11 MOS-Digitalschaltungen

Dieses Kapitel berührt die in Bild 11.1 gezeigten Abstraktions-Ebenen. Dieses Kapitel führt informell in die Technologie und Schaltungstechnik integrierter digitaler MOS-Schaltungen ein. Dabei wird nur ein naives Verständnis elektronischer und elektrotechnischer Grundlagen sowie praktisch keine physikalischen und technologischen Vorkenntnisse vorausgesetzt. Auf diese Weise kann sich der Entwickler voll auf die Anwendung und den System-Aspekt konzentrieren, denn er ist sozusagen für die Organisation von Komplexität zuständig. Bei modernen integrierten Schaltungen sehr hoher Integrationsdichte kann dabei die Arbeitsteilung zwischen Technologie und Design wie folgt charakterisiert werden. Während die Elektrotechnik dafür zuständig ist, wie man Transistoren so klein und so schnell macht, besteht die Aufgabe des Designers in der Organisation: wie bringt man Millionen von Transistoren auf einem einzigen Chip unter und wie verknüpft man diese zur gewünschten Gesamtfunktion.

Diese Arbeitsteilung ist nicht selbstverständlich, sondern Ergebnis einer Revolution, die Ende der 70er Jahre stattfand und zum großen Teil auf Betreiben von Carver Mead zurückgeht (Professor am California Institute of Technology). Zusammen mit seiner Schülerin Lynn Conway (damals am Forschungszentrum Xerox PARC, Palo Alto, Kalifornien) verfaßte er das erste Design-Lehrbuch für Nicht-Technologen [11] - bewußt auf der Basis von nMOS-Technologie. Der Titel *VLSI System Design* mit Betonung auf *System* deutet die Zielgruppe an (s. a. [17]).

Integrierte nMOS-Schaltungen eignen sich besonders gut für eine Einführung in die Mikroelektronik, denn nicht nur der Transistor als *device* ist leichter zu verstehen als der bipolare Transistor, sondern auch die Schaltungstechnik ist einfacher zu realisieren. Integrierte nMOS-Schaltungen haben gegenüber bipolaren Schaltungen sogar eine ganze Reihe von Vorteilen:

- höhere Integrationsdichte
- einfachere Schaltungstechnik
- geringe Anzahl benötigter Masken
- einfacheres und verständlicheres Layout
- (zum Teil) einfachere Entwurfs-Hilfsmittel

Bild 11.2 zeigt die nMOS-Grundschaltungen der Verhältnislogik. Diese Schaltungstechnik ist einfacher als bei Bipolar-Schaltungen gleicher Logik aus Gründen, die im folgenden gezeigt werden. Ein weiterer Vorteil ist die viel höhere Integrationsdichte, wie später gezeigt wird.

Bild 11.3 veranschaulicht die mit nMOS-Technologie erreichbare höhere Integrationsdichte durch einen Vergleich von λ-basiertem Layout (das Layout ist gerastert, basierend auf der Größe einer Grundeinheit λ). Bild a zeigt das Layout eines bipolaren Transistors mit Sperrschicht-Isolierung [4] (vgl. auch Bild 10.5). Bild b zeigt das Layout eines nMOS-Transistors [11]. Wie

11.1 nMOS-Verhältnis-Logik 222
11.2 NMOS-Multiplexer 223
11.3 Dynamische NMOS-Schaltungen 227
11.3.1 KARL-3-Beschreibung dynamischer nMOS-Schaltkreise 228
11.3.2 Verilog®-Beschreibung dynamischer nMOS-Schaltkreise 229
11.4 nMOS-MOL-Schaltungen (tiled logic) 230
11.4.1 nMOS-PLAs 232
11.4.2 Der Weinberger-Array 238
11.5 Literatur 240

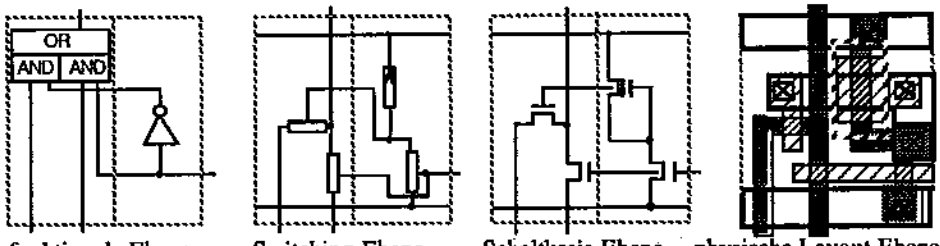


Bild 11.1: Durch dieses Kapitel berührte Abstraktions-Ebenen.

sofort zu sehen ist, verbraucht der MOS-Transistor sehr viel weniger Fläche aus folgenden Gründen. Der MOS-Transistor benötigt keine besonderen Maßnahmen zur Isolierung gegen seine Umgebung. Sämtliche Anschlüsse können sofort als Zuleitungen verlängert werden (Bild 11.3 d), wodurch beispielsweise zwei in Serie geschaltete Transistoren sehr nahe aneinander plaziert werden können (Bild 11.3 c). Jeder Anschluß eines bipolaren Transistors ist jedoch nur über je ein Kontaktloch *(cut)* durch eine Metall-Leitung erreichbar (Bild 11.3 a).

11.1 nMOS-Verhältnis-Logik

Ein weiterer, in vereinfachter Schaltungstechnik resultierender Vorteil der nMOS-Technologie besteht darin, daß der MOS-Transistor spannungsgesteuert ist. Wie Bild 11.4 e zeigt, kann jede MOS-Transistorstufe (hier mit Transistor 1) eine nachfolgende Stufe (hier mit Transistor 2) direkt treiben ohne Verwendung eines Vorwiderstands. Der Gate-Eingang des getriebenen Transistor 2 ist extrem hochohmig (viele MΩ bis GΩ), sodaß im statischen Zustand U_a für die logische "1" praktisch die Betriebsspannung VDD erreicht. Bei bipolaren Transistoren wird diese Schaltungstechnik mit direkter Kopplung als DCTL (directly coupled transistor logic) bezeichnet (Bild 11.4 a).

Da bipolare Transistoren stromgesteuert sind, weshalb der Basis-Eingang sehr niederohmig ist, wird bei DCTL die Spannung für die logische "1" auf Werte von deutlich weniger als 1 Volt beschränkt. Es sei hierzu an das Doppel-Dioden-Modell des Bipolartransistors erinnert (Bild 6.5 d). Das Modell in Bild 11.4 b zu Bild a zeigt, wie die Emitterdiode ED von Transistor T2 als Fangdiode für die Ausgangsspannung U_a wirkt. Die Ausgangsspannung U_a wird auf die Kniespannung der Diode ED (ca. 0,7 Volt) beschränkt. Eine höhere Ausgangsspannung U_a bei

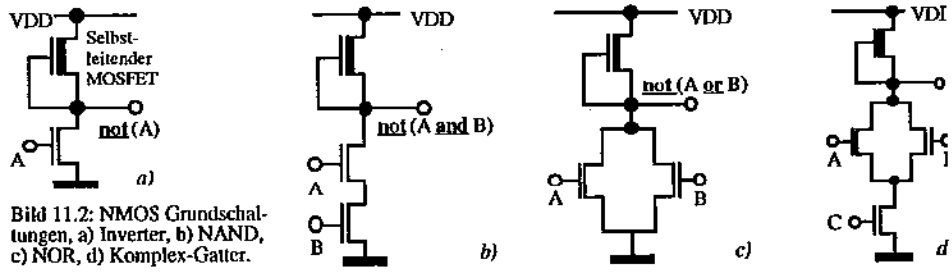


Bild 11.2: NMOS Grundschaltungen, a) Inverter, b) NAND, c) NOR, d) Komplex-Gatter.

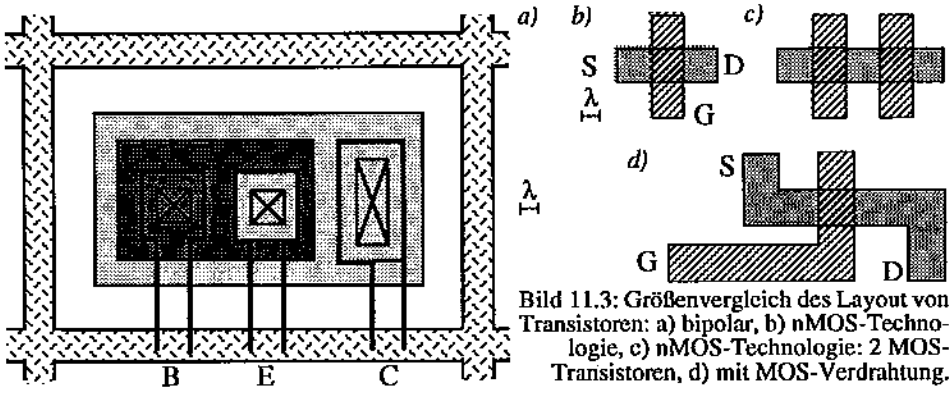


Bild 11.3: Größenvergleich des Layout von Transistoren: a) bipolar, b) nMOS-Technologie, c) nMOS-Technologie: 2 MOS-Transistoren, d) mit MOS-Verdrahtung.

bipolaren Transistorstufen kann nur über die Verwendung eines Vorwiderstands R_V erreicht werden, wie Bild 11.4 c und dessen Modell in Bild d zeigen *(RTL: resistor transistor logic).* Durch solche Vorwiderstände wird weitere Layoutfläche verbraucht.

Es gibt Logik-Schaltungen mit pMOS-Transistoren, die dual sind zu den oben eingeführten Schaltungen mit nMOS-Transistoren. Die Dualität besteht darin, daß bei pMOS alle logischen Relationen invers sind zu denen bei nMOS. Die Betriebsspannung ist invers (VDD statt GND; Last gegen GND statt gegen VDD). Der Transistor ist invers, d. h. ein negativer Schalter (on bei U_g = GND, off bei U_g = VDD). Unter Einbezug solcher pMOS-Transistoren läßt sich nMOS-Logik auf einfache Weise erweitern zu CMOS-Schaltungstechnik (Kapitel 16 , [1]).

11.2 NMOS-Multiplexer

Insbesondere Multiplexer sind in nMOS-Technologie besonders einfach und sehr flächensparend realisierbar. Dies wird in diesem Abschnitt veranschaulicht. Zunächst soll der Multiplexer aus Sicht der Anwendung kurz eingeführt werden. Ein Multiplexer ist ein Quellen-Selektor. Bild 11.5 a zeigt die Aufgabe eines Multiplexers am Beispiel des 4-Wege-Multiplexers: eine von 4 Daten-Quellen Quelle0 bis Quelle3 soll unter Kontrolle durch eine Selektor-Variable (hier aus dem Vorrat 0 bis 3) mit dem Zielanschluß D (Multiplexer-Ausgang) verbunden werden. Bild 11.5 b zeigt das Logik-Diagramm hierzu.

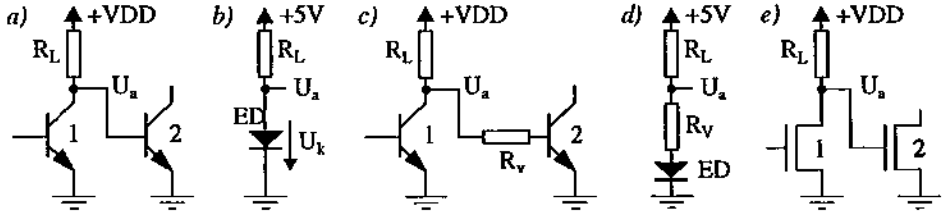


Bild 11.4: Kopplung aufeinanderfolgender Transistorstufen: a) direkt bei DCTL, b) Modell zu (a), c) mit Vorwiderstand R_V bei RTL, d) Modell zu (c), e) bei MOS-Schaltungstechnik.

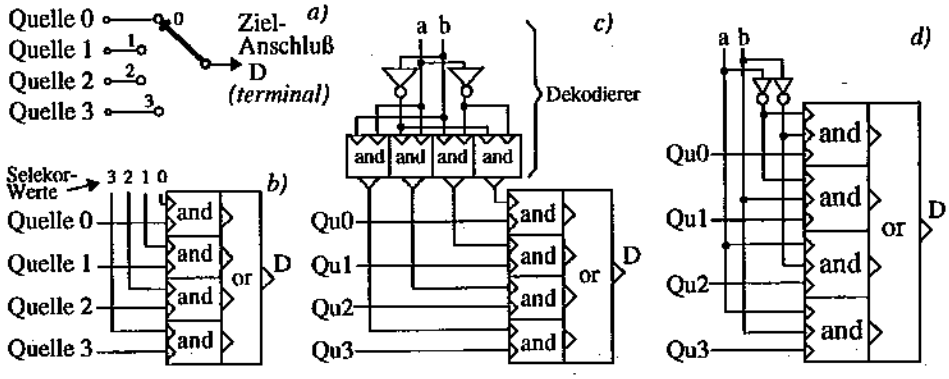


Bild 11.5: Multiplexer-Grundlagen am Vier-Wege-Beispiel: a) Veranschaulichung der Anwendung, b - d) Logikdiagramme: b) undekodierter Multiplexer, c - d) vollständig dekodierter Multiplexer: c) modulare Version (separater Dekodierer), d) Logik-minimierte Version.

Ein Multiplexer dieser Form wird *undekodierter Multiplexer* genannt, da die Selektorvariable im 1-aus-n-Kode (hier 1-aus-4-Kode) ausgewertet wird. Dieser Kode, auch Singulär-Kode genannt, ist eigentlich eine Markierungs-Darstellung, wobei für jeden Wert ein eigenes existiert (vgl. 20.3.4). In unserem Beispiel mit der Wortlänge 4 ist jedem der 4 Werte von 0 bis 3 ein bestimmtes Bit zugeordnet. Ist beispielsweise das Bit Nr. 2 markiert (wobei alle anderen Bits null sein müssen), so wird Quelle2 auf den Ausgang D geschaltet (vgl. Bild 11.5 b).

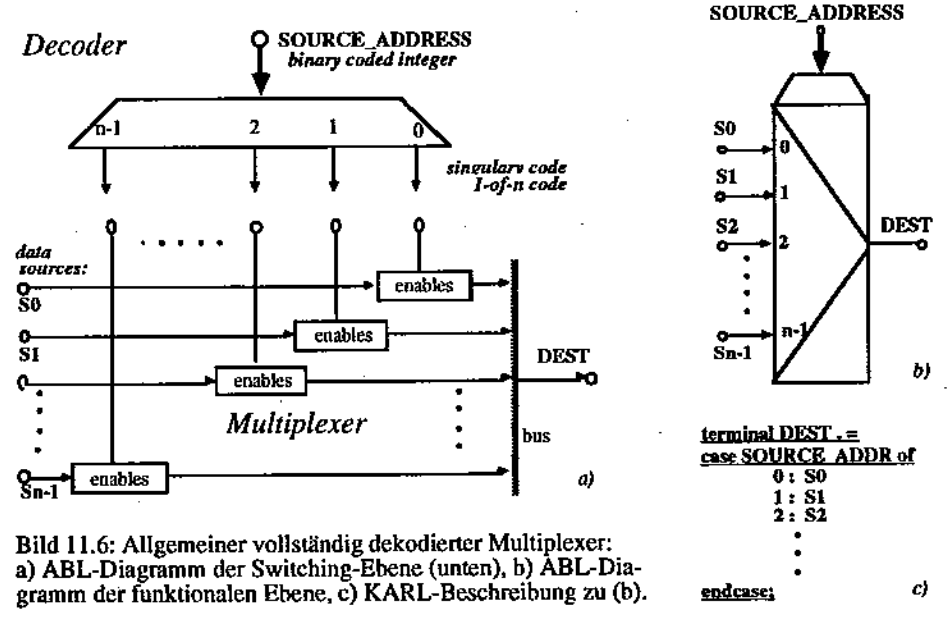


Bild 11.6: Allgemeiner vollständig dekodierter Multiplexer:
a) ABL-Diagramm der Switching-Ebene (unten), b) ABL-Diagramm der funktionalen Ebene, c) KARL-Beschreibung zu (b).

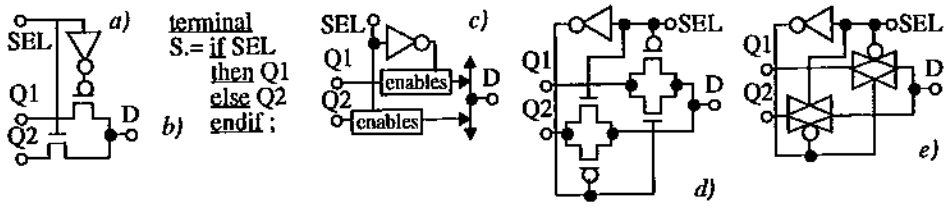


Bild 11.7: Vollständig dekodierter Zwei-Wege-Multiplexer: a) nMOS-Realisierung (Mischdiagramm), b) textuell: KARL-Beschreibung, c) ABL-Diagramm, d - e) CMOS-Realisierung.

Vollständig dekodierter Multiplexer. Häufig liegt die Selektor-Variable für einen Multiplexer nicht im Singulär-Kode, sondern im Binärkode vor (in unserem Beispiel mit einem Wertevorrat von 4 ein 2-Bit Binärkode SEL = (a, b)). Zur richtigen Interpretation einer solchen Selektorvariablen benötigt der Multiplexer zusätzlich noch einen Dekodierer zur Konversion des Binärkodes in einen Singulärkode. Bild 11.5 c zeigt die entsprechend durch einen passenden Dekodierer erweiterte Version des Multiplexers aus Bild b. Ein solcher Multiplexer wird *vollständig dekodierter Multiplexer* genannt, zwecks besserer Unterscheidung von nicht dekodierten Multiplexer. Bei Bild 11.5 c handelt sich um eine modulare Form dadurch, daß Dekodierer und der eigentliche Multiplexerkern als zwei getrennte Moduln deutlich voneinander separiert sind. Eine Minimierung der Gatteranzahl und damit der Transistorzahl und des Flächenbedarfs kann durch eine Verschmelzung dieser beiden Module erreicht werden, wobei aus Bild c der Logikplan nach Bild d entsteht.

Der Mutiplexer allgemein. Bild 11.6 zeigt die allgemeine Darstellung des vollständig dekodierten Multiplexers für beliebige Anzahl von Wegen. Bild a zeigt das ABL-Diagramm des Multiplexerkerns in der Switching-Ebene (der Dekodierer ist durch das Symbol der funktionalen Ebene gezeigt (vgl. auch Bild 11.1), womit insgesamt ein Misch-Diagramm vorliegt). Bild 11.6 b zeigt das ABL-Symbol und Bild c die textuelle KARL-Beschreibung der funktionalen Ebene für den vollständig dekodierten Multiplexer. Bild 11.7 zeigt das Zwei-Wege-Beispiel eines vollständig dekodierten Multiplexers in verschiedenen Darstellungsformen.

Vergleich zwischen bipolar und MOS. Bipolare Schaltungstechniken lassen sich relativ einfach in nMOS-Schaltungen übertragen. Beispielsweise läßt sich aus der Schaltung nach Bild 11.4 a oder (Unter Fortfall des Vorwiderstandes RV) aus der Schaltung nach Bild c Schaltplan nach Bild 11.8 a ableiten (weshalb diese Schaltung *pseudo-bipolar* genannt sei). Bei MOS-Technologie kann dank der Verfügbarkeit des Transfer-Transistor *(pass transistor)* der Multiplexerkern auch durch die Schaltung nach Bild 11.8 b realisiert werden, wobei deutlich weniger Transistoren benötigt werden. (Der Transfer-Transistor ist ein ganz normaler Transistor, der jedoch nicht - wie bei Verhältnis-Logik - quer zur Datenflußrichtung verschaltet ist, sondern mit seinem Kanal in Datenflußrichtung liegt.) Bild 11.8 c zeigt, daß sich ein solcher Transfer-Multiplexer sehr gut regelmäßig strukturiert layouten läßt, was recht flächen-effizient ist.

Bild 11.8 d und e zeigen eine Variante hierzu mit noch größerer Flächen-Effizienz dadurch, daß sich Poly und Diff auch dort kreuzen dürfen, wo eigentlich kein Transistor benötigt wird (vgl.

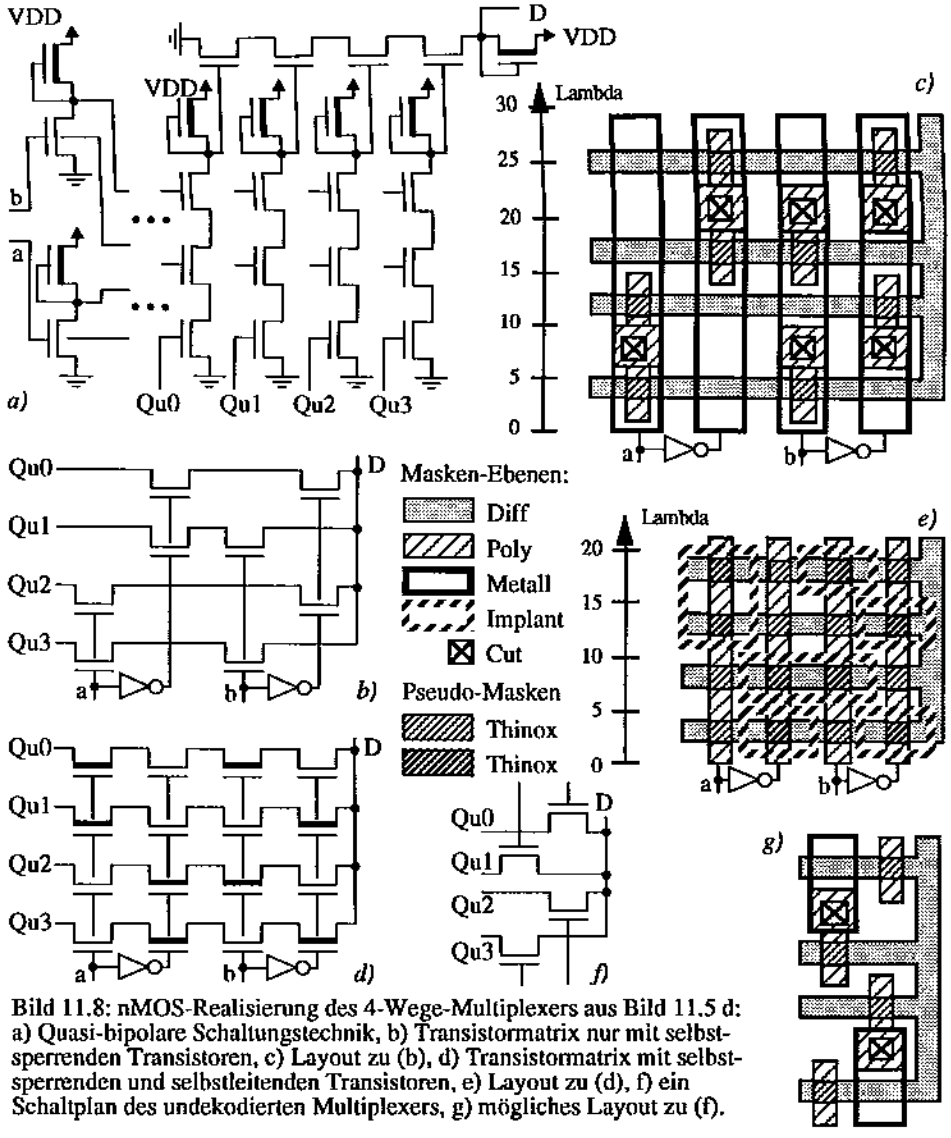
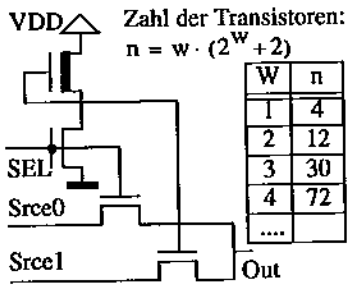
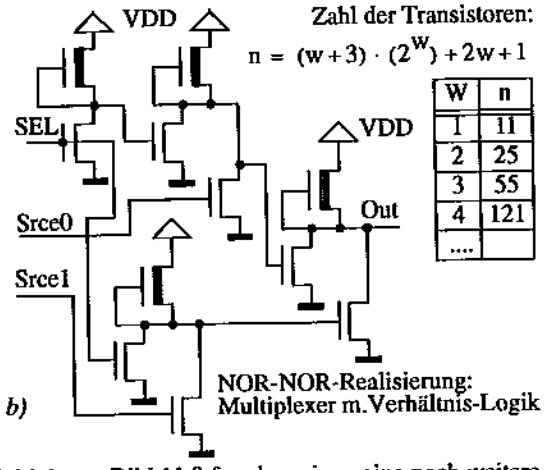


Bild 11.8: nMOS-Realisierung des 4-Wege-Multiplexers aus Bild 11.5 d: a) Quasi-bipolare Schaltungstechnik, b) Transistormatrix nur mit selbstsperrenden Transistoren, c) Layout zu (b), d) Transistormatrix mit selbstsperrenden und selbstleitenden Transistoren, e) Layout zu (d), f) ein Schaltplan des undekodierten Multiplexers, g) mögliches Layout zu (f).

Bild c). Dies wird durch Einfügung selbstleitender Transistoren an diesen Stellen erreicht (vgl. Bild e) [11][16]. Es läßt sich leicht nachprüfen, daß in der Schaltung nach Bild d die selbstleitenden Transistoren immer dort leitend sind, wo die Selektorvariable einen durchgeschalteten Weg verlangt. Somit ist in nMOS-Schaltungstechnik für Multiplexer eine drastisch bessere Flächen-effizienz erreichbar als in bipolarer Schaltungstechnik. Bild 11.9 zeigt eine ähnliche



a)



b)

Bild 11.9: Multiplexer-Beispiel; a) AND/OR mit Transfer-Transistoren, b) NOR/NOR in Verhältnis-Logik.

Gegenüberstellung beim Zwei-Wege-Multiplexer. Bild 11.8 f und g zeigen eine noch weitere Reduzierung des Flächenbedarfs für den Multiplexerkern, wie sie möglich ist wenn die Selektorvariable von außen her schon im Singulärkode vorliegt.

11.3 Dynamische NMOS-Schaltungen

Ein weiterer Vorteil der MOS-Technologie besteht darin, daß unter Verwendung des Transfer-Transistors sehr einfache dynamische Stufen realisiert werden können [11]. Eine entsprechende MOS-Schaltungstechnik läßt beispielsweise eine viel einfachere Realisierung des Flipflop aus Bild 10.19 a zu, als die direkte Umsetzung der bipolaren Schaltungstechnik ergibt (Bild 11.10 a).

Dynamische MOS-Schaltungen nutzen die parasitäre Kapazität (C in Bild 11.10 b) des Transistor-Gates als temporären Speicher aus. Diese Kapazität ergibt sich aus dem Kondensator-ähnlichen Aufbau des MOS-Transistors (vgl. Bild 6.8 e/f). Wurde über den Transfer-Transistor durch einen Takt T eine logische "1" (5 Volt) auf diese Kapazität C "geladen", so wird diese nach dem Abschalten des Transfer-Transistors (T=0) diese "1" noch eine Zeit lang gespeichert. Somit eignet sich die Stufe in Bild 11.10 b als Zwischenspeicher. Bild 11.10 c zeigt ein Modell dieser Stufe. Die Speicherzeit liegt in der Größenordnung von Millisekunden und ist stark von der Temperatur abhängig.

Wenn Langzeitspeicherung beabsichtigt ist, muß der gespeicherte Wert iterativ in genügend kurzen Intervallen wiederaufgefrischt werden. Aus dem Modell in Bild 11.10 c kann jedoch auch eine Flipflop-Schaltung abgeleitet werden, die keinen Wiederauffrischungs-Generator *(refresh generator)* erfordert (Bild 11.10 d). Über Transistor 2 werden bei T=0 zwei Inverter zu einer Rückkopplungsschleife verschaltet: es entsteht die bekannte selbsthaltende Anordnung, welche den Wert Q dauerhaft speichert. Zum Schreiben eines Wertes D in das Flipflop wird kurzzeitig T=1 bewirkt, wobei der erste Inverter als dynamische Stufe gemäß Bild c den neuen Wert solange kapazitiv speichert, bis eine Umschaltung von T=1 auf T=0 beendet ist.

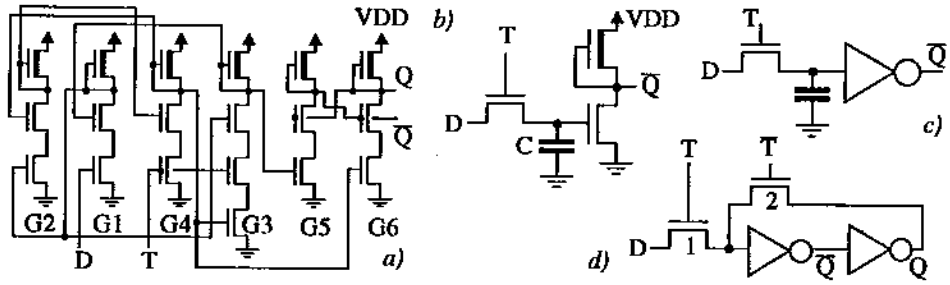


Bild 11.10: Realisierung des D-Flipflop aus Bild 10.19 a: a) quasi-bipolare Schaltung, c) dynamischer Inverter als Speicherglied, c) Modell zu (b), d) (a) als dynamische nMOS-Schaltung.

Das Flipflop nach Bild 11.10 d besteht nur aus 6 Transistoren im Vergleich zu 19 Transistoren in Bild a. Somit ist durch MOS-Schaltungen gegenüber bipolaren Schaltungen eine erhebliche Flächenersparnis möglich. Man beachte, daß die Transistoren 1 und 2 in Bild 11.10 d einen Multiplexer aus Transfer-Transistoren bilden. Die hier gezeigte sparsame Schaltungstechnik geht also eindeutig auch hier auf die Anwendung des Transfer-Transistors zurück. Vergleichbare Schaltungstechniken sind hier im Bereich der Bipolarschaltungen nicht möglich.

11.3.1 KARL-3-Beschreibung dynamischer nMOS-Schaltkreise

Die Standard-Funktion '**DYN[&n]**' modelliert eine dynamische Stufe wie beispielsweise im Eingang eines dynamischen nMOS-Inverters (Bild 11.10 b und c). Der Parameter 'n' (Zerfallzeit) gibt die Zeitspanne an (gemessen in elementaren Simulations-Schritten), nach welcher der gespeicherte logische Wert verfliegt. Ein Werte-Wechsel am Eingang wird verzögerungsfrei weitergegeben. Wenn der Eingabewert jedoch zu '*' (hochohmig) wird, dann wird der unmittelbar vor dem Wechsel zu '*' angelegene Wert n Zeitschritte lang gespeichert. Nach Beendigung der Zerfallzeit schaltet der Ausgang auf '?' um ('undefiniert'), womit der Gedächtnisverlust der Stufe angezeigt wird. Ist 'n' nicht angegeben, so wird ein Default-Wert von '0' benutzt. Das Verhalten des **DYN**-Elements wird durch Fig. 11.11 veranschaulicht. Das

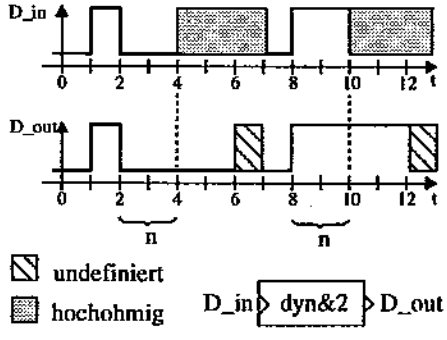
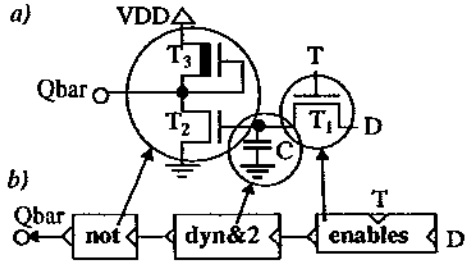


Fig. 11.11: Ein Anwendungs-Beispiel zur KARL-3-Standard-Funktion **DYN**.

Bild 11.12: KARL-3-Modellierung (b) einer dynamischen nMOS-Inverter-Stufe (a).



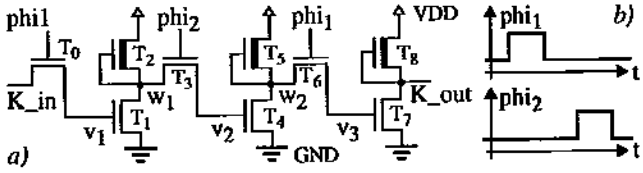


Bild 11.13: nMOS-Schieberegister (a) mit Angabe der Takt-Phasen (b).

DYN-Modell enthält keinerlei Logik-Funktionen zwecks Ermöglichung klarer Modellierungen. Deshalb **muß** beispielsweise ein dynamischer Inverter wie folgt beschrieben werden:

terminal Qbar **.=** **not** (**dyn&2** (T **enables** D) ;

Bild 11.12 veranschaulicht diese Philosophie der Modellierung an Hand des obigen Beispiels. Der **enables**-Operator (Bild b) modelliert den Transfer-Transistor T_1 (Bild a), der **dyn**-Operator den Speicher-Effekt durch die parasitäre Kapazität C und der **not**-Operator die Funktion des nMOS-Inverters aus T_2 und T_3.

11.3.2 Verilog®-Beschreibung dynamischer nMOS-Schaltkreise

(Eine kurze Einführung der Switching-Modellierung in Verilog® ist in Abschnitt 8.5 zu finden.) Bild 11.14 zeigt die Verilog®-Beschreibung einer dynamischen nMOS-Inverterkette (aus Bild 11.13) zur Switching-Modellierung [18]. Zeile 5 zeigt die Deklaration **pullup** für die Modellierung der Lasttransitoren T_2, T_5, und T_8. Hier liegt ein typisches Sprach-Primitiv der Switching-Ebene vor. Zeile 4 zeigt die Bus-ähnliche Deklaration des Netztyp **tri** der Ausgangsknoten w_1, w_2, und K_out der drei Inverter, an welche die Lasttransistoren ja durch das Statement in Zeile 5 angeschlossen werden. Der Netztyp **tri** läßt mehrfache Wertequellen zu und unterstützt die Auflösungs-Funktion bei Quellen-Konflikten. Der Netztyp **trireg** (Zeile 6 in Bild 11.14) modelliert die Ladungs-Speicherung durch die parasitäre Kapazität des Inverter-Eingangs. Die Silbe **reg** in **trireg** deutet diesen Ladungsspeicher an. Die übrigen Transistoren aus Bild 11.13 werden durch die Anweisung **nmos** (Zeilen 8 bis 11) als Switching-Elemente deklariert und instantiiert.

```
module ShiftReg ( K_out, K_in, phi1, phi2);          // siehe Bild 11.13   (1)
   output      K_out;                  // output port declaration          (2)
   input       K_in, phi1, phi2;       // input port declaration           (3)
   tri         w1, w2, K_out;          // tri nets pulled up to VDD        (4)
   pullup      (w1), (w2), (K_out);    // pullups T2, T5, T8               (5)
   trireg      (medium) v1, v2, v3;    // charge storage nodes             (6)
   supply0     GND;                    // gnd supply                       (7)
   nmos        #3                      // transistors and their interconnect (8)
               T0 (v1, K_in, phi1),    T1 (w1, GND, v1)                 (9)
               T3 (v2, w1, phi2),      T4 (w2, GND, v2),                (10)
               T6 (v3, w2, phi1),      T7 (K_out, GND, v3);             (11)
endmodule                                                               (12)
```

Bild 11.14: Verilog®-Beschreibung des Schieberegisters nach Bild 11.13.

```
module   WavesShiftReg;    // sim test pattern generator for ShiftReg   (13)
    wire       shiftout     // net to receive CUT response               (14)
    reg        shiftin;     // register to drive stimulus into CUT       (15)
    reg        phi1, phi2;  // registers to drive clocks into CUT        (16)
    parameter d = 100;  // constant declaration of waveform time step   (17)

    shreg      cct (shiftout, shiftin, phi1, phi2);                     (18)

    initial                                                             (19)
        begin :main                                                     (20)
            shiftin = 0;          // initialize stimuli waveforms       (21)
            phi1 = 0;                                                   (22)
            phi2 = 0;                                                   (23)
            setmon;               // call to the task setmon            (24)
            repeat (2)                                                  (25)
                clockcct;         // call to the task clockcct          (26)
        end;                                                            (27)

task setmon;                      // header display and monitoring setup (28)
    begin                                                               (29)
        $display("          time clks K_in K_out v1-3 w1-3");           (30)
        $monitor ($time,,,,phi1,phi2,,,,,,shiftin,,,, shiftout,,,,,,
                  cct.v1, cct.v2, cct.v3,,,,,cct.w1, cct.w2);           (31)
                                                                        (32)
    end                                                                 (33)
endtask                                                                 (34)

task clockcct;                    // generate dual-phase clock waveform  (35)
    begin                                                               (36)
        #d phi1 = 1;      // for time step constant d see "parameter ..." (37)
        #d phi1 = 0;                                                    (38)
        #d phi2 = 1;                                                    (39)
        #d phi2 = 0;                                                    (40)
    end                                                                 (41)
endtask                                                                 (42)
endmodule                                                               (43)
```

Bild 11.15: Stimuli zur Simulation des Schieberegister n.Bild 11.14 (Verilog®-Beschreibung).

Bild 11.15 zeigt die Verilog®-Beschreibung des zur Simulation dieses Beispiels nötigen Stimuli-Generators. Bild 11.16 zeigt die Antwort des Simulators auf die Beaufschlagung mit den in Bild 11.15 beschriebenen Stimuli. Die Zeilen 30 bis 32 in diesem Testprogramm definierten das Format des in Bild 11.16 gezeigten tabellarischen Simulations-Listings.

Abschnitt 8.5 ist eine Kurzdarstellung von Verilog® in der Switching-Ebene. Da eine umfassende Einführung in Verilog® den Rahmen dieses Buches sprengen würde, sei bezüglich weiterer Einzelheiten auf die Literatur verwiesen, wie z. B. [14] [18].

11.4 nMOS-MOL-Schaltungen (tiled logic)

Wegen der Einfachheit ihrer Schaltungstechnik und ihres Layouts eignet sich nMOS-Technologie gut für einen strukturierten Entwurf zur Realisierung krauser Logik (unregelmäßige Lo-

time	clks	in	out	v1-3	w1-2
0	00	0	x	xxx	xx
100	10	0	x	xxx	xx
103	10	0	x	0xx	xx
106	10	0	x	0xx	1x
200	00	0	x	0xx	1x
300	01	0	x	0xx	1x
303	01	0	x	01x	1x
306	01	0	x	01x	10
400	00	0	x	01x	10
500	10	0	x	01x	10
503	10	0	x	010	10
506	10	0	1	010	10
600	00	0	1	010	10
700	01	0	1	010	10

Bild 11.16: Simulations-Resultat zu Bild 11.15.

gik, *glue logic*, sowohl Schaltnetze, als auch Schaltwerke [11]) durch die Verwendung sehr kleiner, einfacher Zellen, teilweise als Array (d. h. in Matrix-Anordnung). Solche Strukturen können aus gegebenen Boole'schen Gleichungen durch einfach zu implementierende **Modul-Generatoren** automatisch erzeugt werden (z.B.[12]). Solche Zellen-Verbunde werden wegen ihrer Zellen-Anordnung (eine Veranschaulichung erfolgt später) gelegentlich auch tiled logic [13] genannt (d. h. in Kacheln aufgeteilte Logik) oder Matrix-orientierte Logik (MOL [6]).

Kachel-Layout. Bild 11.18 ist ein Beispiel für eine Art "Kachel-Bibliothek" für die PLA-Synthese. Die bekanntesten Kachel-Layout-Strukturen für krause Logik sind der PLA (s.-Abschnitt 11.4.1) und der Weinberger-Array (Abschnitt 11.4.2). Kachel-Logik eignet sich auch zur Im-

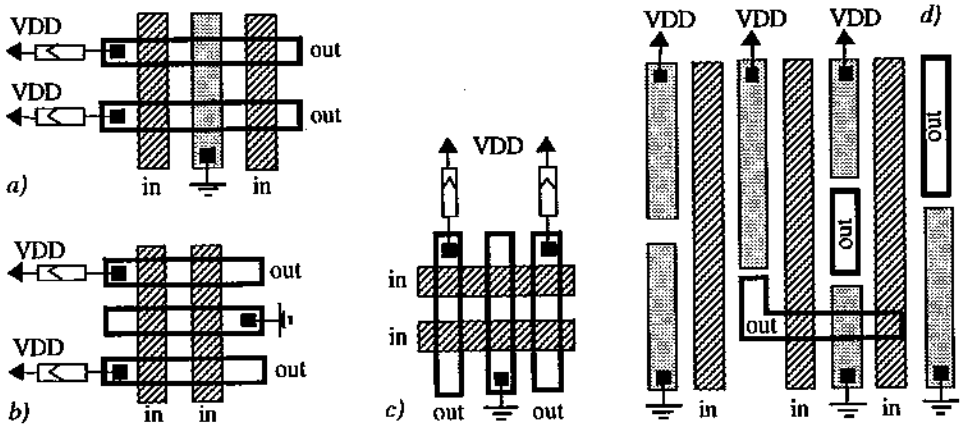


Bild 11.17: Basis-Layout-Schema: a) PLA, b-c) Weinberger-Array, d) Lopez-Law Gate-Matrix.

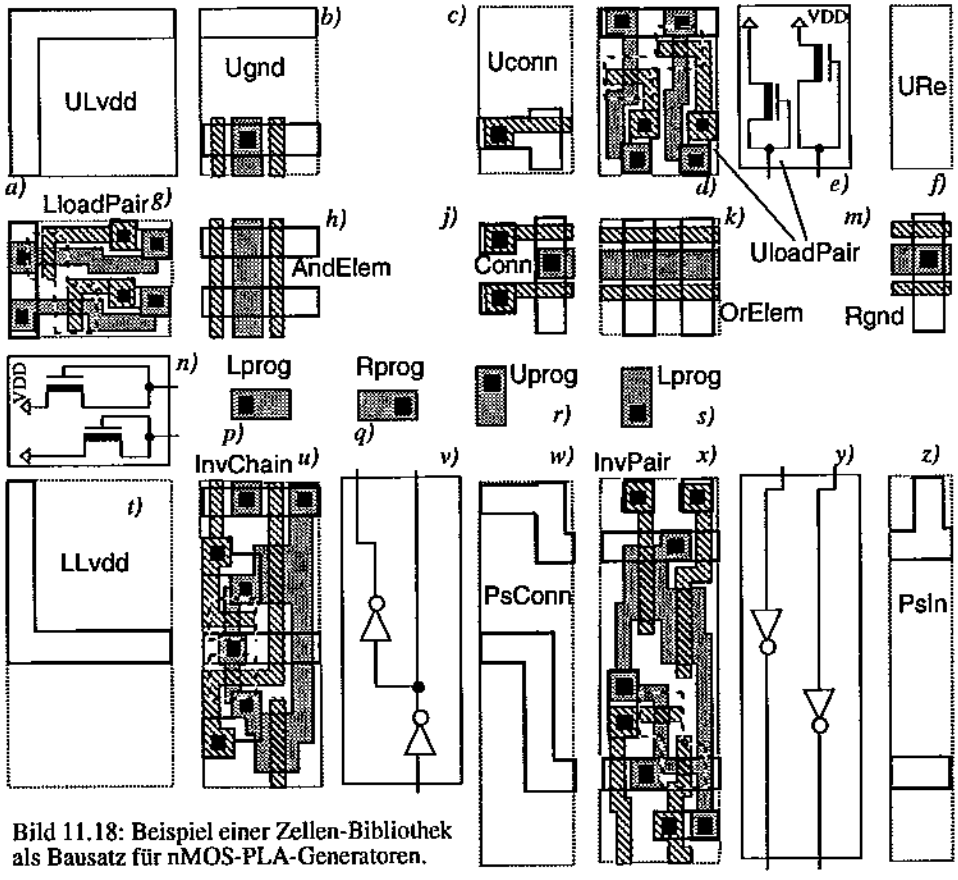


Bild 11.18: Beispiel einer Zellen-Bibliothek als Bausatz für nMOS-PLA-Generatoren.

plementierung von Petri-Netzen als Patil-Arrays (s. Kapitel 24). Neben der Kachel-Struktur ist eine regelmäßige, schematische Anordnung der Leiterbahnen auf dem Chip eine wesentliches Konzept zur Entwicklung einer die Modul-Generierung gut unterstützenden Zellenbibliothek (s. Bild 11.17). Etwas weniger Regelmäßigkeit weist das auch für CMOS-Schaltungen geeignete Layout-Schema nach Lopez und Law auf, genannt Lopez-Law-Gate-Matrix (Bild 11.17 d, s. auch Bild 18.27).

11.4.1 nMOS-PLAs

Das bekannteste Beispiel von Kachel-Logik ist der PLA (programmable logic array). PLA-Generatoren sind besonders leicht zu implementieren. Zu Beginn der akademischen Mead-&-Conway-Bewegung (s. auch Abschnitt 2.2) wurden deshalb Dutzende von PLA-Generatoren implementiert, sozusagen als Lehrstück zur Einführung von Methoden der Informatik in die Mikroelektronik: ein Schlüsselbeispiel zur Veranschaulichung der Grundlagen einer Basis-Innovation. Im folgenden werden Kenntnisse im Logik-Entwurf vorausgesetzt (z.B.[3], [2]).

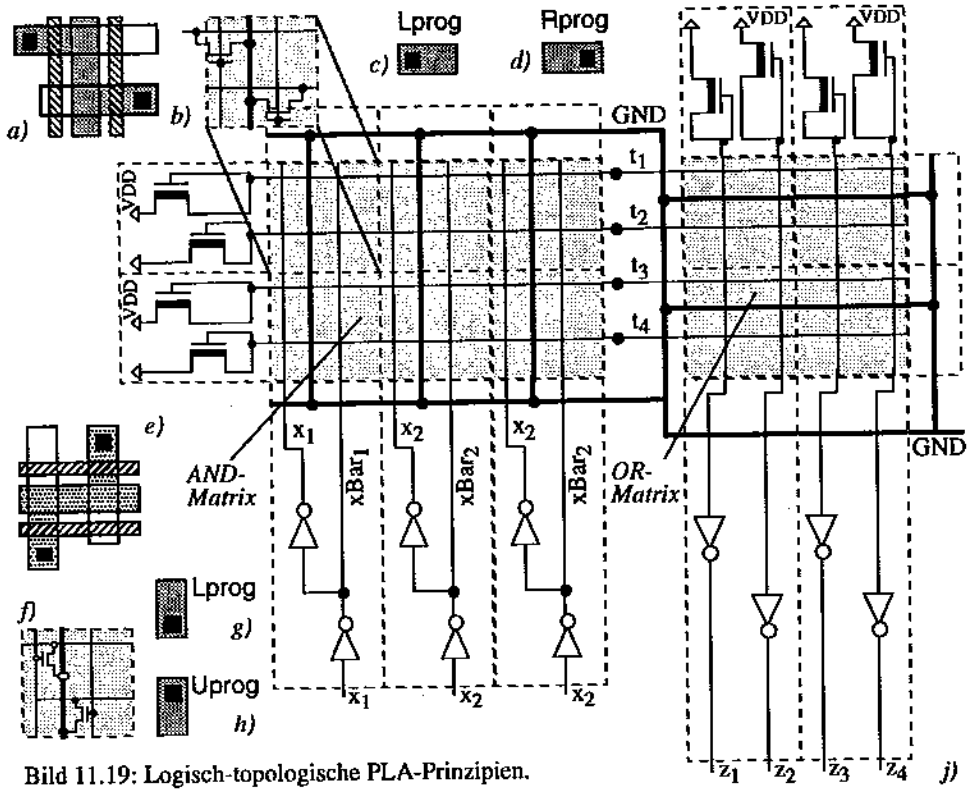


Bild 11.19: Logisch-topologische PLA-Prinzipien.

PLAs (s. a. [11]) dienen der **Realisierung Boole'scher Gleichungen** in disjunktiver Form (Disjunktion von Konjunktionen, wie beispielsweise: $z_1 = x_1 \wedge x_2 \vee \bar{x}_1 \wedge x_3$, wobei eine Konjunktion wie z. B. $x_1 \wedge x_2$ Term genannt wird, z. B.: t_1). Ein PLA ist nur von seinem außen sichtbaren Verhalten her eine AND-OR-Schaltung. Wegen der Realisierung durch Verhältnis-Logik und der damit eingeführten unerwünschten Negation eines jeden Gatters (s. Kapitel 7 und 8) muß aus Minimierungs-Gründen eine andere Schaltungstechnik verwendet werden.

PLA in (negierender) Verhältnis-Logik. Zunächst erscheint eine NAND-NAND-Realisierung oder eine NOR-NOR-Realisierung als sinnvoll. Die NAND-NAND-Realisierung leitet sich ab aus AND-OR (Bild 11.23 a) über folgende Stufen. Aus Regel k in Bild 7.3 folgt Bild 11.24 b. Aus der inversen Regel zu Regel g in Bild 7.3, angewandt auf die OR-Gatter, folgt Bild 11.24 c: die NAND-NAND-Lösung. Die NOR-NOR-Realisierung leitet sich ab aus Bild 11.24 b über folgende Stufen. Aus Regel g in Bild 7.3, angewandt auf das AND-Gatter, folgt Bild 11.24 d. Aus der inversen Regel zu Regel f in Bild 7.3, angewandt auf das OR-Gatter, folgt Bild 11.24 e: die NOR-NOR-Lösung.

NOR-NOR-Realisierung. Wegen hohem Flächenbedarf bei guter Schaltgeschwindigkeit (niederohmige Pulldown-Transistoren) kommt für integrierte Schaltungen die NAND-NAND-Lö-

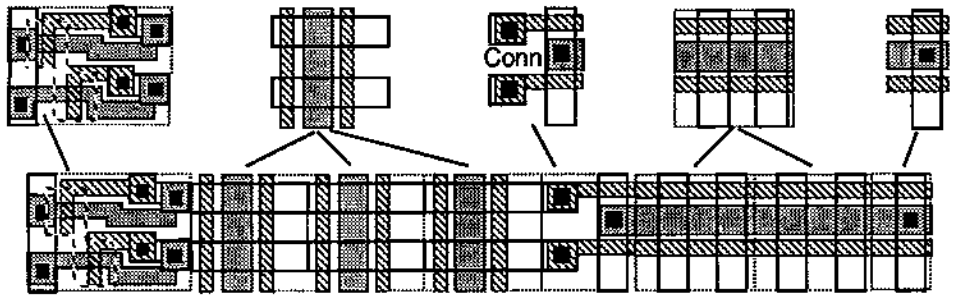


Bild 11.20: Konstruktion eines CenterLayer (PLA-Beispiel): a) Zellen-Satz, b) Zellen-Verbund.

sung nicht in Frage. Dies läßt sich leicht zeigen durch Betrachtungen zur Einhaltung der elektrischen Regeln (vgl. Seite 258 sowie Bild 12.20): ein großer Pulldown-Transistor würde die Basis-Zelle für die AND-Matrix und die OR-Matrix (vgl. Bild Bild 11.18 h und k) auf ein Mehrfaches aufblähen gegenüber der im folgenden beschriebenen NOR-NOR-Lösung. Bild 11.23 f-h erklärt das Zusammenspiel zwischen der AND-Matrix-Zelle und deren Programmierung (Bild 11.19 a-d) und dem Treiber ((Bild 11.18 v).

Kachel-Bibliothek. Bild 11.18 zeigt das Zellen-Repertoir (die Kachel-Bibliothek) für die Generierung von nMOS-PLAs. Bild 11.19 j zeigt den hierzu isomorphen Schaltplan eines PLA-Beispiels, welcher die zugrundeliegende Schaltungstechnik veranschaulicht. Aus Zellen nach Bild 11.18 h wird die AND-Matrix zusammengesetzt (vgl. Bild 11.19), wobei Plättchen nach Bild p oder q der "Programmierung" durch Einsetzen eines Transistors dienen (Bild 11.19 a-b zeigt eine programmierte AND-Zelle). Aus Zellen nach Bild 11.18 k wird die OR-Matrix zusammengesetzt, wobei Programmier-Plättchen nach Bild c oder d verwendet werden (Bild 11.19 e-f zeigt eine programmierte OR-Matrix-Zelle. Eine AND-Matrix-Zelle umfaßt jeweils zwei Terme, sodaß wir nach Hinzufügen des Pullup-Paares nach Bild 11.18 g und einer Matrix-Zeile zwei NOR-Gatter erhalten wie die linke Seite von Bild 11.20 zeigt.

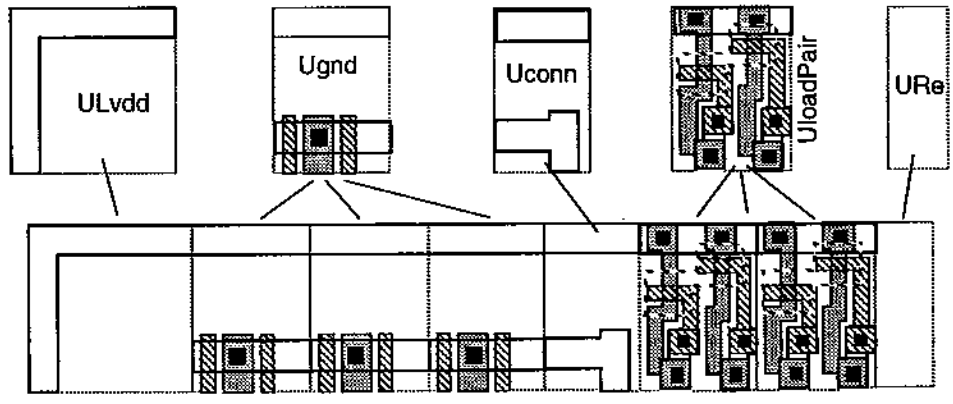


Bild 11.21: PLA-Beispiel: Konstruktion des UpperLayer: a) Zellen-Satz, b) Zellen-Verbund.

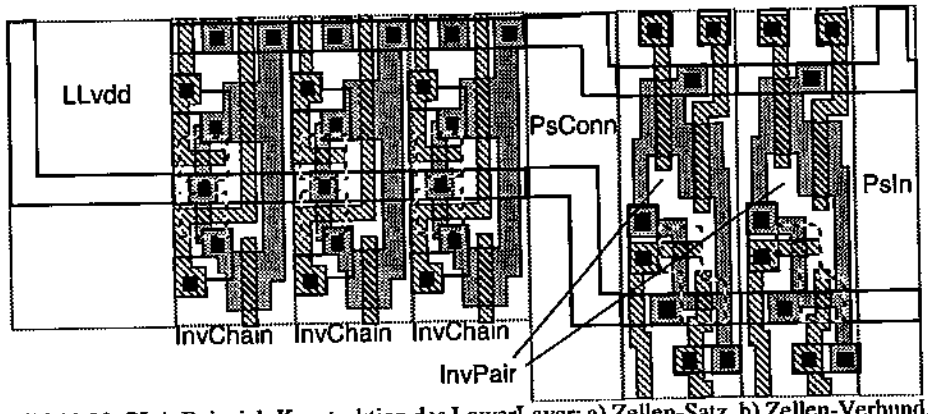


Bild 11.22: PLA-Beispiel: Konstruktion des LowerLayer: a) Zellen-Satz, b) Zellen-Verbund.

Zum Layout-Stil dieses PLAs ist interessant, daß das Pullup-Paar für die OR-Matrix (Bild 11.18 d) nur eine um 90° gedrehte Kopie des AND-Pullup ist (Bild 11.18 g). Auch die OR-Matrix-Zelle ist nur eine rotierte Kopie der AND-Matrix-Zelle. Man beachte auch, daß die Breiten bzw. Längen aller um die AND- und OR-Matrizen herum angeordneten peripheren Zellen genau dem Rapport (16·16 λ) dieser Matrizen angepaßt sind. Die Zelle *InvChain* enthält eine Inverterkette derart, daß am AND-Matrix-Eingang die Variable x_i jeweils nicht negiert und (als $xBar_i$) negiert vorhanden ist. Die Zelle InvPair enthält zwei elektrisch separate Inverter für ein Paar von Ausgangsleitungen aus der OR-Matrix. Für die horizontal von der AND-Matrix zur OR-Matrix verlaufenden Term-Leitungen wird eine Spalte von Konnekt-Zellen Conn benötigt (Bild 11.18 j) wegen des Überganges von Metall auf Poly. Sonstige periphere Zellen (PsConnn, PSin, LLvdd, ULvdd) werden zur Spannungsversorgung benötigt.

PLA als Zellen-Verbund. Bild 11.23 zeigt das Layout eines unprogrammierten, aber sonst vollständigen PLAs mit 3 Eingängen x_1 ... x_3, 4 Termen t_1 ... t_4 und 4 Ausgängen z_1... z_4 (vgl. auch Bild 11.19). Das Format dieses PLAs ist mit drei Parametern vollständig bestimmt: E = 3, T = 4, A = 4 (wobei E, T, oder A für die Anzahl der Eingänge, Terme, bzw. Ausgänge steht). Mit diesen Parametern läßt sich das Layout nach Bild 11.23 durch Abutment von Zellen aus Bild 11.18 automatisch generieren. Dieses Layout ist zusammengesetzt aus drei Layers UpperLayer (Bild 11.21), CenterLayer (Bild 11.20), und LowerLayer (Bild 11.22). Wir können die Synthese des obigen PLAs (zunächst noch ohne Programmier-Plättchen) vollständig angeben durch den folgenden KARL-3-Abutment-Ausdruck:

```
make PLA_344 (ULvdd : Ugnd :3: Uconn : rotr LoadPair :2: URe) @   (*UpperLayer*)
(LoadPair : AndElem :3: Conn : rotr AndElem :2: Rgnd) @2@          (*CenterLayer*)
(LLvdd : InvChain :3: PsConn : InvPair :2: Psin)                   (*LowerLayer*)
(right in GND, VDD, back X[1..3], out Z[1..4]);                    (*external interconnect*)
```

Abutment-Ausdrücke. Solche Abutment-Ausdrücke zusammen mit sogenannten Verdrahtungs-Operatoren (s. Abschnitt 20.4.1) eignen sich gut zur kompakten und präzisen Beschreibung von Kachel-Logik und somit auch für die Konzeption von Modul-Generatoren (und

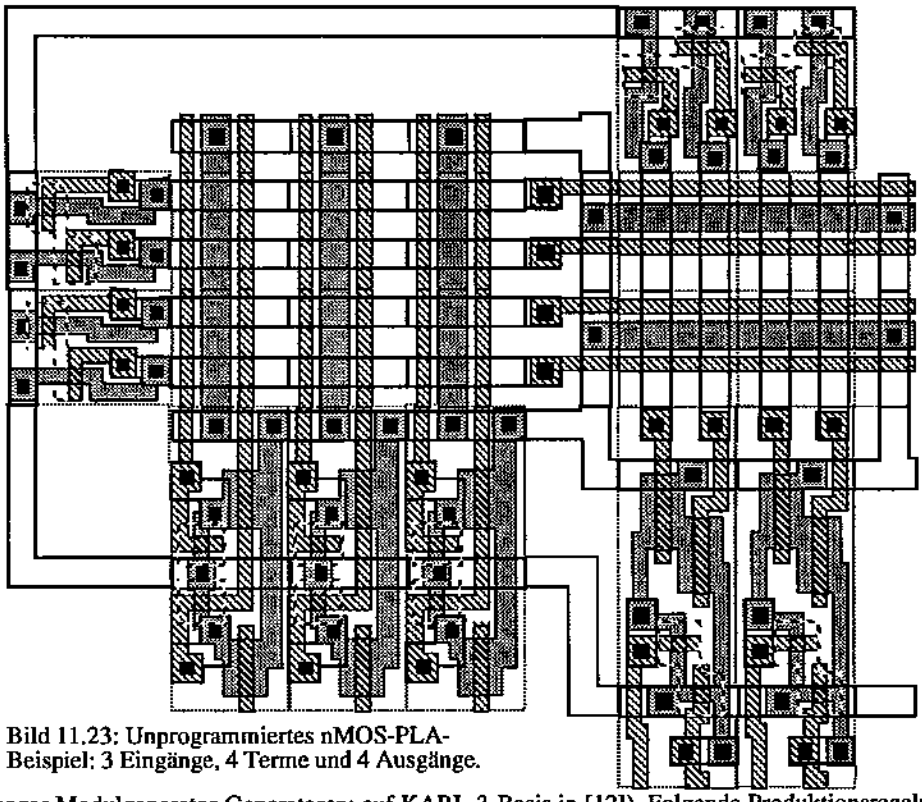


Bild 11.23: Unprogrammiertes nMOS-PLA-
Beispiel: 3 Eingänge, 4 Terme und 4 Ausgänge.

sogar Modulgenerator-Generatoren: auf KARL-3-Basis in [12]). Folgende Produktionsregeln definieren die Syntax für KARL-3-Abutment-Ausdrücke[7][8].

MAKE-STATEMENT... ::='**MAKE**' (LAYERS '(' (ACT-CELL-PARLIST)* ')' ';')+
LAYERS ..... ::=LAYERS '@' SLICES / SLICES .
SLICES ..... ::=SLICES ':' SLICE / SLICE .
SLICE ..... ::=IDENTIFIER ['&' DECIMAL-NUMBER] /
[TRANS-OP] '(' LAYERS ')' .
TRANS-OP ..... ::='**MIRX**' / '**MIRY**' / '**ROTL**' / '**ROTR**' / '**ROTU**' .

Hierbei steht der Operator ':' für Abutment nebeneinanderliegender Zellen und der Operator '@' für Abutment übereinanderliegender Zellen. Das Postfix-Symbol ':n:' bzw. '@n@' steht für das iterative Abutment von n Zellen-Exemplaren. Die Wortsymbole 'MIRX', 'MIRY', 'ROTL, 'ROTR', oder 'ROTU' stehen für 'spiegele die X-Koordinaten' ("flip x"), 'flip y', 'drehe (90°) nach links', 'drehe nach rechts', bzw. 'drehe um 180°'.

PLA-Programmierung. Der Begriff "Programmierung" bei PLAs ist historisch zu verstehen. Die ersten PLAs basierten Dioden-Matrizen (Dioden-Logik: s. Bild 6.20 a, g). Die "Programmierung" für gegebene Boole'sche Gleichungen erfolgte über ein spezielles Progammiergerät

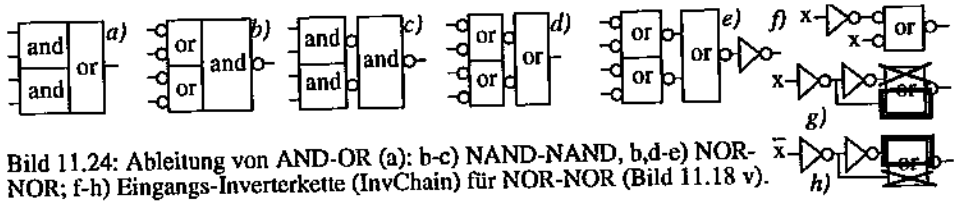


Bild 11.24: Ableitung von AND-OR (a): b-c) NAND-NAND, b,d-e) NOR-NOR; f-h) Eingangs-Inverterkette (InvChain) für NOR-NOR (Bild 11.18 v).

durch gezieltes Wegbrennen bestimmter präparierter Leitungs-Stücke (*fusable link*, bzw. "fuse" in Bild 11.25 d). Bei oben skizziertem PLA-Generator reduziert sich die Programmierung zu einem letzten Synthese-Schritt, nämlich dem Einsetzen der Transistor-Plättchen (Bild 11.19 c, d, g, h) in das bereits richtig formatierte "nackte" PLA-Layout (Bild 11.23).

Streifen-Modell der Gate-Matrix. Die Ableitung der Plazierung der Transistoren in das nackte PLA kann an einem abstrakten PLA-Modell vorgenommen werden. Bild 11.25 f zeigt das Streifen-Modell für den PLA nach Bild 11.23 und den 4 Boole'schen Gleichungen in Bild 11.25 a, die als Problemformulierung vorgegeben sind. Zuerst wird das PLA-Format ermittelt: Bild 11.25 a zeigt 3 Eingangsvariablen x_1 ... x_3, vier Terme t_1 ... t_4 und vier Ausgänge z_1 ... z_4 (Bild 11.25 b), was 4 Gatter der AND-Ebene und 4 Gatter der OR-Ebene ergibt entsprechend Bild 11.23 und Bild 11.25 f. Die Programmierung reduziert sich auf die Plazierung der richtigen Plättchen an den richtigen Punkten in dem durch die drei Format-Parameter mit (E+A)·T = (3+4)·4 großen gegebenen Koordinatensystem.

Personality-Matrix. Die Plazierung der Transistoren in das nackte Layout hinein kann durch die sogenannte *personality matrix* abstrahiert werden. Diese besteht aus zwei Untermatrizen, getrennt durch eine Kommata-Spalte (Bild 11.25 c). Das Alphabet besteht aus den drei Symbolen '1', '0', und '-'. Für die AND-Submatrix steht '1' für "Transistor links", '0' für "Transistor rechts", und '-' für "kein Transistor". Für die OR-Submatrix steht '1' für "Transistor", und '-' für "kein Transistor", wobei darauf zu achten ist, daß für ungeradzahlige Terme "Transistor oben" und für geradzahlige Terme "Transistor unten" zu wählen ist (vgl. auch Bild 11.25 g und Bild 11.19). Die Anordnung aus Bild 11.25 e und f zeigt die Zuordnung zwischen Personality-Matrix und dem Streifen-Modell.

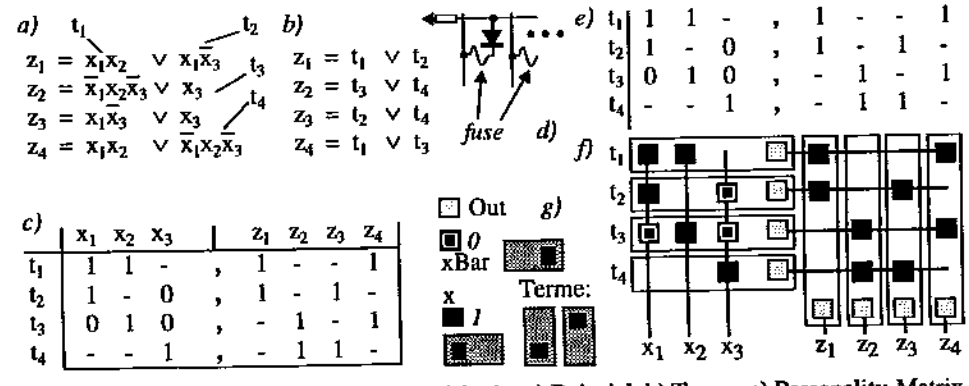


Bild 11.25: Zur Ableitung der Personality-Matrix: a) Beispiel, b) Terme, c) Personality-Matrix.

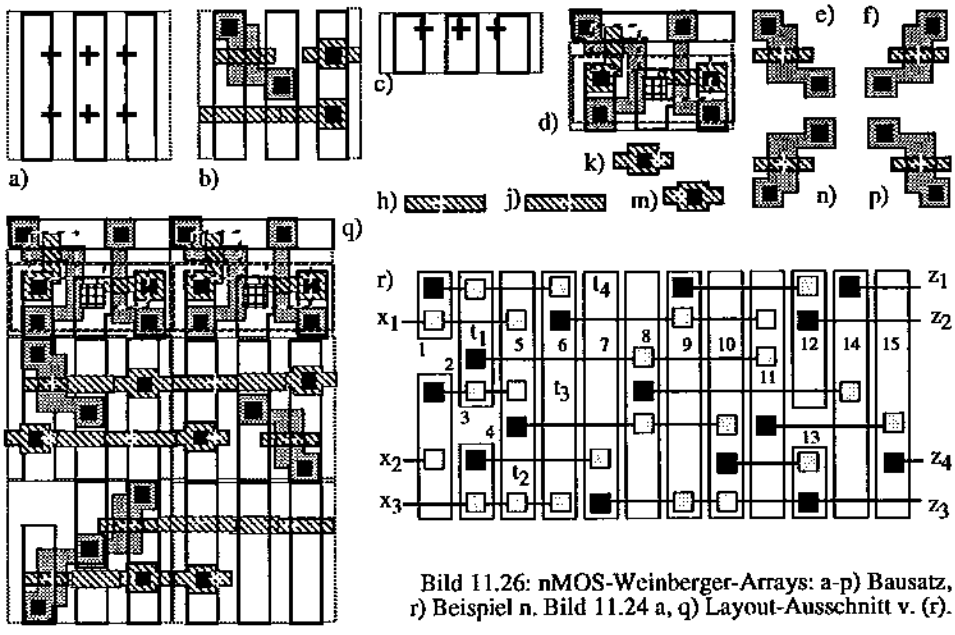


Bild 11.26: nMOS-Weinberger-Arrays: a-p) Bausatz, r) Beispiel n. Bild 11.24 a, q) Layout-Ausschnitt v. (r).

11.4.2 Der Weinberger-Array

Eine bezüglich Format und Form flexiblere Art von Kachel-Logik ist der Weinberger-Array [19]. Bild 11.17 c zeigt das zugrundeliegende nMOS-NOR-NOR-Layout-Schema. Im Gegensatz zum PLA sind hier alle Gatter senkrecht angeordnet, wie das Beispiel eines Streifen-Diagramm zeigt (Bild 11.26 r). Das Alphabet der Personality-Matrix ist wesentlich umfangreicher, da nicht nur Transistor-Plazierungen sondern auch Anfang, Ende und Verlauf waagerechter Poly-Leitungen, sowie die Plazierung von Gatter-Ausgängen angegebenen müssen. (Beim PLA gab es dieses Problem nicht: alle Leitungen haben einheitlich die volle Länge, sind also schematisch und nicht individuell und müssen deshalb nicht "programmiert" werden.) Zusätzlich gegenüber PLA (welches nur kennt: Transistor links, Transistor rechts, kein Transistor): sind individuell anzugeben: Ausgang (von links, von rechts, von beiden Seiten, kein Ausgang), Transistor (mit Eingang von links, von rechts, von beiden Seiten, kein Transistor) und durchgehende Leitung. Bild 11.27 a veranschaulicht das Alphabet und Bild e zeigt das Beispiel einer Personality-Matrix (PM). Bild 11.27 b-e zeigen die Zusammenhänge zwischen Schaltungstechnik, Streifen-Modell PM, und Gatter-Numerierung. Fehlerfreie PMs erfüllen Nachbarschafts-Abhängigkeiten: Bild 11.27 f zeigt legale Symbol-Paare, Bild g illegale Paare.

Weinberger-Array-Generator. Bild 11.26 zeigt Ansätze für einen Bausatz für nMOS-Weinberger-Arrays (geeignet für einen Weinberger-Array-Generator), der von einer Anordnung von Gatter-Paaren ausgeht. Bild 11.26 a zeigt einen 2×2 großen "nackten" Pulldown-Abschnitt für zwei Gatter und davon zwei Zeilen aus der Personality-Matrix, wobei der mittlere Metall-Streifen der Erdung dient. Bild b zeigt ein Programmierungs-Beispiel eines solchen 2×2-Abschnitts

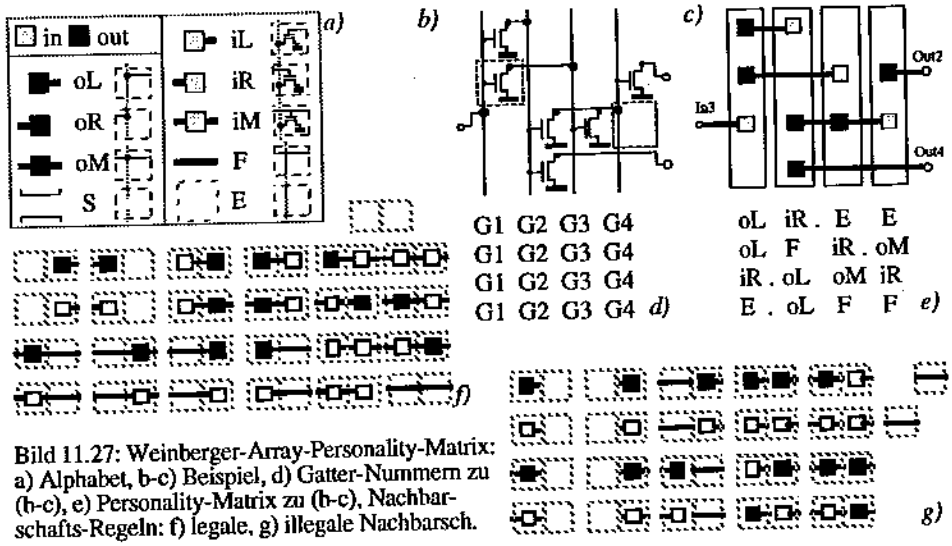


Bild 11.27: Weinberger-Array-Personality-Matrix: a) Alphabet, b-c) Beispiel, d) Gatter-Nummern zu (b-c), e) Personality-Matrix zu (b-c), Nachbarschafts-Regeln: f) legale, g) illegale Nachbarsch.

und Bilder e - p zeigen die Programmier-Plättchen, die zum Teil auch von der Lage innerhalb der 6 Positionen des 2×2-Abschnitts abhängen (Kreuze in Bild a). Bild 11.26 d zeigt ein Pullup-Paar, worin die zusätzliche GND-Zuleitung über eine zweite Metallisierung-Ebene realisiert ist.

Faltung. Wie PLAs [5] können auch Weinberger-Arrays zwecks Minimierung des Flächenbedarfs gefaltet werden (Beispiel in Bild 11.26 r). Dadurch können sich bei geringem Belegungsgrad zwei Gatter in eine Spalte des Gate-Array teilen. Der Pullup für das zweite Gatter wird dann von unten angeschlossen durch eine gespiegelte Version der Zelle aus Bild 11.26 d. Zur Separation der beiden Pullup-Leitung übereinanderliegender Gatter muß im Grenzbereich eine Programmier-Position frei bleiben (vgl. Bild 11.26 r).

Weitere Optimierungs-Möglichkeiten. Die Lage von Netzen kann innerhalb einer Weinberger-Gate-Matrix oft verändert werden, ohne daß dabei die logische Funktion des Array verändert wird (topologische Äquivalenz-Transformation). Beispielsweise das Netz z_2 in Bild 11.26 r kann um eine Zeilenposition nach unten verlagert werden und das Netz z_4 um zwei Positionen nach oben. Durch derart durchgeführte Verbesserung der Plazierungen der Netze können der Flächenbedarf vermindert werden, die Proportionen des Arrays verändert werden (etwa zur optimalen Einpassung in eine Layout-Umgebung), und die Lage der Ports verändert werden (etwa zur Vermeidung oder Minimierung angrenzender Routing-Flächen durch Erreichung von oder Annäherung an Abutment-Fähigkeit über *port matching*). Bild 11.28 veranschaulicht Operationsbeispiele eines hierzu geeigneten interaktiv graphischen Streifen-Modell-Editors (das obere Bild zeigt jeweils den Zustand des Streifen-Modells vor der Manipulation, das untere Bild den Zustand nach der Manipulation). Weitere Verfeinerung: nach Fertigstellung des Weinberger-Array gemäß obigem Schema kann quasi durch *-routing over the cell* - senkrecht verlaufende Poly-Leiter in den freien Zwischenräumen oft eine noch weiter gehende Optimierung erreicht werden. Es kann zusammengefaßt werden, daß der Weinberger-Array ein sehr flexibles Kon-

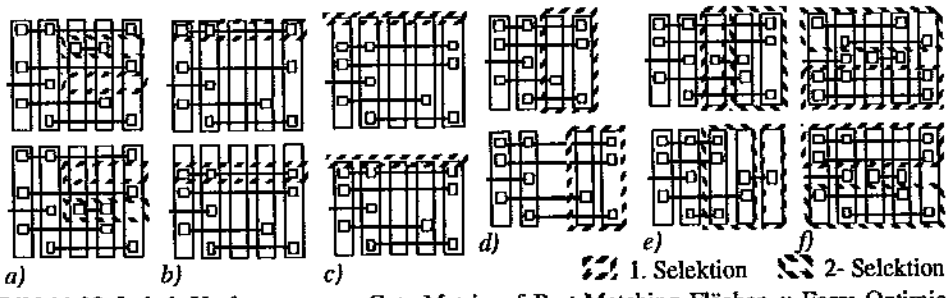


Bild 11.28: Lokale Umformungen an Gate-Matrizen f. Port-Matching, Flächen- u. Form-Optimierung: a) swap nets, b) move nets, c) stretch boxes, d) move boxes, e) swap boxes, f) swap nets.

zept für generisches Layout ist. Weinberger-Array-Generatoren lassen deshalb ein hohes Optimierung-Potential erwarten.

11.5 Literatur

[1] M. Anaratone: Digital CMOS Circuit Design; Kluwer Acad. Publishers; Boston, 1986

[2] R. Brayton et al.: Logic Minimization Techniques for VLSI Synthesis; Kluwer, 1984

[3] D. L. Dietmeyer: Logic Design of digital Systems; Allyn & Bacon, Bonston, 1971

[4] M. Elmasry: Digital Bipolar Integrated Circuits; A Wiley-Interscience; New York 1983

[5] G. Hachtel et al.: An Algorithm for Optimal PLA Folding; IEEE Trans. CAD 1,63 1982

[6] R. Hartenstein, R. Hauck: Functional Extraction from Personality Matrixes of MOL (Matrix-Oriented Logic) Circuits; IFIP CHDL'87, North Holland, Amsterdam, 1987

[7] R. Hartenstein: Introduction to KARL - a Hardware Description Language; IT Press, D-76607 Bruchsal, Germany, 1994

[8] R. Hartenstein: KARL-3 Reference Manual; Univ. Kaiserslautern 1986

[9] R. Hartenstein: KARL and ABL; in (ed.: J. P. Mermet): Fundamentals and Standards in Hardware Description Languages; Kluwer Academic Publishers, Boston, 1993

[10] R. Hauck: KARL-3 User Guide; Univ. Kaiserslautern, 1986

[11] C. Mead, L. Conway: Introduction to VLSI Systems; Addison-Wesley, 1980

[12] V. Moshnyaga, H. Onodera, K. Tamaru, H. Yasuura: A Language for Designing Data-Path Module Generators; IEEE Int'l Design Workshop "Russian Workshop'92", Moscow, Russia; also: IFIP WG 10.5 Workshop on Synthesis, Generation and Portability of Library Blocks for ASIC Design, Grenoble, France, Mar 1992; and: Proc. SASIMI'92 - Synthesis and Simulation Meeting and Int'l Exchange, Kobe, Japan, 1992

[13] M. E. Hofmann: Automated Synthesis of Multi-level combinational Logic in CMOS Technology; Memorandum No. UCB/ERL M85/53, UC Berkeley, 1 July 1985

[14] N. N.: Verilog® Reference Manual; CADENCE Design Systems, Inc., 1989

[15] N. N.: Programming Language Interface CADENCE Design Systems, Inc., 1990

[16] J. Newkirk, R. Mathews: The VLSI Designer's Library; Addison-Wesley, 1983

[17] D. A. Pucknell, K. Eshraghian: Basic VLSI Design; Prentice-Hall 1988

[18] D. Thomas, P. Moorby: The Verilog® Hardware Description Language; Kluwer, 1989

[19] A. Weinberger: Large-Scale Integration of MOS Complex Logic: A Layout Method; IEEE J. Solid State Circuits, SC-2, p. 182-190, 1967