4.6 ALUs (Arithmetic Logical Units) und Rechenwerke

Arithmetisch-Logischen-Einheiten (ALUs) sind Schaltnetze, die typischerweise folgende Operationen ausführen:

Arithmetische Operationen:

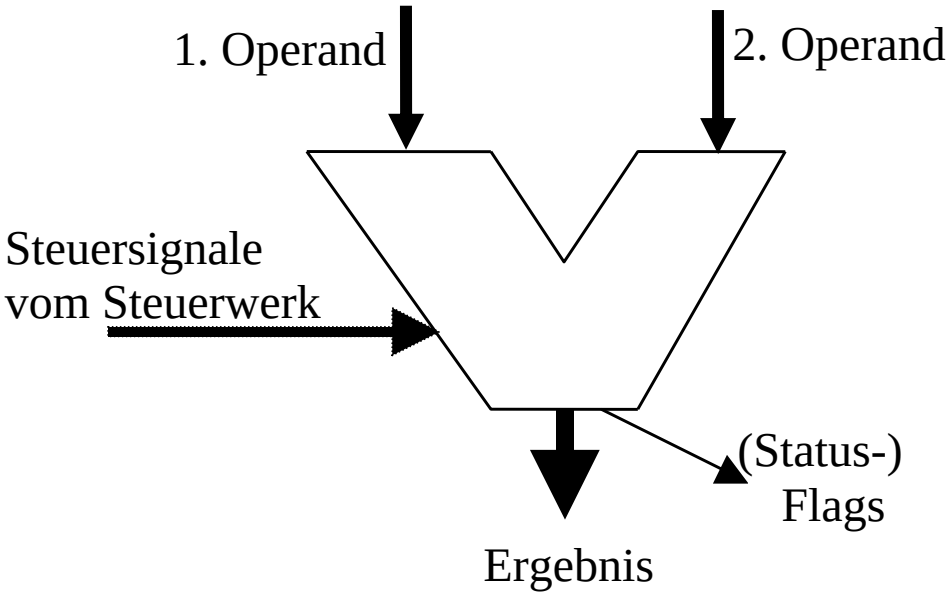
Addition, Subtraktion, ggf. Multiplikation

und

Logische Operationen:

UND, ODER, Exklusiv-ODER,
Löschen, Negation eines Operanden

Häufig verwendetes Symbol für eine ALU:



Die Auswahl der auszuführenden Operationen wird durch entsprechende Steuersignale von einem übergeordneten Steuerwerk vorgenommen.

Dieses koordiniert auch die Bereitstellung und Abholung der Daten (Operanden) sowie das Zwischenspeichern von Zwischenergebnissen.

Durch einen geeigneten übergeordneten Ablauf (im Steuerwerk) können sämtliche Operationen durch eine ALU ausgeführt werden. Sie ist deshalb als eine "Universalschaltung" das Kernstück sämtlicher Computer.

I. d. R. generiert eine ALU auch Statussignale ("Flags"), die dem Steuerwerk spezielle Kriterien anzeigen, z. B. Vorzeichen, Ergebnis = 0 oder Überlauf.

Für größere Wortbreiten kann eine ALU auch in Module aufgeteilt werden, die kleinere Bitgruppen (Bitscheiben, "Bit-Slices") parallel verarbeiten und entsprechend der gewünschten Wortbreite kaskadiert werden. Sie erhalten die Steuersignale gemeinsam.

Realisierung als Integrierte Schaltung

Eine Realisierung als TTL-IC ist der SN74181-Baustein und seine Varianten.

MOTOROLA

SN54/74LS181

4-BIT ARITHMETIC LOGIC UNIT

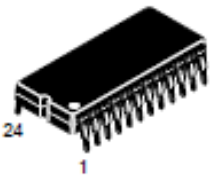
The SN54/74LS181 is a 4-bit Arithmetic Logic Unit (ALU) which can perform all the possible 16 logic, operations on two variables and a variety of arithmetic operations.

- Provides 16 Arithmetic Operations Add, Subtract, Compare, Double, Plus Twelve Other Arithmetic Operations
- Provides all 16 Logic Operations of Two Variables Exclusive — OR, Compare, AND, NAND, OR, NOR, Plus Ten other Logic Operations
- Full Lookahead for High Speed Arithmetic Operation on Long Words
- Input Clamp Diodes

4-BIT ARITHMETIC LOGIC UNIT

LOW POWER SCHOTTKY

J SUFFIX
CERAMIC
CASE 623-05

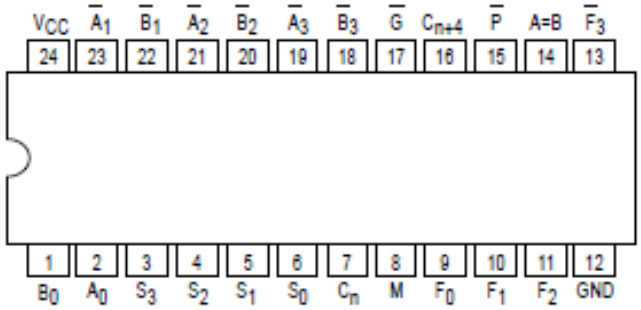


N SUFFIX
PLASTIC
CASE 649-03

ORDERING INFORMATION

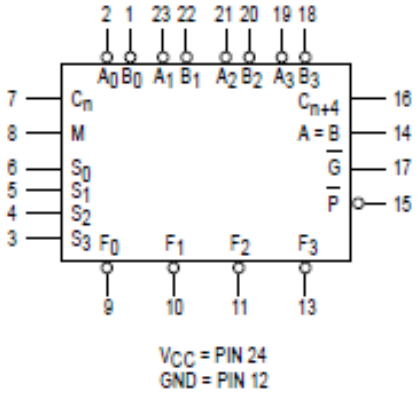
SN54LSXXXJ	Ceramic
SN74LSXXXN	Plastic

CONNECTION DIAGRAM DIP (TOP VIEW)



NOTE:
The Flatpak version has the same pinouts (Connection Diagram) as the Dual In-Line Package.

LOGIC SYMBOL



VCC = PIN 24
GND = PIN 12

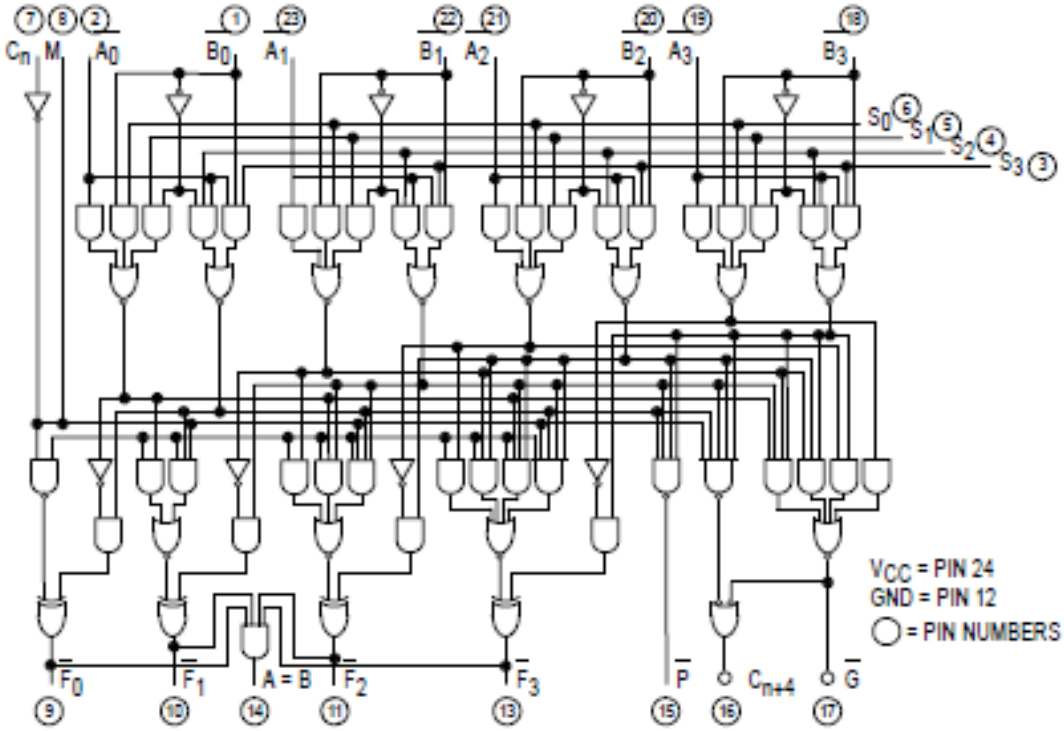
PIN NAMES

		LOADING (Note a) HIGH	LOW
$\overline{A}_0-\overline{A}_3, \overline{B}_0-\overline{B}_3$	Operand (Active LOW) Inputs	1.5 U.L.	0.75 U.L.
S_0-S_3	Function — Select Inputs	2.0 U.L.	1.0 U.L.
M	Mode Control Input	0.5 U.L.	0.25 U.L.
$\overline{C}_n$	Carry Input	2.5 U.L.	1.25 U.L.
$\overline{F}_0-\overline{F}_3$	Function (Active LOW) Outputs	10 U.L.	5 (2.5) U.L.
$\overline{A}=B$	Comparator Output	Open Collector	5 (2.5) U.L.
G	Carry Generator (Active LOW) Output	10 U.L.	10 U.L.
$\overline{P}$	Carry Propagate (Active LOW) Output	10 U.L.	5 U.L.
C_{n+4}	Carry Output	10 U.L.	5 (2.5) U.L.

NOTES:
a. 1 TTL Unit Load (U.L.) = 40 µA HIGH/1.6 mA LOW.
b. The Output LOW drive factor is 2.5 U.L. for Military (54) and 5 U.L. for Commercial (74) Temperature Ranges.

SN74181-Realisierung als Integrierte Schaltung

LOGIC DIAGRAM



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	$\overline{AB}$	AB minus 1	$\overline{A+B}$	A + B
L	L	H	L	A + B	AB minus 1	$\overline{AB}$	A + B
L	L	H	H	Logical 1 minus 1		Logical 0 minus 1	
L	H	L	L	$\overline{A+B}$	A plus (A + B)	$\overline{AB}$	A plus AB
L	H	L	H	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	AB plus A	A + B	(A + 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

Rechenwerke

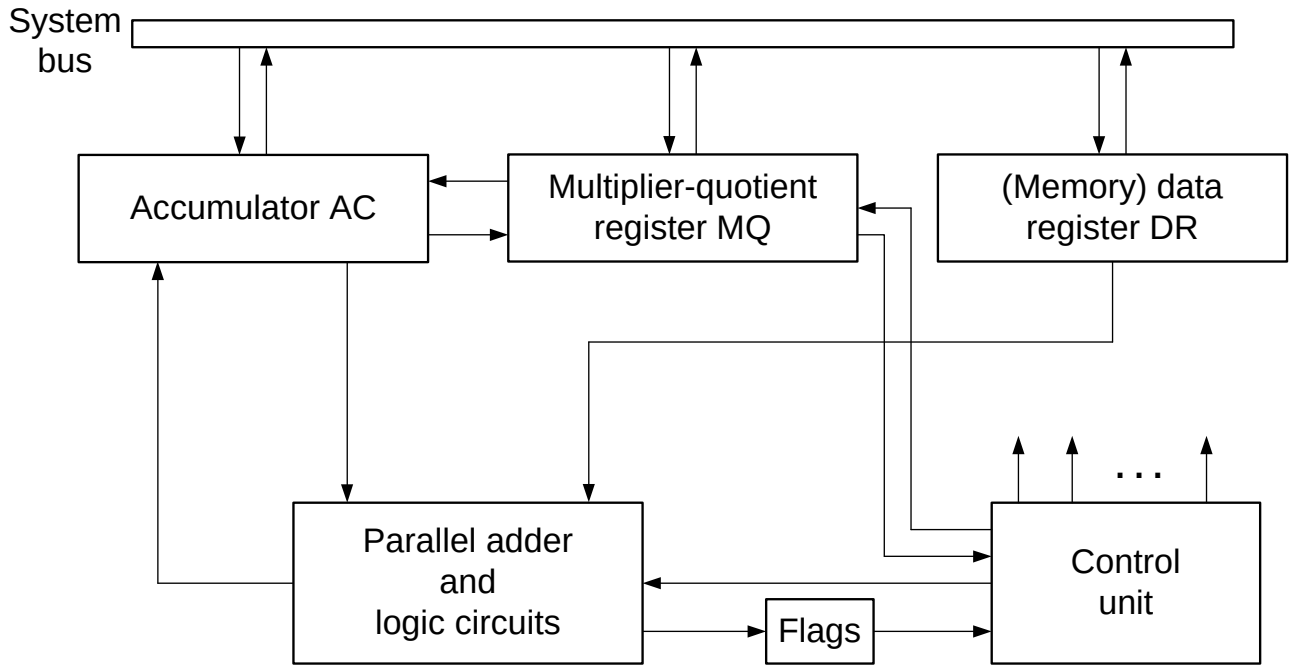
Komplexere Operationen wie Multiplikation und Division werden aus Aufwandsgründen meist in mehreren Schritten realisiert,
d. h. auf eine Sequenz (Ablauf) von einfacheren durch eine einfache Hardware, z.B. eine ALU, realisierbaren Operationen abgebildet.

Die Kombination von ausführender Hardware (Operationswerk) und Steuerwerk (Ablaufsteuerung; Control unit) nennt man Rechenwerk.

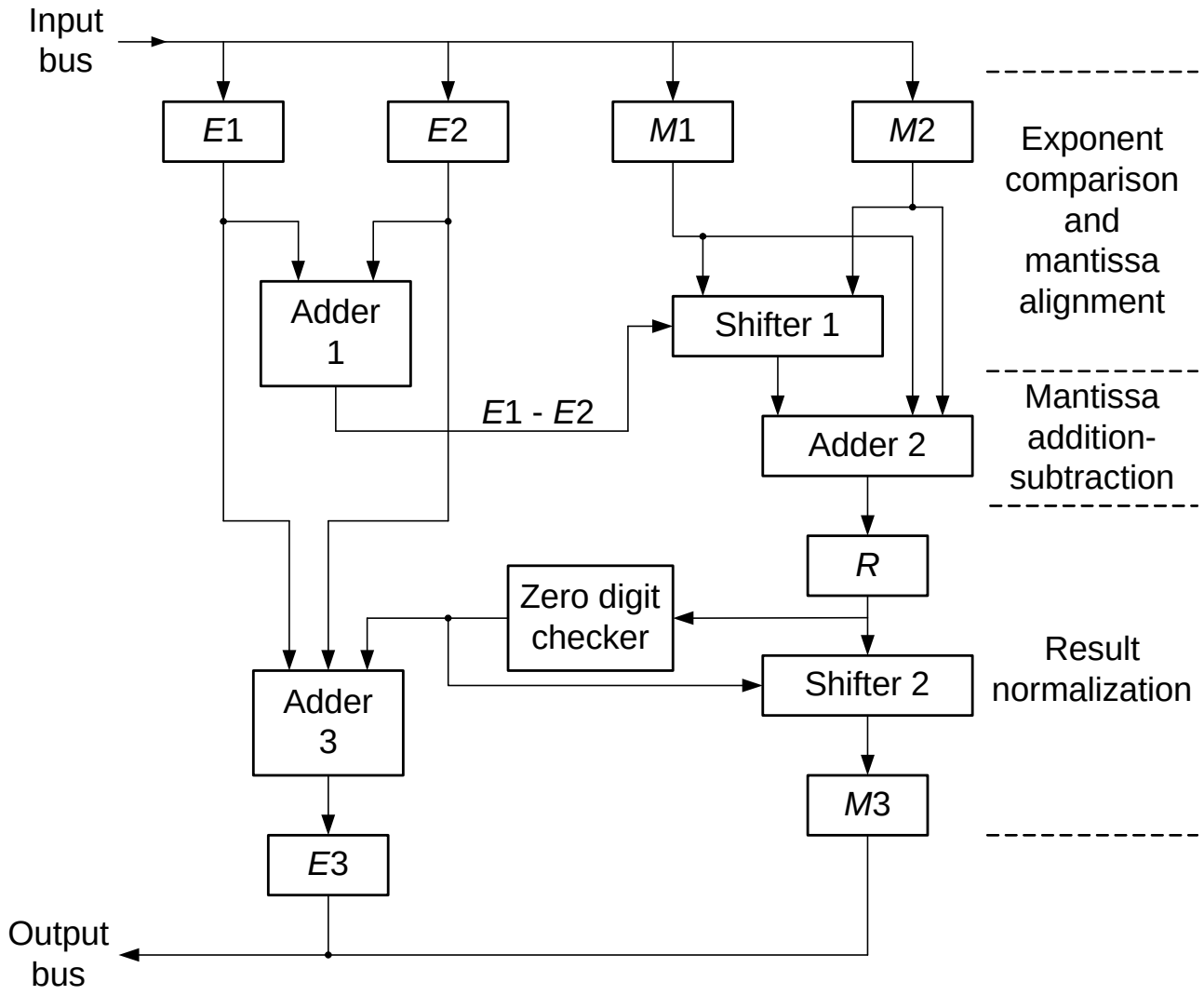
Rechenwerke verfügen i.d.R. über Speicher („Register“) für Zwischenergebnisse.

Sie können eigenständige Bausteine, Coprozessoren oder Bestandteile von Mikroprozessoren sein.

Beispiel: Multiplizier-Rechenwerk



Beispiel: Gleitkomma-Rechenwerk



Getrenntes Exponenten- und Mantissenrechenwerk.

Ein Steuerwerk steuert die einzelnen Schritte der Gleitkomma-operationen.

z. B. Fließ-punkt-addition:
- Exponentenvergleich,
- Exponentenangleich durch Rechts-/Linksshift der Mantisse,
- Addition der Mantissen,
- Normalisierung

4.7 Laufzeiteffekte in Schaltnetzen

Glitches und Hazards

Neben dem funktionalen Verhalten ist das zeitliche, das dynamische bzw. transiente Verhalten der Verknüpfungsglieder wesentlich für das korrekte Verhalten eines Systems. Denn die Ausbreitung elektrische Signale auf Leitungen und das Durchlaufen von Schaltgliedern benötigen Zeit, die (Signal-) Laufzeit.

Die Laufzeiten hängen von vielen Parametern ab, z.B. von der Länge einer Leitung und von der Art und dem Fan Out eines Schaltgliedes.

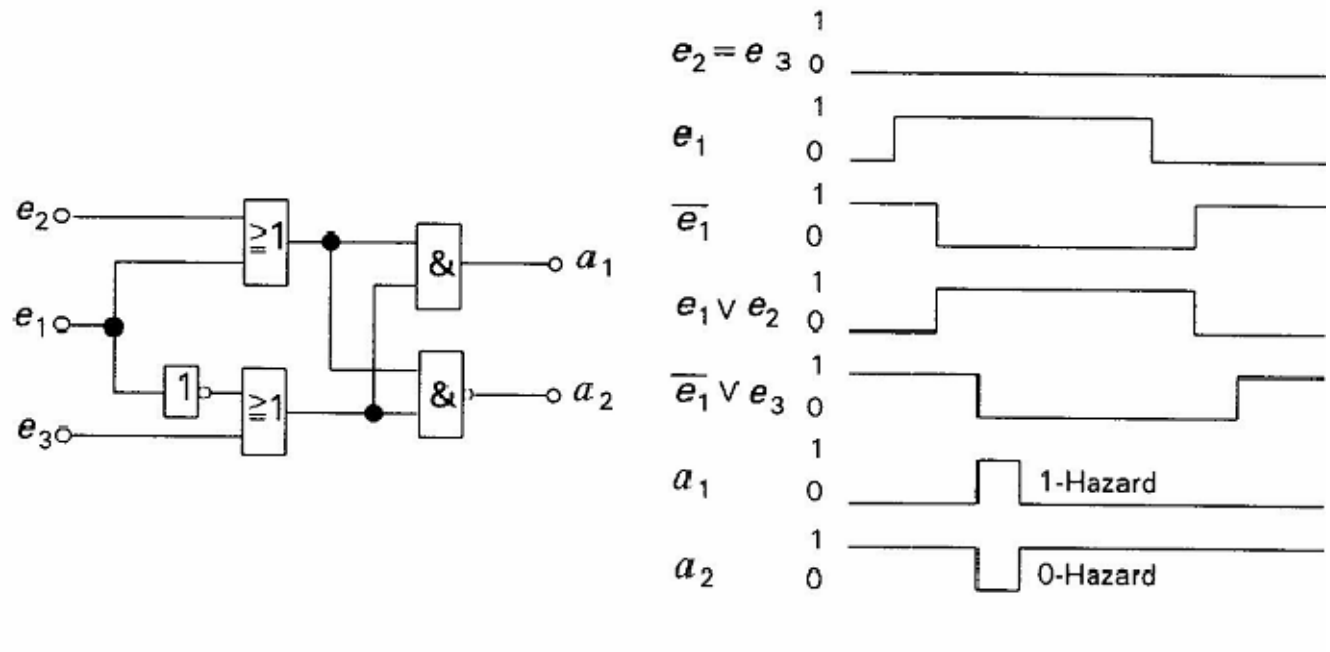
Kritisch wird es, wenn Signale sich über verschiedene Wege innerhalb einer Schaltung ausbreiten und wieder verknüpft werden. Das Betrachten des **stationären Verhaltens** ist dann nicht immer ausreichend. Denn Unterschiede in den Signallaufzeiten können (müssen aber nicht immer) bei mehrstufigen Logikschaltungen zu kurzen, transienten Störimpulsen führen. Um solche Effekte aufzudecken, muss das **dynamische Verhalten** analysiert werden.

Def.: Ein **Glitch** ist eine kurze Signaländerung, die bei statischer Betrachtung der Schaltvariablen theoretisch oder bei idealen Verknüpfungsgliedern nie auftreten würde (dürfte).

Def.: Man spricht von einem **Hazard** (engl.: Zufall, Gefahr, Risiko), wenn in einer Schaltung die Möglichkeit besteht, dass Glitches auftreten.

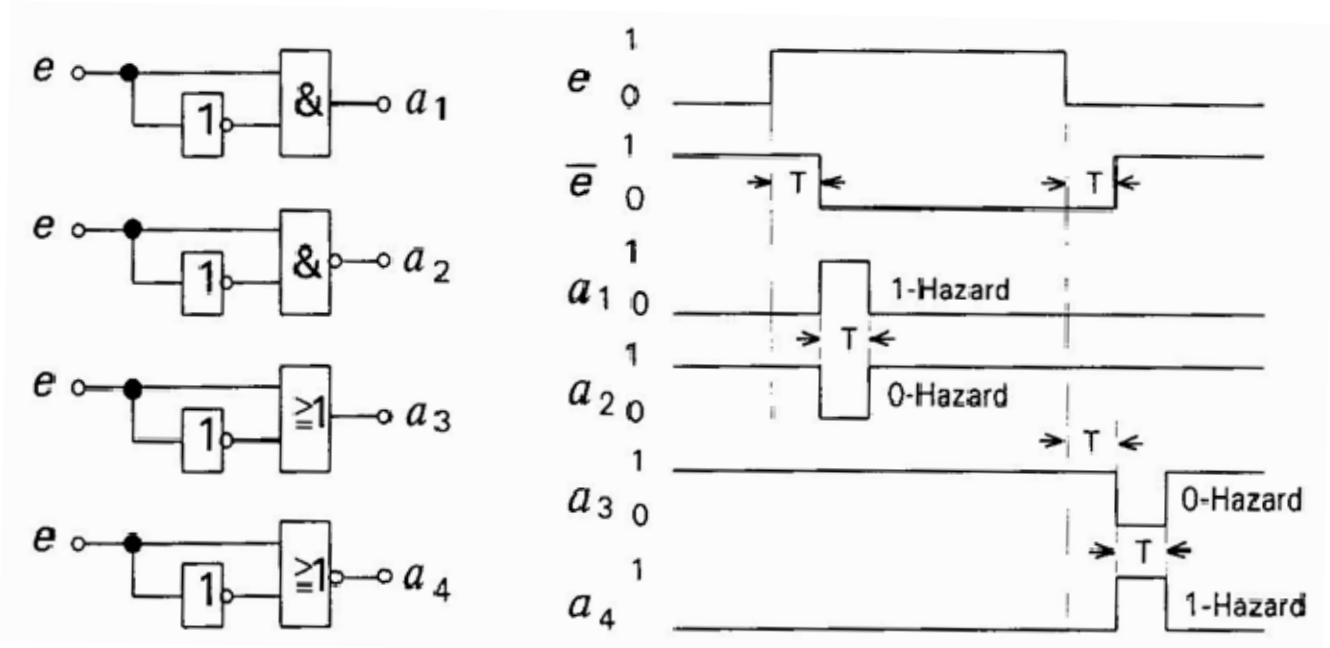
Hazards sind also nur potentiell fehlerhafte Schaltzustände, die an Signalübergängen *möglich* sind.

Beispiel:



Man spricht von statischen 1(0)-Hazards, wenn es eine Paarung von Eingangssignalkombinationen gibt, bei denen potentiell aufgrund von Laufzeiten kurze 1(0)-Störimpulse bei einer Änderung der Eingangssignale auftreten können, das Signal aber eigentlich konstant 0(1) liefern müsste, d. h. sich ohne Laufzeitverzögerung statisch verhalten würde.

Entstehung von (statischen) Hazards (Prinzip)



Analyse der Entstehung von (statischen) Hazards

Werden die Laufzeiten berücksichtigt, können Hazards in den Schaltfunktionen dadurch identifiziert werden, dass in den Verknüpfungen Unterschiede bei den Laufzeitindizes (hier als Exponent angegeben) auftreten.

Beispiel: Hazards bei obigen Schaltungen

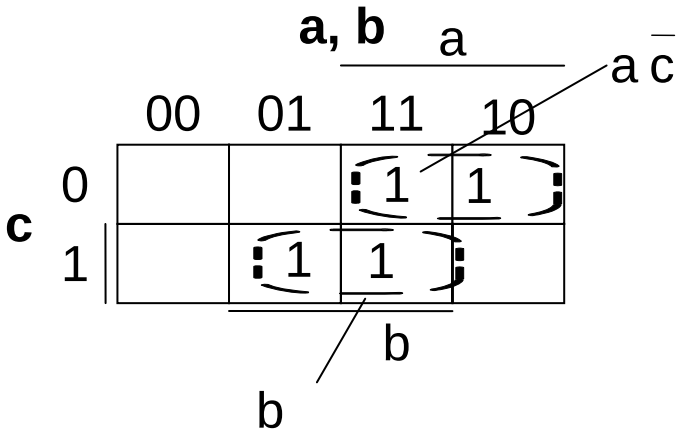
$$a_1 = \left(e \cdot \bar{e}^{-1}\right)^{-1} = e^{-1} \cdot \bar{e}^{-2} \qquad \rightarrow \text{1-Hazard}$$

$$a_2 = \left(\overline{e \cdot \bar{e}^{-1}}\right)^{-1} = \bar{e}^{-1} + e^{-2} \qquad \rightarrow \text{0-Hazard}$$

$$a_3 = \left(e + \bar{e}^{-1}\right)^{-1} = e^{-1} + \bar{e}^{-2} \qquad \rightarrow \text{0-Hazard}$$

$$a_4 = \left(\overline{e + \bar{e}^{-1}}\right)^{-1} = \bar{e}^{-1} \cdot e^{-2} \qquad \rightarrow \text{1-Hazard}$$

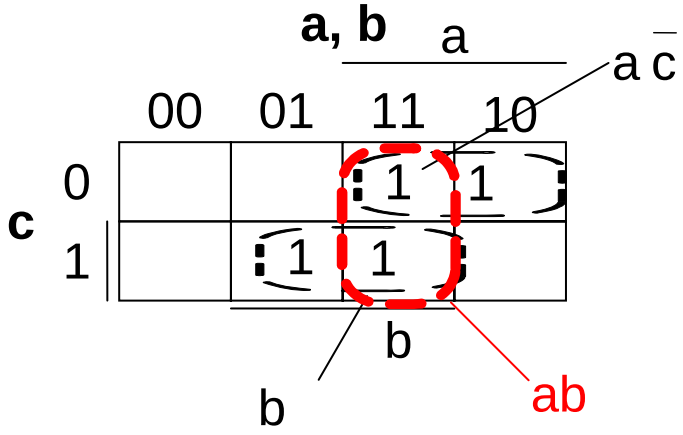
In KV-Diagrammen sind Hazards daran zu erkennen, dass zwei Vereinfachungsschleifen von Primimplikanten aneinander stoßen, sich aber nicht überlappen.



$$f(a,b,c) = bc + a\bar{c}$$

Ein Hazard (und damit auch ein Glitch) wird durch das Hinzufügen eines redundanten Terms, d.h. auch eines Gatters, vermieden, der diesen Übergang abdeckt.

Dieser ist z. B. im KV-Diagramm recht einfach durch eine Vereinfachungsschleife über diesen Wechsel hinweg zu bestimmen.



$$f(a,b,c) = ab + bc + a\bar{c}$$

Dynamische Hazards

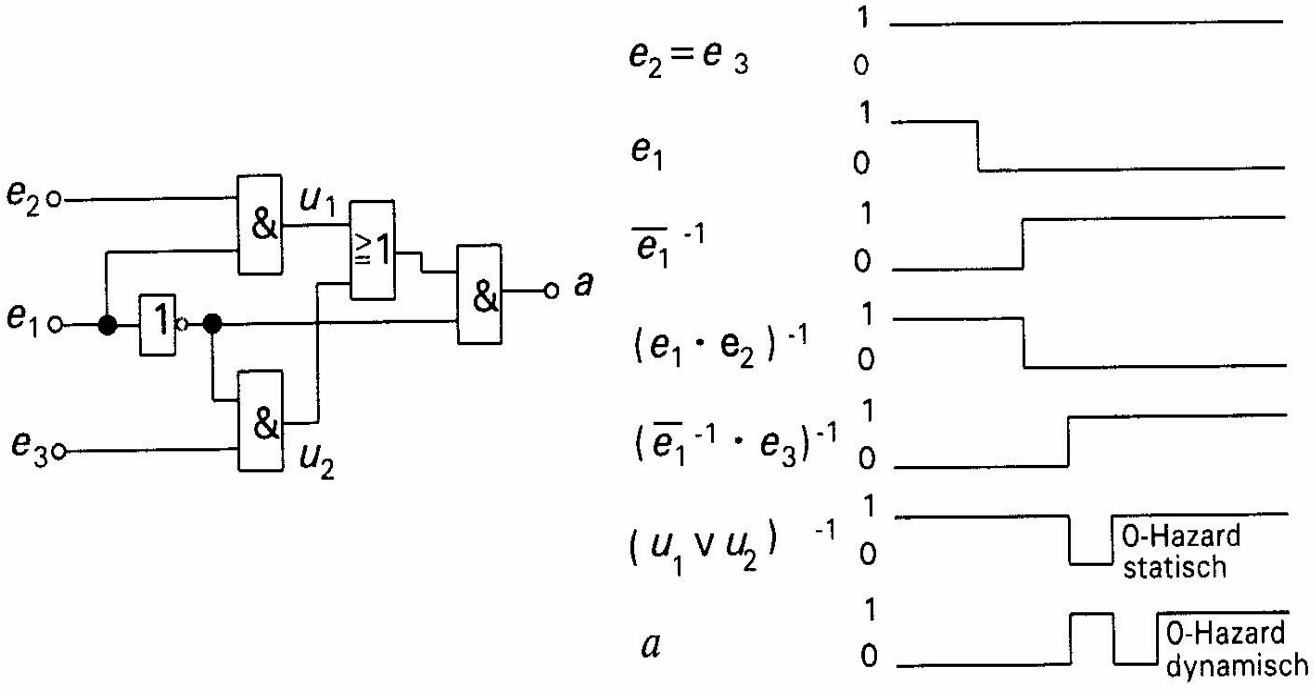
In mehrstufigen Schaltungen können neben statischen Hazards auch dynamische Hazards entstehen, wenn mehrere Wege mit unterschiedlichen Laufzeitverzögerungen ein Ausgangssignal beeinflussen.

Def.: **Dynamische Hazards** sind potentielle Mehrfachtransitionen (Signalübergänge) an Stellen, bei denen ein Ausgangssignal nur einen Signalwechsel vornehmen sollte.

Sie können sich als zusätzliche Signalwechsel äußern, wenn bei einer Änderung eines Eingangssignals das Ausgangssignal zwar zunächst richtig verändert wird, aber durch interne Hazards vorübergehend wieder auf den alten Wert zurückfällt.

Die Ursache sind mehrere Logikpfade mit unterschiedlichen Verzögerungen, über die ein Signalwechsel wirksam ist.

Beispiel:



In der Praxis ist die Hazard-Problematik oft noch komplexer, weil sich die Laufzeiten von Schaltgliedern abhängig von deren Betriebsbedingungen (z.B. Temperatur und Spannung) ändern können und dadurch nicht in jedem Fall Hazards zu Glitches führen.

Echte Glitches können sich bei Schaltwerken (Automaten; s. Kap. 6) kritisch auswirken, weil hier fehlerhafte Übergangszustände ggf. zu dauerhaften Zustandsänderungen und damit zu einem Fehlverhalten des ganzen Schaltwerkes führen können.

Die Eliminierung von Hazards und Glitches kann durch redundante Gatter oder Extraverzögerungen erfolgen, ist aber oft nur eingeschränkt möglich.

Lösung: **getaktete Verarbeitung**, d.h. Trennung von

- Signalverarbeitung in Schaltnetzen
  (incl. aller transienten, laufzeitbedingten Effekte)

und

- Übernahme der Ergebnisse nach dem Einschwingen als fester Ausgangspunkt für den nächsten Verarbeitungsschritt

➔ Zwischenspeichern von zu verknüpfenden Signalen (s. Kap. 5), damit sie während ihrer Verarbeitung stabil sind, und entsprechend verzögerte Übernahme der Verknüpfungsergebnisse in der stationären Phase

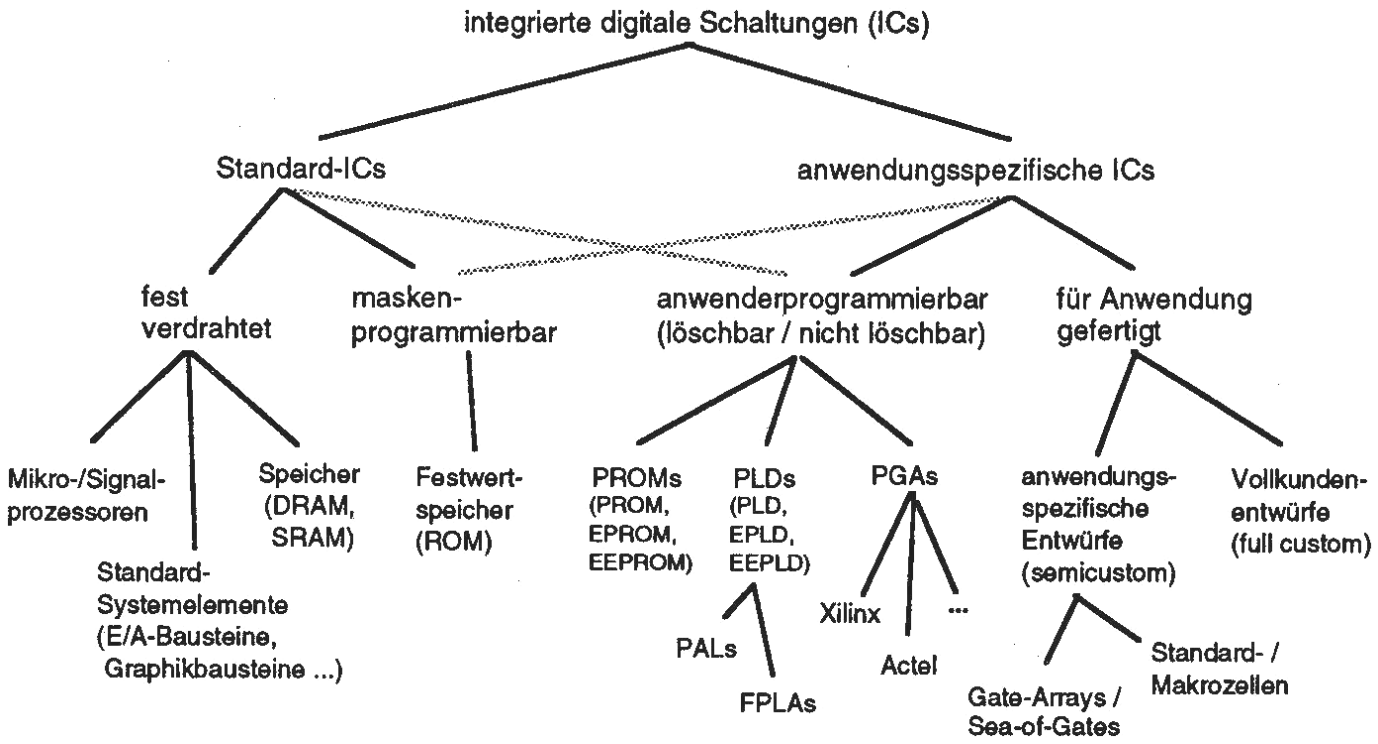
4.8 Schaltnetz-Implementierung mit integrierten Schaltungen

4.8.1 Klassen integrierter Schaltungen

Integrationsstufen

SSI Small Scale Integration (< 10^2 Gatterfunktionen)
MSI Medium Scale Integration (10^2 < Gatterfunktionen < 10^3)
LSI Large Scale Integration (10^3 < Gatterfunktionen < 10^4)
GSI Grand Scale Integration (10^4 < Gatterfunktionen < 10^5)
VLSI Very Large Scale Integration (> 10^5 Gatterf.)

Typologie integrierter Schaltungen



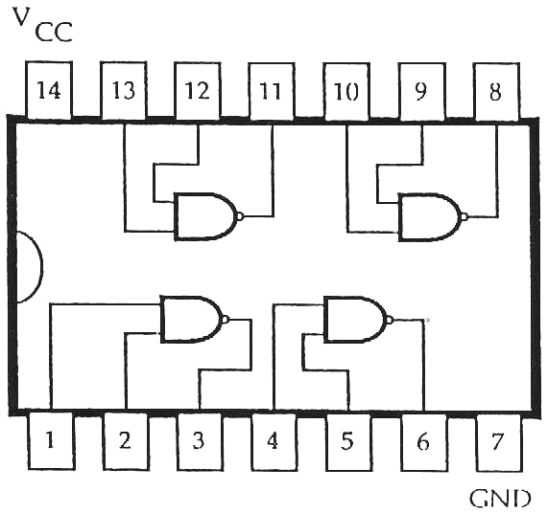
Standardschaltungen

- Entwickler sucht sich aus einem Spektrum angebotener fest vorgegebener Bausteine die passenden aus und verdrahtet sie.
- (einmal oder mehrmals) programmierbare Bausteine vorgegebener Mächtigkeit

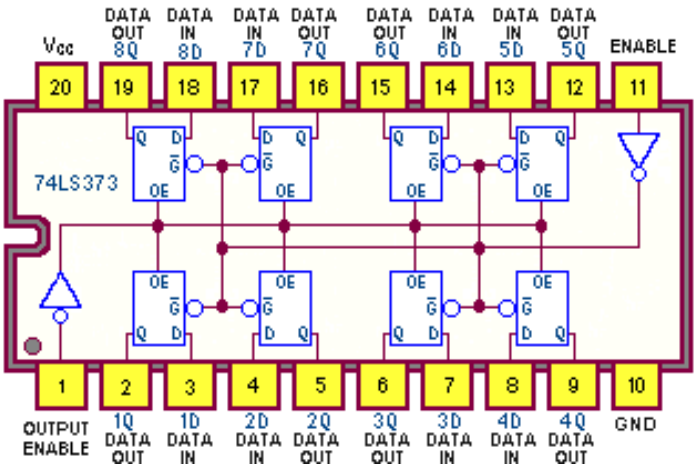
Fest vorgegebene Schaltungen

Praktisch alle digitale Grundgatter und Grundschaltungen sind in niedriger bis mittlerer Integrationsdichte verfügbar.

Beispiel: TTL-Bausteinfamilie



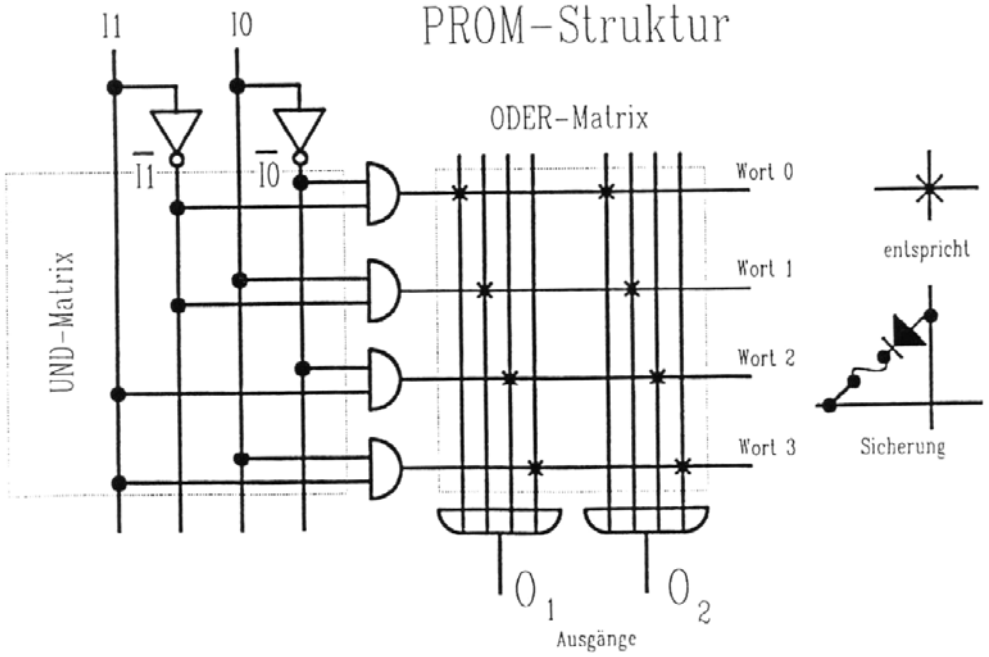
4-fach NAND-Gatter 7400



8-fach D-Latch mit Enable und Tristate-Ausgängen 74LS373

Feste, anwenderprogrammierbare Schaltungen

PROM (Programmable Read Only Memory)



UND-Matrix enthält **alle** Minterme, die fest dekodiert werden (Adressdecoder). Frei programmierbare Zuordnung der Ausgangsvektoren in der ODER-Matrix.

Beispieltechnik: Sicherung (Fuse) vor und nach der Programmierung bei einem PROM (veraltet)

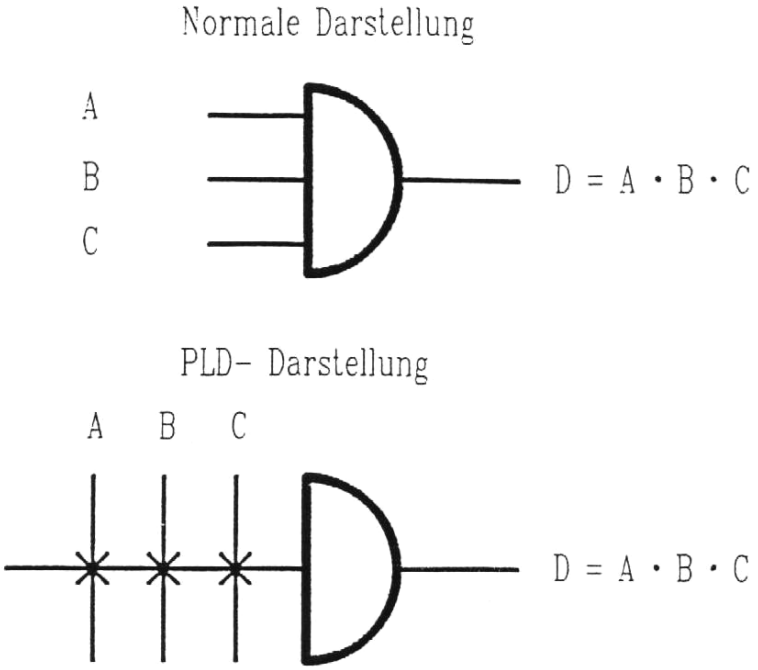
Unprogrammed Fuse

Programmed Fuse

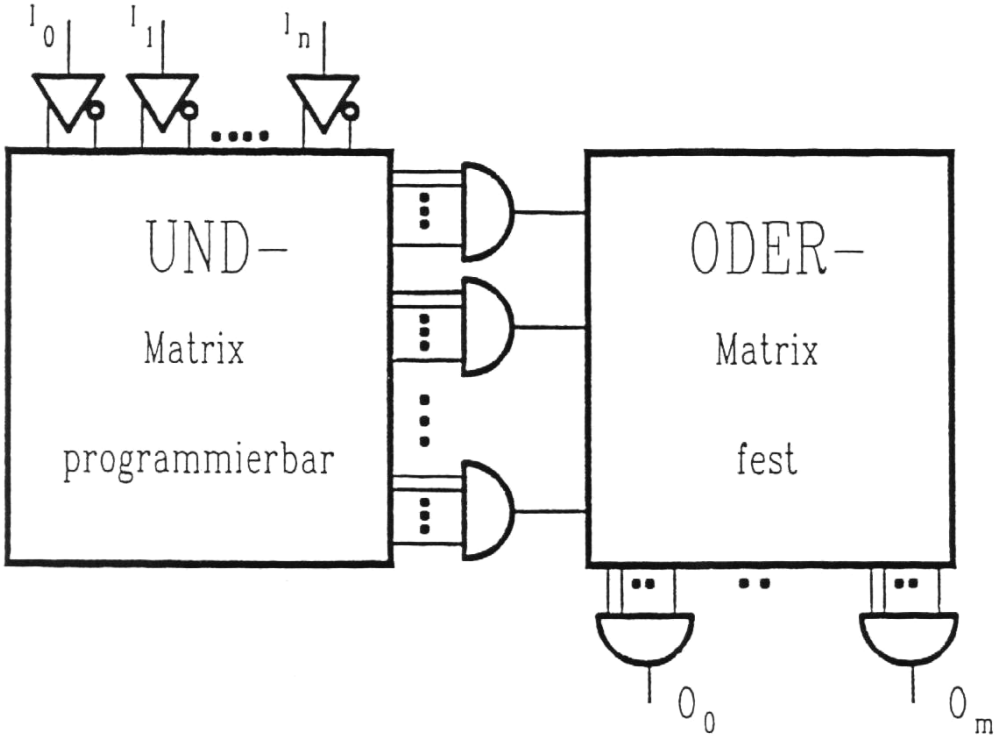
PLD (Programmable Logic Device)

Ähnlich PROM mit *anwenderprogrammierbarer* UND-Matrix, ODER-Matrix oder beidem.

Darstellung:

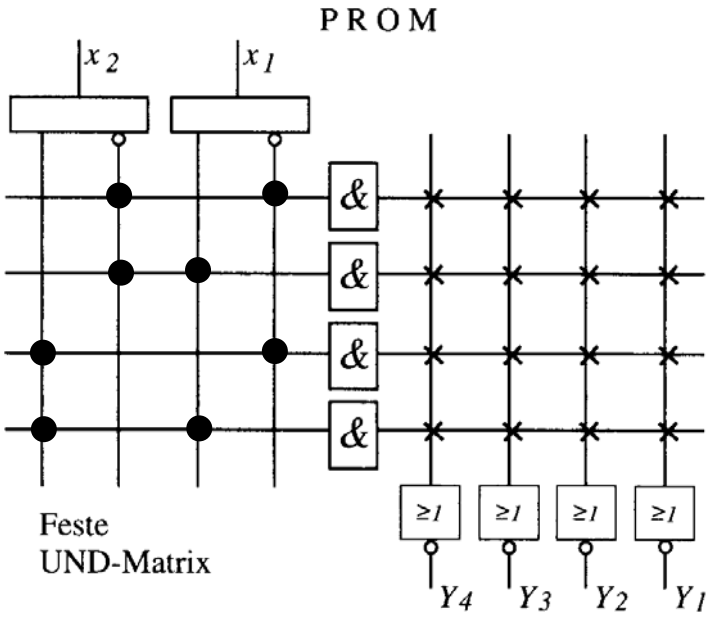


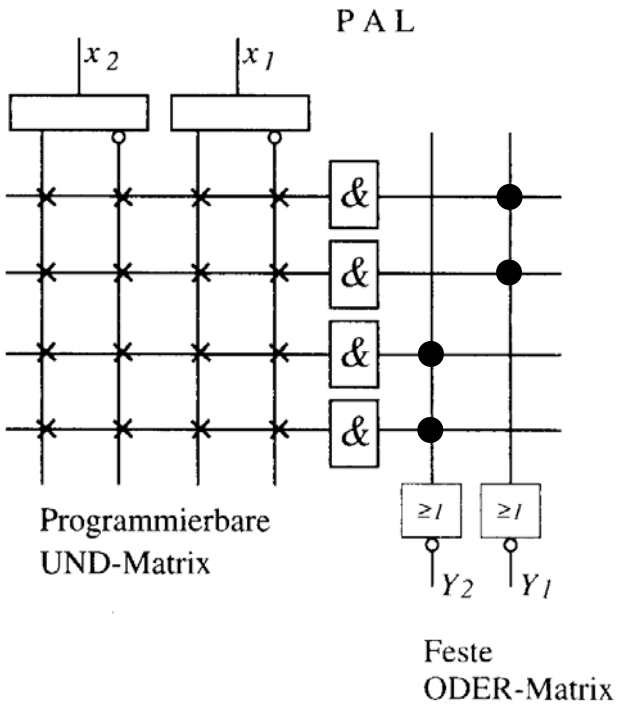
Beispiel: PAL (Programmable Array Logic) mit *programmierbarer* UND-Matrix und *fester* ODER-Matrix



Varianten programmierbarer Logikbausteine

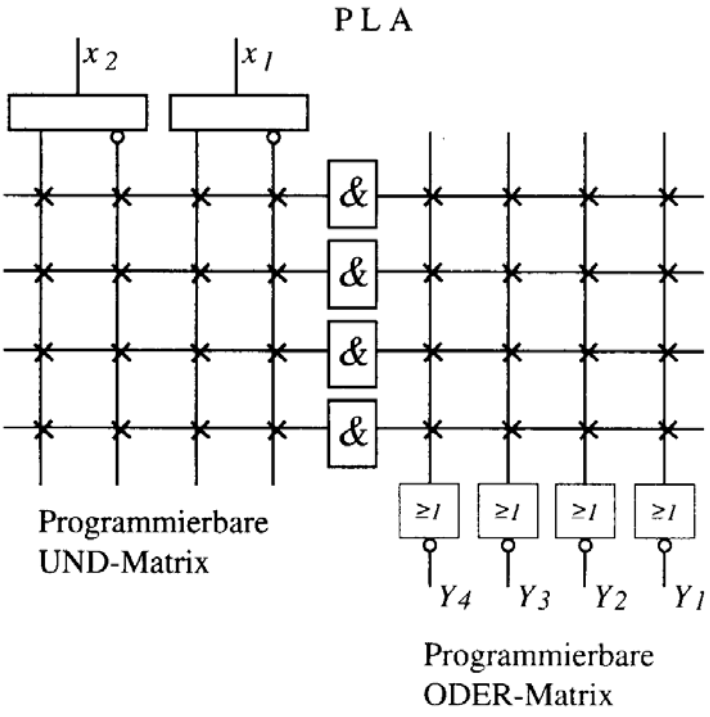
- (P)ROM ((Programmable) Read Only Memory)
- PAL (Programmable Array Logic)
- PLA (Programmable Logic Array)





● = feste Verknüpfung

X = prog.bare Verknüpfung



Anwenderspezifische Schaltungen

ASICs = Application Specific Integrated Circuits

- Entwickler trägt einen Teil oder alles zur Entwicklung der integrierten Schaltung bei.
- Gate Arrays: IC-Hersteller fertigt Gatterstruktur vor, Entwickler legt Verbindungsstruktur fest.
- Standardzellen: IC-Hersteller gibt Bibliothek von vordefinierten Zellen vor, die der Entwickler mittels CAD-System zum speziellen Entwurf kombiniert.
- voll anwenderspezifisch: Anwendungsentwickler entwirft vollständig neue integrierte Schaltung.

FPGAs = Field Porgrammable Gate Arrays

Integrierte Bausteine vorgegebener Mächtigkeit mit einer Vielzahl programmierbarer, niedrig bis mittel komplexer Logikelemente, die anwendungsspezifisch konfiguriert und verschaltet werden.

➔ Der Entwickler beschreibt den Entwurf auf einer abstrakteren Entwurfsebene (s. unten).

Automatisiert arbeitende Entwurfswerkzeuge generieren daraus Konfigurationsdateien, die auf die FPGAs geladen werden und diese dann anwendungsspezifisch programmieren.

4.8.2 Entwurf einfacher integrierter Schaltungen

Entwurfsablauf bei einfachen Schaltnetzen

(1) Aufstellen der Wahrheitstafel oder schaltalgebraischen Gleichungen

(2) Ableitung der DKNF (bzw. KKNF)

(3) Minimierung z.B. mittels KV-Diagramm oder Quine-McCluskey

(4) Zeichnen des Schaltbildes. Gängige Varianten:

- (a) Zweistufige UND/ODER- (bzw. ODER/UND-) Realisierung
- (b) Zweistufige NAND- (bzw. NOR-) Realisierung
- (c) Transformation in mehrstufige UND/ODER- (bzw. ODER/UND-) Realisierungen
- (d) Transformation in mehrstufige NAND- (bzw. NOR-) Realisierungen

oder Transformation in programmierbare Logik

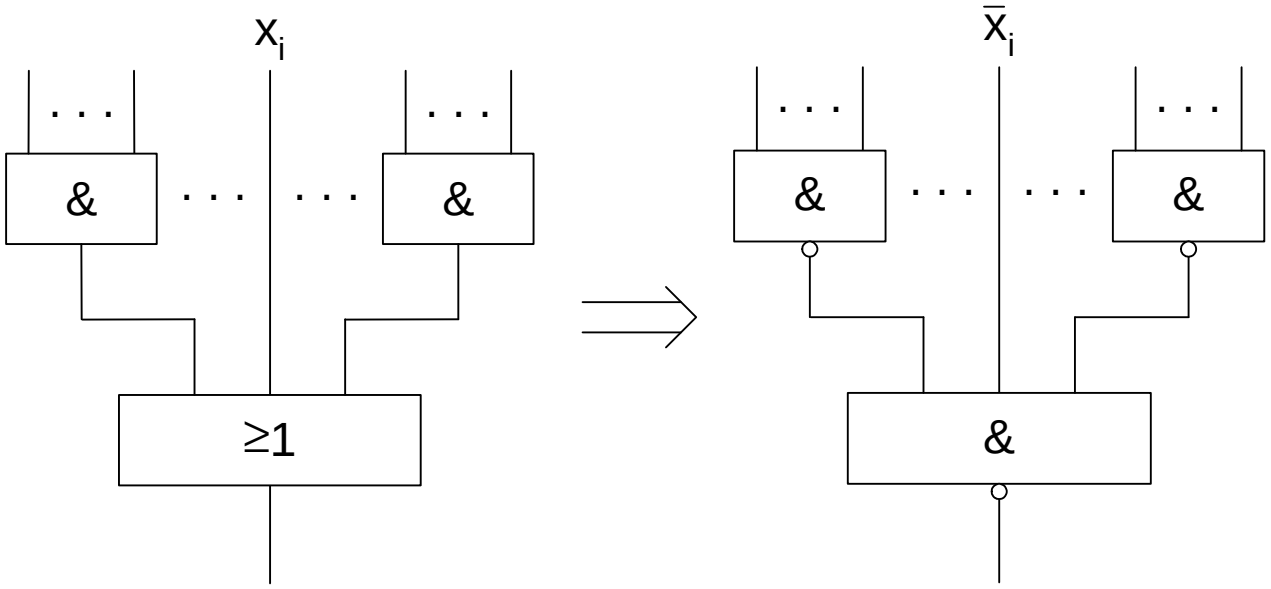
(5) Simulation und Test

(6) Physikalische Realisierung

Transformation (Schritt 4)

In der Regel *zweistufige Realisierung*

Ausgangspunkt: Zweistufige UND/ODER-Realisierung der DMF



UND/ODER NAND

UND- bzw. ODER-Gatter durch NAND-Gatter ersetzen, Struktur bleibt erhalten.

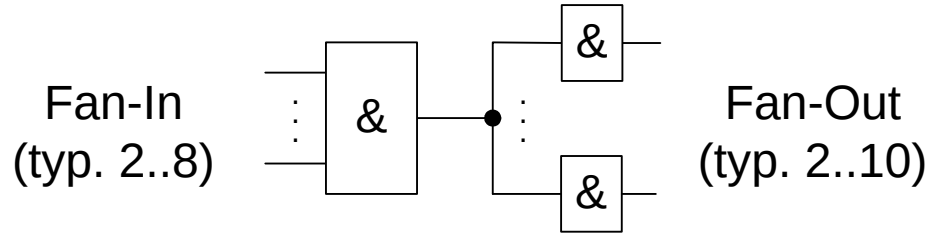
Achtung: Einzelne (direkte) Leitungen in das ODER-Gatter müssen invertiert werden (analog zu einem zum Inverter degenerierten NAND-Gatter).

ODER/UND $\Rightarrow$ NOR: Ersetzen durch NOR-Gatter, sonst ganz analog

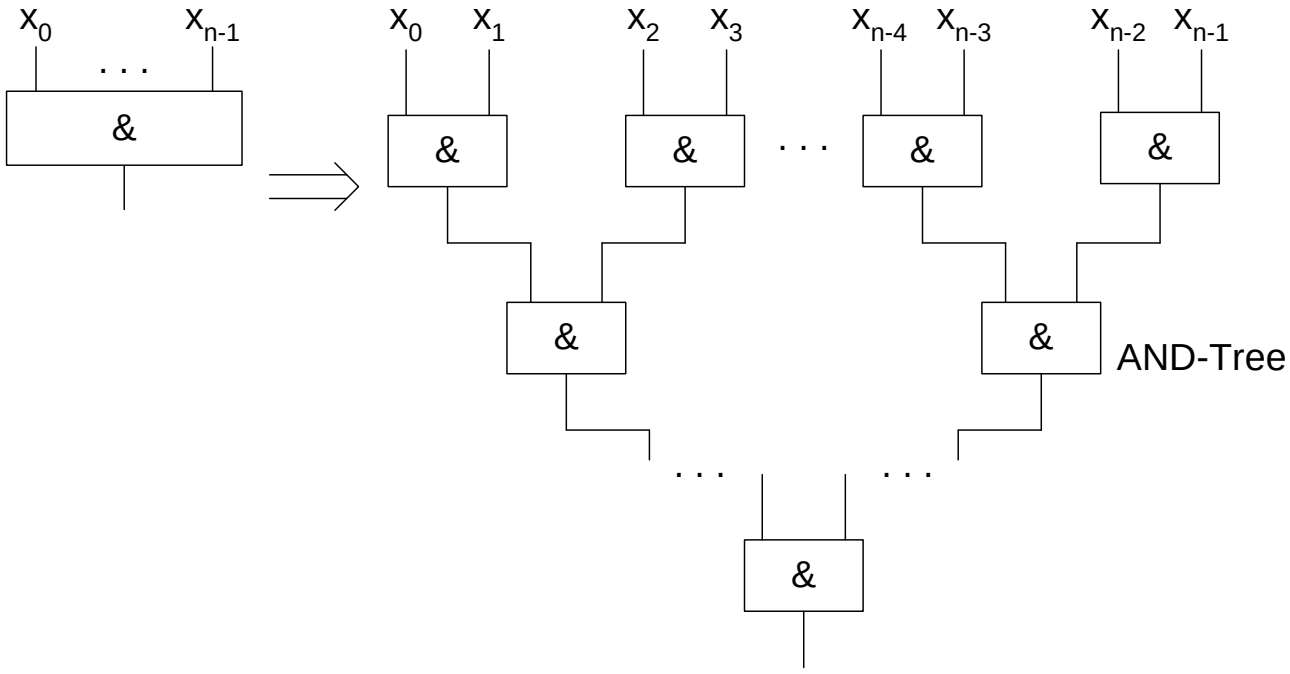
Fazit: Um eine minimierte NAND-Realisierung zu erhalten, geht man von der DMF aus, für einen minimierte NOR-Realisierung von der KMF.

Mehrstufige Realisierung

Die meisten Technologien lassen nur eine begrenzte Zahl von Eingängen je Gatter (**Fan-In**) und Anschlüssen nachgeschaltete Gatter an den Ausgängen (**Fan-Out**) zu.



„Breitere“ Gatter in einer zweistufigen Realisierungen müssen daher in mehrstufige Realisierungen zu „schmalen“ Bäumen aus Gattern transformiert werden.



$$x_0x_1x_2x_3\ldots x_{n-4}x_{n-3}x_{n-2}x_{n-1} = (((x_0x_1)(x_2x_3))\ldots((x_{n-4}x_{n-3})(x_{n-2}x_{n-1})))$$

Nutzung des Assoziativgesetzes möglich

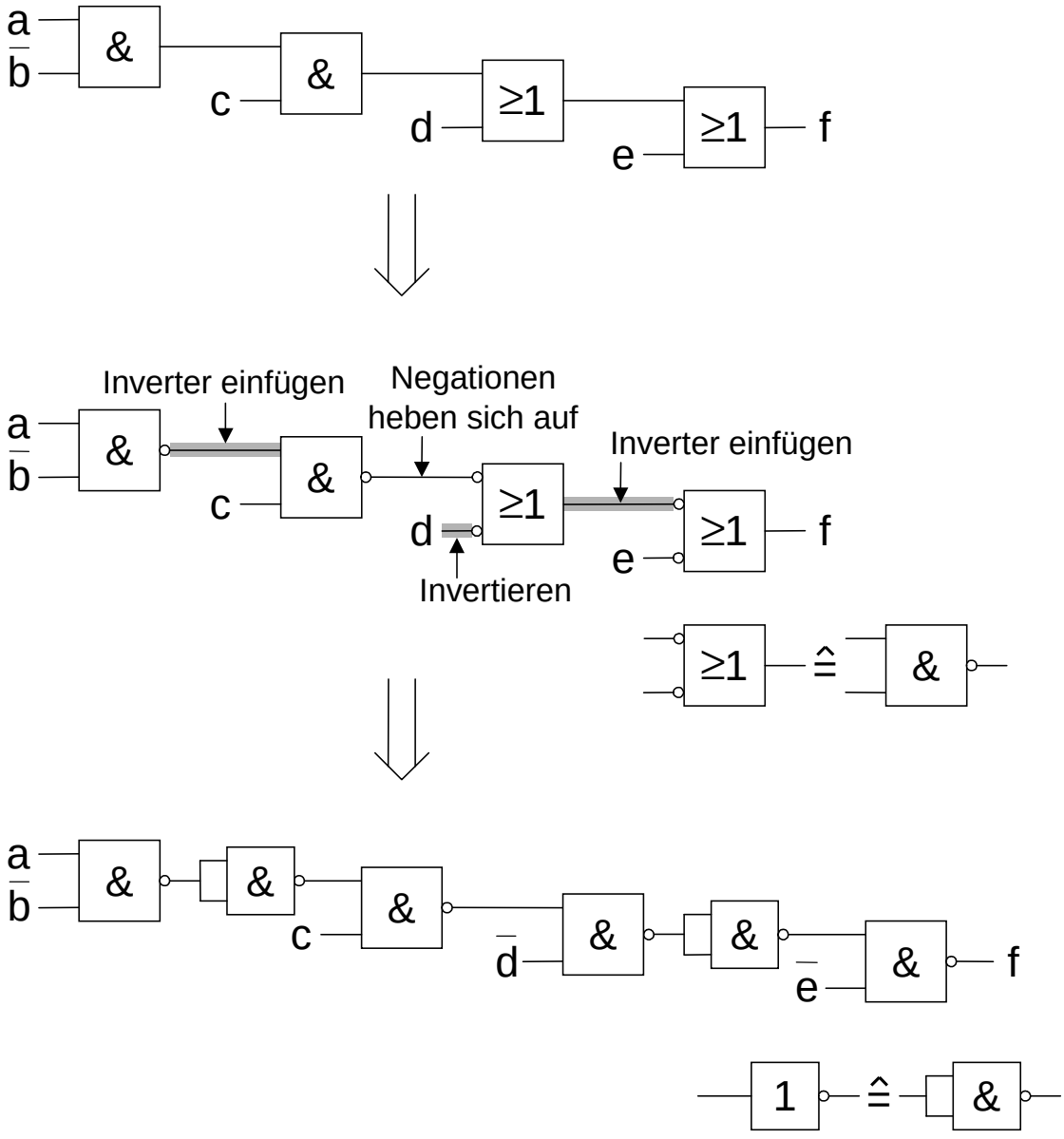
ODER-Gatter ganz analog (OR-Tree)

Funktioniert aber nicht bei NAND- bzw. NOR-Gattern, da nicht assoziativ! $\Rightarrow$ anderer Weg nötig

Transformation mehrstufiger UND-ODER-Verknüpfungen in mehrstufige NAND-Verknüpfungen

Vorgehen:

(1) Konvertiere alle UND-Gatter und alle ODER-Gatter in NAND-Gatter (Anfügen von Negations-Blasen am Ausgang von UND- bzw. den Eingängen von ODER-Gattern).
(2) Treibt ein negierter Ausgang einen negierten Eingang, sind keine weiteren Schritte erforderlich.
(3) Treibt ein negierter Ausgang einen nicht negierten Eingang oder umgekehrt, muss ein Inverter eingeführt werden.
(4) Treibt eine Eingangsvariable einen invertierten Eingang, muss sie invertiert werden.



Alternative: Transformation von zweistufig UND-ODER nach mehrstufig UND/ODER alternierend mit ODER am Ende (Ausklammern der DMF).

Dann können direkt die UND- und ODER-Gatter in NAND-Gatter verwandelt werden (vgl. mehrstufige Transformation)

Transformationen von ODER/UND-Verknüpfungen in NOR-Verknüpfungen erfolgen ganz analog (Dualitätsprinzip)

Weitere Realisierungsmöglichkeiten von Schaltnetzen:

- Decoder (mit Zusatzgattern) etc.
- Multiplexer
- Programmierbare Logik (PLDs oder FPGAs)

4.8.3 Entwurf komplexer Schaltungen

Wenn die Schaltung (Wahrheitstafel) zu groß wird, erfolgt normalerweise eine Dekomposition in sinnvolle Subnetze (vgl. z. B. bitstellenweise Addition bei N-Bit Addierer).
Der Entwurf der Subnetze geschieht dann wie für einfache Schaltnetze.
Danach erfolgt wieder die Komposition zu einem Gesamtnetz.

Problem: Die Laufzeiten können stark ansteigen, daher evtl. Zusatzlogik zur Beschleunigung (vgl. z. B. Carry-Look-Ahead).

Entwurf großer, komplexer digitaler Systeme
Beim Entwurf großer digitaler Systeme spielen neben dem richtigen funktionalen Verhalten weitere Aspekte eine wichtige Rolle:

- Beherrschung immer komplexerer Systeme
- Entwicklungszeit
- Entwicklung im Team
- Entwurfssicherheit
- Wiederverwendbarkeit
- Dokumentation
- Anpassbarkeit an technologische Entwicklungen
- . . .

⇒ **Komplexere Hardwareschaltungen können nicht mehr manuell auf der bisherigen Beschreibungsebene (Detaillierungsgrad) beschrieben werden.**

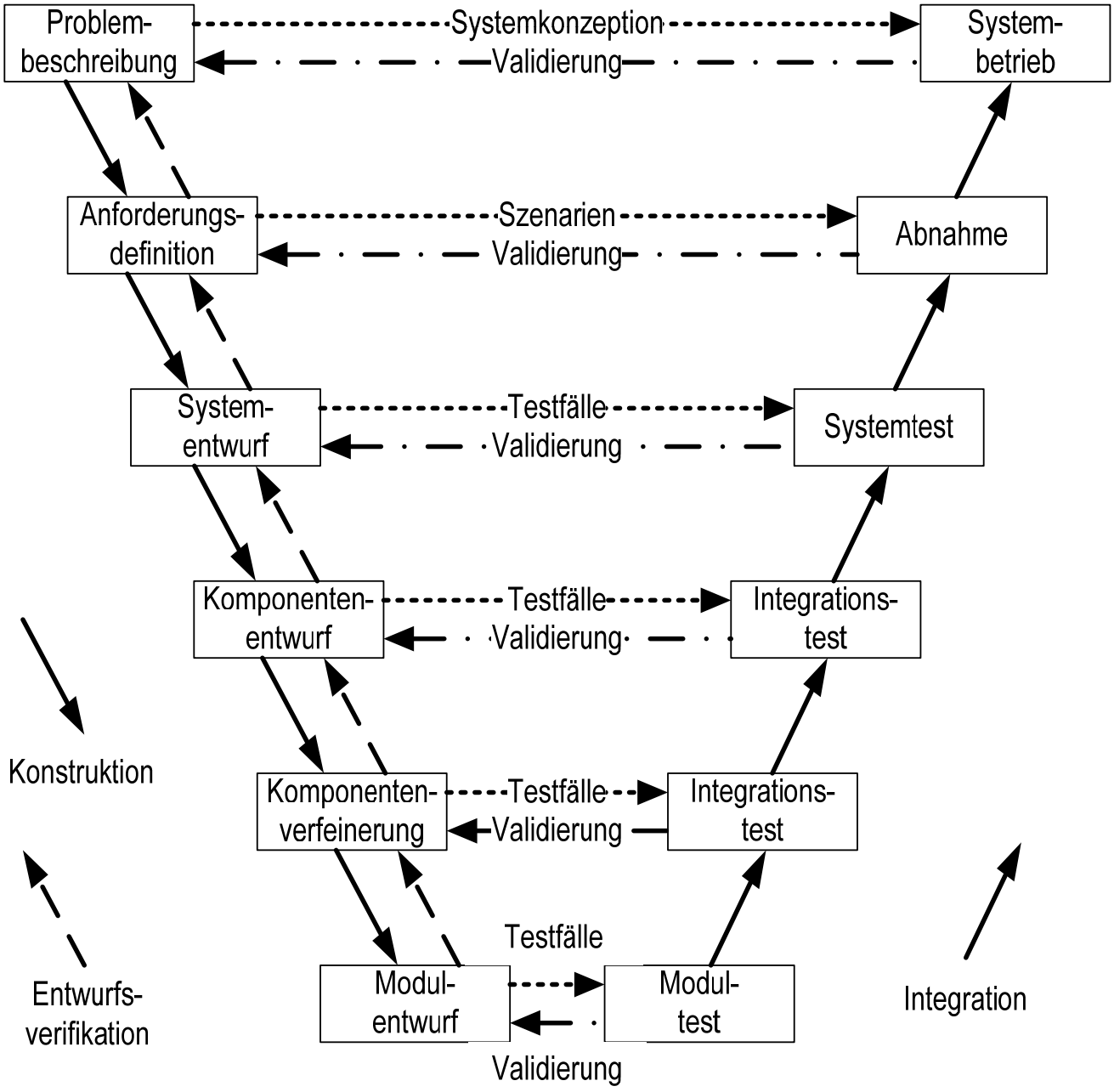
➔ Lösungsprinzip: abstrahierte Beschreibung (ggf. auf verschiedenen Abstraktionsebenen) und

automatisierte Umsetzung auf Gatterebenen (Netzliste)

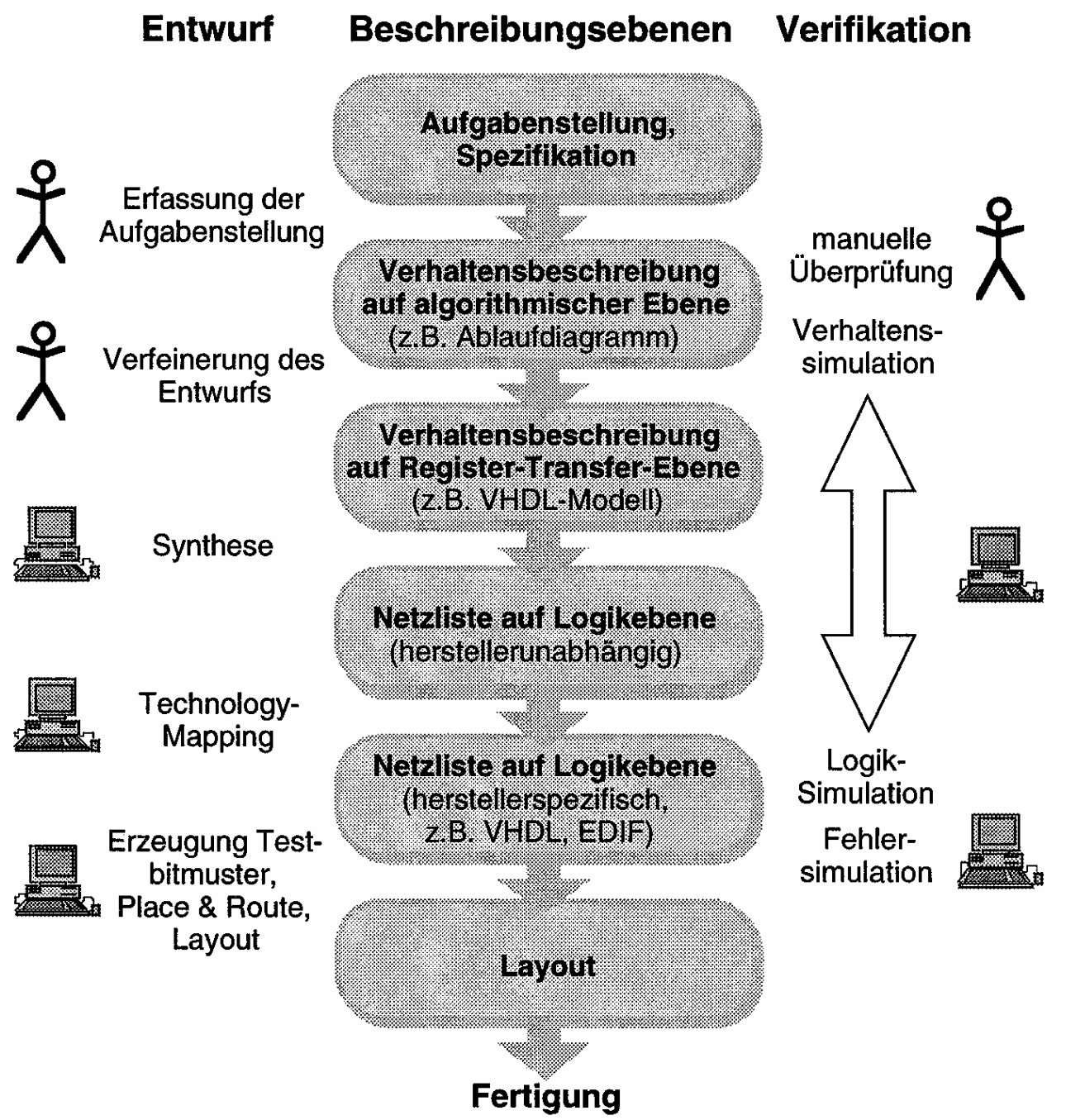
Idealer Entwurfsablauf

Daher wird das Systemverhalten meist auf einer abstrakteren Ebene beschrieben und soweit verfeinert, bis es automatisch durch Entwurfs- und Synthesewerkzeuge optimiert und in die zu realisierende digitale Schaltung umgesetzt werden kann.

- Ausgehend von der Spezifikation auf Systemebene wird die Schaltung partitioniert („Funktionale Dekomposition“).
- Der Entwurf wird dann Stück für Stück feiner strukturiert bzw. detailliert.



Prinzipieller Entwurfsablauf in VHDL

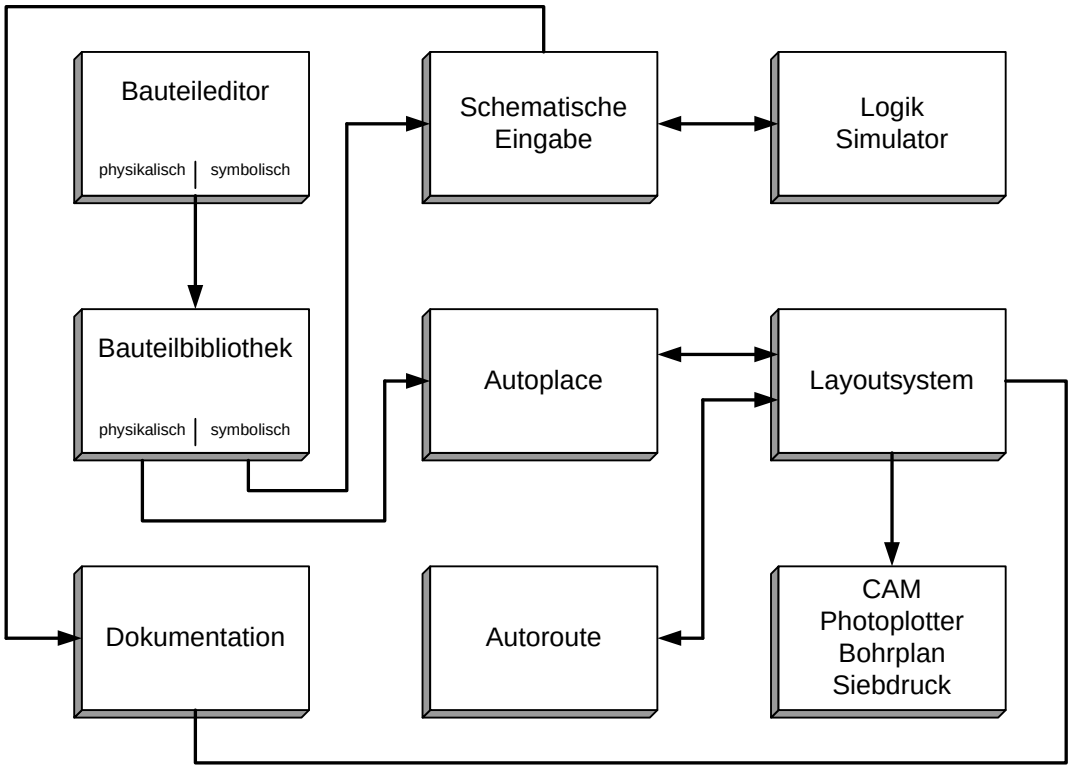


CAD-Systeme

CAD-Systeme (Computer Aided Design) enthalten neben schematischer und/oder textueller Eingabe i. Allg. leistungsfähige Simulatoren zum Austesten der Schaltung vor der sehr kostenintensiven**!!!** Fertigung von ICs.

Die Ausgabe liefert dann Daten für die IC-Fertigung (z.B. Gate Arrays) oder zur Programmierung von Bausteinen (z. B. PLDs, FPGAs) und zum Testen der gefertigten ICs.

Aufbau eines CAD-Systems für die Entwicklung von Schaltungen



<u>Vorteile:</u>
- Verwendung von Bibliotheken
- Prüfen des Entwurfs (Funktion <u>und</u> Zeitverhalten) vor der Fertigung des ICs
- leichte Änderbarkeit und Fehlersuche
- autom. Dokumentation und Testmustergen.

Beispiel: Entwurfshilfsmittel für PLDs

WinCUPL von Atmel als Beispielentwicklungsumgebung für anwenderprogrammierbare Schaltungen mäßiger Komplexität:

- geeignet für verschiedene PLD-Bausteine
- enthält:
  - Editor für
    - Pinbelegung
    - Logikfunktionen
  - Compiler mit Minimierung
  - Logik-Simulator
- verschiedene Eingabeformate
  - logische Gleichungen
  - Wertetabelle
- verschiedene Ausgaben
  - minimierte Gleichungen
  - Dokumentation
  - Programmierfiles (Standardformat: JEDEC)
- unterstützt Makros
- verschiedene äquivalente Beschreibungsformen

Funktionsbeschreibung

Ein-Ausgangsspezifikation

```
Device g16v8a;
/*************** INPUT PINS ******/
PIN 1 = x0;  /* Eingang 2 hoch 0 */
PIN 2 = x1;  /* Eingang 2 hoch 1 */
PIN 3 = x2;  /* Eingang 2 hoch 2 */
PIN 4 = x3;  /* Eingang 2 hoch 3 */

/******** OUTPUT PINS ******/
PIN 12 = y0;  /* Ausgang 1 */
PIN 13 = y1;  /* Ausgang 2 */
PIN 14 = y2;  /* Ausgang 3 */
PIN 15 = y3;  /* Ausgang 4 */
```

Spezifikation der Logik

```
/* typical logical equation */
Y0 = !x0 & x1 # x0 & !x1;

/* typical truth table in mixed format*/
TABLE  [x3..0] => [y3..0] {
    'h'0 => 'b'0000;
    'h'1 => 'b'0001;
    'h'2 => 'b'0011;
    'h'3 => 'b'0010;
        . . .
    'h'F => 'b'1000; }
```

Spezifikation eines Automaten (Prinzip)

```
SEQUENCE state_Var_list  {
   PRESENT state_n0
      IF (condition1) NEXT state_n1;
      IF (condition2) NEXT state_n2;
      DEFAULT         NEXT state_n0;
   PRESENT state_n1
      NEXT state_n2;
   . . .
```

Nützliche Links zu *WinCUPL*:

Download von *WinCUPL*:

http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2759

Anschauliche Einführungen:

http://www.informatik.fh-lausitz.de/sreichel/Digitaltechnik/Uebung/WinCuplAnleitung.pdf (nicht mehr verfügbar)

http://www.rexfisher.com/Downloads/CUPL Tutorial.htm

Umfassende Anleitung des Tool-Herstellers *Atmel*:

http://www.atmel.com/dyn/resources/prod_documents/DOC0737.PDF

Beschreibung auf Register-Transfer-Ebene

Auf Register-Transfer-Ebene/Level (RTL) erfolgt die Beschreibung durch Transfer von und Operationen auf zu verarbeitenden Daten. Das zeitliche Verhalten wird auf Taktebene durch Einbeziehung von Taktsignalen berücksichtigt.

Die Verhaltensbeschreibung erfolgt z. B. durch endliche Automaten (s. Kap. 6) oder Register-Transfer-Sprache (s. Kap. 7).

Beispiel: Umsetzung zweier Assemblerbefehle

```
FETCH:   AR<-PC;
         read M;
         IR<-DR, PC<-PC+1|
         . . .
         if IR=JMP then goto JUMP        else
         if IR=BRA then goto BRANCH      else
         . . . ;

JUMP:    read M;
         TEMP<-DR | PC<-PC+1;
         AR<-PC | PC(15:8)<-TEMP;
         read M;
         PC(7:0)<-DR | goto FETCH;

BRANCH:  PC<-PC+DR | goto FETCH;
         . . .
```

Auf der Register-Transfer-Ebene werden alle für die Funktion notwendigen Elemente (z. B. Register, ALU) und deren Verschaltung durch Signale bestimmt.

Daraus wird die Struktur der Schaltung und das Zeitverhalten (auf Taktebene) abgeleitet.

Hardwarebeschreibungssprache VHDL

Eine Hardwarebeschreibung sollte es erlauben, einerseits den Entwurf einer (komplexen) digitalen Schaltung schnell und komfortabel (auf hoher Abstraktionsebene) durchzuführen, aber auch andererseits die Laufzeit kritischer Schaltungsteile (auf einer niedrigen Entwurfsebene) optimieren zu können.

Am weitesten verbreitet ist hier die Spezifikation komplexer digitaler Systeme durch **VHDL** (VHSIC Hardware Description Language; VHSIC: Very High Speed Integrated Circuit).

VHLD ist so etwas wie eine „Programmiersprache“ für Hardware und syntaktisch an die Programmiersprache ADA angelehnt, aber um hardware-spezifische Elemente erweitert.

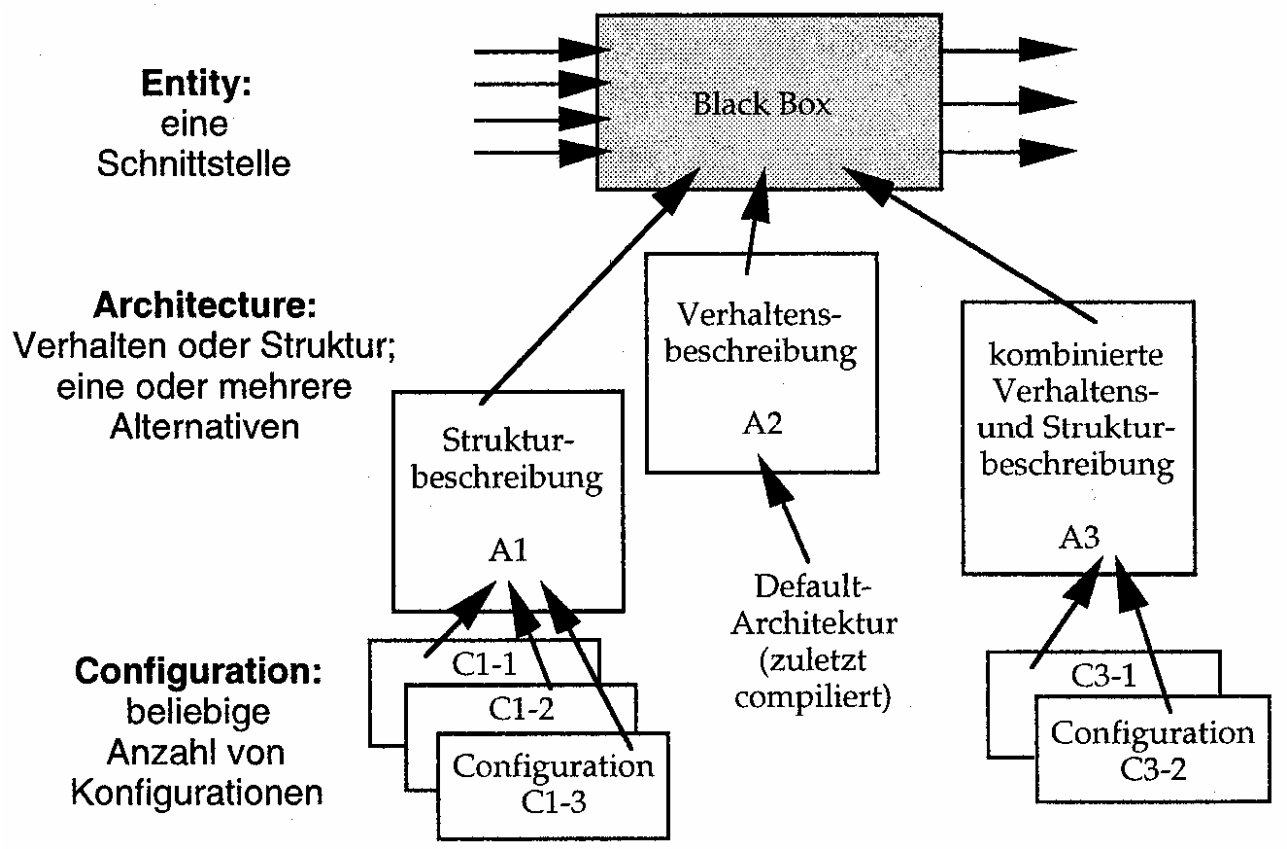
Ein (digitale) Schaltung wird in VHDL grundsätzlich als aus **nebenläufigen Einheiten** bestehendes System modelliert. Zwischen den einzelnen Schaltungseinheiten werden (physikalische) **Signale** ausgetauscht. Darin unterscheidet sich VHDL grundlegend von anderen Programmiersprachen (Assignment-Statement).

Beispiel für nebenläufige Anweisungen: Halbaddierer

```
ARCHITECTURE b_par OF halfadder IS
BEGIN
  sum <= a XOR b;
  c <= a AND b;
ND ARCHITECTURE b_par;
```

I. d. R. wird auch noch zusätzlich das Timing angegeben, d.h. bei welchem Ereignis oder zu welchem Zeitpunkt Signale ihren Wert ändern (können).

Aufbau einer Hardwarebeschreibung mit VHDL



VHDL unterstützt durch die Modularisierung auf verschiedenen Hierarchieebenen und verschiedenen Modellierungsarten den Entwurf komplexer digitaler Systeme:

- Strukturmodellierung
- Verhaltensmodellierung
- gemischte Modellierung

VHDL erlaubt auch, für die entworfene Schaltung ebenfalls eine Testumgebung (Testbed) in der gleichen Weise wie die Schaltung zu beschreiben, in der diese dann für Test- und Simulationszwecke eingebunden werden kann.

Näheres siehe

- Hilfsmaterial „*VHDL-Übersicht*“

- Vorlesung „*Entwurf digitaler Systeme*“