

Hardware-Entwurfstechnik

Heinrich Krämer

**Hochschule für Technik, Wirtschaft und Kultur Leipzig (FH)
Fachbereich Informatik, Mathematik und Naturwissenschaften**

A thick horizontal bar, split into black and grey sections, runs across the bottom of the slide.

1 Einleitung

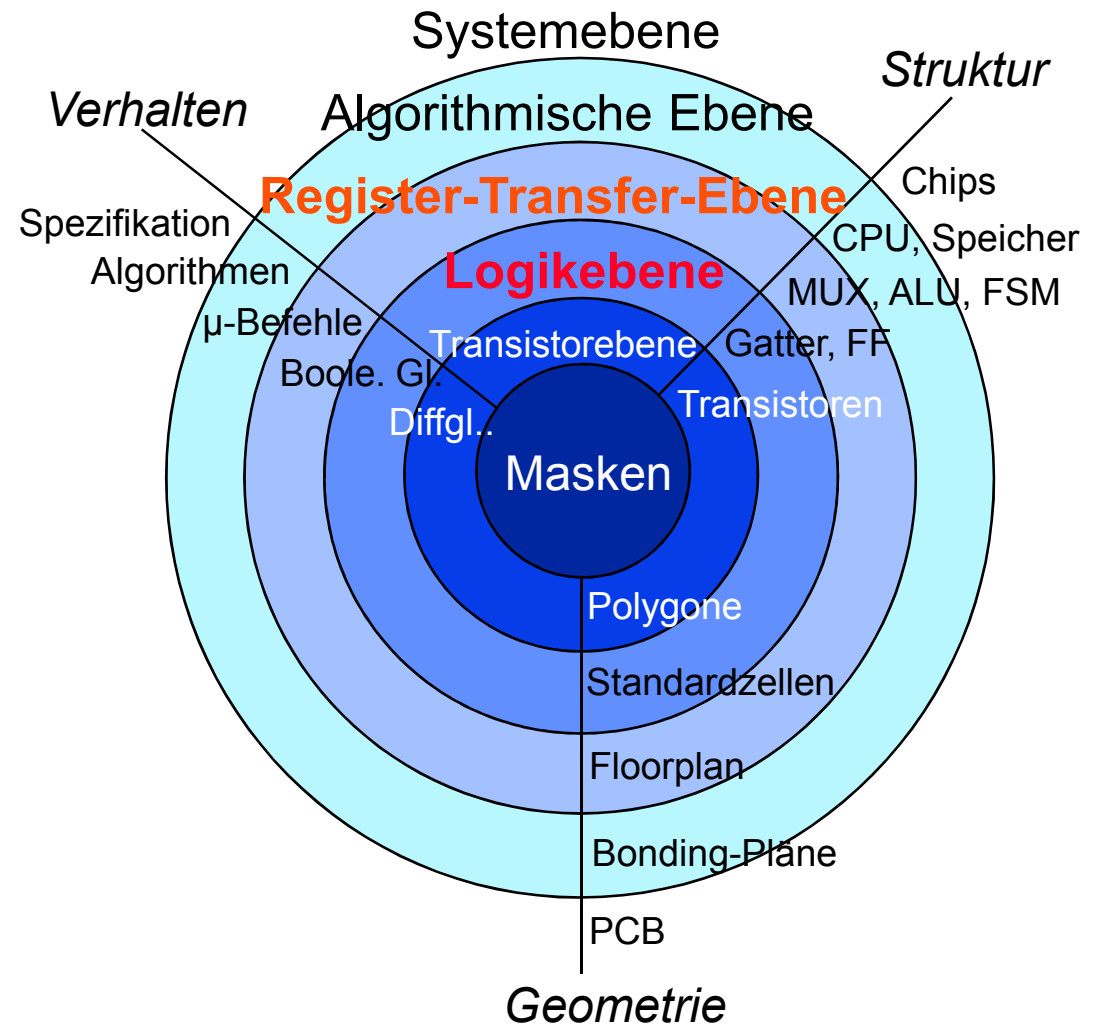
Entwurf von Schaltungen

• Ebenen

- Systemebene
- Algorithmische Ebene
- Register-Transfer-Ebene
- Logikebene
- Transistorebene
- Masken/Layout

• Aspekte

- Verhalten
- Struktur
- Geometrie



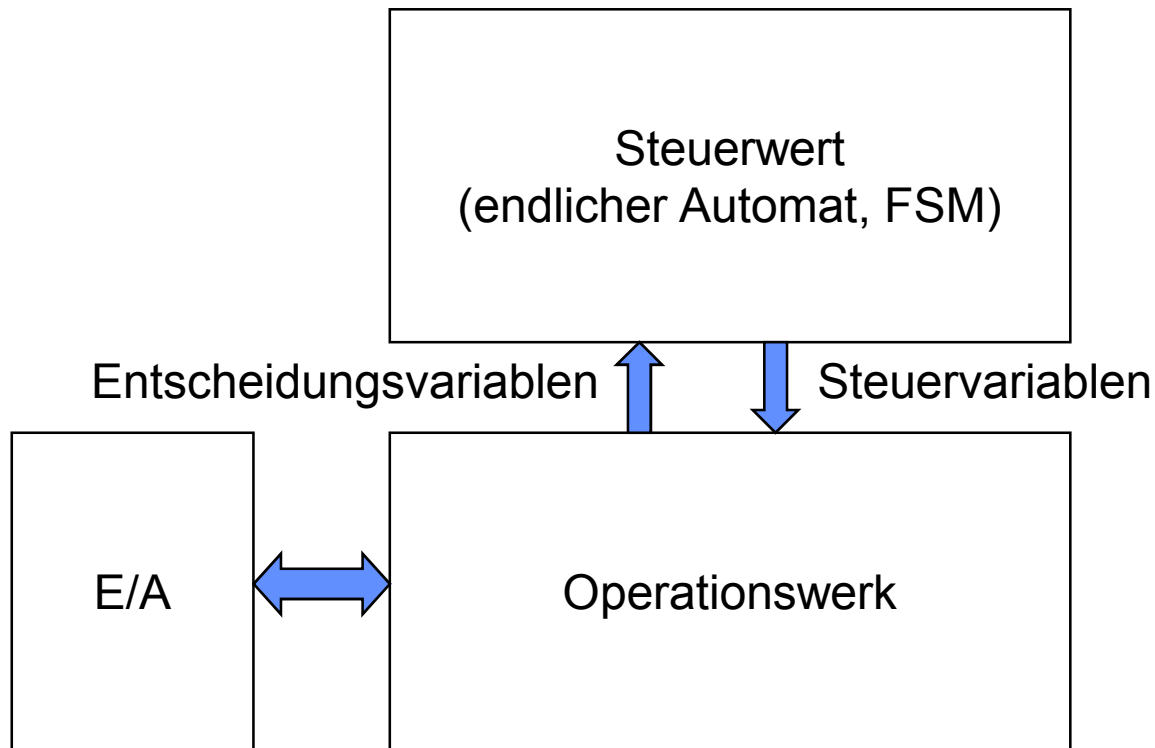
1 Einleitung

Entwurf auf der Register-Transfer-Ebene

- **Trennung Operations- / Steuerwerk**
- **Komponenten**
 - Register, Flipflops,
 - Addierer, ALU, Multiplizierer, ...
 - Logikblöcke
 - FSM (Finite State Machine)
- **Das Verhalten ist gegeben durch boolesche Gleichungen.**
- **Das Ziel ist die Umsetzung der Komponenten in eine Implementierung (Struktur) aus einzelnen Gattern und Flipflops.**

1 Einleitung

Register-Transfer-Ebene



Kennzeichen:

unregelmäßige Struktur
der Logik (Random logic)

Kennzeichen:

regelmäßige Struktur der
Verarbeitungseinheiten

zum Teil unregelmäßige
Struktur der Verbindungslogik

1.1 Darstellung von Logikfunktionen

Def. 1.1: Boolesche Funktion (Schaltfunktion, Logikfunktion)

Sei $\mathbf{B} = \{0, 1\}$ und $\mathbf{B}^* = \{0, 1, \text{Ⓚ}\}$. Eine boolesche Funktion ist eine Funktion $F : \mathbf{B}^m \rightarrow \mathbf{B}^n$. Hierbei bezeichnet Ⓚ eine nicht spezifizierte Ausgabe (Don't care). F wird als **unvollständig spezifizierte** boolesche Funktion bezeichnet. Nimmt F nur Werte aus \mathbf{B}^n an, so handelt es sich um eine **vollständig spezifizierte** boolesche Funktion. Eine Funktion $f : \mathbf{B}^m \rightarrow \mathbf{B}$ ($n = 1$) wird als **einwertige** boolesche Funktion bezeichnet. Für $n > 1$ wird die Funktion **Bündelfunktion** genannt.

Eine Bündelfunktion kann auch als n einwertige Funktionen $F = (f_1, f_2, \dots, f_n)$ über dem gleichen Definitionsbereich aufgefaßt werden.

Für jede Komponente f_i $i = 1 \dots m$ von F bezeichnet $f_i^{\text{ON}} \subseteq \mathbf{B}^m$ die Teilmenge für die $f_i(x) = 1$ gilt, $f_i^{\text{OFF}} \subseteq \mathbf{B}^m$ die Teilmenge für die $f_i(x) = 0$ gilt und $f_i^{\text{DC}} \subseteq \mathbf{B}^m$ die Teilmenge für die $f_i(x) = *$ gilt. Eine unvollständig spezifizierte Funktion kann also durch drei vollständig spezifizierte Funktionen dargestellt werden

1.1 Darstellung von Logikfunktionen

Boolesche Variablen x_1, x_2, x_3, \dots und deren Komplemente $\overline{x}_1, \overline{x}_2, \overline{x}_3, \dots$

Die Variablen können die Werte 0 und 1 annehmen.

$F = (f_1(x_1, x_2, \dots, x_m), f_2(x_1, x_2, \dots, x_m), f_3(x_1, x_2, \dots, x_m), \dots, f_n(x_1, x_2, \dots, x_m))$

wobei die f_i einwertige boolesche Funktionen sind.

Faktorierte Form

$$f = a \wedge (b \vee c) \vee c \wedge (d \vee e \vee f)$$

Disjunktive Form(DF)

$$f = a \wedge b \vee a \wedge c \vee c \wedge d \vee c \wedge e \vee c \wedge f = a b \vee a c \vee c d \vee c e \vee c f$$

Der durch UND verknüpfte Teilausdruck heißt (UND-)Term.

Ist in einem Term jede Variable oder deren Komplement enthalten so handelt es sich um einem *Minterm*. Ist eine Funktion nur durch Minterme dargestellt und sind die Variablen in einer festen Reihenfolge dargestellt, so handelt es sich um eine *disjunktive Normalform* (DNF).

Konjunktive Form(KF)

$$f = (a \vee c) (c \vee b) (a \vee d \vee e \vee f) (b \vee c \vee d \vee e \vee f)$$

Die ODER-ausdrücke heißen Maxterme. Hier läßt sich ebenfalls eine Normalform, die *konjunktive Normalform* (KNF) herstellen.

1.1 Schaltalgebra

Rechenregeln für boolesche Variablen

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

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

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

$$a \vee b = b \vee a$$

$$a \wedge a = a$$

$$a \vee a = a$$

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

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

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

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

$$a \wedge 1 = a$$

$$a \vee 0 = a$$

$$a \wedge \bar{a} = 0$$

$$a \vee \bar{a} = 1$$

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

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

Assoziativgesetze

Kommutativgesetze

Idempotenzgesetze

Distributivgesetze

Absorptionsgesetze

Neutrale Elemente

Inverse Elemente

DeMorgan-Regeln

1.1 Darstellung von Logikfunktionen

1.1.2 Wahrheitstabellen (Espresso-Format)

Bedeutung

- .i Anzahl der Eingänge
- .o Anzahl der Ausgänge
- .p Anzahl der Logikzeilen
- .e Ende
- 0 Komplement
- 1 Variable / Funktion ist 1
- - Don't care
- ~ Funktion ist 0

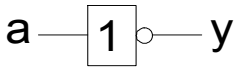
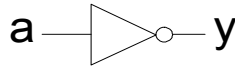
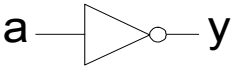
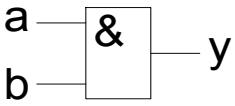
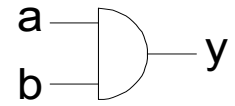
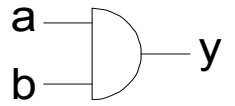
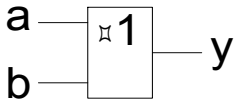
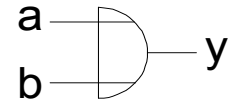
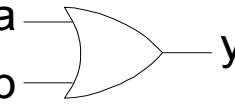
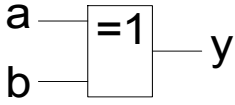
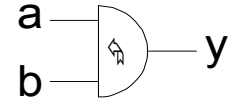
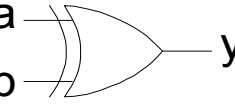
Eingangsteil

Ausgangsteil

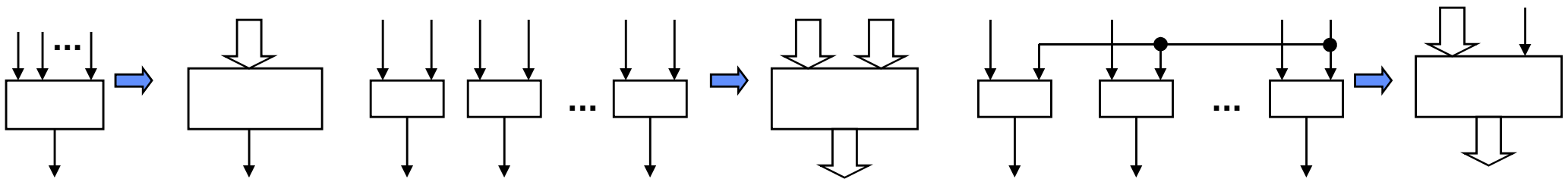
```
.i 5
.o 28
.p 17
00000 ~~~~~~1
-00-0 ~~~~~~1~~~~~
0-100 ~~~1~~~~~1~~~1~~~~~
1-0-0 ~~~1~~~~~1~~~~~
1-01- ~~~1~1~~~~~1~~~~~
0-001 ~~~1~~~~~1~~~~~
--010 ~~~1~~~1~~~~~1~~~~~
11000 -~~~~~--~~~~~
10011 -~~~~~--~~~~~
11011 -~~~~~--~~~~~
00100 ~~-~~~~~--~~~~~
10000 -~~~~~--~~~~~
00001 ~~~~~~--~~~~~
01010 -~~~~~--~~~~~
00010 -~~~~~--~~~~~
01000 -~~~~~--~~~~~
01001 ~~~~~~--~~~~~
.e
```


1.1 Schaltzeichen

Logikgatter

Funktion	Schaltzeichen (DIN)	Schaltzeichen (DIN alt)	Schaltzeichen (USA)
$y = \bar{a}$			
$y = a \cdot b$			
$y = a \vee b$			
$y = a \oplus b = \bar{a}b \vee a\bar{b}$			

Vereinfachungen



1.1 Darstellung von Logikfunktionen

Hierarchische Beschreibungen

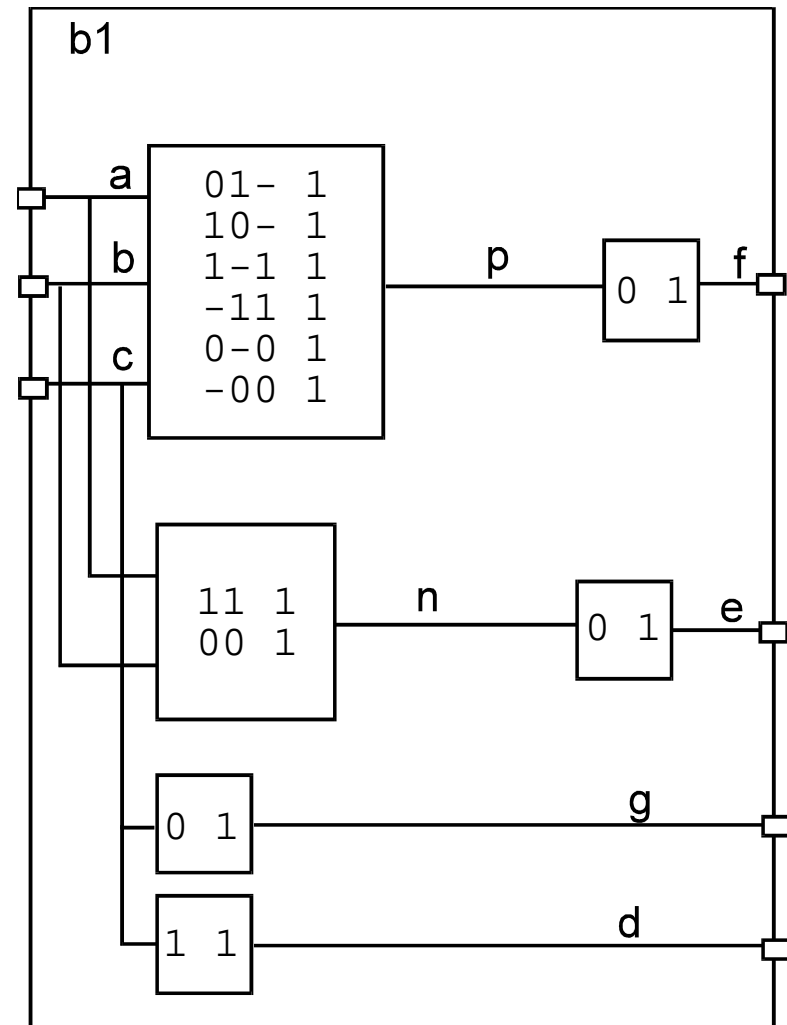
- Schaltungsedatoren bieten oft die Möglichkeit eine Schaltung hierarchisch zu beschreiben.
 - Die Basis für die Beschreibung bilden die vordefinierten Grundgatter einer Bibliothek.
 - Ein Block besteht aus Grundgattern oder anderen Blöcken, die bereits entworfen sind.
 - Jeder selbst entworfene Block besteht aus
 - Einer internen Struktur und
 - einem Symbol
- Gemischte Beschreibung (Mixed Level)
Hierbei ist es möglich, Grundgatter auch funktional (z. B. Logikfunktion) oder algorithmisch (z. B. FSM) zu beschreiben.

1.1 Darstellung von Logikfunktionen

1.1.4 Textuelle Form

BLIF (Berkeley Logic Interchange Format)

```
.model b1
.inputs a b c
.outputs d e f g
.names n e
0 1
.names p f
0 1
.names c g
0 1
.names a b n
11 1
00 1
.names a b c p
01- 1
10- 1
1-1 1
-11 1
0-0 1
-00 1
.names c d
1 1
.end
```



1.1 Darstellung von Logikfunktionen

1.1.5 VHDL (VHSIC Hardware Description Language)

- **Spezifikation von Schaltungen**
 - Einheitliche Dokumentation von Schaltungen
 - Austauschbarkeit von Designs
- **Simulation**
 - Partieller Entwurf
 - Systementwurf
- **Synthese**
 - Automatische Umsetzung aus verschiedenen Abstraktionsebenen
- **Kennzeichen der Sprache**
 - Prozedural mit Erweiterungen für den Hardware-Entwurf
 - Strenge Typisierung
- **Sprachumfang deckt System- bis zur Logikebene ab**
- **Keine Geometrieinformation**

1.1.5 VHDL

Hier nur Strukturelle Beschreibung Aufbau für eine Komponente

Verwendete Bibliotheken

Entity-Deklaration

Name der Komponente

Schnittstellen-Signale

Architektur-Deklaration

Beschreibung des internen Aufbaus

Konfigurations-Deklaration

Bindung der verwendeten
Komponenten an eine Bibliothek

```
library IEEE;
    use IEEE.std_logic_1164.all;
library ECPD10_IND;
    use ECPD10_IND.components.all;

entity TRI2 is
    Port (  A : In    std_logic_vector (1 downto 0);
           EN : In    std_logic;
           Y  : Out   std_logic_vector (1 downto 0) );
end TRI2;

architecture SCHEMATIC of TRI2 is
begin
    I_1 : LIBTRI3
        Port Map ( A=>A(3), ENB=>EN, Y=>Y(0) );
    I_2 : LIBTRI3
        Port Map ( A=>A(2), ENB=>EN, Y=>Y(1) );
end SCHEMATIC;

configuration CFG_TRI2_SCHEMATIC of TRI2 is
    for SCHEMATIC
        for I_1, I_2: LIBTRI3
            use entity ECPD10_IND.LIBTRI3(FTSM);
        end for;
    end for;
end CFG_TRI4_SCHEMATIC;
```

1.2 Realisierung von Logikfunktionen

Technische Realisierung von Logikfunktionen

- **Programmierbare Logik**

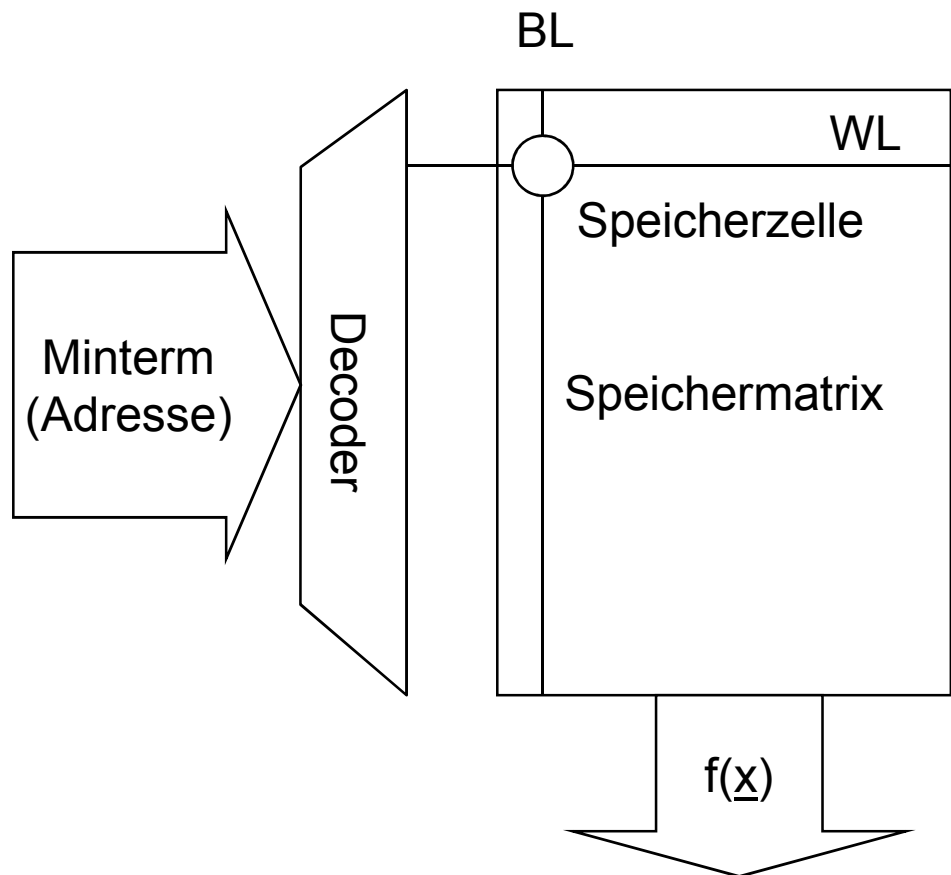
- PROMs
- PALs, PLAs
- Programmierbare Makro-Logik (PML)
- Field programmable Gate-Arrays (FPGA)

- **Semikunden-ICs**

- Gate-Arrays, Sea of Gates
- Standardzellen-ICs

1.2 Realisierung von Logikfunktionen

PROM

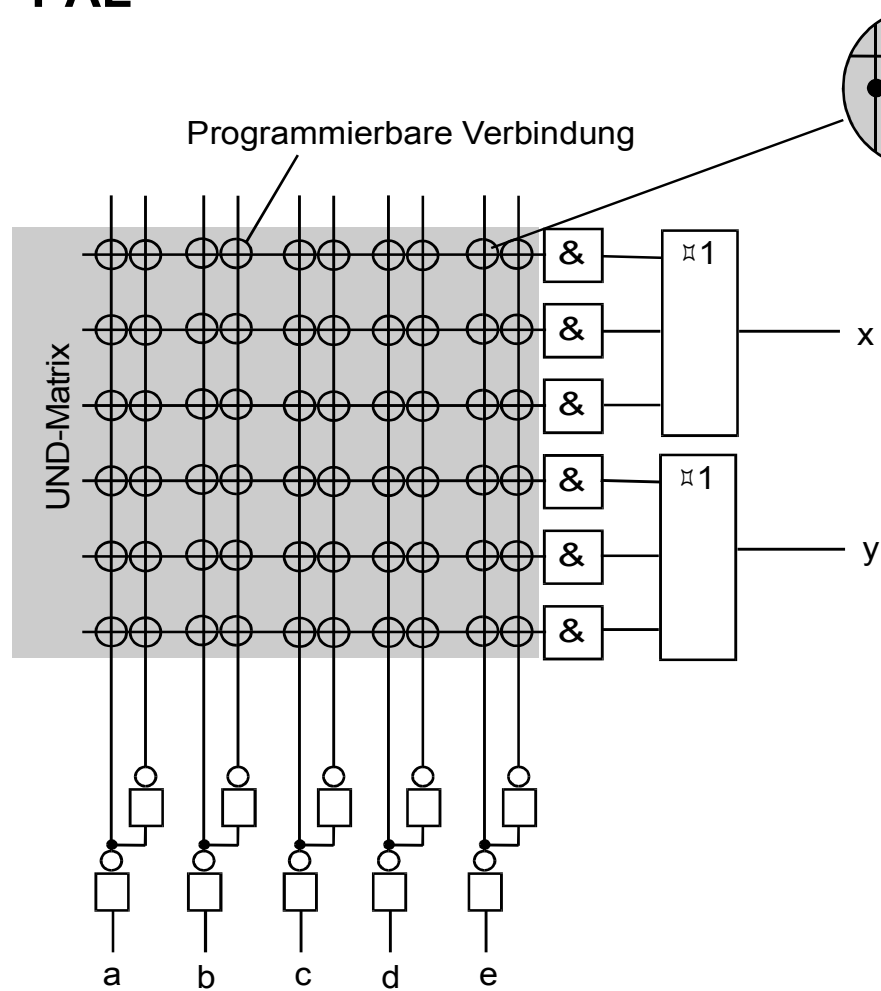


Jeder Minterm muss realisiert werden. Diese Art der Realisierung bietet Vorteile, falls wenige Don't cares in der Funktion vorhanden sind und wenige Ausgabe lauter Nullen liefern.

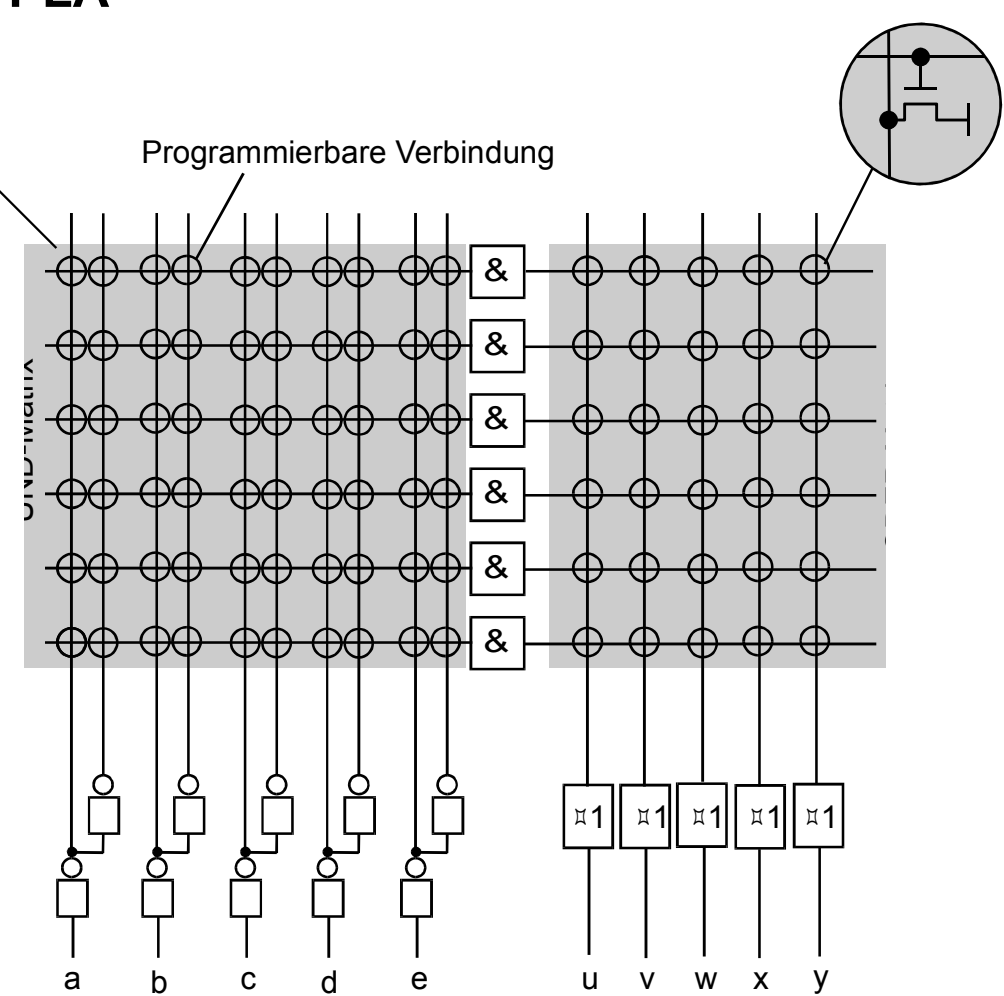
Bei einer Realisierung auf einem Semicustom-IC muss eine geeignete Aufteilung zwischen Zeilen- und Spaltenadresse gefunden werden (quadratische Speichermatrix)

1.2 Realisierung von Logikfunktionen

PAL

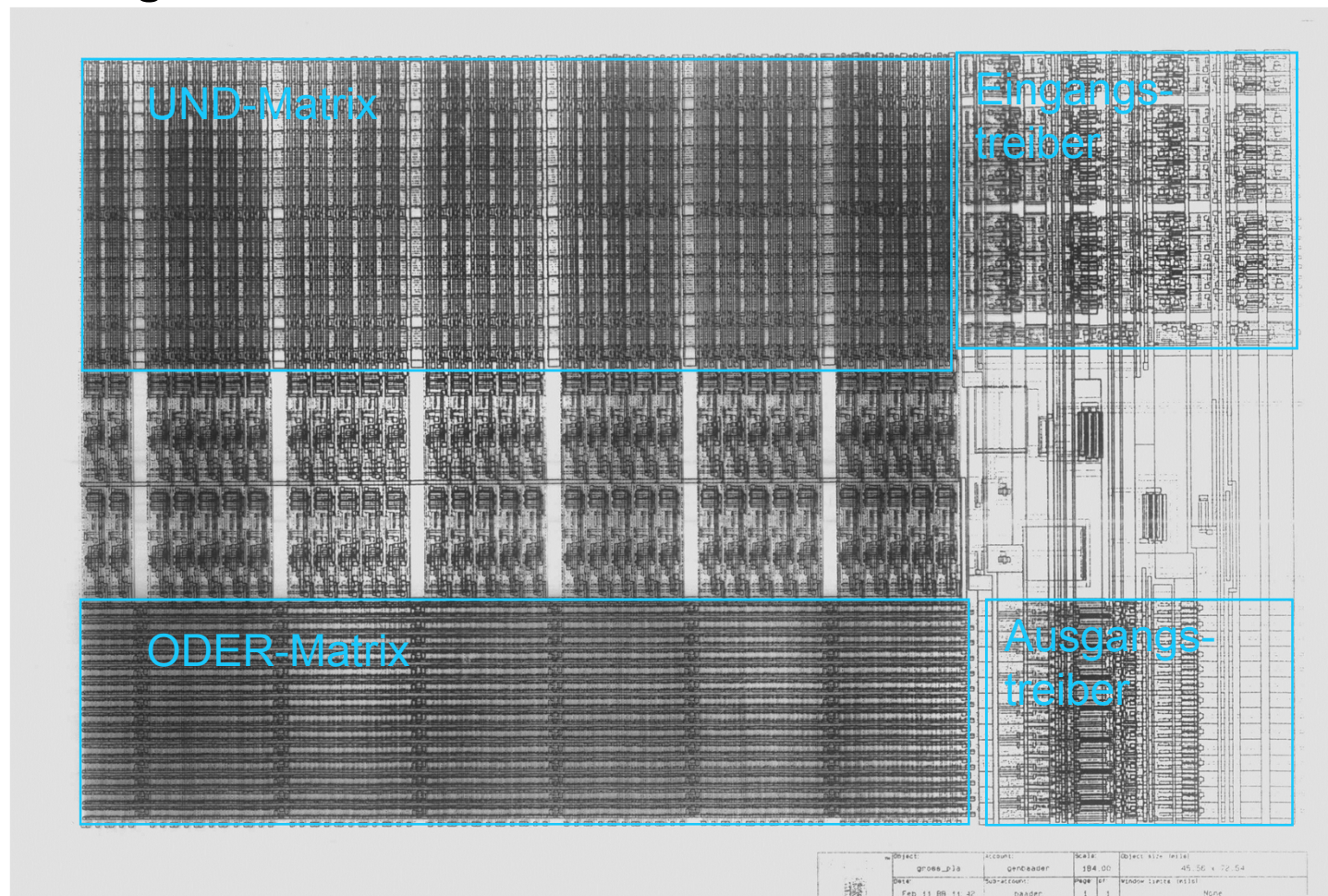


PLA



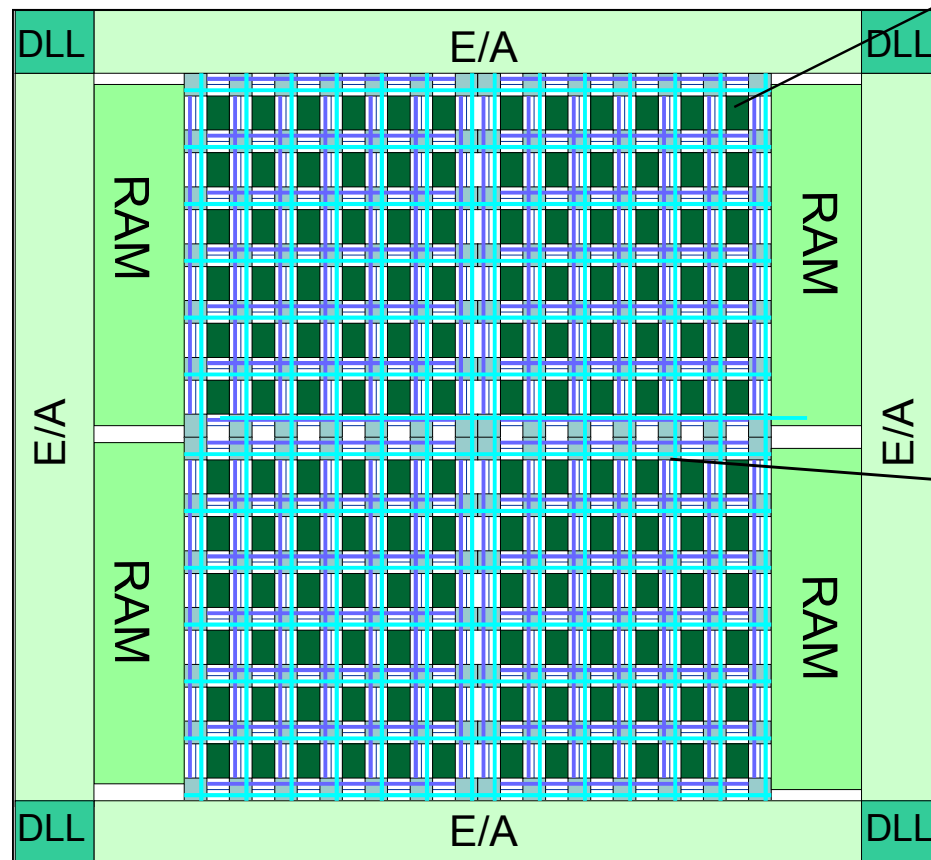
1.2 Realisierung von Logikfunktionen

PLA generiert für ein ASIC mit PLA-Generator

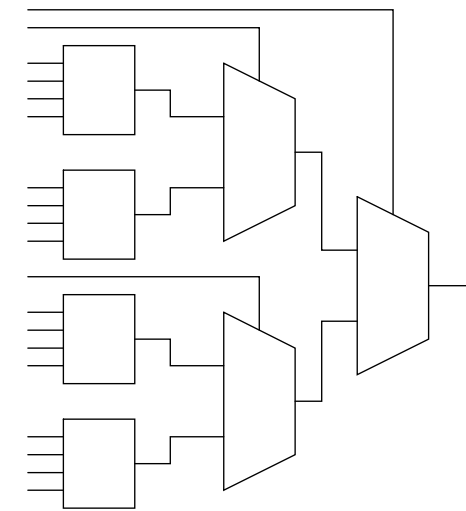


1.2 Realisierung von Logik

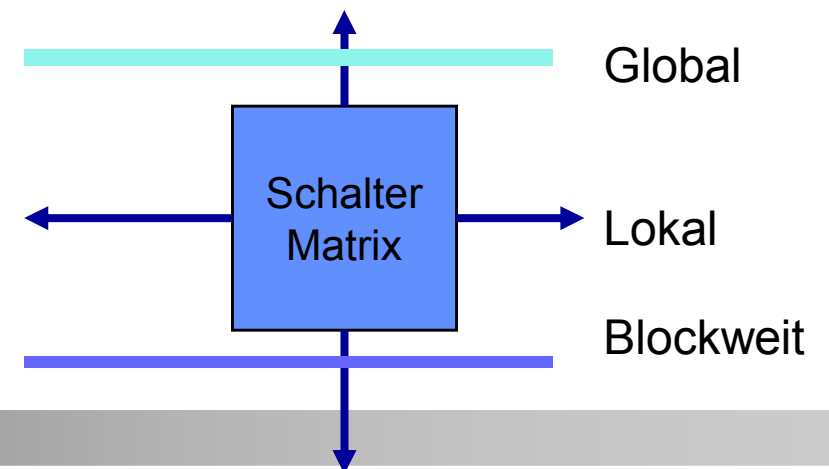
FPGA (Field Programmable Gate Array)



CLB (Core Logic Block)

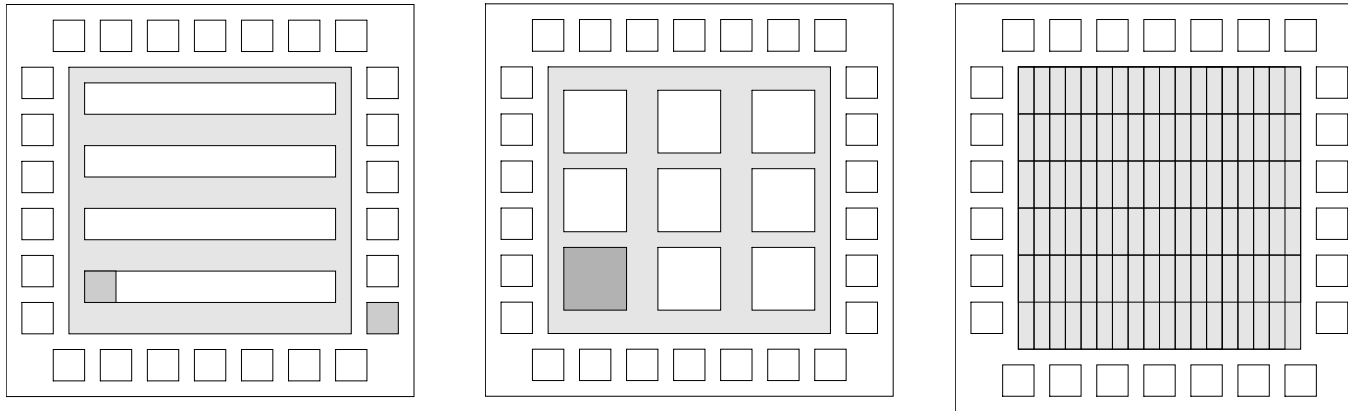


Programmierbare Verdrahtung

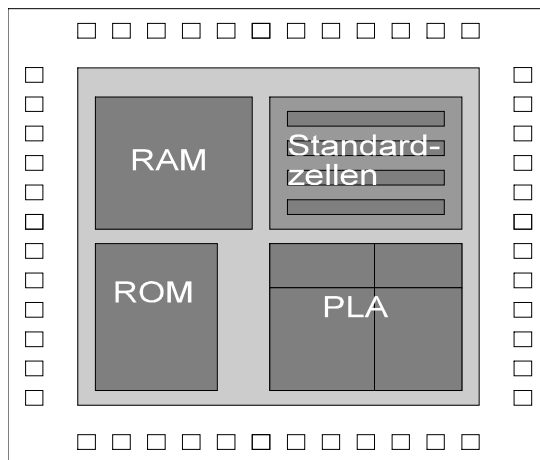


1.2 Realisierung von Logik

Semicustom Gate Arrays



Standardzellen/Makrozellen



1.3 Werkzeuge

Werkzeuge

- **Schaltungseditoren**

dienen zur graphischen Eingabe von Schaltplänen. Moderne Schaltungseditoren gestatten den hierarchischen Entwurf von Logikschaltungen. Hierbei können einzelnen Blöcke auch durch andere Beschreibungsformen spezifiziert werden.

- **Logiksynthese**

wandelt eine RT-Beschreibung der Logik in eine Implementierung um. Falls es sich bei der Zieltechnologie um PLAs, PALs handelt spricht man auch von Logikminimierung.

- **Simulatoren (hier Logiksimulatoren)**

dienen zur Validierung der erstellten Logik. Hierbei können nacheinander einzelne Eingangskombinationen, die Stimuli, an die Schaltung angelegt werden. Der Simulator berechnet dann die zugehörigen Ausgangssignale.

1.3.1 Logiksimulation

- **Signalwerte [Marw93]**

- Zweiwertige Simulation

- 1 Logisch "wahr"
 - 0 Logisch "falsch"

- Mehrwertige Simulation (zusätzliche Werte)

- U,X Wert ist elektrisch weder 0 noch 1
 - Z Wert ist hochohmig

Hierbei gilt $U > 1$, $0 > Z$

- Switch-Level Simulation

- Für 1, 0, U werden auch "schwache" Werte eingeführt.
Dies dient zur Modellierung von als Widerstände verwandte Verarmungstransistoren. Schwache Werte werden von starken überschrieben.

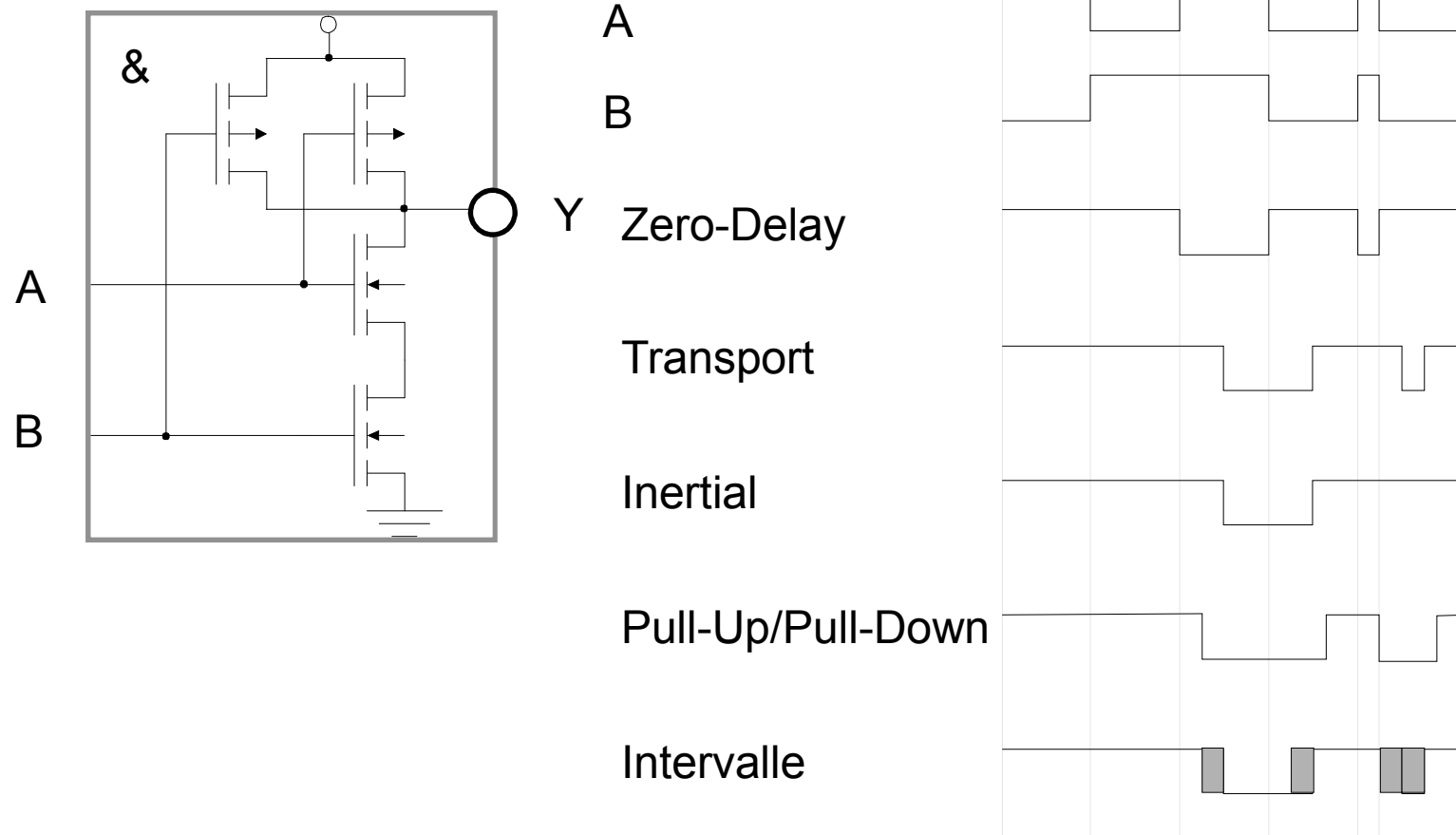
1.3.1 Logiksimulation

- **Simulation der Verzögerungszeiten**

- Zero-Delay
keine Modellierung der Verzögerung. Hier erfolgt nur eine funktionale Simulation.
- Unit-Delay
Alle Elemente erhalten eine einheitliche Verzögerungszeit
- Transportverzögerungen
Signale werden um einen gewissen Zeitraum verzögert. Hierbei kann noch zwischen reiner Gatterverzögerung und Leitungsverzögerung unterschieden werden.
- Inertialverzögerungen
Signalwechsel werden nur dann weitergeleitet, falls sie eine gewisse Zeit anliegen
- Pull-Up/Pull-Down-Verzögerung
- Verzögerungsintervalle

1.3.1 Logiksimulation

Beispiel für verschiedene Verzögerungsmodelle



1.3.1 Logiksimulation

- **Eingabe**

- Beschreibung der Schaltung Schaltungsgraph
- Beschreibung der Testdaten (Stimuli)

- **Tabellengesteuerte Simulation**

- Für jedes Bauelement gibt es eine tabellarische Beschreibung der Funktion.
 - Interpretative Simulation
Nach Anlegen eines Eingabemusters werden alle Signale nach den in den Tabellen gespeicherten Vorschriften neu berechnet.
 - Ereignisgesteuerte Simulation
Hierbei werden nur Signaländerungen mit ihrem Zeitpunkt in einer Liste gehalten. Eine Auswertung erfolgt nur in den von der Signaländerung betroffenen Schaltungsteilen.

- **Compilierte Simulation**

Auswertung der Schaltung wird in eine Zielsprache (ASM oder Hochsprache) übersetzt. Die Simulation erfolgt durch Programmabarbeitung.

2 Entwurf der Komponenten des Operationswerkes

Funktionsumfang

- **Logikkomponenten**

AND, OR, NOT, NAND, NOR, EXOR, EXNOR

Encoder, Decoder, Multiplexer, Demultiplexer

allgemeine Schaltnetze

- **Vergleicher**

EQU, NEQ, LT, GT, GTE, LTE

- **Arithmetik**

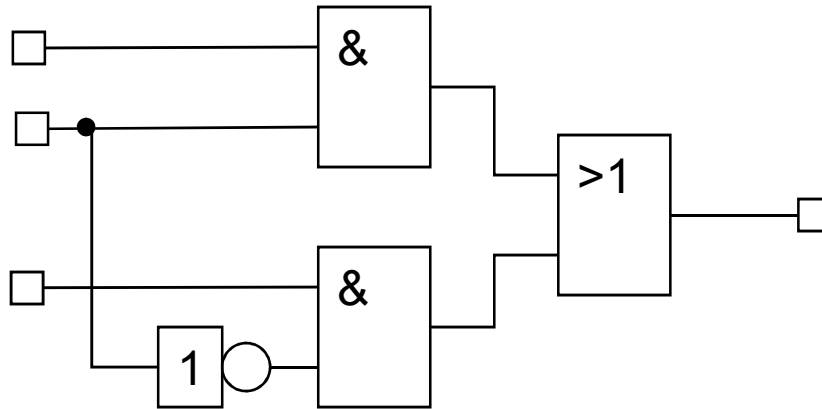
ADD, SUB, SHIFT

Arithmetische Logische Einheit (ALU)

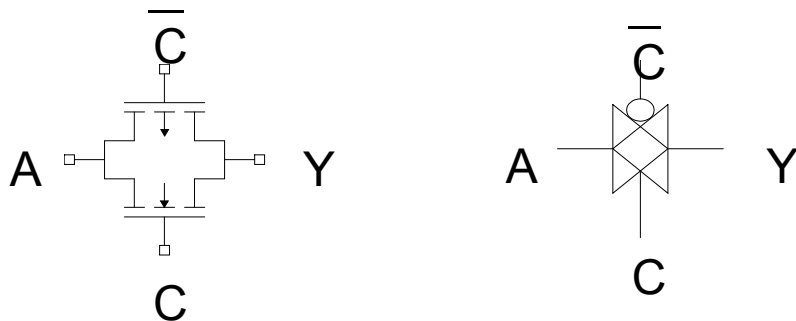
Multiplikation, Division, elementar-transzendente Funktionen

2.1 Elementare Komponenten

Gatterrealisierung eines 2:1 Multiplexers



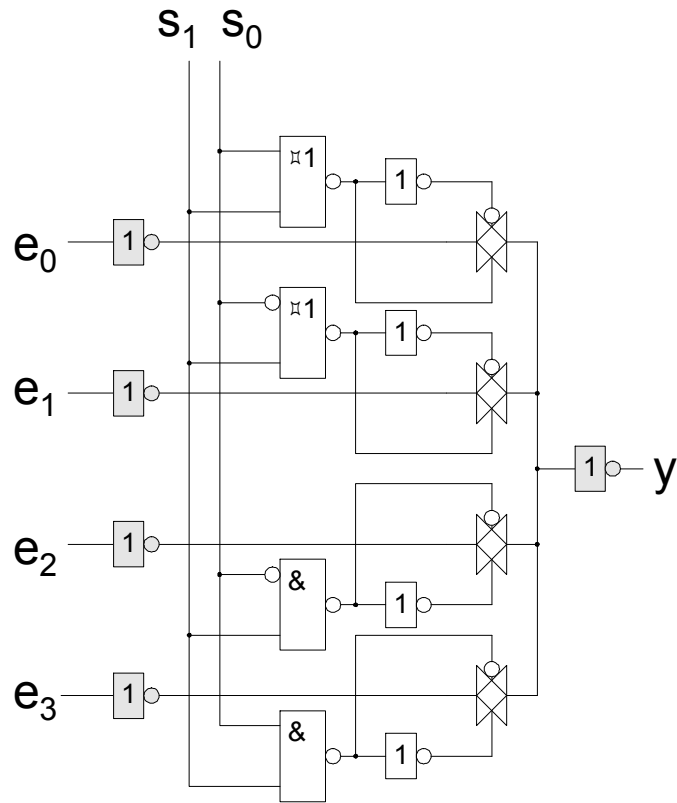
Transmissions-Gatter (CMOS-Schalter)



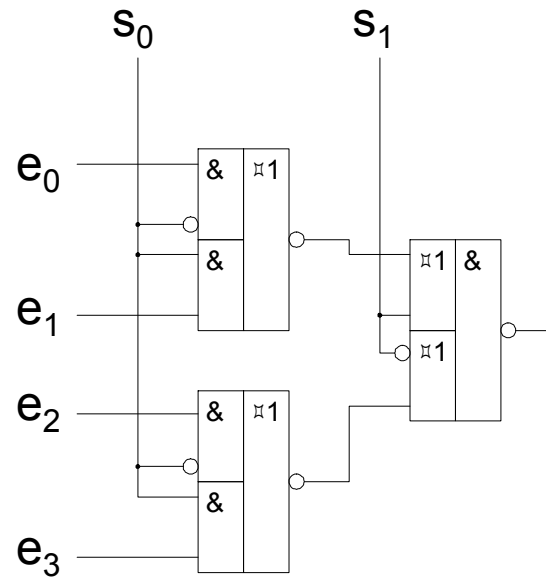
Bem.: Durch geeignete Ansteuerung kann ein Transmission-Gatter als Analogschalter verwendet werden (z. B. 4066)

2.1 Elementare Komponenten

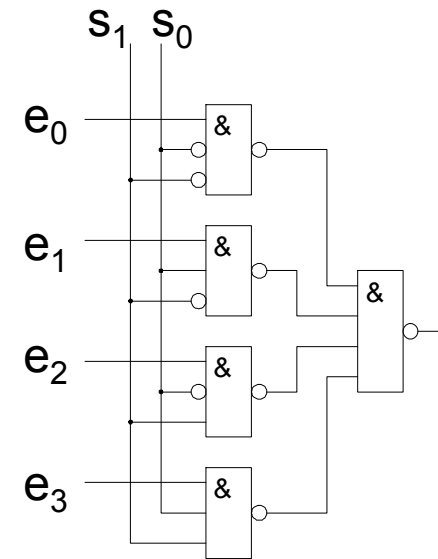
Realisierungsvarianten eines 4:1



44 Transistoren



26 Transistoren



32 Transistoren

Zahlendarstellung

Ganzzahlige Werte



(n-1)

0

Zahlenbereiche

positive Zahlen: $0.. 2^n-1$

$$\sum_{i=0}^{n-1} b_i \cdot 2^i$$

2er-Komplement: $-2^{(n-1)}..2^{(n-1)}-1$

$$\sum_{i=0}^{n-1} b_i \cdot 2^i - 2^{(n-1)}$$

MSB Most significant bit (meist Bit (n-1))

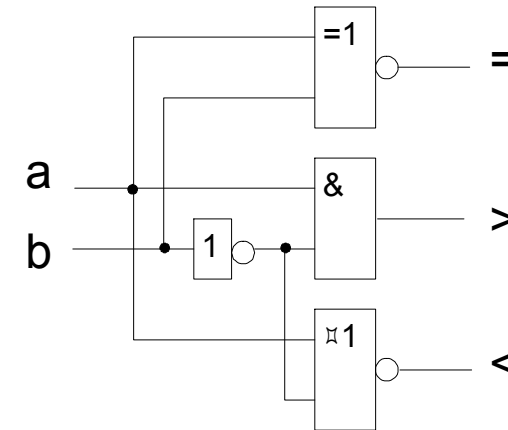
LSB Least significant bit

Bemerkung: Bei der 2er-Komplement-Darstellung kann das MSB als Vorzeichenbit interpretiert werden.

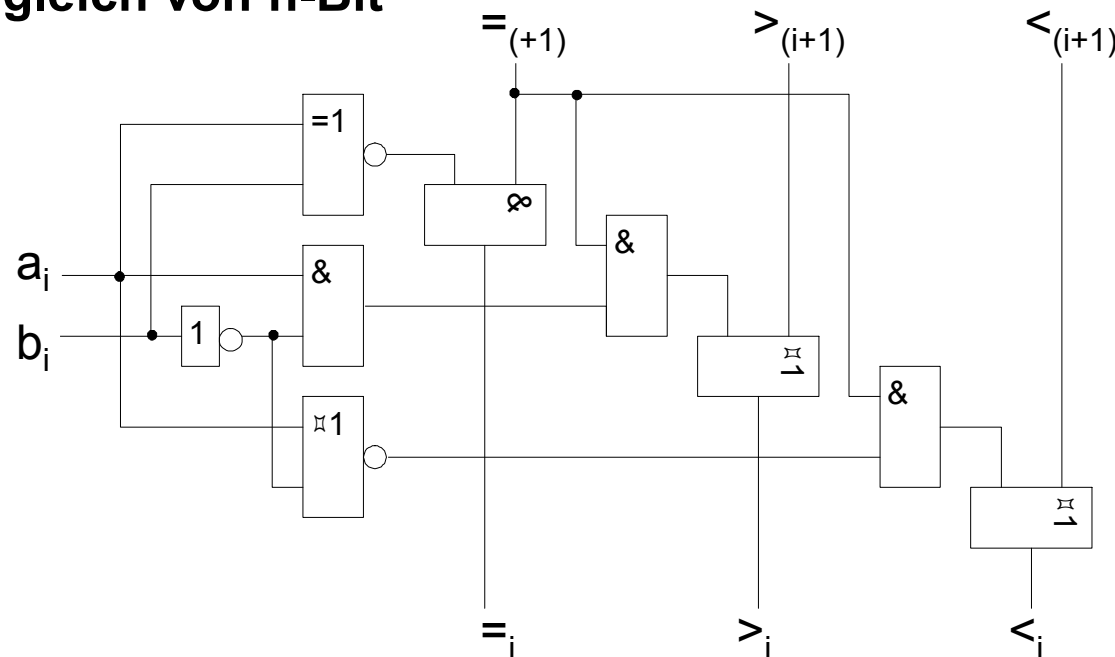
2.1 Elementare Komponenten

Vergleich von einem Bit Wahheitstabelle

a	b	=	>	<
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0



Vergleich von n-Bit



Alternativ

Berechne $c := a - b$

$a = b$ falls $c = 0$ (Zero-Flag)

$a > b$ falls $c > 0$ (MSB = 0)

$a < b$ falls $c < 0$ (MSB = 1)

2.2 Addierer

Halbaddierer

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

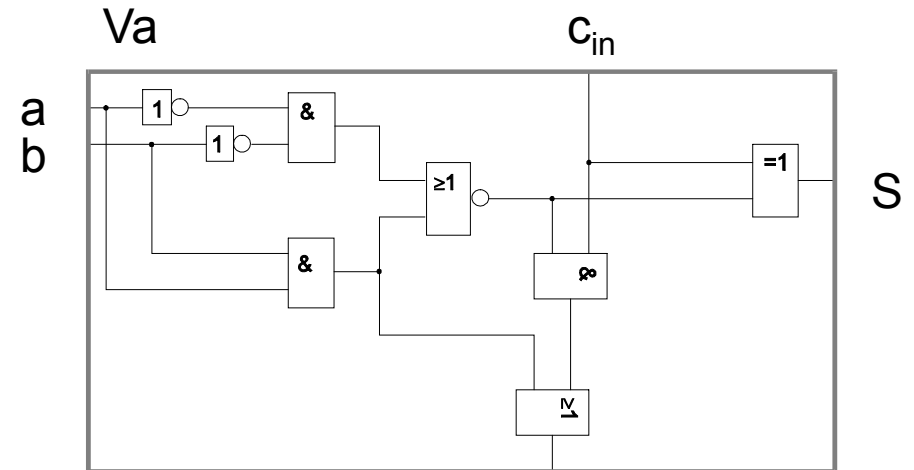
$$S = a \oplus b$$

Volladdierer

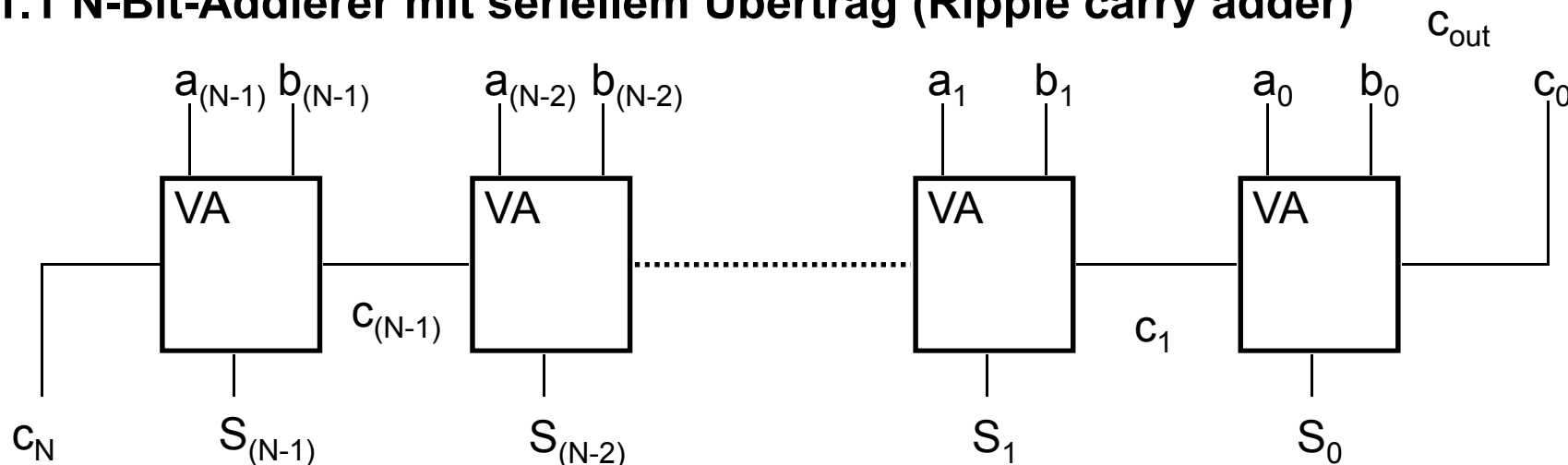
$$S' = \overline{a \wedge b \vee \bar{a} \wedge \bar{b}}$$

$$S = S' \oplus c_{in}$$

$$c_{out} = a \wedge b \vee c_{in} \wedge S'$$



2.1.1 N-Bit-Addierer mit seriellem Übertrag (Ripple carry adder)



Die maximale Verzögerungszeit ergibt sich zu $6\blacklozenge + 2(N-1)\blacklozenge$ (Gatterverzögerung \blacklozenge)

2.2.2 Addierer mit parallelem Übertrag

Das Carry-Look-Ahead-Prinzip

In der i-ten Stufe wird ein Übertrag generiert, falls

$$G_i = A_i \uparrow B_i = 1 \text{ (**Generate**)}$$

Ein Übertrag $C_{(i-1)}$ wird durch die i-te Stufe propagiert falls

$$P_i = A_i \Downarrow B_i = 1 \text{ (**Propagate**)}$$

Die Summe ergibt sich zu $S_i = P_i \Downarrow C_i$ (Bem.: $C_{-1} = C_{in}$)

Der i-te Übertrag ergibt sich zu $C_i = P_i \uparrow C_{i-1} \Downarrow G_i$

Bei einem Vier-Bit-Addierer ergeben sich folgende Gleichungen:

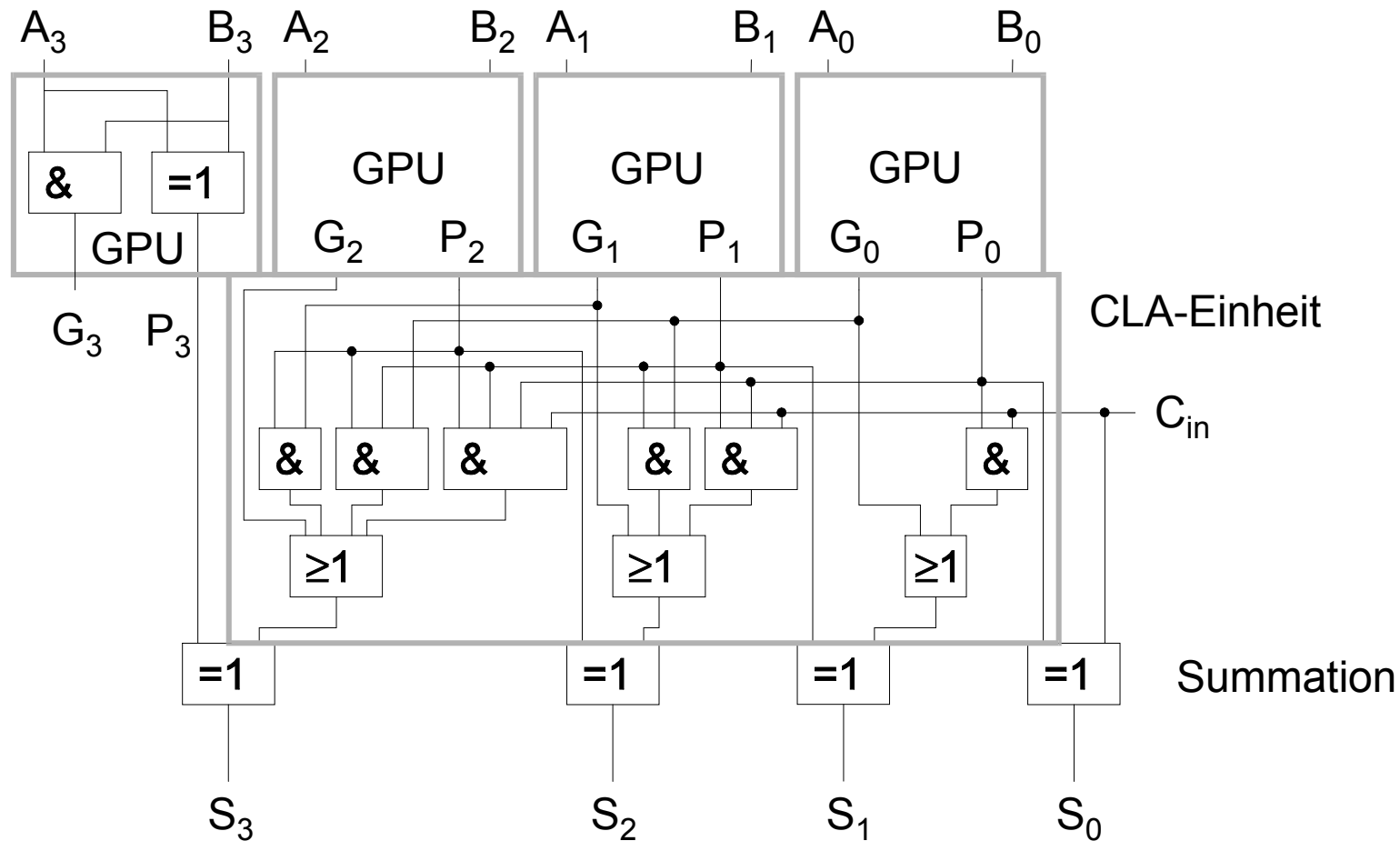
$$C_0 = P_0 \uparrow C_{in} \Downarrow G_0$$

$$C_1 = P_1 \uparrow C_0 \Downarrow G_1 \quad C_1 = (P_1 \uparrow P_0 \uparrow C_{in}) \Downarrow (P_1 \uparrow G_0) \Downarrow G_1$$

$$C_2 = P_2 \uparrow C_1 \Downarrow G_2 \quad C_2 = (P_2 \uparrow P_1 \uparrow P_0 \uparrow C_{in}) \Downarrow (P_2 \uparrow P_1 \uparrow G_0) \Downarrow (P_2 \uparrow G_1) \Downarrow G_2$$

2.2.2 Addierer mit parallelem Übertrag

Ein Vierbit-CLA-Addierer



2.2.2 Addierer mit parallelem Übertrag

Wegen der steigenden Anzahl von Eingängen für das UND-Gatter ist es nicht sinnvoll, mehr als vier Bits in der angegebenen Weise zu verwenden

Hierzu ist eine hierarchische Zusammenfassung zu Vierergruppen notwendig (Block-CLA-Addierer)

Eine solche Vierergruppe generiert einen Übertrag bei

$$G_G = (P_3 \wedge P_2 \wedge P_1 \wedge G_0) \vee (P_3 \wedge P_2 \wedge G_1) \vee (P_3 \wedge G_2) \vee G_3$$

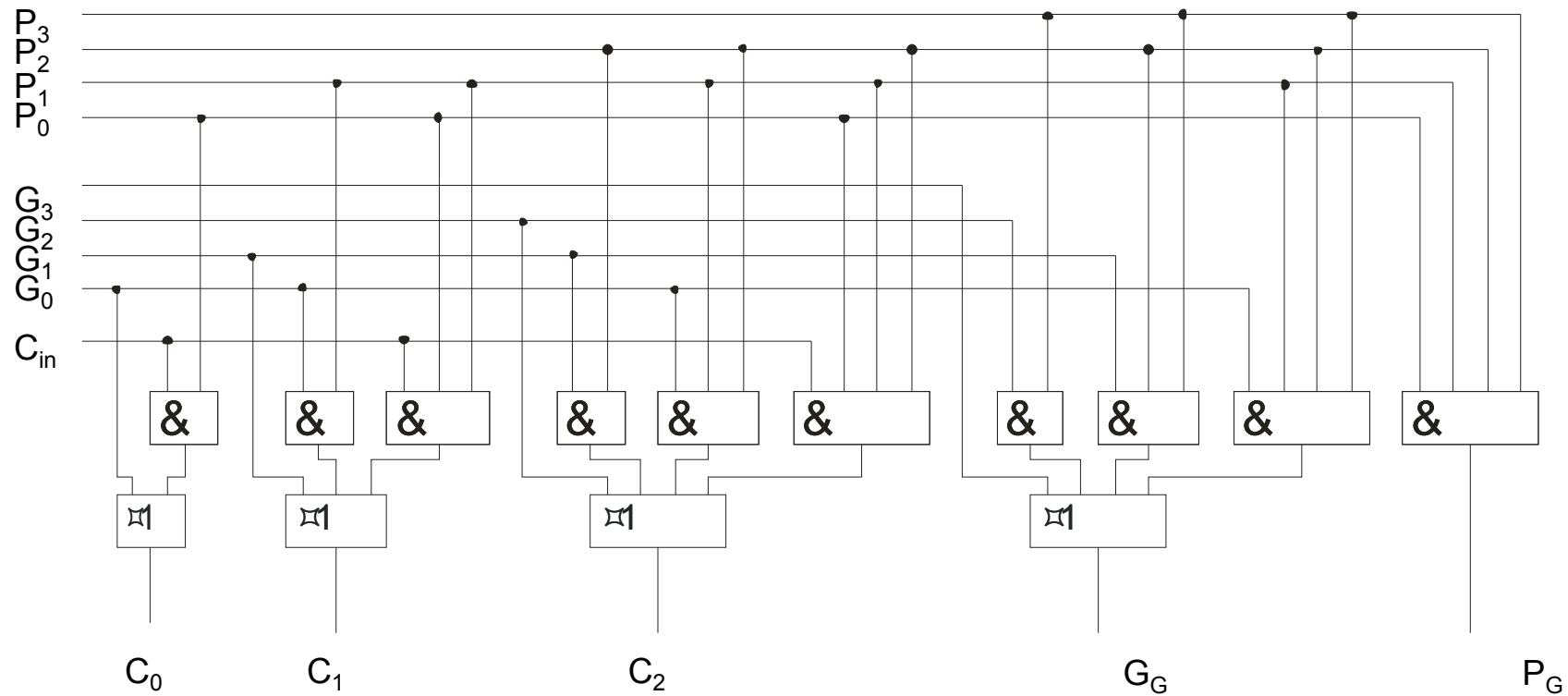
und propagiert einen Übertrag bei

$$P_G = (P_3 \wedge P_2 \wedge P_1 \wedge P_0)$$

Für größere Addierer kann also eine hierarchische Einheit für die Übertragvorrausschau verwendet werden. Diese besitzt den gleichen Aufbau wie die für die normalen Überträge. Die Ausgänge C_i werden dann an die entsprechenden C_{in} -Eingänge der einzelnen Blöcke angeschlossen.

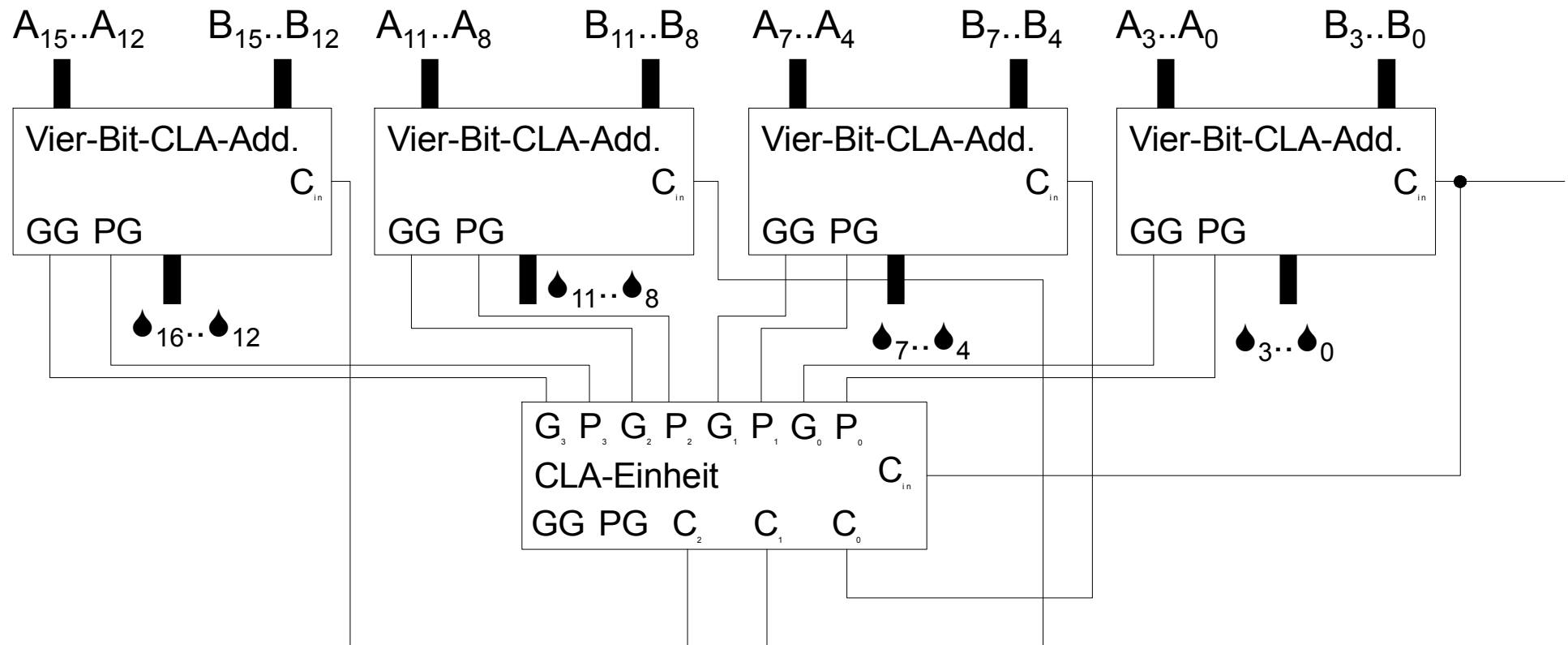
2.2.2 Addierer mit parallelem Übertrag

Block-Carry-Look-Ahead-Einheit (BCLA)



2.2.2 Addierer mit parallelem Übertrag

16-Bit Block-CLA-Addierer

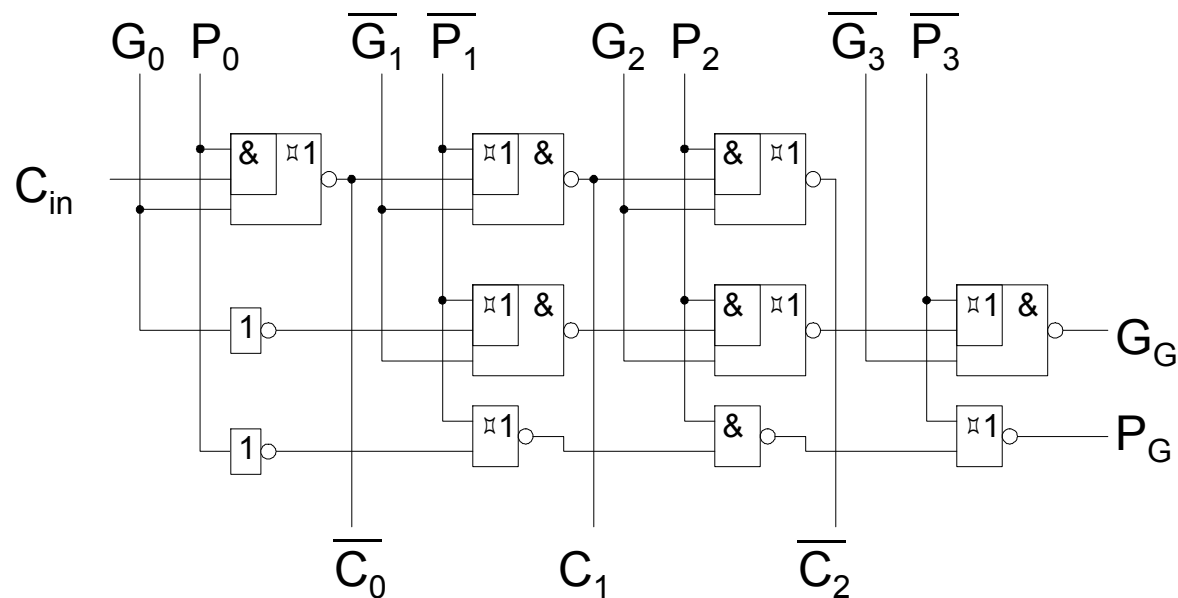


2.2.3 Addierer mit parallelem Übertrag

Verminderung des Flächenverbrauchs

In der bisherigen Implementierung ist der Flächenverbrauch relativ hoch. Der wesentliche Laufzeitvorteil ergibt sich durch die baumartige Berechnung des Übertrags. Ein Kompromiß zwischen Laufzeit und Flächenverbrauch kann durch eine Modifikation der Übertragseinheit erzielt werden.

Die Übertragseinheit (optimiert für CMOS)

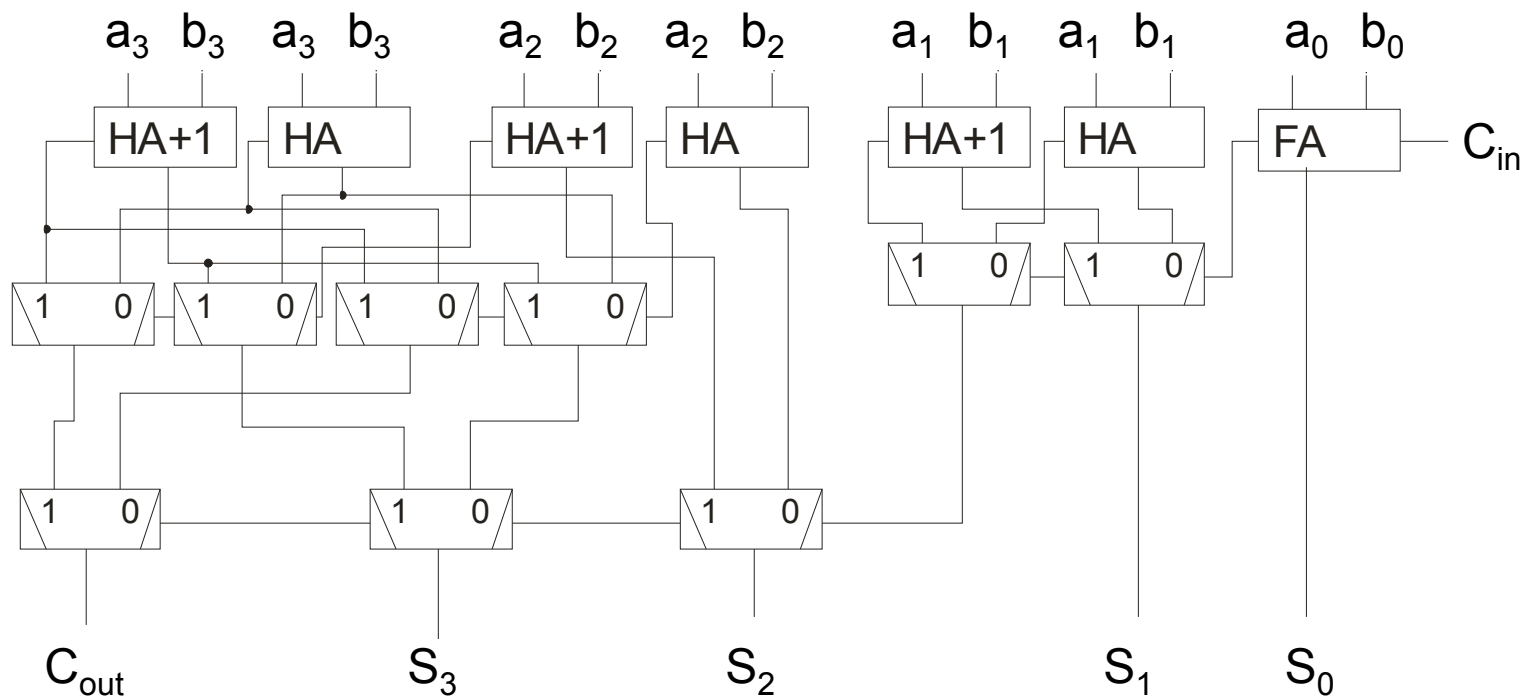


Bem.: Da P_1 , C_0 und P_3 , C_2 invertiert generiert werden, muß die Summiereinheit nicht verändert werden.

Probleme ergeben sich bei der Zusammenschaltung da P_G , G_G und C_{in} immer positiv sind

2.2.4 Carry-Select Addierer

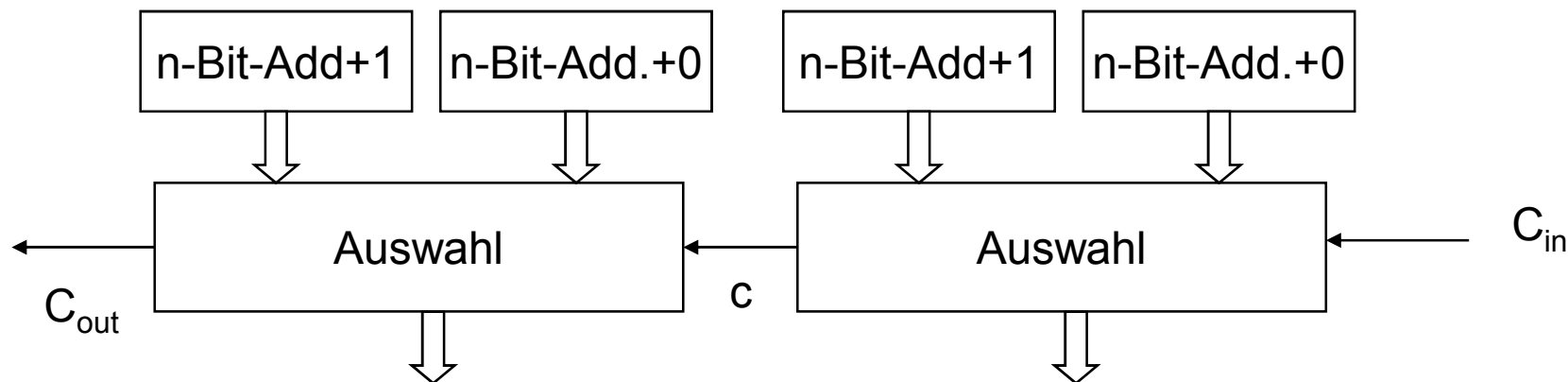
Die Summe wird sowohl für eine Addition mit als auch ohne Übertrag berechnet.
 Die Auswahl erfolgt in einer nachgeschalteten Stufe über Multiplexer.
 Dies wird als Addierer mit bedingter Summe bezeichnet



HA+1			
a_i	b_i	g_i	s_i
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	1

2.2.4 Carry-Select Addierer

Einen Addierer mit Übertragsauswahl (Carry-Select-Adder) erhält man durch die Kaskadierung von n-Bit Addierern (z. B. mit seriellem Übertrag). In jeder Stufe wird die Summe mit und ohne Übertrag berechnet. Die vorige Stufe wählt dann das korrekte Ergebnis aus.



Durch Anpassung der Stufen an die Laufzeit des Übertrags (kleine Bitanzahlen für niederwertige Bits große für höherwertige) kann eine Geschwindigkeitssteigerung gegenüber einem Addierer mit seriellem Übertrag erzielt werden.

2.2.5 Präfix-Baum-Addierer

Zur Berechnung der Überträge definieren wir eine Verknüpfung $(a, b) \blacktriangleright (c, d)$ mit

$$(a, b) \circ (c, d) = (a \vee (b \wedge c), b \wedge d)$$

Hierbei ist die Operation \blacktriangleright assoziativ[†]

Die Überträge lassen durch den Ausdruck

$$(SC_{i+1}, SP_{i+1}) = (G_i, P_i) \circ (SC_i, SP_i)$$

$$SC_i = C_i \text{ für } i = 1 \dots n \text{ und } SC_0 = C_0 = C_{in}$$

$$SP_0 = 1$$

berechnen. Ausmultipliziert erhält man

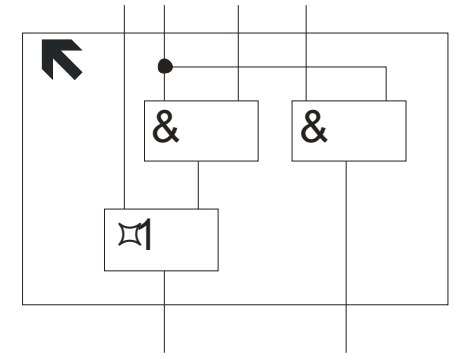
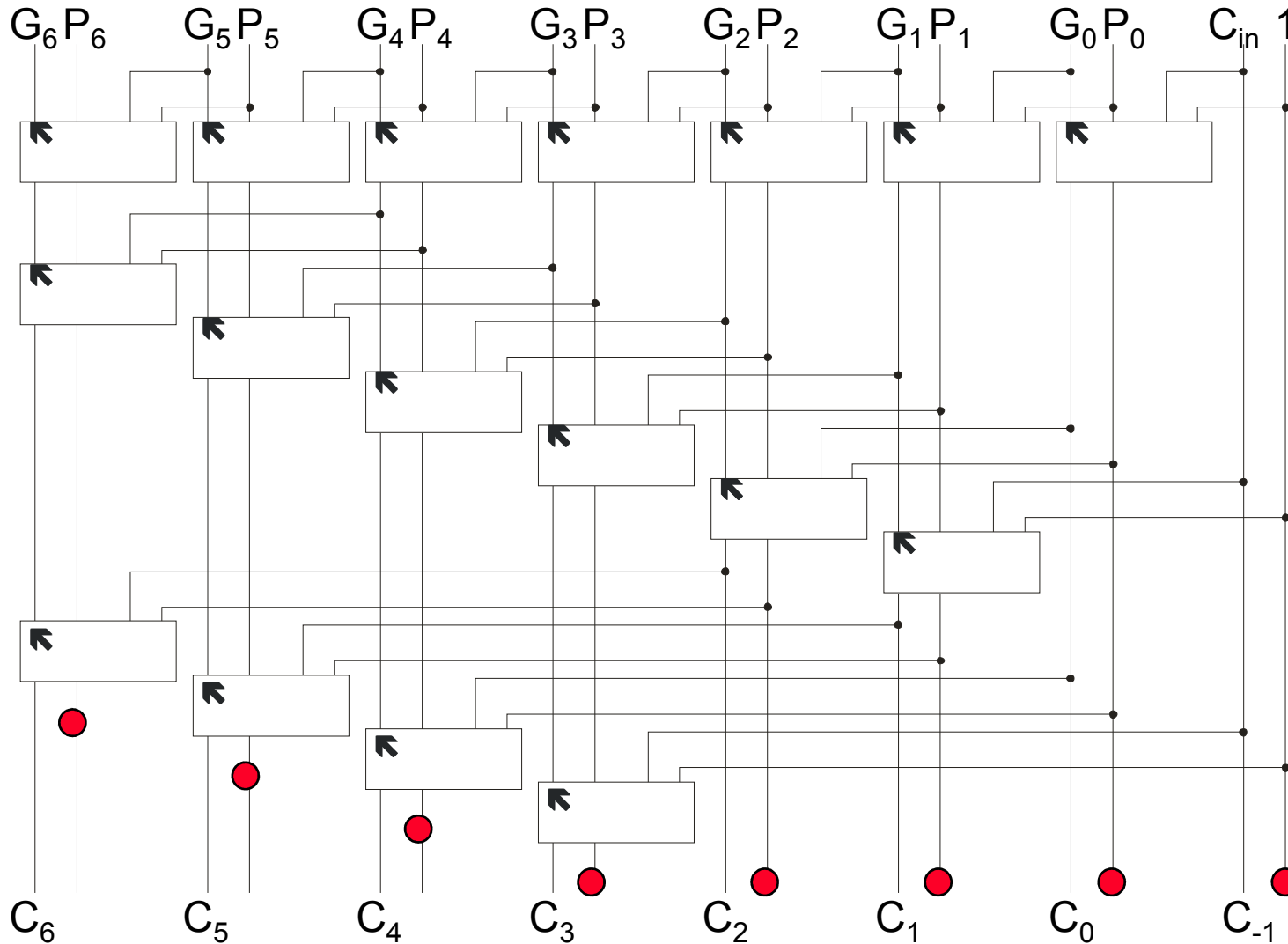
$$(SC_{i+1}, SP_{i+1}) = (G_i, P_i) \circ (G_{i-1}, P_{i-1}) \circ (G_{i-2}, P_{i-2}) \circ \dots \circ (G_0, P_0) \circ (SC_0 = C_{in}, SP_0 = 1)$$

Wegen der Assoziativität kann die Berechnung baumartig erfolgen. Hierzu sind verschiedene Schemata publiziert.

$$\begin{aligned} &^\dagger ((a, b) \circ (c, d)) \circ (e, f) = ((a \vee (bc), bd)) \circ (e, f) = \\ &((a \vee bc) \vee ((bd)e), (bd)f) = (a \vee b(c \vee de), b(df)) = \\ &(a, b) \circ ((c, d) \circ (e, f)) \end{aligned}$$

2.2.5 Präfix-Baum-Addierer

Kogge-Stone-Addierer (Schema der Übertragsberechnung)

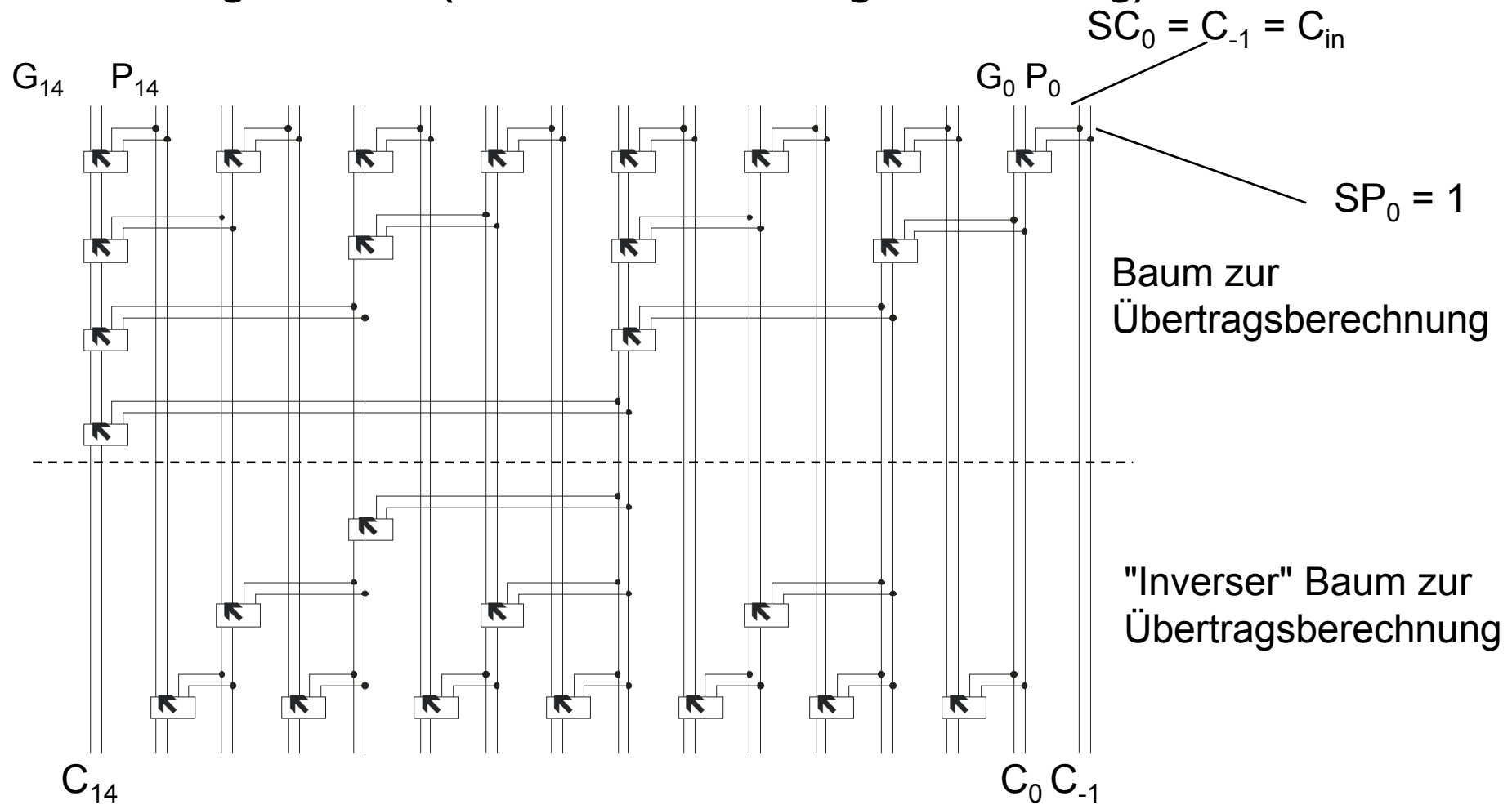


Die konstanten Werte und die offenen Ausgänge können zur Minimierung genutzt werden.
($C_0 = C_{in}$)

● offene Ausgänge

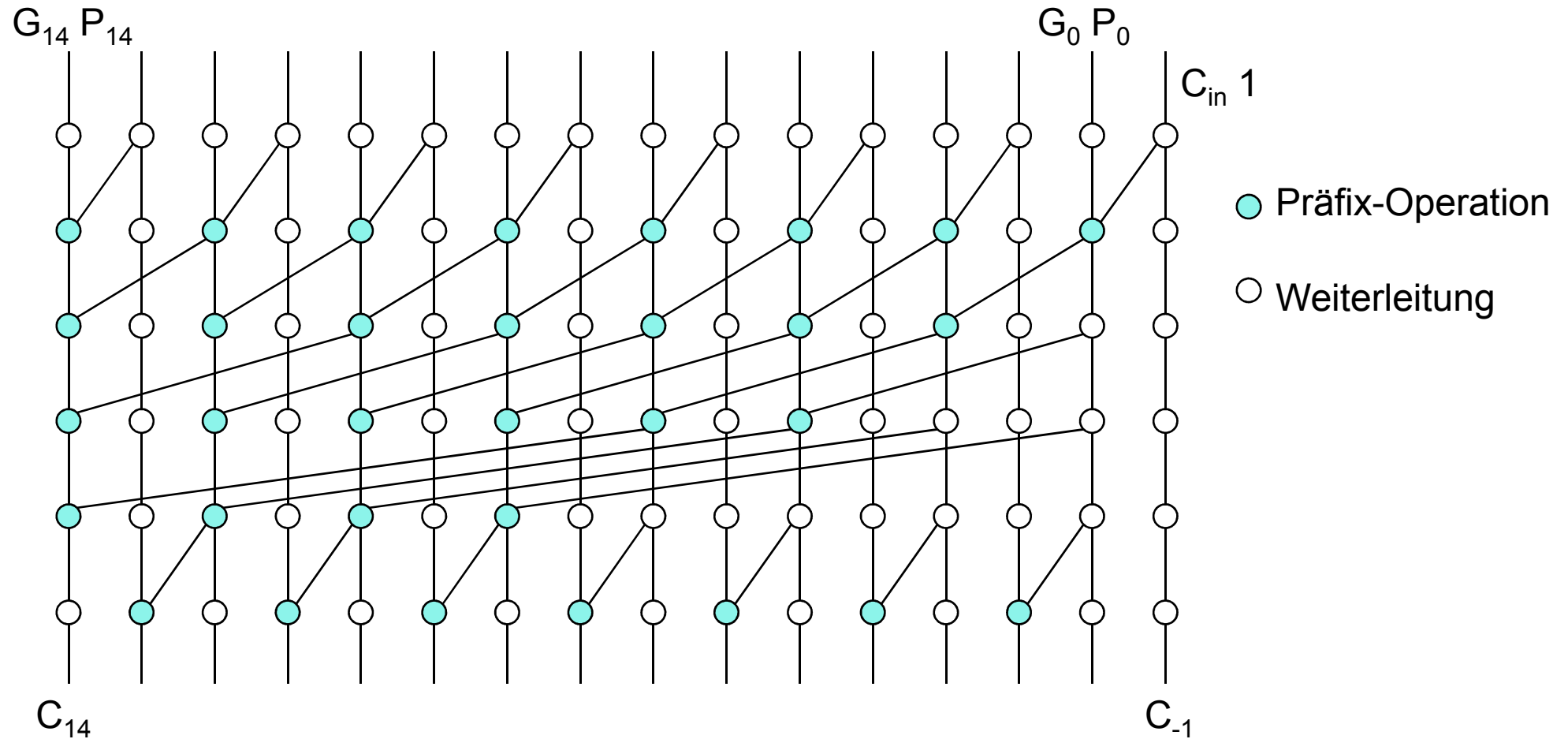
2.2.5 Präfix-Addierer

Brent-Kung Addierer (Schema der Übertragsberechnung)



2.2.5 Präfix-Addierer

Han-Carlson-Addierer

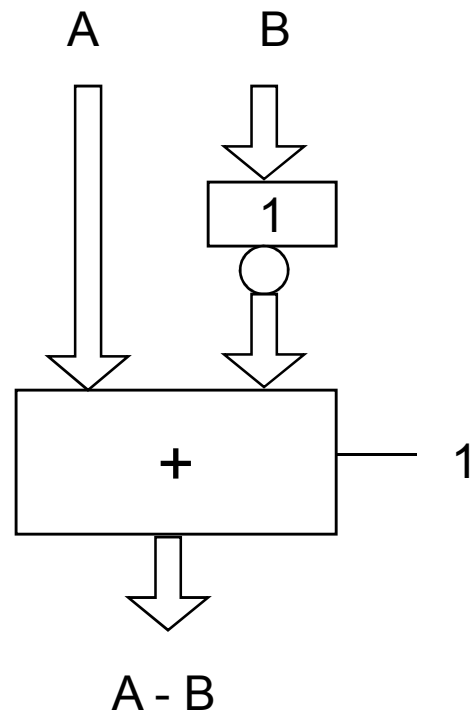


2.3 Subtrahierer

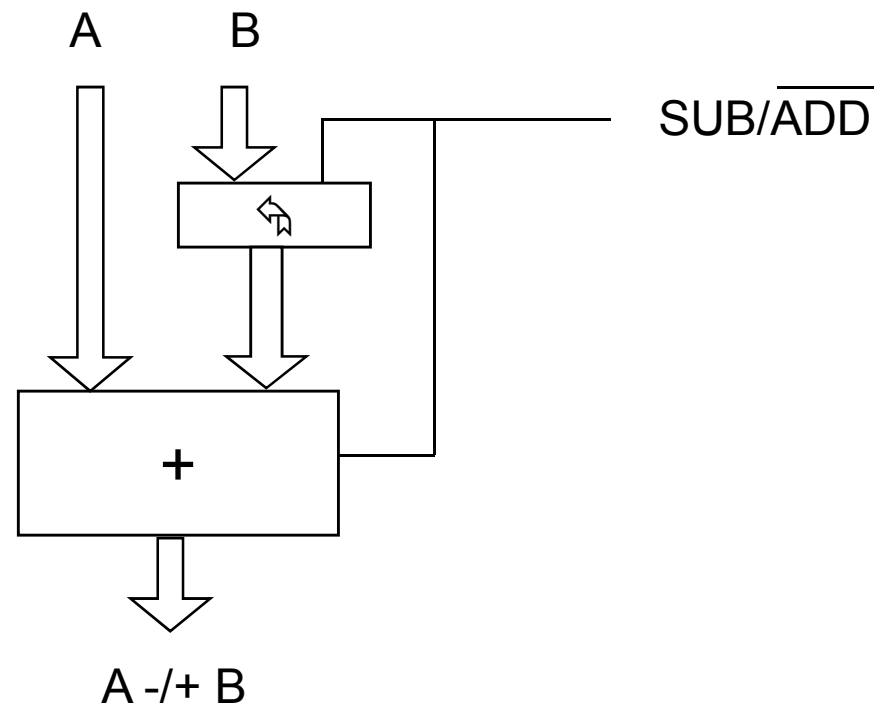
Die Subtraktion von zwei Dualzahlen $A = (a_{(N-1)} .. a_0)$, $B = (b_{(N-1)} .. b_0)$ wird durch Addition des Zweierkomplements realisiert, d. h. $A - B = A + (-B)$.

Hier berechnet sich das Zweierkomplement $(-B)$ zu $(\overline{b_{(N-1)}} .. \overline{b_0}) + 1$

Subtrahierer

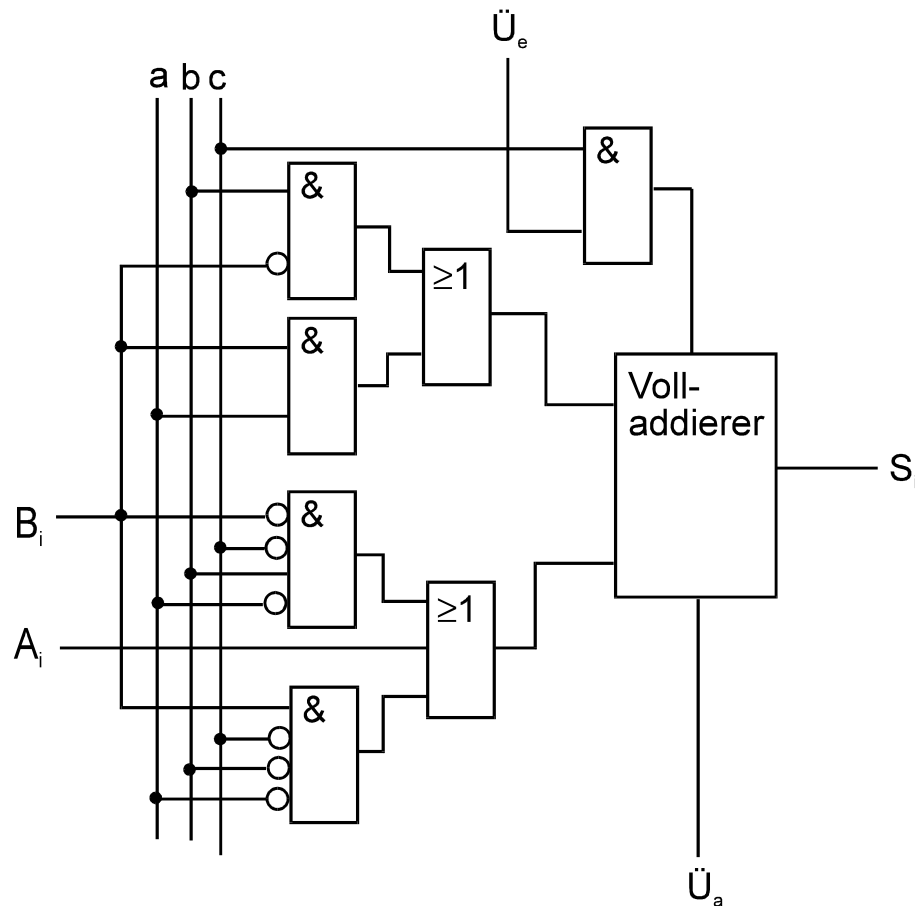


Addierer/Subtrahierer



2.3 ALU (Beispiel)

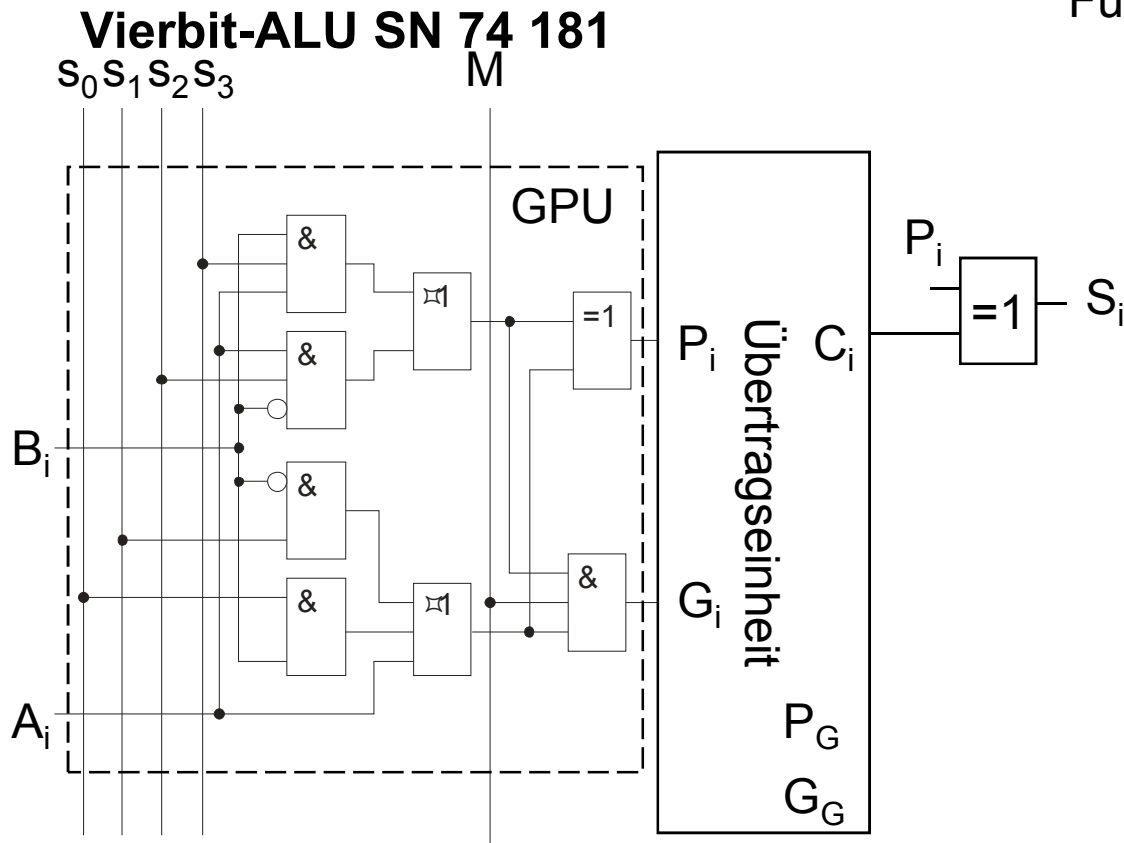
ALU Realisierung [Coy92]



Funktionen

a	b	c	\ddot{U}_e	S	Funktion
					Arithmetisch
0	0	1	0	A_i	Identität
0	0	1	1	$A_i + 1$	Inkrement
0	1	1	0	$A_i - B_i$	Subtraktion (Einerkomplement)
0	1	1	1	$A_i - B_i$	Subtraktion (Zweierkomplement)
1	0	1	0	$A_i + B_i$	Addition
1	0	1	1	$A_i + B_i + 1$	Addition mit Übertrag
1	1	1	0	$A_i - 1$	Dekrement
1	1	1	1	A_i	Identität
					Logisch
0	0	0	-	$A_i \vee B_i$	Disjunktion
0	1	0	-	$A_i \wedge B_i$	Konjunktion
1	0	0	-	$A_i \nabla B_i$	Antivalenz
1	1	0	-	$\overline{A_i}$	Negation

2.3 ALU



Funktionstabelle

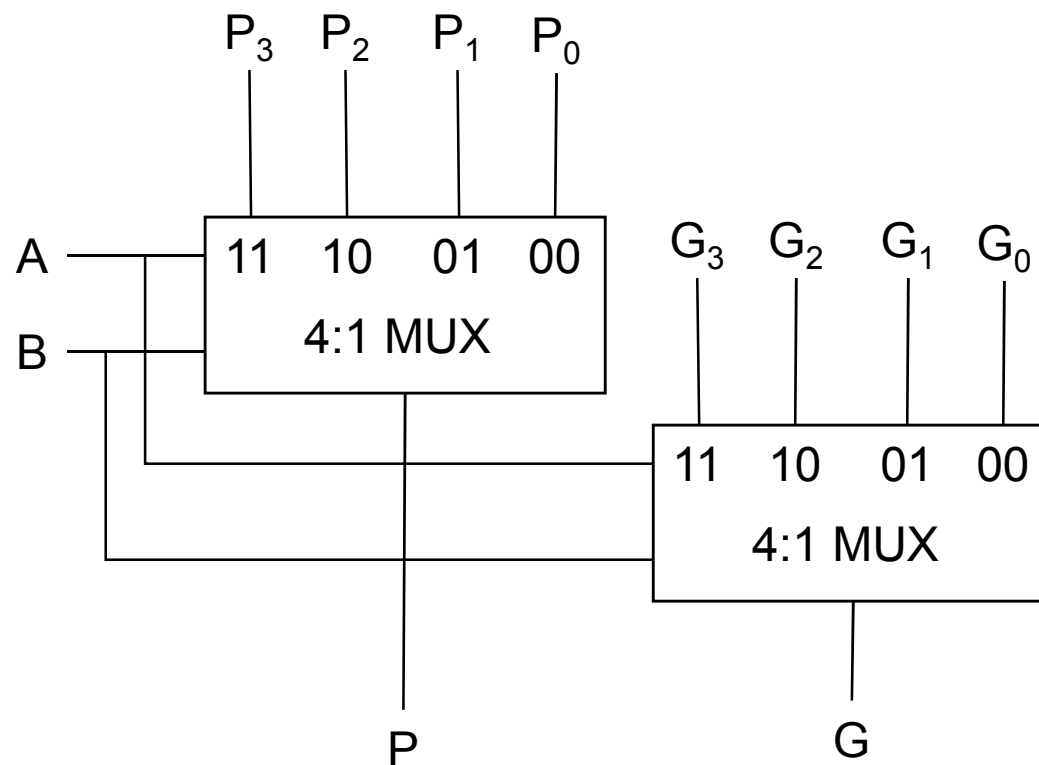
$S_3 S_2 S_1 S_0$	$M = 1$ Logische Funktionen	$M=0$ Arithmetische Funktionen	
		$c = 0$	$c = 1$
0000	\overline{A}	$A - 1$	A
0001	$\overline{A}B$	$AB - 1$	AB
0010	$\overline{A} \vee B$	$A \overline{B} - 1$	$A \overline{B}$
0011	1	-1	0
0100	$\overline{A} \vee \overline{B}$	$A + (A \vee \overline{B})$	$A + (A \vee \overline{B}) + 1$
0101	\overline{B}	$AB + (A \vee \overline{B})$	$AB + (A \vee \overline{B}) + 1$
0110	$\overline{A} \oplus B$	$A - B - 1$	$A - B$
0111	$A \vee \overline{B}$	$A \vee \overline{B}$	$(A \vee \overline{B}) + 1$
1000	$\overline{A} B$	$A + (A \vee B)$	$A + (A \vee B) + 1$
1001	$A \oplus B$	$A + B$	$A + B + 1$
1010	B	$A \overline{B} + (A \vee B)$	$(A \overline{B}) + (A \vee B) + 1$
1011	$A \vee B$	$A \vee B$	$(A \vee B) + 1$
1100	0	$A + A$	$A + A + 1$
1101	$A \overline{B}$	$AB + A$	$AB + A + 1$
1110	AB	$A \overline{B} + A$	$A \overline{B} + A + 1$
1111	A	A	$A + 1$

Zusammen mit dem Baustein SN 74 182 kann ein paralleler Übertrag realisiert werden

2.3 ALU

ALU nach [Mead Convay 78]

Hierbei werden die Funktionen P und G jeweils durch eine 4:1 Multiplexer realisiert.



A	B	P ₃	P ₂	P ₁	P ₀
0	0	-	-	-	1
0	1	-	-	0	-
1	0	-	0	-	-
1	1	1	-	-	-
Steuerwort					
-	-	1	0	0	1
A	B	G ₃	G ₂	G ₁	G ₀
0	0	-	-	-	0
0	1	-	-	0	-
1	0	-	1	-	-
1	1	0	-	-	-
Steuerwort					
-	-	0	1	0	0

2.3 ALU

Flags

Eine ALU generiert meist folgende Flags

S Vorzeichenflag (Sign)

Das Vorzeichen einer Zweierkomplementzahl ergibt sich aus dem MSB

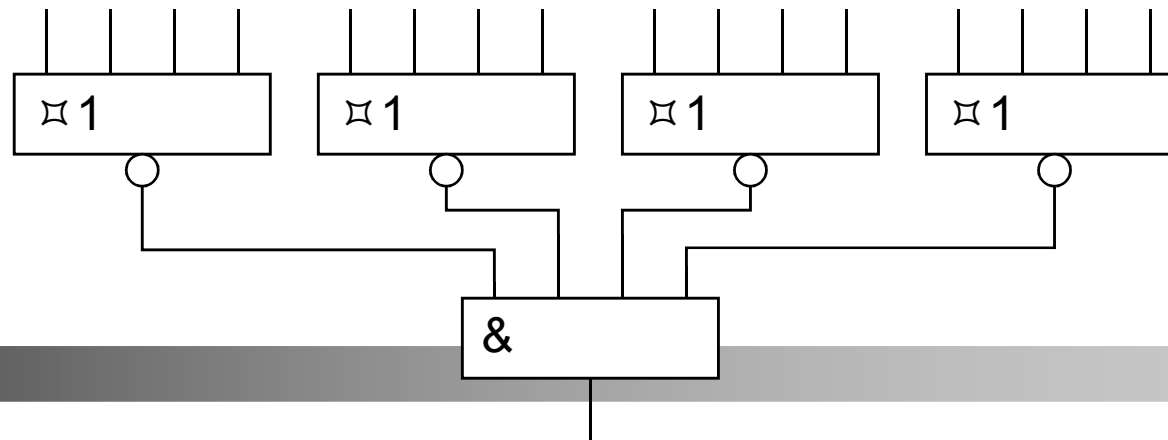
1 : Zahl ist negativ, 0 : Zahl ist positiv

O Über- bzw. Unterlauf (Overflow)

Bei Zweierkomplementzahlen ergibt sich bei der Addition ein Über-, Unterlauf, falls ($C_{out} \neq MSB$)

Z Nullflag (Zero)

Der Test auf eine Ergebnis $= 0$ macht den Test jeden einzelnen Bits notwendig. Zur schnelleren Berechnung sollte der Test als Baum realisiert werden. z. B.:



2.3 ALU

Vergleichsoperationen durch $A - B$ (Zweierkomplement-Subtraktion)

Gleichheit

$$A = B : Z = 0$$

vorzeichenloser Vergleich

$$A > B : C = 0 \wedge Z = 0$$

$$A < B : C = 1$$

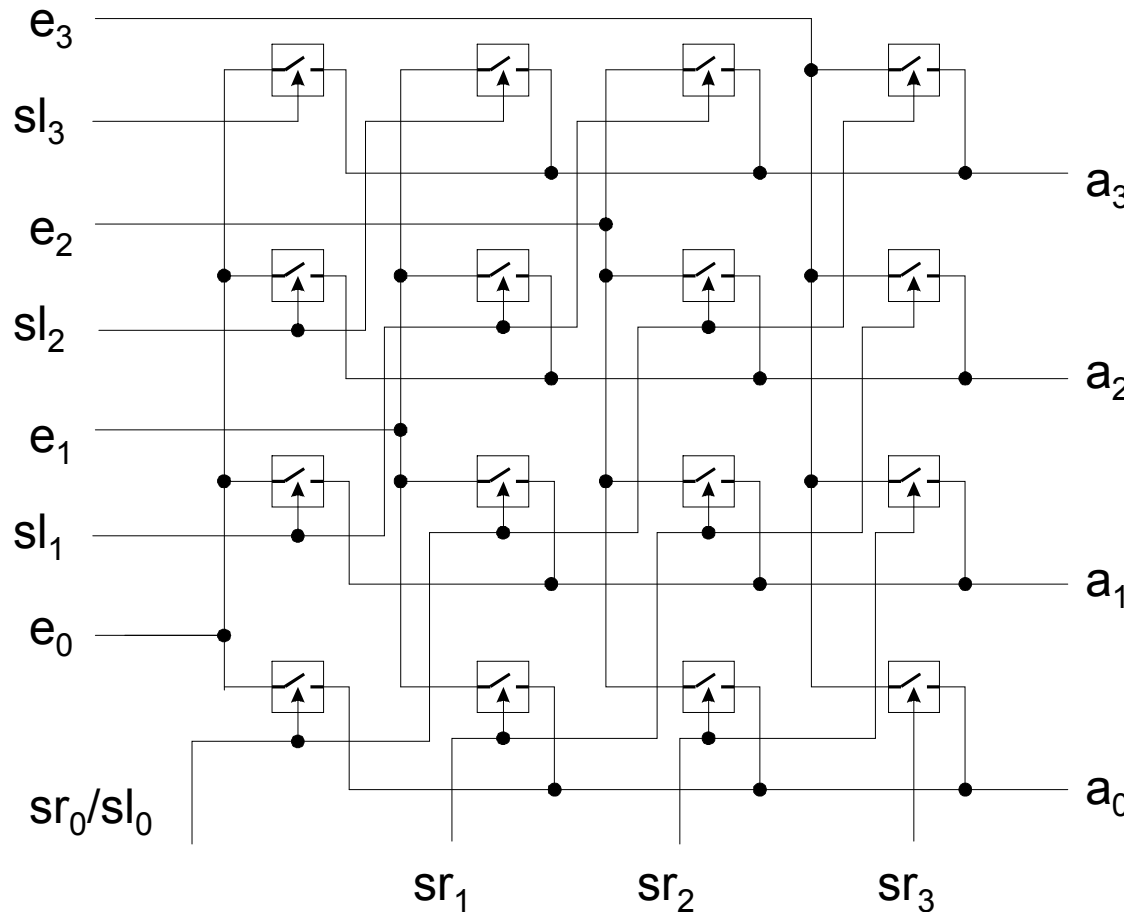
vorzeichenbehafteter Vergleich (Zweierkomplement)

$$A > B : S = 0 \wedge Z = 0$$

$$A < B : S = 1$$

2.4 Shifter

4-Bit-Barrel-Shifter auf der Basis eines Kreuzschienenverteilers



Für eine n -Bit Barrel-Shifter werden n^2 Schalter benötigt.

In nebenstehender Schaltung sind die nachgeschobenen Bits nicht definiert.

Hierbei müssen folgende Fälle unterschieden werden.

Schieben nach links:

a_2 Es wird immer 0 nachgeschoben

Schieben nach rechts

Arithmetisch: Es wird $e_{(N-1)}$

Logisch: es wird 0

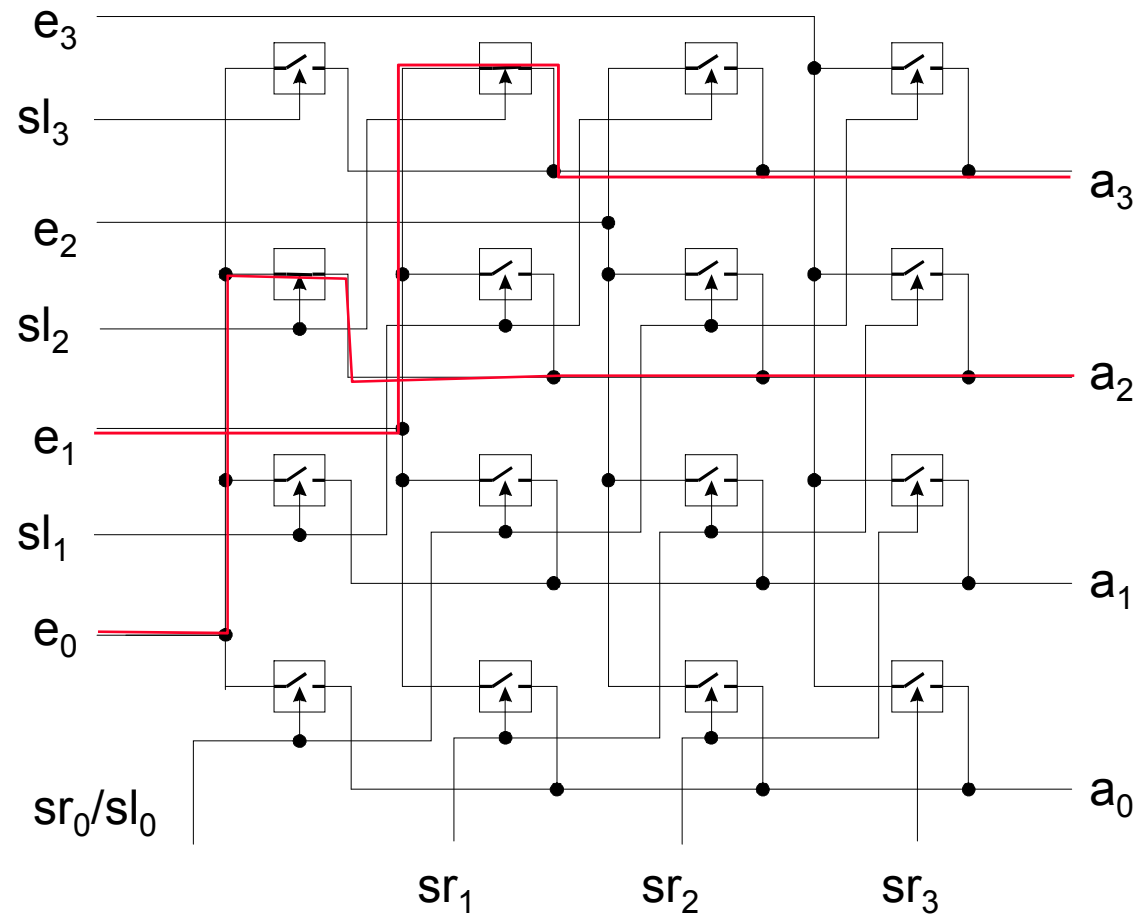
a_1 nachgeschoben.

Hierfür ist eine gesonderte Logik notwendig

a_0 Durch die Ansteuerung von $sr_k, sl_{(N-k)}$, bzw. $sl_k, sr_{(N-k)}$ wird eine Rotation um k nach rechts bzw. links realisiert.

2.4 Shifter

4-Bit-Barrel-Shifter



2.4 Speicherelemente

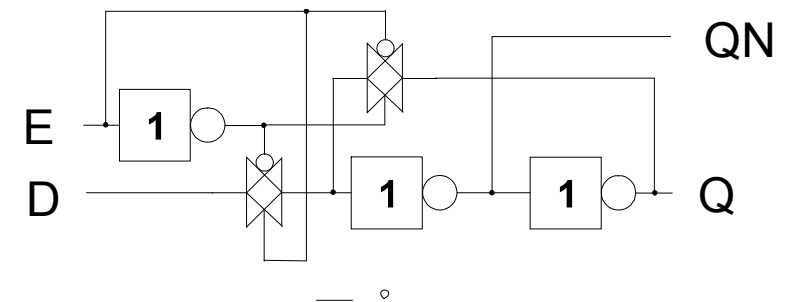
Pegelgesteuerte Latches

- Funktionsweise:
Bei hohem Pegel am Eingang E wird das Latch transparent geschaltet. Bei einem niedrigen Pegel werden die Daten gehalten.

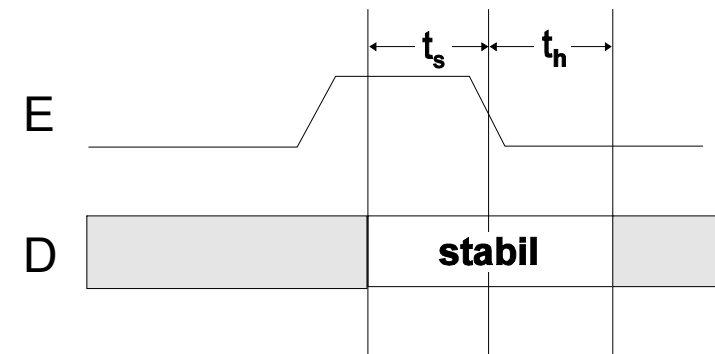
Zeitbedingungen

- **Set-up-Zeit t_s**
Die Daten müssen t_s vor der fallenden Flanke stabil anliegen
- **Hold-Zeit t_h**
Die Daten müssen t_h nach der fallenden Flanke noch stabil anliegen.

CMOS-Realisierung eines D-Latches

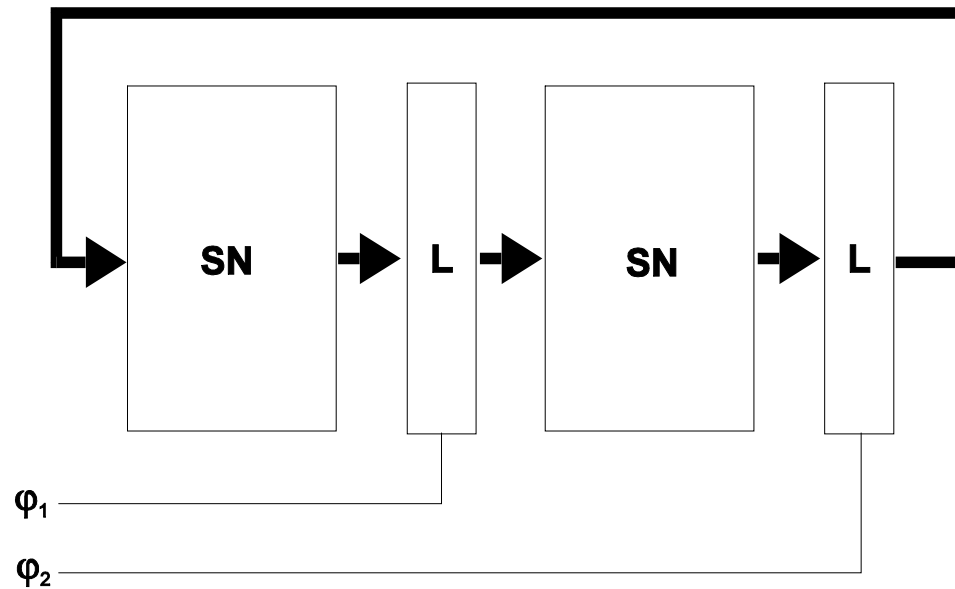


Zeitdiagramm

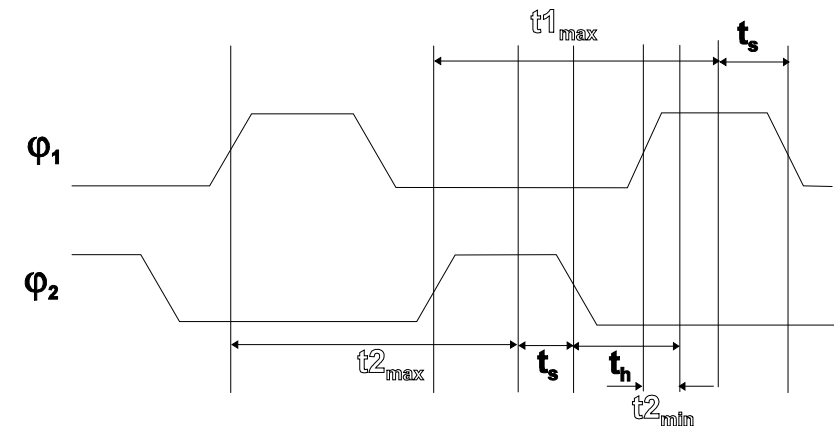


2.4 Speicherelemente

Aufbau einer Schaltung mit pegelgesteuerten Latches



Ansteuerung durch nicht überlappenden Zweiphasen-Takt. Auf ein mit ϕ_1 angesteuertes Latch muß immer ein mit ϕ_2 angesteuertes Latch folgen.



Zeitbedingungen

t_{max} ist maximale Verzögerungszeit

t_{min} ist die minimale Verzögerungszeit

2.4 Speicherelemente

Flankengetriggerte Flipflops

– Funktionsweise:

Bei niedrigem Pegel am Takteingang C ist das Latch L1 transparent geschaltet. Bei steigender Taktflanke wird das Latch L2 transparent während das Latch L1 die Daten speichert.

Zeitbedingungen

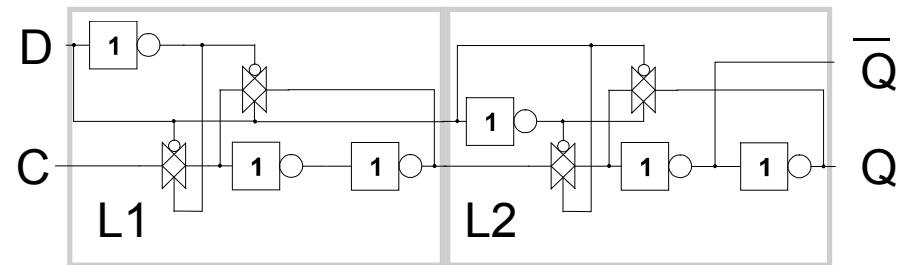
– **Set-up-Zeit t_s**

Zeitspanne vor der steigenden Taktflanke, während der die Daten bereits stabil anliegen müssen

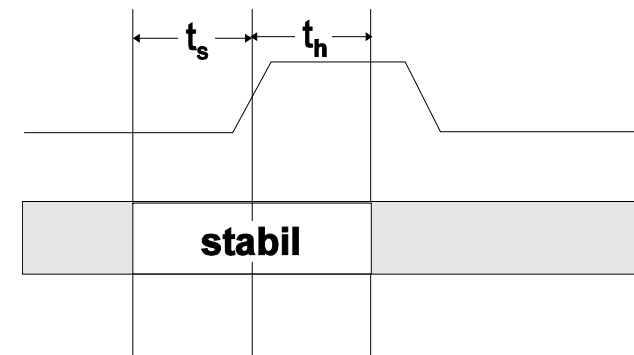
– **Hold-Zeit t_h**

Zeit nach der Taktflanke, in der die Daten stabil bleiben müssen.

Aufbau eines D-Flipflops in CMOS

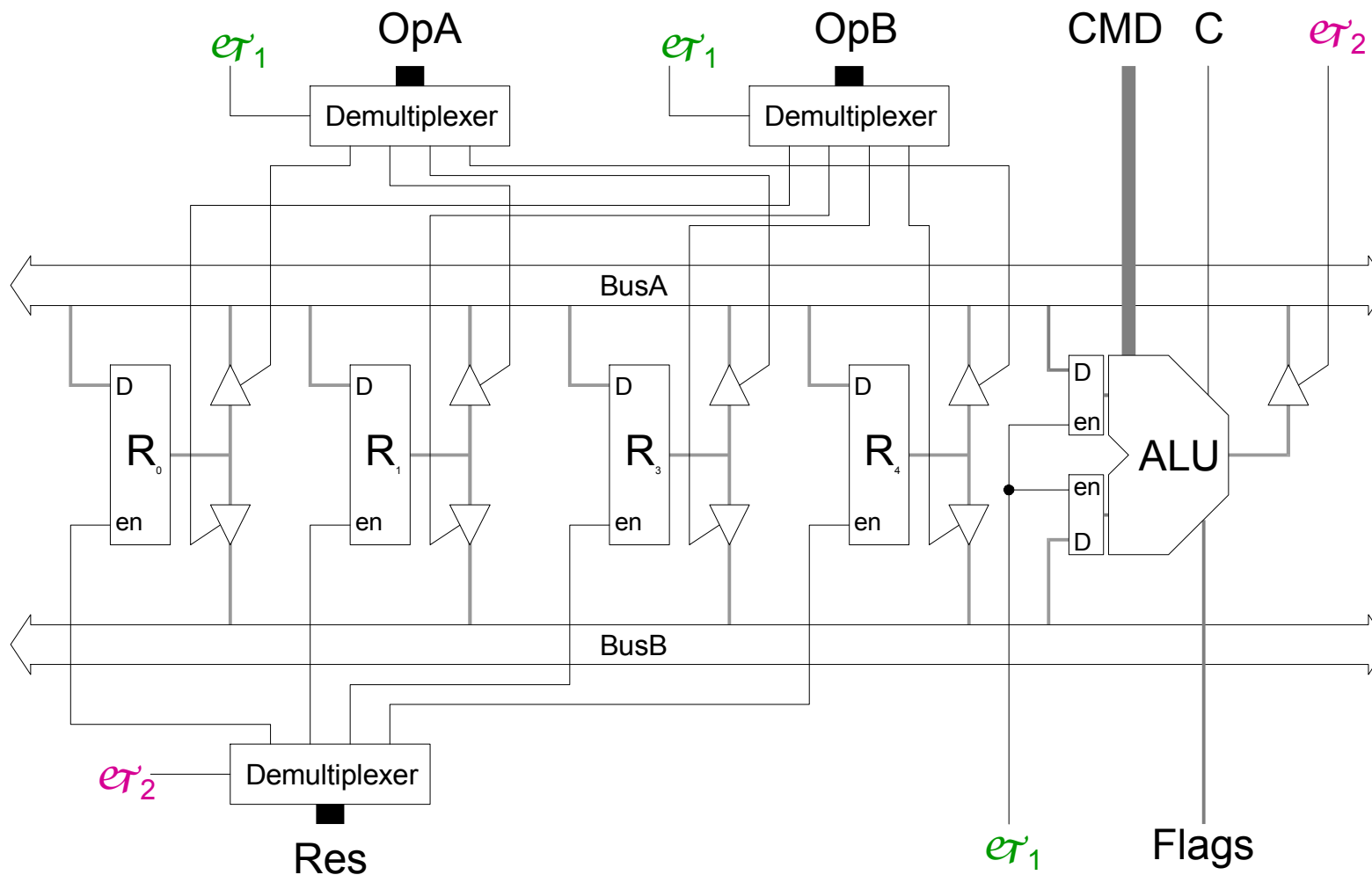


Zeitdiagramm



2.5 Datenpfad

Beispiel für einen Datenpfad



2.6 Multiplizierer

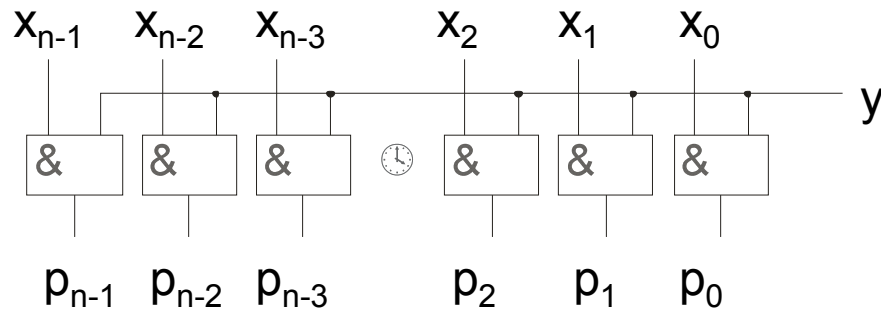
Multiplikation von zwei 1-Bit Werten

Es handelt sich hierbei um eine UND-Verknüpfung

x	y	p
0	0	0
0	1	0
1	0	0
1	1	1

Multiplikation von $N \times M$ Bit Zahlen

Multiplikation von mehreren Bits
(partielles Produkt PP)

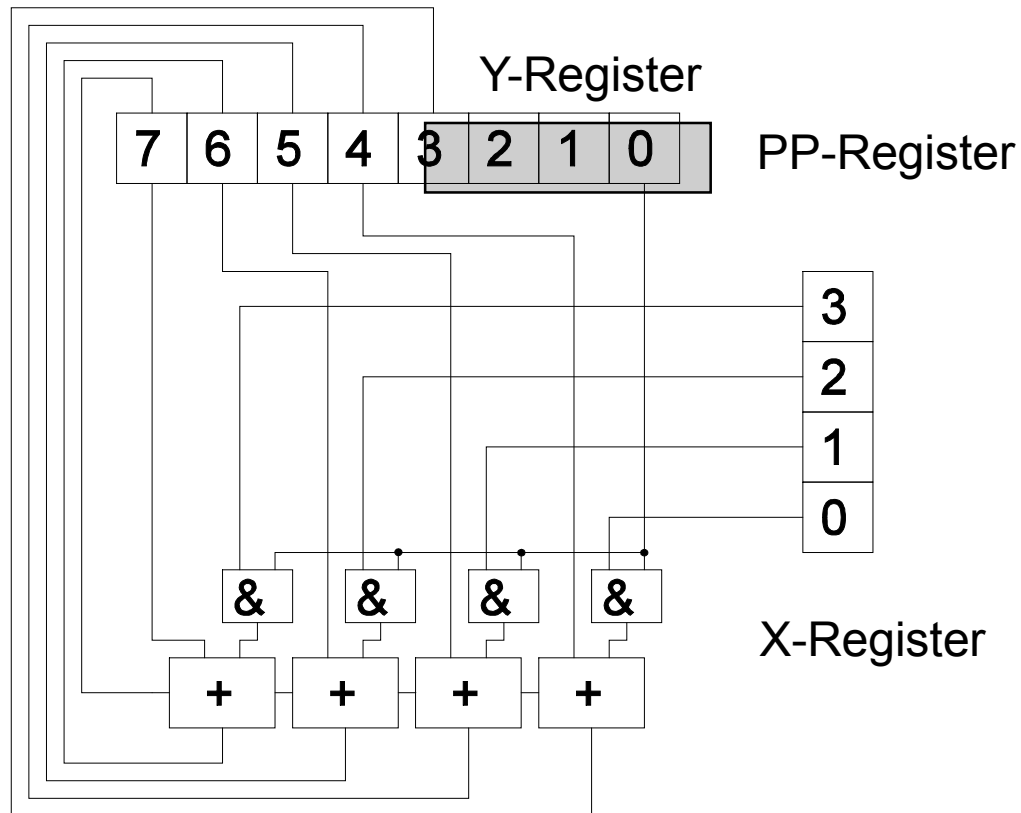


Multiplikation

				x_3	x_2	x_1	x_0	
PP_0				$x_3 y_0$	$x_2 y_0$	$x_1 y_0$	$x_0 y_0$	y_0
PP_1			$x_3 y_1$	$x_2 y_1$	$x_1 y_1$	$x_0 y_1$		y_1
PP_2		$x_3 y_2$	$x_2 y_2$	$x_1 y_2$	$x_0 y_2$			y_2
PP_3	$x_3 y_3$	$x_2 y_3$	$x_1 y_3$	$x_0 y_3$				y_3
	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

2.6.1 Sequentielle Multiplizierer

Aufbau eines 4-Bit sequentiellen Multiplizierers



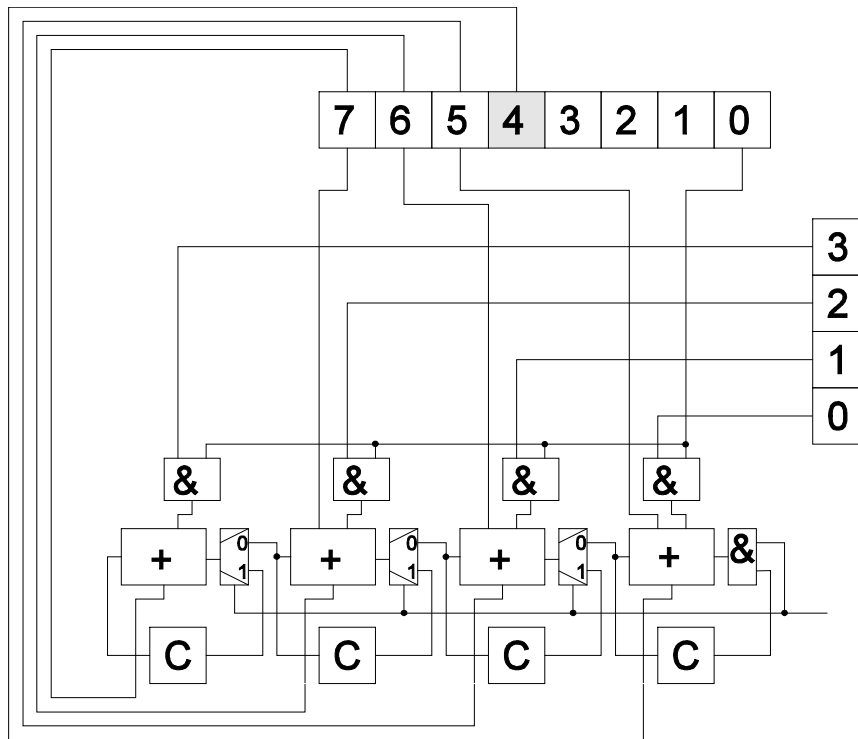
Multiplikationsalgorithmus

```

PP(3 to 0) := Y(3 to 0);
PP(7 to 0) := "0000";
for i in 0 to 3 do
  if PP(0) = '1' then
    PP(7 to 4) := X(3 to 0) + PP(7 to 4);
  end if;
  for j in 0 to 7 do
    PP(j) := PP(j+1);
  end for;
  PP(7) := '0';
end for;
  
```

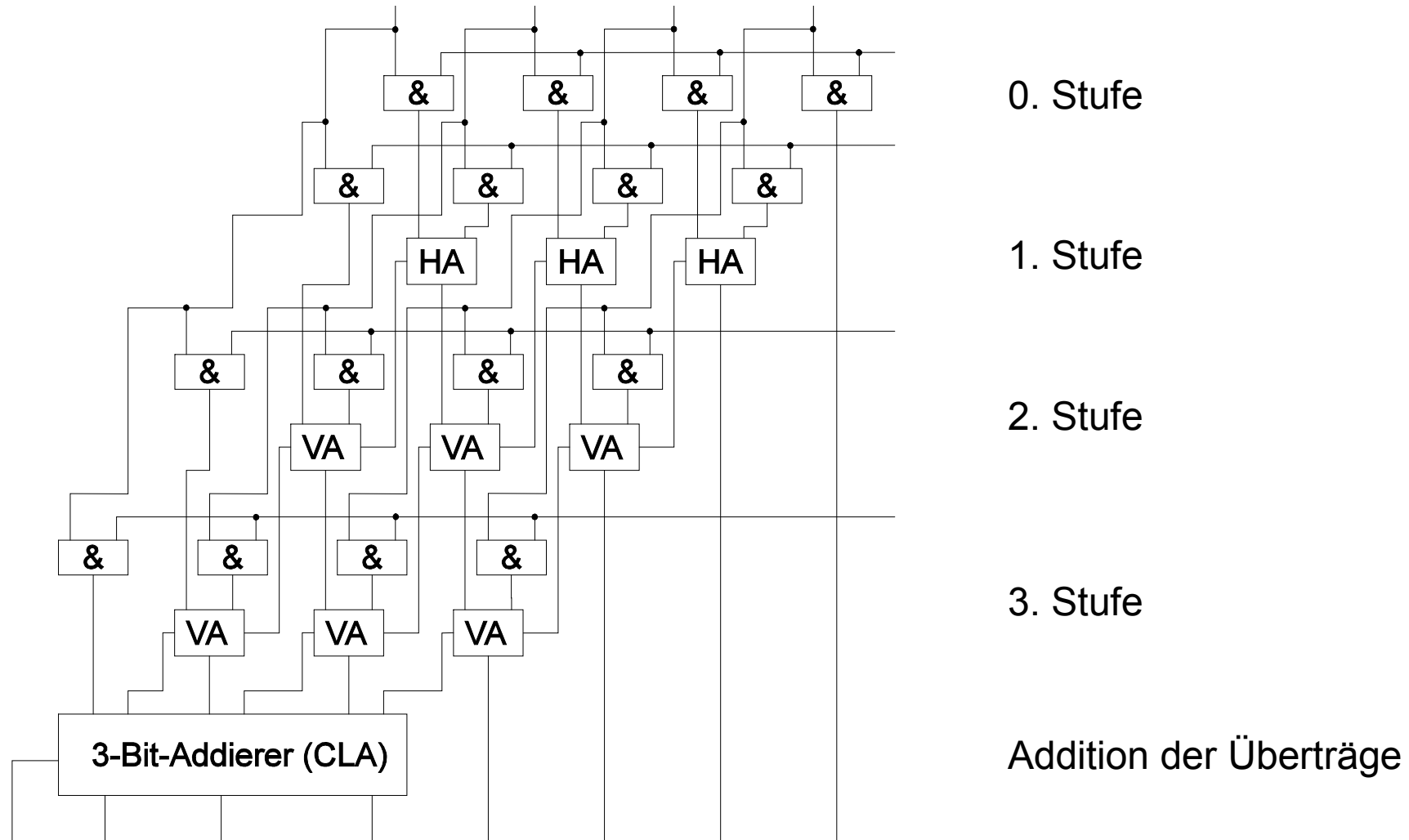

2.6.1 Sequentielle Multiplizierer mit Carry-Save

Problem: Laufzeit ist bestimmt durch Addierer



Nach der Multiplikation ist noch eine weitere Addition der Überträge erforderlich

2.6.2 Paralleler Multiplizierer(Carry Save)



2.6.1 Sequentielle Multiplizierer (Booth)

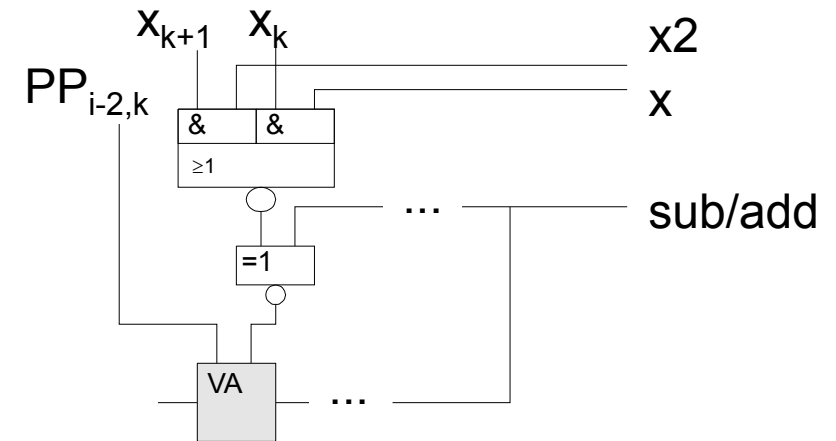
Multiplikation von Zweierkomplementzahlen

Modifizierte Booth-Decodierung (3-Bit)

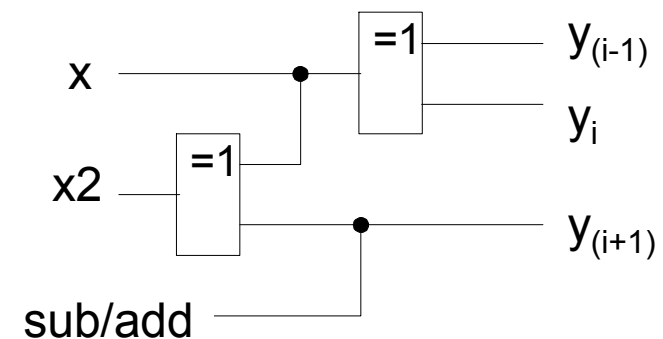
y_{i+1}	y_i	y_{i-1}	Operation
0	0	0	$PP_i = (1/4) PP_{i-2}$
0	0	1	$PP_i = (1/4) PP_{i-2} + x$
0	1	0	$PP_i = (1/4) PP_{i-2} + x$
0	1	1	$PP_i = (1/4) PP_{i-2} + 2x$
1	0	0	$PP_i = (1/4) PP_{i-2} - 2x$
1	0	1	$PP_i = (1/4) PP_{i-2} - x$
1	1	0	$PP_i = (1/4) PP_{i-2} - x$
1	1	1	$PP_i = (1/4) PP_{i-2}$

Der Multiplikator muß rechts vom LSB ein zusätzlichen Bit erhalten. Initial ist dieses 0.

Addierstufe



Decodierlogik



2.6.3 Bauth-Wooley-Multiplizierer

Darstellung von Zweierkomplement-Zahlen (Qn-Format)

$$z = -b_{(n-1)} \cdot 2^0 + \sum_{k=1}^{n-1} b_{((n-1)-k)} \cdot 2^{-k}$$

Bsp.:

$$0.1010000_2 = 0,5625_{10}$$

$$1.0110000_2 = -0,5625_{10}$$

Die Multiplikation von zwei Zahlen x und y ergibt sich zu

$$\underbrace{\sum_{k=1}^{n-1} x_{((n-1)-k)} \cdot 2^{-k}}_{x_{mant}} \cdot \underbrace{\sum_{k=1}^{n-1} y_{((n-1)-k)} \cdot 2^{-k}}_{y_{mant}}$$

$$x \cdot y = x_{(n-1)} \cdot y_{(n-1)} + \underbrace{x_{mant} \cdot y_{mant}}_{\text{normale Multiplikation}} + \underbrace{(-x_{(n-1)}) \cdot y_{mant} + (-y_{(n-1)}) \cdot x_{mant}}_{\text{Sonderbehandlung (Korr)}}$$

Der Korrekturterm $Korr$ berechnet sich zu

$$Korr = (-x_{(n-1)}) \cdot y_{mant} + (-y_{(n-1)}) \cdot x_{mant}$$

$$Korr = \sum_{k=1}^{n-1} x_{(n-1)} \cdot y_{((n-1)-k)} 2^{-k} + 2^{-(n-1)} + \sum_{k=1}^{n-1} y_{(n-1)} \cdot x_{((n-1)-k)} 2^{-k} + 2^{-(n-1)}$$

$$Korr = \sum_{k=1}^{n-1} (x_{(n-1)} \cdot y_{((n-1)-k)} + y_{(n-1)} \cdot x_{((n-1)-k)}) 2^{-k} + 2^{-(n-2)}$$

2.6.3 Bauth-Wooley-Multiplizierer

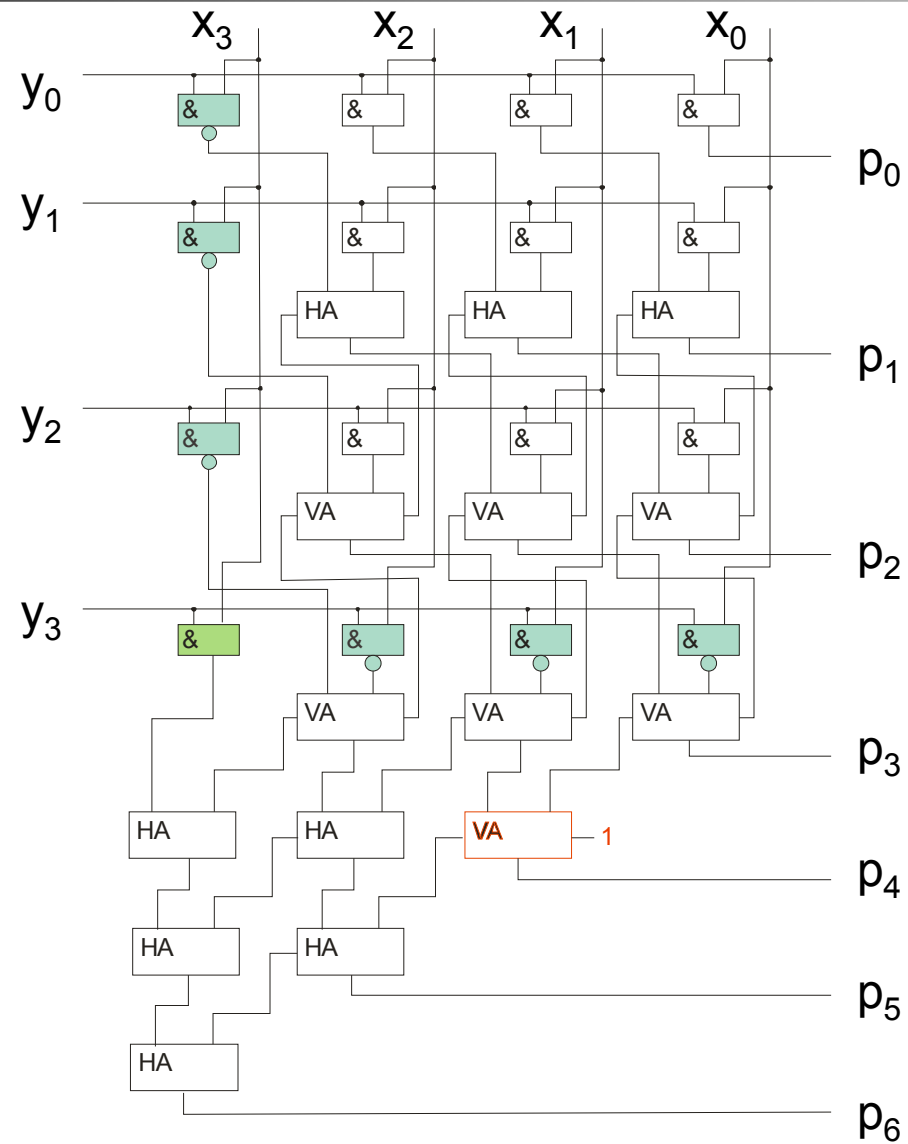
Beispiel: $x = x_3x_2x_1x_0$, $y = y_3y_2y_1y_0$

$$\begin{aligned}
 x \cdot y &= (-x_3 + x_2 2^{-1} + x_1 2^{-2} + x_0 2^{-3}) (-y_3 + y_2 2^{-1} + y_1 2^{-2} + y_0 2^{-3}) \\
 &= \underbrace{x_3 y_3 + (x_2 2^{-1} + x_1 2^{-2} + x_0 2^{-3}) (y_2 2^{-1} + y_1 2^{-2} + y_0 2^{-3})}_{P} \\
 &\quad - x_3 (y_2 2^{-1} + y_1 2^{-2} + y_0 2^{-3}) + -y_3 (x_2 2^{-1} + x_1 2^{-2} + x_0 2^{-3}) \\
 &= P + (\overline{x_3 y_2} 2^{-1} + \overline{x_3 y_1} 2^{-2} + \overline{x_3 y_0} 2^{-3} + 2^{-3}) + (\overline{y_3 x_2} 2^{-1} + \overline{y_3 x_1} 2^{-2} + \overline{y_3 x_0} 2^{-3} + 2^{-3}) \\
 &= P + (\overline{x_3 y_2} + \overline{y_3 x_2}) 2^{-1} + (\overline{x_3 y_1} + \overline{y_3 x_1} + 1) 2^{-2} + (\overline{x_3 y_0} + \overline{y_3 x_0}) 2^{-3} \\
 &\quad = 2 \triangleleft 2^{-3}
 \end{aligned}$$

Schema

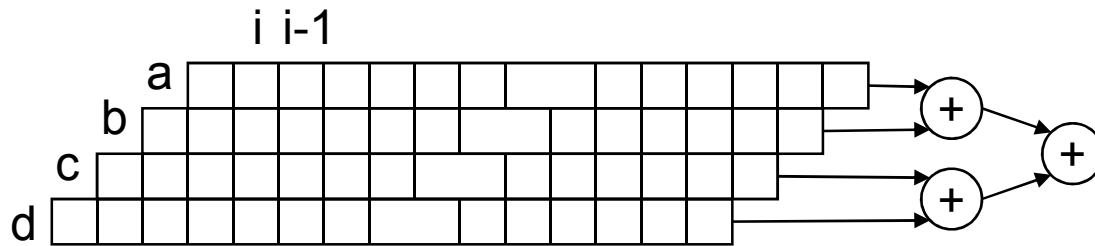
$2^0(\text{VZ})$	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
			$\overline{x_3 y_0}$	$x_2 y_0$	$x_1 y_0$	$x_0 y_0$
		$\overline{x_3 y_1}$	$x_2 y_1$	$x_1 y_1$	$x_0 y_1$	
	$\overline{x_3 y_2}$	$\overline{x_2 y_2}$	$\overline{x_1 y_2}$	$x_0 y_2$		
$x_3 y_3$	$\overline{x_2 y_3}$	$\overline{x_1 y_3}$	$\overline{x_0 y_3}$			
		1				
p_6	p_5	p_4	p_3	p_2	p_1	p_0

2.6.3 Bauth-Wooley-Multiplizierer



2.6.3 Wallace-Baum-Multiplizierer

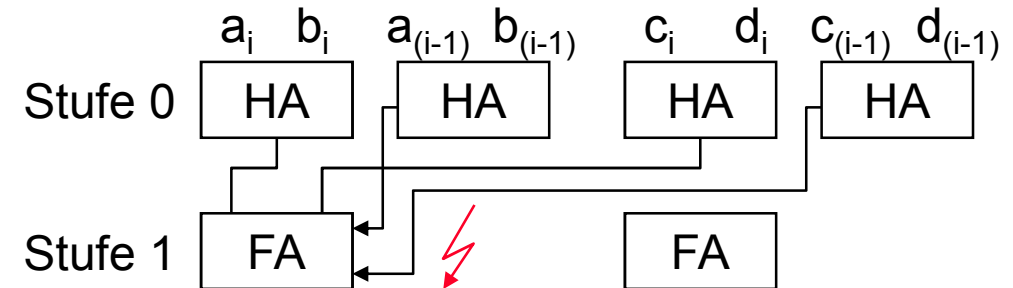
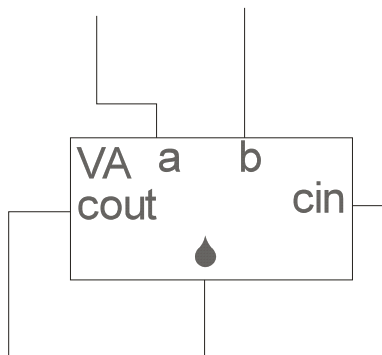
Folgender Ansatz ist ungeeignet



Lösung: Mehrbitaddierer (sog. Zähler counter)

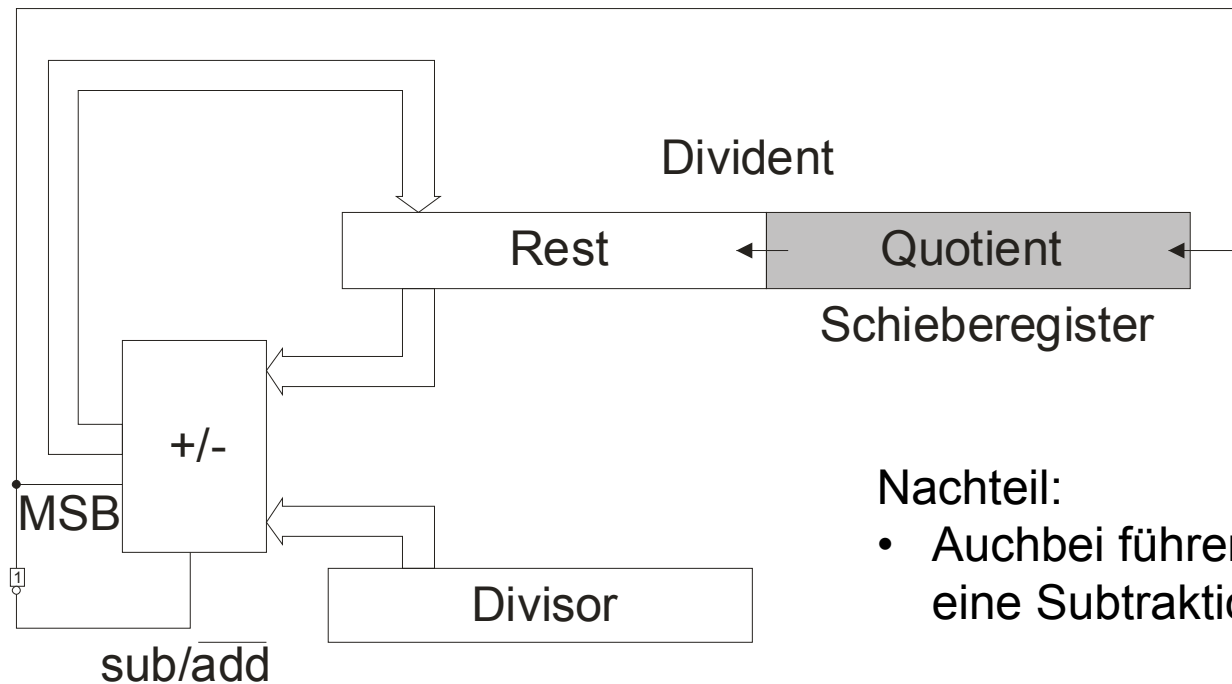
3:2 Zähler

7:3 Zähler



2.7 Hardware-Dividierer

Division ohne Wiederherstellung des Zwischenergebnisses (vorzeichenlos)



Nachteil:

- Auch bei führenden Nullen wird eine Subtraktion ausgeführt

Hardware-Dividierer

Redundante Zahlendarstellung

Bisher erfolgte die Darstellung einer Zahl bezüglich einer Basis B durch

$$z = \sum_{k=0}^{n-1} b_k B^k \quad \text{mit } b_k \in \{0, \dots, (B-1)\}$$

d. h. als Ziffern sind nur positive Werte zugelassen. Allerdings können Zahlen auch mit negativen Ziffern dargestellt werden. Damit sind die $b_k \in \{-(B-1), \dots, (B-1)\}$

Somit kann die Zahl

$Z = 1957_{10}$ dargestellt werden durch $2-16-30_R$

SRT-Division

Benannt nach Sweeney, Robertson und Tochter

Vorraussetzungen (Divisor D , $R^{(i)}$ partieller Rest)

$0.5 < |D| < 1$, $0.5 < |2R^{(i)}| < 1$ (wird durch Schieben erreicht)

die Ergebnisbits q_i sind aus $\{-1, 0, 1\}$

Division zur Basis $b > 1$ (High-Radix-Division)

2.7 Iterative Division

Newton-Raphson-Verfahren

Approximiere Nullstelle einer Funktion $f(x) \equiv 0 \big|_{x=\frac{1}{D}}$

Funktion $f(x) = \frac{1}{D} - 1$ ungeeignet, aber $f(x) = \frac{1}{x} - D$ möglich

Iterationsvorschrift

$$x^{(i+1)} = x^{(i)} - \frac{f(x)}{f'(x)} = x^{(i)} - \frac{\frac{1}{x^{(i)}} - D}{-\frac{1}{(x^{(i)})^2}} = x^{(i)} + x^{(i)} - D(x^{(i)})^2 = x^{(i)}(2 - Dx^{(i)})$$

Wahl von $x^{(0)}$

$$x^{(0)} = T_1 - T_2 D \text{ mit } T_1 = 2.9142, T_2 = 2 \quad \text{Anzahl Iterationen } k = \left\lceil \log_2 \frac{-\text{Bitanz.}}{\log_2 0,0858} \right\rceil$$

Bei 32 Bit ist $k = 4$ bei 64 Bit ist $k = 5$

2.8. Der CORDIC-Algorithmus

CORDIC (Coordinate Rotation Digital Computer)

Rotation eines Vektors um den Winkel φ

$$x' = x \cos(\varphi) - y \sin(\varphi)$$

$$y' = x \sin(\varphi) + y \cos(\varphi)$$

$$x' = \cos(\varphi)(x - y \tan(\varphi))$$

$$y' = \cos(\varphi)(y + x \tan(\varphi))$$

Beschränkung auf Drehungen mit $\tan(\varphi_i) = \pm 2^{-i}$. Hierbei soll eine beliebige Drehung durch eine iterative Folge von immer kleineren Drehungen in positive oder negative Richtung ersetzt werden. Da weiterhin $\cos(\varphi) = \cos(-\varphi)$, kann zu jedem φ_i eine Konstante $K_i = \cos(\arctan(2^{-i}))$ bestimmt werden.

$$x_{i+1} = K_i(x_i - y_i \cdot d_i \cdot 2^{-i})$$

$$y_{i+1} = K_i(y_i + x_i \cdot d_i \cdot 2^{-i})$$

mit

$$K_i = \cos(\arctan(2^{-i})) = \frac{1}{\sqrt{1+2^{-2i}}} \quad \text{und} \quad d_i = \pm 1$$

2.8. Der CORDIC-Algorithmus

Die Schaltung besitzt somit einen Verstärkungsfaktor von

$$A_n = \prod_{i=1}^n \sqrt{1 + 2^{-2i}}$$

Durch die Einführung eines Speichers für den Drehwinkel

$$z_{i+1} = z_i - d_i \arctan(2^{-i})$$

Der Algorithmus kann in zwei Modi betrieben werden:

Rotationsmodus

$$x_{i+1} = x_i - y_i d_i 2^{-i}$$

$$y_{i+1} = y_i + x_i d_i 2^{-i}$$

$$z_{i+1} = z_i - d_i \arctan(2^{-i})$$

$$d_{i+1} = \begin{cases} -1 & \text{falls } z_i > 0 \\ 1 & \text{sonst} \end{cases}$$

Vektormodus

$$x_{i+1} = x_i - y_i d_i 2^{-i}$$

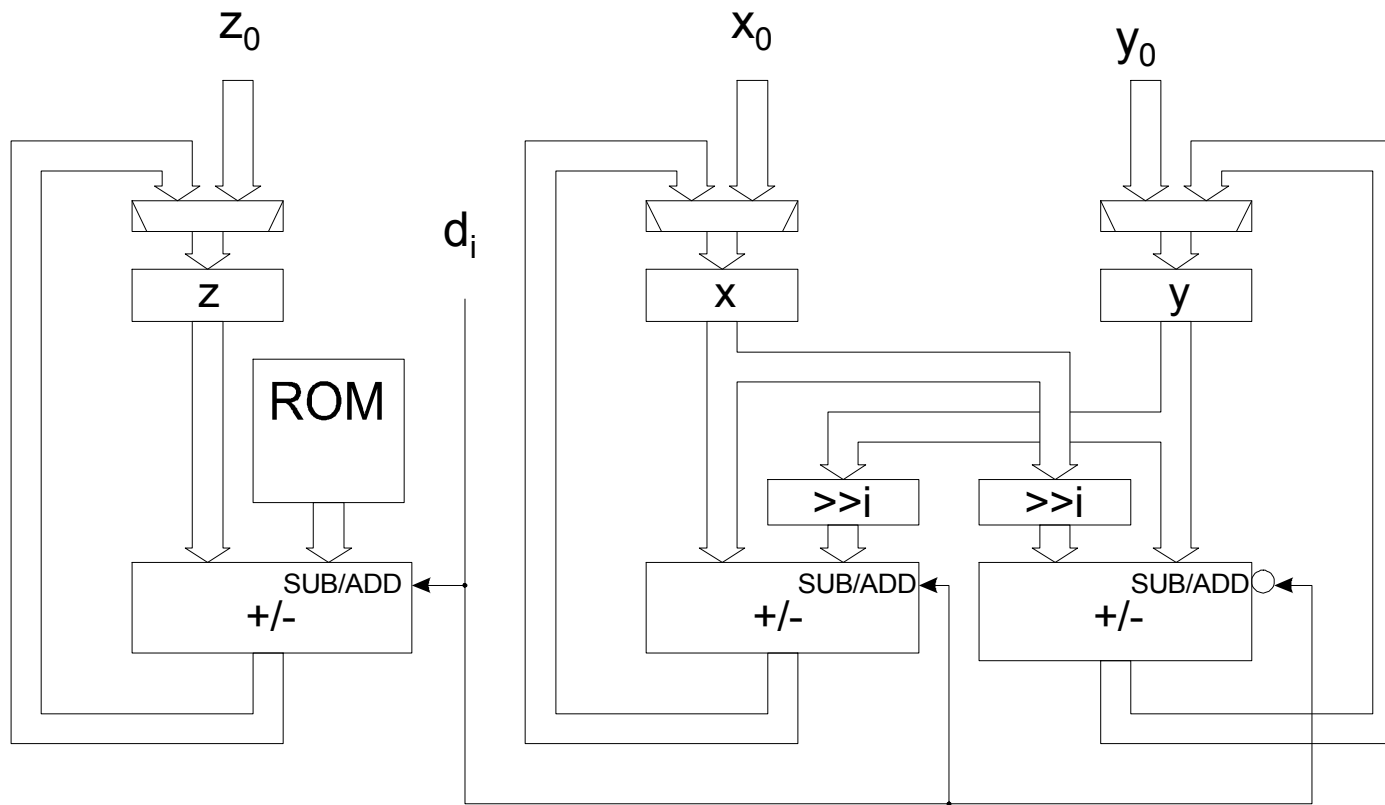
$$y_{i+1} = y_i + x_i d_i 2^{-i}$$

$$z_{i+1} = z_i - d_i \arctan(2^{-i})$$

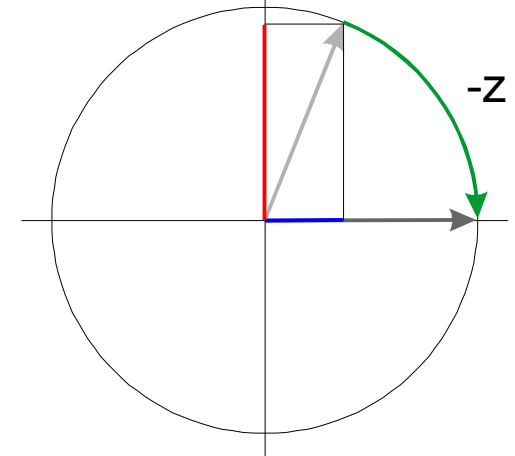
$$d_{i+1} = \begin{cases} -1 & \text{falls } y_{i+1} < 0 \\ 1 & \text{sonst} \end{cases}$$

2.8. Der CORDIC-Algorithmus

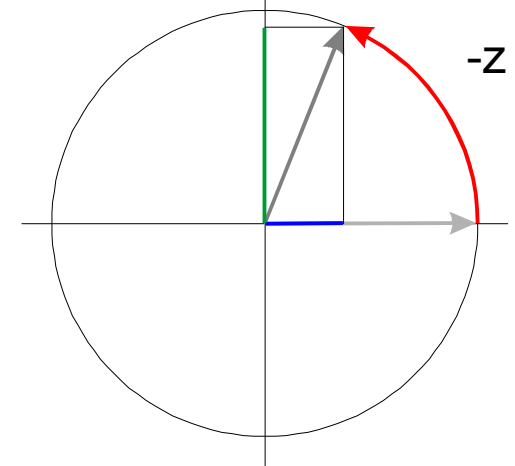
Schaltungsstruktur (bit-parallele Implementierung)



Vektormodus



Rotationsmodus



2.8 Der CORDIC-Algorithmus

Allgemeines Iterationsschema

$$x_i \leftarrow x_i \oplus y_i d_i 2^{-i}$$

$$y_i \leftarrow y_i \oplus x_i d_i 2^{-i}$$

$$z_i \leftarrow z_i \oplus d_i m_i$$

Trigonometrische Funktionen

$$x_n \leftarrow \cos \quad y_n \leftarrow \sin$$

$$m_i \leftarrow \arctan(2^{-i}) \quad d_i \leftarrow \text{sgn } z_i \quad m \leftarrow 1$$

$$x_0 \leftarrow \cos(m) \quad C \quad y_0 \leftarrow 0 \quad z_0 \leftarrow C$$

$$z_n \leftarrow \arctan(y_0 / x_0) \quad x_n \leftarrow C \sqrt{x_0^2 + y_0^2}$$

$$m_i \leftarrow \arctanh(2^{-i}) \quad d_i \leftarrow \text{sgn } y_i \quad m \leftarrow 1$$

$$z_0 \leftarrow 0$$

Exponentialfunktion und Logarithmus

$$x_n \leftarrow \cosh \quad y_n \leftarrow \sinh \quad e^x \leftarrow x_n \oplus y_n$$

$$m_i \leftarrow \operatorname{arctanh}(2^{-i}) \quad d_i \leftarrow \text{sgn } z_i \quad m \leftarrow 1$$

$$x_0 \leftarrow \cosh(m) \quad C' \quad y_0 \leftarrow 0 \quad z_0 \leftarrow C'$$

$$z_n \leftarrow \operatorname{arctanh}(y_0 / x_0) \quad x_n \leftarrow C' \sqrt{x_0^2 + y_0^2}$$

$$\ln x \leftarrow 2 \operatorname{arctanh} \left(\frac{x-1}{x+1} \right) \quad m \leftarrow 1$$

$$m_i \leftarrow \operatorname{arctanh}(2^{-i}) \quad d_i \leftarrow \text{sgn } y_i$$

$$z_0 \leftarrow 0$$

Multiplikation und Division

$$y_n \leftarrow x_0 \cdot y_0$$

$$m_i \leftarrow 2^{-i} \quad d_i \leftarrow \text{sgn } z_k$$

$$y_0 \leftarrow 0$$

$$y_n \leftarrow y_0 / x_0$$

$$m_i \leftarrow 2^{-i} \quad d_i \leftarrow \text{sgn } y_k$$

$$z_0 \leftarrow 0$$

2.8 Der CORDIC-Algorithmus

Sinus/Kosinus

m	d	x	y	z
	1	0,8588	0	0,7854
0,46365	1	0,8588	0,4294	0,3218
0,24498	1	0,7514	0,6441	0,0768
0,12435	-1	0,6709	0,738	-0,0476
0,06242	1	0,7171	0,6961	0,0148
0,03124	-1	0,6953	0,7185	-0,0164
0,01562	-1	0,7065	0,7076	-0,0008
0,00781	1	0,7121	0,7021	0,007
0,00391	1	0,7093	0,7049	0,0031
Taschenrechner:		0,7071	0,7071	

Exponentialfunktion

m	d	x	y	z
	1	1,2051	0	1
0,54931	1	1,2051	0,6026	0,4507
0,25541	1	1,3558	0,9038	0,1953
0,12566	1	1,4688	1,0733	0,0696
0,06258	1	1,5358	1,1651	0,007
0,03126	-1	1,5722	1,2131	-0,0242
0,01563	-1	1,5533	1,1885	-0,0086
0,00781	-1	1,544	1,1764	-0,0008
0,00391	1	1,5394	1,1704	0,0031
		e=x+y	2,7098	
Taschenrechner:			2,7183	

Fehler \oplus 8.Bit = \oplus 0,00390625

2.8 Der CORDIC-Algorithmus

Arkustangens

m	d	x	y	z
	-1	1	1	0
0,46365	-1	1,5	0,5	0,4636
0,24498	-1	1,625	0,125	0,7086
0,12435	1	1,6406	-0,0781	0,833
0,06242	-1	1,6455	0,0244	0,7706
0,03124	1	1,6463	-0,027	0,8018
0,01562	1	1,6467	-0,0013	0,7862
0,00781	-1	1,6467	0,0116	0,7784
0,00391	-1	1,6467	0,0051	0,7823
Taschenrechner:				0,7854

Logarithmus

m	d	x	y	z
	1	1,5	-0,5	0
0,54931	-1	1,25	0,25	-0,5493
0,25541	1	1,1875	-0,0625	-0,2939
0,12566	-1	1,1797	0,0859	-0,4196
0,06258	-1	1,1743	0,0122	-0,357
0,03126	1	1,1739	-0,0245	-0,3257
0,01563	1	1,1736	-0,0061	-0,3413
0,00781	-1	1,1735	0,003	-0,3491
0,00391	1	1,1735	-0,0016	-0,3452
		ln 0,5	-0,6905	
Taschenrechner:			-0,6931	

3.1 Logikminimierung

Zwei Arten von Logikminimierung

- Zweistufig:

Ausgangspunkt ist eine disjunktive Form (DF)

Das Ziel ist eine DF mit möglichst wenigen Termen

- Mehrstufig (Logiksynthese)

Ausgangspunkt bel. Logikbeschreibung

Ziel faktorisierte Form mit möglichst wenigen Literalen

Mögliche Zielarchitekturen sind:

- FPGA: Hier muß die Logikfunktion so ungeformt werden, daß alle Teilfunktionen durch CLBs realisiert werden können.
- Zellbasierte Technologie (Standardzellen, Gate-Arrays) : Die Logikfunktion muß auf Bibliotheksgatter abgebildet werden (Technologieabbildung)

3.1.1 Der Cube-Kalkül

Boolesche Funktionen, Cubes und die Realisierung als PAL

$$F : B^3 \rightarrow B^2$$

in algebraischer Darstellung

$$f_1 = x_1 \bar{x}_2 \vee \bar{x}_2 x_3$$

$$f_2 = x_2 \bar{x}_3 \vee \bar{x}_2 x_3 \vee \bar{x}_1 x_2 x_3$$

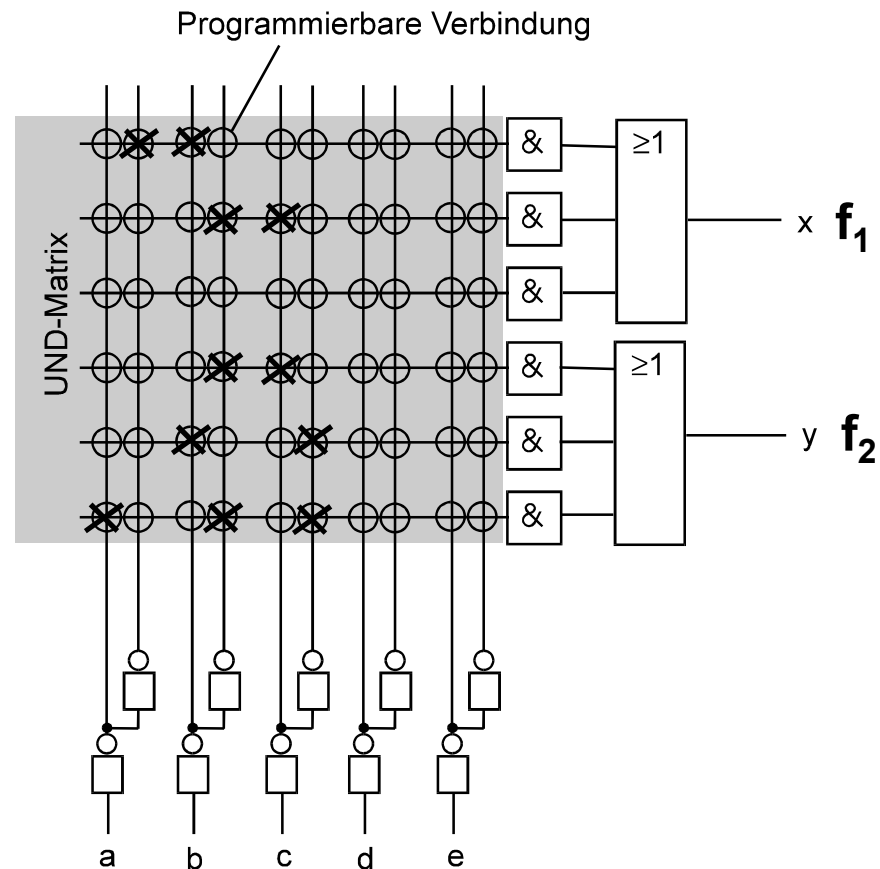
Zugehörige Cubedarstellung

$$f_1 = \{(102\ 43), (210\ 43)\}$$

$$f_2 = \{(210\ 34), (201\ 34), (011\ 34)\}$$

Matrixdarstellung

$$M(F) = \begin{array}{l} 102\ 43 \\ 210\ 43 \\ 210\ 34 \\ 201\ 34 \\ 011\ 34 \end{array}$$



3.1 Definitionen

Def 3.1: Cubes

Ein Cube ist ein m -Tupel $c = (c_1, c_2, \dots, c_m, c_{m+1}, \dots, c_{m+n})$

mit $c_i \in \{\emptyset, 0, 1, 2\}$ für $i = 1..m$ und $c_i \in \{\emptyset, 4, 3\}$ für $i = m+1..m+n$.

Die c_i mit $i = 1..m$ heißen der Eingangsteil $I(c)$ eines Cubes, die c_i mit $i = m+1..m+n$ heißen der Ausgangsteil $O(c)$ eines Cubes. Der Wert \emptyset bezeichnet den leeren Eintrag. Ist ein $c_i = \emptyset$ so wird der Cube als leerer Cube \emptyset bezeichnet.

Def. 3.2: Überdeckung

Eine Überdeckung $C = \{ c \mid i = 1..I \}$ ist eine Menge von Cubes.

3.1 Definitionen

Zusammenhang zwischen Überdeckungen und boolesche Funktionen

Sei $F : B^m \rightarrow B^n$ eine boolesche Funktion mit den Eingangsvariablen x_1, x_2, \dots, x_m und den Teilfunktionen f_1, f_2, \dots, f_n .

Eine Überdeckung $C(F)$ kann als Darstellung der Funktion in disjunktiver Darstellung angesehen werden mit:

Der i -te Cube c^i stellt den i -ten UND-Term dar, wobei x_k im Term enthalten ist falls c_k^i gleich 1 ist und \bar{x}_k im Term enthalten ist falls c_k^i gleich 0 ist. Der Term enthält x_k nicht, falls c_k^i gleich 2 ist. Falls der i -te Term zur j -ten Funktion f_j beiträgt ist c_{m+j}^i gleich 4 sonst ist c_{m+j}^i gleich 3.

Def.3.3: Überdeckung von Cubes:

Ein Cube c **überdeckt** Cube d ($c \supseteq d$), falls für jedes Element c_i entweder $c_i \supseteq d_i$ oder $c_i \neq d_i$ bezüglich d_i nach nebenstehender Tabelle gilt.

Der Cube d wird von Cube c **echt** überdeckt ($c \subsetneq d$), falls $c_i \neq d_i$ für ein c_i gilt.

		d_j		
		0	1	2
c_i	0	\supseteq	$\not\supseteq$	$\not\supseteq$
	1	$\not\supseteq$	\supseteq	$\not\supseteq$
	2	\supseteq	\supseteq	\supseteq

3.1 Definitionen

Beispiel

$$F : B^3 \rightarrow B^2$$

in algebraischer Darstellung

$$f_1 = \bar{x}_1 x_2 \vee \bar{x}_2 x_3$$

$$f_2 = x_2 \bar{x}_3 \vee \bar{x}_2 x_3 \vee x_1 x_2 x_3$$

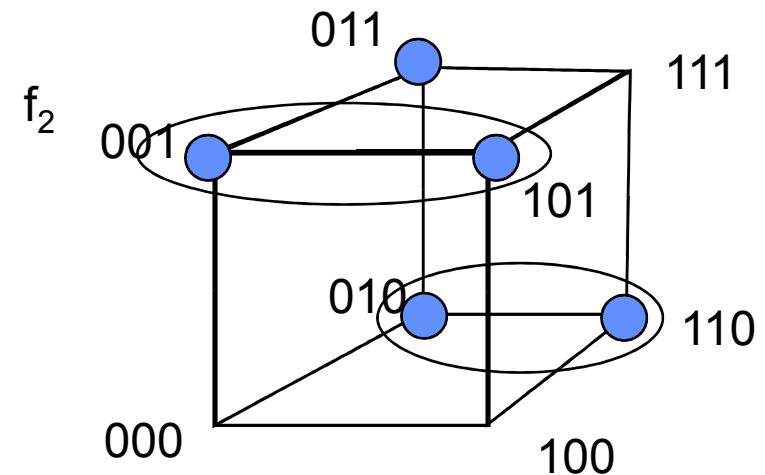
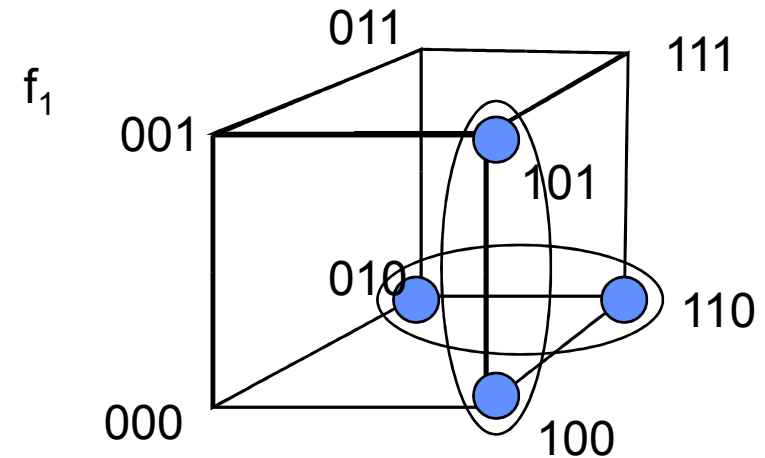
Zugehörige Cubedarstellung

$$C(f_1) = \{(102\ 43), (210\ 43)\}$$

$$C(f_2) = \{(210\ 34), (201\ 34), (011\ 34)\}$$

Matrixdarstellung

	102 43		102 43
	210 43		210 44
M(F)=	210 34	M(F)=	201 34
	201 34		011 34
	011 34		



3.1.1 Der Cube-Kalkül

Boolesche Funktionen, Cubes und die Realisierung als PLA

$$F : B^3 \rightarrow B^2$$

mit algebraischer Darstellung

$$f_1 = x_1 \bar{x}_2 \vee \bar{x}_2 x_3$$

$$f_2 = x_2 \bar{x}_3 \vee \bar{x}_2 x_3 \vee x_1 x_2 x_3$$

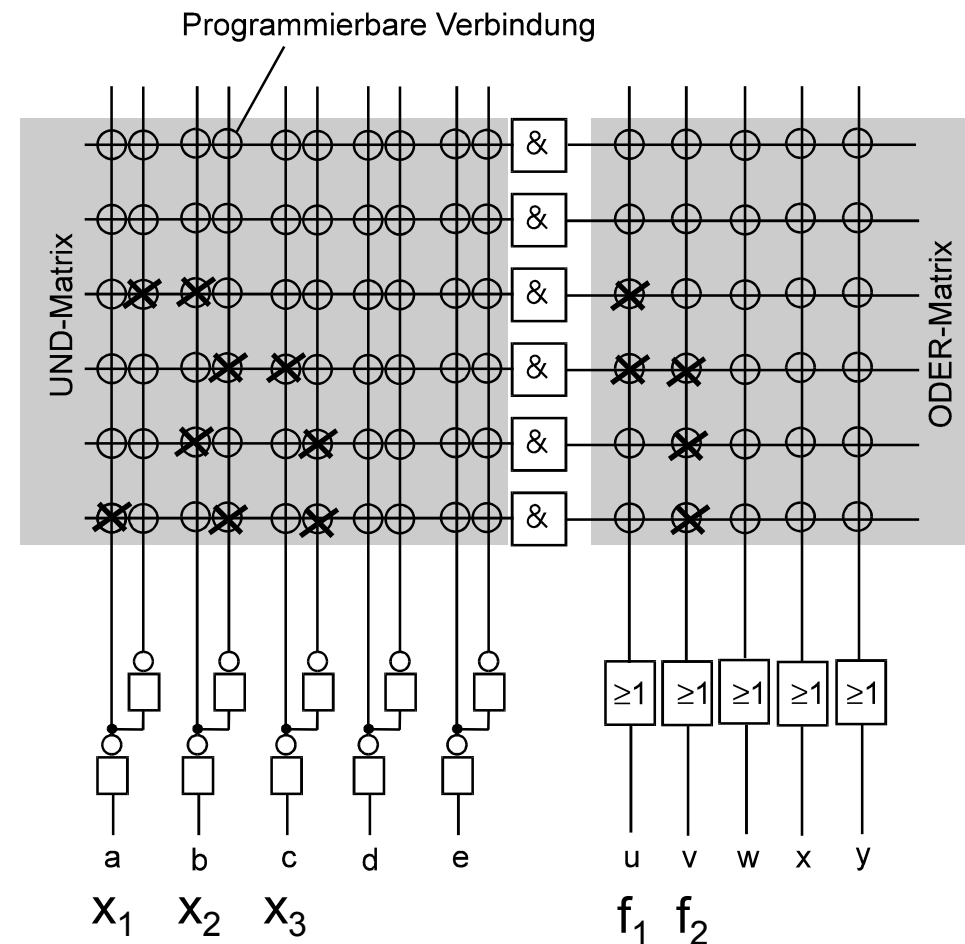
Zugehörige Cubedarstellung

$$f_1 = \{(102\ 43), (210\ 43)\}$$

$$f_2 = \{(210\ 34), (201\ 34), (011\ 34)\}$$

Matrixdarstellung

$$M(F) = \begin{array}{cc} & \begin{matrix} 102\ 43 \\ 210\ 44 \\ 201\ 34 \\ 011\ 34 \end{matrix} \end{array}$$



3.1.1 Der Cube-Kalkül

Operationen mit Cubes

Def 3. Der Durchschnittscube $c = (c_1, c_2, \dots, c_m, c_{m+1}, \dots, c_{m+k})$ von zwei Cubes $a = (a_1, a_2, \dots, a_m, a_{m+1}, \dots, a_{m+k})$ und $b = (b_1, b_2, \dots, b_m, b_{m+1}, \dots, b_{m+k})$ ist elementweise definiert. Wobei gilt

$$C_i = \begin{cases} a_i \cap b_i & \text{falls } a_i \cap b_i \neq \phi \\ \phi & \text{(leer) sonst} \end{cases}$$

Die elementweise Operation ist durch nebenstehende Tabelle definiert.

	0	1	2	3	4
0	0	\nearrow	0	\nearrow	\nearrow
1	\nearrow	1	1	\nearrow	\nearrow
2	0	1	2	\nearrow	\nearrow
3	\nearrow	\nearrow	\nearrow	3	3
4	\nearrow	\nearrow	\nearrow	3	4

3.1.1 Der Cube-Kalkül

Beispiel

$$a_1 = (2 \ 1 \ 2 \ 4)$$

$$b_1 = (1 \ 2 \ 2 \ 4)$$

$$a_1 \hat{\curvearrowright} b_1 = (1 \ 1 \ 2 \ 4)$$

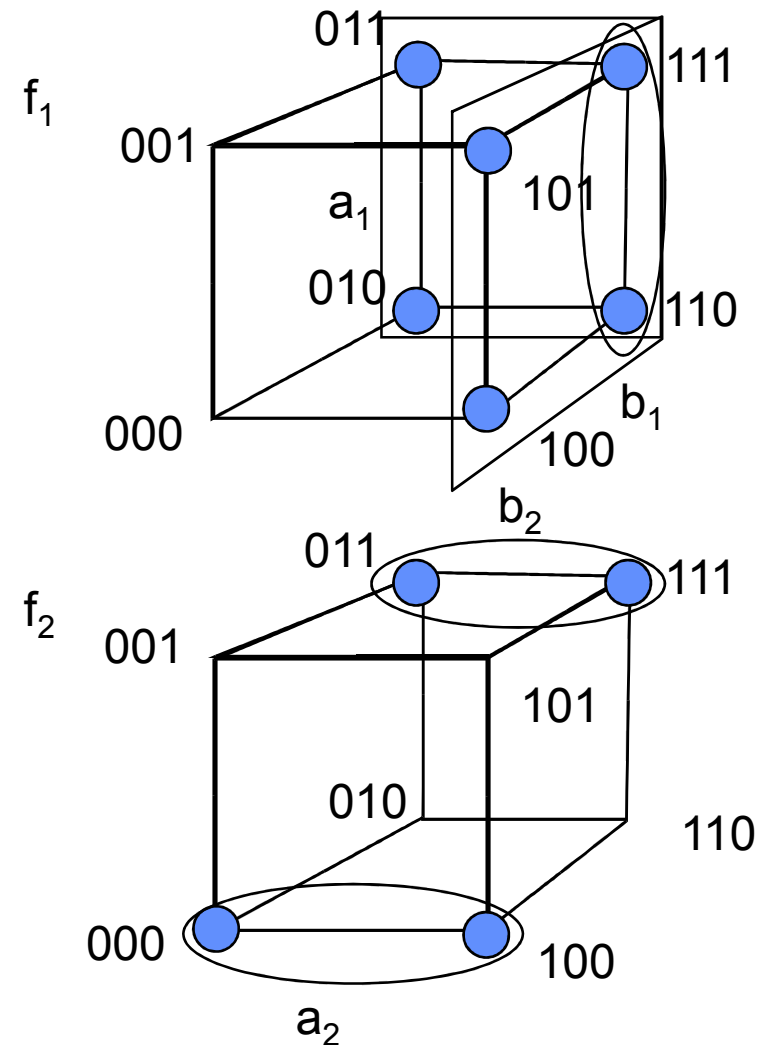
Der Durchschnittscube enthält also die Minterme (111) und (110)

$$a_2 = (2 \ 0 \ 0 \ 4)$$

$$b_2 = (2 \ 1 \ 1 \ 4)$$

$$a_2 \hat{\curvearrowright} b_2 = \times^{\nearrow}$$

Der Durchschnittscube ist leer.



3.1.1 Definitionen

Die Vereinigung von zwei booleschen Funktionen $F_1 \uplus F_2$ mit ist dann definiert als $C(F_1) \uplus C(F_2)$. (ODER-Verknüpfung von zwei Funktionen)

Bem.: Die Oder-Verknüpfung von zwei Cubes stellt i. a. keinen Cube dar.

Der Durchschnitt von zwei booleschen Funktionen $F_1 \frown F_2$ ist definiert über die Überdeckung (UND-Verknüpfung von zwei Funktionen)

$$C(F_1 \frown F_2) = \{ a \frown b \mid c \supseteq C(F_1), b \supseteq C(F_2) \text{ und } a \frown b \neq \emptyset \}$$

Ein **Minterm** ist ein Cube der im Eingangsteil keine 2 enthält und im Ausgangsteil genau eine 4 enthält. Jeder allgemeine Cube kann als Menge von Mintermen angesehen werden.

Def.: Ein **Implikant** einer booleschen Funktion ist ein Cube c mit

$$c \frown F^{\text{OFF}} = \emptyset$$

Ein **Primimplikant** ist ein Implikant, der in keinem anderen Implikanten enthalten ist.

3.1 Der Quine-McCluskey-Algorithmus

Zusammenfassen von Termen (Finden von Primtermen)

Nr.	Terme 0.Ord.						Nr.	Terme 1.Ord.						Nr.	Terme 2. Ord.					
	a	b	c	d	u	v		a	b	c	d	u	v		a	b	c	d	u	v
2	0	0	1	0	x	-	2,3	0	0	1	-	x	-	3,7,11,15	-	-	1	1	-	x
3	0	0	1	1	x	x	2,6	0	-	1	0	x	-	6,7,14,15	-	1	1	-	-	x
4	0	1	0	0	x	x	3,7	0	-	1	1	-	x	9,11,13,15	1	-	-	1	x	-
6	0	1	1	0	x	x	3,11	-	0	1	1	x	x	10,11,14,15	1	-	1	-	-	x
7	0	1	1	1	-	x	4,6	0	1	-	1	x	x							
9	1	0	0	1	x	x	6,7	0	1	1	-	-	x							
10	1	0	0	0	-	x	6,14	-	1	1	0	-	x							
11	1	0	1	1	x	x	7,15	-	1	1	1	-	x							
13	1	1	0	1	x	-	9,11	1	0	-	1	x	x							
14	1	1	1	0	-	x	9,13	1	-	0	1	x	-							
15	1	1	1	1	x	x	10,11	1	0	1	-	-	x							
							10,14	1	-	1	0	-	x							
							11,15	1	-	1	1	x	x							
							13,15	1	1	-	1	x	-							
							14,15	1	1	1	-	-	x							

3.1 Minimisierung von Bündelfunktionen

Min- und Primterme

Minterme	a	b	c	d	u	v	
10,11,14,15	1	-	1	-	-	x	A
9,11,13,15	1	-	-	1	x	-	B
6,7,11,15	-	1	1	-	-	x	C
3,7,11,15	-	-	1	1	-	x	D
2,3	0	0	1	-	x	-	E
2,6	0	-	1	0	x	-	F
4,6	0	1	-	0	x	x	G
3,11	-	0	1	1	x	x	H
9,11	1	0	-	1	x	x	I
11,15	1	-	1	1	x	x	K

2.Quinesche Tabelle

	u								v								
	2	3	4	6	9	11	13	15	3	4	6	7	9	10	11	14	15
A														x	x	x	x
B					x	x	x	x									
C											x	x			x		x
D									x			x			x		x
E	x	x															
F	x			x													
G			x	x						x	x						
H		x				x			x						x		
I					x	x							x		x		
K						x		x							x		x

Überdeckungsfunktion

$$\ddot{u}_u = (E \vee F)(E \vee H)G(F \vee G)(B \vee I)(B \vee H \vee I \vee K)B(B \vee K)$$

$$\ddot{u}_v = (D \vee H)G(C \vee G)(C \vee D)IA(A \vee C \vee D \vee H \vee I \vee K)A(A \vee C \vee D \vee K)$$

3.1 Definitionen

Die **Distanz** $\delta(c,d)$ der Cubes c, d ist gegeben durch $\delta(c,d) = \delta(l(c),l(d)) + \delta(o(c),o(d))$
wobei

$$\delta(l(c),l(d)) = \begin{cases} 0 & \text{falls } c_i \cap d_i = 4 \text{ für ein } i > m \\ 1 & \text{sonst} \end{cases}$$

Def. : Konsenscube $e = c \odot d$

- Falls $\delta(c, d) = 0$

$$e = \begin{cases} c \cap d & \text{falls } \Delta(c, d) = 0 \\ \emptyset & \text{falls } \Delta(c, d) \geq 2 \end{cases}$$

- Falls $\delta(c, d) = 1$

Falls $\delta(l(c), l(d)) = 1$

Falls $\delta(o(c), o(d)) = 1$

$$e_i = \begin{cases} c_i \cap d_i & \text{falls } c_i \cap d_i \neq \emptyset \\ 2 & \text{sonst} \end{cases}$$

$$e_i = \begin{cases} c_i \cap d_i & \text{für } 1 \leq i \leq m \\ 4 & \text{falls } c_i \text{ oder } d_i = 4 \text{ für } m+1 \leq i \leq m+n \\ 3 & \text{sonst} \end{cases}$$

3.1 Definitionen

Beispiel für Konsenscubes

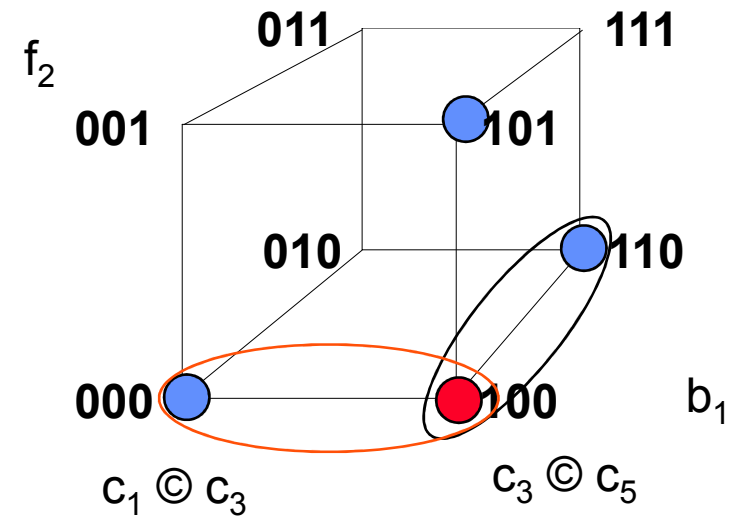
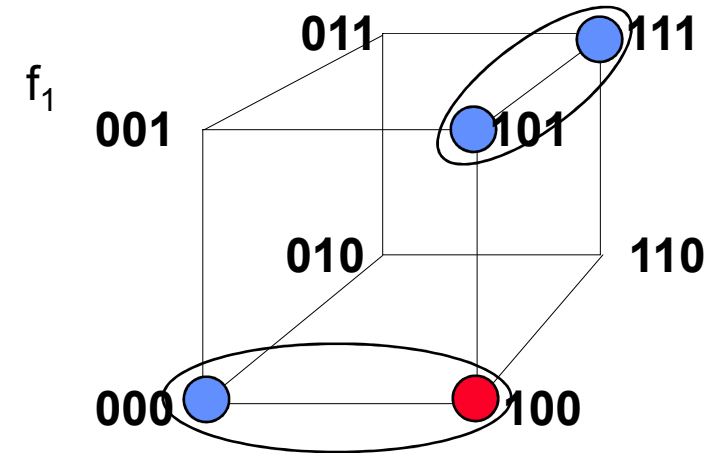
$$f_1 = \bar{a}\bar{b}\bar{c} \vee \bar{a}bc \vee \bar{a}c$$

$$f_2 = ac \vee \bar{b}\bar{c}$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 4 & 3 \\ 1 & 0 & 1 & 4 & 3 \\ 1 & 2 & 0 & 4 & 3 \\ 1 & 2 & 1 & 3 & 4 \\ 2 & 0 & 0 & 3 & 4 \end{bmatrix}$$

$$c_1 \odot c_3 = [200 \ 43]$$

$$c_3 \odot c_5 = [100 \ 44]$$



3.1 Der Quine-McCluskey-Algorithmus

Diskussion

Schritte bei Quine-McCluskey-Algorithmus

- Finden aller Primterme

Am Ende des Zusammenfassens erhält man eine Überdeckung aus Primtermen. Da jedoch alle Primterme gesucht werden kann es vorkommen, daß diese Überdeckung umfangreicher ist als die ursprüngliche Funktion

- Finden einer minimalen Überdeckung aus Primtermen

Verfahren: Es wird eine Überdeckungsfunktion aufgestellt. In dieser Funktion wird anschließend der größte Cube gesucht. Diese Suche vereinfacht sich, da es sich bei der Funktion um eine monotone Funktion handelt. Jedoch ist das Aufstellen dieser Funktion äußerst schwierig. Das Problem ist eigentlich das Finden einer minimalen Anzahl von Reihen, so daß jeder Spalteneintrag mindestens einmal enthalten (überdeckt) ist.

✱ Überdeckungsproblem (NP-vollständig)

3.2 Zweistufenlogikminimierung

Eine **vollständig spezifizierte** boolesche Funktion $F: B^n \rightarrow B^m$ $B = \{0, 1\}$ ordnet jedem Eingabewert einen Wert aus $B = \{0, 1\}$

Eine **unvollständig spezifizierte** boolesche Funktion läßt als Ausgabewerte Elemente aus $B^* = \{0, 1, \square\}$ zu. Hierbei bezeichnet \square einen unspezifizierten Eintrag.

Sei F eine unvollständig spezifizierte boolesche Funktion so können drei Mengen unterschieden werden

F^{ON} die Menge der Eingaben für die F eine 1 liefert (ON set).

F^{DC} die Menge der Eingaben für die F nicht spezifiziert ist (Don't care set).

F^{OFF} die Menge der Eingaben für die F eine 0 liefert (OFF set).

Für eine gegebene boolesche Funktion F sind diese drei booleschen Funktionen vollständig spezifiziert. Es gelten folgende Beziehungen

$$F^{ON} \cup F^{OFF} = \text{alle Eingaben} \quad F^{ON} \cap F^{DC} = \emptyset \quad F^{OFF} \cap F^{DC} = \emptyset \quad 1, \text{ d.h. } F^{OFF} = \overline{F^{ON} \cup F^{DC}} \text{ bzw. } F^{DC} = F^{ON} \cup F^{OFF}$$

3.2 Zweistufenlogikminimierung

Schritte bei der Zweistufenminimierung mit Espresso

- Berechnen von F^{OFF} aus $\overline{F^{\text{ON}}}$ $\xrightarrow{\text{AND}} F^{\text{DC}}$ oder F^{ON} aus $\overline{F^{\text{OFF}}}$ $\xrightarrow{\text{AND}} F^{\text{DC}}$
(Hierbei wird die Komplementierung benutzt)
- Aufspalten der Terme in F^{OFF} in einzelne Funktionen (nur eine 4 pro Term)
- Berechnen einer Überdeckung aus Primimplikanten
- Berechnen der essentiellen Primterme. Diese werden in F^{DC} gespeichert.
- Suche einer irredundanten Überdeckung

3.2 Zweistufenlogikminimierung

Expandieren von Cubes

Problem: Generierung aller Primterme ist sehr zeitaufwendig

Lösung: Die vorhandenen Cubes werden soweit expandiert (Ersetzung von 0/1 durch 2) bis sie sich gerade noch nicht mit dem OFF-Set schneiden.

Finden einer irredundanten Überdeckung aus Primtermen

Problem ist NP-hart

Lösung

1. Finden von Essentiellen Termen und Reduktion des Problems
2. Weitere Reduktion der Problemgröße
3. Heuristische Lösung des reduzierten Problems

3.2 Zweistufenlogikminimierung

Expandieren der Cubes

Überdeckungsmatrix C

F^{ON}	0 1 0 4 3	0 0 0 0 0	\ddot{u}_{ij}	falls $c_i = 1$ und $f_{ij} = 1$ oder $c_i = 0$ und $f_{ij} = 0$ ($i = 1..n$) falls $c_i = 3$ und $f_{ij} = 4$ ($i = n+1..m$) sonst
	1 0 1 4 4	1 1 1 0 1		
	0 1 0 3 4	0 0 0 0 1		
	0 1 1 4 3	0 0 1 0 0		
	1 1 0 4 4	1 0 0 0 1		
	0 0 1 3 4	0 1 1 0 1		

Blockungsmatrix B $L = \{ 1, 5 \}$ oder $\{ 2, 3 \}$

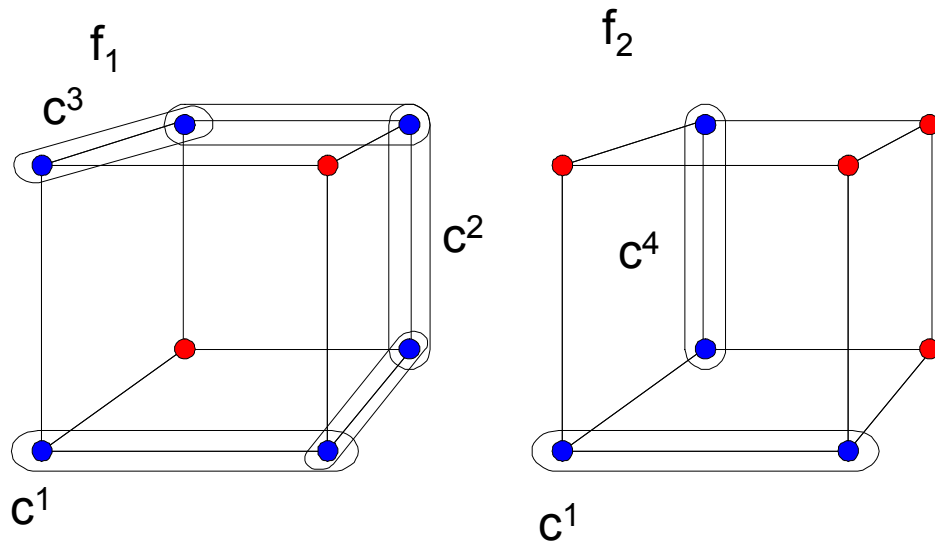
F^{OFF}	1 1 1 4 3	1 0 1 0 0	b_{ij}	falls $c_i = 1$ und $\bar{f}_{ij} = 0$ oder $c_i = 0$ und $\bar{f}_{ij} = 1$ ($i = 1..n$) falls $c_i = 3$ und $\bar{f}_{ij} = 4$ ($i = n+1..n+m$) sonst
	2 1 1 3 4	0 0 1 0 1		
	0 0 0 3 4	0 1 0 0 1		
	1 0 2 4 3	1 1 0 0 0		

Gesucht ist eine minimale Menge L von Spalten die alle Zeilen in B und so wenig wie möglich Zeilen in C überdeckt. Alle Variablen c_i mit $i \notin L$ werden nicht zu 2 verändert, $c^+ = (2 \ 1 \ 0 \ 4 \ 4)$

Ergebnis ist einer Überdeckung aus Primtermen (Dieses Verfahren ist reihenfolgeabhängig)

3.2.1 Expandieren von Cubes

Sonderfälle: Essentielle Spalten und der überexpandierte Cube c^*



$$F^{ON} = \begin{matrix} & 1 & 0 & 0 & 4 & 4 \\ \begin{matrix} 0 & 1 & 2 & 4 & 3 \\ 1 & 2 & 1 & 4 & 3 \\ 1 & 1 & 2 & 3 & 4 \end{matrix} \end{matrix}$$

$$F^{OFF} = \begin{matrix} & 0 & 0 & 1 & 4 & 3 \\ \begin{matrix} 1 & 1 & 0 & 4 & 3 \\ 2 & 0 & 1 & 3 & 4 \\ 0 & 1 & 2 & 3 & 4 \end{matrix} \end{matrix}$$

essentielle Spalten

$$B^1 = \begin{matrix} & 1 & 0 & 1 & 0 & 0 \\ \begin{matrix} 0 & \textcircled{1} & 0 & 0 & 0 \\ 0 & 0 & \textcircled{1} & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{matrix} \end{matrix}$$

$$c2' = (0 \ 1 \ 0 \ 4 \ 3)$$

$$B^{2'} = \begin{matrix} & 0 & 1 & 1 & 0 & 0 \\ \begin{matrix} \textcircled{1} & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & \textcircled{1} \\ 0 & 0 & 0 & 0 & \textcircled{1} \end{matrix} \end{matrix}$$

Auswahl

Besitzt eine Zeile nur eine 1, so muß die entsprechende Spalte in die Menge L^* aufgenommen werden (essentielle Spalte)
 der Cube c^* mit $c_i = 2$ bzw. 4 für alle $i \in L^*$ wird als überexpandierte Cube bezeichnet. Dieser ist i. a. kein Implikant von F ; nur in Sonderfällen gilt $c^* = c^+$

3.2.1 Expandieren der Cubes

Das Ergebnis von Expand ist eine Überdeckung P aus Primtermen.

Im Gegensatz zum Quine-McCluskey-Algorithmus werden aber nur solche Primterme generiert, die als nützlich für die Lösung ansehen werden.

Nun muß aus diesen Primtermen eine (möglichst) minimale Auswahl getroffen werden, so daß F^{ON} überdeckt wird. ($P \not\supseteq F^{OFF} = \emptyset$ ist sichergestellt)

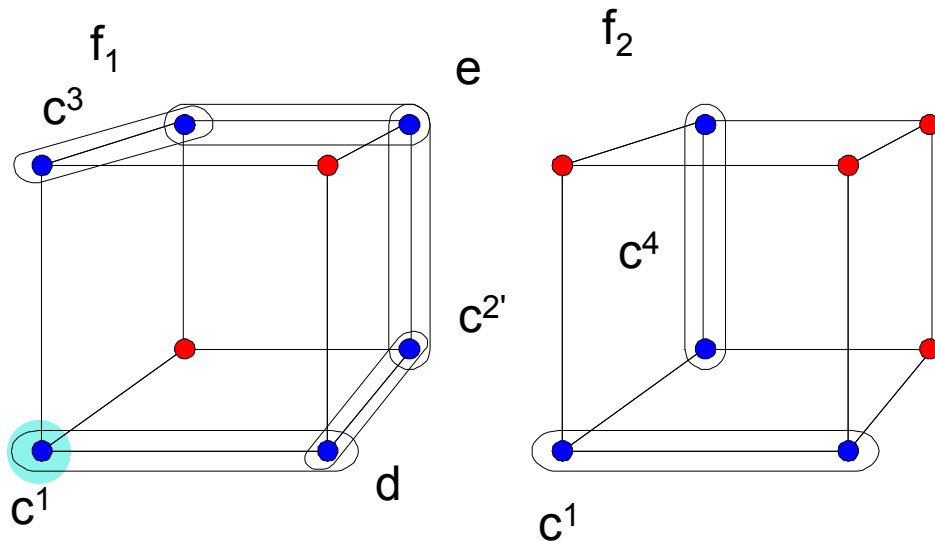
Zunächst soll die Problemgröße reduziert werden.

Essentielle Primterme sind Primterme, die einen Minterm enthalten, der von keinem anderen überdeckt wird.

Diese müssen in die Lösung aufgenommen werden.

3.2.2 Essentielle Cubes

Bestimmen der essentiellen Cubes (während Expand)



$$F^{ON} = \begin{matrix} & 1 & 0 & 0 & 4 & 4 \\ \begin{matrix} 1 \\ 0 \\ 1 \\ 1 \end{matrix} & 0 & 1 & 2 & 4 & 3 \\ & 1 & 2 & 1 & 4 & 3 \\ & 1 & 1 & 2 & 3 & 4 \end{matrix}$$

$$F^{OFF} = \begin{matrix} & 0 & 0 & 1 & 4 & 3 \\ \begin{matrix} 1 \\ 1 \\ 2 \end{matrix} & 1 & 1 & 0 & 4 & 3 \\ & 2 & 0 & 1 & 3 & 4 \\ & 0 & 1 & 2 & 3 & 4 \end{matrix}$$

$$c2' = (0 \ 1 \ 0 \ 4 \ 3)$$

$$B^{2'} = \begin{matrix} & 0 & 1 & 1 & 0 & 0 \\ \begin{matrix} 1 \\ 0 \\ 0 \\ 0 \end{matrix} & 0 & 0 & 0 & 0 & 0 \\ & 0 & 1 & 1 & 0 & 1 \\ & 0 & 0 & 0 & 0 & 1 \end{matrix}$$

$$B^1 = \begin{matrix} & 1 & 0 & 1 & 0 & 0 \\ \begin{matrix} 0 \\ 0 \\ 0 \\ 1 \end{matrix} & 1 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 1 & 0 & 0 \\ & 1 & 1 & 0 & 0 & 0 \end{matrix}$$

Beim Minterm $c^{2'}$ besteht die Auswahl zwischen Spalte 2 und Spalte 3.. Daher gibt es weitere Primterme die den Minterm $c^{2'}$ enthalten. Damit kann der expandierte Cube kein essentieller Cube sein.

Bei c^1 besteht keine Auswahlmöglichkeit. Daher gibt es keinen weiteren Primterm der c^1 enthält.

Für essentielle Cubes gilt

$$c^* = c^+$$

3.2.3 Irredundante Überdeckung

Bestimmen einer möglichst minimalen irredundanten Überdeckung

Das Ergebnis von Expand ist eine Überdeckung F^{ON} aus Primtermen.

Diese kann in folgende Teilmengen unterteilt werden $F^{ON} = E \uplus R_e \uplus R_p \uplus R_t$

- E ist die Menge der essentiellen Terme. Diese werden aus F^{ON} entfernt und dem Don't Care Set F^{DC} zugeschlagen. $F^{DC} := F^{DC} \uplus E$; $F^{ON} := F^{ON} - E$
- R_e ist eine Menge von Cubes c^k (bezogen auf F^{ON}) für die gilt, daß $F^{ON} - \{c^k\}$ keine Überdeckung der Funktion mehr darstellt. Diese müssen also in der Lösung vorkommen und werden daher nicht weiter betrachtet.
- R_t ist die Menge der vollkommen redundanten Terme.
 $\nexists c^k \cap R_t : c^k \not\subset F^{DC} \uplus E \uplus R_e$ Diese werden aus der Menge F^{ON} entfernt.

Die Auswahl Beschränkt sich demnach nur noch auf die Terme der Menge R_p

Anstatt die Überdeckungsfunktion \bar{u} zu berechnen, wird hier die einfacher zu berechnende Funktion \bar{u} bestimmt.

$\bar{u} = f(p_1, p_2, \dots, p_r)$ mit $r = |R_p|$ und $\bar{u}(\underline{p}) = 1$ falls $F^p = E \uplus R_e \uplus \{c^k \cap R_t \mid p_k = 1\}$ eine Überdeckung der Funktion ist

3.2.3 Irredundante Überdeckung

Konstruktion der Funktion \bar{u}

für jedes $c \in R_p$ bestimme alle Mengen $N \in N(c)$ mit $c \not\in (R_p - N) \Rightarrow F^{DC} \Rightarrow E \Rightarrow R_e$

Definiere den Cube $n(N)$ durch

$$n_i(N) = \begin{cases} 0 & c^i \in N \\ 2 & c^i \notin N \end{cases}$$

$$\bar{u} = \bigvee_{c^k \in R_p} \bigvee_{N \in N(c^k)} n(N)$$

Begr.: Jeder Minterm in \bar{u} stellt keine Überdeckung von F dar. Da $R_p \Rightarrow F^{DC} \Rightarrow E \Rightarrow R_e$ eine Überdeckung darstellt muß ein Cube in R_p nicht übedeckt werden ($c \not\in F^{DC} \Rightarrow E \Rightarrow R_e$ wurden entfernt). Daher ist dieser Minterm in einem $n(N)$ enthalten.

Nach Konstruktion ist jeder Minterm der in einem Cube $n(N)$ enthalten ist keine Überdeckung von F daher ist $\bar{u} = 1$

3.2.3 Irredundante Überdeckung

Aus \bar{u} kann nun die Blockungsmatrix $B = \{ b_{ij} \}$ von u konstruiert werden.

$$b_{ij} = \begin{cases} 1 & \bar{u} = 0 \\ 0 & \bar{u} = 2 \end{cases}$$

Aus der Blockungsmatrix B kann der größte Primterm von u als minimale Menge L von Spalten bestimmt werden, so daß jede Zeile mit einer 1 überdeckt wird, (Zeilenüberdeckung vgl. Expand) berechnet werden.

Allerdings sind Überdeckungsprobleme NP-hart

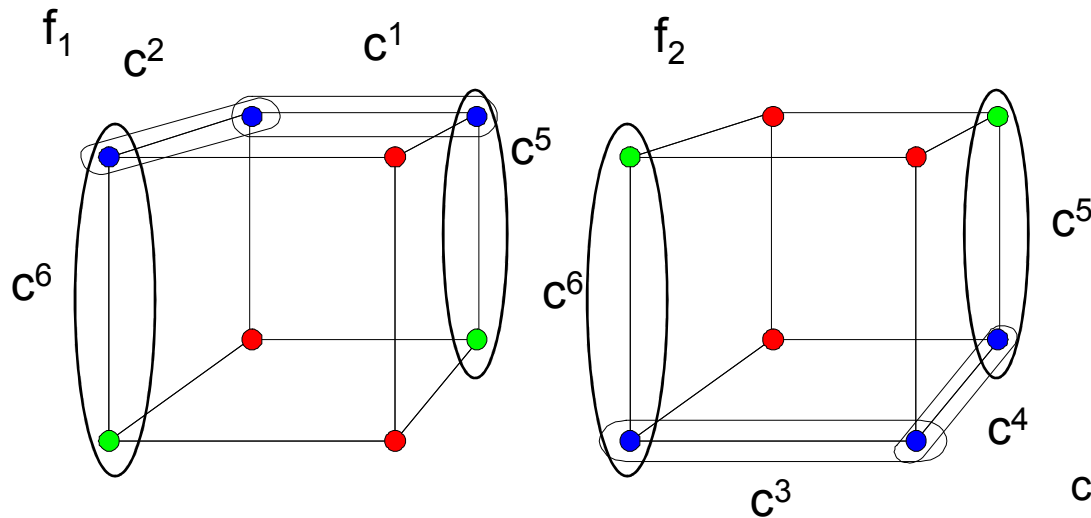
Berechnung einer möglichst minimalen irredundanten Überdeckung

Bestimme

BB^T eine 1 an der Stelle i,j bedeutet, daß die Zeilen i und j mit der gleichen Spalte überdeckt werden können.

3.2.3 Irredundante Überdeckung

Beispiel



Nach Expand

	2	1	1	4	3
	1	2	1	4	3
R_p	2	0	0	3	4
	0	2	0	3	4
	0	1	2	4	4
	1	0	2	4	4

	2	1	1	4	3
F^{ON}	1	2	1	4	3
	2	0	0	3	4
	0	2	0	3	4
	0	0	1	4	3
F^{OFF}	1	1	0	4	3
	0	0	1	3	4
	1	1	0	3	4

	c^1	c^2	c^3	c^4	c^5	c^6	
$B =$	1	0	0	0	1	0	n_1
	1	1	0	0	0	0	n_2
	0	0	1	1	0	0	n_3
	0	0	0	1	1	0	n_4
	0	0	1	0	0	1	n_5
	0	0	1	0	0	1	n_6
	0	1	0	0	0	1	n_7

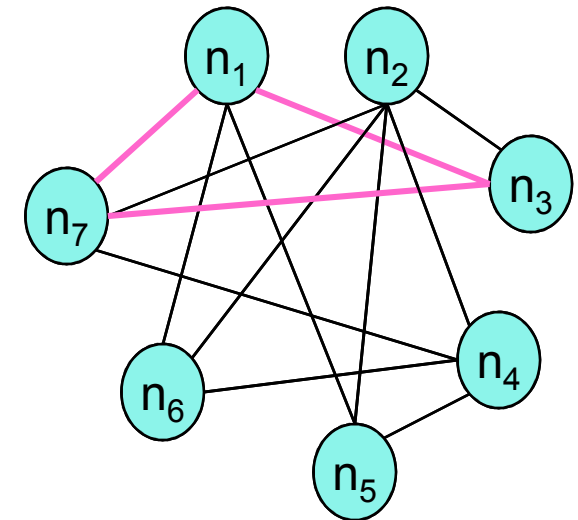
3.2.3 Irredundante Überdeckung

Beispiel

	c^1	c^2	c^3	c^4	c^5	c^6	
$B =$	1	0	0	0	1	0	n_1
	1	1	0	0	0	0	n_2
	0	0	1	1	0	0	n_3
	0	0	0	1	1	0	n_4
	0	0	1	0	0	1	n_5
	0	0	1	0	0	1	n_6
	0	1	0	0	0	1	n_7

Problem: Auswahl der Spalten
 weitere Heuristiken notwendig
 Hier minimale Auswahl
 (c_1, c_4, c_6)

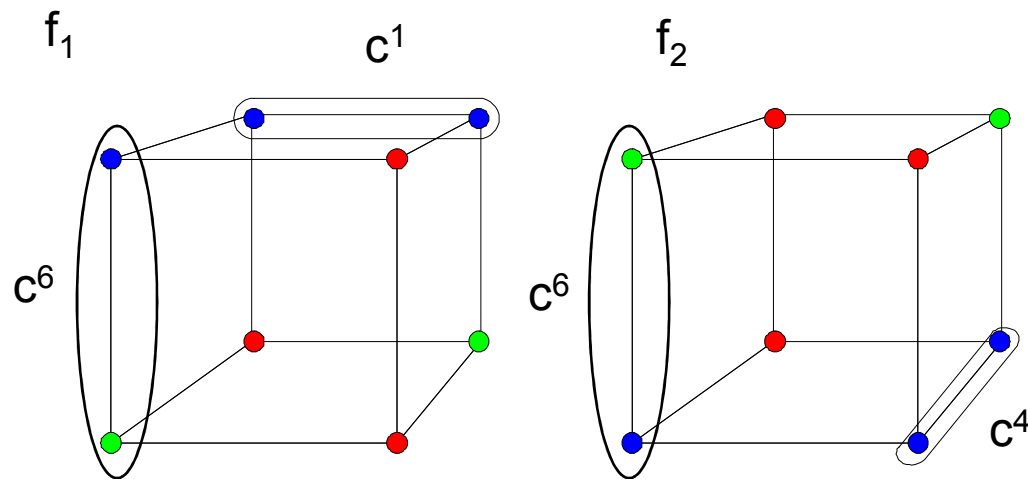
$$\begin{aligned}
 BB^T &= \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \\
 \overline{BB^T} &= \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}
 \end{aligned}$$



Clique { 1, 3, 7 }

3.2.3 Irredundante Überdeckung

Beispiel



Ergebnis

	2	1	1	4	3
F^{ON}	0	2	0	3	4
	1	0	2	4	4

	2	1	1	4	3
F^{ON}	1	2	1	4	3
	2	0	0	3	4
	0	2	0	3	4
	0	0	1	4	3
F^{OFF}	1	1	0	4	3
	0	0	1	3	4
	1	1	0	3	4

3.2.4 Ergänzende Ausführungen

Def.: Kofaktor

Sei $C = (c^1, c^2, \dots, c^r)$ eine Menge von Cubes und $p = (p_1 \dots p_n)$ ein Cube. Dann berechnet sich der Kofaktor $(C)_p$ durch Berechnung des Kofaktors jedes Cubes c^i bezüglich eines Cubes p elementweise zu

$$(c_k)_p^i = \begin{cases} \emptyset & \text{falls } c^i \cap p = \emptyset \\ 2 & \text{falls } p_k = 0 \text{ oder } 1 \\ 4 & \text{falls } p_k = 3 \\ (c_k)^i & \text{sonst} \end{cases}$$

Shannon-Expansion und Kofaktoren

eine Funktion f kann als Shannon-Expansion dargestellt werden durch $f = x_i g_{x_i} \vee \bar{x}_i g_{\bar{x}_i}$ mit $g_{x_i}, g_{\bar{x}_i}$ sind die jeweiligen Kofaktoren bezüglich x_i, \bar{x}_i .

3.2.4 Ergänzende Ausführungen

Eine Menge von Cubes $C = \{ c^k \mid k = 1..r \}$ überdeckt einen Cubes c , falls $(C)_c \neq 1$

Es stellt sich damit das Problem der Tautologieprüfung.

Tautologie-Prüfung

Gegeben eine boolesche Funktion F in Matrixdarstellung $F = C(F)$

Sonderfälle:

- F ist eine Tautologie, falls es eine Zeile aus 2 bzw. 4 gibt
- F ist **keine** Tautologie, falls es eine Spalte aus 0,1 oder 3 gibt
- F ist **keine** Tautologie, falls die Anzahl der Minterme kleiner als $m \leq 2^n$ ist.
- Eine Tautologie kann durch Auswertung der Wahrheitstabelle festgestellt werden. (Machbar für kleine Anzahlen von Variablen)

Zerlege $C(F)$ in eine Darstellung der Form

3.2.4 Ergänzende Ausführungen

Rekursion: Zerlege Matrix in sechs Untermatrizen

(durch tauschen von Spalten und Zeilen)

$Q(F) = \begin{pmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \end{pmatrix}$
 mit C_{11} ist keine Tautologie (Spalte aus 1 oder 0)
 C_{21} besteht nur aus Zweien
 C_{13}, C_{23} sind die Matrizen der zugehörigen Ausgangsteile

Auswahl der Matrix C_{11} :

- Enthalten Spalten nur 1(0) und 2, so ist die zugehörige Funktion nur dann eine Tautologie, falls es eine Zeile aus Zweien gibt.
- Zerlege F

3.2.4 Ergänzende Ausführungen

Der einfache Algorithmus liefert immer nur ein lokales Minimum

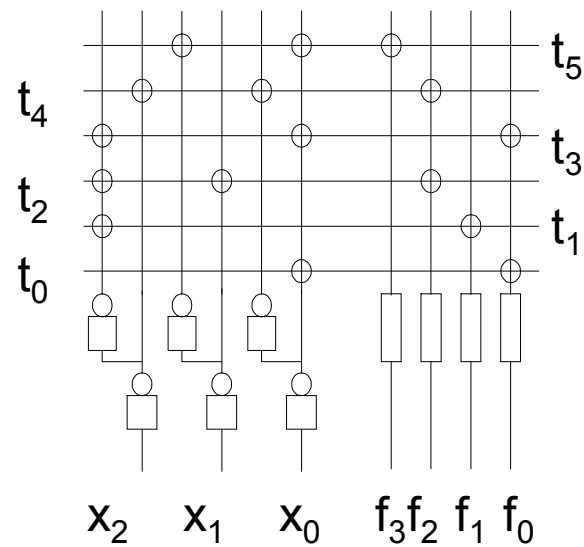
Modifikationen

- Für bessere Resultate werden die Eingabecubes für jede Funktion aufgespalten
- Nach einem Minimierungsschritt werden die Ergebniscubes reduziert, d.h. $2 \rightarrow 1/0$, $4 \rightarrow 3$, wobei allerdings die Überdeckungseigenschaft erhalten bleiben muß. Anschließend wird wieder minimiert, solange bis sich keine Reduktion mehr einstellt
- In einem letzten Schritt wird die Expansion nicht mehr Reihenfolgeabhängig durchgeführt
- Schließlich wird versucht, in den Ergebniscubes die 4 zu einer 3 zu reduzieren (das führt zu besserem elektrischen Verhalten und unterstützt die \nearrow Faltung von PLAs)

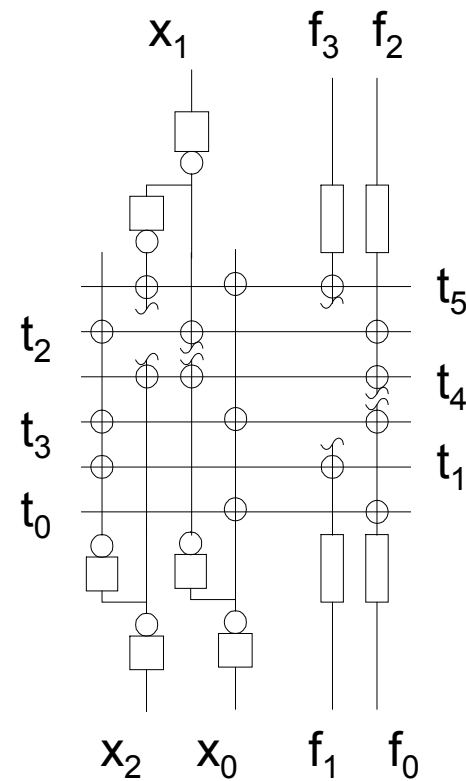
3.2.5 Optimierung der PLA-Struktur

Faltung von PLAs

Standardarchitektur



Gefaltete Struktur



3.2.5 Optimierung der PLA-Struktur

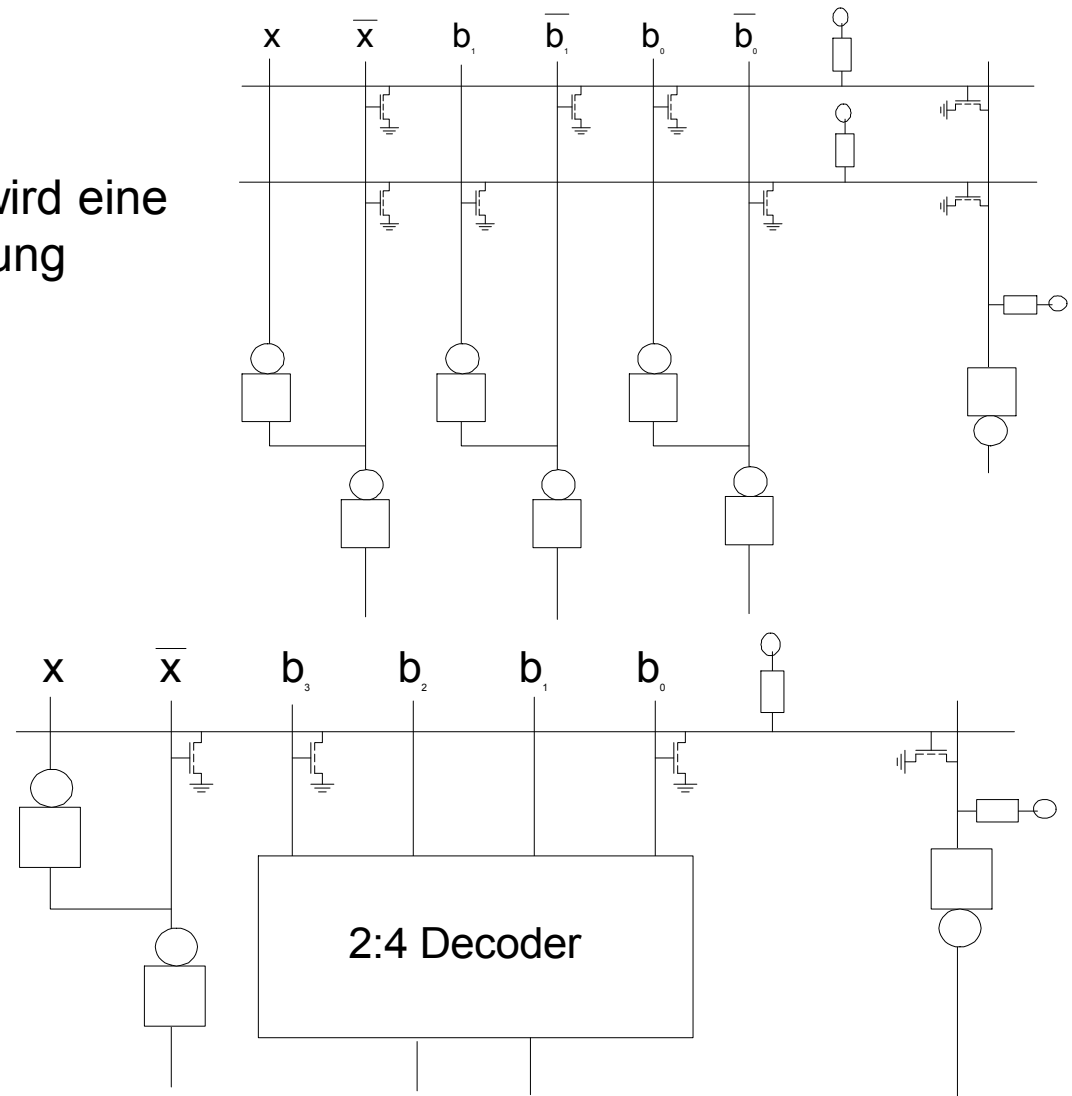
Einfügen von Decodern (Bsp.)

x	b ₁	b ₀	f
1	1	0	1
1	0	1	1
1	0	0	0
1	1	1	0

In der PLA-Struktur wird eine NOR-NOR-Verknüpfung realisiert.

x	11	10	01	00	f
1	0	1	0	0	1
1	0	0	1	0	1
1	0	1	1	0	1
1	0	0	0	1	0
1	1	0	0	0	0

✱ Symbolische Minimierung



3.3 Symbolische Minimierung

Beispiel

INDEX	AND	CNTA
INDEX	OR	CNTA
INDEX	JMP	CNTA
INDEX	ADD	CNTA
DIR	AND	CNTB
DIR	OR	CNTB
IND	AND	CNTB
DIR	JMP	CNTC
DIR	ADD	CNTC
IND	ADD	CNTC
IND	OR	CNTD
IND	JMP	CNTD

Durch Zusammenfassen erhält man folgenden Tabelle. Bitte beachten Sie, daß die zweite und dritte Zeile nicht zusammengefaßt werden darf, da sonst bei Eingabe von IND OR unklar ist ob CNTB oder CNTD ausgegeben werden soll. (**Eingangscodierbedingungen**)

INDEX	AND OR JMP ADD	CNTA
DIR	AND OR	CNTB
IND	AND	CNTB
DIR	JMP	CNTC
DIR IND	ADD	CNTC
IND	OR JMP	CNTD

3.3 Symbolische Minimierung

Bei Codierung der Symbole durch

INDEX = 00	AND = 00	CNTA = 11
DIR = 01	OR = 01	CNTB = 01
IND = 11	ADD = 10	CNTC = 10
	JMP = 11	CNTD = 00

erhalten wir

00	--	11
01	0-	01
11	00	01
-1	10	10
01	11	10
11	-1	00

Zeile kann gestrichen werden

Die minimierte boolesche Funktion ist optimal.

Bei Codierung der Symbole durch

INDEX = 00	AND = 00	CNTA = 00
DIR = 01	OR = 01	CNTB = 01
IND = 11	ADD = 10	CNTC = 10
	JMP = 11	CNTD = 11

erhalten wir jedoch

00	--	00
01	0-	01
11	00	01
-1	10	10
01	11	10
11	-1	11

00	--	00
-1	0-	01
-1	1-	10
11	-1	11

Hier kann die linke Funktionstabelle noch minimiert werden.

3.3 Symbolische Minimierung

Die endgültige boolesche Funktionstabelle entspricht der zusammengefaßten, symbolischen Tabelle

INDEX	AND OR JMP ADD	CNTA
DIR IND	AND OR	CNTB
DIR IND	ADD JMP	CNTC
IND	OR JMP	CNTD

Hierbei wird davon ausgegangen, daß bei Eingabe von IND OR die Ausgabe CNTB von der Ausgabe CNTD überschrieben wird (Analoges gilt für IND JMP).

Dies ist nur möglich, falls der Code von CNTD an jeder Stelle an der der Code von CNTB eine Eins hat ebenfalls eine Eins besitzt. D. h. der Ausgangscode von CNTD **überdeckt** den Code von CNTB.

Das bedeutet bei einer symbolischen Minimierung erhält man eine Halbordnung auf den Ausgangssymbolen (**Ausgangscodierbedingungen**). Die Codierung der Ausgangssymbole muß also bezüglich der Überdeckungsrelation der Codes verträglich mit dieser Halbordnung sein.

3.3 Symbolische Minimierung

3.3.1 Definitionen

Def.: **Symbolische Variable**

Eine symbolische Variable s ist ein Element aus einer endliche Menge S von Symbolen.

$$S^+ = S \uparrow \{ \bowtie, \boxtimes \}, \quad S^* = S \uparrow \{ \boxtimes \}$$

Bsp.: $S^+ = \{ \bowtie, \text{DIR}, \text{IND}, \text{INDEX}, \boxtimes \}$, s kann also die Werte DIR, IND, und INDEX annehmen. Die Menge S bildet hierbei einen Verband wobei \bowtie das Bottom-Element und \boxtimes das Top-Element (Don't care) bezeichnet. Auf S kann eine beliebige Halbordnung $>$ definiert sein. Hierbei sei $>_S$ die transitive Hülle von $>$.

3.3 Symbolische Minimierung

Operationen auf symbolischen Variablen

- Produkt(UND) $s \blacktriangle s' = \text{größtes } s'' \text{ mit } s >_s s'' \text{ und } s' >_s s''$ (Minimum)
- Summe(ODER) $s \blacktriangledown s' = \text{kleinstes } s'' \text{ mit } s'' >_s s \text{ und } s'' >_s s'$ (Maximum)
- Die Summe von zwei n-Tupeln ist gegeben durch eine elementweise Summenbildung

Def.: Symbolische Funktion

Eine vollständig spezifizierte symbolische Funktion ist eine Abbildung

$$f: S_i^0 \times S_i^1 \dots S_i^n \rightarrow S_o^0 \times S_o^1 \dots S_o^m$$

Eine unvollständig spezifizierte symbolische Funktion läßt auch S^* als Wertebereich zu.

3.3 Symbolische Minimierung

Def.: Symbolisches Literal

Sei A eine Teilmenge von S und s eine symbolische Variable. Ein symbolisches Literal X^A ist eine Funktion $X^A : S \rightarrow \{TRUE, FALSE\}$ mit

$$X^A(s) = \begin{cases} TRUE & \text{falls } s \in A \\ FALSE & \text{sonst} \end{cases}$$

Def.: Unvollständig spezifiziertes symbolisches Literal

Seien A, B Teilmengen von S und $A \nsubseteq B = \nsubseteq$. Ein unvollständig spezifiziertes symbolisches Literal $X^{A[B]}$ ist eine Relation

$$X^{A[B]}(s) = \begin{cases} TRUE & \text{falls } s \in A \\ FALSE & \text{falls } s \notin A \cup B \end{cases}$$

Ein (unvollständig spezifiziertes) symbolisches Literal X^A , $A \not\subseteq S$ lässt sich durch einen Vektor $(x_i)_{i=1 \dots |S|}$ mit $x_i \in B$ (B^*) darstellen. Bsp.: $S = \{s_1, s_2, s_3, s_4\}$

$X^{\{s_1, s_2\}}$ hat die Darstellung (1100)

$X^{\{s_1, s_4\}[\{s_3\}]}$ hat die Darstellung (10⊠1)

3.3 Symbolische Minimierung

Def.: Ein **Symbolischer Produktterm** ist ein n -Tupel $X = (X^A_1, X^B_2, \dots, X^N_n)$, wobei die $X^L_i \in S^I_i$ symbolische Literale sind. Ein **symbolisches Produkt** $p(s^I, X, \Diamond)$ mit $\Diamond \in S^O$ und $s^I \in S^I$. Das Symbol \Diamond ist der Ausgangsteil. Das Resultat ist definiert durch:

$$p(s^I, X, \tau) = \begin{cases} \tau & \text{falls } X^L_i(s^I_i) = \text{TRUE}, \forall i = 1..n \\ \emptyset & \text{sonst} \end{cases}$$

Zwei Produkte p_1, p_2 **überschneiden sich** ($p_1 \bowtie p_2 \circlearrowright$), falls $\exists s^I \in S^I$ mit $p_1(s^I, X_1, \Diamond_1) \circlearrowleft$ und $p_2(s^I, X_2, \Diamond_2) \circlearrowleft$. Zwei Produkte sind **ausgangsseitig disjunkt** falls sie sich entweder nicht überschneiden oder sie den gleichen Ausgangsteil haben, d. h. aus $p_1 \bowtie p_2 \circlearrowright$ folgt $\Diamond_2 = \Diamond_1$. Eine Menge von Produkten ist ausgangsseitig disjunkt, falls die Produkte paarweise ausgangsseitig disjunkt sind.

3.3 Symbolische Minimierung

Ein symbolischer **Implikant** einer symbolischen Funktion ist ein symbolisches Produkt $p(s^I, X, \Diamond)$ wobei gilt $p(s^I, X, \Diamond) >_{s^O} f(s^I)$ ($>_{s^O}$ ist die transitive Hülle der Ordnungsrelation auf den Ausgangssymbolen)

Eine symbolische **Überdeckung** $C(P, >_{s^O})$ einer symbolischen Funktion f ist eine Menge $P = \{p_1, p_2, \dots, p_{|P|}\}$ von symbolischen Implikanten mit

$$\exists s^I : f(s^I) >_{s^O} p_1 \vee p_2 \vee \dots \vee p_{|P|}.$$

Die Kardinalität einer symbolischen Überdeckung ist $|P|$. Eine Überdeckung ist **minimal**, falls keine Teilmenge $P' \subsetneq P$ die Funktion überdeckt. Eine **Minimumsüberdeckung** ist eine Überdeckung mit minimaler Kardinalität.

Bem.: Die Kardinalität hängt von der Relation $>_{s^O}$ ab. Das Ziel bei einer symbolischen Minimierung ist also auch diese Relation so zu konstruieren, daß die Kardinalität der Überdeckung möglichst klein wird.

3.3 Symbolische Minimierung (nach DeMicheli)

Berechnung der Eingangscodierbedingungen

- Führe eine 1 aus n Codierung der Zustände durch.

Sei $S = \{ s_0, s_1, \dots, s_n \}$ die Menge der Symbole, so wird das Symbol s_i durch eine Bitfolge $000\dots 1_{(i)} \dots 0$ mit einer 1 an der i-ten Stelle dargestellt.

(sog. positionelle Cubes)

Für diese symbolischen Einträge gelten andere Regeln für das Zusammenfassen von Termen, so ergibt die Zusammenfassung von 1000, 0010 das symbolische Literal 1010 (dargestellt als Vektor).

- Minimiere Tabelle

Es wird eine zweistufige Minimierung durchgeführt.

- Die booleschen Variablen werden nach den üblichen Rechenregeln behandelt.
- Die Symbole werden "symbolisch" minimiert

3.3 Symbolische Minimierung (nach DeMicheli)

Erfüllen von Eingangscodierbedingungen.

n_b ist die Anzahl der Zustandsvariablen $q_0 \dots q_{n_b-1}$ (Anzahl der Codebits)

$n_s = |S|$ ist die Anzahl der symbolischen Werte (Anzahl der Zustände)

Eine Codierung ist gegeben durch die $n_s \times n_b$ Matrix

$$C = \begin{pmatrix} q_{00} & \cdots & q_{1(n_b-1)} \\ \vdots & \ddots & \vdots \\ q_{(n_b-1)0} & \cdots & q_{(n_s-1)(n_b-1)} \end{pmatrix}$$

Hierbei entspricht eine Zeile von C $q_{i0} \dots q_{i(n_b-1)}$ der Codierung des i -ten Symbols.

Eine Selektion von $x \in \{ \text{true}, 1, 0, \text{false} \}$ durch $a \in \{ \text{true}, 1, 0, \text{false} \}$ ist definiert als

$$a \bullet x = \begin{cases} x & \text{falls } a = 1 \\ \phi & \text{sonst} \end{cases}$$

3.3 Symbolische Minimierung (nach DeMicheli)

Die Selektion läßt sich auf Matrizen ausdehnen: $A \bullet X = C = \{c_{ij}\}^{p \times q}$ wobei gilt

$$c_{ij} = \bigvee_{k=0}^{p-1} a_{ik} \bullet x_{kj}$$

Die Matrix $F = EC \bullet C$, die sog. **Face-Matrix**, gibt in der i-ten Zeile die Codierung der i-ten Eingangscodierbedingung an.

Bsp.:

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \bullet \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} - & 0 & 0 \\ - & 1 & - \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & - & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

Boolescher Unterraum (Face) der 2.
 Codierbedingung. Diese umfaßt die Codierungen der Zustände s1, s2 und s7.

Code des vorletzten Zustandes

3.3 Symbolische Minimierung (nach DeMicheli)

Def.: Erfüllung von Eingangscodierbedingungen

Sei $\overline{EC} = \{\overline{ec}_{ij}\}$ die Matrix zu $EC = \{ec_{ij}\}$ mit $\overline{ec}_{ij} = 1$ falls $ec_{ij} = 0$ sonst ist $\overline{ec}_{ij} = 0$.

Die Matrix $\overline{F}^k = \{f^k_{ij}\}$ mit $f^k_{ij} = \overline{ec}_{ik} \bullet c_{kj}$. Eine Codematrix C erfüllt die Eingangscodierbedingungen EC, wenn für alle $k = 0 \dots (n_s-1)$ gilt: Die Matrix $\overline{F}^k \wedge F = \overline{x}$, d.h. jede Zeile beinhaltet mindestens einen Eintrag \overline{x} .

$$\begin{aligned}
 F = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \bullet \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{red}{1} \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} &= \begin{pmatrix} 1 & - & - \\ - & 1 & - \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ - & 1 & - \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \quad \overline{F}^4 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} \bullet (1 \ 1 \ 1) = \begin{pmatrix} \phi & \phi & \phi \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ \phi & \phi & \phi \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad F \wedge \overline{F}^4 = \begin{pmatrix} \phi & \phi & \phi \\ \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{red}{1} \\ \phi & \phi & 1 \\ 1 & \phi & 1 \\ 1 & \phi & \phi \\ \phi & 1 & 1 \\ \phi & \phi & \phi \\ \phi & \phi & 1 \\ 1 & \phi & 1 \end{pmatrix} \neq \Phi
 \end{aligned}$$

3.3 Symbolische Minimierung (nach DeMicheli)

Reduktion der Matrix EC

Jede Zeile in der Matrix EC stellt ein symbolisches Literal dar.

Eine Zeile in der Matrix EC heißt **Verbindung**, falls sie den Durchschnitt aus zwei Zeilen darstellt, sonst heißen die Zeilen **prim**.

Satz: Falls eine Codematrix C die Codierbedingungen in EC erfüllt, so erfüllt C auch die Codierbedingungen Matrix EC' mit

$$EC' = \begin{pmatrix} EC \\ ec' \end{pmatrix}$$

wobei ec' eine Verbindung der Zeilen in EC ist.

Es ist also zulässig, Verbindungen aus der Matrix EC zu streichen.

Zeilen mit nur einer Eins enthalten keine eigentlichen Codierbedingungen. Daher können sie ebenfalls aus EC gestrichen werden. (Bem.: Die Erfüllung der Codierbedingungen garantiert **nicht** einen eindeutigen Code. Dies könnte durch die Hinzunahme von Zeilen mit nur einer Eins pro Zustand erreicht werden. Hier soll ein eindeutiger Code allerdings durch eine Nachverarbeitung garantiert werden.)

3.3 Symbolische Minimierung (nach DeMicheli)

Spalten aus lauten Nullen können ebenfalls aus der Matrix EC gestrichen werden. Die Codes für die dadurch repräsentierten Zustände müssen aus den noch unbesetzten Bitkombinationen gewonnen werden.

Satz: Die Matrix $C = EC^T$ erfüllt die Codierbedingung EC.

$$F = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \bullet \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & - \\ 0 & 1 & - \\ - & - & 1 \end{pmatrix}$$

$$\bar{F}^{2,3} = \begin{pmatrix} 0 & 1 & 0 \\ \phi & \phi & \phi \\ 0 & 1 & 0 \end{pmatrix}$$

$$\bar{F}^4 = \begin{pmatrix} \phi & \phi & \phi \\ 1 & 0 & 1 \\ \phi & \phi & \phi \end{pmatrix}$$

$$\bar{F}^{5,6} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\bar{F}^7 = \begin{pmatrix} 0 & 1 & 1 \\ \phi & \phi & \phi \\ \phi & \phi & \phi \end{pmatrix}$$

$$\bar{F}^1 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \bullet (1 \ 0 \ 0) = \begin{pmatrix} \phi & \phi & \phi \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Bem.: Die Codes für die Zustände s1,s2 und s4, s5 sind identisch. Es müsste also noch eine weitere Codespalte angefügt werden, die eine Unterscheidung ermöglicht.

3.3 Symbolische Minimierung

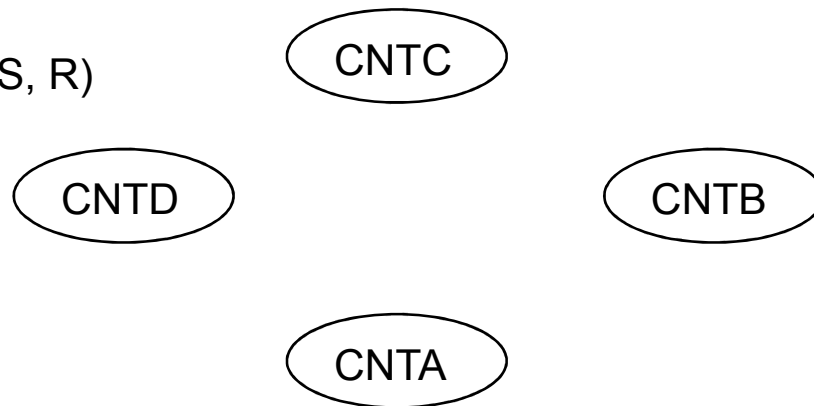
Bsp.:

Symbolische Überdeckung*

CNTA	100	1111	1000
CNTB	001	1000	0100
	010	1100	0100
CNTC	001	0001	0010
	010	0011	0010
CNTD	001	0110	0001

P

Graph $G = (S, R)$



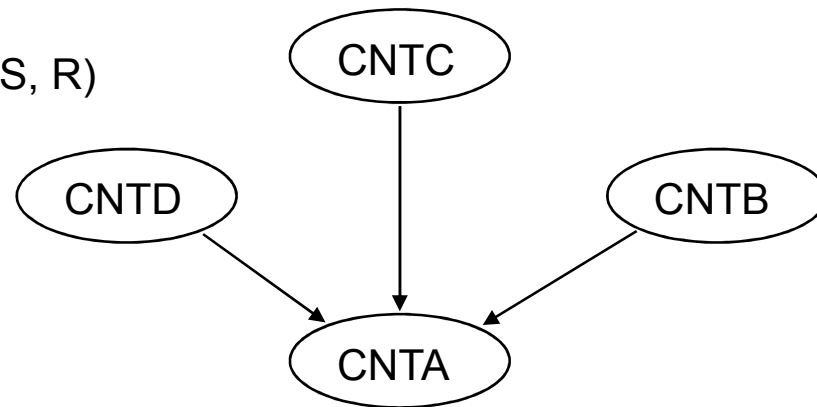
* Im Gegensatz zum Algorithmus ist diese bereits minimiert

3.3 Symbolische Minimierung

Bsp.:

ON	CNTA	100	1111	1000	P	CNTA	111	1111	1000
	CNTB	001	1000	0100					
DC		010	1100	0100					
	CNTC	001	0001	0010					
		010	0011	0010					
	CNTD	001	0110	0001					

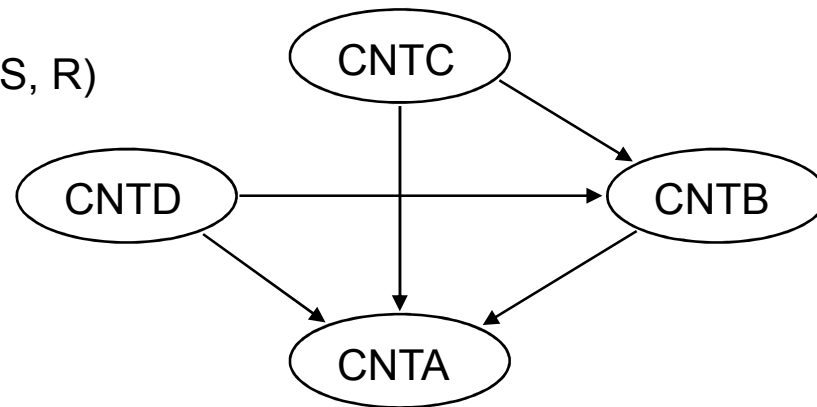
Graph $G = (S, R)$



3.3 Symbolische Minimierung

					P				
OFF	CNTA	100	1111	1000	→	CNTA	111	1111	1000
	CNTB	001	1000	0100		CNTB	011	1111	0100
ON		010	1100	0100					
	CNTC	001	0001	0010					
DC		010	0011	0010					
	CNTD	001	0110	0001					

Graph $G = (S, R)$



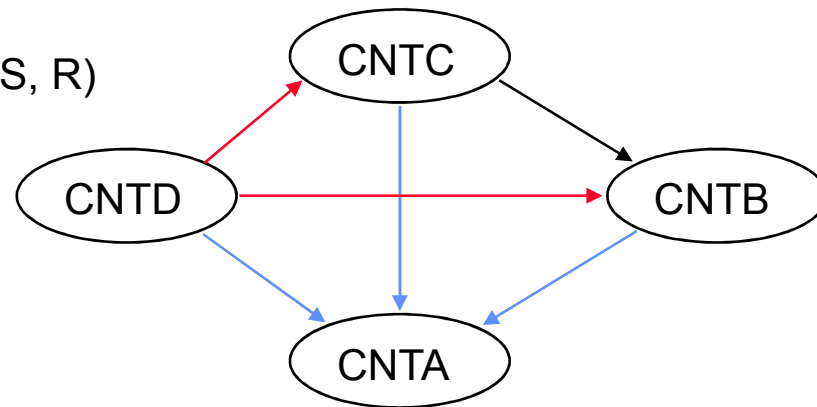
3.3 Symbolische Minimierung

P

OFF	CNTA	100	1111	1000
	CNTB	001	1000	0100
		010	1100	0100
ON	CNTC	001	0001	0010
		010	0011	0010
DC	CNTD	001	0110	0001

CNTA	111	1111	1000
CNTB	011	1111	0100
CNTC	011	0011	0010
CNTD	001	0110	0001

Graph $G = (S, R)$



Eine mit R konsistente Codierung

CNTA	000
CNTB	001
CNTC	011
CNTD	111

3.3 Symbolische Minimierung

Berechnung der Ausgangscodierbedingungen

Eingabe: ON_i , $i = 0..|S^O|-1$ // Die ON-Mengen der Ausgabesymbole

Algorithmus

Graph $G = (S, R)$ $V = S$, $R = \Rightarrow^+ \equiv$ // Die transitive Hülle der Ordnungsrelation

$P = \Rightarrow$; // Optimierte symbolische Überdeckung

for $k := 0$ to $|S^O|-1$ do

$i := \text{select}(k)$ // Auswahl einer der Überdeckung für $s_i \in S$

$OFF_i = \bigcup_{j \in J} ON_j$; $J = \{j \mid (v_i, v_j) \in R\}$;

$M_i = \text{minimize}(ON_i, OFF_i)$; // Minimierung mit Zweistufen-Minimierer

$R = R \setminus \{(v_j, v_i) \mid M_i \subseteq ON_j \subseteq \}$;

$P = P \setminus M_i$;

od;

Ausgabe: P und G

3.3 Symbolische Minimierung

Diskussion des Algorithmus

- Unnötiger Aufwand bei der Minimierung der ON-Menge für ein Ausgangssymbol.
- Ziel sollte sein, die ON-Menge durch eine große DC-Menge zu reduzieren, welche die meisten Produktterme besitzt.
 - ☞ Minimierung der einzelnen ON-Mengen getrennt.
- Es werden unnötige Kanten im Graph erzeugt.
 - ☞ Kontrolle der Überschneidungen der Mengen.
 - ☞ Verhinderung der Expansion der Literale über ein zur Minimierung notwendiges Maß.
- Expansion der Literale
 - Ein Literal aus lauter Einsen stellt ein *don't care* dar.
 - Bei weniger Einsen bleibt unklar, ob dies für die spätere Codierung nützlich oder hinderlich ist.
 - ☞ Dies kann durch *don't care* Einträge für die Codierung ausgenutzt werden.

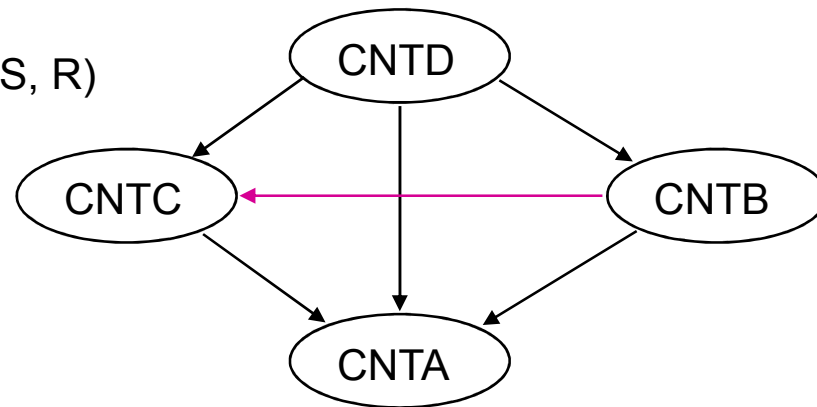
3.3 Symbolische Minimierung

Verbesserter Algorithmus

CNTA	100	1111	1000
CNTB	001	1000	0100
	010	1100	0100
CNTC	001	0001	0010
	010	0011	0010
CNTD	001	0110	0001

CNTA	111	1111	1000
CNTA	1**	1111	
CNTB	011	1111	0100
CNTB	011	1100	0100
CNTC	011	0011	0010
CNTD	001	0110	0001

Codierung

Unnötige
ExpansionGraph $G = (S, R)$ 

Funktion

CNTA	00
CNTB	01
CNTC	10
CNTD	11

--	--	00
-1	0-	01
-1	1-	10
11	-1	11

3.3 Symbolische Minimierung

Codierung der Symbole

Entwicklung eines iterativen Algorithmus

Spaltenweise Codierung

- In jedem Schritt wird eine neue Codespalte (q_i) generiert

Zeilenweise Codierung

- In jedem Schritt wird ein neuer Zustand $s_i \in S$ codiert.

Gemischte Vorgehensweise

- Es werden abwechselnd Spalten und Zeilen der Codiermatrix C hinzugefügt

Bem.: Zeilenweise Codierungsverfahren generieren meist kürzere Codelängen. Diese sind jedoch, wegen der Suche nach noch freien Codes, in der Rechenzeit aufwendiger.

3.4 Mehrstufen-Logikminimierung

Einsatzgebiete der Mehrstufen-Logikminimierung

- Synthese von mehrstufigen Implementierungen aus Logikfunktionen
- Umsetzung eines Entwurfs auf eine andere Technologie (target remapping)
- Optimierung eines Entwurfs für eine Technologie
- Optimierung des Flächen-Zeit-Produkts

Aufgaben der Mehrstufen-Logikminimierung

- Zerlegung einer zweistufigen Funktion in Teilfunktionen (Dekomposition)
- Minimierung einer der Funktion(en)
 - nach dem Flächenverbrauch
 - nach der Verarbeitungszeit
 - ...
- Abbildung auf eine Zielbibliothek (Technology mapping)

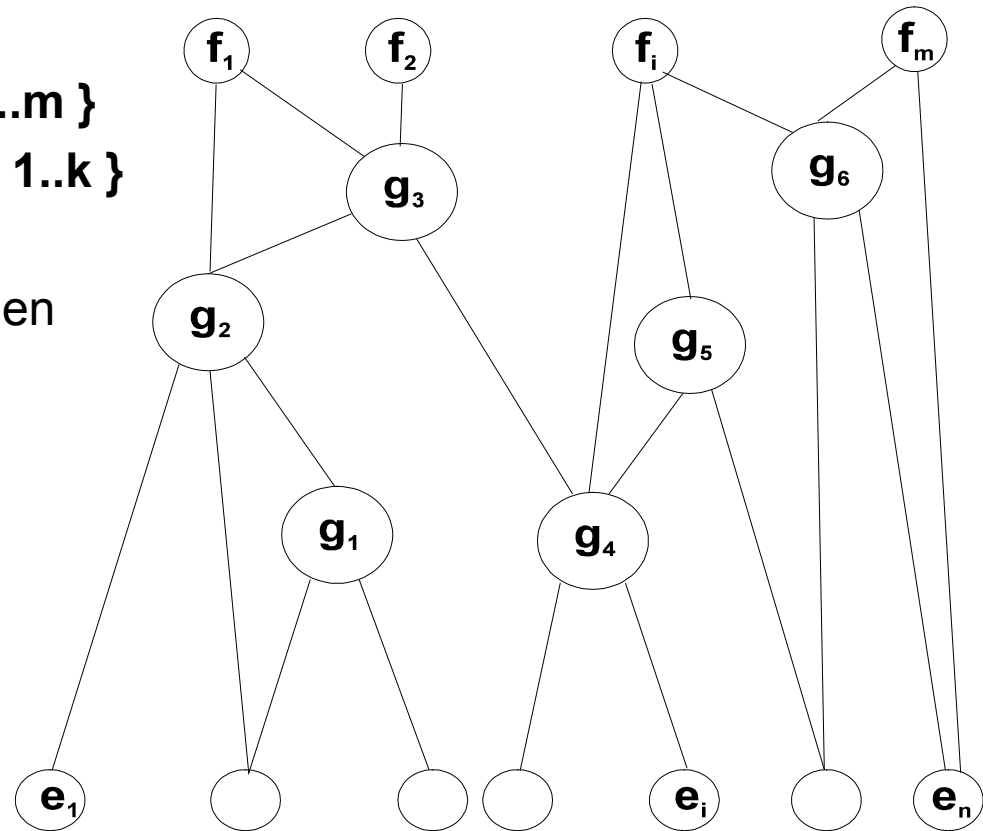
3.4 Mehrstufen-Logikminimierung

Grunddatenstruktur

- **Primäre Eingänge** $PI = \{e_i \mid i = 1..n\}$
- **Menge von Funktionen** $PO = \{f_i \mid i = 1..m\}$
- **Menge von Teilfunktionen** $N = \{g_i \mid i = 1..k\}$
 - Jede Teilfunktion g_i ist zweistufig
 - Üblicherweise besitzt jedes g_i nur einen Ausgang.
- **Der Graph** $(PI \rightrightarrows N \rightrightarrows PO, E)$ ist ein **gerichteter, azyklischer Graph (DAG)**
 - PO sind die Senken
 - PI sind die Quellen

Optimierungskriterien

- Anzahl der Literale
- Anzahl der Variablen pro g_i
- Stufenzahl



3.4 Mehrstufen-Logikminimierung

Es soll hier die Umwandlung einer zweistufigen Beschreibung in ein mehrstufiges untersucht werden

Aufgaben

- Faktorisierung
Die einzelnen Funktionen in disjunktiver Form werden in ihre Teilfunktionen zerlegt (Klammerausdrücke).
- Dekomposition
gemeinsame Teilfunktionen werden als getrennte Funktionen realisiert.

3.4 Mehrstufen-Logikminimierung

Beispiel

$$F = ace \vee acd \vee bce \vee bcd$$

Faktorisieren

$$F = c (a (d \vee e) \vee b (d \vee e))$$

$$F = c ((a \vee b) (d \vee e))$$

Dekomposition

$$f_1 = abc$$

$$f_2 = abd$$

$$f_3 = abc \vee abd = ab (c \vee d)$$

$$g_1 = ab$$

$$g_2 = (c \vee d)$$

$$f_1 = g_1 c$$

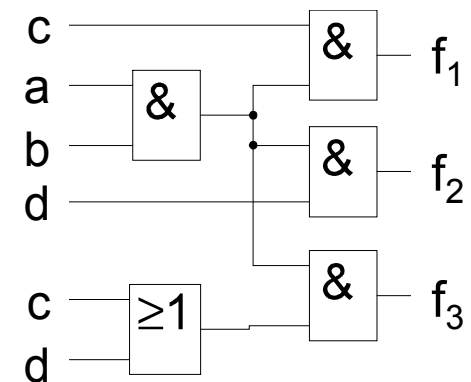
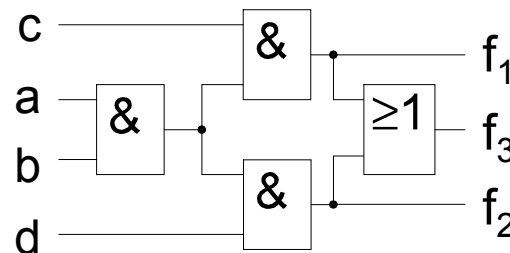
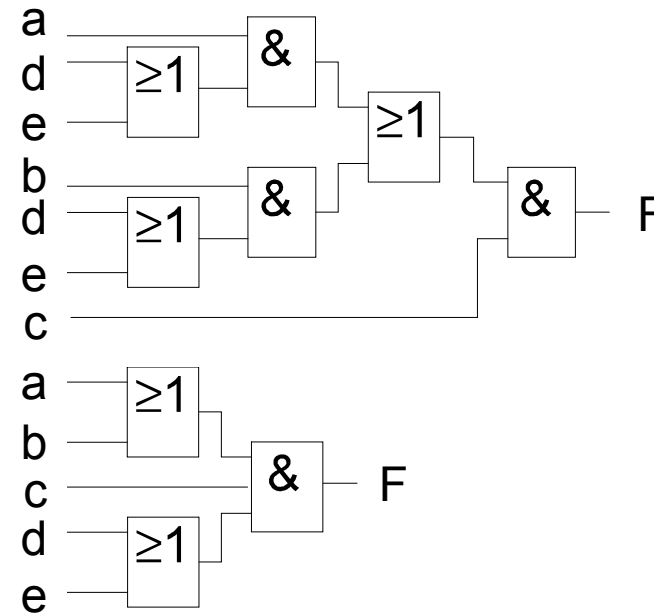
$$f_1 = g_1 c$$

$$f_2 = g_1 d$$

$$f_2 = g_1 d$$

$$f_3 = f_1 \vee f_2$$

$$f_3 = g_1 g_2$$



3.4.1 Das MIS-System

MIS (Multilevel Interactive Synthesis System)

- Algorithmisches System
- Interaktiv
- Gewisser Automatisierungsgrad durch Scripts

3.4 Mehrstufen-Logikminimierung

Algebraische und boolesche Darstellung von Logikfunktionen

Ein **algebraischer Ausdruck** ist eine disjunktive Form in der kein Term in einem anderen enthalten ist (z. B. irredundante Überdeckung aus Primtermen). Bei **algebraischen Operationen** werden Idempotenzgesetz, Absorptionsgesetz und DeMorgan-Regeln nicht berücksichtigt, d. h. eine UND-Verknüpfung von zwei Logikfunktionen wird wie eine Multiplikation von Zahlen behandelt.

Bsp.:

$$y = (a \vee b) \wedge (c \vee d) = ac \vee ad \vee bc \vee bd$$

aber

$$y = (a \vee b) \wedge (\bar{a} \vee b) = a\bar{a} \vee ab \vee b\bar{a} \vee bb = b$$

Algebraische Operationen sind weniger rechenzeitintensiv, ergeben aber meist schlechtere Resultate.

3.4 Faktorisierung

Division

Def.: eine Logikfunktion g ist ein **boolescher Divisor** von f , falls

$$f = gh \vee r$$

wobei h und r Logikfunktionen sind und h nicht Null ist (d. h. $r = f$).

h ist der, Quotient r ist der Rest. (Hierbei sind r und h nicht eindeutig bestimmt)

Eine Logikfunktion g ist ein **boolescher Faktor** falls gilt

$$f = gh$$

Boolesche und algebraische Division

Ein boolescher Ausdruck $f = a \vee bc$ läßt sich schreiben als $f = (a \vee b)(a \vee c)$ da

$$f = aa \vee ac \vee ab \vee bc = a \vee bc \quad (aa = a \text{ (Idempotenz)} \quad ac, ab \vee a = a \text{ (Absorbtion)})$$

andererseits läßt sich $f = ac \vee ad \vee bc \vee bd$ nur durch algebraische Umformungen darstellen als $f = (a \vee b)(c \vee d)$

3.4 Faktorisierung

Algebraische Division (Weak Division)

Gegeben einwertige boolesche Funktion f und g in Cube-Darstellung

$$f = \{ c^i \mid i = 1..n \}, g = \{ d^i \mid i = 1..m \bullet n \}$$

Gesucht: Funktionen h und r , so daß $f = gh \vee r$, wobei die Anzahl der Cubes in r minimal ist (Hierdurch sind h und r eindeutig bestimmt)

Algorithmus:

$U = \{ u^i \mid i = 1..n \}$ die Cubes in f reduziert auf die Literale in g

$V = \{ v^i \mid i = 1..n \}$ die Cubes in f reduziert auf die Literale nicht in g

$$V^k = \{ v^j \mid \exists s : u^i = d^s \bullet 1 \bullet s \bullet n \}$$

$$h = \bigcap V^k$$

$$r = f - gh$$

3.4 Faktorisierung

Beispiel und Implementierungshinweise

$$f = ace \vee ade \vee bc \vee bd \vee be \vee a'b \vee ab$$

$$g = ae \vee b$$

Codiere die Cubes in g, U und V durch ganze Zahlen ($ae = 5$, $b = 2$)

Für jeden Cube (Zahl) in g markiere die Zeile in der Hashtabelle mit x

Durchsuche die Codespalte von U

und trage für den zugehörigen Cube (Zahl) in V eine 1 ein.

Verunde die Zeilen in der Hashtabelle (nicht markierte Einträge werden zu 0 angenommen)

Cube	a	\bar{a}	b	c	d	e	U	V	Code in U	Code in V
ace	1	0	0	1	0	1	ae	c	5	2
ade	1	0	0	0	1	1	ae	d	5	1
bc	0	0	1	1	0	0	b	c	2	2
bd	0	0	1	0	1	0	b	d	2	1
be	0	0	1	0	0	1	be	-	3	0
$\bar{a}b$	0	1	1	0	0	0	b	\bar{a}	2	4
ab	1	0	1	0	0	0	ab	-	6	0

$$h = c \vee d \quad hg = ace \vee ade \vee bc \vee bd$$

$$r = be \vee a'b \vee ab$$

Hashtabelle zur
Berechnung der V^k

U \ V	M	0	1	2	3	4
2	x		1	1		1
3		1				
4						
5	x		1	1		
6		1				
\cap			1	1		

3.4 Mehrstufen-Logikminimierung

Die Multiplikation von algebraischen Ausdrücken f und g ist nur sinnvoll, falls g und verschiedene Variablen enthalten.

Einige Eigenschaften von algebraischen Faktoren und Divisoren

- Eine Logikfunktion g ist ein boolescher Faktor von f gdw $\overline{f} \cdot g = 0$ d.h. $g \nmid f$
- Falls $fg \neq 0$ dann ist g ein boolesche Divisor von f
- Falls g ein algebraischer Divisor (algebraischer Faktor) von f ist, dann ist g ein boolescher Divisor (boolescher Faktor) von f

3.4 Faktorisierung

Boolesche Division

$$F_1^{DC} \quad \square F_1^{DC} \quad \Rightarrow \bar{x}g \quad \Rightarrow x\bar{g}$$

$$F_1^{ON} \quad \square F_1^{ON} F_1^{DC}$$

$$F_1^{OFF} \quad \square F_1^{ON} \quad \Rightarrow F_1^{DC} \quad \square F_1^{ON} F_1^{DC} \quad \square F_1^{ON} F_1^{DC}$$

$$F_3^{ON} \quad \square \text{Minimum_Literal}(F_2^{ON}, F_1^{OFF}, x, \bar{x})$$

$$F_4^{ON} \quad \square \text{Minimiere}(F_3^{ON}, F_1^{DC}, F_1^{OFF})$$

$$H_0 \quad \square F_4^{ON} / \bar{x} \text{ (Quotient)}$$

$$H_1 \quad \square F_4^{ON} / x \text{ (Quotient)}$$

$$R \quad \square F_4^{ON} \quad \square (H_0 \Rightarrow H_1)$$

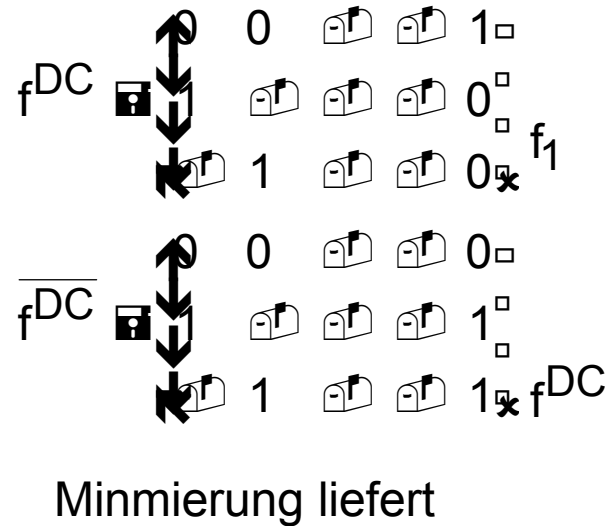
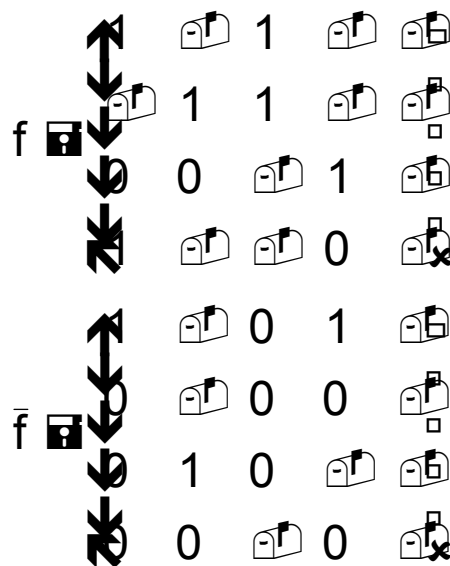
$$F \quad \square H_0 \bar{g} \vee H_1 g \vee R$$

3.4 Faktorisierung

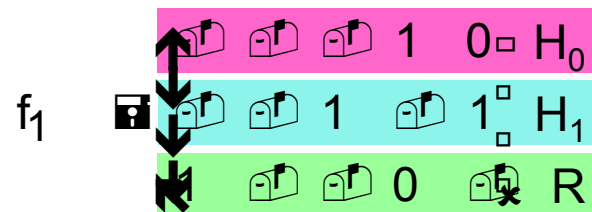
Bsp: Boolesche Division

$$f \models ac \vee bc \vee \bar{a}\bar{b}d \vee \bar{a}\bar{d} \models (a \vee b)c \vee (a \vee b)d \vee \bar{a}\bar{d}$$

$$g \models (a \vee b) \quad f^{DC} \models (a \vee b)\bar{g} \vee (a \vee b)g$$



Minimierung liefert



$$f \models cg \vee d\bar{g} \vee \bar{a}\bar{d}$$

Minimale
Spaltenüberdeckung

	0	0	1	0
	0	0	0	1
	1	0	0	1
	0	0	0	1
	0	1	0	0
	0	1	0	0
	0	0	0	1
	0	0	1	0
	0	0	0	0
	1	0	0	1
	0	1	0	0
	0	1	0	0

B =

3.4 Faktorisierung

Def.: Kernfunktionen (Kerne) $K(f)$ einer einwertigen booleschen Funktion f

Kernfunktionen sind (algebraische) Quotienten von f mit folgenden zwei Eigenschaften

- Ein Kern k eines Ausdrucks f ist der Quotient von f und einem Cube c
 $k = f/c$
- Ein Kern k hat keinen Cube als Faktor.

Beispiel

$$f = abc + abde$$

$f/a = bc + bde$ ist zwar ein Quotient aber der Cube b ist ein Faktor von f/a . Daher ist f/a kein Kern. aber $f/ab = c + de$ besitzt keine Cube als Faktor ist also ein Kern.

Zu jedem Kern k einer Funktion f gehört ein Co-Kern, der Cube c wobei $k = f/c$. Der Co-Kern ist nicht eindeutig bestimmt.

3.4 Mehrstufen-Logikminimierung

Kernels(f)

c ist größter Cube-Faktor von f

$K = \text{Kernel1}(0, f/c)$

if (f besitzt keinen Cube-Faktor $\neq 1$) then

return {f} u K

fi

Kernels1(j,g)

$R = \{g\}$

for i := j+1 to nl do

if (l_i kommt in mehreren Termen vor) then

c = größter Cube-Faktor von g/l_i

if ($l_k \nmid c$ für $1 \leq k \leq i$) {

$R = R \cup \text{Kernel1}(i, g/(l_i \wedge c))$

fi fi return R

Bsp.: Bestimmen von Kernfunktionen

$$f = abcd \vee abce \vee abf \vee abg \vee h$$

	a	b	c	d	e	f	g	h
f =	1	1	1	1	-	-	-	-
	1	1	1	-	1	-	-	-
	1	1	-	-	-	1	-	-
	1	1	-	-	-	-	1	-
	-	-	-	-	-	-	-	1

ab	a	b	c	d	e	f	g	h
	-	-	1	1	-	-	-	-
	-	-	1	-	1	-	-	-
	-	-	-	-	-	1	-	-
	-	-	-	-	-	-	1	-

c	a	b	c	d	e	f	g	h
	-	-	-	1	-	-	-	-
	-	-	-	-	1	-	-	-

Fakttorisierte Form

$$f = ab(c(d \vee e) \vee f \vee g) \vee h$$

Es kann aber auch sehr viele Kernfunktionen geben

3.4 Mehrstufen-Logikminimierung

Kernfunktionen der Stufe n

$$k \in K(f)$$

$$\text{Stufe 0 : } K^0(k) = \{k\}$$

$$\text{Stufe n : } \text{level}(q) \leq l-1 \wedge q \in K(k) : q \in K$$

$$\wedge q \in K(k) \text{ level}(q) = l-1$$

Beispiele

$$f = a(bd' + c(b'+d) + bc'd)$$

$$K^0(f) = \{ b'+d, ad' + c'd, bc' + ac \}$$

$$K^1 = K^0 \cup \{ bd' + c(d+b') \}$$

$$K^2 = K^1 + \{f\}$$

Falls f und g einen nicht trivialen gemeinsamen schwachen Divisor haben, so gibt es

$$k_f \in K(f), k_g \in K(g) \text{ mit } k_f \wedge k_g \text{ ist nicht trivial}$$

(Besitzt mehr als zwei Cubes)

3.4.1 Dekomposition

Dekomposition als Überdeckungsproblem

Zu jeder booleschen Funktion kann eine boolesche Matrix $\{b_{ij}\}$ definiert werden mit:

- Jede Zeile korrepondiert zu einem Term
- Jede Spalte korrespondiert zu einer Variablen
- b_{ij} ist gleich 1 fall die j-te variable im i-ten Term enthalten ist

Beispiel

$$f = ab\bar{d} + acd + bcd$$

3,4.1 Algorithmische Logikminimierung

Ein Rechteck von B ist eine Teilmenge der Z Zeilen mit $|Z| > 1$ und S Spalten, so daß alle $b_{ij} = 1$ für alle $i \in Z$ und $j \in S$

Ein primes Rechteck ist ein Rechteck, das in keinem anderen enthalten ist.

Ein Co-Rechteck (Z, S') zu einem Rechteck (Z, S) ist eine Rechteck, mit allen Spalten S' , die nicht in S enthalten sind.

3.4.1 Algorithmische Logikminimierung

Bsp.:

$$f_1 = ab(c(d \vee e) \vee f \vee g) \vee h$$

$$f_2 = ai(c(d \vee e) \vee f \vee j) \vee k$$

Finden von gemeinsamen Kernfkt. $K^0(f_1) = K^0(f_2) = \{ (d \vee e) \}$

$$g_0 = d \vee e$$

$$f_1 = ab(c g_0 \vee f \vee g) \vee h$$

$$f_2 = ai(c g_0 \vee f \vee j) \vee k$$

$$K^0(f_1) = \{ (c g_0 \vee f \vee g) \} \quad (c g_0 \vee f \vee g) \not\sim (c g_0 \vee f \vee j) \quad g_1 = c g_0 \vee f$$

$$K^0(f_2) = \{ (c g_0 \vee f \vee j) \}$$

$$f_1 = ab(g_1 \vee g) \vee h$$

$$f_2 = ai(g_1 \vee j) \vee k$$

Finden von gemeinsamen Cubes $g_2 = ag_1$

$$f_1 = b(g_2 \vee ag) \vee h$$

$$f_2 = i(g_2 \vee aj) \vee k$$

3,4 Algorithmische Logikminimierung

$$g_0 = d \vee e$$

$$g_1 = c g_0 \vee f$$

$$g_2 = a g_1$$

7 Literale

$$g_2 = a (c (d \vee e) \vee f)$$

5 Literale

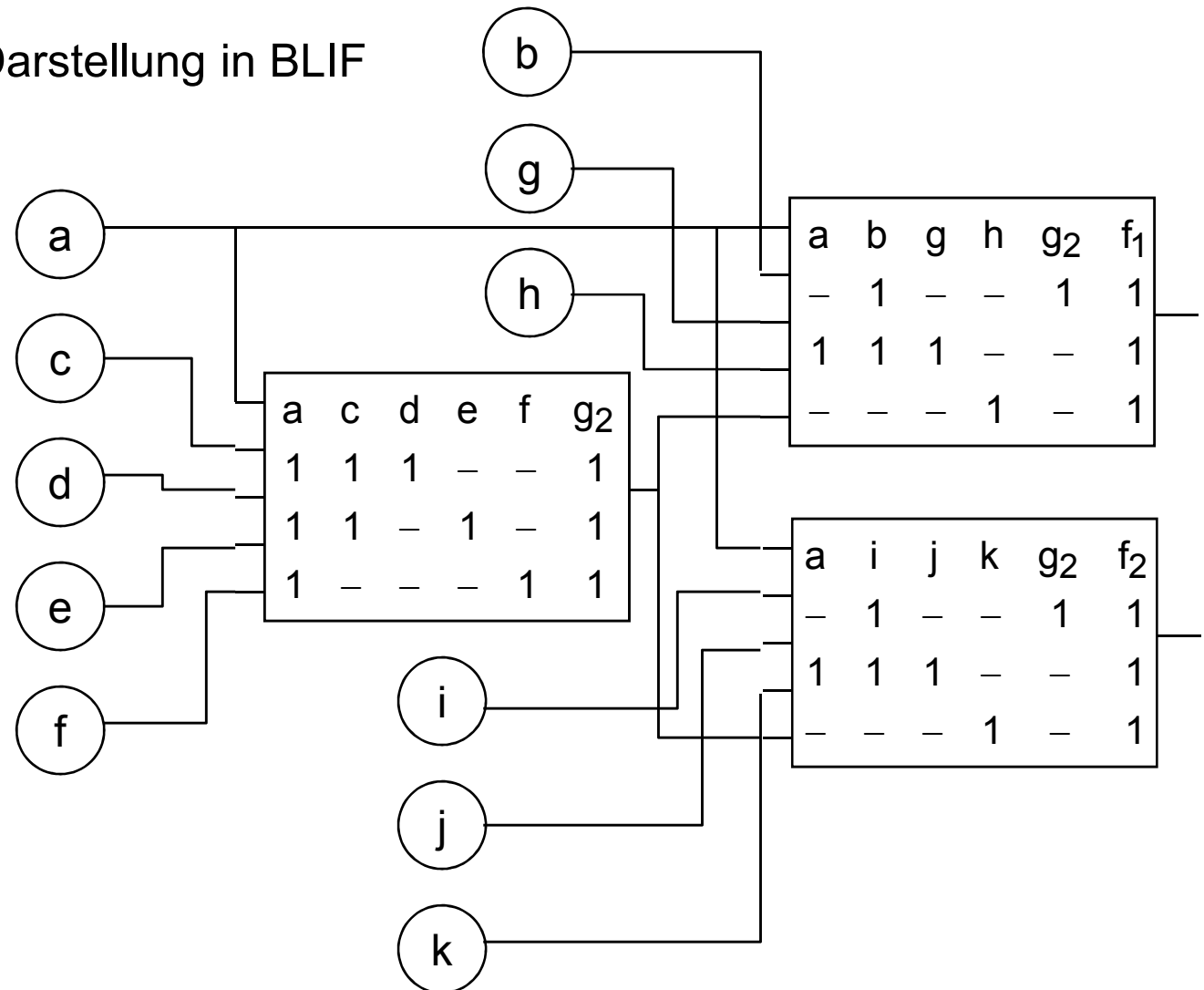
$$f_1 = b(g_2 \vee a g) \vee h$$

$$f_2 = i(g_2 \vee a j) \vee k$$

Realisierungsformen

- FPGA: Hier kann jede Teilfunktion direkt in ein CLB umgesetzt werden
- Bibliotheksbasierte Verfahren: weitere Verarbeitung notwendig

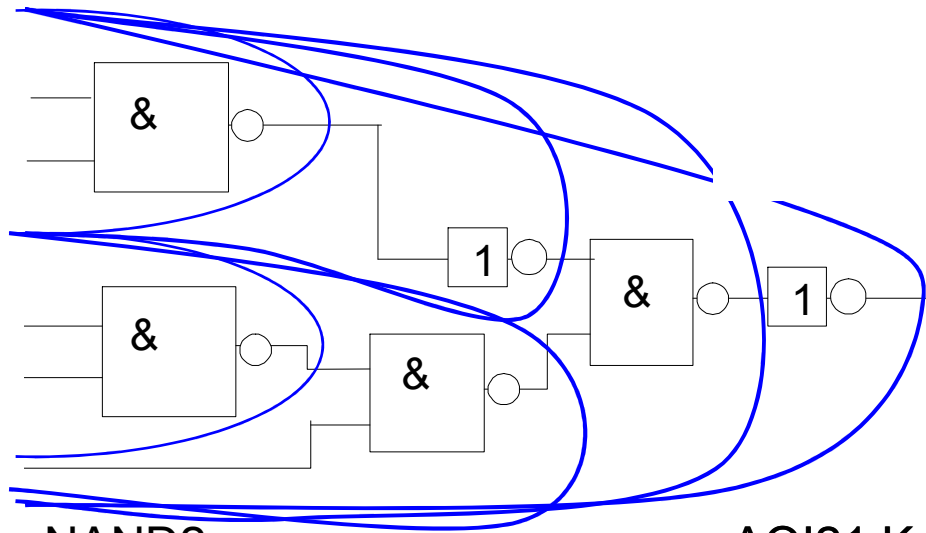
Darstellung in BLIF



3.4.2 Technologie-Abbildung (DACON)

**Eindeutige Darstellung durch NAND/NOR-Struktur
(vollständige Logik notwendig)**

Schaltungsgraph (NAND-Baum)
Dynamische Programmierung
NAND2 K = 2

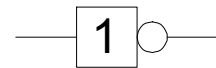


NAND2
K = 2

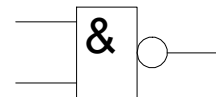
AOI21 K = 8
INV1 K = 10

Bibliothek

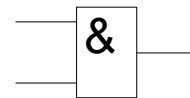
INV1



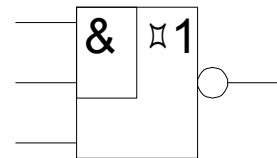
NAND2



AND2



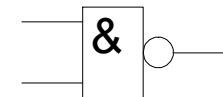
AOI21



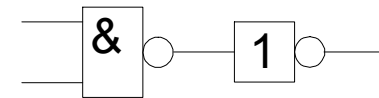
Vergleichsmuster Kosten



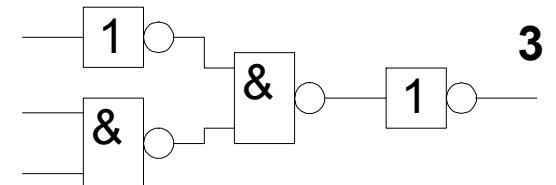
1



2



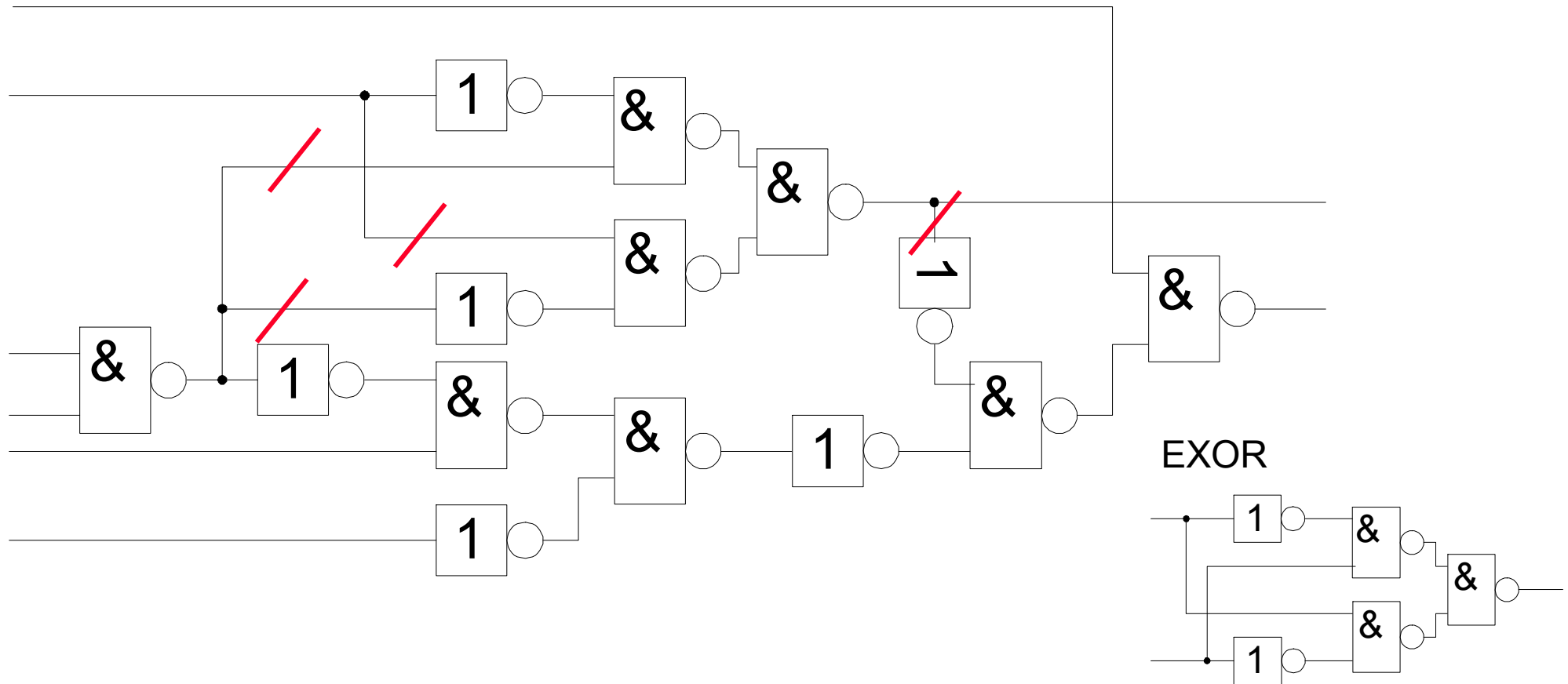
3



3

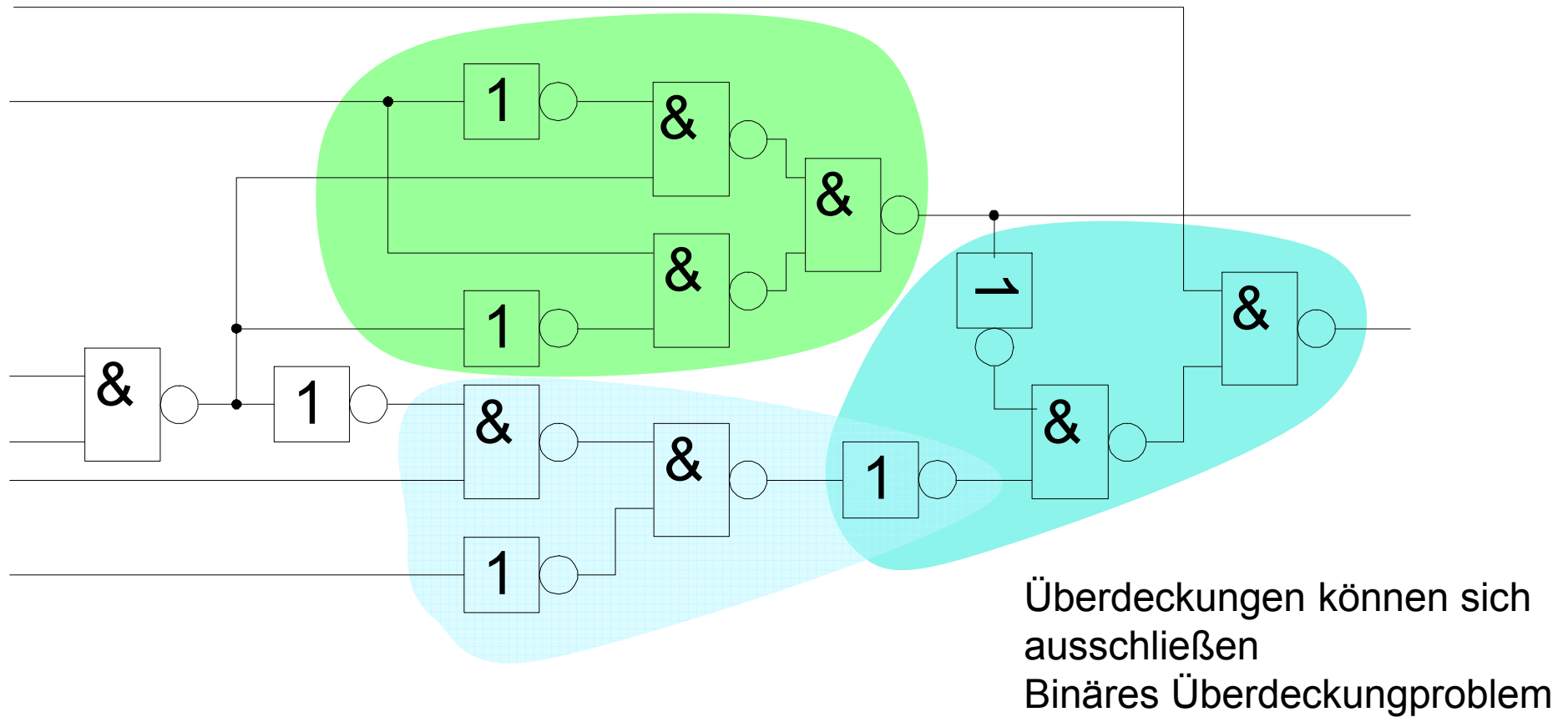
3.4.2 Technologie-Abbildung

Allgemeine gerichtete Graphen Zerlegen in Bäume (Schneiden)



3.4.2 Technologie-Abbildung

Allgemeine gerichtete Graphen Subgraphmatching



3.4.2 Technologieabbildung

Das LSS-System

Eingabe: Eine Beschreibung der Schaltung auf der RT-Ebene.

Ausgabe: Optimierte(s) Schaltnetz(e) in bezug auf eine Zielbibliothek.

Zunächst wird die Spezifikation in umgeformt und in einzelne Schaltnetze, dargestellt als UND-ODER-Verknüpfung, zerlegt.

Vorverarbeitung:

- Eliminierung gemeinsamer Unterausdrücke
- Konstantenpropagierung

Bsp.:

Multiplexer

$$y = x \text{ sel} \vee '1' \overline{\text{sel}}$$

$$y = x \text{ sel} \vee \overline{\text{sel}} = x \vee \overline{\text{sel}}$$

Vergleich mit '0' ($x = '0'$)

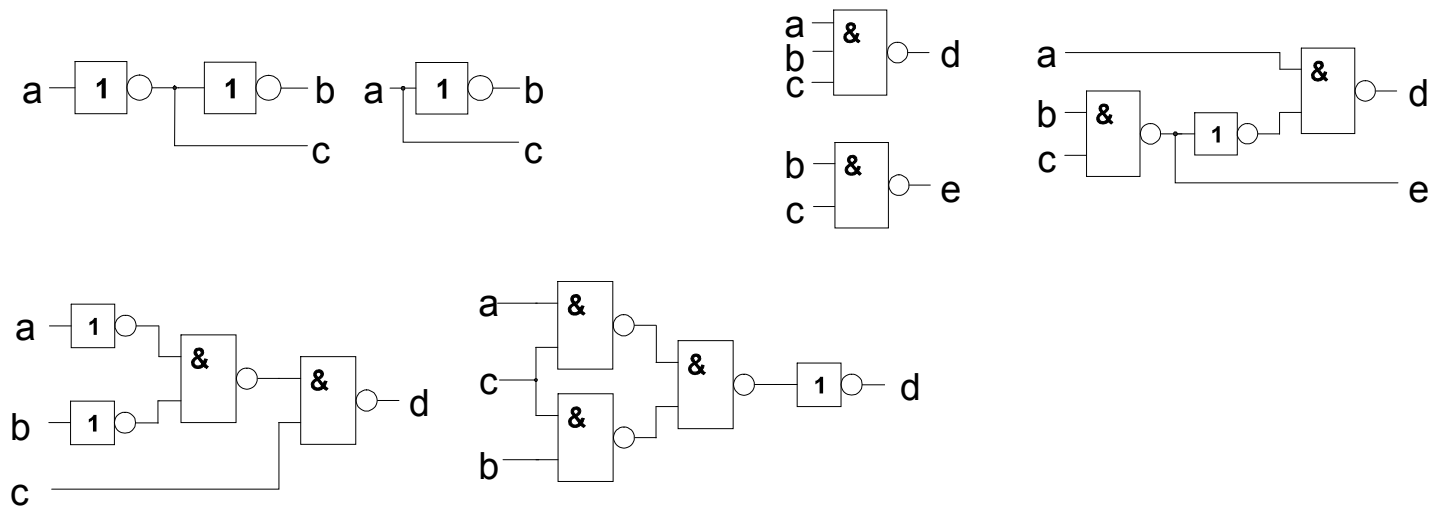
$$y = (x_{n-1}'0' \vee \overline{x_{n-1}}'1') \wedge \dots \wedge (x_0'0' \vee \overline{x_0}'1')$$

$$y = \overline{x_{n-1}} \wedge \dots \wedge \overline{x_0} = \overline{x_{n-1} \vee \dots \vee x_0}$$

- Umformung in eine NAND-Struktur und Regelanwendung

3.4.2 Technologieabbildung

Beispiele für NAND-Regeln [DJBT81]

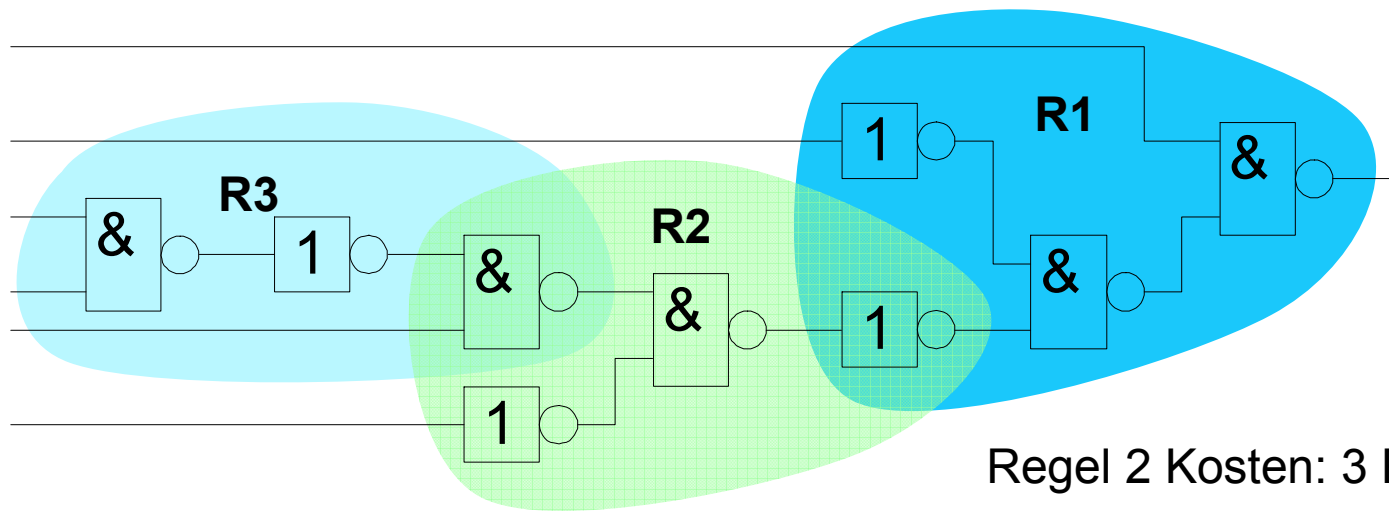


Das System sollte auch zur Logikminimierung eingesetzt werden

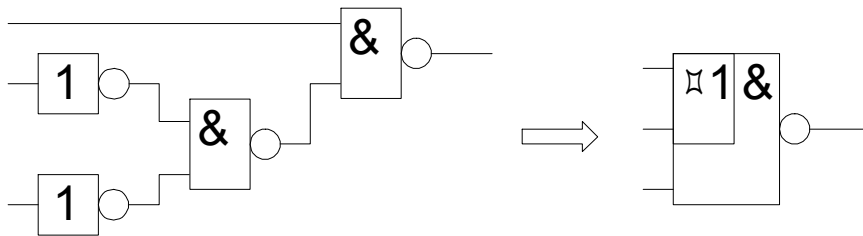
3.4.2 Das LSS-System

Regeln können sich gegenseitig beeinflussen

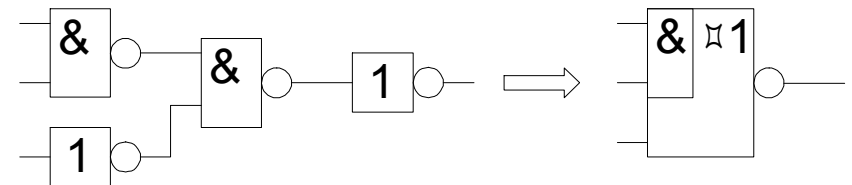
Anwendung Regel 2 : Kosten 11
Anwendung Regel 1: Kosten 9



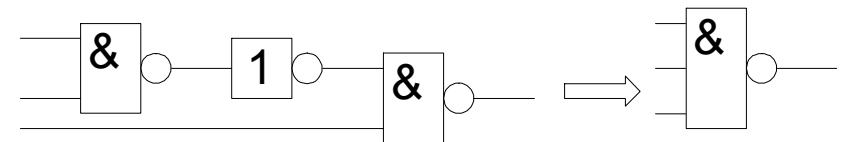
Regel 1 Kosten: 3 Red.: 3



Regel 2 Kosten: 3 Red.: 3

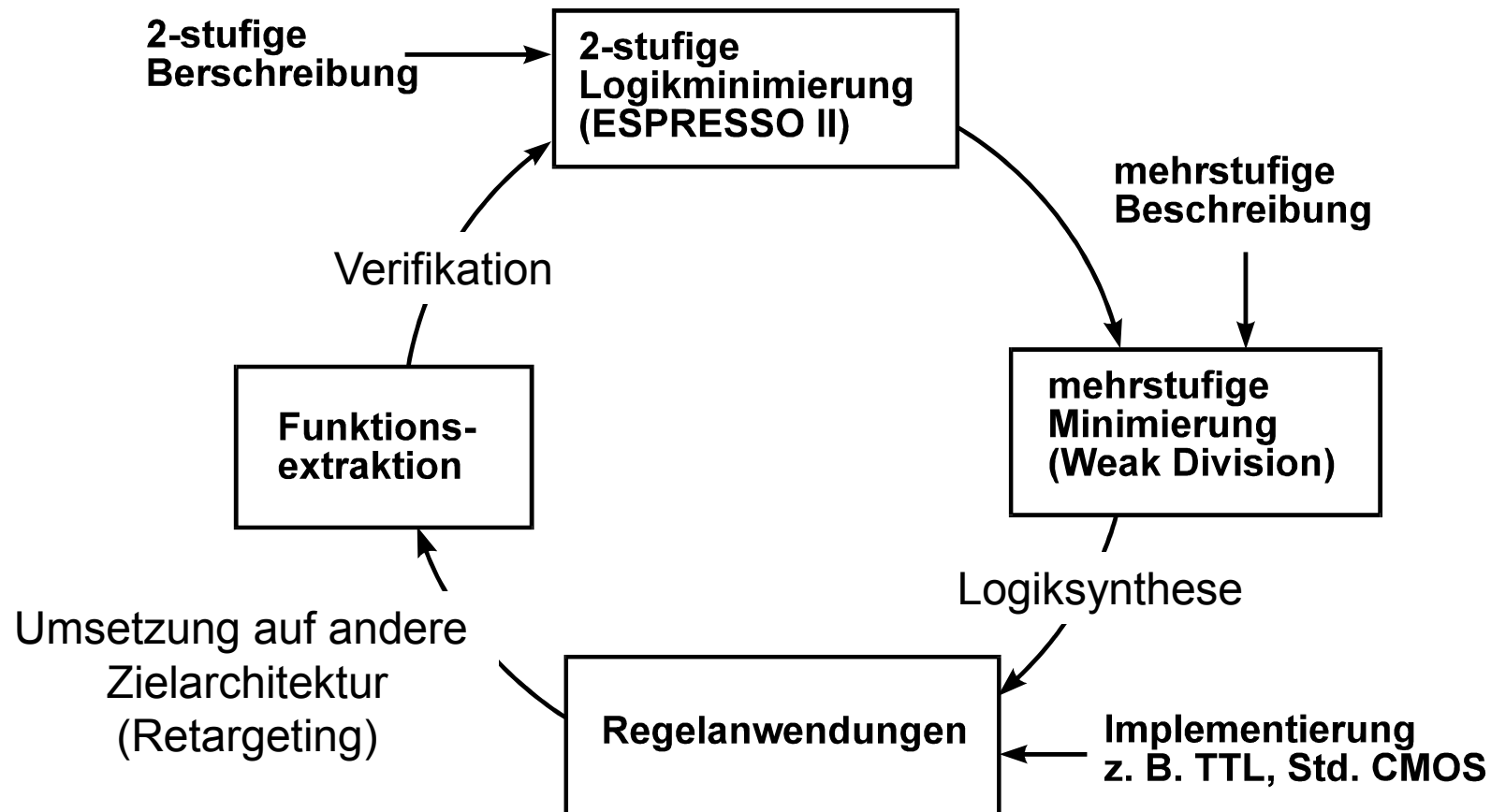


Regel 2 Kosten: 3 Red.: 2



3.4.2 Technologieabbildung

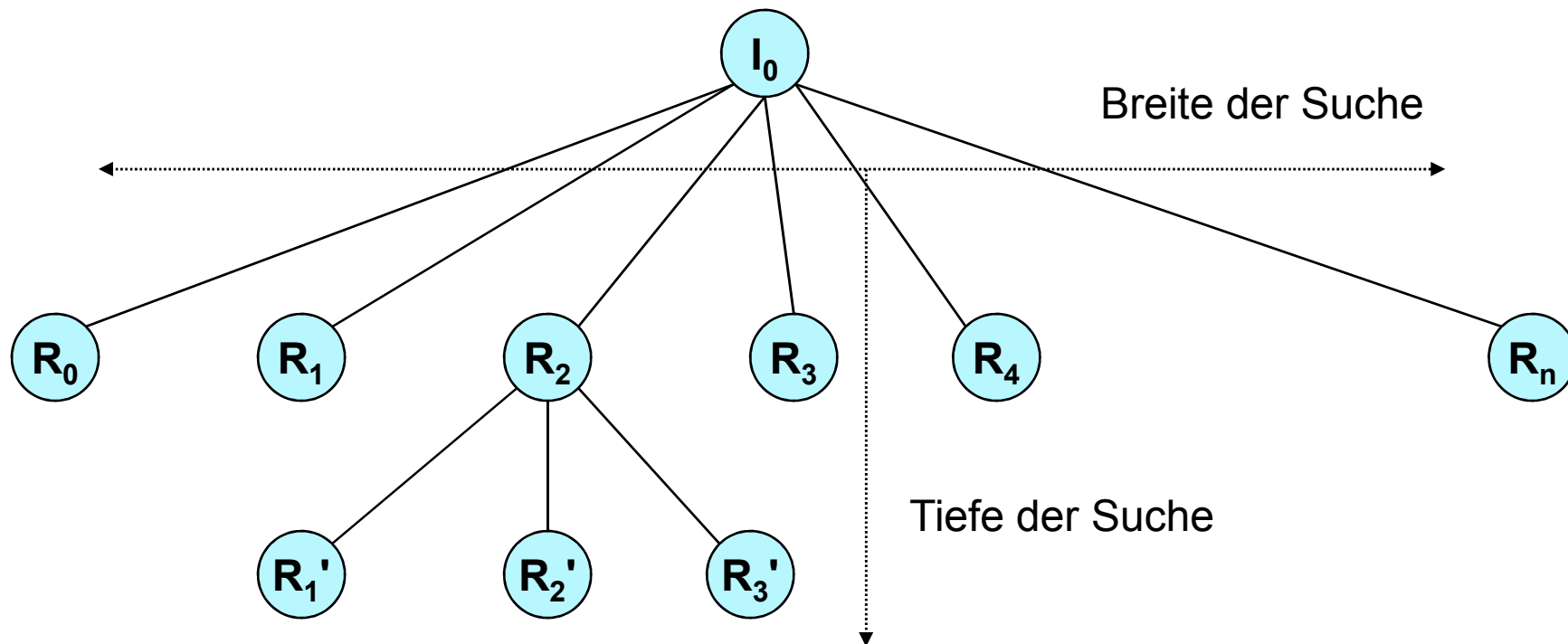
Das SOCRATES System



3.4.2 Das SOCRATES-System

Regelanwendung

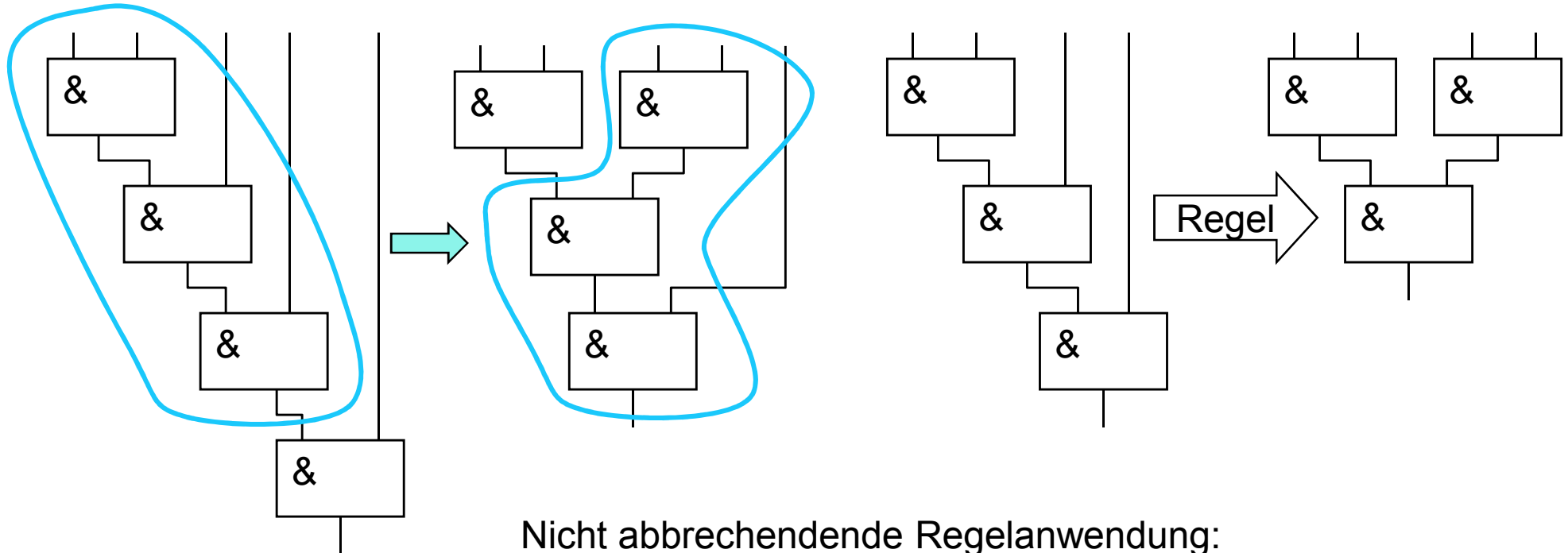
Es werden verschiedene Möglichkeiten probiert.



Das System muß in der Lage sein vorgenommene Änderungen rückgängig zu machen. Daher benötigt das Verfahren viel Speicherplatz.

3.4.2 Das SOCRATES-System

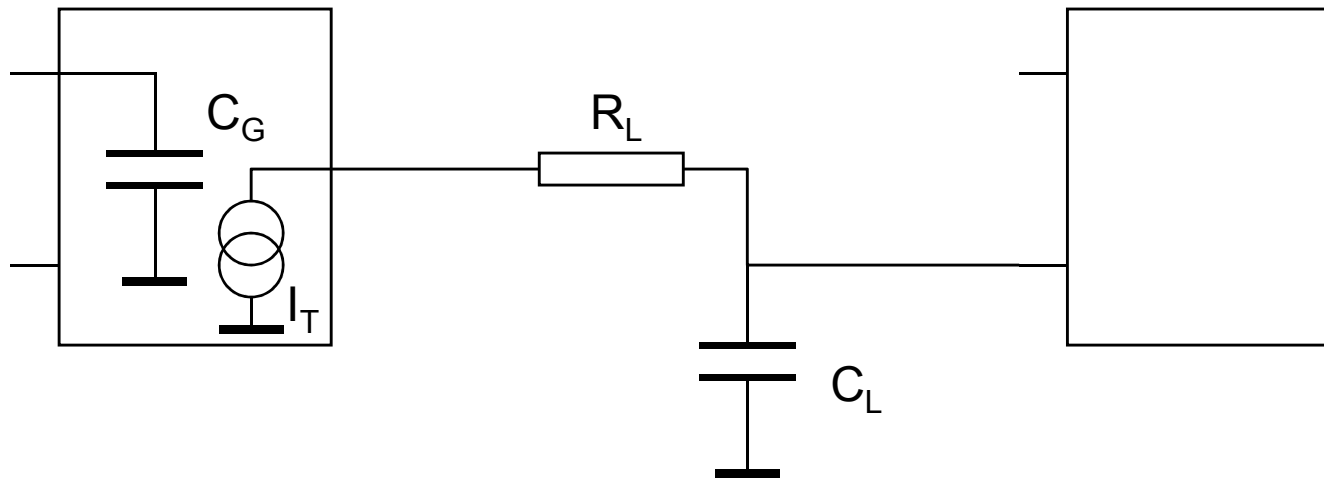
Problem: Verkürzung von Schaltzeiten



Nicht abbrechende Regelanwendung:
Es muß daher die Laufzeit der Schaltung brechnet werden.
Dies ist allerdings keine lokale Eigenschaft mehr

3.4.2 Technologie-Abbildung

Optimierung von Schaltzeiten Einfaches Gattermodell (CMOS)



Problem: C_L und R_L sind erst nach dem Layout bekannt * Schätzverfahren

Bem.: Die Technologieabbildung kann auch auf eine handentworfene Schaltung angewandt werden. Dies wird als inkrementelle Synthese bezeichnet. Durch die bessere Anpassung an die Bibliothek kann damit eine Verkürzung der Schaltzeit um den Faktor 2 bis 5 erreicht werden.

1 Einleitung

Bisher Entwurf von Schaltnetzen für synchrone Schaltungen

✱ **Entwurf des Operationswerkes**

Jetzt:

- **Entwurf von synchronen Schaltwerken**

Entwurf des Steuerwerks

Verkürzung der Zykluszeit (Retiming)

4.1 Theoretische Überlegungen

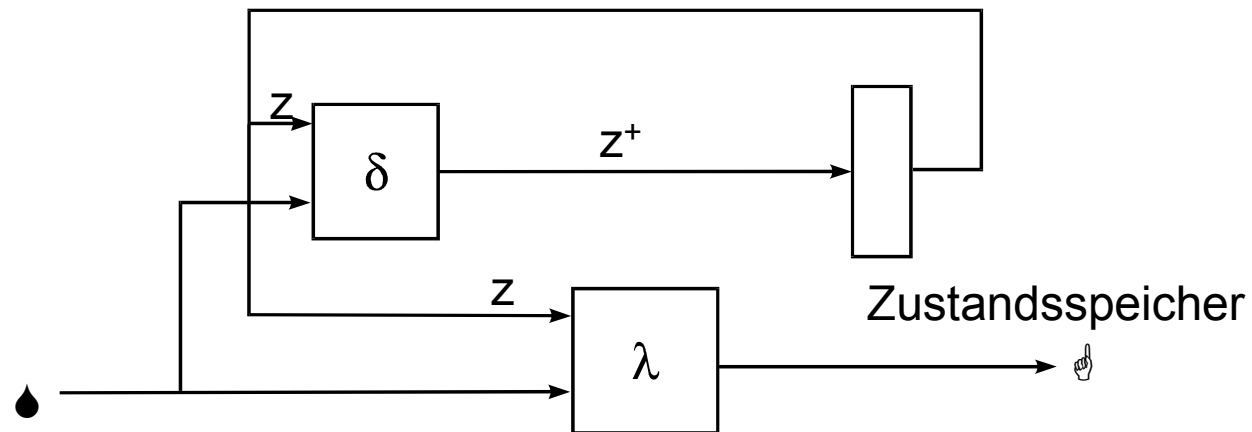
Endlicher Automat A ist definiert durch ein Quintupel:

$A = (\Sigma, \Gamma, Z, \delta, \gamma)$ mit

- Σ ist eine endliche Menge $\{\sigma_i \mid i = 1 \dots n\}$ von Eingabesymbolen
- Γ ist eine endliche Menge $\{\gamma_i \mid i = 1 \dots m\}$ von Ausgabesymbolen
- Z ist eine endliche Menge $\{z_i \mid i = 1 \dots k\}$ von internen Zuständen
- δ ist die Überföhrungsfunktion $\delta : \Sigma \times Z \rightarrow Z$
- γ ist die Ausgabefunktion. Hier werden zwei Typen von Automaten unterschieden:
 - Der **Moore-Automat**: $\gamma : Z \rightarrow \Gamma$
Hier ist der Wert der Ausgabe nur vom internen Zustand abhängig.
 - Der **Mealy-Automat**: $\gamma : Z \times \Sigma \rightarrow \Gamma$
Hier ist der Wert der Ausgabe sowohl vom internen Zustand als auch vom jeweiligen Eingabewert abhängig

4.1 Typen von Automaten

Mealy-Automat

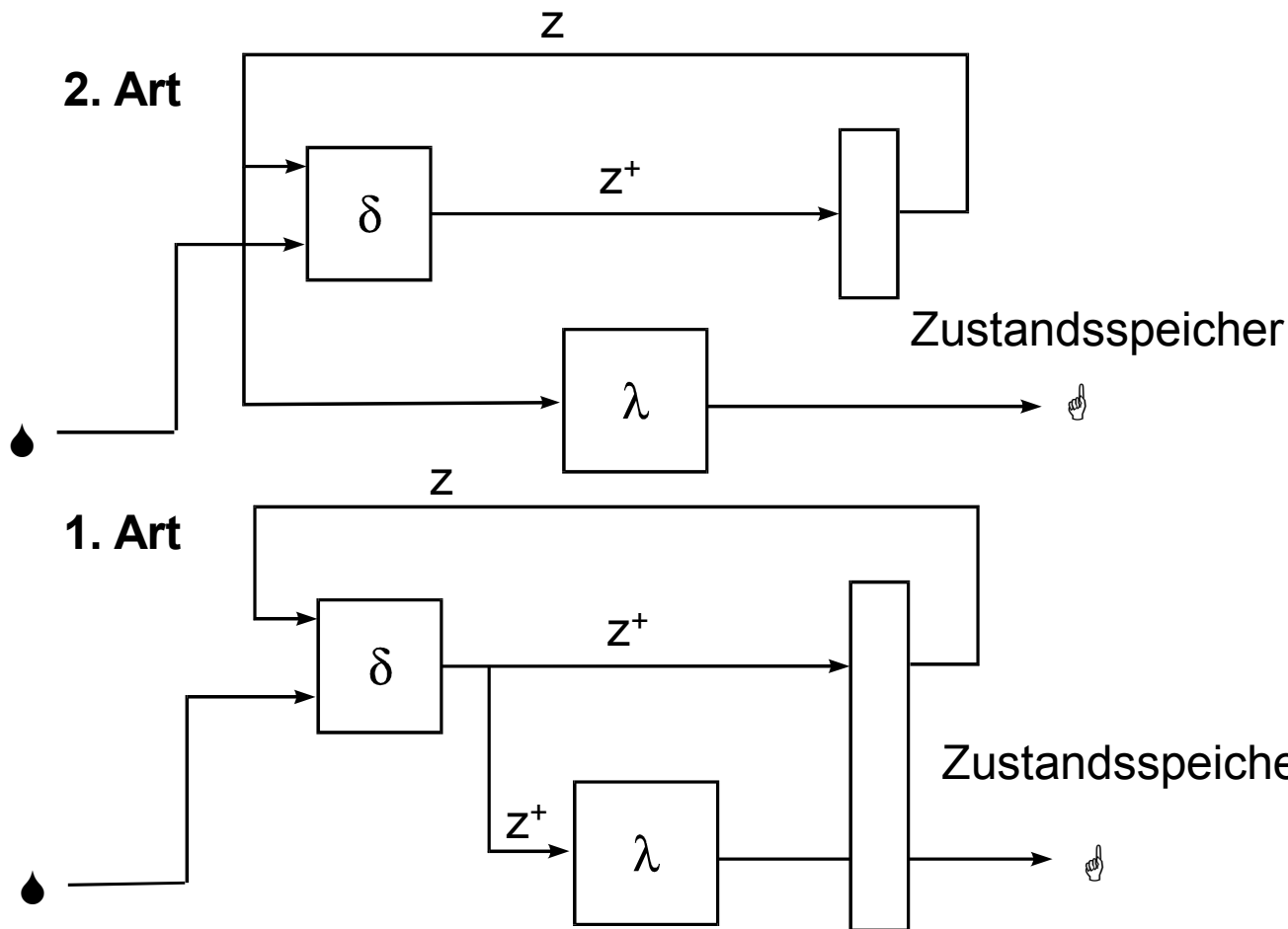


Bei einer Realisierung müssen die Eingangs-, die Ausgangssymbole und der Zustand durch boolesche Variablen $x = (x_1, x_2, \dots)$, $y = (y_1, y_2, \dots)$ und $q = (q_1, q_2, \dots)$ codiert werden. Die x_i heißen dann **Entscheidungs-**, die y_i **Steuer-** und die q_i **Zustandsvariablen**. Die Funktionen δ und λ sind dann geeignete boolesche Funktionen.

Beim Mealy-Automaten hängen also die Steuervariablen vom Zustand und den Entscheidungsvariablen ab.

4.1 Typen von Automaten

Moore-Automaten

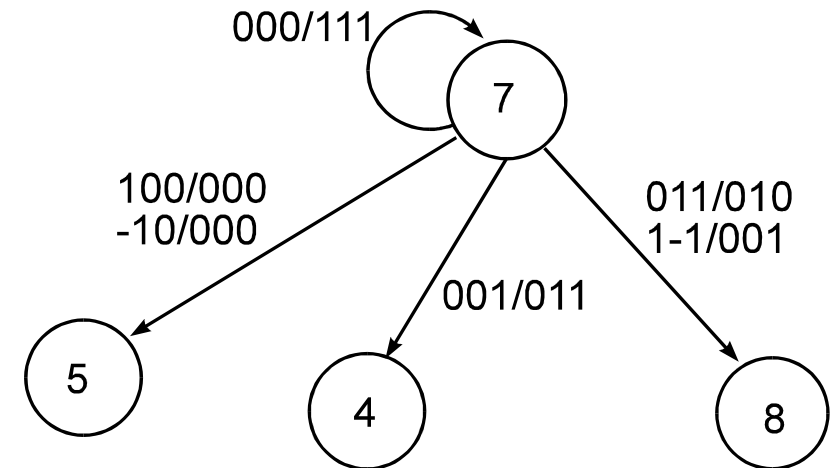
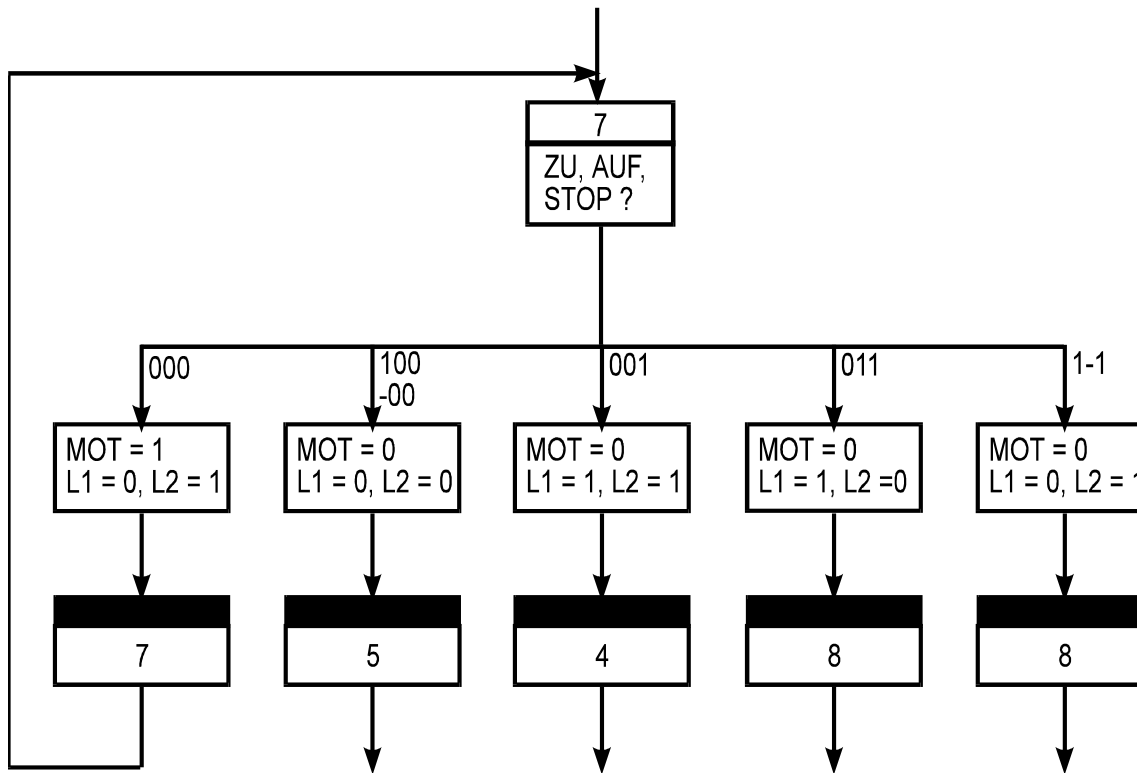


Beim Moore-Automaten hängen die Steuervariablen nur vom Zustand z bzw. Nachfolgezustand z^+ ab.

Hier werden die Steuervariablen zwar von der Eingabe beeinflusst, diese werden aber zwischengespeichert.

4.2 Darstellung von Automaten

1.2.1 Ablaufdiagramm, Zustandsübergangsgraph



4.2 Darstellung von Automaten

1.2.2 Flußtabelle

		Symbole			
		Eingabesymbol			
Nachfolgezustand		A	B	C	D
aktueller Zustand	1	2,0	-,1	3,-	2,0
	2	3,0	5,1	2,0	-, -
	3	3,0	4,1	-, -	5,0
	4	-, -	1,1	2,-	-, -
	5	-, -	-, -	1,1	-, -

unspezifizierter Eintrag

unvollständig spezifizierter Automat

4.2 Darstellung von Automaten

Automatentabelle (KISS-Format)

Symbolisch

Entscheidungs- variable(n)	EV	Z	Z ⁺	SV
	0	start	state6	00
	0	state2	state5	00
Aktueller Zustand	0	state3	state5	00
	0	state4	state6	00
	0	state5	start	10
Nachfolge- zustand	0	state6	start	01
	0	state7	state5	00
	1	start	state4	00
	1	state2	state3	00
	1	state3	state7	00
	1	state4	state6	10
	1	state5	state2	10
	1	state6	state2	01
	1	state7	state6	10

Steuervariablen

Codiert

EV	Z	Z ⁺	SV
0	100	101	00
0	110	001	00
0	011	001	00
0	000	101	00
0	001	100	10
0	101	100	01
0	010	001	00
1	100	000	00
1	110	011	00
1	011	010	00
1	000	101	10
1	001	110	10
1	101	110	01
1	010	101	10

4.3 Realisierung von Automaten

Entwurfsschritte

Systementwurf,
Partitionierung in Teilsysteme

Zustandsreduktion

Auswahl der
Steuerwerksarchitektur

Zustandscodierung

Logiksynthese
Logikminimierung

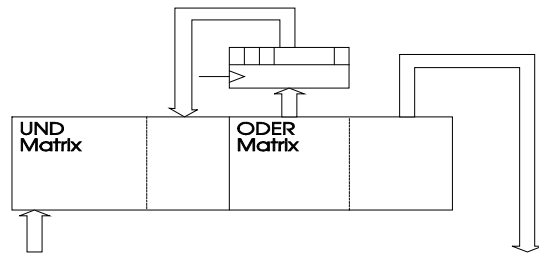
5 Realisierung von Steuerwerken

- **Synchrone Steuerwerke**
- **Realisierung der Logikfunktionen**
 - Gatterlogik
 - PLA/PAL
 - ROM
- **Zustandsspeicher**
 - Flipflops (D-FF, JK-FF, T-FF)
 - Zähler (Ringzähler, Johnsonzähler, Binärzähler)

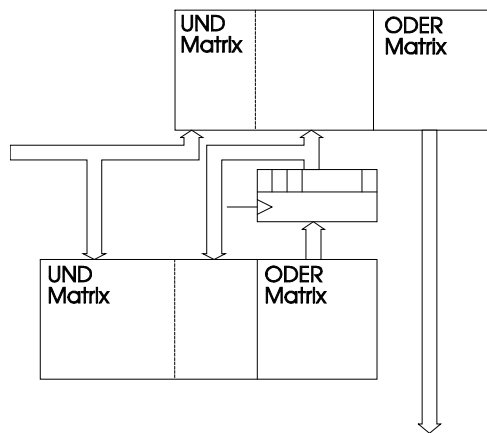
5.1 Realisierung der Logikfunktion

Realisierung von Automaten

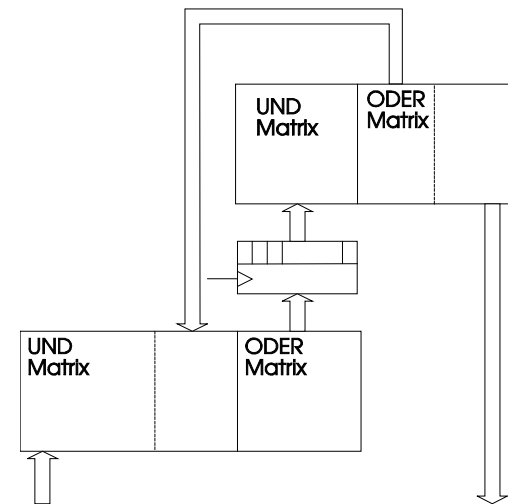
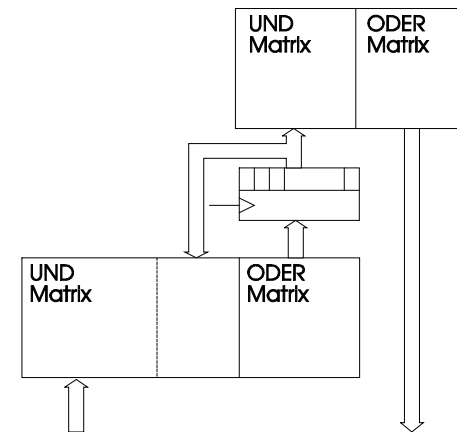
Einfach PLA-Lösung



Doppel PLA-Lösung

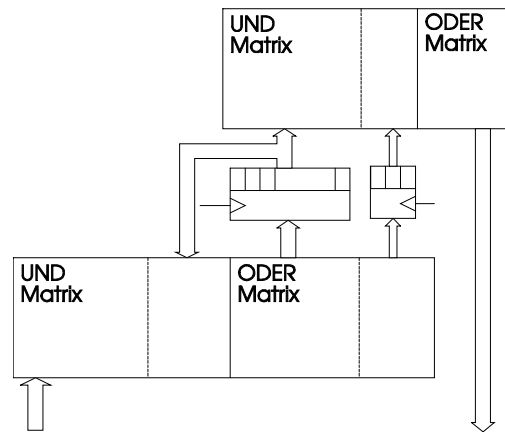


Doppel-PLA-Lösung für Moore-Automaten

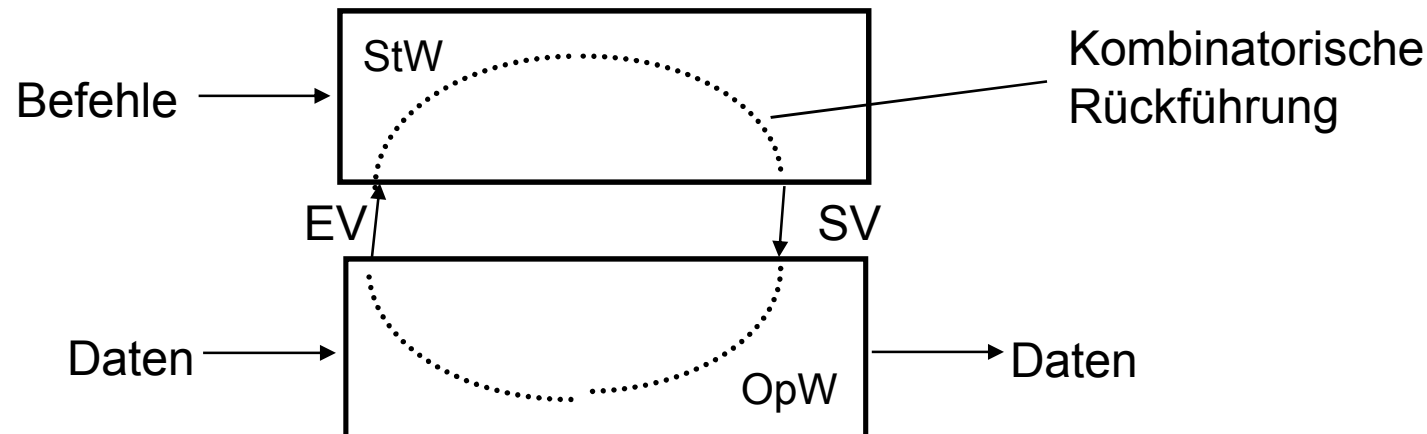


5.1 Realisierung der Logikfunktion

Realisierung für lokal transformierte Moore-Automaten



Problem bei der Verwendung eines Mealy-Automaten als Steuerwerk



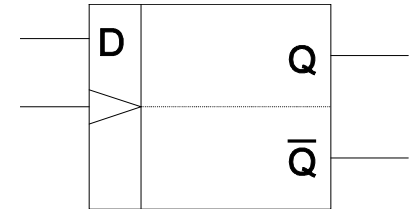
5.2 Realisierung des Zustandsspeichers

D-Flipflop

–immer

$$D = z_i^+$$

D	Q^t	Q^{t+1}
0	Q	0
1	Q	1



JK-Flipflop

$$-z_i = z_i^+ = 1$$

J = don't care

$$K = 0$$

$$-z_i = z_i^+ = 0$$

$$J = 0$$

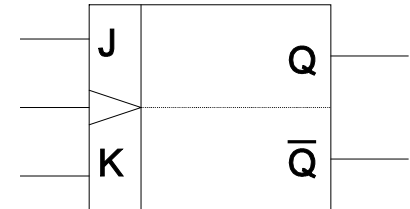
K = don't care

$$-z_i \text{ (clock) } z_i^+$$

$$J = 1$$

$$K = 1$$

J	K	Q^t	Q^{t+1}
0	0	Q	Q
0	1	Q	0
1	0	Q	1
1	1	Q	\overline{Q}



T-Flipflop

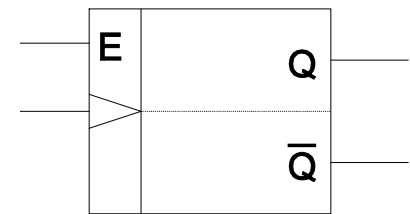
$$-z_i = z_i^+$$

$$E = 0$$

$$-z_i \text{ (clock) } z_i^+$$

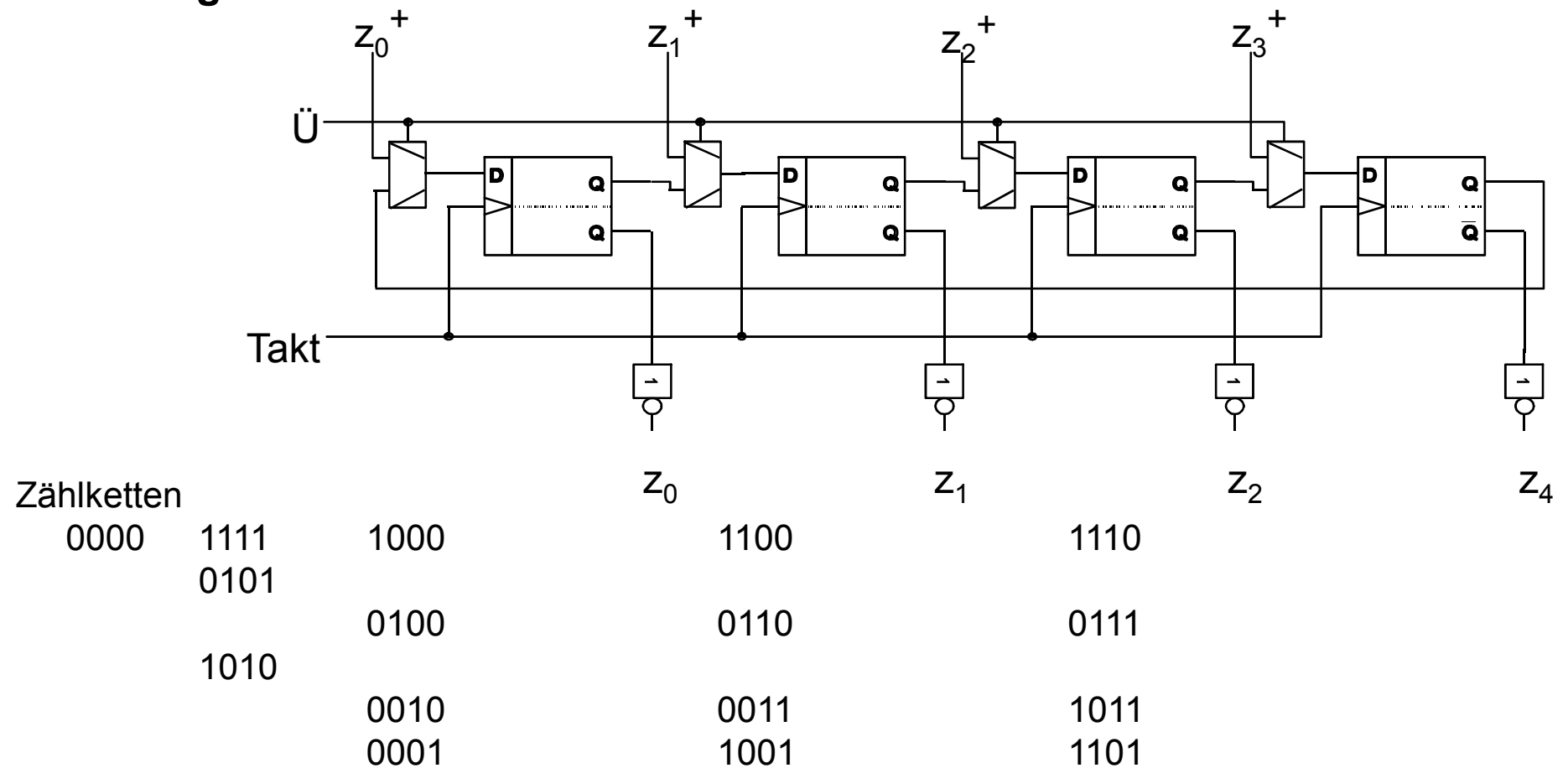
$$E = 1$$

E	Q^t	Q^{t+1}
0	Q	Q
1	Q	\overline{Q}



5.2 Zählersteuerwerke

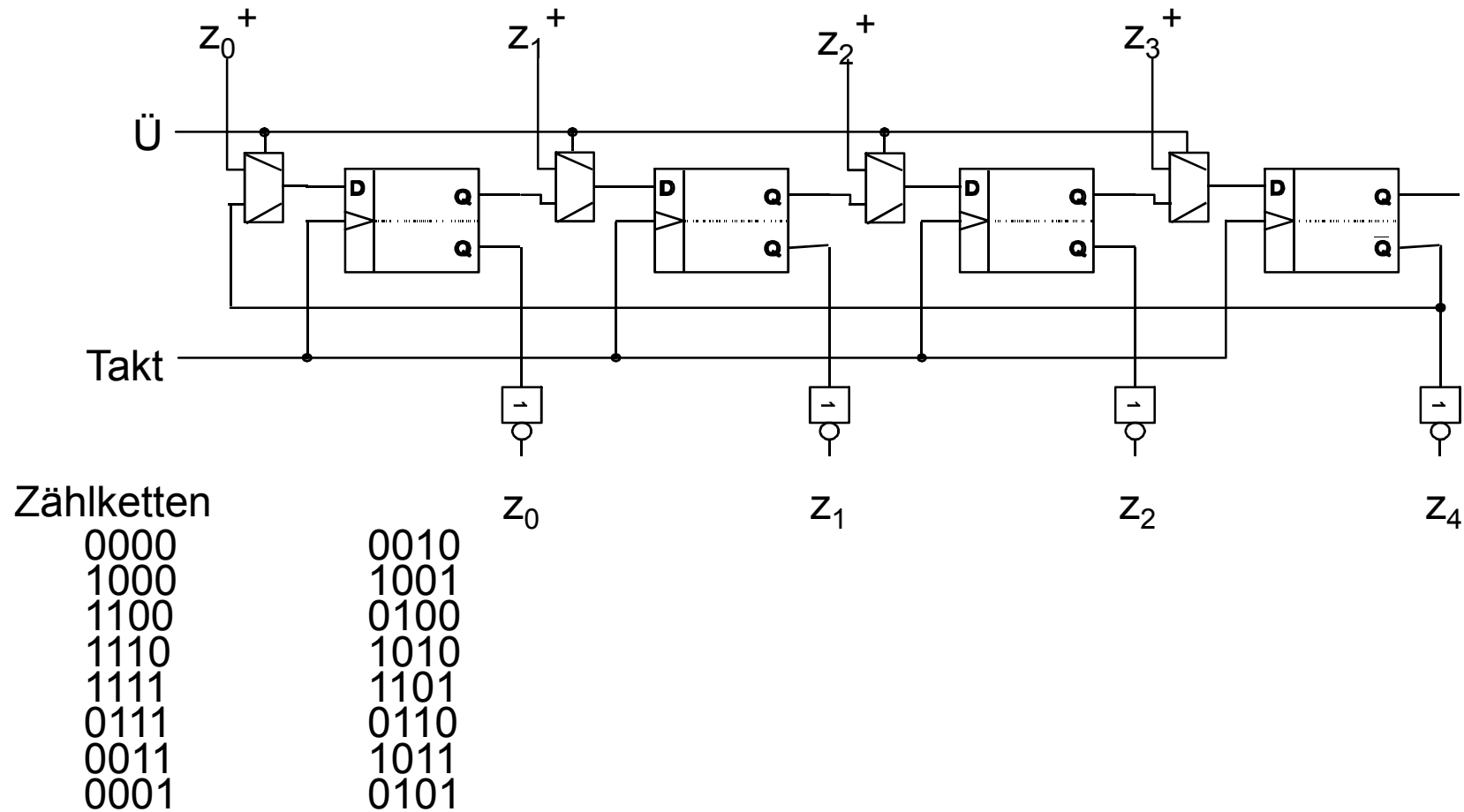
Ladbarer Ringzähler



Ein n-Bit-Ringzähler besitzt Zählfolgen mit einer maximalen Länge von n.

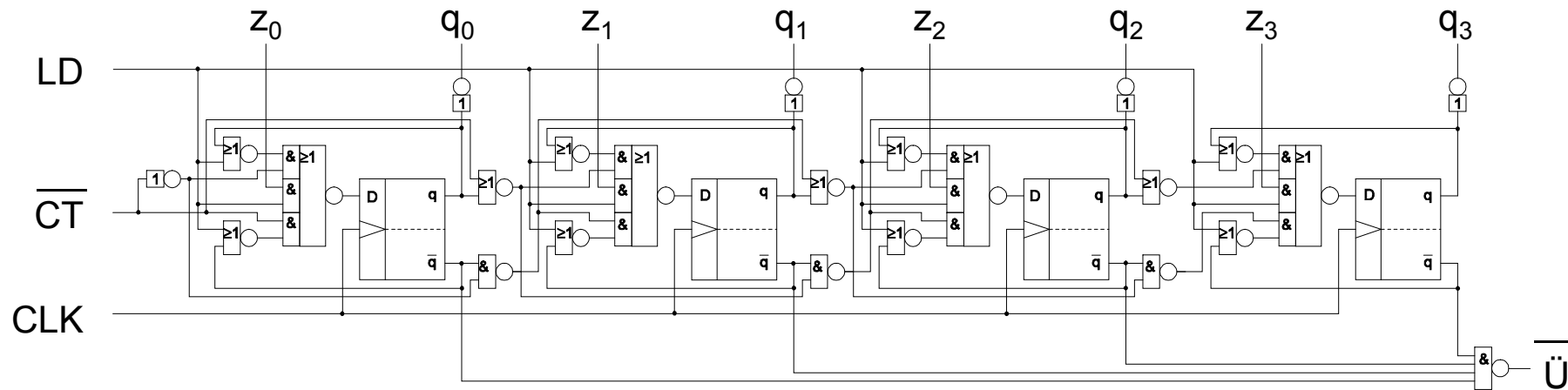
5.4 Zählersteuerwerke

Ladbarer Johnsonzähler



5.4 Zählersteuerwerke

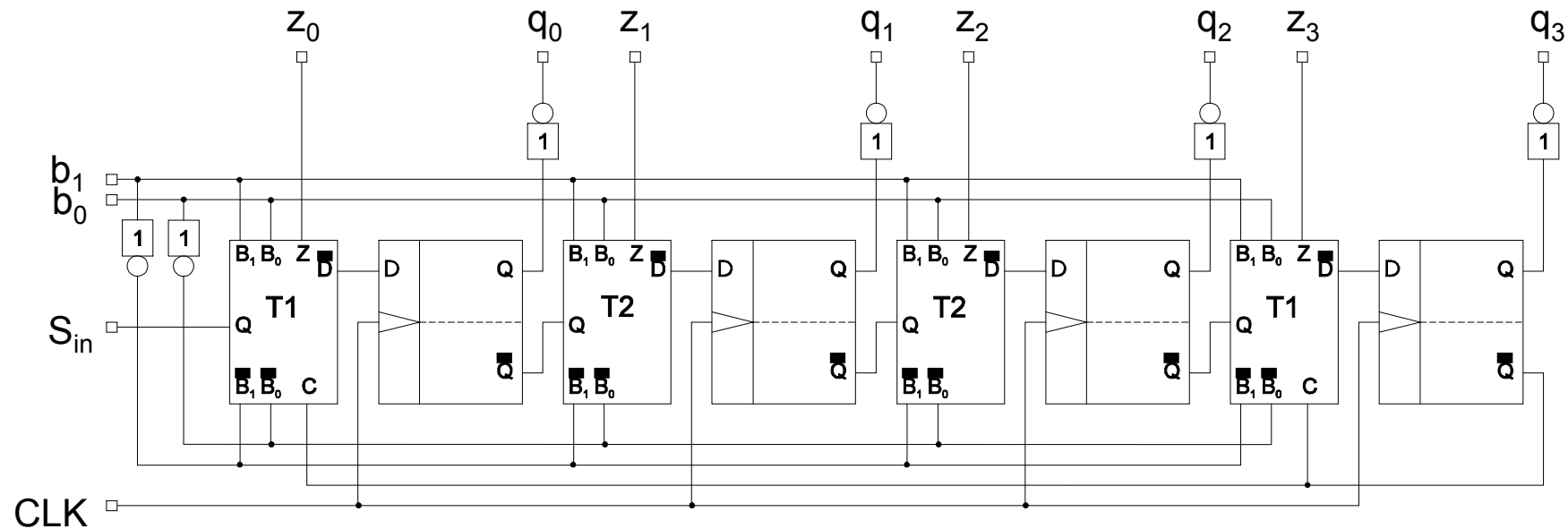
Ladbarer, synchroner 4-Bit-Binärzähler



Die Zählkette umfaßt alle 2^n Codierungen

5.4 Zählersteuerwerke

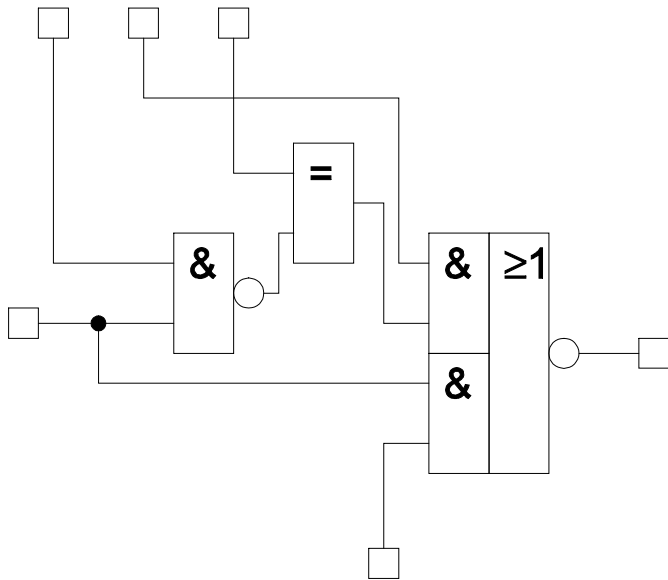
MISR (Multiple Input Signature Register, modulares LRSR nach [Wund91])



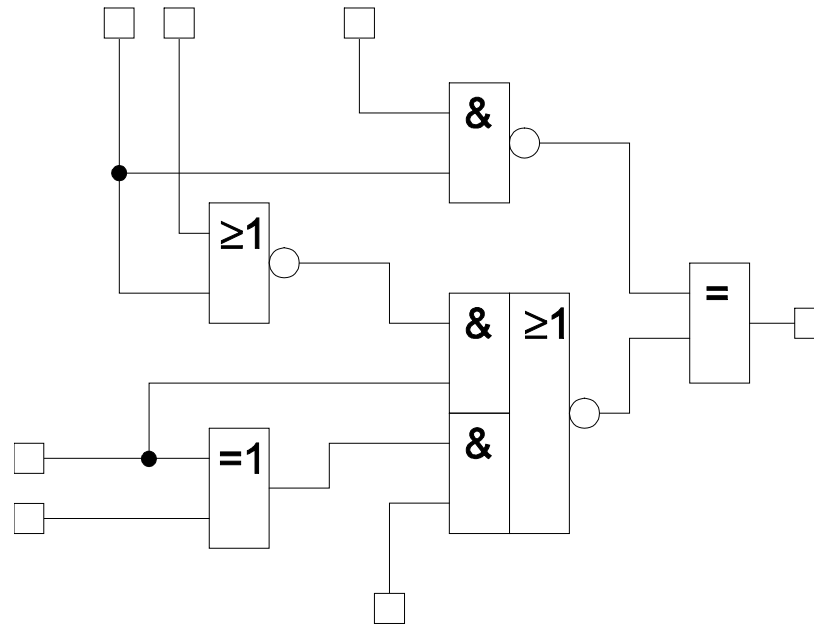
Linear rückgekoppeltes Schieberegister zum Polynom $x^4 + x^3 + 1$
Funktion

5.4 Zählersteuerwerke

T2

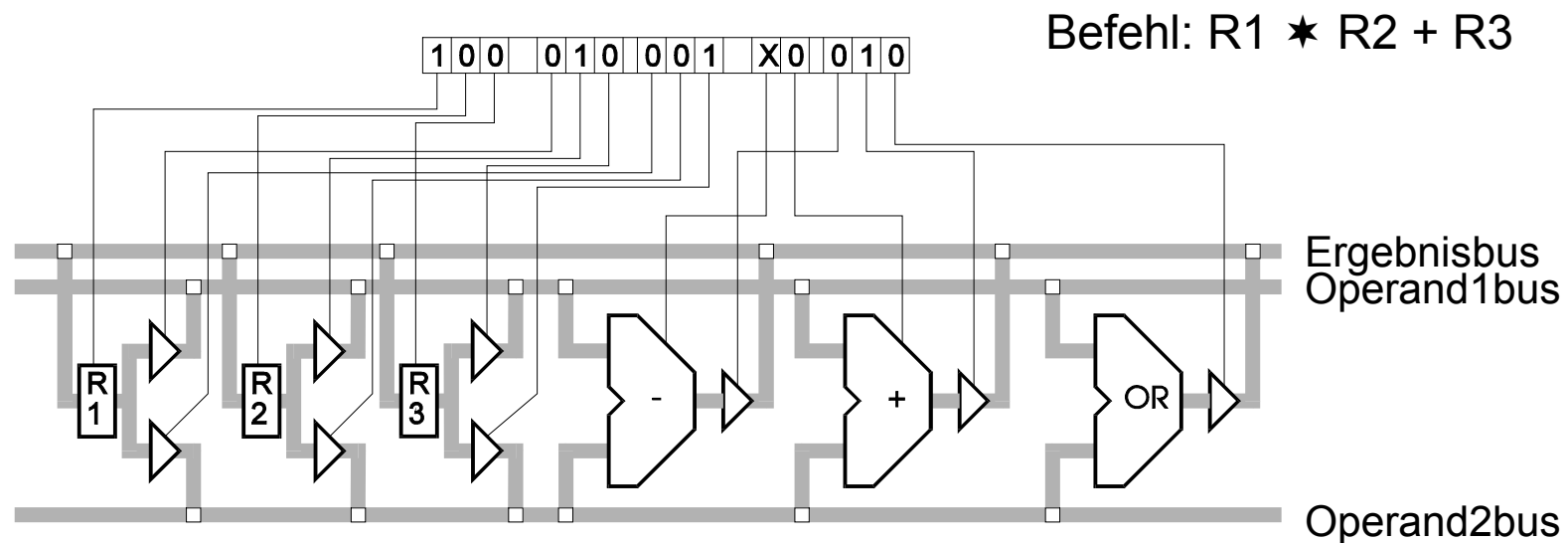


T1



5.5 Mikroprogrammsteuerwerke

Direkte Steuerung



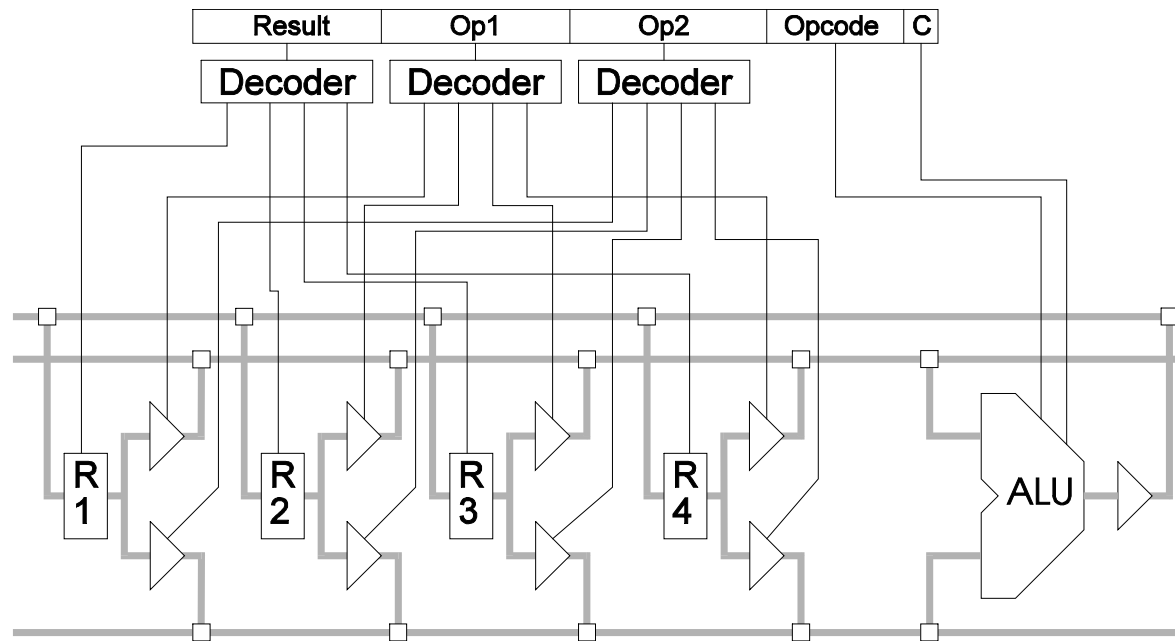
Kennzeichen

- Jede Operation wird durch eine Verarbeitungseinheit realisiert
- Die Auswahl der Operationen erfolgt über Tore (Gates) oder Treiber

Bemerkung: Diese Art der Implementierung kommt in der Praxis nicht vor

5.5 Mikroprogrammsteuerwerke

Direkte Codierung



Kennzeichen

- Sich gegenseitig ausschließende Operationen werden in einem Befehlsfeld codiert
- Es ergeben sich (nahezu) keine Einschränkungen in der möglichen Parallelarbeit

5.5 Mikroprogrammsteuerwerke

Direkte Codierung mit "sharing"

- Die Steuerbits für verschiedene Operationseinheiten werden gemeinsam benutzt. Beispielsweise können Operationen auf bestimmte Registergruppen beschränkt werden

Horizontale Mikroprogrammierung

- Durch das Steuerwort kann jeder **mögliche** Befehl realisiert werden. Hierdurch wird der Programmspeicher breiter und damit größer.

Vertikale Mikroprogrammierung

- Alle **zulässigen** Befehle werden in einem Steuerwort codiert. Hierdurch kann die Bitbreite stark reduziert werden. Allerdings sind spätere Änderungen nur sehr schwer möglich.

5.5 Mikroprogrammsteuerwerke

Weitere Möglichkeiten

- **Residual control**

Die Funktion der Hardware hängt nicht nur von dem Befehlswort sondern auch von einem inneren Zustand ab. (Residual control register)

- ✱ Partitionierung der Automaten

- **Zweistufige Codierung**

- Bit steering

- Die Bedeutung eines Befehlsfeldes hängt von einem anderen Befehlsfeld ab

- Format shifting

- Die Bedeutung eines Befehlsfeldes hängt von vom inneren Zustand ab

5.6 PLA-Steuerwerke

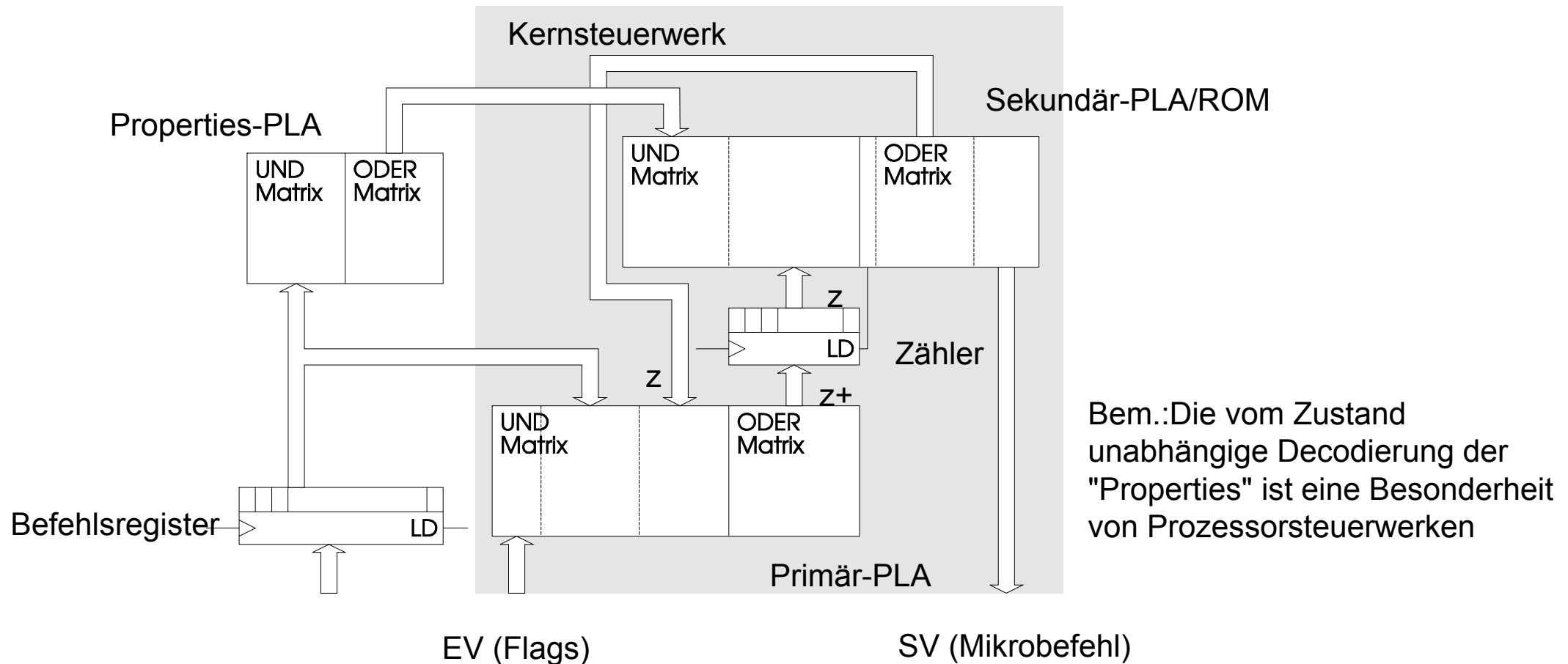
Steuerwerke für Mikroprozessoren

Ziel ist hierbei die Reduktion der Zustandsanzahl

Idee: Die Generierung der Steuersignale läßt sich aus dem Maschinenbefehl ableiten. Beispielsweise können gleiche Operationen mit unterschiedlichen Operanden die gleiche Zustandssequenz benutzen.

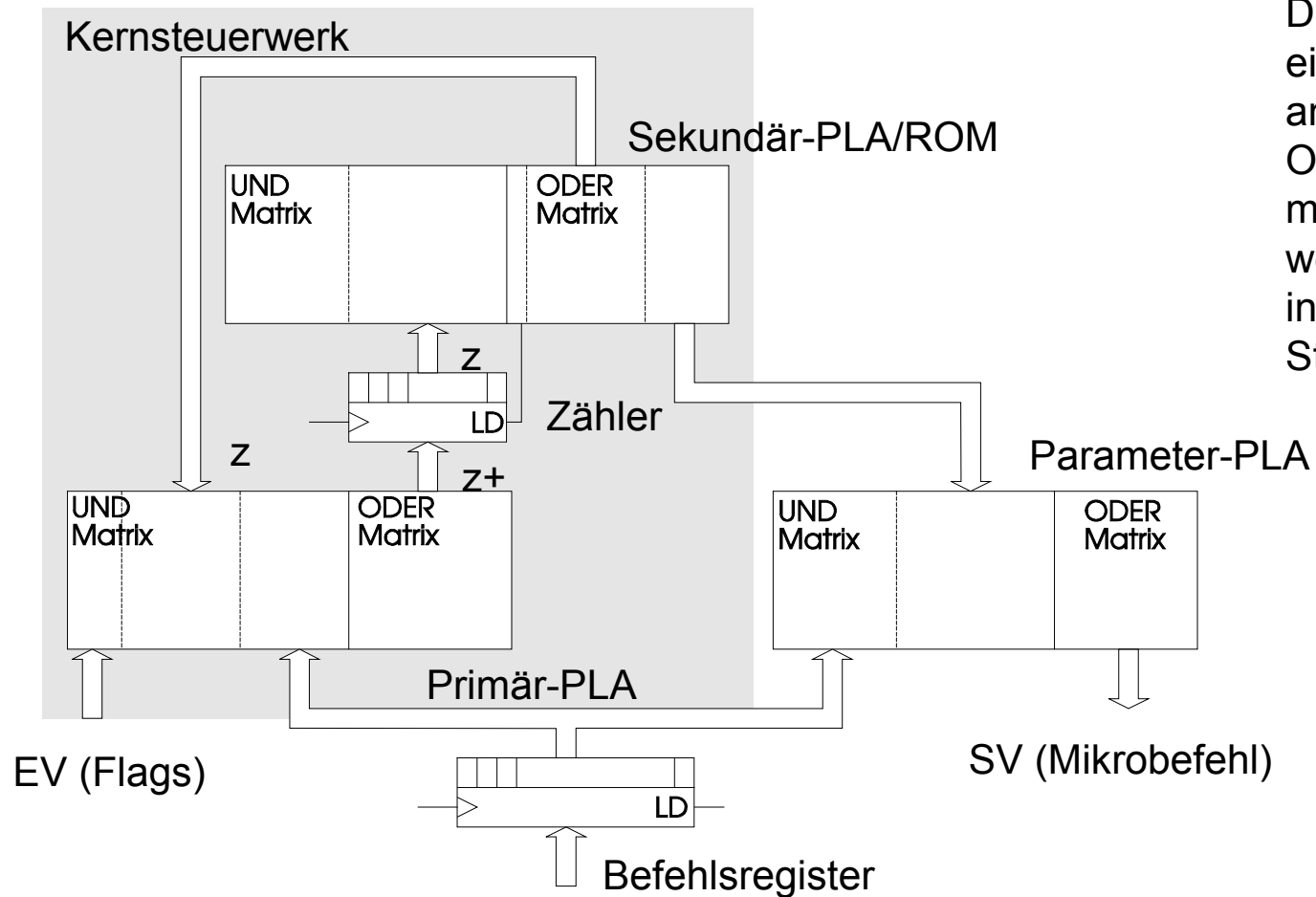
5.6 PLA-Steuerwerke

Steuerwerk mit Properties-PLA (Vorverarbeitung)



5.6 PLA-Steuerwerke

Steuerwerk mit Nachverarbeitung (Parameter-PLA)



Die während der Abarbeitung eines Befehls konstant anliegenden "Parameter" (z.B. Operationscodes für Befehle mit gleichen Operanden) werden direkt aus dem Befehl in entsprechende Steuersignale gewandelt.

5.6 PLA-Steuerwerke

Moderne Prozessoren mit Mikroprogramm-Steuerwerk haben meist sowohl Vor- als auch Nachverarbeitungschaltnetze

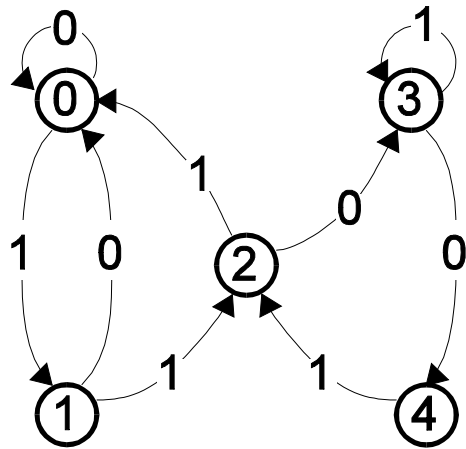
Hardwarestack

- Implementierung von Mikro-Unterprogrammen
Hierdurch können Sequenzen, die häufig in Befehlen vorkommen (z. B. Operanden holen) als Unterprogramm realisiert werden.
- **Vorteil:**
Minimierung der benötigten Einträge
- **Nachteil:**
Relativ großer Hardware-Aufwand und beschränkte Kellertiefe.
Daher wird heute üblicherweise nur wenige Kellereinträge realisiert (z. B. einer beim PIC)

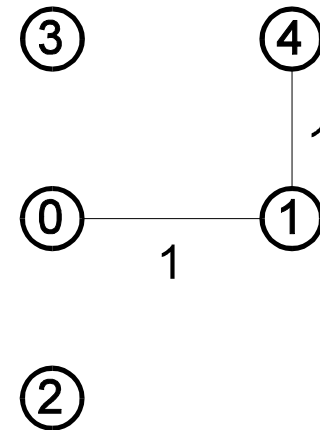
5.7 Zusammenfassung Steuerwerke

6 Zustandskodierung

4.1 Zustandskodierung nach [Armstrong 62]



	0	1
s0	s0	s1
s1	s0	s2
s2	s3	s0
s3	s4	s3
s4	-	s2



Graph zur Repräsentation der Bedingungen. Das Kantengewicht stellt die Anzahl dar.

Codierung

s1 = 100, s4 = 101, s2 = 010

Minimierung von

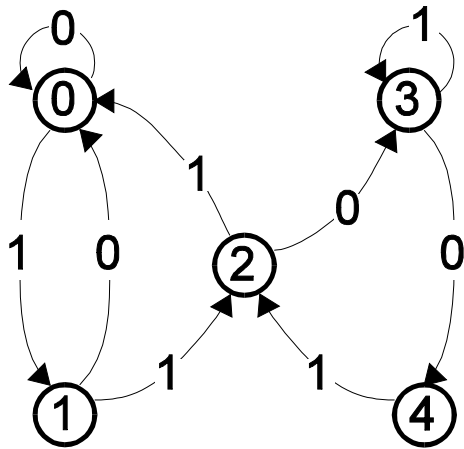
1 100 010

1 101 010 liefert 1 10- 010

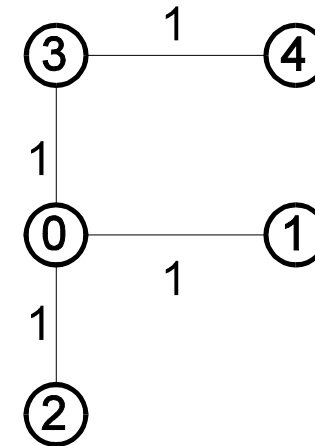
Alle Zustände mit gleicher Eingabe, gleichem Ausgabezustand und gleicher Ausgabe sollten in einem booleschen Unterraum codiert werden
(**Eingangscodierbedingung**)

6.1 Zustandskodierung nach Armstrong

Weitere Codierbedingungen



	0	1
s0	s0	s1
s1	s0	s2
s2	s3	s0
s3	s4	s3
s4	-	s2



Graph zur Repräsentation der Bedingungen. Das Kantengewicht stellt die Anzahl dar.

Codierung

s3 = 001, s4 = 101

Minimierung von

0 001 101 liefert 0 001 100

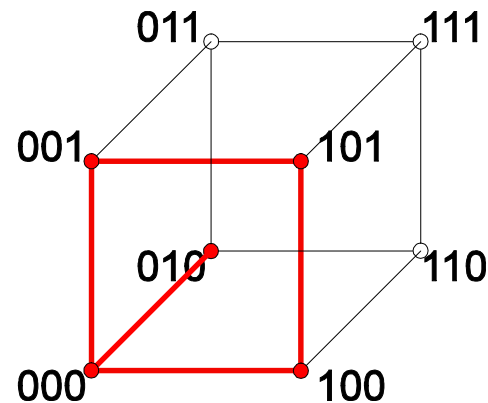
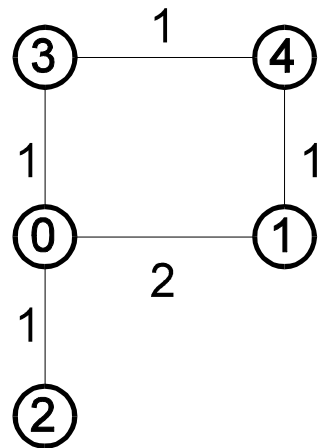
1 001 001 - 001 001

Vorteilhaft bei einer PAL-Implementierung

Alle Zustände mit verträglicher Eingabe und gleichem Eingabezustand sollten in einem booleschen Unterraum codiert werden.

6.1 Zustandskodierung nach Armstrong

Die Zusammenfassung der Graphen und Einbettung in einen Hypercube ergibt die Codierung



Eine Minimierung ergibt

EV	z	z+
0	010	001
1	000	100
1	10-	010
0	-01	100
-	001	001

EV	z	z+
0	000	000
1	000	100
0	100	000
1	100	010
0	010	001
1	010	000
0	001	101
1	001	001
0	101	---
1	101	010

6.2 Zustandskodierung (nach DeMicheli)

Probleme der symbolischen Minimierung [DeMicheli 86]

Gegeben sei eine Schaltfunktion mit symbolischen Einträgen. Die symbolischen Einträge können sowohl im Eingabeteil als auch im Ausgabeteil stehen. Es können vier verschiedene Probleme unterschieden werden:

- **Eingabecodierung** (P1)

Finde eine Codierung der **Eingänge**, welche die Kosten der Schaltnetzimplementierung minimiert. (minimale Codierung)

- **Ausgabecodierung** (P2)

Finde eine Codierung der **Ausgänge**, welche die Kosten der Schaltnetzimplementierung minimiert.

- **Ein- Ausgabecodierung** (P3)

Finde eine minimale Codierung der Ein- und Ausgänge.

- **Zustandskodierung** (P3)

Finde eine minimale Ein- Ausgabecodierung, wobei die Codes der (einiger)Ein- Ausgänge identisch sind. (später)

6.2 Zustandskodierung (nach DeMicheli)

Sei $A = \{ a_{ij} \}^{p \times q}$ und $X = \{ x_{ij} \}^{q \times r}$. Die boolesche Selektion $A \circ X = C = \{ c_{ij} \}^{p \times r}$
 mit $c_{ij} = \bigvee_{k=1}^q a_{ik} \wedge x_{kj}$

(Bem.: Es gelten die üblichen booleschen Rechenregeln)

Def.: Erfüllen von Ausgangscodierbedingungen

Sei $AC = \{ ac_{ij} \}^{ns \times ns}$ die Matrix der transitiven Hülle der Ordnungsrelation $>_S$, d. h. $ac_{ij} = 1$ falls $s_i >_S s_j$, 0 sonst, und C die Codematrix. Die Matrix $G = AC \circ C$

Eine Codierung C erfüllt die Ausgangscodierbedingungen falls gilt

$$C \wedge G = 0,$$

Satz.: Sei $EC = \{ ec_{ij} \}$ eine Matrix der Eingangscodierbedingungen und $>_S: S \times S$ eine partielle Ordnungsrelation auf den Symbolen (Ausgangscodierbed.). Es existiert eine Codierung C die beide Codierbedingungen erfüllt genau dann, wenn $\nexists s_r, s_s, s_t \in S$ mit $s_r >_S s_s$ und $s_s >_S s_t : \bigoplus k$ mit $ec_{kr} = 1$ $ec_{ks} = 0$ $ec_{kt} = 1$.

6.2 Zustandskodierung (nach DeMicheli)

Beweisskizze

$$EC = \begin{pmatrix} & r & & s & & t \\ \vdots & & \vdots & & \vdots & \\ \dots & 1 & \dots & 0 & \dots & 1 & \dots \\ \vdots & & \vdots & & \vdots & \end{pmatrix} k$$

$$J^0 = \{j \mid \tilde{c}_j = 0 \wedge \exists k : \tilde{c}_k = 1 \wedge s_k >_s s_j\}$$

$$J^1 = \{j \mid \tilde{c}_j = 0 \wedge \exists k : \tilde{c}_k = 1 \wedge s_j >_s s_k\}$$

$$\text{Falls } J^1 = \emptyset \text{ und } J^0 = \emptyset \rightarrow c_{\bullet n} = \tilde{c}$$

$$\text{Falls } J^1 = \emptyset \text{ und } J^0 \neq \emptyset \rightarrow c_{\bullet n} = \tilde{c}$$

$$\text{Falls } J^1 \neq \emptyset \text{ und } J^0 = \emptyset \rightarrow c_{\bullet n} = \bar{\tilde{c}}$$

$$\text{Falls } J^1 \neq \emptyset \text{ und } J^0 \neq \emptyset \rightarrow c_{jn} = t_j$$

$$s_r >_s s_s \text{ und } s_s >_s s_t$$

Fälle				
s_r	0	1	1	1
s_s	0	1	0	1
s_t	0	1	0	0
Spalte in Face-Matrix				
	0	1	-	-

$$t_j = \begin{cases} 01 & \forall j : \tilde{c}_j = 1 \\ 11 & \forall j \in J^1 \\ 00 & \forall j \in J^0 \\ \tilde{c}_j \tilde{c}_j & \text{sonst } (\tilde{c}_j \tilde{c}_j = 00) \end{cases}$$

6.2 Zustandskodierung (nach DeMicheli)

Algorithmus

Eingabe: Matrizen EC, AC

$n_b = 0$;

EC = clean(EC); // Identische Zeilen werden gewichtet und gelöscht. Zeilen aus
 // lauter Einsen oder mit nur einer Eins werden gelöscht

EC = compress(EC); // Löschen von nicht primen Zeilen

verify_constraints(EC,AC) // Existiert Codierung? (Modifikation der *
 Einträge)

do

 c = column_select

 if ($n_b = 0$) then C = c else C = [C|c] fi;

 nb = Anzahl Spalten in C

 EC = reduce_constraints(EC);

while (Codierung noch unvollständig)

6.2 Zustandskodierung

Beispiel

Ausgangstabelle (KISS-Format)

EV	Z	Z+	SV
0	start	state6	00
0	state2	state5	00
0	state3	state5	00
0	state4	state6	00
0	state5	start	10
0	state6	start	01
0	state7	state5	00
1	start	state4	00
1	state2	state3	00
1	state3	state7	00
1	state4	state6	00
1	state5	state2	10
1	state6	state2	01
1	state7	state6	00

1-aus-N Codierung

EV	Z	Z+	SV
0	1000000	0000010	00
0	0100000	0000100	00
0	0010000	0000100	00
0	0001000	0000010	00
0	0000100	1000000	10
0	0000010	1000000	01
0	0000001	0000100	00
1	1000000	0001000	00
1	0100000	0010000	00
1	0010000	0000001	00
1	0001000	0000010	00
1	0000100	0100000	10
1	0000010	0100000	01
1	0000001	0000010	00

6.2 Zustandskodierung

EV	Z	Z+	SV		EV	Z	Z+	SV
0	1000000	0000010	01		0	1001000	0000010	01
0	0100000	0000100	00		0	0110001	0000100	00
0	0010000	0000100	00					
0	0001000	0000010	01					
0	0000100	1000000	10		0	0000110	1000000	10
0	0000010	1000000	10					
0	0000001	0000100	00					
1	1000000	0001000	00		1	1000000	0001000	00
1	0100000	0010000	00		1	0100000	0010000	00
1	0010000	0000001	00		1	0010000	0000001	00
1	0001000	0000010	00		1	0001001	0000010	00
1	0000100	0100000	10		1	0000110	0100000	10
1	0000010	0100000	10					
1	0000001	0000010	00					
					-	0000100	0000000	10

Matrix EC der
Eingangscodier-
bedingungen

6.2 Zustandskodierung

EV	Z	Z+	SV
0	1001000	0000010	00
0	0110001	0000100	00
0	0000110	1000000	00
1	1000000	0001000	00
1	0100000	0010000	00
1	0010000	0000001	00
1	0001001	0000010	00
1	0000110	0100000	00
-	0000100	0000000	10

EV	Z	Z+	SV
-	1001001	0000010	00
0	0110001	0000100	00
0	0000110	1000000	00
1	1000000	0001000	00
1	0100000	0010000	00
1	0010000	0000001	00
1	0000110	0100000	00
-	0000100	0000000	10

Ausgangs-
codier-
bedingungen
state6 < state5
state6 < state4

6.2 Zustandskodierung (nach DeMicheli)

Reduktion der Matrix EC

$$EC = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$EC = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \begin{matrix} \text{Häufigkeit} \\ 1 \\ 1 \\ 2 \end{matrix}$$

Doppelte Zeilen werden eliminiert,
wobei die Häufigkeit sich gemerkt wird.

Zeilen aus lauter Einsen oder mit nur einer Eins
werden gestrichen.

Zeilen, die nicht prim sind, werden gelöscht.

6.2 Zustandskodierung (nach DeMicheli)

Auswahl einer Codespalte c

$$EC = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \begin{matrix} 1 \\ 1 \\ 2 \end{matrix}$$

Eingangscodierbedingungen

Auswahl einer Codespalte $\tilde{c} = ec \oplus_k$ aus EC
nach:

- Anzahl der erfüllten Eingangscodierbedingungen
- Weitere Heuristiken

Berücksichtigung der Ausgangscodierbedingungen

$$J^0 = \{j \mid \tilde{c}_j = 0 \wedge \exists k : \tilde{c}_k = 1 \wedge s_k >_s s_j\}$$

$$J^1 = \{j \mid \tilde{c}_j = 0 \wedge \exists k : \tilde{c}_k = 1 \wedge s_j >_s s_k\}$$

$$\text{Falls } J^1 = \emptyset \text{ und } J^0 = \emptyset \rightarrow c_{\bullet n} = \tilde{c}$$

$$\text{Falls } J^1 = \emptyset \text{ und } J^0 \neq \emptyset \rightarrow c_{\bullet n} = \tilde{c}$$

$$\text{Falls } J^1 \neq \emptyset \text{ und } J^0 = \emptyset \rightarrow c_{\bullet n} = \bar{\tilde{c}}$$

$$\text{Falls } J^1 \neq \emptyset \text{ und } J^0 \neq \emptyset \rightarrow c_{jn} = t_j$$

$$t_j = \begin{cases} 01 & \forall j : \tilde{c}_j = 1 \\ 11 & \forall j \in J^1 \\ 00 & \forall j \in J^0 \\ \tilde{c}_j \tilde{c}_j & \text{sonst } (\tilde{c}_j \tilde{c}_j = 00) \end{cases}$$

6.2 Zustandskodierung (nach DeMicheli)

Reduktion der Codierbedingungen in EC

$$F = EC \circledast E$$

for i := 0 to $n_p - 1$ do

 for j := 1 to $n_s - 1$ do

 if ($ec_{ij} = 0$ und $f_i \circledast c_{i\bullet}$)

 then $ec_{ij} = \text{X}$; fi

 od;

od;

clean(EC)

$$EC = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

$$\tilde{c} = (0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0)^T$$

$$c_{1\bullet} = (1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1)^T$$

$$F = EC \bullet C = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

$$EC = \begin{pmatrix} 1 & 0 & 0 & 1 & * & * & 1 \\ 0 & 1 & 1 & 0 & * & * & 1 \\ * & * & * & * & 1 & 1 & * \end{pmatrix}$$

6.2 Zustandskodierung (nach DeMicheli)

$$EC = \begin{pmatrix} 1 & 0 & 0 & 1 & * & * & 1 \\ 0 & 1 & 1 & 0 & * & * & 1 \\ * & * & * & * & 1 & 1 & * \end{pmatrix}$$

$$\tilde{c} = (1 \ 0 \ 0 \ 1 \ * \ * \ 1)^T$$

$$c_{2\bullet} = (1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1)^T$$

$$F = EC \bullet C = \begin{pmatrix} 1 & 1 \\ 1 & * \\ 0 & * \end{pmatrix}$$

$$\tilde{c} = (0 \ 1 \ 1 \ 0 \ * \ * \ 1)^T$$

$$c_{3\bullet} = (0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1)^T$$

$$F = EC \bullet C = \begin{pmatrix} 1 & 1 & 0 \\ 1 & * & 1 \\ 0 & * & 0 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Identische Codes müssen durch das Hinzufügen einer weiteren Codespalte eindeutig gemacht werden

Zustandscodierung nach de Micheli

Zusammenfassung

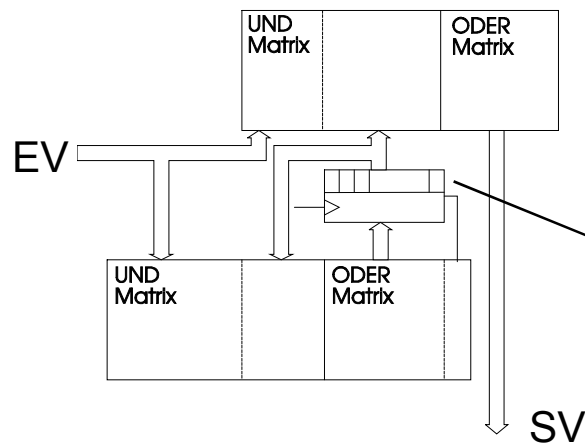
- **Verfahren für PLAs**
- **Spaltenweise Codierung**
- **Beachtung von Eingangs- und Ausgangscodierbedingungen**
- **Verwendung von Dont't Cares in den Eingangscodierbedingungen**
- **Kontrolle der Codelänge durch gezieltes Weglassen von Codierbedingungen**

6.3 Zustandscodierung für Zählersteuerwerke

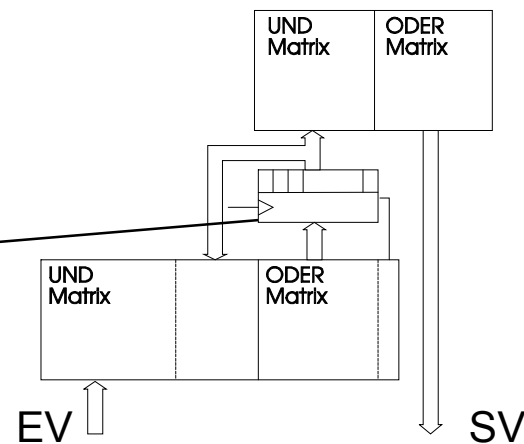
Realisierung von Steuerwerken mit einem Zähler

Idee: Bestimmte Zustandsübergänge werden durch den Zähler realisiert. Daher muß die Zustandsübergangsfunktion Δ nur für die restlichen Übergänge als Schaltnetz dargestellt werden.

Mealy-Automat



Moore-Automat

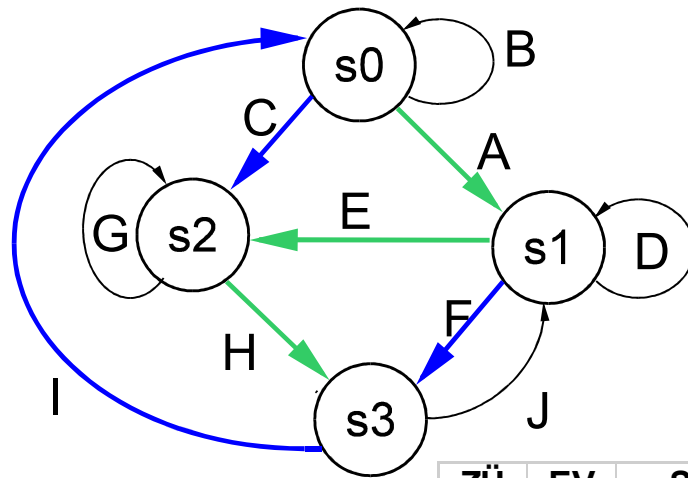


Schaltnetz für Ausgabefunktion kann beim Mealy-Automaten sehr groß werden

☞ Zunächst nur Moore-Automaten

6.3 Zustandscodierung für Zählersteuerwerke

Einleitendes Beispiel



ZÜ	EV	S	S+
A	00	s0	s1
B	01	s0	s0
C	1-	s0	s2
D	00	s1	s1
E	10	s1	s2
F	-1	s1	s3
G	-0	s2	s2
H	-1	s2	s3
I	-0	s3	s0
J	-1	s3	s1

Minimierte Tabelle

ZÜ	EV	S	S+
A,D	00	s0, s1	s1
B	01	s0	s0
C	1-	s0	s2
E	10	s1	s2
F,H	-1	s1, s2	s3
G	-0	s2	s2
I	-0	s3	s0
J	-1	s3	s1

ZÜ	EV	S	S+	Ü
A	00	s0	-	0
B	01	s0	s0	1
C	1-	s0	s2	1
D	00	s1	s1	1
E	10	s1	-	0
F	-1	s1	s3	1
G	-0	s2	s2	1
H	-1	s2	-	0
I	-0	s3	s0	1
J	-1	s3	s1	1

ZÜ	EV	S	S+	Ü
A	00	s0	s1	1
B	01	s0	s0	1
C	1-	s0	-	0
D	00	s1	s1	1
E	10	s1	s2	1
F	-1	s1	-	0
G	-0	s2	s2	1
H	-1	s2	s2	1
I	-0	s3	-	0
J	-1	s3	s1	1

6.3 Zustandskodierung für Zählersteuerwerke

Bestimmung der Zählketten

Zwei Zustandsübergänge $T_1: s_i \rightarrow s_k$ und $T_2: s_n \rightarrow s_m$ heißen **verträglich**, falls gilt

$B_1: s_i \rightarrow s_n, s_k \rightarrow s_m, s_k \rightarrow s_n$ und $s_i \rightarrow s_m$, oder,

$B_2: s_i \rightarrow s_m$ und $s_k = s_n$

Bestimmung von maximalen Mengen von paarweise verträglichen Zuständen

☞ Überdeckungsproblem (NP-hart)

Weitere Randbedingungen

- Auswahl der Zählübergänge so, daß möglichst viele Eingangscodierbedingungen erhalten bleiben.
- Unklar ist ob es günstig ist, ob man möglichst lange oder jeweils nur einzelne (kurze) Zählketten bestimmt werden.

Die Generierung der Zählketten kann durch eine Abschätzung der möglichen Reduktion der Terme gesteuert werden.

6.3 Zustandskodierung für Zählersteuerwerke

Schätzen der eingesparten Terme

Sei $A = \{ a_{ij} \}^{n_s \times n_s}$ die Adjazenzmatrix mit $a_{ij} = n$, falls ein Zustandsübergang $s_i \rightarrow s_j$ im Automaten n -Mal enthalten ist, und $EC' = \{ ec'_{ij} \}$ die Matrix der Eingangscodierbedingungen in folgender Darstellung ec'_{ij} ist = 1 falls eine Codierbedingung der Form $\{ s_i, s_k \dots \} \rightarrow s_j$ in der Tabelle enthalten ist.

Sei $P_{\max} = P_Z + P_{EC}$ die obere Schranke für die maximal möglichen eingesparten Terme.

P_{EC} , die Anzahl der eingesparten Terme durch Eingangscodierbedingung, kann nach einer symbolischen Minimierung abgeschätzt werden. Es wird davon ausgegangen, daß alle Eingangscodierbedingungen eingehalten werden können.

Abschätzung von P_Z , der durch die Zählerrealisierung eingesparten Terme:

Zunächst wird die Matrix $R = A - EC$ berechnet (Einträge < 0 werden zu 0 gesetzt)

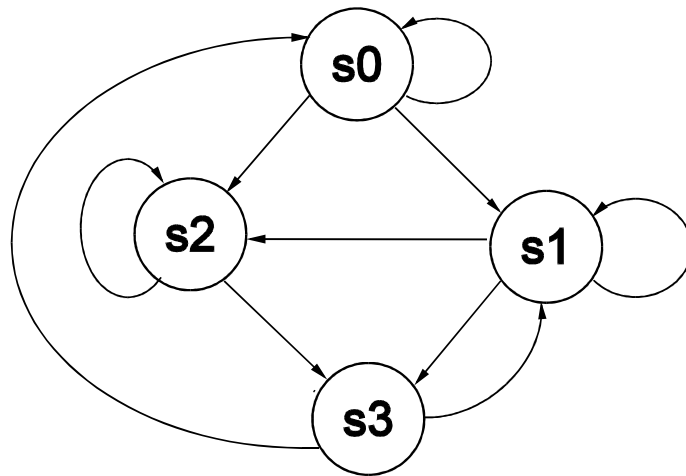
Die Anzahl der eingesparten Terme kann durch

$$P_Z = \sum_{k=1}^{n_s} \max_{j=1}^{n_s} (r_{jk})$$

abgeschätzt werden.

6.3 Zustandskodierung für Zählersteuerwerke

Beispiel



Einvorgängerkonflikte: Die Transitionen $s_3 \rightarrow s_0$ und $s_3 \rightarrow s_1$ lassen sich nicht gleichzeitig in einem Zähler unterbringen. Dies sog.

Einvorgängerkonflikte lassen sich dadurch erkennen, daß mehrere Spalten in R in einer Zeile nur genau einen Eintrag haben. Die Einträge in diesen Spalten müssen dann bis auf den größten zu Null gesetzt werden. Es ergibt sich dann $P_Z = 2$.

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} \quad EC' = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad R = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \quad P_Z = 3$$

Die gleichen Betrachtungen lassen sich auch für Nachfolger anstellen.

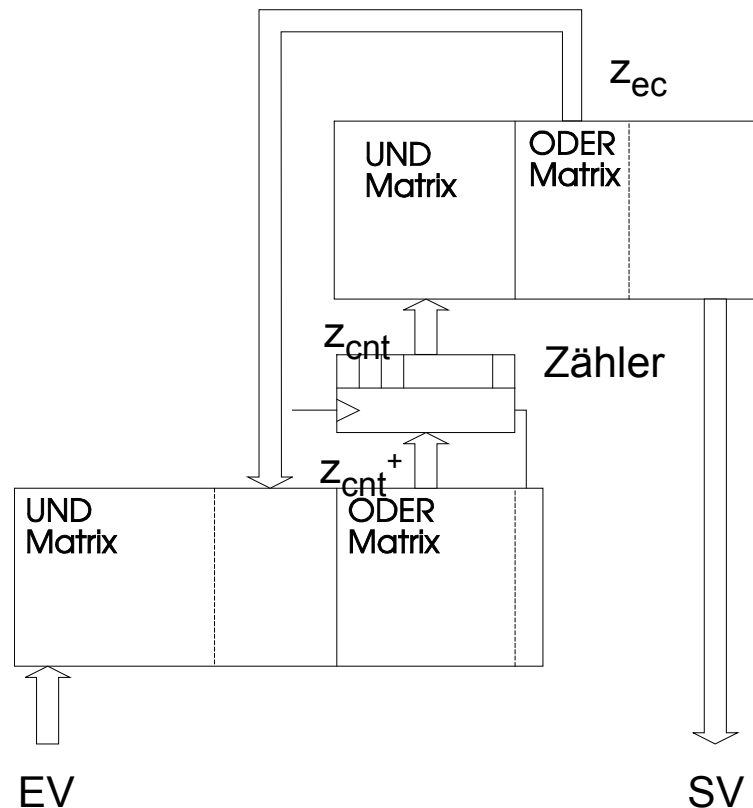
6.3 Zustandskodierung für Zählersteuerwerke

Kombinierte Codierung für Eingangscodierbedingungen und Zählübergänge

- **Probleme**
- **Codierung der Zustände**

6.3 Zustandskodierung für Zählersteuerwerke

Architektur zur Erfüllung von Eingangscodierbedingungen mit doppelter Codierung



Hierbei ist z_{ec} der Zustandscode zur Erfüllung der Eingangscodierbedingungen und z_{cnt} der Zustandscode für eine Zählerimplementierung

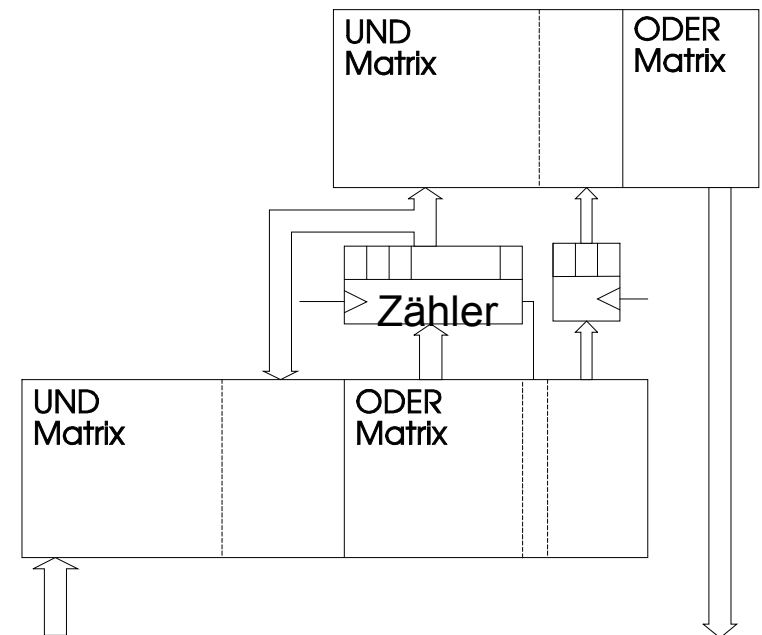
Durch die Zählerrealisierung von Zuständen können einige Zustände kompatibel werden. Daher lassen sich diese zu einem Zustand zusammenfassen (Zustandsreduktion). Das kann dazu führen, daß für z_{ec} weniger Zustandsbits benötigt werden.

6.3 Zustandskodierung für Zählersteuerwerke

Lokale Moore/Mealy-Umwandlung

Für jeden Zustandsübergang mit gleichem Nachfolgezustand aber unverträglicher Steuerbelegung wird ein "Auswahlcode" bestimmt. Die Anzahl von Codebits n_a für den Auswahlcode ist für ein gegebenes Steuerwerk konstant $n_a = \lceil \log_2 t_{\max} \rceil$.
 □. Wobei t_{\max} die maximale Anzahl von unverträglichen Steuerbelegungen für Nachfolgezuständen ist. (In der Praxis ist n_a relativ klein ● 3)

Steuerwerksarchitektur für eine lokale Mealy-Moore-Umwandlung



6.3 Zustandskodierung für Zählersteuerwerke

Auswahl der Art der Zählers [Amann]

- **Johnson-Zähler**

Vorteil: Geringer zusätzlicher Aufwand gegenüber von D-Flipflops

Nachteil: Nur wenige kurze Zählzyklen

– Der Einsatz von Johnson-Zählern erscheint vor allem bei kleineren Steuerwerken, eventuell als Einfach-PLA-Lösung sinnvoll

- **Binärzähler**

Vorteil: Der Zählzyklus umfaßt den gesamten booleschen Raum

Nachteil: Großer zusätzlicher Aufwand

– Binärzähler sollten vor allen in großen Steuerwerken mit Doppel-PLA-Lösung und evtl. doppelter Codierung eingesetzt werden.

- **MISR-Register**

Vorteil: Bei geeigneter Wahl des Polynoms besitzt er die gleiche Anzahl von Zuständen. Die Struktur kann im Selbsttest der Schaltung eingesetzt werden.

Nachteil: Sehr großer zusätzlicher Aufwand.

7 Retiming

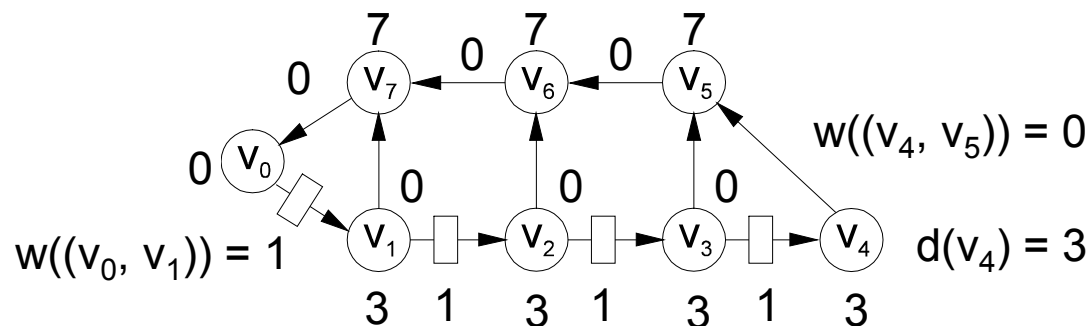
5.1 Strukturelles Retiming [Leieron, Saxe 91]

Einschränkungen

- D-Flipflops, die bei jeder Taktflanke geladen werden
- Alle kombinatorischen Elemente haben eine feste Verzögerungszeit

Modellierung der Schaltung durch einen gerichteten Graphen $CG = (V, E, d, w)$

- $V = \{v_i\}$ jeder Knoten v_i repräsentiert einen kombinatorischen Schaltungsteil.
- $E = \{e_k \mid e_k = (v_i, v_j)\}$ Der Ausgang des Schaltungsteils v_i ist mit einem Eingang des Schaltungsteils v_j verbunden.
- $d: V \rightarrow \mathbb{R}$ $d(v_i)$ ist die Verzögerung des Schaltungsteils v_i .
- $w: E \rightarrow \mathbb{N}$ $w(e_k)$ ist die Anzahl der DFF auf der Kante e_k .



7.1 Strukturelles Retiming

Ein **Pfad** $p = (v_0, v_1, \dots, v_k)$ ist eine Folge von Knoten die paarweise mit einer Kante verbunden sind. Ein einfacher Pfad enthält einen Knoten nur einmal. Wir benötigen zwei Größen:

Pfadgewicht

$$w(p) = \sum_{i=0}^{k-1} w(e_i)$$

Pfadverzögerung

$$d(p) = \sum_{i=0}^k d(v_i)$$

Bedingungen für eine sinnvolle physikalische Realisierung:

- B1: $\forall v_i \in V$ gilt $d(v_i) \geq 0$ keine negative Verzögerung
- B2: $\forall e_k \in E$ gilt $w(e_k) \geq 0$

Da wir nur synchrone Schaltungen betrachten wollen, soll ferner gelten:

- B3: Auf jedem Zyklus z gilt $w(z) \geq 1$ (mindestens ein Register)

7.1 Strukturelles Retiming

Berechnung der minimalen Taktperiode $\tau(CG)$

1. Sei CG_0 der Subgraph von CG , der alle Kanten e_k mit $w(e_k) = 0$ enthält.
2. Da nach B2 CG_0 azyklisch ist liefert eine topologische Sortierung eine totale Ordnung auf CG_0 .
3. Berechne in der Reihenfolge der Sortierung die Größe $D(v)$ durch
 Falls v keine eingehenden Kanten in CG_0 hat ist $\tau(v) = d(v)$
 sonst ist $\tau(v) = d(v) + \max\{\tau(u) \mid (u, v) \in E_0\}$
4. Die minimale Taktperiode $\tau(CG)$ ist $\max_v \tau(v)$.

7.1 Strukturelles Retiming

Es soll nun ein Algorithmus gefunden werden, der die Register so in der Schaltung "verschiebt", daß die minimale Taktperiode möglichst klein wird. Genauer: Es wird eine Funktion $r: V \rightarrow \mathbb{Z}$, das **Retiming**, bestimmt. Die neuen Kantengewichte $w_r((u,v))$ berechnen sich dann zu:

$$w_r((u, v)) = w((u,v)) + r(v) - r(u)$$

Das Pfadgewicht $w(p)$ mit $p = (u, \dots, v)$ berechnet sich dann zu

$$w_r(p) = w(p) + r(u) - r(v)$$

Insbesondere gilt für jeden Zyklus z : $w_r(z) = w(z)$

Für ein **legales** Retiming muß gelten:

$$w_r(e_k) \geq 0 \text{ (B2)}$$

Bem: B3 gilt

7.1 Strukturelles Retiming

Charakterisierung der minimalen Takperiode

$W(u,v) = \min\{ w(p) \mid p = (u, \dots, v) \}$ (minimale Anzahl von Registern zw. u und v)

$D(u, v) = \max\{ d(p) \mid p = (u, \dots, v) \}$

Es gilt nun: $\exists (CG) \bullet c \in \mathbb{N} \wedge \forall u,v \in V : D(u, v) > c \Rightarrow W(u, v) \geq 1$

Berechnung von $W(u,v)$ und $D(u,v)$

1. Markiere alle Kanten (u, v) mit $(w(e), -d(u))$
2. Berechne kürzeste Pfade (Vergleich durch Gewichte über lexikographische Ord.)
3. Sei (x, y) das Gewicht auf dem kürzesten Pfad von u nach v, setze
 $W(u, v) = x$ und $D(u, v) = d(v) - y$.

Für ein Retiming berechnen sich die Größen zu

$$W_r(u,v) = W(u, v) + r(v) - r(u)$$

$$D_r(u, v) = D(u, v)$$

7.1 Strukturelles Retiming

Ein Retiming $r: V \rightarrow \mathbb{Z}$ ist genau dann legal mit der minimalen Taktperiode c

☞ (CG) $\Leftrightarrow c$, falls $r(u) - r(v) \leq w((u,v))$ und $r(u) - r(v) \leq W((u,v)) - 1$ für alle $D((u,v)) > c$

Bem.: Die kleinste Taktperiode muß gleich $D(u,v)$ für ein Paar (u,v) sein.

Algorithmus zur Berechnung eines Retimings mit minimaler Taktperiode

1. Berechne W und D
2. Sortiere die Elemente in nach D
3. Bestimme durch Intervallschachtelung (binäre Suche in D) die minimale Taktperiode. Hierzu müssen mit Hilfe des Bellman-Ford-Algorithmus obige Bedingungen überprüft und ein Retiming gefunden werden.
4. Verwende das in 3 gefundene Retiming

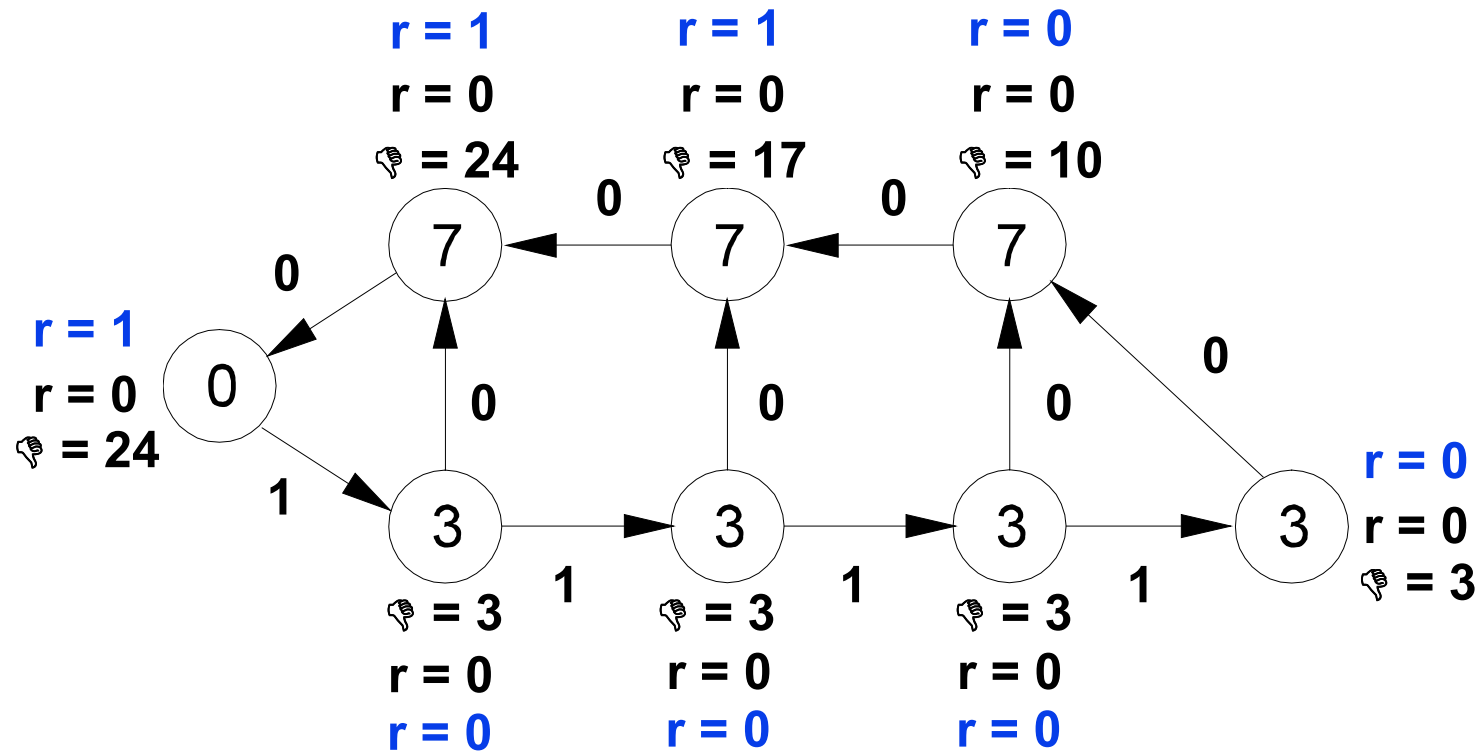
7.1 Strukturelles Retiming

Verbesserter Algorithmus

1. For all $v \in V$ do $r(v) := 0$;
2. $CG^0 = CG$
3. for $i = 1$ to $|V| - 1$ do
 - 3.1 $CG^i := \text{Retiming}(CG^0)$
 - 3.1 Berechne $\delta(v)$ für CG^i (nach dem vorherigen Algorithmus)
 - 3.3 For all $v \in V$ do
 - 3.3.1 if $\delta(v) > c$ then $r(v) := r(v) + 1$;
- 4 If $\delta(CG_r) > c$ then Taktperiode zu kurz

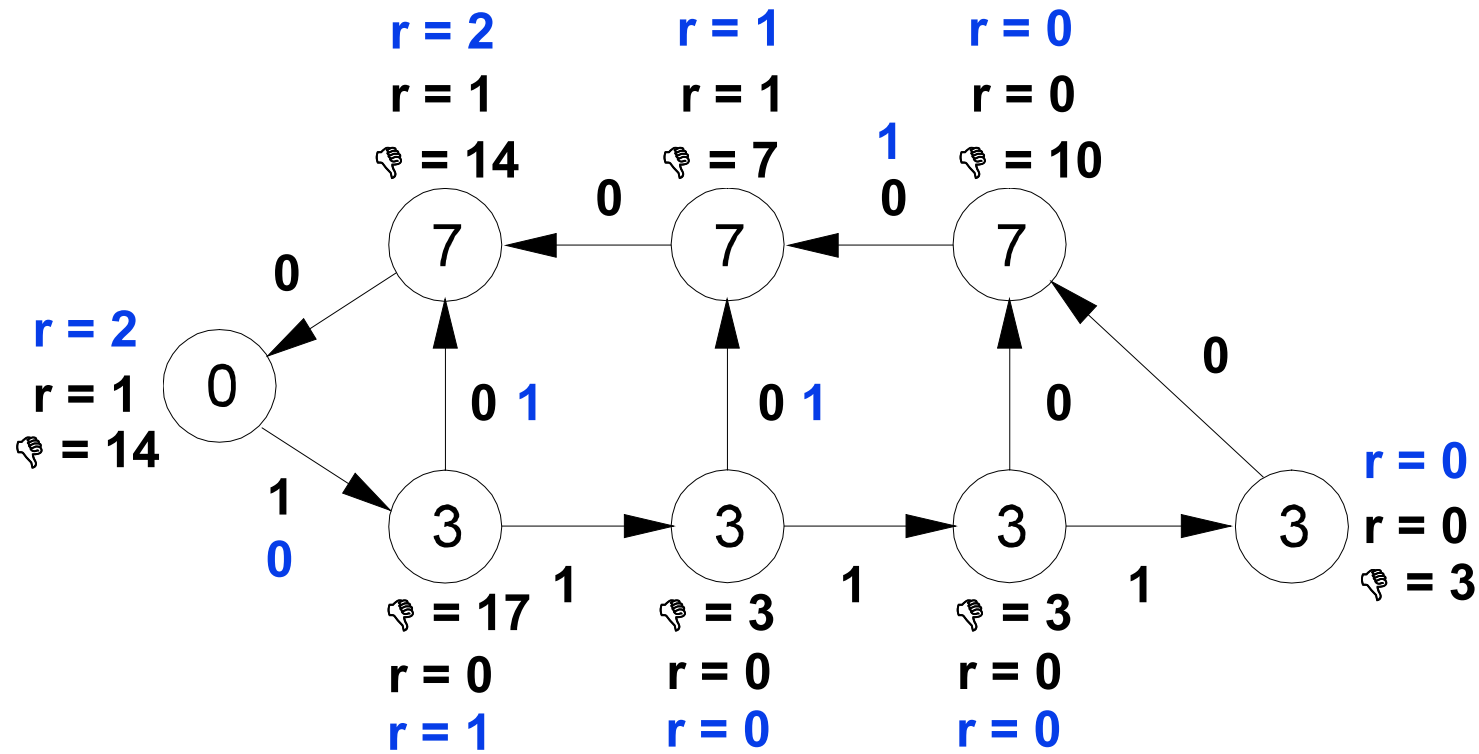
7.1 Strukturelles Retiming

Beispiel ($c = 13$)



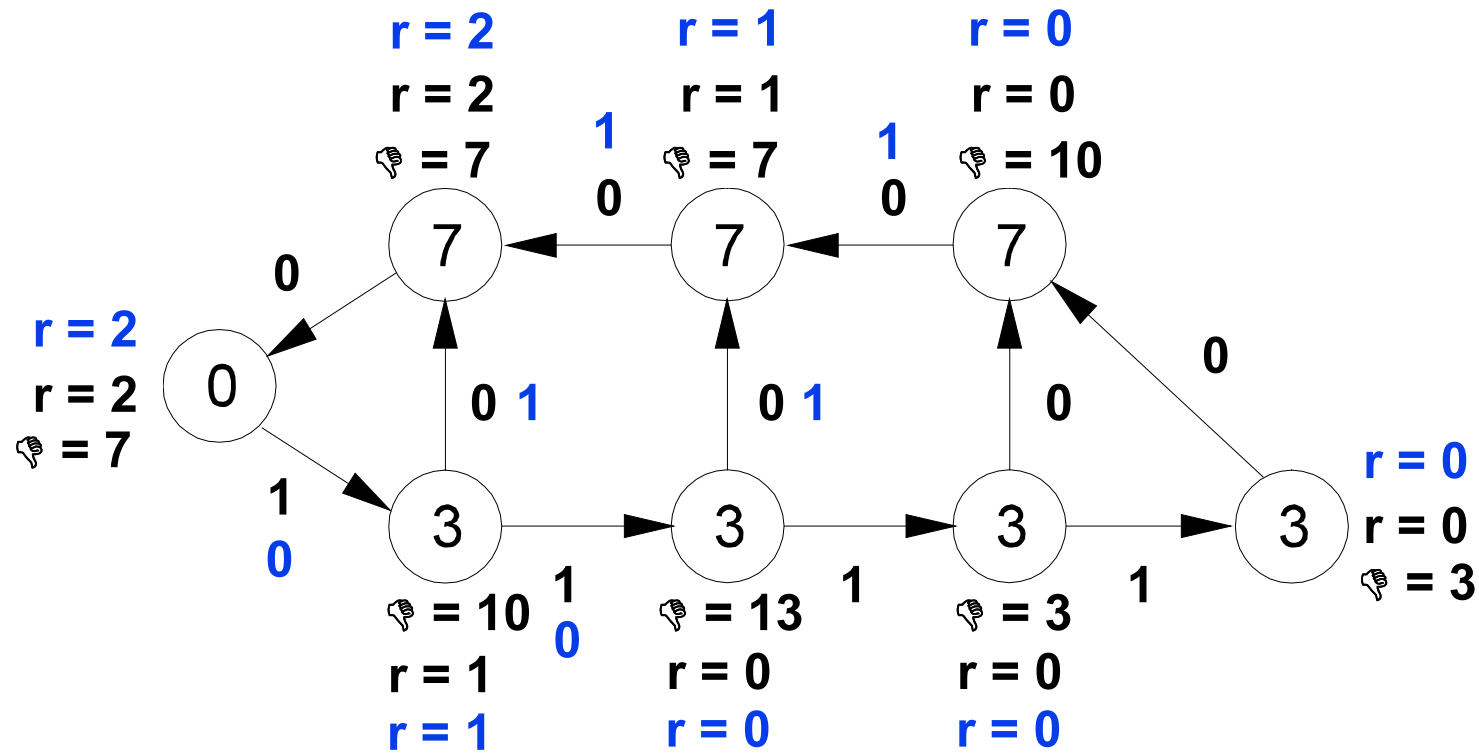
7.1 Strukturelles Retiming

Beispiel ($c = 13$)



7.1 Strukturelles Retiming

Beispiel ($c = 13$)

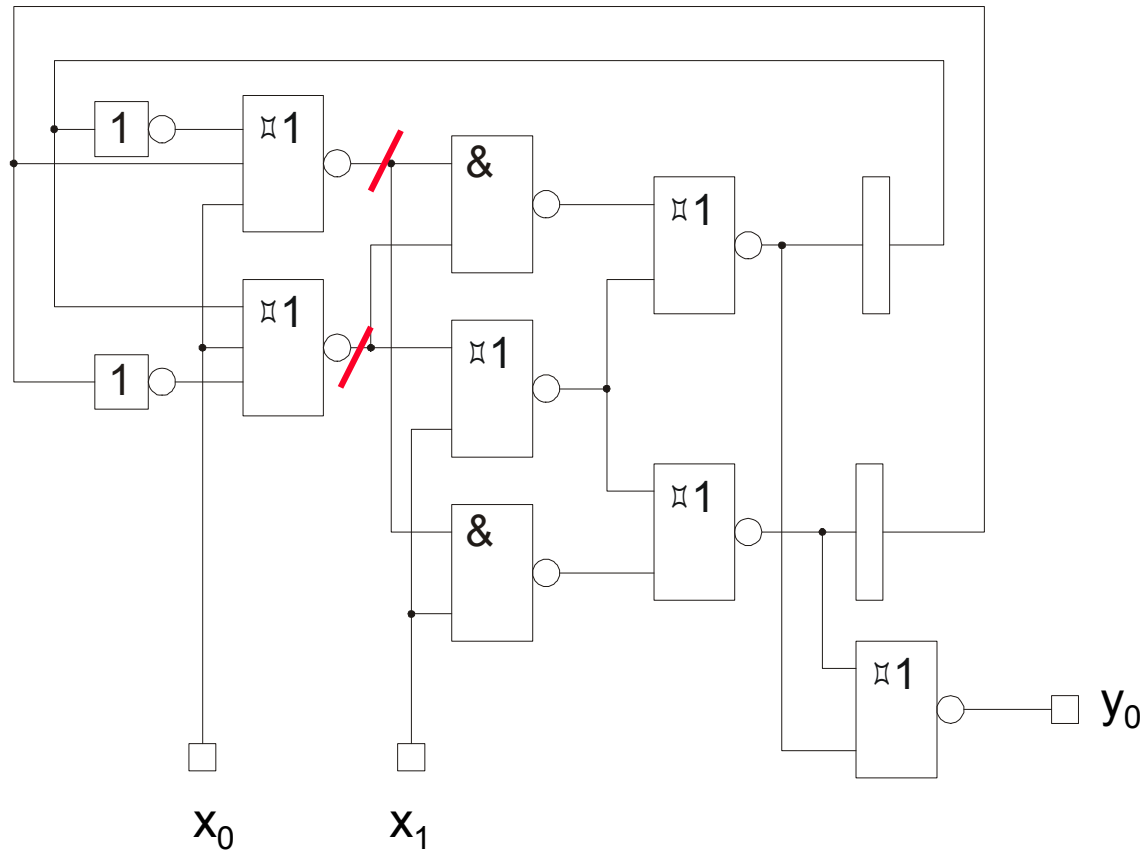


7.1 Strukturelles Retiming

- **Erweiterung für nichtüberlappenden Zweiphasentakt und Latches als Speicherelemente**

7,2 Funktionales Retiming

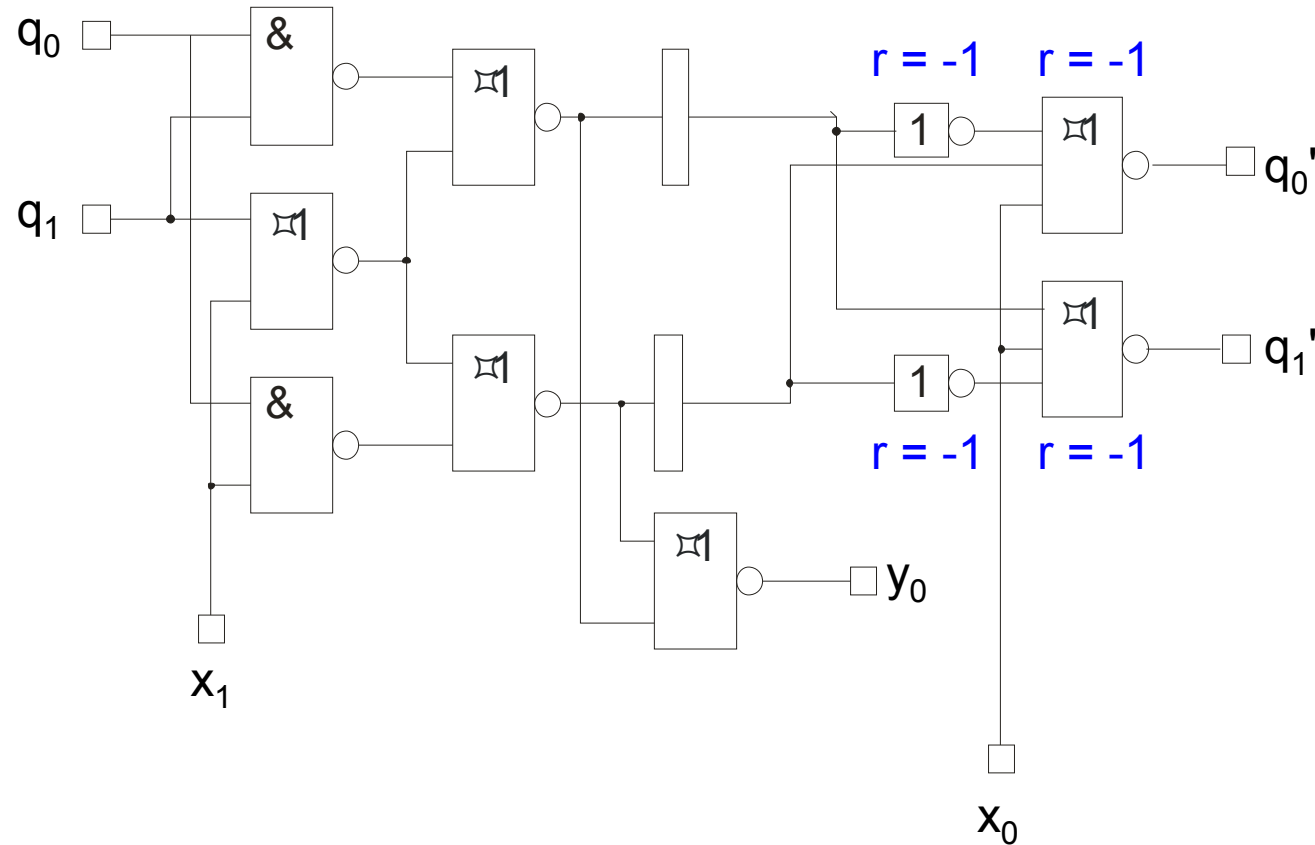
Grundsätzliche Idee



Auftrennen der
Rückkopplungen

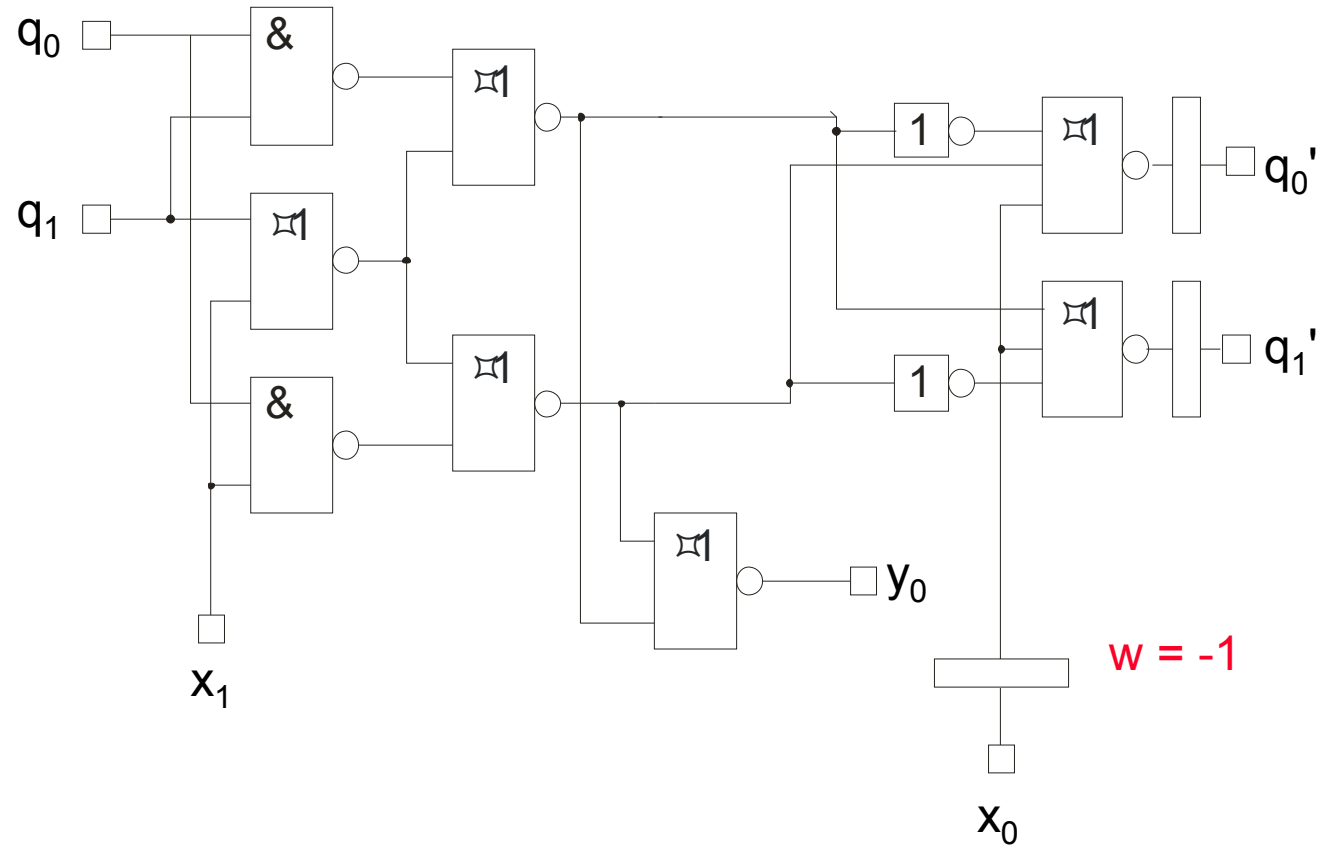
7,2 Funktionales Retiming

- Alle Register werden an die Ein-, Ausgänge verlegt (Peripheral retiming)



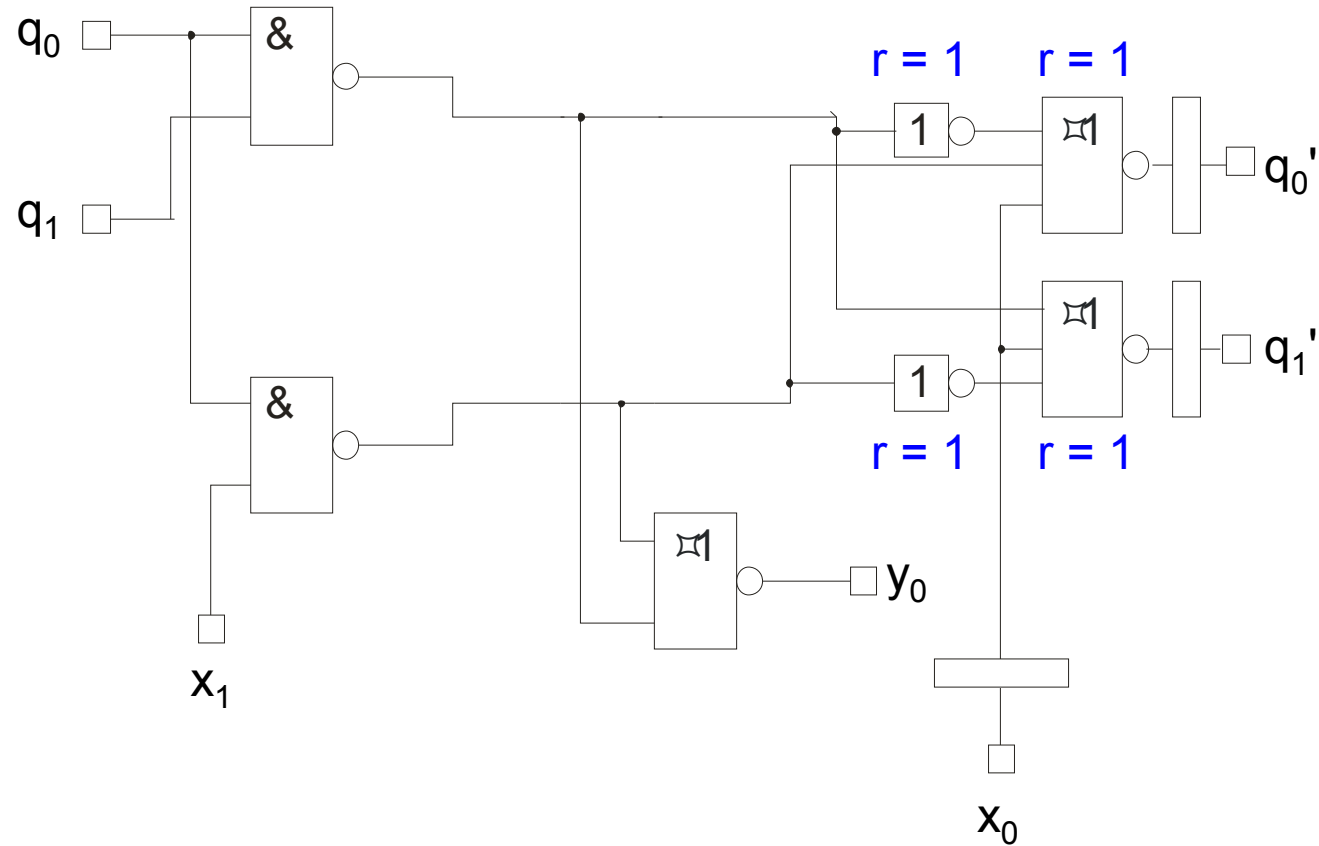
7,2 Funktionales Retiming

Durch das Retiming soll es zulässig sein, dass im Laufe des Verfahrens an den Eingängen negative Register entstehen

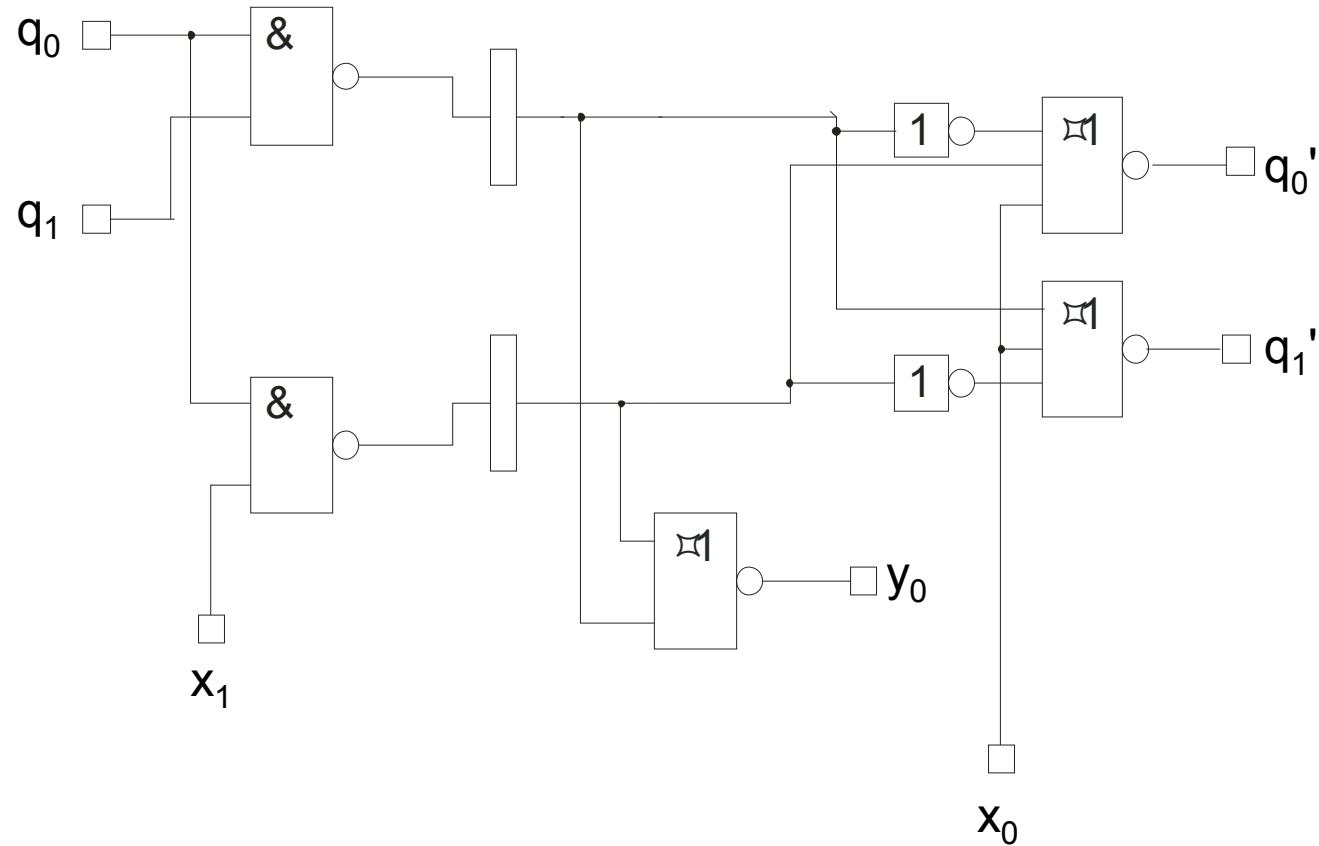


7,2 Funktionales Retiming

- Minimierung der Logik



7,2 Funktionales Retiming



7,2 Funktionales Retiming

8 Taktung von Schaltungen

Die Taktung von Schaltungen richtet sich nach dem Typ der Speicherelemente

- **flankengetriggerte Flipflops**

Ein Zustandswechsel der Schaltung erfolgt bei definierten Taktflanken.

Es wird meist ein Einphasentakt verwendet

- **phasengesteuerte Latches**

Es soll hier nur der sogenannte „level sensitive“-Entwurfstil untersucht werden.

Hierbei erfolgt die Taktung der Latches durch einen nicht überlappenden Zweiphasentakt.

- ☞ **Eine Mischung der Taktschemata sollte vermieden werden. Dies führt meist zu einem völlig unübersichtlichen Entwurf dessen Zeitverhalten nur sehr schwer nachvollziehbar ist.**

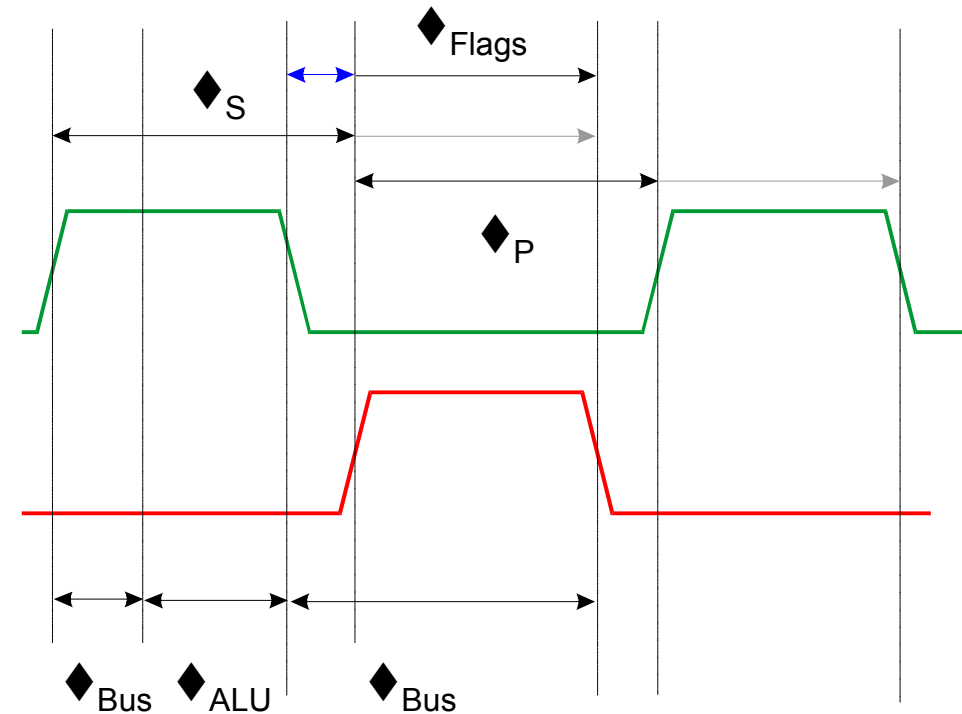
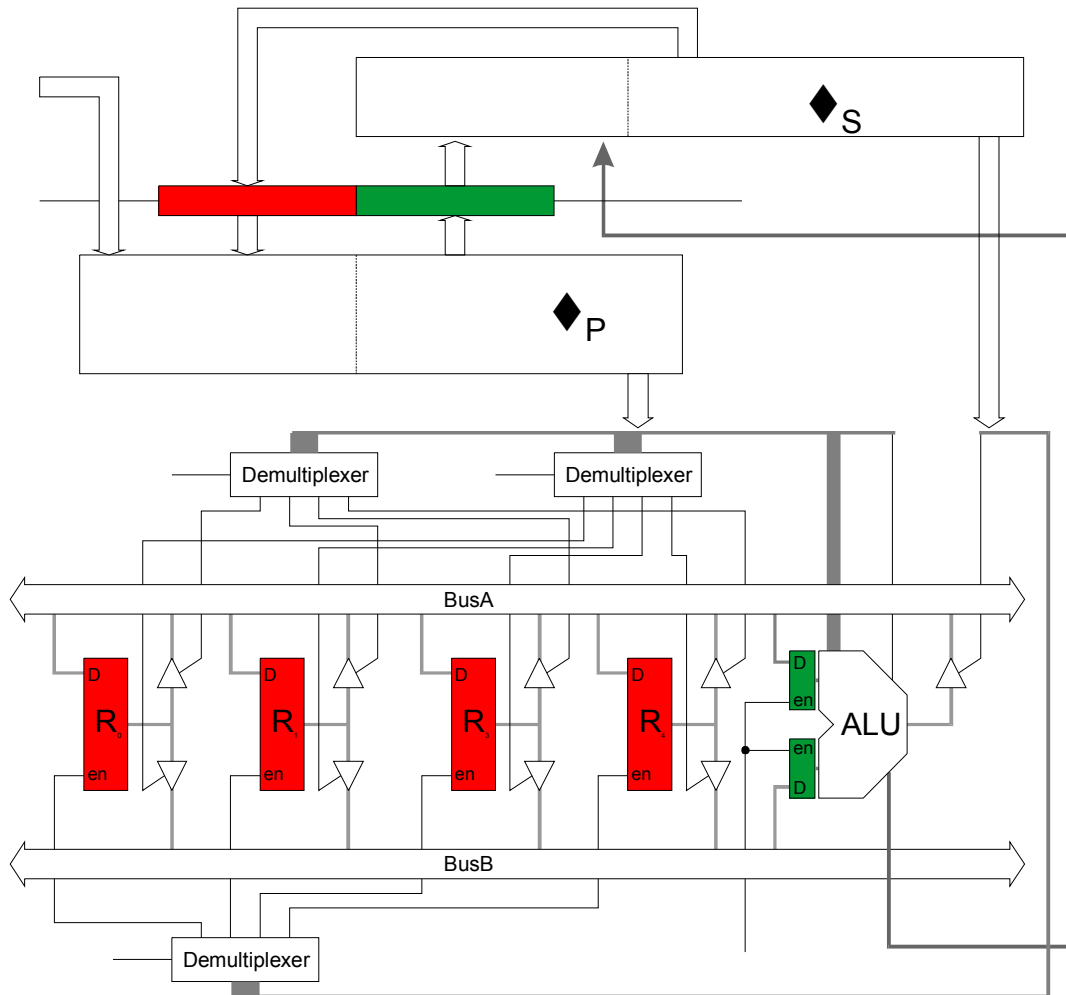
Beim Entwurf von integrierten Schaltungen führt eine Mischung der Schemata zu Problemen beim ✱ Test integrierter Schaltungen

8.1 Flankengetriggerte Flipflops

3-Bus-Struktur

8.2 Phasengesteuerte Latches

Zwei-Bus-Struktur



Die blau eingezeichnete, verkürzte Verarbeitungszeit $\blacklozenge_{\text{Flags}}$ ist nur notwendig, falls das Laden der Register in einem Befehl von Flags abhängig ist. Ansonsten kann $\blacklozenge_{\text{ALU}}$ und $\blacklozenge_{\text{Flags}}$ verlängert werden

9 Asynchrone Schaltungen

SIC single input change

MIC multiple output change

Funktionshazards

Statische Hazards

- statischer 1-Hazard
- statischer 0-Hazard

Bsp.:

Satz: Eine

Dynamische Hazards

9 Asynchrone Schaltungen

Der Zustandsübergangsgraph (STG)

USC (Unique state code) Bedingung

CSC (Conforming state Code) Bedingung

9.1 Zustandskodierung

Wettläufe (Races)

STT single transition time

Normale Flußtabelle

SOC single output change

MOC

9.1 Zustands-codedierung

Liu-Algorithmus

9.1 Zustandskodierung

Der Algorithmus nach [Tracy 66]

Beschränkung auf normale Flußtabellen (SOC-Bedingung)

Def.: Die **n-Partition** p einer Menge S (von Zuständen) ist eine Liste von Teilmengen, den **Blöcken**, so daß jeder paarweise Durchschnitt von zwei Teilmengen leer ist. Eine **Zweierpartition** $\Pi = \{\Pi^0, \Pi^1\}$ ist eine Partition mit zwei Blöcken.

Def.: eine Partition **teilt** zwei Elemente $s_1, s_2 \in S$, falls $s_1 \in \Pi^0_i$ und $s_2 \in \Pi^1_i$ oder $s_1 \in \Pi^1_i$ und $s_2 \in \Pi^0_i$

Ziel.: Finde (eine minimale Anzahl von) Zweierpartitionen $\mathcal{P} = \{\Pi_1, \Pi_2, \dots, \Pi_{nq}\}$ mit der Eigenschaft: für jeden Übergang mit dem Eingangsprodukt $c_{ab} \stackrel{\Delta}{=} (c_{ab}, s_a) \star s_b$, $a \in \Pi_i$ gilt:

$\exists s_k$ mit $c_{km} \in c_{ab}$ $\exists \Pi_j \stackrel{\Delta}{=} (c_{km}, s_k) \star s_m$ mit $b \in \Pi_j$ $\exists \Pi_i : \Pi_i$ teilt s_m und s_b .

9.1 Zustandskodierung

Algorithmus

Beispiel für Simplify

- $F =$
- 112120 220020
- 021212
- 121221
- 121002
- 220021
- 121012
- $Fb = 121221022122222201220021220020122121122120$ $Fb =$
- $Fb = 121221022122222201220021220020122121122120$ $Fbf =$
 $121222022122222202220022122122$ $Fbf = 022122220022122122$
- $Fbf = 121222022122222202220022122122$ $Fbfa =$
 221222222202220022222122 $Fbfa = 222122222202220022$
- $Fbfa = 221222222202220022222122$ $Fbfac = 222222222202222122$
 $= 222222$ $Fbfac = 222202222022222121$

- $Fbfac = 222202222022222121$ $Fbfacd = 222202222222 = 222222$



3.2.1 Definitionen

Def.: **Monotone Funktionen**

Eine boolsche Funktion F heißt **monoton steigend** (**monoton fallend**) in einer Variablen x_i , wenn bei einer Änderung von x_i von 0 nach 1 alle sich ändernde Ausgänge von 0 nach 1 steigen (von 1 nach 0 fallen). Eine boolsche Funktion heißt **monoton in x_i** , falls sie entweder monoton steigend oder monoton fallend in x_i ist. Eine boolsche Funktion heißt **monoton**, wenn sie in allen Variablen monoton ist.

Eine Überdeckung C ist monoton steigend (fallend) in der Variablen in x_i , wenn jeder Cube entweder eine 1 (eine 0) oder eine 2 an der i -ten Position besitzt.

Wenn die Überdeckung $C(F)$ einer boolschen Funktion monoton ist, so ist auch F monoton. (nicht notwendigerweise umgekehrt)

Satz 3.1 : Eine boolsche Funktion F ist monoton steigend (fallend) in x_i , genau dann wenn kein Primimplikant von F eine 0 (1) in der i -ten Position hat.

Satz 3.2: Eine monotone Überdeckung C ist eine Tautologie, genau dann wenn sie einen Cube aus lauter Zweien besitzt.

3.2.1 Monotone Funktionen

Komplementbildung von monotonen Funktionen

Satz 3.3: Sei F eine monotone Funktion, so kann das Komplement dargestellt werden als

$$\overline{F} = \overline{x_i} \overline{F_{\overline{x_i}}} \vee \overline{F_{x_i}} \quad \text{falls } F \text{ in } x_i \text{ monoton steigend ist, oder als}$$

$$\overline{F} = x_i \overline{F_{x_i}} \vee \overline{F_{\overline{x_i}}} \quad \text{falls } F \text{ in } x_i \text{ monoton fallend ist.}$$