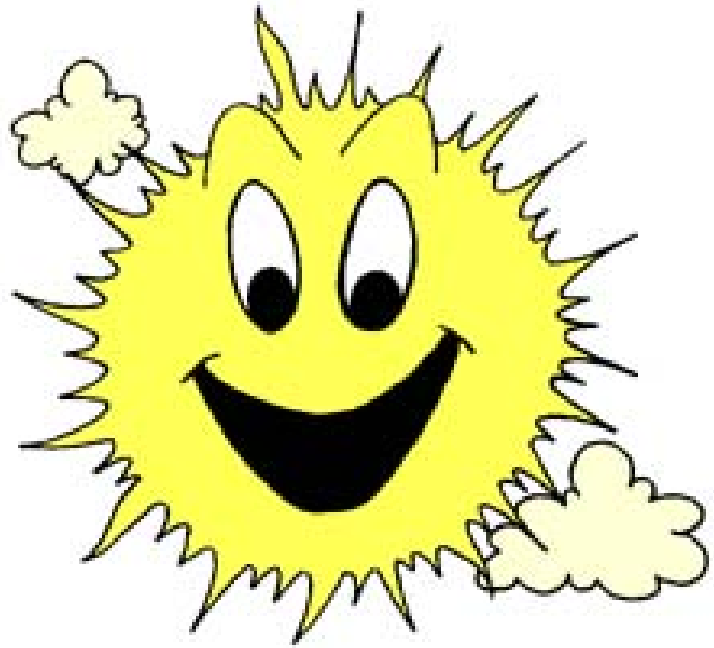
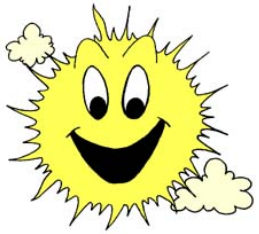
Wiederholung der 8. Vorlesung



Ende der Wiederholung



Bemerkung zum Stoff

Der folgende Foliensatz enthält in der Vorlesung teilweise nicht detailliert besprochene Folien

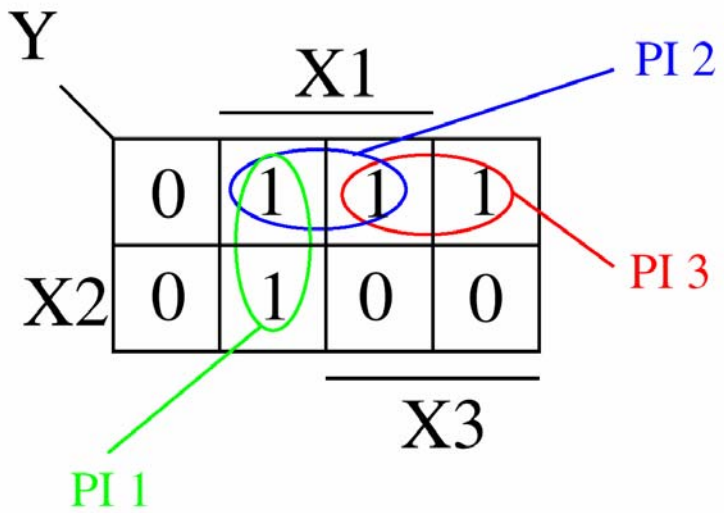
- Vertiefungen zum Thema Hazards. Bearbeiten Sie sie diese **zur Übung** zu Hause und stellen Sie ggf. zu Beginn der nächsten Vorlesung Fragen
- Details zu CPLD‘s und FPGA‘s zur Vertiefung für Interessierte.
- Vorausschau auf **Zustandsgraphen** und **Zustandsfolgetabellen**. Machen Sie sich als Vorbereitung für die nächste Vorlesung mit der Thematik vertraut.

12. Hazards

- Kurzzeitige und unerwartete Änderungen der Werte auf Signalleitungen.
- Eine Schaltung, die eine Gefahr (**Hazard**) enthält, hat das Potenzial einen Störimpuls von kurzer Dauer (**Glitch**) zu produzieren.
- Hazards können zu instabilem Verhalten in Schaltungen führen und müssen daher schon beim Entwurf vermieden werden.

12.1 Logik-Hazards – Beispiel

X3	X2	X1	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



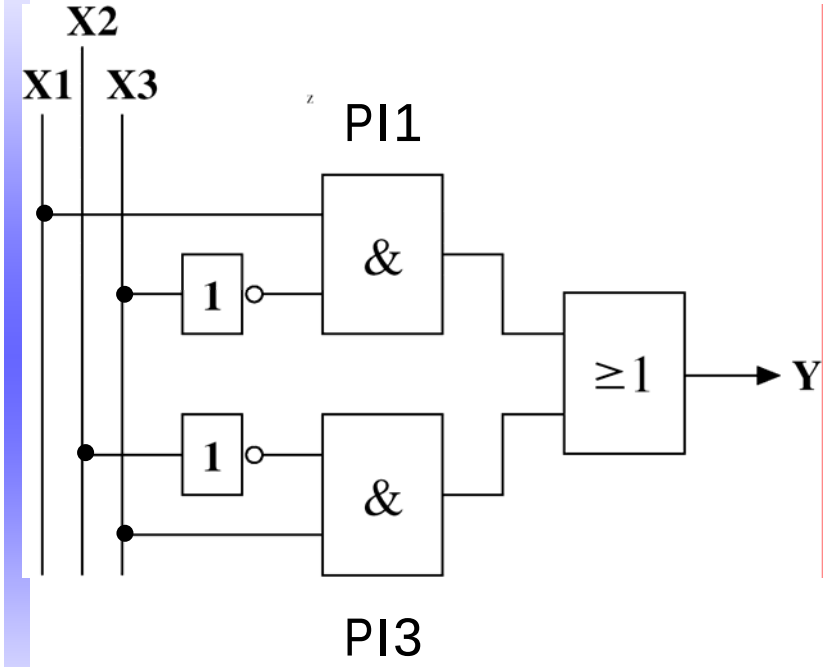
$$PI1 = X1 \wedge \overline{X}3$$

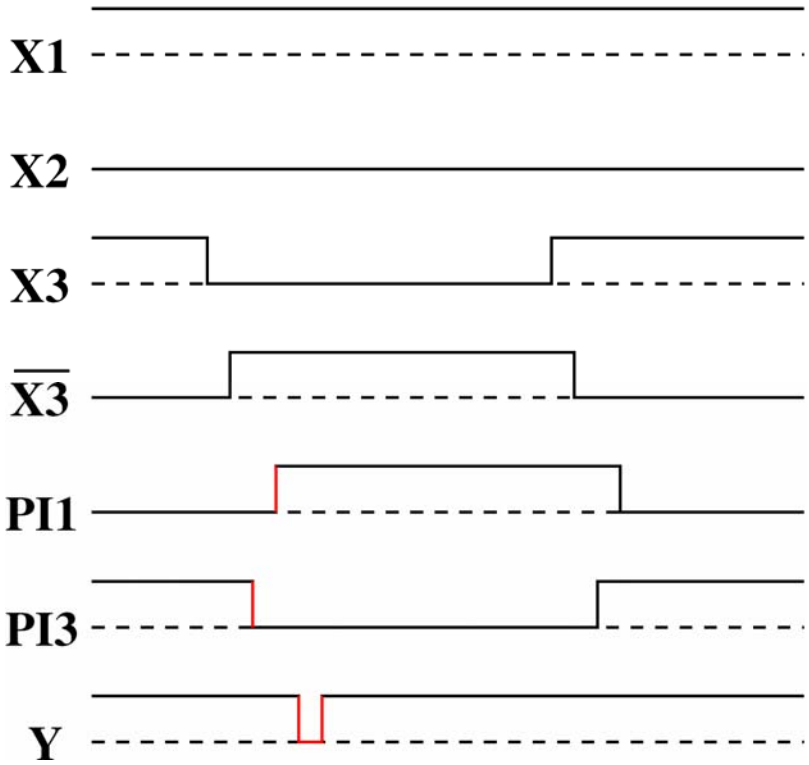
$$PI2 = \overline{X}2 \wedge X1$$

$$PI3 = \overline{X}2 \wedge X3$$

Übergang von 101 zu 001

12.1 Logik-Hazards – Beispiel (2)

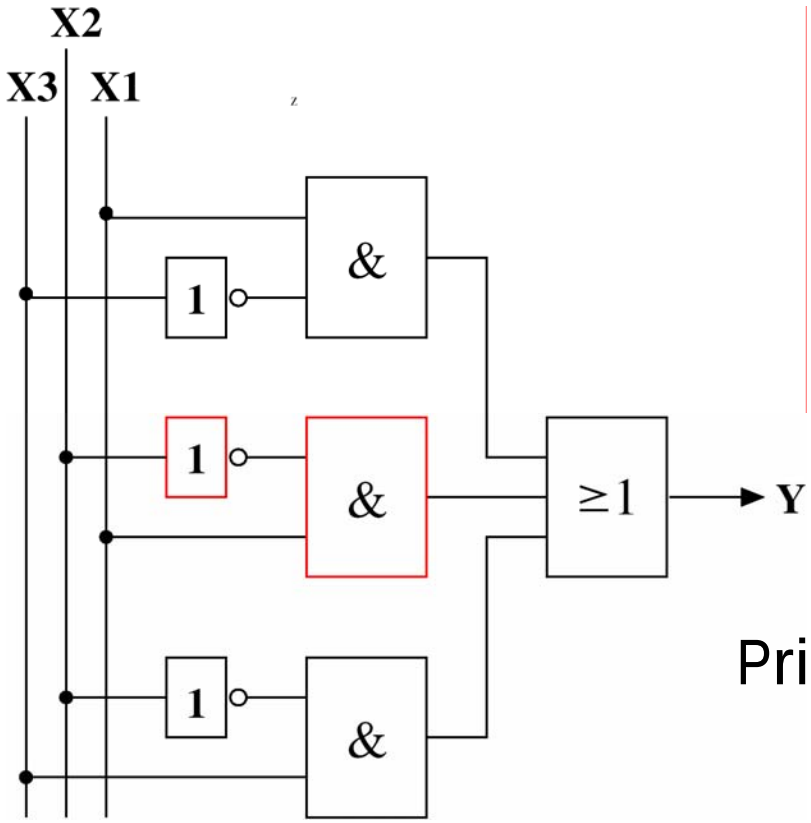


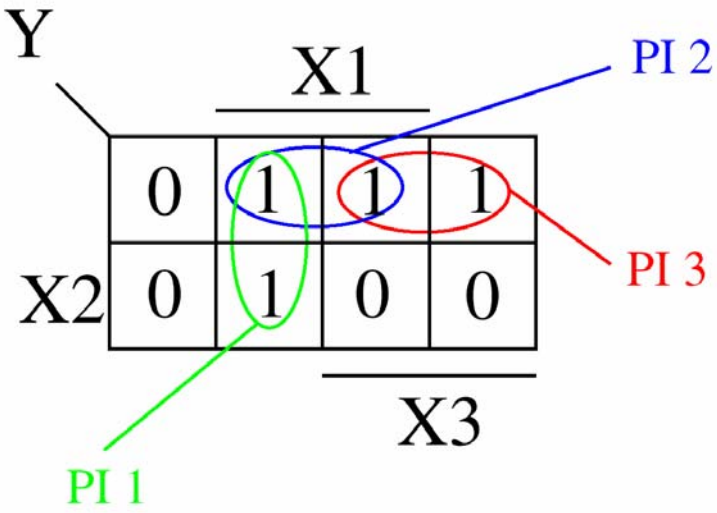


Übergang von 101 zu 001

Annahme (Laufzeiten):
und/oder: 4ns
Negation: 2ns

12.1 Logik-Hazards – Beispiel Lösung





Übergang von 101 zu 001

Primimplikant 2 wird (trotz Redundanz) schaltungstechnisch realisiert.

Logik-Hazards werden auch als kombinatorische Hazards bezeichnet.

12.1 Vermeidung von Logik-Hazards

- Erzeugung Minimalform.
- Wenn bei dem Übergang einer Komponente der aktive Primimplikant gewechselt wird, kann ein redundanter Primimplikant hinzugefügt werden, so dass der Übergang eliminiert wird.
- Einsatz von getakteten Schaltungen.

12.2 Funktions-Hazards

- werden auch als **Race** (Wettlauf) bezeichnet.
- können auftreten, wenn sich mehr als ein Eingang gleichzeitig ändert (Mehrkomponenten-Übergang)
- Das Ergebnis hängt von dem Ausgang des Wettlaufs ab.
- Dies kann zu Fehlfunktionen im System führen.

12.2 Vermeidung von Funktions-Hazards

- Überführung von Mehrkomponenten-Übergängen in mehrere Einkomponentenübergänge (Änderung einer einzelnen Eingangsvariablen)
- Einsatz von getakteten Schaltungen

12.2 Zusammenfassung – Fazit: Hazards

- Bei komplexen asynchronen Schaltungen ist die Wahrscheinlichkeit für Hazards hoch.
- Analyse von asynchronen Schaltungen bezüglich Hazards ist aufwändig.

- In der Praxis dominiert vor diesem Hintergrund der Entwurf von synchronen Schaltungen

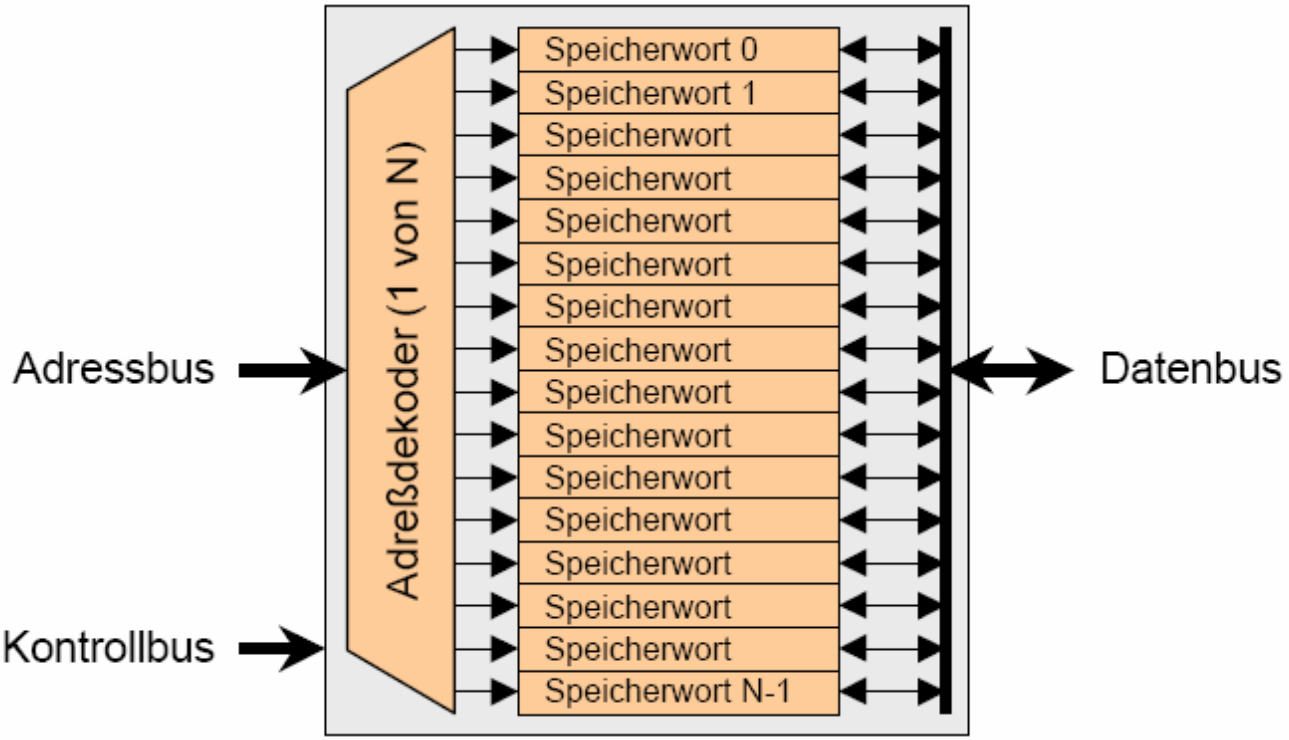
16.6 Speicherarchitekturen

Bei Speichern unterscheidet man zwischen

- expliziter
- impliziter

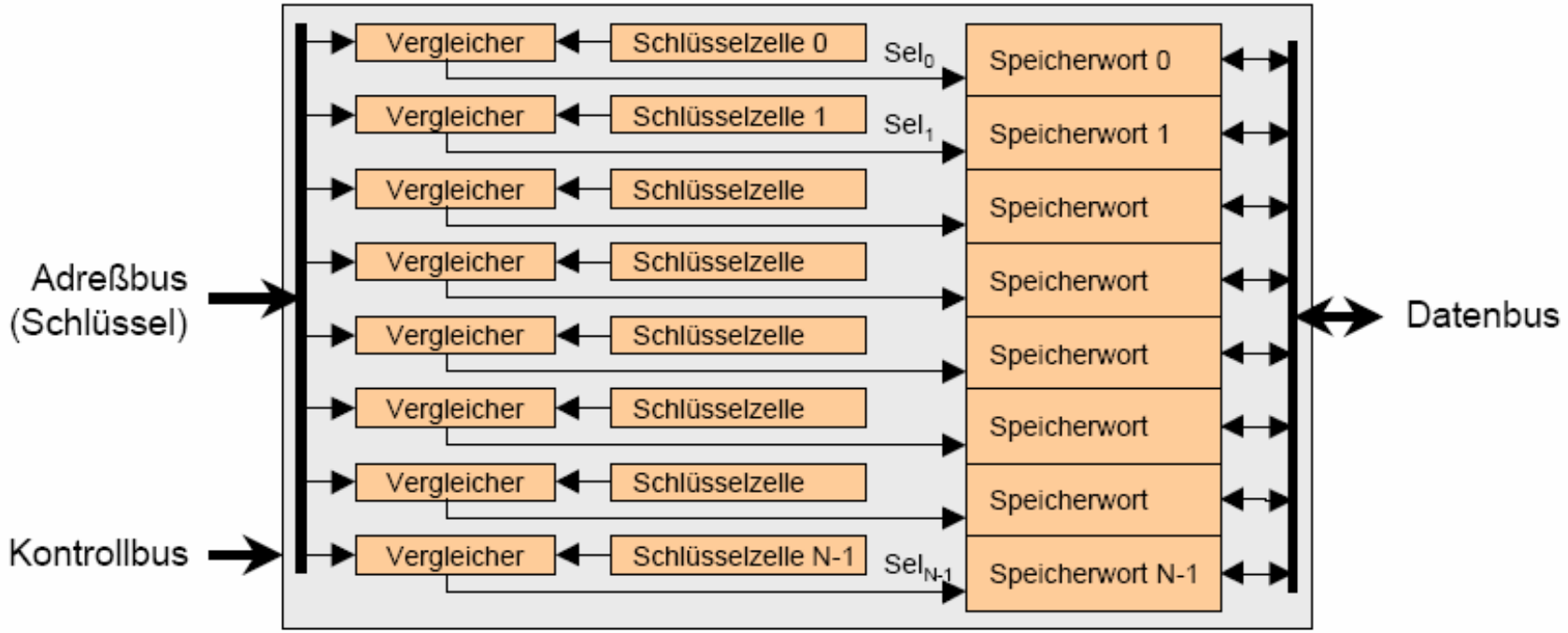
Adressierung

16.6.1 Explizite Adressierung



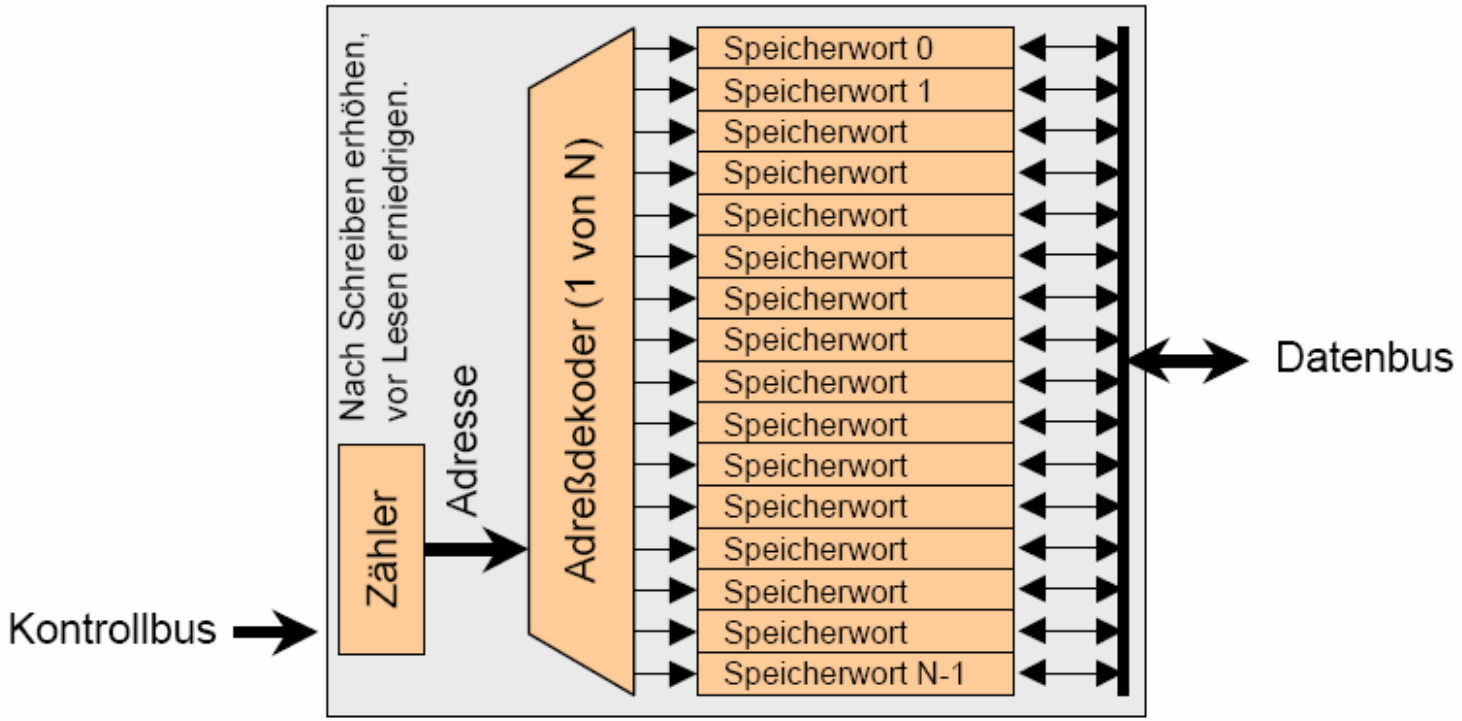
- Die Adresse besitzt einen Wertebereich von 0 bis N-1, es können also N Werte gespeichert werden.
- Speichertyp mit fester Datenwortlänge beispielsweise als Hauptspeicher bei PC‘s.

16.6.2 Assoziativspeicher

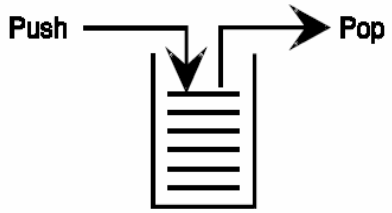


- Explizite Adressierung
- Auswahl über unterscheidbare Schlüssel
- Schlüssel bilden keinen kontinuierlichen Adressraum.
- Bei fester Speicherwortlänge z.B in Cache-Speichern

16.6.3 Implizite Adressierung



- Datenwörter werden aufeinandergestapelt (Stapelspeicher / Stack)



- Speicher endlicher Größe
- Behandlung von Über- und Unterlauf zu klären.

13. Realisierung digitaler Lösungen

Eine Schaltung soll in gewisser Stückzahl realisiert werden.

Dann gibt es aus technischer und kommerzieller Sicht mehrere Ansätze:

- Full Custom IC
- ASIC (**A**pplication **S**pecific **I**ntegrated **C**ircuit)
- Bausteine mit programmierbarer Logik

13. Full Custom IC

- individuelle Entwicklung eines digitalen Systems
- lange Entwicklungszeiten
- sehr große Stückzahlen
- individuelle Fertigung
- geringer Stückpreis

13. Application Specific IC (ASIC)

- Chip entsteht auf Basis existierender Funktionsblöcke
- Weniger aufwändig als Neudesign des Chip
- Hersteller stellt umfangreiche Bibliotheken für Funktionen zur Verfügung
- verkürzte Entwicklungszeiten
- Schaltung wird mit einer Hardware-Beschreibungssprache beschrieben
- Hersteller realisiert Schaltung auf der Basis eines adäquaten ASIC
- große Stückzahlen
- günstiger Stückpreis

13. Programmierbare Bausteine

- Hersteller bieten programmierbare Logik-Bausteine an
- Lösung wird vom Anwender entwickelt
- hohe Flexibilität
- kleine Stückzahlen
- hoher Stückpreis

13. Programmierbare Bausteine

- PLD (Programmable Logic Device)
- programmierbare Logikelemente (seit Mitte der 70er)
- PLD stellen eine logische Grundstruktur zur Verfügung, die vom Entwickler nach Bedarf konfiguriert (programmiert) werden kann.
- Für hoch integrierte PLD stehen Beschreibungssprachen zur Verfügung.

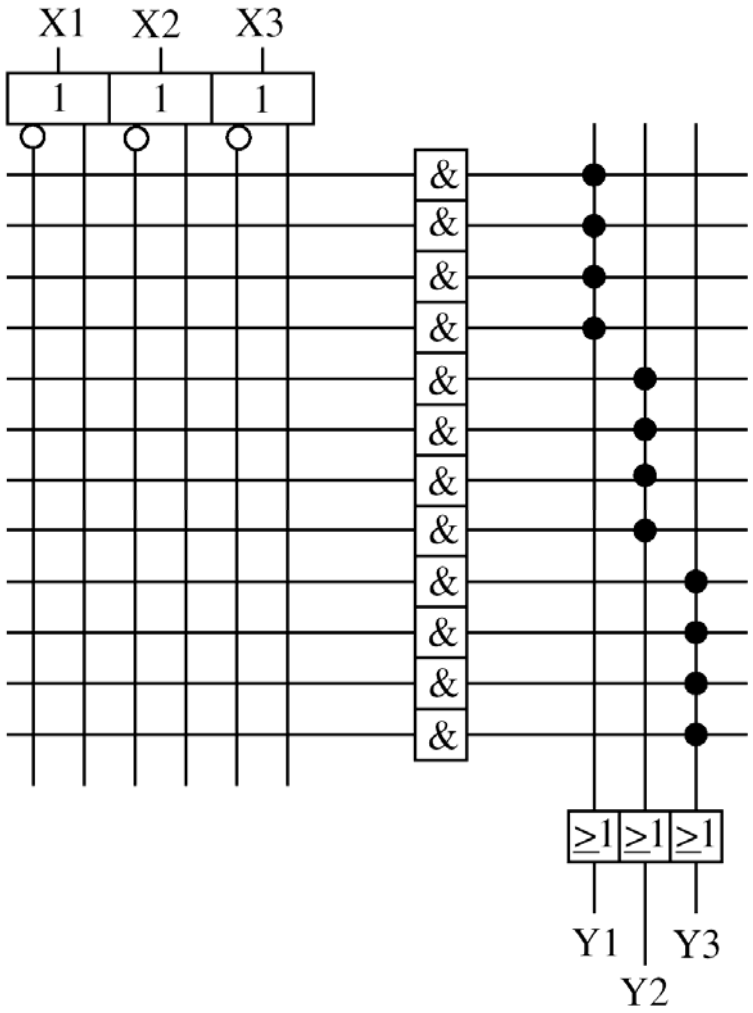
13. Verfahren zur Programmierung

- PROM (Programmable Read Only Memory)-Prinzip: Durchbrennen einer Sicherung (Fuse) oder Entferner einer Isolierung (Antifuse), Programmierung ist irreversibel
- EPROM (Erasable PROM)-Prinzip: Programmierung kann durch Bestrahlung mit UV-Licht wieder gelöscht werden
- EEPROM (Electrical Erasable PROM)-Prinzip: Programmierung kann durch elektrische Impulse wieder gelöscht werden

13.1 PAL (Programmable Array Logic)

- Realisierung logischer Gleichungen in disjunktiver Form.
- Alle Eingangsgrößen werden in negierter und nicht-negierter Form zur Verfügung gestellt.
- Programmierbares UND-Feld das mit den Eingangsgrößen verbunden ist.
- Fest verdrahtetes ODER-Feld.

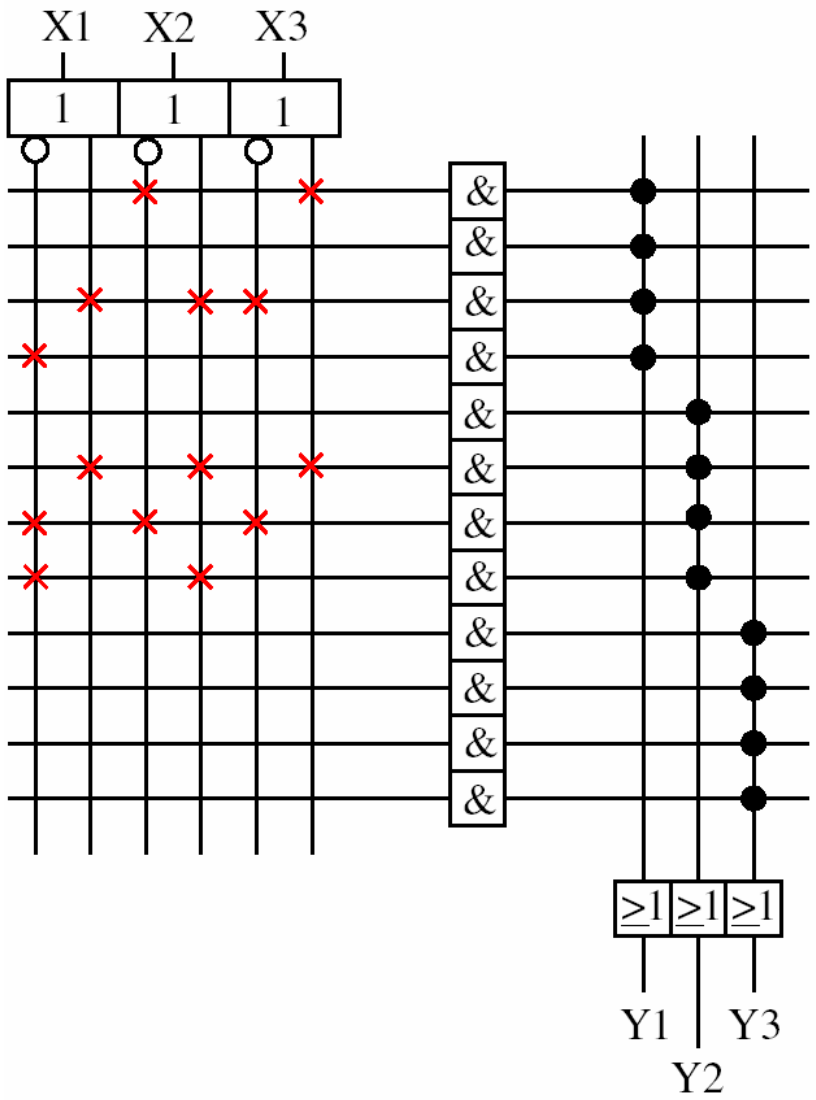
13.1 Prinzip PAL



Frei programmierbare UND-GATTER

Fest verschaltete ODER-Gatter

13.1 Beispiel PAL



$$Y1 = (\bar{X}2 \wedge X3) \vee (X1 \wedge X2 \vee \bar{X}3) \vee \bar{X}1$$

$$Y2 = (X1 \wedge X2 \wedge X3) \vee$$

$$(\bar{X}1 \wedge \bar{X}2 \wedge \bar{X}3) \vee$$

$$(\bar{X}1 \wedge X2)$$

13. Typisierung

- PAL (Programmable Array Logic):

  Programmierbare UND-Matrix, feste Oder-Matrix, von einem Hersteller auch als GAL (Generic Array Logic) bezeichnet
- PLE (Programmable Logic Element):

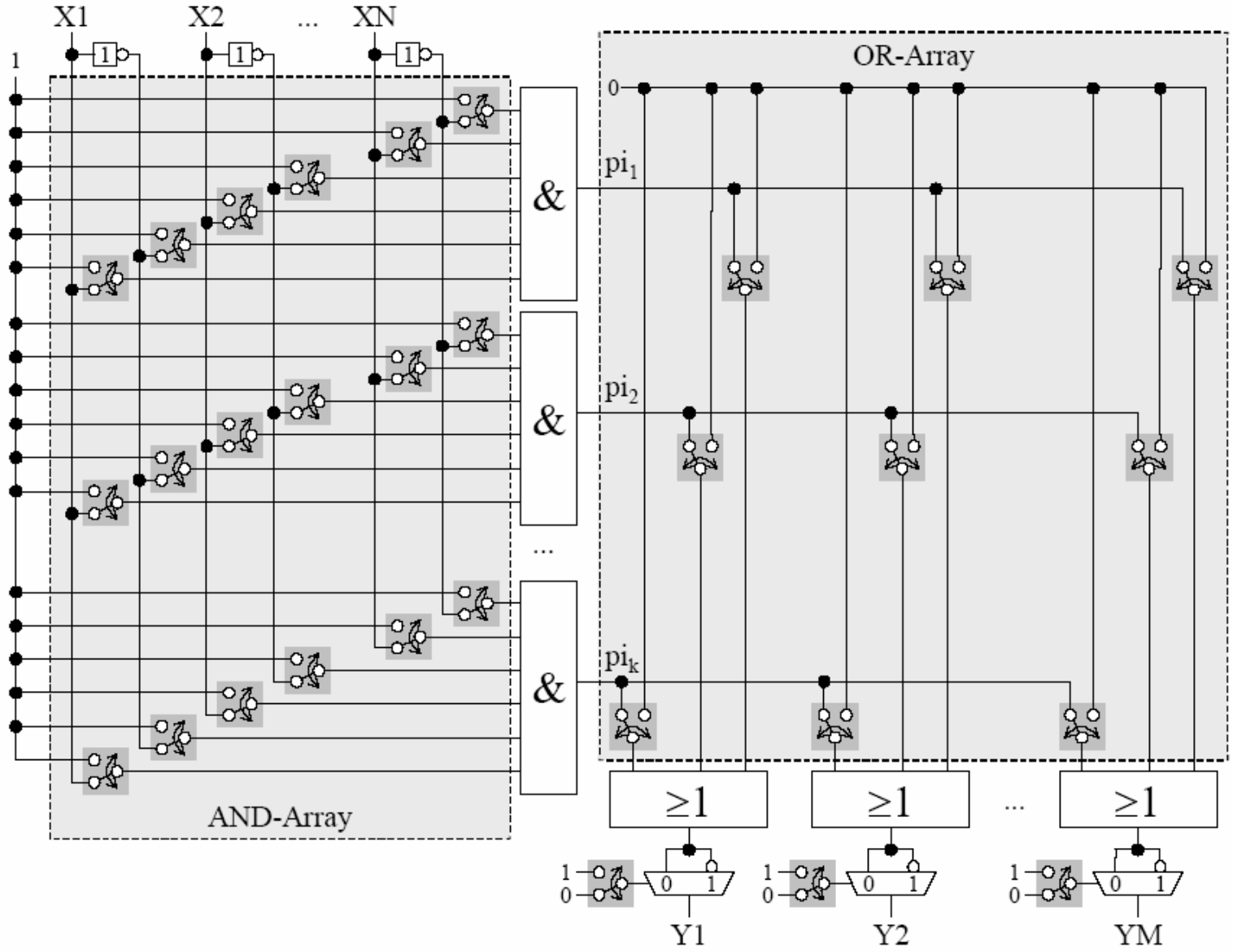
  Programmierbare Oder-Matrix, feste Und-Matrix
- PLA (Programmable Logic Array):

  Programmierbare UND-Matrix und programmierbare ODER-Matrix

13.2 PLA (Programmable Logic Array)

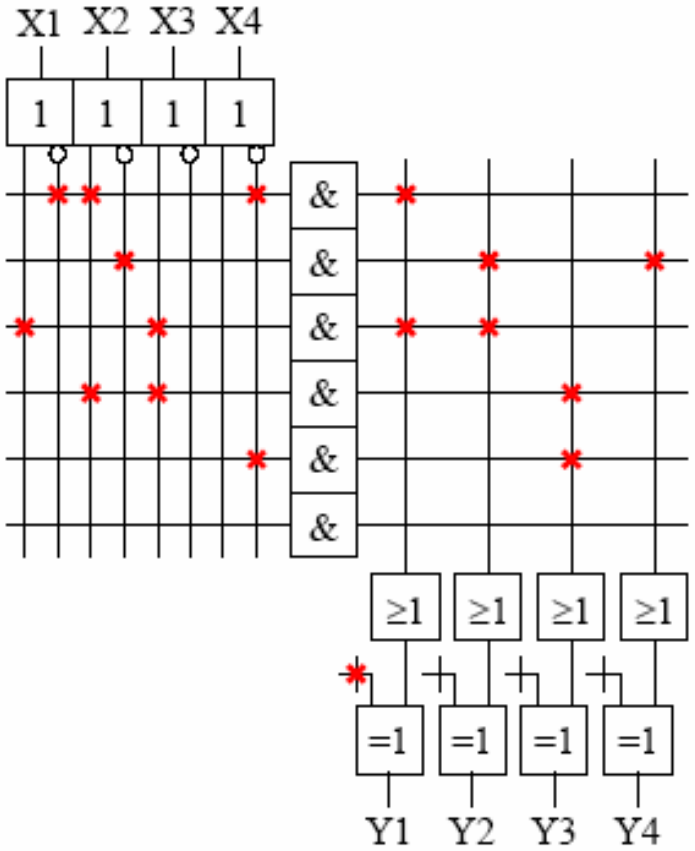
- Mit dem frei programmierbaren AND- und OR-Array können Funktionen in disjuktiver Normalform realisiert werden.
- Durch die programmierbaren Ausgangsinverter ist aufgrund des Shannonschen Gesetzes auch die Realisierung der konjunktiven Form möglich.

13.2 PLA (Programmable Logic Array) (2)



13.2 PLA (Programmable Logic Array) (3)

Üblicherweise wird diese logische Grundstruktur in einer kompakten Form dargestellt, wo bei jedem Gatter eine Linie die Gattereingänge symbolisiert.



$$Y1 = \overline{(\overline{X1} \wedge X2 \wedge \overline{X4}) \vee (X1 \wedge X3)}$$

$$= (X1 \vee \overline{X2} \vee X4) \wedge (\overline{X1} \vee \overline{X3})$$

$$Y2 = \overline{X2} \vee (X1 \wedge X3)$$

$$Y3 = (X2 \wedge X3) \vee \overline{X4}$$

$$Y4 = \overline{X2}$$

Implikanten können in mehreren Funktionen gemeinsam genutzt werden.

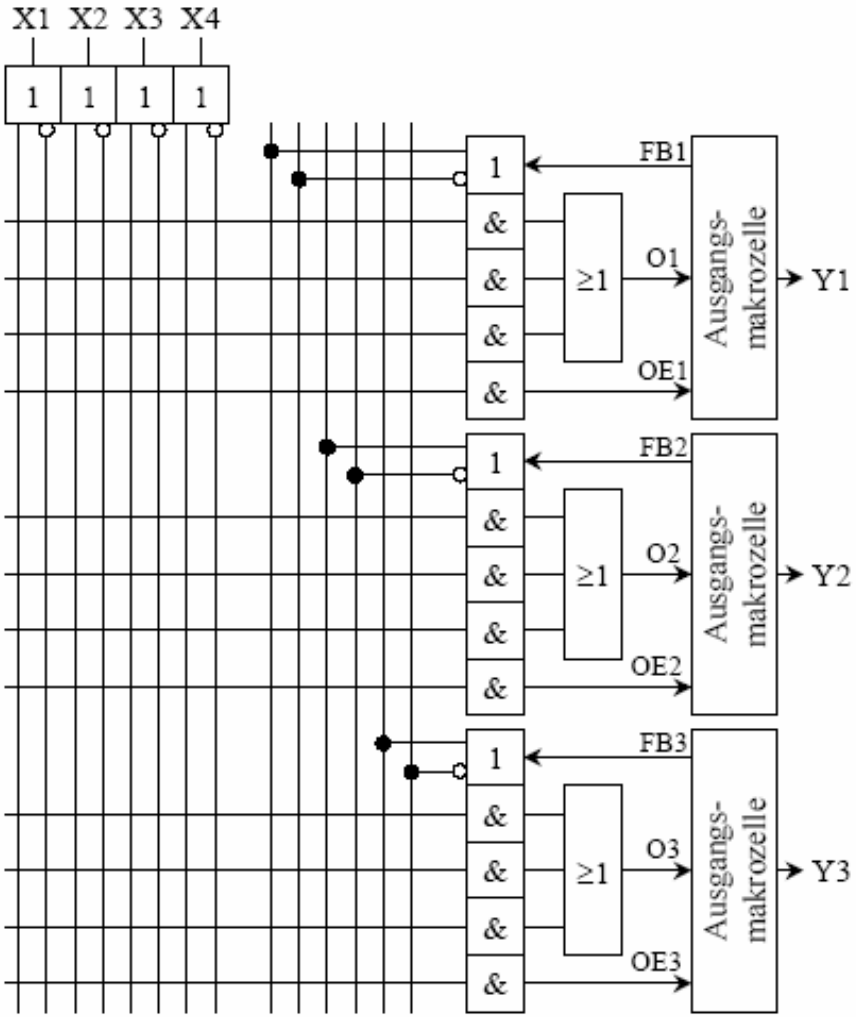
13.2 PLA (Programmable Logic Array) (4)

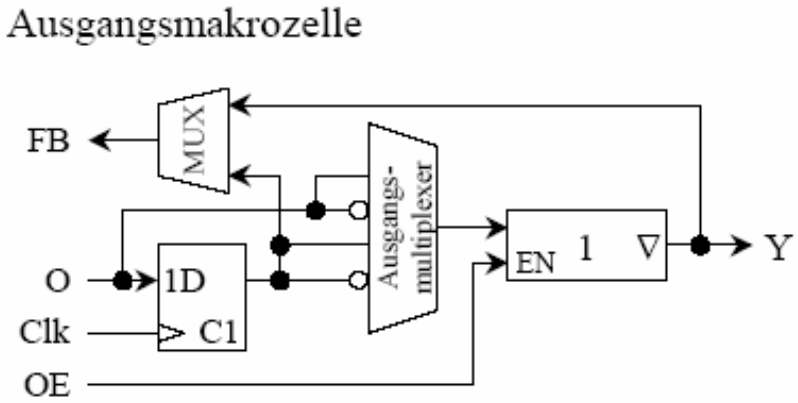
- Die Verwendung von PLA-Bausteinen ist jedoch eher selten.
- Die gebräuchlichen PAL- und Speicher-Bausteine schränken die Möglichkeiten der Programmierung auf ein sinnvolles Maß ein, so dass deren Verwendung gut handhabbar ist.

13.1 Erweiterung von PAL

- Einfache PAL Elemente haben mindestens 8 Ein- und Ausgänge
- Moderne PAL Bausteine verfügen über komplexe, programmierbare Makrozellen
- die Ausgänge verfügen über Register
- die Ausgänge können zurück gekoppelt werden

13.1 PAL, erweiterte Struktur

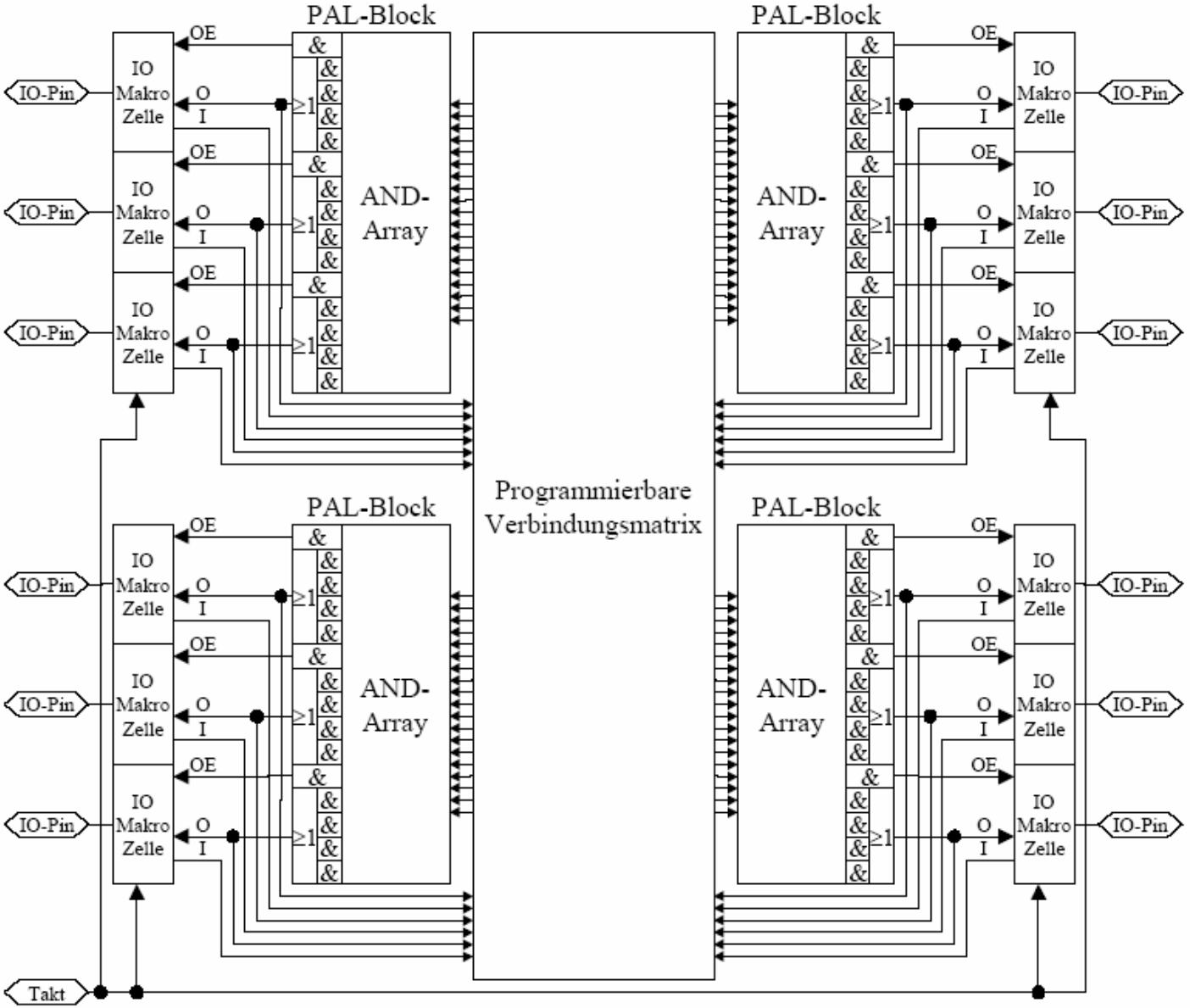




13.3 CPLD (Complex Programmable Logic Device)

- komplexe PLDs mit einer Block-Struktur
- jeder Block entspricht einem einfachen PAL
- die Blöcke werden über eine programmierbare Schaltmatrix miteinander verbunden
- ein einzelner Block enthält typischerweise ca. 50 Eingänge und 10-20 Ausgänge
- jeder Ausgang kann aus 10-15 Produkttermen gebildet werden

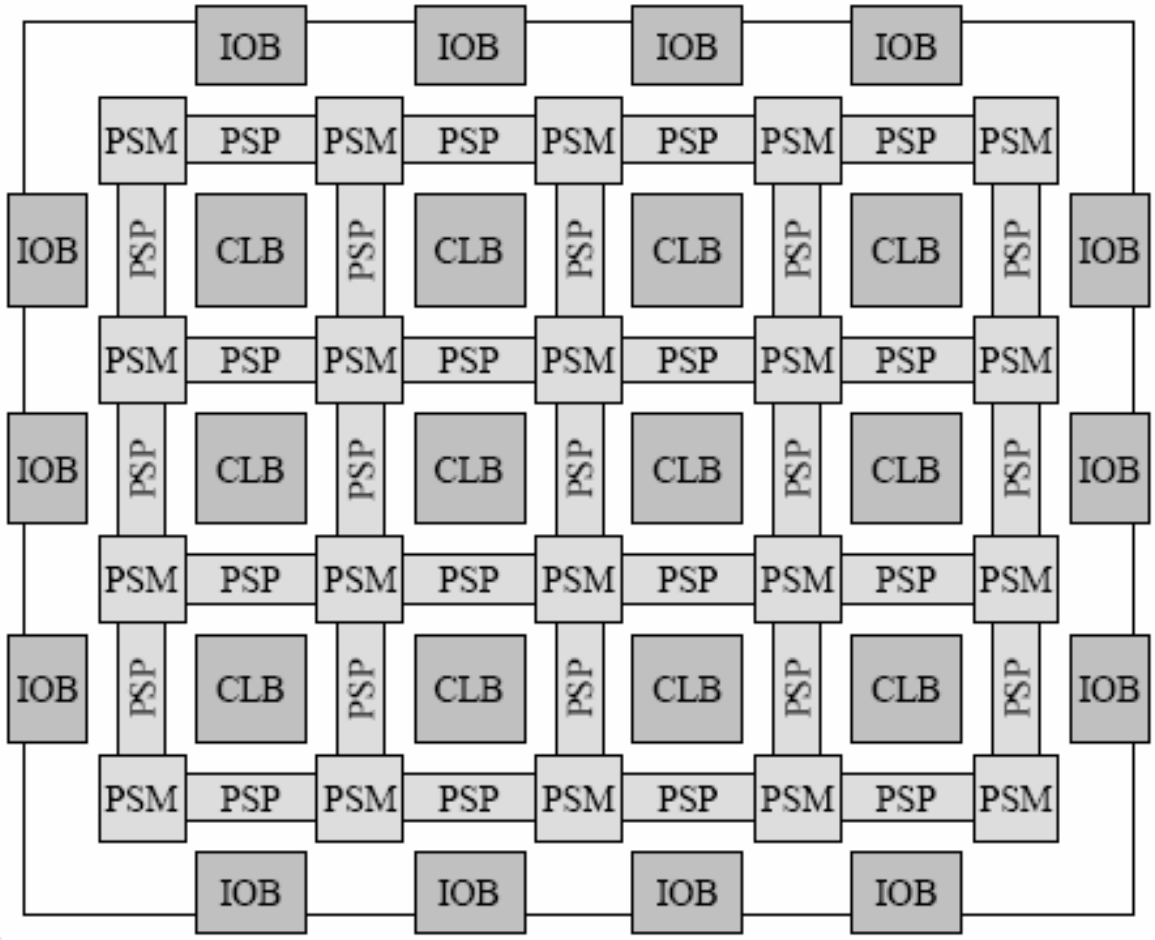
13.3 CPLD (2)


PAL-Block
PAL-Block
PAL-Block
PAL-Block
AND-Array
AND-Array
AND-Array
AND-Array
Programmierbare Verbindungsmatrix
IO Makro Zelle
IO-Pin
OE
O
I
&
≥1
Takt

15 FPGA (Field Programmable Gate Array)

- frei programmierbarer Logikschaltkreis
- aus einzelnen Logikblöcken (CLBs Configurable Logic Blocks) aufgebaut
- in den einzelnen Blöcken werden einfache Operationen und auch Flip-Flop-Logik zur Verfügung gestellt
- teilweise werden FPGAs ausschließlich über Look-Up Tabellen (LUT) realisiert
- hohe Komplexität
- Selbstkonfigurierende Systeme werden möglich

15 FPGA (2)



CLB: Configurable Logic Block

PSP: Programmable Switching Points

PSM: Programmable Switching Matrix

IOB: I/O Block

13+15 Vergleich CPLD und FPGA

CPLD	FPGA
Wenige Logikblöcke mit großer Anzahl an Makrozellen	Viele Logikblöcke mit kombinatorischer Logik
Kurze Wege	Lange Wege
Platzierung und Routing fest vorgegeben	Platzierung und Routing variabel
Schaltzeiten einfach vorhersagbar	Schaltzeiten sind von der Größe des Designs sowie Platzierung und Routing abhängig
Hohe Taktfrequenzen unabhängig von der konkreten Schaltung	Taktfrequenz ist von der Größe der Schaltung abhängig
Kleine und mittelgroße Schaltungen	Für sehr komplexe Schaltungen geeignet

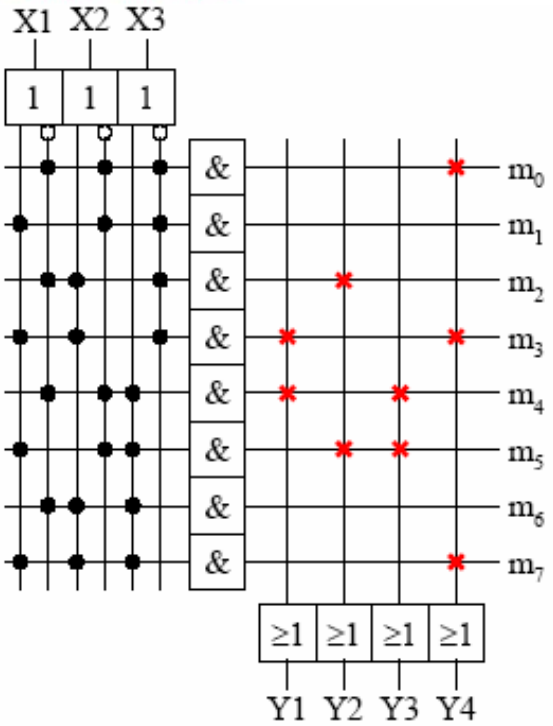
13+15 VHDL

- VHSIC (Very High Speed Integrated Circuit)
- VHDL (VHSIC Hardware Description Language)
- ursprünglich für Zwecke der Dokumentation entwickelt
- kommt auch für Schaltungssynthese zum Einsatz
- hat sich zwischenzeitlich als Hardwarebeschreibungssprache durchgesetzt
- Alternative ist bspw. Verilog

14. Speicherbausteine als programmierbare Logik

- Bei Speicherbausteinen ist das AND-Array fest programmiert.
- besitzen bei n Eingängen genau $N=2^n$ UND-Gatter.
- Mit der festen Programmierung des AND-Arrays stehen alle Minterme zur Verfügung
- durch Programmierung des OR-Arrays mit jedem ODER-Gatter eine beliebige boolesche Funktion erzeugt werden.
- Bausteine mit nicht-flüchtigen OR-Arrays: PROM (**P**rogrammable **R**ead-**O**nly **M**emory)

14. PROM als programmierbare Logik (Beispiel)



$$Y1 = m_3 \vee m_4 = (X1 \wedge X2 \wedge \overline{X3}) \vee (\overline{X1} \wedge \overline{X2} \wedge X3)$$

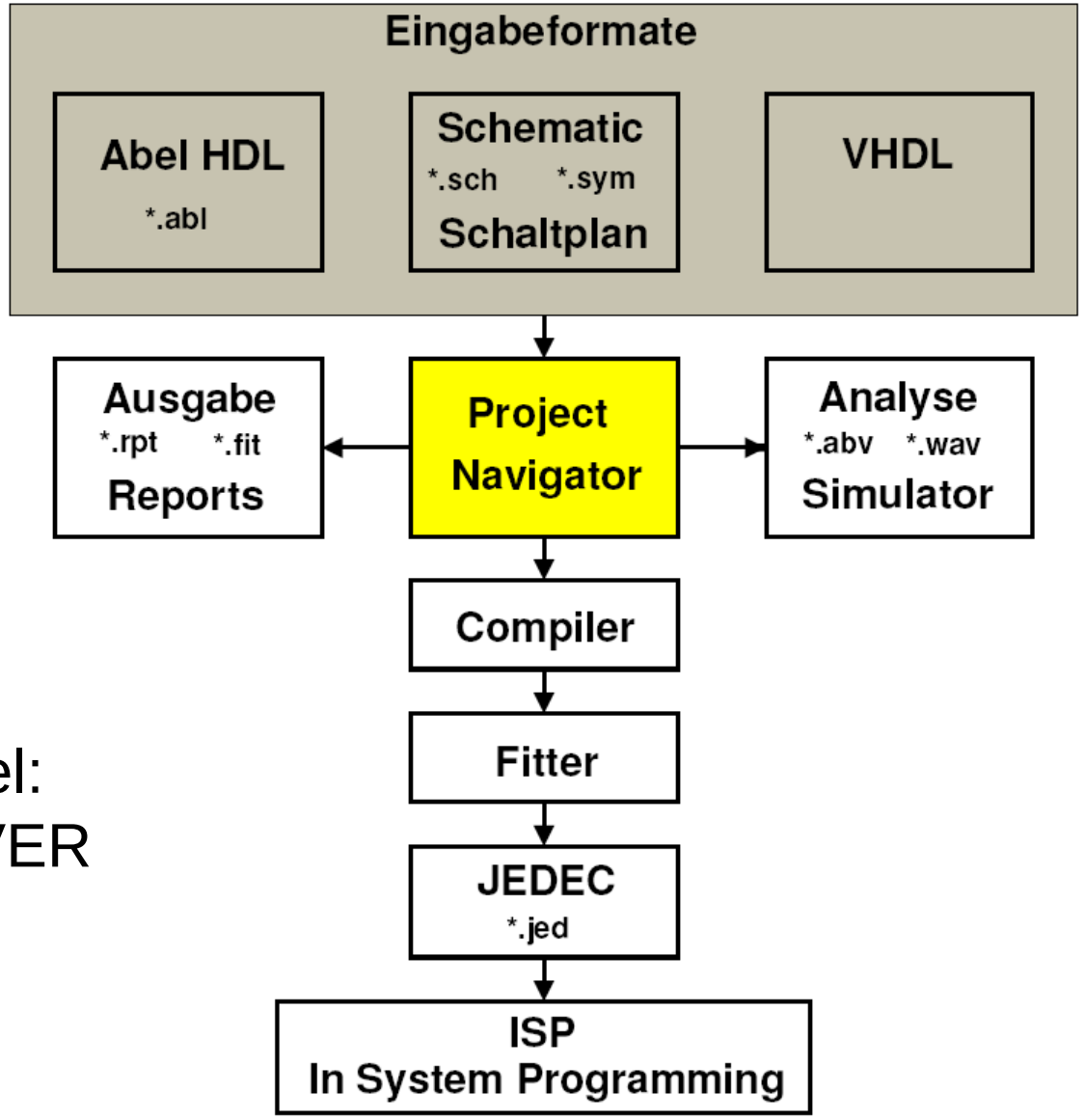
$$Y2 = m_2 \vee m_5 = (\overline{X1} \wedge X2 \wedge \overline{X3}) \vee (X1 \wedge \overline{X2} \wedge X3)$$

$$Y3 = m_4 \vee m_5 = (\overline{X1} \wedge \overline{X2} \wedge X3) \vee (X1 \wedge \overline{X2} \wedge X3)$$

$$Y4 = m_0 \vee m_3 \vee m_7$$
$$= (\overline{X1} \wedge \overline{X2} \wedge \overline{X3}) \vee (X1 \wedge X2 \wedge \overline{X3})$$
$$\vee (X1 \wedge X2 \wedge X3)$$

- beim Übergang von einem Minterm zu nächsten können immer Hazards auftreten.
- daher für asynchrone Schaltungen wenig geeignet.
- gut geeignet für Look-Up-Tabellen (LUT)

Programmierung (CPLD, FPGA)



Beispiel:
ispLEVER

Programmierung

- ISP (In System Programming)
- HDL (Hardware Description Language)
- VHDL (VHSIC HDL)
- VHSIC (Very High Speed Integrated Circuit)
- Abel (Advanced Boolean Expression/Equation Language)
- Abel wurde in den 80er Jahren entwickelt und ist für kleinere Schaltungen hinreichend.
- VHDL und Verilog sind die weltweit am meisten genutzten Hardware-Beschreibungssprachen und sind beide von IEEE standardisiert.

19. Automaten

- Ein **Automat** ist eine **Modellmaschine**, die ein System beschreibt.

- Ein **Automat**
  - reagiert auf eine **Eingabe**
  - produziert eine **Ausgabe**, die von
    - der **Eingabe** und
    - dem momentanen **Zustand des Systems**

    abhängt.

19.1 Endliche Automaten

- Ein Automat wird als **endlicher Automat** bezeichet, wenn die Menge
  - der möglichen Eingabezeichen (Eingabealphabet)
  - der möglichen Ausgabezeichen (Ausgabealphabet)
  - der Zustände

  endlich ist.

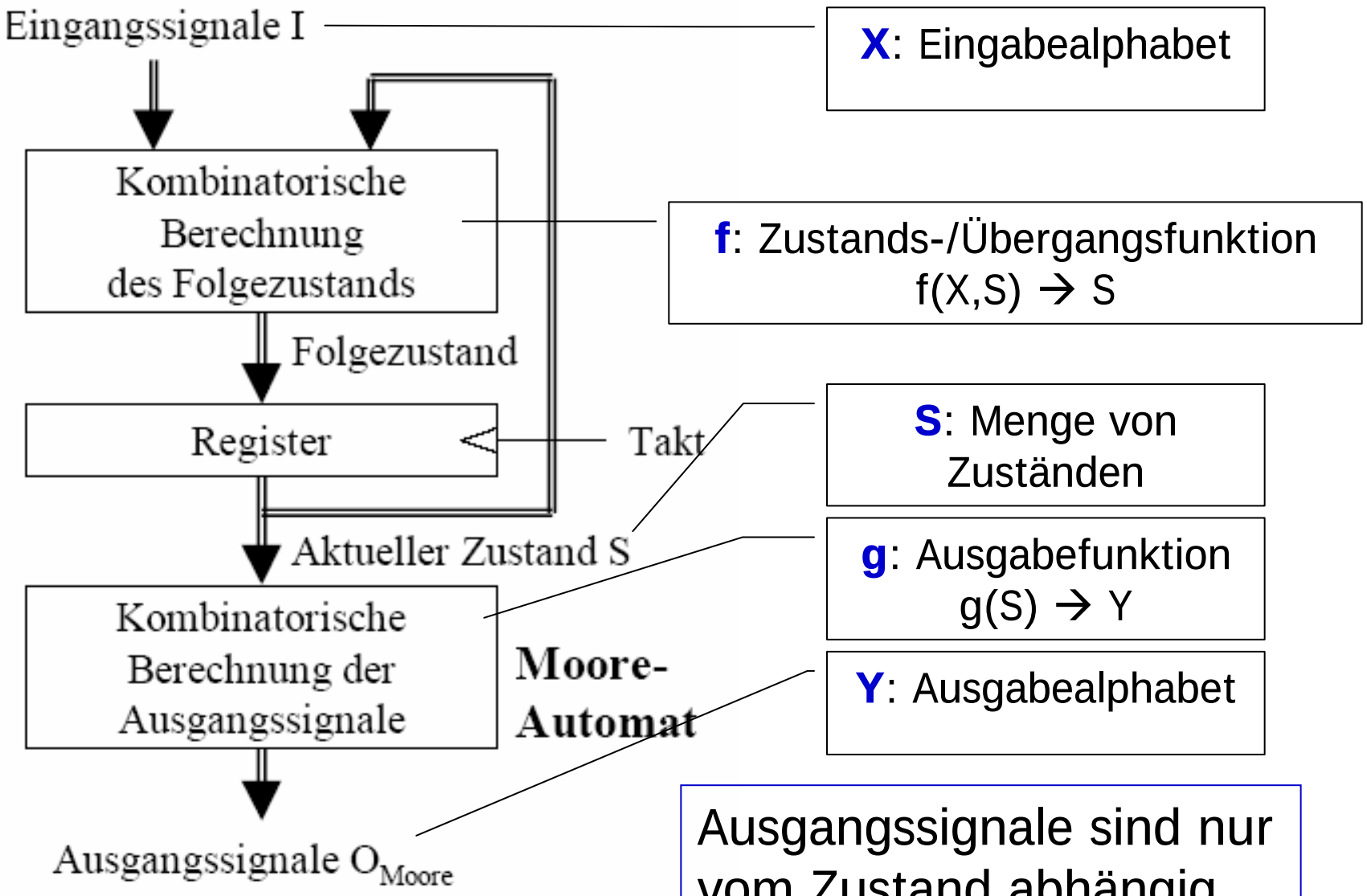
Es gilt:

Jeder endliche Automat
lässt sich in ein Schaltwerk umsetzen

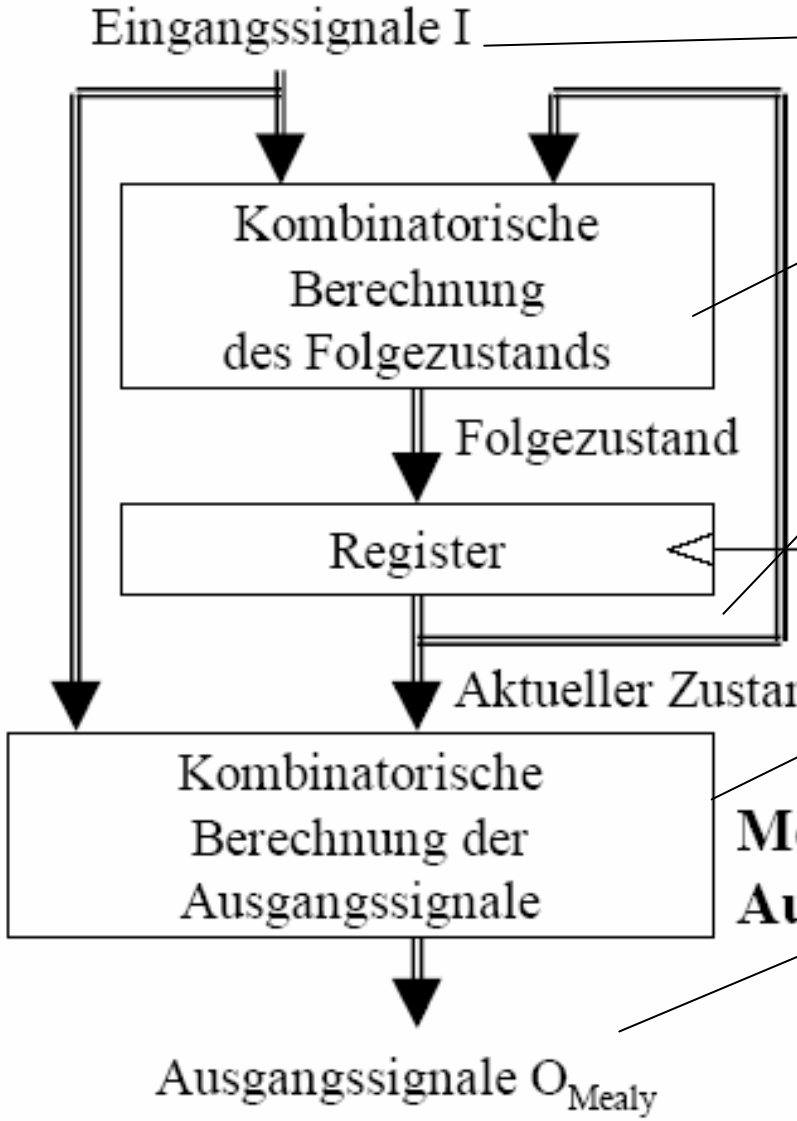
19.2 Formale Beschreibung Automaten

- Ein endlicher Automat ist ein Fünftupel
  A = (X, Y, S, f, g) .
- **X** ist ein endliches nichtleeres **Eingabe**alphabet.
- **Y** ist ein endliches nichtleeres **Ausgabe**alphabet.
- **S** ist eine endliche nichtleere Menge von **Zustände**n.
- **f**: **Zustands**(überführungs)**funktion**
- **g**: **Ausgabefunktion**

19.3 Moore-Automaten



19.4 Mealy-Automaten



X: Eingabealphabet

f: Zustands-/Übergangsfunktion
f(X,S) → S

S: Menge von Zuständen

g: Ausgabefunktion
g(**X**,S) → Y

Y: Ausgabealphabet

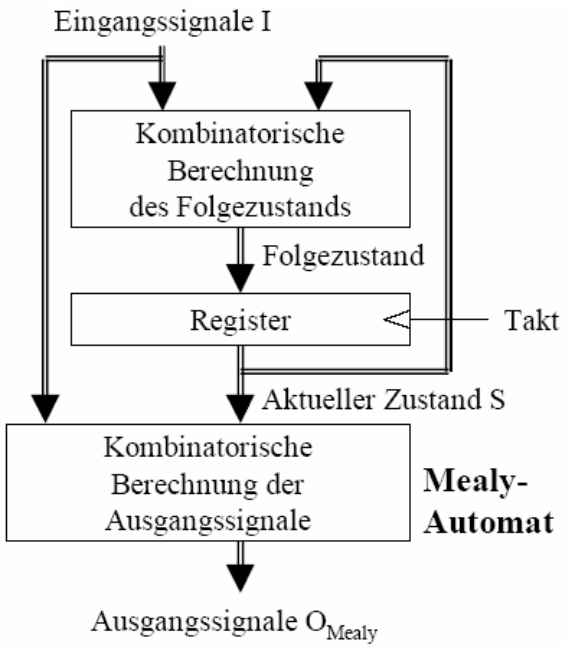
Ausgangssignal wechselt schon bei Änderung des Eingangssignal.

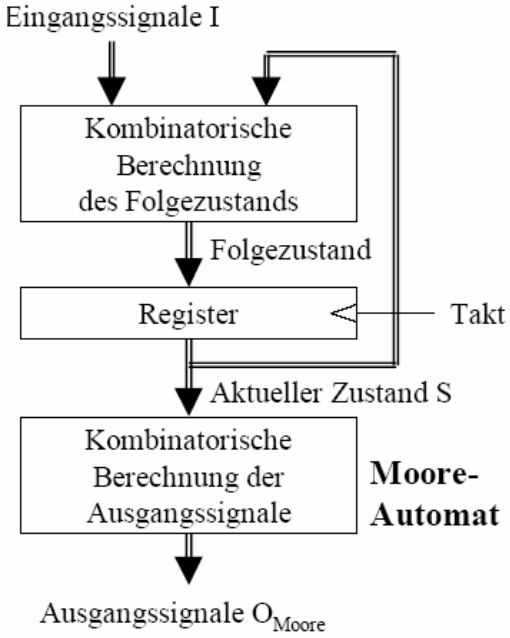
19.2 Formale Beschreibung Automaten (2)

Beschreibung eines Automaten **A = (X, Y, S, f, g)**

- **X** = $x_1, x_2, \ldots, x_i$ **Eingabe**alphabet.
- **Y** = $y_1, y_2, \ldots, y_m$ **Ausgabe**alphabet.
- **S** = $s_1, s_2, \ldots, s_n$ **Zustandsmenge**.
- **f**: $X \times S \rightarrow S$ oder $f: (x_i,s_j) \rightarrow s_k$
  Zustandsfunktion, Übergangsfunktion
  $\mathbf{S}(t_{n+1}) = \mathbf{f}(X(t_n), S(t_n))$
- **g**: $X \times S \rightarrow Y$ oder $g: (x_i,s_j) \rightarrow y_k$
  Ausgabefunktion
  $\mathbf{Y}(t_n) = \mathbf{g}(X(t_n), S(t_n))$

19.3.-4. Mealy – Moore Automaten





Mealy-Automat:

$\mathbf{Y}(t_n)=\mathbf{g}(\ X(t_n),\ S(t_n)\ )$

Moore-Automat:

$\mathbf{Y}(t_n)=\mathbf{g}(\ S(t_n)\ )$

Ausgang hängt nur von Zustand ab:

Zustandsorientiertes Schaltwerk

19.5 Beispiele für Automaten

- Synchrone Zähler sind Beispiele für Moore-Automaten. Der Eingangsvektor könnte beispielsweise ein Signal zum Umschalten der Zählrichtung sein.

- Was ist Zyklische Folgeschaltung?

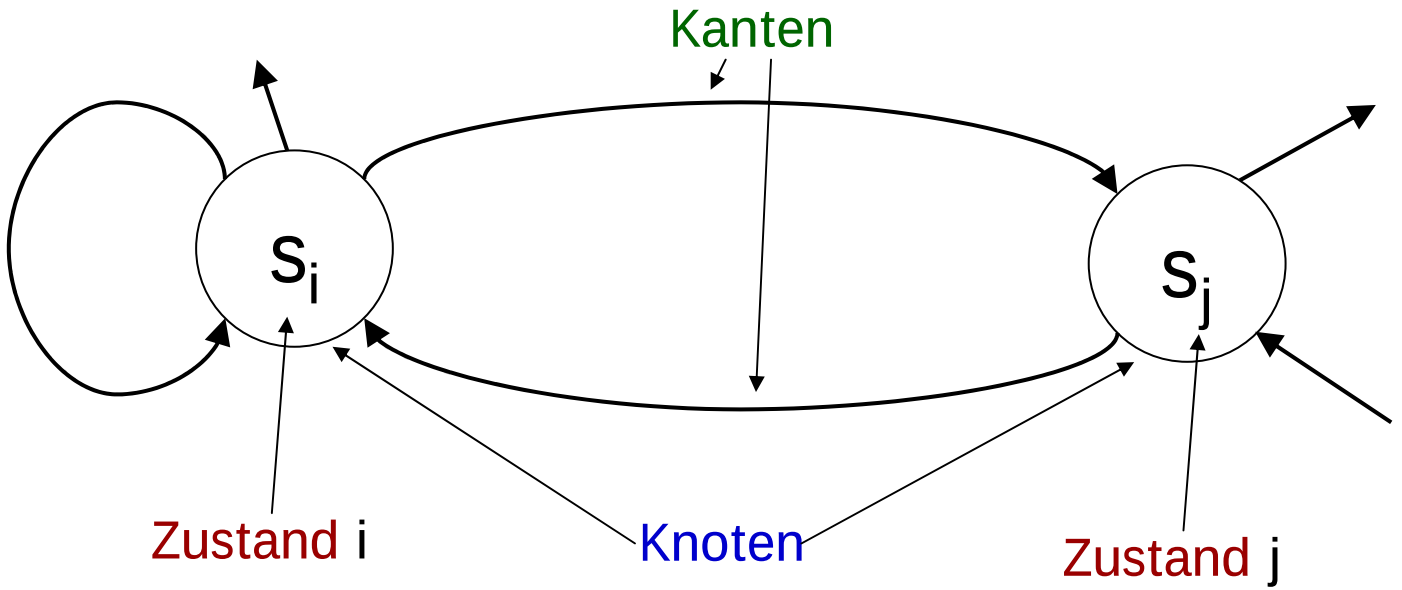
Schaltwerke, die außer dem Taktsignal kein Eingangssignal haben, nennt man **autonom** oder **autonome Automaten**.

19.6 Beschreibung endlicher Automaten

Endliche Automaten, Schaltwerke können beschrieben werden durch:

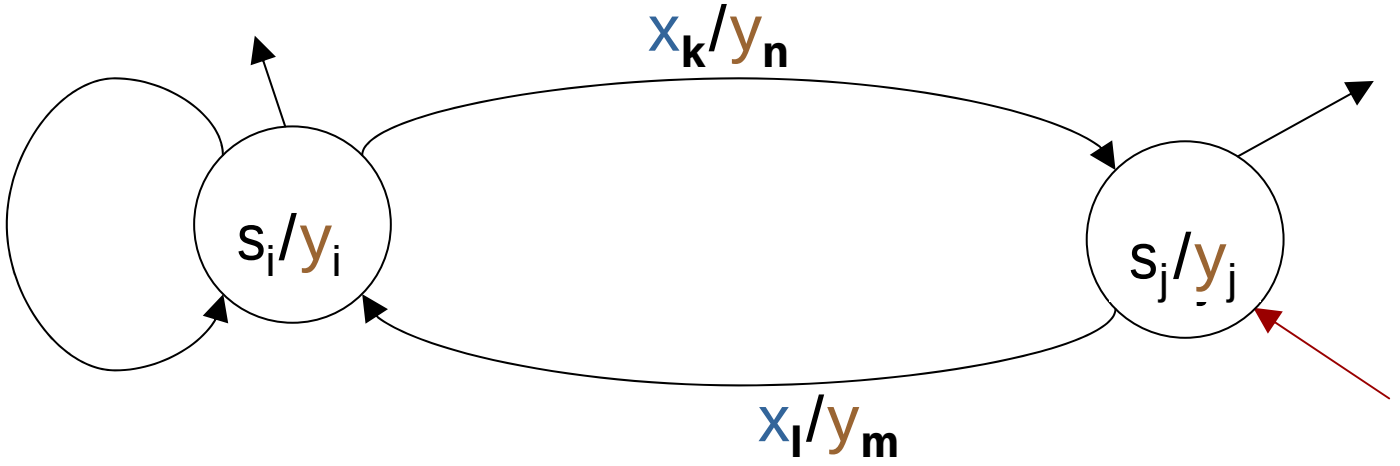
- Zustandsfolgetabellen
- KV-Diagramme
- Schaltfunktionen oder Vektorfunktionen (enthalten die Ausgangs- und Übergangs-Funktionen)
- Zustandsgraphen

19.6 Zustandsgraphen



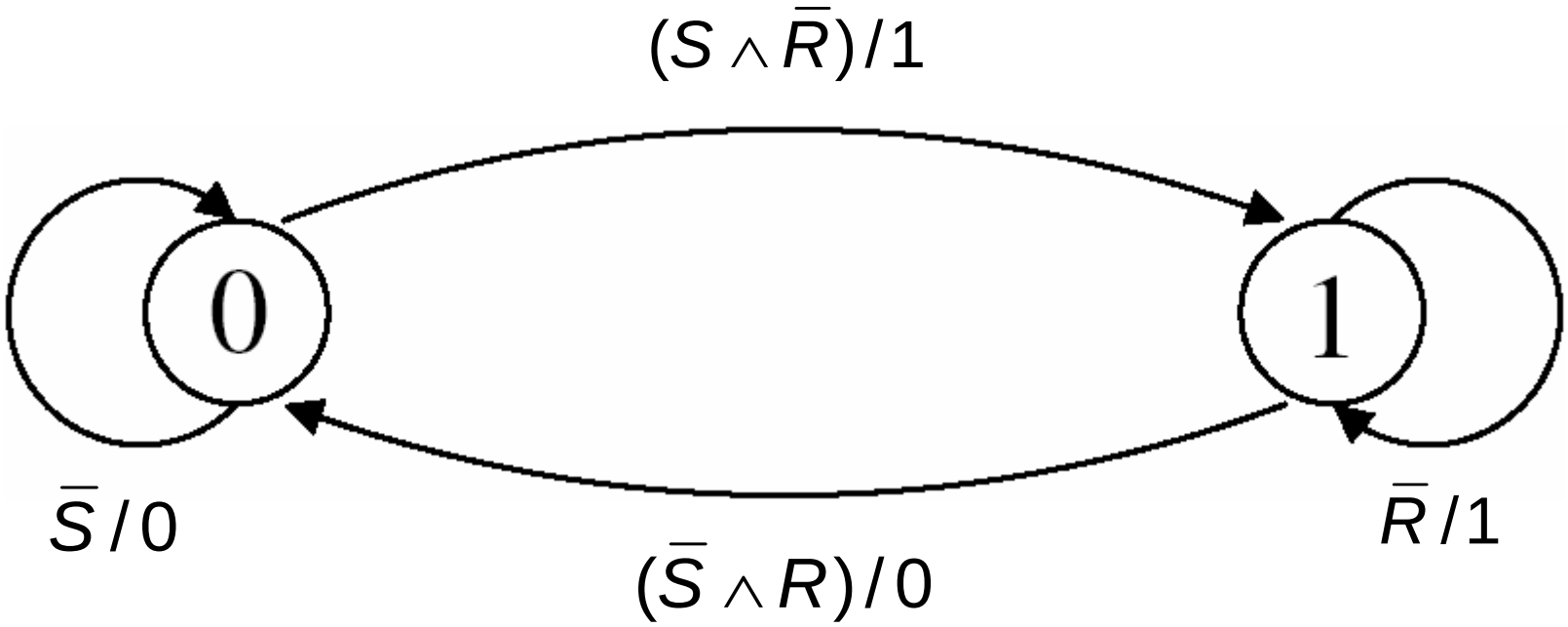
- Ein Zustandsgraph besteht aus Knoten und gerichteten Kanten. Die Knoten (Kreise) beschreiben die Zustände. Die Kanten stellen in Abhängigkeit vom anliegenden Eingabezeichen die Übergänge zwischen dem gegenwärtigen und dem nächsten Zustand her (nächste Folie).
- Zustandsgraph und Automaten sind äquivalent.

19.6 Zustandsgraphen

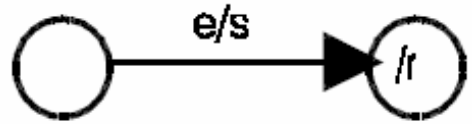


- Pfeil auf ersten Zustand ist Anfangszustand
- Vor dem '/' steht die Eingangsbedingung, unter der der Ausgangszustand verlassen wird, hinter dem '/' steht das Ausgangssignal
  → Mealy-Automat
- Im Knoten vor dem '/' steht wie bisher der Zustand, dahinter das Ausgangssignal
  → Moore-Automat

19.6 Zustandsgraphen (Beispiel FF)

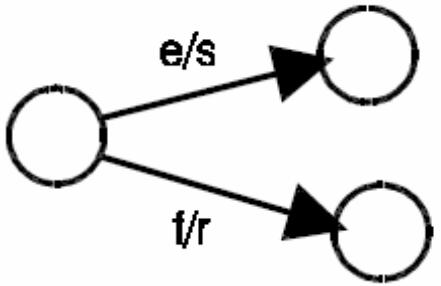


19.6 Zustandsgraphen



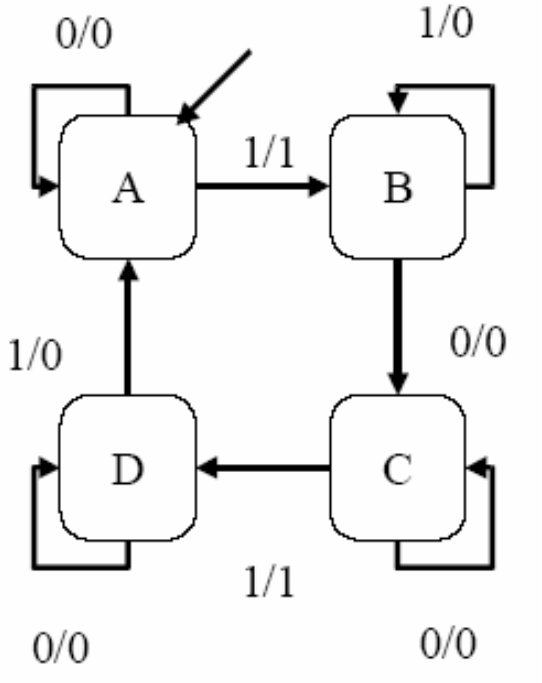
Zustandsübergang

e auslösendes Ereignis
/ s und /r erzeugte Ereignisse



- Statecharts beschreiben endliche Automaten.
- Statecharts sind äquivalent zu endlichen Automaten.
- Ein Zustand im Chart ist elementar oder selbst wieder ein Statechart. Statecharts sind also hierarchisch und parallel zerlegbar.
- Das Zusammenfassen von Zuständen erzeugt Superstates

19.7 Beispiel Mealy-Automat

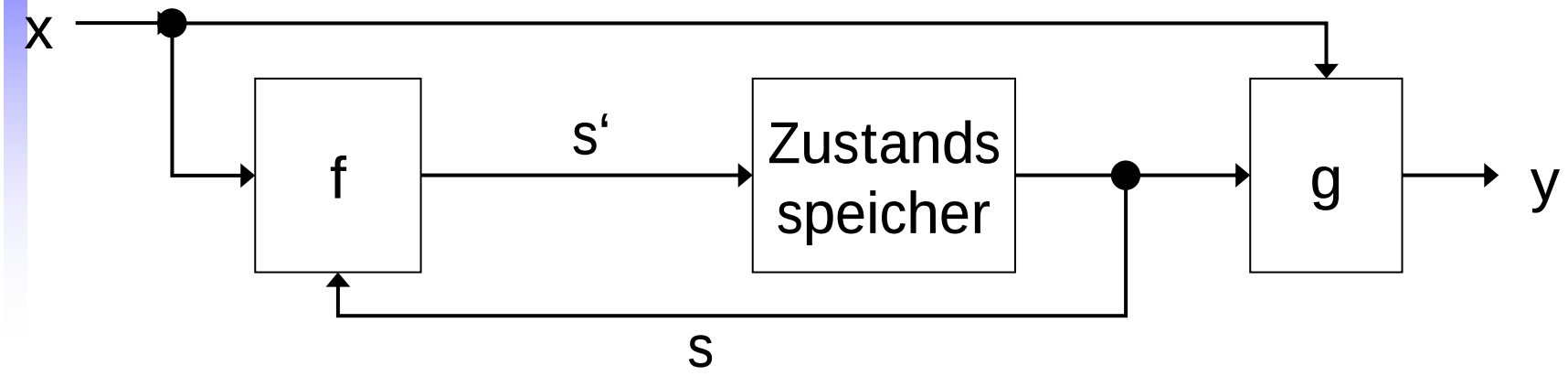


- Beschreiben Sie den Automaten mit einem 5-Tupel

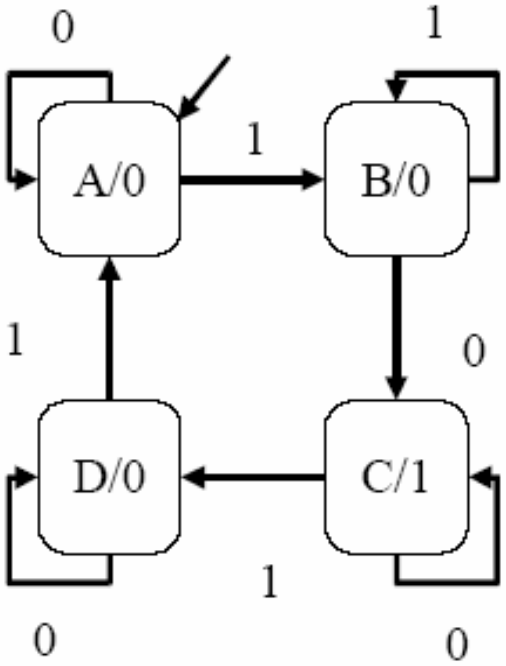
A = ({0,1}, {0,1}, {A,B,C,D}, f, g);
mit f: f(X,S) ; g: g(X,S)

- Skizzieren Sie den Automaten

ergänzen Sie ...



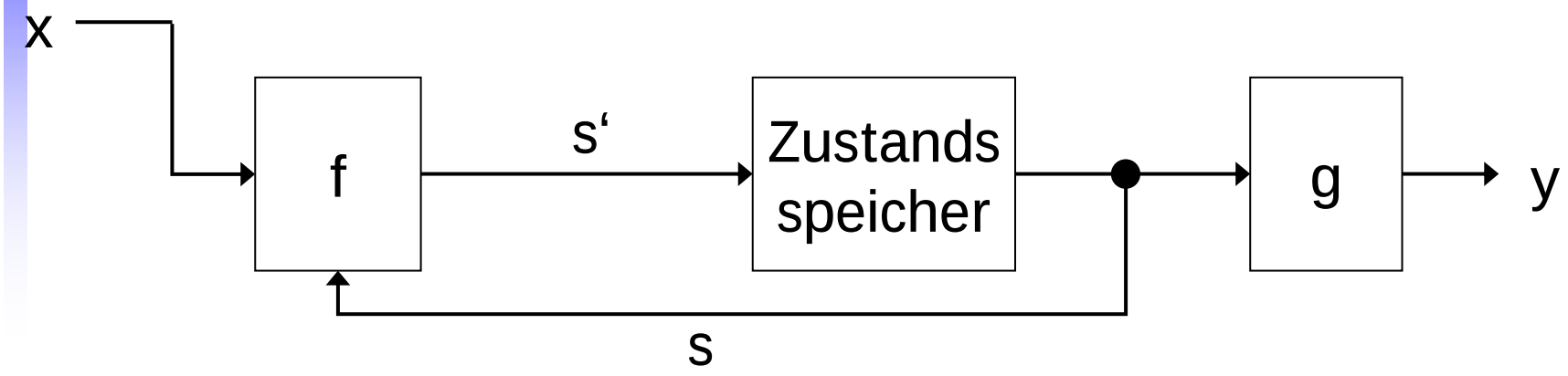
19.7 Beispiel Moore-Automat



- Beschreiben Sie den Automaten mit einem 5-Tupel

A = ({0,1}, {0,1}, {A,B,C,D}, f, g);
mit f: f(X,S) ; g: g(S)

- Skizzieren Sie den Automaten



19.8 Zustandsfolgetabelle

Eingangsvariable		Ausgangsvariable	
$s_0, \ldots, s_n$	$x_0, \ldots, x_i$	$s'_0, \ldots, s'_n$	$y_0, \ldots, y_m$

Die Zustandsfolgetabelle enthält:

- die Komponenten des Eingangsvektor X und die Komponenten des Zustandsvektors $S(t_n)$
- Als Ausgangsvariable die Komponenten des Ausgangsvektors Y und die Komponenten des Folgezustandsvektors $S(t_{n+1})$