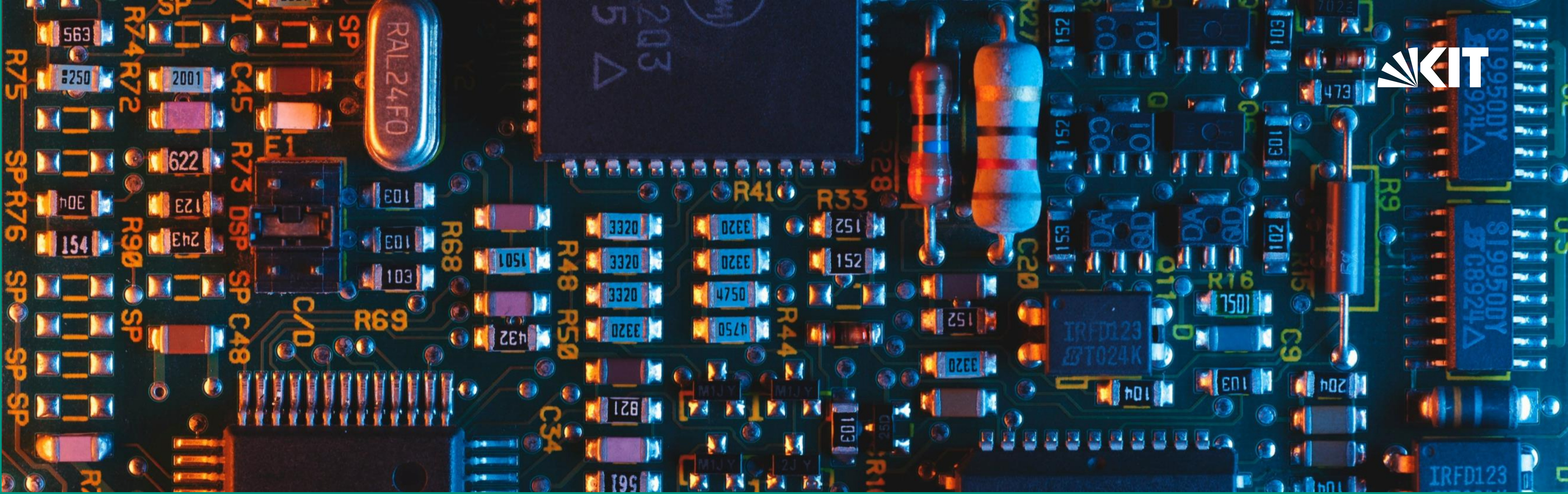


Digitaltechnik
Kapitel 7a: Schaltnetze und Schaltwerke

Prof. Jürgen Becker

Institut für Technik der Informationsverarbeitung (ITIV)

08.01.2026

Praktische Anwendung

K. 7.4

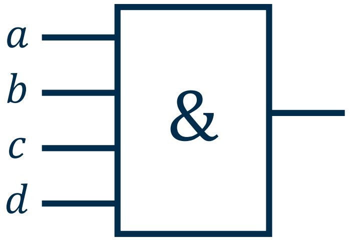
Definition (angelehnt an DIN IEC 748)

Ein **Schaltnetz** ist eine Digitalschaltung, in der es für jede mögliche Kombination von digitalen Signalen an den Eingängen eine – und nur eine – Kombination von digitalen Signalen gibt.

- Notwendig: für jeden Operatortyp eine **passende technische Realisierung**
- Schaltglieder (**Gatter**) für Konjunktion, Disjunktion und Negation

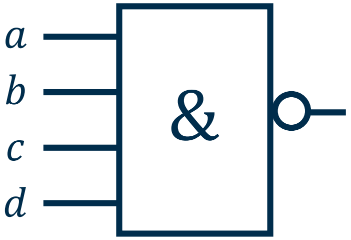
Schaltzeichen nach der Norm DIN 40900

AND-Gatter:



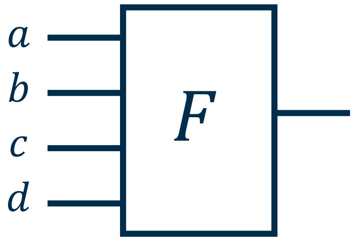
$y = a \,\&\, b \,\&\, c \,\&\, d$

NAND-Gatter:



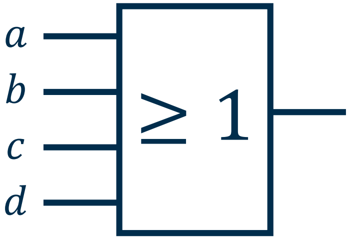
$y = \overline{a \,\&\, b \,\&\, c \,\&\, d}$

Beliebige Funktion:



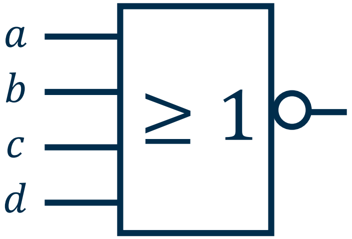
$y = F(a, b, c, d)$

OR-Gatter:



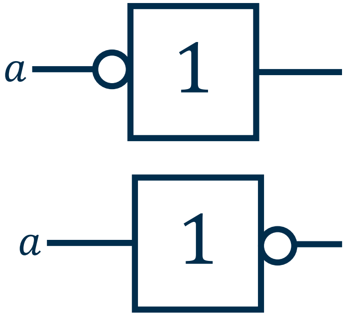
$y = a \vee b \vee c \vee d$

NOR-Gatter:



$y = \overline{a \vee b \vee c \vee d}$

Negationsgatter:



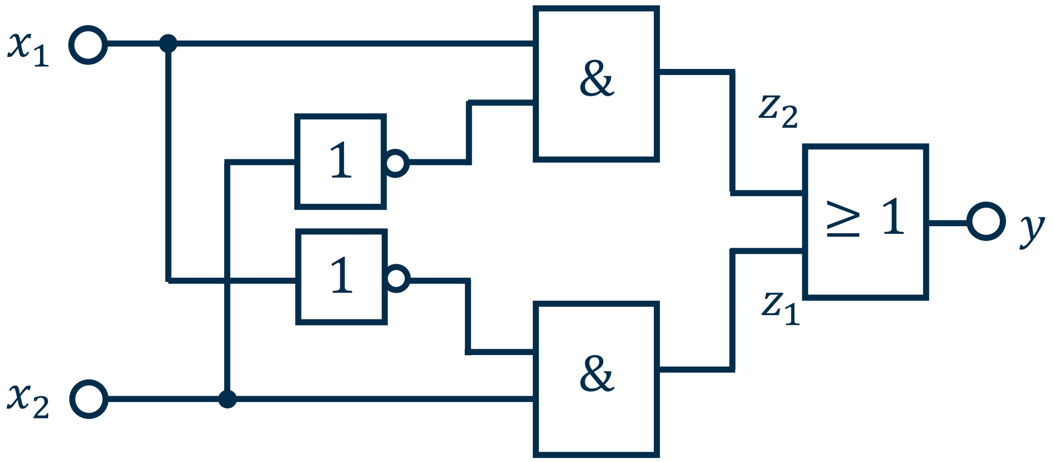
$y = \bar{a}$

$y = \bar{a}$

Schaltnetze

K. 7.4

- In der Regel bestehen logische Ausdrücke aus mehr als einem Operator
- Aus den logischen Ausdrücken kann unmittelbar eine **Konstruktionsvorschrift** für Digitalschaltungen erstellt werden
- Die maximale Anzahl von Schaltgliedern von den Eingängen zum Ausgang wird **Stufenzahl** oder **Tiefe** der Schaltung genannt (Inverter werden nicht gezählt)
- **Beispiel**: $y = (x_2 \,\&\, \overline{x_1}) \vee (\overline{x_2} \,\&\, x_1) = [z_2 \vee z_1]$



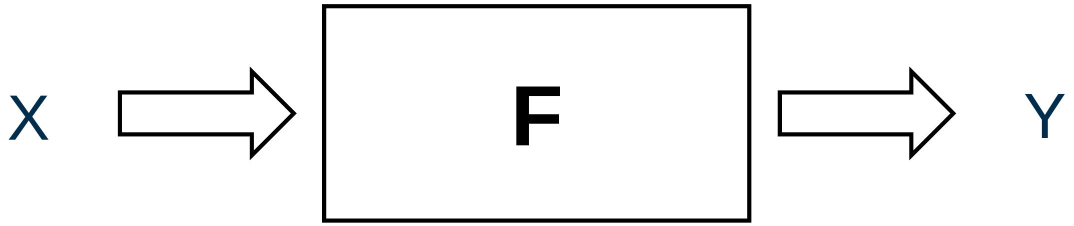
Schaltnetze

K. 7.4

Definition (nach DIN IEC 748 Teil 2)

Ein **Schaltnetz** ist eine Digitalschaltung, in der es für jede mögliche Kombination von digitalen Signalen an den Eingängen eine – und nur eine – Kombination von digitalen Signalen an den Ausgängen gibt

- Schaltnetze sind Schaltungen, die unmittelbar einem Strukturausdruck entsprechen
- Die Ausgänge hängen **nur von den Signalen an den Eingängen** ab
  - Hierbei werden Schaltzeiten der Logik-Gatter und Signallaufzeiten zunächst vernachlässigt



$Y^t = F(X^t)$

^{t} ist hier ein Hinweis darauf, dass Y zeitlich unmittelbar aus X folgt

Entwurf von Schaltnetzen

K. 7.4

1. Umsetzung der **verbalen Aufgabenstellung** in eine formale Form, z.B eine **Funktionstabelle**
2. Bildung einer **Normalform**; Bildung einer vollständigen Blocküberdeckung
3. Bildung einer **Minimalform** (z.B. mit S-Diagramm oder Nelson/Patrick)
4. Umformung in das gewählte **Basissystem**
5. Umformung in den **Strukturausdruck**
6. Umsetzen in das entsprechende **Schaltnetz**
   - (Unter Umständen sind nicht alle Schritte notwendig)

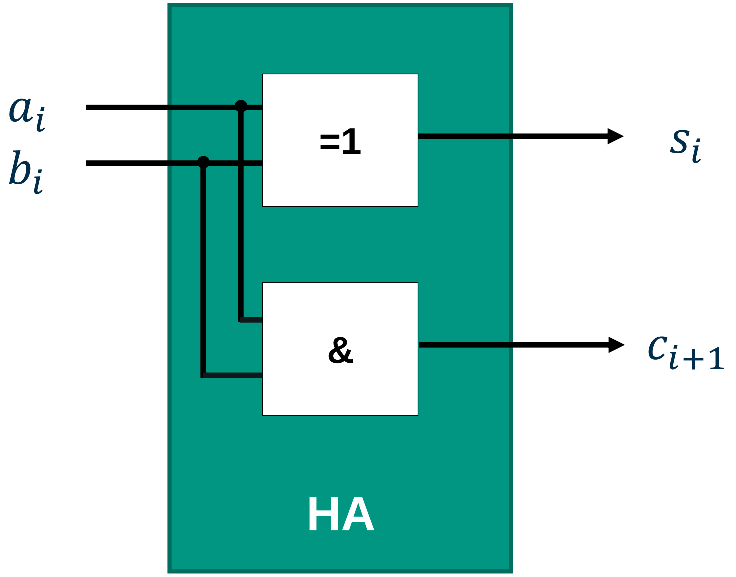
In Challenge 2 wird mit diesem Ablauf die Ansteuerung einer 7-Segment Anzeige entworfen.

Schaltnetzbeispiel: Addiererbaustein

- Digitale Systeme sollen nicht nur **logische** sondern auch **arithmetische Operationen** ausführen können
  - z.B. Addition, Subtraktion, Multiplikation
- Hierfür benötigen wir Grundbausteine, die diese Operationen realisieren
- **Addierer** sind Schaltungen, die uns ermöglichen digital eine Addition durchzuführen
- Es gibt verschiedene **Addiererstrukturen**, die sich in Laufzeit und Kosten (Gatteranzahl) unterscheiden

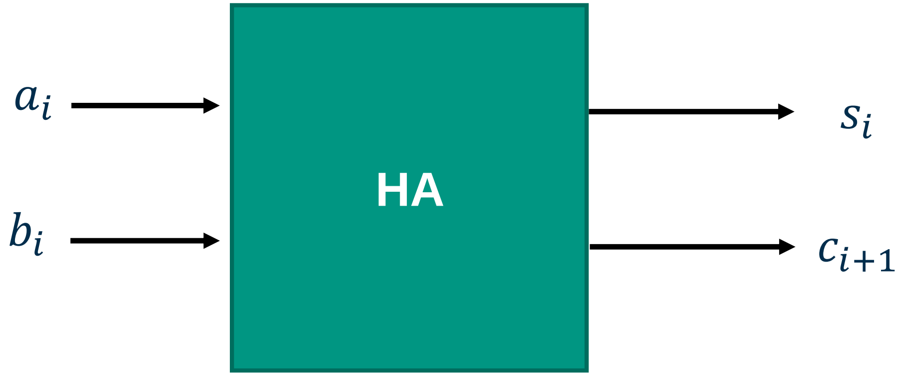
Schaltnetzbeispiel: Halbaddierer

- Summieren die beiden **Eingangsbits a_i** und **b_i** und legen die **Summe** auf den Ausgang **s_i**
- Zusätzlich wird ein **Übertragsbit c_{i+1}** erzeugt



- $s_i = a_i \oplus b_i$
- $c_{i+1} = a_i \cdot b_i$

a_i	b_i	s_i	c_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

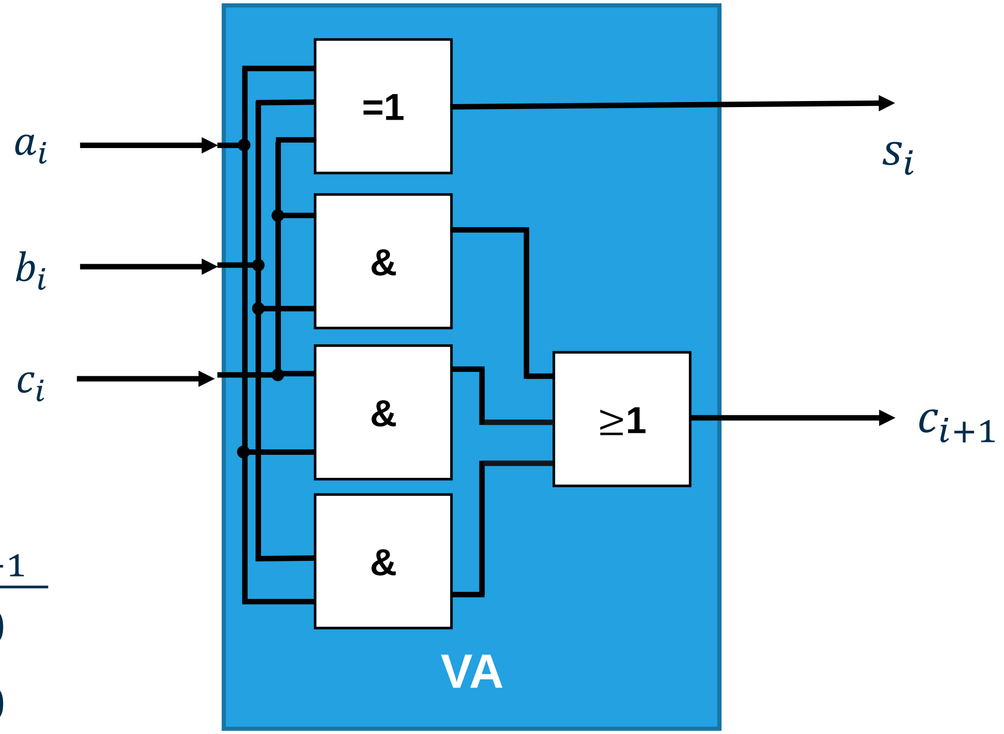


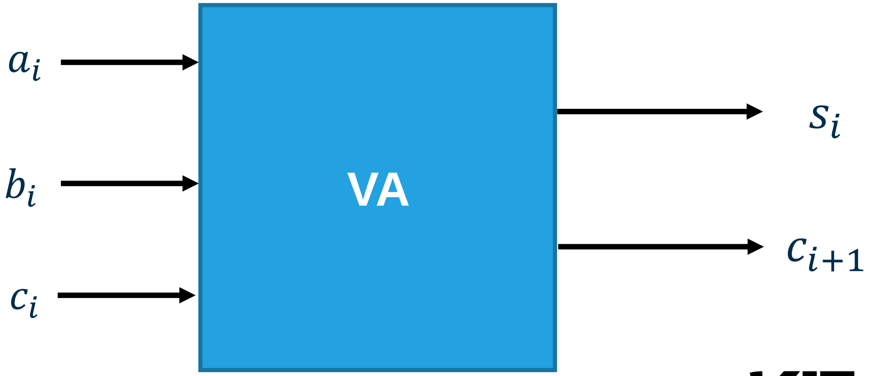
Schaltnetzbeispiel: Volladdierer

- Besitzen zusätzlich einen **Übertragseingang** und sind somit in der Lage, vorhergehende Stellen in die Berechnung einzubeziehen

- $s_i = a_i \oplus b_i \oplus c_i$
- $c_{i+1} = a_i \cdot b_i + a_i \cdot c_i + b_i \cdot c_i$

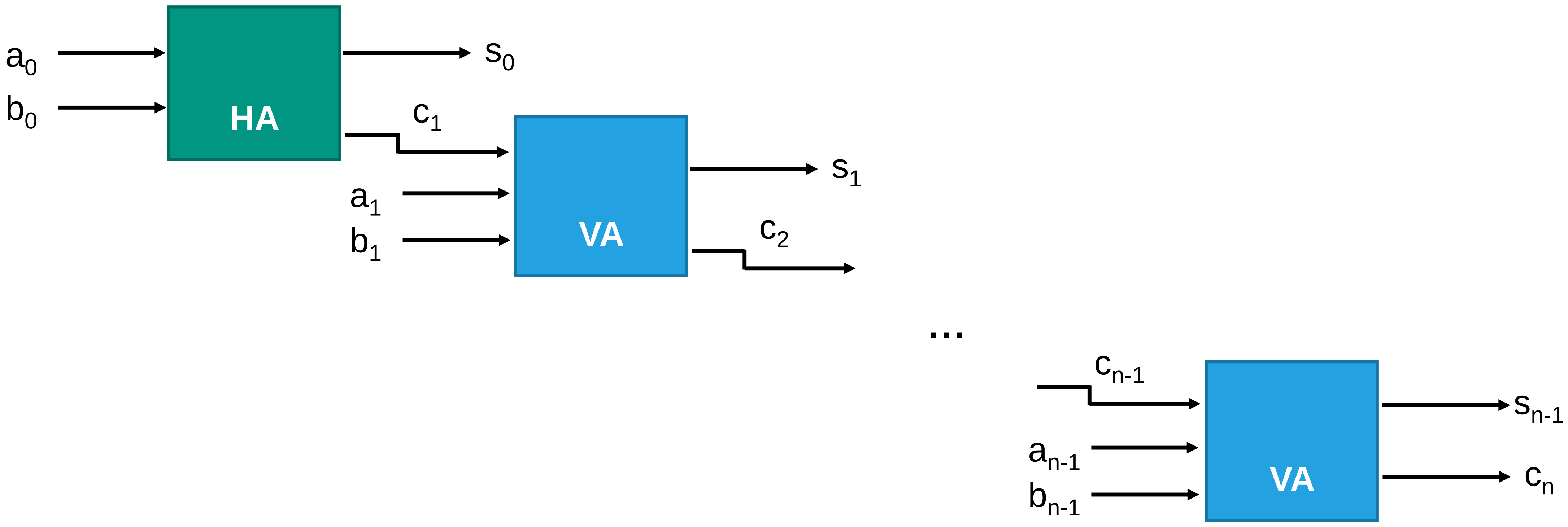
a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1





Schaltnetzbeispiel: n-Bit Ripple-Carry-Addierer

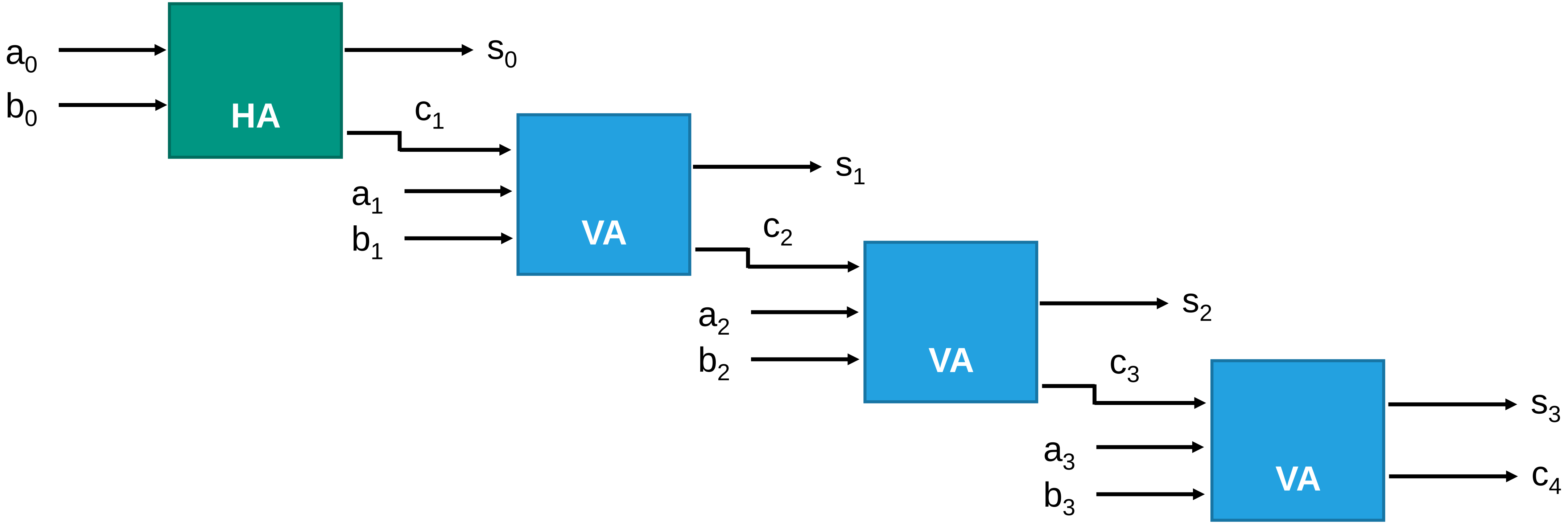
- 1. Baustein ist ein HA, die darauffolgenden sind VAs



Schaltnetzbeispiel: 4-Bit Ripple-Carry-Addierer

- **Nachteil**: sehr langer kritischer Pfad
- Die Laufzeit beträgt 2n Gatter für die Berechnung von c_n

Weitere Ansätze, z. B. der Carry-Look-Ahead Adder, werden in „Mikroelektronische Schaltungen und Systeme“ (**MSS**) behandelt.

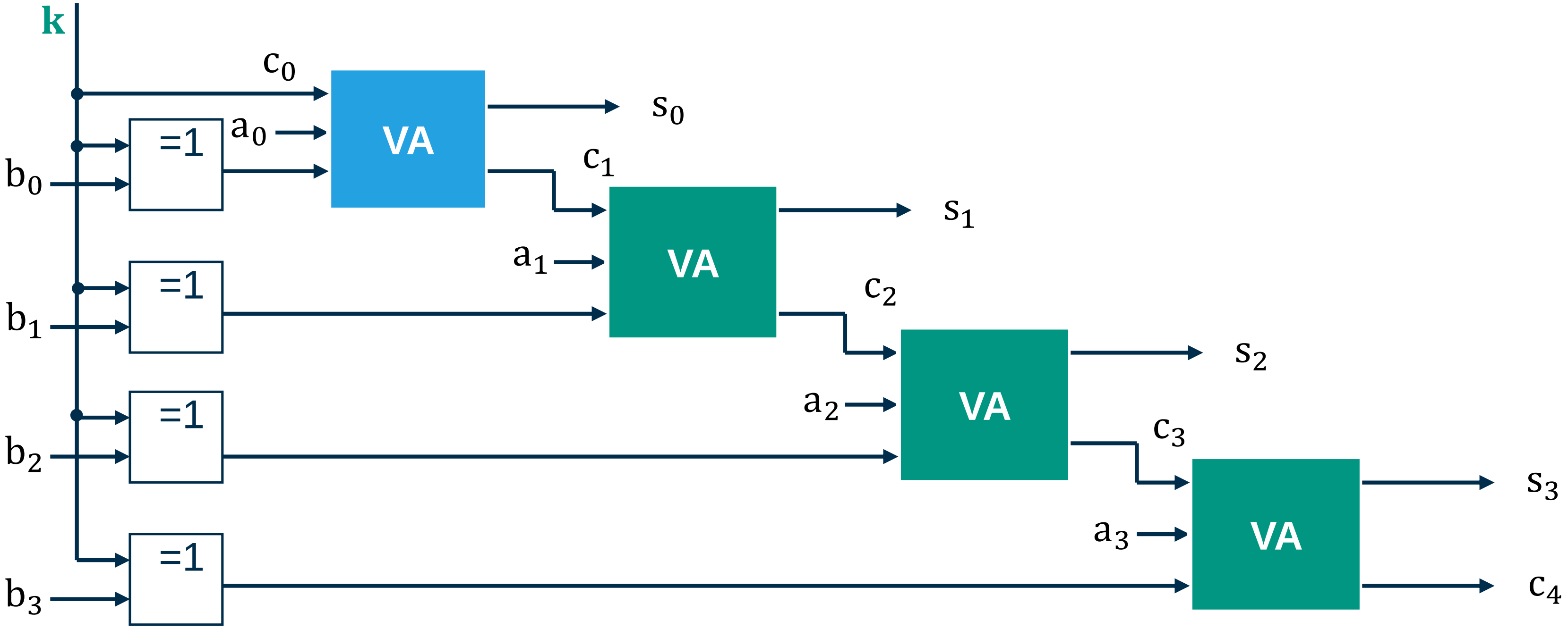


Schaltnetzbeispiel: Subtrahiererbaustein

- Die Subtraktion lässt sich auf die Addition des **K2-Komplements** zurückführen
- Dies bedeutet die **bitweise Negation** und Addition einer 1
- Für einen **kombinierten Addierer/Subtrahierer** lässt sich die bitweise Negation über **XOR** steuern

Schaltnetzbeispiel: Subtrahierbaustein

- **Beispiel**: 4-Bit Addierer/***Subtrahierer***
- **Addierer** mit **k=0**, ***Subtrahierer*** mit **k=1**



Schaltnetze

K. 7.4

- Anwendung schaltalgebraischer Regeln auf logische Ausdrücke erlaubt **vielfältige Formen für Schaltnetze**
  - Im Allgemeinen ist dann kein durchgängiges Ordnungsprinzip erkennbar („*krause Logik*")
  - Bei **komplexen Schaltnetzen**: unübersichtliche und teilweise schwierige Verhältnisse

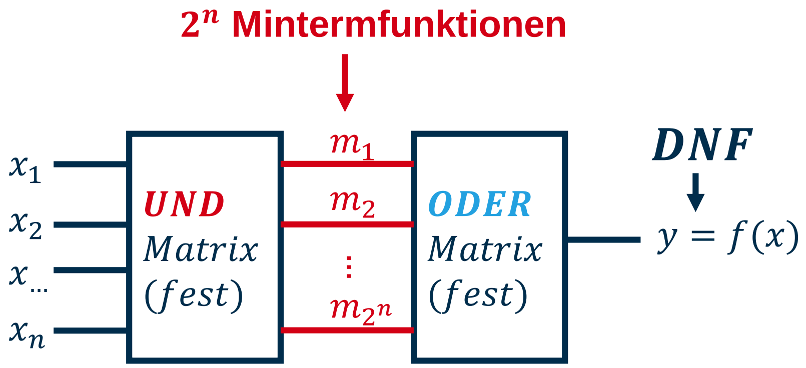
- Wir benötigen eine übersichtliche Methode zur **schnellen** und **kostengünstigen** Realisierung von Schaltnetzen in Form integrierter Schaltungen (ICs)

- **Grundprinzip**: Weglassen/Hinzufügen von Signalverbindungen in universell nutzbaren Hardwarestrukturen (Matrizen)
  - Minterm bzw. Maxtermorientiert (DNF/KNF)
  - Blockorientiert (DMF/KMF)

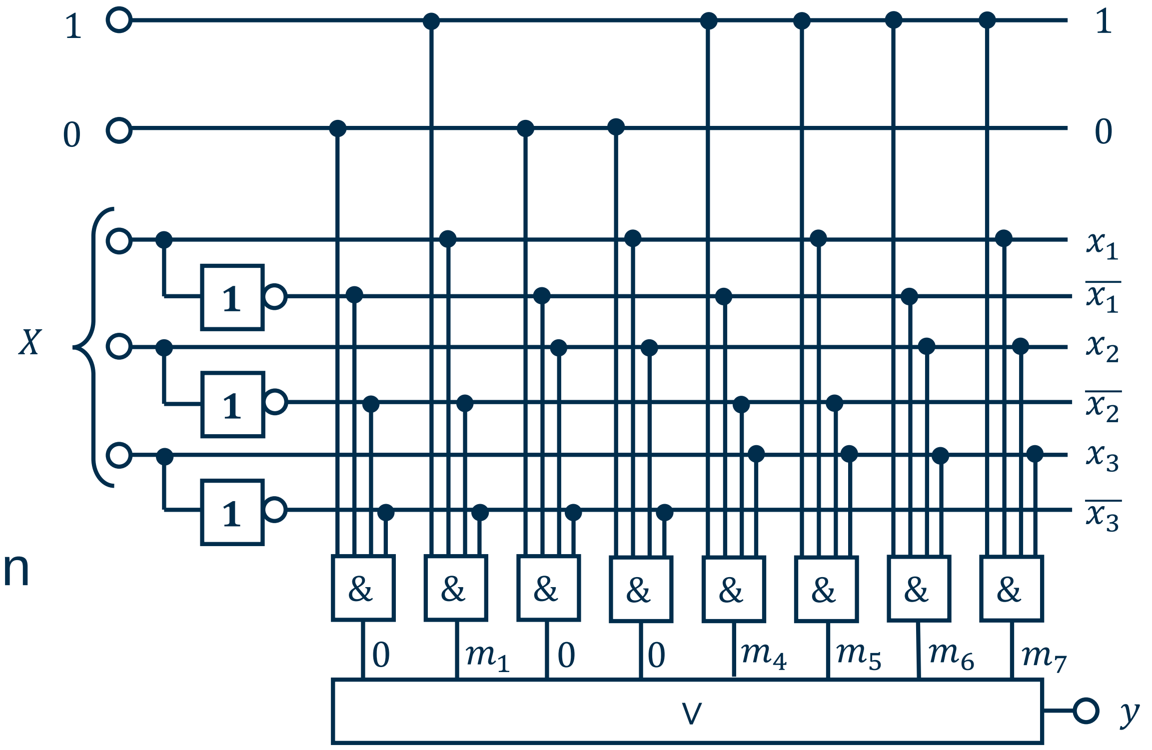
Normalformorientierte Struktur von Schaltnetzen

K.7.4

- **Normalformorientierte Struktur (DNF)**:
  - 2^n UND-Glieder in der ersten Stufe und ein bzw. mehrere ODER-Glieder in der zweiten Stufe
  - **Möglichkeiten zur Programmierung**:
    1. Einfluss: **Bilden** der ***Minterme*** (1. Stufe)
    2. Einfluss: **Einkoppeln** der ***Minterme*** (2. Stufe)

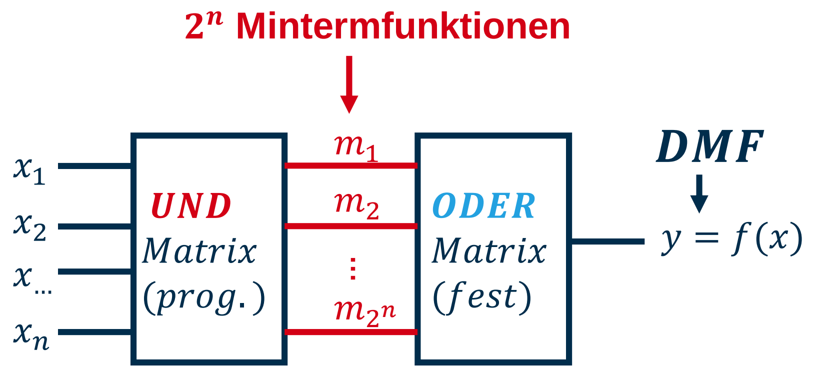


- <u>**Beispiel 1**</u>: **ULA (Universal Logic Array)**
  - $y = m_1 \vee m_4 \vee m_5 \vee m_6 \vee m_7$
  - **Programmierung über die 1. Stufe**:
    - **Konjunktive Verknüpfung** der **Minterme m_j** mit **0** oder **1**, um Wirkung ein- bzw. auszuschalten

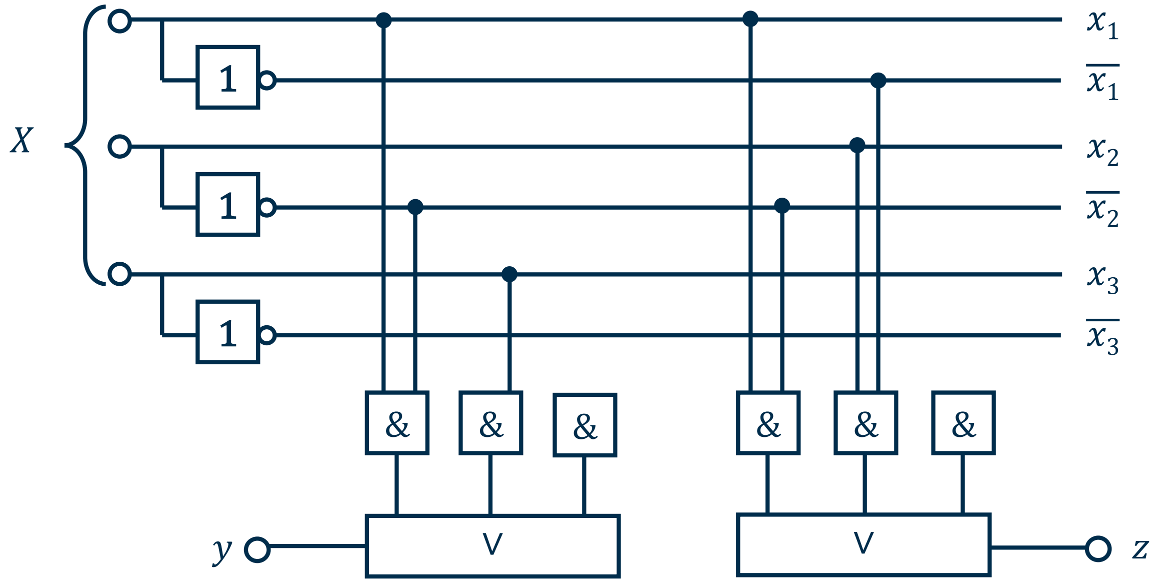


Blockorientierte Struktur von Schaltnetzen

- **Blockorientierte Struktur (DMF):**
  - $< 2^n$ UND-Glieder in der ersten Stufe und ein bzw. mehrere ODER-Glieder in der zweiten Stufe
  - **Möglichkeiten zur Programmierung**:
    1. Nur über die UND-Matrix (PAL-Baustein)
    2. UND- und ODER-Matrix (PLA-Baustein)



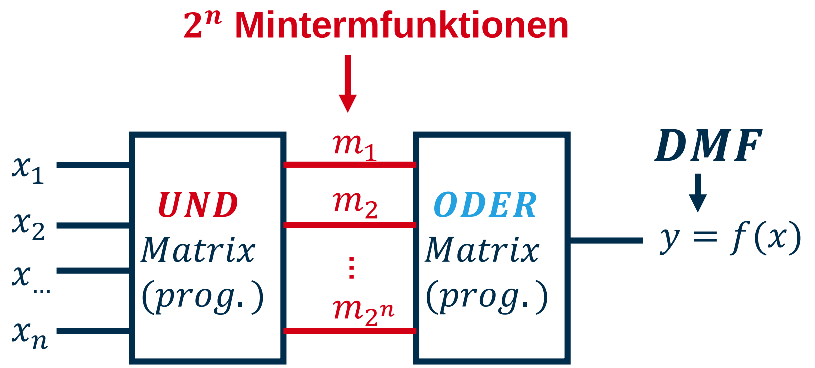
- **Beispiel 2**: **PAL (Programmable Array Logic)**
  - $y = (\overline{x_2} \,\&\, x_1) \vee (x_3)$
  - $z = (\overline{x_2} \,\&\, x_1) \vee (x_2 \,\&\, \overline{x_1})$
  - → Da die Gruppierung der UND-Verknüpfungen fest vorgegeben ist, muss der identische Term $(\overline{x_2} \,\&\, x_1)$ doppelt realisiert werden

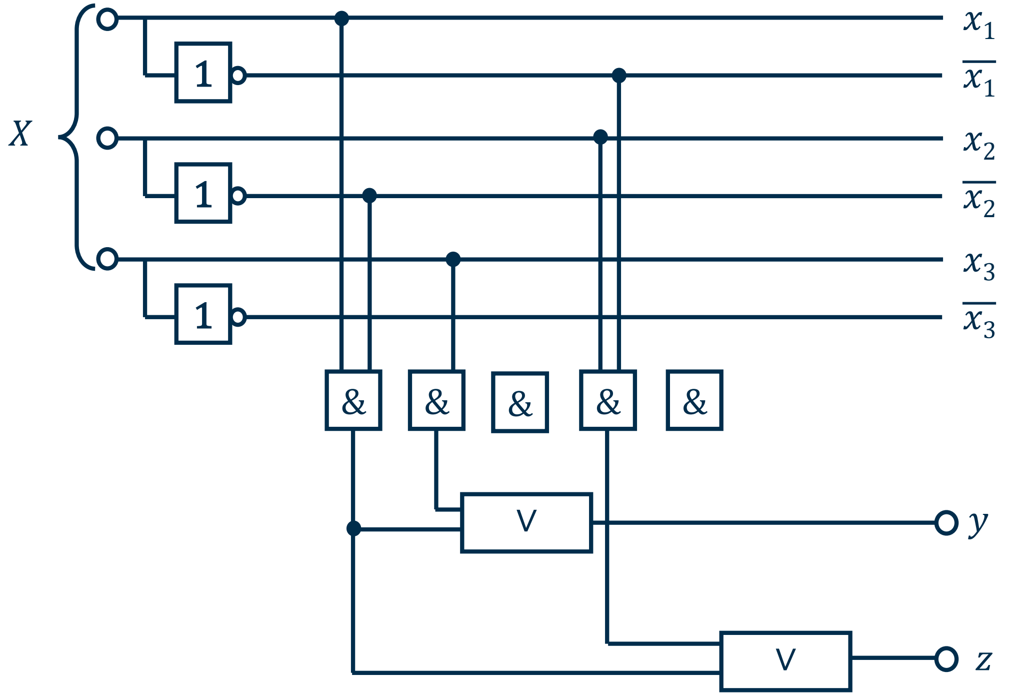


Blockorientierte Struktur von Schaltnetzen

K.7.4

- **Beispiel 3**: **PLA (Programmable Logic Array)**
  - $y = (\overline{x_2} \,\&\, x_1) \lor (x_3)$
  - $z = (\overline{x_2} \,\&\, x_1) \lor (x_2 \,\&\, \overline{x_1})$
  - → Da hier **beide** Matrizen beeinflussbar sind, kann der Term $(\overline{x_2} \,\&\, x_1)$ in beiden ODER-Matrizen verwendet werden

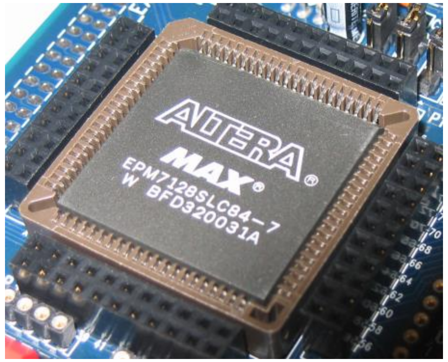


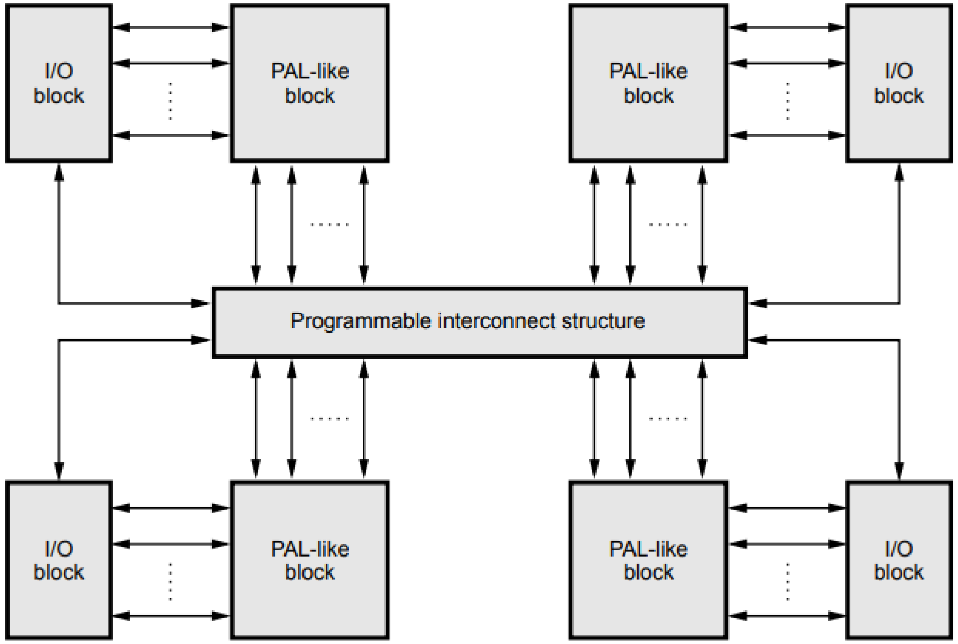


CPLD (Complex Programmable Logic Device)

- **Enthält mehrere Function Blocks (FBs)**
  - PAL-ähnliche Bauteile
- Jeder FB besitzt mehrere Macrocells
- Zentrale Interconnect-Matrix verbindet FBs und I/Os

- **<u>Eigenschaften</u>**:
  - Deterministische Signaldurchlaufzeit
  - „Instant-On“ Verhalten
  - Geringer Stromverbrauch





Entwicklung programmierbarer Logik

- **PLA/PAL**, ab den 1970er Jahren
  - Erste Realisierungen programmierbare Hardware
  - Sind inzwischen durch CPLDs und FPGAs ersetzt worden

- **CPLD** (Complex Programmable Logic Device), 1988
  - Technologischer Nachfolger von PAL/PLA
  - Heutzutage verwendet für einfache, deterministische Steuer- und Glue-Logic

- **FPGA** (Field Programmable Gate Array), 1985
  - Array aus Configurable Logic Blocks (CLBs)
  - Heutzutage verwendet für hochgradig parallele, flexible Hardwarebeschleunigung

FPGAs werden z. B. in **MSS** behandelt und in diversen Mastervorlesungen bzw. Praktika vertieft.

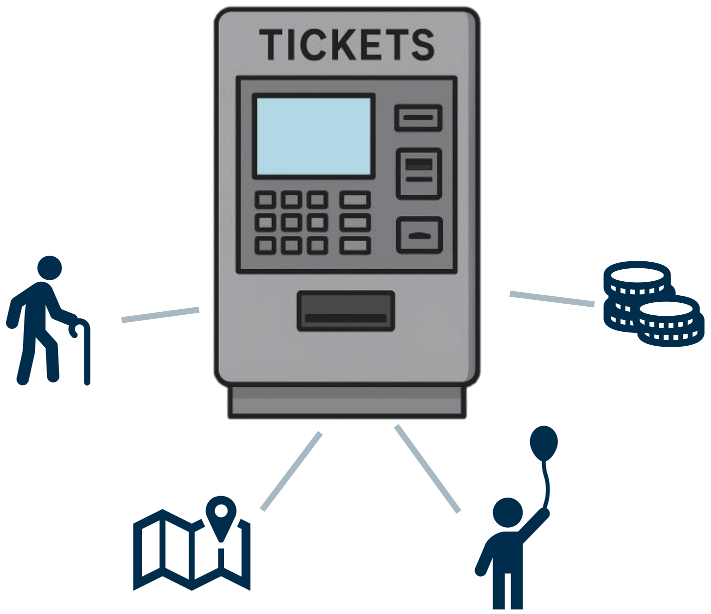
Wenn Sie den Hardwaretest der Challenges 1-3 auf dem Zedboard durchführen, programmieren Sie durch den generierten Bitstream einen **FPGA**

Automaten: Motivation

K. 7.6

- **Bisher wurden Schaltnetze behandelt:**
  - Ermöglichen die Bildung von Ausgangsgrößen unmittelbar aus Eingangsgrößen
  - **Maximal sinnvolle Größe** der Schaltnetze ist jedoch aus wirtschaftlichen und technischen Gründen beschränkt

- **Beispiel: Fahrkartenautomat**
  - **Eingangswerte:** Münzgröße/Scheine, Entfernung, Sondertarif, Kind/Erwachsene
  - **Realisierung als Schaltnetz:**
    - Es müssten entweder alle vier Eingaben **zeitgleich** betätigt werden oder ein Tastensatz verfügbar sein, dessen Mächtigkeit gleich der des kartesischen Produktes aller Mengen ist, um alle Kombinationen abzubilden



Geldeinwurf $\boldsymbol{G} = \{0{,}5\ € \cdot k \mid k \in \mathbb{N}, k < 100\}$

Wabenanzahl $\boldsymbol{W} = \{1,2,3,4,5\}$

Ermäßigung $\boldsymbol{E} = \{ja, nein\}$

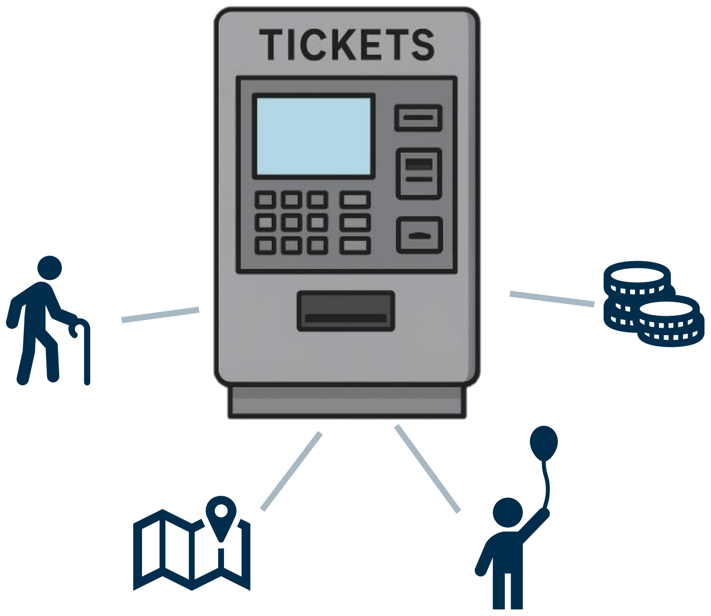
Kind $\boldsymbol{K} = \{ja, nein\}$

Automaten: Motivation

- **Reales Verhalten eines Fahrkartenautomaten**:
  - Die **Eingabe** einzelner Parameter erfolgt **nacheinander**

- → Dies entspricht einem **neuen Verarbeitungskonzept**, bei dem die Mächtigkeit des Eingabealphabets reduziert und die Komplexität der Ausgabegröße in eine andere Dimension verlagert wird

- Es werden vorherige **Eingaben gespeichert** und **Zustände** verwendet



Geldeinwurf $G = \{0{,}5\ € \cdot k \mid k \in \mathbb{N}, k < 100\}$

Wabenanzahl $W = \{1,2,3,4,5\}$

Ermäßigung $E = \{ja, nein\}$

Kind $K = \{ja, nein\}$

Automaten: Allgemein

K. 7.6

- **Verallgemeinerung des Problems**:
  - Endliches **Eingabealphabet**: $E = \{E_1, E_2, \dots, E_g, \dots, E_u\}$
  - Endliches **Ausgabealphabet**: $A = \{A_1, A_2, \dots A_h, \dots A_v\}$
  - ***Einheit AT***, die die **zeitliche Folge** von Elementen des **Eingabealphabets** $\boldsymbol{F_E}$ in eine **zeitliche Folge** von Elementen des **Ausgabealphabets** $\boldsymbol{F_A}$ abbildet

$$F_E \longrightarrow \boxed{\textbf{AT}} \longrightarrow F_A$$

- **Einführung** eines **Ordnungsindizes** $\boldsymbol{v}$ zur Unterscheidung ***zeitlicher*** Reihung:
  - $F_E = E_g^v \, E_g^{v-1} \, E_g^{v-2} \; \dots$ $\qquad$ $F_A = A_h^v \, A_h^{v-1} \, A_h^{v-2} \; \dots$

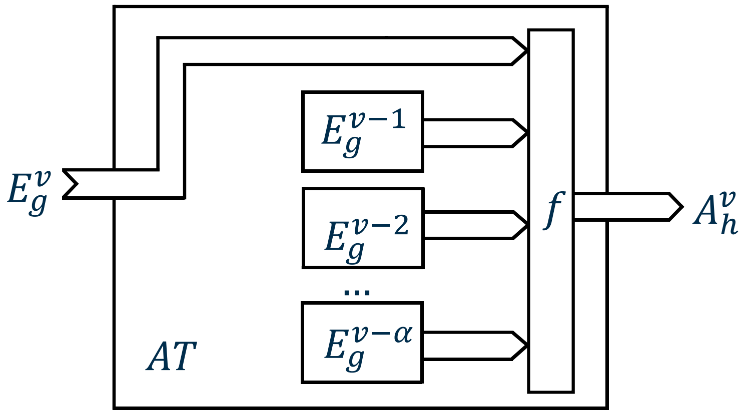
- **Für die *Einheit AT* gilt:** $A_h^v = f(E_g^v, E_g^{v-1}, E_g^{v-2}, \dots, E_g^{v-\alpha})$
  - Insgesamt werden $\boldsymbol{\alpha + 1}$ **Elemente** des Eingabealphabets beobachtet
  - $\boldsymbol{\alpha}$ ist die **zeitliche Tiefe** der Verarbeitung

> $\alpha \geq 1,\ \alpha = 0$ wäre der Sonderfall eines Schaltnetzes

Automaten: Allgemein

K. 7.6

- **Nachteil des Konzepts**: **Um A_h^v** berechnen zu können, müssen **α Eingabeelemente gespeichert** werden



- **Bei Erhalten eines neuen Elementes**:
  - **Ältestes Element $E_g^{v-\alpha}$** entfällt aus Folge mit Länge $(\alpha + 1)$
  - Bei Normierung der Indizes dem Alter nach erhält man:
    - v: $\quad \mathrm{E_g^v \; E_g^{v-1} \; E_g^{v-2} \; ... \; E_g^{v-(\alpha+1)} \; E_g^{v-\alpha}}$
    - $v + 1$: $\quad \mathrm{E_g^{v+1} \; E_g^v \; E_g^{v-1} \; E_g^{v-2} \; ... \; E_g^{v-(\alpha+1)} \; E_g^{v-\alpha}}$
    - $v + 1$: $\quad \mathrm{E_g^{v+1} \; E_g^v \; E_g^{v-1} \; E_g^{v-2} \; ... \; E_g^{v-(\alpha+1)} \; \cancel{E_g^{v-\alpha}}}$ **reduziert**
    - $v + 1 \Rightarrow v$: $\quad \mathrm{E_g^v \; E_g^{v-1} \; ... \; E_g^{v-2} \; E_g^{v-(\alpha+1)} \; E_g^{v-\alpha}}$ **normiert**

Automaten: Allgemein

K. 7.6

- **Abbildung der Folge**: $E_g^{v-1} \dots E_g^{v-\alpha}$ auf ein **neues Alphabet** $S = \{S_1, \dots, S_k, \dots S_w\}$ mit **geringerer Mächtigkeit**:
  - $\left(E_g^{v-1} \dots E_g^{v-\alpha}\right) \rightarrow S$, mit $|E|^\alpha \geq |S|$
  - Elemente $\boldsymbol{S_k} \in \boldsymbol{S}$ werden **Zustände** von AT genannt

- Die **Ausgabe** hängt nun von den **Eingabeelementen** und den **Zuständen** ab:
  - $\boldsymbol{A_h^v = \lambda\left(E_g^v, S_k^v\right)}$ mit $\boldsymbol{\lambda}$ als **Ausgabefunktion** **von** $\boldsymbol{AT}$

- Die **Zustände** müssen in Abhängigkeit des bisherigen Zustands und der Eingabe aktualisiert werden:
  - $\boldsymbol{S_k^{v+1} = \delta\left(E_g^v, S_k^v\right)}$ mit $\boldsymbol{\delta}$ als **Überführungsfunktion** der **Zustände**

Automaten: Allgemein

K. 7.6

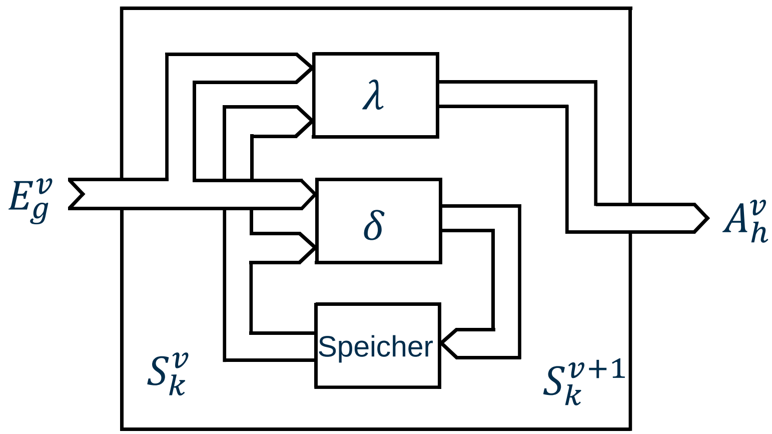
Definition

Ein endlicher **Automat** $\boldsymbol{AT}$ wird durch ein **Quintupel** aus drei endlichen Mengen und zwei Abbildungen zwischen diesen Mengen beschrieben:

$$\boldsymbol{AT = (E, A, S, \delta, \lambda)}$$

mit: E: Eingabealphabet, A: Ausgabealphabet, S: Zustandsalphabet,
δ: Überführungsfunktion, λ: Ausgabefunktion

- **Darstellung des Automaten durch folgende Struktur**:
  - ***Rekursive Anordnung*** zur Bestimmung des neuen Zustands
  - Der **Speicher** nimmt den momentanen Zustand S_k^v auf
  - Der neue **Zustand** $\boldsymbol{S_k^{v+1}}$ muss zum **richtigen Zeitpunkt** übernommen werden



Automaten: Typen

K. 7.6

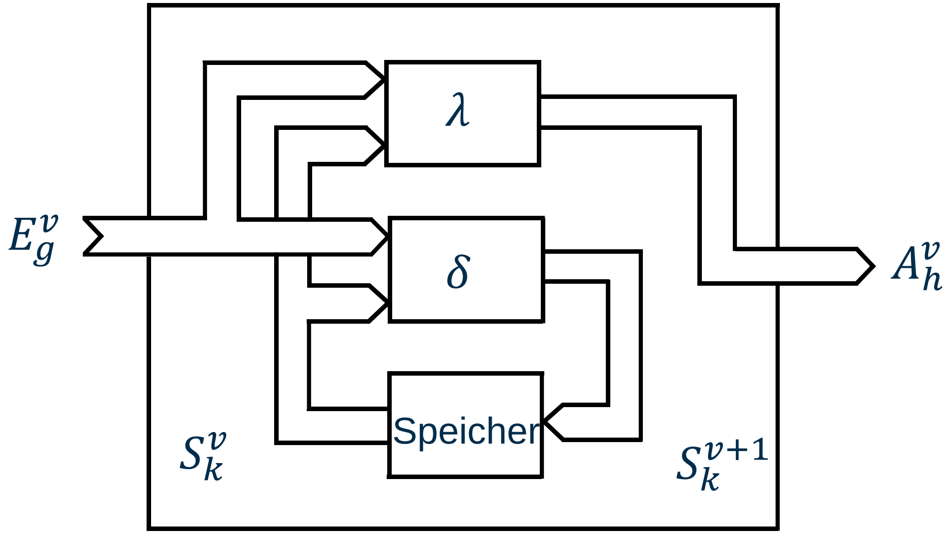
- Wichtige ***Typklassen*** **von Automaten** im Zusammenhang mit Digitalschaltungen:

  Generell: **Endliche**, **diskrete** und **deterministische** **Automaten**:
  - **Endlich**: erreicht einen definierten Endzustand, d. h. **E, A, S** sind **endliche Mengen**
  - **Diskret**: es werden „*gültige*" Eingaben an **diskreten Zeitpunkten** verarbeitet
  - **Deterministisch**: Automat ist „***eindeutig***" in der **Berechnungssequenz** der Eingaben, d. h. wann der Automat welchen **Zustand** + **Ausgabe** einnimmt ist immer **eindeutig**

- Anhand **Abhängigkeiten** der **Ausgabefunktion** von E_g^v und S_k^v unterscheidet man drei Typen von Automaten:
  - ***Mealy*-Automat**
  - ***Moore*-Automat**
  - ***Medwedew*-Automat**

Automaten: Typen

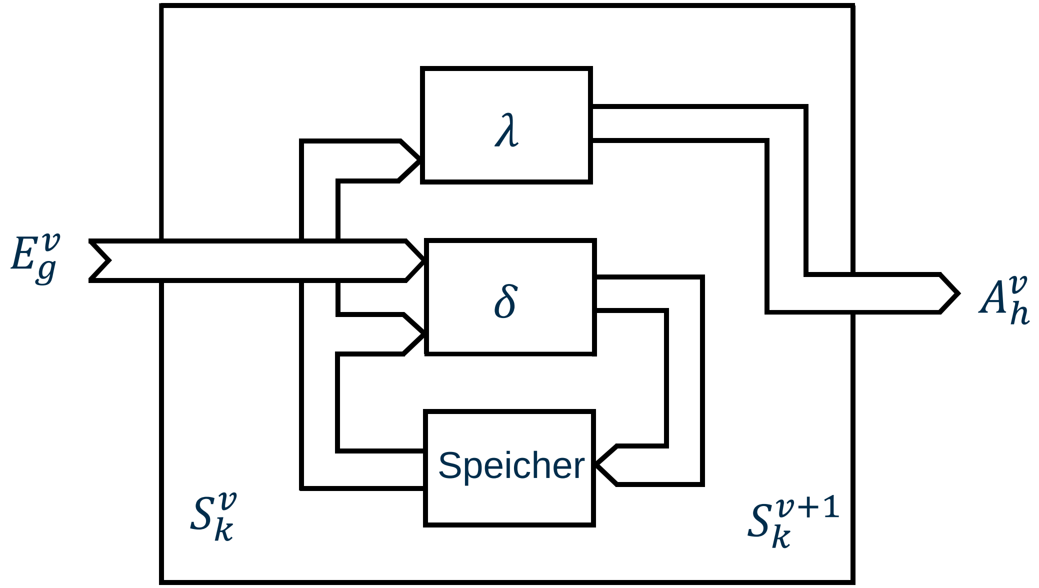
- **Mealy-Automat**
  - $A_h^v = \lambda\,(E_g^v, S_k^v)$ als **allgemeinster** Fall
  - **Ausgabe** hängt sowohl von der **Eingabe** als auch vom **aktuellen Zustand** ab



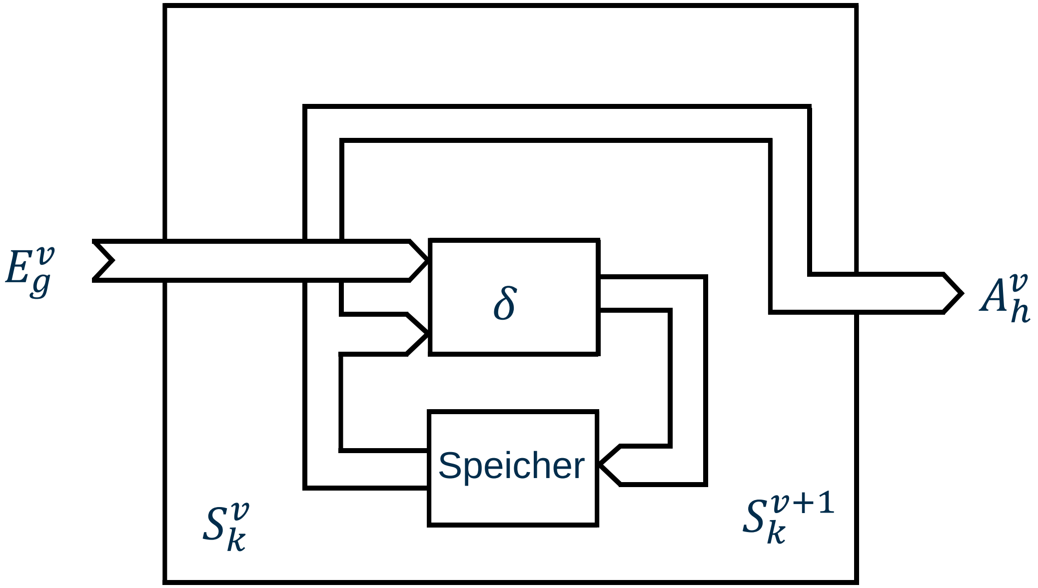
Automaten: Typen

K. 7.6

- **Moore-Automat**
  - $A_h^v = \lambda\,(S_k^v)$ als ein Spezialfall, bei dem die **Ausgabe alleine** vom **Zustand** abhängt



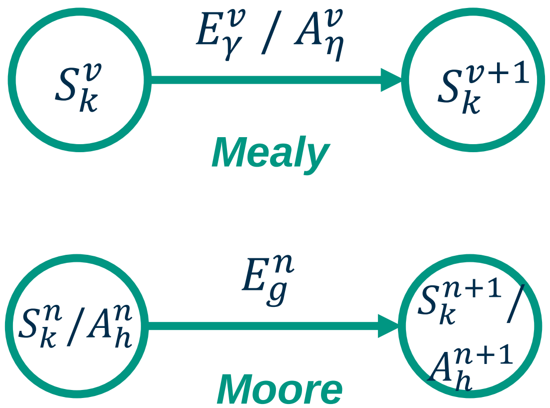
- **Medwedew-Automat**
  - $A_h^v = S_k^v$ als Spezialfall eines Moore-Automaten, bei dem der **Zustand selbst** als **Ausgabewert** dient



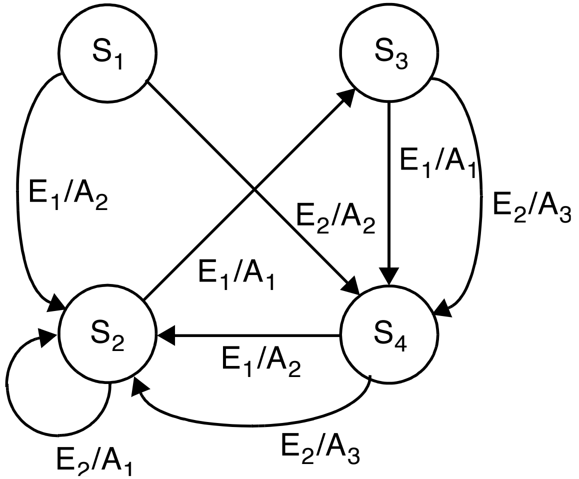
Automaten: Beschreiben des Verhaltens

K. 7.6

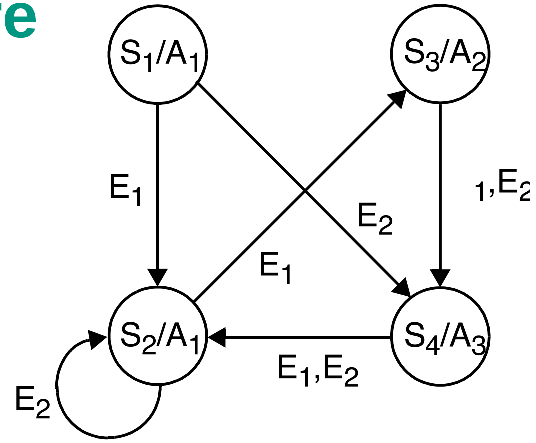
- **Automatengraphen**
  - **Knoten** repräsentieren **Zustände**, Kanten als **Zustandsübergange**
  - **Beide Elemente** des Graphen werden mit ***Attributen*** versehen, die sich auf **Eingangs-** und **Ausgangselemente** beziehen



- **Beispiel**: **Mealy**



- **Beispiel**: **Moore**



Automaten: Beschreiben des Verhaltens

K. 7.6

- **Automatentafel**
  - Bilden des **kartesischen Produktes** aus **Eingabe-** und **Zustandsmenge**
  - An **Kreuzungsstellen** werden beim **Mealy-Automaten** der jeweilige **Folgezustand** und die **Ausgabe**, beim **Moore-Automaten** nur der **Folgezustand** eingetragen

Mealy

	S^{v+1} / A^v			
S^v	E_1^v	E_2^v	$E_{\dots}^v$	E_n^v
S_1^v	…	S_k^{v+1}/A_h^v	…	S_k^{v+1}/A_h^v
S_2^v	…	S_k^{v+1}/A_h^v	…	…
$S_{\dots}^v$	…	…	S_k^{v+1}/A_h^v	…
S_n^v	S_k^{v+1}/A_h^v	…	…	…

Moore

	S^{v+1}				
S^v	E_1^v	E_2^v	$E_{\dots}^v$	E_n^v	
S_1^v	…	S_k^{v+1}	…	S_k^{v+1}	A^v
S_2^v	…	S_k^{v+1}	…	…	…
$S_{\dots}^v$	…	…	S_k^{v+1}	…	…
S_n^v	S_k^{v+1}	…	…	…	A^v

Schaltwerke

K. 7.6

Definition

Die **technische Realisierung** eines endlichen, diskreten und deterministischen Automaten wird ***Schaltwerk*** genannt (im Gegensatz zum „gedächtnislosen“ Schaltnetz)**.**

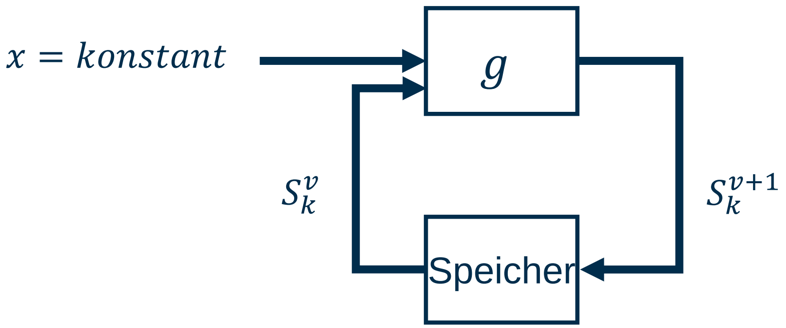
- **Unterscheidung:** zwischen **Mealy-**, **Moore-** und **Medwedew-Schaltwerken**

- Beim Übergang von **Automaten** zum ***Schaltwerk*** ist eine **eineindeutige Abbildung der Automatenelemente** in binäre Größen (Bit-Leitung) erforderlich:
  - $E \leftrightarrow \{X_j\}$ **mit:** $|E| \leq 2^n$ bei $X = \{x_n, \dots, x_1\}$
  - $A \leftrightarrow \{Y_i\}$ **mit:** $|A| \leq 2^m$ bei $Y = \{y_m, \dots, y_1\}$
  - **Zustandskodierung**: $S \leftrightarrow \{Q_k\}$ **mit:** $|S| \leq 2^r$ bei $Q = \{q_r, \dots q_1\}$
  - **Ausgabefunktion** λ und **Zustandsüberführungsfunktion** δ können als **Schaltnetze** abgebildet werden

Schaltwerke: asynchrone Schaltwerke

K. 7.6

- Formal eingeführte Indizierung v der **zeitlichen Ordnung** muss nun in **technische Realisierung** überführt werden:
  - **Rückführung der Zustände über einen Speicher**:



- **Schaltwerke**, bei denen **Rückführungen direkt wirksam** werden heißen **asynchron**
  - Diese sind schwieriger zu entwerfen bzw. zu betreiben und daher nur für spezielle Anwendungen vorgesehen

Schaltwerke: Synchrone Schaltwerke

- Stattdessen wird ein **Speicher** benötigt, der **nur zu bestimmten Zeitpunkten** die am Speicher anliegenden Werte übernimmt

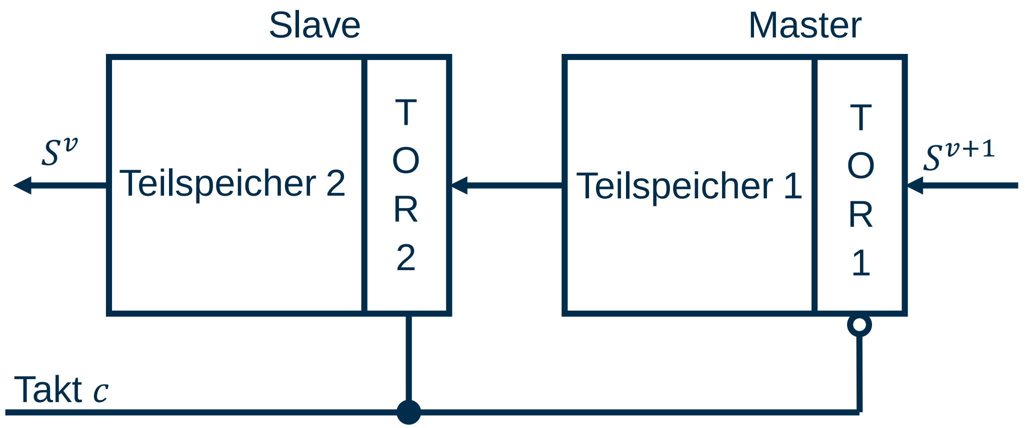
- **Schaltwerke** mit taktgesteuerten Speichern werden **getaktet betrieben** genannt
  - Sind die Takte an allen Schaltungsteilen gleichzeitig aktiv, ist es ein **synchrones Schaltwerk**

- Als zusätzliches *„**informationsfreies Binärsignal**"* wird der sogenannte **Takt $\mathbf{c}$** benötigt:
  - $\mathbf{c} = \mathbf{0}$: angeschlossene Elemente sind **nicht aktiviert**
  - $\mathbf{c} = \mathbf{1}$: angeschlossene Elemente sind **aktiviert**

Schaltwerke: Synchrone Schaltwerke

K. 7.6

- Auftrennung und partielle Weiterleitung des Zustandes bzw. Folgezustandes:
  - $\underline{c = 0}$:
    - Durch Negation an **TOR 1** ist es **aktiv**
    - S^{v+1} wird in Teilspeicher 1 geladen
  - $\underline{c = 1}$:
    - **TOR 2** ist **aktiv**
    - Teilspeicher 2 speichert Inhalt von Teilspeicher 1
    - nächster Zustand wird zum aktuellen Zustand $S^{v+1} \rightarrow S^{v}$

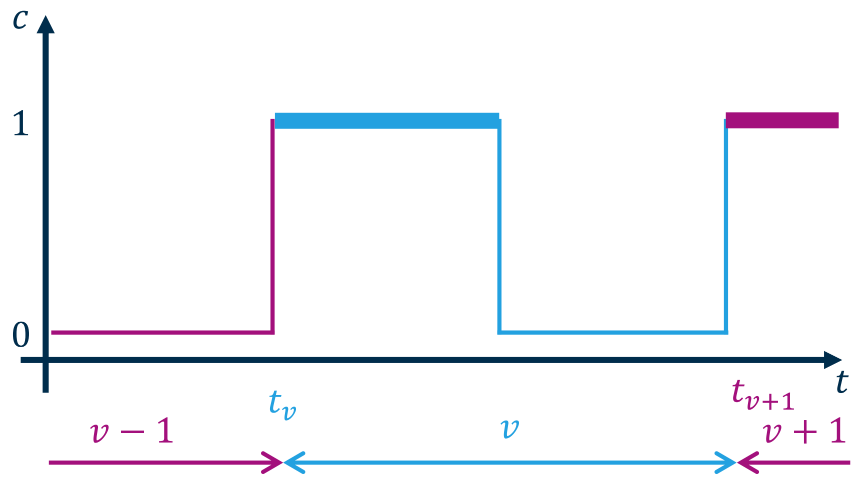


- **Tor 1 und Tor 2 sind nie gleichzeitig aktiv!**

Schaltwerke: Steuerarten des Taktes

K. 7.6

- **Mealy-Automat:**
  - **Änderung** der **Eingabe** bewirkt asynchrone **Ausgabenberechnung**
  - **Zustandsübergänge** werden ausschließlich vom Takt bestimmt
    - Die **Zählung** des **Indexes** v muss aus dem **Taktsignal abgeleitet** werden

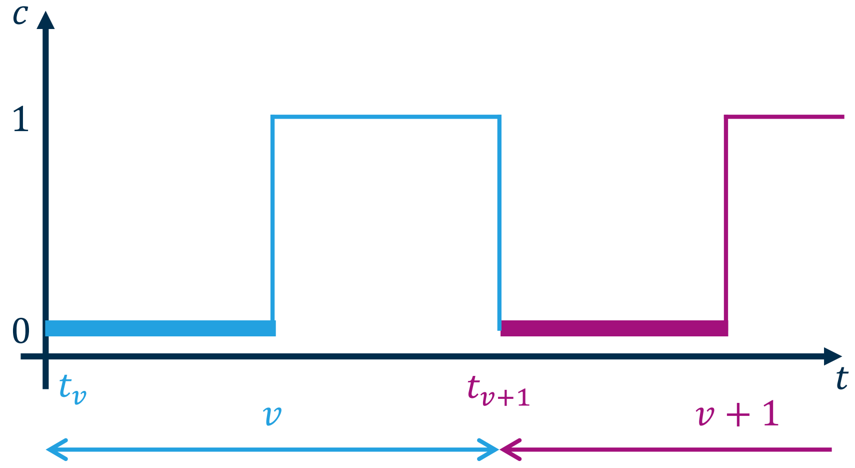


- **Steuerungsarten** des **Taktes**:
  - **Pegelgesteuerter Takt auf 1**

Schaltwerke: Steuerarten des Taktes

K. 7.6

- **Mealy-Automat:**
  - **Änderung** der **Eingabe** bewirkt asynchrone **Ausgabenberechnung**
  - **Zustandsübergänge** werden ausschließlich vom Takt bestimmt
    - Die **Zählung** des **Indexes** v muss aus dem **Taktsignal abgeleitet** werden

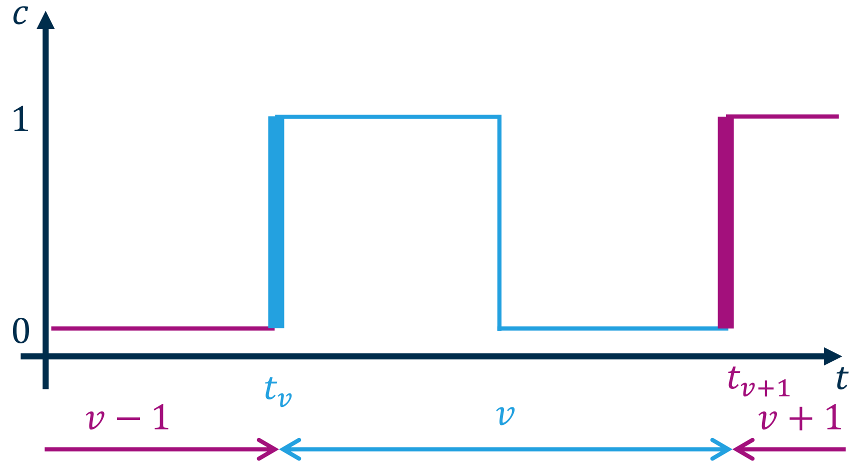


- **Steuerungsarten** des **Taktes**:
  - Pegelgesteuerter Takt auf 1
  - **Pegelgesteuerter Takt auf 0**

Schaltwerke: Steuerarten des Taktes

K. 7.6

- **Mealy-Automat:**
  - **Änderung** der **Eingabe** bewirkt asynchrone **Ausgabenberechnung**
  - **Zustandsübergänge** werden ausschließlich vom Takt bestimmt
    - Die **Zählung** des **Indexes** v muss aus dem **Taktsignal abgeleitet** werden

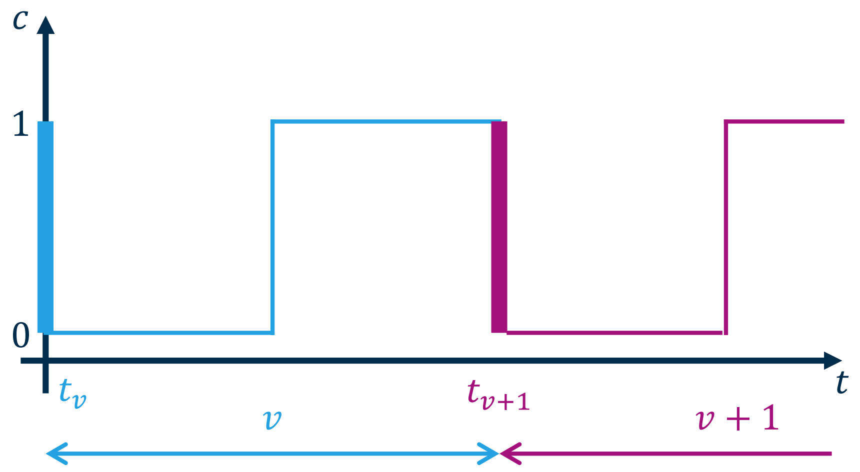


- **Steuerungsarten** des **Taktes**:
  - Pegelgesteuerter Takt auf 1
  - Pegelgesteuerter Takt auf 0
  - **Flankengesteuerter Takt auf $0 \rightarrow 1$**

Schaltwerke: Steuerarten des Taktes

K. 7.6

- **Mealy-Automat:**
  - **Änderung** der **Eingabe** bewirkt asynchrone **Ausgabenberechnung**
  - **Zustandsübergänge** werden ausschließlich vom Takt bestimmt
    - Die **Zählung** des **Indexes v** muss aus dem **Taktsignal abgeleitet** werden



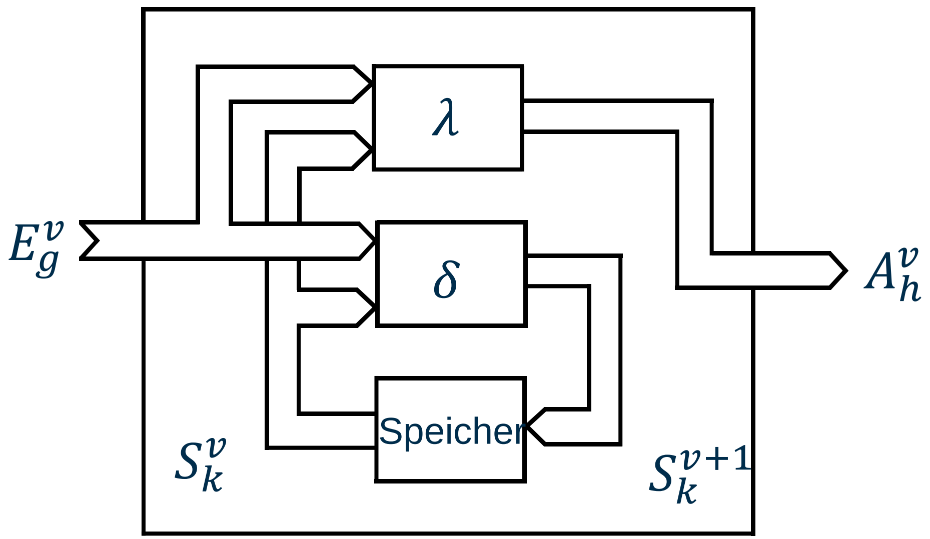
- Steuerungsarten des Taktes:
  - Pegelgesteuerter Takt auf 1
  - Pegelgesteuerter Takt auf 0
  - Flankengesteuerter Takt auf $0 \rightarrow 1$
  - **Flankengesteuerter Takt auf $1 \rightarrow 0$**

Schaltwerke: Speichern der Zustände

- Die Zustände eines Schaltwerks müssen gespeichert werden

- Ist $|S|$ die **Zahl der Zustände**, berechnet sich die **Größe des Binärvektors $\boldsymbol{Q}$** zu:
  - $r = \lceil ld\ |S| \rceil$ mit $Q = (q_r, \dots, q_k, \dots, q_1)$
  - mit q_k einer beliebigen Zustandsvariable des Zustandsvektors

- Einzelne Zustandsvariablen q_k werden **rekursiv** mit δ berechnet



Binärspeicher (FlipFlops)

K. 7.6

Definition

Eine **Einheit**, die zwei Werte speichert, wird als ***Binärspeicher*** bzw. ***FlipFlop*** bezeichnet.
Ein **Speicher** mit r ***FlipFlops*** kann 2^r **unterschiedliche Zustände speichern**.

- **Die grundsätzlichen Funktionen eines *Binärspeichers* sind:**
  - Er muss **wahlweise** eine $\mathbf{0}$ oder $\mathbf{1}$ einspeichern können (***schreiben***)
  - Der **Wert** im **Speicher** muss extern **abrufbar** sein (***lesen***)
  - Beim **Vorhandensein** eines **Taktes** muss die **Rekursion** $\mathbf{v+1 \rightarrow v}$ vom ***Binärspeicher*** unter Einwirkung des Taktes verwirklicht werden

- Es sind verschiedene **Varianten** der ***Binärspeicher*** möglich
  - Werden im Folgenden diskutiert

RS-FlipFlop: Funktionsweise

K. 7.6

- **RS-FlipFlop**
  - **Lesesignal ≙ Zustandswert**
    - $\boldsymbol{q_k} \in \{0, 1\}$
  - **Schreiben** einer 0: **Rücksetzen** (**R**-Bit)
    - $\boldsymbol{R} \in \{0, 1\}$, 0 ≙ inaktiv
1 ≙ Schreiben 0
  - **Schreiben** einer 1: **Setzen** (**S**-Bit)
    - $\boldsymbol{S} \in \{0,1\}$, 0 ≙ inaktiv
1 ≙ Schreiben 1
  - Das **gleichzeitige Schreiben** von 0 und 1 ($R = 1, S = 1$) ist **unzulässig**

- **Funktionstabelle RS-FlipFlop:**

q_k^{alt}	R	S	q_k^{neu}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	-
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	-

D-FlipFlop: Funktionsweise

K. 7.6

- **D-FlipFlop**
  - **Lesesignal ≙ Zustandswert**
    - $\boldsymbol{q_k} \in \{0, 1\}$
  - **Datensignal ≙ Zustandswert neu**
    - $\boldsymbol{D} \in \{0, 1\}$
  - **Schreiben** des Wertes von **D**:
    $\boldsymbol{W} \in \{0,1\}$, 0 ≙ inaktiv
    1 ≙ Schreiben

- **Funktionstabelle D-FlipFlop**

q_k^{alt}	D	W	q_k^{neu}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

FlipFlops: Charakteristische Gleichungen

K. 7.6

- **Übertragung** der **zeitlichen Indizes** v und $v+1$ auf die **Schaltnetzgrößen**:
  - $q_k{}^{alt} = q_k^v,$ $\qquad q_k^{neu} = q_k^{v+1}, R^v, S^v, D^v, W^v$

- **Überführungsfunktion** zur Bildung von q_k^{v+1} für beide Fälle:

		S^v	S^v	
	0 (0)	1 (1)	1 (5)	1 (4)
R^v	0 (2)	– (3)	– (7)	0 (6)
			q_k^v	q_k^v

		W^v	W^v	
	0 (0)	0 (1)	0 (5)	1 (4)
D^v	0 (2)	1 (3)	1 (7)	1 (6)
			q_k^v	q_k^v

RS-FlipFlop

q_k^{alt}	R	S	q_k^{neu}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	-
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	-

D-FlipFlop

q_k^{alt}	D	W	q_k^{neu}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- → Die **charakteristischen Gleichungen** lauten:
  - $q_k^{v+1} = S^v \vee (q_k^v \,\&\, \overline{R^v})$, mit $S^v \,\&\, R^v = 0$ $\qquad q_k^{v+1} = (\mathrm{D}^{\mathrm{v}} \,\&\, W^v) \vee (q_k^v \,\&\, \overline{W^v})$

FlipFlops: Ansteuerfunktionen

K. 7.6

- **Zusammenhang** für einen bestimmten Werteübergang: $q_k^v \rightarrow q_k^{v+1}$
  - Gibt die sogenannte **Ansteuerfunktion** an
- **Beispiele**:

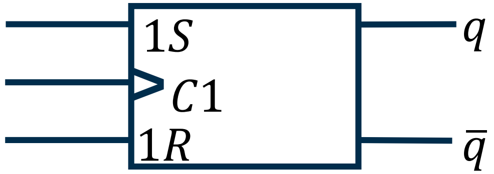
q^v	q^{v+1}	R^v	S^v
0	0	-	0
0	1	0	1
1	0	1	0
1	1	0	-

q^v	q^{v+1}	D^v	W^v
0	0	-	0
0	1	1	1
1	0	0	1
1	1	-	0

  - **Daraus ergibt sich**:
    - $R^v = \overline{q_k^{v+1}}, \; S^v = q_k^{v+1}$ $\quad$ $D^v = q_k^{v+1}$ oder $D^v = \overline{q_k^v}, \; W^v = q_k^{v+1} \not\equiv q_k^v$
- neben der formalen Behandlung des Binärspeicherverhaltens ist die **technische Realisierung** solcher Elemente von Bedeutung

FlipFlops: Zusammenfassung

Symbole:



Charakteristische Gleichungen:

RS-FlipFlop

$$q^{v+1} = S \vee (q^v \,\&\, \bar{R})$$

D-FlipFlop

$$q^{v+1} = D$$

Ansteuerfunktionen:

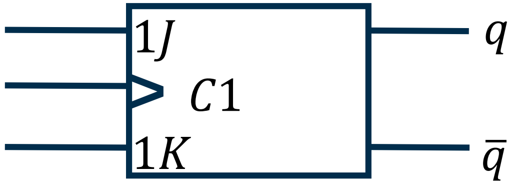
q^v	q^{v+1}	R^v	S^v
0	0	-	0
0	1	0	1
1	0	1	0
1	1	0	-

q^v	q^{v+1}	D^v
0	0	0
0	1	1
1	0	0
1	1	1

Binärspeicher: FlipFlop Zusammenfassung

K. 7.6

Symbole:



Charakteristische Gleichungen:

T-FlipFlop

$$q^{v+1} = (T \,\&\, \overline{q^v}) \vee (\bar{T} \,\&\, q^v)$$

JK-FlipFlop

$$q^{v+1} = (\overline{K} \,\&\, q^v) \vee (J \,\&\, \overline{q^v})$$

Ansteuerfunktionen:

q^v	q^{v+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

q^v	q^{v+1}	K	J
0	0	-	0
0	1	-	1
1	0	1	-
1	1	0	-

Ablaufdiagramm und Ablauftabelle

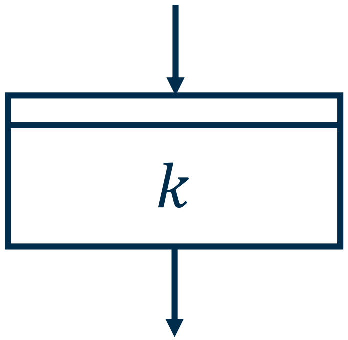
K. 7.6

- Nicht alle Komponenten des Eingabevektors beim Übergang zum Folgezustand sind relevant und müssen berücksichtigt werden
- Schaltwerkstafel bzw. Schaltwerksgraph ist als **Beschreibungsmittel oftmals ungünstig**

- Daher: Einführung des Ablaufdiagramms und der Ablauftabelle
  - **Platzsparende** und **effizientere** Beschreibungsform

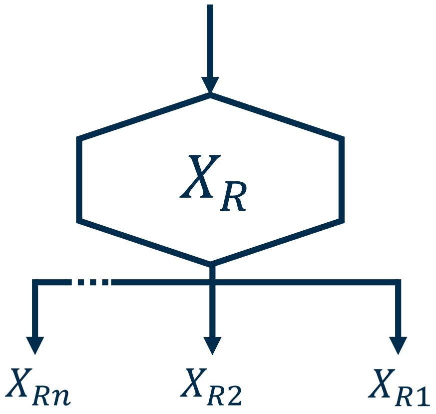
Ablaufdiagramm

- Zustandsübergang



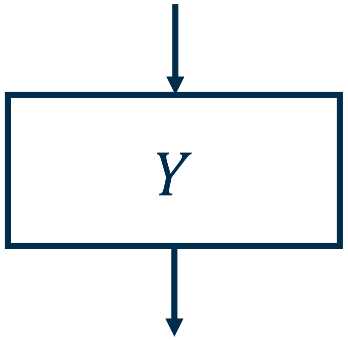
Übergang in den Zustand k,
Ausgelöst durch das Taktsignal

- Abfrage (Verzweigung)



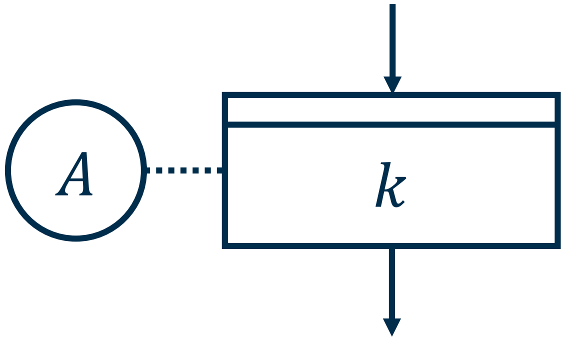
Abfrage der im aktuellen Zustand k relevanten Eingangsvariablen auf ihre Werte $X_R = (x_{Rn}, \dots, x_{R2}, x_{R1})$

- Ausgabe



Wert des Ausgangsvektors $Y = (y_m, \dots, y_1)$ im aktuellen Zustand

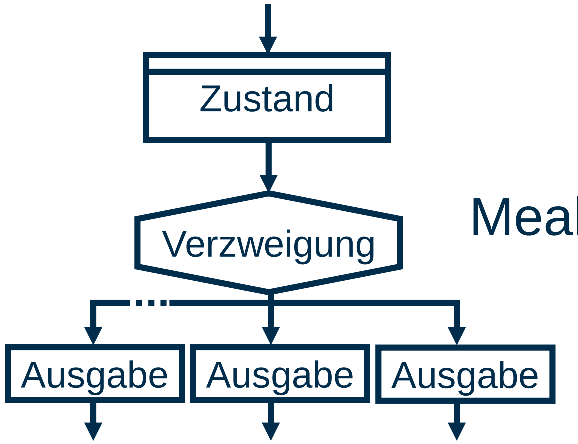
- Markierung des Anfangszustandes



Ablaufdiagramm

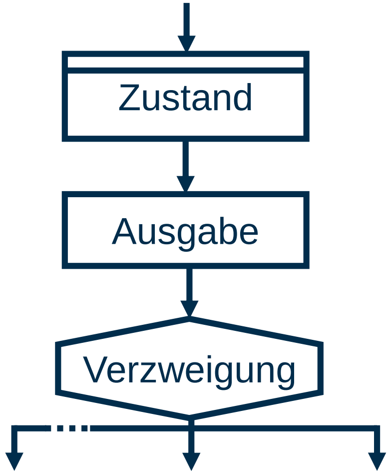
K. 7.6

- Mealy- und Moore-Schaltwerke werden entsprechend der Abhängigkeiten $Y^v = \lambda(Q^v, X^v)$ bzw. $Y^v = \lambda(Q^v)$ durch die Reihenfolge der Symbole für Verzweigungen und Ausgabe berücksichtigt
- **Definition des Ablaufschrittes**
  - Tripel aus **Zustandsübergang, Ausgabe und Verzweigung**, das jeweils zu einem Wert des Index v gehört



Mealy-Ablaufschritt

Moore-Ablaufschritt



Ablauftabelle

K. 7.6

- **Komplexere Schaltwerksbeschreibungen**:
  - Größere Zustandsanzahl
  - Es eignet sich eine tabellenorientierte Darstellung besser

- **Ablauftabellen** setzen sich aus Teiltabellen für jeden Ablaufschritt zusammen
  - Teiltabellen bestehen aus je vier Spalten
    - Für den **momentanen Zustand Q^v**
    - Für die Eingabeblöcke der jeweils **relevanten Eingabevariablen $X_R(k)$**
    - Für den **Folgezustand Q^{v+1}**
    - Für die **Ausgabeblöcke $Y(k)$**

Q^v	$X_R(k)$	Q^{v+1}	$Y(k)$
k			

- **Zustände** sind in **Ablauftabellen** i. A. nicht kodiert

Ablauftabelle

K. 7.6

- **Beispiel**:
  - I_1, I_2, I_3, I_4 seien Folgezustände zum Zustand I_1
  - Y_1 sei die Ausgabe gemäß Schaltwerkstyp (hier Moore)

Q^v	$X_R(3)$				Q^{v+1}	$Y(3)$
	x_6	x_4	x_3	x_1		
I_1	0	–	0	1	I_1	Y_1
	0	1	0	0	I_2	
	0	1	1	1	I_2	
	1	1	0	1	I_2	
	1	0	–	–	I_3	
	–	0	1	1	I_3	
	1	1	1	0	I_4	

Entwurf von Schaltwerken

K. 7.6

- Der **Schaltwerksentswurf** gliedert sich in **folgende Schritte**:

1. **Definition** der **Ein- und Ausgangsgrößen**
2. Wahl des **Schaltwerktyps** und Erstellen des **Ablaufdiagramms** bzw. der **Ablauftabelle** gemäß **Aufgabenstellung**
3. **Zustandskodierung**
4. **Wahl** des **FlipFlop-Typs** und Aufstellen der **Ansteuerfunktionen**
5. **Entwurf** des **Schaltnetzes** für die **Überführungsfunktion** auf Basis der Ansteuerfunktion
6. **Entwurf** des **Schaltnetzes** für die **Ausgabefunktion**
7. Eventuelle **Umformung** der **logischen Ausdrücke** in geeignete **Strukturausdrücke**
8. **Umsetzung** in das **Schaltbild** des Schaltwerks

Entwurf von Schaltwerken

K. 7.6

- **Beispiel**: Verbale Aufgabenstellung
  - Eine Schaltung für einen sehr einfachen Anrufbeantworter soll zunächst ein Klingelzeichen des Telefons abwarten.
  - Wenn bis zum Beginn des zweiten Klingelzeichens der Hörer nicht abgenommen wurde, soll der Anrufbeantworter eingeschaltet werden.
  - Dieser spielt dann seine Mitteilung ab, egal wie lange der Anrufer tatsächlich zuhört.
  - Erst wenn das Band vollständig abgespielt ist, wird der Anrufbeantworter wieder ausgeschaltet.
  - Der Ausschaltvorgang beinhaltet ein automatisches Rückspulen des Bandes

Entwurf von Schaltwerken

K. 7.6

1. Variablen

- **Eingangsvariablen**:
  - Klingelzeichen: $K = 1$
  - Klingelpause: $K = 0$
  - Timeout: $T = 1$
  - Kein Timeout: $T = 0$
  - Bandende: $B = 1$
  - Sonst: $B = 0$
- **Ausgangsvariablen**:
  - Anrufbeantworter eingeschaltet $A = 1$
  - Anrufbeantworter ausgeschaltet $A = 0$

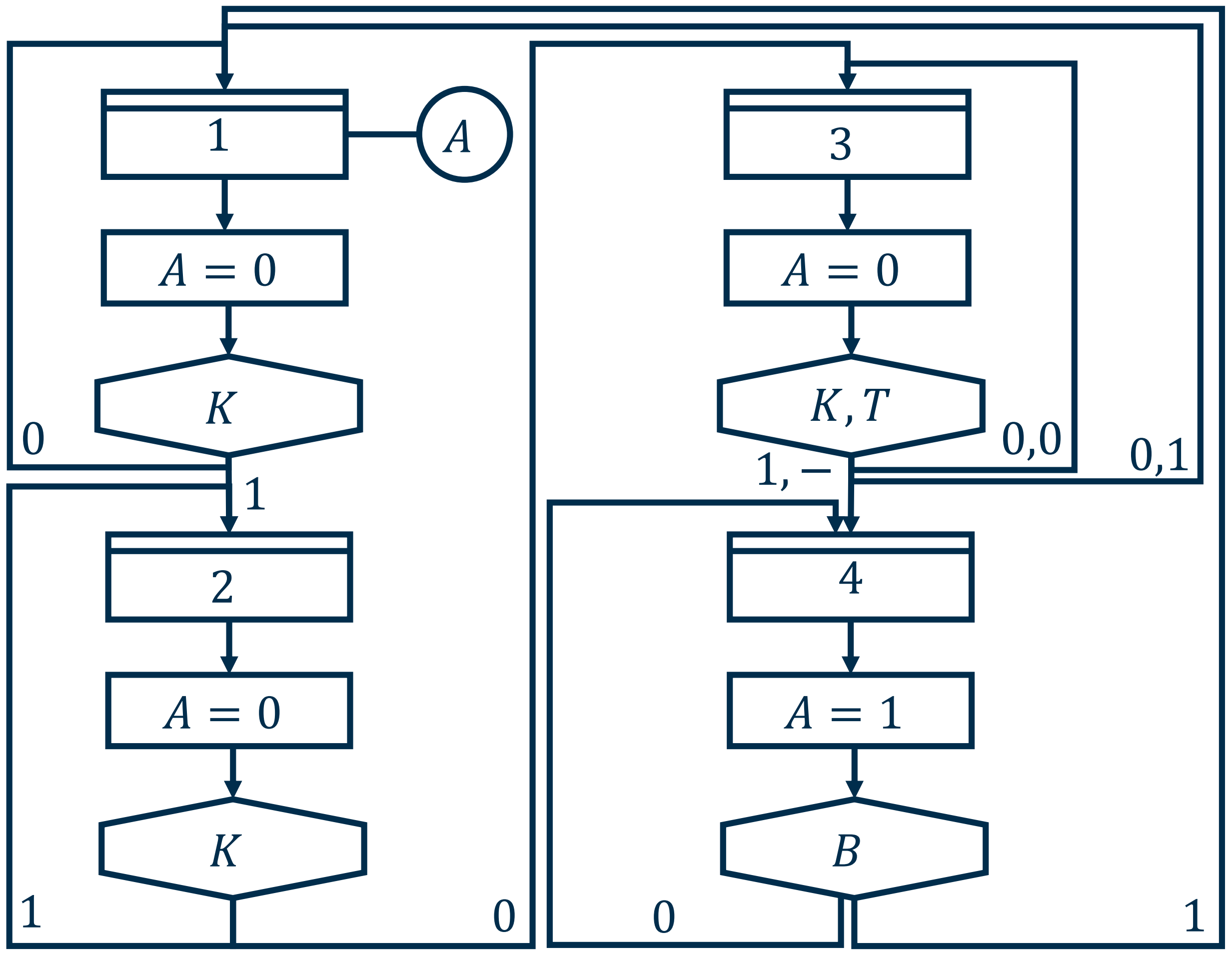
Timeout: Klingelpause dauert zu lange, Anrufer hat nach dem ersten Klingeln aufgelegt

Entwurf von Schaltwerken

K. 7.6

2. Ablaufdiagramm

- **Schaltwerkstyp**: Moore



Entwurf von Schaltwerken

3. Zustandskodierung

- Für **4 Zustände** werden **2 Zustandsvariablen** (q_2, q_1) benötigt.
- Die Kodierung wird willkürlich, z.B. als Zählerkodierung ausgeführt

	q_2	q_1
Zustand 1	0	0
Zustand 2	0	1
Zustand 3	1	0
Zustand 4	1	1

Entwurf von Schaltwerken

K. 7.6

4. Ablauftabelle

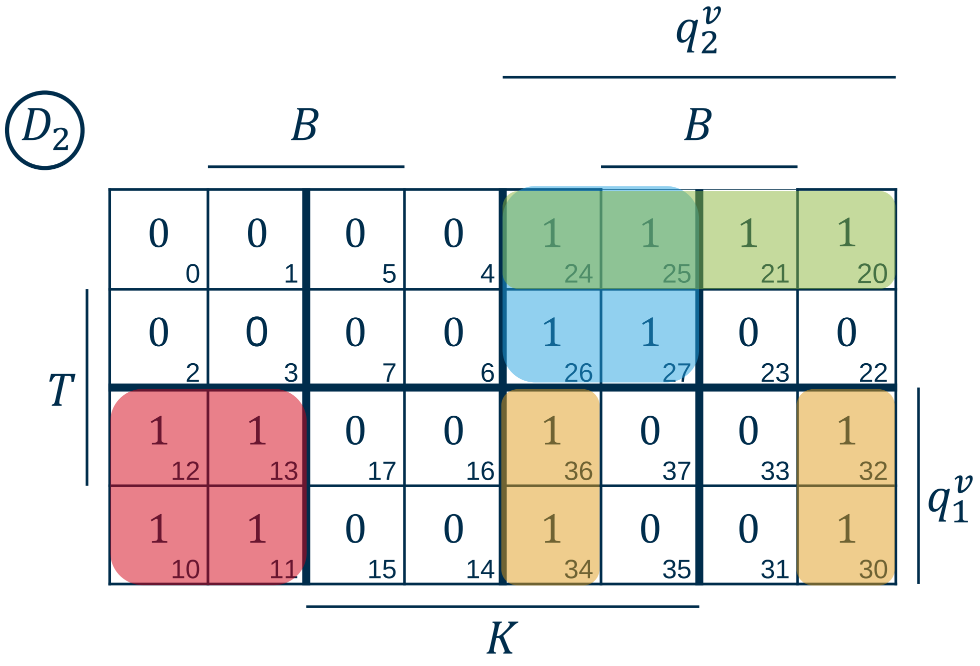
- Es sollen **D-FlipFlops** verwendet werden, d.h. $q_2 \mathrel{\widehat{=}} D_2$ und $q_1 \mathrel{\widehat{=}} D_1$

$\boldsymbol{Q^v}$				$\boldsymbol{Q^{v+1}}$		$\boldsymbol{Y}$
q_2^v	q_1^v			q_2^{v+1}	q_1^{v+1}	A
		$x_R(1): K$				
0	1	0		1	0	0
		1		0	1	
		$x_R(2): K$				
0	1	0		1	0	0
		1		0	1	
		$x_R(3): K, T$				
1	0	0	0	1	0	0
		1	–	1	1	
		0	1	0	0	
		$x_R(4): B$				
1	1	0		1	1	1
		1		0	0	

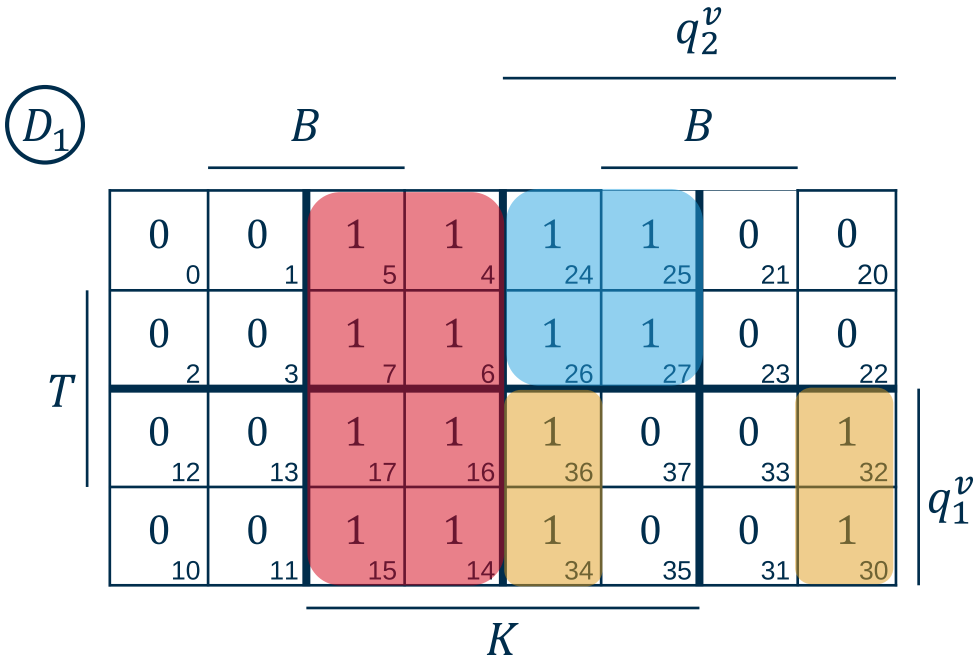
Entwurf von Schaltwerken

K. 7.6

5. Symmetriediagramme für die Ansteuerfunktion



$$D_2 = \overline{q_2^v} q_1^v \overline{K} \vee q_2^v \overline{q_1^v}\,\overline{T} \vee q_2^v \overline{q_1^v} K \vee q_2^v q_1^v \overline{B}$$



$$D_1 = \overline{q_2^v} K \vee q_2^v \overline{q_1^v} K \vee q_2^v q_1^v \overline{B}$$

Entwurf von Schaltwerken

K. 7.6

6. Ausgabefunktion

- Aus der Ablauftabelle folgt sofort $A = q_2^v q_1^v$

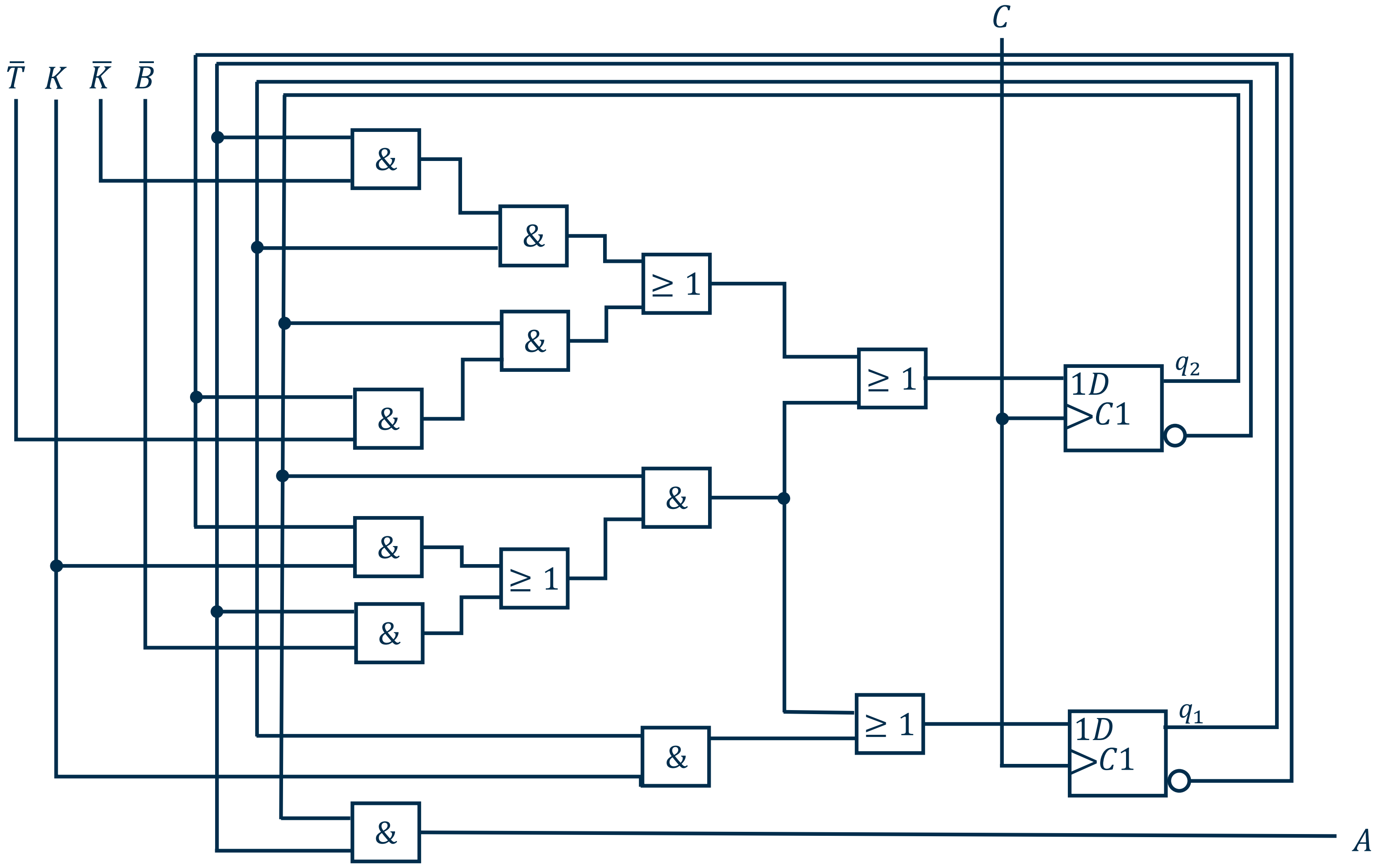
7. Umformungen

- Wenn zum Beispiel ausschließlich Gatter mit 2 Eingängen zur Verfügung stehen, müssen entsprechende Umformungen vorgenommen werden:
  - $D_2 = \left[\overline{q_2^v}\left(q_1^v \overline{K} \right) \vee q_2^v\left(\overline{q_1^v}\overline{T}\right)\right] \vee q_2^v\left(\overline{q_1^v} K \vee q_1^v \overline{B}\right)$
  - $D_1 = \overline{q_2^v} K \vee q_2^v\left(\overline{q_1^v} K \vee q_1^v \overline{B}\right)$
  - $A = q_2^v q_1^v$

Entwurf von Schaltwerken

K. 7.6

8. Blockschema (Schaltbild) des Schaltwerkes



Entwurf von Schaltwerken

K. 7.6

8. Blockschema (Schaltbild) des Schaltwerkes

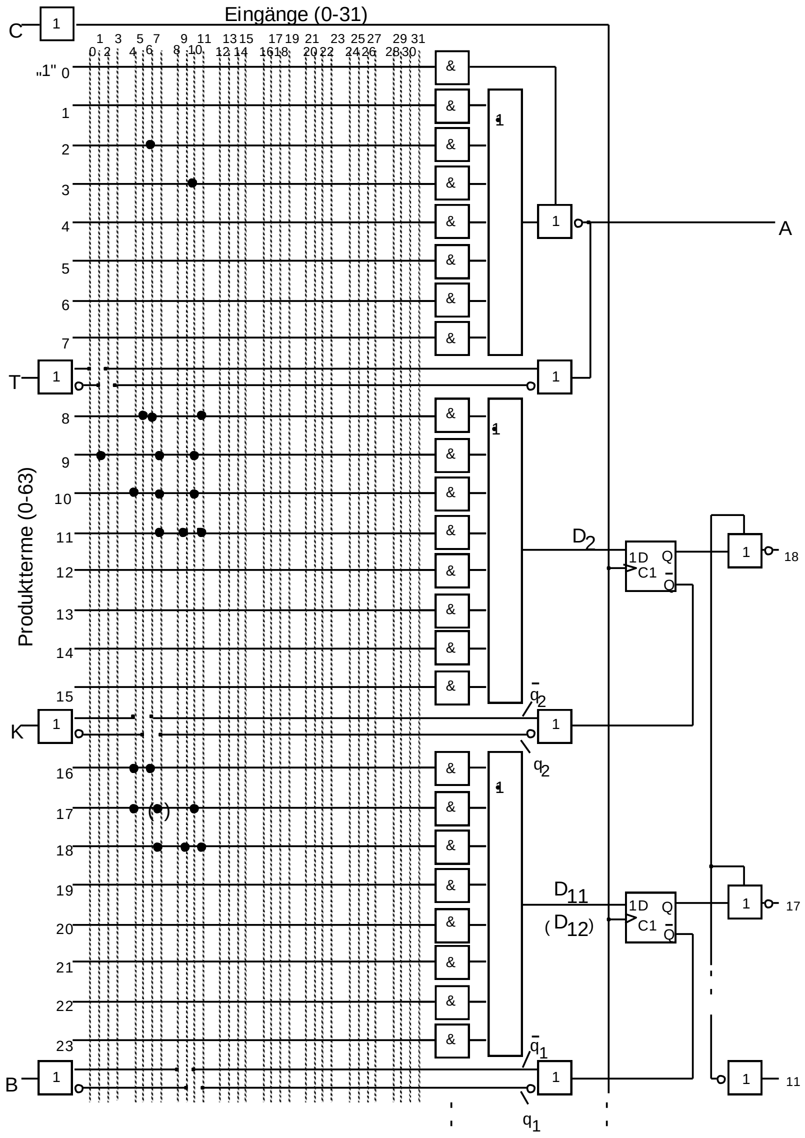
- Es besteht die Möglichkeit, zweistufige Logik in programmierbare **integrierte Schaltungen** einzubetten:
  - Kann ebenso bei Schaltwerken geschehen
  - Anhand des Bausteins AmPAL16R6 soll dies für das Beispiel gezeigt werden
  - Wegen negierter Ausgänge des Bausteins muss unsere Lösung umgeformt werden:
    - $A = q_2^v q_1^v = \overline{\overline{q_2^v q_1^v}} = \overline{\overline{q_2^v} \vee \overline{q_1^v}}$
  - Ansteuerfunktionen für D_1 und D_2 können Schritt 5 entnommen werden

Entwurf von Schaltwerken

K. 7.6

8. Blockschema (Schaltbild) des Schaltwerkes

- Einbettung in einen AmPAL16R6



Poster „Automaten“

- Zusammenfassung des Kapitels als Poster

- Verfügbar auf Ilias: