8 Modelle der Switching-Ebene

Aufbauend auf Kapitel 7 wird im hier beginnenden Kapitel die Modellierung von Potential-Logik behandelt. Im letzten Abschnitt wird gezeigt, wie eine Hardware-Beschreibungssprache (KARL-3 [5]) eingesetzt werden kann, um Schaltungen in der Switching-Ebene durch "Pseudo-Switching-Level"-Beschreibungen zu modellieren und zu simulieren.

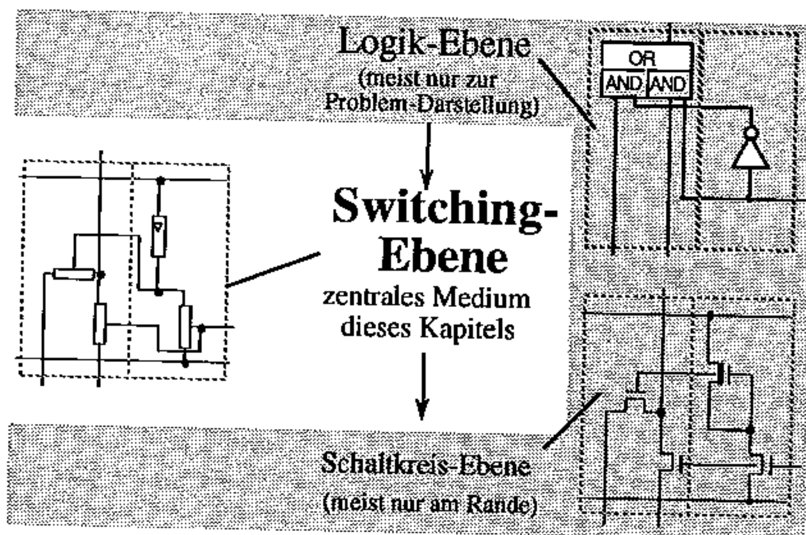


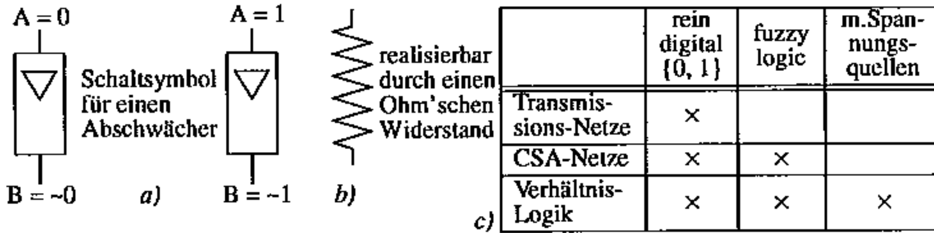
Bild 8.1 Die von diesem Kapitel betroffenen Abstraktionsebenen

8.1 CSA-Modelle und Niveau-Logik (Potential-Logik)

Bisher (im vorigen Kapitel) haben wir nur Schalternetze aufgrund ihrer Transmission charakterisiert mit einem Wertevorrat v_T = {on, off}, bzw. {durchlässig, undurchlässig}, weshalb wir

8.1 CSA-Modelle und Niveau-Logik (Potential-Logik) 159
8.1.1 CSA-Modellierung von Verhältnis-Logik 161
8.2 Logische Schaltkreisfamilien 163
8.2.1 Synthese logischer Transistor-Schaltungen 166
8.3 Schalternetz-Optimierung 167
8.4 Modellierung in der Switching-Ebene mit KARL-3 170
8.6 Literatur 177

© Reiner W. Hartenstein, 76646 Bruchsal, Germany 1988 - 1993



	rein digital {0, 1}	fuzzy logic	m.Spannungs-quellen
Transmissions-Netze	×		
CSA-Netze	×	×	
Verhältnis-Logik	×	×	×

Bild 8.2: Switching-Ebenen: a) Abschwächer-Symbol, b) Realisierung v. (a), c) Wertebereiche.

von Transmissions-Logik sprachen. Wir erhalten neue Kategorien von Netzen (vgl. Bild 8.2 c), wenn wir neben abstrakten Schaltern weitere Elemente wie Widerstände und Spannungsquellen zulassen: CSA-Netze [1][8][9], Potential-Logik und Verhältnis-Logik (Bild 8.3 gibt einen Überblick).

CSA-Netze. Durch die Einführung von Abschwächern (vgl. Bild 8.2 a/b, engl.: *attenuators*, wie etwa durch Widerstände realisierbar), die in reinen T-Netzwerken (vgl. Kapitel 7) nicht vorkommen, erhalten wir einen erweiterten Wertevorrat, der über v_T = {on, off} hinausgeht, wie beispielsweise v_T = {stark durchlässig, schwach durchlässig, sehr schwach durchlässig, undurchlässig}. Algebraisch gesehen kommen wir damit in Bereiche der *fuzzy logic* [2][3][10]. Die dabei auftretenden Netze werden meist *CSA-Netzwerke* genannt, wobei CSA für "Connector, Switch, Attenuator" steht.

Potential-Logik (Niveau-Logik). Durch Einführung von Spannungsquellen kommen wir von der Transmissionslogik (ohne Abschwächer) zu einer dritten Kategorie von Logik: nämlich der *Niveau-Logik* (oder *Potential-Logik*, auch *Verhältnis-Logik* genannt), bei der das statische Niveau N eines Signales (z. B. das Spannungs-Niveau in Volt, wenn es sich um ein elektrisches

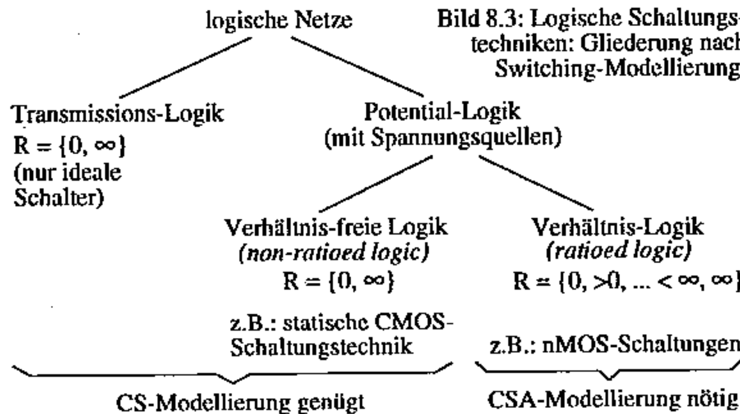
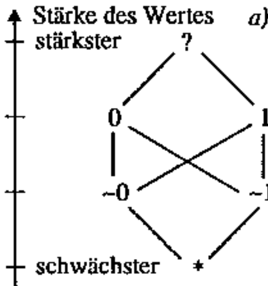
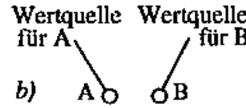


Bild 8.3: Logische Schaltungstechniken: Gliederung nach Switching-Modellierung.





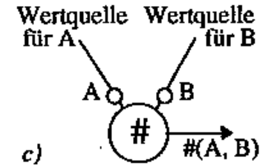


Bild 8.4: a) Hasse-Diagramm der Sprache KARL-3, b) und c) der join-Operator.

Signal handelt) relevant ist mit dem Wertevorrat v_N = {low, high} der Realisierung logischer Werte dienend, wobei 'low' etwa die logische '0' realisiert, und 'high' die logische '1'.

Verhältnis-Logik ***(ratioed logic).*** Durch die Kombination von CSA-Logik mit Potential-Logik erhalten wir einer vierte Art der Logik, nämlich Verhältnis-Logik. Da neben reinen Schalterzuständen entsprechend dem Wertevorrat R = 0 ("kein Widerstand") und R = ∞ ("undurchlässig") noch Zwischenwerte vorkommen (wie "niedriger Widerstand" oder "hoher Widerstand"), kommen in einem solchen Netz oft Knoten vor, deren Potential weder "high", noch "low" ist, also einen Zwischenwert einnimmt. Bei solchen Netzen ist die Wahl richtiger Widerstandsverhältnisse wichtig. Bei statischer (komplementärer) CMOS-Schaltungstechnik (vgl. Kapitel 16) liegt nur Potentiallogik vor, da keine Lastwiderstände verwendet werden und somit nur die Werte "on" und "off" vorkommen. Bei nMOS-Schaltungstechnik hingegen (vgl. Kapitel 11) liegt Verhältnislogik vor, weshalb hier auf Inverterverhältnisse (Widerstands-Verhältnisse) geachtet werden muß.

8.1.1 CSA-Modellierung von Verhältnis-Logik

Wir wenden uns der Switching-Modellierung von Verhältnislogik zu (wie z.B. bei nMOS-Schaltungstechnik). Es kommen zum Wertevorrat R = 0 und R = ∞ je nach Zahl der unterschiedlichen Abschwächer-Typen noch ein oder mehrere Zwischenwert(e) hinzu.Solche Netze sind wegen des Vorkommens von Widerständen keine reinen Schalternetze mehr. Wir müssen daher zu CSA-Modellierung übergehen (vgl. Abschnitt 8.2).

Wir werden es im folgenden Abschnitt zwischen R = 0 und R = ∞ bei einem einzigen Zwischenwert bewenden lassen (z. B. R = 1 kΩ), somit kämen wir auf den Widerstandswerte-Vorrat v_R mit: v_R = {0, 1kΩ, ∞}. Da die Potentialwerte 0 und 1 (geliefert von Spannungsquellen "high" und "low") auch über Abschwächer laufen können, erhalten wir zusätzlich zur harten 0 und zur harten 1 noch die weiche Null ~0 und die weiche Eins ~1. Wird ein Knoten mit keinem Wert beliefert (etwa wegen eines Schalters im Zustand "off"), so hat er den "Wahrheitswert" "hochohmig" oder "don't care", angegeben durch das Symbol *. Außerdem ist es zweckmäßig, ein Symbol "?" für "undefiniert" zu haben. Unser Wertevorrat an Wahrheitswerten (Potentialen) in der CSA-Ebene ergibt sich damit zu v={0, 1, ~0, ~1, *, ?}.

Der Konnekt-Operator. Eine wesentliche Grundlage der CSA-Modellierung bildet die Einführung des Verbindungsoperators '#', den wir auch "join"-Operator oder "Konnekt"-Operator

nennen wollen. Zwei oder mehr Knoten (z.B. A und B in Bild 8.4 b) werden durch den Konnekt-Operator einfach miteinander kurzgeschlossen (Bild 8.4 c). Durch den Kurzschluß zwischen A und B stellt sich ein resultierender Wert #(A, B) ein. Das Hasse-Diagramm definiert, welcher Wert #(A,B) dabei entsteht. Für A=0 und B= ~1 ergibt z.B. sich #(A,B)=0.

Für zwei Quellen (A und B, vgl. Bild 8.5 a) allgemein ergibt sich für das Resultat #(A,B) gemäß Hasse-Diagramm in Bild 8.4 a. Die Tabelle in Bild 8.5 b zeigt die Auflösungsfunktion (auch *resolution function* genannt) tabellarisch, aus welcher sich das Hasse-Diagramm ergibt. Bevor wir nun speziell auf die Modellierung von Verhältnis-Logik eingehen, demonstrieren wir noch einmal kurz die Wirkung eines Abschwächers anhand des Beispiels in Bild 8.6. Die Abschwächer bewirken, daß an den Knoten a und n eine *schwache Eins* bzw. an den Knoten b und o eine *schwache Null* anliegt. Der Eingabevektor X(x1,x2,x3,x4,x5)=01110 zur Steuerung der Schalter hat dann die Wirkung:

Quelle A	Quelle B	#(A,B)
0	~0	0
0	0	0
0	~1	0
0	1	?
0	?	?
0	*	0
1	~0	1
1	~1	1
1	1	1
1	?	?
1	*	1
~0	~0	~0
~0	~1	?
~0	?	?
~0	*	~0
~1	~1	~1
~1	?	?
~1	*	~1
?	?	?
?	*	?
*	*	*

b)

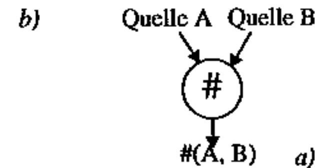


Bild 8.5: Verknüpfung zweier Leitungen durch den Konnekt-Operator (a) und Ergebnis am Ausgang (b).

Der Knoten c ist hochohmig (*), da der Schalter mit x1 als Eingang nicht durchschaltet. Knoten d hat den Wert '1' und am Knoten e liegt eine schwache Null an. Die Zusammenschaltung dieser drei Knoten bewirkt eine '1' am Knoten f. Am Knoten k liegt eine schwache Eins an, der Knoten l ist hochohmig und der Knoten m hat den Wert '0'. Die Zusammenschaltung dieser drei Knoten bewirkt eine '0' am Knoten i.

Da x4=1 ist, werden die Werte der Knoten f und i an die Knoten g und h weitergegeben. Die Funktion #(g, h) ergibt einen Widerspruch, da der Wert von g=1 und h=0 ist. Somit ist die Funktion F(x1,x2,x3,x4,x5) für den Eingabevektor X = 01110 undefiniert. In Bild 8.7 sind alle Ergebnisse dieser Funktion für alle x-Kombinationen in Tabellenform aufgelistet.

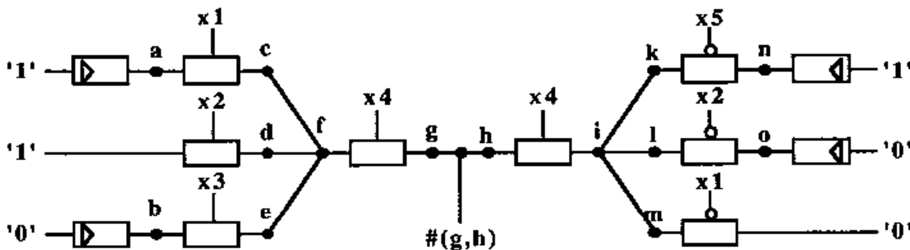


Bild 8.6: Schalternetz mit Abschwächern dargestellt in der CSA-Ebene

x1	x2	x3	x4	x5	c	d	e	f	k	l	m	i	g	h	#(g,h)	F(x1,...x5)
0	0	0	0	0	*	*	*	*	~1	~0	0	0	*	*	*	*
0	0	0	0	1	*	*	*	*	*	~0	0	0	*	*	*	*
0	0	0	1	0	*	*	*	*	~1	~0	0	0	*	0	0	false
0	0	0	1	1	*	*	*	*	*	~0	0	0	*	0	0	false
0	0	1	0	0	*	*	~0	~0	~1	~0	0	0	*	*	*	*
0	0	1	0	1	*	*	~0	~0	*	~0	0	0	*	*	*	*
0	0	1	1	0	*	*	~0	~0	~1	~0	0	0	~0	0	0	false
0	0	1	1	1	*	*	~0	~0	*	~0	0	0	~0	0	0	false
0	1	0	0	0	*	1	*	1	~1	*	0	0	*	*	*	*
0	1	0	0	1	*	1	*	1	*	*	0	0	*	*	*	*
0	1	0	1	0	*	1	*	1	~1	*	0	0	1	0	?	undef.
0	1	0	1	1	*	1	*	1	*	*	0	0	1	0	?	undef.
0	1	1	0	0	*	1	~0	1	~1	*	0	0	*	*	*	*
0	1	1	0	1	*	1	~0	1	*	*	0	0	*	*	*	*
0	1	1	1	0	*	1	~0	1	~1	*	0	0	1	0	?	undef.
0	1	1	1	1	*	1	~0	1	*	*	0	0	1	0	?	undef.
1	0	0	0	0	~1	*	*	~1	~1	~0	*	?	*	*	*	*
1	0	0	0	1	~1	*	*	~1	*	~0	*	~0	*	*	*	*
1	0	0	1	0	~1	*	*	~1	~1	~0	*	?	~1	?	?	undef.
1	0	0	1	1	~1	*	*	~1	*	~0	*	~0	~1	~0	?	undef.
1	0	1	0	0	~1	*	~0	?	~1	~0	*	?	*	*	*	*
1	0	1	0	1	~1	*	~0	?	*	~0	*	~0	*	*	*	*
1	0	1	1	0	~1	*	~0	?	~1	~0	*	?	?	?	?	undef.
1	0	1	1	1	~1	*	~0	?	*	~0	*	~0	?	~0	?	undef.
1	1	0	0	0	~1	1	*	1	~1	*	*	~1	*	*	*	*
1	1	0	0	1	~1	1	*	1	*	*	*	*	*	*	*	*
1	1	0	1	0	~1	1	*	1	~1	*	*	~1	1	~1	1	true
1	1	0	1	1	~1	1	*	1	*	*	*	*	1	*	1	true
1	1	1	0	0	~1	1	~0	1	~1	*	*	~1	*	*	*	*
1	1	1	0	1	~1	1	~0	1	*	*	*	*	*	*	*	*
1	1	1	1	0	~1	1	~0	1	~1	*	*	~1	1	~1	1	true
1	1	1	1	1	~1	1	~0	1	*	*	*	*	1	*	1	true

Bild 8.7: Funktionstabelle der Schaltung aus Bild 8.6.

8.2 Logische Schaltkreisfamilien

Logische (oder digitale) Schaltkreisfamilien und deren Entwicklung sind Gegenstand dieses Abschnitts. Eine bekannte Schaltungstechnik für die ***Verhältnislogik*** ist die nMOS-Technologie (siehe auch folgende Kapitel). Sie baut sich im wesentlichen als Serienschaltung wie folgt auf: "oben" mit Verbindung zu VDD (Versorgungsspannung) und dem Funktionsausgang f befindet sich ein Lasttransistor (*Pullup-Transistor*) und "unten" ein Transmissionsnetz (Bild 8.8). Die Funktion f der Schaltung hängt von der Transmissionsfunktion dieses Netzes ab. Ist die Transmissionsfunktion '1', d.h. das Netz ist stromdurchlässig und damit sein Widerstand Null, ist die Spannung am Gatterausgang 0 Volt und umgekehrt. Wenn wir davon ausgehen, daß high (5V) = '1' und low (0V) = '0', ergibt sich für f der Funktionswert f = not (T (x1, x2, ... , x_n)), d.h. es ergibt sich stets eine Negation. Diese Art von Logik hat ihre Vorzüge. Sie ist nämlich

© Reiner W. Hartenstein, 76646 Bruchsal, Germany 1988 - 1993

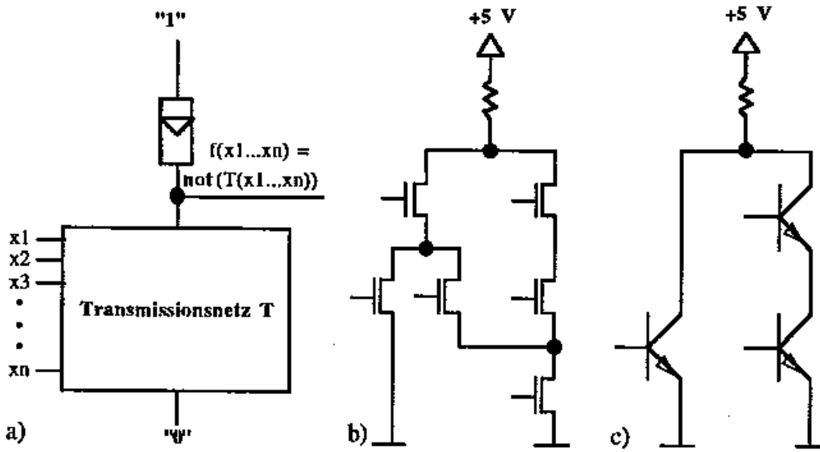


Bild 8.8: Prinzip invertierender Verhältnislogik (a) (z.B. NMOS (b), DCTL (c))

regenerativ, d.h. es werden Verstärkungseffekte wirksam. Die einfachst mögliche Transmissionsfunktion in dieser Schaltungstechnik hat der Inverter (Bild 8.9 a).

Die hier verwendete Logik bezeichnet man auch als *Pullup/Pulldown-Logik*.. Über den Lasttransistor (*Pullup-Transistor*) wird eine parasitäre Kapazität aufgeladen, die gegebenenfalls über das *pull-down-Netz* entladen wird (Abschnitt 12.2.2 behandelt mehr Einzelheiten), d.h. der Funktionswert wird durch diese Kapazität eine Weile gespeichert. In Bild 8.9 b und c sind dann noch eine **nand**- und eine **nor**-Schaltung in dieser Technologie dargestellt.

Warum sprechen wir bei der vorliegenden Technik von "Verhältnislogik"? Transistoren sind keine idealen Schalter. In der nMOS-Technologie hat der Transistor zwar einen in etwa idealen "off-Zustand" (Widerstand im Giga-Ohm-Bereich) jedoch keinen idealen "on-Zustand" (viele Kilo-Ohm). Es muß folgende elektrische Regel eingehalten werden: der Pullup-Widerstand

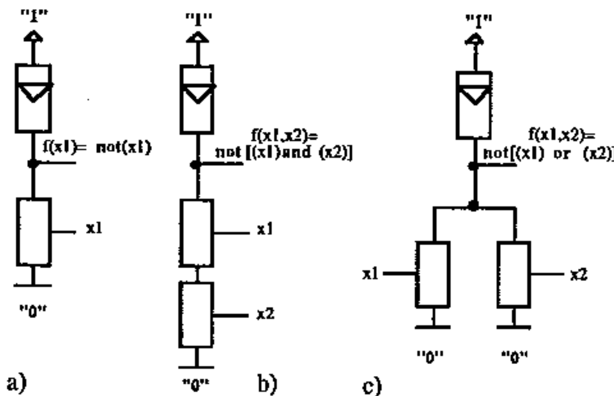


Bild 8.9: invertierende Gatter in Verhältnislogik in der Switch-Ebene dargestellt a) Inverter b) NAND-Schaltung c) NOR-Schaltung

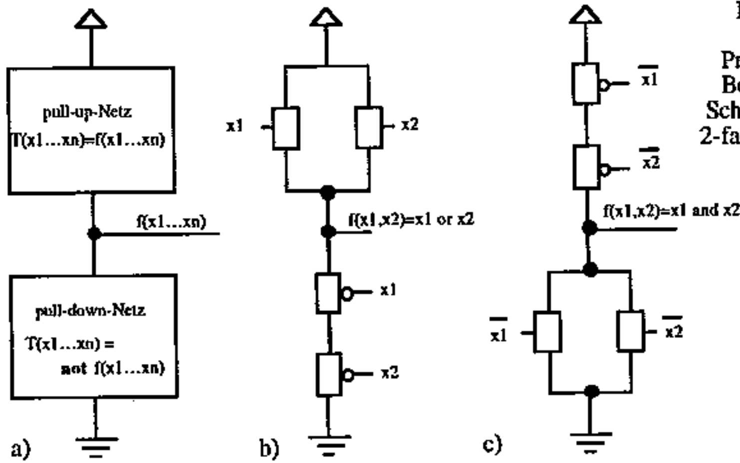


Bild 8.10: Komplementäre Logik a) Prinzip allgemein, b) Beispiel: 2-fach **OR**-Schaltung, c) Beispiel: 2-fach **AND**-Schaltung

muß so groß sein, daß beim "on-Transistor" eine genügend niedrige Ausgangsspannung (z.B. 1 Volt) erreicht wird (z.B. durch ein Mindest-Pullup/Pulldown-Verhältnis von 4/1). Deshalb wird diese Logik Verhältnislogik genannt (ratioed logic).

Komplementäre Logik Eine andere Schaltungstechnik für Logik ist die komplementäre Schaltungstechnik. Hier wird der Widerstand aus den vorangegangenen Schaltungen durch ein zweites zum pull-down-Netz duales Transmissionsnetz (*Pullup-Netz*) ersetzt, das die Funktion f direkt realisiert (vgl. Bild 8.10 a).

Hier muß darauf geachtet werden, daß die beiden Netze streng komplementär zueinander entworfen werden, d.h. daß immer genau ein Netz bei einer bestimmten Beschaltung durchlässig ist. Die bekannteste Technik bei der diese Logik verwendet wird ist die statische CMOS-Schaltungstechnik, auf die in Kapitel 16 näher eingegangen wird. Im allgemeinen verwendet sie für das Pullup-Netz negative Schalter und für das Pulldown-Netz positive. Ein Hauptvorteil dieser Logik ist es, daß man kein Verhältnis (Widerstandsverhältnis)

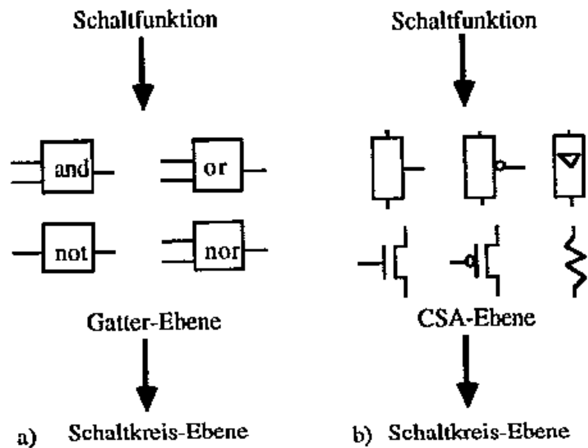


Bild 8.11: Alternativen des Schaltkreis-Entwurf; a) Klassische Methode, b) über die CSA-Ebene (Switching-Ebene)

© Reiner W. Hartenstein, 76646 Bruchsal, Germany 1988 - 1993

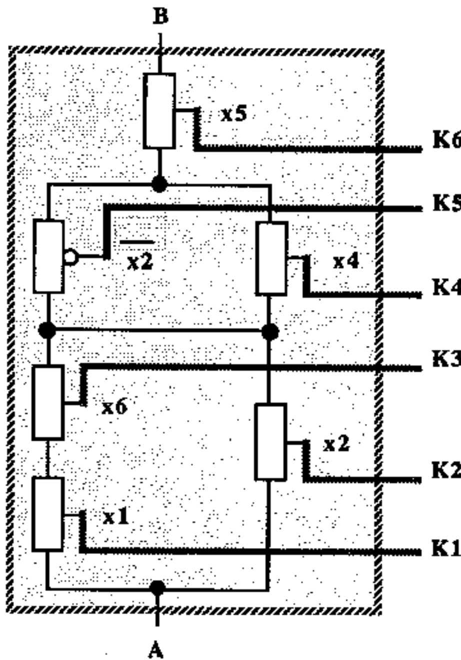


Bild 8.12: Zweipol mit Steuereingang $K_1 \ldots K_6$.

der beiden Netze zueinander beachten muß. Deshalb wird diese Logik auch als verhältnisfrei (ratio-less) bezeichnet.

8.2.1 Synthese logischer Transistor-Schaltungen

In diesem Abschnitt befassen wir uns mit der Synthese von einfachen und komplexen Logik-Gattern. Ausgehend von einer gegebenen Logikfunktion (Boole'scher Ausdruck) wird eine Transistorschaltung als Lösung des Problems gesucht. Dies ist eine Alternative zum klassischen Logik-Entwurf (vgl. Bild 8.11), die für die Mikroelektronik oft bessere Flächenersparnis erlaubt. Bis hierher haben wir eine Abstraktion betrieben. In Wirklichkeit müssen die Schalter betätigt werden, der Zweipol muß also irgendwelche Eingänge haben, durch die man die Schalter ansteuern kann. Wir brauchen dann Eingänge x_1 bis x_n, mit denen dann die Schalter betätigt werden, somit haben wir also gar keinen echten Zweipol. Im Beispiel in Bild 8.12 werden diese Eingänge mit K1 bis K6 bezeichnet.

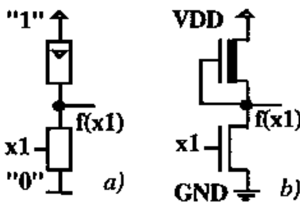


Bild 8.13: nMOS-Inverter, a) CSA-Ebene, b) Schaltkreisebene

Elementare Gatter (NOT, NAND, NOR) Wir bleiben bei der Verhältnislogik, d.h. der Widerstand gehört dazu. Dann ist die einfachst mögliche Transmissionsfunktion, ein einziger Schalter (vgl. Bild 8.13). Wir nennen den Steuereingang des Schalters x1. Wenn dies ein positiver Schalter ist, - positiver Schalter heißt: bei x1 = 1 ist der Schalter durchlässig -, bekommen wir als Funktion f die not-Funktion mit f = not(x1). Diese Schaltung ist der Inverter.

In Bild 8.14 haben wir eine **nand**-Schaltung mit zwei Eingängen. Das Gatter hat die Funktion f = **not**(x1 **and** x2). Der Ausgang f der Schaltung wird auf Null "gezogen", wenn beide Schalter durchlässig sind, d.h. die Signale x1 und x2 müssen den Wert '1' haben (Voraussetzung: positive Schalter). Sperrt ein Schalter oder auch beide, also x1=0 und/oder x2=0, dann liegt am Ausgang 5V an und somit ist der Funktionswert f(x1,x2)=1. Durch einfache Serienschaltung mehrerer Schalter bekommt man den entsprechend der Anzahl eine 3, 4, 5, ... -fach **nand**-Schaltung.

Entsprechendes gilt für die 2-fach **nor**-Schaltung in Bild 8.15. Sie hat die Funktion f = **not**(x1 **or**x2), d.h. der Ausgang der Schaltung wird "heruntergezogen", wenn einer der

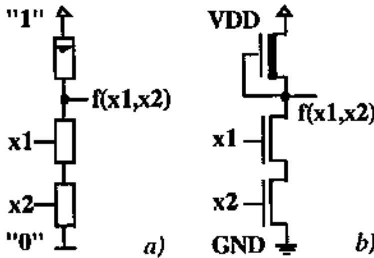


Bild 8.14: Eine 2-fach- **nand**-Schaltung in NMOS-Technologie, a) in der CSA-Ebene, b) Transistorschaltung.

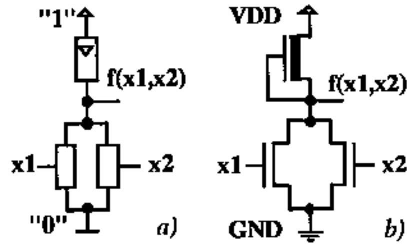


Bild 8.15: Eine 2-fach-**nor**-Schaltung in NMOS-Technologie; a) in der CSA-Ebene, b) in der Schaltkreis-ebene.

Schalter x1 oder x2 durchlässig ist (unter der Voraussetzung: positive Schalter). Ebenso ist diese **nor**-Schaltung durch einfache Parallelschaltung weiterer Schalter erweiterbar.

8.3 Schalternetz-Optimierung

Eine Möglichkeit heuristischen Vorgehens beim der Optimierung besteht in der Anwendung sukzessiver algebraischer Äquivalenz-Transformation. Zunächst werden Regeln der Schaltalgebra vorgestellt, die den heuristischen Entwurfsprozeß unterstützen und helfen die Anzahl der Gatter zu senken. Diese Regeln, die über die Netzumwandlung auf Gatterebene veranschaulicht werden (Bild 7.3), sind überwiegend einfach anzuwenden und werden in der digitalen Logik gelehrt. Als Beispiele hierfür seien die "de Morgan'sche Regel" (Bild 7.3 g, h) genannt und

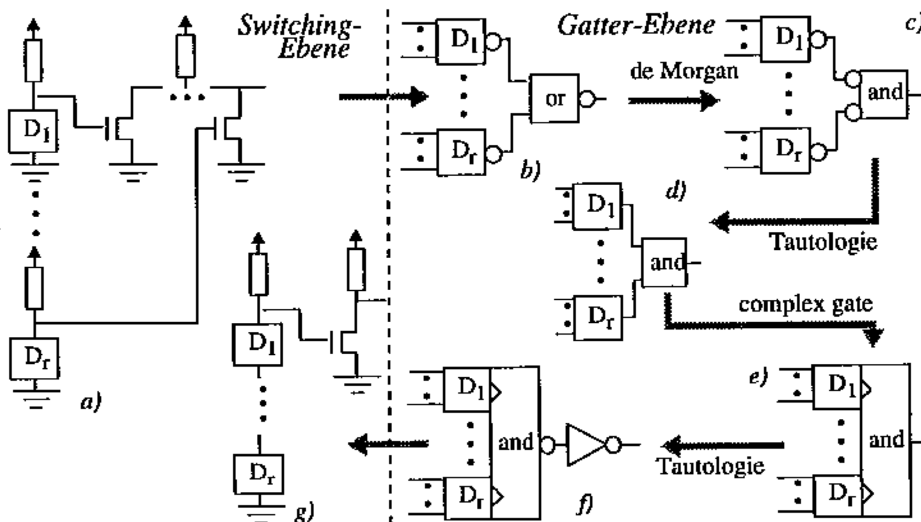


Bild 8.16: Beispiel zur Anwendung der Schaltalgebra zur Schalternetz-Umwandlung.

© Reiner W. Hartenstein, 76646 Bruchsal, Germany 1988 - 1993

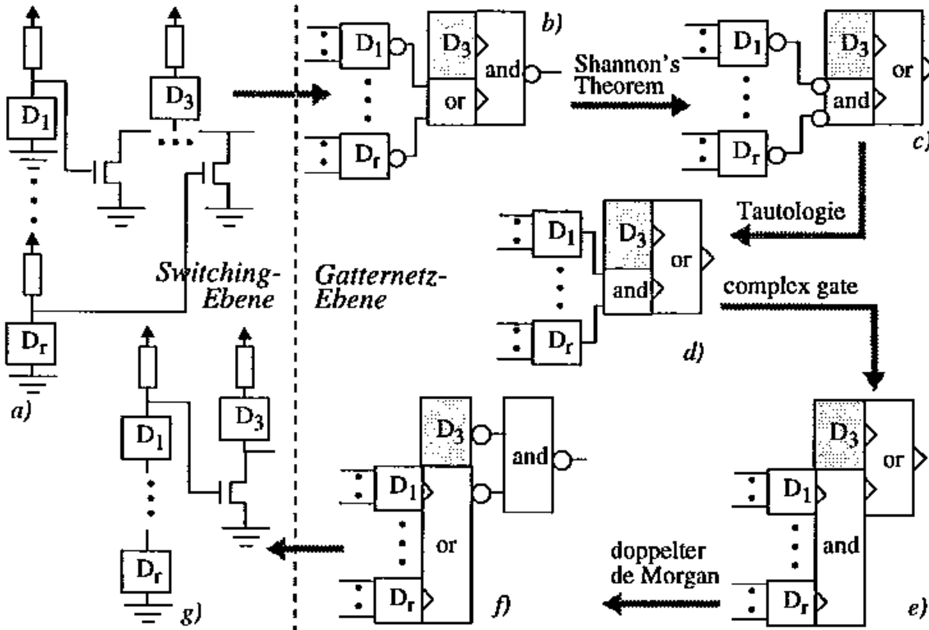


Bild 8.17: Ein zweites Beispiel zur Anwendung der Schaltalgebraregeln

die Tautologie: Doppelte Negation von A ergibt A (Bild 7.3 f). Etwas umfangreicher, aber nicht zu kompliziert ist das "Shannon'sche Theorem" (Bild 7.3j) sowie das "doppelte Shannon'sche Theorem" (Bild 7.3 k). Am besten kann man diese Umformungen mit Hilfe von Beispielen verdeutlichen. In Bild 8.16 soll aus dem Schaltnetz der Switch-Ebene (Bild 8.16 a) eine Version mit weniger Gattern bzw. mit weniger Transistoren erzeugt werden.

Dazu begeben wir uns von der Switch-Ebene in die Gatter-Ebene. Die Durchlaßfunktion der Durchlaßnetze D_1 bis D_r gehen, wegen der für positive Verhältnislogik typischen Negation, je in ein NOR-Gatter über (Bild 8.16 b). Nach "de Morgan" läßt sich das NOR in ein AND mit vorgeschaltetem Invertern auf der Eingangsseite umwandeln (Bild 8.16 c).

Die Tautologie liefert dann das Gatterdiagramm in Bild 8.16 d. Die Durchlaßnetze D_1 ... D_r gehören jetzt direkt in ein AND. Das kann als Komplex-Gatter wie in

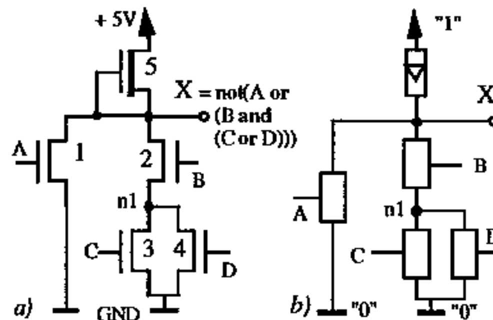


Bild 8.18: nMOS-Komplex-Gatter, a) in der Schaltkreis-Ebene), b) in der Switching-Ebene.

Primitive		OCO	OEO	enables	disables	passes*	cuts*
Realisierungen	Logik-Ebene (dieses Symbol nur b. CMOS übl.)						
	MOS Layout						
	MOS Sticks						
	Schaltkreis-Ebene		+				
'switching-symbol'		in, out, "0"	"1", in, out	control, in, out	control, in, out	control, in	control, in, out
KARL-Notation (Pseudo-Switching-Ebene)		out := oco in;	out := oeo in;	out := control enables in;	out := control disables in;	out := control passes in;	out := control cuts in;
ABL-Notation		in, oco, out	in, oeo, out	control, in, enables, out	control, in, disables, out	control, in, passes, out	control, in, cuts, out

Bild 8.19: KARL-3-Switching Elemente

© Reiner W. Hartenstein, 76646 Bruchsal, Germany 1988 - 1993

Netz-Klasse	Netz-Typen (Deklaration)	Statement zum Quellen-Anschluß	Anzahl Quellen	Wert ohne Quelle
Pfad	**node, lightnode, constant,** (Ausgang von:) **register, ram, rom, delayer**	**terminal**	1	?
Bus	**upbus**	**bus**	≥ 1	~ 1
	downbus			~ 0
	tribus			x
Extern-Quelle	**switch**	(entfällt)	0	per SCIL

Bild 8.20: Netztypen in KARL-3.

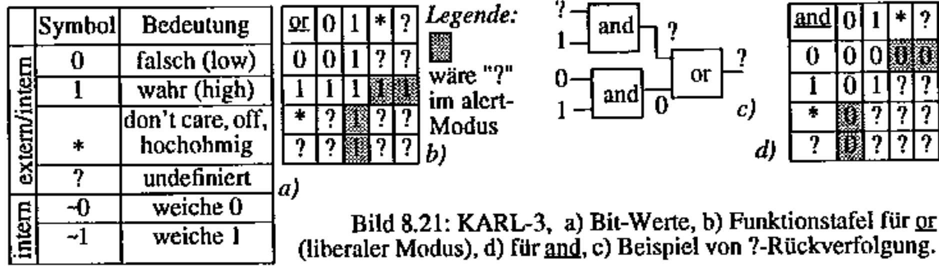
Bild 8.16 e dargestellt werden. Da diese Version eines Gatterdiagrammes nicht direkt in nMOS-Schaltungstechnik umgesetzt werden kann, bedienen wir uns nochmals der Tautologie. Aus dem AND wird ein NAND mit nachgestelltem Inverter (Bild 8.16 f). Dieses Ergebnis kann nun wieder in eine Darstellung der Switching-Ebene umgewandelt werden (Bild 8.16 g). Durch diese Transformationen werden also je r-1 Pullup- und Pulldown-Transistoren eingespart. Ein weiteres ähnliches Beispiel in Bild 8.17 soll die Anwendung der Schaltalgebra-Regeln bei heuristischer Minimierung noch weiter vertiefen. Auch hier werden je r-1 Pullup- und Pulldown-Transistoren eingespart.

8.4 Modellierung in der Switching-Ebene mit KARL-3

Zum Abschluß dieses Kapitels wollen wir zeigen, wie die Mehrebenen-Hardware-Beschreibungssprache KARL-3 [5] zur Modellierung auf Switching-Ebene verwendet werden kann. Wir verwenden nur eine Untermenge der Sprache, die wir KARLchen nennen (vgl. Grammatik in Bild 8.23 und Bild 8.24). Auch von der Simulator-Kommandosprache SCIL (Simulator Command and I/O Language) verwenden wir nur eine Untermenge SCILchen (Bild 8.29).

KARL-3 ist eine nicht-prozedurale Hardware-Beschreibungssprache, welche die Spezifikation von Hardware auf Register-Transfer-Ebene, auf Logik-Ebene und in etwas eingeschränkter Form auch in der Switching-Ebene erlaubt. Zur Modellierung auf Switching-Ebene wird die Netzklasse des Bus von KARL-3 verwendet mit den Netztypen **upbus**, **downbus**, und **tribus** (vgl. Bild 8.20). Ein Netz (z. B. ein Draht) ist Träger eines Signales. Als Signal-Quellen dienen hier Treiber, die per **bus**-Statement auf Netze aufgeschaltet werden (s. Bild 8.20). Es handelt sich also nicht um eine "echte" Switch-Level Beschreibung (da Treiber gerichtet sind), weshalb diese Ebene als Pseudo-Switching-Ebene bezeichnet wird. Unter unauffälliger Führung durch die Grammatik bemerkt der Benutzer dies kaum (infolge der KARL-Mnemonik).

Die Sprache KARL-3 und deren Simulator unterstützen extern (Ein- und Ausgabe) vier verschiedene Signal-Werte (0, 1, *, ?) vor (s. Bild 8.21 a). Intern verwendet die KARL-Semantik noch zwei weitere Signal-Werte (~0,~1). Diese internen Werte sowie der Wert * (hochohmig) unterstützen Modellierungen in der Switching-Ebene. Der Wert ? (undefiniert) zeigt an, daß Eingangs-Ports, Register und auch interne Knoten keinen Wert zugewiesen erhielten. Darüber hinaus wird der Wert "?" auch von KARL-Operatoren als Resultat generiert, wenn auf Grund fehlender oder widersprüchlicher Operanden kein konkreter Wert angegeben werden kann. Entsprechende sorgfältig abgestimmte Definitionen aller Sprach-Primitive bilden einen wesentlichen Bestandteil der Semantik von KARL-3 [5]. Bild 8.21 b zeigt dies am Beispiel der



	Symbol	Bedeutung
extern/intern	0	falsch (low)
	1	wahr (high)
	*	don't care, off, hochohmig
	?	undefiniert
intern	~0	weiche 0
	~1	weiche 1

a)

or	0	1	*	?
0	0	1	?	?
1	1	1	1	1
*	?	1	?	?
?	?	1	?	?

b)

and	0	1	*	?
0	0	0	0	0
1	0	1	?	?
*	0	?	?	?
?	0	?	?	?

d)

Bild 8.21: KARL-3, a) Bit-Werte, b) Funktionstafel für or (liberaler Modus), d) für and, c) Beispiel von ?-Rückverfolgung.

Wahrheitstafel eines **or**-Operators mit 2 Eingängen. In diesem Fall wird ein Operand sowohl mit "?" als auch mit "*" als "undefiniert" verstanden, d. h. es ist unbekannt, welcher Wert aus dem Vorrat {0, 1} unterstellt werden soll. Bei der Eingabekombination (1, ?) oder (1,*) ist das Resultat in jedem Fall eine "1" (vgl. Bild 8.21 b). Die Eingabekombination (0, ?) oder (0, *) ist jedoch nicht entscheidbar, sodaß der Operator ein "?" als Resultat abgibt.

Ein weiteres Beispiel ist der **and**-Operator gemäß Bild 8.21 d. Hier kann das Resultat nur dann "1" sein, wenn (1, 1) die Eingabekombination ist. Bei Eingabe (0, ?) oder (0, *) hingegen kann das Resultat nur "0" sein. Wie Bild 8.21 c zeigt, läßt sich längs eines Pfades das Auftreten der Werte "?" bis an den Ort der Ursache zurückverfolgen. Dies ist eine für Fehlerdiagnose wichtige Eigenschaft von KARL-3. Die in beiden Beispielen vorkommende Interpretation der Operanden "?" und "*" sei als *liberal* bezeichnet. Dieser liberale Modus erlaubt bei vielen fehlenden einzelnen Werten und vielen Wertekonflikten oft dennoch eine vernünftige Simulation. Für Zwecke der Fehlerdiagnose ist jedoch der strengere *Alert-Modus* günstiger. Solche Spracheigenschaften unterstützen beispielsweise Testbarkeits-Analyse und die Entwicklung von Testmustern [6]. Hierbei ist die Wahrheitstafel abweichend von der Darstellung in Bild 8.21 b und d: an den schattierten Stellen wird ein "?" als Resultatwert eingesetzt.

Anhang. Nun sei kurz eine Untermenge der Hardware-Beschreibungs-Sprache KARL-3 eingeführt, die wir KARLchen nennen wollen. Eine KARL-Beschreibung besteht aus zwei Teilen: einem Anweisungsteil (statement part) und einem vorangestellten Deklarationsteil. Bild 8.23 zeigt die Deklarations-Syntax und die Ausdrucks-Syntax von KARLchen. Diese Ausdrucks-Syntax ist Teil der Anweisungs-Grammatik, deren Rest in Bild 8.24 dargestellt ist. Eine HDL (Hardware Description Language) hat eine Anwendung, die von der einer Programmiersprache

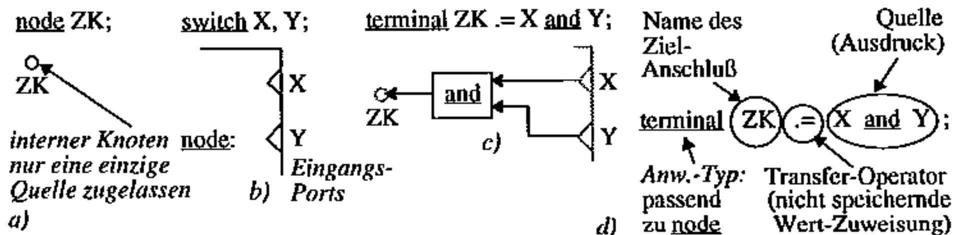


Bild 8.22: Erläuterung des Beispiels einer einfachen Verbindungs-Anweisung, a) Deklaration des Zielknoten, b) Deklaration von Eingangs-Ports, c) eigentliche Verbindungs-Anweisung (nicht speichernde Transfer-Anweisung), d) Erläuterung des Anweisungs-Formates zu (c).

© Reiner W. Hartenstein, 76646 Bruchsal, Germany 1988 - 1993

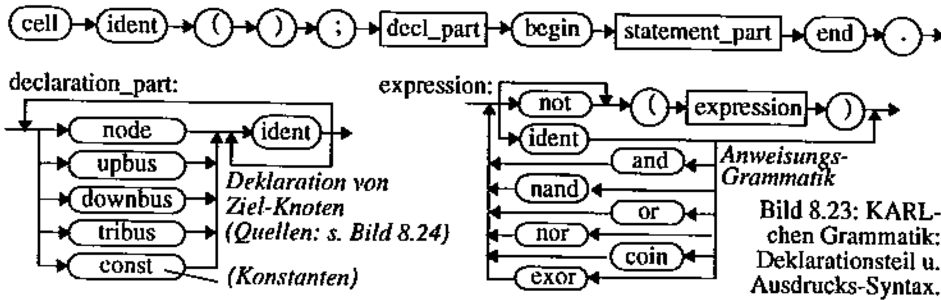


Bild 8.23: KARLchen Grammatik: Deklarationsteil u. Ausdrucks-Syntax.

grundsätzlich verschieden. Eine Programm ist nur eine Prozedur zur Berechnung von Zwischenresultaten und Resultaten ist, die auf einem Rechner nacheinander ausgeführt werden können. Deshalb werden Programmiersprachen gern als *prozedurale Sprachen* bezeichnet (abgesehen von applikativen Programmiersprachen). Bei einer HDL steht jedoch die Beschreibung des Vorrates physisch existierenden Operatoren und der Verbindungsleitungen zwischen diesen Operatoren im Vordergrund. Wie sich dies auf den Aufbau einer HDL auswirkt, wird an folgenden einfachen Beispielen veranschaulicht.

Es soll beispielsweise eine Hardware mit einem **and**-Gatter beschrieben werden, dessen Eingänge an zwei von der Außenwelt kommende Anschlüsse X und Y angeschlossen sind, und dessen Ausgang mit einem Anschluß ZK (Ziel-Knoten) verbunden ist (vgl. Bild 8.22 c). In KARL-3 muß zuerst der Zielknoten ZK deklariert worden sein (Bild 8.22 a), sowie die beiden Anschlüsse X und Y, welche die Operanden liefern (Bild 8.22 b). Sodann wird eine gerichtete Verbindung von der Quelle zum Zielanschluß beschrieben durch die Verbindungs-Anweisung gemäß Bild 8.22 c. Bild 8.22 d erläutert Format und Semantik dieser Transfer-Anweisung. Man beachte die ungewohnte Form der Wertzuweisung ".=", die einen Zustand angibt, nämlich das Bestehen einer Verbindungsleitung (im Gegensatz zur Wertzuweisung ":=" bei Programmiersprachen, welche ein Ereignis beschreibt, nämlich das Kopieren eines Wertes in ein speicherndes Element, wie beispielsweise ein Register). Wenn eine Beschreibung auf hoher Abstraktionsebene vorliegt, steht auch bei einer HDL auf der rechten Seite einer Verbindungsanweisung häufig nur ein Ausdruck.

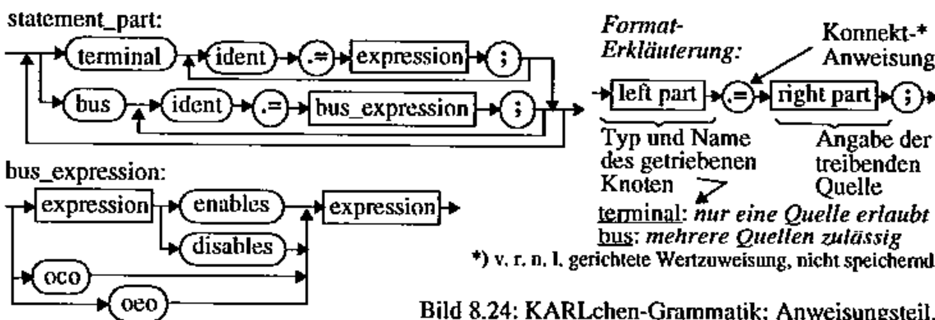


Bild 8.24: KARLchen-Grammatik: Anweisungsteil.

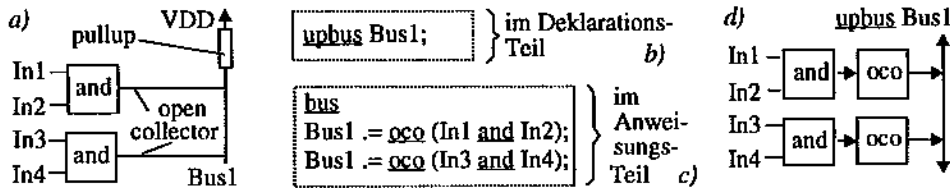


Bild 8.25: Bus-Beispiel, a) Problem, b) Deklaration, c) Konnekt-Anweisungen, c) ABL-Diagr.

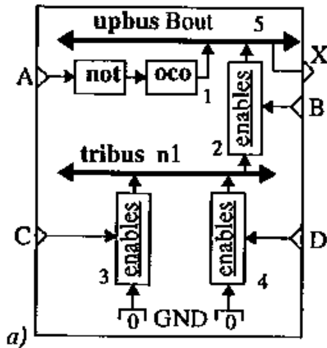
Der in obigem Beispiel vorkommende als **node** deklarierte Knoten ZK darf nur mit einer einzigen Quelle verbunden werden. Mit solchen Sprach-Konstrukten kann der für die Switching-Ebene typische Konnekt-Operator "#" nicht angewandt werden (Bild 8.4 bzw. Bild 8.5). Deshalb wurden für die Beschreibung von Bus-Schaltungen und ähnlichen Strukturen andere Sprach-Primitive eingeführt. Ein als **upbus**, **downbus**, oder **tribus** deklarierter Zielknoten (Syntax in Bild 8.23) darf mit mehreren Quellen verbunden werden, wie dies der #-Operator erfordert. Ein solcher Bus-Knoten kann über eine oder mehrere **bus**-Anweisung(en) mit einer oder mehreren Quell(en) verbunden werden (**bus** statement: Syntax in Bild 8.24).

Die KARL-Beschreibung einer Bus-Schaltung sei an folgendem einfachen Beispiel demonstriert. Zuerst wird die Deklaration formuliert (Bild 8.26 a): ein 1-Bit-Bus vom Typ **tribus** (weder Pullup- noch Pulldown-Widerstand) mit dem Namen BITB. An diesen Bus sollen zwei Bustreiber (vgl. Bild 8.19) angeschlossen werden mit den Datenquellen D1 und D2 und den Steuerungsvariablen Q und R (Bild 8.26 b). Die beiden Treiber können durch getrennte Konnekt-Anweisungen angeschlossen werden (Text in Bild 8.26 d, s. auch Bild f). Bild 8.26 c erläutert die Syntax der Konnekt-Anweisung. Die Verbindung kann aber auch in einer einzigen Konnekt-Anweisung beschrieben werden (Bild 8.26 e, veranschaulicht durch Bild 8.26 g).

An zwei weiteren Beispielen soll die Modellierung auf Switching-Ebene mit KARL-3 nochmals gezeigt werden.Bild 8.25 a zeigt eine wired-**and**-Schaltung, die in der Switching-Ebene beschrieben werden soll. Bild Bild 8.25 b und c zeigen Deklaration und Konnekt-Anweisung und Bild d das ABL-Diagramm. Der **oco**-Operator modelliert den <u>o</u>pen <u>c</u>ollector <u>o</u>utput der **and**-Gatter in Bild a. Bild 8.18 zeigt ein etwas komplexer zu beschreibendes Beispiel, eine einfache NMOS-Schaltung. Bild 8.19 zeigt das Blockdiagramm, dargestellt in ABL (<u>A</u> <u>B</u>lockdiagram <u>L</u>anguage [4]). Danach wird die äquivalente textuelle KARL-3 Beschreibung entwickelt (siehe Bild 8.27 b). Jeder interne Knoten der Schaltung wird als **bus** beschrieben. Ausgangsknoten *Bout* wird als **upbus** deklariert wegen des Pullup-Transistor 5 und Knoten n1 als **tribus**, weil weder Pullup- noch Pulldown-Widerstand vorhanden sind. GND (Null-Anschluß der Versorgungsspannung) und VDD können in KARL-3 durch Konstanten-Deklaration GND = 0 beschrieben (Zeile 3 in Bild 8.27). Die Eingänge A, B, C und D von der Außenwelt werden als "switch" deklariert (**switch**-Beispiel: Schalter an einer Konsole).

Nachdem nun alle Ziel-Anschlüsse deklariert sind, wenden wir uns den Treibern zu (vgl. Erläuterung in Bild 8.24 und Verhaltensbeschreibungen in Bild 8.19 und Bild 8.28). Der Transistor 1 mit Eingang A wird als **oco** dargestellt, wobei darauf zu achten ist, daß der Eingang negiert angelegt wird (Zeile 8 in Bild 8.27 b, siehe auch Bild 8.19). Die Transistoren 2, 3, und

© Reiner W. Hartenstein, 76646 Bruchsal, Germany 1988 - 1993




```
1   cell  beispiel (A,B,C,D) ; }  Header
2     node A, B, C, D;
3     const  GND .= 0 ;            Deklarations-Teil
4     upbus  Bout ;                (Ziel-Knoten)
5     tribus n1 ;
6   begin
7     bus
8      Bout .= oco not (A) ;       Anweisungs-Teil
9      Bout .= B enables n1 ;      (Treiber mit
10     n1  .= C enables GND ;      Quellen-
11     n1  .= D enables GND ;      angabe)
12  end .
```

b)

Bild 8.27: Switching-Modell zu Bild 8.18: a) ABL-Diagramm, b) KARL-3-Beschreibung.

4 mit den Eingängen B, C und D werden durch einen Bus-Ausdruck (**bus** expression) mit **enables**-Klausel als Treiber an Bus n1, bzw. Bus Bout angeschlossen.

Die Verbindungen zwischen den einzelnen Elementen werden im Anweisungsteil durch ein Bus-Statement durchgeführt. Da es sich um eine nicht-prozedurale Sprache handelt und hier eine statische Verbindungs-Struktur beschrieben wird, ist die Reihenfolge der Statements im Anweisungsteil irrelevant. Die nachfolgende Abbildung (Bild 8.19) zeigt die KARL-Primitive zur Switching-Modellierung in ABL- und KARL-Notation. Außerdem wird das dazugehörende Switching-Symbol, sowie Realisierungen in der Schaltkreis-Ebene, in der symbolischen Layout-Ebene und Layout-Ebene gezeigt.

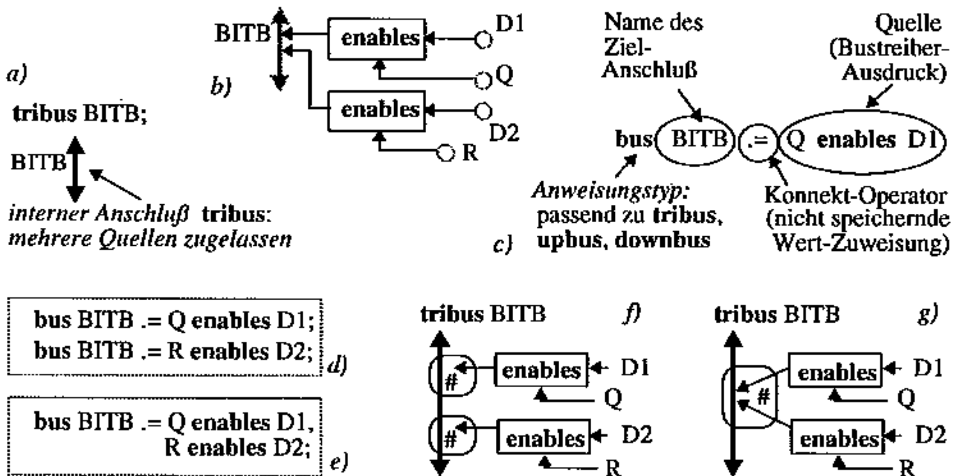


Bild 8.26: Ein (1-Bit-)Bus-Schaltungs-Beispiel: a) Deklaration des eigentlichen Bus, b) hierzu der Anschluß von 2 Bus-Treibern, c) hierzu die Syntax-Erläuterung zur Bus-Konnekt-Anweisung, d) getrennte Konnekt-Anweisungen, e) Mehrfach-Konnektanweisung, f) zu (d), g) zu (e).

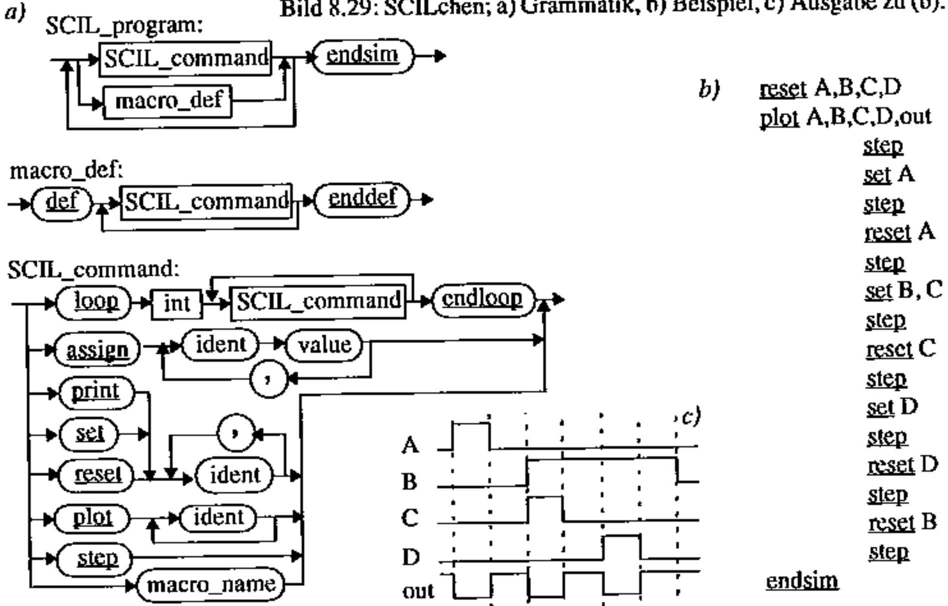
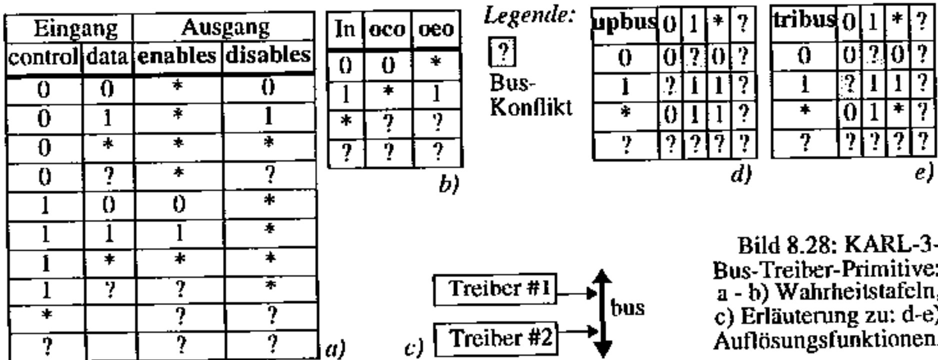


Bild 8.29: SCILchen; a) Grammatik, b) Beispiel, c) Ausgabe zu (b).

Wir wollen uns nun noch kurz der Arbeitsweise des KARL-3-Simulators zuwenden. Er arbeitet im Gegensatz zu Switching-Simulatoren gerichtet, d.h. er kennt vor der Simulation schon Quellen und Senken. Diese sind bei der ungerichteten Simulation nicht bekannt und müssen erst ermittelt werden. Ein Lösungsansatz war die Einführung des "Konnekt"-Operators zur Ermittlung des Wertes eines Knotens (siehe Abschnitt 8.1). Bild 8.30 zeigt die Struktur des KARL-3-Simulationssystems. Der KARL-3-Compiler generiert eine RTcode genannte exekutierbare Netzlistenstruktur [7], die vom Simulator ausgeführt wird unter Steuerung durch Stimuli und Kommandos, die in der Sprache SCIL formuliert sind. Auf der Basis von RTcode sind



Eingang		Ausgang	
control	data	enables	disables
0	0	*	0
0	1	*	1
0	*	*	*
0	?	*	?
1	0	0	*
1	1	1	*
1	*	*	*
1	?	?	*
*		?	?
?		?	?

a)

In	oco	oeo
0	0	*
1	*	1
*	?	?
?	?	?

b)

Legende: ? Bus-Konflikt

upbus	0	1	*	?
0	0	?	0	?
1	?	1	1	?
*	0	1	1	?
?	?	?	?	?

d)

tribus	0	1	*	?
0	0	?	0	?
1	?	1	1	?
*	0	1	*	?
?	?	?	?	?

e)

Bild 8.28: KARL-3-Bus-Treiber-Primitive: a - b) Wahrheitstafeln, c) Erläuterung zu: d-e) Auflösungsfunktionen.

© Reiner W. Hartenstein, 76646 Bruchsal, Germany 1988 - 1993

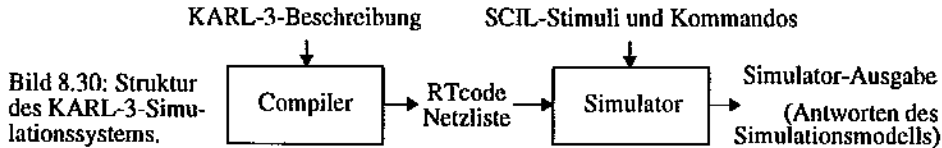


Bild 8.30: Struktur des KARL-3-Simulationssystems.

auch andere Entwurfswerkzeuge entwickelt worden (eine Übersicht gibt [6]). Über SCIL (Simulator Command and I/O Language) kann der Simulator interaktiv und im Stapelbetrieb betrieben werden [7].

Der KARL-3-Simulator verwendet extern folgenden Wertevorrat: {0, 1, *, ?}. Intern unterscheidet der KARL-3-Simulator auch "schwache Werte": der Wert auf einem **upbus** (ohne daß andere Quellen vorhanden sind) ist eine schwache Eins (~1) ist. Ein nackter **downbus** hat entsprechend eine weiche Null "~0" als Wert. Bild 8.29 a zeigt die Grammatik einer Untermenge von SCIL, die SCILchen genannt wird. Bild c zeigt eine SCILchen-Kommandofolge zur Hardwarebeschreibung nach Bild 8.27, welche die Stimuli *A, B, C, und D* und die Ausgabe der Anwort *Bout* definiert wie in Bild b gezeigt wird.

8.5 Switching-Modellierung in Verilog®

Auch das Verilog®-Simulations-System unterstützt die Modellierung der Switching-Ebene [12] [13] [14]. Dabei sind nicht nur explizite Primitive der Switching-Ebene vorgesehen. Verilog® unterscheidet 11 verschiedene Typen elektrischer Knoten, bzw. Netztypen (Tabelle in Bild 8.31), wobei die Unterscheidungs-Merkmale oft nicht eindeutig sind. Im Vergleich hierzu unterscheidet KARL-3 nur 5 Netztypen: **node** (nur eine einzige Quelle zulässig), **upbus**, **downbus** und **tribus** (mehrere Quellen zulässig), **switch** (keine Quelle zulässig). Die Netztyp-Bezeichnung **lightnode** in KARL ist nur ein Synonym für **node**.

Auch für Primitive der Logik-Ebene können die Treiber-Stärken der Ausgänge spezifiziert werden. Für die Ausgänge der folgenden Typen von Gattern und Switching-Elementen können

Netz-Typ	Werte-Vorrat		automatische Konflikt-Auflösung	Anwendung	
	physisch	Artefakt		bevorzugt	zulässig
wire	0, 1, z	x		für 1 Treiber	mehrere Treiber
tri	0, 1, z	x		f. mehrere Treiber	1 Treiber
wand	0, 1	x	ja	für 1 Treiber	mehrere Treiber
triand	0, 1	x	ja	f. mehrere Treiber	1 Treiber
wor	0, 1	x	ja	für 1 Treiber	mehrere Treiber
trior	0, 1	x	ja	f. mehrere Treiber	1 Treiber
tri1	0, 1	x		resistiver Pullup	
tri0	0, 1	x		resistiver Pulldown	
trireg	0, 1			speichert letzten Wert (Ladung)	
supply0	0			Masse-Anschluß (GND)	
supply1	1			Versorgungsspannung (VDD)	

Bild 8.31: Netztypen in Verilog® (z steht für hochohmig und x für unbekannt).

in Verilog® Treiber-Stärke-Werte (driver strength) individuell vereinbart werden.

and	**buf**	**nmos**	**tran**	**pullup**
nand	**not**	**pmos**	**tranif0**	**pulldown**
nor	**bufif0**	**cmos**	**tranif1**	
or	**bufif1**	**rnmos**	**rtran**	
xor	**notif0**	**rpmos**	**rtranif0**	
xnor	**notif1**	**rcmos**	**rtranif1**	

Die Treiber-Stärke-Vereinbarung erfolgt entsprechend dem Format in folgendem Beispiel vor der eigentlichen Instantiierung.

```
nor (highz1, strong0)
    N1 (enableOut, in1, in2),
    N2 (enableOut, in3, in4);
```

Bei Stärke-Deklarationen für die "1" (an erster Stelle innerhalb der Klammern) sind die folgenden Angaben zulässig: **supply1, strong1, pull1, weak1,** und **highz1.** Für die "0" (an zweiter Stelle innerhalb der Klammern) sind die folgenden Angaben zulässig: **supply0, strong0, pull0, weak0,** und **highz0.** Die Tabelle in Bild 8.32 a zeigt die in Verilog® unterstützten Stärkegrade. Bei Anwendung wird jeweils eine **0** oder **1** angehängt (s. obiges Beispiel). Bild 8.34 zeigt einige kommentierte Beispiele von Verilog®-Beschreibungen der Switching-Ebene.

8.6 Literatur

[1] R. Bryant: Boolean analysis of MOS circuits; IEEE Trans-CAD 6,4 (July) 1987
[2] B. Cosco: Neuronal Networks and Fuzzy Systems; Prentice-Hall, 1992
[3] E. Cox: Fuzzy Fundamentals; IEEE Spectrum, Oct. 1992
[4] R. Hartenstein: Fundamentals of structured hardware design; North Holland, 1977
[5] R. Hartenstein: KARL-3 Reference Manual; Univ. Kaiserslautern 1986
[6] R. Hartenstein: KARL and ABL; in (ed.: J. P. Mermet): Fundamentals and Standards in Hardware Description Languages; Kluwer Academic Publishers, Boston, 1993
[7] R. Hauck: KARL-3 User Guide; Univ. Kaiserslautern, 1986
[8] J. Hayes: Digital Simulation with Multiple Logic Values; IE³ T-CAD 5,2 (Apr.1986)

a)

Stärke-Name	Stärke-Niveau	Deklaration	modelliert:
Supply drive	7	**supply**	Spannungsversorg.
Strong drive	6	**strong**	Default von gate und assign
Pull drive	5	**pull**	gate und assign
Large capacitor	4	**large**	trireg-Netz-Kapaz.
Weak drive	3	**weak**	gate und assign
Medium capacitor	2	**medium**	trireg-Netz-Kapaz
Small capacitor	1	**small**	trireg-Netz-Kapaz
High impedance	0	**highz**	ungetriebener A.

b)

Input Strength	Reduced Strength
supply drive	pull drive
strong drive	pull drive
pull drive	weak drive
weak drive	medium capacitor
large capacitor	medium capacitor
medium capacitor	small capacitor
small capacitor	small capacitor
high impedance	high impedance

Bild 8.32: Treiber-Stärken (Strength) in VEROLIG: a) verschiedene Stärken-Werte, b) Stärken-Reduktion durch Gatter der Typen **rnmos, rpmos, rcmos, rtran, rtranif1,** und **rtranif0.**

© Reiner W. Hartenstein, 76646 Bruchsal, Germany 1988 - 1993

Element-Typ	Typ-Klasse	Beispiel oder Bemerkung	Bemerkung
and, nand, or, nor, xor, xnor	logisches Gatter	and (outw, in1, in2, ...)	Ausgang skalar 0, 1, oder 2 Delays
buf, not	Fan-out	buf (out1, out2, ... inw)	
bufif0, bufif1, notif0, notif1	Three-State-Treiber	bufif0 (outw, inw, control)	Ausgang: zusätzliche Werte H und L
nmos, pmos	MOS-Transistor (f. MOS-Netz)	nmos (w, datain, ncontrol);	
rnmos, rpmos			Stärken-Reduktion
tran	bidirektionale Transfer-Gatter	kein Steuereingang	
rtran		kein Steuereingang	Stärken-Reduktion
tranif0, tranif1		tranif1(inout1, inout2, control	zusätzl.Werte H, L
rtranif0, rtranif1		Ausgang: zusätzl. Werte[a] H und L	Stärken-Reduktion
pullup, pulldown	Lastwiderstand	pullup (netname)	
cmos	CMOS-Transistor (f. CMOS-Netz)	cmos (w, datain, ncntrol, pcntrol);	
rcmos			Stärken-Reduktion

a. unbekannt, ob Treibereingang getrieben oder hochohmig; H für 1 oder z, L für 0 oder z.

Bild 8.33: Verilog-Typen von Gattern und Switching-Elementen.

[9] J. Hayes: A unified switching theory with applications to VLSI design; Proc. IEEE, vol. 70 Oct 1982
[10] G.Klear, T. Folger: Fuzzy Sets, Uncertainty, and Information, Prentice-Hall, 1988
[11] S. Muroga: Logic Design and Switching Theory; Wiley - Interscience, 1979
[12] N. N.: Verilog® Reference Manual; CADENCE Design Systems, Inc., 1989
[13] N. N.: Programming Language Interface CADENCE Design Systems, Inc., 1990
[14] D. E. Thomas, Ph. R. Moorby: The Verilog® Hardware Description Language; Kluwer Academic Publishers, Boston et al., 1989

```
and (out,in1,in2);          //a 2-input and gate
nand #(6,9) (w1, w2, w3);
            //rize delay of 6, fall delay of 9
bufif1 (strong1,weak0) (outw,inw,controlw);
notif0 gate_name(wire_a, wire_b, wire_c);
      or (highz1,strong0) #(3,2);
        // drives a wired-and configuration
      nor #(26,43)
      g1(n1,n2,n3,n4),
        (n2,n3,n4),              // no name
      g2(n3,n4,n5);
            // three nor's with same delay
      nand #25 (out, a[3], a[2], a[1], a[0]);
                  // bit selects on inputs
```

```
bufif1 #(13,19,100)
   d1(out1, data, control); //delay of 13
                     //for transition to 0,
                     //19 to 1, and 100 to Z.
      wire netA, netB;
      pullup (netA), (netB);
      bufif1
            driver1(netA, data1, c),
            driver2(netB, data2, c);
```

Bild 8.34: Verilog®-Beschreibungs-Beispiele der Switching-Ebene.