Entwurf kundenspezifischer integrierter Schaltungen

von H.-G. Zipperer
(überarbeitete Version der Vorlesung von S. Rust)

Inhaltsverzeichnis

1 Einleitung

1.1 Einführung ASICs

Integrierte Schaltungen werden schon seit über 20 Jahren zum Aufbau von digitalen Systemen benutzt. Früher handelte es sich aber in den meisten Fällen um Standardbausteine (standard chips), die von den Halbleiterfirmen entworfen wurden. Die Halbleiterindustrie hatte schon sehr früh festgestellt, das eine integrierte Schaltung nur ökonomisch hergestellt werden kann, wenn sie in großen Stückzahlen zu verkaufen ist. Die Gründe dafür waren:

- Die Entwurfskosten waren sehr hoch. Es existierten noch keine strukturierten Entwurfsmethoden, keine computergestützten Hilfsmittel und keine speziellen Bausteine, die einen vereinfachten Entwurf ermöglichten.
- Die relativ hohen Kosten der Masken, des Herstellungsprozesses und des Tests integrierter Schaltungen waren nur bei höheren Stückzahlen sinnvoll.

Es wurden deswegen die Logikfamilien, z. B. 74xx, mit typischen, weitverbreiteten Logikfunktionen entwickelt. Jeder Baustein dieser Logikfamilie enthält nur eine kleine Anzahl von Gattern oder Flip-Flops (SSI: Small Scale Integration) und kann in vielen unterschiedlichen digitalen Systemen verwendet werden (Bild 1).

In den 60er Jahren wurde die IC-Technologie weiterentwickelt, man konnte immer mehr Funktionen auf einem Chip unterbringen. Statt wenigen Gattern (= 10 bis 20 Transistoren) wie bei der SSI konnten nun mehrere hundert Transistoren untergebracht werden (MSI: Medium Scale Integration). Damit konnten komplexere logische Subsysteme wie Binärzähler oder Multiplexer auf einem Chip realisiert werden. Die Funktion von MSI-ICs war schon wesentlich spezialisierter als bei den grundlegenden SSI-ICs. Das machte aber nicht so viel aus, da man versuchte, möglichst häufig vorkommende Funktionen zu integrieren.

Mit weiter fortschreitender Technologie war es dann Anfang der 70er Jahre möglich, integrierte Schaltungen mit mehreren tausend Transistoren herzustellen (LSI: Large Scale Integration). Bei dieser großen Anzahl von Transistoren wurde es immer schwieriger, Bausteine zu entwerfen, die sich häufig genug verkaufen ließen (z. B. Taschenrechner und Digitaluhren). Andere Systeme waren meist zu spezialisiert, um sie in größeren Stückzahlen verkaufen zu können.

Beispiel LSI-Uhren-IC:
Entwicklungskosten 100 000 Dollar. Es müßten also ungefähr 100 000 Stück verkauft werden, um einen realistischen Preis zu erzielen.

Nur bei Systemen, die in sehr großen Stückzahlen benötigt wurden und bei denen Bedingungen wie kompakter Aufbau und geringer Stromverbrauch erfüllt werden mußten, wurde von größeren Firmen der Entwurf einer speziellen integrierten Schaltung durchgeführt (custom chips).

Ein wichtiger Durchbruch war dann der Mikroprozessor. Er ist ein Standard-LSI-Baustein, der in großen Stückzahlen verkauft werden kann, da die Funktion nicht vollständig vom Hersteller vorbestimmt ist, sondern vom Anwender programmiert wird (Bild 1).

Mikroprozessoren eignen sich aber durch ihre sequentielle Arbeitsweise nicht zur Bearbeitung hochgradig paralleler Probleme mit hoher Geschwindigkeitsanforderung. Das bedeutet, daß es öfters keine zufriedenstellende Alternative zum Entwurf einer integrierten Schaltung nach den speziellen Wünschen eines Anwenders gibt, vorausgesetzt daß dies ökonomisch vertretbar ist. Lange Zeit gab es dann nur diese zwei Möglichkeiten:

- Verwendung von Standardbausteinen (SSI / MSI / LSI) mit dem Nachteil des hohen Platzbedarfs und der Fehleranfälligkeit, da viele diskrete Einheiten bei der Realisierung komplexer Funktionen notwendig waren, bzw. bei Mikroprozessoren das Problem der Geschwindigkeit und Parallelität der Bearbeitung.
- Entwurf eines voll kundenspezifischen Bausteins (custom chip), der eine optimale Lösung bezüglich Platzbedarf bietet, aber sehr hohe Entwicklungskosten hat.

Die Technologie der Herstellung integrierter Schaltungen entwickelte sich immer schneller, es wurden integrierte Schaltungen mit zigtausenden Transistoren (VLSI: Very Large Scale Integration) möglich. Somit wurde auch das Entwurfsproblem für integrierte Schaltungen immer größer. Die Entwurfskosten stiegen sehr stark an und es wurde immer schwieriger, halbwegs fehlerfreie integrierte Schaltungen herzustellen. Man befürchtete, daß die hohen Integrationsdichten bald nicht mehr genutzt werden könnten, da man nicht in der Lage sei, funktionierende Schaltungen mit hunderttausend Transistoren zu annehmbaren Kosten zu entwerfen. Es gab auch zu wenige Designer, die die nötige Erfahrung hatten, um korrekte Entwürfe zu erstellen.

Beispiel Mikroprozessoren:

Typ	Wortlänge	Jahr	Transistoren	Entwicklungzeit (Mannjahre)
Intel 4004	4 bit	1971	2 300	1
Intel 8086	16 bit	1979	30 000	13
Motorola 68000	16 / 32 bit	1979	70 000	52
iAPX432	32 bit	1981	200 000	> 100
i860	64 bit	1989	1 Mio.	< 100

Der Designer eines integrierten Bausteins hatte bislang die einzelnen Transistoren eines Chips mit der Hand entworfen. Ein Durchbruch wurde durch die Entwicklung von Entwurfsmethoden erreicht, die es auch Designern mit relativ wenig Wissen von der Technologie der Chipherstellung ermöglichte, den Entwurf komplexer digitaler integrierter Schaltungen durchzuführen. Der Entwurf wurde verlagert vom Entwurf einzelner Transistoren auf Chipebene zu einem Entwurf in einer höheren, mehr funktional orientierten Abstraktionsebene. Die Entwicklung von immer leistungsfähigeren Entwurfsrechnern (z. B. Workstations) mit entsprechenden Entwurfstools für den computerunterstützten Entwurf von integrierten Schaltungen (CAE: Computer Aided Engineering) brachten weitere Fortschritte, so daß sich in den letzten Jahren eine radikale Änderung in der Einschätzung der Möglichkeiten ergab. Wurde es noch 1980 als unmöglich angesehen, komplexe Schaltungen mit 50 000 Transistoren in wenigen Wochen zu entwerfen und galten Chips mit 1 Million Transistoren als nicht nutzbar, so erreichte man schon 1983 durch die neuen Entwurfsmethoden wesentlich verkürzte Entwurfszeiten auch bei sehr komplexen Chips. An Universitäten lernen Studenten in wenigen Wochen, selber Chips zu entwerfen. Diese Entwicklung hat starke Auswirkungen auf den Entwurf digitaler Systeme in der Industrie und an den Universitäten. Die Vorteile von integrierten Schaltungen können bei wesentlich kleineren Stückzahlen zu annehmbaren Preisen genutzt werden.

Beispiel:
Beim oben erwähnten i860 wurden ca. 10 000 Transistoren manuell plaziert und verdrahtet, weitere ca. 40 000 mit automatischen Tools. Alle übrigen Transistoren wurden durch Vervielfältigung von Standardfunktionen erzeugt (vgl. auch Bild 2).

Die neuen Entwurfsverfahren beruhen auf der Verwendung vorentworfener Funktionsgruppen (Zellen), so daß sich der Designer nicht mehr um die Eigenschaften eines einzelnen Transistors kümmern muß. Der Nachteil ist natürlich ein Verlust an Flexibilität und eine geringere Packungsdichte. Je einfacher und schneller der Entwurf durchgeführt werden kann, desto größer sind im allgemeinen diese Nachteile.

Dieses Konzept beeinflußte auch die Entwicklung der halbkundenspezifischen (semi-custom) ICs. Das sind Bausteine mit vorgefertigten Grundmustern, die kostengünstigeren Entwurf und Herstellung von integrierten Schaltungen ermöglichen (Bild 1). Die Idee für solche Bausteine gab es schon in den 60er Jahren. Sie wurden aber erst in größerem Rahmen eingesetzt, als sie durch den steigenden Bedarf an höher integrierten Schaltungen interessant wurden.

Man unterscheidet heute Full-Custom- und Semi-Custom-Entwurf:

Full-Custom: Vollentwurf für einen speziellen Anwender. Entwurf weitgehend auf Transistorebene, es wird eine optimale Packungsdichte ange-

strebt.

Semi-Custom: Entwurf von Schaltungen unter Verwendung von vorentworfenen, allgemein verwendbaren Schaltungsteilen.

Die wichtigsten Vorteile der Verwendung kundenspezifischer und halbkundenspezifischer Bausteine sind:

- Reduzierung von Raumbedarf und Gewicht: Durch Zusammenfassung mehrerer Bauteile zu einem einzigen IC erreicht man eine wesentliche Reduzierung der Bauteileanzahl des Endprodukts. So läßt sich unter Umständen der ursprüngliche Platzbedarf von einer Europakarte mit Standard-ICs auf einen einzigen Chip reduzieren. Dies führt wiederum zur Vereinfachung der nächsthöheren Packungsebene. Die Platinen können vereinfacht werden oder die Logik mehrerer Platinen auf einer einzigen untergebracht werden. Insgesamt ergeben sich so Einsparungen bei Entwurf, Fertigung, Bestückung und Test. Leichtere und kleinere Systeme ermöglichen einen Einsatz in völlig neuen Einsatzgebieten.

- Reduzierung der Verlustleistung: Durch Verwendung von höherintegrierten anwendungsspezifischen ICs statt Standard-Bausteinen kann eine wesentliche Verringerung der Verlustleistung erreicht werden.

  Beispiel Impulsgenerator mit 25 Standardbausteinen:

ECL Standard-ICs	8 W
STTL Standard-ICs	6 W
LSTTL Standard-ICs	1.8 W
1 Gate-Array	0.035 W

  Dies ermöglicht dann auch Anwendungen wie tragbare, batteriebetriebene Geräte. Es ergeben sich weniger Kühlprobleme und höhere Zuverlässigkeit, da die Zuverlässigkeit mit höherer Betriebstemperatur sinkt.

- Erhöhung der Systemzuverlässigkeit: Durch die Zusammenfassung vieler ICs zu einer einzigen integrierten Schaltung reduziert sich die Anzahl der Bauteile und auch die Anzahl der Verbindungsleitungen, Lötstellen und Verbindungselemente beträchtlich. Dadurch sinkt die Anzahl der Fehlerquellen sehr stark und es folgt eine Steigerung der Zuverlässigkeit (MTBF 5- bis 20-mal größer).

- Schutz der Schaltung vor Nachbau: Eine Systemrealisation durch Standard-ICs mit bekannter Funktion ist recht einfach zu kopieren. Die Verwendung von anwendungsspezifischen ICs macht es einem Konkurrenten nahezu unmöglich, die exakte Auslegung der Schaltung nachzuvollziehen. Dies bringt unter Umständen einen Vorsprung vor anderen Wettbewerbern und einen entsprechenden Wettbewerbsvorteil.

- Vereinfachung der Lagerhaltung, Dokumentation und Service: Dies ist bedingt durch die wesentliche Reduzierung der Anzahl der verwendeten Bauteile.

- Reduzierung der Systemkosten: Entscheidend dafür sind nicht nur die Bauelementekosten sondern die Gesamtkosten des Systems unter Berücksichtigung der oben genannten Fakten. Je nach verwendetem IC-Typ ergeben sich unterschiedliche Designkosten und Stückkosten, so daß sich auch je nach verwendetem Baustein eine unterschiedliche Stückzahl ergibt, ab der sich die Verwendung lohnt. Unter Umständen kann aber ein anderer oben genannter Grund so wichtig sein, daß das Kriterium der Stückzahl nicht unbedingt ausschlaggebend ist.

Diese Gründe haben mit zu einer starken Ausbreitung des Entwurfs von kundenspezifischen integrierten Schaltungen, auch anwendungsspezifische integrierte Schaltungen (ASIC: Application Specific Integrated Circuit) genannt, geführt. Die geschätzte jährliche Zuwachsrate für ASICs beträgt weltweit 32 %.

Als ASICs werden hauptsächlich folgende Chiptypen bezeichnet (Bild 3):

PLD (Programmable Logic Device): für den unteren Komplexitätsbereich, kann vom Anwender selber programmiert werden (z. B. PLA).

Gate Array: vorgefertigtes IC, dessen Grundstruktur schon festliegt. Nur die Verbindung der Grundelemente wird kundenspezifisch vom Hersteller vorgenommen. Dadurch vermindert sich die Anzahl der Herstellungsschritte und der Entwurfsaufwand.

Standardzellen-IC: Dabei sollen die Vorzüge der Vollkunden-ICs und der Gate-Arrays kombiniert werden. Der Entwurf wird vollständig aus vorgefertigten Funktionsmakros aus einer Bibliothek zusammengesetzt, die aber relativ frei auf dem Chip plaziert und verbunden werden können.

Diese ASIC-Typen unterscheiden sich vor allem in Chipkomplexität, Designaufwand, Design-Flexibilität und Wirtschaftlichkeit als Funktion der Stückzahl.

1.2 Geschichtlicher Überblick

30er Jahre	Entdeckung des Feldeffekts in Si
1947	Erfindung des Transistors (Bell Labs; Ge)
1954	Si-Transistor (TI)
1958	erste integrierte Schaltung (TI; Ge)
	Si-Planar-Prozeß (Fairchild)
1959	MOS-FET
60er Jahre	SSI: $\leq$ 10 Gatter auf einem Chip
	Standard-Logikfamilien:
	RTL (Resistor Transistor Logic)
	DTL (Diode Transistor Logic)
	TTL (Transistor Transistor Logic; 74xx-Familie 1963)
1963	CMOS-Technologie (RCA)
	MSI: 10-100 Gatter auf einem Chip
	komplexere Standardschaltungen wie Zähler, Register, Multiplexer oder einfache ALUs
70er Jahre	LSI: 100-100 000 Gatter auf einem Chip
	Speicherchips mit > 1Kbit Speicherkapazität
1971	4-bit Mikroprozessor (Intel 4004; PMOS)
1972	8-bit Mikroprozessor (Intel 8008; PMOS)
1974	8-bit Mikroprozessor in CMOS (RCA 1802)
1975	12-bit Mikroprozessor in CMOS (IM 6100)
1976	16-bit Mikroprozessor (TI TMS9900; NMOS)
1978	64 Kbit DRAM
	Mead / Conway-Methode
	Johannson: Bristle Blocks (Silicon Compiler)
80er Jahre	VLSI: > 100 000 Gatter, > 1 Mio. Transistoren auf e. Chip
	Spezielle Bausteine für ASICs: PLD, Gate-Arrays
	Design Automation: Automatisierung des Entwurfsvorgangs
	Silicon Compiler: Automatische Umsetzung der funktionalen Beschreibung eines Bausteins in das Layout einer integrierten Schaltung
1980	32-bit Mikroprozessor (HP 9000; NMOS)
1981	256 Kbit DRAM
1983	one-month chip
1986	1 Mbit DRAM
1989	4 Mbit DRAM
	64-bit Mikroprozessor (Intel i860; CMOS)

(vgl. auch Bild 4)

Vorschau auf die 90er Jahre:

- one week chip (Laser-Direktschreibeverfahren ohne Maske)

- 10 Millionen bis 1 Milliarde Transistoren auf einem Chip
- 100 000 IC-Designer gesucht
- Markt für ASICs steigt um das 20-fache
- Verbreitung von Silicon Compilern
- Expertensysteme unterstützen Chip-Entwurf

1.3 Entwurfs- und Fertigungsmöglichkeiten

An den Universitäten gab bzw. gibt es mehrere Projekte, die den Entwurf und die Fertigung von integrierten Schaltkreisen unterstützen.

E.I.S-Projekt (Entwurf integrierter Schaltungen):
Verbundprojekt von Hochschulen, der Gesellschaft für Mathematik und Datenverarbeitung und der Fraunhofer-Gesellschaft. Kooperation mit AEG, Siemens und Telefunken. Laufzeit: 1983-1987. Chipfertigung: CMOS-Gate-Arrays, Standardzellen und Full-Custom-Entwurf. Ziele:

- Entwicklung von Programmen für den rechnerunterstützten Entwurf
- Entwurf von Schaltungen für Forschungs- und Lehrzwecke
- Verstärkung der Ausbildung von Informatikern und Elektrotechnikern
- Lösung der technischen, logistischen und finanziellen Probleme der Fertigung in kleinen Stückzahlen

DARPA (USA) Fast Turn Around Fabrication: MOSIS (MOS Implementation Service)

NORCHIP: seit 1981. Teilnehmer: Norwegen, Finnland, Schweden, Dänemark. 10 Universitäten, 11 Institute, 13 Industriefirmen. 9 Fabrikationen pro Jahr.

Teilweise werden die Entwürfe auf Multiprojektchips zusammengefasst (Bild 5).

1.3.1 Beispiele

Einige Beispiele sollen zeigen, welche ASIC-Entwürfe an Universitäten durchgeführt wurden:

- Johann-Wolfgang-Goethe-Universität Frankfurt, Technische Informatik: Entwurf eines Stack-Controllers in Standardzellen-Technik: 209 Standardzellen. Assoziativspeicher: 428 Zellen (entspr. 1004 Gatter).
- Fachhochschule Furtwangen: Druckersteuerung auf Gate-Array.
- Universität Stuttgart, Institut für Informatik: Gate-Array IFI-ST2: dezentrale Kreuzschienensteuerung für Multiprozessorsystem. 2 Knotensteuerungen auf einer Europakarte bei Verwendung von LS-TTL. 4 Knoten wurden auf ein Gate-Array mit 3080 Transistoren integriert.

Beispiele von Industrieentwürfen:

- FE2200 PC-Bus Monochrome Display Controller (Gate-Array): ersetzt 36 Logik-Chips.
- CS8220 Chip-Set (5 Gate-Arrays): ersetzen 63 der 94 ICs, die zum Aufbau einer IBM PC/AT CPU-Platine nötig sind.
- PEGA 1: Ein-Chip-Implementierung der IBM EGA-Karte (8000 Gatter auf einem Gate-Array).
- PLDs verwendet in MV 8000 von Data General, Gate-Arrays in VAX 11/750 von DEC.

2 Halbleitertechnologie

2.1 Überblick

Definition 'Integrierte Schaltung': Alle Bauteile (Transistoren, Dioden, Kapazitäten und Widerstände) und die Verdrahtung einer Schaltung werden in einem Herstellungsprozeß als integrierte Teile (Zonen) eines Halbleiterchips hergestellt.

Die einzelnen Integrationstechniken unterscheiden sich durch:

- Integrationsdichte
- Schaltgeschwindigkeit
- Verlustleistung

Vergleichskriterium ist das Produkt aus Schaltzeit und Verlustleistung (power-delay product, in Ws).

Alle wichtigen Technologien beruhen derzeit auf der Verwendung von Silizium (Si). Germanium (Ge) ist als Halbleitermaterial weitgehend veraltet, andere Technologien wie Galliumarsenid (GaAs) stehen erst am Anfang ihrer Entwicklung. Aus kommerziellen Gründen (z. B. wegen des Problems der second source) gibt es eine gewisse Standardisierung der verwendeten Technologien. Es werden hauptsächlich noch die geometrischen Ausmaße der einzelnen Transistoren immer weiter verkleinert (scaling). Die wichtigsten Technologien sind:

Bipolartechnik: Bipolartransistor, sehr kurze Gatterlaufzeiten, geringe Integrationsdichten, hohe Verlustleistung, aufwendige Herstellung

TTL (Transistor-Transistor Logic)

ECL (Emitter Coupled Logic): sehr schnell, hohe Verlustleistung

Unipolartechnik: MOS-Feldeffekttransistor (MOSFET), Transistor 10- bis 100-mal kleiner als bei Bipolartechnik, höhere Integrationsdichte, einfachere Herstellung, langsamere Schaltzeiten

PMOS (P-Channel Metal Oxide Semiconductor): nur p-Kanal-Transistoren verwendet, veraltet

NMOS (N-Channel Metal Oxide Semiconductor): nur n-Kanal-Transistoren verwendet

CMOS (Complementary Metal Oxide Semiconductor): n-Kanal- und p-Kanal-Transistoren, geringe Verlustleistung

Für Standarddesignverfahren wird meist ECL oder CMOS, für Full-Custom auch noch NMOS (wegen optimalerer Schaltungsmöglichkeiten) verwendet (Bild 6).

2.2 Aufbau einer integrierten NMOS-Schaltung

Der MOS-Transistor gehört zu den spannungsgesteuerten Halbleitern, den Feldeffekttransistoren (FET). Er besteht aus einem Halbleiter und einer Steuerelektrode, die durch eine isolierende Oxidschicht getrennt sind (MOS = Metal Oxide Semiconductor). Der Halbleiter besteht aus zwei hochdotierten Anschlußzonen (Source und Drain) und einem dazwischenliegenden niedrig dotierten Bereich mit einer entgegengesetzten Dotierung (Kanal). Durch das niedrig dotierte Gebiet kann kein Strom fließen. Wird an die Steuerelektrode (Gate) eine Spannung gelegt, so wird eine Ladung influenziert und es bildet sich ein leitender Kanal zwischen Source und Drain (Inversion). Je nach Dotierung von Source und Drain handelt es sich um einen n-Kanal oder p-Kanal; daraus leitet sich die Bezeichnung ab (Bild 7).

Eine integrierte NMOS-Schaltung besteht aus mehreren Schichten von leitenden Materialien, die durch isolierendes Material getrennt sind:

- Metall (Aluminium)
- Polysilizium (polykristallines Si)
- Diffusion (n-Dotierung mit Arsen, Phosphor)

Als Substrat dient ein schwach p-dotierter Si-Einkristall. Die Metallebene hat keine Auswirkung auf die anderen Ebenen, sie wird zur Verdrahtung verwendet. Die Verbindung zu den anderen Ebenen wird durch Kontakte in der Isolierung erreicht.

Ein NMOS-Transistor wird durch die Überlappung von Polysilizium und Diffusion gebildet. Das Gate (Polysilizium) steuert den Stromfluß in der darunterliegenden Diffusion. Übersteigt die Gate-Source Spannung V_{gs} die Durchschaltspannung V_{th} (typ. 0.2 V_{dd}, positive Versorgungsspannung), so wird der leitende Kanal zwischen Source und Drain gebildet (enhancement mode transistor, Transistor vom Anreicherungstyp, selbstsperrend; vgl. Bild 8).

Die Eigenschaften des Transistors sind abhängig von der verwendeten Technologie (Implantierung) und den geometrischen Ausmaßen (Bild 9). Es gilt:

- Schaltzeit: $\tau = \frac{L}{v}$. Mit μ = Mobilität der Ladungsträger im Kanal, typ. 500 cm^2/Vs, wird $\tau = \frac{L}{\mu E} = \frac{L^2}{\mu V_{ds}}$.
- Gate-Kapazität: $C_g = \frac{\epsilon W L}{D}$ (ϵ = Dielektrizitätskonstante des Isolators, typ. 35×10^{-14} F/cm).
- Source-Drain-Strom: $I_{ds} = \frac{Q}{\tau}$. Mit $Q = C_g(V_{gs} - V_{th})$ wird $I_{ds} = B\frac{W}{L}(V_{gs} - V_{th})V_{ds}$ ($B = \frac{\mu\epsilon}{D}$ = konst.).

- Drain-Source-Widerstand: $R_{ds} = \frac{V_{ds}}{I_{ds}} = \frac{L^2}{\mu C_g (V_{gs} - V_{th})}$

Wenn $V_{ds} > (V_{gs} - V_{th})$ kommt der Transistor in den Sättigungsbereich, es gilt dann: $I_{ds} = \frac{BW}{2L}(V_{gs} - V_{th})^2$.

2.3 Grundschaltungen in NMOS-Technik

2.3.1 Inverter

Für den Aufbau eines Inverters ist zusätzlich noch ein pull-up-Widerstand erforderlich. Aus Platzgründen wird dieser nicht durch eine lange Poly-Leitung o. ä., sondern durch einen depletion mode transistor realisiert. Beim depletion mode transistor (Transistor vom Verarmungstyp, selbstleitend) ist die Durchschaltspannung $V_{dep} < 0$ V. Dies wird durch eine besondere Dotierung (Ionenimplantation mit Bor) bei der Herstellung erreicht. Wird der depletion mode transistor als pull-up verwendet, so wird das Gate mit der Source verbunden, so daß der Transistor immer durchgeschaltet ist (Bild 10).

Die Widerstände des pull-up-Transistors und des pull-down-Transistors müssen so abgestimmt werden, daß für die nachfolgenden Schaltungsteile entsprechende eindeutige Logikpegel über bzw. unter der Durchschaltspannung entstehen. Insbesondre ist der L-Pegel vom Widerstandsverhältnis der Transistoren abhängig. Es muß gelten: $V_{out_L} < V_{th}$ und $I_{ds_{pd}} = I_{ds_{pu}}$.

Der pull-up-Transistor arbeitet immer im Sättigungsbereich; der pull-down-Transistor hier nicht. Mit $Z = W/L$ und $V_{in} = V_{dd}$ gilt: $BZ_{pd}(V_{dd} - V_{th})V_{out} = \frac{B}{2}Z_{pu}(-V_{dep})^2$. Meist wird $V_{th} = 0.2\ V_{dd}$ und V_{dep} = -0.6 V_{dd} gewählt. Bei $V_{out_L} \approx 0.2$ - 0.3 V erhält man $k := Z_{pd}/Z_{pu} \approx 4$ - 5.

Eine andere Analyse führt über die logische Schwellspannung des Inverters $V_{in} = V_{out} =: V_{inv}$. Unter der Annahme, daß beide Transistoren im Sättigungsbereich arbeiten, muß gelten: $Z_{pd}(V_{inv} - V_{th})^2 = Z_{pu}(-V_{dep})^2$, und damit $V_{inv} = V_{th} - (V_{dep}\sqrt{Z_{pu}/Z_{pd}})$. Bei einem Verhältnis $Z_{pd}/Z_{pu} = 4$ ergibt sich genau $V_{inv} = 0.5\ V_{dd}$. Bei dieser Dimensionierung erfüllt der Inverter die oben genannte Bedingung $V_{out_L} < V_{th}$.

Bei einem fan-out von f beträgt die Verzögerung der fallenden Flanke $t_f \approx f\tau$ und die Verzögerung der steigenden Flanke $t_r \approx kf\tau$. Durch parasitäre Kapazitäten (z. B. auf den Signalleitungen) vergrößert sich die Verzögerungszeit, ebenso beim Treiben großer kapazitiver Lasten (z. B. Pad oder Array von Speicherzellen). Näherungsformeln: $t_r \approx \frac{4C_l}{BZ_{pu}V_{dd}}$ und $t_f \approx \frac{5C_l}{BZ_{pd}V_{dd}}$.

2.3.2 NAND

S. Bild 11. Die pull-down-Transistoren sind in Reihe geschaltet. Deshalb muß entweder ihre Breite oder die Länge des pull-up-Transistors vergrößert werden, um das Widerstandsverhältnis zu bewahren: $W_{pd_{nand}} = nW_{pd_{inv}}$ oder $L_{pu_{nand}} = nL_{pu_{inv}}$ mit n = Anzahl der Eingänge.

Im letzteren Fall wächst die Verzögerung eines n-Eingang-NAND auf $\tau_{nand} \approx n\tau_{inv}$.

2.3.3 NOR

S. Bild 11. Da die pull-down-Transistoren parallel geschaltet sind, muß ihre Dimensionierung gegenüber dem Inverter nicht verändert werden.

Die Verzögerung eines NOR beträgt $\tau_{nor} \approx \tau_{inv}$. Das NOR-Gatter ist also i. a. vorzuziehen.

2.4 CMOS-Schaltungen

Bei CMOS-Schaltungen werden p-Kanal und n-Kanal-Transistoren so verwendet, daß nur beim Schalten kurzzeitig ein größerer Strom fließt. Dies führt zu dem geringen Stromverbrauch der CMOS-Schaltungen. Das Netzwerk aus p-Kanal-Transistoren ist dafür verantwortlich, den Ausgang auf H zu ziehen; die n-Kanal-Transistoren sind für den L-Pegel zuständig. Sie müssen immer komplementäre logische Funktionen darstellen (Bild 12).

2.4.1 Inverter

Im stationären Zustand (logisch 0 oder 1) ist immer ein Transistor gesperrt und einer leitend. Der Sperrstrom eines FET ist typisch 0.1 - 1 nA, die entsprechende Verlustleistung von $P_S = 5$ nW bei $V_{dd} = 5$ V also vernachlässigbar klein. Im Übergangszustand (L $\rightarrow$ H und umgekehrt) gibt es dagegen einen spürbaren Leistungsverbrauch, da ein momentaner Strom fließt (beide FETs sind gleichzeitig leitend). Für einen vollständigen Zyklus wird dabei folgende Energie verbraucht:

V_{dd}	E_T
3.3 V	0.15 nWs
5 V	0.3 nWs
10 V	2 nWs

Die zugehörige Verlustleistung bei einem periodischen Rechtecksignal mit Frequenz f ist dann $P_T = fE_T$. Ausserdem wird beim Übergang von 0 nach 1 ein zusätzlicher Strom benötigt, um die externen Kapazitäten aufzuladen:

$E_C = CV^2$, $P_C = fE_C$. Gesamte Verlustleistung eines Inverters: $P_{inv} = P_S + P_T + P_C$.

Da der Ausgangspegel nicht vom Größenverhältnis der Transistoren abhängt, könnte man alle Transistoren gleich groß machen. Jedoch ist die Beweglichkeit der Ladungsträger im p-Kanal-Transistor (Löcher) um den Faktor 2 - 3 geringer als beim n-Kanal-Transistor (Elektronen). Der p-Kanal-Transistor muß deshalb eine entsprechend größere Breite W_P erhalten, um gleich schnelle ansteigende und abfallende Ausgangsflanken zu erzielen.

2.4.2 NAND

Hier sind die n-Kanal-Transistoren in Reihe geschaltet. Um gleich schnelle Schaltflanken zu erzielen, muß $W_{N_{nand}} = nW_{N_{inv}}$ sein (n = Anzahl der Eingänge), während $W_{P_{nand}} = W_{P_{inv}}$ bleibt.

2.4.3 NOR

Hier sind die p-Kanal-Transistoren in Reihe geschaltet. Deshalb muß $W_{P_{nor}} = nW_{P_{inv}}$ werden, während $W_{N_{nor}} = W_{N_{inv}}$ bleibt. Da die p-Kanal-Transistoren ohnehin schon groß sein müssen, ist hier das NAND-Gatter i. a. vorzuziehen.

Weitere Eigenschaften der CMOS-Technik im Vergleich zu NMOS (vgl. Bild 13):

- Die Übertragungscharakteristik der CMOS-Transistorlogik ist steiler als bei der NMOS-Logik.
- Die Eingangsimpedanz eines CMOS-Schaltkreises ist extrem hoch, so daß diese einen CMOS-Ausgang kaum belastet, der fan-out ist daher sehr groß (ungefähr 50).
- Der Platzbedarf ist wegen der größeren Anzahl an Transistoren, des Flächenbedarfs der p-Kanal-Transistoren und gewissen Minimalabständen 1.5- bis 2-mal größer als bei NMOS.
- Die Schaltzeit entspricht in etwa der NMOS-Technologie, sie ist bei 3 μm-CMOS-Technologie 2 ns. Derzeit wird eine 0.5 μm-CMOS-Technologie mit 150 ps Schaltzeit experimentell gefertigt.

2.5 Scaling

Die Verkleinerung der Maße eines Transistors zur Erreichung höherer Integrationsdichten nennt man scaling (Bild 14). Dabei wird gefordert, daß die

Kennlinien der Transistoren gleich bleiben sollen. Um dies zu erreichen, muß V_{dd} um den Verkleinerungsfaktor α reduziert ($V' = V/\alpha$) und die Dotierung um Faktor α erhöht werden. Bei einer Verkleinerung der Maße für L, W, D und $V = (V_{gs} - V_{th})$ um α gilt:

- Fläche $A' = A/\alpha^2$,
- $\tau' = \tau/\alpha$,
- $C' = C/\alpha$ (gut für Schalter, schlecht für Speicher),
- $I' = I/\alpha$,
- Leistungsverbrauch $P' = P/\alpha^2$
- power-delay-product $(P\tau)' = (P\tau)/\alpha^3$;

aber

- Stromdichte $(I/A)' = \alpha(I/A)$
- Kontaktwiderstand $R'_k = \alpha^2 R_k$
- Leitungswiderstand $R' = \alpha R$.

Aus Kompatibilitätsgründen wird V_{dd} auf 5 V gehalten. Dies wird durch zusätzliche Dotierung und weitere Verkürzung von W erreicht.

Die Verkleinerung kann nicht unbegrenzt fortgeführt werden. Grundlegende Einschränkungen sind gegeben durch $V_{dd_{min}} = 0.5$ - 0.1 V (wegen Schaltcharakteristik), $L = 0.25$ μm und $D = 5$ nm (sonst Tunneleffekt). Die Grenze der Auflösung bei Belichtung mit UV-Licht ist 1 μm, dann muß auf andere Belichtungsverfahren (Elektronenstrahl, Röntgenstrahlen) ausgewichen werden.

2.6 Bipolar-Technologie

Ein Bipolartransistor (npn-Typ) besteht aus zwei n-dotierten Schichten, die durch eine p-dotierte Schicht getrennt sind (Bild 15). Auf diese Weise entstehen zwei pn-Übergänge. Der eine Übergang (Emitter-Basis) wird so an eine Spannungsquelle angeschlossen, das der pn-Übergang in Durchlassrichtung gepolt ist und ein Strom fließen kann. Der andere pn-Übergang (Kollektor-Basis) wird in Sperrichtung angeschlossen, so daß nur ein kleiner Sperrstrom fließen kann. Die Elektronen, die vom Emitter in die Basis gelangen, werden jedoch von dem elektrischen Feld an dem gesperrten pn-Übergang angezogen und gelangen so in das n-Gebiet des Kollektors. Wieviele Elektronen

zum Kollektor fließen, hängt von der Spannung und von der Dicke des Basisbereichs ab.

Die Technologien unterscheiden sich darin, ob der Transistor im Sättigungsbereich oder im nicht gesättigten Bereich betrieben wird (Bild 16). Im nicht gesättigten Bereich können wesentlich kürzere Schaltzeiten erreicht werden. Bei TTL-Schaltungen kann durch Schottky-Dioden eine Sättigung vermieden werden (Schottky-TTL).

2.7 Herstellung von integrierten Schaltungen

Das Grundmaterial für integrierte Schaltungen sind Silizium-Einkristalle von bis zu 2 m Länge und 75 - 125 mm Durchmesser. Diese werden in Scheiben (wafer) von 0.2 - 0.3 mm Dicke geschnitten. Durch Photolithographie werden die einzelnen Strukturen von Masken auf den Wafer übertragen. Dazu wird ein photosensitiver Lack auf den Wafer aufgebracht und dieser belichtet. Dann kann entweder der belichtete oder der unbelichtete Teil entfernt werden. Danach werden die freiliegenden Teile je nach Layer mit Metall bedampft (für die Verdrahtung), oder entsprechende Ionen implantiert (für die Herstellung der Transistoren). Für die Isolierschichten wird das Silizium oxidiert, die Oxidschicht wird an den Kontaktstellen weggeätzt (vgl. Bild 17).

Fabrikationsschritte für PMOS-metal-gate-Prozeß (veraltet): Bild 18

Fabrikationsschritte für NMOS-silicon-gate-Prozeß (vgl. Bild 19):

1. Oxidation (field oxide, dick)
2. Maske 1: Lithographie für alle aktiven Flächen (Diffusionsbezirke)
3. selektives Abätzen des Oxids
4. Maske 2: Lithographie für depletion-mode-Transistoren
5. selektive Ionenimplantierung
6. Oxidation (gate oxide, dünn)
7. Beschichten mit Polysilizium
8. Maske 3: Lithographie für Gates und Poly-Leitungen
9. selektives Abätzen des Polysiliziums und des Oxids
10. Diffusion für Source und Drain
11. Oxidation

12. Maske 4: Lithographie für Kontakte
13. selektives Abätzen des Oxids
14. Beschichten mit Aluminium
15. Maske 5: Lithographie für Kontakte und Metall-Leitungen
16. selektives Abätzen des Metalls
17. Oxidation (Passivierung; dick)
18. Maske 6: Lithographie für Pads
19. selektives Abätzen des Oxids

Bei der Herstellung ist es nicht notwendig, die Source- und Drain-Zonen separat herzustellen. Vielmehr werden sie in den Masken als zusammenhängende Fläche dargestellt. Man macht sich dabei die Tatsache zunutze, daß die Diffusion das Poly-Gate und das Gate-Oxid nicht durchdringen kann, so daß unter dem Gate ein nicht dotierter Bereich entsteht. Die Dotierung des Polysiliziums ist sogar vorteilhaft, weil dadurch dessen Widerstand verringert wird. Außerdem ist sichergestellt, daß der Kanal vollständig unter dem Gate liegt, ohne daß kritische Maskenjustierungen nötig wären.

Die Herstellung von CMOS-Schaltungen erfordert mehr Prozeßschritte und Masken. S. Bild 20 u. 21 für einen NMOS-kompatiblen n-Wannen-CMOS-Prozeß.

Um eine sichere Herstellung der Chips zu gewährleisten, ist es notwendig, beim Entwurf gewisse Regeln (design rules) einzuhalten. Damit soll sichergestellt werden, daß die Fertigungstoleranzen nicht die Funktion der Transistoren beeinträchtigen. Die design rules legen Minimalwerte für bestimmte Abstände, Längen und Überlappungen fest. Um vom scaling unabhängig zu sein, werden die design rules in einer Längeneinheit λ angegeben, die der Auflösung der Belichtung des Herstellungsprozesses entspricht. Typische λ-Werte für kommerzielle Prozesse sind heute 1.5 - 2 μm. S. Bild 22 für NMOS design rules und Bild 23 für p-well CMOS design rules.

Auf einem Wafer werden viele gleichartige Schaltungen erzeugt, die jedoch nicht alle funktionsfähig sind. Gründe für Herstellungsfehler:

- Fehler in der monokristallinen Struktur des Halbleiters
- Staubpartikel auf dem Wafer
- Maskenfehler
- Justierung, Toleranzen

Die Fehler sind meist stark lokalisiert und statistisch über den Wafer verteilt. Die Defektdichte D beträgt typisch 2 bis 5 Defekte/cm^2. Als Ausbeute (yield) bezeichnet man das Verhältnis Y = Anz. funktionierender Chips/Anz. Chips auf dem Wafer.

Die Ausbeute hängt stark von der Chipgröße A ab. Abschätzungen: $Y \approx e^{-(AD)}$ oder $Y \approx Y_0^{(A/A_0)}$ (Y_0 ist die Ausbeute für einen Chip der Fläche A_0 bei dem jeweiligen Herstellungsprozeß) (Bild 24). Man versucht also die Chipfläche möglichst klein zu halten, um eine große Ausbeute zu erzielen.

Bei neuen Prozessen ist Y oft < 0.1. Nach Weiterentwicklung des Herstellungsprozesses steigt Y auf 0.3 - 0.5. Der Mehraufwand für noch größere Ausbeute lohnt sich kostenmäßig nicht, zumal dann schon die nächste Chipgeneration aktuell ist. Die Verbesserung der Ausbeute mit der Zeit entspricht der Lernkurve $Y \approx 1 - e^{-(t/\tau)}$ (τ konstant).

Ein interessanter Sonderfall ist die wafer scale integration (WSI). Die integrierte Schaltung erstreckt sich dabei über den gesamten Wafer. Die Schaltungen müssen fehlertolerant ausgelegt sein, um eine Funktion trotz Defekten zu erreichen.

3 Entwurfsprozeß für integrierte Schaltungen

3.1 Hierarchischer Entwurf

Der Entwurf von immer komplexeren integrierten Schaltungen zwingt den Entwerfer zu einer strukturierten Vorgehensweise. Der Entwurfsprozeß für integrierte Schaltungen ähnelt dem Entwurfsprozeß für große Software-Systeme (strukturierte Programmierung, top-down-Vorgehen, Wiederverwendbarkeit von Komponenten) (Bild 25). Die Verwendung von Hierarchien ist dabei von grundlegender Bedeutung. Beim Entwurf von integrierten Schaltungen unterscheidet man verschiedene Hierarchien.

3.1.1 Hierarchie von Abstraktionsebenen

Aufgrund der Komplexität einer integrierten Schaltung ist es notwendig, verschiedene Abstraktionsebenen für den Entwurf einzuführen, um eine geistige Bewältigung der Schaltungsfunktion zu ermöglichen (Bild 26).

System-Ebene: Verhaltensbeschreibung des Systems.

Register-Transfer-Ebene: Spezifikation des Datenflusses und Kontrollflusses im System.

Logik-Ebene / Gatter-Ebene: Netz von logischen Gattern, Registern und speichernden Elementen. Algorithmische Abläufe sind in der Logik-Ebene nicht mehr zu erkennen.

Schaltkreisebene / Transistorebene: Technologieabhängige Umsetzung der logischen Funktion (ECL-, NMOS-, CMOS-Realisierung von Gattern).

Layout-Ebene: Geometrische Anordnung und Realisierung der Komponenten wird festgelegt.

Es gibt spezielle Hardwarebeschreibungssprachen, z. B. VHDL (VHSIC Hardware Description Language). Sie dienen

- zur Beschreibung (Dokumentation, formale Beschreibung, Spezifikation)
- zur Verifikation und Simulation (Syntaxprüfung, Simulationsmöglichkeiten)
- als Ausgangspunkt für automatische Entwurfsverfahren

auf der funktionalen Entwurfsebene oder für den Gesamtentwurf (multi-level).

Entwurfsschritt	Darstellungsebene	Entwurfsmodell	Beschreibung / Verifikation
Systementwurf	Blockdarstellung	Verhaltens-beschreibung	ISPS
RT-Entwurf	funktionale Blöcke	REG, ALU, ...	KARL III
Logikentwurf	logische Gatter	AND, OR, ...	DISIM
Schaltungsentwurf	Schaltplan	Transistoren	SPICE
Layout	Stick-Diagramm	Zellen, Polygone	CIF

Auf jeder Abstraktionsebene wird der Entwurf beschrieben und verifiziert. Meist erfolgt die Verifikation durch Simulation, da eine formale Verifikation zu komplex ist. Die Verifikation dient zur Sicherstellung der horizontalen (in der selben Abstraktionsebene) und vertikalen Konsistenz (über verschiedene Abstraktionsebenen hinweg).

Vorgabe für die Konstruktion der Ebene n:

- Ergebnis der Ebene $n-1$ (strukturierter Datensatz in einer Datenbank)
- Entwurfsregeln für die Ebene n

Eine Entwurfsregelüberprüfung kann erfolgen, wenn formale Entwurfsregeln existieren, sonst erfolgt eine horizontale Verifikation durch Simulation. Dann ist zu prüfen, ob die Sollvorgabe der Ebene $n-1$ erfüllt ist, d.h. ob vertikale Konsistenz vorliegt. Dies dient zur Sicherung der Entwurfstreue. Mit einem Extraktor wird eine Beschreibung der Ebene $n-1$ aus Ebene n zurückgewonnen. Danach kann ein Soll-Ist-Vergleich oder eine Re-Simulation durchgeführt werden (Bild 27).

Beim Entwurf wird nun top-down eine funktionelle Beschreibung in eine geometrische Beschreibung umgesetzt, indem eine Umsetzung von einer Abstraktionsebene in die nächste vorgenommen wird. Dabei steigt die Komplexität stark an.

Beispiel:

RT-Beschreibung	1 000 RT-Elemente
Gatter	30 000 Gatter
Transistoren	100 000 Transistoren
Layout	1 Mio. Rechtecke

Der allgemeine IC-Designprozeß (full custom) und der Standard-Designprozeß (Gate-Array, Standardzellen) unterscheiden sich im Grad der Verwendung von vorgefertigten Modellen und im Grad der Automatisierbarkeit (Bild 28, 29).

Der Übergang von einer höheren in eine tiefere Entwurfsebene führt zu einer detaillierteren Darstellung der integrierten Schaltung. Dabei spielt mit der immer detaillierteren Darstellung die verwendete Technologie eine immer größere Rolle. Ziel ist es nun, die höheren Entwurfsebenen unabhängig von der verwendeten Technologie zu halten, um bei der schnellen Weiterentwicklung der Technologie nicht die komplette Schaltung auf allen Entwurfsebenen neu entwickeln zu müssen. Dies ist bei den Vollkundenschaltungen meist nicht möglich, da dort unter Verwendung der speziellen Möglichkeiten einer Technologie versucht wird, eine möglichst platz- und geschwindigkeitsoptimale Lösung zu finden. Bei einer Semikundenschaltung ist eine höhere Technologieunabhängigkeit vorhanden. Die verwendeten Grundzellen werden dort einfach durch neue Zellen ersetzt, der Entwurf muß auf höherer Ebene nicht geändert werden.

3.1.2 Syntaktische Hierarchie

Die VLSI-Schaltung wird dabei als eine zusammengesetzte Zelle betrachtet, deren Teilzellen Grundzellen sind oder wieder aus anderen zusammengesetzten Zellen aufgebaut sind (Bild 30). Bei Verwendung einer syntaktischen Hierarchie kann die Datenmenge reduziert werden, wenn eine Gruppe von Elementen wiederholt im Entwurf vorkommt.

3.1.3 Semantische Hierarchie

Das Konzept der semantischen Hierarchie führt über eine syntaktische Hierarchie hinaus. Semantische Hierarchien dienen dazu, eine Gruppe von Daten einer bestimmten syntaktischen Ebene in einer einfacheren Form zu beschreiben. Diese abstrakte Form bedeutet, daß die Information der Zelle reduziert wird, aber für die Analyse auf der höheren Ebene ausreicht. Die abstrakte Beschreibung einer Zelle besteht aus einer einfacheren Beschreibung, die anstelle der Zelle selbst in die hierarchische Struktur eingesetzt wird.

Beispiel design rule check auf der geometrischen Layout-Ebene:
Das Layout einer integrierten Schaltung muß auf die Einhaltung geometrischer Regeln geprüft werden, die für den Fertigungsprozeß vorgeschrieben sind (vgl. Kap. 2). Wird nun eine Zelle geprüft, so wird unabhängig von der Plazierung die innere Layoutstruktur geprüft. Entspricht diese den Entwurfsregeln, so braucht bei Verwendung dieser Zelle als Teil einer anderen Zelle beim DRC der zusammengesetzten Zelle die Layoutbeschreibung der untergeordneten Zelle nicht mehr geprüft werden; diese kann durch das Ergebnis der Prüfung ersetzt werden.

3.2 Die Entwurfsphasen

Beim hierarchischen Entwurf ist von besonderer Bedeutung, daß sich bei einer stufenweisen Verfeinerung des Entwurfs (top-down) die hierarchischen Strukturen mit den verschiedenen Betrachtungsweisen (funktional, Logik, Layout) vervielfachen. Das zu entwerfende System wird zuerst hierarchisch zerlegt. Dann werden die Endknoten der Hierarchie in verschiedenen Abstraktionsebenen beschrieben. Von großer Bedeutung ist dabei die Entwurfsverifizierung in den einzelnen Abstraktionsebenen. Diese wird meist durch Simulation oder spezielle Testprogramme durchgeführt. Sie wird durch Verwendung von semantischen Hierarchien vereinfacht.

Der Entwurf wird meist nicht im reinen top-down-Verfahren durchgeführt. Das Ergebnis einer Simulation kann zeigen, daß die Vorgaben (z. B. des Zeitverhaltens) nicht eingehalten wurden, so daß die Struktur des Entwurfs wieder verändert werden muß. Bei Entwurf von kritischen Teilen oder zur Ausnützung spezieller Technologiemöglichkeiten wird oft bottom-up vorgegangen (vom Layout für einzelne Schaltungsteile ausgehend) und der Entwurf wird dann auf diesen Teilentwürfen aufgebaut.

4 Systementwurf

Auf dieser Ebene wird eine rein funktionale oder verhaltensmäßige Beschreibung des Systems gegeben, auf das Zeitverhalten wird meist noch nicht eingegangen. Es wird dann durch Simulation überprüft, ob die gestellten Anforderungen an das System erfüllt werden. Gegebenenfalls wird der Entwurf geändert.

Der funktionale Entwurf kann in einer universellen Programmiersprache (ADA, C, Occam usw.) erfolgen. Für diesen Zweck gibt es aber auch spezielle Systementwurfssprachen und Simulatoren. Sie erlauben die Beschreibung und Simulation komplexer Systeme. Bei diesen Systembeschreibungssprachen kann eine Aufteilung des Systems in unabhängige Module vorgenommen werden. Jede Einheit besteht aus einem strukturellen Teil und einem Kontrollteil für die Beschreibung des sequentiellen Ablaufs. Die Module können synchron und asynchron verbunden werden.

Dies führt zur Aufteilung des Entwurfs in Kontrollfluß und Datenfluß und bildet eine erste Strukturierung des Entwurfs. Die Realisierung kann z. B. durch Verwendung einer FSM (Finite State Machine, endlicher Automat) erfolgen (Bild 31).

Beispiel ISPS (Instruction-Set Processor Specification):
Mit ISPS kann das Verhalten einer CPU vollständig definiert werden, ohne daß damit jedoch Aussagen über interne Struktur oder timing verbunden sind (Bild 32).

5 Register-Transfer-Beschreibung

Die Notation von Register-Transfer-Sprachen zur Beschreibung und Simulation auf Register-Transfer-Ebene benutzt Register, Speicher, Busse, Multiplexer und funktionale Einheiten (Addierer usw.) als Sprachelemente. Damit wird die Struktur des zu entwerfenden Schaltkreises bereits teilweise festgelegt.

Beispiel: KARL (Kaiserslautern Register Transfer Language)
KARL-III ist eine nicht-prozedurale Register-Transfer-Sprache mit Sprachelementen für RT-Ebene, Logikebene und Schaltkreisebene (Bild 33).

Eine Simulation auf RT-Ebene benötigt nur wenig detaillierte Informationen. Diese Simulation ist sehr nützlich, wenn die Architektur eines digitalen Systems entworfen werden soll, da in dieser Phase der Datenfluß im System von hauptsächlichem Interesse ist. Der Kontrollfluß muss durch expliziten Entwurf eines Steuerwerks (mikroprogrammiert oder FSM) beschrieben werden, was allerdings weniger übersichtlich ist als bei den Systementwurfssprachen (Bild 34, 35).

Oft liegen spezielle Anforderungen an den Baustein vor (Fehlertoleranz, Geschwindigkeit usw.). Ein erster Ansatz zur Erfüllung der Spezifikation wird meist mit Register-Transfer-Blöcken wie ROMs, RAMs, Register, Multiplexer, Bussen durchgeführt.

6 Logik-Entwurf

6.1 Logikkonstruktion

Der Systementwurf wird in die Ebene der Logikelemente umgesetzt. Logikelemente sind Gatter (AND, NAND, NOR, ...), Flip-Flops, Multiplexer usw. Das Resultat dieses Entwurfsschrittes ist die Verknüpfung dieser Logikelemente (Bild 36, 37). Die Darstellung auf der Logikebene erfolgt als Logikplan oder Netzliste.

Zur Optimierung des Logikentwurfs werden übliche Verfahren angewandt wie

- Zustandsminimierung bei Verwendung von FSM
- Logikminimierung
  - Quine-McCluskey-Verfahren: Generierung aller Prim-Implikanten, Extraktion einer minimalen Überdeckung
  - ESPRESSO-Algorithmus zur PLA-Minimierung (heuristisch)

Eine wichtige Aufgabe ist die Bestimmung des Zeitverhaltens der Schaltungen. Die funktionale Beschreibung gibt meist keine genauen Hinweise auf das zeitliche Verhalten der Schaltung. Für die VLSI-gerechte Implementierung von logischen Funktionen spielen folgende Punkte eine wichtige Rolle:

- Abhängigkeit der Verzögerungszeiten vom fan-in und fan-out der einzelnen Gatter
- Verzögerungszeiten durch die Verdrahtung
- Gatteranzahl und die damit verbundene Chipfläche und Verlustleistung
- Verwendung möglichst regulärer Strukturen, um Plazierung und Verdrahtung der Gatter im Layout zu vereinfachen (Verwendung der disjunktiven Normalform für kombinatorische Logik zur PLA-Realisierung).

Sequentielle Netze können auf zwei Arten entworfen werden:

Asynchroner Entwurf: Das zeitliche Verhalten ist sehr schwierig abzuleiten, deshalb ist diese Methode nicht zu empfehlen.

Synchroner Entwurf: Alle Zustandsänderungen werden von einem Takt abgeleitet. Das Taktintervall zwischen den Zustandsänderungen wird bestimmt durch die Verzögerungszeiten der Gatter und Leitungen. Es muß berücksichtigt werden, das die Verzögerungszeiten in einem gewissen Bereich schwanken können (bedingt durch Toleranzen des Herstellungsprozesses), deshalb Verwendung von Maximalzeiten (worst case design). Schleifen zwischen Ein- und Ausgängen der kombinatorischen Logik müssen über getaktete Speicherelemente laufen, da sonst kein eindeutig bestimmtes Verhalten erreicht werden kann. Externe asynchrone Signale werden meist synchronisiert.

In sequentiellen Schaltnetzen müssen auch die setup- und hold-Zeiten der speichernden Elemente berücksichtigt werden.

6.2 Logikverifikation

Die Logikverifikation dient zur Feststellung der vertikalen Konsistenz (Vergleich mit der Vorgabe auf funktionaler Ebene) und der horizontalen Konsistenz (Prüfung ob logische Entwurfsregeln (z. B. keine Rückkopplung in kombinatorischer Logik) eingehalten wurden).

Die Verifikation erfolgt durch Logik- und Timing-Simulation. Logiksimulatoren simulieren das logische Verhalten und das Zeitverhalten digitaler Schaltungen. Es gibt viele derartige Programme mit unterschiedlichen Eigenschaften:

- Zeitverhalten der Gatter in der Simulation
  - keine Verzögerung: einfach, aber unrealistisch
  - konstante Verzögerung (unit delay)
  - variable Verzögerung: aufwendig
- strukturelle oder Verhaltenssimulation
  - Bei Verhaltenssimulation wird ein Baustein als 'black box' mit idealisiertem Verhalten dargestellt.
  - Bei struktureller Simulation wird auch die interne (reale) Struktur in der Simulation nachgebildet.

  Kombination: mixed mode simulation
- PME (Physical Model Extension): Teile der simulierten Schaltung werden durch Hardware realisiert, die mit dem Simulationsrechner gekoppelt ist.

Eine Logiksimulation von Schaltungen mit mehreren tausend Gattern ist sehr zeitaufwendig. Eine Beschleunigung der Logiksimulation ist durch spezielle Simulationsrechner (bit-slice-Rechner) möglich. Diese sind meist wesentlich schneller als Simulationssoftware auf einem Großrechner.

Beispiel: Zycad Logic Evaluator
1 Mio. Ereignisse (Signalwechsel)/sec.
1 Mio. Gatter max.

8 000 Gatter, 250 000 Zeitschritte auf	Simulationszeit
VAX 11/780 (Supermini)	1.66 h
Cyber 205 (Super)	10.5 min
Logic Evaluator	1.92 sec

Die einfachsten Logiksimulatoren verwenden nur die Logikzustände 0 und 1. Meist werden auch noch 'tri-state' (hochohmig), 'Übergang $0 \rightarrow 1$', 'Übergang $1 \rightarrow 0$' und 'undefiniert' berücksichtigt.

6.3 Testbarkeitsanalyse

Um die integrierte Schaltung auf einem Prüfautomaten auf Funktion prüfen zu können, ist ein Satz von Testmustern (testpatterns) nötig. Die Qualität der Testmuster ist bestimmt durch

- Länge (möglichst kurz, damit schnelle Prüfung)
- Fehlerabdeckung (Zahl der Fehler, die durch diese Testmuster erkannt werden, bezogen auf die Zahl der möglichen Fehler)

Es können allerdings nur statische Fehler berücksichtigt werden (vgl. Bild 38):

- Totalausfälle
- Haftfehler (stuck-at-faults) auf 0 oder 1 an den Ausgängen
- Haftfehler auf 0 oder 1 an den Eingängen
- Kurzschlüsse zwischen benachbarten Anschlüssen
- Leitungsunterbrechungen und Leitungskurzschlüsse

Die Erzeugung von Testmustern erfolgt mit Hilfe von Fehlersimulatoren. Hauptaufgabe der Fehlersimulation ist die Bestimmung der Fehlerüberdeckung (Anteil der auffindbaren Fehler an der Gesamtzahl der statischen Fehler). Dazu werden in das aus der Logiksimulation bekannte Modell Fehler

eingebaut und festgestellt, ob sich der Fehler durch entsprechende Eingangsbelegung am Ausgang der Schaltung bemerkbar macht. Deduktive Verfahren, d. h. die Bestimmung der Belegung, die einen Fehler am Ausgang sichtbar macht, sind sehr aufwendig; die Simulation ist deshalb speicher- und rechenzeitintensiv. Eine Abschätzung für den Rechenzeitbedarf für die Testmustererzeugung ist $R \sim aN^b$ mit N = Anzahl der Gatteräquivalente; a = 0.5 für kombinatorische Schaltungen, a = 1 für sequentielle Schaltungen; b = 2 - 3 für nicht testfreundlichen Entwurf, $b \leq 1.5$ für testfreundlichen Entwurf (s. u.).

6.4 Testfreundlicher Entwurf (design for testability)

Durch die zunehmende Integrationsdichte wird es immer schwieriger, alle Punkte in einer integrierten Schaltung zu testen. Deswegen müssen schon beim funktionalen, hauptsächlich aber beim Logikentwurf entsprechende Vorkehrungen getroffen werden, um die Testbarkeit der integrierten Schaltung sicherzustellen (Bild 39, 40).

Partitionierung: Durch Aufteilung der Schaltung in einzelne Blöcke, die getrennt von außen zu testen sind, läßt sich der Testaufwand senken.

Testpunkte: Durch Multiplexer können verschiedene interne Punkte an die Anschlüsse der integrierten Schaltung gelegt werden.

scan path design: Die Schaltung wird aufgeteilt in einen kombinatorischen Teil und einen sequentiellen Teil. Alle speichernden Elemente der Schaltung werden zu einem Schieberegister zusammengeschaltet. Zu Beginn eines Tests kann sequentiell ein Testmuster eingelesen werden, nach einem oder mehreren Takten kann das Resultat ausgelesen werden. Der Test reduziert sich damit auf den Test des kombinatorischen Netzwerks (Bild 41). Ziel: jeder Punkt in der Schaltung ist erreichbar (testbar).

Integrierte Testmittel:

Signaturanalyse: Aus linear rückgekoppelten Schieberegistern (LFSR) wird ein Signaturregister gebildet (Bild 41). Daten von bestimmten Testpunkten werden in diese Register geschrieben und dann in einem Schiebemodus über EXOR-Gatter mit neuen Eingabemustern verknüpft (vgl. CRC bei Datenübertragung). Das Ergebnis ist ein Pseudozufalls-Muster, das charakteristisch für die verschiedenen Eingabemuster ist und mit einem gespeicherten Testmuster verglichen werden kann. Die Wahrscheinlichkeit, einen Fehler zu erkennen, beträgt

$P = 1 - \frac{2^{n-b}-1}{2^n-1}$ mit b = Breite des Signaturregisters, n = Länge des Bitstroms.

Selbsttest: Ein on-chip controller steuert den Testablauf mit Scan-Test, Zufallsgeneratoren für Testmustererzeugung und Signaturregistern (= BILBO (Built In Logic Block Observer)) und wertet die Ergebnisse aus (Bild 42). Testbare Entwürfe können unter Verwendung von standardisierten testbaren Bausteinen (z. B. Register mit scan-path) einfach zusammengesetzt werden.

Der Platzbedarf bei design for testability steigt im Durchschnitt um 10 - 20 %.

7 Schaltkreisentwurf

Ziel des Schaltkreisentwurfs ist es, die einzelnen logischen Elemente durch Transistoren zu realisieren (Bild 43). Die Schaltkreisebene kann je nach Sichtweise aufgeteilt werden in switch level (Transistoren werden als Schalter dargestellt) und circuit level (Transistoren werden mit ihren realen Parametern berücksichtigt). Die Schaltkreisebene bezieht sich immer auf eine bestimmte Zieltechnologie, ist also nicht mehr technologieunabhängig. Die gängigsten Technologien für kundenspezifische Schaltkreise sind CMOS und NMOS. Bei speziellen Geschwindigkeitsanforderungen wird ECL-Technik verwendet.

Der Schaltkreisentwurf wird im Normalfall nur bei Vollkundenentwurf und eventuell noch bei Gate-Arrays durchgeführt. Bei Standardentwurfsverfahren (Standardzellen, Gate-Arrays) wird nach dem Logikentwurf durch Auswahl vorgefertigter Zellen aus einer Bibliothek direkt auf den Layout-Entwurf übergegangen.

Bei Vollkundenentwürfen werden viele Optimierungen durch Entwurf von speziellen Schaltungen für bestimmte Logikteile unter Ausnützung aller Möglichkeiten einer bestimmten Technologie erreicht. Bei Standardentwurfsverfahren werden die üblichen Realisierungen der einzelnen Logikgatter verwendet, die ohne Optimierungsmöglichkeiten zusammengeschaltet werden. Es gibt nur die Möglichkeit der Auswahl zwischen verschiedenen Varianten einer Logikschaltung in der Bibliothek.

7.1 Schaltungsentwurf in NMOS

7.1.1 NMOS Static Logic

Statische Logik auf der Basis von enhancement- und depletion-mode-Transistoren. Sie wird auch als ratioed logic bezeichnet, da die Logikpegel durch das Größenverhältnis der Transistoren bestimmt werden. S. Bild 44 und Beispiele in Kap. 2.

7.1.2 Enhancement FET Logic

Statt eines depletion-mode-Transistors, dessen Gate und Source verbunden sind, kann auch ein enhancement-mode-Transistor, dessen Gate und Drain verbunden sind ($V_{ds} = V_{gs}$), als Lasttransistor eingesetzt werden (Bild 44). Die Funktion des Inverters ist wie folgt: Bei H am Eingang sind beide Transistoren durchgeschaltet, die Ausgangsspannung ist bestimmt durch die beiden Transistoren. Deren Größenverhältnis muß so bestimmt werden, daß das Widerstandsverhältnis im durchgeschalteten Zustand eine

Ausgangsspannung unter V_{th} gewährleistet. Wenn der Eingang auf L liegt, geht der Ausgang auf $V_{out} \leq V_{dd} - V_{th}$, da der Lasttransistor sperrt, wenn $V_{gs} = V_{dd} - V_{out} < V_{th}$ wird.

Diese Technologie wurde eingesetzt, bevor depletion-mode-Transistoren hergestellt werden konnten. Um dem geringen Signalhub entgegenzuwirken, wurde meist $V_{dd} = 12$ V benutzt.

7.1.3 PLA

Komplexe Logik-Funktionen, z. B. die Ansteuerfunktionen der Flip-Flops in FSMs, werden sinnvollerweise nicht aus Logikgattern zusammengesetzt, sondern in ihre distributive Normalform überführt und als PLA (Programmable Logic Array) realisiert. Die AND-OR-Struktur wird als NOR-NOR-Struktur realisiert (Bild 45). Das PLA ist eine regelmäßige Struktur und deshalb einfach zu entwerfen, allerdings ist es nicht flächenoptimal.

7.1.4 Pass Transistors

Pass transistors werden als elektronische Schalter verwendet. Damit können z. B. Ladungen kurzzeitig isoliert oder komplexe Logikfunktionen realisiert werden (Bild 46). Dabei ist allerdings zu beachten, daß jeder pass transistor das Signal verschlechtert, d. h. es dürfen nicht allzuviele hintereinander geschaltet werden.

7.1.5 Takte in NMOS-Schaltungen

Der zeitliche Ablauf in MOS-Schaltungen wird meist nicht flankengesteuert, sondern durch Pass-Transistoren gesteuert. Daraus ergeben sich dann folgende Möglichkeiten der Taktversorgung (Bild 47):

Ein-Phasen Takt: Einfach, aber riskant. Die Signale durchlaufen die kombinatorische Logik mit unterschiedlicher Geschwindigkeit. Das langsamste Signal muß schneller sein als T_p. Das schnellste Signal muß langsamer sein als T_{high}, da sonst Teile der kombinatorischen Logik schon in den übernächsten Zustand schalten. Diese Bedingungen sind nicht einfach zu erfüllen, zumal sie auch unter dem Einfluß von Fertigungstoleranzen, Temperatur usw. nicht verletzt werden dürfen. Deshalb wird dieses Verfahren heute nicht mehr angewandt.

Zwei-Phasen Takt: Etwas komplizierter, aber sicher. Der nichtüberlappende 2-Phasen-Takt kann aus einem normalen Takt mit geeignetem Tastverhältnis gewonnen werden.

7.1.6 Dynamic Logic

Durch dynamische Logik kann eine Schaltungsvereinfachung und eine Verringerung des Stromverbrauchs erreicht werden (Bild 48). Es wird ein nichtüberlappender 2-Phasen-Takt benötigt.

$\phi 1$ lädt den Ausgang auf $V_{dd} - V_{th}$ auf, $\phi 2$ entlädt bedingt durch die logische Funktion den Ausgang dann wieder auf 0. Hierbei kommt es bei der Bestimmung der Ausgangspegel nicht auf die Größenverhältnisse der Transistoren an (ratioless logic). Problematisch ist, daß solche Gatter nicht einfach miteinander verbunden werden können, da sich die Eingänge während $\phi 2$ nicht ändern dürfen. Um dies zu gewährleisten, können abwechselnde Logikstufen mit nichtüberlappenden Takten versorgt werden (4-Phasen-Takt).

Da über die Taktsignale viele Kapazitäten geladen werden, werden hohe Anforderungen an die Takttreiber gestellt. Außerdem muß eine minimale Taktfrequenz berücksichtigt werden, da sich die Ladungen sonst verflüchtigen (typ. 10 kHz). Schließlich wird das Layout durch die vielen Taktleitungen erschwert.

7.1.7 Schutz der Eingänge

Das Gate eines MOS-Transistors bildet mit dem Substrat einen Kondensator mit einer sehr dünnen Oxidschicht (25 - 100 nm) als Dielektrikum. Bei einer Spannung von mehr als ca. 50 V wird diese Isolierschicht durchschlagen und der Transistor zerstört. Aus diesem Grund müssen MOS-Schaltungen vor statischer Elektrizität geschützt werden. An den Eingangspads werden zusätzlich besondere Schaltungen angebracht, die Überspannungen ableiten sollen, z. B. Schutzdioden oder punch-through-Strecken, die auf der Ausdehnung des n-Gebietes bei hoher Spannung beruhen (Bild 49).

7.1.8 Ausgangstreiber

Ausgänge, die hohe Ströme liefern sollen, benötigen große Transistoren. Dadurch wird jedoch die Eingangskapazität dieser Treiber sehr hoch, so daß die Schaltzeit durch die Belastung der Vorgängerstufe ansteigt. Deshalb werden Ausgangstreiber durch Hintereinanderschaltung von mehreren Stufen realisiert. Die optimale Anzahl und Leistung der Stufen läßt sich analytisch bestimmen. Typisch sind 3 Stufen, wobei jede die dreifache Leistung der Vorgängerstufe hat (Bild 49).

7.2 CMOS-Schaltungstechnik

7.2.1 Statische CMOS-Logikschaltungen

Bild 50, siehe auch Beispiele in Kap. 2.

7.2.2 Transmission Gate

Ein transmission gate besteht aus einem p-Kanal- und einem n-Kanal-Transistor, die parallelgeschaltet sind und gegensinnig angesteuert werden. Es bildet das CMOS-Äquivalent zum pass transistor, hat jedoch bessere elektrische Eigenschaften (Bild 51). Mit CMOS-Invertern und CMOS-transmission-gates kann man Flip-Flops, Schieberegister, Multiplexer usw. realisieren (Bild 52).

7.2.3 Domino Logic

Domino-Logik ist eine Erweiterung der NMOS dynamic logic. Der Ausgang eines dynamischen Gatters wird immer mit einem statischen Inverter komplementiert (Bild 53). Während $\phi = 0$ wird der Ausgangsknoten auf $V_{dd} - V_{th}$ geladen, damit wird der Ausgang des Inverters L. Die nachfolgenden Stufen bleiben somit inaktiv. Während $\phi = 1$ wird der Knoten bedingt entladen, wodurch der Inverterausgang auf H geht. Dadurch wird die nachfolgende Stufe aktiviert usw. wie beim Umfallen einer Reihe von Dominosteinen.

Vorteilhaft bei dieser Schaltung ist die geringere Transistorzahl bei komplexen Logikfunktionen ($n + 4$ statt $2n$ bei n Eingängen), zumal die großflächigen p-Kanal-Transistoren entfallen. Nachteilig ist jedoch, daß keine invertierenden Funktionen realisiert werden können, außer in der letzten Stufe vor dem Übergang in statische Schaltungen oder Register.

7.2.4 Latch-up-Effekt

Ein Problem, das nur bei CMOS-Schaltungen aufteten kann, ist die Bildung eines parasitären Thyristors (pnpn-Anordnung) im Chip. Wird dieser Thyristor durch eine Störspannungsspitze gezündet, kann er leitend bleiben und eine direkte Verbindung zwischen V_{dd} und V_{ss} bilden (Bild 54). Dadurch wird die Funktion der Schaltung gestört. Um dies zu verhindern, müssen beim Schaltungsentwurf und Layout besondere Maßnahmen ergriffen werden, z. B. räumliche Trennung (unerwünscht), Substratkontakte oder trench isolation.

7.3 Verifikation auf Schaltkreisebene

Mit einem ERC-Programm (Electrical Rules Check) kann die Schaltung auf grobe Fehler (Kurzschlüsse, nicht angeschlossene Schaltungsteile, gegeneinander geschaltete Ausgänge usw.) geprüft werden. Die vertikale Konsistenz wird durch Extraktion der Logikebene aus den Transistornetzen oder durch Simulation überprüft.

Schaltkreissimulatoren sind Simulationsprogramme für elektrische Netzwerke aus Transistoren, Kondensatoren und Widerständen. Für die einzelnen Bausteine werden entsprechende mathematische Modelle eingesetzt. Die Spannungen und Ströme in den einzelnen Punkten der Schaltung werden in Abhängigkeit von der Zeit dargestellt. Damit läßt sich das Timing oder die Einhaltung der Logikpegel genau überprüfen.

Die Simulatoren unterscheiden sich hauptsächlich durch die Komplexität der verwendeten Modelle und der Berechnungsverfahren. Beispiele:

SPICE: Iterative Methode für die Berechnung der nichtlinearen Gleichungen. Es werden diskrete Spannungswerte ermittelt.

RELAX: waveform relaxation-Methode: Lösung der Differentialgleichung für die einzelnen Elemente. Es wird der Spannungsverlauf (die Kurvenform) approximiert (Bild 55).

8 Layoutentwurf

Beim Layoutentwurf wird die geometrische Struktur und Anordnung der Transistoren auf einem Chip festgelegt. Bei einem full-custom-Entwurf ist dies ein großer Teil des Entwurfsaufwandes.

Die Layouterstellung geschieht in mehreren hierarchischen Schritten.

Partitioning: Aufteilung einer Schaltung auf mehrere Module oder Chips

Floorplanning: Verteilung der Module auf einem Chip

Placement: Plazierung der einzelnen Gatter oder Zellen in einem Modul

Layout: Entwurf der einzelnen Gatter

- symbolisches Layout (stick diagram)
- geometrisches Layout

Routing: Verdrahtung der einzelnen Gatter, Zellen und Module (hierarchisch)

Die Komplexität eines Layouts kann durch die mehrfache Verwendung vorentworfener Zellen und Module (Makrozellen) wesentlich verringert werden. Diese Reduzierung der Komplexität wird durch die Regularität eines Entwurfs ausgedrückt: Regularität = gesamte Chipfläche/individuell entworfene Chipfläche. Bei modernen Layouts ist der Regularitätsfaktor > 20.

8.1 Partitionierung

Das System soll so in eine vorgegebene Anzahl von Teilsystemen zerlegt werden, daß die Anzahl der Verbindungen zwischen den Modulen minimiert wird. Die Module mit den meisten gegenseitigen Verbindungen müssen im selben Teilsystem liegen. Man geht von wenigen Elementen aus und führt dann einen Integrationsprozeß durch.

Algorithmus (agglomerative Clusterbildung) (Bild 56):
Gegeben ist die Verbindungsmatrix (diagonalsymmetrisch).

1. Suche das größte Matrixelement; falls es mehrere gibt, wähle ein beliebiges.
2. Lies die Koordinaten i und j des Maximalelementes ab.
3. Addiere Spalte i zu Spalte j und Zeile i zu Zeile j.
4. Streiche Zeile i und Spalte i.

5. Notiere i unter Spalte j.

6. Prüfe, ob die Anzahl der notierten addierten Elemente gleich der maximalen / gewünschten Zahl der Elemente pro Modul ist.
   Falls ja: fertiges Modul notieren.
   Falls nein: $\rightarrow$ 1.

Weitere Zerlegung des Rests: mit dem Rest von vorne beginnen.

Die Clusteranalyse macht keine Aussage über die Anzahl der Außenkontakte. Abschätzung (Rent's Rule): $X_m \approx X_e n^r$, wobei X_m = mittlere Anzahl der Außenkontakte eines Moduls, X_e = mittlere Anzahl der Kontakte eines Elements, n = Anzahl der Elemente im Modul und r = Konstante (früher 0.6 - 0.7, heute 0.3 - 0.5 durch bessere Partitionierung).

Eine weitere Überlegung zeigt, daß mit dem Ansteigen der Anzahl von Elementen in einem Modul die Anzahl der Außenkontakte wächst, dann aber nach Erreichen eines Maximums auf einen konstanten Endwert fällt, der gerade der Anzahl der Außenkontakte des Gesamtsystems entspricht.

8.2 Floorplanning

Beim floorplanning wird die Anordnung der Module, der Datenpfade zwischen diesen Modulen und die Verlegung der Stromversorgung festgelegt (Bild 57). Die Ergebnisse der Clusteranalyse können dabei angewendet werden.

Wichtigstes Ziel ist hierbei die Minimierung der Verbindungslängen und der Chipfläche. Dabei müssen die parasitären Kapazitäten und Widerstände der Verdrahtungen berücksichtigt werden, um einen störungsfreien Betrieb zu gewährleisten. Bei der Spannungsversorgung der Module muß der Widerstand und die Stromdichte in der Verbindung berücksichtigt werden. Bei immer höher integrierten Schaltkreisen mit immer mehr Modulen treten Probleme auf:

- Beschränkung der Geschwindigkeit des Schaltkreises durch die Verdrahtung: Beim Verkleinern der Verbindungsleitungen steigt deren Widerstand an. Die durchschnittliche Verdrahtungslänge bleibt aber ungefähr gleich, da immer noch viele Verbindungen über den ganzen Chip gehen (Bild 58).

- Gesamtlänge der Verdrahtung steigt mit der Komplexität des Chips an.

Ab einer gewissen Integrationsdichte dominiert dann die Chipfläche, die durch Verdrahtung belegt ist, über die Chipfläche, die durch Gatter belegt ist. Durch Verwendung mehrerer Metall-Ebenen im Chip ist es möglich,

die Verdrahtung auf mehreren Ebenen auszuführen. Dies erhöht jedoch die Fertigungskosten erheblich. Deswegen ist es sehr wichtig, die Verdrahtung sorgfältig zu planen und durch günstige Plazierung und Modifikation der Logik die Verdrahtung zu vereinfachen (Bild 59).

8.3 Placement

Beim placement (Plazierung) wird die Anordnung der Gatter und Makros (Multiplexer, Register, Addierer, Zähler usw.) in einem Modul festgelegt.

Grundlage ist wieder die Anzahl und Länge der Verbindungen zwischen den einzelnen Bausteinen. Die Plazierungskomplexität hängt von der Freiheit der Plazierungsmöglichkeiten ab (Bild 60).

- Gate-Array: feste Plätze für die Gatter vorgegeben
- Standardzellen: lineare Plazierung
- Full-Custom: fast beliebige Plazierungsmöglichkeiten

Matrix-Plazierung: Plazierung auf matrixartiger Grundstruktur

Linear-Plazierung: reihenförmige Anordnung

Manhattan-Plazierung: beliebige Plazierung, zwischen allen Zellen Verdrahtungskanäle

8.3.1 Lineare Plazierung

Das Ziel der linearen Plazierung ist eine derartige lineare Anordnung der Bausteine / Zellen, daß die Gesamtlänge der Verdrahtung minimiert wird und möglichst viele Verbindungen durch einfaches Aneinanderreihen entstehen (wiring by abutment). Der folgende Algorithmus ist hierbei nützlich.

Algorithmus *MINCUT* für Teilung eines Graphen:

1. Teile die Knoten in 2 Teilmengen.
2. Tausche zwischen diesen so lange paarweise diejenigen Knoten aus, die die Anzahl der durchtrennten Kanten am meisten verringern, bis keine weitere Verbesserung möglich ist.

Die Plazierung erfolgt nun durch rekursive Anwendung des Mincut-Verfahrens (Bild 61). Die Schaltung wird als Graph betrachtet, wobei die Zellen den Knoten und die Verbindungen den Kanten entsprechen. Auf diesen Graphen wird nun *MINCUT* angewandt. Sodann wird auf jeden Teilgraphen wieder *MINCUT* angewandt und so weiter bis jeder Teilgraph nur noch einen Knoten enthält.

8.3.2 Freie Plazierung

Für die Plazierung im allgemeinen Fall gibt es verschiedene deterministische und nichtdeterministische Verfahren.

Algorithmus auf Basis von Kräftegleichungen: Die Zellen werden so plaziert, daß die Gesamtverdrahtungslänge möglichst klein wird. Es wird die optimale Nachbarschaftskonstellation aller Zellen als Lösung eines auf Kräftegleichungen basierenden nichtlinearen Gleichungssystems bestimmt. Ausgehend von der optimalen Nachbarschaftskonstellation werden dann die endgültigen Positionen der Zellen unter Berücksichtigung ihrer Abmessungen berechnet.

Simulated Annealing: Es ist möglich, daß durch Verlängern einer Verbindung zwischen Bausteinen mehrere andere Verbindungen verkürzt werden können. Eine solche Verlängerung wird durch eine zufallsbestimmte Methode (Monte-Carlo-Methode) erreicht.

Durch das Studium der Ähnlichkeiten zwischen der Plazierung von Zellen und der Simulation des Abkühlungsprozesses in Kristallstrukturen von Metallen wurde beim T. J. Watson Research Center ein Plazierungsverfahren entwickelt.

Gegeben ist:

- Zustandsraum S mit der Menge der möglichen Lösungen
- Bewertungsfunktion: S → R zur Bewertung der Qualität jeder Lösung
- Menge von erlaubten Transformationen zwischen Zuständen
  - umkehrbar
  - ergeben nur eine kleine Veränderung der Bewertung

Algorithmus:
S = augenblicklicher Zustand
S' = ein nächster Zustand
E(S) = Bewertungsfunktion
MOVE(S) = führt eine beliebige mögliche Transformation aus
RANDOM: Zufallszahl 0 .. 0.9999

```
PROCEDURE SIMULATED_ANNEALING is
-- Bestimme einen beliebigen Anfangszustand
```

```
-- Bestimme Starttemperatur T

  REPEAT
    REPEAT
      REPEAT
        S' := MOVE(S);
      UNTIL (ACCEPT(S, S', T))
      S := S';
    UNTIL (keine Verbesserung mehr)
    T := T - dT;
  UNTIL (Ende-Kriterium erreicht)
END

FUNCTION ACCEPT (S, S', T): BOOLEAN is
  dE = E(S') - E(S);
  IF (dE <= 0) RETURN (TRUE)
  ELSE BEGIN
    Y := EXP(-dE/T);
    IF (RANDOM < Y) RETURN (TRUE)
    ELSE RETURN (FALSE)
  END
END
```

Es wird also die Boltzmann'sche Wahrscheinlichkeit $P(E) = e^{-(E/kT)}$ berechnet, die mitbestimmt, ob eine Verschlechterung akzeptiert wird oder nicht, bevor eine neue Plazierung errechnet wird. Durch die Verwendung dieser Methode haben Werte von dE, die eine große Verschlechterung bewirken, abhängig von der Temperatur eine geringere Wahrscheinlichkeit, akzeptiert zu werden als kleine Verschlechterungen. Je höher die Temperatur ist, um so größer ist die Wahrscheinlichkeit, daß eine größere Verschlechterung noch akzeptiert wird.

Wichtige Einflußgrößen sind die Starttemperatur, die Geschwindigkeit der Temperaturreduzierung und das Endekriterium.

8.4 Layout

Das Layout von Registern, PLAs usw., die einen regelmäßigen Aufbau haben, kann durch Aneinanderreihung von Grundstrukturen leicht, sogar automatisch, erzeugt werden. Für die Erstellung des Layouts einer logischen Funktion kann man keine festen Regeln geben. Vereinfachend ist aber auch hier die Verwendung von regelmäßigen, mehrfach verwendbaren, aneinanderreihbaren Strukturen (slicing, wiring by abutment).

8.4.1 Stick-Diagramm (symbolisches Layout)

Das Stick-Diagramm ist eine Darstellung auf der Transistor-Ebene, die neben der reinen Verknüpfung der Elemente auch Informationen über die relative Plazierung von Transistoren, Diffusionsgebieten, Metallverbindungen, Polysiliziumverbindungen usw. enthält. Sie werden durch bestimmte Symbole in einem Standardfarbcode dargestellt (blau: Metall, rot: Polysilizium, grün: Diffusion, gelb: Implantierung, schwarz: Durchkontaktierung). Es enthält somit fast alle Informationen des endgültigen Layoutplans. Das geometrische Layout kann aus dem symbolischen Layout erzeugt werden, indem dieses unter Berücksichtigung der design rules und anderer Einschränkungen (z. B. Lage von Anschlüssen) kompaktiert wird (Bild 62).

8.4.2 Weinberger-Array

Die Schwierigkeit des Layouts einer großen Logikschaltung wurde schon früh erkannt. Das Weinberger-Array ist ein Versuch, die Komplexität zu verringern, indem nur eine Art von Gattern (NOR- oder NAND-Gatter) verwendet werden und diese in einer matrixartigen Struktur angeordnet werden. Dabei verlaufen die Ein- und Ausgangssignale horizontal, die pull-up- und Masseleitungen vertikal (Bild 63). Diese Anordnung kann sowohl manuell als auch automatisch leicht erzeugt werden, ist jedoch nicht sehr effizient.

8.4.3 Line-of-Diffusion-Methode

Viele Schaltungen können in Form einer ununterbrochenen Reihe von Transistoren, deren Source bzw. Drain jeweils ineinander übergehen, realisiert werden. Bei CMOS-Gattern werden zwei solcher Reihen (mit p-Kanal- und n-Kanal-Transistoren) benötigt. Das Layout einer derartigen Schaltung kann mit folgendem Algorithmus erzeugt werden (Bild 64):

1. Konstruiere den Graphen G, wobei die Kanten von G den Transistoren und die Knoten den Source- bzw. Drainanschlüssen entsprechen.

   CMOS: Konstruiere Graphen G_n und G_p für die n- bzw. p-Kanal-Teile der Schaltung.

2. Suche einen Euler-Pfad in G, d. h. einen (nicht geschlossenen) Pfad, der jede Kante genau einmal durchläuft. Ein solcher Pfad existiert genau dann, wenn nicht mehr als 2 Knoten einen ungeraden Grad haben.

   CMOS: Suche Euler-Pfade in G_n und G_p, die die Kanten in der selben Reihenfolge durchlaufen. Damit können die Gates kreuzungsfrei verlegt werden.

3. Können keine Euler-Pfade gefunden werden, muß der Graph partitioniert und der Algorithmus auf die Teilgraphen angewendet werden.

4. Ordne die Transistoren in der gefundenen Reihenfolge als ununterbrochene Reihe (CMOS: 2 parallele Reihen) in der Diffusionsebene an. Die Poly-Gates kreuzen die Reihen im rechten Winkel. Zusätzliche Verbindungen werden in Metall über bzw. zwischen den Reihen verlegt.

8.5 Routing

Nach der Plazierung muß die genaue Verlegung der Verbindungsleitungen bestimmt werden. Dabei müssen bestimmte Verdrahtungsregeln beachtet werden:

- Metallverbindungen sind Polysilizium-Verbindungen vorzuziehen
- Taktleitungen für synchronisierte Bausteine (FF, Zähler) sind immer am Anfang des Verdrahtungsprozesses und in Metall anzufertigen
- Spannungsversorgungsleitungen sind ebenfalls in Metall und mit genügender Breite zu verlegen

Wenn nur eine Metallebene vorhanden ist, stellt sich das Problem der Planarisierung, d. h. das Aufbringen der Verdrahtung, ohne daß Überkreuzungen der Leitungen auftreten. Ein Algorithmus zur Planarisierung ist das Verfahren von Bader (Bild 69):
Gegeben ist der Graph G, dessen Knoten den Bauelementen und dessen Kanten den Verbindungen entsprechen.

1. Finde einen Hamilton-Zyklus in G (d. h. einen geschlossenen Pfad, der jeden Knoten genau einmal durchläuft).

2. Stelle G so dar, daß der Hamilton-Zyklus einen Kreis bildet und alle anderen Kanten in seinem Inneren liegen (normierte Darstellung).

3. Konstruiere den Hilfsgraphen G' wie folgt: Jeder Kante von G, die nicht zum Hamilton-Zyklus gehört, wird ein Knoten von G' zugeordnet. Zwischen zwei dieser Knoten gibt es eine Kante genau dann, wenn sich die zugehörigen Kanten in der normierten Darstellung kreuzen.

4. Wähle einen beliebigen Knoten in G' und markiere ihn mit '+'. Markiere alle seine Nachfolger mit '−', alle Nachfolger von diesen wieder mit '+' usw. Falls G' partitioniert ist, führe dies für alle Teilgraphen durch. Tritt dabei ein Konflikt auf, so ist der Graph G nicht planarisierbar.

5. Werden die Verbindungen in G, deren Knoten in G' mit '+' markiert wurden, außerhalb der Hamiltonlinie verlegt, die mit '−' markierten innerhalb (oder umgekehrt), so ist eine kreuzungsfreie Darstellung möglich.

Im Bedarfsfall kann auch die Polysiliziumebene zur Verdrahtung verwendet werden. Bei Mehrebenenverdrahtung werden die Leitungen so vorsortiert, daß Leitungen mit ungefähr demselben Richtungsverlauf in eine Ebene aufgenommen werden. Heutige Fertigungsprozesse erlauben 3 oder 4 Metallebenen.

8.5.1 Routing-Verfahren

Lee-Algorithmus: Der klassische Lee-Algorithmus ist ein wavefront-Mechanismus, d. h. vom Startpunkt ausgehend werden jeweils die nächsten Nachbarn in einem Gitter untersucht, bis das Ziel erreicht ist (Bild 70).

Ein Nachteil des Lee-Algorithmus ist, daß unter Umständen erst ein sehr großer Teil der Verdrahtungsfläche untersucht werden muß, bis eine Lösung gefunden wird. Auch werden besondere Eigenschaften der Topologie des Chips nicht berücksichtigt (z. B. Verdrahtungskanäle nur in x- und y-Richtung). Dies kann durch heuristische Modifikationen des Lee-Algorithmus verbessert werden (z. B. Suche von Start- und Zielpunkt gleichzeitig, schnellere Ausbreitung in Vorzugsrichtung).

Beispiel:
Eine sinnvolle Modifikation zielt darauf ab, den Flächenbedarf der gesamten Verdrahtung zu minimieren. Dabei wird jedem Gitterelement ein Gewicht zugeordnet, das der Anzahl benachbarter freier Zellen entspricht. Durch Aufsummieren der Gewichte über alle möglichen Pfade wird dann der optimale Pfad gefunden.

Line-Search-Algorithmen: 75 % aller Verbindungen sind relativ einfach und können als fast direkte Verbindung gelegt werden. Es ist also lohnend, vom Start- und Zielpunkt parallele Linien vorzutreiben und zu versuchen, diese durch eine rechtwinklig dazu liegende zu verbinden. Line-Search-Algorithmen gehen von dieser Überlegung aus und versuchen, mit heuristischen Erweiterungen die komplizierteren Fälle zu lösen.

Channel Routers: Die Verdrahtung verläuft in rechteckigen Kanälen, wobei nur an zwei gegenüberliegenden Seiten (normalerweise den Längsseiten) Anschlüsse liegen. Das Verfahren hat 2 Phasen:

1. Zuordnung der Verdrahtung zu den Verdrahtungskanälen

2. Verdrahtung der einzelnen Kanäle, wobei lange Leitungen in die Mitte des Kanals gelegt werden und kurze nach außen.

Hierarchical Routing: Verdrahtung wird hierarchisch durchgeführt:

1. Verdrahtung der Zellen eines Moduls
2. Verdrahtung zwischen Modulen
3. Verdrahtung der Makromodule

Dadurch wird die Komplexität verringert und unnötige Abwege verhindert. Wegen der geringeren Anzahl von Leitungen in jedem Schritt können auch Verfahren mit exponentieller Laufzeit eingesetzt werden. Jedoch ist die optimale Zusammenfügung von lokal optimierten Blöcken nicht unbedingt ein globales Optimum.

8.6 Zellenorientierter Entwurf

Grundidee: Bereitstellung von wiederverwendbaren, vorgefertigten und verifizierten Schaltungsteilen. Der Designaufwand für das Layout dieser Schaltungsteile wird eingespart. Nicht mehr Transistoren und die technologischen Randbedingungen bilden die Basis des Entwurfs, sondern logische Einheiten wie Gatter, Flip-Flops oder Speicher. Diese Einheiten nennt man Zellen (Bild 66). Der Layout-Entwurf beschränkt sich auf die Plazierung und Verdrahtung dieser Zellen.

Zellen und Zellenbibliotheken werden nach bestimmten Richtlinien entworfen. Die Zellen sind durch ihre logische Funktion, ihr elektrisches Verhalten, ihre geometrische Form und ihre Anschlußstruktur charakterisiert. Die Auswahl der Grundschaltungen für eine Bibliothek von Zellen erfolgt aufgrund der universellen Einsetzbarkeit der Zellen. Eine zu große Artenvielfalt hochspezialisierter Zellen verschlechtert die Handhabbarkeit.

Nachteile zellenorientierter Designs:

- ineffektive Flächenausnutzung
- niedrigere Verarbeitungsgeschwindigkeit
- u. U. höhere Verlustleistung

Vorteile:

- hohe Designsicherheit
- kürzere Designzeit
- geringere Designkosten

8.6.1 Zellenorientierte Entwurfsverfahren

S. Bild 60, 67.

Gate-Array: fest vorgegebene regelmäßige Struktur von Transistor-Grundzellen mit dazwischenliegenden Verdrahtungskanälen. Aufbau auf vorgefertigtem Master-Chip.

Standardzellendesign: Zellen mit konstanter Höhe, aber variabler Breite. Oben und unten sind die Ein- / Ausgänge herausgeführt, seitlich die Betriebsspannungen und evtl. die Taktsignale, deren Verbindungen durch Anreihen der Zellen erfolgen. Zwischen den linearen Anordnungen der Standardzellen liegen Verdrahtungskanäle für die Signale.

Makro- / Universalzellen: komplexe Zellen beliebiger Größe bis hin zu kompletten Mikroprozessoren. Anordnung und Verdrahtung beliebig.

8.6.2 Zellgeneratoren

Zur automatischen Erzeugung von 'ähnlichen' Zellen gibt es Zellgeneratoren.

Einfach parametrisierbare Zellen: Zusammenstellung von vorentworfenen Grundelementen (Zellelemente, subcells). Die Kombination aus einer Anfangs-, mehreren Mittel- und einer Endzelle führt zu einer funktionierenden Zelle (z. B. einem Gatter mit n Eingängen). Die Verbindungen werden durch Aneinandersetzen (butting) hergestellt (Bild 68).

Parameter: Verarbeitungsbreite bei ALUs, Zählerbreite, Anzahl der Eingänge usw.

Funktional parametrisierbare Zellen: Automatische Generierung der Zellen nach einer funktionalen Beschreibung.

Beispiele:
PLA aus boole'scher Gleichung
ROMs, FSMs

Ein wichtiger Aspekt eines Zellgenerators ist, daß nicht nur das Zellenlayout, sondern auch ein Simulationsmodell und Prüfmuster generiert werden.

8.6.3 Layout-Erstellung für zellenorientierten Entwurf

Ablauf der automatischen Layout-Erstellung für zellorientierte Entwurfsverfahren bei CAD-Systemen:

1. Logischer Entwurf auf Basis der Zellbibliothek der gewünschten Standardentwurfsverfahren und Technologie (z. B. 3 μ-CMOS-Standardzellen)

2. Simulation auf Basis der zu den Zellen gehörenden Simulationsmodelle (timing, fan-in-, fan-out-abhängige Ausgangssignale), Fehlersimulation

3. Automatische Umsetzung der Netzliste des Logikentwurfs in Zellen und ihre Verdrahtung

   (a) relative Plazierung: Nachbarschaftsverhältnisse zwischen den einzelnen Zellen werden festgelegt. Die relative Plazierung nimmt noch keine Rücksicht auf die geometrischen Ausmaße der Zellen. Optimierungskriterium ist das Minimum der Summe der Verbindungslängen.

   (b) Festlegung der Pinanschlüsse (automatisch oder manuell)

   (c) absolute Plazierung: Zuordnung der Zellen zu entsprechenden festgelegten Zellenreihen

   (d) globale Verdrahtung: Es wird untersucht, ob die Schaltung prinzipiell verdrahtbar ist. Resultat: Belegungsdichte der einzelnen Kanäle.

   Manuelle Eingriffsmöglichkeiten: Verschiebung, Vertauschen, Zusammenschieben von Zellen.

   (e) Verdrahtung: Es wird sukzessiv Kanal für Kanal verdrahtet

4. Extraktion der Leitungslängen und Simulation unter Einbeziehung der Leitungsverzögerungen

5. Aus den Zellendaten und den Verdrahtungsdaten wird ein Band erzeugt (z. B. im CIF-Format, s. u.) und zum Halbleiterhersteller geschickt. Meist sind noch Testmuster beigefügt, so daß die Funktionsfähigkeit schon beim Hersteller überprüft werden kann.

8.7 Layout-Sprache CIF

Mit CIF (Caltech Intermediate Format) können graphische Strukturen auf Masken beschrieben werden. Die Darstellung einer integrierten Schaltung in CIF ist standardisiert, sie kann direkt zur Herstellung der Masken verwendet werden. Die Maßangaben erfolgen in 1/100 μm, die Richtungsangaben durch Richtungsvektoren.

CIF-Kommandos (vgl. Bild 65):

Box: B 25 60 80 40 0 -1;
Box length 25, width 60, center 80, 40, direction 0, -1;

Polygon: P 0 0 0 20 -30 20;
Polygon a 0, 0 b 0, 20 c -30, 20 d 0, 0;

Wire: W 5 3 9 3 5 10 5 10 2;
Wire width 5 a 3, 9 b 3, 5 c 10, 5 d 10, 2;

Layer: L ND;
Layer N Diffusion;

Define: DS 57 100 1; ... ; DF;
Definition Start #57 a/b = 100/1; ... ; Definition Finish;
Anmerkung: alle Maßangaben zwischen DS und DF werden mit a/b skaliert.

Call: C 57 MX R 0 -1 T 10 20;
Call symbol 57 Mirrored in X Rotated to 0, -1 then Translated to 10, 20;

Delete: DD 57;
Delete Definition #57;

8.8 Layout-Verifikation

Diese ist hauptsächlich bei Vollkundenentwürfen nötig, da Zellen der Standardentwurfsverfahren vor der Verwendung verifiziert werden ('correct by construction').

Es kann bei komplexen Chips über 250 000 Dollar kosten, die Masken und den ersten Prototyp eines Chips zu erstellen. Deswegen wird versucht, durch entsprechende Verifikationsmaßnahmen zu erreichen, daß der Chip beim ersten Prototyp schon funktioniert ('working on first silicon'). Dies gelingt heute bei entsprechenden Entwurfssystemen bei über 90 % aller Chipentwürfe.

Tools für die Layout-Verifikation:

DRC (Design Rule Checker): Überprüfung der prozessabhängigen Parameter des Layouts wie Minimalabstände und Minimalbreiten. Es gibt auch komplexere Designregeln , z. B. für die Breite der Stromversorgungsbahnen oder die Anzahl und Länge der Polysiliziumverbindungen.

Hierarchischer DRC: Wenn eine Zelle mehrmals verwendet wird, so wird deren innere Struktur nur einmal überprüft. Später werden nur noch ihre Verbindungen zu anderen Zellen geprüft.

Circuit Extractor: Rückgewinnung des Transistornetzes oder der logischen Funktion aus dem Layout zur funktionalen Verifikation. Es werden auch die Leitungskapazitäten und -widerstände extrahiert und dann in der Simulation verwendet.

9 Bausteine für halbkundenspezifische ICs

9.1 Einleitung

Es gibt eine Anzahl von Bausteinen, die sich besonders für einen schnellen Entwurf kundenspezifischer Schaltungen eignen. Der Entwurf wird dabei einerseits durch schon vorgefertigte Strukturen auf dem Chip und andererseits durch konsequente Verwendung von vorgefertigten Zellen vereinfacht und beschleunigt. Diese Bausteine werden auch als semicustom integrated circuits bezeichnet, da nur ein Teil des Entwurfes kundenspezifisch durchgeführt wird, während der größte Teil des Entwurfs auf vorgefertigten Strukturen beruht. Das Gegenteil dazu ist der full-custom-Entwurf, wobei der gesamte Chip in allen Details kundenspezifisch und für eine bestimmte Anwendung entworfen wird.

9.2 Programmable Logic Devices (PLD)

Programmierbare Logikbausteine bestehen aus regelmäßig angeordneten Logikgattern mit vorgegebener Verdrahtungsstruktur, deren logische Funktion durch Programmierung festgelegt werden kann. Die Programmierung erfolgt entweder permanent durch Durchbrennen bestimmter Verbindungen der Verdrahtungsstruktur (fusible links) oder mit EPROM-, EEPROM- oder RAM-Zellen. Das grundlegende Prinzip ist die Realisierung von kombinatorischer Logik durch die disjunktive Normalform (sum of products). PLDs ermöglichen die Programmierung der DNF in einen integrierten Baustein (Bild 71, 72).

Diese Bausteine bestehen aus einem UND-Feld und einem ODER-Feld und unterscheiden sich darin, ob das UND-Feld, das ODER-Feld oder beide Felder programmierbar sind (Bild 73).

PROM (Programmable Read Only Memory): Das UND-Feld ist so aufgebaut, daß es die n Eingänge voll dekodiert. Alle möglichen $m = 2^n$ Produktterme können durch Programmierung des ODER-Arrays realisiert werden. Dies entspricht dem Aufbau eines Speichers.

Die Programmierung von ROMs wird vom Hersteller durch Festlegung der Maske nach Wunsch des Kunden durchgeführt; PROMs sind vom Anwender programmierbar (durch Durchbrennen bestimmter Verbindungen); EPROMs können vom Anwender programmiert werden und durch UV-Licht wieder gelöscht werden; EEPROMs können elektrisch gelöscht werden.

Vorteil von ROMs: Jede Eingangskombination kann dekodiert werden (sinnvoll bei Adressdekodierung)
Nachteil: Großer Platzbedarf

Beispiel 1-bit-Volladdierer:

C_{in}	A	B	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

PLA (Programmable Logic Array): Beim PLA ist das UND-Feld und das ODER-Feld programmierbar. Es kann eine beliebige Teilmenge aller möglichen Produktterme programmiert werden und es können beliebige Kombinationen von Produkttermen im ODER-Feld zusammengefaßt und auf die Ausgänge geführt werden. PLAs werden durch den Hersteller programmiert; FPLAs können durch den Anwender programmiert werden (Durchbrennen der fusible links); EPLAs können auch wieder gelöscht werden. PLAs besitzen eine große Flexibilität. Produktterme können durch die Programmierbarkeit des ODER-Feldes mehrfach benützt werden.

Beispiel 1-bit-Volladdierer:

$S = \overline{C_{in}}\,\overline{A}B + \overline{C_{in}}A\overline{B} + C_{in}\overline{AB} + C_{in}AB$

$C_{out} = \overline{C_{in}}AB + C_{in}\overline{A}B + C_{in}A\overline{B} + C_{in}AB = AB + C_{in}A + C_{in}B$

PAL (Programmable Array Logic): Bei PALs ist nur das UND-Feld programmierbar, das ODER-Feld ist festgelegt. Es werden immer Gruppen von Produkttermen an einem Ausgang zusammengefaßt. Dadurch kann eine höhere Integrationsdichte und eine kürzere Verzögerungszeit gegenüber PLAs erreicht werden.

PLDs erlauben wesentlich kompaktere Entwürfe von kombinatorischen Schaltungen als bei der Verwendung von SSI- / MSI-Bausteinen ohne wesentliche Steigerung der Entwurfszeit. Der Entwurf wird durch Software der PLD-Hersteller unterstützt, die Programmierung erfolgt durch entsprechende Programmiergeräte. Es gibt eine große Auswahl von PLDs mit der unterschiedlichsten Anzahl von UND- und ODER-Gattern. Zusammen mit Registern kann mit PLDs ein Steuerwerk (endlicher Automat) einfach realisiert werden.

Varianten von PLDs:

- EEPLDs: Elektrisch löschbare PLDs (s. o.)
- Kombinationen von Registern oder RAMs auf PLDs

- verschiedene Ein- / Ausgangsstufen: Latch, tri-state, invertierend usw.
- Logic Cell Arrays / Programmable Gate Arrays: Ein PGA enthält eine Matrix von 'configurable logic blocks', die eine beliebige kombinatorische Funktion von mehreren Eingangsvariablen sowie eine Latch- / Flip-Flop-Funktion realisieren können; außerdem I/O-Blöcke. Die Verbindungen zwischen ihnen sind frei programmierbar (Bild 74).

9.3 Gate-Arrays

Gate-Arrays (uncomitted logic arrays) bestehen aus einer matrixförmigen Anordnung von Transistorzellen, die beliebig untereinander und mit den Pins verbunden werden können (Bild 75). Der größte Teil der Chipstruktur ist vorfabriziert, es wird nur die Maske mit der Verdrahtungsinformation benötigt. Die Anzahl der Gatter variiert zwischen 300 und 50 000. Der Aufbau der Zellen unterscheidet sich je nach verwendeter Technologie:

- CMOS-Gate-Arrays mit mittlerer Schaltgeschwindigket
- ECL-Gate-Arrays für sehr hohe Schaltgeschwindigkeit.

Bei großer Gatterzahl können zwei Verdrahtungsebenen benutzt werden.

Beim Gate-Array-Entwurf steht meistens eine größere Anzahl von vorentworfenen Funktionsblöcken zu Verfügung, die dann nur noch auf dem Gate-Array plaziert und verdrahtet werden müßen. Sonst sind für das Layout Methoden wie das Weinberger-Array oder die line-of-diffusion-Methode anwendbar.

9.3.1 Das CMOS-Gate-Array UA4

Das Gate-Array UA4 (AMI) besteht aus 22 x 14 Kernzellen. Jede Kernzelle besteht aus einer großen Halbzelle mit 3 n-Kanal- und 3 p-Kanal-Transistoren, einer kleinen Halbzelle mit 2 + 2 Transistoren und underpasses in Polysilizium (Platz für Verdrahtungskanäle) (Bild 76-77). Am Rand des Chips befinden sich verschiedene Typen von Peripheriezellen, die als Input, Output, tri-state-Output, open-drain-Output oder bidirektionaler I/O konfiguriert werden können (Bild 78).

Bei Inputs von TTL-Signalen muß der Spannungspegel angepasst werden, da die Logikpegel von TTL für CMOS ungeeignet sind.

	CMOS	TTL
V_{low}	0 V	$\leq$ 0.4 V
V_{high}	5 V	$\geq$ 2.4 V

Dies wird durch einen Level-Translator erreicht, der aus einem großen n-Kanal- (LSN) und einem kleinen p-Kanal-Transistor (SP) aufgebaut ist (Bild 79). Bei V_{low} leitet nur der p-Kanal-Transistor, da $V_{low} < V_{th_{LSN}}$. Bei V_{high} sind SP und LSN durchgeschaltet. Durch unterschiedliche Widerstände von SP und LSN liegt aber der Ausgangspegel auf $< V_{th}$.

9.4 Standardzellen

Beim Standardzellenentwurf wird eine Schaltung aus grundlegenden funktionalen Bausteinen (Zellen) zusammengesetzt und diese untereinander verdrahtet. Die Zellen sind in der Bibliothek eines Entwurfssystems gespeichert. Es existiert keine vorbereitete Chipstruktur, so daß der Herstellungsaufwand derselbe wie beim Vollentwurf ist. Durch die Beschränkung auf vorentworfene und getestete Standardzellen kann aber der Entwurf wesentlich vereinfacht und beschleunigt werden. Die Zellen besitzen alle die gleiche Höhe, alle Ein- und Ausgänge befinden sich am oberen und unteren Ende der Zellen. Die Breite der Verdrahtungskanäle kann den Anforderungen angepaßt werden (Bild 80). Der Standardzellenentwurf erlaubt eine wesentlich höhere Integrationsdichte als Gate-Arrays. Es können auch Analogzellen und dynamische Zellen verwendet werden.

Beispiel 3 μ-CMOS-Standardzellen:
Das Layout erfolgt auf einem Grundraster von 10 μm. Linien dürfen nur auf diesem Raster laufen und Zellen dürfen nur in diesem Raster gesetzt werden. Dies sichert die Einhaltung der geometrischen Entwurfsregeln (Bild 66, 81).

Die Zellen bestehen aus einem Bereich für n-Kanal- und einem Bereich für p-Kanal-Transistoren. V_{dd} und V_{ss} werden am oberen bzw. unteren Ende der Zelle durchgeführt und durch spezielle Randzellen angeschlossen. Die Zellen müssen in Reihen liegen, um die Stromversorgung nicht zu unterbrechen. Die Verdrahtung zwischen den Zellen läuft in Polysilizium und Metall zwischen den Reihen. Verbindungen durch die Zellreihen erfolgen mit speziellen feedthrough-Zellen. Die p-Wannen der einzelnen Zellreihen sollen gegeneinander zeigen und nicht zu den Peripheriezellen hin (zur Verhinderung des latch-up-Effekts). Die Peripheriezellen enthalten die bonding pads und werden an den Rändern plaziert.

9.5 Makrozellen

Bei Makrozellen besteht nicht mehr die Beschränkung der Größe der einzelnen Zellen. Die Zellbibliothek besteht aus wesentlich komplexeren Bausteinen wie ROMs, RAMs, PLAs oder Mikroprozessoren. Die Plazierung und Verdrahtung ist völlig frei.

9.6 Auswahl der ASIC-Bausteine

Am Beginn einer ASIC-Entwicklung steht die Analyse der Kundenschaltung. Die erste und wichtigste Entscheidung ist, ob man überhaupt ASICs verwenden will oder die Schaltung mit Standard-Chips realisiert. Falls man sich für ASICs entscheidet, folgt dann zusammen mit anderen Kriterien wie Stückzahl, Einsatzbedingungen usw. die Auswahl des ASIC. Kriterien sind:

- Kosten (Entwicklungskosten, Stückkosten)
- Entwicklungszeit
- technologische Möglichkeiten (Geschwindigkeit, Stromverbrauch)
- Realisierungsmöglichkeit durch bestimmten ASIC-Typ
- Entwicklungsmöglichkeiten (Designer, CAD-Unterstützung usw.)

Vergleich der Realisierungsmöglichkeiten einer Schaltung:

Standard-Chips (SSI / MSI):

- Entwicklungskosten sehr niedrig
- Stückkosten bei größeren Stückzahlen sehr hoch
- Entwicklungszeit für Prototyp klein
- alle Geschwindigkeiten, aber relativ hoher Stromverbrauch
- praktisch alle Schaltungen sind möglich

PLDs:

- geringe Entwicklungskosten
- bei großer Stückzahl relativ teuer
- kurze Entwicklungszeit
- größere Auswahl an Geschwindigkeiten und Komplexität
- nur für einfache Schaltungen
- einfache Entwicklung, leistungsfähige Design-Software

Gate-Arrays:

- Entwicklungskosten abhängig von der Komplexität der Schaltung und der Entwurfsunterstützung durch CAD-System; mäßige Prototypkosten
- Bei großer Stückzahl relativ hohe Kosten
- Gate-Array kann meist nicht voll ausgenutzt werden

- Entwurfszeit stark abhängig von der Entwurfsunterstützung
- in CMOS und ECL, teilweise auch für Analoganwendungen
- für einfache bis mittlere Komplexität; ROMs, RAMs ungünstig zu realisieren
- sehr gute Unterstützung und Automatisierung durch CAD-Systeme

Standardzellen:

- Entwicklungskosten ähnlich wie Gate-Arrays; Prototypkosten wesentlich höher
- Bei großen Stückzahlen billiger als Gate-Arrays
- Entwicklungszeit wie Gate-Arrays
- große Auswahl an Technologien
- Analogzellen möglich
- sehr große Flexibilität, auch für komplexere Schaltungen
- CAD-Systeme wie bei Gate-Arrays

Vollkundenentwurf:

- sehr aufwendige Entwicklung, hohe Entwicklungskosten
- Bei sehr großen Stückzahlen kostengünstig
- lange Entwicklungszeit
- sehr große Auswahl an Technologien
- extrem große Flexibilität, auch für sehr komplexe Schaltungen
- aufwendige CAD-Systeme, geringer Automatisierungsgrad

Auswahl des ASIC-Herstellers:

- abhängig von gewünschter Technologie und Komplexität
- bei PLDs spezielle Versionen nur von einzelnen Herstellern
- bei Gate-Arrays und Standardzellen meist mehrere Hersteller und kompatible Bibliotheken
- wichtig ist Unterstützung durch Hersteller und entsprechende CAD-Systeme
- second source vorhanden?

Der Entwurf kann auch durch ein Design-Center des Herstellers unterstützt werden, so daß die hohen Investitionskosten für CAD-Systeme und die Lernkurve für den Kunden entfallen. Es gibt unterschiedliche Möglichkeiten der Aufteilung des Entwurfs auf Kunden, Design-Center und Hersteller (Bild 82, 83).

10 Computerunterstützter Entwurf

10.1 Einleitung

Heutzutage sind Schaltungen und Chips so komplex, das man sie ohne entsprechende computerunterstützte Entwurfssysteme nicht entwickeln könnte. Die Computerunterstützung reicht von Editoren für die Bearbeitung eines Entwurfs auf verschiedenen Ebenen, der Verifikation auf jeder Ebene und der zentralen Datenhaltung der Entwurfsdaten in einer Datenbank bis zur Automatisierung einzelner Entwurfsschritte.

10.2 Teile eines Entwurfssystems

Die frühen Programme zur Unterstützung des Entwurfs deckten nur einzelne Teilaspekte des Entwurfs ab. Jedes hatte eine eigene Datenhaltung mit eigenen Datenformaten, eine eigene Benutzerschnittstelle usw.

Die Komponenten eines modernen integrierten Entwurfssystems sind (vgl. Bild 84):

- Datenbank zur zentralen Verwaltung aller Entwurfsdaten. Die Datenbank enthält den hierachisch strukturierten Entwurf, dargestellt in verschiedenen Abstraktionsebenen (funktional, logisch, Schaltkreis, Layout). Ebenso dient sie zur Verwaltung der Zellbibliotheken (logische Modelle, Simulationsmodelle, elektrische Parameter, Dokumentation usw.).
- Benutzerschnittstelle zur einheitlichen Benutzerführung
- Editoren für Text, RT-Symbole, Logiksymbole, Transistornetze, symbolische und geometrische Layouts
- Zugehörige Plot- und Druckprogramme
- Ausgabeprogramme für bestimmte Formate (z. B. CIF)
- Simulationsprogramme für die einzelnen Abstraktionsebenen
- Prüfprogramme (ERC, DRC)
- Tools, die einzelne Entwurfsschritte automatisch durchführen (Logikoptimierer, Router)

10.3 Silicon Compiler

Eine exakte Definition dieses Begriffs zu geben, ist derzeit schwierig, da sich dieses Feld noch in der Entwicklungsphase befindet. Im allgemeinen bezeichnet man als silicon compiler design-automation-Programme, die mehrere Entwurfsschritte automatisch durchführen (Bild 85). Dies reicht von relativ einfachen Tools, die z. B. aus einer Beschreibung einer FSM automatisch ein Layout eines PLA und einiger Register erzeugen, bis hin zu komplexen Programmsystemen, die z. B. eine ISPS-Beschreibung einer CPU in ein entsprechendes Layout umsetzen. Einfache Programme, die eine struktur- oder logikorientierte Eingabe benötigen, werden manchmal auch als silicon assembler bezeichnet.

Ein silicon compiler im engeren Sinne, der aus einer Verhaltensbeschreibung ein Layout erzeugt, muß folgende Schritte durchführen (vgl. Bild 86):

Analyse: Umsetzung der Verhaltensbeschreibung in eine strukturelle Beschreibung (Extraktion von Kontroll- und Datenfluß). Hierbei werden Techniken wie bei Compilerbau angewandt (Datenflußgraph, Erkennen von Parallelität)

scheduling: Bestimmung der benötigten Hardware-Funktionselemente unter Berücksichtigung verschiedener Einschränkungen (Platzbedarf, Geschwindigkeit)

Synthese: Zerlegung der Funktionselemente in Grundelemente (Gatter, Flip-Flops)

Modulerzeugung: Erzeugung von (verschieblichem) Layout für die einzelnen Module (bei Bedarf)

Layout: Plazierung und Verdrahtung der Module

Um näherungsweise die Qualität eines handoptimierten Entwurfs zu erreichen, muß der Entwurf in jedem Schritt optimiert werden (Bild 87). Für die höheren Ebenen ist dies sehr schwierig. Derzeit wird der Einsatz von regelbasierten Systemen für diesen Zweck untersucht. Der Design Automation Assistant (CMU / Bell Labs) ist z. B. ein regelbasiertes System, das ISPS-Beschreibungen in Layouts umsetzen kann, die mit denen guter menschlicher Designer vergleichbar sind (Bild 88).

Kommerzielle silicon compiler, z. B. Genesil, sind erhältlich und werden in der Industrie zunehmend eingesetzt (Bild 89, 90). So soll der Mikroprozessor Motorola 68040 unter Einsatz eines silicon compilers entwickelt worden sein.

A Literaturverzeichnis

1. Mead, C., Conway, L.: Introduction to VLSI Systems, Addison-Wesley 1980
2. Hörbst, E., Nett, M., Schwärtzel, H.: VENUS: Entwurf von VLSI-Schaltungen, Springer-Verlag 1986
3. Hicks, P. J.: Semi-Custom IC Design and VLSI, Peregrinus Ltd. 1983
4. Informatik Spektrum, Band 9, Heft 4, August 1986 (Sonderheft VLSI)
5. Ayres, R. F.: Silicon Compilation and the Art of Automatic Microchip Design, Prentice-Hall, 1983
6. Glasser, L. A., Dobberpuhl, D. W.: The Design and Analysis of VLSI Circuits, Addison-Wesley 1985
7. Hartenstein, R. W.: KARL-III Manual, Lehrstuhl für Rechnerstrukturen und Technische Informatik, Universität Kaiserslautern 1986
8. White, J. K, Kundert, K., Moore, P., Saleh, R. A., Sangiovanni-Vincentelli, A., Newton, A. R.: RELAX 2.2 User's Guide, Department of Electrical Engineering and Computer, Science, University of California, Berkeley 1984
9. Austria Microsystems International: SuperSceptre Design System User's Handbook, July 1986
10. AEG Telefunken: DISIM Version 2.5 User's Manual, Stand 1983
11. Brayton, R. K., Hachtel, G. D., McMullen, C. T., Sangiovanni-Vincentelli, A. L.: Logic Minimization Algorithms for VLSI Synthesis, Kluver Academic Publishers 1986
12. Kowalski, T. J.: An Artificial Intelligence Approach to VLSI Design, Kluver Academic Publishers 1986
13. Antognetti, P., Anceau, F., Vuillemin, J.: Microarchitecture of VLSI Computers, Martinus Nijhoff Publishers 1985
14. Hilberg, W.: Grundprobleme der Mikroelektronik, Oldenburg Verlag 1982
15. Wölcken, K.: E.I.S., Informatik Spektrum, S. 108, Band 7, Heft 2, April 1984
16. Abel, E., Gorges, M., Heckl, H., Plöger, P. G., Vierhaus, H. T., Wölcken, K.: E.I.S. Entwurfswerkzeuge, Informationstechnik, S. 177, Jahrgang 28, Heft 3, 1986

17. MOSIS: Chips and Boards through MOSIS, COMPCON Spring 1985

18. E.I.S.: E.I.S. Zeitung, Nr. 3, April 1986

19. Aylor, J., Parrish, E. A., Pocek, K. L.: A Semicustom VLSI Design Course Supported by the General Electric Microelectronics Center, IEEE Transactions on Education, S. 85, Vol. 29, No. 2, May 1986

20. Oberschelp, W., Vossen, G.: Rechneraufbau und Rechnerstrukturen, Oldenburg Verlag 1986

21. Robinson, P.: Overview of Programmable Hardware, Byte, Jan 1987, S. 197

22. Coli, V. J.: Introduction to Programmable Array Logic, Byte, Jan. 1987, S. 207

23. AMI: AMI Uncomitted Logic Array Design Manual, American Microsystems Inc., Santa Clara, 1982

24. Elgert, C.: ASIC Layout auf Personal Computern, Elektronik, Nr. 3, 1987, S. 149

25. Silicon Compilers Inc.: Intelligent Designing means Intelligent Silicon Compilation, Seminar Series, March 1987

26. Waldschmidt, K.: Simulation in Modern Electronics, ESC 83, S. 44, Springer Verlag

27. Grass, W.: Steuerwerke, Springer-Verlag 1978

28. Pantano, G.: Grundlagen der Logiksimulation, Teil 1: Elektronik 12/13. 6. 86 S. 55, Teil 2: Elektronik 13/27. 6. 86 S. 75

29. Horstmann, J.: Testproblematik komplexer VLSI-Schaltungen, Teil 1: Elektronik 5/6. 3. 87 S. 133, Teil 2: Elektronik 6/20. 3. 87 S. 113

30. Sangiovanni-Vincentelli, A.: Simulated Annealing: Theory and Application to the Placement of Integrated Circuits, E.I.S. CAD-VLSI Sommerschule, Juni 1986

31. Weste, N., Eshraghian, K.: Principles of CMOS VLSI Design, Addison-Wesley, 1985

32. Goto, S.: Design Methodologies, Advances in CAD for VLSI, Volume 6, North Holland, 1986

33. Hartenstein, R. W.: Hardware Description Languages, Advances in CAD for VLSI, Volume 7, North Holland, 1987

34. Borrione, D.: From HDL Descriptions to Guaranteed Correct Circuit Designs, North Holland, 1987

35. Kern, W.: Anwendungsspezifische integrierte Schaltungen, Hüthig Verlag Heidelberg, 1986

36. Post, H.-U.: Entwurf und Technologie hochintegrierter Schaltungen, Teubner Verlag Stuttgart, 1989

37. Gajski,D.: Silicon Compilation, Addison-Wesley, 1988

B Abbildungen

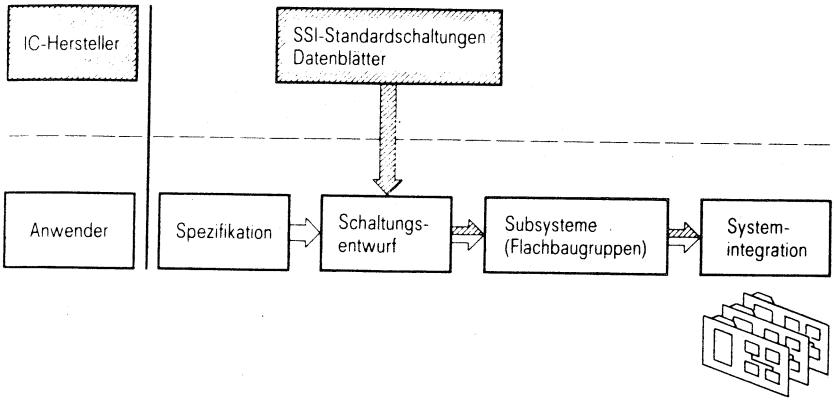


Bild 1.1. Problemlösung mit SSI-Bausteinen

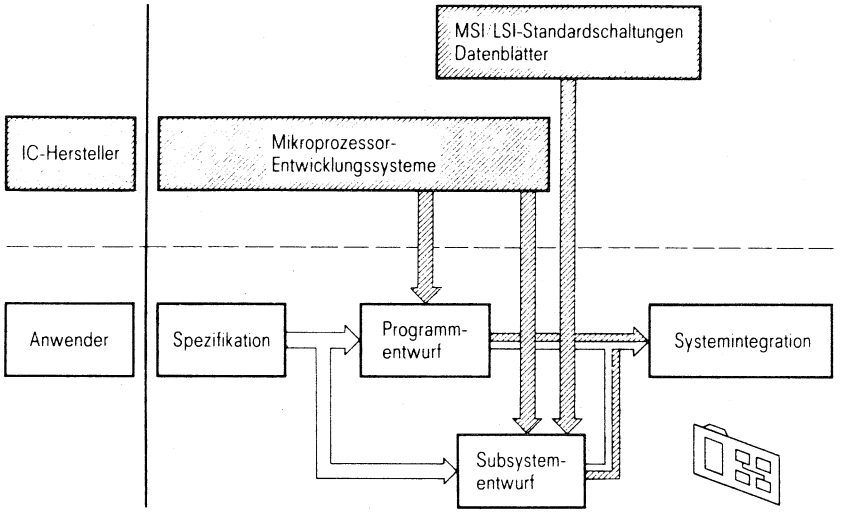


Bild 1.2. Problemlösung mit Mikroprozessor und MSI LSI-Bausteinen

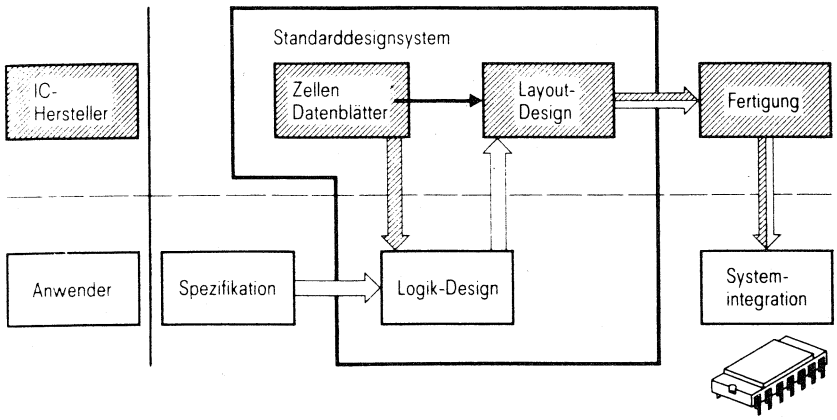


Bild 1.3. Problemlösung mit problemspezifischen VLSI-Bausteinen

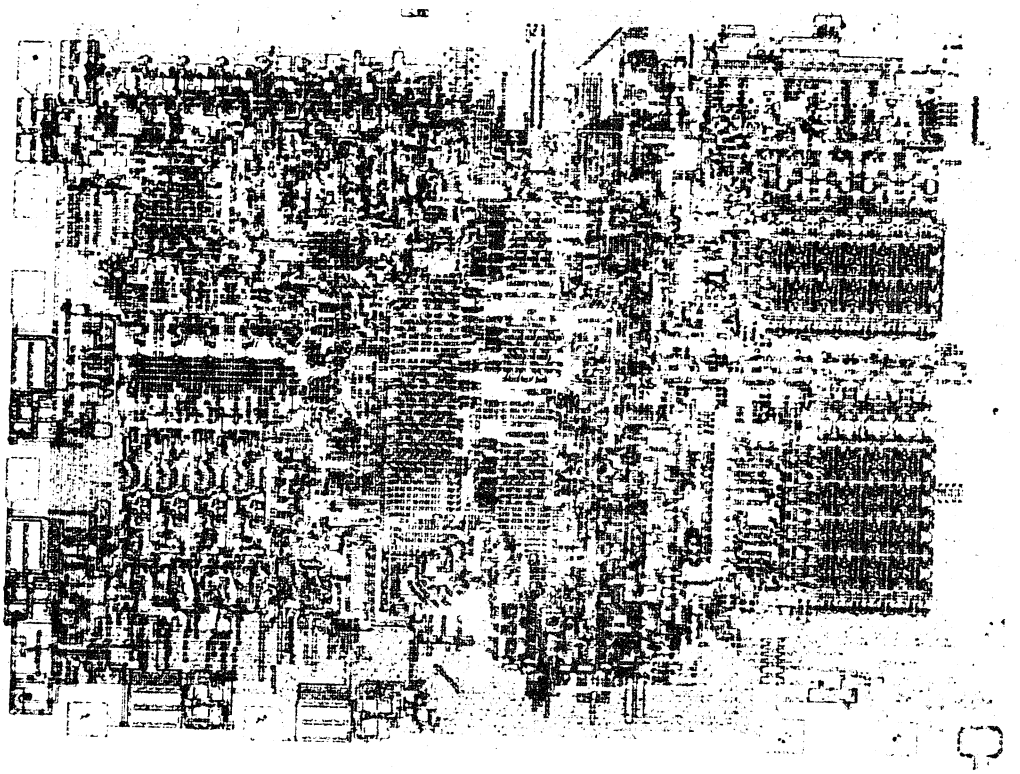
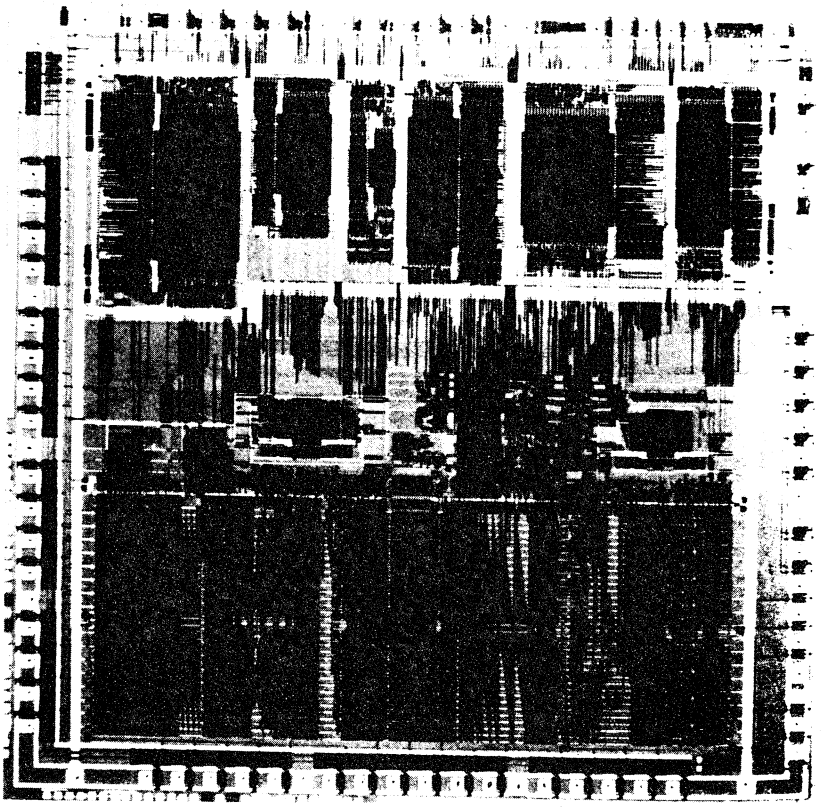


Fig. 3.2 Photomicrograph of a 4004 chip with pin designations. (Reprinted with permission of Intel Corporation.)

4 bit-CPU

1971, 2300 Transistoren



AT&T WE32000 32 bit-CPU

1981, 146000 Transistoren

Klassifizierung von ASICs

PAL

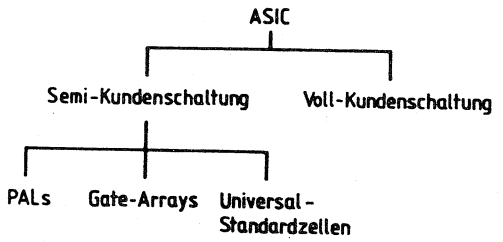
Grundsätzlicher Aufbau dieses Bausteins ist eine Matrixstruktur, wobei die Zeilen fest als „Oder-Funktion" verdrahtet sind, die Spalten als programmierbare „Und-Funktion" der Eingänge. Programmiert werden diese Bausteine entweder bei der Fertigung (maskenprogrammierbar) oder bei Kleinstserien und Prototypen mit einem Programmiergerät (feldprogrammierbar), das die Zeilen/Spalten-Verbindungen je nach zu realisierender Schaltfunktion unterbricht („Fusible Link") [3].

Gate-Arrays

Gate-Arrays sind standardmäßig gefertigte Chips, deren Einzelelemente (bestehend aus Dioden, Transistoren, Widerständen) nicht miteinander in Verbindung stehen, also nicht verdrahtet sind. Diese Einzelelemente sind matrixförmig in Reihen und Spalten angeordnet, wobei Zwischenräume für die Verdrahtung freigelassen werden. Die einzelnen Gatter sind umgeben von Peripheriezellen, über deren Anschlußflächen das Chip mit dem Gehäuse verbunden wird. Diese Peripheriezellen enthalten Treiberstufen, Pegelumsetzer usw. Die Herstellung der Verbindungen der einzelnen Elemente untereinander erfolgt nach Anwenderspezifikation entsprechend der Aufgabenstellung.

Voll-Kundenschaltungen

Voll-Kunden-ICs werden erst ab größeren Stückzahlen oder bei extremen Produktspezifikationen rentabel, da jedes Detail der Bausteine individuell entworfen wird. Das heißt also, daß alle Entwicklungsschritte auf das Endprodukt abgestimmt werden. Zur optimalen Nutzung der Chipfläche unter Beachtung aller Entwurfsregeln ist ein größeres Know-how des Entwicklers als bei Gate-Arrays oder Standardzellen notwendig. Dagegen besitzen die Voll-Kundenschaltungen aber als einzige auch das vollständige Potential aller Einflußmöglichkeiten des Entwicklers. Sie ermöglichen oft Produktrealisierungen, die ohne diese Technik unmöglich wären.



Standardzellen

Standardzellen haben eine vergleichbare bis etwas längere Entwicklungszeit als Gate-Arrays. Die ICs werden nicht vorgefertigt, sondern der Kunde wählt sich aus einer sogenannten Zellenbibliothek diejenigen Funktionen aus, die er zur Realisierung seines Problems benötigt. Der eigentliche Entwurf besteht aus der Plazierung und Verdrahtung dieser Zellen. Alle Maskenschritte werden kundenspezifisch ausgeführt. Man verwendet hierbei automatische Plazierungs- und Verdrahtungsprogramme, die die ausgewählten Zellen auf dem Chip möglichst optimal anordnen. Die Chipfläche wird also in der Regel besser ausgenutzt werden als bei den Gate-Arrays. Nachdem mit Hilfe der Daten aus der Zellenbibliothek eine Logiksimulation durchgeführt wurde, können die Masken der Schaltung erzeugt werden. Man unterscheidet zwischen Standardzellen- und Universalzellen-Chips. Standardzellen besitzen eine einheitliche Höhe und werden in Zeilen angeordnet. Zwischen den Zeilen befinden sich Verdrahtungskanäle. Universalzellen sind ebenfalls Zellen aus einer Baustein-Bibliothek, jedoch sind ihre Höhe und Breite beliebig. Hierdurch ist das Universalzellen-Konzept flexibler, jedoch ist die automatische Plazierung und Verdrahtung wesentlich aufwendiger, so daß z. Zt. flächenoptimale Entwürfe nur durch Handplazierung und -verdrahtung möglich sind.

Gate-Array:	40... 80 TDM
Standardzellen:	70...100 TDM
Universalzellen:	etwa 100 TDM
Voll-Kundenschaltung:	150...250 TDM

Kosten für ein 2000-Gatter-ASIC inklusive Entwurf, Prototypenfertigung und Test

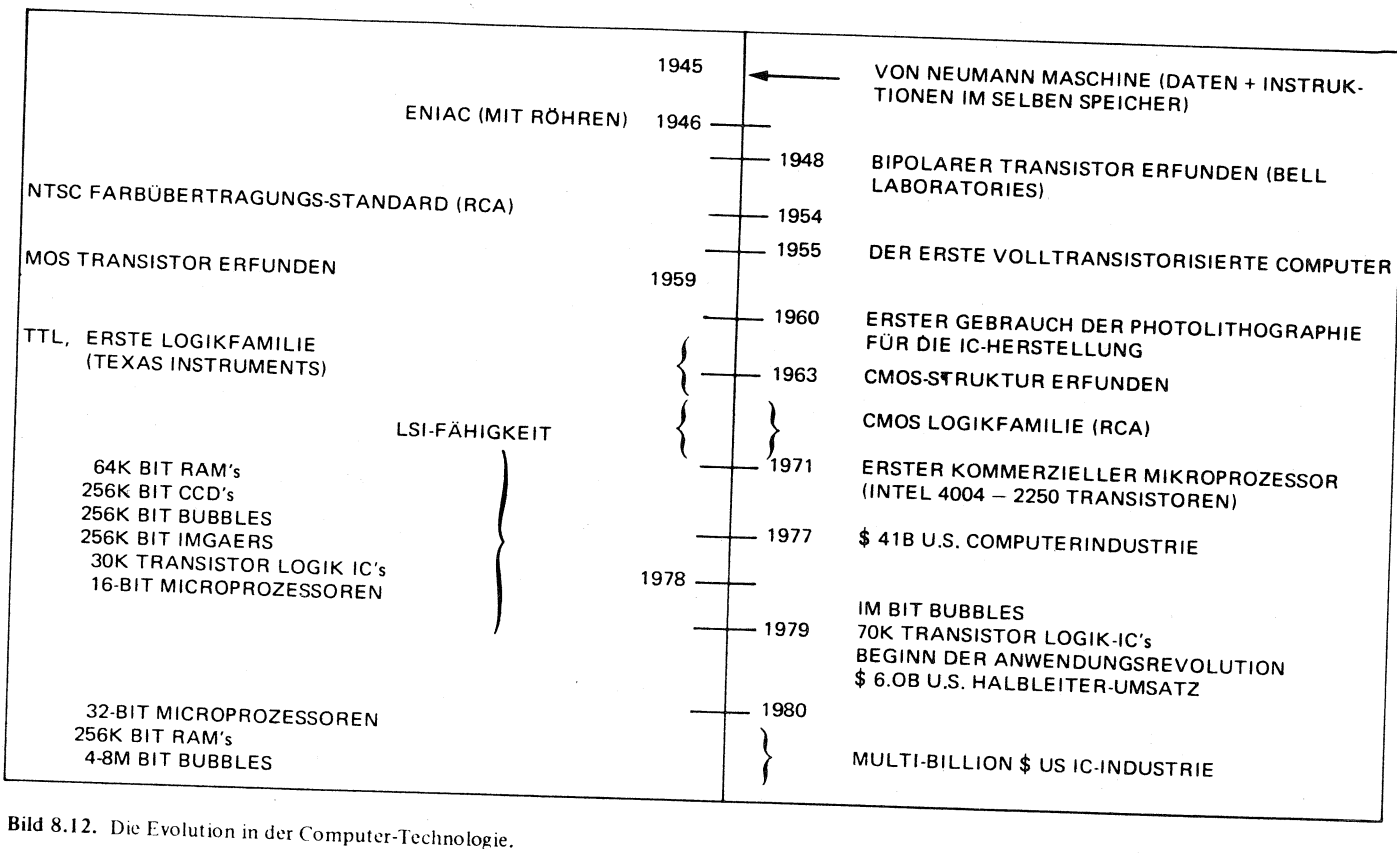


Bild 8.12. Die Evolution in der Computer-Technologie.

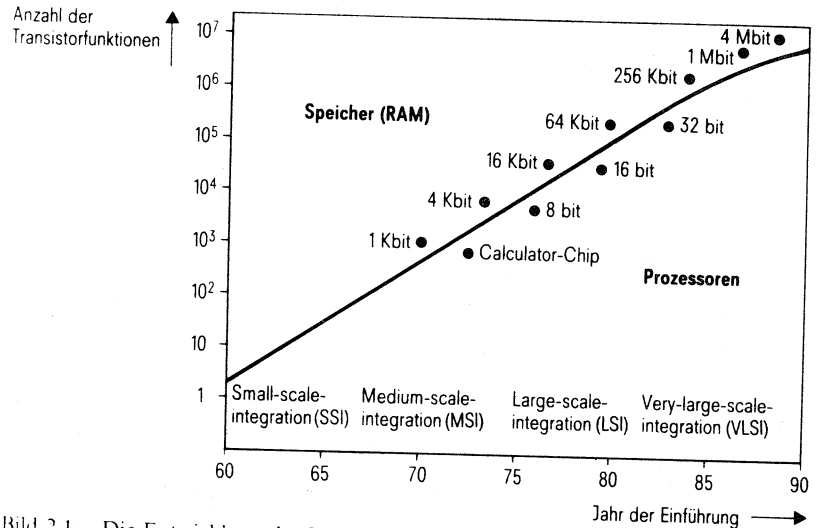


Bild 2.1. Die Entwicklung der Integrationsdichte pro Chip

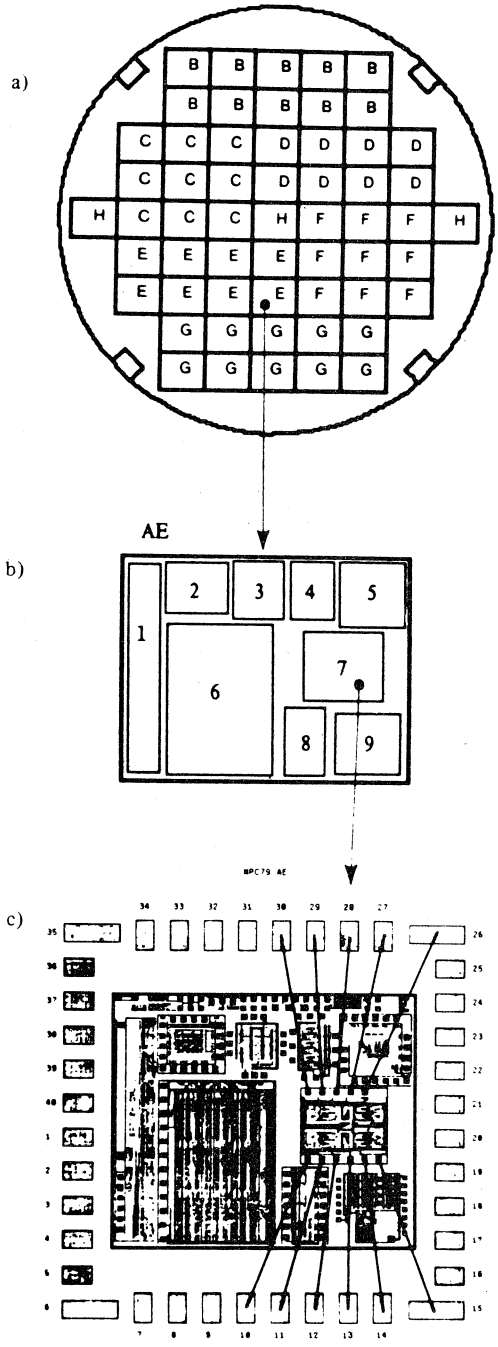


Bild 8.36. Veranschaulichung des „Multiproject-Chip"-Konzeptes. [8.15].
a) Die Verteilung verschiedener Chips auf dem Wafer.
b) Der Inhalt eines einzelnen Chips.
c) Die Kontaktierung nur eines einzigen Teilchips für einen Benutzer.

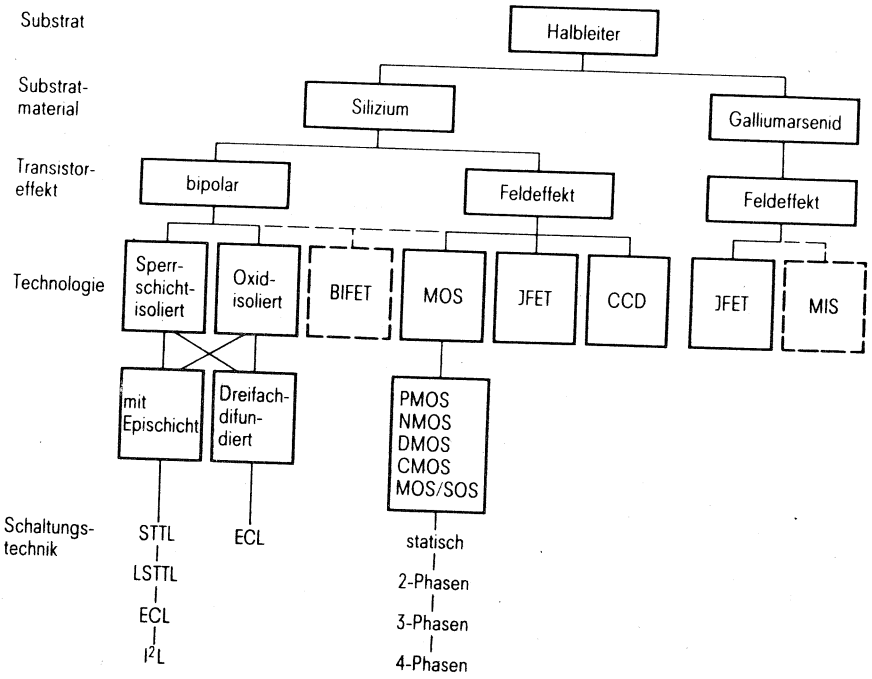


Bild 2.2. Übersicht über die wichtigsten Integrationstechniken [2.1]

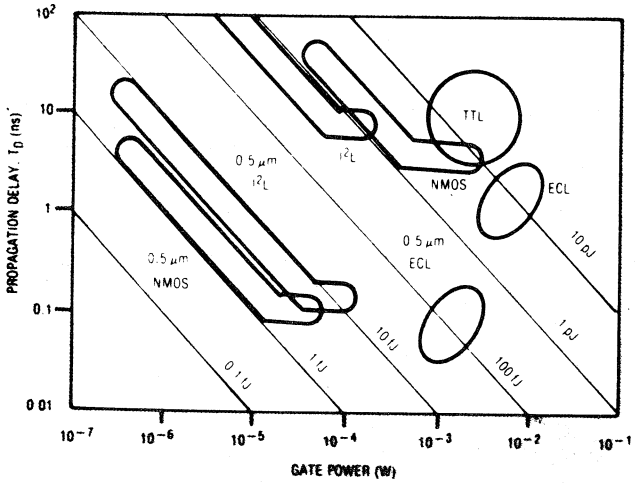


Bild 9.10. Schaltzeit und Verlustleistung verschiedener IC-Technologien. [8.18].

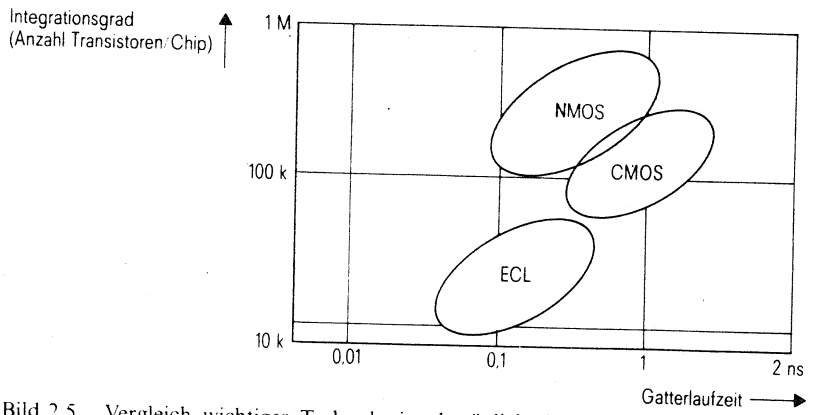


Bild 2.5. Vergleich wichtiger Technologien bezüglich Geschwindigkeit (Gatterlaufzeit) und Integrationsgrad (Stand 1985)

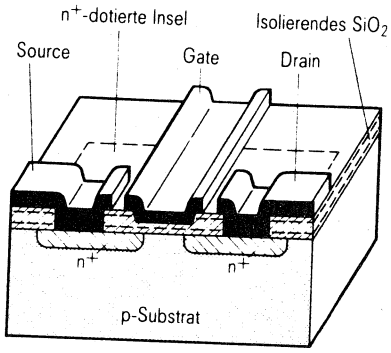


Bild 2.4. Schematische Darstellung eines MOS-Feldeffekttransistors

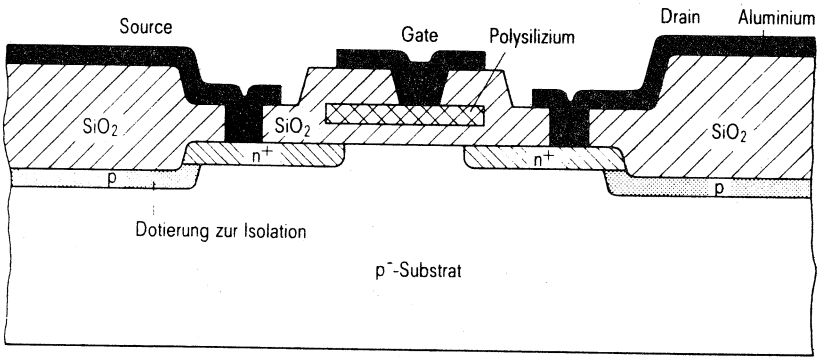


Bild 2.12. Querschnitt eines MOS-Feldeffekttransistors vom n-Typ (n-Kanal Transistor)

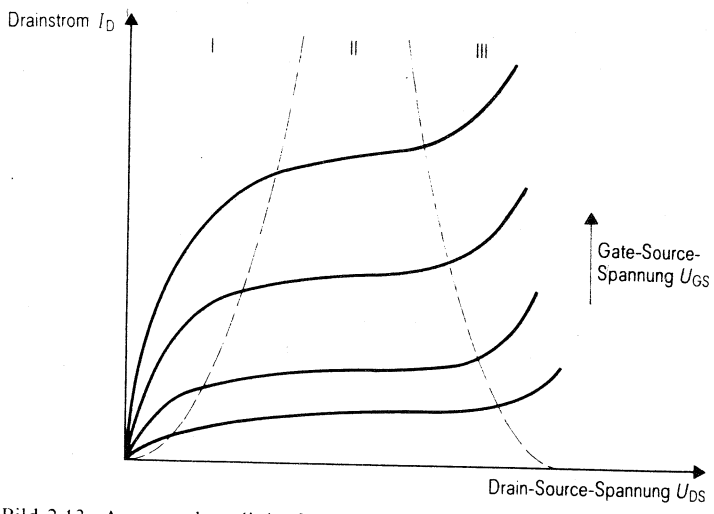
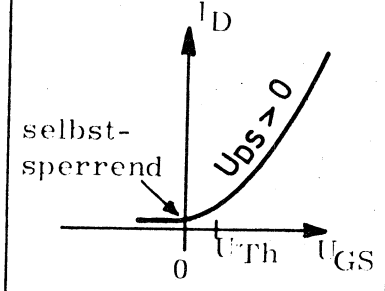
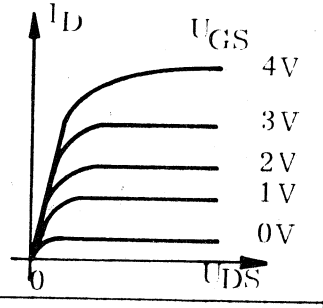
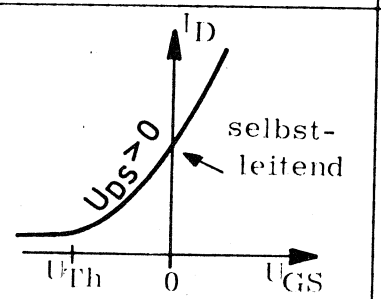
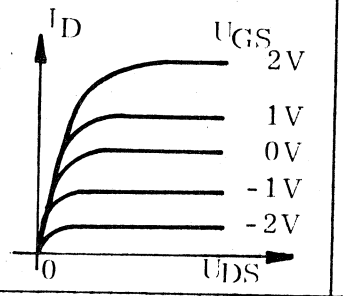
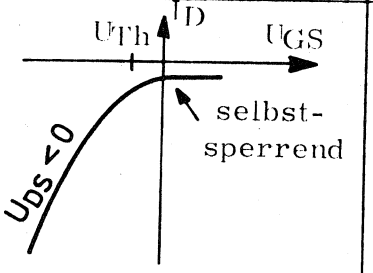
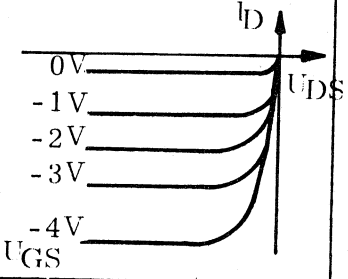
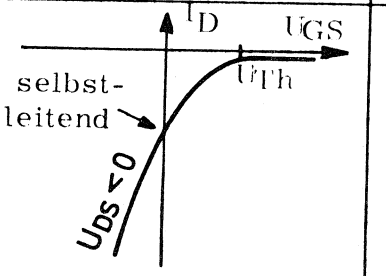
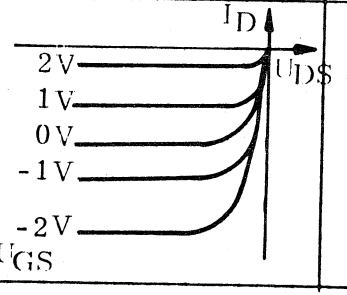


Bild 2.13. Ausgangskennlinienfeld eines n-Kanal MOS-Transistors

DIE VIER VERSCHIEDENEN MOS - FET - TYPEN

	Polarität	Typ	I_D-U_{GS}-Kennlinie	I_D-U_{DS}-Kennlinie	Bemerkungen
1	n - Kanal	Anreicherung			Entsteht nur bei höherer Substratdotierung, da sonst Inversion der Oberfläche in einen selbstleitenden n-Kanal erfolgt. Höhere Substratdotierung führt jedoch zu geringerer Steilheit. ("Atmen" der Raumladungszone erfordert größere Feldstärke.)
2		Verarmung			"Natürlicher" Typ, da p^--Substrat an der Oberfläche durch positive Oxydladungen einen selbstleitenden n-Kanal bildet (Inversion). Kann in Typ 1 umgewandelt werden durch negative Vorspannung am Substrat (Ausdehnung der Raumladungszone, Einschnürung des Kanals). Diese Vorspannung ist auch notwendig, um leitende Kanäle unter dickem Oxyd zu unterbinden. (Es hilft auch "Channel-Stopper"- p^+-Diffusion.)
3	p - Kanal	Anreicherung			"Natürlicher" Typ, da positive Oxydladungen die n-Natur der Oberfläche des n-Substrats eher verstärkt.
4		Verarmung			Entsteht durch Implantation von Akzeptoren (z.B. Bor) in die Kanalzone. Die so erzeugten Defektelektronen überkompensieren die durch die positiven Oxidladungen influenzierten Elektronen und bilden somit einen selbstleitenden Kanal.

- Schaltzeit: $\tau = \frac{L^2}{\mu V_{ds}}$ (μ ist Mobilität der Elektronen im Kanal (in cm^2/Vs))
- Gate-Kapazität: $C_g = \frac{\epsilon WL}{D}$ (ϵ = Dielektrizitätskonstante des Materials (in F/cm))
- Source-Drain-Strom: $I_{ds} = \frac{\mu\epsilon W}{LD}(V_{gs} - V_{th})V_{ds}$
- Drain-Source-Widerstand: $R_{ds} = \frac{L^2}{\mu C_g(V_{gs}-V_{th})}$

Wenn $V_{ds} > (V_{gs} - V_{th})$ kommt der Transistor in den Sättigungsbereich, es gilt dann: $I_{ds} = \frac{\mu\epsilon W}{2LD}(V_{gs} - V_{th})^2$

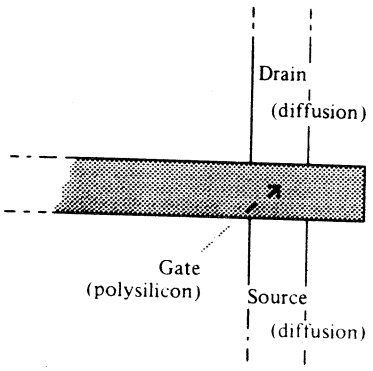


Fig. 1.1 MOS transistor, top view.

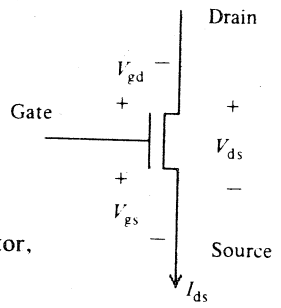


Fig. 1.2 MOS transistor symbol, subscripts in + to − direction sequence.

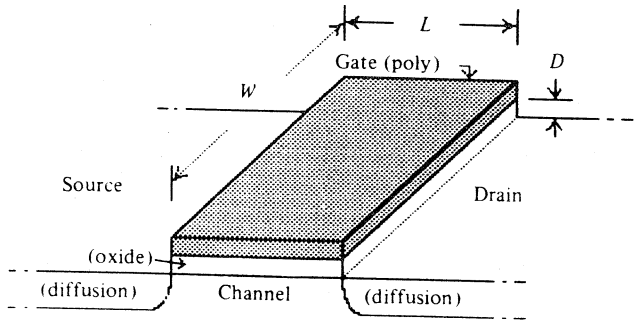


Fig. 1.3 MOSFET gate dimensions.

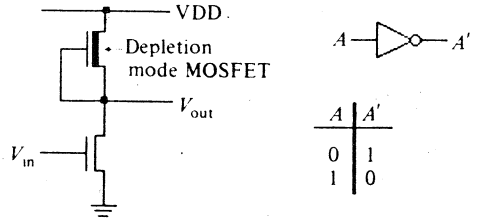


Fig. 1.7 The basic inverter circuit diagram, logic symbol, and logic function.

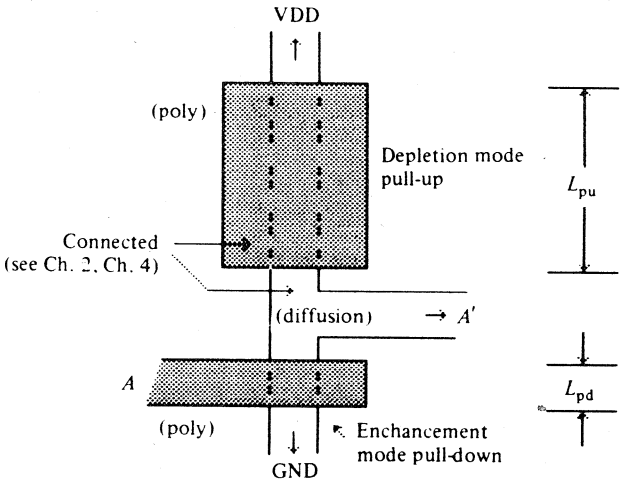


Fig. 1.8 Basic inverter layout.

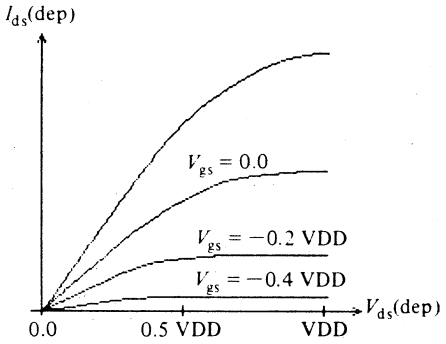


Fig. 1.9 Inverter pull-up characteristics.

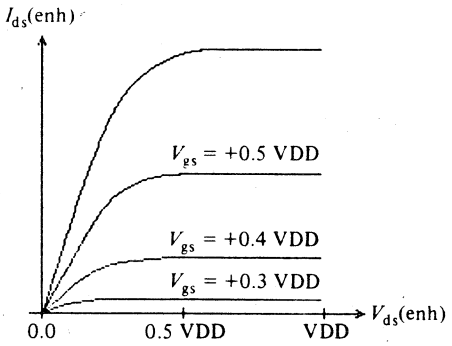
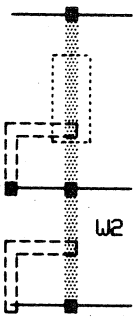
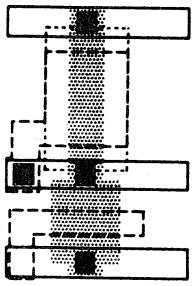
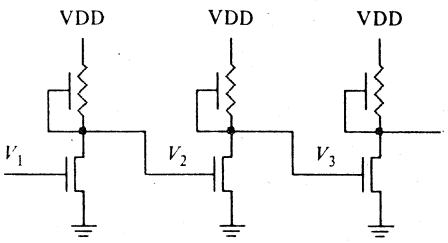


Fig. 1.10 Inverter pull-down characteristics.

Bild 4.7
Stickdiagramm und
Layout eines
Depletion-Last-
Inverters







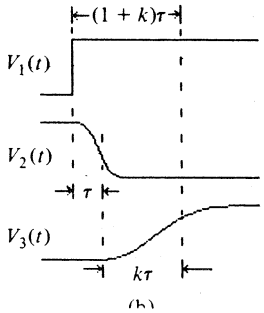


Fig. 1.13 Inverter delay.

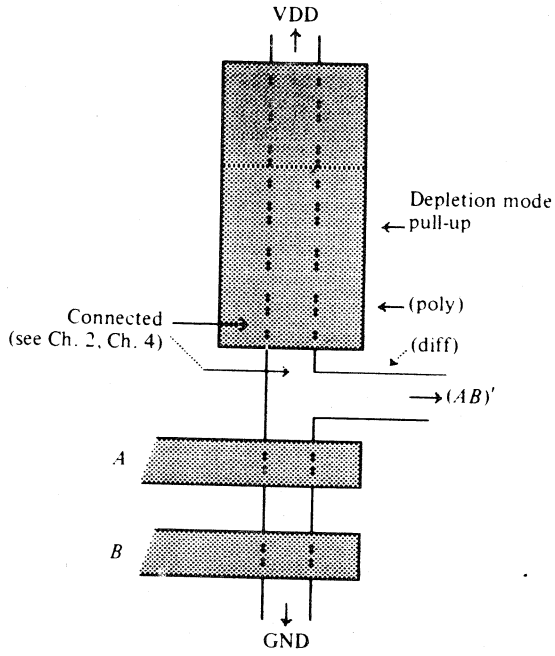


Fig. 1.17 NAND gate, top view of layout.

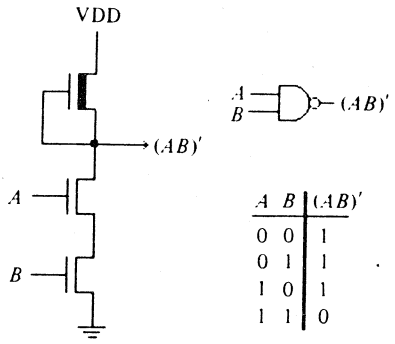


Fig. 1.18 NAND gate circuit diagram, logic symbol, and logic function.

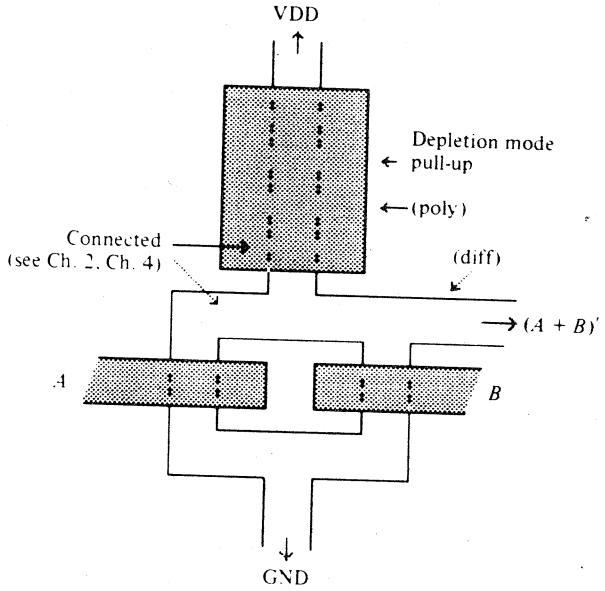


Fig. 1.19 NOR gate, top view of layout.

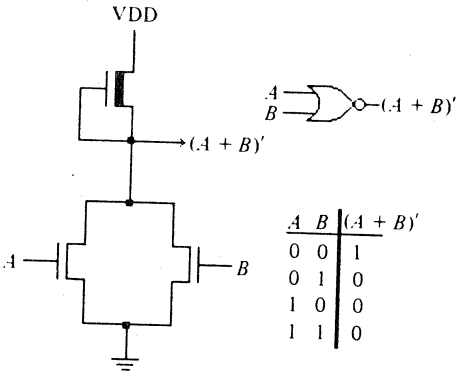


Fig. 1.20 NOR gate circuit diagram, logic symbol, and logic function.

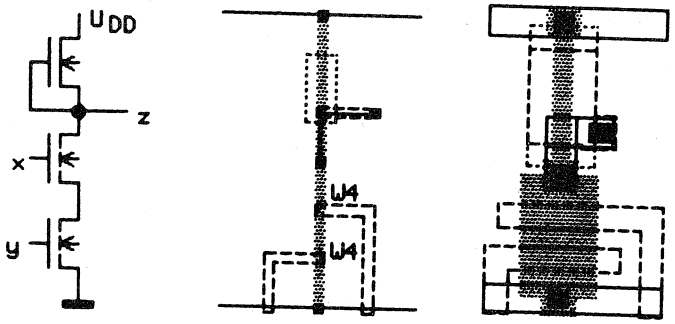


Bild 4.8 Schaltung, Stickdiagramm und Layout eines NAND-Gatters in NMOS-Depletion-Technik

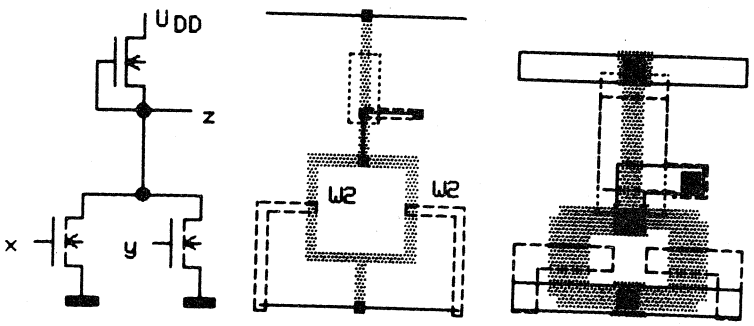


Bild 4.9 Schaltung, Stickdiagramm und Layout eines NOR-Gatters in NMOS-

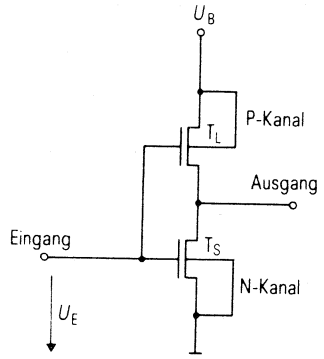


Bild 2.25. Schaltbild eines MOS-Inverters in Komplementärkanaltechnik

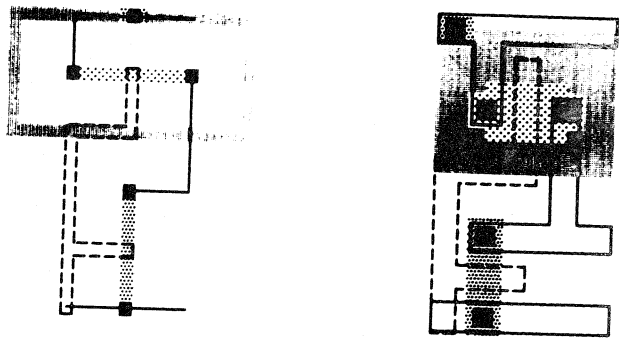


Bild 4.11 Stickdiagramm und Layout eines CMOS-Inverters

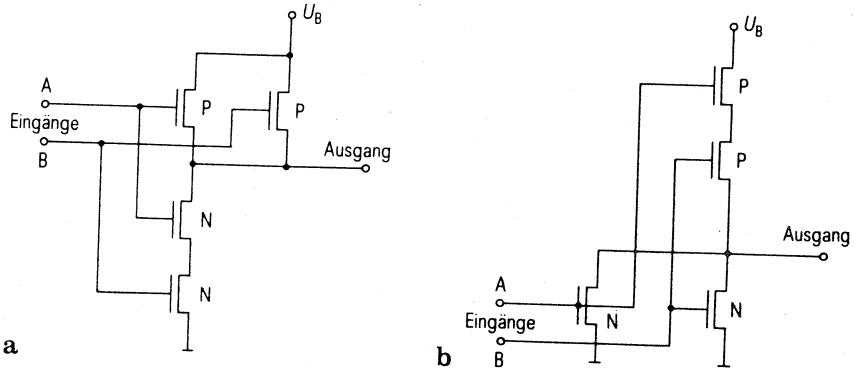


Bild 2.29. Schaltbild eines NAND-Gatters (a) und eines NOR-Gatters (b) in Komplementärkanaltechnik (CMOS)

Bild 4.12
NAND-Gatter in CMOS-Technik:
Schaltung, Layout
und Stickdiagramm

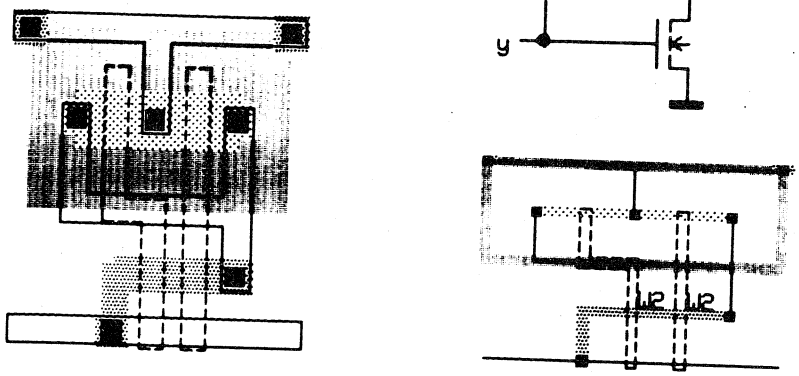


Bild 4.13
NOR-Gatter in CMOS-Technik:
Schaltung, Stickdiagramm
und Layout

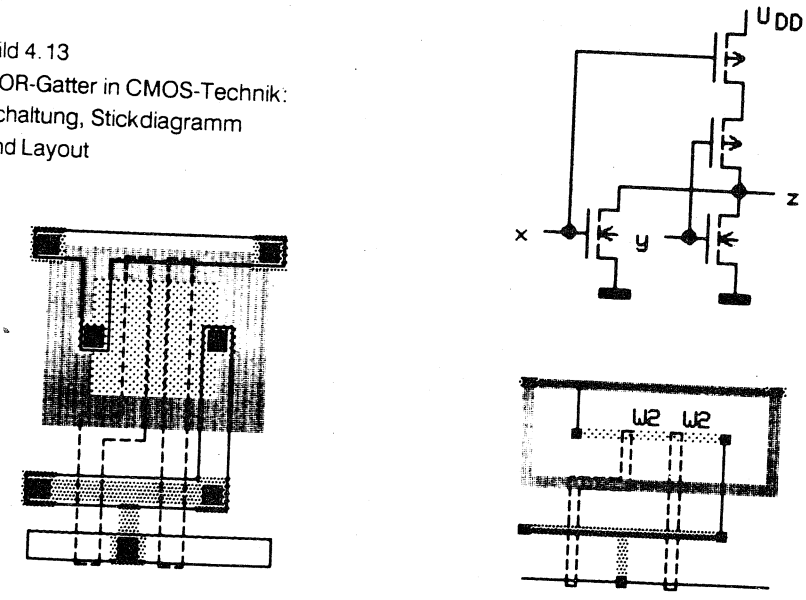
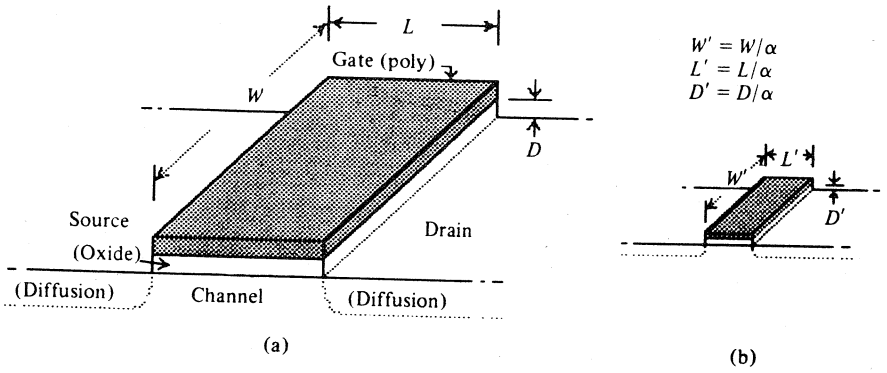


TABLE 1.6. CMOS-nMOS quick summary

CMOS	nMOS
(i) Logic Levels	
• Fully restored logic, i.e., output settles at V_{DD} or V_{SS} (GND).	• Output does not settle at V_{SS} (GND) — hence degraded noise margin.*
(ii) Transition Times	
• Rise and fall times are of the same order.	• Rise times are inherently slower than Fall times.*
(iii) Transmission Gates	
• Transmission gate passes both logic levels well. The output of transmission gate can be used to drive the input of other transmission gates.	• Pass transistor transfers logic '0' well but logic '1' is degraded. Pass transistor cannot drive the gate of a second pass transistor.
(iv) Power Dissipation	
• Almost zero static power dissipation. However power is dissipated during logic transition.	• With output of a given gate = '0' power is dissipated in the circuit in addition to power dissipated during logic transitions.*
(v) Precharging Characteristics	
• Both n-type and p-type devices are available for precharging a bus to V_{DD} and V_{SS}. Nodes can be charged fully to V_{DD} or alternatively to V_{SS} in a short time.	• With enhancement mode transistor the best one can do (with normal clocking) is to charge a bus to $(V_{DD} - V_t)$. Generally use of bootstrapping or hot clocking is needed to precharge to V_{DD}.
(vi) Power Supply	
• Voltage required to switch a gate is a fixed percentage of V_{DD}.	• Somewhat dependent on supply voltage.
• Variable range 1.5 to 15 volts.	• Fixed.
(vii) Packing Density	
• Require 2N devices for N inputs for complementary static gates. Less for dynamic gates.	• Require (N + 1) devices for N inputs.
(viii) Pull-up to Pull-down Ratio	
• Load-to-driver device ratio is typically 1:1 or 2:1.	• Load-to-enhancement-driver ratio is typically 4:1 to optimize the logic '0' output level and minimize current consumption.
(ix) Layout	
• CMOS encourages regular layout styles.	• Depletion load and different driver transistor sizes inhibit layout regularity.



Charakteristische Größen	1985	1990	1995
Minimale Strukturbreite	1 µm	0,5 µm	0,3 µm
Durchschnittliche Chipfläche	50 mm²	100 mm²	200 mm²
Verdrahtungs-Ebenen (*) (Speicher)	3	5	7
Trans./Chip (Speicher)	$5 \cdot 10^6$	10^7	$10^8 \ldots 10^9$

(*) Metall und Poly-Silizium

Bild 3. Gegenwärtiger Stand und Entwicklungsziele der IC-Technologie

Im Jahre 1978 haben Folberth und Bleher [9.3] eine sehr abgewogene Übersicht über die Grenzen der digitalen Halbleitertechnik veröffentlicht. Sie interessierten sich hierbei im wesentlichen für diejenigen Grenzwerte, die wirklich realisierbar erscheinen. Einige interessante Werte sind z.B.

- Maximale Integrationsdichte für Gatter

  $2{,}5 \cdot 10^7$ Gatter/cm²

- Maximale Integrationsdichte für Speicher

  $2{,}5 \cdot 10^7$ Bits/cm²

- Minimale Fläche einer Speicherzelle

  2 µm x 2 µm

- Minimale Zugriffszeit für Speicher

  100 ps

- Minimale Laufzeit durch einen Bipolartransistor

  1,2 ps

- Minimale Schaltzeit eines belasteten Bipolartransistors

  2,3 ps

- Produkt von Schaltzeit und Verlustleistung (CMOS mit L = 1 µm und V_S = 1V)

  $5 \cdot 10^{-16}$ Ws

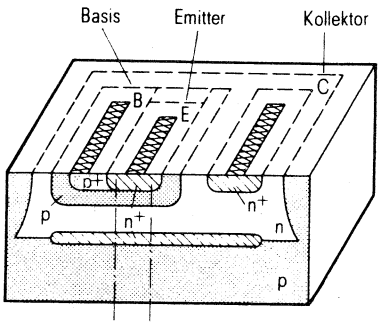


Bild 2.3. Schematische Darstellung eines bipolaren npn-Transistors

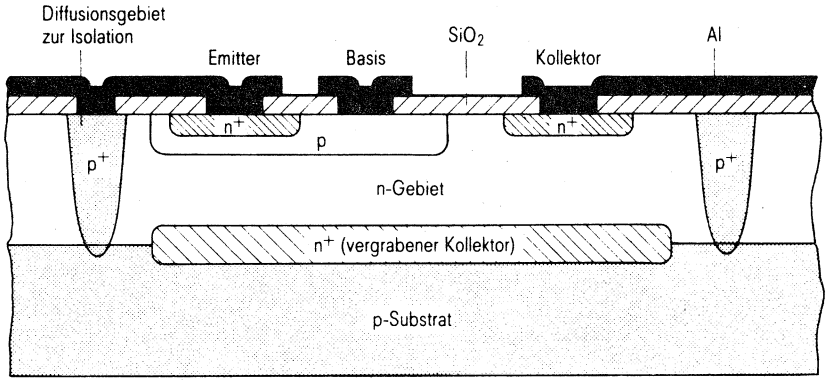


Bild 2.8. Querschnitt eines Bipolartransistors vom npn-Typ

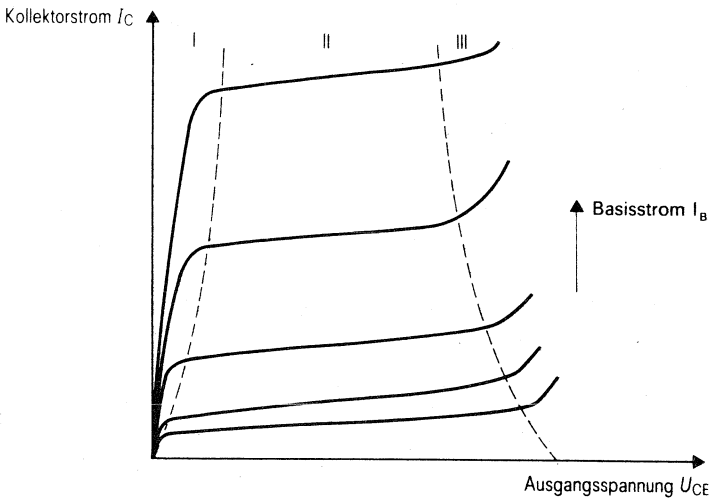


Bild 2.9. Ausgangskennlinienfeld eines Bipolartransistors vom npn-Typ

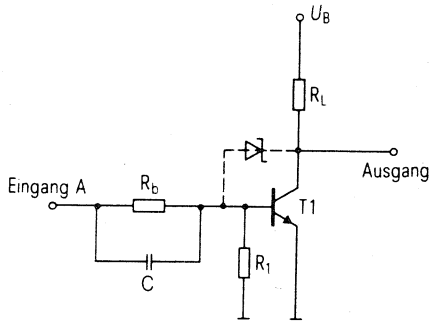


Bild 2.15. Schaltbild eines Inverters mit einem Bipolartransistor und einem ohmschen Widerstand als Lastelement

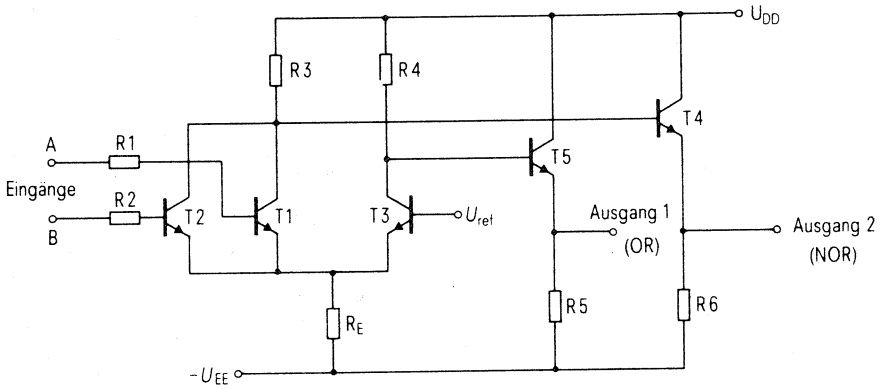


Bild 2.19. Schaltbild eines OR/NOR-Gatters in der ECL-Schaltungstechnik

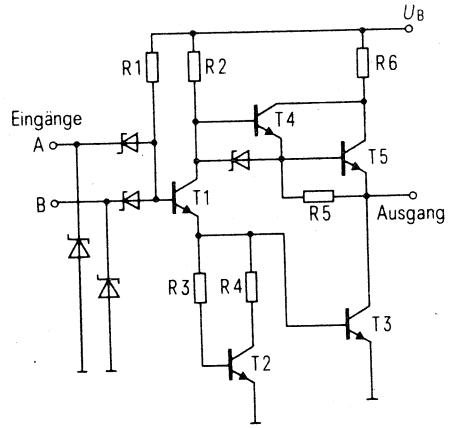


Bild 2.18. Schaltbild eines NAND-Gatters in Low-Power Schottky-TTL-Schaltungstechnik

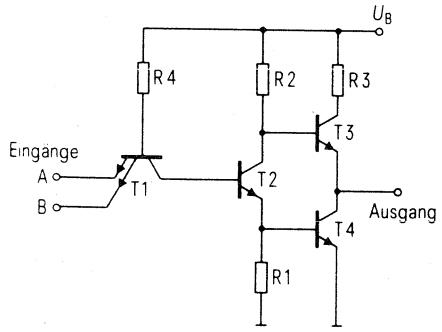


Bild 2.17. Schaltbild eines NAND-Gatters in TTL Schaltungstechnik

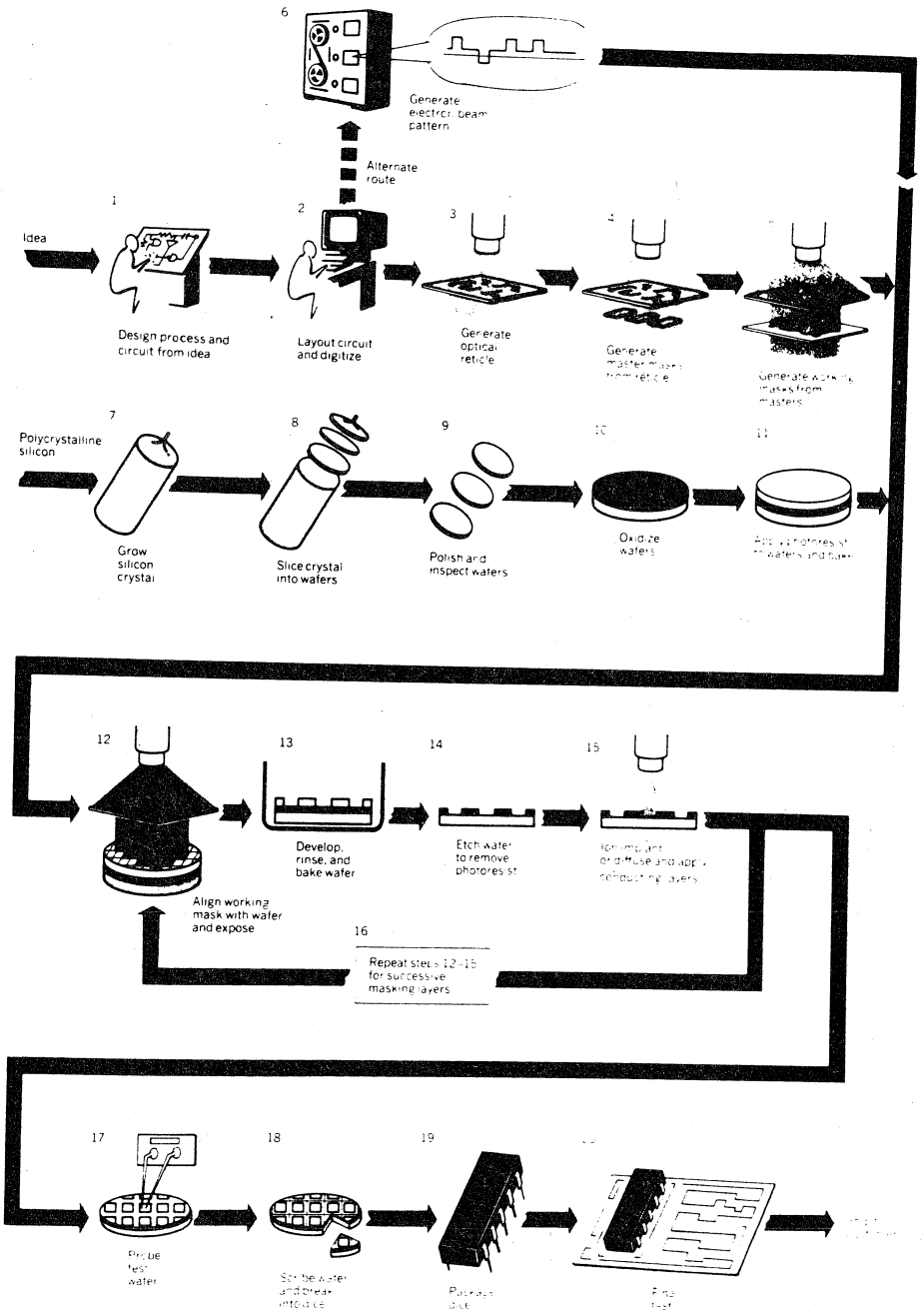


Bild 1.28a. Anschauliche Darstellung der wichtigsten Prozeßschritte zur Herstellung integrierter Schaltungen. [1.5].

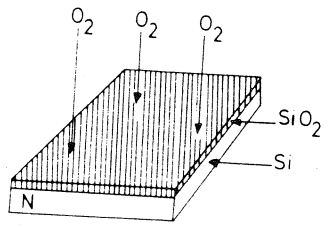


Bild 1.2. Oxydation des Wafers.

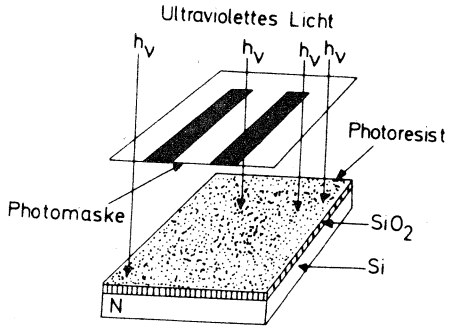


Bild 1.3. Exposition der ersten Maske. (Source und Drain)

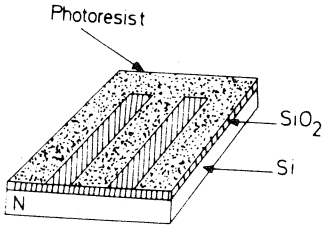


Bild 1.4. Photoresist entwickelt und bereit zum Ätzen.

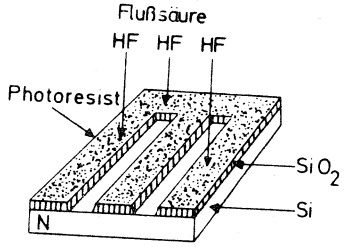


Bild 1.5. Ätzen, um SiO_2 zu entfernen.

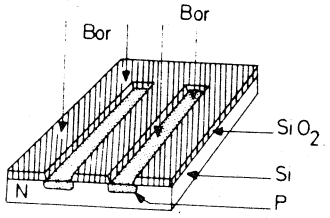


Bild 1.6. P-Diffusion.

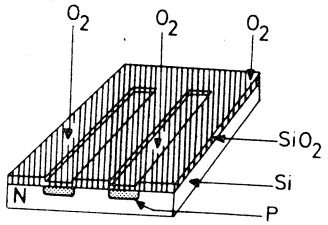


Bild 1.7. Erneute Oxydation, um dickes Feldoxyd zu bilden.

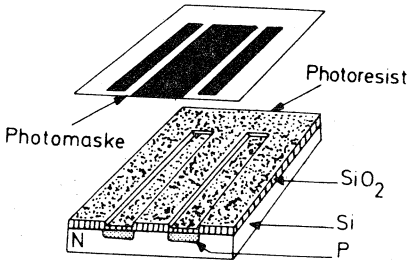


Bild 1.8. Zweite Maskierung (Gate und Kontakte).

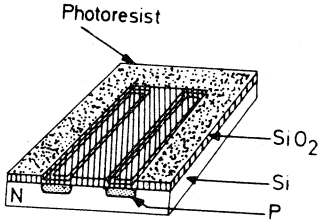


Bild 1.9. Photoresist entwickelt und fertig zum Ätzen.

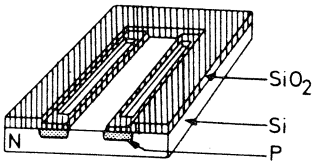


Bild 1.10. Nach der Ätzung des Gates.

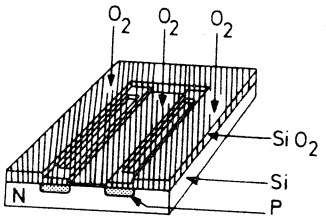


Bild 1.11. Nach der Gate-Oxydation.

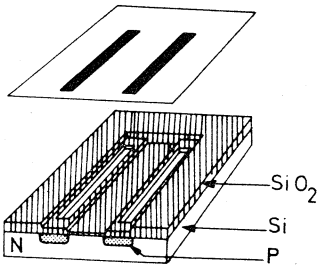


Bild 1.12. Dritte Maske (Kontaktöffnungen).

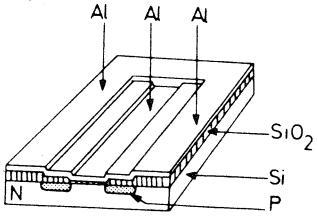


Bild 1.13. Aufbringen von Aluminium.

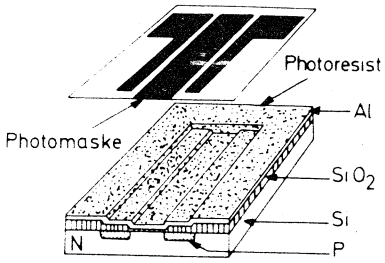


Bild 1.14. Vierte Maske (Kontakte und Verbindungen).

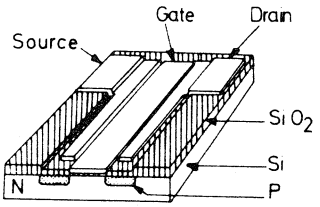


Bild 1.15. Vollständiges Bauelement.

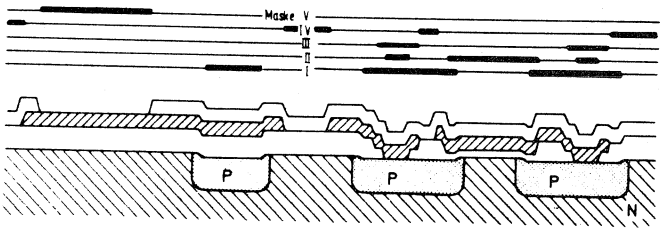


Bild 1.16. Zuordnung der Masken zu dem Oberflächenprofil.

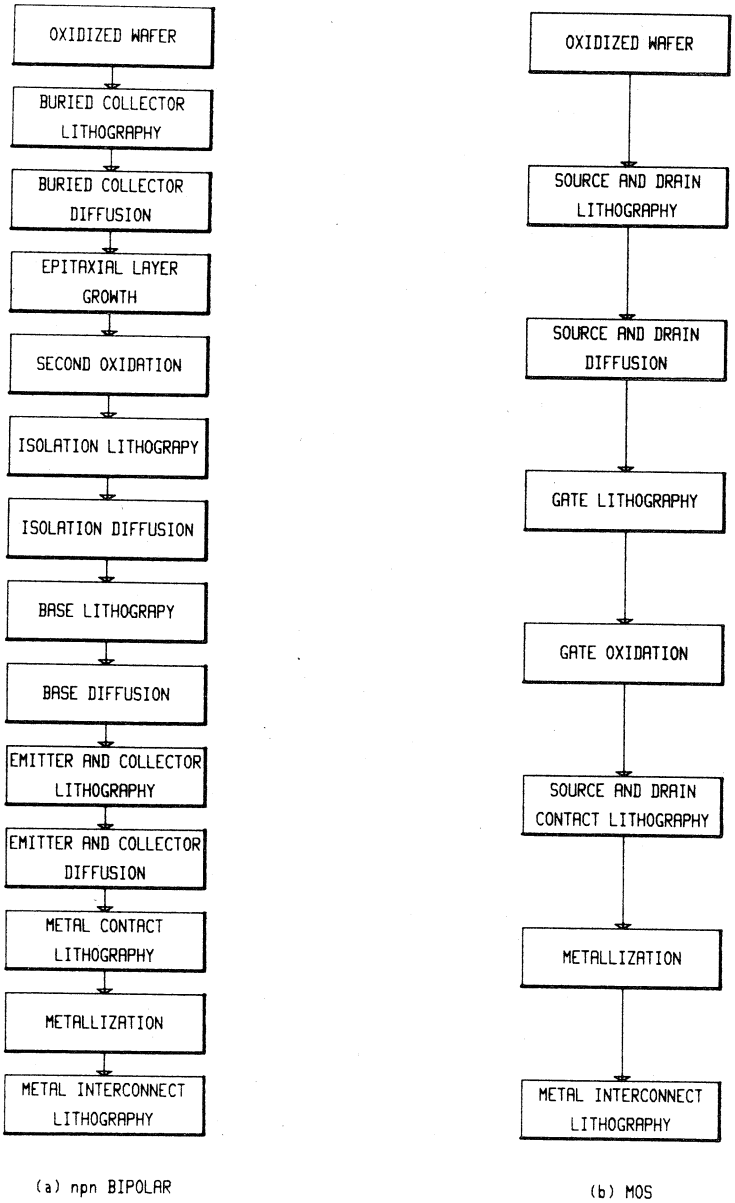


Fig 2.5 Fabrication processing for (a) bipolar (b) MOS transistors

Die Prozeßfolge für einen Silizium-Gate-Prozeß ist in Bild 2.32 dargestellt.

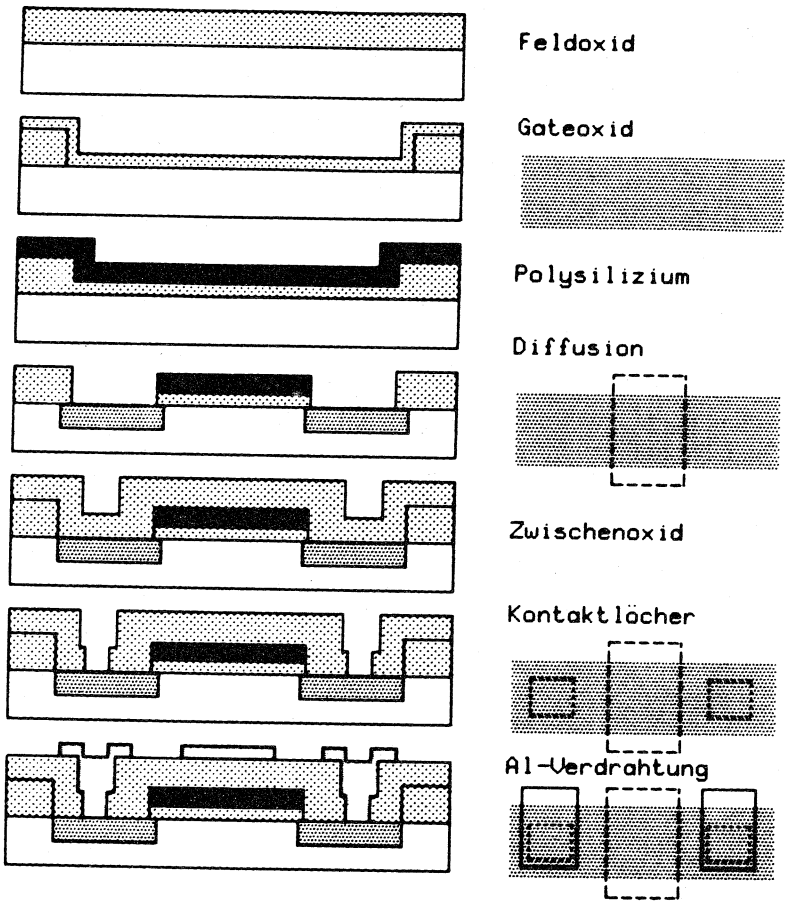


Bild 2.32 Prozeßfolge einer Si-Gate n-Kanal-Technologie

Für den 5 Volt-Prozeß werden die Schwellspannungen auf U_{TE} = 0,8 V und U_{TD} = - 2,5 V eingestellt. Mit einer minimalen Gate-Länge L = 2.5 μm sind Taktfrequenzen im Bereich von 20 MHz möglich. Je nach Prozeßvariante sind 8 bis 12 Masken erforderlich.

- SUBSTRATE <NAME> (*TYPE=[P,N] IMPURITY=[])
  Specifies the substrate name, type, and impurity level.
- OXIDE <NAME> THICKNESS = []
  Specifies oxide layer and thickness.
- DEPOSITION <NAME> (*) THICKNESS=[]
  Specifies a layer and thickness of a deposited layer. (*) is followed by TYPE=[] IMPURITY=[] if it is silicon.
- ETCH <NAME> DEPTH=[]
  Specifies a material and an etch depth.
- DOPE TYPE=[P,N] PEAK=[] DEPTH=[] DELTA=[] BLOCK=[]
  Specifies parameters necessary to define a diffusion step.
- MASK <RESIST NAME> <EXPOSED NAME> <MASK NAME> <POLARITY OF MASK>
  Specifies a resist layer and associated information.

The complete process input file is as follows (with abbreviations):


```
1. LEVEL 1
2. SUBS SILICON TYPE=P IMPU=1e13
```

Initial oxidation

```
3. OXIDE OX1 THICK=0.1
```

n-well definition

```
 4. DEPO NTRD THICK=0.5
 5. DEPO RST THICK=0.5
 6. MASK RST DRST MNWL POSI
 7. ETCH DRST DEPTH=0.6
 8. ETCH NTRD DEPTH=0.6
 9. ETCH RST DEPTH=0.6
10. OXIDE OX2 THICK=0.5
11. ETCH NTRD DEPTH=0.6
12. DOPE TYPE=N PEAK=1.5e15 DEPTH=0.0 DELTA=1.5
    BLOCK=0.2
13. ETCH OX DEPTH=0.7
14. OXIDE OX3 THICK=0.1
```

All active area definition

```
15. DEPO NTRD THICK=0.5
16. DEPO RST THICK=0.5
17. MASK RST DRST MAA POSI
18. ETCH DRST DEPTH=0.6
19. ETCH NTRD DEPTH=0.6
20. ETCH RST DEPTH=0.6
```

Field dope for n-channel

```
21. DEPO RST THICK=1.0
22. MASK RST DRST MNWL POSI
23. ETCH DRST DEPTH=1.1
24. DOPE TYPE=P PEAK=1e21 DEPTH=0.05 DELTA=0.15
    BLOCK=0.2
25. ETCH RST DEPTH=1.1
26. OXIDE OX4 THICK=0.7
27. ETCH NTRD DEPTH=0.6
```

Threshold adjust dope

```
28. DOPE TYPE=P PEAK=1e20 DEPTH=0.0 DELTA=0.05
    BLOCK=0.2
```

Regrow gate oxide

```
29. ETCH OX DEPTH=0.1
30. OXIDE OX5 THICK=0.1
```

Poly gate definition

```
31. DEPO POLY THICK=0.30
32. DEPO RST THICK=0.5
33. MASK RST DRST MSI POSI
35. ETCH DRST DEPTH=0.6
35. ETCH POLY DEPTH=0.6
36. ETCH RST DEPTH=0.6
```

Arsenic dope for n-channel source and drain

```
37. DEPO RST THICK=1.0
38. MASK RST DRST MIIN POSI
39. ETCH DRST DEPTH=1.1
40. DOPE TYPE=N PEAK=1e22 DEPTH=0.0 DELTA=0.2
    BLOCK=0.2
41. ETCH RST DEPTH=1.1
```

Boron dope for p-channel source and drain

```
42. DEPO RST THICK=1.0
43. MASK RST DRST MIIN NEGA
44. ETCH DRST DEPTH=1.1
45. DOPE TYPE=P PEAK=1e22 DEPTH=0.0 DELTA=0.2
    BLOCK=0.2
46. ETCH RST DEPTH=1.1
```

LPCVD oxide (**L**iquid **P**hase **C**hemical **V**apor **D**eposition Oxide)

```
47. DEPO OX6 THICK=0.5
```

Contact definition

```
48. DEPO RST THICK=1.0
49. MASK RST DRST MCC NEGA
50. ETCH DRST DEPTH=1.1
51. ETCH OX DEPTH=1.1
52. ETCH RST DEPTH=1.1
```

Metallization

```
53. DEPO METL THICK=1.0
54. DEPO RST THICK=1.0
55. MASK RST DRST MME POSI
56. ETCH DRST DEPTH=1.1
57. ETCH METL DEPTH=1.1
58. ETCH RST DEPTH=1.1
```

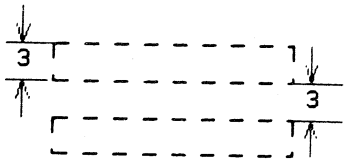
Some of the abbreviations are as follows:

```
NTRD - Nitride
RST - Resist
METL - Metal (Aluminum)
NEGA - Negative
POSI - Positive
MNWL - N-well mask
MAA - Thin-oxide mask
MSI - Polysilicon mask
MIIN - Nplus mask
MCC - Contact mask
MME - Metal mask
```

Polysilizium NP

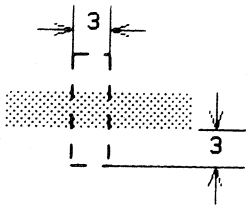
Breite

Abstand

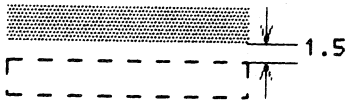


Transistorlänge

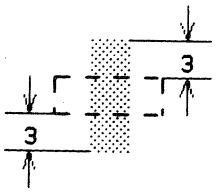
Überlappung Gate



Abstand



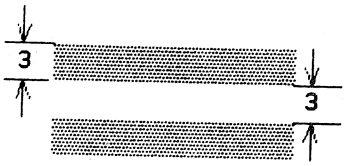
Überlappung von
Source und Drain



n+-Gebiet ND

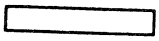
Breite

Abstand



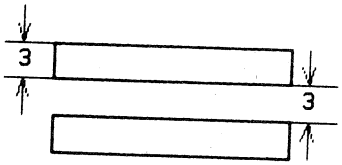
Transistorbreite

Metall NM



Breite

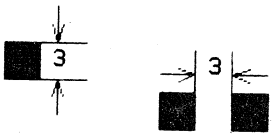
Abstand



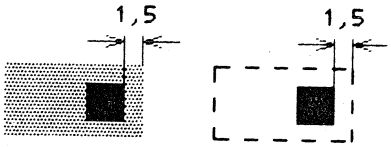
Kontaktloch NC

Breite, Länge

Abstand

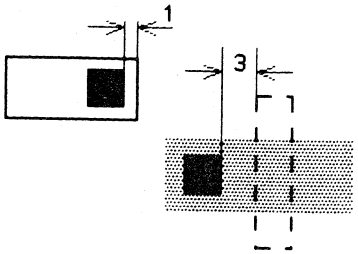


Überlappung
ND,PD,NP

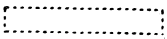


Überlappung NM

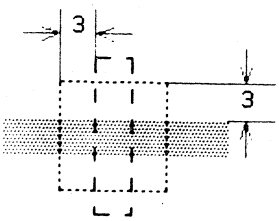
Abstand Gate



Depletion NI

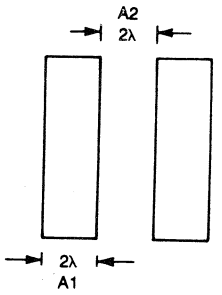


Überlappung



NMOS design rules

FIGURE 3.22. Lambda-based rules for a p-well process

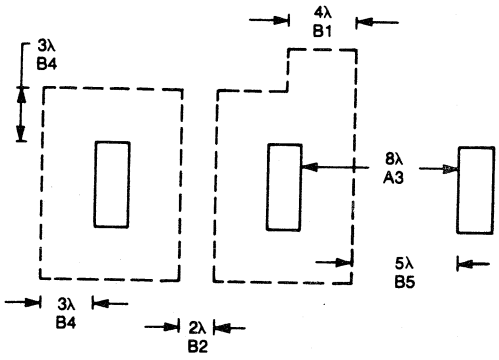


MASK 1: THINOX

A1. MINIMUM THINOX WIDTH 2λ

A2. THINOX SPACING 2λ
(n^+ to n^+ or p^+ to p^+)

A3. p^+ to n^+ SPACING 8λ



(6λ FOR WELLS AT DIFFERENT POTENTIALS)

MASK 2: p-WELL

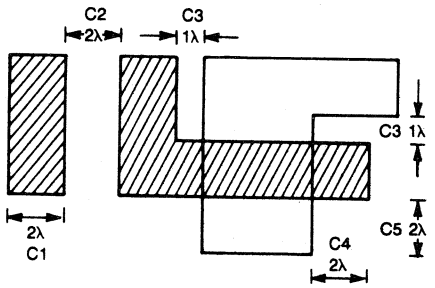
B1. MINIMUM p-WELL WIDTH 4λ

B2. MINIMUM p-WELL SPACING 2λ
(SAME POTENTIAL)

B3. MINIMUM p-WELL SPACING 6λ
(DIFFERENT POTENTIAL)

B4. MINIMUM OVERLAP OF INTERNAL THINOX 3λ

B5. MINIMUM SPACING TO EXTERNAL THINOX 5λ



MASK 3: POLYSILICON

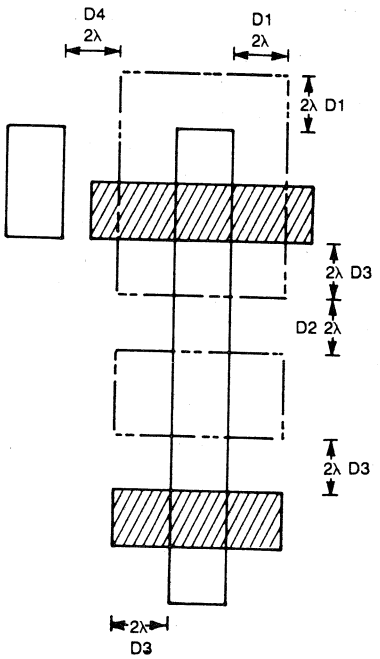
C1. MINIMUM POLY WIDTH 2λ

C2. MINIMUM POLY SPACING 2λ

C3. MINIMUM POLY-THINOX SPACING λ

C4. MINIMUM POLY GATE EXTENSION 2λ

C5. MINIMUM THINOX SOURCE/DRAIN EXTENSION 2λ



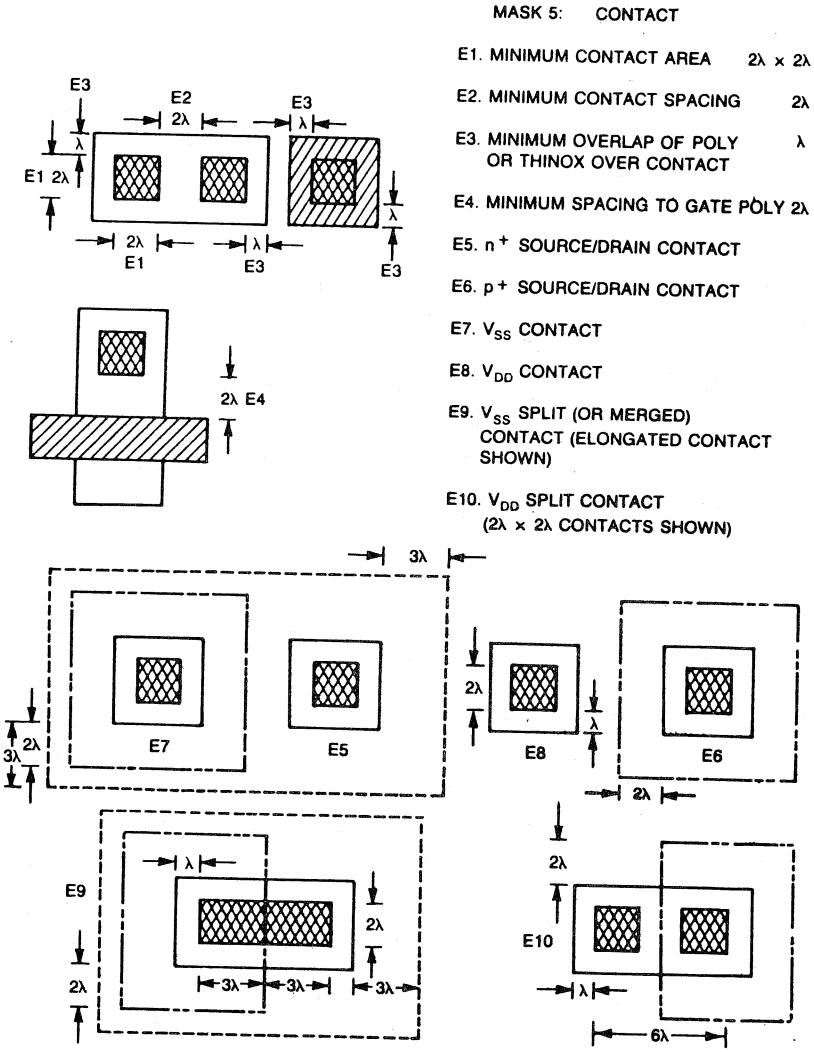
MASK 4: p-PLUS

D1. MINIMUM OVERLAP OF THINOX 2λ

D2. MINIMUM p-PLUS SPACING 2λ

D3. MINIMUM GATE OVERLAP OR DISTANCE TO GATE EDGE 2λ

D4. MINIMUM SPACING TO UNRELATED THINOX 2λ



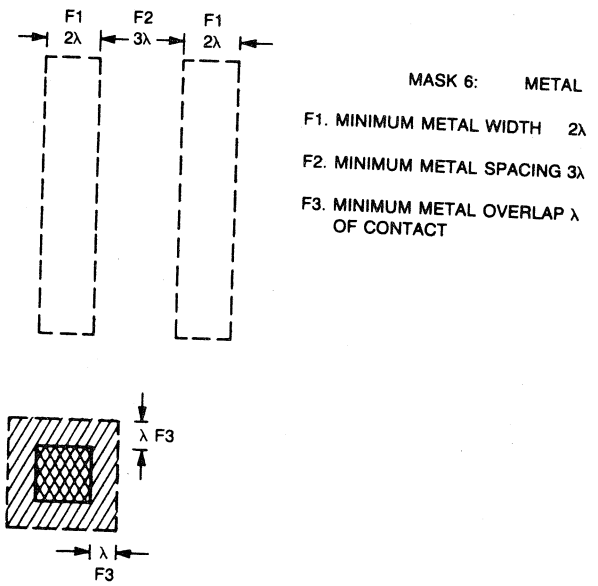
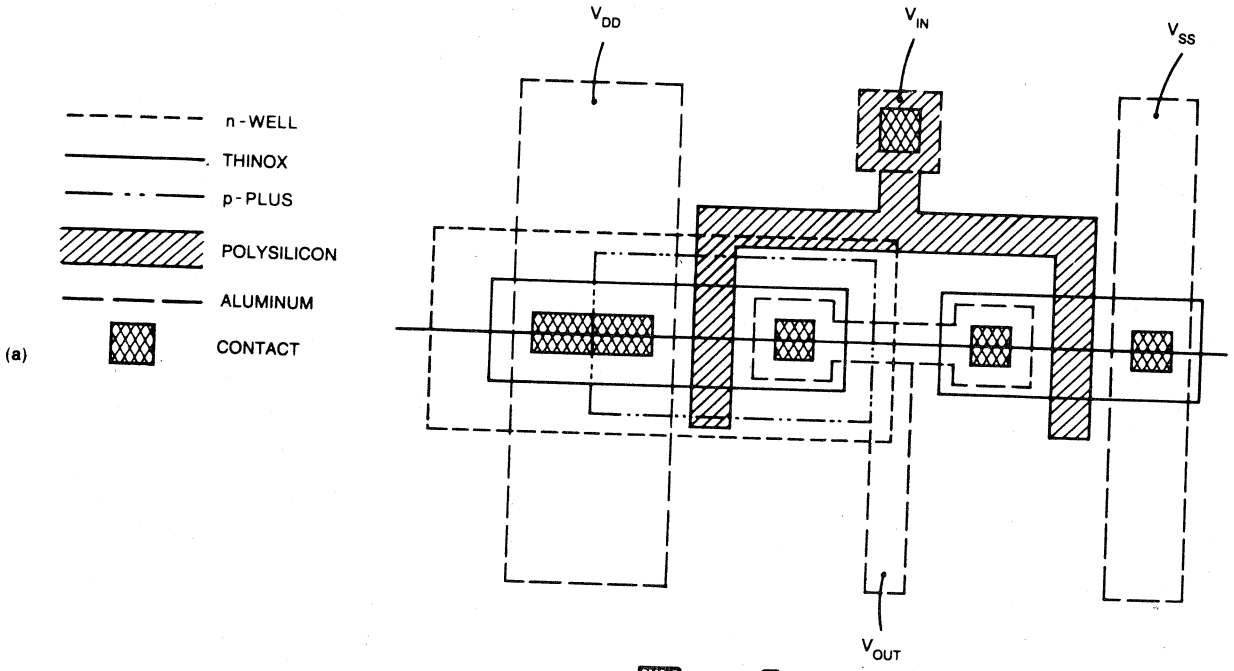
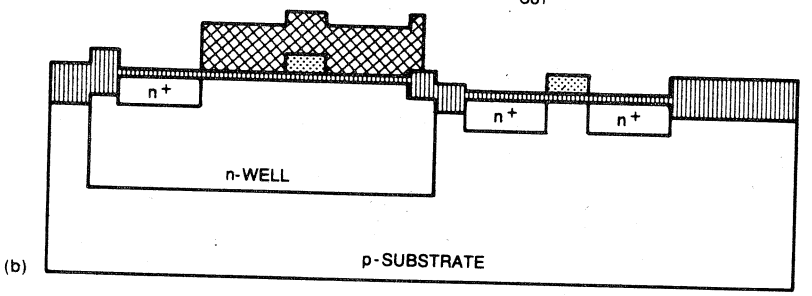
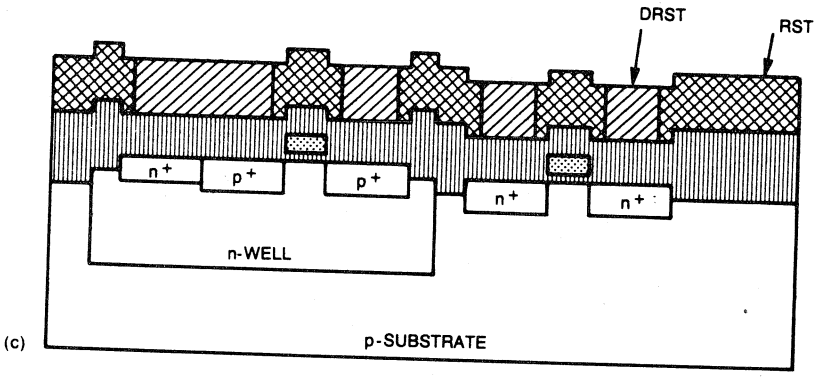


FIGURE 3.22. (Continued)

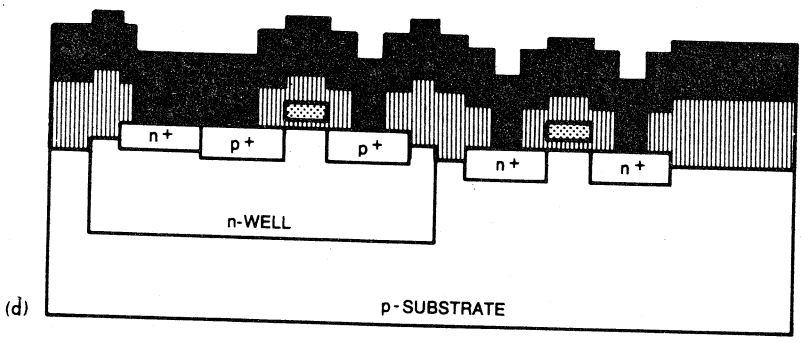




STEP 40



STEP 49



STEP 53

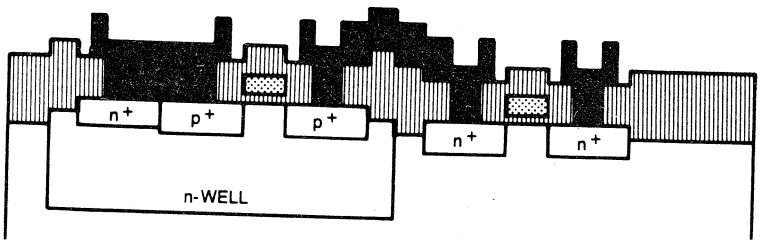


FIGURE 3.12. Berkeley n-well process snap-shots and layout for n-well inverter © IEEE 1983 ([GrLN83])

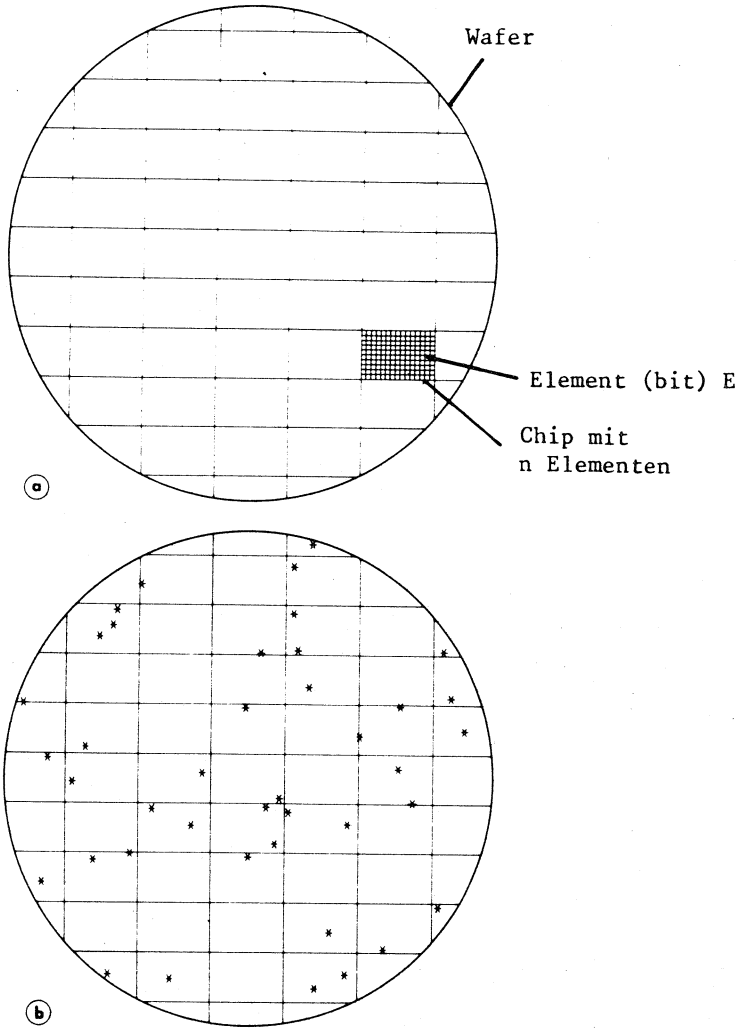


Bild 2.1. Unterteilung des Wafers in Chips.
a) Angedeutet ist ein Chip mit n Elementen.
b) Eine gleichmäßig statistische Verteilung lokaler Fehler über dem ganzen Wafer. [3.4 und 1.1].

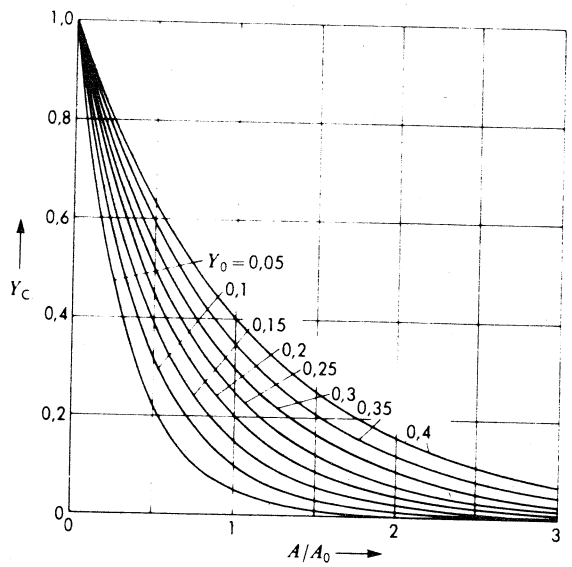


Bild 2.4. Die Chipausbeute in Abhängigkeit der relativen Fläche für verschiedene Bezugswerte der anfänglichen Ausbeute. [3.4 und 1.1].

Phase			Aktion	Ergebnis
1	Studie		Definition der Anforderungen als Zielvorgaben	Pflichtenheft
2	Spezifikation		Erarbeiten des Lösungskonzeptes und der Testmethodik	Spezifikation - Objektspezifikation - Testspezifikation
3	Systemdesign		Entwicklung von Architektur, Funktionsdiagramm, Modulstruktur, Testumgebung	Objektstruktur - Architektur - Funktionsstruktur - Modulstruktur Testrahmen - Testsystem - Testumgebung
4	Implementierung		Konstruktion der Moduln und Testprofile (Codierung, Compilation)	Getestete Moduln
5	Integration und Test		Integration der Module und Entwurfsverifikation des Gesamtobjekts	Prototyp
6	Produktion	HW	Überführung ins Produktmanagement und Fertigung	Freigegebenes Produkt
		SW	Überführung ins Produktmanagement	

Bild 1.6. Phasenmodell des Problemlösungsprozesses

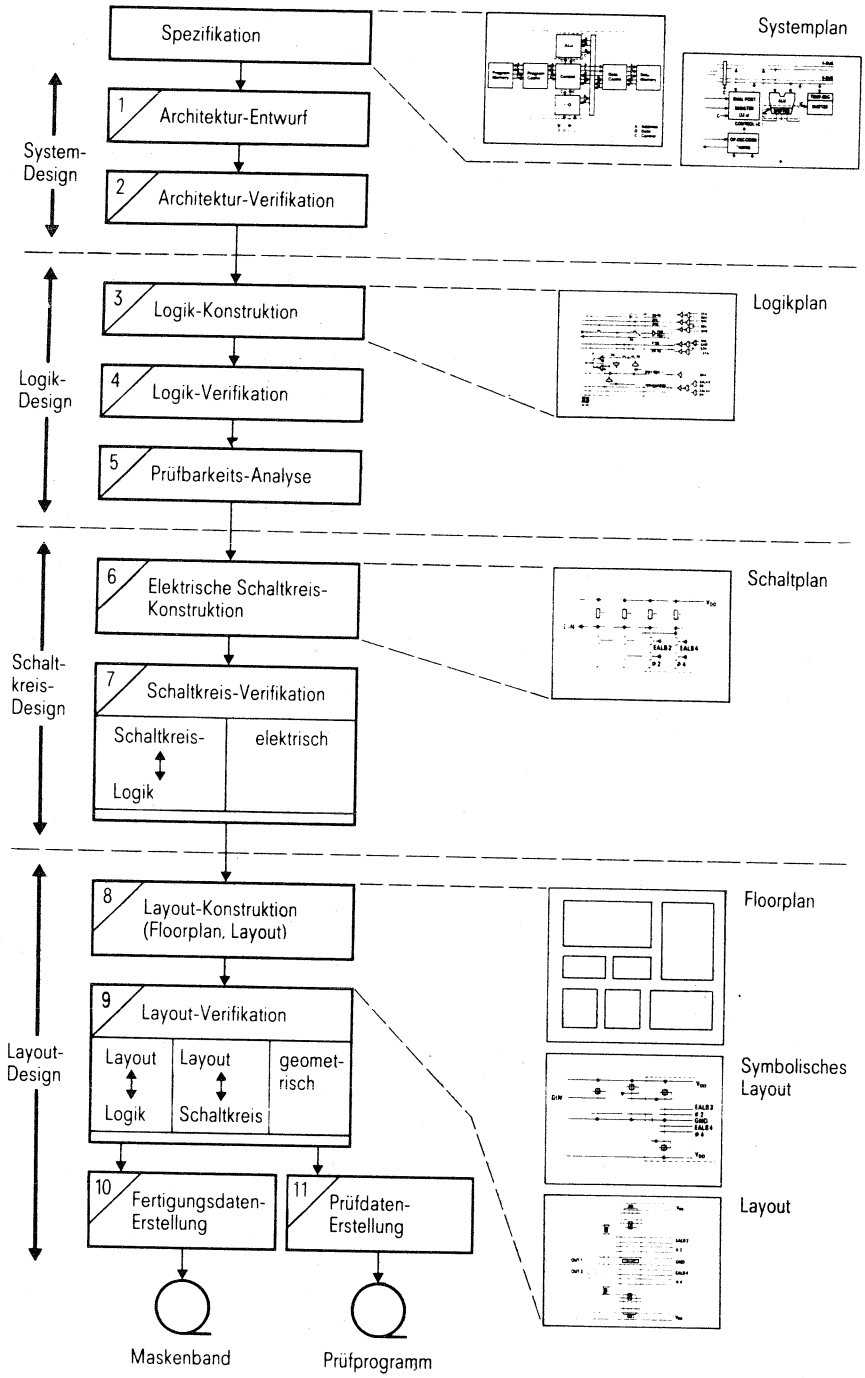


Bild 1.18. Allgemeiner IC-Designprozeß

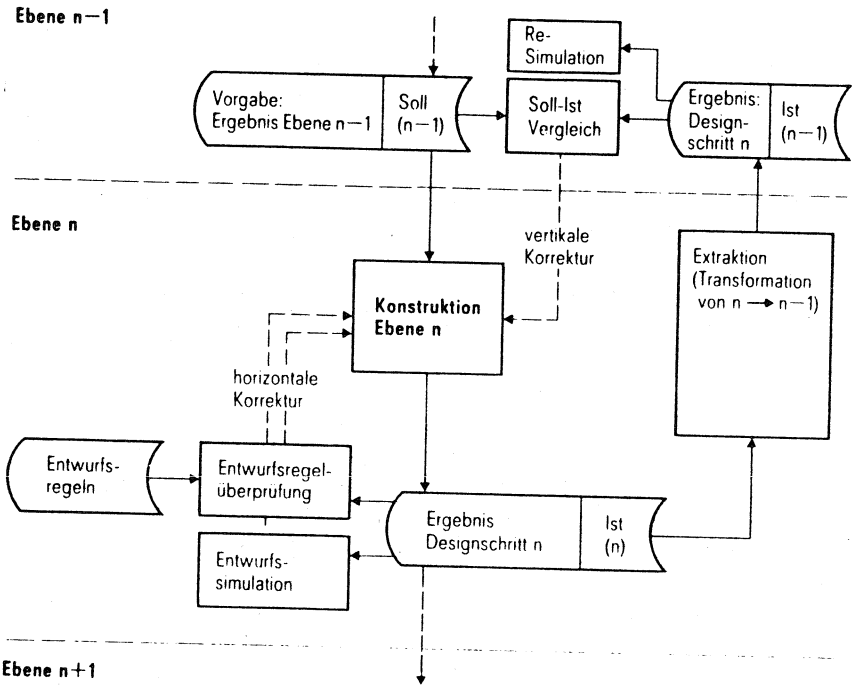


Bild 1.26. Verifikationsprozeß

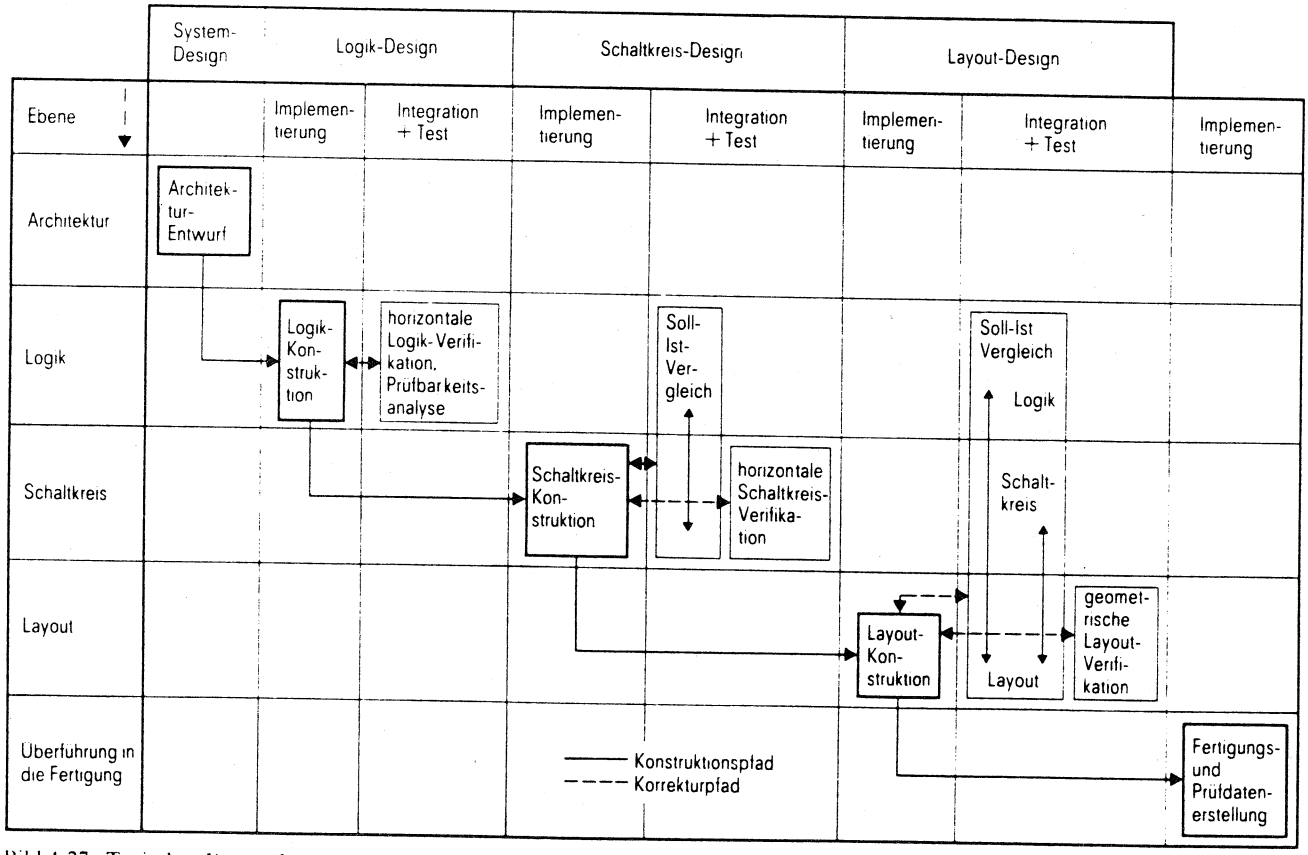


Bild 1.27. Typischer Entwurfs- und Verifikationsablauf

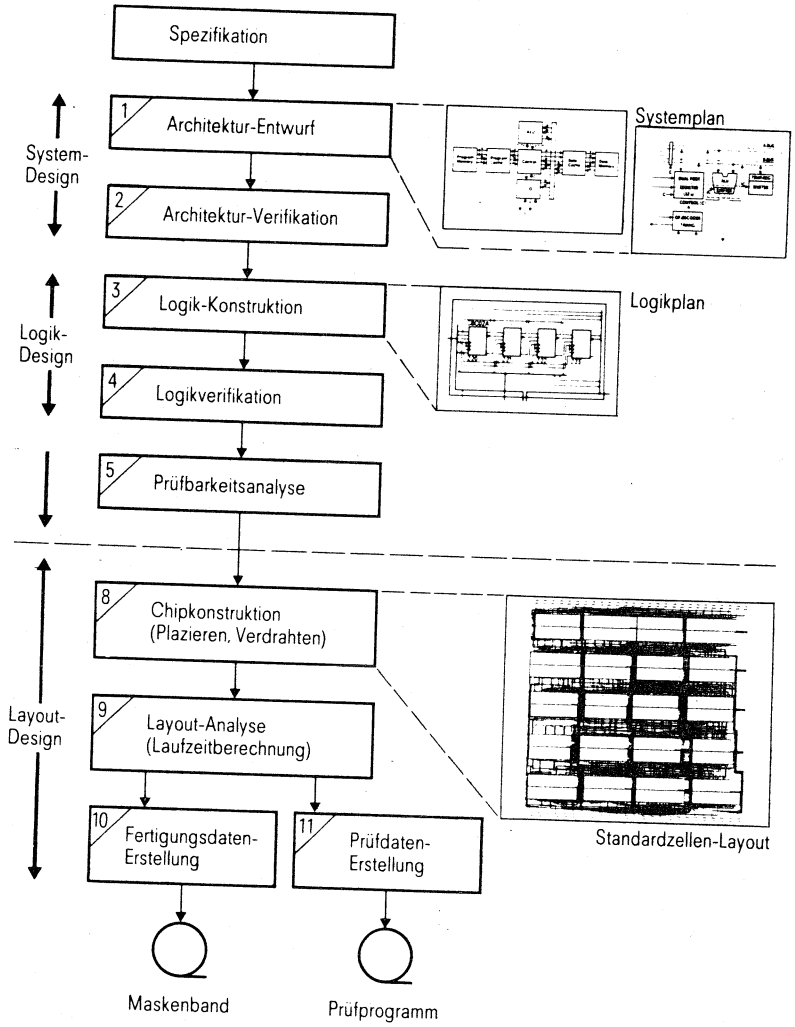


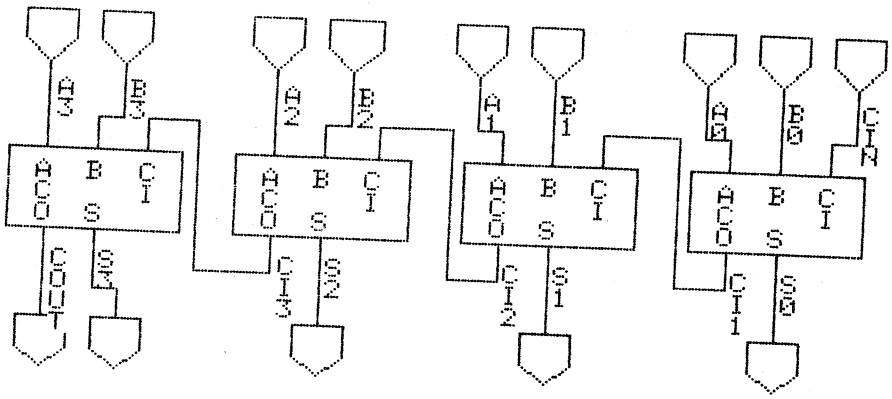
Bild 1.31. Standard-IC-Designprozeß mit „sichtbaren" Darstellungen des Schaltkreises

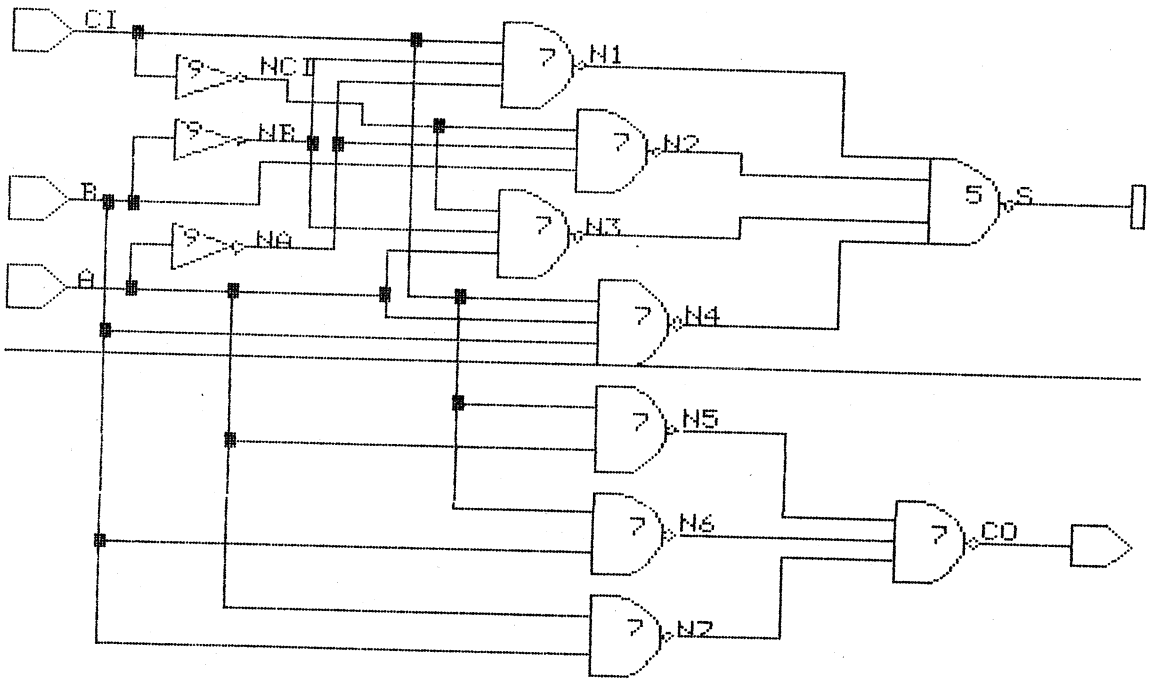
Automatisierbarkeit		Entwurfsschritt
Allgemeiner Designprozeß	Standard-Designprozeß	
interaktiv	interaktiv	Architekturentwurf
interaktiv	interaktiv	Architekturverifikation
interaktiv		Logikkonstruktion
interaktiv/ automatisch	interaktiv/ automatisch	Logikverifikation
interaktiv/ automatisch	automatisch	Prufbarkeitsanalyse
interaktiv	-	Elektrische Schaltkreiskonstruktion
interaktiv/ automatisch	-	Schaltkreisverifikation
interaktiv	automatisch	Layoutkonstruktion
interaktiv/ automatisch	(Analyse: automatisch)	Layoutverifikation
automatisch	automatisch	Fertigungsdaten-erstellung
interaktiv	automatisch	Prufdatenerstellung

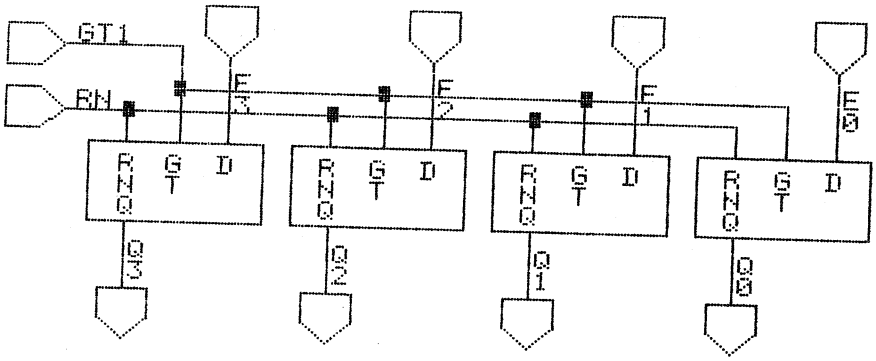
Bild 1.32. Automatisierbarkeit von Entwurfsschritten und Vorfertigung von Modellbibliotheken beim Allgemeinen und beim Standarddesignprozeß

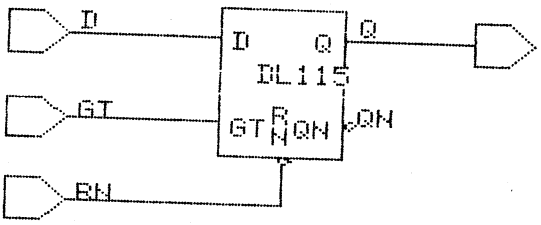
Modell	Grad der Technologieabhängigkeit
Blocksymbole	o
Register-Transfer-Modelle	o
Schaltzeichen	o
Logik-Entwurfsregeln	o
Logik-Modelle	o
Prüftechnische Entwurfsregeln	⊗
Prüftechnische Modelle	⊗
Elektrische Schaltelementsymbole	⊗
Elektrische Entwurfsregeln	⊗
Schaltkreis-Modelle	⊗
Stickelemente	●
Layout-Konturen	●
Layout-Entwurfsregeln (geometrisch und elektrisch)	●
Layout-Teile	●
Technologievorhalte	●
Modellprüfprogramme	●
Elektrische Daten	●

o unabhängig
⊗ beeinflußt
● abhängig

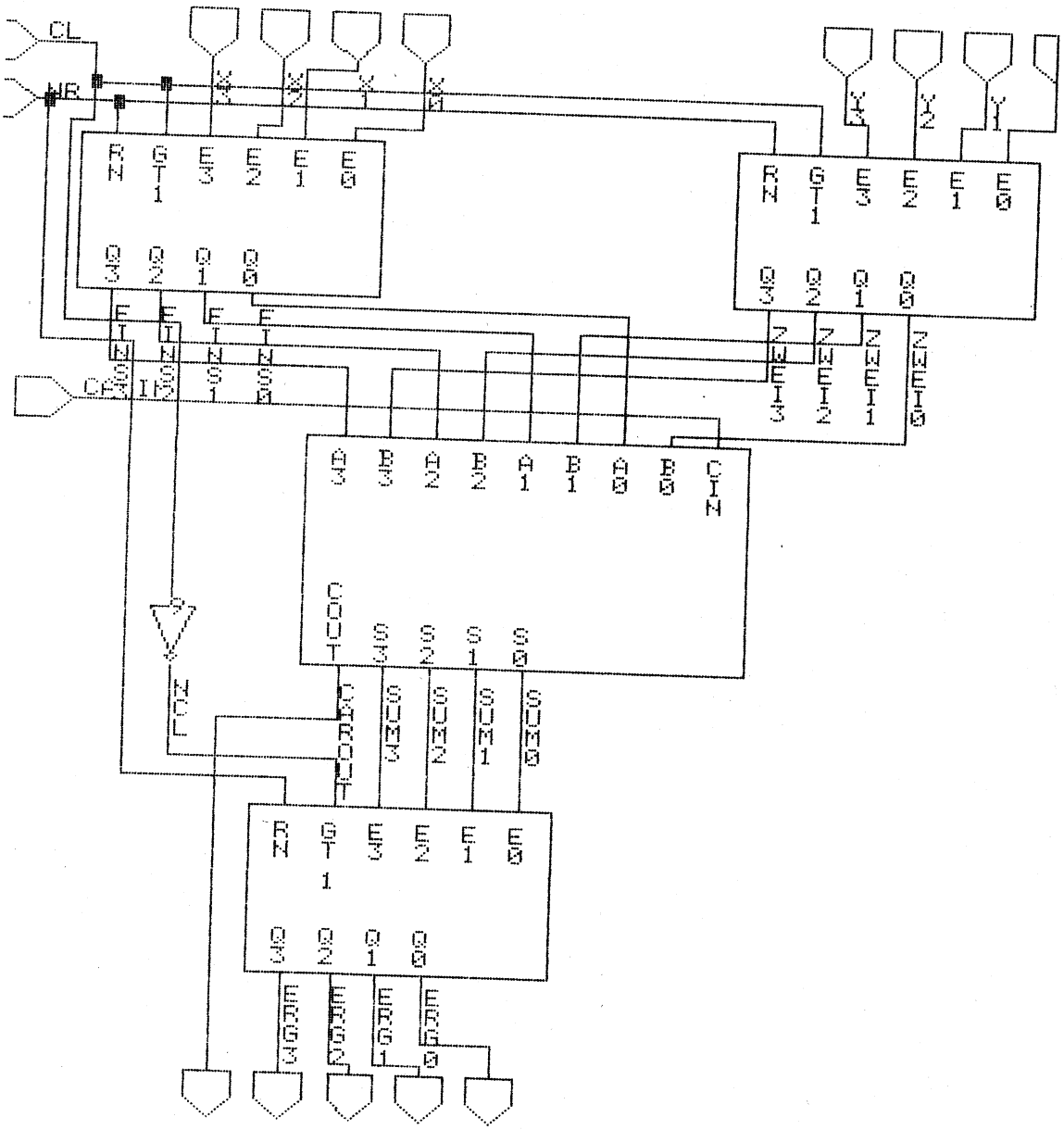

A3
B3
A
B
CI
CO
S
COUT
S3
A2
B2
CI3
S2
A1
B1
CI2
S1
A0
B0
CIN
CI1
S0


CI
NCI
NB
B
NA
A
N1
N2
N3
N4
N5
N6
N7
S
CO





4 Bit Reg, 1 Bit Reg


CL
WR
X4
X3
X2
X1
Y4
Y3
Y2
Y1
RN
G1
E3
E2
E1
E0
Q3
Q2
Q1
Q0
EIN
ENS3
ENS2
ENS1
ENS0
CRY
NWEI3
NWEI2
NWEI1
NWEI0
A3
B3
A2
B2
A1
B1
A0
B0
CIN
COUT
S3
S2
S1
S0
NC
CARRY
SUM3
SUM2
SUM1
SUM0
ERG3
ERG2
ERG1
ERG0

4 Bit ALU

7.3 DESIGN EXAMPLE

A case study will be used to provide examples in the text and therefore the specification of the example is now stated.

A video game is to be controlled by logic which performs the following functions. (A simplified specification is given here for clarity).

(i) Coins of 2p and 10p are recognised by a coin unit.

(ii) A game costs 10p only.

(iii) The game is started when the correct money is provided and a start button is pressed.

(iv) The player is given 4 lives, but can get an extra life for every 500 points obtained in the game.

(v) The game terminates when all the lives are expended.

(vi) The controller will indicate, using two lamps driven by logic signals, (a) 'game over' when the game is completed and more money may be inserted, (b) 'start game' when sufficient money has been inserted.

(vii) The game may be activated by a logic signal and cleared, ready to start again, by a second logic signal. The game will report, by logic signals, when a life has been lost and when another 500 points have been obtained.

(viii) The machine is to be clocked fairly rapidly; say every 1mS, and all input signals will be active for a single clock cycle.

(ix) All outputs must be held in the active or inactive state all the time the action is performed. Clearing the game need only be for 1 clock cycle.

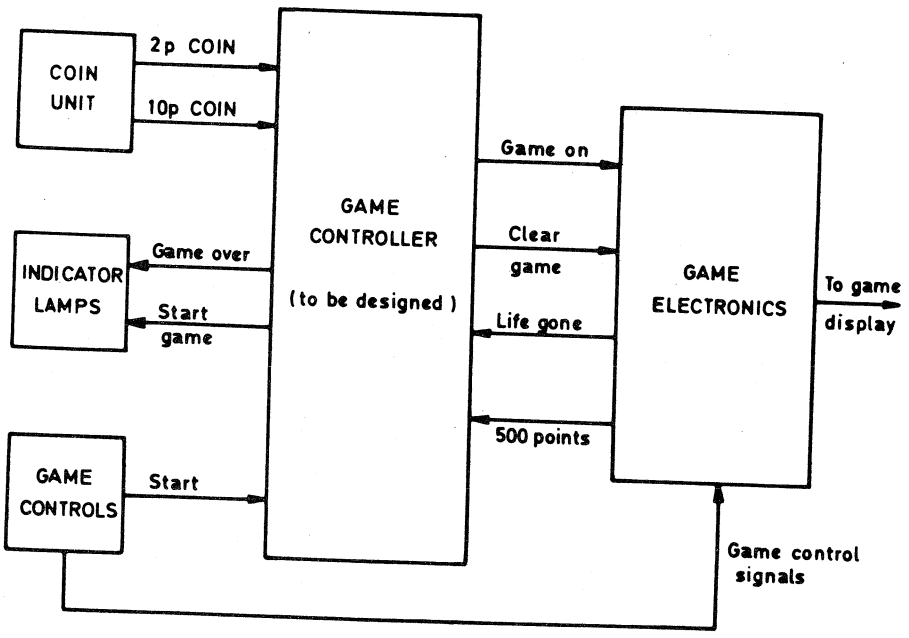


Fig 7.1 Overall view of game controller

```
Startgame:=  false
while true do
     clear game:=  true;  gameover:=  true
     coincount:=  0
     while coin count < 10p do
          game over:=  true
          if 2p detected then coin count:=  coin count +1
            *count in 2p units
          if 10p detected then coin count:=  coin count + 5

     gameover:=  false;  startgame:=  true
     while not start do nothing
     while start do idle      *wait for release
     lives:=  4;  startgame:=  false
     while lives > 0 do
          game on:=  true
          if 500 points then lives:=  lives + 1
          if life gone then lives:=  lives - 1
End of program
```

Fig 7.2 Algorithmic description of solution

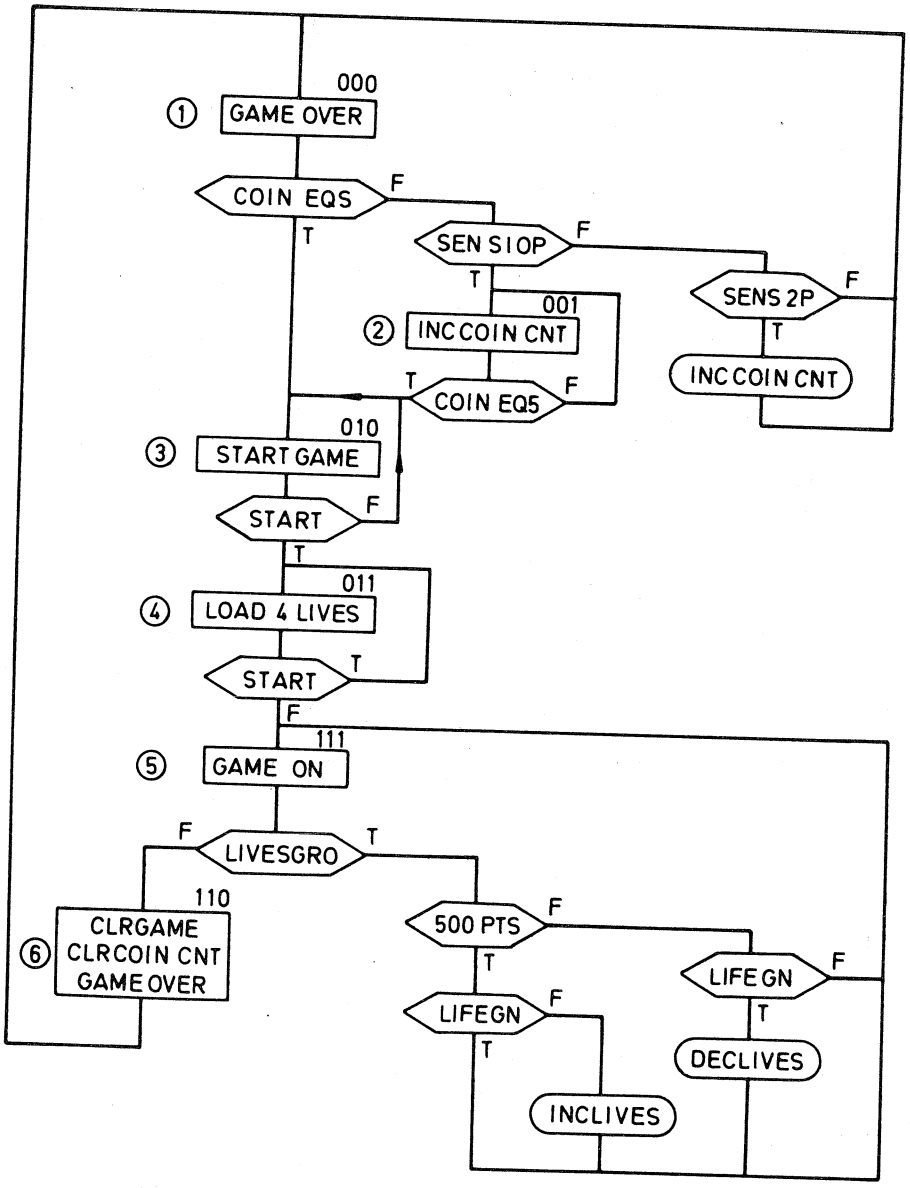


Fig. 7.8. SM chart of the Video Game Controller

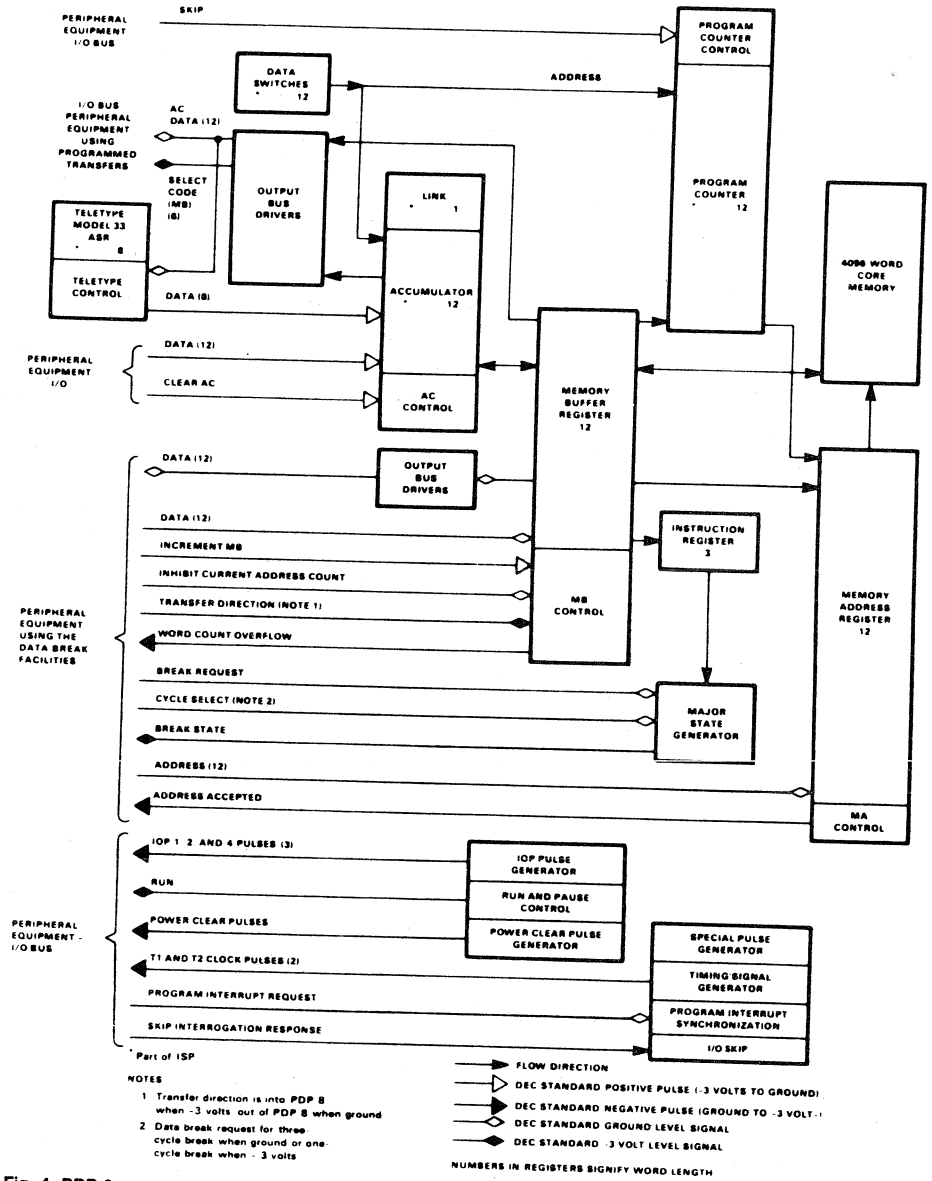
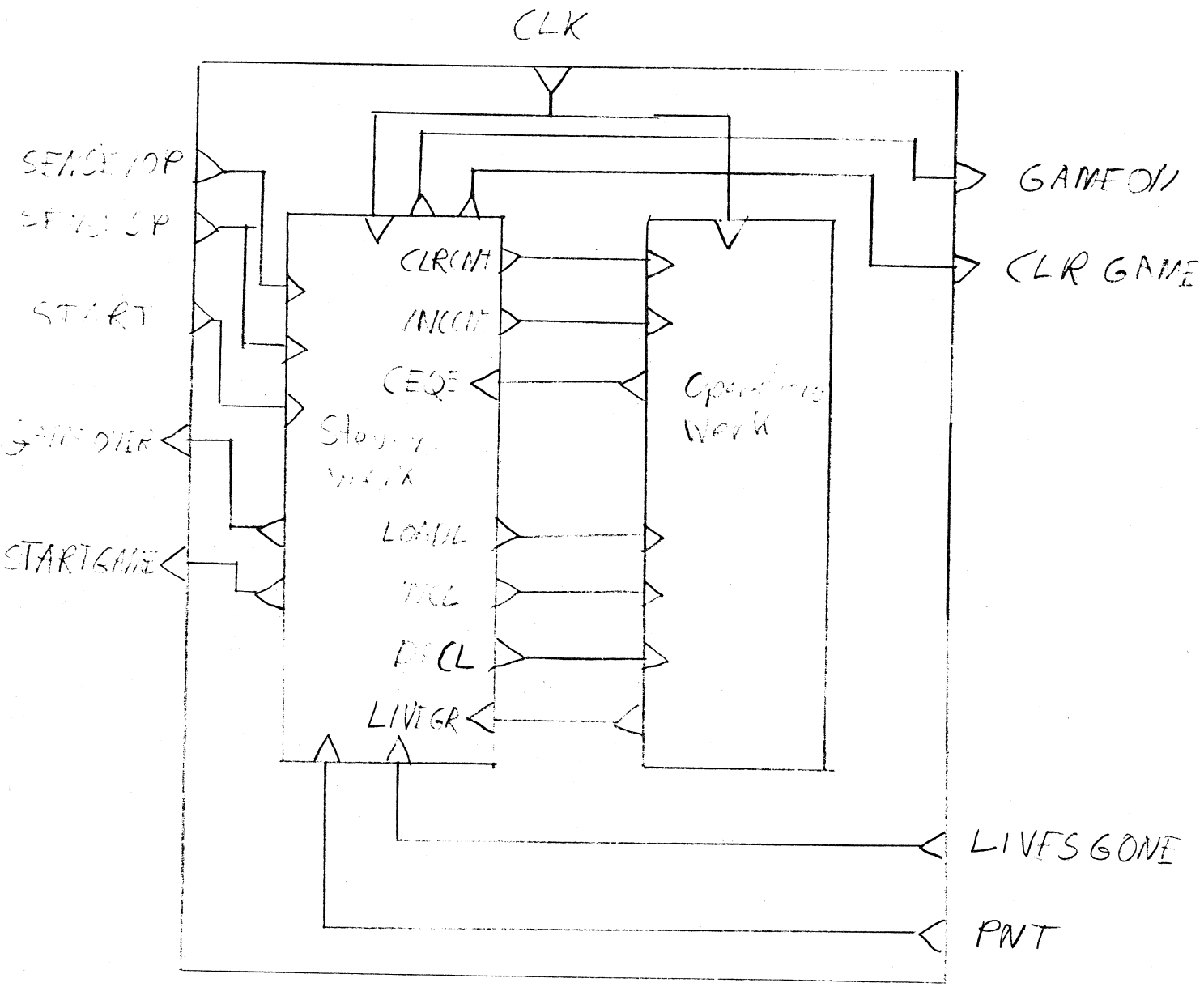


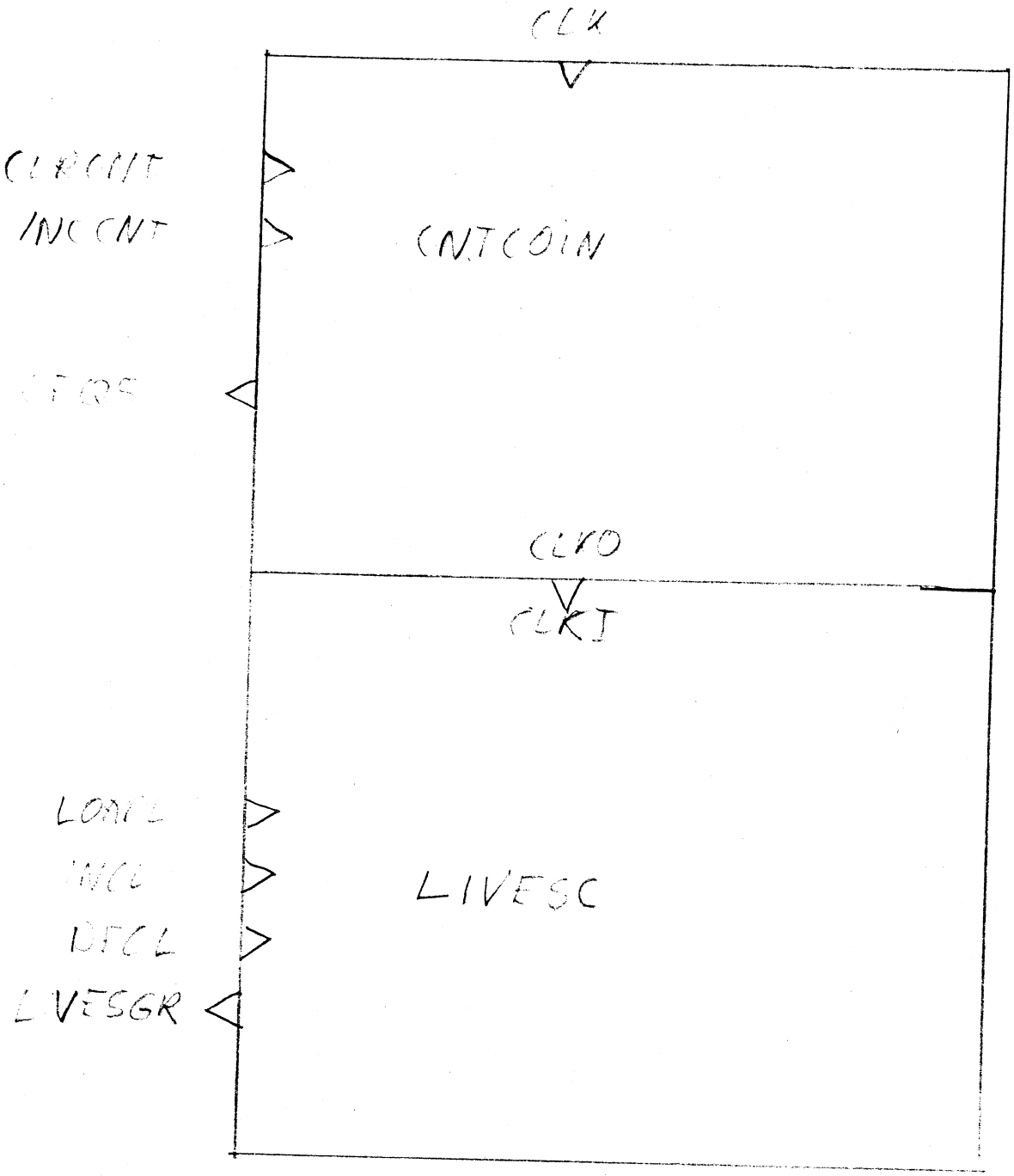
Fig. 4. PDP-8 processor block diagram.

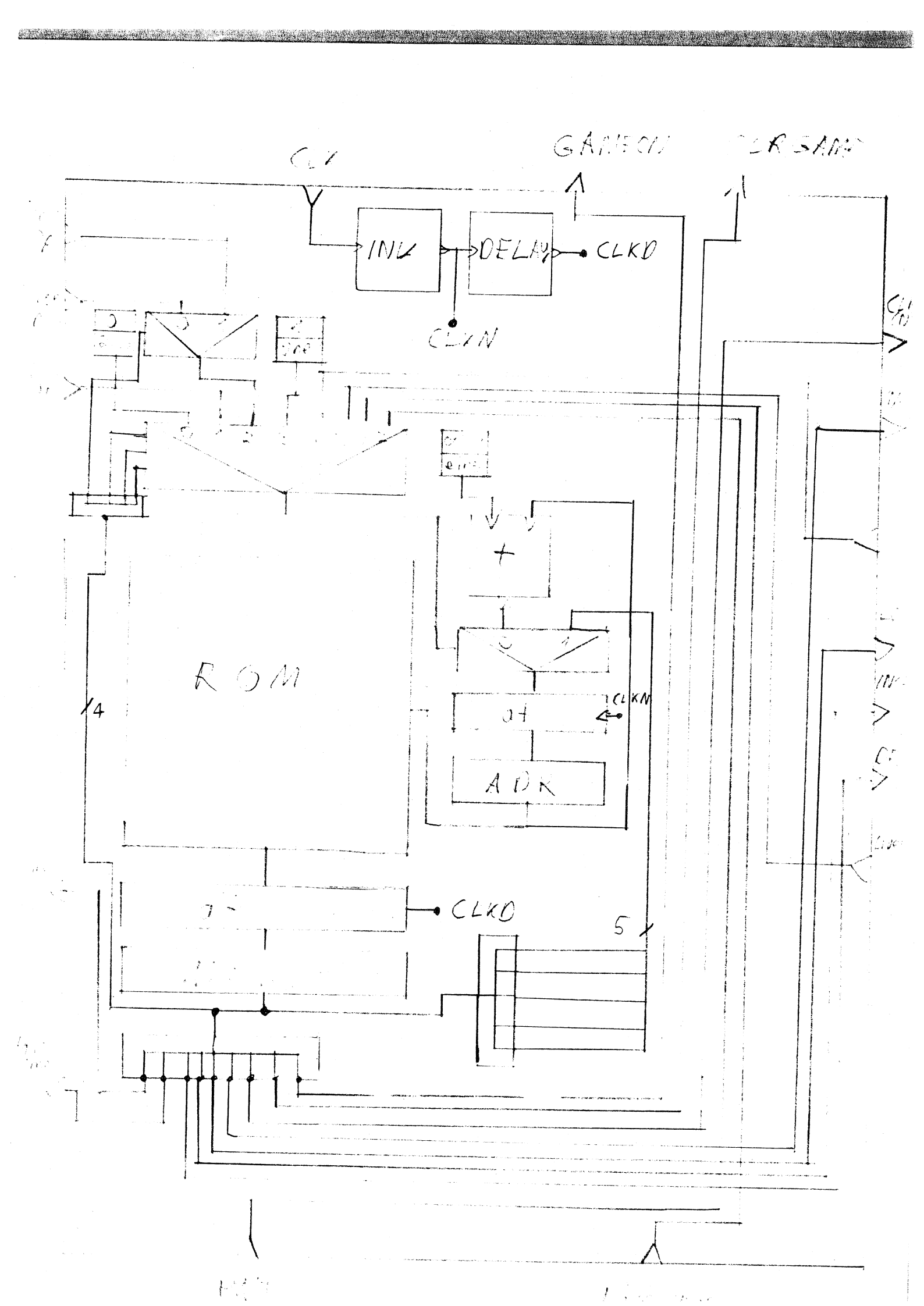
```
-------------------- Kurzbeschreibung von KARL -------------------------------

Module:   - Hardwarezelle: rechteckiger Kasten mit den Kanten:
                    FRONT (= oben), BACK (= unten), RIGHT, LEFT
            Parameter: - IN : Eingang
                       - OUT: Ausgang
                       - BI : bidirektionaler Anschluss
            Beispiel:  CELL bsp (FRONT IN  ein[7..0]
                                 OUT q
                            LEFT  OUT a,b[15..0]    (* = a[15..0]; b[15..0]  *)
                            RIGHT BI  bus[15..0]);
                       <Deklarationen>
                       BEGIN
                         <Statements>
                       END;                              (* bzw 'END.' *)

          - log. Funktion: keine Orts- und Richtungsangaben
            Parameter: - Eingaenge
                       - genau ein Ausgang (= Name der Funktion)
            Beispiel:  FUNC und[3..1] (a[3..1], b[3..1]);
                       NODE und[3..0];                   (* und weitere Deklarationen *)
                       BEGIN und .= a AND b END;

Deklarationen:
          - Konstanten: muessen unbedingt deklariert werden, da Zahlen nicht in
                        Ausdruecken auftreten duerfen.   CONSTANT zero[7..0] .= 0;
          - Taktgeneratoren: Angabe von Periode, steigender und fallender Flanke.
                                                         CLOCK clk[8;3..7];
          - Verzoegerungsglieder: Angabe von exakter Delay-Zeit bzw. von Ober- und
                        Unterschranke.                   DELAYER del1 BY 3; del2 BY 3 TO 5;
          - ROM:        Groesse und Wortlaenge:          ROM mem[65535..0; 7..0];
          - Schalter:   nur durch Benutzer veraenderbar. SWITCH input;
          - Abgriffe:   = Testpunkt, Leitung, Knoten     NODE a,b[15..0]; q
                        alle Ausgaenge mussen als NODE deklariert werden!
          - Laempchen:  kann nicht gelesen werden        LIGHTNODE <wie NODE>;
          - Register:                                    REGISTER  <wie NODE>;
          - RAM:        Groesse und Wortlaenge:          RAM mem[32767..0; 7..0];
          - Busse: - Downbus: wired AND, mit pull-up Widerstand  DOWNBUS b1[7..0];
                   - Upbus:   wired OR,  mit pull-down Wid.      UPBUS   <analog>
                   - Tristate-Bus: high impedance moeglich       TRIBUS  <analog>

Statements:
          - Verbindungen: wichtig: '.=' statt ':=' !     TERMINAL a .= b AND c; q .= d;
          - Laempchen:    --------- " ----------         LIGHT l .= NOT(q);
          - Verzoegerung: Beschreibung der Eingaenge der Delay-Glieder. DELAY del1 .= q;
          - Busverbind.:  nur hier duerfen einem Element mehrere Werte zugewiesen werden.
                          BUS upb[2..1] .= OEO b AND c;    (* open emitter *)
                              tri[7..0] .= read ENABLES data;
          - Zusammensetzen von Zellen: aneinanderliegende Ein- und Ausgaenge muessen genau
                          passen, d.h. IN muss auf OUT treffen, usw.
            - vertikal:   MAKE upper_cell @ lower_cell (FRONT IN ...    );
            - horizontal: MAKE left_cell  : right_cell ( ...           );
            Die anzugebenden Parameter stellen die Verbindung zur Aussenwelt dar, d.h.
            bei Eingaengen koennen bel. Ausdruecke stehen, bei Ausgaengen Nodes.
            Zur richtigen Plazierung der Teilzellen sind auch Transformationen verfuegbar:
            - Spiegeln: MIRX, MIRY
            - Rotieren: ROTR (90 Grad rechts), ROTL (90 links), ROTU (180 Grad)
          - Register:     wichtig: ':=' !
            - IF:   IF <expr> THEN <stmtlist> ELSE <stmtlist> ENDIF;
            - CASE: CASE <expr> OF <valuelist> : <stmtlist>
                                              . . .
                                 ELSE <stmtlist>
                    ENDCASE
            - flankengetriggerte Flipflops: AT   clk DO q := d; ...  ENDAT;
            - Master-Slave-Flipflops:       ON   clk DO q := d; ...  ENDON;
            - Latch-Statement:
              - pulsgetriggert:             WILE clk DO q := d; ...  ENDWILE; (* ohne 'H' *)
              - mit asynchr. SET/CLEAR:     WILE res DO q := 0
                                                    OTHERWISE AT clk TAKE d; ENDWILE

Ausdruecke:
          - Teilbereiche:                   a .= b[7..5];
          - zusammengesetzte Ausdruecke:    c .= a . b;
          - Multiplexer: - IF:   a .= IF high THEN b[1] ELSE b[0];
                         - CASE: a .= CASE sel[2..0] OF
                                      0,4 : b + c
                                      1   : b - c
                                      ELSE b AND c  ENDCASE;
                          'ELSE' darf nur weglassen werden, wenn alle Werte angegeben sind.

Operatoren:
          - arithm.: *  /  MOD  +  -  =
          - relat.:  =  <>  >  <  >=  <=
```

```
Standard-Funktionen:
          - Shifts:   SHR, SHL, DSHR, DSHL, CSHR, CSHL, NSHR, NSHL, ESHR, ESHL,
                      CIRSHR, CIRSHL, PUSH, POP, DPUSH, DPOP, ...
          - Shuffle:  - bitshuffle:  FOLD
                      - wordshuffle: MERGE
          - Spiegeln: - bit:  REFLECT
                      - word: REVERSE
          - Auswahl:  MSB, LSB, MSW, LSW
          - Code-Konvertierung: DECODE, ENCODE, DECOUNT, ENCOUNT
          - Tests:    EQU, EVEN, ODD, TESTSINGULARY, TESTUNARY
          - sonstige: - DYN: dynamisches Speicherelement: TERMINAL q .= DYN&2(d);
                      - NOT: bitweise Invertierung
                      - INC: erhoehen um einen best. Wert:        q .= INC&3(b); (* b + 3 *)
                      - DEC: erniedrigen ------ " -------
                      - PRIL: Ausfiltern der am weitesten links stehenden '1'.
                      - PRIR: ------------"------------  rechts -----"------.

Auswahl von speziellen Zellen/Funktionen:
         Durch Anhaengen von &<Nummer> an den Namen:     TERMINAL a .= counter&5;


------------------------------- Simulator: ------------------------------------------
Definition von Makros:  DEF <name> <befehl1> ... ENDDEF

Verzweigung:            IF counter IS 255 THEN WRITE 'max value'
                                          ELSE STEP            ENDIF

Selektion von Zellen:   IN counter&2 SET clk, reset ... ENDIN
                        (= SET counter&2.clk, counter&2.reset)

Schleifen:   - LOOP:    LOOP 10  (* Anzahl der Durchlaeufe *)
                          STEP ...      ENDLOOP
                        Verlassen der Schleife: SKIP
             - REPEAT:  REPEAT ... UNTIL counter IS 0
             - WHILE:   WHILE counter ISNOT 0 ... ENDWHILE    (* mit 'H' *)

Zuweisungen: - ASSIGN:  ASSIGN opd1 '0110, opd2 #ff, digit 7
             - INIT:    alle HW-Elemente auf Null oder die angegebene Konstante setzen.
                        INIT  (* = INIT 0 *)
             - INVERT:  die angegebenen Bits invertieren. INVERT a(15..10, 5..1)
             - KEEP:    Wert setzen und festhalten.       KEEP a 15, b 1
             - LOCK:    haelt momentanen Wert.
             - RELEASE: Aufheben der Sperren von KEEP und LOCK.
             - RESET:   angegebene Bits auf Null setzen   RESET clk, a[13..9]
             - SET:     ----------"-------- Eins --"---

Einzelschritt in Simulation: STEP. Fuer mehrere Schritte Makro oder LOOP verwenden.

Ausgabe:     - WRITE:   Text ausgeben.                    WRITE 'hello world'
             - PLOTALL: angegebene Werte bei jedem Simulationsschritt ausgeben.
                        Aufeinanderfolgende PLOTALL-Befehle ueberschreiben sich nicht
                        sondern fuegen die Werte zur Ausgabeliste hinzu.
             - PLOTNEW: wie PLOTALL, nur dass die Werte erst ausgegeben werden, wenn sich
                        mindestens einer geaendert hat.   PLOTNEW a:2, b:3, c:1
             - weitere: PRINT, PRINTACT, PRINTRTC, PRINTUNIT, PRINTBIT.

Ein- und Ausgabe-Umlenkung:
             - NEWIN:   hole nachfolgende Simulator-Befehle von ASCII-File. Nach Aufruf
                        dieses Befehls wird nach dem Filenamen gefragt.
             - NEWOUT:  schreibe Ausgabe auf File.        NEWOUT
             - OLDIN:   Zurueckschalten der Eingabe.
             - OLDOUT:  -------"-------     Ausgabe.
             - STDIN:   Eingabe von Standard-Input (meist Tastatur). Muss am Ende des
                        Eingabe-Files stehen!
             - STDOUT:  Ausgabe auf Standard-Output (meist Bildschirm).
             - STARTPRT: alle von nun an gegebenen Kommandos kommen auf ein File.
             - ENDPRT:   Aufheben von STARTPRT

Editieren des Eingabe-Files: EDIT  (tut auf Apollo nicht)

Verlassen des Simulators:
             - ENDSIM:  zurueck ins KARL-System
             - RETURN:  -----"----- und behalten aller Werte fuer Fortsetzung der Simulation.

Speicher (RAM/ROM) lesen:     LOADMEM:                   LOADMEM microcode[2..2047]
--------"--------- schreiben: DUMPMEM:                   DUMPMEM cache[0..511]
           Nach Aufruf des Befehls wird nach dem Filenamen gefragt. Die Files sind
           einfache ASCII-Files, in denen eine Reihe von Konstanten steht.
```


CLK
GAMEON
CLR GAME
START
CLRCNT
INCCNT
CEQE
LOADL
LIVEGR
LIVESGONE
PNT

Operations work




INV
DELAY
CLKD
CLKN
ROM
+
CLKM
ADR
CLKD
4
5

Mikroprogramm fuer Game-Controller

Bit 0 - 4	Sprungadresse
Bit 5 - 7	Conditions 0 = never 1 = START 2 = SENSE 3 = always 4 = CEQ5 5 = LIVESGR 6 = LIVESGONE 7 = PNT
Bit 8	SENSE 0 = SENSE2P 1 = SENSE10P
Bit 9	GAMEON
Bit 10	CLRGAME
Bit 11	CLRCNT
Bit 12	INCCNT
Bit 13	LOADL
Bit 14	INCL
Bit 15	DECL
Bit 16	STARTGAME
Bit 17	GAMEOVER

		ADR	MPW
STATE1_0:	GAMEOVER if CEQ5 then STATE3_0	00000	1000000000 100 00111
STATE1_1:	GAMEOVER,SENSE10P if SENSE then STATE2_0	00001	1000000001 010 00101
STATE1_2:	GAMEOVER if SENSE then STATE1_4	00010	1000000000 010 00100
STATE1_3:	GAMEOVER if ALWAYS then STATE1_0	00011	1000000000 011 00000
STATE1_4:	GAMEOVER,INCCNT if ALWAYS then STATE1_0	00100	1000010000 011 00000
STATE2_0:	GAMEOVER,INCCNT if CEQ5 then STATE3_0	00101	1000010000 100 00111
STATE2_1:	GAMEOVER if ALWAYS then STATE2_0	00110	1000000000 011 00101
STATE3_0:	STARTGAME if START then STATE4_0	00111	0100000000 001 01001
STATE3_1	STARTGAME if ALWAYS then STATE3_0	01000	0100000000 011 00111

```
STATE4_0:  LOADLIVES                          01001   0000100000 001 01001
           if START then STATE4_0

STATE5_0:  GAMEON                             01010   0000000010 101 01100
           if LIVESGR then STATE5_2

STATE5_1:  CLRGAME,CLRCNT,GAMEOVER            01011   1000001100 011 00000
           if ALWAYS then STATE1_0

STATE5_2:  GAMEON                             01100   0000000010 111 10000
           if PNT then STATE5_6

STATE5_3:  GAMEON                             01101   0000000010 110 01111
           if LIVESGN then STATE5_5

STATE5_4:  GAMEON                             01110   0000000010 011 01010
           if ALWAYS then STATE5_0

STATE5_5:  GAMEON,DECLIVES                    01111   0010000010 011 01010
           if ALWAYS then STATE5_0

STATE5_6:  GAMEON                             10000   0000000010 110 01010
           if LIVESGN then STATE5_0

STATE5_7:  GAMEON,INCLIVES                    10001   0001000010 011 01010
           if ALWAYS then STATE5_0
```

```
PDP8 :=
    begin

! The basic PDP-8 instruction set (without extended arithmetic element)
! is implemented.  No I/O devices are included in the description.
! I/O instruction execution is limited to the instructions that
! deal with the internal interrupt enable flags and status.

! Reference: "The DIGITAL Small Computer Handbook", 1967 Edition,
!            Digital Equipment Corporation.

**MP.State**

    MP[#0:#7777]<0:11>,                     ! Main memory (4k words)
        page.zero[#0:#177]<0:11> := MP[#0:#177]<0:11>,
        auto.index[#0:#7]<0:11>  := MP[#10:#17]<0:11>,

    MB<0:11>                                ! Memory buffer

**PC.State**

    L<>,                                    ! Link bit
    AC<0:11>,                               ! Accumulator

    PC<0:11>                                ! Program counter

**External.State**

    switches<0:11>,                         ! Console data switches
    interrupt.request<>                     ! Any device requesting interrupt

**Implementation.Declarations**

    go<>,                                   ! 1 when running
    interrupt.enable<>,                     ! 1 when Pc can be interrupted
    last.pc<0:11>,
    skip<>                                  ! Skip flag

**Instruction.Format**

    IR\instruction.register<0:2>,           ! Operation code
    ib\indirect.bit<>     := MB<3>,         ! Memory reference:
                                            ! 0 = direct; 1 = indirect
    pb\page.bit<>         := MB<4>,         ! 0 = zero page; 1 = current page
    pa\page.address<0:6> := MB<5:11>,
    io.select<0:5>        := MB<3:8>,       ! I/O device select
    io.pulse<0:2>         := MB<9:11>,      ! I/O pulse control bits
       io.pulse.1<>       := io.pulse<0>,
       io.pulse.2<>       := io.pulse<1>,
       io.pulse.4<>       := io.pulse<2>,
    group<>               := MB<3>,         ! Microinstruction group
    CLA<>                 := MB<4>,         ! Clear AC
    CLL<>                 := MB<5>,         ! Clear Link
    CMA<>                 := MB<6>,         ! Complement AC
    CML<>                 := MB<7>,         ! Complement Link
    RAR<>                 := MB<8>,         ! Rotate right
    RAL<>                 := MB<9>,         ! Rotate left
    RTx<>                 := MB<10>,        ! Rotate twice
    IAC<>                 := MB<11>,        ! Increment AC
    SMA<>                 := MB<5>,         ! Skip on minus AC
    SPA<>                 := MB<5>,         ! Skip on positive AC
    SZA<>                 := MB<6>,         ! Skip on zero AC
    SNA<>                 := MB<6>,         ! Skip on AC not zero
    SNL<>                 := MB<7>,         ! Skip on Link not zero
    SZL<>                 := MB<7>,         ! Skip on Link zero
    is<>                  := MB<8>,         ! Invert skip sense
    OSR<>                 := MB<9>,         ! Logical or AC with switches
    HLT<>                 := MB<10>         ! Halt the processor

**Address.Calculation**

    MA\effective.memory.address<0:11> :=
        begin
        MA = '00000 @ pa next                   ! Zero page
        IF pb => MA<0:4> = last.pc<0:4> next    ! Current page
        IF ib =>                                ! Indirect bit
            begin
            IF MA<0:8> eql #001 =>              ! Auto index
                MP[MA] = MP[MA] + 1 next
            MA = MP[MA]                         ! Indirect address
            end
        end

**Instruction.Interpretation**

    start{main} :=
        begin
        go = 1 next
        run()
        end,

    run\instruction.interpretation :=
        begin
        IF go =>
            begin
            MB = MP[PC]; last.pc = PC next
            PC = PC + 1 next
            exec() next
            IF interrupt.enable and interrupt.request =>
                begin
                MP[0] = PC next
                PC = 1
                end next
            RESTART run
            end
```

```
(*************************************************************************)
(*                      GAME CONTROLLER                                  *)
(*************************************************************************)

cell gamecontr (left out gameon, clrgame
                    in sense10p, sense2p, start
                    out gameover, startgame
                back in pnt, livesgone);

 cell cntcoin (left in clk, clrcnt, inccnt
                    out ceq5
                back out clko);

 constant zero[2..0] .= '000;
 constant one[2..0] .= '001;
 constant five[2..0] .= '101;
 node ceq5, clko;
 register count[2..0];

 begin
   at clk do count := case clrcnt.inccnt of
                        1: count + one
                        2: zero
                        else count
                      endcase;
   endat;
   terminal ceq5 .= (count = five);
   terminal clko .= clk;
end;

cell livesc (front in clki
             left in loadl, incl, decl
                  out livesgr);

constant one[3..0] .= '0001;
constant zero[3..0] .= 0;
constant four[3..0] .= '0100;
node livesgr;
register lcount[3..0];

begin
  at clki do lcount := if loadl then four
                       else case decl.incl of
                              '00: lcount
                              '01: lcount + one
                              '10: lcount - one
                              '11: lcount
                            endcase
                       endif;
  endat;
  terminal livesgr .= (lcount > zero);
end;

cell steuer (front in clk
                 out gameon, clrgame
             right out clko, clrcnt, inccnt
                 in ceq5
                 out loadl, incl, decl
                 in livesgr
             back in pnt, livesgone
             left in sense10p, sense2p, start
                  out gameover, startgame);

constant zero .= 0;
```

```
    constant one .= 1;
    constant eins[4..0] .= '00001;
    node gameover, startgame, decl, incl, loadl, inccnt, clrcnt, clrgame;
    node gameon, clko, s1, s2, sense;
    register adr[4..0];
    register mpw[17..0];
    rom mpr[31..0;17..0];
    delayer clkd by 3;

    begin
      delay clkd .= not (clk);
      terminal clko .= clk;
      terminal sense .= if mpw[8] then sense10p else sense2p endif;
      terminal s1 .= case mpw[7].mpw[6].mpw[5] of
                        '000: zero
                        '001: start
                        '010: sense
                        '011: one
                        '100: ceq5
                        '101: livesgr
                        '110: livesgone
                        '111: pnt
                      endcase;
      at not (clk) do adr := if s1 then mpw[4].mpw[3].mpw[2].mpw[1].mpw[0]
                             else adr + eins
                             endif;
      endat;
      terminal s2 .= clkd;
      at s2 do mpw := mpr[adr]; endat;
      terminal gameon .= mpw[9];
               clrgame .= mpw[10];
               clrcnt .= mpw[11];
               inccnt .= mpw[12];
               loadl .= mpw[13];
               incl .= mpw[14];
               decl .= mpw[15];
               startgame .= mpw[16];
               gameover .= mpw[17];
   end;

   cell operate (left in clk, clrcnt, inccnt
                      out ceq5
                      in loadl, incl, decl
                      out livesgr);

   node ceq5, livesgr;

   begin
     make cntcoin @ livesc (left in clk; clrcnt; inccnt
                                out ceq5
                                in loadl; incl; decl
                                out livesgr);
   end;

clock clk[4;2..0];
node gameon, clrgame, gameover, startgame;

begin
  make steuer : operate (front in clk
                               out gameon; clrgame
                         left in sense10p; sense2p; start
                              out gameover; startgame
                         back in pnt; livesgone);
end.
```

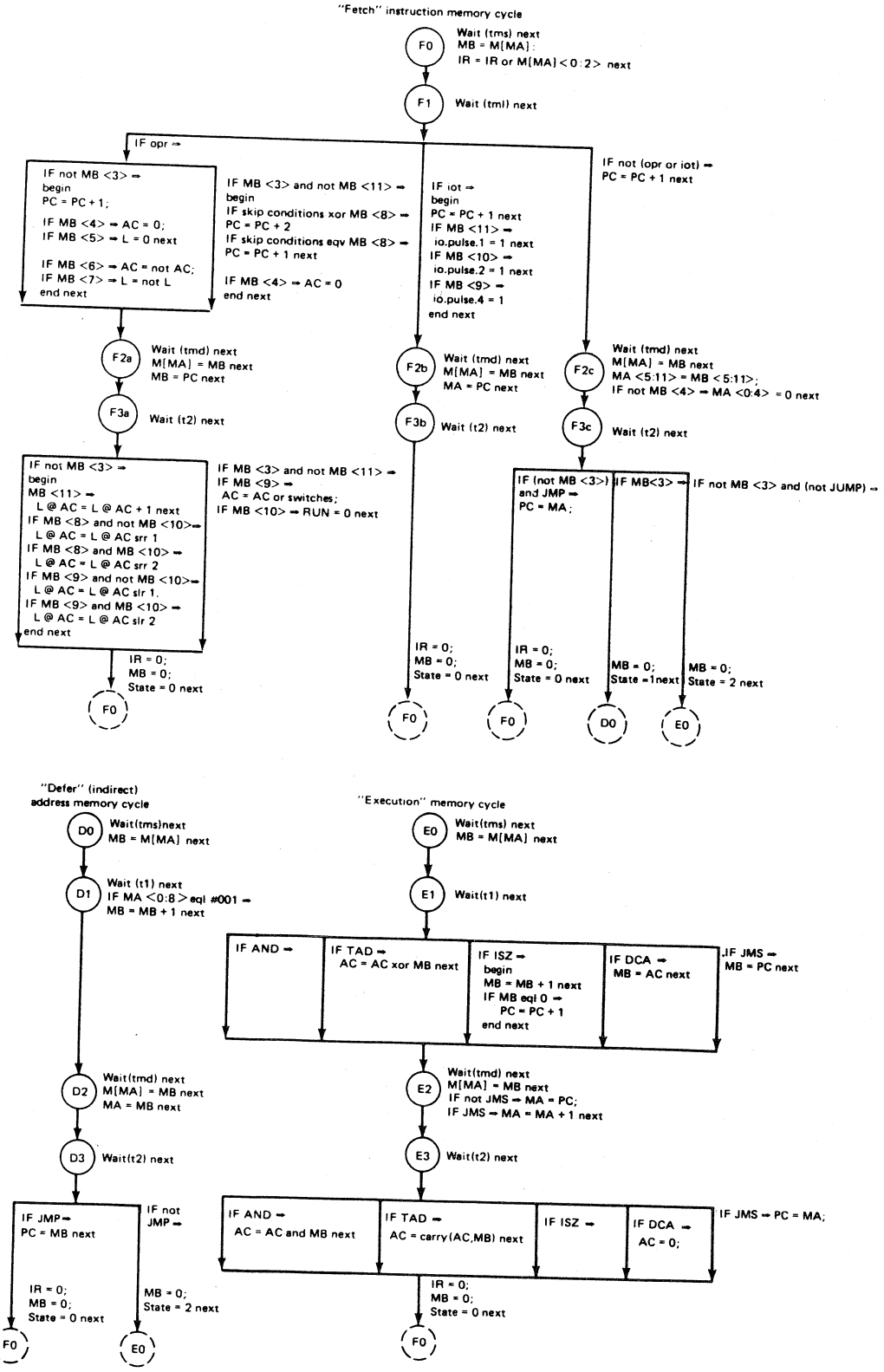


Fig. 7. PDP-8 Pc state diagram.

```
FUNC COUNT[2..0]();
CONSTANT ONE[2..0] .= 1;
         ZERO[2..0] .= 0;
SWITCH   INC, RESET;
CLOCK    CLK[2;1..0];
REGISTER  COUNT[2..0];
BEGIN
  TERMINAL
    INCOUNT[2..0] .= IF RESET THEN ZERO
                     ELSE IF INC THEN COUNT + ONE
                                 ELSE COUNT
                          ENDIF
                ENDIF;
    AT CLK DO COUNT[2..0] := INCOUNT; ENDAT;
END.
```

Register transfer description of a counter

```
FUNC COUNT[2..0]();
SWITCH   INC, RESET;
CLOCK    CLK[2;1..0];
REGISTER  COUNT[2..0];
BEGIN
  TERMINAL
    AND1 .= INC AND NOT(RESET);
    OR1 .= AND1 OR RESET;
    AND2 .= AND1 AND COUNT[0];
    OR2 .= AND2 OR RESET;
    AND3 .= AND2 AND COUNT[1];
    OR3 .= AND3 OR RESET;
    NOT1[2..0] .= NOT(OR3.OR2.OR1);
    NOT2[2..0] .= NOT(COUNT);
    AND4[2..0] .= AND3.AND2.AND1 AND NOT2;
    AND5[2..0] .= NOT1 AND COUNT;
    INCOUNT[2..0] .= AND4 OR AND5;
    AT TAKT DO COUNT := INCOUNT; ENDAT;
END.
```

Logic description of a counter

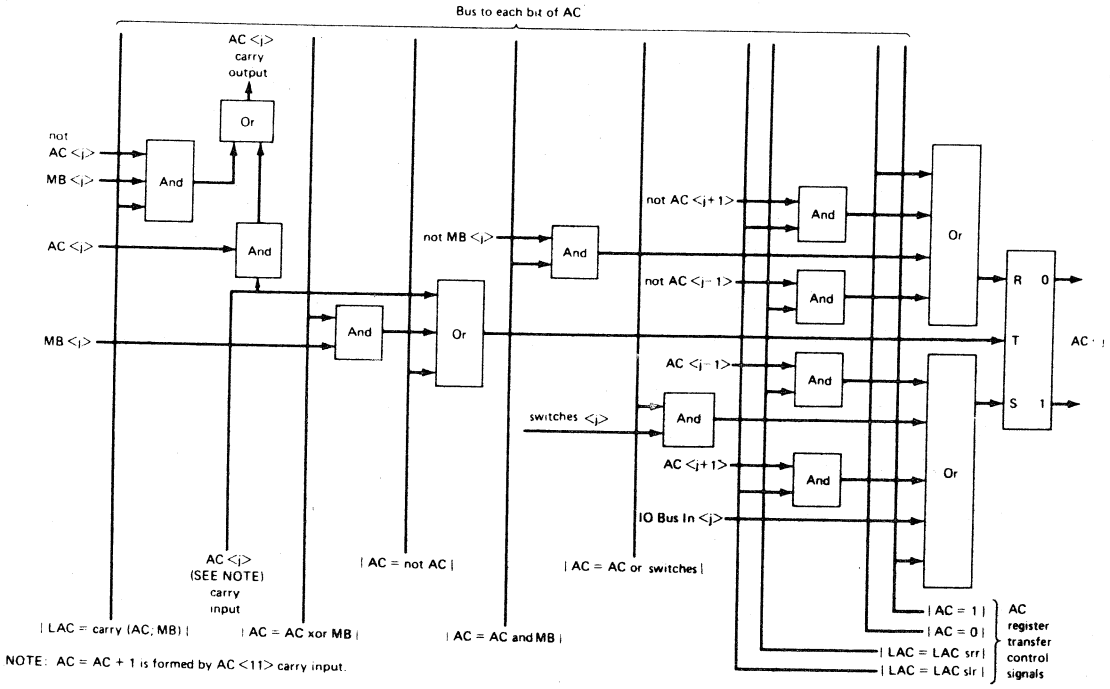


Fig. 8. PDP-8 AC<J> bit logic diagram.

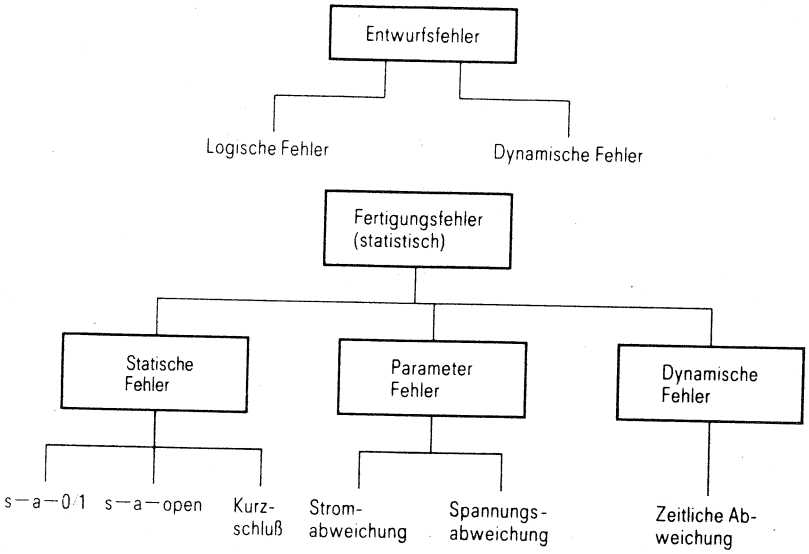


Bild 4.5. Fehlerklassifizierung

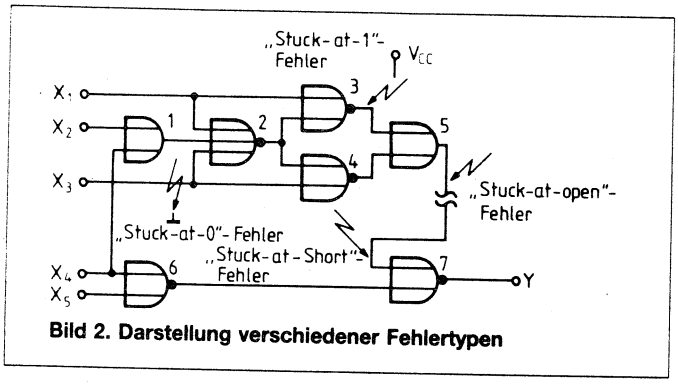


Bild 2. Darstellung verschiedener Fehlertypen

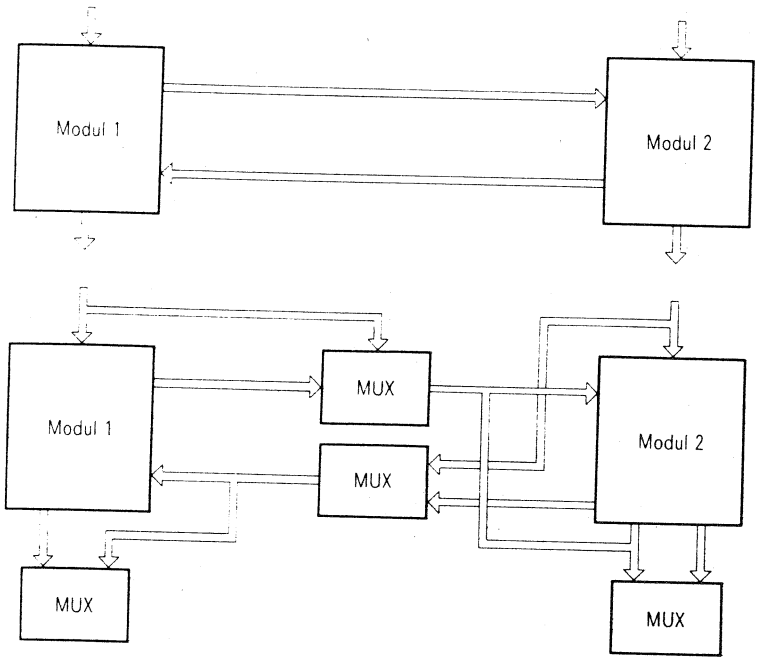


Bild 4.8. Partitionierung zweier Module auf einem Baustein

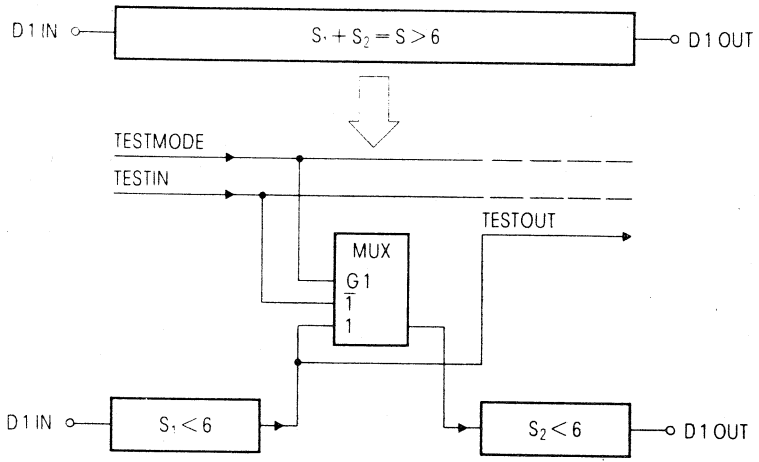


Bild 4.6. Multiplexer zur Reduzierung der sequentiellen Tiefe

Tabelle 4.1. Allgemeine prüftechnische Entwurfsregeln des Designsystems VENUS

Regel 1:	Jedes Pad der Schaltung muß mit einem Anschluß des Bausteins (Gehäuses) direkt verbunden sein.
Regel 2:	Im Stromlaufplan muß logische und schaltungstechnische Redundanz vermieden werden.
Regel 3:	Ein definierter Prüfanfangszustand der Schaltung muß extern einstellbar sein.
Regel 4:	Erzeugung und Ableitung von Impulsen dürfen nicht durch schaltungsinterne Verzögerungen erfolgen.
Regel 5:	Zur Prüfung der Schaltung sind Takte erforderlich, die im TESTMODE unabhängig voneinander extern einstellbar sind.
Regel 6:	Schaltungen, die Störimpulse auf Taktleitungen erzeugen können, sind unzulässig.
Regel 7:	Takte und Daten müssen in getrennten Pfaden von den Bausteinanschlüssen an die Eingänge der Schaltelemente (Zellen) geführt werden.
Regel 8:	Im TESTMODE dürfen in einem Datenpfad der Schaltung zwischen Eingangs- und Ausgangsanschluß nur maximal sechs speichernde Schaltelemente liegen.
Regel 9:	Der rückgekoppelte Teil eines Datenpfades darf im TESTMODE nur maximal zwei speichernde Schaltelemente einschließen.
Regel 10:	Rückkopplungen in rein kombinatorischen Schaltungsteilen sind nicht zugelassen.

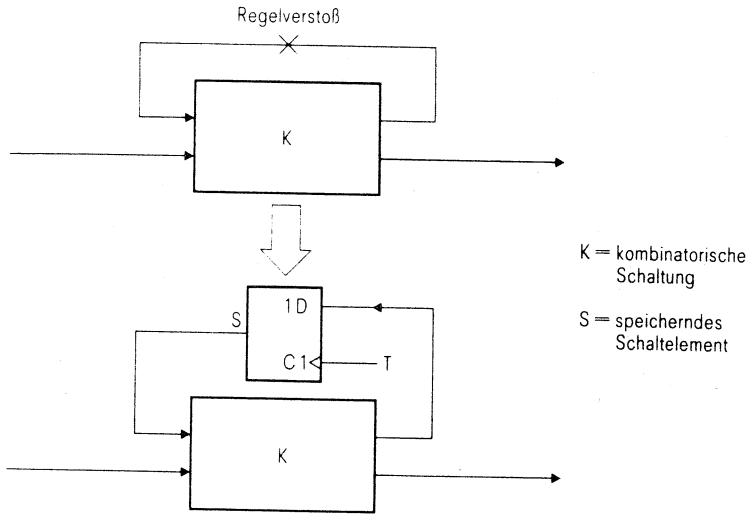


Bild 4.7d. Trennung kombinatorischer Rückkopplungen

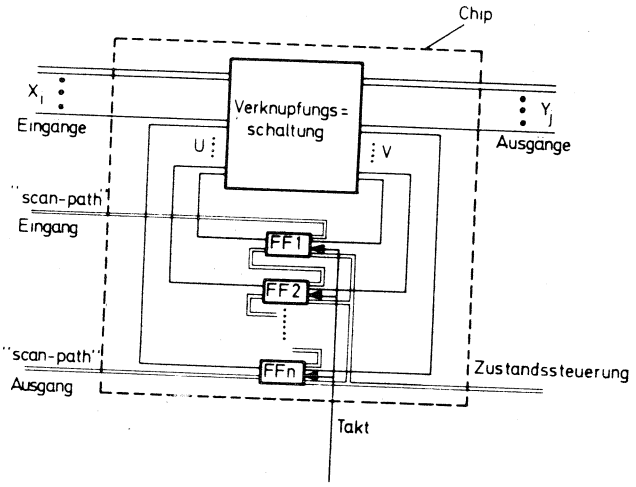


Bild 4.7. Schema eines sequentiellen Systems mit „Scan-Path". [4.10].

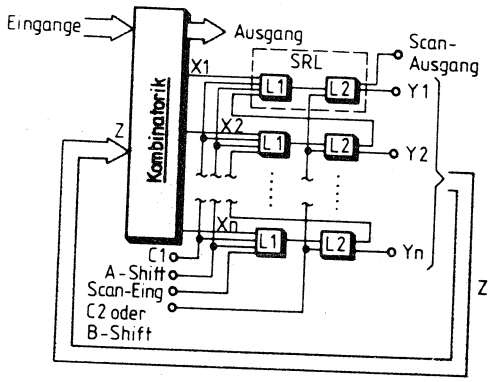


Bild 7. Struktur einer Schaltung mit LSSD, Betrieb mit zwei Takten

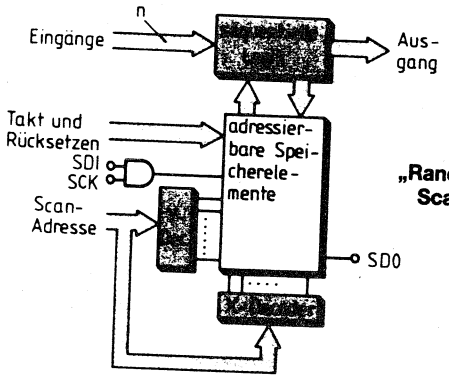


Bild 9. „Random-Access-Scan"-Netzwerk

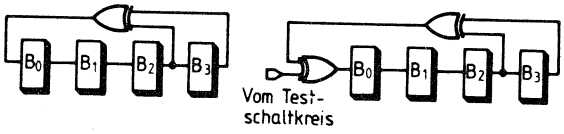
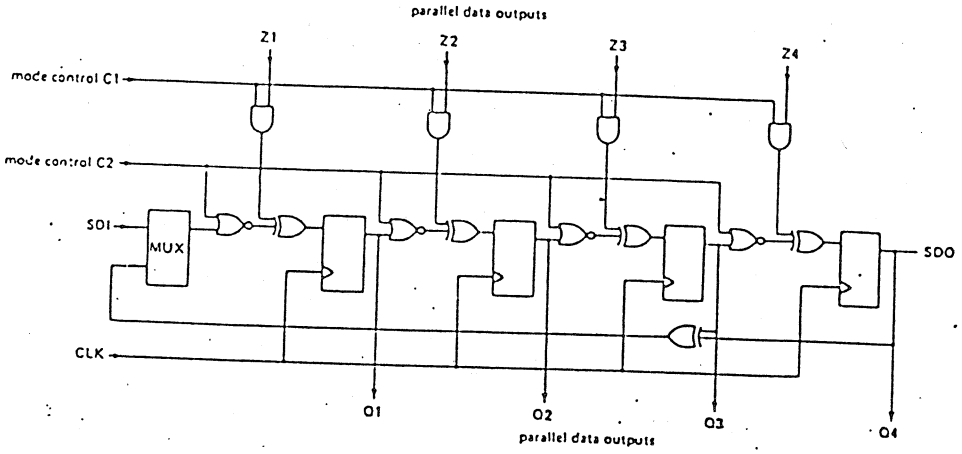


Bild 10. Pseudo-Zufallsgenerator (links) und Signatur-analysator (rechts)



Multifunktionaler Test-Baustein mit 4 Betriebsarten

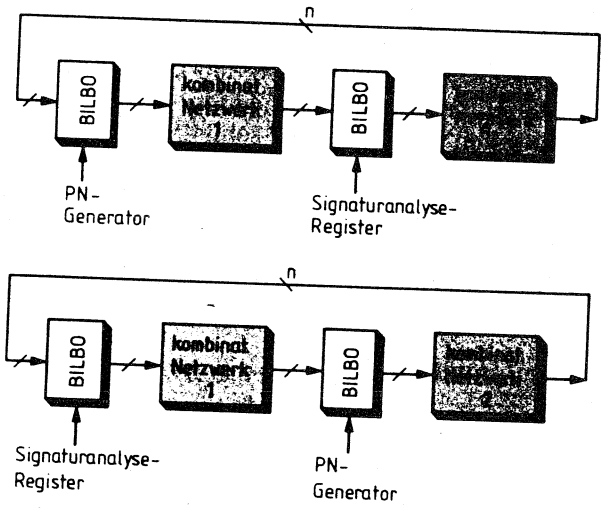
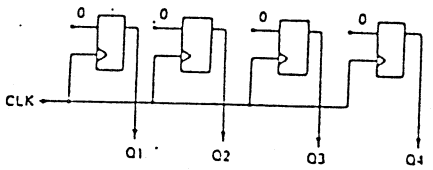
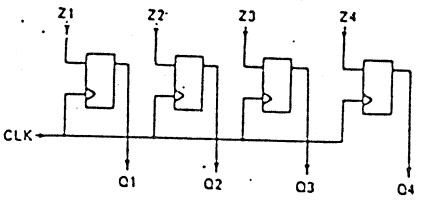


Bild 13. Testkonfiguration eines BILBO

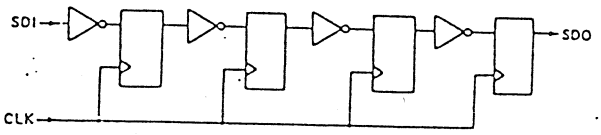
BILBO = Built In Logic Block Observer



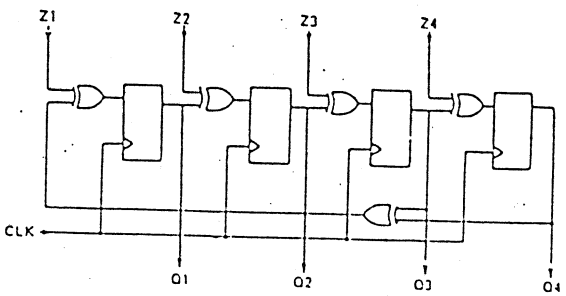
1) BILBO reset mode: C1=0, C2=1.



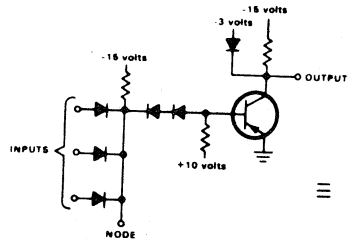
2) BILBO normal latch mode: C1=1, C2=1.



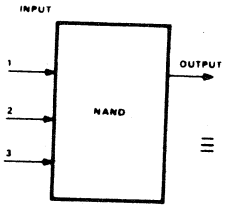
3) BILBO scan path mode: C1=0, C2=0.



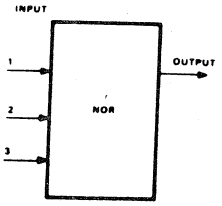
4) BILBO LFSR mode: C1=1, C2=0.



(a) Multiple input inverter circuit.



(b) NAND logic element.



(c) NOR logic element.

Table of Circuit Behavior

Input			
1	2	3	Output
0	0	0	-3
0	0	-3	-3
0	-3	0	-3
0	-3	-3	-3
-3	0	0	-3
-3	0	-3	-3
-3	-3	0	-3
-3	-3	-3	0

Table of NAND Behavior

Input			
1	2	3	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Table of NOR Behavior

Input			
1	2	3	Output
1	1	1	0
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	1

Fig. 11. PDP-8 combinational circuit and logic diagram.

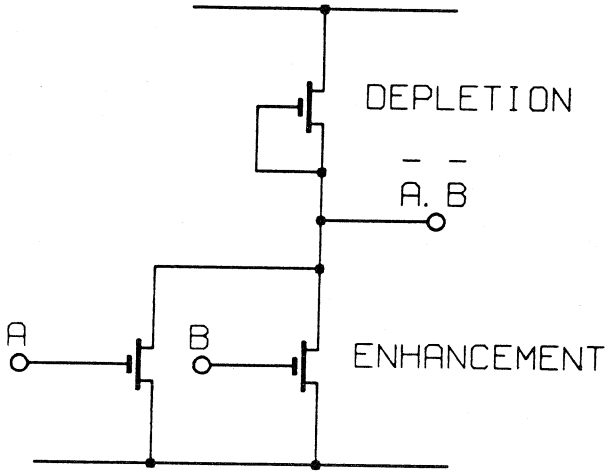


Fig.6.11 The basic NMOS static NOR gate

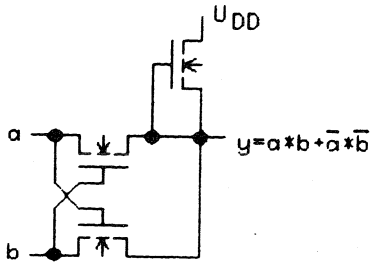


Bild 6.10 ÄQUIVALENZ-Schaltung in NMOS-Technik

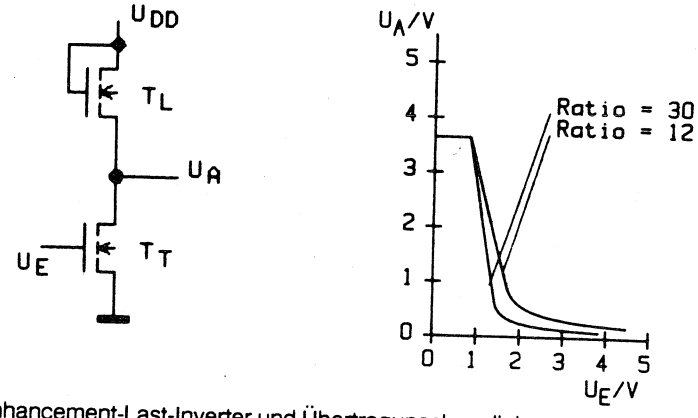


Bild 4.4 Enhancement-Last-Inverter und Übertragungskennlinie

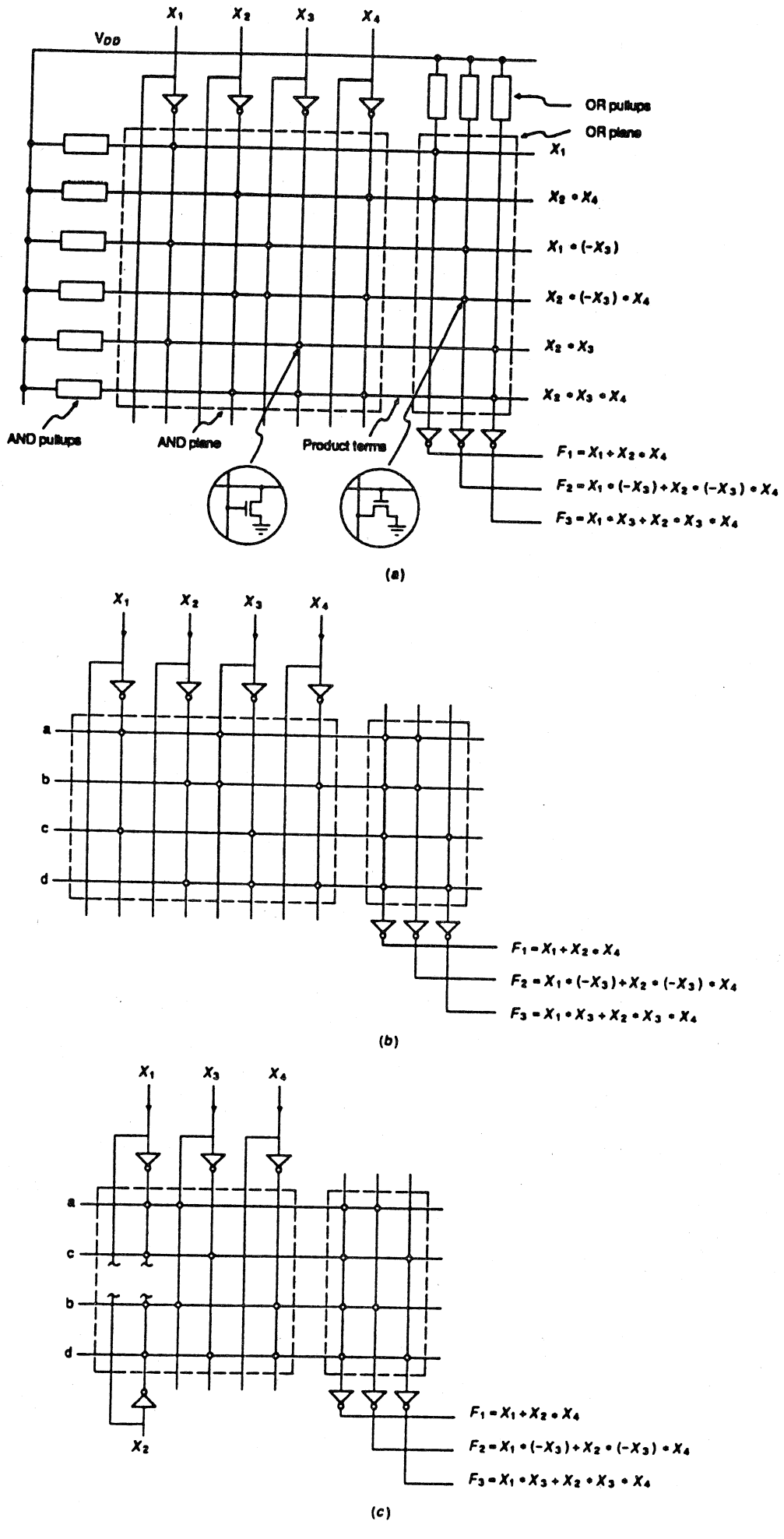
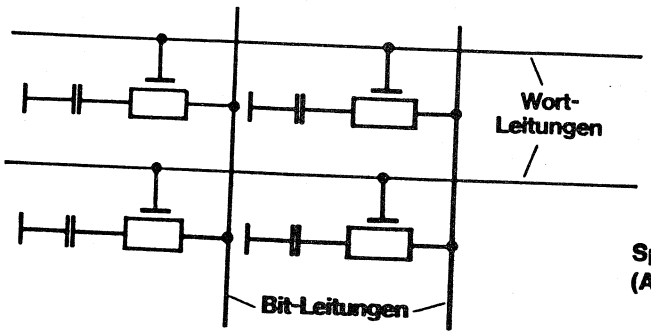
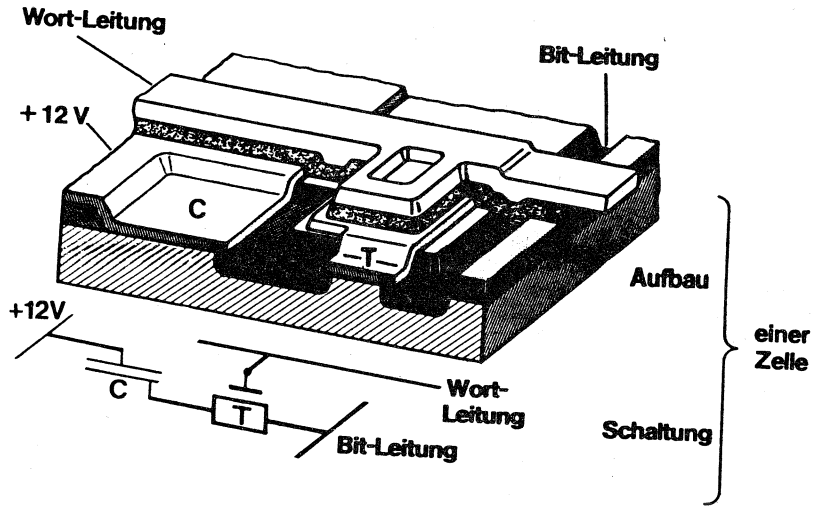


Figure 1.9 PLA layout architecture. (a) Nonminimized. (b) Minimized. (c) Folded.





Ein-Transistor-Speicherzelle

H.J. Harloff,
Phys. Bl. 32, 119 (76)

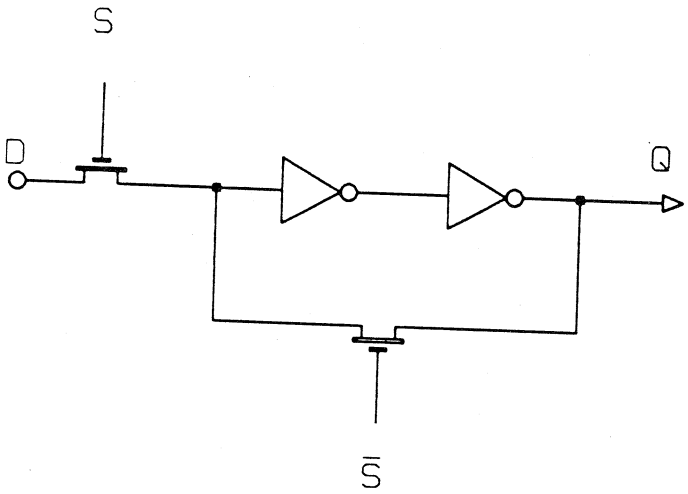


Fig.6.12 Using pass transistors to form a simple data latch in NMOS technology

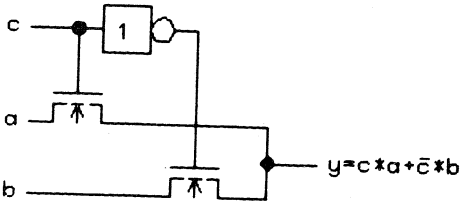
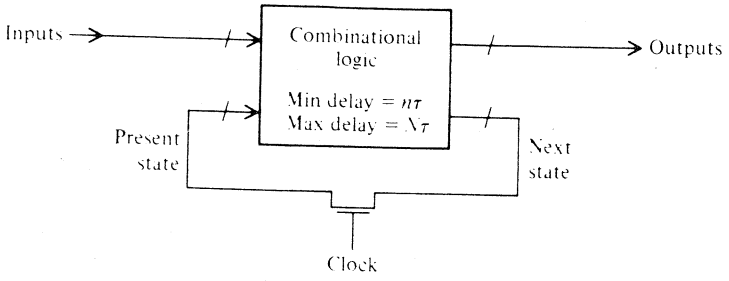
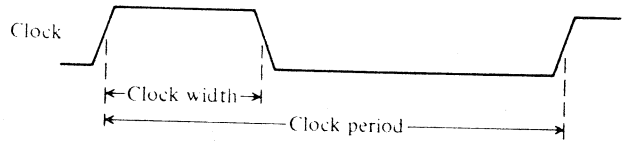


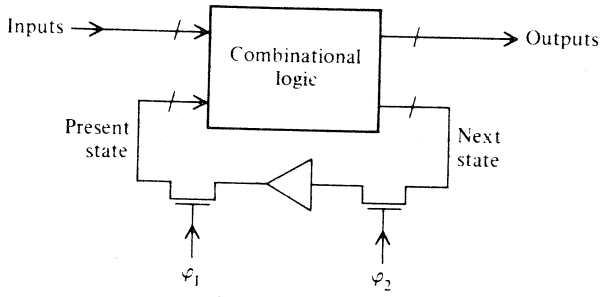
Bild 6.7 2-Bit Multiplexer in NMOS-Technik





$(R_{on}C_{in})_{max} <$ clock width $< n\tau$
$N\tau <$ clock period $<$ refresh period

Fig. 7.4 Cheap, fast, and risky clocking scheme. R_{on} is the effective on resistance of the pass transistor, and C_{in} is the input capacitance of the combinational logic.



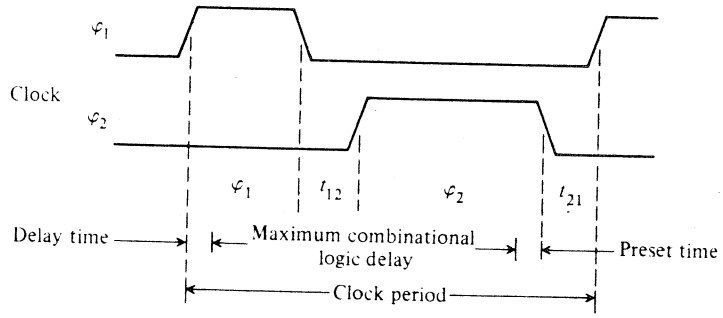
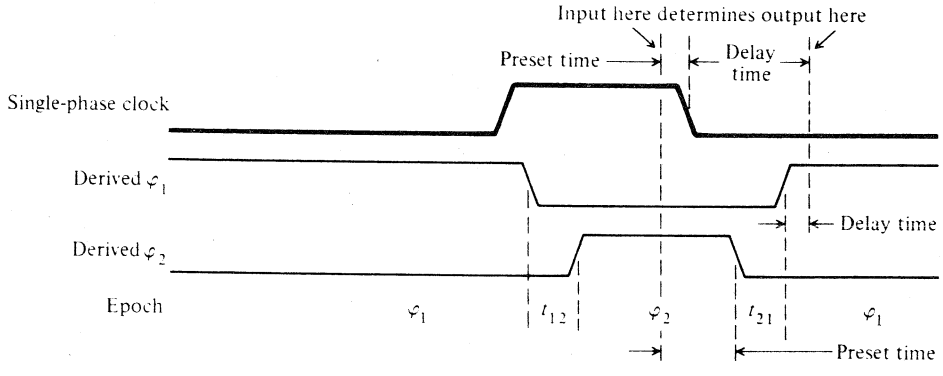
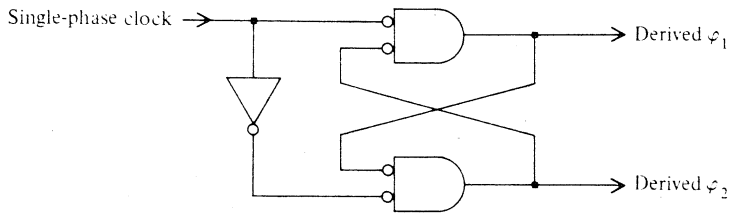


Fig. 7.5 Two-phase clocking.

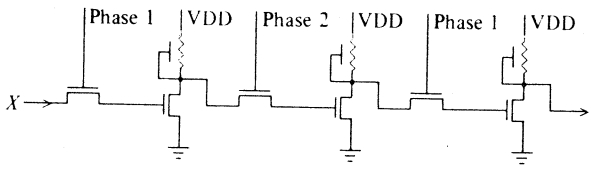


(a)

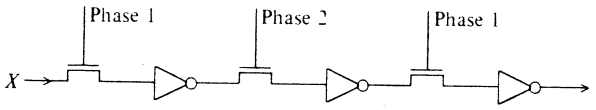


(b)

Fig. 7.6 (a) Relationship between a single-phase clock and a two-phase clock. (b) Logic circuit to derive a two-phase nonoverlapping clock from a single-phase clock.



(a)



(b)

Fig. 3.4 (a) Shift register circuit diagram. (b) Shift register in mixed notation.

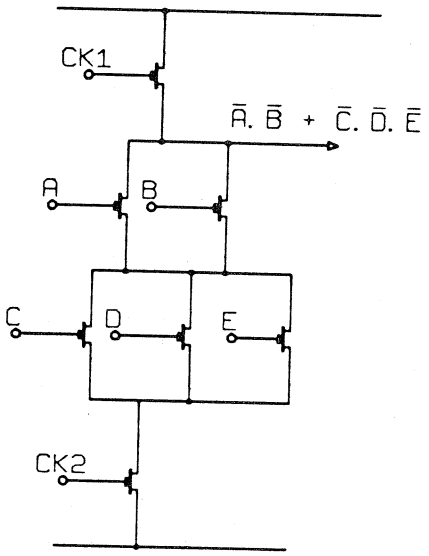


Fig.6.13 An example of an NMOS complex gate using dynamic logic

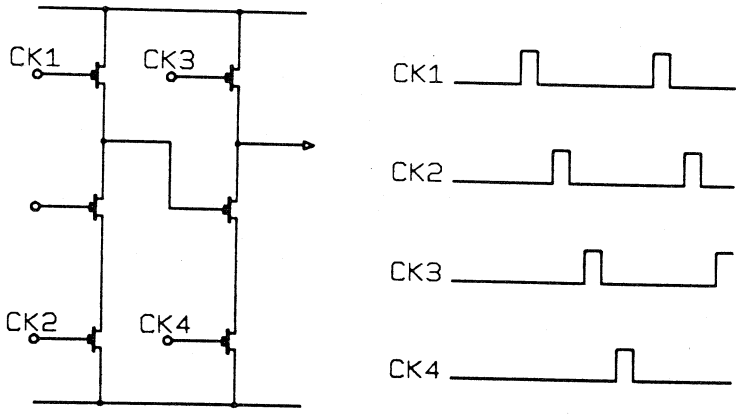


Fig.6.14 A single stage of a 4-phase dynamic shift register

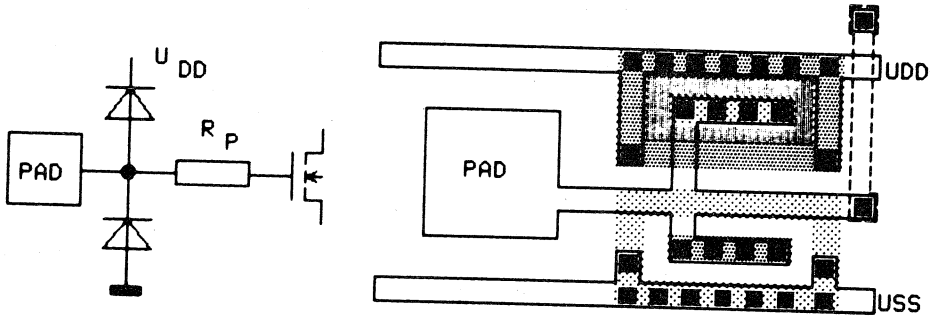


Bild 12.3 Schutzstruktur mit Dioden

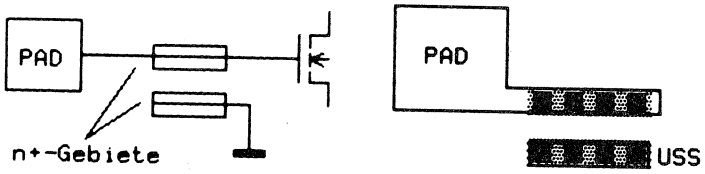


Bild 12.2 Schutzstruktur auf der Basis des Punch-Through-Effekts

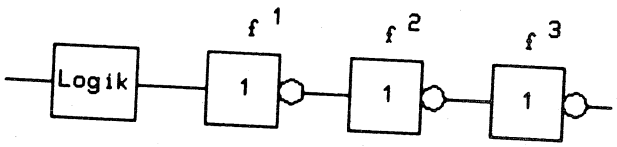


Bild 12.4 Kaskadierung von Inverterstufen zur Optimierung der Treiberfunktion

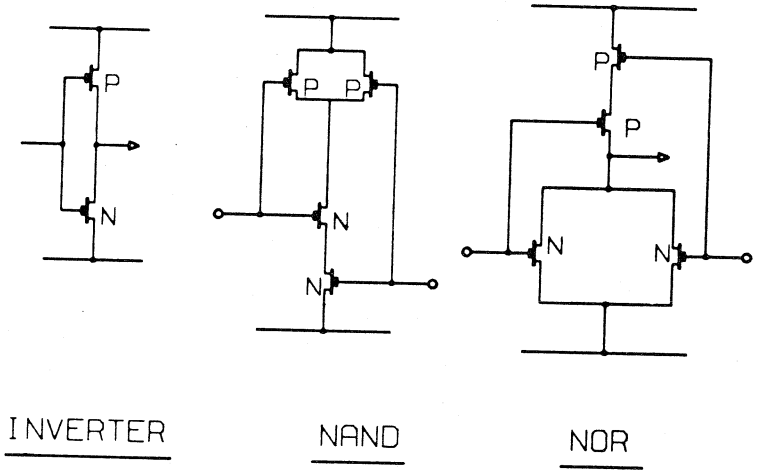


Fig.6.15 A selection of simple CMOS logic functions

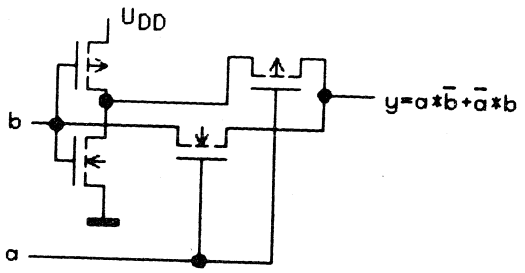
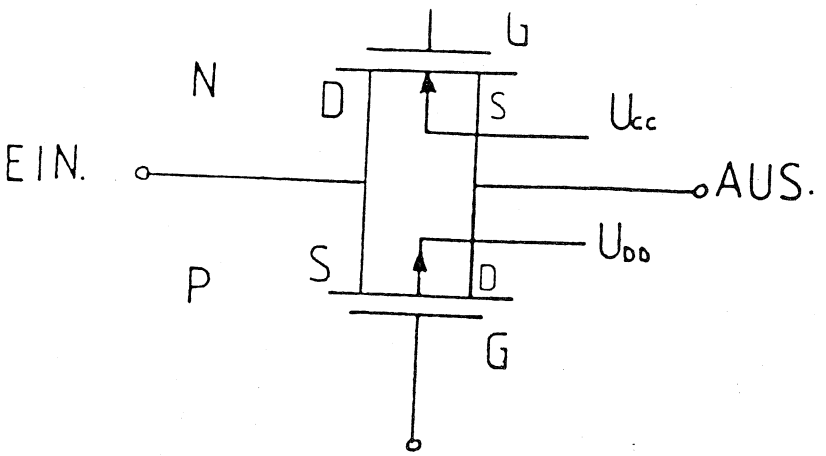


Bild 6.8 XOR-Gatter in CMOS-Technik



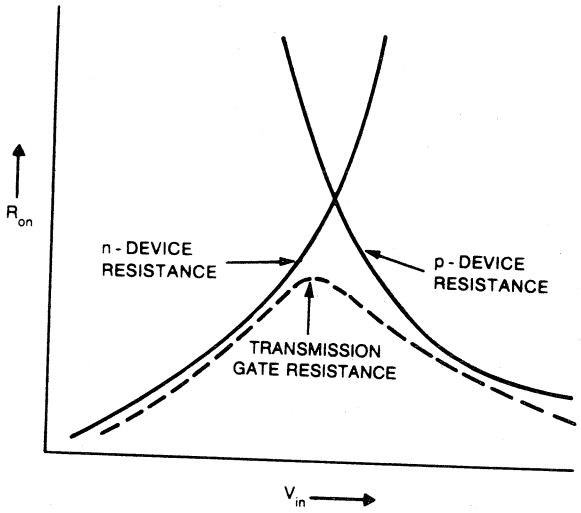
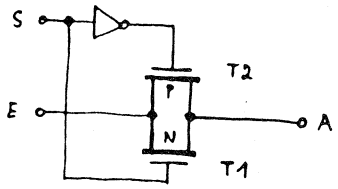


FIGURE 2.24. Transmission gate output characteristic

CMOS transmission gate



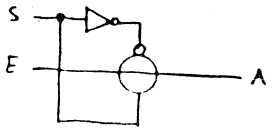
Bei 1-Potential an S leiten beide Transistoren; bei 0-Potential sind beide gesperrt:

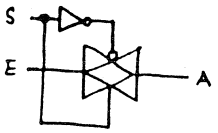
S = 1 : E ——— A

S = 0 : E —/— A

⇒ bilateraler (bidirektionaler) Schalter

Schaltsymbol:



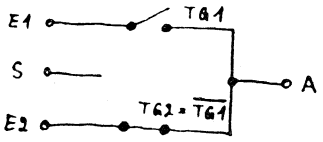


Vereinfachung im Schaltbild:

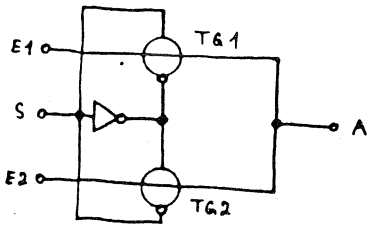
Durch die Verwendung von TG's können bei Flipflops und komplexeren Schaltungen einige Transistoren eingespart werden. (Table 1)

2-Weg-Multiplexer:

Prinzip:



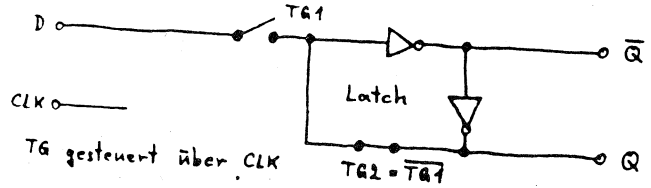
Realisierung:



Mit dem 2-Weg-Multiplexer lassen sich Zähler, Schieberegister, usw. wesentlich einfacher aufbauen als mit herkömmlichen Gattern.

Grund-FF zum Aufbau komplexerer FF's in CMOS-Gate-Arrays.

Prinzip:



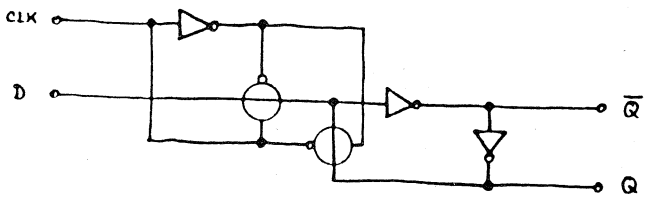
TG gesteuert über CLK

Annahme: 1 gespeichert → Q = 1, $\overline{Q}$ = 0, CLK = 0, D beliebig

– D = 0, CLK = 1 → TG1 schließt, TG2 = $\overline{TG1}$ öffnet → $\overline{Q}$ = 1, Q = 0

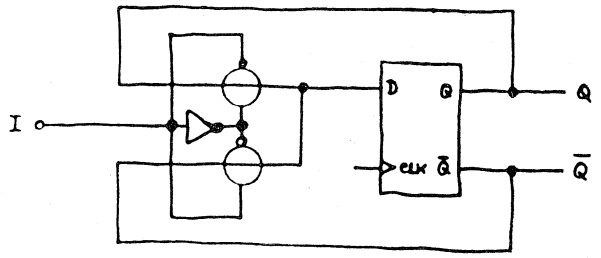
– CLK = 0 → TG1 öffnet, TG2 schließt → Q = 0, $\overline{Q}$ = 1 ⇒ 0 gespeichert

Realisierung:



Nur 1 Inverter für TG-Ansteuerung notwendig!

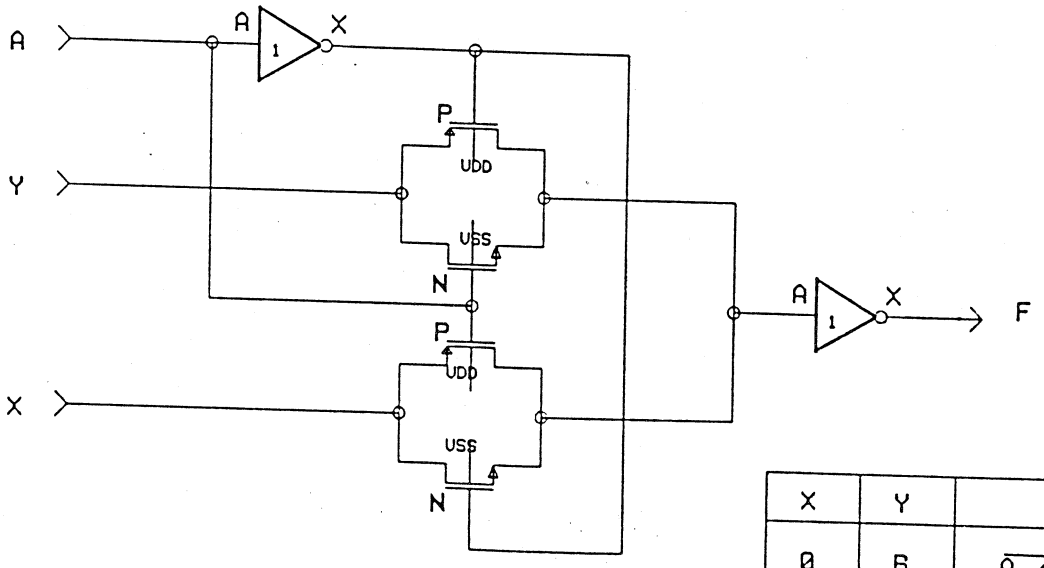
d) Toggle-FF: T-FF (Figure 3)



Multiplexer schaltet wahlweise Q oder $\overline{Q}$ auf den D-Eingang:

I = 0: Q ist auf den D-Eingang geschaltet
→ FF schaltet beim nächsten CLK nicht
I = 1: $\overline{Q}$ ist auf den D-Eingang geschaltet

CMOS UNIVERSALGATTER



X	Y	F
0	B	$\overline{A \wedge B}$
1	$\overline{B}$	$A \wedge B$
B	1	$\overline{A \vee B}$
$\overline{B}$	0	$A \vee B$
B	$\overline{B}$	$\overline{A \oplus B}$
$\overline{B}$	B	$A \oplus B$

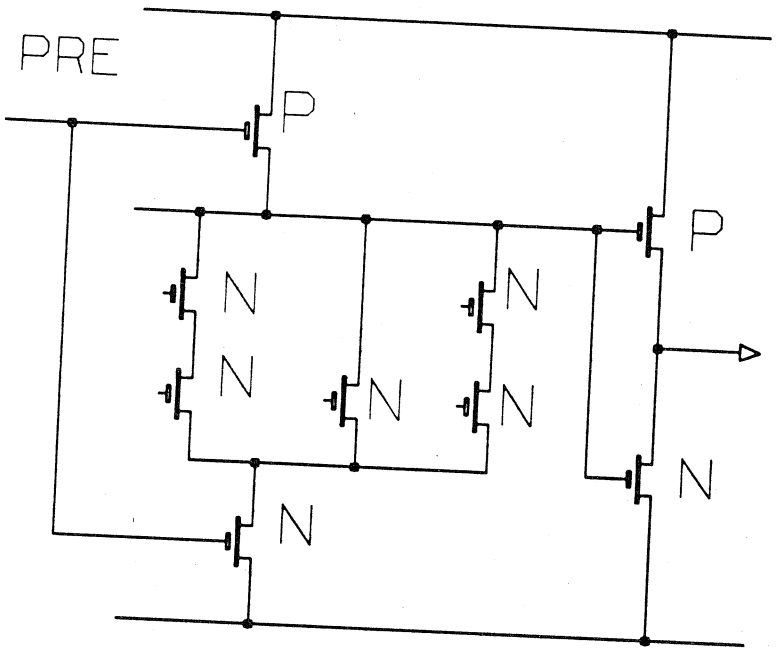
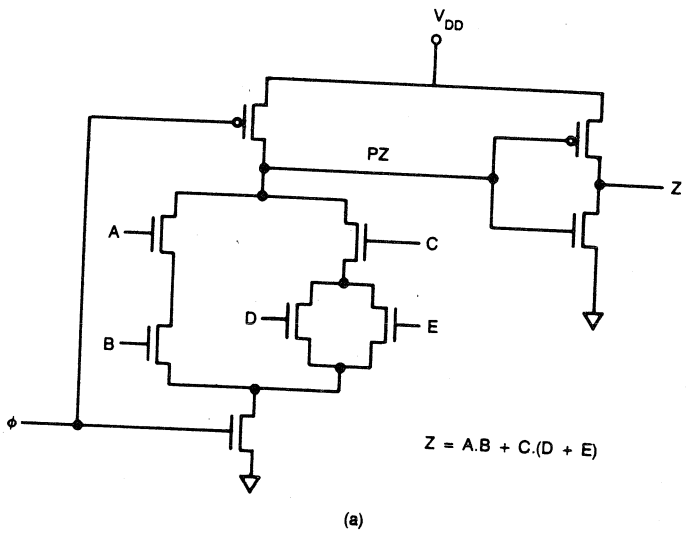


Fig.6.16 'Domino' logic utilises a pre-charging technique



(a)

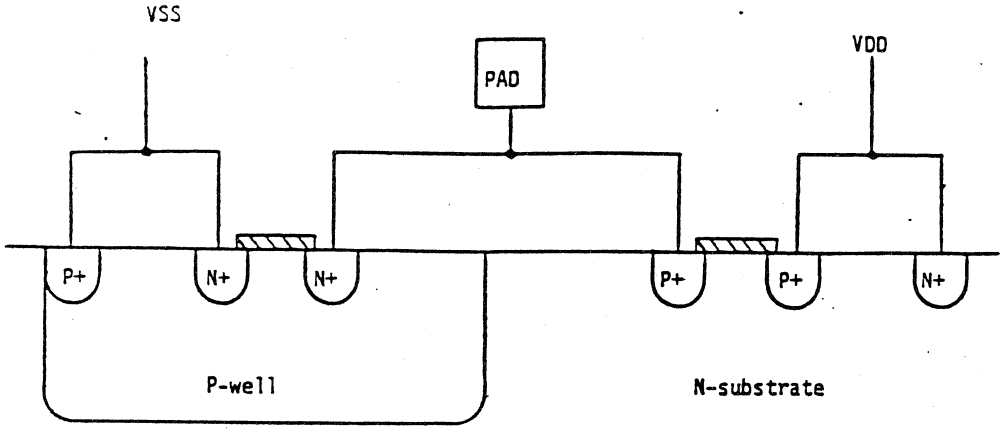


Figure 2.9 (a)

Simplified cross-section of an unprotected CMOS output buffer.

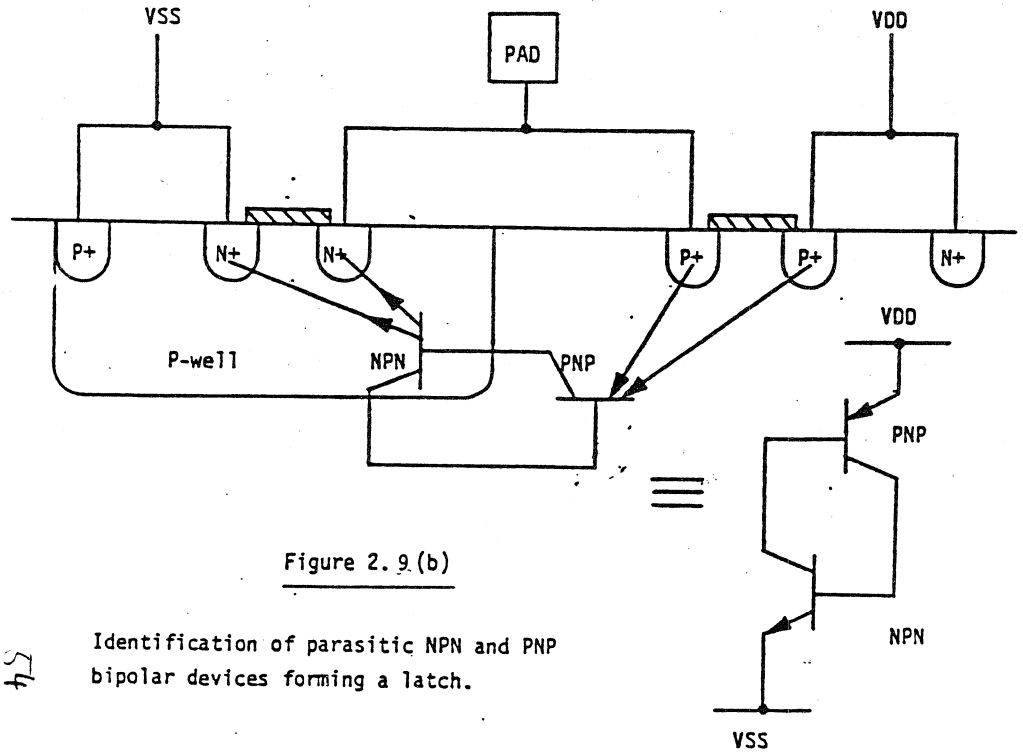


Figure 2.9 (b)

Identification of parasitic NPN and PNP bipolar devices forming a latch.

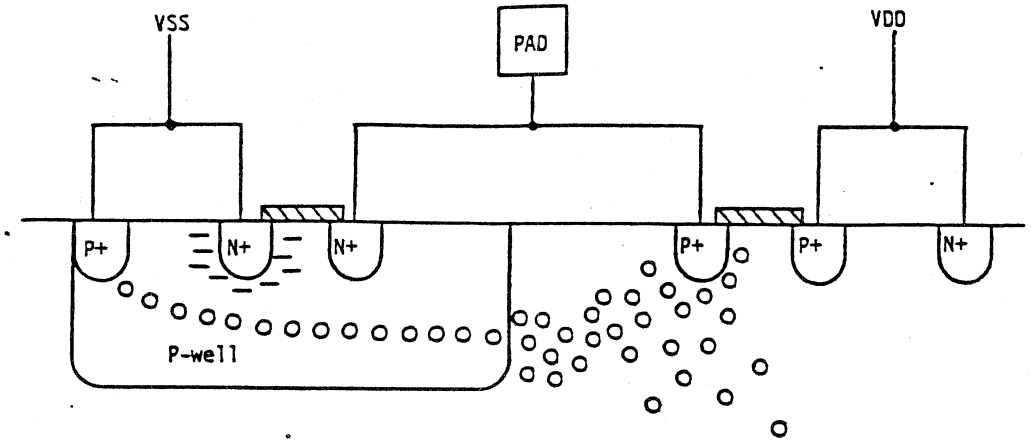


Figure 2.10(a)

Positive spike on the bonding pad which goes above VDD injects holes into the substrate. These propagate into the P-well by transistor action.

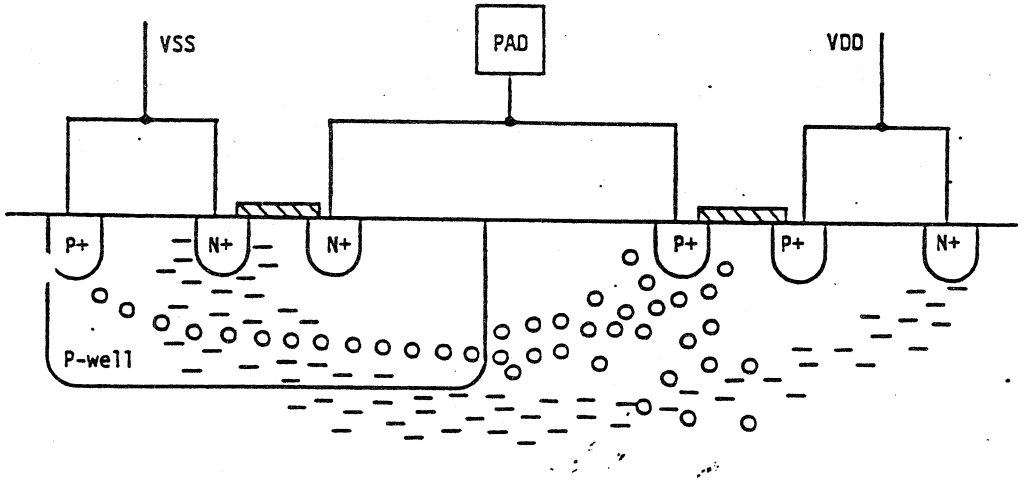


Figure 2.10(b)

Due to the resistance of the P-well, the voltage of the well rises underneath the N+ diffusions. Electrons are injected into the P-well. These propagate into the N-substrate.

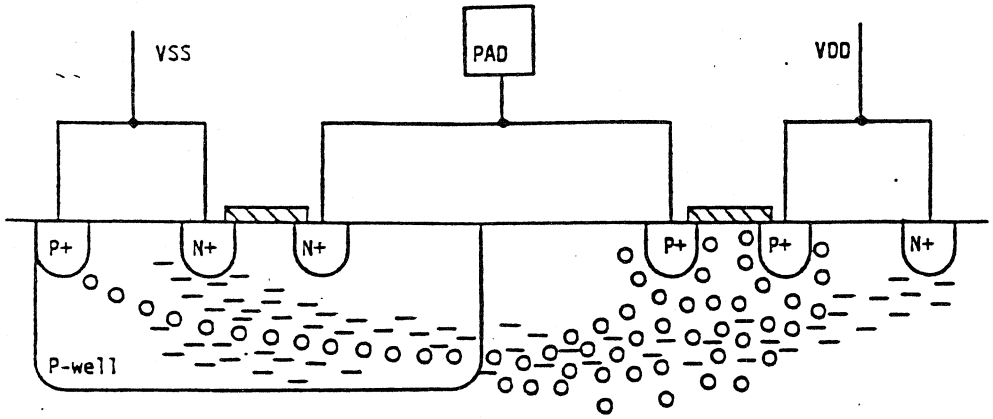


Figure 2.10(c)

Due to substrate resistance, the voltage underneath the P+ diffusions will fall, causing more holes to be injected from P+ diffusions NOT connected to the pad.

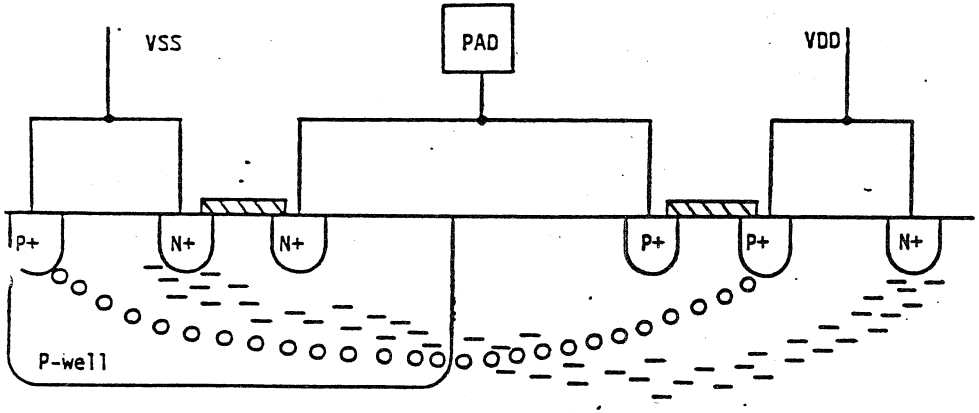


Figure 2.10(d)

If the disturbance on the pad is removed,a current still flows between VDD and VSS. This current is self sustaining as long as the circuit is powered up. This problem is known as latch-up.

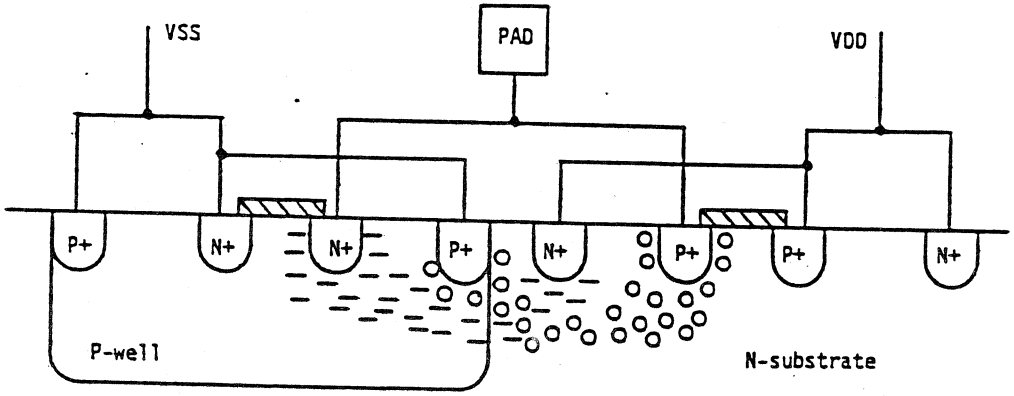
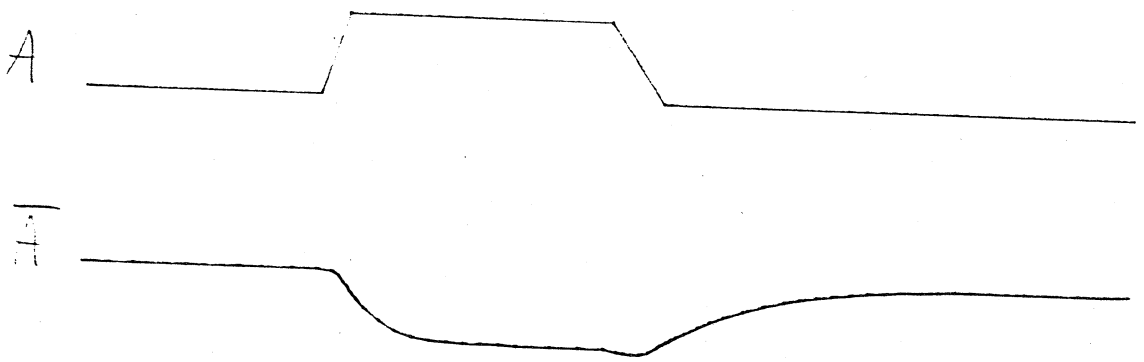


Figure 2.11

Diffusion guardbars are used to reduce the collector resistance of the parasitic bipolar devices and collect injected carriers before they cause latch-up. As shown in the figure, the protection covers both positive and negative overdrives on the pad.

```
; NMOS-Inverter

global 0

model pullup nmos vto=-2.5v kp=10u
model pulldn nmos vto=0.8v  kp=20u

t1 vdd out out 0  pullup w=5 l=10
t2 out in 0 0     pulldn w=10 l=5
c1 out 0          c c=0.05pf

v vdd 0 dc v=5.0
vin in 0 pwl t0=0 v0=0.0 t1=45ns v1=0.0 t2=50ns v2=5.0 t3=195ns v3=5.0 \
 t4=205ns v4=0.0

relax2 options stop=250ns

plot in out
```



NMOS - Inverter
Relax - Simulation

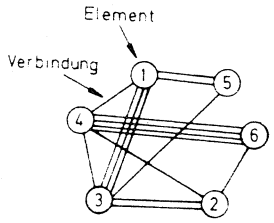


Bild 5.8. Beispiel eines Systems aus 6 miteinander verbundenen Elementen. [5.3].

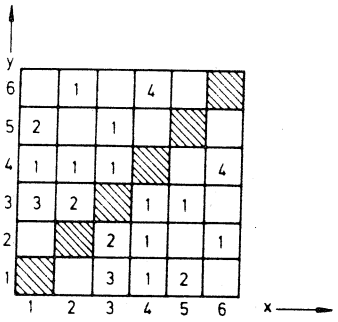


Bild 5.9. Verbindungsmatrix zu dem Beispiel im vorigen Bild. [5.3].

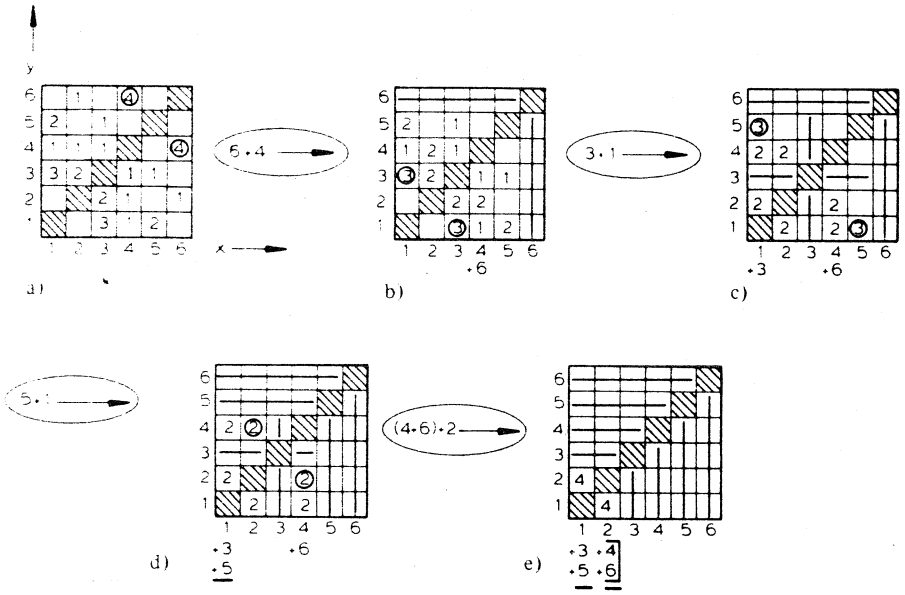


Bild 5.10. Schrittweise Durchführung von Verschmelzungsvorgängen in der Verbindungsmatrix (Addition von Zeilen und Spalten).

a) Wahl der Elemente 6 und 4.
b) Durchführen der Additionen, Streichen der Spalte und Zeile 6 und Wahl der Elemente 3 und 1.
c) Durchführen der Additionen, Streichen der Spalte und Zeile 3 und Wahl der Elemente 5 und 1.
d) Durchführen der Additionen und Streichen der Spalte und Zeile 5. Der erste Modul ist vollständig. Wahl der Elemente (4 + 6) und 2.
e) Durchführen der Additionen und Streichen der Spalte und Zeile 4. Auch der zweite Modul ist vollständig.

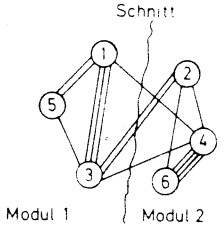


Bild 5.11. Graphische Darstellung des partitionierten Systems. [5.3].

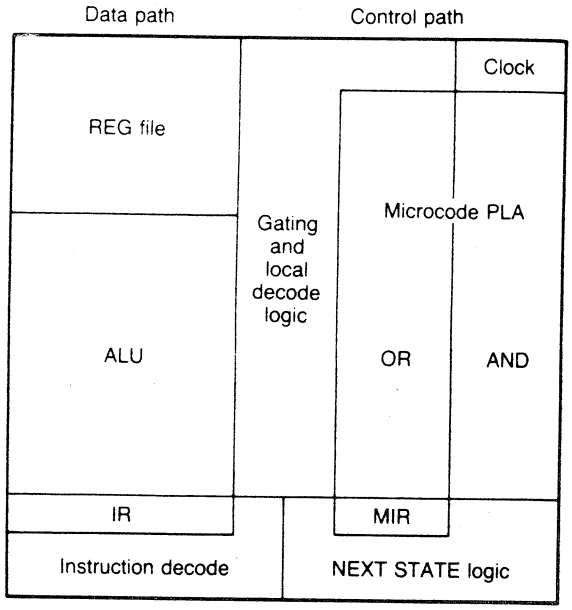
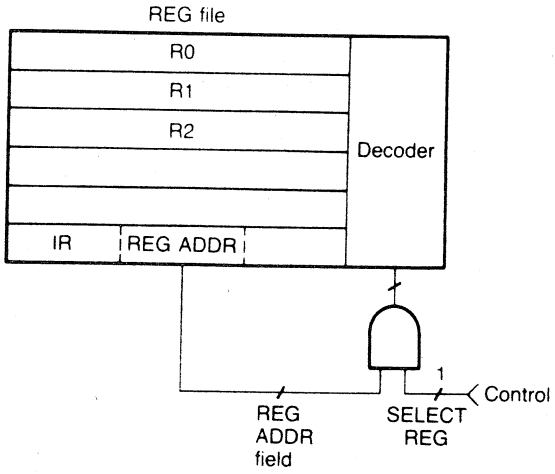
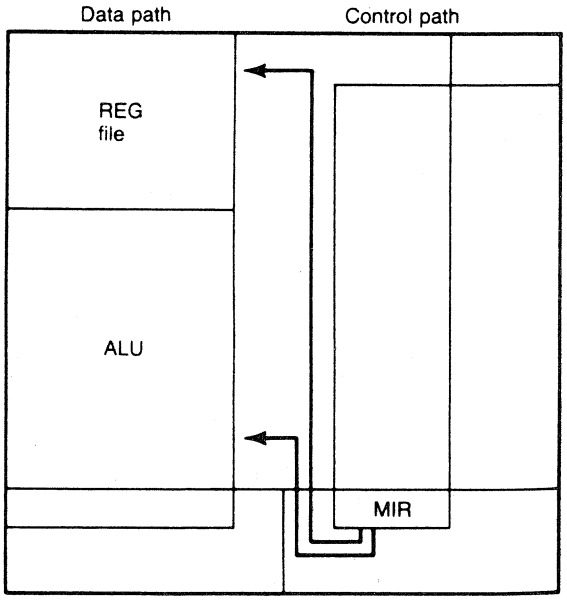


FIGURE 5.70 Floor plan of a simple microprocessor.

FIGURE 5.69 Register file with decoder. The selection of the register number is made locally.





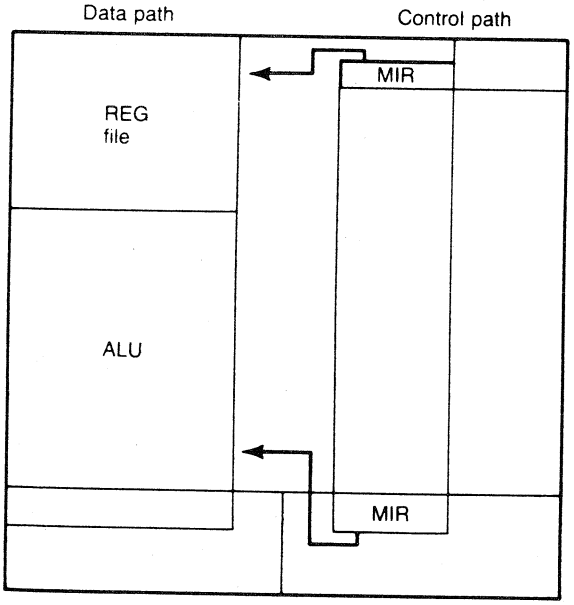


FIGURE 5.72 A better routing of the data-path control wires with a redundant MIR.

10mm connection - metal

λ	5μ	2.5μ	0.5μ	0.1μ
R	25Ω	100Ω	2.5K	62.5K
C	4pF	4pF	4pF	4pF
RC	100pS	400pS	10nS	250nS
Gate	5nS	2.5nS	0.5nS	0.1nS
	←		→	
	Performance dominated by circuits		Performance dominated by connections	

(Polysilicon connections have resistance approximately 100 x R)

Fig.12.1 Interconnection performance

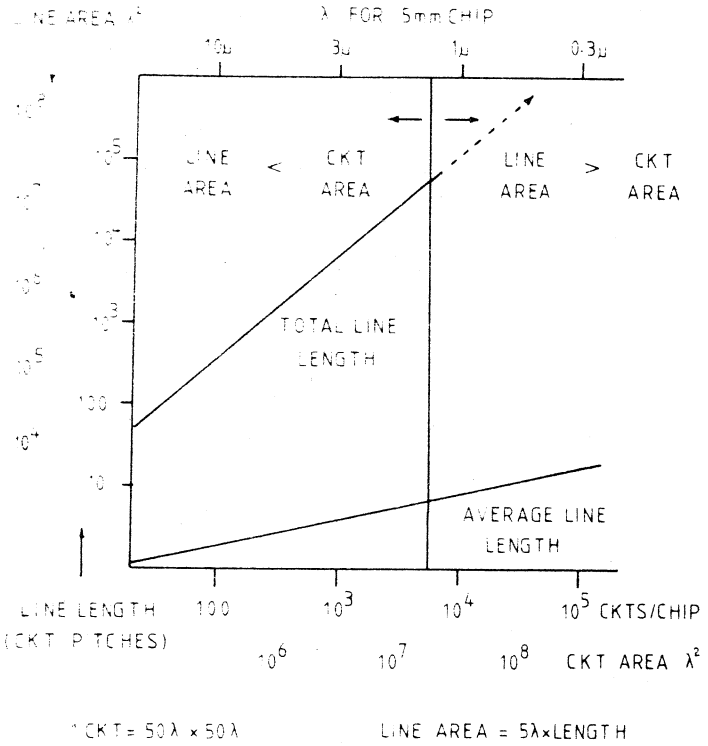


Fig.12.2 Relative connection area
λ = minimum feature size

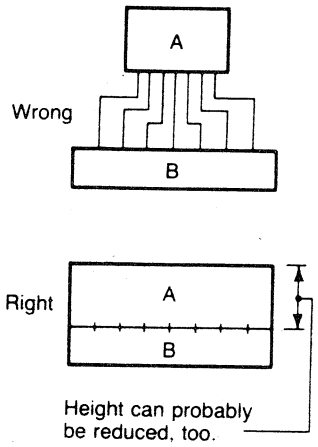
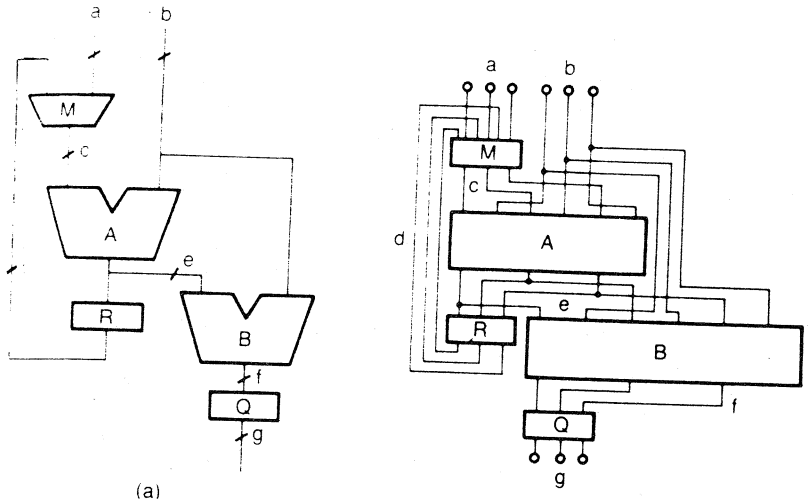


FIGURE 5.75 The usually wrong and usually right ways of performing intermodule wiring.



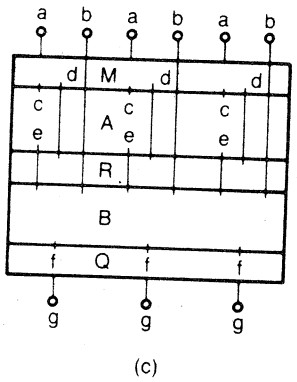
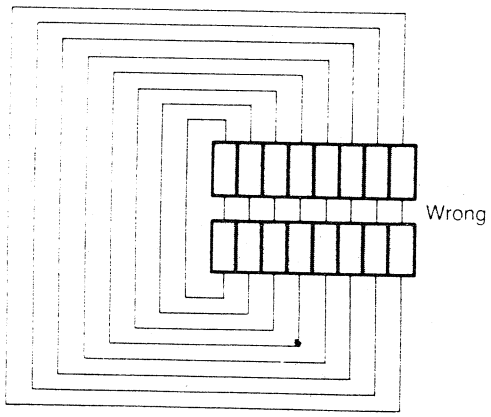


FIGURE 5.74 Transferring a block diagram into VLSI: (a) the block diagram, (b) transliteration to obtain floor plan, (c) VLSI-style floor plan.



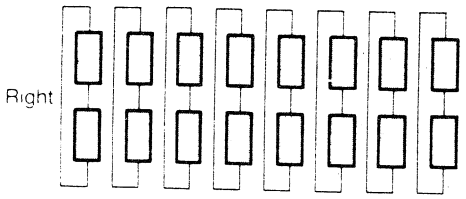
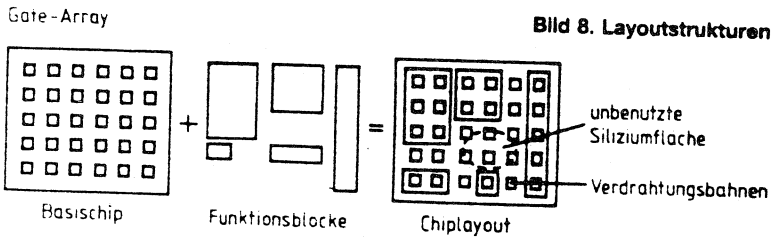


FIGURE 5.76 The usually wrong and usually right ways of adding feedback loops to intermodule wiring.



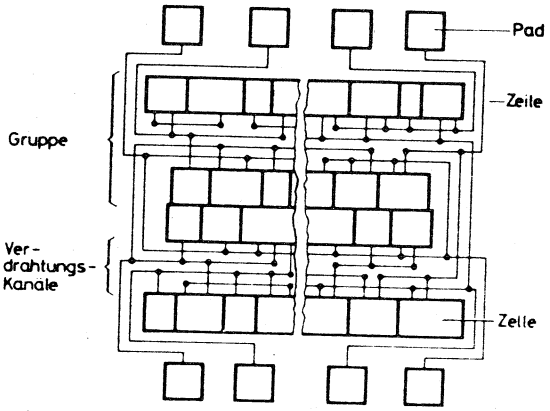


Bild 8.35. Standardzellenschaltungen. [8.26].
a) Prinzip.

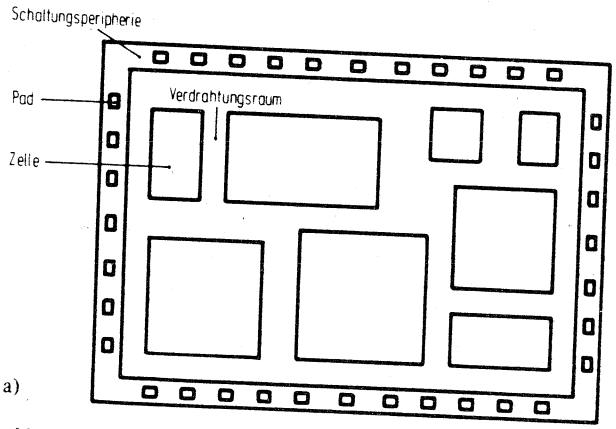


Bild 8.33. Der Makrozellenentwurf.
a) Schema eines allgemeinen Makrozellenentwurfs. [8.26].

MINCUT HEURISTIC

An efficient heuristic for partitioning graphs has been given by Kernighan & Lin.

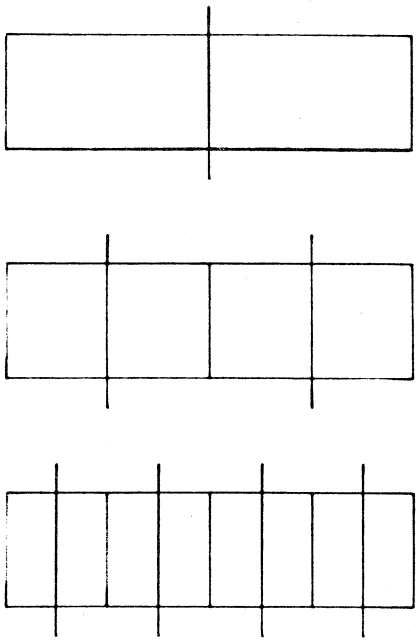
Algorithm:

1. Divide the nodes into 2 arbitrary sets
2. Exchange a pair af nodes that reduces the number of common edges most.
3. Repeat 2. until no improvement is possible.

The best pair can be found efficiently by recording the possible improvements for all pairs, and updating them after each swap.

USING MINCUT

Linear placement can be done by recursively applying mincut.
Divide modules in two sets, minimizing mutual connections. Divide each set again, until each set contains 1 module.



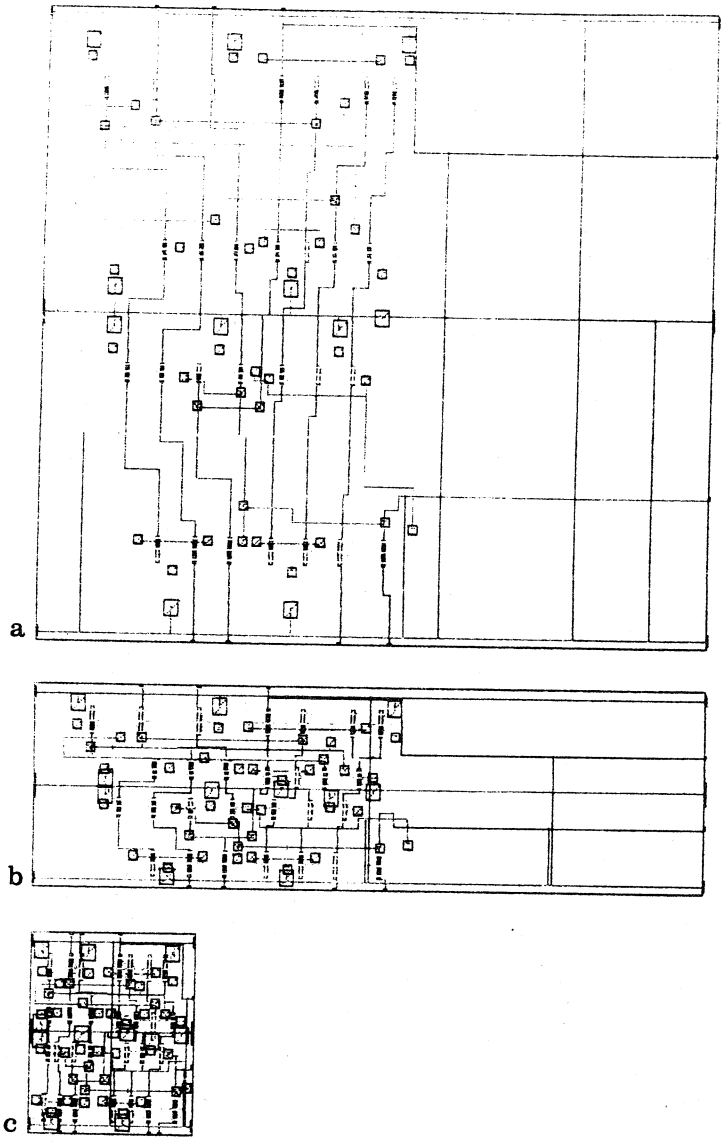


Bild 3.23. Stick-Diagramm (a) mit vertikaler (b) und zusätzlich horizontaler (c) Kompaktierung

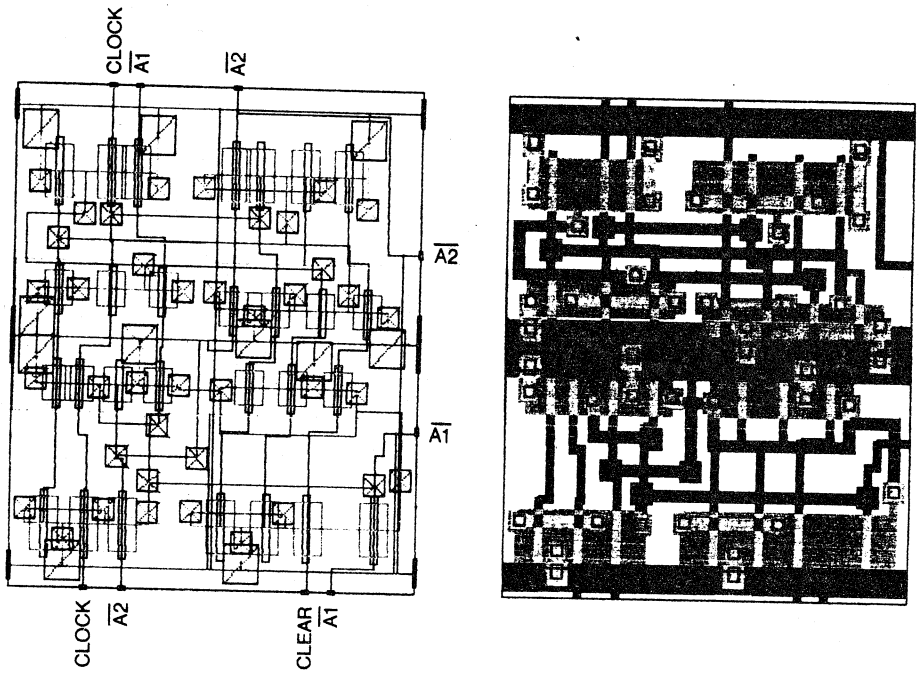


Bild 3.24. Vergrößerte Darstellung des kompaktierten Stick-Diagramms und daraus generiertes Layout

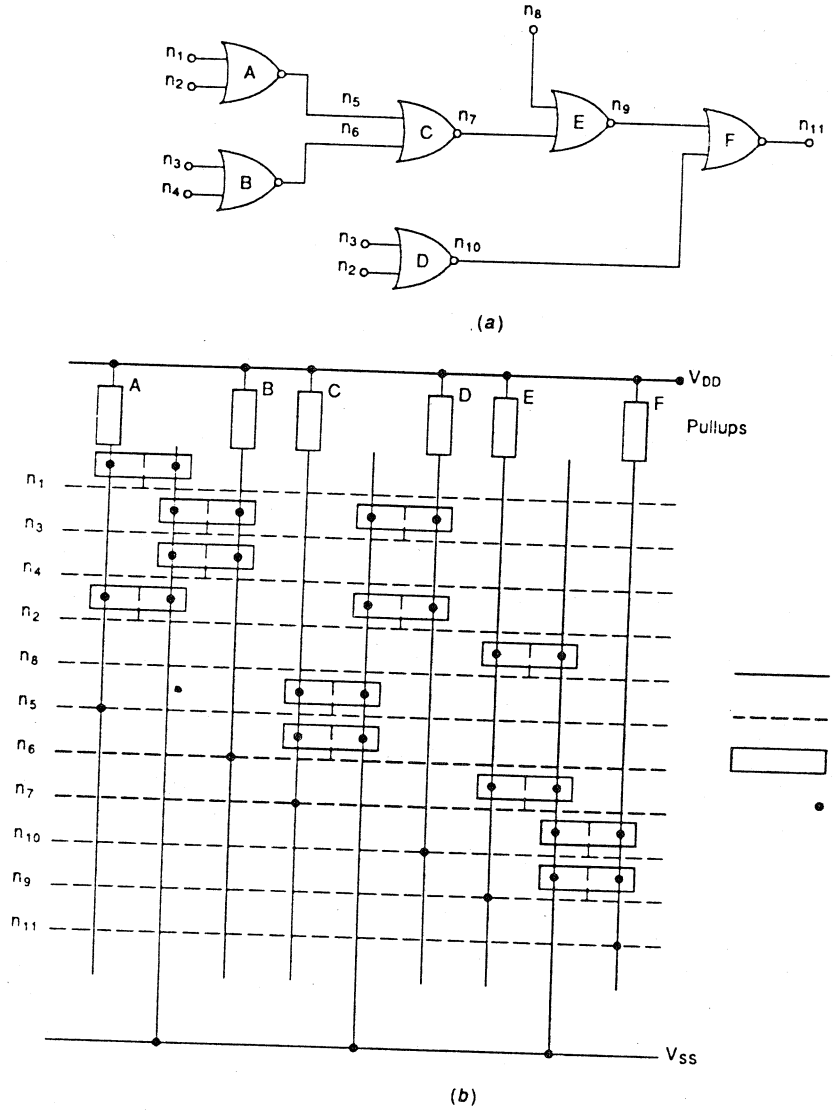


Figure 1.4 Weinberger array architecture. (a) Schematic. (b) Layout.

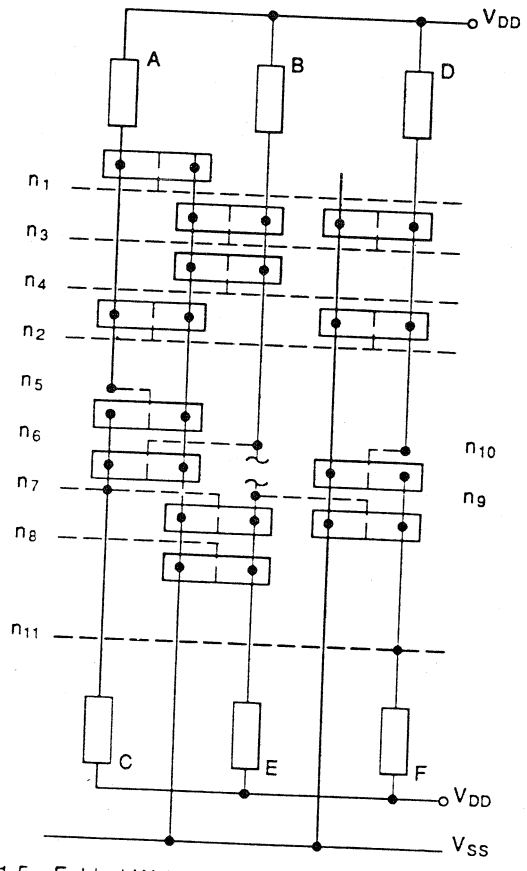


Figure 1.5 Folded Weinberger array of Figure 1.4.

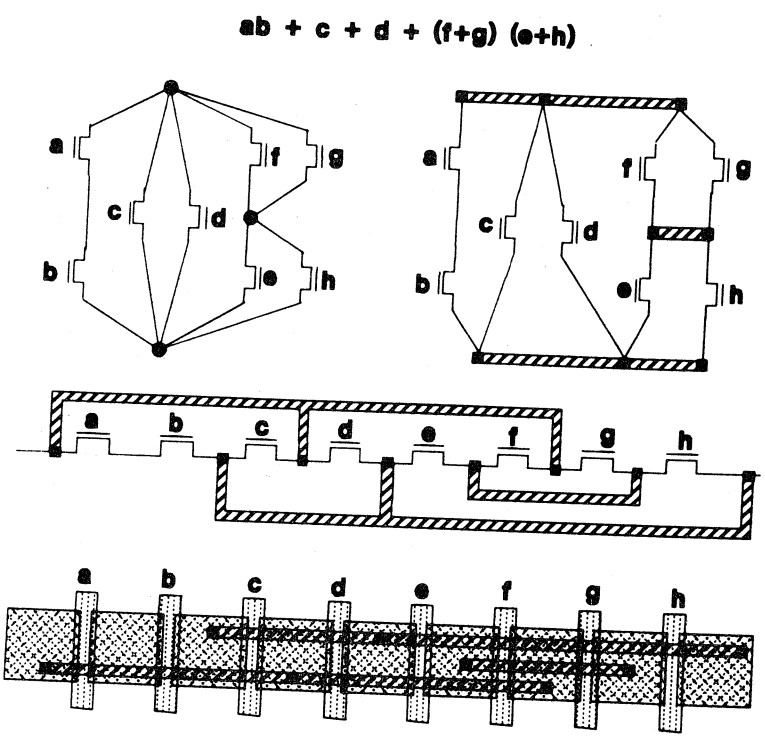
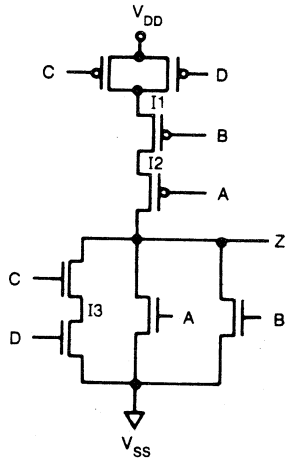
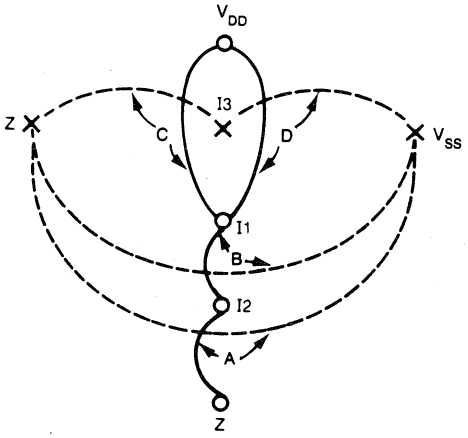


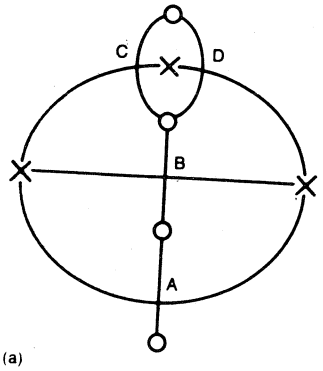
Figure 7.16 Translating an AND-OR switching function in a linear transistor array.



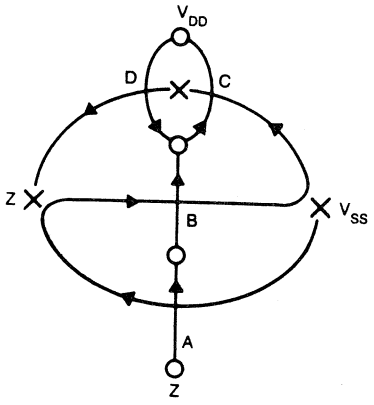
(a)

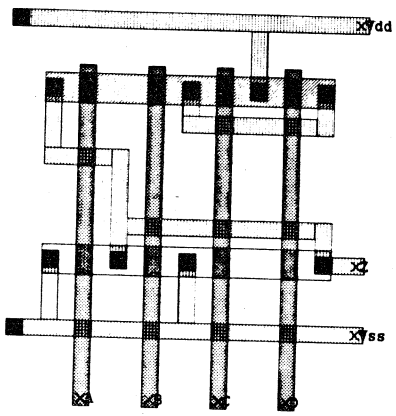


(b)



(a)





(b)

Command	Form
Polygon with a path	P path
Box with length, width, center, and direction (direction defaults to (1,0) if omitted)	B integer integer point point
Round flash with diameter and center	R integer point
Wire with width and path	W integer path
Layer specification	L shortname
Start symbol definition with index, *a*, *b* (*a* and *b* both default to 1 if omitted)	DS integer integer integer
Finish symbol definition	DF
Delete symbol definitions	DD integer
Call symbol	C integer transformation
User extension	digit userText
Comments with arbitrary text	(commentText)
End marker	E

```
cifFile          = { { blank } { command } semi } endCommand { blank }.
command          = primCommand | defDeleteCommand |
                   defStart Command semi { { blank } { primCommand } semi } defFinishCommand.
primCommand      = polygonCommand | boxCommand | roundFlashCommand | wireCommand |
                   layerCommand | callCommand | userExtensionCommand | commentCommand.
polygonCommand   = "P" path.
boxCommand       = "B" integer sep integer sep point [ sep point ].
roundFlashCommand = "R" integer sep point.
wireCommand      = "W" integer sep path.
layerCommand     = "L" [ blank ] shortname.
defStartCommand  = "D" [ blank ] "S" integer [ sep integer sep integer ].
defFinishCommand = "D" [ blank ] "F".
defDeleteCommand = "D" [ blank ] "D" integer.
callCommand      = "C" integer transformation.
userExtensionCommand = digit userText.
commentCommand   = "(" comment Text ")".
endCommand       = "E".
transformation   = { [ blank ] ( "T" point | "M" [ blank ] "X" | "M" [ blank ] "Y" | "R" point) }.

path             = point { sep point }.
point            = sInteger sep sInteger.

sInteger         = [ sep ] [ "-" ] integerD.
integer          = [ sep ] integerD.
integerD         = digit { digit }.

shortname        = c [ c ] [ c ] { c }.
c                = digit | upperChar.
userText         = { userChar }.
commentText      = { commentChar } | commentText "(" commentText ")" commentText.

semi             = [ blank ] ":" [ blank ].
sep              = upperChar | blank.
digit            = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
upperChar        = "A" | "B" | "C" | . . . | "Z".
blank            = any ASCII character except digit, upperChar, "-", "(", ")", or ":".
userChar         = any ASCII character except ":".
commentChar      = any ASCII character except "(" or ")".
```

October 1984

3u Single-Metal CMOS
Standard Cell

Description:

AØØ55 is an AND-OR circuit which consists of one 3-input AND gate and one 2-input AND gate into a 2-input OR gate.

Area = 2Ø.9 Sq. Mils (9Øu x 15Øu)
Eq. Gates = 3.5
BOLT Syntax: Q .AØØ55 A B C D E ;

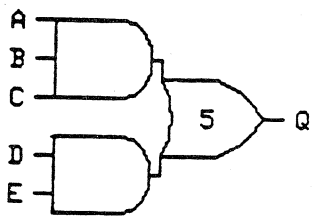
Truth Table:

A	B	C	D	E	Q
Ø	X	X	Ø	X	Ø
X	Ø	X	Ø	X	Ø
X	X	Ø	Ø	X	Ø
Ø	X	X	X	Ø	Ø
X	Ø	X	X	Ø	Ø
X	X	Ø	X	Ø	Ø
1	1	1	X	X	1
X	X	X	1	1	1

Input Capacitance:

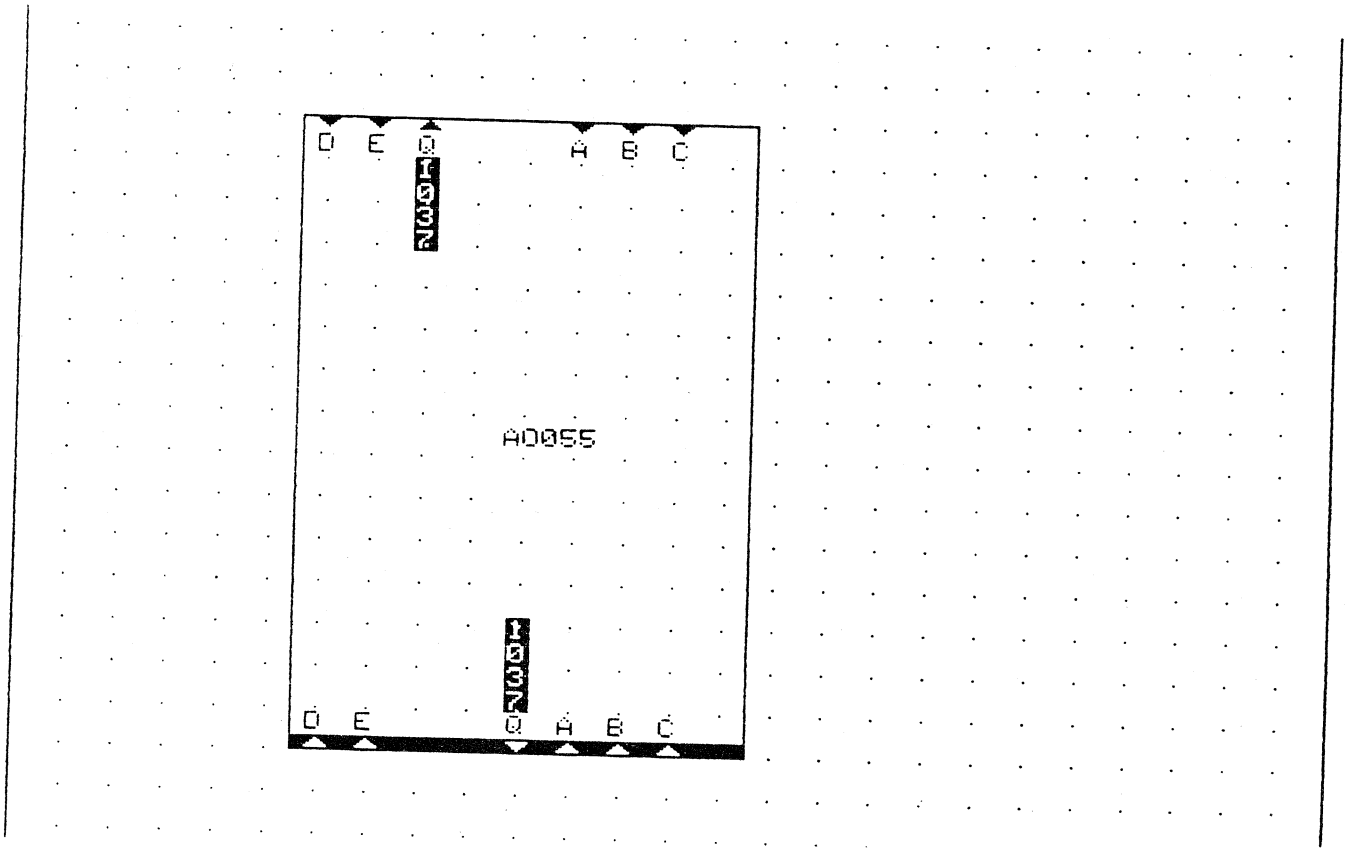
	Ci (pF)
A	Ø.13
B	Ø.13
C	Ø.13
D	Ø.1Ø
E	Ø.1Ø

Logic Symbol:



Characteristics: t(CL) = tdx + ktdx*CL Tj = 25C VDD = 5.ØV Nominal Process

Characteristic	Symbol	tdx (ns)	ktdx (ns/pF)	t(1.ØpF) (ns)
Max Input to Q Delay	tpd	1.7	2.7	4.4



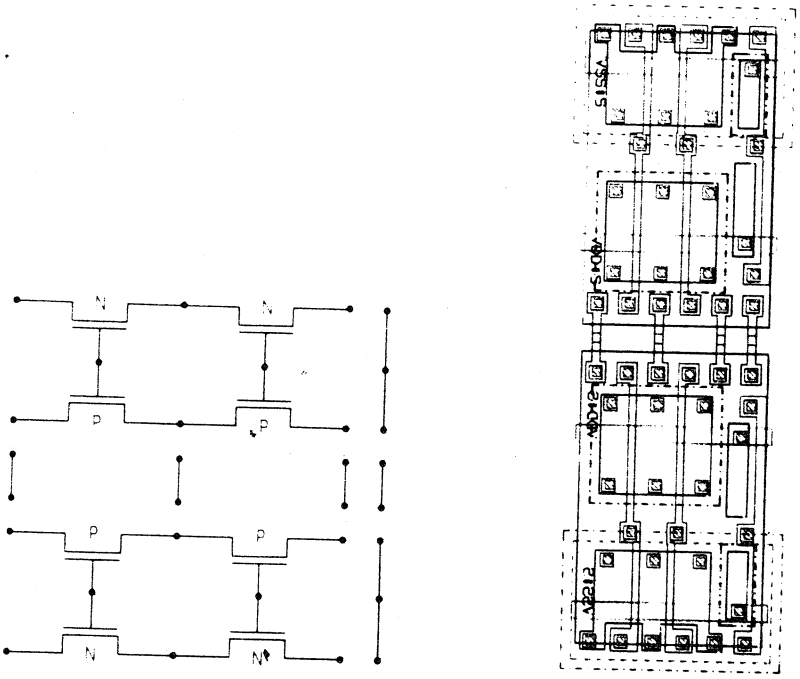


Bild 3.2. Schaltbild und Layout einer CMOS-Gate-Array-Doppelgrundzelle aus vier Transistorpaaren einschließlich Polysiliziumdurchführungen

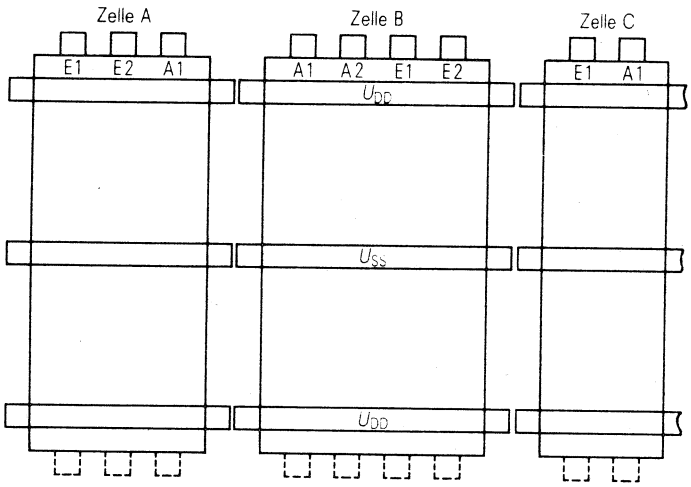


Bild 3.7. Zellenschema von VENUS-Standardzellen

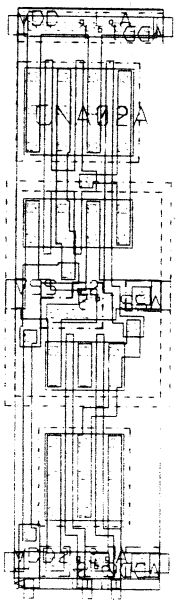


Bild 3.8. 2-input-NAND-Standardzelle

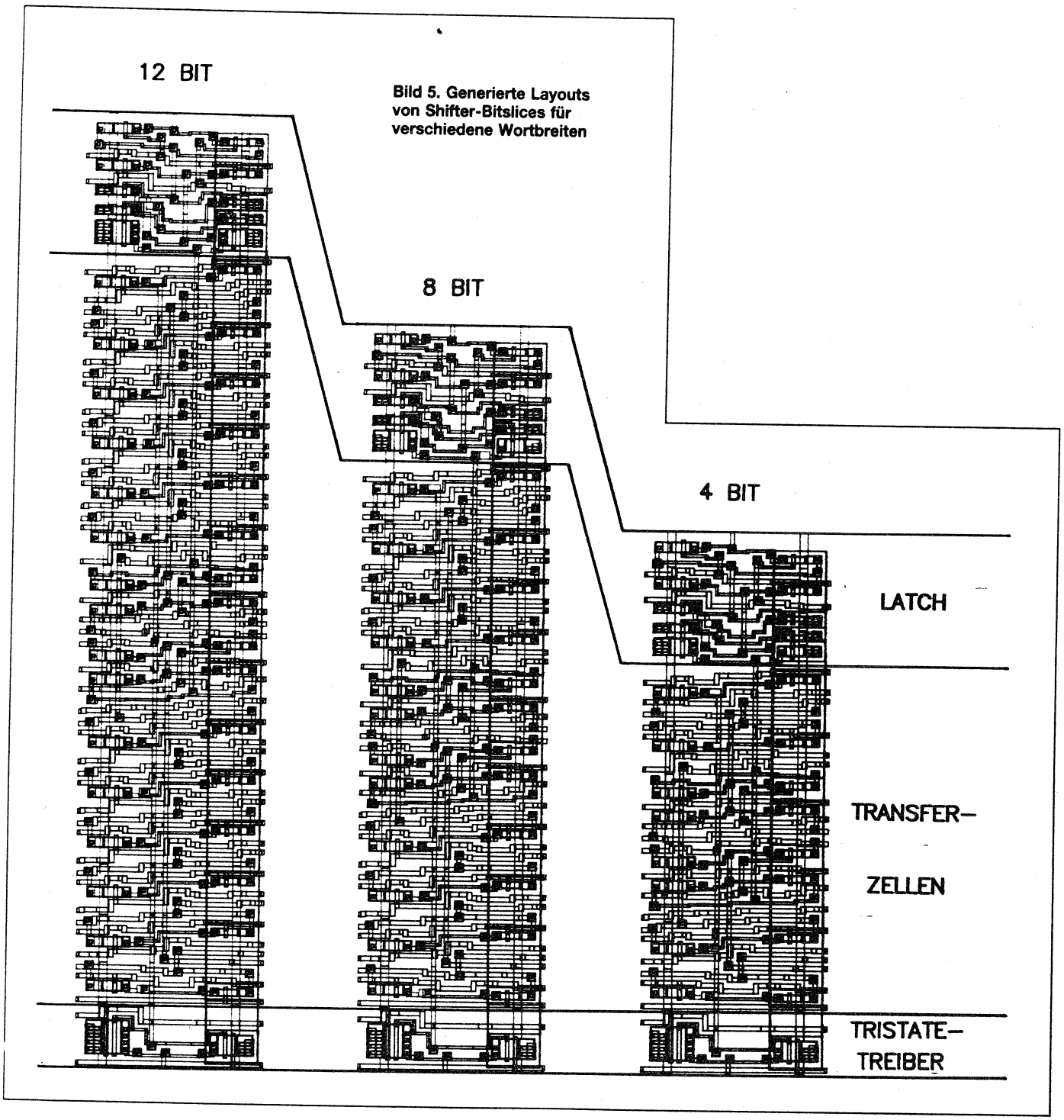


Bild 5. Generierte Layouts von Shifter-Bitslices für verschiedene Wortbreiten

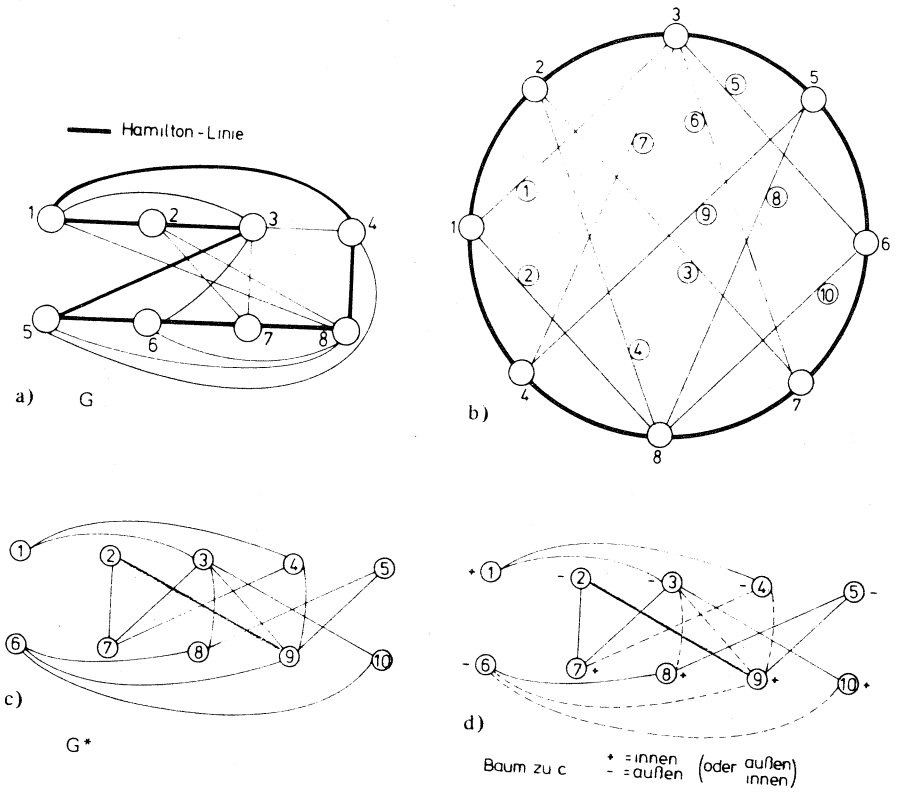
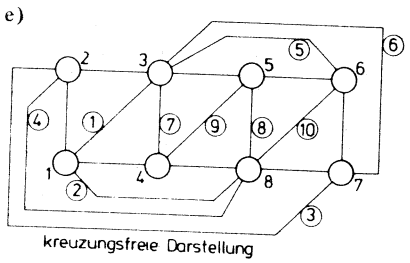
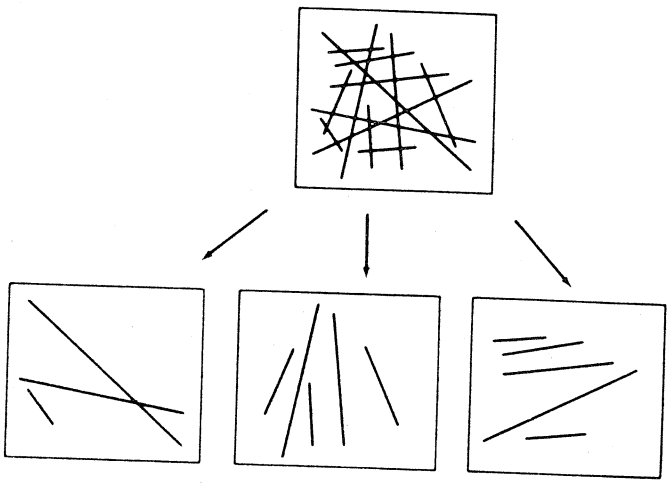
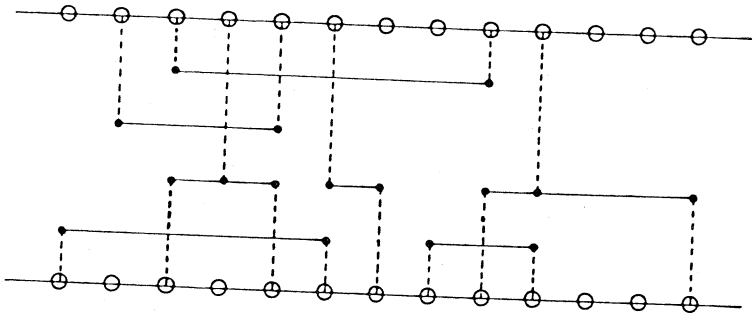


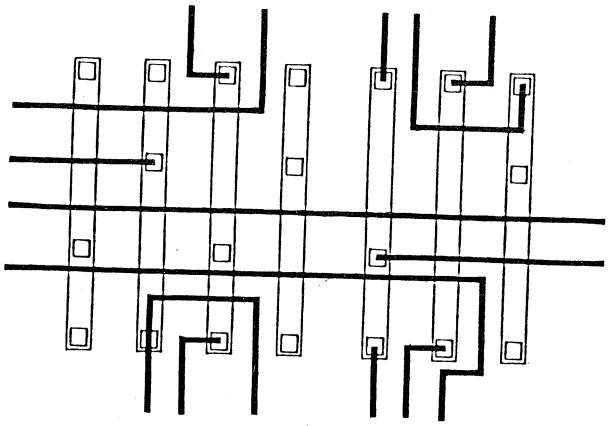
Bild 5.1. Beispiel für die Planarisierung eines Graphen nach Bader [5.4].
a) Gegebener Graph G.
b) Umwandlung in eine Normalform mit kreisförmiger Hamilton-Linie.
c) Hilfsgraph G*.
d) Einzeichnung eines Baumes in den Hilfsgraphen.
e) Zeichnen der ermittelten kreuzungsfreien Darstellung.







(a) This diagram shows how the routing of tracks can be achieved using two metal layers. The lower layer (dotted) runs only in the vertical direction while the upper layer (solid) is constrained to the horizontal.



(b) Crossunders in polysilicon or diffusion placed at fixed, pre-arranged locations on the chip are used here to achieve pseudo two-layer routing of interconnections (metal tracks are drawn as solid black lines: the squares represent contact holes through to the crossunders).

Fig. 4.3 Orthogonal routing of interconnections on two layers

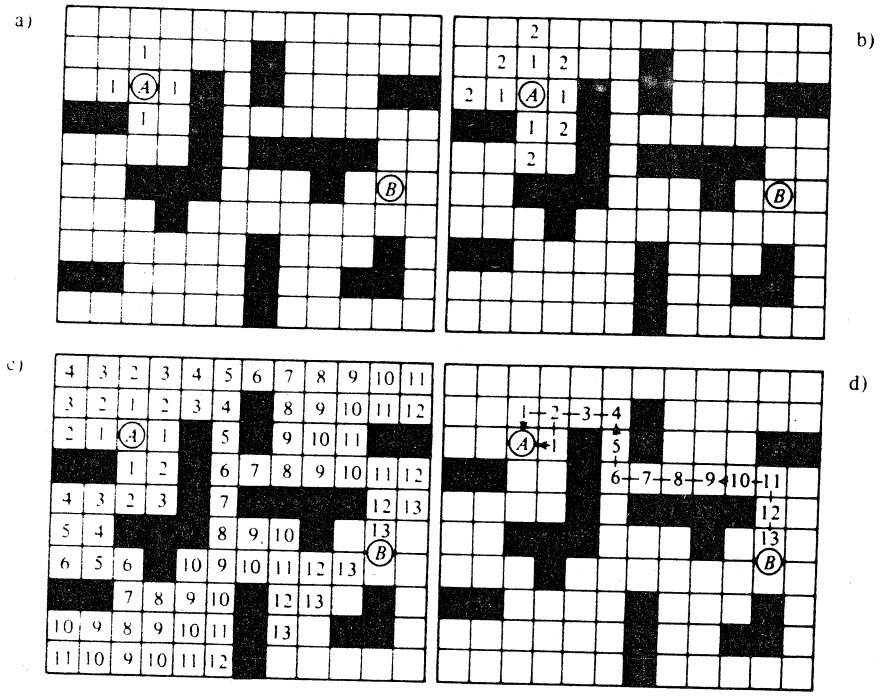


Bild 5.35. Veranschaulichung des Lee-Algorithmus. [5.29].
a) Erster Schritt.
b) Zweiter Schritt.
c) Letzter Schritt.
d) Wegefindung durch Rückverfolgung der kürzesten Spur.

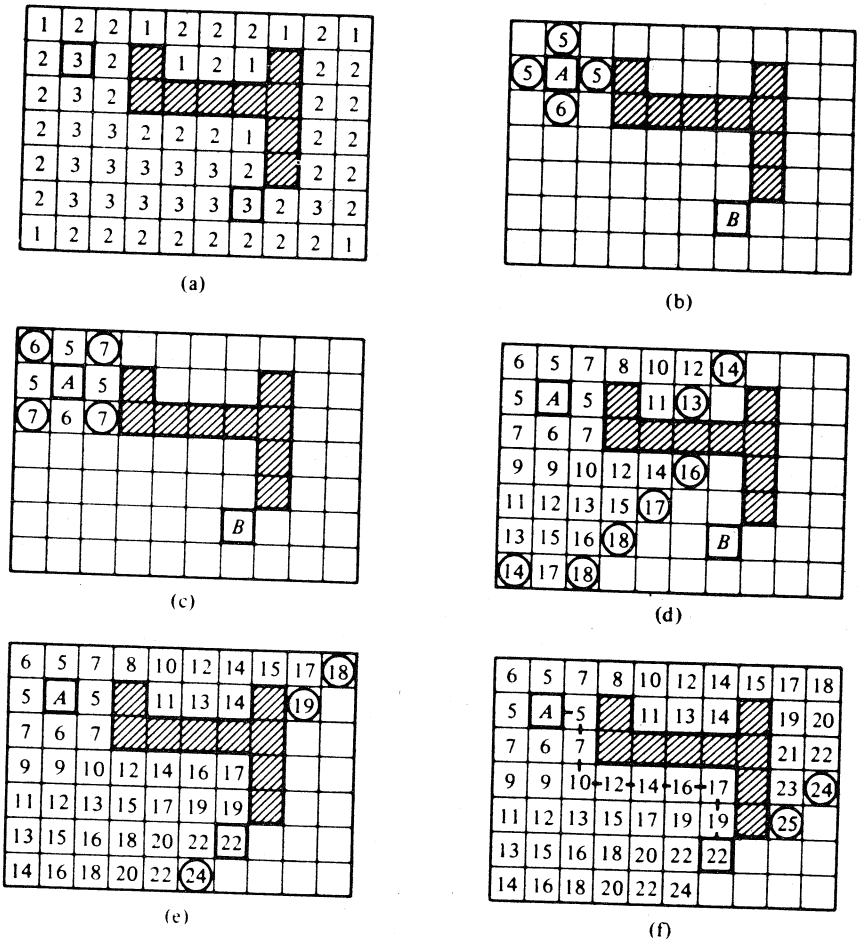


Bild 5.40. Modifizierung des Lee-Algorithmus für die Wegefindung mit dem geringsten Verbrauch an Flächenelementen. [5.29].
a) Belegung der Zellen mit einem Gewicht W. (In diesem Beispiel ist W gleich der Anzahl der benachbarten freien Zellen minus Eins.)
b) Vom Startpunkt an fortschreitende neue Belegung S nach der Vorschrift S = A + W. Erster Schritt.
c) Zweiter Schritt. Nennt man die vorhergehende Belegung jetzt S*, so ergibt sich die Belegung der dazukommenden Zellen aus S = S* + W.
d) Sechster Schritt; wieder S = S* + W.
e) Erreichen des Zielpunktes.
f) Vervollständigen der alternativen Wege und Wegefindung durch Zurückverfolgung der kürzesten Spur.

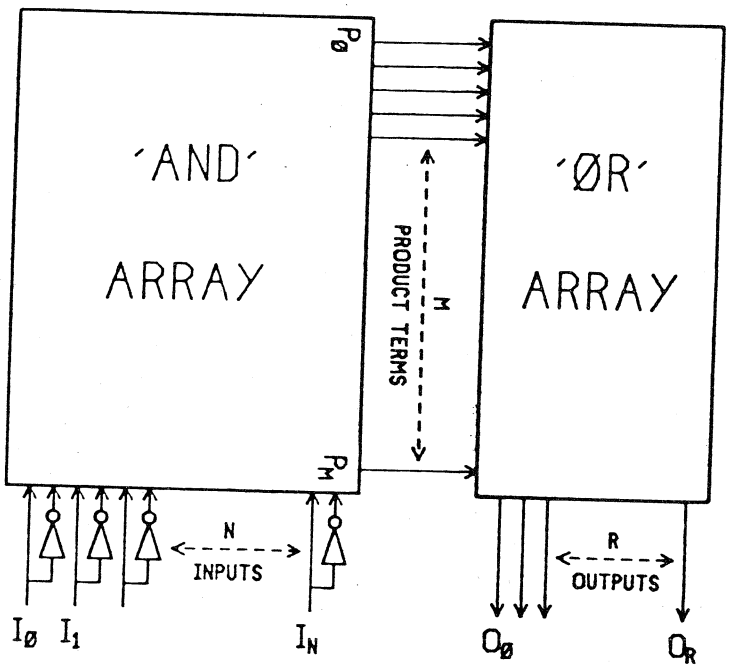


Fig.4.7 A comparison of programmable logic devices.

(a) PROM : 'AND' array pre-programmed. Fully decodes N inputs to yield all $M = 2^N$ possible product terms.
'OR' array user-programmable. Any combination of the M products can be ORed onto any of the R outputs.

(b) FPLA : 'AND' array user-programmable. Generates a chosen subset of M ($<2^N$) product terms from N inputs.
'OR' array user-programmable. Any combination of the M products can be ORed onto any of the R outputs.

(c) PAL : 'AND' array user-programmable. Generates a chosen subset of M ($<2^N$) product terms from N inputs.
'OR' array pre-programmed. Groups of product terms are ORed onto the R outputs according to a pre-arranged pattern.

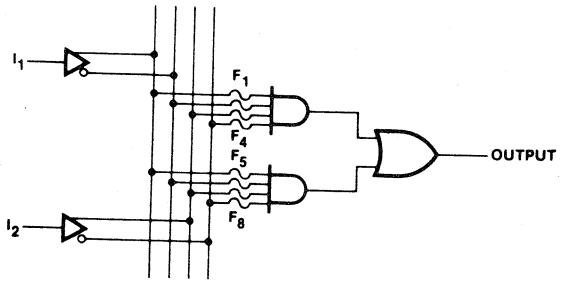


Figure 1

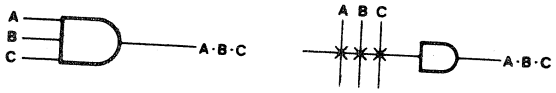


Figure 2

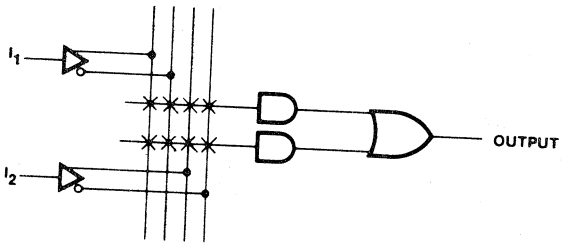


Figure 3

As a simple PAL example, consider the implementation of the transfer function:

$$\text{Output} = I_1\overline{I_2} + \overline{I_1}I_2$$

The normal combinatorial logic diagram for this function is shown in figure 4, with the PAL logic equivalent shown in figure 5.

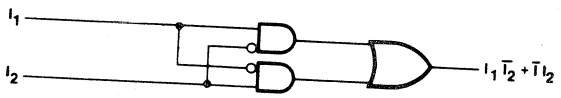


Figure 4

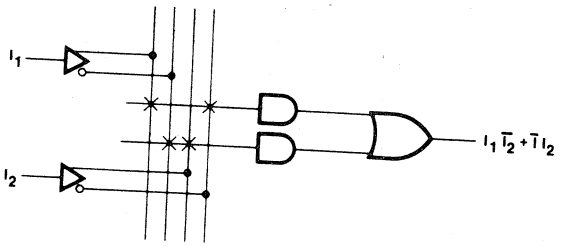


Figure 5

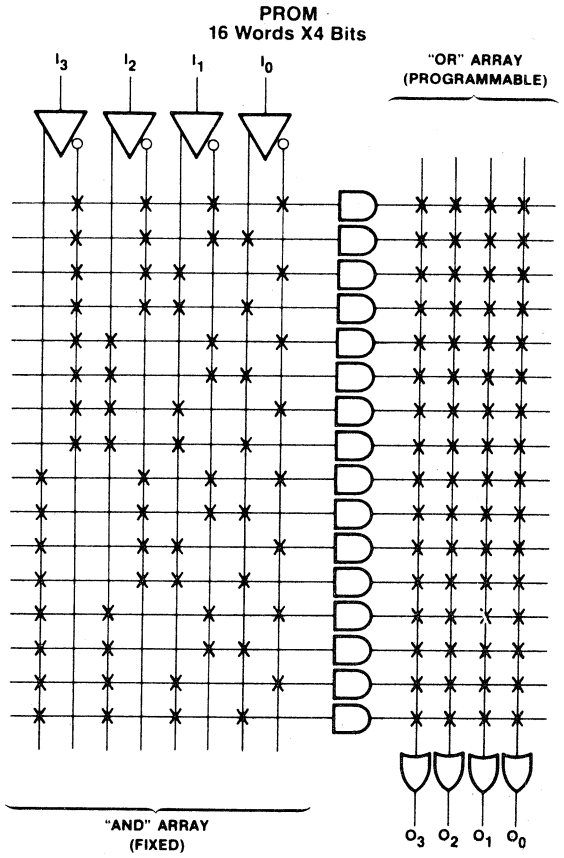


Figure 6

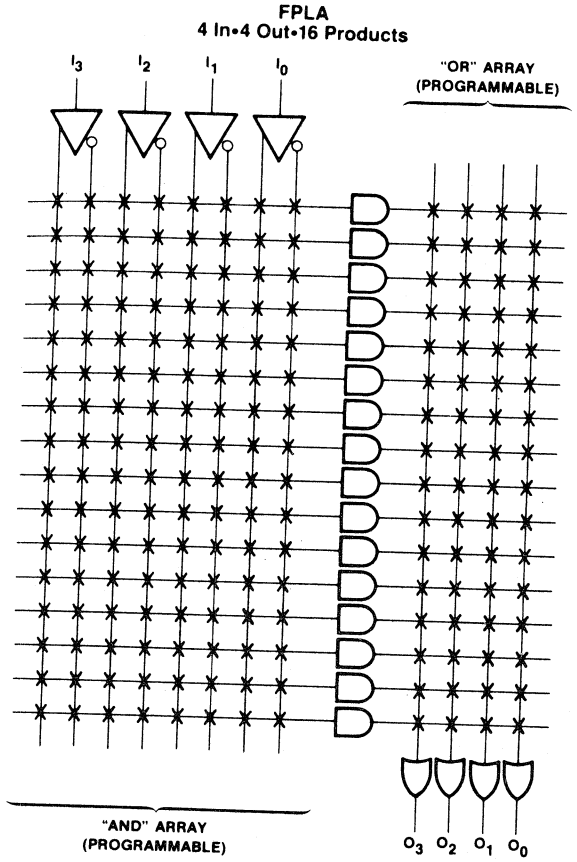


Figure 7

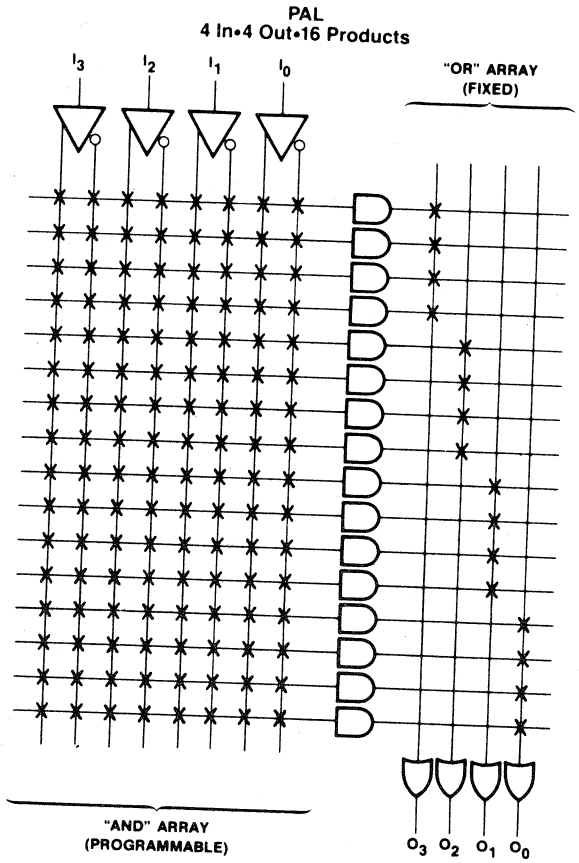


Figure 8

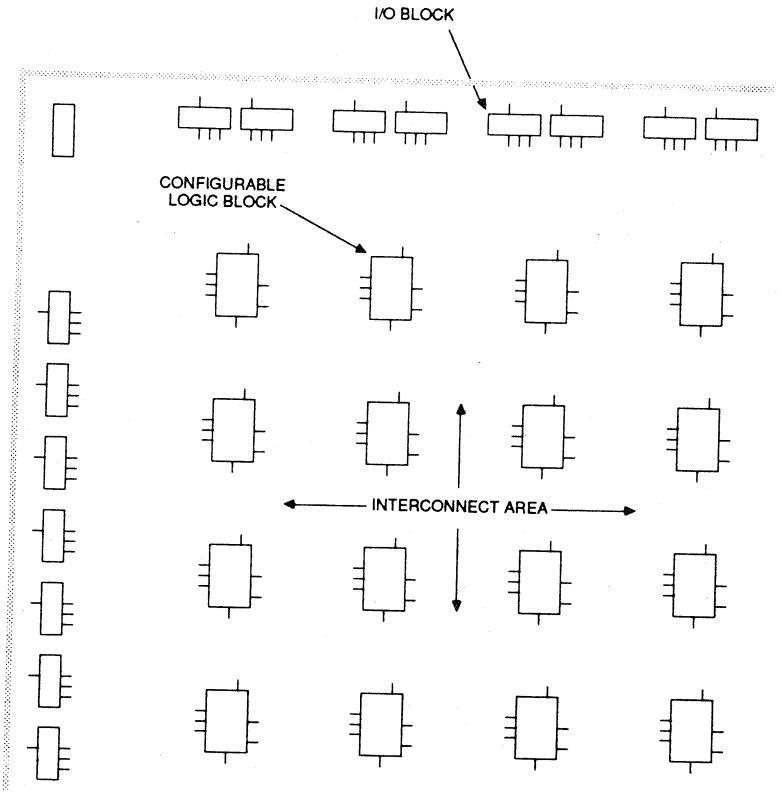


Figure 1. Logic Cell Array Structure

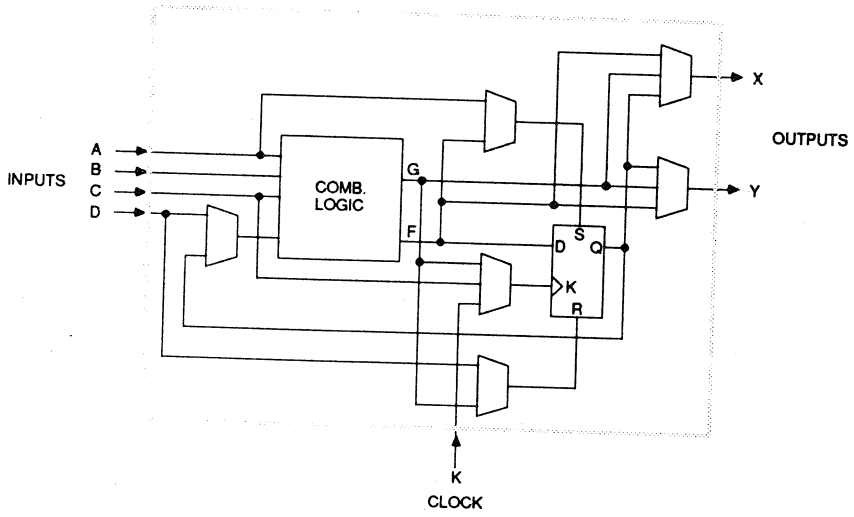


Figure 4. Configurable Logic Block

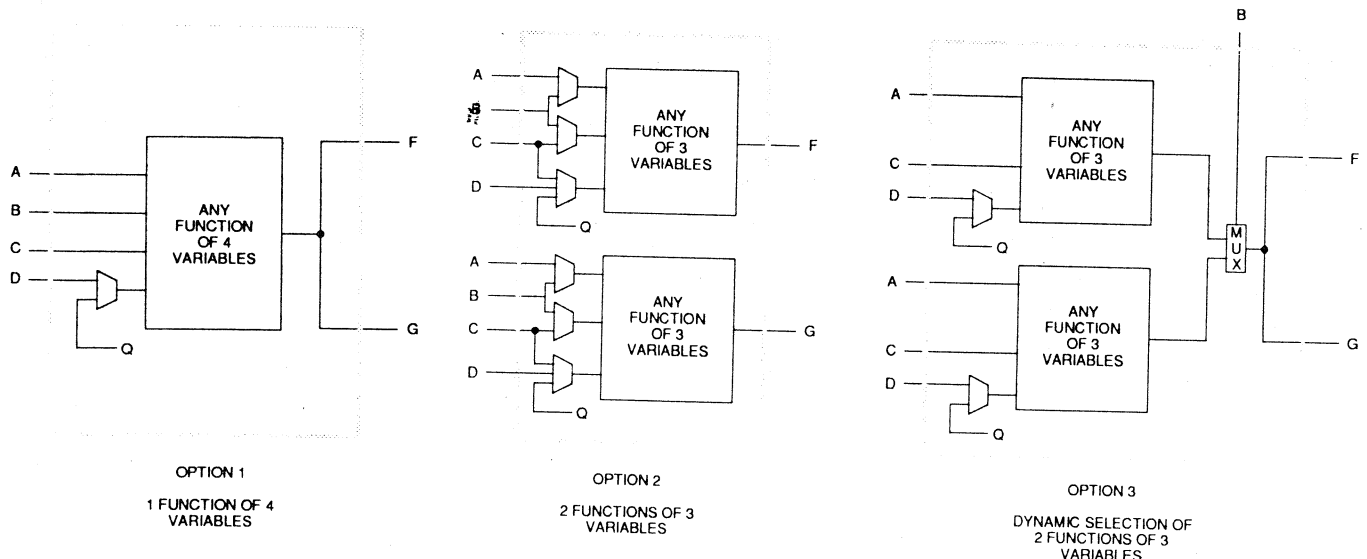


Figure 5. CLB Combinatorial Logic Options
Note: Variables D and Q can not be used in the same function.

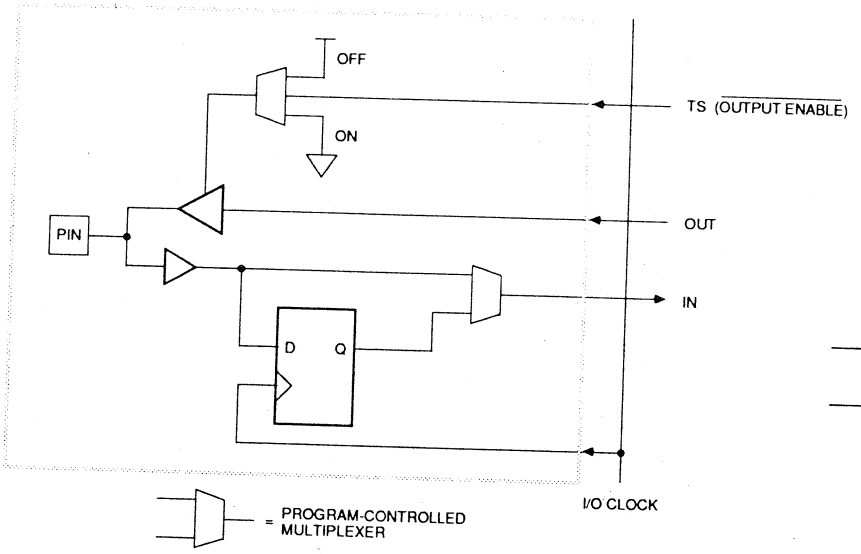


Figure 3. I/O Block

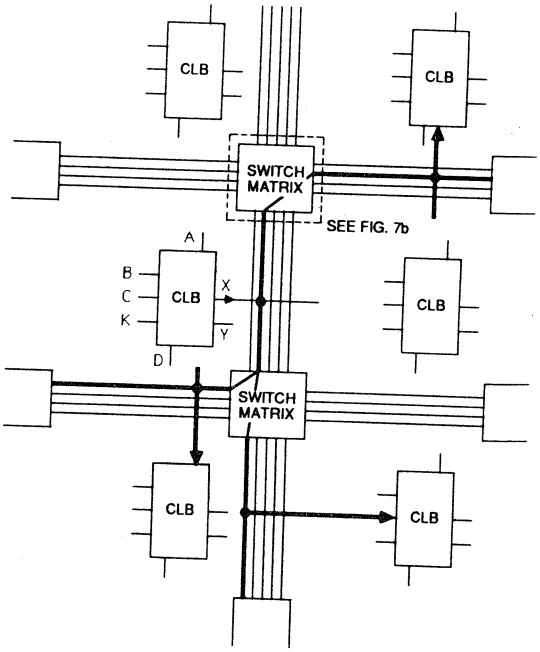


Figure 7a. General-Purpose Interconnect

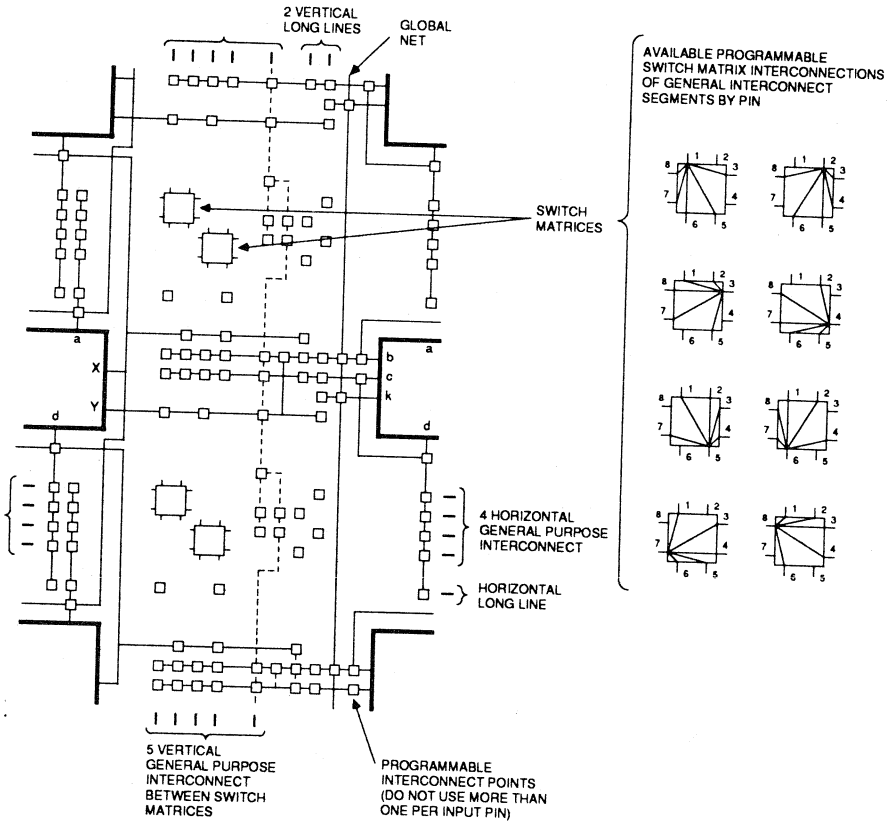


Figure 7b. Routing and Switch Matrix Connections

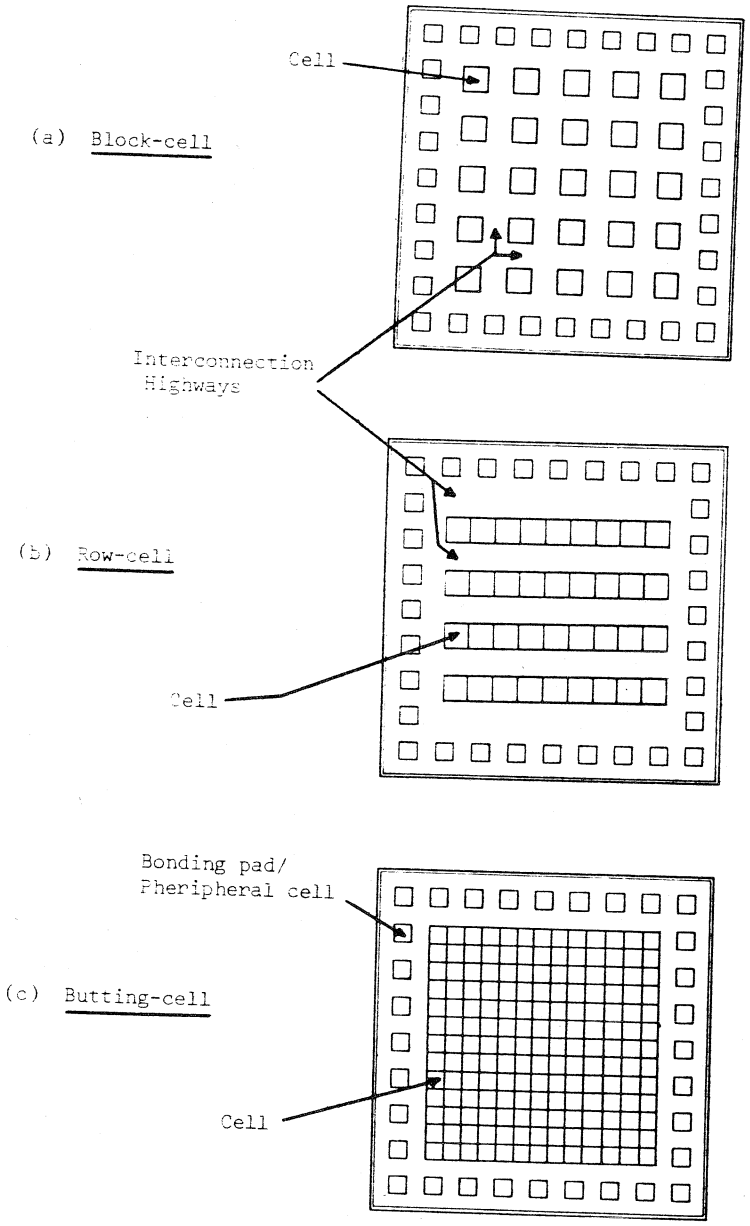


Fig. 4.2 Organisation of gate arrays

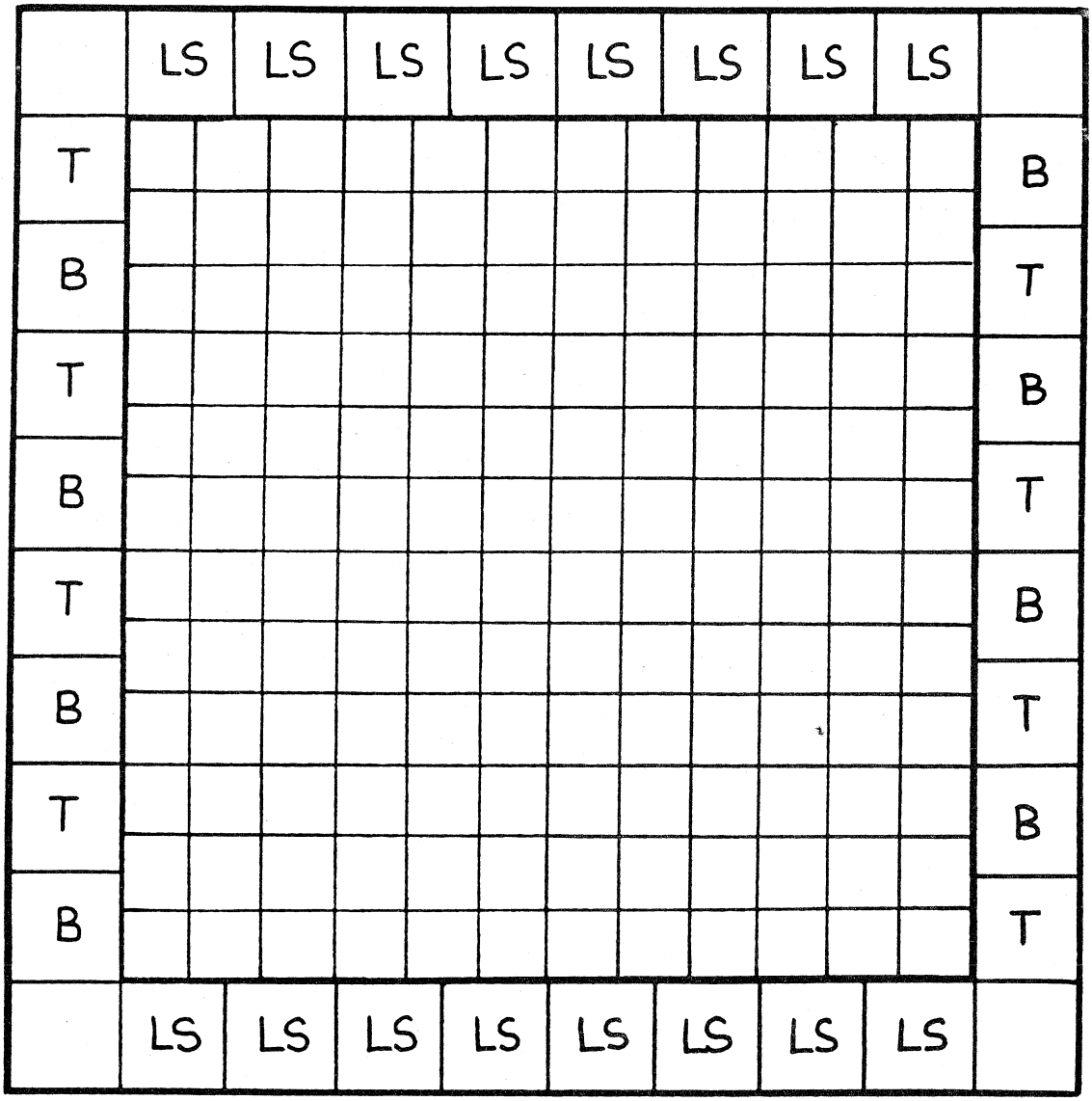
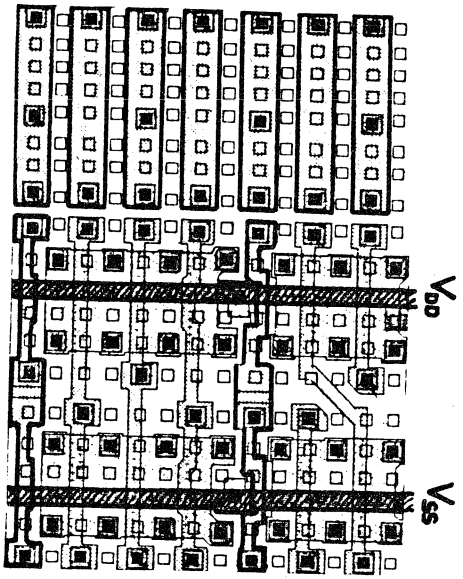


Abb.2: Gesamtaufbau des ULA (schematisch)



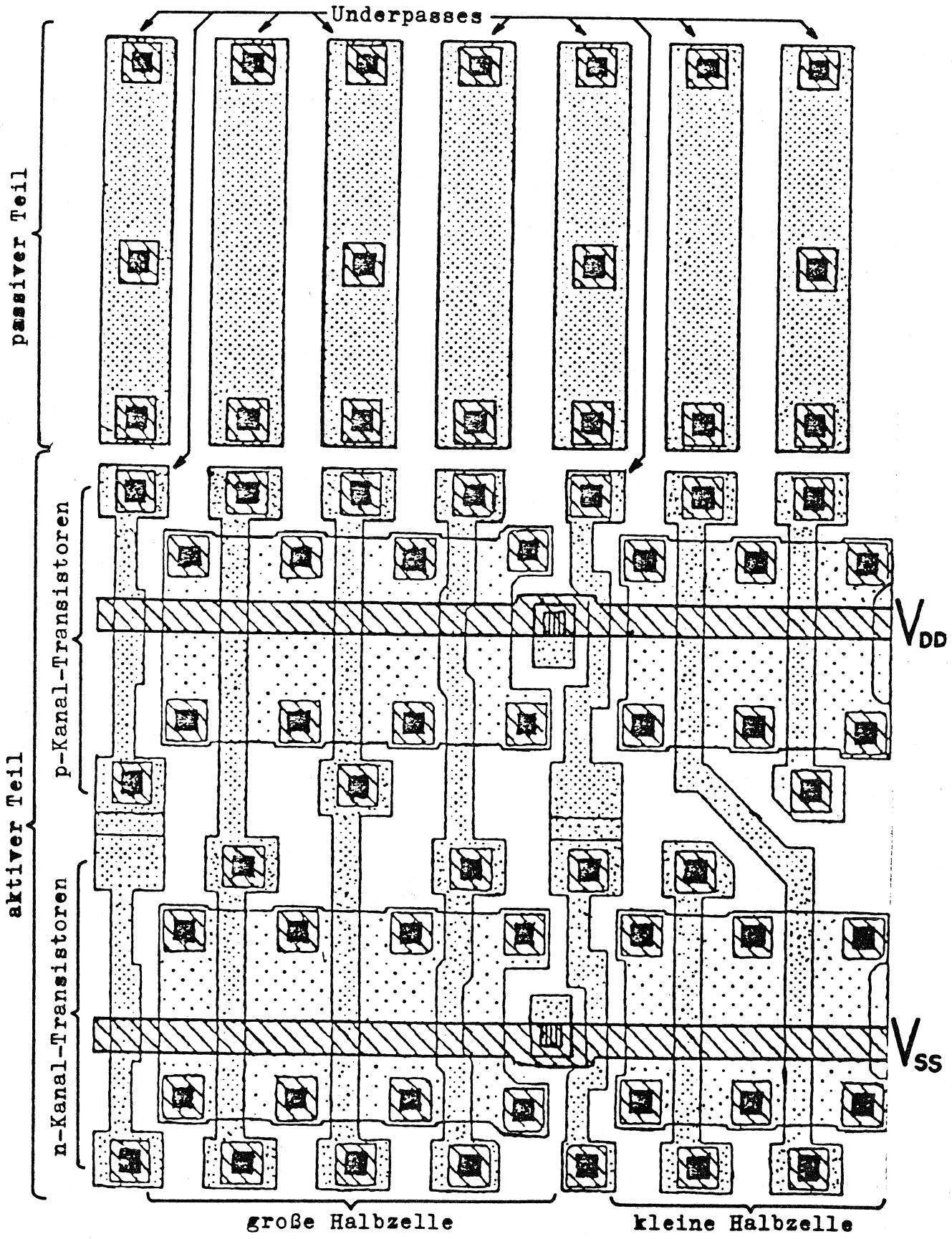
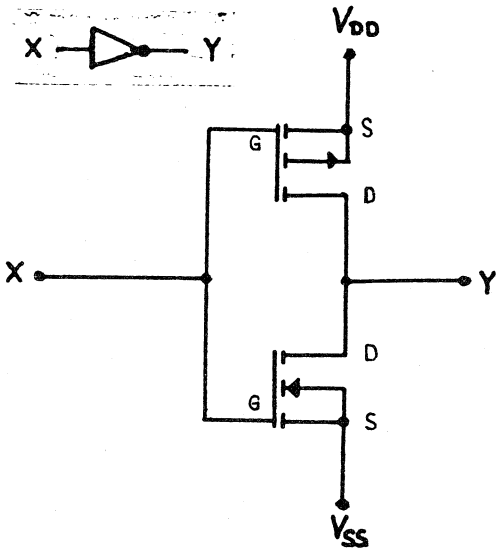
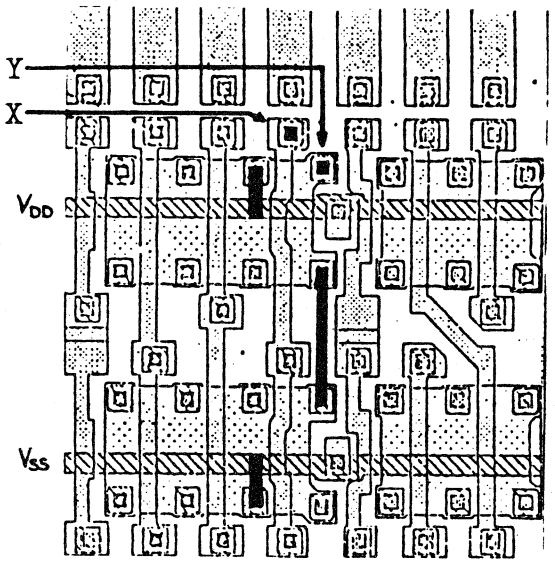


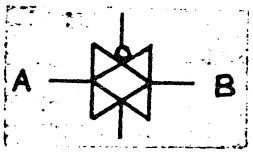
Abb.3: Kernzelle des ULA

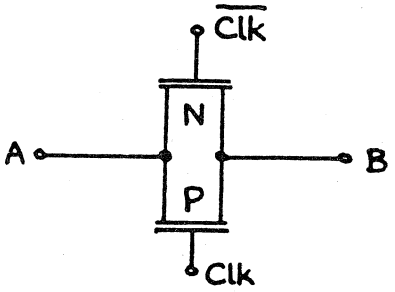
4.1. Inverter (große Halbzelle)

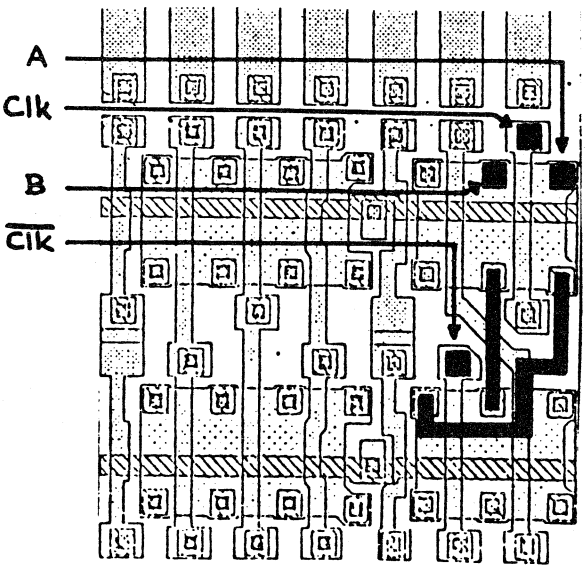




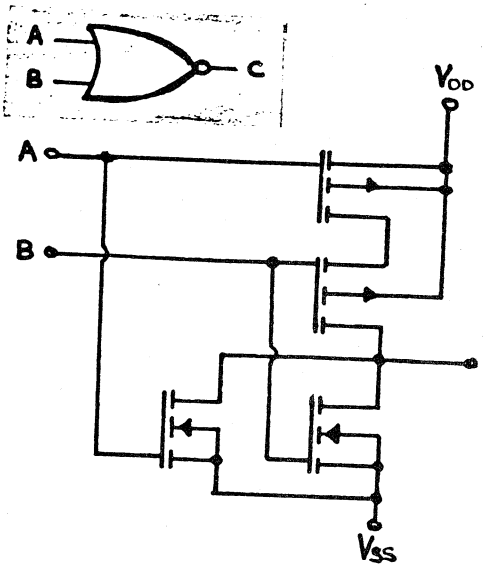
4.6. Transmission Gate (kleine Halbzelle)

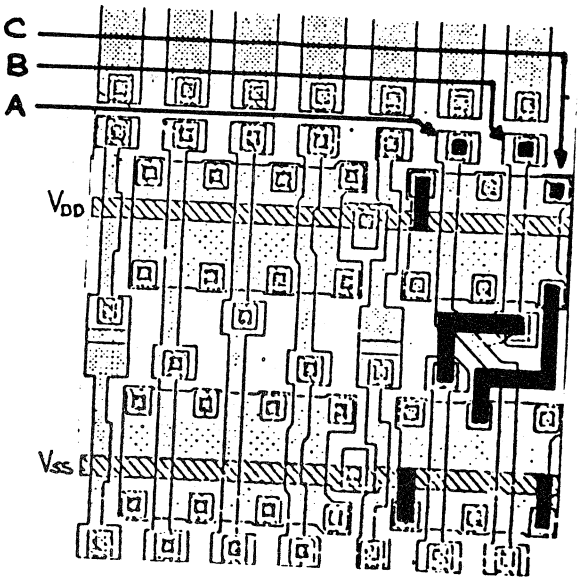




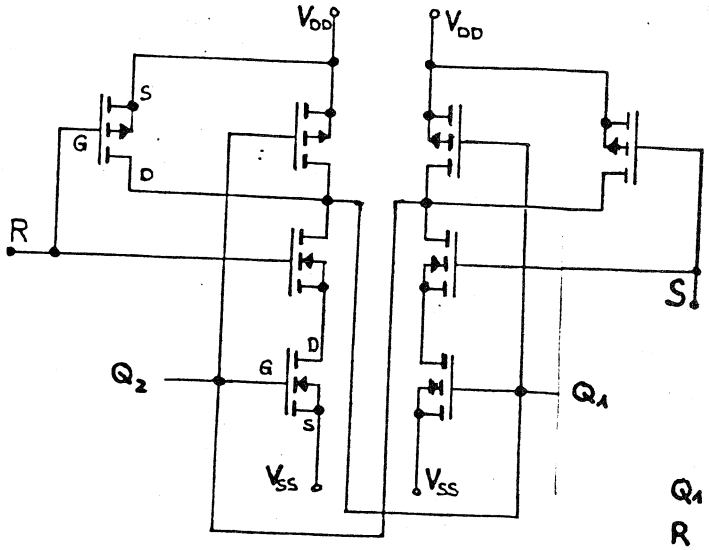


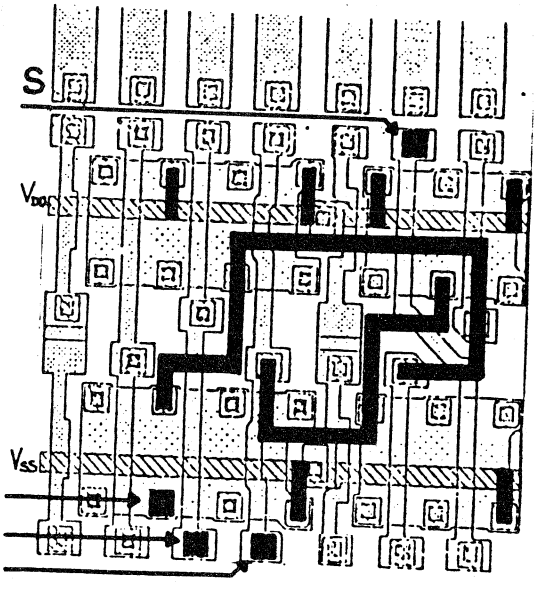
4.4. Two Input NOR (kleine Halbzelle)

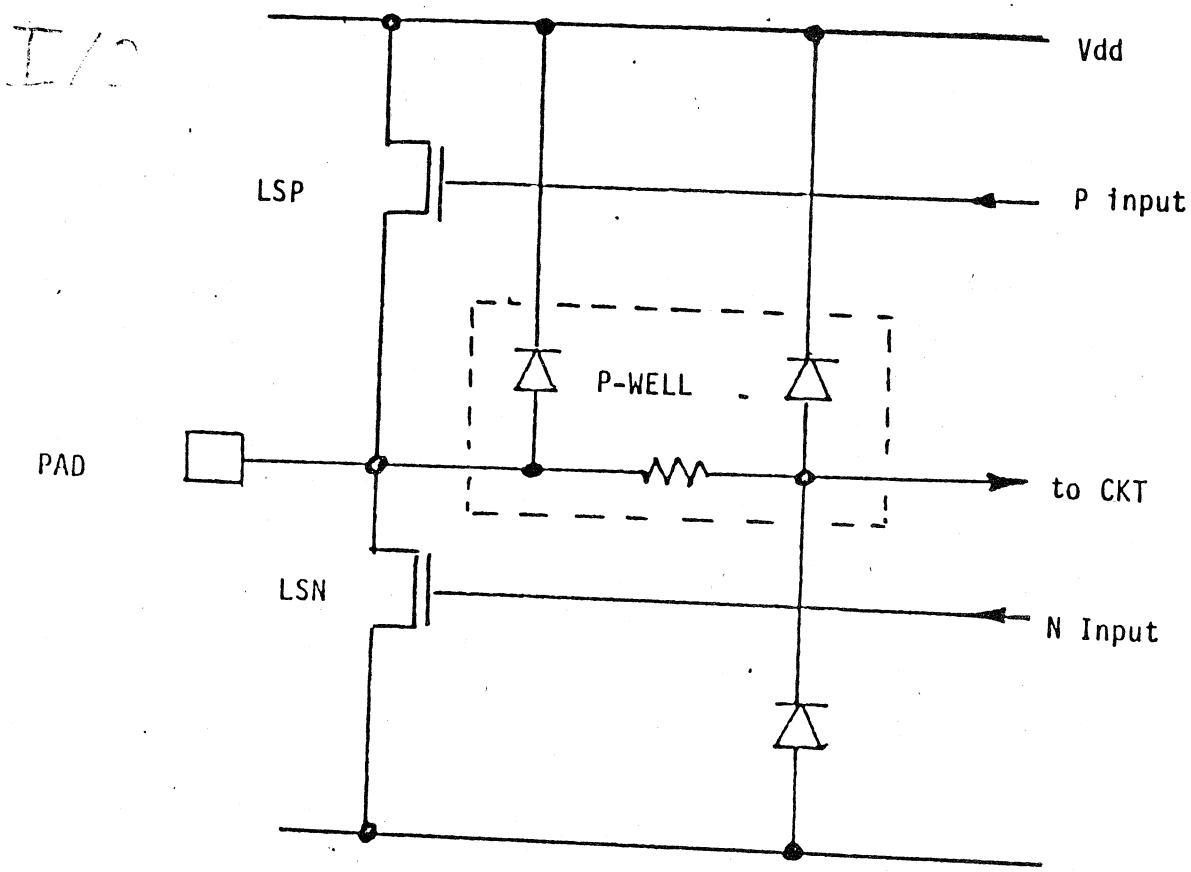


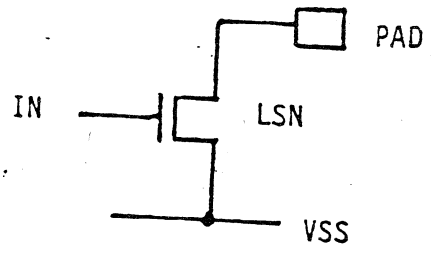


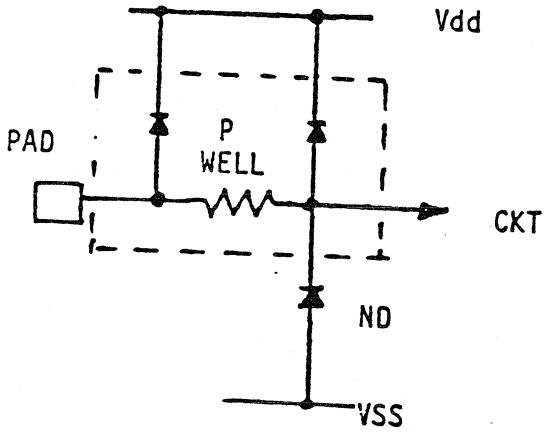
4.8. RS-Flip-Flop

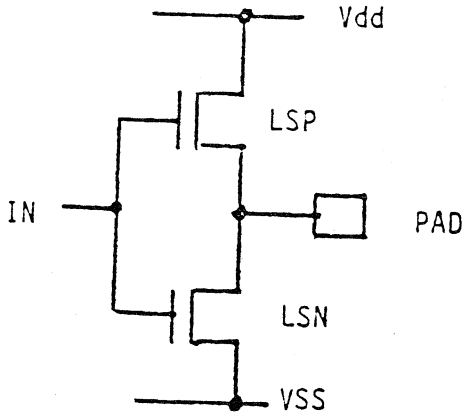


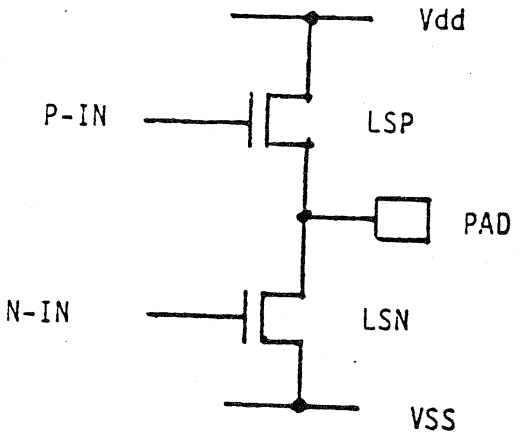



Vdd
LSP
P input
P-WELL
PAD
to CKT
LSN
N Input


PAD
IN
LSN
VSS


Vdd
PAD
P
WELL
CKT
ND
VSS


Vdd
LSP
IN
PAD
LSN
VSS


Vdd
P-IN
LSP
PAD
N-IN
LSN
VSS

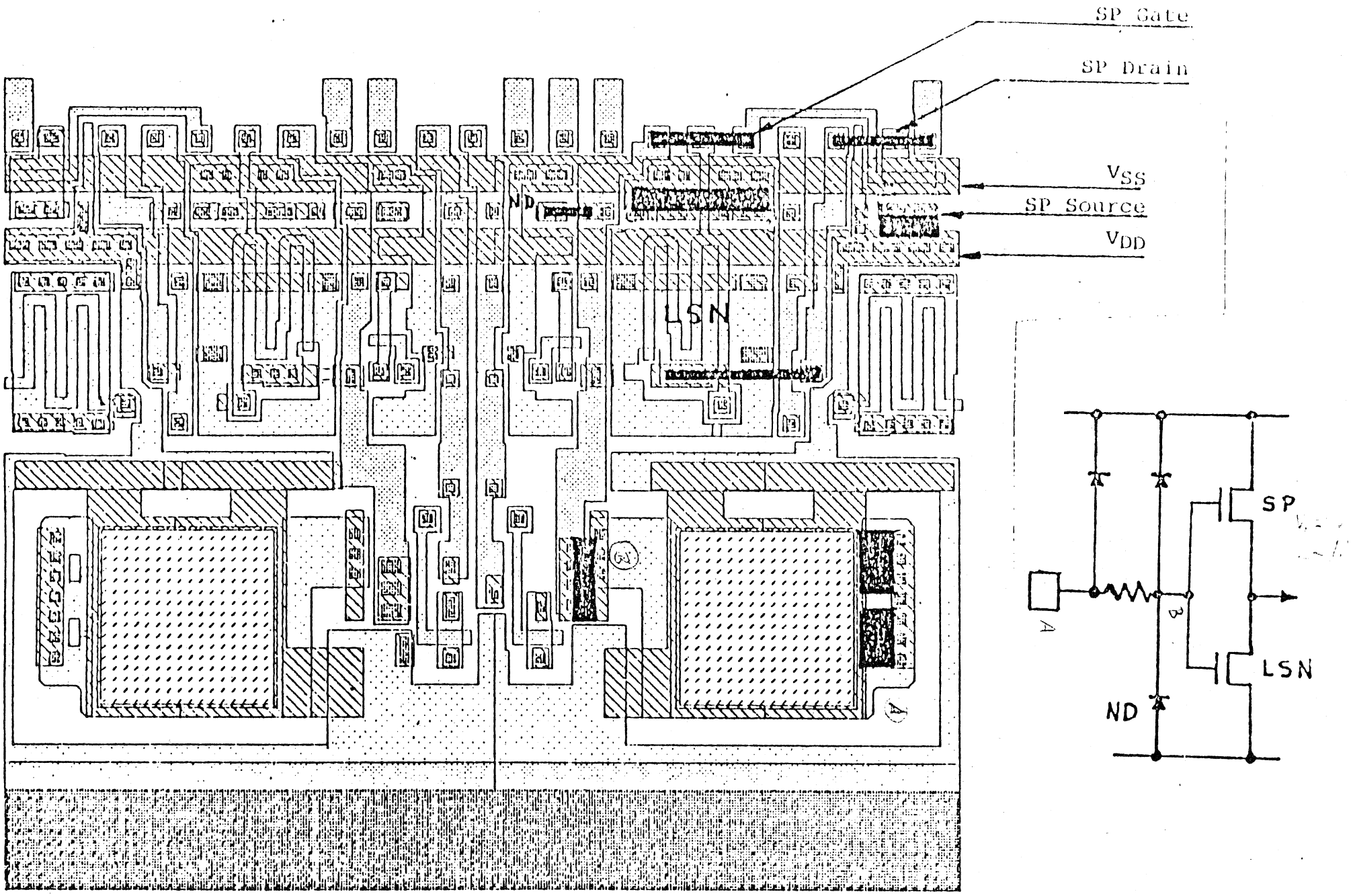


Figure 3.26: LS Level Translator

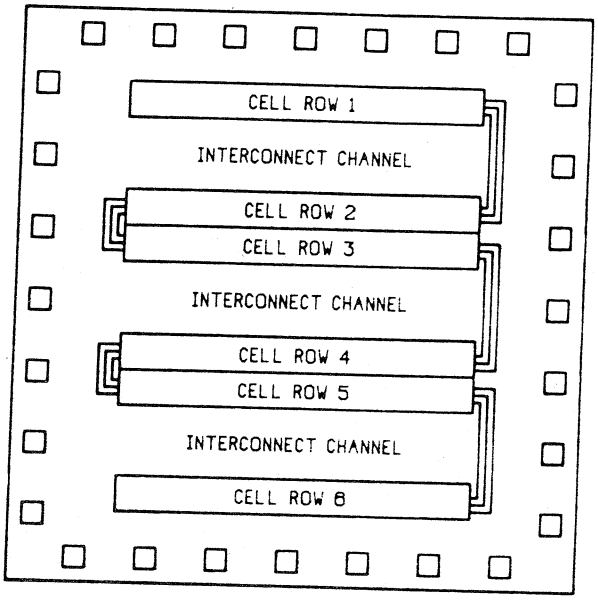


Fig.4.6. Organisation of a Standard Cell IC.

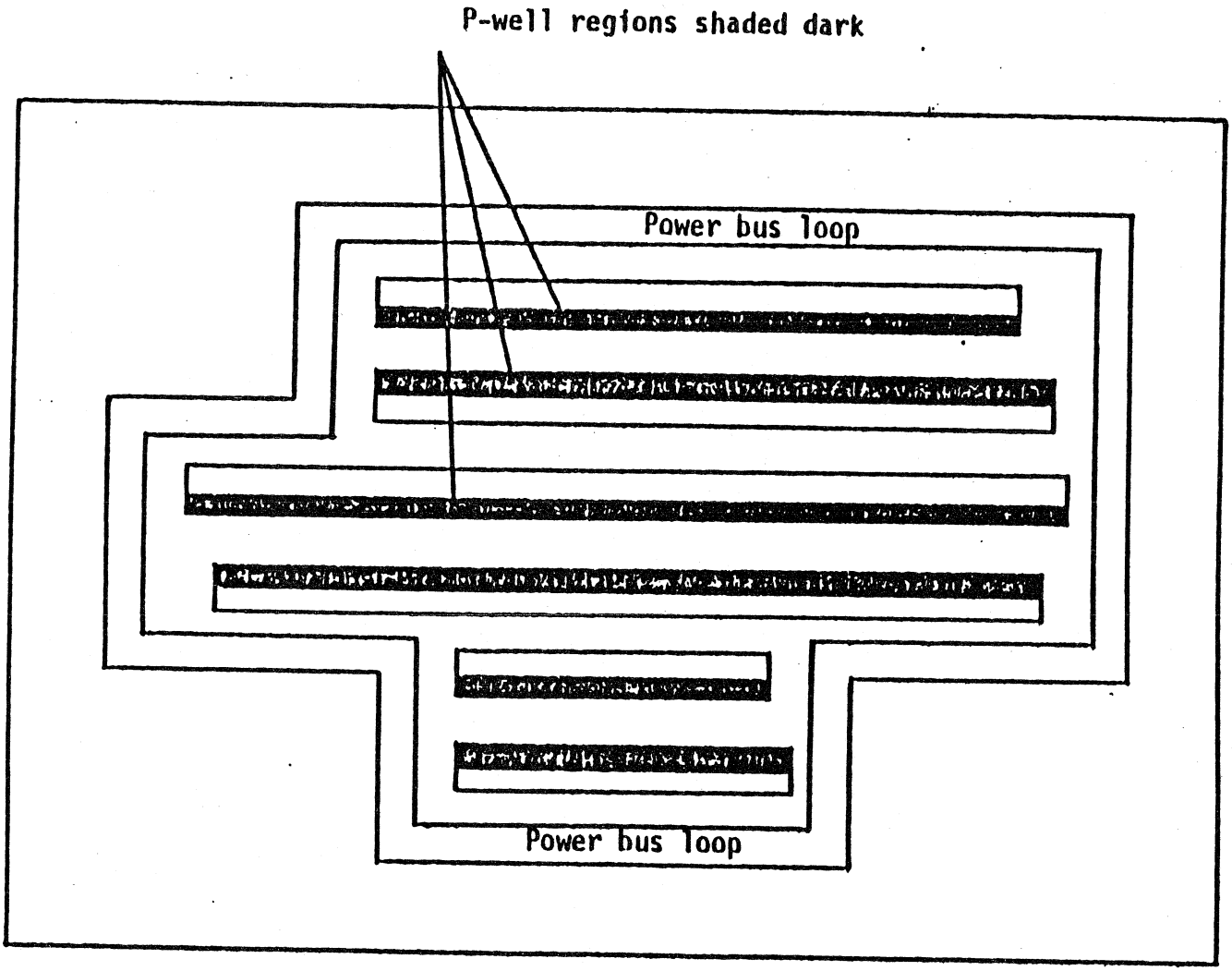


Figure 2.7.

An example of a chip core showing pairs of rows of nearly equal length with the P-wells in each pair facing one another.

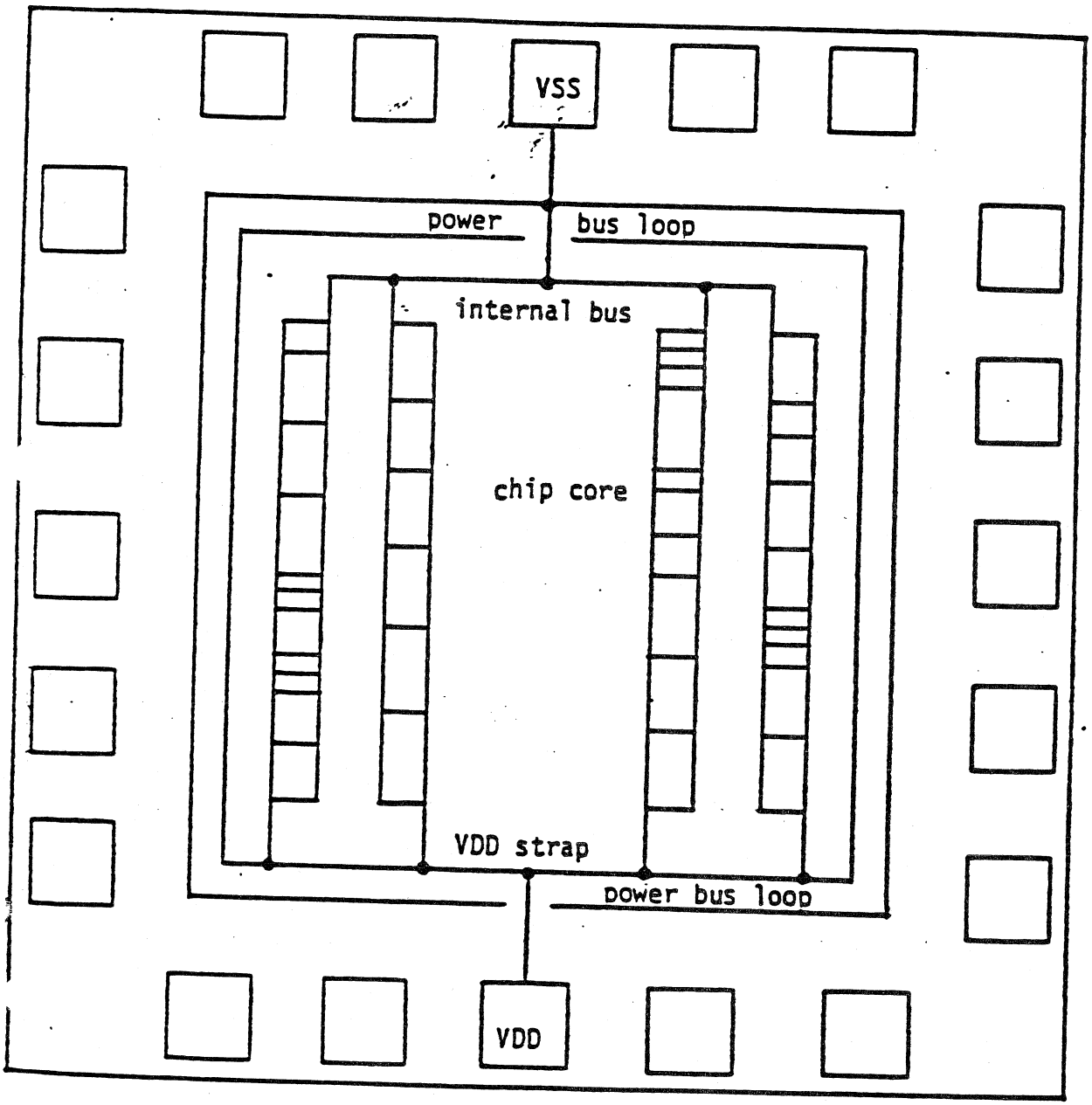


Fig. 2,14

The simplest method for connecting up the power supply. VDD and VSS pads are at opposite ends of the chip, this structure is recommended.

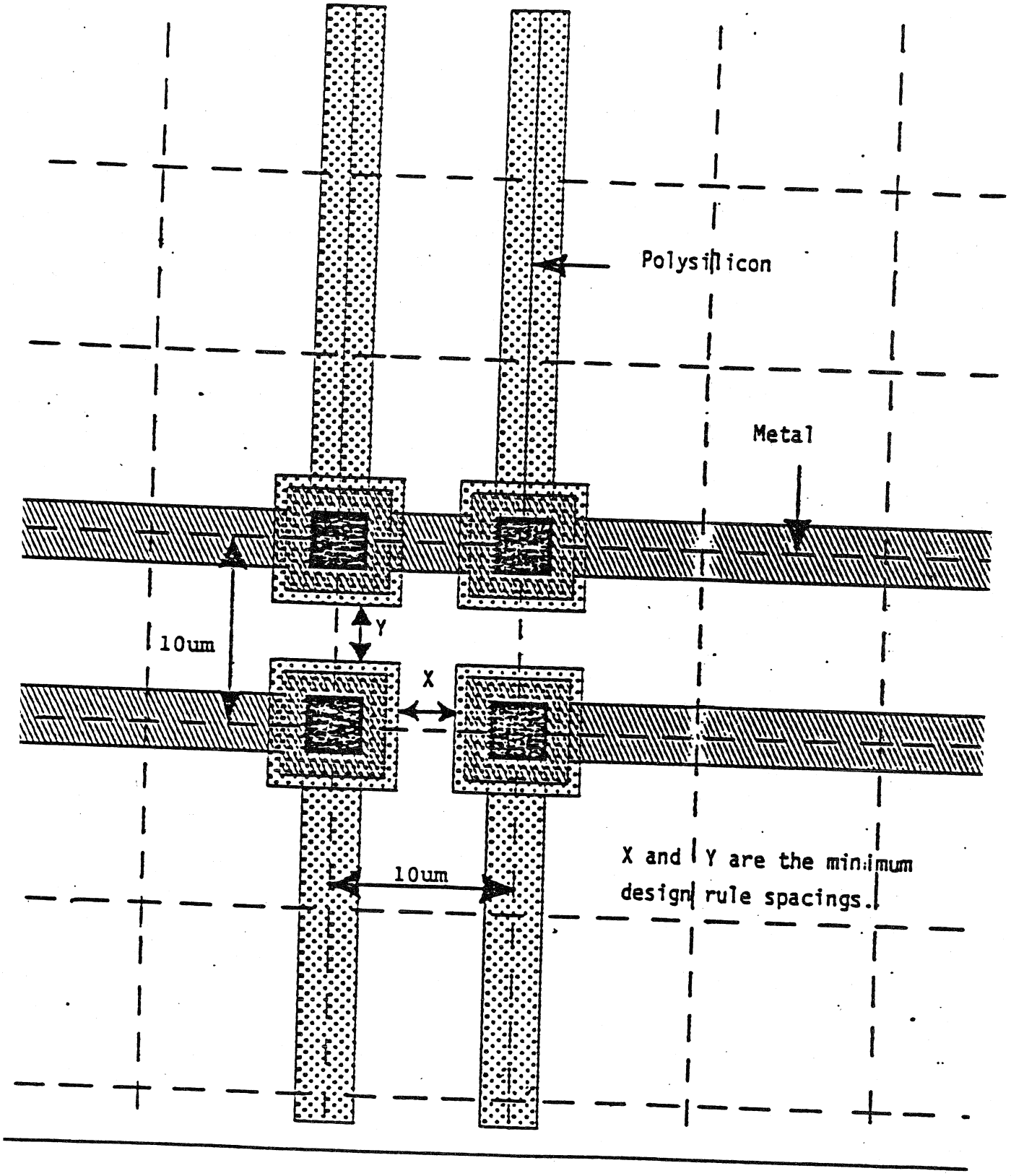


Figure 2.1

Interconnect lines and contacts lie on the 10 um grid. This gives the minimum allowable spacing between contact overlaps.

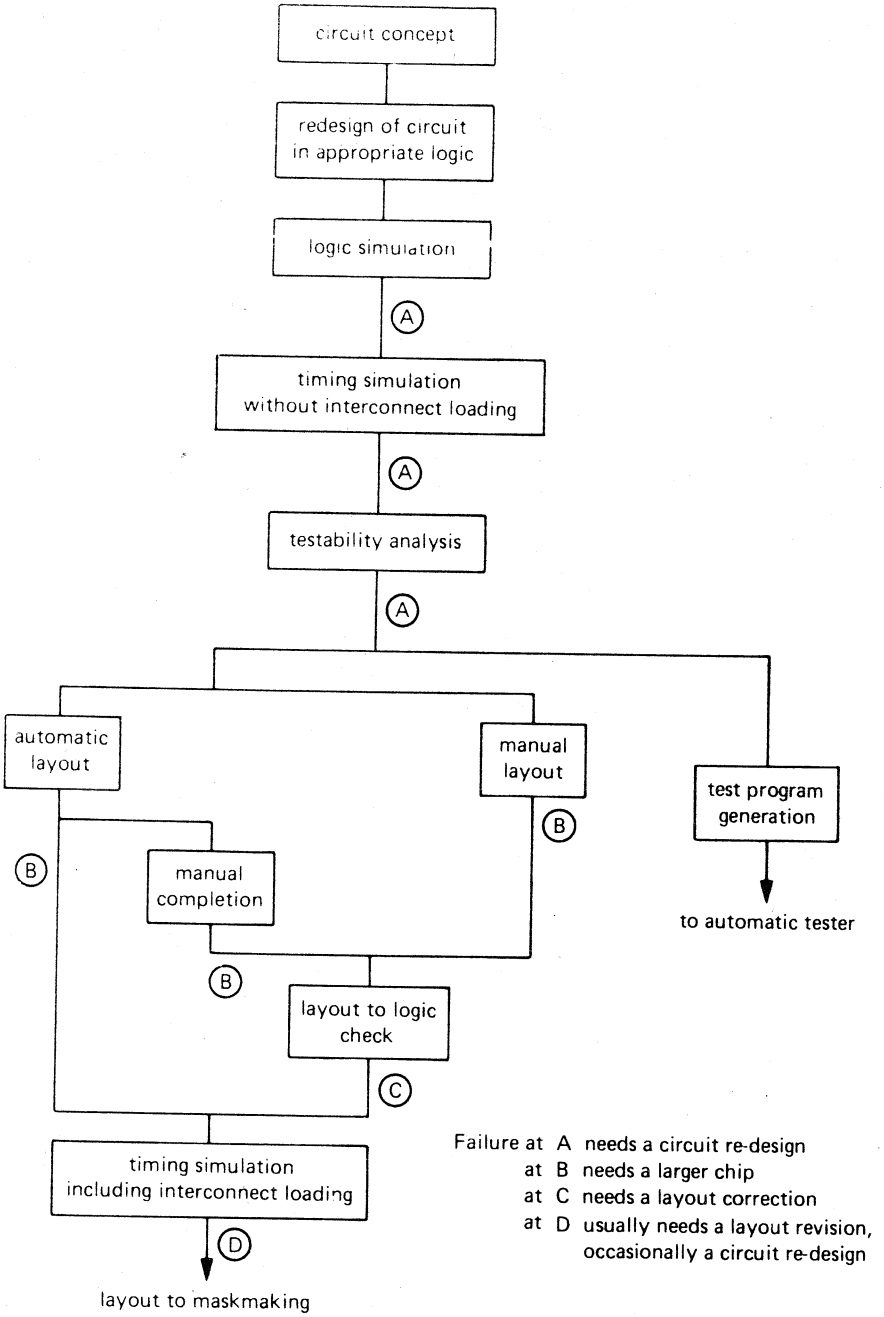
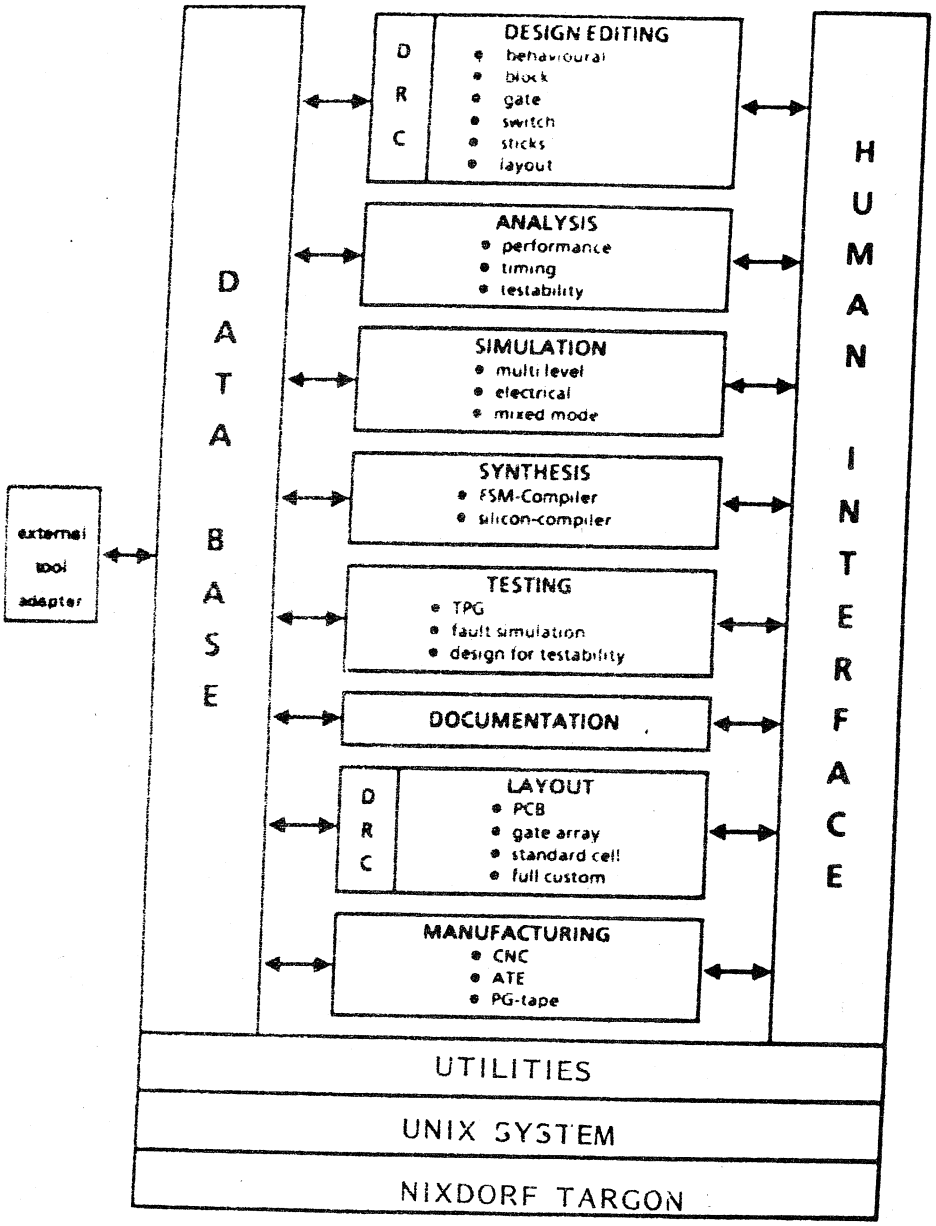


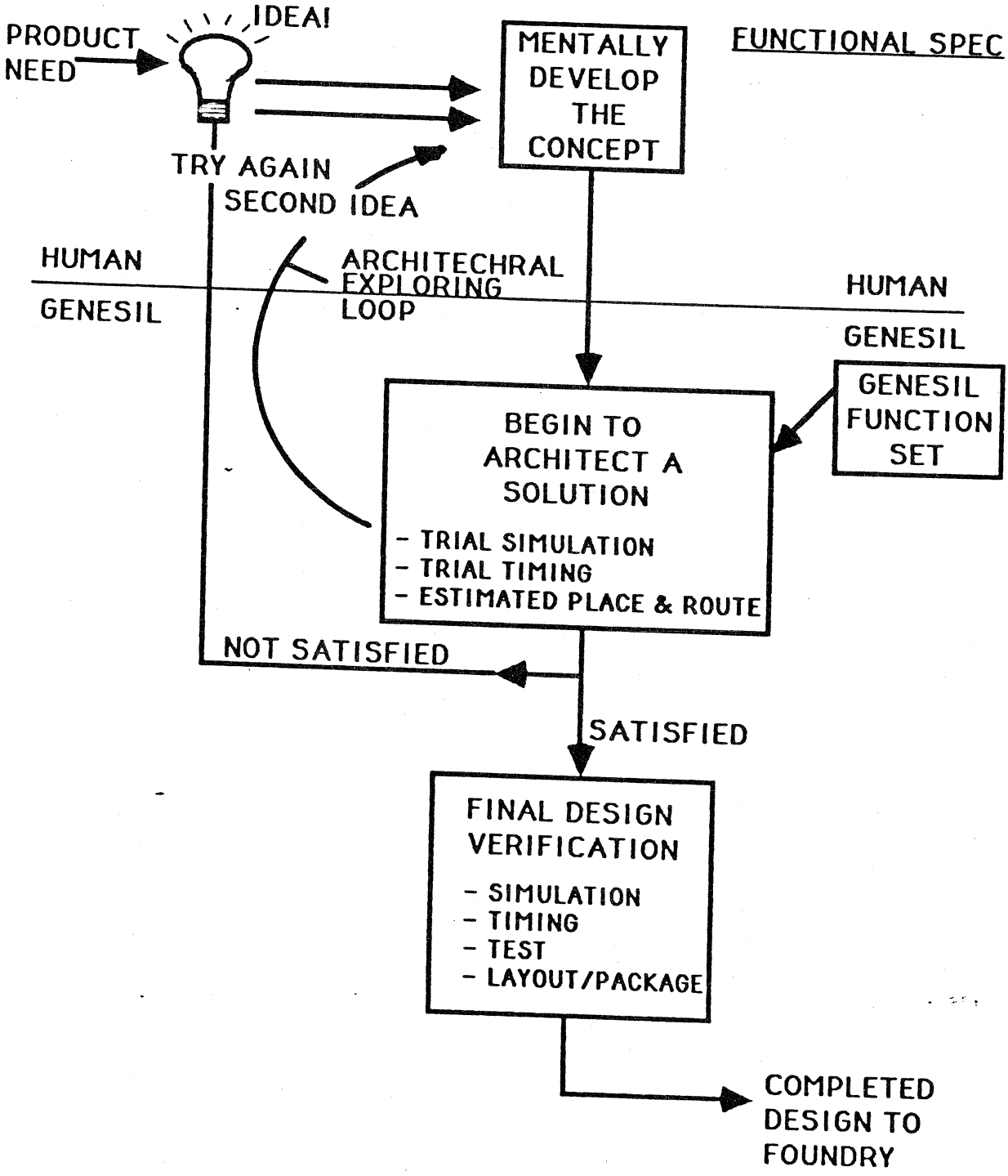
Fig 5.2 Design path for a gate array or cell-based system

Tabelle 6.1. Mögliche Aufgabenteilung zwischen Kunde (K), Hersteller (H) und Designcenter (DC)

Ablauf → / Schritt ↓	1	2	3	4	5	6
Systemspezifikation	K	K	K	K	K	K
Logikentwicklung	DC	K	K	K	K	K
VENUS Logikkonstruktion	DC	DC	K im DC	K im DC	K im DC	K im DC
VENUS Logikverifikation	DC	DC	K im DC	K im DC	K im DC	K im DC
VENUS Prüfbarkeitsanalyse	DC	DC	DC	K im DC	K im DC	K im DC
VENUS Chipkonstruktion	DC	DC	DC	DC	K im DC	K im DC
VENUS Layoutanalyse	DC	DC	DC	DC	K im DC	K im DC
VENUS Fertigungsdatenerstell.	DC	DC	DC	DC	K im DC	K im DC
VENUS Prüfdatenerstellung	DC	DC	DC	K im DC	K im DC	K im DC
Fertigung	H	H	H	H	H	H
Test	H	H	H	H	H	K


DESIGN EDITING
behavioural
block
gate
switch
sticks
layout
D
R
C
ANALYSIS
performance
timing
testability
SIMULATION
multi level
electrical
mixed mode
SYNTHESIS
FSM-Compiler
silicon-compiler
TESTING
TPG
fault simulation
design for testability
DOCUMENTATION
LAYOUT
PCB
gate array
standard cell
full custom
MANUFACTURING
CNC
ATE
PG-tape
external tool adapter
DATA BASE
HUMAN INTERFACE
UTILITIES
UNIX SYSTEM
NIXDORF TARGON

WHAT IS SILICON COMPILATION?



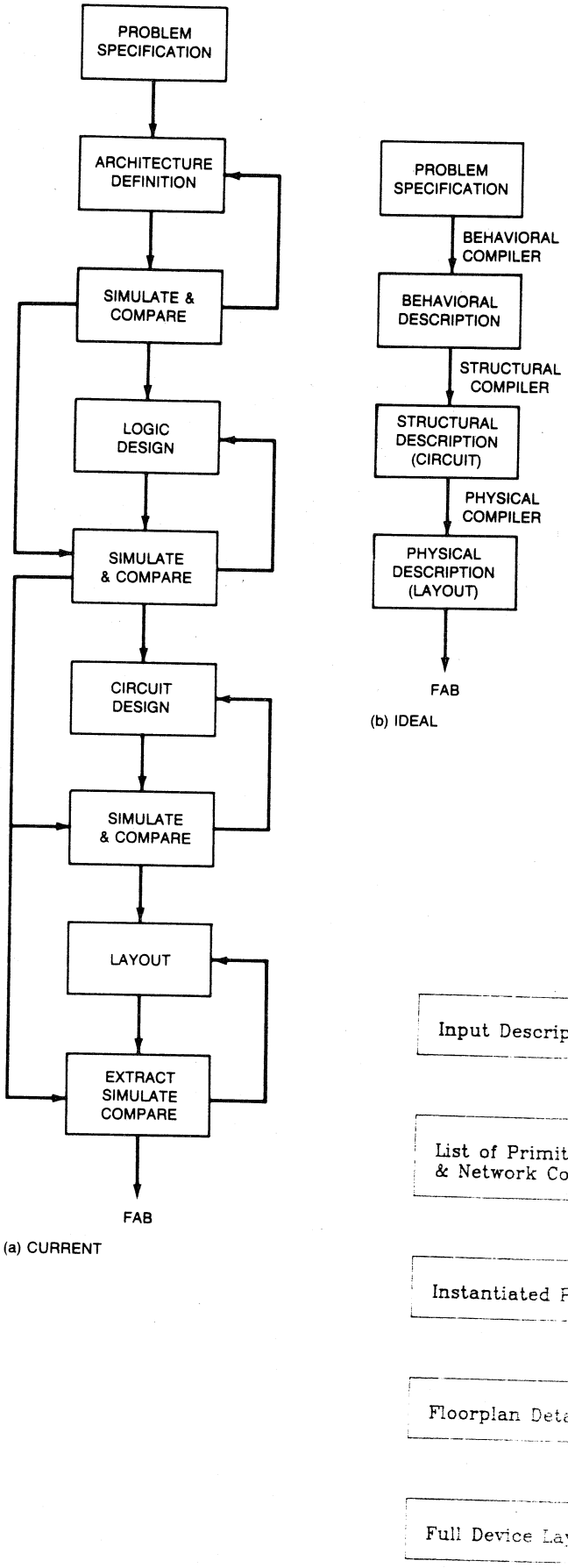
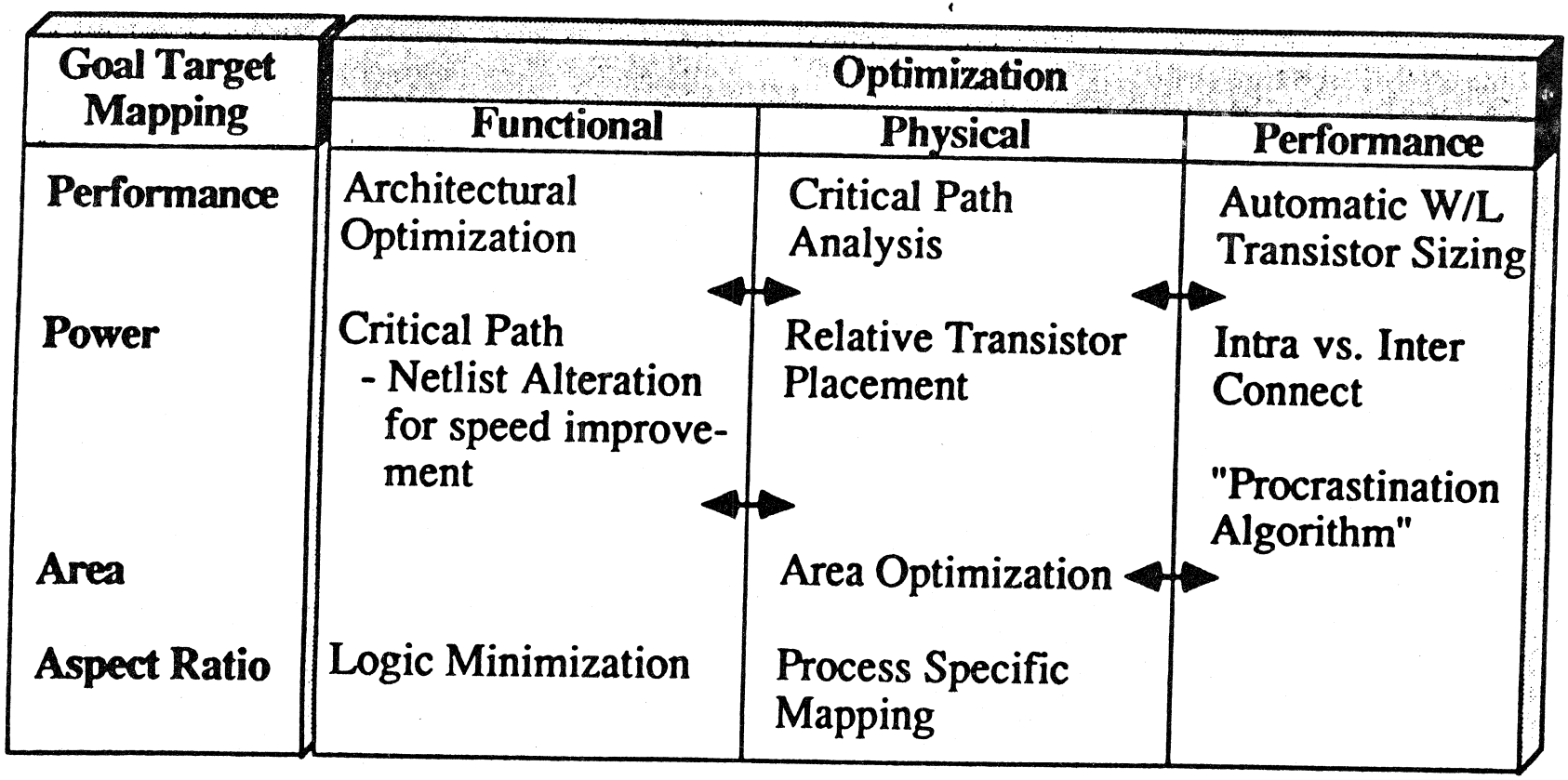


Figure 14.3 Five steps to silicon compilation.


Goal Target Mapping
Optimization
Functional
Physical
Performance
Performance
Power
Area
Aspect Ratio
Architectural Optimization
Critical Path - Netlist Alteration for speed improvement
Logic Minimization
Critical Path Analysis
Relative Transistor Placement
Area Optimization
Process Specific Mapping
Automatic W/L Transistor Sizing
Intra vs. Inter Connect
"Procrastination Algorithm"

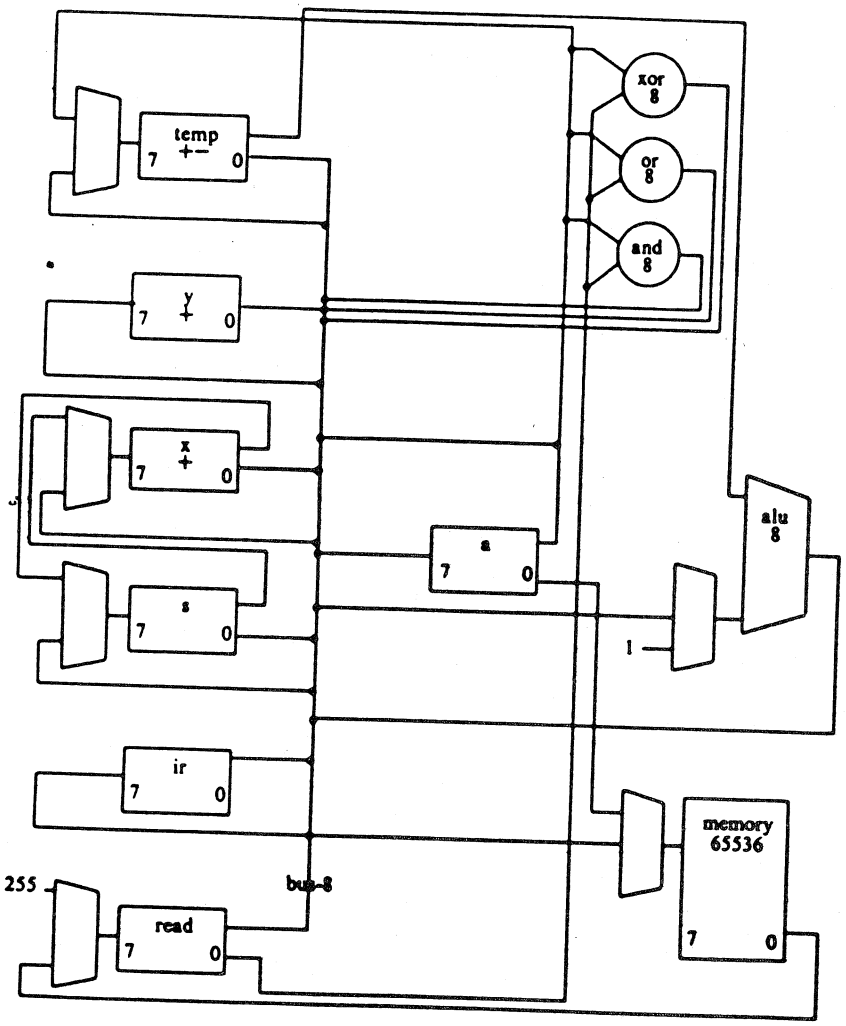
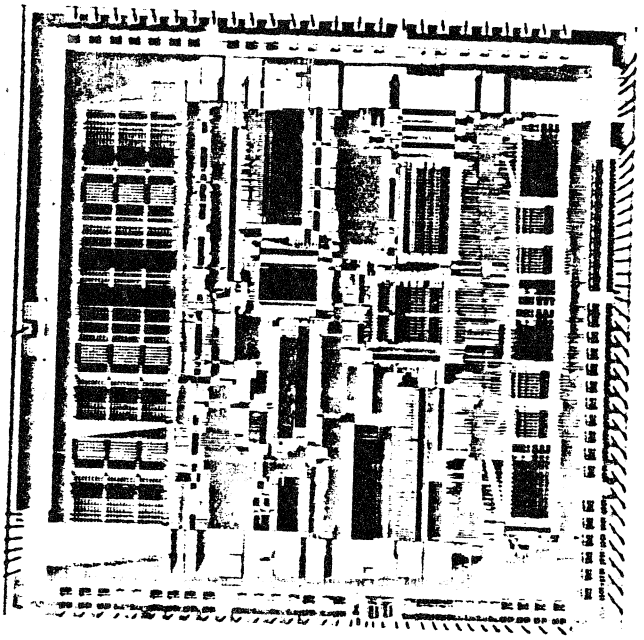


Figure 5.2 MCS6502—8-bit BUS data paths.

Automatisch aus der ISPS-Beschreibung erzeugte Architektur



Die photo of a 32-bit minicomputer's floating point unit.

USER DATA

Company Name: Prime Computers

Customer Business: Minicomputer developer and manufacturer.

Compiler User Since: November 1984

[C**Number of Circuits Under Design:** Multiple.

Reason for choosing Silicon Compilation: In early 1984, Prime formulated a set of goals for the design of a high-performance CPU chip set that would meet future demands:

1. A short product development time
2. Low design and fabrication costs
3. Small Size
4. Low power consumption
5. High performance
6. Leverage in state-of-the-art semiconductor technology
7. Minimization of the chip design experience required
8. Retention of the expertise within Prime.

Prime had no IC design experience. The chip was too big to fit in either standard cells or gate arrays; the GENESIL system was chosen so that system designers could design complex IC's without IC knowledge.

Experience of Engineers using GENESIL:
Gate Array designs: Yes
Standard Cell designs: Yes
Full custom designs: No

APPLICATION DATA

Application: This is a floating point processor for the Prime floating point chip set and will be used in a 32-bit minicomputer.

The floating-point unit (FPU) will perform all single- and double-precision instructions, including addition, subtraction, multiplication, division, and data conversion.

Features Utilized: The chip contains 10 main blocks, comprising 70 primitive blocks.

Operating Frequency: 6 MHz
Technology: 2μ NMOS
Die Size: 420 x 420 mils
Power: 1.9W at 5V
Number of Transistors: 78,000
Number of Pins: 144
Package Type: Pin Grid Array

DESIGN TIME

Elapsed time to develop: 9 months (18 man months)

```
        FIRST COMPILER   - Copyright Denyer,Renshaw,Bergmann - 1982
SOURCE FILE: LMSFIR1

!adaptive(LMS)Transversal(FIR)Filter
!filter stage of multiplexed filter sections

CONSTANT rounds=1
CONTROL INPUT c0,c1,init
CONTROL INPUT c1t0,c1t1,c1t2,c1t3,c1t5,c1t6,c2t0
INPUT Xin,TMSIN,WMSBin,WLSBin
OUTPUT Xout,TMSBout,TISBout,TLSBout

   OPERATOR FIR[wordlength,idles,stages]

      SIGNAL s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,
          s16,s19,s20,s21,s22,ss1,ss2,ss3,ss4,ss5,ss6

       Multiplex[1,0,0] (c2t0) s3,Xin -> s1
       Multiplex[1,0,0] (INIT) s15,WMSBin -> s7
       Multiplex[1,0,0] (INIT) s16,WLSBin -> s6
       Multiplex[1,0,0] (c1t2) s6,s5 -> s6
       Multiplex[1,0,0] (c1t6) TMSBout,TISBout -> TMSBout
       Multiply[round,wordlength-2,0,0] (c1t0->nc) s2,TMSin -> s5,nc
       Multiply[round,wordlength-2,0,0] (c1t5->nc) s22,s15 -> TLSBout,TISBout
       Add[1,0,0,0] (c1t3) s6,s11,gnd -> s15,nc
       Add[1,0,0,0] (c1t3) s5,s12,gnd -> s16,nc
       constant p1=(idles+1)*(wordlength)
       Bitdelay[1] s1 -> ss1
       Bitdelay[1] ss1 -> ss2
       Bitdelay[1] ss2 -> ss3
       Bitdelay[1] ss3 -> ss4
       Bitdelay[1] ss4 -> ss5
       constant p2 = p1-90 , p3 = p2/2 , p4 = p2-p3
       Bitdelay[p3] ss5 -> ss6
       Bitdelay[p4] ss6 -> Xout
       Worddelay[idles+stages-1,wordlength-2,0] (c1t1) s7 -> s10
       Worddelay[idles+stages-1,wordlength,0] (c1t1) s8 -> s11
       Constant half = stages/2,
                rest = (stages - 1) - half
       Worddelay[half,wordlength-2,0] (c1t0) Xout -> s4
       Worddelay[rest,wordlength-2,0] (c1t0) s4 -> s2
       Bitdelay[wordlength] s1 -> s19
       Bitdelay[wordlength] s19 -> s20
       Bitdelay[wordlength] s20 -> s21
       Bitdelay[wordlength-1] s2 -> s3
       Bitdelay[wordlength-4] s9 -> s11
       Bitdelay[wordlength-4] s10 -> s12
       Bitdelay[wordlength/2+1] s21 -> s22

   END

   ! ALLOCATE VALUES TO PARAMETERS : wordlength,idles,stages

   FIR[14,16,8];

ENDOFPROGRAM
```

Figure 14.7 An example of silicon compilation using FIRST.
(a) Input system description file

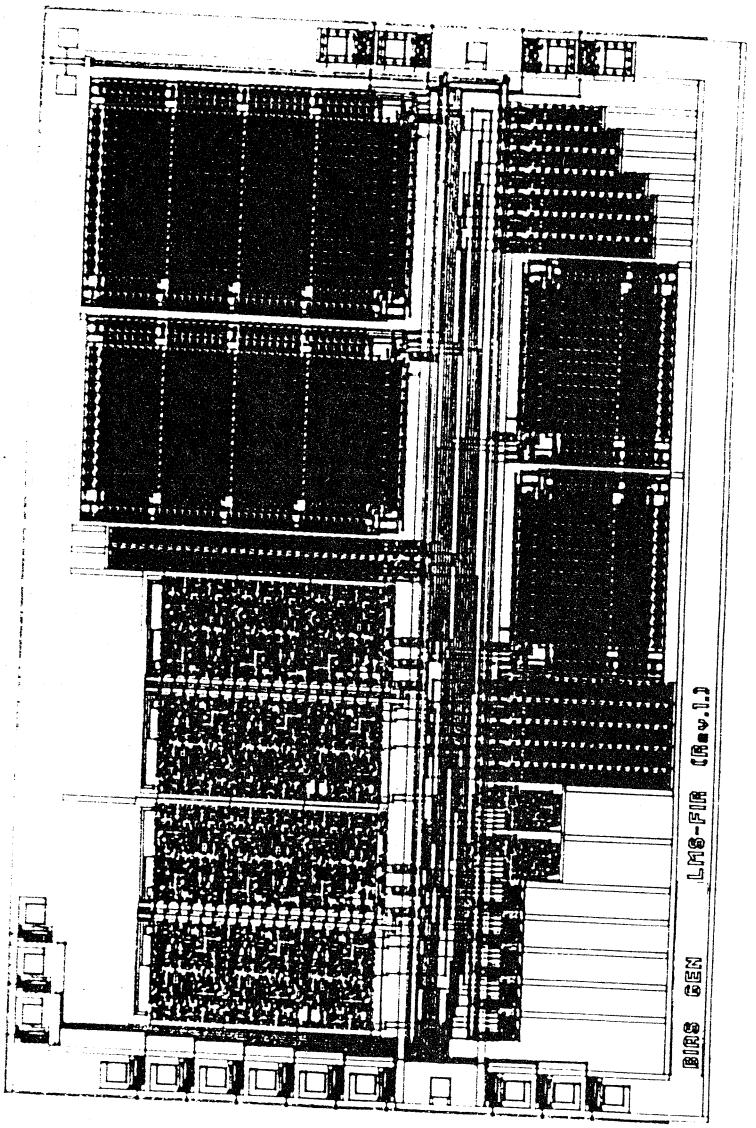


Figure 14.7 (b) Resulting