2$$

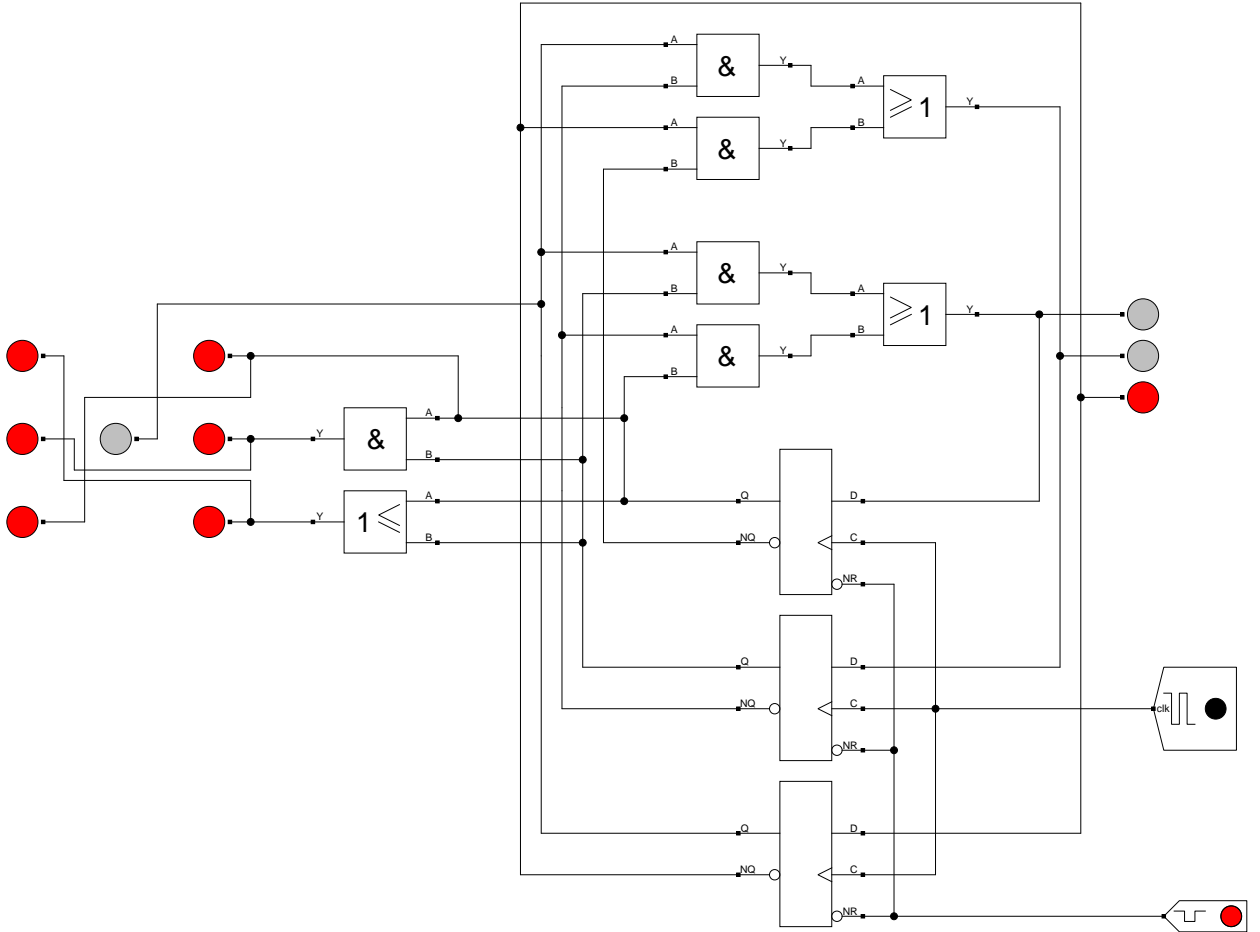
$$L_2 = L_6 = Q_1 \wedge Q_2$$

$$L_3 = L_5 = Q_1 \vee Q_2$$

$$L_4 = Q_0$$

Schaltwerkssynthese mit D-FF

- ▶ Schaltwerk des Würfels
- ▶ Hades: wuerfel.hds

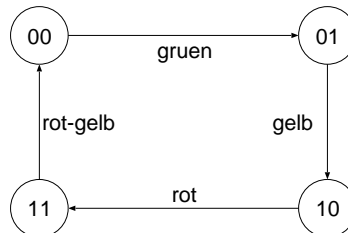


Schaltwerkssynthese mit JK-FF

- ▶ Beispiel: einfache Ampelsteuerung
- ▶ Signalfolge
 - rot; rot-gelb; grün; gelb; rot ...
 - 4 Zustände \rightarrow 2 FF (Q_0, Q_1)
- ▶ Zustandskodierung

Q_1	Q_0	Ausgabe
0	0	grün
0	1	gelb
1	0	rot
1	1	rot-gelb

- ▶ Zustandgraph



- ▶ Zustandsfolgetabelle

Q_1	Q_0	Q_1^+	Q_0^+	r	ge	gr
0	0	0	1	0	0	1
0	1	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	1	1	0

- ▶ Ausgangsfunktionen

$$ro = Q_1$$

$$ge = Q_0$$

$$gr = \overline{Q_0 \vee Q_1}$$

Schaltwerkssynthese mit JK-FF

► Übergangsfunktion eines JK-FF

- Zustandsfolgetabelle für JK-FF

Q ist	Q^+ soll werden	dann muß J/K sein	
		J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

- Zustandsfolgetabelle für Zähler

Q_1	Q_0	Q_1^+	Q_0^+	J_1	K_1	J_0	K_0
0	0	0	1	0	×	1	×
0	1	1	0	1	×	×	1
1	0	1	1	×	0	1	×
1	1	0	0	×	1	×	1

► KV-Tafeln Zähler

J_1	Q_0		
	Q_1	0	1
0		0	1
1		×	×

K_1	Q_0		
	Q_1	0	1
0		×	×
1		0	1

J_0	Q_0		
	Q_1	0	1
0		1	×
1		×	1

K_0	Q_0		
	Q_1	0	1
0		×	1
1		1	×

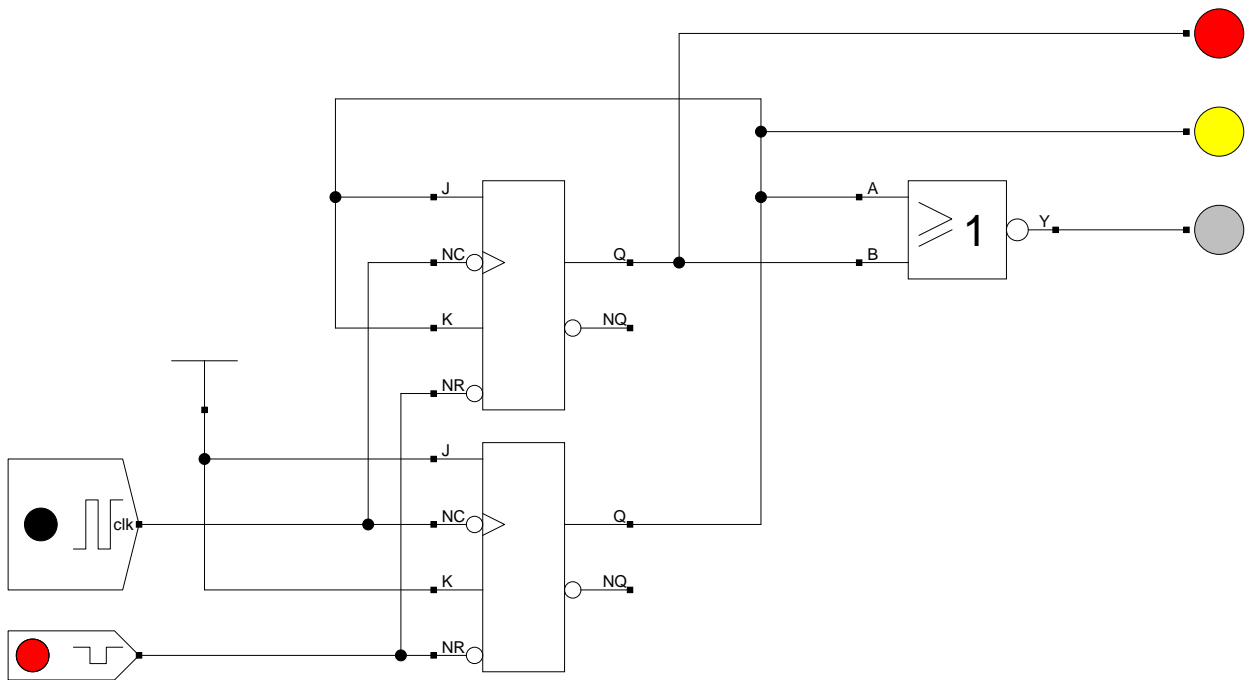
► Übergangsfunktionen

$$J_1 = K_1 = Q_0$$

$$J_0 = K_0 = 1$$

Schaltwerkssynthese mit JK-FF

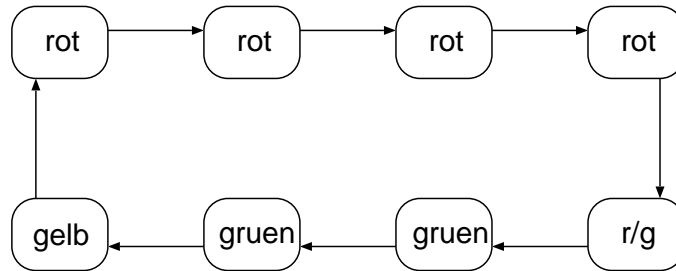
- ▶ Beispiel: einfache Ampelsteuerung
- ▶ Schaltwerk
- ▶ Hades: jk_synthese.hds



Schaltwerkssynthese mit JK-FF

► Kreuzungsampel mit verschiedenen langen Phasen

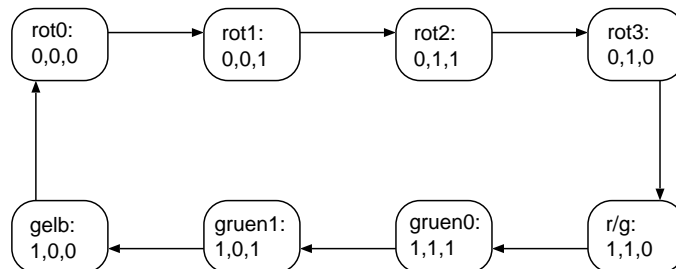
- Rotphase solange wie rot/gelb, grün und gelb zusammen



► keine direkte Kodierung möglich (da Übergang rot → rot)

- Differenzierung der farbgleichen Phasen durch Indizes

► Kodierung über 3-Bit Graycode:



Q_2	Q_1	Q_0	r	ge	gr
0	0	0	1	0	0
0	0	1	1	0	0
0	1	1	1	0	0
0	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	1
1	0	1	0	0	1
1	0	0	0	1	0

Schaltwerkssynthese mit JK-FF

- ▶ Kreuzungsampel mit verschieden langen Phasen
- ▶ Zustandsfolgetabelle für Zähler:

Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	×	0	×	1	×
0	0	1	0	1	1	0	×	1	×	×	0
0	1	0	1	1	0	1	×	×	0	0	×
0	1	1	0	1	0	0	×	×	0	×	1
1	0	0	0	0	0	×	1	0	×	0	×
1	0	1	1	0	0	×	0	0	×	×	1
1	1	0	1	1	1	×	0	×	0	1	×
1	1	1	1	0	1	×	0	×	1	×	0

- ▶ KV-Tafeln:

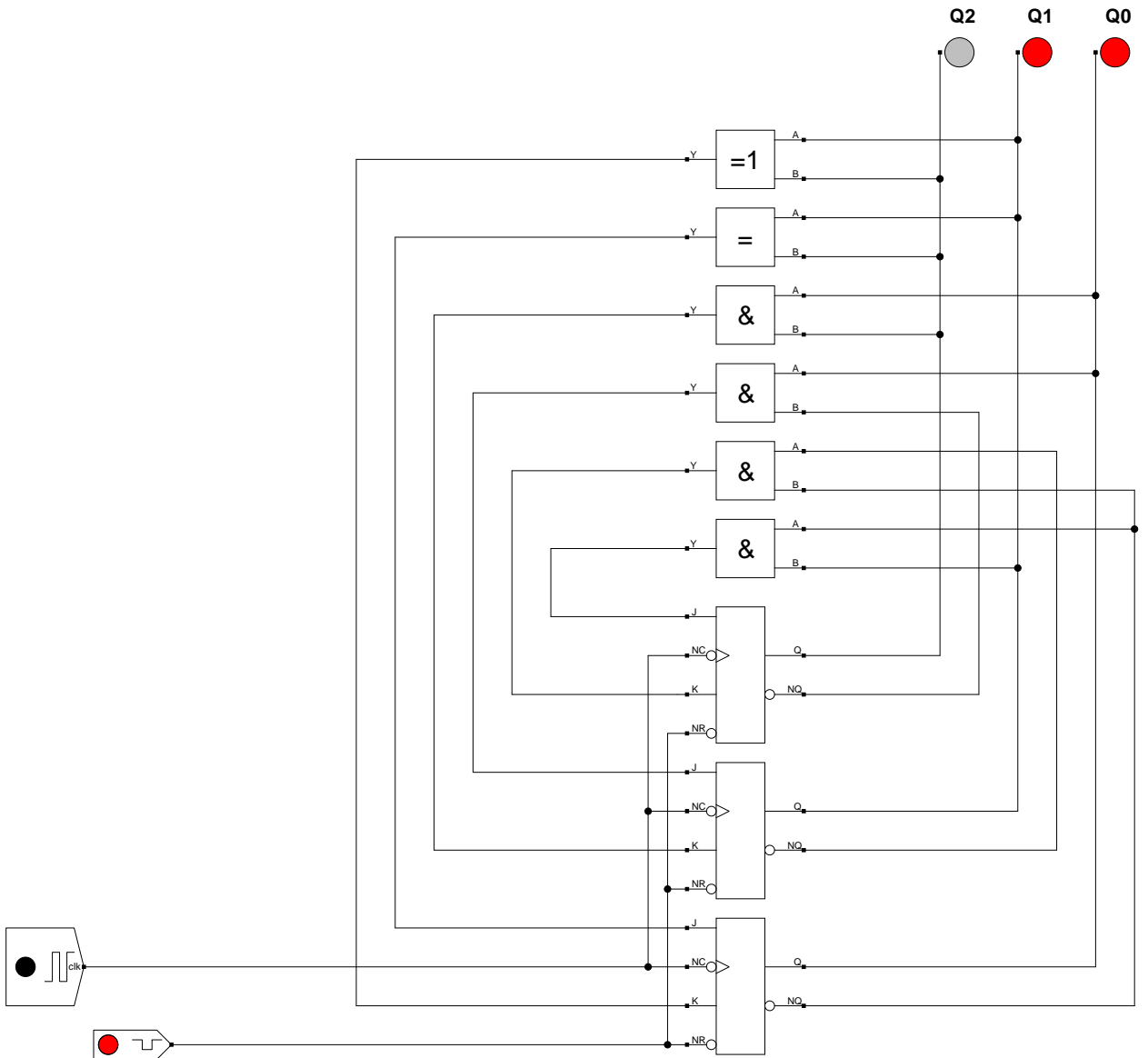
J_2	Q_1Q_0 Q_2	00	01	11	10		K_2	Q_1Q_0 Q_2	00	01	11	10
	0	0	0	0	1			0	x	x	x	x
	1	x	x	x	x			1	1	0	0	0
J_1	Q_1Q_0 Q_2	00	01	11	10		K_1	Q_1Q_0 Q_2	00	01	11	10
	0	0	1	x	x			0	x	x	0	0
	1	0	0	x	x			1	x	x	1	0
J_0	Q_1Q_0 Q_2	00	01	11	10		K_0	Q_1Q_0 Q_2	00	01	11	10
	0	1	x	x	0			0	x	0	1	x
	1	0	x	x	1			1	x	1	0	x

- ▶ Übergangsfunktionen:

$$\begin{array}{ll}
 J_2 = \bar{Q}_0Q_1 & K_2 = \bar{Q}_0\bar{Q}_1 \\
 J_1 = Q_0\bar{Q}_2 & K_1 = Q_0Q_2 \\
 J_0 = Q_1 \equiv Q_2 & K_0 = Q_1 \neq Q_2
 \end{array}$$

Schaltwerkssynthese mit JK-FF

- ▶ Kreuzungsampel mit verschieden langen Phasen
- ▶ ampel_jk_zaebler.hds



Schaltwerkssynthese mit JK-FF

- ▶ Kreuzungsampel mit verschieden langen Phasen
- ▶ Ausgangsfunktionstabelle:

Q_2	Q_1	Q_0	r	ge	gr
0	0	0	1	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	1	1	0
1	1	1	0	0	1

- ▶ KV-Tafeln:

rot	$Q_1 Q_0$ Q_2	00	01	11	10		gelb	$Q_1 Q_0$ Q_2	00	01	11	10
	0	1	1	1	1			0	0	0	0	0
	1	0	0	0	1			1	1	0	0	1

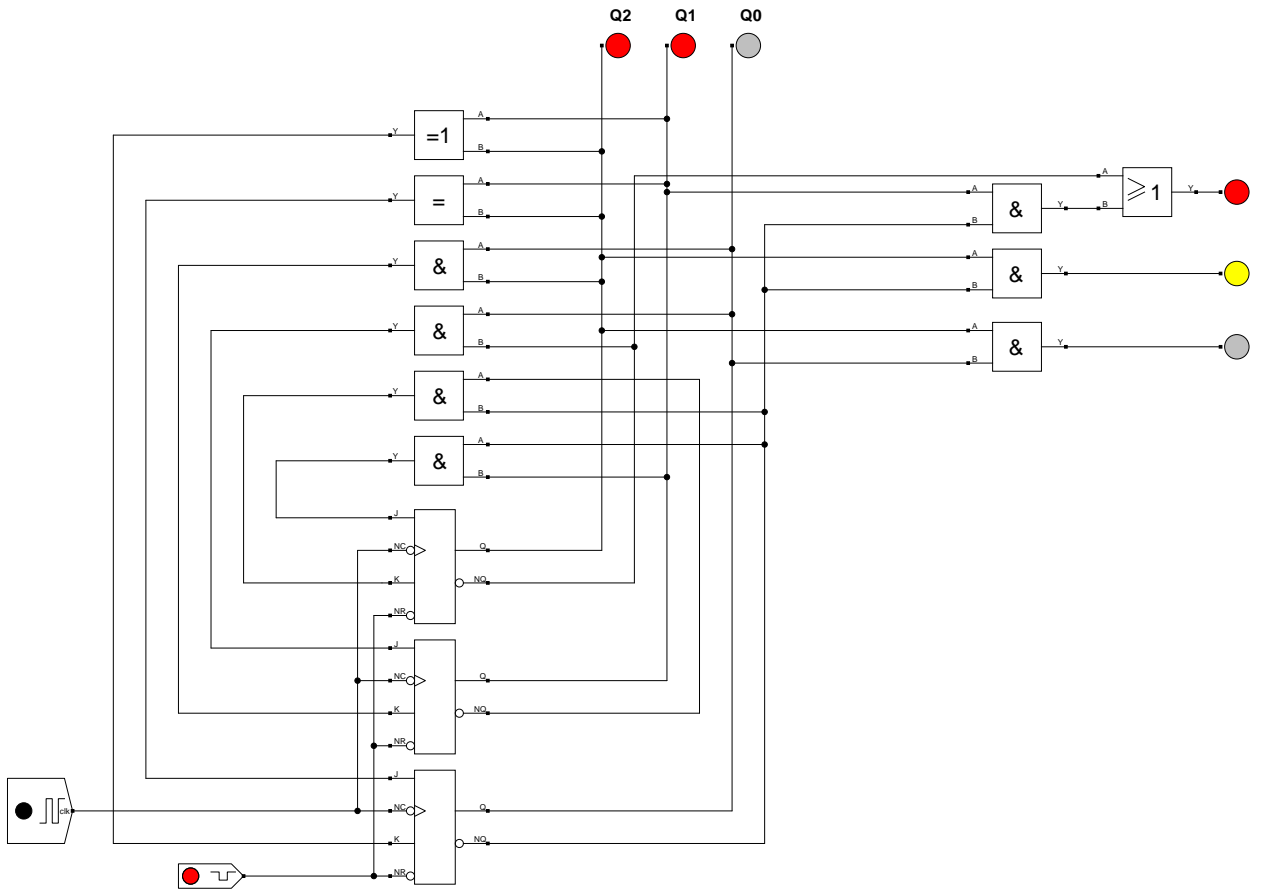
gruen	$Q_1 Q_0$ Q_2	00	01	11	10
	0	0	0	0	0
	1	0	1	1	0

- ▶ Ausgangsfunktionen:

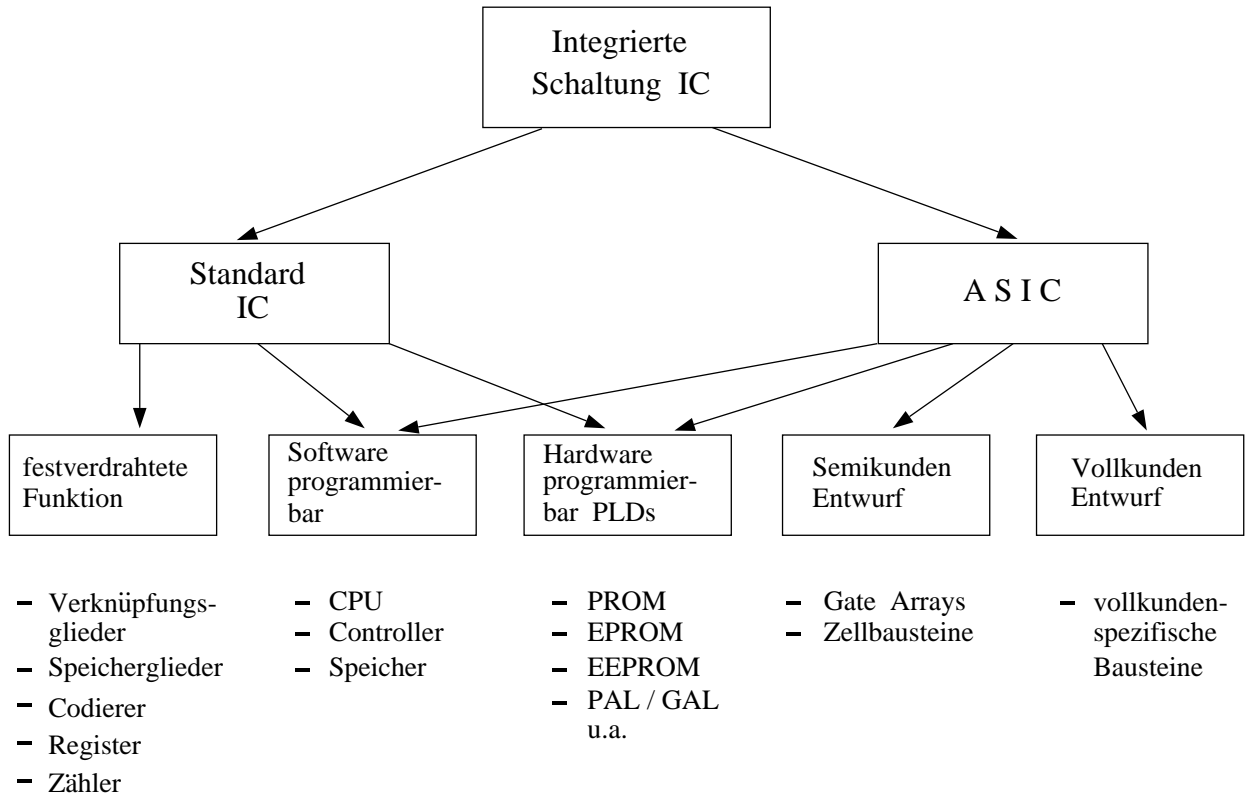
$$\begin{aligned}
 rot &= \bar{Q}_2 \vee Q_1 \bar{Q}_0 \\
 gelb &= \bar{Q}_0 Q_2 \\
 grün &= Q_0 Q_2
 \end{aligned}$$

Schaltwerkssynthese mit JK-FF

- ▶ Kreuzungsampel mit verschiedenen langen Phasen
- ▶ ampel_jk.hds



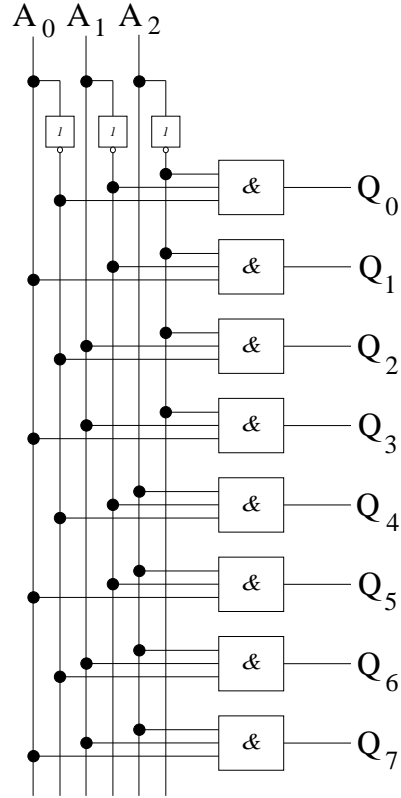
Integrierte Schaltungen



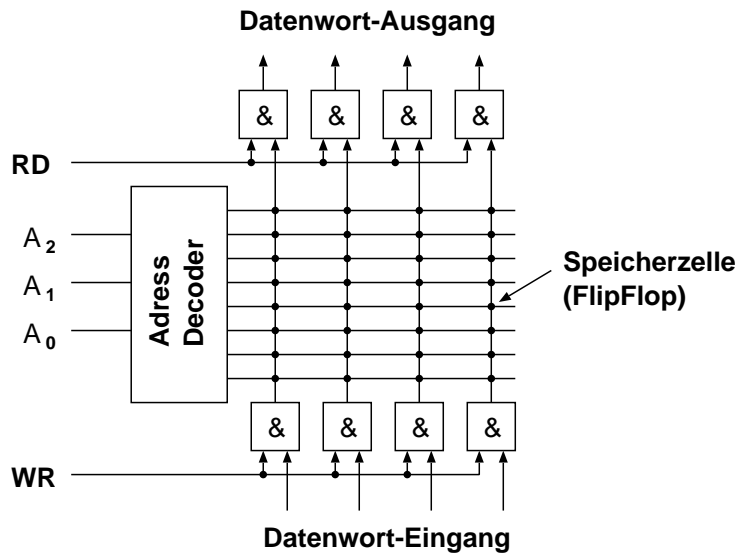
Speicherbausteine

- ▶ Adressdecodierer
- ▶ Minterme

$$\begin{aligned}
 Q_0 &= \bar{A}_0 \wedge \bar{A}_1 \wedge \bar{A}_2 \\
 Q_1 &= A_0 \wedge \bar{A}_1 \wedge \bar{A}_2 \\
 Q_2 &= \bar{A}_0 \wedge A_1 \wedge \bar{A}_2 \\
 Q_3 &= A_0 \wedge A_1 \wedge \bar{A}_2 \\
 Q_4 &= \bar{A}_0 \wedge \bar{A}_1 \wedge A_2 \\
 Q_5 &= A_0 \wedge \bar{A}_1 \wedge A_2 \\
 Q_6 &= \bar{A}_0 \wedge A_1 \wedge A_2 \\
 Q_7 &= A_0 \wedge A_1 \wedge A_2
 \end{aligned}$$



- ▶ Speicherbaustein



Speicherbausteine

- ▶ programmierbare Bausteine sind universelle Schaltnetze
- ▶ Beispiel: RAM
 - 8 Bit Wortbreite
 - jede Speicherzelle besteht aus 8 Bit (Werte von \$0 bis \$FF)
- ▶ Annahme: folgende Werte wurden einprogrammiert

- Inhalt:

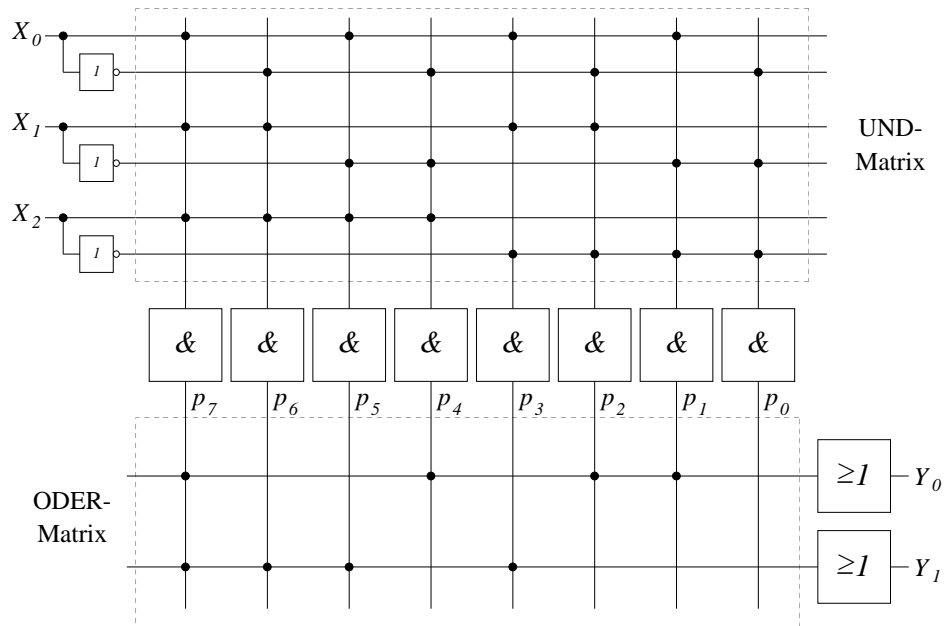
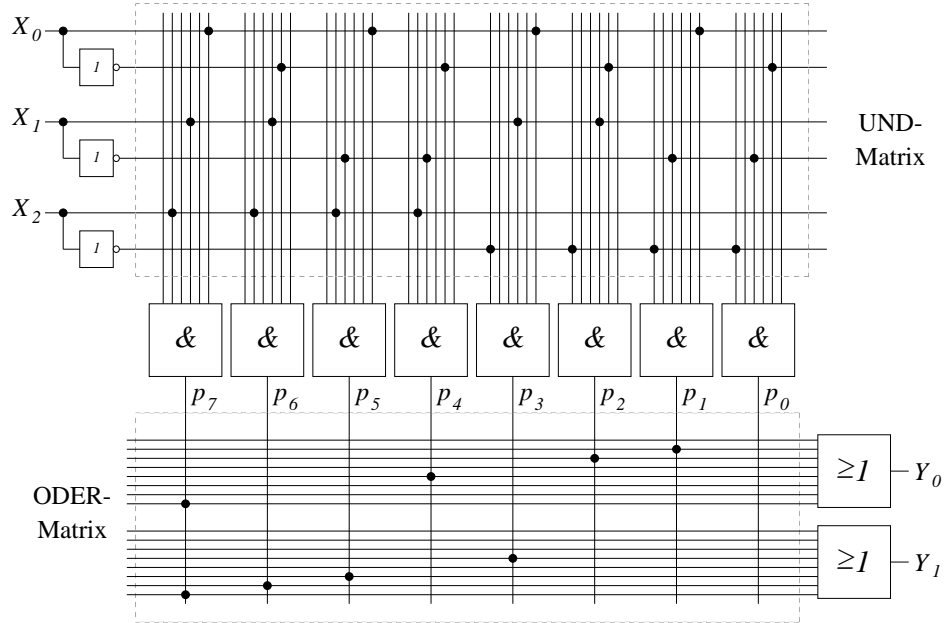
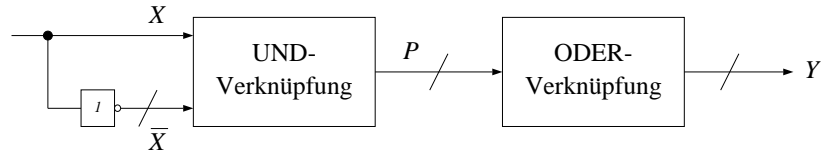
Adresse	Inhalt
0	\$EC
1	\$96
2	\$56
3	\$23

- im Dualcode

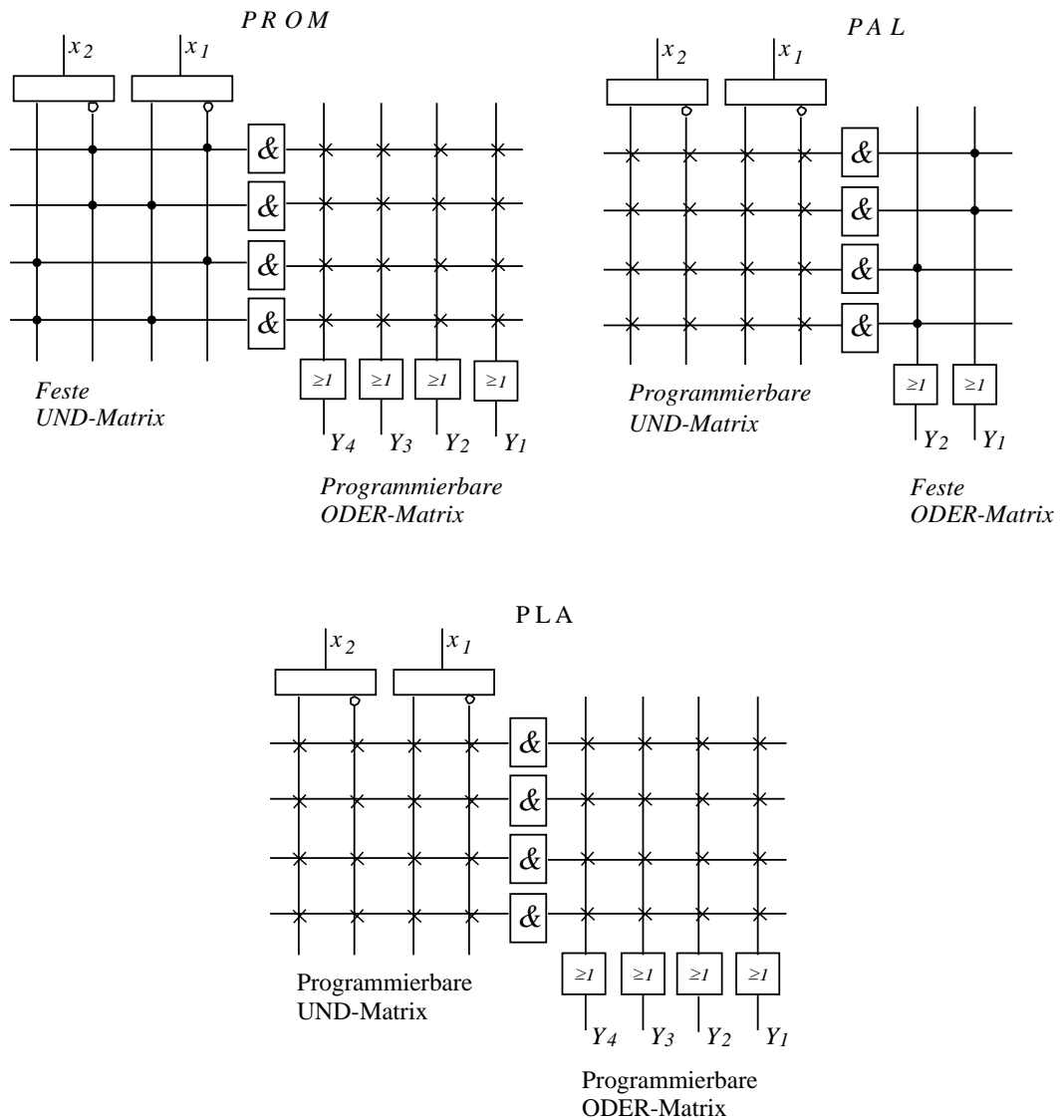
Adresse		Inhalt							
A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	1	1	1	0	1	1	0	0
0	1	1	0	0	1	0	1	1	0
1	0	0	1	0	1	0	1	1	0
1	1	0	0	1	0	0	0	1	1
Funktion		\bar{A}_1	\bar{A}_0	$A_0 \equiv A_1$	$A_0 \neq A_1$	$\bar{A}_0 \vee \bar{A}_1$	$\bar{A}_0 \wedge \bar{A}_1$	$A_0 \vee A_1$	$A_0 \wedge A_1$

- ▶ m-Bit Wortbreite \rightarrow m Funktionen realisierbar
- ▶ n Adressleitungen \rightarrow n Eingangsvariablen
- ▶ Speicherbausteine realisieren *alle* Funktionen in DNF

PLD-Struktur



Programmable Logic Devices (PLD)



	UND-Matrix	ODER-Matrix
ROM	fest	fest
(E)PROM	fest	programmierbar
PAL	programmierbar	fest
PLA	programmierbar	programmierbar

Übersicht PLD

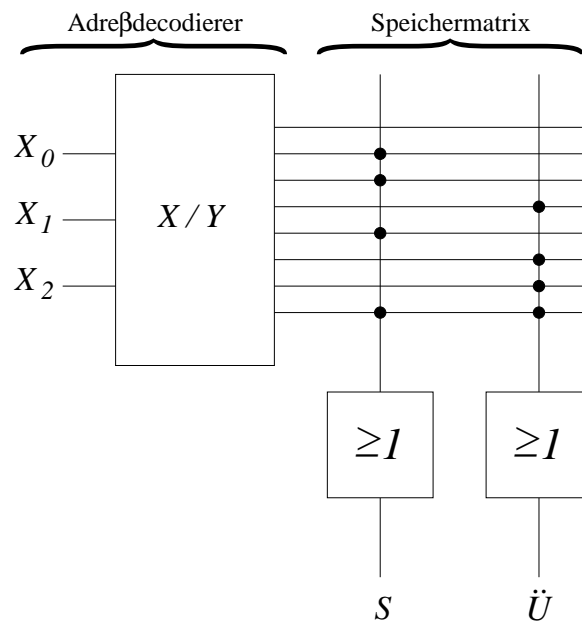
PLD-Baustein Typen	UND- Matrix	ODER- Matrix	Besonderheiten
PROM	fest	prog-bar	
EPROM (Erasable PROM)	fest	prog-bar	UV – löschbares PROM
EEPROM (Electrically Erasable PROM)	fest	prog-bar	elektrisch löschbares PROM
PAL (Programmable Array Logic)	prog-bar	fest	PAL ist ein eingetragenes Warenzeichen der Firma Monolithic Memories Inc., USA
GAL (Genetic Array Logic)	prog-bar	fest	elektrisch programmierbar und elektrisch löschbares PAL, GAL ist ein Warenzeichen der Firma Lattice Semiconductor
HAL (Hardware Array Logic)	fest	fest	PAL mit ab Hersteller kundenspezifisch festverdrahteter UND-Matrix
IFL (Integrated Fuse Logic)			Familienbezeichnung der von Valvo hergestellten PLDs. Dazu gehören die Typen:
FPGA (Field Programmable Gate Array)	prog-bar	fest	PAL ohne Register
FPLA (Field Programmable Logic Array)	prog-bar	prog-bar	PLA ohne Register
FPLS (Field Programmable Logic Sequencer)	prog-bar	prog-bar	PLA mit JK/D-Flipflops
LCA (Logic Cell Array)	prog-bar	prog-bar	programmierbare E-/Ausgabeblocke, konfigurierbare Logikblöcke und Verbindungsleitungen

ROM/PROM/(E)EPROM

► Aufbau

- Adressdecoder
- Speichermatrix mit programmierbaren Koppellementen

► Beispiel: VA

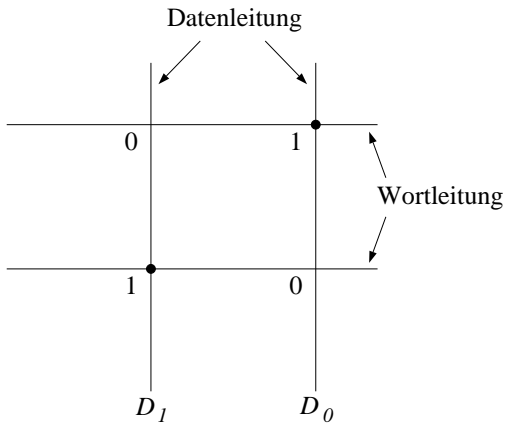


► Programmcode ab Adresse 0: 0,2,2,1,2,1,1,3

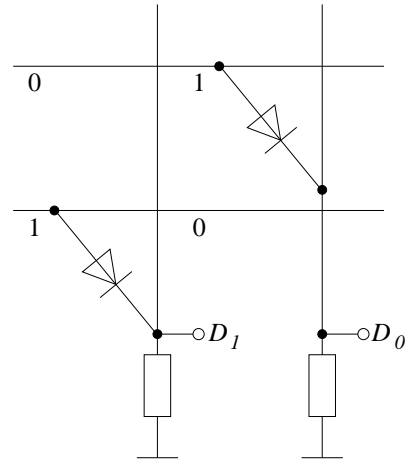
► Ausführungen:

- ROM: Read-Only-Memory
 - maskenprogrammierbare ODER-Matrix
- PROM: Programmable Read-Only-Memory
 - einmalprogrammierbare ODER-Matrix
- (E)EPROM: (Electrically) Erasable Programmable Read-Only-Memory
 - ODER-Matrix programmierbar und löschar

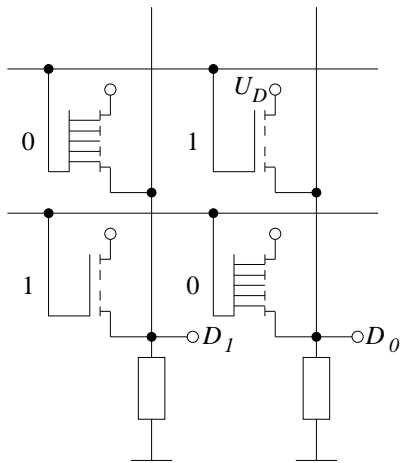
Koppelemente



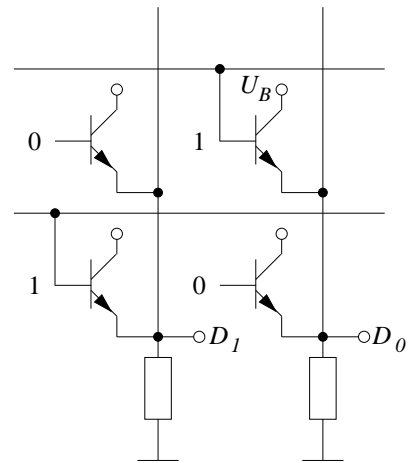
a) Schematische Darstellung



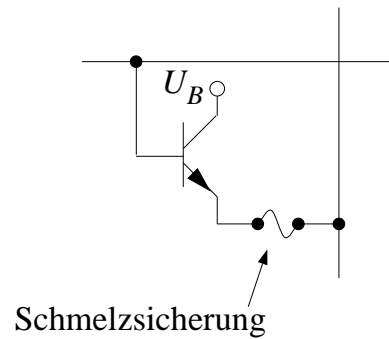
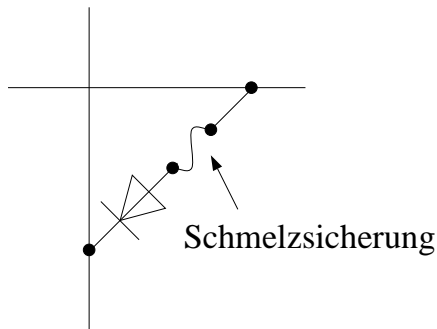
b) mit Dioden



c) mit MOS-Transistoren



d) mit NPN-Transistoren



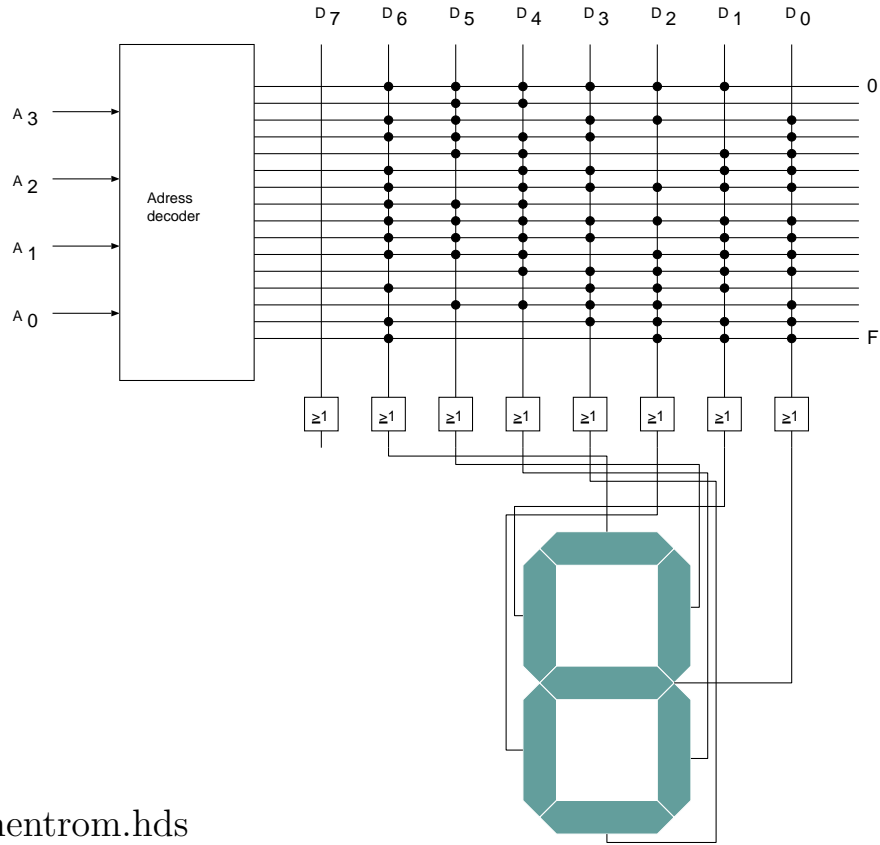
Schaltnetzentwurf mit Eproms

- ▶ 7-Segmentanzeige
- ▶ 7Segmente → 7 Funktionen
- ▶ 4 Eingangsvariablen (\$0 bis \$F) → 4 Adressleitungen, 16 Speicherworte
- ▶ Segment high-activ, dh, ON bei 1 bzw. OFF bei 0

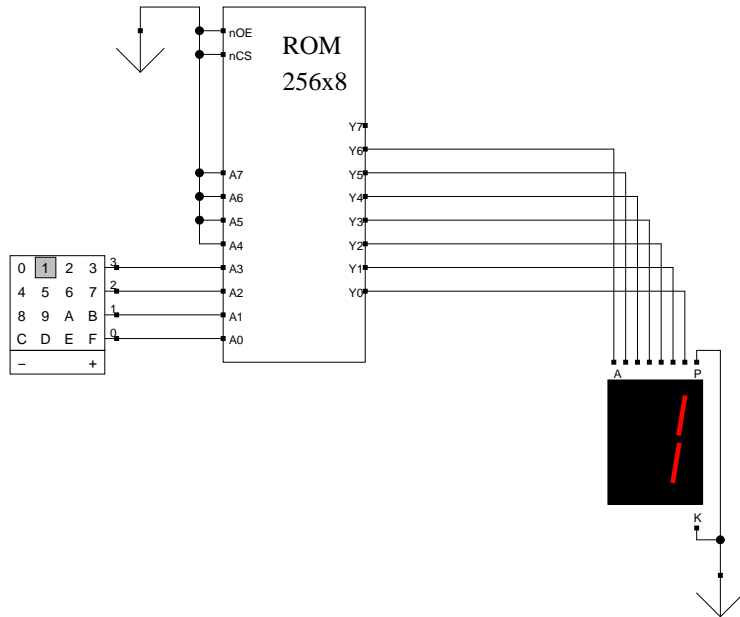
A_3	A_2	A_1	A_0	D_7	a D_6	b D_5	c D_4	d D_3	e D_2	f D_1	g D_0	HEX
0	0	0	0	0	1	1	1	1	1	1	0	7E
0	0	0	1	0	0	1	1	0	0	0	0	30
0	0	1	0	0	1	1	0	1	1	0	1	6D
0	0	1	1	0	1	1	1	1	0	0	1	79
0	1	0	0	0	0	1	1	0	0	1	1	33
0	1	0	1	0	1	0	1	1	0	1	1	5B
0	1	1	0	0	1	0	1	1	1	1	1	5F
0	1	1	1	0	1	1	1	0	0	0	0	70
1	0	0	0	0	1	1	1	1	1	1	1	7F
1	0	0	1	0	1	1	1	1	0	1	1	7B
1	0	1	0	0	1	1	1	0	1	1	1	77
1	0	1	1	0	0	0	1	1	1	1	1	1F
1	1	0	0	0	1	0	0	1	1	1	0	4E
1	1	0	1	0	0	1	1	1	1	0	1	3D
1	1	1	0	0	1	0	0	1	1	1	1	4F
1	1	1	1	0	1	0	0	0	1	1	1	47

Schaltnetzentwurf mit Eproms

► 7 Segment Programmierung



► 7segmentrom.hds



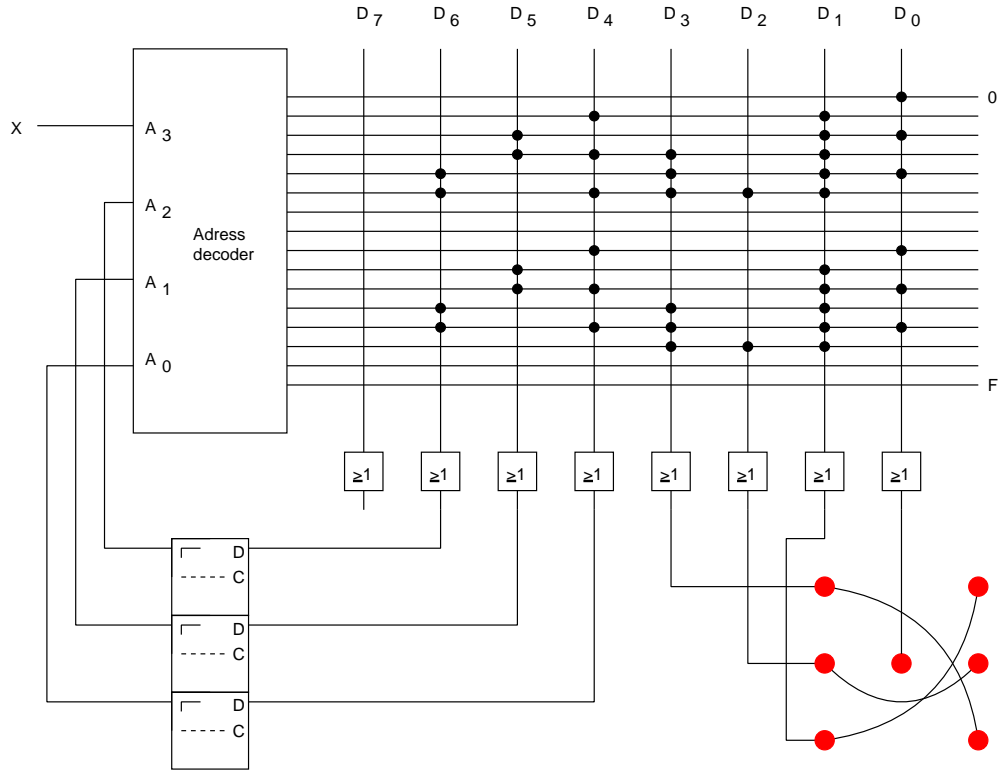
Schaltwerksentwurf mit Eproms

- ▶ Würfel
- ▶ $x = 1 \rightarrow$ zyklischer Zähler 0-5 mit Würfelausgabe
- ▶ $x = 0 \rightarrow$ Halten des aktuellen Zustands mit Würfelausgabe
- ▶ Trennung: Übergangsfunktion und Ausgangsfunktion

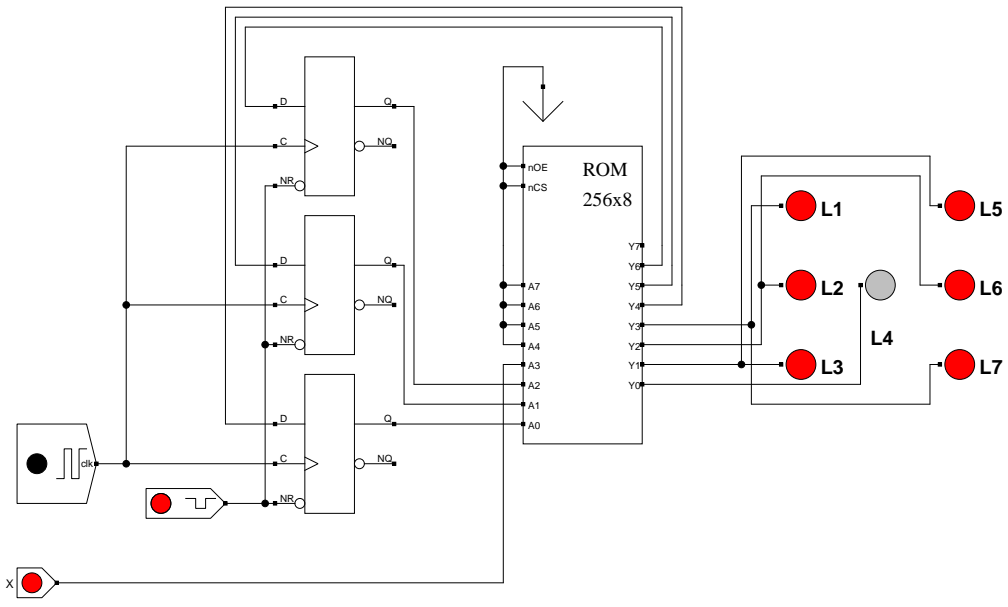
X	Q_2	Q_1	Q_0		Q_2^+	Q_1^+	Q_0^+	L_1	L_2	L_3	L_4	
A_3	A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Hex
0	0	0	0	0	0	0	0	0	0	0	1	01
0	0	0	1	0	0	0	1	0	0	1	0	12
0	0	1	0	0	0	1	0	0	0	1	1	23
0	0	1	1	0	0	1	1	1	0	1	0	3A
0	1	0	0	0	1	0	0	1	0	1	1	4B
0	1	0	1	0	1	0	1	1	1	1	0	5E
⋮				⋮				⋮				⋮
1	0	0	0	0	0	0	1	0	0	0	1	11
1	0	0	1	0	0	1	0	0	0	1	0	22
1	0	1	0	0	0	1	1	0	0	1	1	33
1	0	1	1	0	1	0	0	1	0	1	0	4A
1	1	0	0	0	1	0	1	1	0	1	1	5B
1	1	0	1	0	0	0	0	1	1	1	0	0E
⋮				⋮				⋮				⋮

Schaltwerkkentwurf mit Eproms

► Würfel-Programmierung

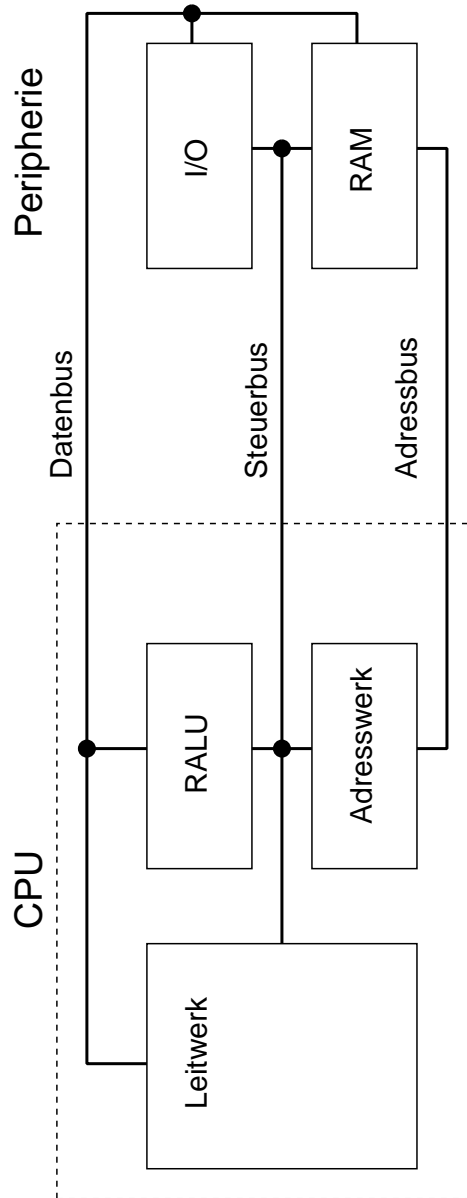


► wuerfelrom.hds



Rechnerarchitektur

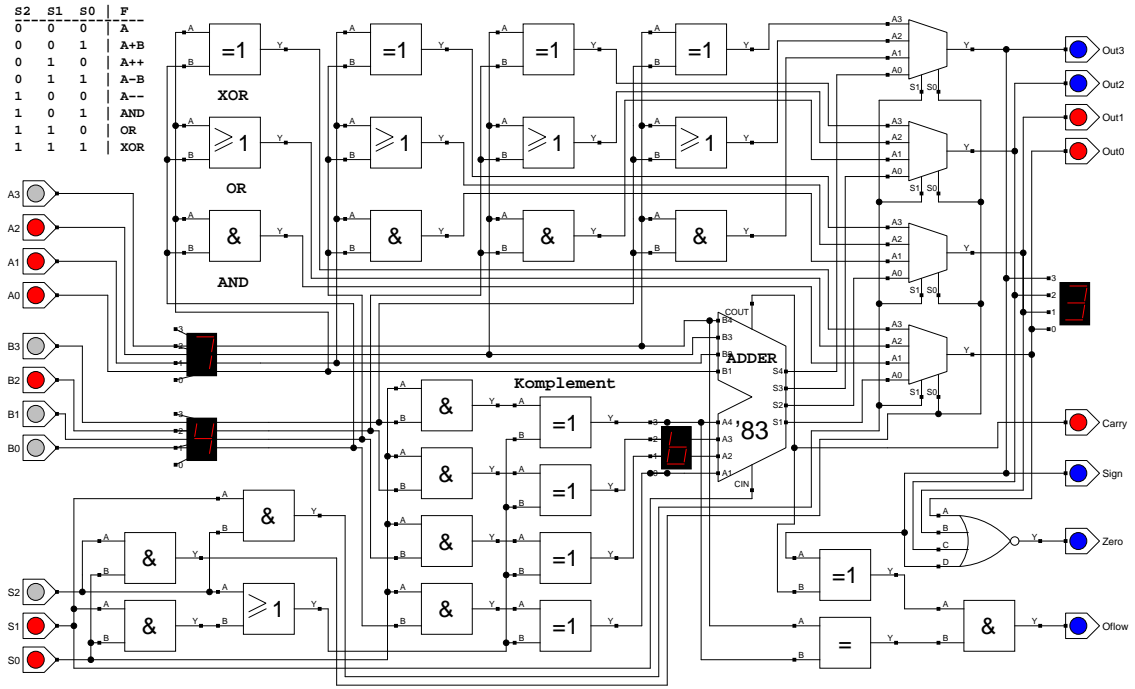
- ▶ Komponenten einer grundlegenden Rechnerarchitektur



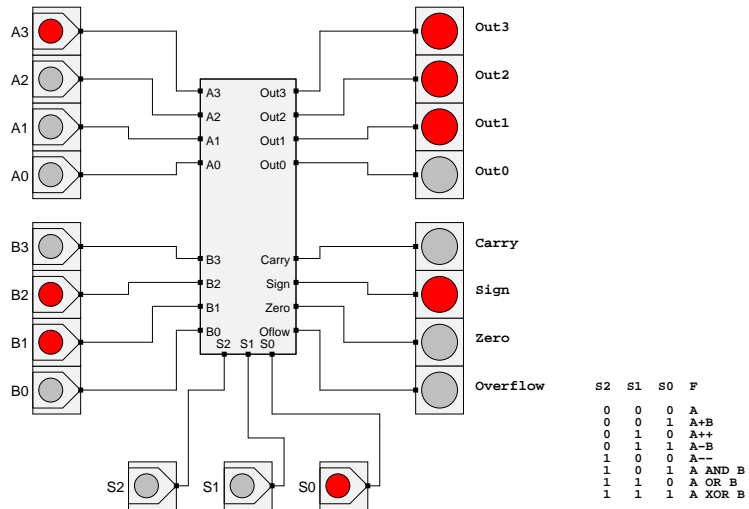
- ▶ Entwicklung: Step by Step

Entwicklung einer Mini CPU

► ALU wird zu

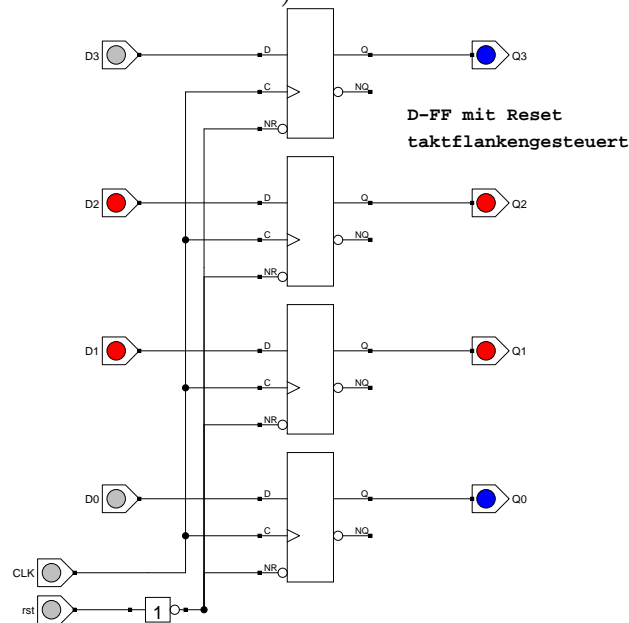


► einem Hades Subdesign



Arithmetisch-logische Einheit mit Register (RALU)

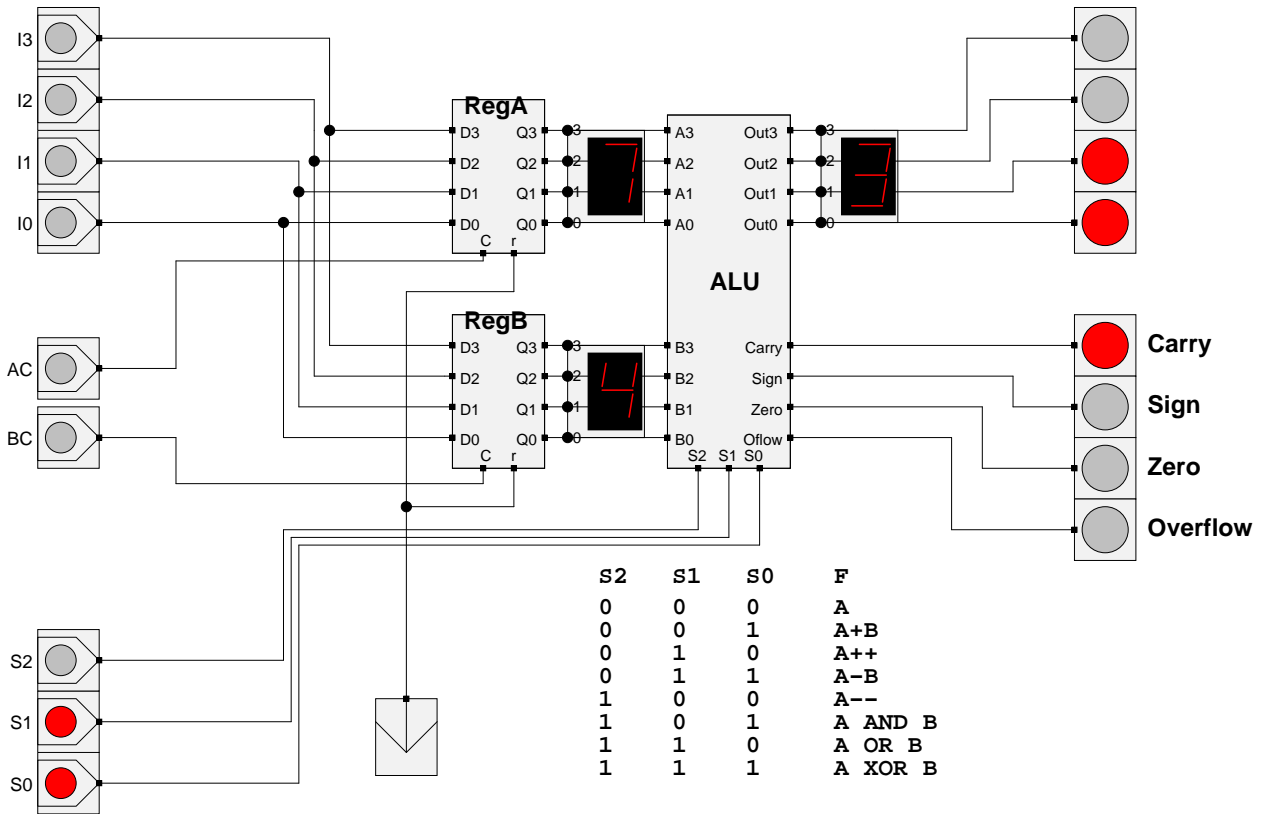
- ▶ Verarbeitung *zweier* Datenworte
 - Eingabeeinheit kann nur ein Wort zu einem Zeitpunkt liefern
 - Zwischenspeicher nötig
- ▶ Register (4xD-FF: TTL 74175)



- taktflankengesteuerte D-FF (4 Bit)
- Datenübernahme durch Takt gesteuert
- Daten müssen stabil zum Zeitpunkt der Taktflanke(!) sein
- \bar{Q} ermöglicht weitere Verknüpfungen
 - $\bar{A} \vee \bar{B} = \overline{A \wedge B}$ NAND
 - $\bar{A} \wedge \bar{B} = \overline{A \vee B}$ NOR
 - $A \neq \bar{B} = A \equiv B$ XNOR
- Reset wird im folgenden nicht benötigt

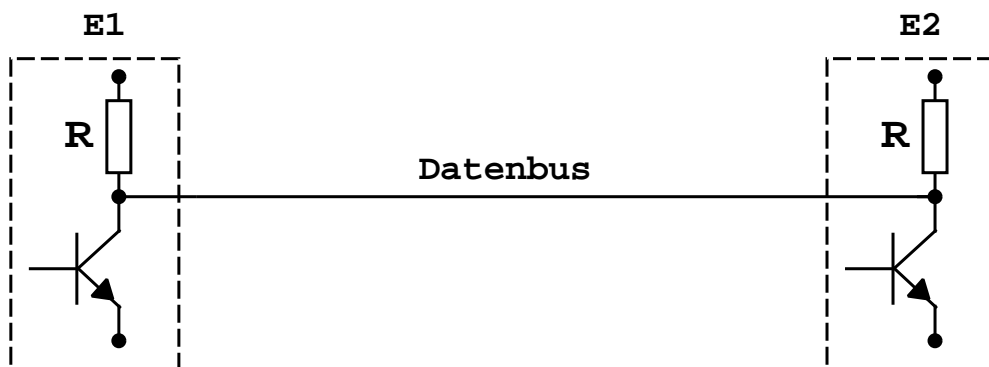
Arithmetisch-logische Einheit mit Register (RALU)

► Hades: 7_ralu/ralu.hds



Datenbus

- ▶ Datenfluss
 - Woher kommen die Daten (Lieferant)
 - Wohin fließen die Daten (Konsument)
- ▶ Datenbus
 - Verbindungselement zwischen Lieferanten und Konsumenten
 - mehrere Einheiten greifen lesend und schreibend auf den Bus zu
 - ↳ Eingabeeinheit schreibend
 - ↳ Register lesend
 - ↳ ALU schreibend
 - ↳ (Ausgabeeinheit lesend)
- ▶ Verhalten nicht schreibender Einheiten



- ▶ zwei Möglichkeiten:
 - E1 will nicht auf den Bus schreiben und gibt LOW-Pegel aus

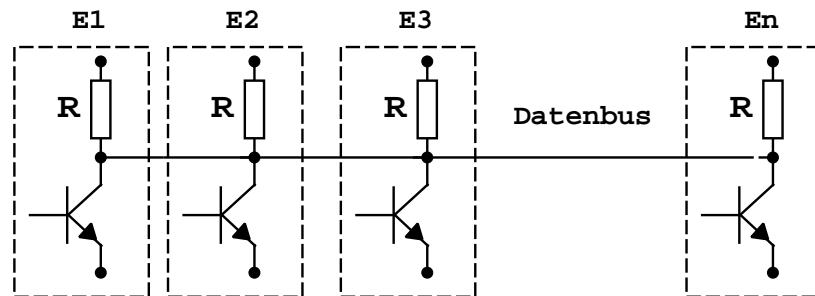
E1	E2	Datenbus
0	0	0
0	1	0

- E1 will nicht auf den Bus schreiben und gibt HIGH-Pegel aus

E1	E2	Datenbus
1	0	0
1	1	1

↳ prinzipiell möglich

TriState



► Problem bei vielen Busteilnehmern:

- $R_g = \frac{R}{n}$
- $R_{T-sperr} \gg R_g \gg R_{T-leit}$ gilt nicht mehr

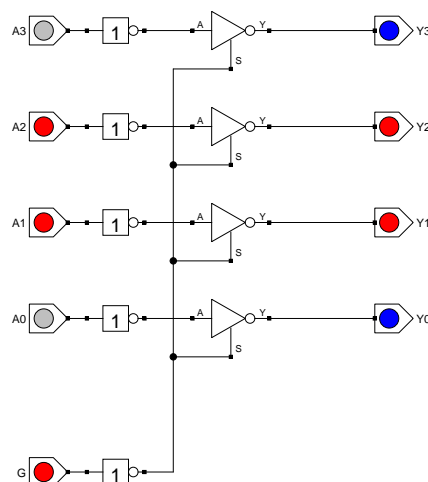
► Deshalb: gleichzeitiger schreibender Zugriff wird verboten

- schreibende Einheiten müssen vom Bus abgekoppelt werden können

► Tri-State Bausteine (zB. TTL 74126)

- 3 Zustände 0, 1, *hochohmig*

S	E	Q
0	0	H (hochohmig)
0	1	H (hochohmig)
1	0	0
1	1	1

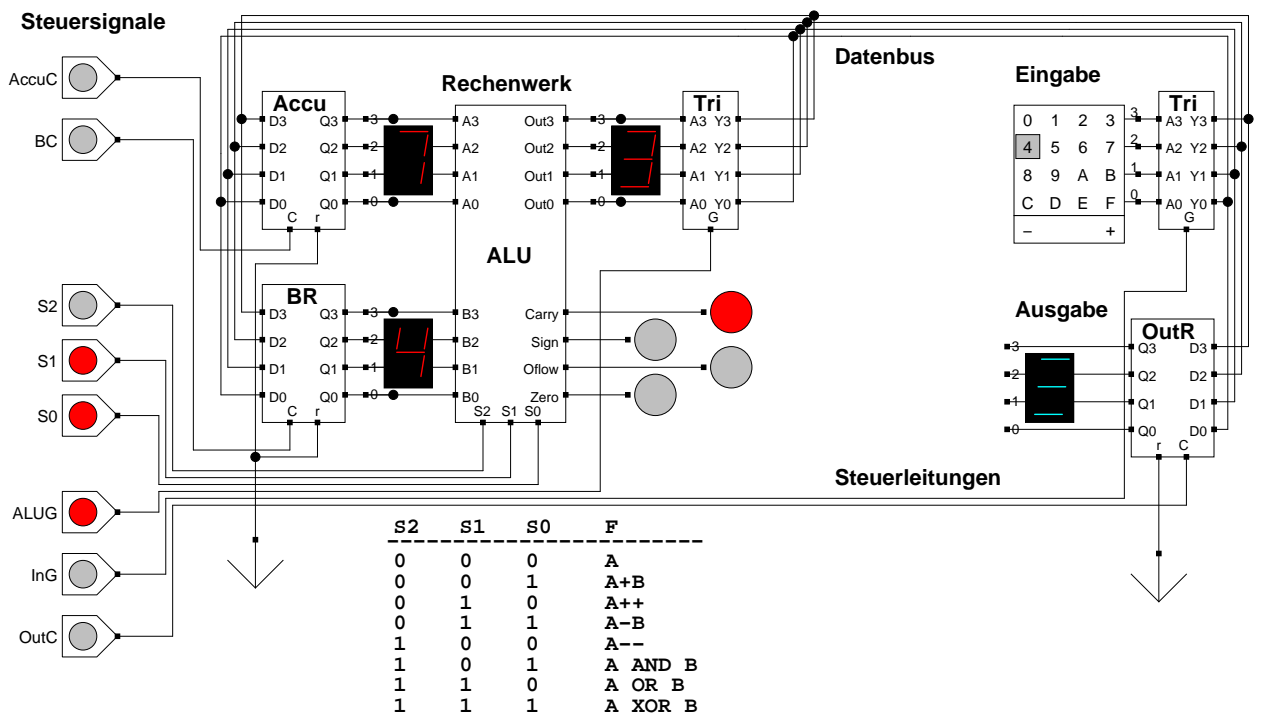


RALU mit Datenbus

► Accu

- Ergebnis einer Operation muss ebenfalls zwischengespeichert werden
- Möglichkeit: Ergebnisregister am Ausgang der ALU
- Alternative: Ergebnis landet immer im RegA
- Umbenennung RegA in **Accu** (Accumulator: Sammler)

► Hades: 8_ralubus/ralu.hds

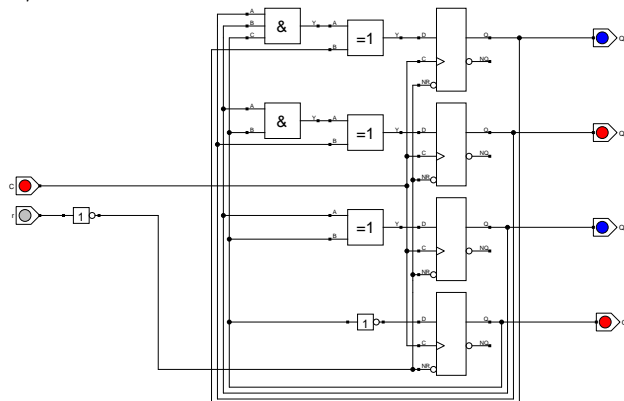


RALU mit Datenbus

- ▶ Beispiel: Berechne 7-4
- ▶ Vorgehensweise:
 - 7 in Accu laden
 - 4 in Reg B laden
 - Ergebnis in Accu laden
- ▶ entspricht 2 Assemblerbefehlen: IN, SUB
- ▶ IN
 - 1. Schritt
 - $InG = 1$ (Datum von Eingabeeinheit (EE) auf den Bus)
 - 2. Schritt
 - $InG = 1$
 - $AccuC = 1$ (Datum in Accu übernehmen)
 - 3. Schritt
 - alle Steuerleitungen auf null
- ▶ SUB
 - 1. Schritt
 - $InG = 1$ (Datum von EE auf den Bus)
 - 2. Schritt
 - $InG = 1$
 - $BC = 1$ (Datum in RegB übernehmen)
 - 3. Schritt
 - $BC = 0$ (Datum wurde gelesen)
 - $InG = 0$ (EE vom Bus)
 - $S2 = 0$
 - $S1 = 1$
 - $S0 = 1$ (ALU auf subtrahieren einstellen)
 - $ALUG = 1$ (Ergebniss auf den Bus)
 - 4. Schritt
 - $AccuC = 1$ (Datum in Accu übernehmen)
 - 5. Schritt
 - alle Steuerleitungen auf null

RALU mit Leitwerk

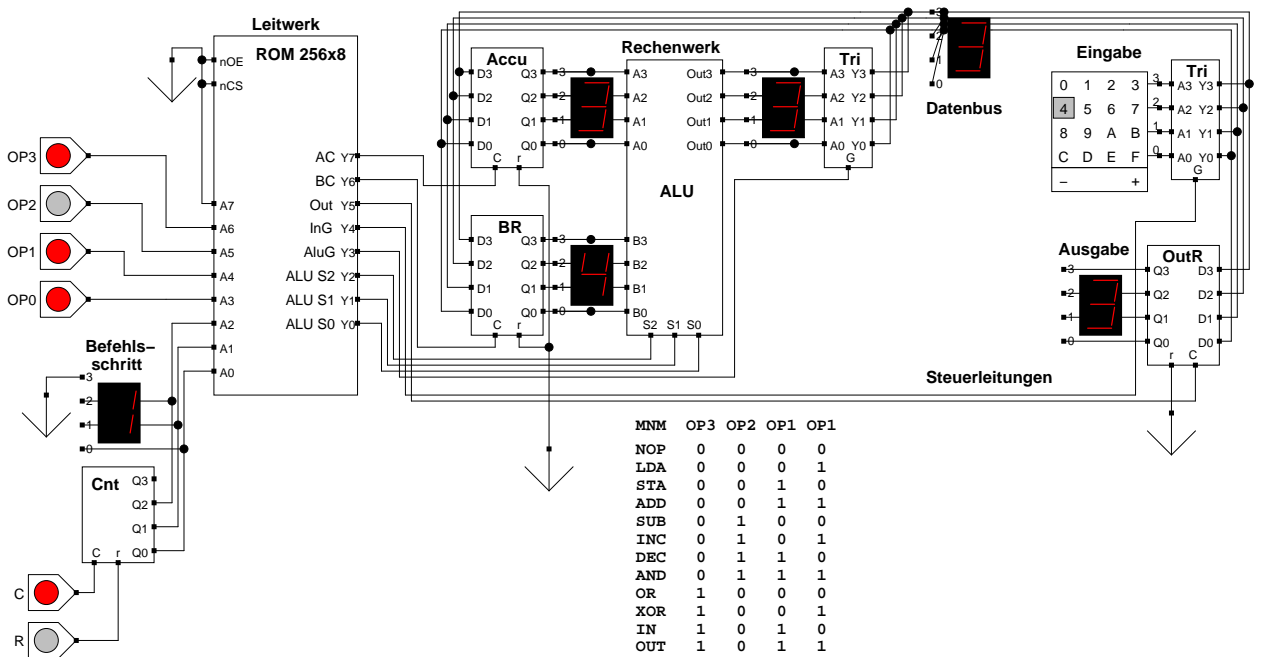
- ▶ Prozessor (DIN 44300)
 - Funktionseinheit innerhalb eines digitalen Rechensystems, die Rechenwerk und Leitwerk umfasst
- ▶ Rechenwerk (ALU)
 - Funktionseinheit innerhalb eines digitalen Rechensystems, die Rechenoperationen ausführt.
- ▶ Leitwerk (oder Steuerwerk)
 - Funktionseinheit innerhalb eines digitalen Rechensystems, die
 - die Reihenfolge steuert, in der die Befehle eines Programms ausgeführt werden
 - diese Befehle entschlüsselt und dabei ggf. modifiziert
 - die für ihre Ausführung erforderlichen Signale abgibt
- ▶ Ziel: Schritte automatisieren
 - Microprogramm (EPROM)
 - Microprogramm soll autonom laufen
 - für die Microschritte wird ein Zähler benötigt
 - für den Zähler wird ein Takt benötigt
- ▶ Zähler
 - innerhalb des Microprogramms realisierbar
 - externer Zähler möglich
 - Vereinbarung: 8 Microschritte pro Befehl
- ▶ Hades: counter/counter.hds



- ▶ 4Bit-Zähler: TTL 74161

RALU mit Leitwerk

- ▶ Code eines Assemblerbefehls (OpCode) ist Startadresse ($A_3 - A_6$) des Microprogramms
- ▶ Hades: 9_raluleit/ralu.hds
 - ROM: leithigh.rom



- ▶ Assembler Befehle

Mnemo	Binär	Hex	Funktion
NOP	0 0 0 0	0x00	Nix
LDA	0 0 0 1	0x01	(reserviert: aus RAM lesen)
STA	0 0 1 0	0x02	(reserviert: ins RAM schreiben)
ADD	0 0 1 1	0x03	Addiert Accu mit RegB, Ergebniss in Accu
SUB	0 1 0 0	0x04	Subtrahiert Regb vom Accu, Ergebniss in Accu
INC	0 1 0 1	0x05	Incrementiert Accu
DEC	0 1 1 0	0x06	Decrementiert Accu
AND	0 1 1 1	0x07	Bitweise AND Verknüpfung von Accu und RegB
OR	1 0 0 0	0x08	Bitweise OR Verknüpfung von Accu und RegB
XOR	1 0 0 1	0x09	Bitweise XOR Verknüpfung von Accu und RegB
IN	1 0 1 0	0x0A	Liest von Eingabeeinheit in Accu
OUT	1 0 1 1	0x0B	Schreibt Accuinhalt in Ausgabeeinheit

- ▶ 4 Weitere Befehle möglich

Microprogrammierung des Leitwerk

Codierung:

ALU-Funktion			
AS2	AS1	AS0	F
0	0	0	A
0	0	1	A+B
0	1	0	A++
0	1	1	A-B
1	0	0	A--
1	0	1	AND
1	1	0	OR
1	1	1	XOR

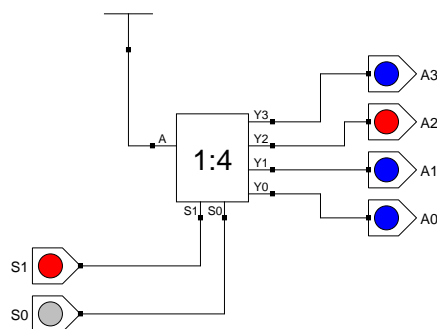
MNM	OPC	AC	BC	OUT	ING	ALG	AS2	AS1	AS0	ADR	HEX	Funktion
NOP	0000	0	0	0	0	0	0	0	0	00	00	NIX
LDA	0001	1	0	0	0	0	0	0	0	08	90	Accu uebernimmt
STA	0010	0	0	0	0	1	0	0	0	10	18	IR auf Datenbus
ADD	0011	0	0	0	1	0	0	0	0	18	10	Eingabeeinheit auf Datenbus
		0	1	0	1	0	0	0	1	19	51	RegB uebernimmt; ALU A+B
		0	0	0	0	1	0	0	1	1A	09	Ergebniss auf Bus
		1	0	0	0	1	0	0	1	1B	49	Accu uebernimmt
SUB	0100	0	0	0	1	0	0	0	0	20	10	Eingabeeinheit auf Datenbus
		0	1	0	1	0	0	1	1	21	53	RegB uebernimmt; ALU A-B
		0	0	0	0	1	0	1	1	22	0B	Ergebniss auf Bus
		1	0	0	0	1	0	1	1	23	8B	Accu uebernimmt
INC	0101	0	0	0	0	1	0	1	0	28	0A	ALU: A++
		1	0	0	0	1	0	1	0	29	8A	Accu=A++
DEC	0110	0	0	0	0	1	1	0	0	30	0C	ALU: A--
		1	0	0	0	1	1	0	0	31	8C	Accu=A--
AND	0111	0	0	0	1	0	0	0	0	38	10	Eingabeeinheit auf Datenbus
		0	1	0	1	0	1	0	1	39	55	RegB uebernimmt; ALU AND
		0	0	0	0	1	1	0	1	3A	0D	Ergebniss auf Bus
		1	0	0	0	1	1	0	1	3B	8D	Accu uebernimmt
OR	1000	0	0	0	1	0	0	0	0	40	10	Eingabeeinheit auf Datenbus
		0	1	0	1	0	1	1	0	41	56	RegB uebernimmt; ALU OR
		0	0	0	0	1	1	1	0	42	0E	Ergebniss auf Bus
		1	0	0	0	1	1	1	0	43	8E	Accu uebernimmt
XOR	1001	0	0	0	1	0	0	0	0	48	10	Eingabeeinheit auf Datenbus
		0	1	0	1	0	1	1	1	49	57	RegB uebernimmt; ALU XOR
		0	0	0	0	1	1	1	1	4A	0F	Ergebniss auf Bus
		1	0	0	0	1	1	1	1	4B	8F	Accu uebernimmt
IN	1010	0	0	0	1	0	0	0	0	50	10	Eingabeeinheit auf Bus
		1	0	0	1	0	0	0	0	51	90	Accu liest Bus
OUT	1011	0	0	0	0	1	0	0	0	58	08	ACCU auf Datenbus
		0	0	1	0	1	0	0	0	59	28	Ausgabe liest Bus

RALU mit RAM

- ▶ Erweiterung: Hauptspeicher
- ▶ Lieferant für Befehle *und* Daten
 - Neu: Instruction-Register IR
 - Befehle über Datenbus ins IR
 - neue Steuerleitung IRC
- ▶ Schreib- und Lesezugriff auf Datenbus
- ▶ 3 schreibende Einheiten → 3 Steuerleitungen ??
 - RAM
 - ALU
 - Eingabeeinheit
- ▶ Codierung durch ein einfaches Schaltnetz

DBusW1	DBusW0	ALU	IN	RAM
0	0	0	0	0
0	1	1	0	0
1	0	0	1	0
1	1	0	0	1

- DBusW1 und DBusW0 ersetzen InG und AluC
- ▶ Einfacher 2Bit Adressdecoder (realisiert mit Demultiplexer)



RALU mit RAM

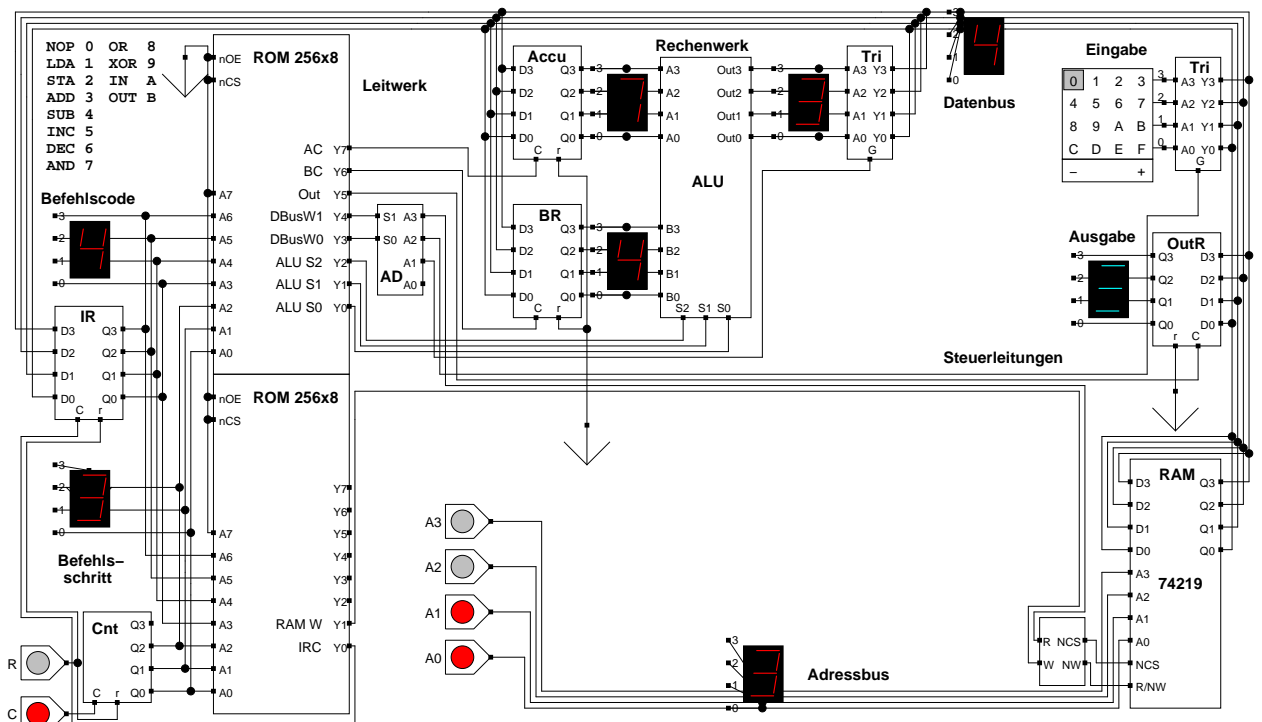
- ▶ kleine Logik zur Umsetzung der low-aktiven Steuereingänge des Rams auf high-aktive Signale:

R	W	NCS	R/NW	Funktion
0	0	1	x	Abgekoppelt
0	1	0	0	RAM beschreiben
1	0	0	1	RAM auslesen
1	1	x	x	–

$$R/NW = \overline{W} = \overline{W \vee W}$$

$$NCS = \overline{R \vee W}$$

- neue Steuerleitung RAM $W \rightarrow W$
- ▶ 16x4 Bit RAM: TTL 74219
- ▶ Hades: 10_raluram/ralu.hds
 - oberes ROM: leithigh.rom
 - unteres ROM: leitlow.rom



Leitwerk: RALU mit RAM

Codierung: Datenbus Schreiben

DW1 DW0 | F

```

0 0 | -
0 1 | ALU
1 0 | Input
1 1 | RAM

```

ALU-Funktion

AS2 AS1 AS0 | F

```

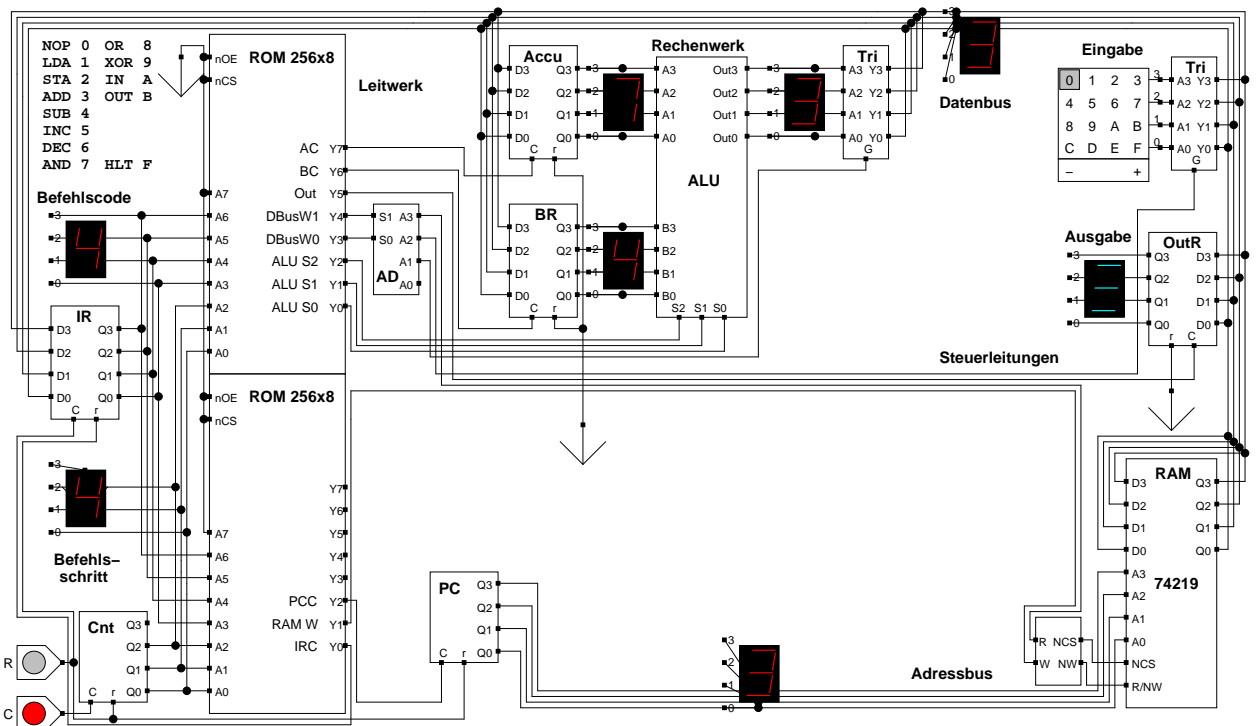
0 0 0 | A
0 0 1 | A+B
0 1 0 | A++
0 1 1 | A-B
1 0 0 | A--
1 0 1 | AND
1 1 0 | OR
1 1 1 | XOR

```

MMN	OPC	AC	BC	OUT	DW1	DW0	AS2	AS1	AS0	RMW	IRC	ADR	HEX	Funktion
NOP	0000	0	0	0	1	1	0	0	0	0	0	00	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	1	01	18 01	IR uebernimmt
LDA	0001	0	0	0	1	1	0	0	0	0	0	08	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	1	09	18 01	IR uebernimmt
		0	0	0	1	1	0	0	0	0	0	0A	18 00	Datum auf Datenbus
		1	0	0	1	1	0	0	0	0	0	0B	98 00	Accu uebernimmt
STA	001	0	0	0	1	1	0	0	0	0	0	10	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	1	11	18 01	IR uebernimmt
		0	0	0	1	1	0	0	0	0	0	12	18 00	Datum auf Datenbus
		0	0	0	0	1	0	0	0	0	1	13	08 02	RAM uebernimmt
ADD	0011	0	0	0	1	1	0	0	0	0	0	18	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	1	19	18 01	IR uebernimmt
		0	0	0	1	1	0	0	0	0	0	1A	18 00	Datum auf Datenbus
		0	1	0	1	1	0	0	1	0	0	1B	59 00	RegB uebernimmt; ALU A+B
		0	0	0	0	1	0	0	1	0	0	1C	09 00	Ergebniss auf Bus
		1	0	0	0	1	0	0	1	0	0	1D	89 00	Accu uebernimmt
SUB siehe ADD														
INC	0101	0	0	0	1	1	0	0	0	0	0	28	18 00	Befehl auf Bus
		0	0	0	1	1	0	1	0	0	1	29	1A 01	IR uebernimmt
		0	0	0	0	1	0	1	0	0	0	2A	0A 00	ALU: A++
		1	0	0	0	1	0	1	0	0	0	2B	8A 00	Accu=A++
DEC	0110	0	0	0	1	1	0	0	0	0	0	30	18 00	Befehl auf Bus
		0	0	0	1	1	1	0	0	0	1	31	1C 01	IR uebernimmt
		0	0	0	0	1	1	0	0	0	0	32	0C 00	ALU: A--
		1	0	0	0	1	1	0	0	0	0	33	8C 00	Accu=A--
AND OR XOR siehe ADD														
IN	1010	0	0	0	1	1	0	0	0	0	0	50	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	1	51	18 01	IR uebernimmt
		0	0	0	1	0	0	0	0	0	0	52	10 00	Eingabeeinheit auf Bus
		1	0	0	1	0	0	0	0	0	0	53	90 00	Accu liest Bus
OUT	1011	0	0	0	1	1	0	0	0	0	0	58	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	1	59	18 01	IR uebernimmt
		0	0	0	0	1	0	0	0	0	0	5A	08 00	ACCU auf Datenbus
		0	0	1	0	1	0	0	0	0	0	5B	28 00	Ausgabe liest Bus

CPU mit PC

- ▶ Erweiterung: Program-Counter (PC)
- ▶ PC zeigt auf nächsten Befehl/Operand
- ▶ 4-Bit Counter
- ▶ neue Steuerleitung PCC
- ▶ neuer Befehl HLT
 - schaltet PC nicht weiter
- ▶ Hades: 11_minicpu/cpu.hds



Leitwerk: CPU mit PC

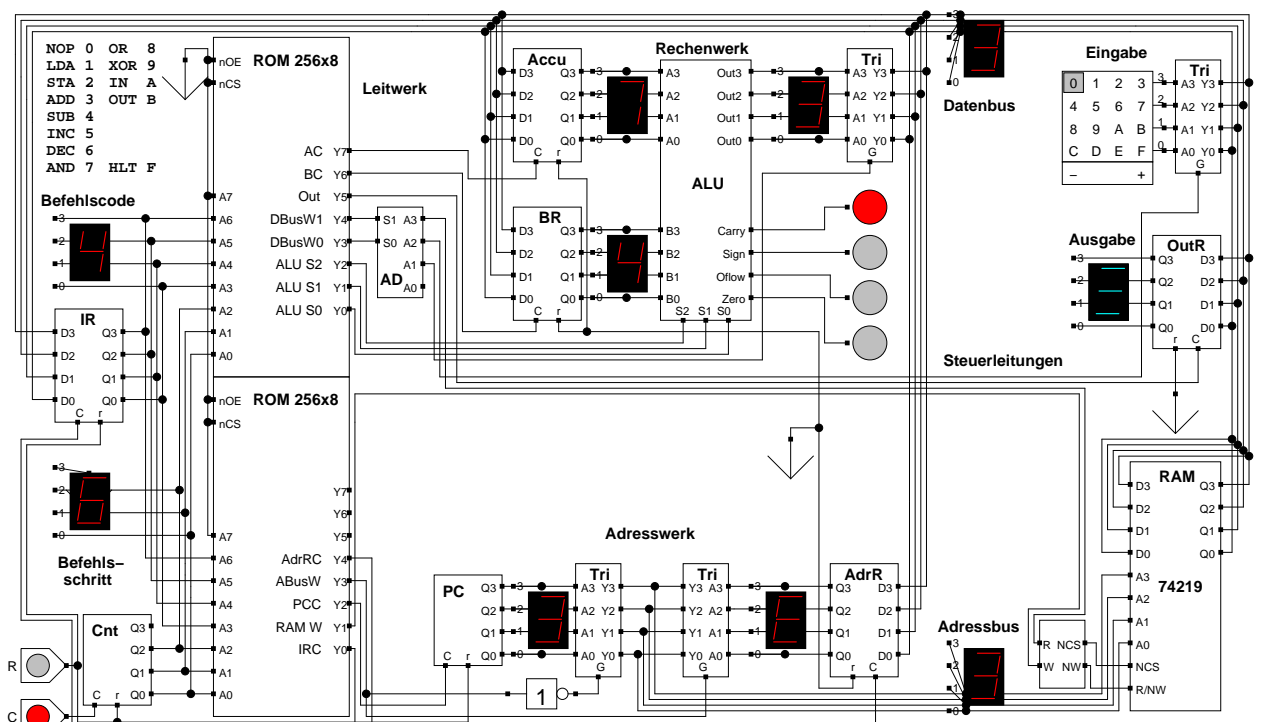
MMN OPC	AC	BC	OUT	DW1	DW0	AS2	AS1	AS0	PCC	RMW	IRC	ADR	HEX	Funktion
NOP 0000	0	0	0	1	1	0	0	0	0	0	0	00	18 00	Befehl auf Bus
	0	0	0	1	1	0	0	0	0	0	1	01	18 01	IR uebernimmt
	0	0	0	0	0	0	0	0	1	0	0	02	00 04	PC erhoehen
LDA 0001	0	0	0	1	1	0	0	0	0	0	0	08	18 00	Befehl auf Bus
	0	0	0	1	1	0	0	0	0	0	1	09	18 01	IR uebernimmt
	0	0	0	1	1	0	0	0	1	0	0	0A	18 04	PC++; Datum auf Datenbus
	1	0	0	1	1	0	0	0	0	0	0	0B	98 00	Accu uebernimmt
	0	0	0	0	0	0	0	0	1	0	0	0C	00 04	PC++
STA 001	0	0	0	1	1	0	0	0	0	0	0	10	18 00	Befehl auf Bus
	0	0	0	1	1	0	0	0	0	0	1	11	18 01	IR uebernimmt
	0	0	0	1	1	0	0	0	1	0	0	12	18 04	PC++; Datum auf Datenbus
	0	0	0	0	1	0	0	0	0	1	0	13	08 02	RAM uebernimmt
	0	0	0	0	0	0	0	0	1	0	0	14	00 04	PC++
ADD 0011	0	0	0	1	1	0	0	0	0	0	0	18	18 00	Befehl auf Bus
	0	0	0	1	1	0	0	0	0	0	1	19	18 01	IR uebernimmt
	0	0	0	1	1	0	0	0	1	0	0	1A	18 04	PC++; Datum auf Datenbus
	0	1	0	1	1	0	0	1	0	0	0	1B	59 00	RegB uebernimmt; ALU A+B
	0	0	0	0	1	0	0	1	0	0	0	1C	09 00	Ergebniss auf Bus
	1	0	0	0	1	0	0	1	0	0	0	1D	89 00	Accu uebernimmt
	0	0	0	0	0	0	0	0	1	0	0	1E	00 04	PC++
SUB siehe ADD														
INC 0101	0	0	0	1	1	0	0	0	0	0	0	28	18 00	Befehl auf Bus
	0	0	0	1	1	0	1	0	0	0	1	29	1A 01	IR uebernimmt
	0	0	0	0	1	0	1	0	0	0	0	2A	0A 00	ALU: A++
	1	0	0	0	1	0	1	0	1	0	0	2B	8A 04	Accu=A++; PC++
DEC 0110	0	0	0	1	1	0	0	0	0	0	0	30	18 00	Befehl auf Bus
	0	0	0	1	1	1	0	0	0	0	1	31	1C 01	IR uebernimmt
	0	0	0	0	1	1	0	0	0	0	0	32	0C 00	ALU: A--
	1	0	0	0	1	1	0	0	1	0	0	33	8C 04	Accu=A--; PC++
AND OR XOR siehe ADD														
IN 1010	0	0	0	1	1	0	0	0	0	0	0	50	18 00	Befehl auf Bus
	0	0	0	1	1	0	0	0	0	0	1	51	18 01	IR uebernimmt
	0	0	0	1	0	0	0	0	0	0	0	52	10 00	Eingabeeinheit auf Bus
	1	0	0	1	0	0	0	0	1	0	0	53	90 04	Accu liest Bus; PC++
OUT 1011	0	0	0	1	1	0	0	0	0	0	0	58	18 00	Befehl auf Bus
	0	0	0	1	1	0	0	0	0	0	1	59	18 01	IR uebernimmt
	0	0	0	0	1	0	0	0	0	0	0	5A	08 00	ACCU auf Datenbus
	0	0	1	0	1	0	0	0	1	0	0	5B	28 04	Ausgabe liest Bus; PC++
HLT 1111	0	0	0	1	1	0	0	0	0	0	0	78	18 00	Befehl auf Bus
	0	0	0	1	1	0	0	0	0	0	1	79	18 01	IR uebernimmt

direkte Adressierung

- ▶ bisher: Operand folgt Befehl unmittelbar
 - unmittelbare Adressierung
- ▶ Problem: STA
 - Im Program "Platz" lassen für Ergebniss
 - aber wie später darauf zugreifen??
- ▶ Lösung: direkte Adressierung
 - *Adresse* des Operanden folgt Befehl

Adressierung	Beispiel	und Funktion
unmittelbar	LDA 0x0F	Accu wird mit der Zahl F geladen
direkt	LDA 0x0F	Accu wird mit Inhalt der Speicherstelle F geladen

- ▶ Neu: Adresregister (AdrR) mit Zugriff auf Adressbus (TriState)
 - neue Steuerleitungen ABusW und AdrRC
- ▶ Hades: 12_adressierung/cpu.hds

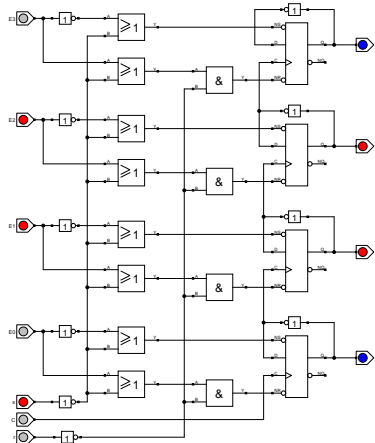


Leitwerk: direkte Adressierung

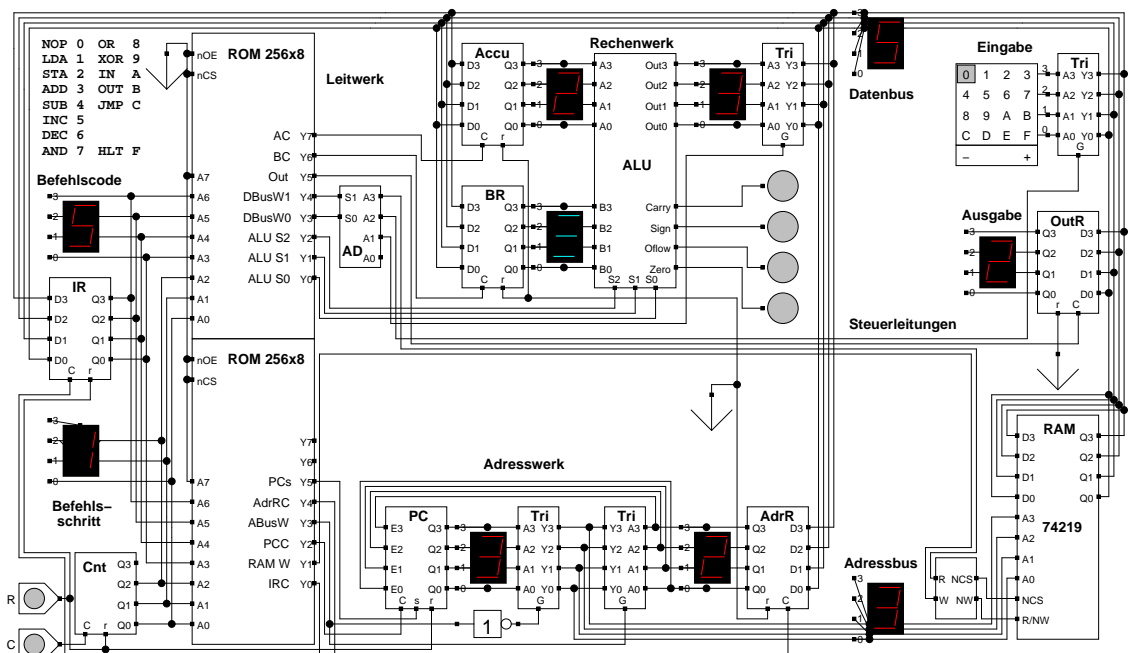
MMN OPC	AC	BC	OUT	DW1	DWO	AS2	AS1	AS0	ARC	ARW	PCC	RMW	IRC	ADR	HEX	Funktion
NOP 0000	0	0	0	1	1	0	0	0	0	0	0	0	0	00	18 00	Befehl auf Bus
	0	0	0	1	1	0	0	0	0	0	0	0	1	01	18 01	IR uebernimmt
	0	0	0	0	0	0	0	0	0	0	1	0	0	02	00 04	PC erhoehen
LDA 0001	0	0	0	1	1	0	0	0	0	0	0	0	0	08	18 00	Befehl auf Bus
	0	0	0	1	1	0	0	0	0	0	0	0	1	09	18 01	IR uebernimmt
	0	0	0	1	1	0	0	0	0	0	1	0	0	0A	18 04	PC++; Adresse auf Datenbu
	0	0	0	1	1	0	0	0	1	0	0	0	0	0B	18 10	AdrR uebernimmt
	0	0	0	1	1	0	0	0	0	1	1	0	0	0C	18 08	Datum auf Datenbus
1	0	0	1	1	0	0	0	0	0	1	1	0	0	0D	98 0C	Accu uebernimmt;PC++
STA 001	0	0	0	1	1	0	0	0	0	0	0	0	0	10	18 00	Befehl auf Bus
	0	0	0	1	1	0	0	0	0	0	0	0	1	11	18 01	IR uebernimmt
	0	0	0	1	1	0	0	0	0	0	1	0	0	12	18 04	PC++; Adresse auf Datenbu
	0	0	0	1	1	0	0	0	1	0	0	0	0	13	18 10	AdrR uebernimmt
	0	0	0	0	1	0	0	0	0	1	0	0	0	14	08 08	Alu auf Datenbus
0	0	0	0	1	0	0	0	0	0	1	1	0	15	08 0E	RAM uebernimmt; PC++	
ADD 0011	0	0	0	1	1	0	0	0	0	0	0	0	0	18	18 00	Befehl auf Bus
	0	0	0	1	1	0	0	0	0	0	0	0	1	19	18 01	IR uebernimmt
	0	0	0	1	1	0	0	0	0	0	1	0	0	1A	18 04	PC++; Adresse auf Datenbu
	0	0	0	1	1	0	0	0	1	0	0	0	0	1B	18 10	AdrR uebernimmt
	0	0	0	1	1	0	0	0	0	1	0	0	0	1C	18 08	Datum auf Datenbus
	0	1	0	1	1	0	0	1	0	1	0	0	0	1D	59 08	RegB uebernimmt; ALU A+B
	0	0	0	0	1	0	0	1	0	0	0	0	0	1E	09 00	Ergebniss auf Bus
1	0	0	0	1	0	0	1	0	0	1	0	0	1F	89 04	Accu uebernimmt; PC++	
SUB siehe ADD																
INC 0101	0	0	0	1	1	0	0	0	0	0	0	0	0	28	18 00	Befehl auf Bus
	0	0	0	1	1	0	1	0	0	0	0	0	1	29	1A 01	IR uebernimmt
	0	0	0	0	1	0	1	0	0	0	0	0	0	2A	0A 00	ALU: A++
	1	0	0	0	1	0	1	0	0	0	1	0	0	2B	8A 04	Accu=A++; PC++
DEC 0110	0	0	0	1	1	0	0	0	0	0	0	0	0	30	18 00	Befehl auf Bus
	0	0	0	1	1	1	0	0	0	0	0	0	1	31	1C 01	IR uebernimmt
	0	0	0	0	1	1	0	0	0	0	0	0	0	32	0C 00	ALU: A--
	1	0	0	0	1	1	0	0	0	0	1	0	0	33	8C 04	Accu=A--; PC++
AND OR XOR siehe ADD																
IN 1010	0	0	0	1	1	0	0	0	0	0	0	0	0	50	18 00	Befehl auf Bus
	0	0	0	1	1	0	0	0	0	0	0	0	1	51	18 01	IR uebernimmt
	0	0	0	1	0	0	0	0	0	0	0	0	0	52	10 00	Eingabeeinheit auf Bus
	1	0	0	1	0	0	0	0	0	0	1	0	0	53	90 04	Accu liest Bus; PC++
OUT 1011	0	0	0	1	1	0	0	0	0	0	0	0	0	58	18 00	Befehl auf Bus
	0	0	0	1	1	0	0	0	0	0	0	0	1	59	18 01	IR uebernimmt
	0	0	0	0	1	0	0	0	0	0	0	0	0	5A	08 00	ACCU auf Datenbus
	0	0	1	0	1	0	0	0	0	0	1	0	0	5B	28 04	Ausgabe liest Bus; PC++
HLT 1111	0	0	0	1	1	0	0	0	0	0	0	0	0	78	18 00	Befehl auf Bus
	0	0	0	1	1	0	0	0	0	0	0	0	1	79	18 01	IR uebernimmt

unbedingter Sprung

- ▶ bisher: lineare Programmierung
- ▶ Erweiterung; unbedingter Sprung (JMP)
 - Setzen des Program-Counter (PC)
- ▶ Programmierbarer Zähler: TTL 74161



- ▶ Ablauf:
 - Sprungadresse ins Adressregister
 - PC übernimmt Sprungadresse
- ▶ neue Steuerleitung: PCs
- ▶ Hades: 13_jmp/cpu.hds

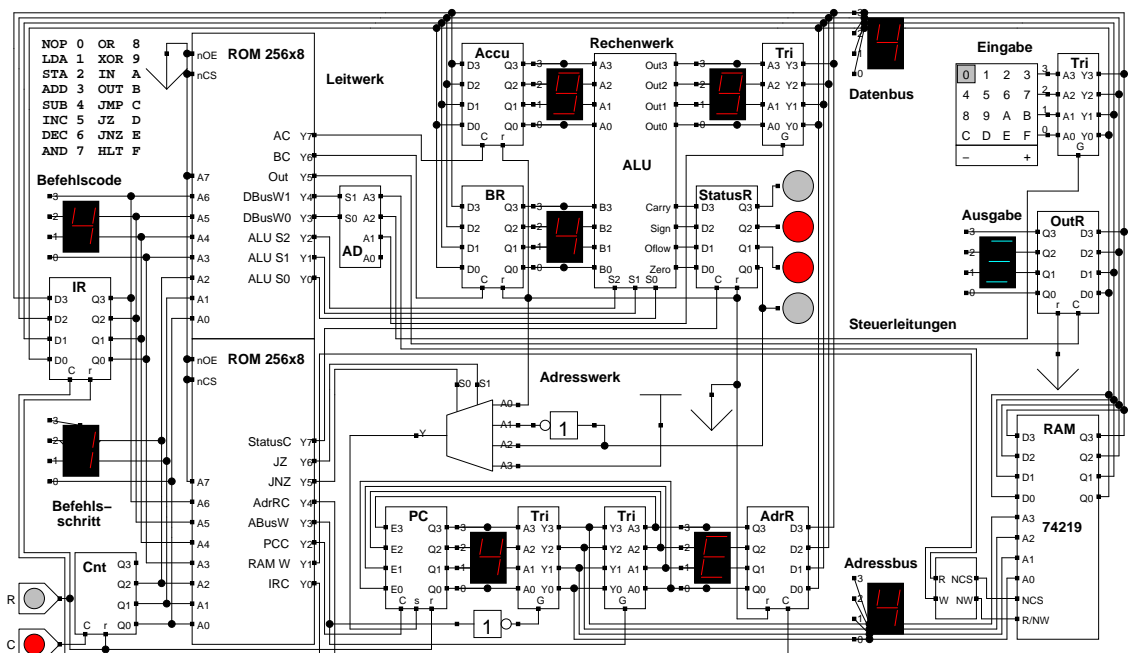


Leitwerk: unbedingter Sprung

MMN	OPC	AC	BC	OUT	DW1	DW0	AS2	AS1	AS0	PCs	ARC	ARW	PCC	RMW	IRC	ADR	HEX	Funktion
NOP	0000	0	0	0	1	1	0	0	0	0	0	0	0	0	0	00	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	0	0	0	0	1	01	18 01	IR uebernimmt
		0	0	0	0	0	0	0	0	0	0	0	1	0	0	02	00 04	PC erhoehen
LDA	0001	0	0	0	1	1	0	0	0	0	0	0	0	0	0	08	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	0	0	0	0	1	09	18 01	IR uebernimmt
		0	0	0	1	1	0	0	0	0	0	0	1	0	0	0A	18 04	PC++; Adresse auf Dat
		0	0	0	1	1	0	0	0	0	0	1	0	0	0	0B	18 10	AdrR uebernimmt
		0	0	0	1	1	0	0	0	0	0	0	1	0	0	0C	18 08	Datum auf Datenbus
		1	0	0	1	1	0	0	0	0	0	0	1	1	0	0D	98 0C	Accu uebernimmt;PC++
STA	001	0	0	0	1	1	0	0	0	0	0	0	0	0	0	10	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	0	0	0	0	1	11	18 01	IR uebernimmt
		0	0	0	1	1	0	0	0	0	0	0	1	0	0	12	18 04	PC++; Adresse auf Dat
		0	0	0	1	1	0	0	0	0	0	1	0	0	0	13	18 10	AdrR uebernimmt
		0	0	0	0	1	0	0	0	0	0	0	1	0	0	14	08 08	Alu auf Datenbus
		0	0	0	0	1	0	0	0	0	0	0	1	1	1	15	08 0E	RAM uebernimmt; PC++
ADD	0011	0	0	0	1	1	0	0	0	0	0	0	0	0	0	18	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	0	0	0	0	1	19	18 01	IR uebernimmt
		0	0	0	1	1	0	0	0	0	0	0	1	0	0	1A	18 04	PC++; Adresse auf Dat
		0	0	0	1	1	0	0	0	0	0	1	0	0	0	1B	18 10	AdrR uebernimmt
		0	0	0	1	1	0	0	0	0	0	0	1	0	0	1C	18 08	Datum auf Datenbus
		0	1	0	1	1	0	0	1	0	0	1	0	0	0	1D	59 08	RegB uebernimmt; ALU
		0	0	0	0	1	0	0	1	0	0	0	0	0	0	1E	09 00	Ergebniss auf Bus
1	0	0	0	1	0	0	1	0	0	0	1	0	0	1F	89 04	Accu uebernimmt; PC++		
SUB siehe ADD																		
INC	0101	0	0	0	1	1	0	0	0	0	0	0	0	0	0	28	18 00	Befehl auf Bus
		0	0	0	1	1	0	1	0	0	0	0	0	0	1	29	1A 01	IR uebernimmt
		0	0	0	0	1	0	1	0	0	0	0	0	0	0	2A	0A 00	ALU: A++
		1	0	0	0	1	0	1	0	0	0	0	1	0	0	2B	8A 04	Accu=A++; PC++
DEC siehe INC																		
AND OR XOR siehe ADD																		
IN	1010	0	0	0	1	1	0	0	0	0	0	0	0	0	0	50	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	0	0	0	0	1	51	18 01	IR uebernimmt
		0	0	0	1	0	0	0	0	0	0	0	0	0	0	52	10 00	Eingabeeinheit auf Bu
		1	0	0	1	0	0	0	0	0	0	0	1	0	0	53	90 04	Accu liest Bus; PC++
OUT	1011	0	0	0	1	1	0	0	0	0	0	0	0	0	0	58	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	0	0	0	0	1	59	18 01	IR uebernimmt
		0	0	0	0	1	0	0	0	0	0	0	0	0	0	5A	08 00	ACCU auf Datenbus
		0	0	1	0	1	0	0	0	0	0	0	1	0	0	5B	28 04	Ausgabe liest Bus; PC
JMP	1100	0	0	0	1	1	0	0	0	0	0	0	0	0	0	60	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	0	0	0	0	1	61	18 01	IR uebernimmt
		0	0	0	1	1	0	0	0	0	0	0	1	0	0	62	18 04	PC++; Adresse auf Dat
		0	0	0	1	1	0	0	0	0	0	1	0	0	0	63	18 10	AdrR uebernimmt
		0	0	0	0	0	0	0	0	0	1	0	0	0	0	64	00 60	PC uebernimmt
HLT	1111	0	0	0	1	1	0	0	0	0	0	0	0	0	0	78	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	0	0	0	0	1	79	18 01	IR uebernimmt

bedingter Sprung

- ▶ Erweiterung: bedingter Sprung (condition jump)
 - Schleifenabbruch und Verzweigungen
 - ▶ Statusregister
 - sichert den Status der letzten logischen oder arithmetischen Operation
 - ▶ In Abhängigkeit eines Statusbits PC setzen
 - PCs mit Registerausgang des Statusbits "verbinden"
 - Beispiel: Zeroflag Z
 - ▶ Lösung mit Multiplexer
 - vereinigt den unbedingten Sprung
- | JZ | JNZ | PCs | Funktion |
|----|-----|-----------|-----------------------|
| 0 | 0 | 0 | kein Sprung |
| 0 | 1 | \bar{Z} | Sprung falls NOT ZERO |
| 1 | 0 | Z | Sprung falls ZERO |
| 1 | 1 | 1 | unbedingter Sprung |
- ▶ neue Steuerleitungen:
 - StatusC, JNZ, JZ (PCs entfällt)
 - ▶ Hades: 14_jmpcond/cpu.hds



Leitwerk: bedingter Sprung

NOP, LDA, STA, ADD, SUB, INC, DEC, AND, OR, XOR, IN, OUT siehe Leitwerk unbedingter Sprung

MNM	OPC	AC	BC	OUT	DW1	DWO	AS2	AS1	AS0	StC	JZ	JNZ	ARC	ARW	PCC	RMW	IRC	ADR	HEX	Funktion
JMP	1100	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	60	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	61	18 01	IR uebernimmt
		0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	62	18 04	PC++; Adresse a
		0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	63	18 10	AdrR uebernimmt
		0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	64	00 60	PC uebernimmt

MNM	OPC	AC	BC	OUT	DW1	DWO	AS2	AS1	AS0	StC	JZ	JNZ	ARC	ARW	PCC	RMW	IRC	ADR	HEX	Funktion
JZ	1101	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	68	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	69	18 01	IR uebernimmt	
		0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	6A	18 04	PC++; Adresse a
		0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	6B	18 10	AdrR uebernimmt
		0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	6C	00 44	PC uebernimmt o

MNM	OPC	AC	BC	OUT	DW1	DWO	AS2	AS1	AS0	StC	JZ	JNZ	ARC	ARW	PCC	RMW	IRC	ADR	HEX	Funktion
JNZ	1110	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	70	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	71	18 01	IR uebernimmt	
		0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	72	18 04	PC++; Adresse a
		0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	73	18 10	AdrR uebernimmt
		0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	74	00 24	PC uebernimmt o

MNM	OPC	AC	BC	OUT	DW1	DWO	AS2	AS1	AS0	StC	JZ	JNZ	ARC	ARW	PCC	RMW	IRC	ADR	HEX	Funktion
HLT	1111	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	78	18 00	Befehl auf Bus
		0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	79	18 01	IR uebernimmt	

Programmierung der MiniCPU

► Aufgabe:

- Subtrahiere des Speicherinhalt der Speicherstelle E vom Inhalt der Speicherstelle F und speichere das Ergebnis in die Speicherstelle D

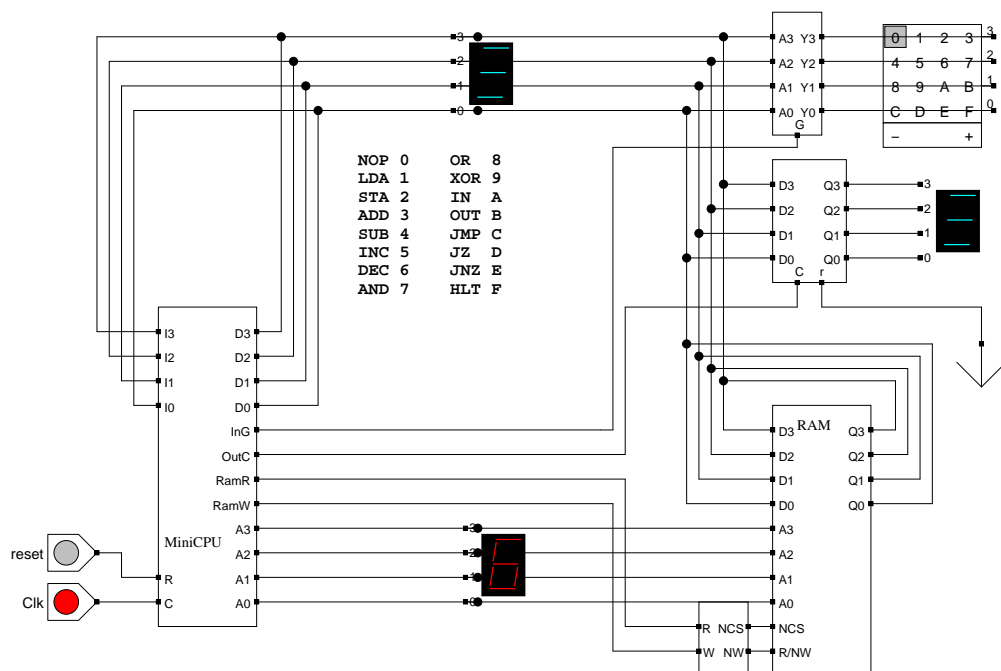
Befehlssatz							
NOP	0	SUB	4	OR	8	JMP	C
LDA	1	INC	5	XOR	9	JZ	D
STA	2	DEC	6	IN	A	JNZ	E
ADD	3	AND	7	OUT	B	HLT	F

0: 1 F ; LDA F : Speicherinhalt von F in Accu
 2: 4 E ; SUB E : Speicherinhalt von E vom Accu subtrahieren
 4: 2 D ; STA D : Ergebniss nach D speichern
 6: F ; HLT : Ende

► Programmierung des RAMs:

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	1	F	4	E	2	D	F							4	7

► Hades: subtraktion/cpu.hds



Programmierung der MiniCPU

► Aufgabe: 4 Bit Inverter

- Entwerfen Sie ein Programm das das Komplement der Eingabe auf der Ausgabe erzeugt

Befehlssatz							
NOP	0	SUB	4	OR	8	JMP	C
LDA	1	INC	5	XOR	9	JZ	D
STA	2	DEC	6	IN	A	JNZ	E
ADD	3	AND	7	OUT	B	HLT	F

```

0:  A      ; IN      : Eingabeport lesen
1:  9  F   ; XOR F   : Mit Inhalt von F antivalent verknuepfen
3:  B      ; OUT     : Ergebniss auf den Ausgabeport
4:  C  0   ; JMP 0   : Endlosschleife

```

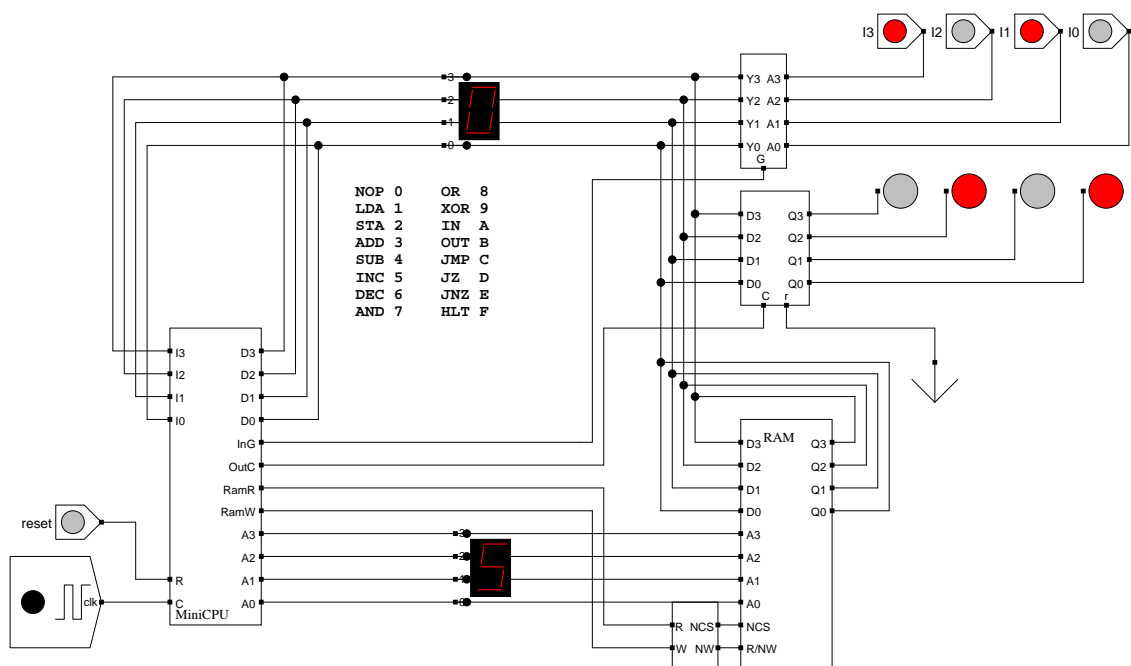
► Programmierung des RAMs:

```

      0 1 2 3 4 5 6 7 8 9 A B C D E F
A  9 F B C 0
      F

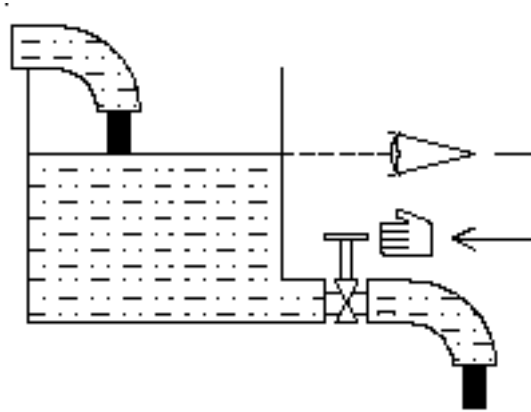
```

► Hades: inverter/cpu.hds

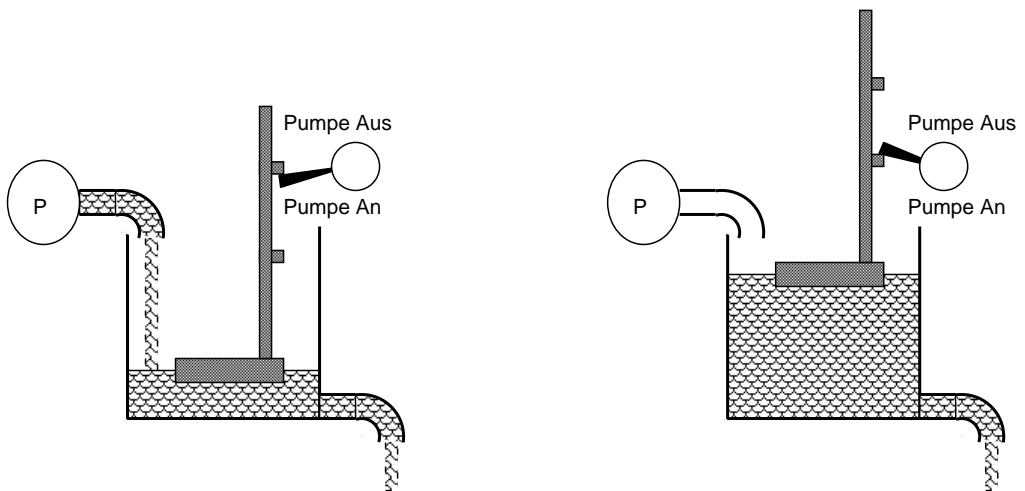


2-Punkt Regler

- ▶ Wasserstandsregelung manuell

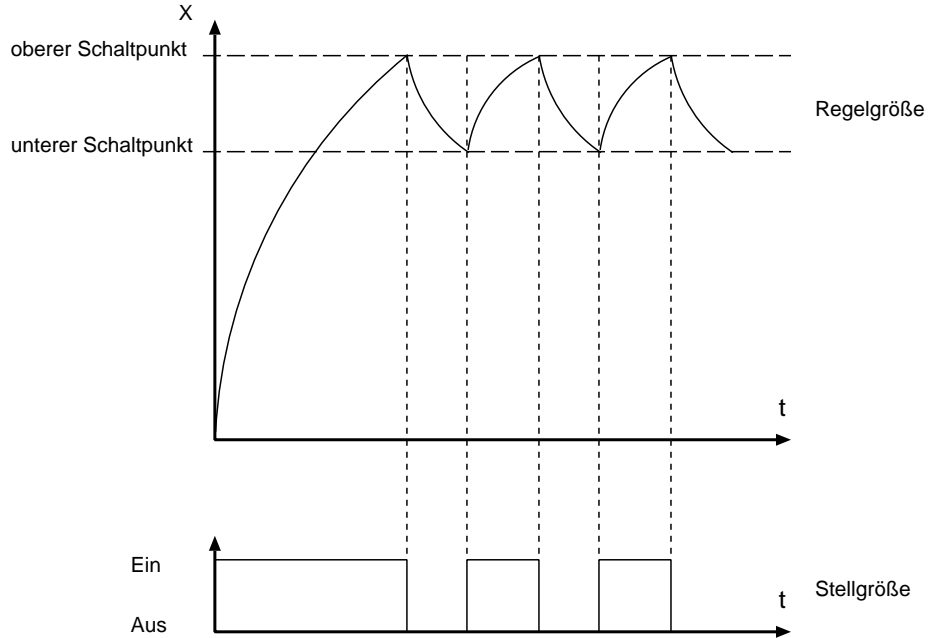


- ▶ Wasserstandsregelung mechanisch

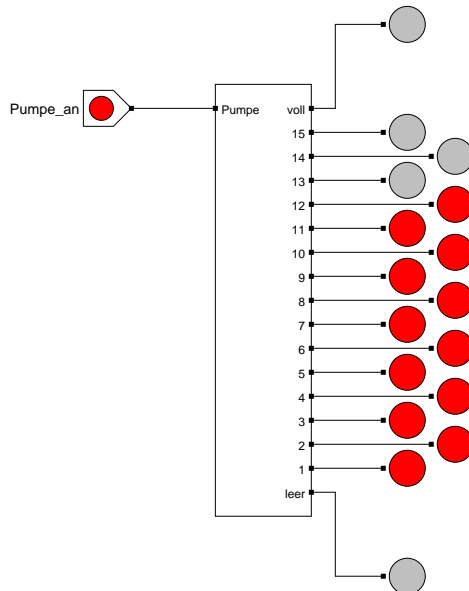


2-Punkt Regler

- ▶ Verhalten einer 2-Punkt Regelung



- ▶ Hades: wassertank/wasserdemo.hds



Programmierung der MiniCPU

► Aufgabe: 2–Punkt Regler

- Entwerfen Sie ein Programm (2–Punkt Regler) zur Wasserstandsregelung

```

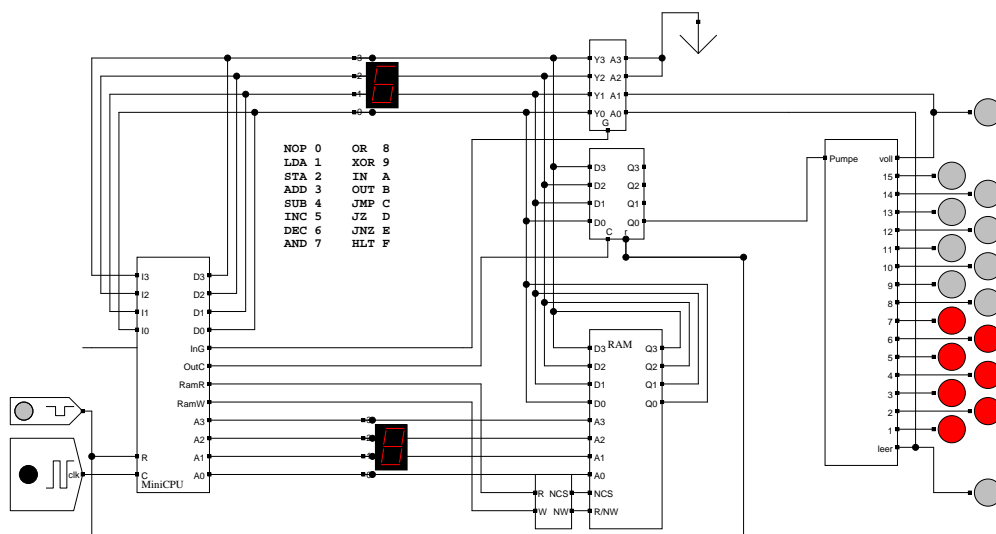
0:  A      ; IN      : Eingabeport lesen
1:  6      ; DEC     : Decrementieren, Check ob In==1
2:  E 0    ; JNZ 0   : Wenn nicht; zurueck nach 0
4:  5      ; INC     : Accu++ damit 1 im Accu steht
5:  B      ; OUT     : Ausgabe, Pumpe einschalten
6:  A      ; IN      : Eingabeport lesen
7:  6      ; DEC     : Zweimal decrementieren
8:  6      ; DEC     : Check ob IN==2
9:  E 6    ; JNZ 6   : Wenn nicht; zurueck nach 6
B:  B      ; OUT     : Ausgabe, im Accu steht 0, Pumpe aus
C:  C 0    ; JMP 0   : und wieder von vorne

```

► Programmierung des RAMs:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A	6	E	0	5	B	A	6	6	E	6	B	C	0			

► Hades: regelung/cpu.hds



Programmierung der MiniCPU

► Aufgabe: Würfel

- Entwerfen Sie ein Programm zur Würfelsteuerung

```

0:  1  F  ; LDA F : Accu mit MEM[f]=6 vorladen
2:  B      ; OUT   : Ausgabe
3:  2  E  ; STA E : Wuerfelstand sichern
5:  A      ; IN    : Eingabeport lesen
6:  6      ; DEC   : decrementieren
7:  D 5   ; JZ   5 : wenn NULL, dann weiter einlesen
9:  1  E  ; LDA E : alten Wuerfelstand einladen
B:  6      ; DEC   : decrementieren
C:  E 2   ; JNZ  2 : wenn nicht NULL, dann zurueck zur 2
E:  C 0   ; JMP  0 : und wieder von vorne

E:  _      : Speicher fuer Wuerfelstand
F:  6      : Startwert fuer den Wuerfel

```

► Problem:

- zuwenig Speicher

► STA und LDA zur Speicherung des Würfelstandes kostenintensiv

- 5 Speicherplätze

► Idee:

- Ausgaberegister als Speicher mitbenutzen
- unteren 3 Bit in die Eingabe rückkoppeln
- oberes Bit für die Steuerung

Programmierung der MiniCPU

- ▶ Aufgabe: Würfel
 - neue Version

```

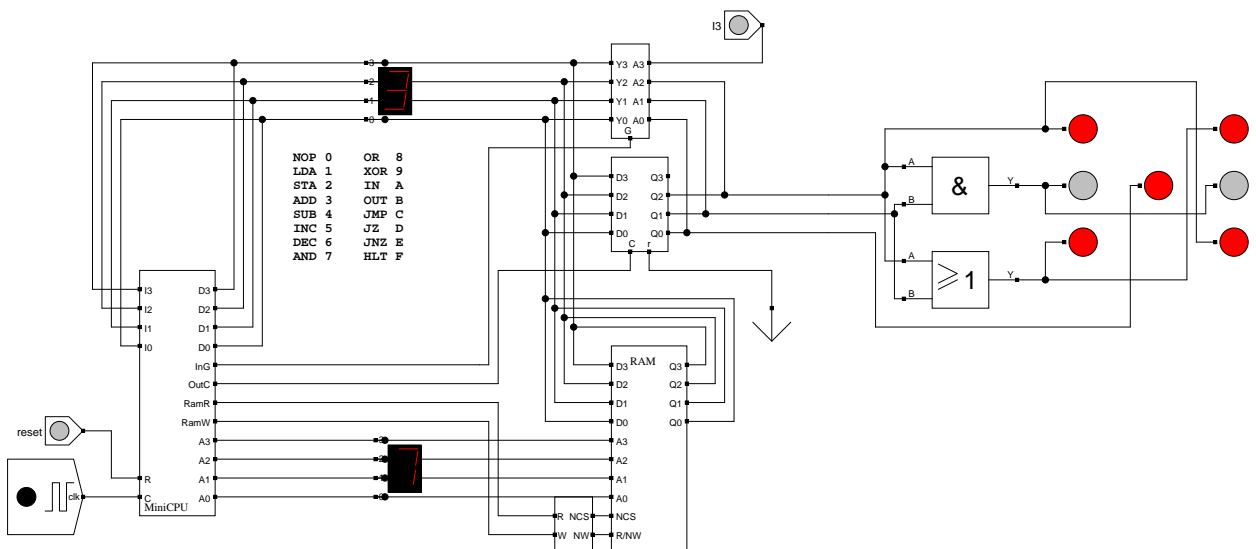
0:  1  F  ; LDA F : Accu mit MEM[f]=6 vorladen
2:  B    ; OUT  : Ausgabe
3:  A    ; IN   : Eingabeport lesen
4:  7  E  ; AND E : mit MEM[E]=1000 UND-Verknuepfen
6:  E  3  ; JNZ 3 : wenn nicht NULL, dann zurueck zur 3
8:  A    ; IN   : Eingabeport lesen, alter Wuerfelstand
9:  6    ; DEC  : decrementieren
A:  E  2  ; JNZ 2 : wenn nicht NULL, dann zurueck zur 2
C:  C  0  ; JMP 0 : und wieder von vorne
E:  8    : Konstante 8=1000 zur Maskierung
F:  6    : Startwert fuer den Wuerfel

```

- ▶ Programmierung des RAMs:

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	F	B	A	7	E	E	3	A	6	E	2	C	0	8	6

- ▶ Hades: wuerfel/cpu.hds



Ausblick

Mikrocontroller und Robotik

- Messtechnik
- TTL Schaltkreisfamilie
- Treiber, Gleichstrom/Schritt/Servomotoren
- Prozessorentwicklung CISC/RISC
- Mikrocontroller
 - Assembler und C
 - I/O
 - Unterprogramme
 - Interrupts
 - DA/AD Wandler
 - serielle Schnittstellen
- Einchipcomputer
 - VGA
 - Audio
 - Ethernet
- Spielekonsole