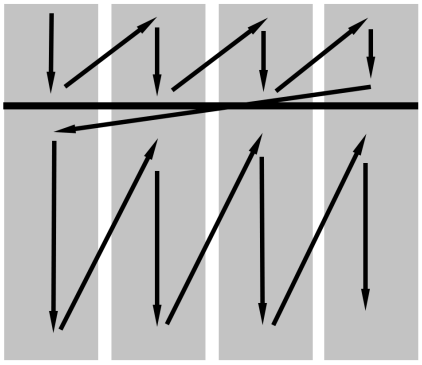
Kommentar:
- Basis: Skript Prof. Rauh (SS2011)
- Der Aufbau des Inhalts orientiert sich am dortigen Inhaltsverzeichnis & den Praktikas
- Obwohl es mehrmals quergelesen wurde, können vorallem im VHDL Bereich immernoch ein paar Fehler stecken
- Es ist (fast) alles drauf, falls euch was fehlt schreibt es in die Lücken. Die Syntax von VHDL wird in der Prüfung gestellt.
- Einige Grafiken dienen lediglich als Gedächnisstützte die man dann mit ein bisschen Wissen auffüllen muss (Platzmangel)
- Druckt die folgenden 4 Seiten mit maximaler Auflösung und der Einstellung im Adobe Reader: „Auf Druckbereich verkleinern“ dann holt ihr das Maximum an Randabstand heraus, alles (!) ist lesbar. Solltet ihr Probleme mit Unschärfe haben braucht ihr evtl einen besseren Drucker oder müsst es halt von Hand dicker schreiben.
- Lesen: Jede Seite besteht aus 4 Spalten, jeweils von oben nach unten bis zum Querstrich bzw. Seitenende lesen dann Wechsel in die nächste Spalte



# 1. Einleitung

Integrierte Schaltung (IC)
= mehrere Bauelemente auf einem Halbleiterkristall (Si, GaAs)
- einige cm^2 (ausbeutebegrenzt) mit einige 10^9 Bauelementen

MEMS = Mikro Electrical Mechanical System (IC mit Aktoren)

Einteilung nach Herstellung, Entwurfsverfahren & Anwendung

Schlüsseltechnolgie, da unverzichtbar für die meisten technischen Geräte und sonst Abhängigkeit von Zulieferung

Aufgabe der Schaltung (Text, Spezifikation)
Logische (1./2.) ↔ Physikalische (3./4./5.) Beschreibungsebene
=>Entwurf (Top - Down)
1.) Verhaltensebene
Verhalten, Fktsblöcke, Schnittstellen, Algorithmus
2.) Logische Ebene
Digitalschaltung, logische Gatter, Zellen
3.) Elektrische Ebene
Analogschaltung, elektronische Bauelemente
4.) Bauelementebene
Halbleiterstruktur, Halbleiterphysik
5.) Technologieebene
Herstellungsprozeß, Festkörperstruktur, Designregeln

Moore's Law: Komplexität der ICs steigt jedes Jahr konstant an
=> Hohe Entwicklungsgeschwindigkeit der ICs
=> Schneller Preisverfall
=> Vorgesehene Entwicklung auf Technology Roadmap (ITRS)

Merkmale der Mikroelektronik (Konsequenz aus Miniaturisierung)
- kompakt (Fläche ↓)
- energiesparend (Verbrauch ↓)
- schnell (Geschwindigkeit ↑)
- zuverlässig (Störanfällig Bauteile ↓)
- preiswert (Preis ↓)

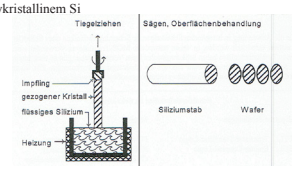
Entwicklungsphasen:
1. Entdeckung (1833): Gleichrichter
2. Theor. Grundlagen (1900): Quantenmechanik, Festkörperphysik
3. Spezielle Theorie(1939) Halbleiterphysik
4. Basisinnovation (1948): Transistor, IC
5. Großind, Nutzung (1970): Mikroeletronik

# 2. Herstellung von ICs

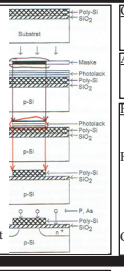
Basistechnologien: Bipolar und MOS Transistoren
=> Herstellung auf einkristallinen, homogen dotierten Halbleitern
=> Bezeichnung des HL:
Substrat (in der Struktur) bzw. Wafer (im Herstellungsprozeß)

Schaltkreisfamilien: TTL, CMOS, (BICMOS=Bipolar und MOS)
Bipolare Schaltungen:
TTL Transistor Transistor Logic
ECL Emitter Coupled Logic
PMOS:P-Channel-Mos-Technology
NMOS:N-Chanel-Mos-Techn.
CMOS:Complementary MOS Tech.

Waferherstellung
Tiegelziehen: Erzeugung von hochreinem einkristallinen Si aus polykristallinem Si
Zersägen in 500µm HLscheiben




Standardherstellverfahren von ICs (Planartechnik):
Prozeßschritte bei der Chiperstellung
1.) Schichterzeugung
Herstellung einer (homogenen Schicht)
2.) Photolithographie
Strukturübertragung von Vorlage/Maske in (Hilfs-)schicht aus Fotolack
3.) Ätzen
Selektives Entfernen einer Schicht, Strukturierung
4.) Dotieren
Ändern Eigenschaften einer Halbleiterschicht




Gehäuse
Kontakt Gehäusepin ↔ Chip (Pad) mittels Bonddraht
=> Drahtgebondete Chips

Abschließende Prozessschritte
Scheibentest => Zerlegung in Chips => Chipmontage => Kontaktierung => Verkapselung => Endtest

Funktionstest (=Vergleich Ist ↔ Soll)
1.) Nach Bearbeitung des Wafers => Scheibentest
2.) Nach Monatage der Chips => Endest
Fehlerklassen:
1.) Lokale Fehler (statisch verteilt) auf einzelnen Chips
z.B. Staub in Herstellung
2.) Globale /Systemmatische Fehler in ganzen Charge
z.B. falsche Prozeßparameter, falsche Maschinen
Güte der Fertigung wird mit Fertigungsausbeute gemessen

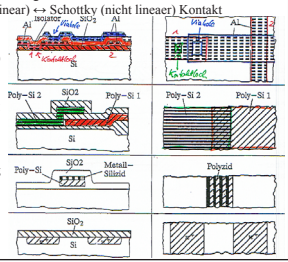
# 3. Bauelemente in ICs

Spezielle Eigenschaften in IC
Aktive Bauelemente billiger => Passive Bauelemente aus Transistor

Verhalten aufteilen in
- Inneren Transistor (eigentliche Funktion)
- Modellrahmen (Parasitäre Komponenten)

Technologie legt vertikale Schichtedicke fest
=> Bauelemente aus charakterischter Schichtung erzeugen
Schaltungsentwurf legt horizontale Abmessungen der Schicht fest
=> Länge (L) & Weite (W)
=> Berücksichtigung der Designregeln des Prozesses
z.B. minimale Abmessungen, Überlappungen
=> Abbildung im Layout (= Maske)

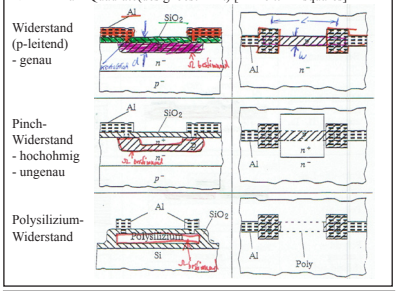
Leiterbahnen
haben endlichen Widerstand und weitere elektr. Eigenschaften
=> Signale mit endlicher Geschwindigkeit und veränderlicher Form
=> Beläge (Größe pro Längeneinheit)
=> Kontakt zur Halbleiter per Kontaktloch in Isolationsschicht
=> Durchkontaktierung mehrerer Leiterebenen mittels VIA-Holes
=> Ohmscher (linear) ↔ Schottky (nicht lineaer) Kontakt



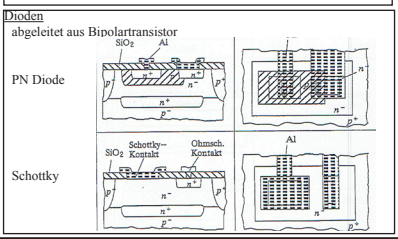

Widerstände

$$R = \varrho \frac{L}{A} = \varrho * \frac{L}{d*W} = \frac{\varrho}{d} \frac{W}{L} = R_{\square} * N_{\square}$$

R □ = Schichtwiderstand [Einheit: Ω/ □ = ohms per square];
N □ = Anzahl Quadrate(designbestimmt) [Einheit: □ =squares]



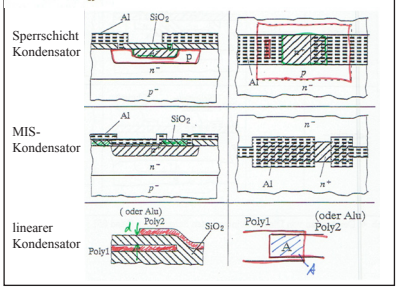

Dioden
abgeleitet aus Bipolartransistor



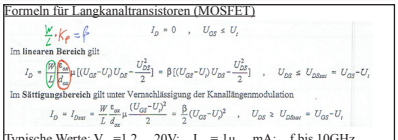

Kondensatoren

$$C = \varepsilon_0 \cdot \varepsilon_r \cdot \frac{A}{d}$$

Nichtlineare Kondensatoren mit V abhängiger C: Sperrschicht, MIS

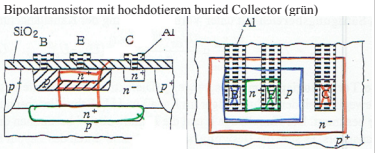



Formeln für Langkanaltransistoren (MOSFET)

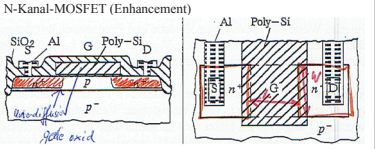


Typische Werte: $V_{DS}$=1,2 ... 20V; $I_{DS}$ = 1µ ... mA; f bis 10GHz
L (kleinst) = 30nm; $d_{ox}$ (kleinst) = 2nm;
Abhängig von Prozesstechnologie:
Dielektrizitätskonstante des Oxids, Oxiddicke d, Beweglichkeit der LT µ und Schwellenspannung $U_T$
Designparameter:
Kanallänge L, Kanalweite W

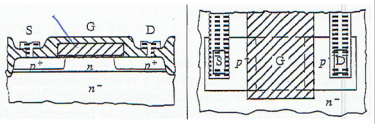
Transistoren
Bipolartransistor mit hochdotierem buried Collector (grün)

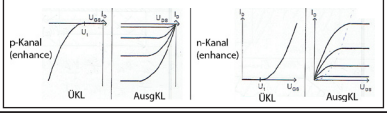



N-Kanal-MOSFET (Enhancement)




P-Kanal-MOSFET (Enhancement)

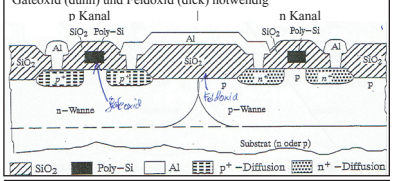


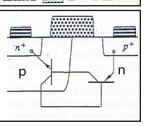

# 4. Schaltungstechnik für IC

Kenngrößen von Schaltkreisfamilien
- Verzögerungszeit τd (Laufzeit von Eingang bis Ausgang)
Einfluss von Gatterlaufzeiten und Leitungslaufzeiten
- Störabstand ΔU (max. zulässige Abweichung vom Nennpegel)
- Verlustleistung $P_v$ (statisch: in R; dynamisch/schalten: in C)
Freisetzung durch Wärme => Wärmestrom => evtl. kühlen
- Energie pro Schaltvorgang $E = P_v * \tau d$

Grundstruktur von CMOS Schaltungen
Verwendung vom zum Substrat komplementären Dotierwannen (tub)
Gateoxid (dünn) und Feldoxid (dick) notwendig



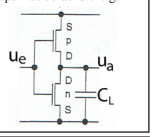

Latch-Up
Erzeugen eines parasitären Thyristors bei CMOS Schaltungen
=> Zünden des Thyristors (= Latch-Up) kann Schaltung zerstören
=> Geeignete Abmessungen / Dotierungen / Schutzmaßnahmen nötig



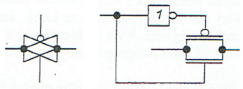

Eigenschaften CMOS Schaltungen
- großer Störabstand (Eingangssignalhub = Vdd)
- großerAusgangssignalhub (Ausgangssignalhub = Vdd)
- geringe Verlustleistung (keine statische Verlustleistung, dynamische Verlustleistung (Querstrom) proportional zur Frequenz)

Mittlere Leistung: $P = C_L * f * V_{DD}^2$
Bei modernen Schaltungen zusätzlich Kurzkanaleffekte
Bei komplexen Schaltungen: Auch Verlustleistung im stationären Zustand (Summe von Gateströmen und Unterschwellströmen)

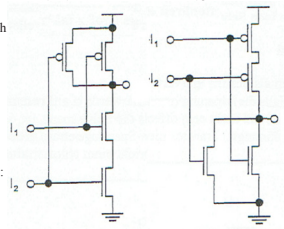
CMOS Inverter
Signalverzögerungen => Verhalten in Lastkapazität berücksichtigt
Aufladen Wechsel L →H (p-Kanal)
Entladen Wechsel H → L(n-Kanal)
=> Ungleichmäßig da LT Beweglichkeit unterschiedlich
=> I(n-Kanal) > I(p-Kanal)
=> Kanalweite p-Kanal vergrößern
Anstiegs-/Abfallszeiten von 10% bis 90%
Verzögerungszeiten zwischen 50% Pegeln



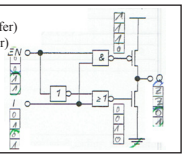

Pass-Transistoren und Transmission Gates
Transistoren als Schalter in Signalpfaden (=Transmission Gate)
=> Aber: Einzelner Transistor kann nicht volles Signal schalten, da unterhalb von $U_T$ nicht aktiv
=> Parellelschalten eines n-&p-Kanal Transistors mit Inverter



Statische CMOS Gatter
Maß für Komplexizität = Gätteräquivalenten (Basis NAND)
NAND Logik:
AND & OR durch anhängen eines Inverters
2 MOSFETS:
Serienschaltung:
p leitend: NOR
n leitend: AND
Parallelschaltung:
p leitend: NAND
n leitend: AND




Tri-State-Treiber
schaltet Eingangssignal(Tri-State-Buffer) oder invertiertes ES (Tri-State-Inverter) durch falls Enable Signal gesetzt
=> Ausgangssignale:
0 / 1 (mit Enable)
Z (hochohmig, kein Enable)




Latch
Bei Clk = 1 Latch speichert
Bei Clk = 0 Latch transparent

Synchrone Schaltungen
Daten durchlaufen in einer, von einem gemeinsamen Takt gesteuerten, Taktperiode eine kombinatorische und eine sequentielle Stufe
Beschreibung: Struktur (Register-Transfer-Ebene) ↔ Verhalten

Zeitbedingungen (alle nötig):
1.) Taktperiode T: $t_{df} + t_{dk} < T - t_{ds}$
2.) max. Frequenz $f_{max} = 1 / T_{min} = 1 / ( t_{df} + t_{dk} + t_{ds} )$
3.) Zeitbedingung: $t_{dj} + t_{dk} > t_{dH}$
FF: $t_{df}$ = Verzögerungszeit
$t_s$ = Set-Up-Zeit (Vor Clk Wechsel)
$t_{dS}$ = Hold-Zeit (Nach Clk Wechsel)
$t_{dH}$
Kombinatorik: $t_{dk}$ = Verzögerungszeit
=> müssen für jede sequentielle Stufe erfüllt sein (einzeln prüfen)

Häufig Probleme in der Realität aufgrund von Taktversatz (clock skew) aufgrund von kapazivtver Last durch Taktleitung
=> baumartige Strukturen mit verteilten Verstärkern (buffer tree) nötig

Ground Bounce = Abweichende Ground Pegel durch induzierte Spannungen in Zuleitungsinduktivitäten, ausgelöst durch Spannungsänderung in der Lastkapazität => Verschiebungen der Gate Spannungen

Zusätzliche Probleme durch Störimpulse (Glitches, Spikes, Hazards oder Races)

Reset Möglichkeit zum initialisieren eines definierten Anfangszustand, per Software oder Hardware nötig, sonst Selbstblockade, Zerstörung (Kurzschluss) möglich

# 5. Typen von IC

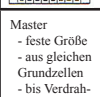
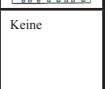
Typeinteilung
Nach Anwendung:
Standard ICs ↔ ASICs
allgemeine Anw. spezielle Anw.
Nach Fertigungsmethode:
Maskenprogrammiert ↔ Feldprogrammiert / PLD
Fkt bei der Herstellung Fkt durch Progammierung (re- und irreversibel)
Nach Entwurfsmthode:
Full-Custom ↔ Zellorientiert
Entwurf bis Layout Entwurf nur bis Logikebene Standardgatter aus Bibliothek
Unterarten Zellorientiere Schaltungen:
- Gate Arrays
- Standard-Zellen
- Allgemeine Zellen
- PLD = Programmable Logic Devices
ASIC = Application Specific Integrated Circuit

Full Custom Schaltungen (Grundlage aller anderen Entwurfsmethoden)
Vorteile:
Volle Optimierungsmöglichkeit nach Chip Fläche, Schaltungsgeschwindigkeit, Leistungsverbrauch, Testbarkeit
Nachteile:
Hoher Aufwand, Lange Entwurfsdauer
=> nur bei hohen Stückzahlen

Zellenorientierter Entwurf
Entwurf nur bis Schaltung aus vorentwickelten Zellen aufgebaut werden kann (auch Semi-Custom-Schaltung)
Zusätzliche Verwendung von Zellen-Generatoren zur parametrisierten Erzeung von komplexen Schaltungsteilen z.B. ALUs
Zusätzliche Verwendung von eingekauften Zellen (IP Cores)

Zelleneigenschaften im Datenblatt (logische Fkt und Zeitverhalten):
fan out darf Output Drive (Ausgangstreiberlast) nicht überschreiten
Last der Zelle für vorhergehende Zelle = Input Load
Vorteile: Entwurfskosten und Entwurfsdauer gering
Nachteile: Weniger Optimiert, Leistung geringer, Stückkosten höher

Unterarten Zellenorientierter Entwurf:

| | Gate Array | Standardzellen | Allgemeine Zellen |
|---|---|---|---|
| |  |  |  |
| Vorfertigung | Master - feste Größe - aus gleichen Grundzellen - bis Verdrahtung fertig | Keine | Keine |
| Layout | - Matrix aus Grundzellen - Zellen aus Blöcken von Grundzellen | - Zellen mit gleiche Höhe aber verschiedener Breite - Reihen aus Zellen - Dazwischen Verdrahtungskanäle | - Zellen sind rechteckförmig mit beliebigen Abmessungen |
| schaltungsspezifische Masken? | nur Verdrahtung | Alle | Alle |
| Fertigung | Verdrahtung | komplett | komplett |
| Ausnutzung HL Fläche | < 100% | fast 100% | fast 100% |

Gate-Array-Schaltungen:
Intrazellverdrahtung = Verdrahtung Grundzellen zu Schaltungszellen
Interzellverdrahtung = Schaltungszellen zu kompletter Schaltung
Typen von Gate Array Master:
- Channeled- Gate-Arrays (Reihenweise Zellen mit Verdrahtungskanälen dazwischen)
- Sea-of-Gate-Arrays (Durchgänge Matrix, Verdrahtung über Zellen hinweg)

Standard-Zellen-Schaltungen:
Zum Entwurf nur Layout Kontur (Zellengröße mit Schnittstellen) nötig

Allgemeine-Zellen-Schaltungen:
Platzierung rechtförmiger Zellen mit belieber Größe (komplexer)

Halbleiterspeicher
da schneller Speicher teuer => Speicherhierachie z.B. PC
1.) Register an ALU => FFs => Zugriffszeit <ns
2.) Cache => SRAM => Zugriffszeit 1...10ns
3.) Hauptspeicher => DRAM => Zugriffszeit 10 ... 100ns
4.) Massenspeicher => magn. optisch => Zugriffszeit 10ms

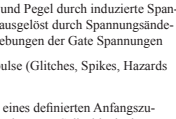
Einteilung(generell):
wahlfreier Speicher(RAM)=Zugriffszeit unabhängig v. Speicherplatz
serieller Zugriffsspeicher =Zugriffszeit abhängig vom Speicherplatz

Flüchtige Speicher (Information gehen beim Ausschalten verloren):
- Dynamische Speicher (Information über Zeitkonstante verloren)
z.B. DRAM, CCD
- Statische Speicher (Information bleibt erhalten)
z.B. SRAM

Nichtflüchtige Speicher (Infos bleiben bei Ausschalten erhalten):
- Nur lesbare Speicher
z.B. ROM
- Beschreibare Speicher
z.B. PROM, EPROM, EEPROM, FLASH

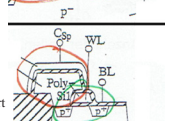
Organisation:
Logischer Zugriff: fortlaufende Speicherplatzadressen
Physischer Zugriff: matrixförmige Anordnung, Zugriff auf Zeile per Wortleitung (Zeilenadresse) und Spalte per Bitleitung (Spaltenadresse)

Dynamische Halbleiterspeicher
Eine Zelle = C & T
C = Speicherkondensator
T = Steuer-/Passtransistor
Planare Eintransistorzelle →



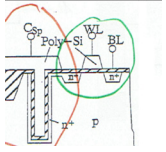

Bitspeicherung als Kondensatorladung

Stapel- oder „Stacked Capacitor" →




Daten müssen regelmäßig regeneriert werden (Refresh)
=> Totzeit von ca. 1%

Graben- oder Trenchzelle →




Organisation
Adressansteuerung per Multiplex
=> RAS und CAS Signale

Lese-Zyklus:
- Vorladen der Bitleitung (precharge)
- Speicherkondensator und Leitung verschalten über Transfer-Gate
- Ladungsausgleich

Zugriffszeit = Zeit zwischen Anlegen der Adresse und Datenausgabe
Zykluszeit = Zeit zwischen 2 Zugriffen auf Speicher
Zykluszeit = ca. 2* Zugriffszeit

Zugriffszeitverbesserung z.B. durch Page-Mode, Static Column Mode, Nibble Mode oder Aufteilung in Speicherbänke

DRAM-Chips häufig als Speichermodule d.h. steckbare Leitplatten mit Ansteuerlogik

## Statische Halbleiterspeicher

Bitinformation als Zustand eines Flip-Flops gespeichert
Verbindung über Auswahltransistoren
Vorteile:
- stationär sehr geringen Stromverbraucht
- schneller als DRAM

Nachteile:
- teuer
- maximale verfügbare Größe geringer als DRAM

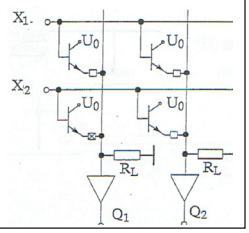
## Nichtflüchtige Halbleiterspeicher

Speicherzelle = Serienschaltung Transitor mit Lastwiderstand zwischen Ground und $V_{DD}$, Ansteuerung über Gate von Worleitung, Bitleitung zwischen Transistor & R
Programmierung durch „Herausnehmen" des Transistors
=> Bei Herstellung = ROM; Im Feld = PROM
=> Irreversibel ↔ Reversibel (EPROM)

## PROMs

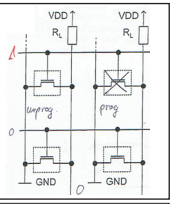
irreversible Programmierung durch Zerschmelzen einer Leiterbahn nach dem Prinzip der Schmelzsicherung (Fuse)
- Noch exsistente Zellen schalten auf Ausgang
- Beispiel hier:
$X_1 = 1$
=> $Q_1 = 0$
=> $Q_2 = 1$

- nur eine Wortleitung ansteuerbar (X)




## ROMs (z.B. als Boot ROM, BIOS)

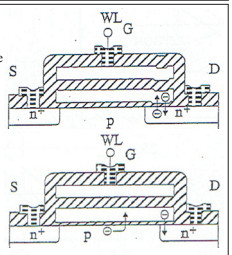
- Maskenprogrammiert bei Herstellung
- Programmierung durch fehlende Verdrahtung eines Transistoranschlußes
- Programmierte / Herausgenommene Zellen schalten auf Ausgang
- Beispiel hier:
$X_{oben} = 1$
=> $Q_{links} = 0$
=> $Q_{rechts} = 1$
- nur eine Wortleitung ansteuerbar (X)



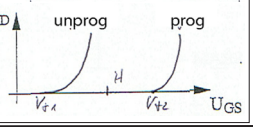

## EEPROMs und FLASH

EEPROM
Programmierung durch tunneln der e ins floating gate (unteres) bei pos. $U_{GD}$
Bitweises Löschen durch neg. $U_{GD}$

FLASH
Programmierung durch „heiße Elektronen" ins floating Gate bei pos. $U_{GD}$
Blockweises Löschen durch neg. $U_{GD}$

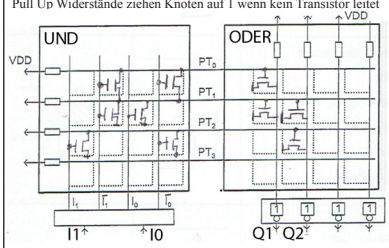



Übertragungskennlinie
Zugriffszeiten
SRAM (0-50ns)
DRAM (20 - 100ns)
EPROM (50-200ns)
FLASH & EEPROM (50 - 200 ns)




## Programmierbare logische Schaltungen (PLD)

komplexer: Field Programmable Gate Array (FPGA)
besteht aus einer UND - und einer ODER Matrix
UND durch Parallelschalten 2er Transistoren & Invertierten Signalen
ODER durch Parallelschalten der 2er Transistoren
Im unprogrammierten Zustand überall Transistoren
Pull Up Widerstände ziehen Knoten auf 1 wenn kein Transistor leitet




Hier $PT_0 = (I_1 \& I_2)$ $PT_1 = (I_1 \& \overline{I_0})$ $PT_2 = (\overline{I_1} \& I_0)$
$PT_3$ = leer $Q_2 = PT_2 \vee PT_1$ $Q_1 = PT_1 \vee PT_0$

## Programmierarten & - prinzipien

- Hardware Sicherung (fuse) - irreversibel - vgl PROM
  = Zerstören einer Verbindung
- Hardware Antisicherung (anti fuse) - irreversibel - wie Kondensator
  = Zerstören einer Isolationsschicht / Herstellen einer Verbindung
- Floating Gate Zellen - reversibel - vgl EEPROM
  = Verschieben der Schwellenspannung
- SRAM Speicher - reversibel - Parallel 2x Inv + Transistor
  = Setzen von FF und Pass-Transistor

## Typen von programmierbaren Schaltungen

SPLDs = Simple PLDs (nur eine UND - & ODER Matrix)
CPLDs = Complex PLDs (mehere UND - & ODER Matrizen & FFs)

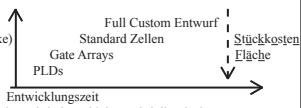
- PAL (Programmable Array Logic) / UND progr. | ODER fest
  PAL / EPLD (Ersable PLD) / EEPLDs (Electricaly Erasable PLD)
- PLA (Programmable Logic Array / UND progr. | ODER progr.
  FPLA (Field PLA)/FPLS (FPL Sequencer)/GAL (Gate Array Logic)
- PROM / UND fest | ODER progr.
  PROM / EPROM / EEPROM

## Field Programmable Gate Arrays (FPLA)

progr. logischen Blöcken (grün)
progr. Verdrahtung (gelb)
Rand: progr. Ein-,Ausgangszellen (rot)
Hardware Prog.: Anti-Fuses
Software Prog: Floating-Gate-Zellen oder SRAM (mit Bootspeicher)

## Auswahl eines Schaltungstyp

Kosten pro Exemplar = (Einmalkosten / Stückzahl) + Stückkosten




= Je mehr optimiert wird, desto kleiner wird die Fläche
=> Je nach Stückzahl unterschiedliche Schaltungstypen wählen

# 6. Entwurf von ICs

Anfangspunkt: Spezifikation des Systems
technische Anforderungen
- Anpassung an Technologie (interne Kompabilität)
- Anpassung an System (Schnittstellenkompabilität)

wirtschaftliche Anforderungen:
- Hohe Entwurfszuverlässigkeit
- Kurze Entwicklungszeit
- Niedrige Entwicklungskosten

Fehlererkennung (nach jedem Entwurfsschritt):
- Verifikation = Korrektheit der Entwicklung
- Validation = Korrektheit bezüglich der Anforderungen / Spezifikation

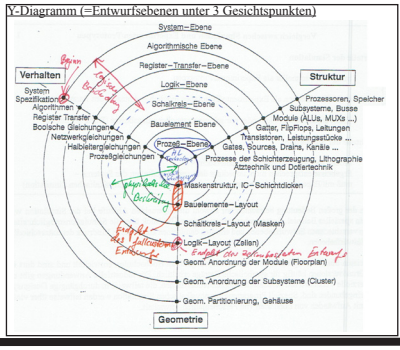
wichtig da: Je früher Fehler im Entwurf erkannt werden, desto günstiger lassen sie sich beseitigen

full custom: Alle Entwurfsebenen
Zellenbasiert: nur logische Ebene

## Entwurfsebenen

| | | | |
|---|---|---|---|
| Entwurf | logische Ebene | Systemebene | Spezifikation, Funktionalität, Partitionierung, Systemkomponenten, Schnittstellen |
| | | Algorithmische Ebene | Verhaltensbeschreibung, abstrahiert, Datenstrukturen, Algorithmen, Progr. Sprache |
| | | Register-Transfer-Ebene | Verhalten, Struktur, techn. Komponenten, endliche Zustandsautomaten, HDL |
| | | Logikebene | Verhalten, Struktur, Gatter (Zellen), Bool Algebra, Bool Variablen, HDL |
| | physikalische Ebene | Schaltkreisebene | elektr. Schaltung, Bauelemente, Kirchhoff Gesetze |
| | | Bauelementebene | HL-Struktur, elektr. Verhalten, HL-Gleichungen |
| | | Prozeßebene | HL-Struktur, Herstellung Polygone, Prozessgleichungen |
| Fertigung | | Maskenherstellung => IC-Herstellung => IC-Test => Montage => Endtest => fertiges Produkt | |

## Y-Diagramm (=Entwurfsebenen unter 3 Gesichtspunkten)

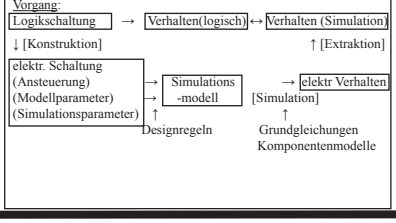



## Simulation (im Vergleich zu Experiment)

Vorteile:
- billiger & schneller
- höhere Anzahl möglich => besseres Abtasten der Parameter
- liefert experimentell nur schwierig zugängliche Ergebnisse

Nachteile:
- Muss nicht umbeedingt der Realität entsprechen da nur Modell

Vorgang:




# 7. Full-Custom-Entwurf

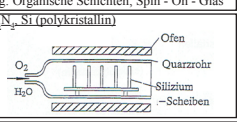
## Schichterzeugung

Materialien:
- Halbleiter: Si (ein- und polykristallin)
- Isolatoren: $SiO_2$, $Si_3N_4$
- Metalle (Leiter): Al, Al/Si, Al/Si/Cu, W, Ti, Ta, Mo
- Silizide (Leiter): $TaSi_2$, $MoSi_2$, $TiSi_2$, $WSi_2$
- Organische Schichten: Resists, Polymid

Herstellverfahren:
- Thermische Oxidation $SiO_2$
- Chemische Abscheidung: Si, $SiO_2$, $Si_3N_4$, Silzide, Refraktärmetall
- Sputter & Aufdampen: Metalle, Silizide
- Schleuderbeschichtung: Organische Schichten; Spin - On - Glas

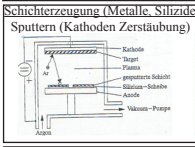
## Schichterzeugung ($SiO_2$, $Si_3N_4$, Si (polykristallin))

1.) Therm. Oxidation
=> Chem. Abscheidungen
=> Chemical Vapour
2.) Deposition(CVD)
3.) Expitaxie

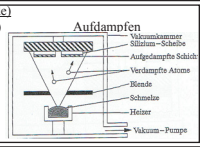



## Schichterzeugung (Metalle, Silizide)

Sputtern (Kathoden Zerstäubung)

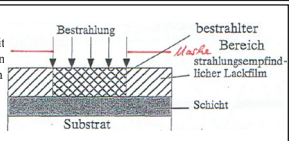



Aufdampfen




## Lithographie

Schritte:
- Bestrahlung mit Licht, Elektronen oder Ionen durch Maske
- Entwicklung
- Ätzprozeß
- Lackstrippen

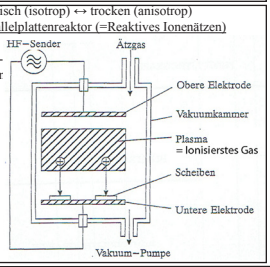



Verfahren:
- Photo- (Linienbreite: 10µm bis 0,1µm > )
- Röntgen-
- Elektronen- (Linienbreite: 2µm bis 0,1 µm > )
- Ionenlithograhpie (Linienbreite: 2µm bis 0,1 µm)

## Ätztechnik: nasschemisch (isotrop) ↔ trocken (anisotrop)

Plasmaätzen im Parallelplattenreaktor (=Reaktives Ionenätzen)

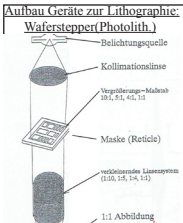
Ionen erzeugen eine Reaktion im Schichtmaterial und wandelr dies in ein Gas um (Kombination aus physik. und chem. Effekten)



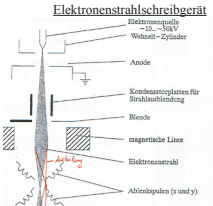

Ätzgase:
für Si, $SiO_2$, $Si_3N_4$:
F, Cl, Br
für Al:
Cl, Br
für Organisches: O

## Aufbau Geräte zur Lithographie:

Waferstepper(Photolith.)




Elektronenstrahlschreibgerät




## Gesamtprozeße (Prozeßintegration):

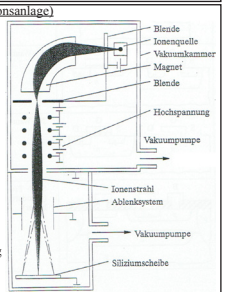
- NMOS = Poly-Si-Gate-Prozess
- PMOS = Al-Gate-Prozess
- CMOS = P- oder N-Wannen-Prozess
- Bipolar = SBC-Process (Standard-Buried-Collector)

## Dotiertechnik (Ionenimplantationsanlage)

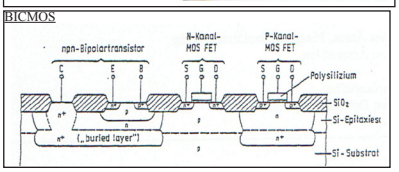
Dotierstoffe:
Bor, Phosphor, Arsen
Diffusion:
therm. Eindringen von Dotieratomen in HLkristall
Ionenimplantation:
Einschießen von Dotieratomen in HLkristall

Oberes Rechteck:
Masseseperator mit Beschleunigung (8 Punkte)

Kristallstruktur wird zerstört
=>Ausheilen (annealing) nötig




## BICMOS




## Prozeß Simulation der Verfahren:

- Schichterzeugung: Oxidation, Chem. Abscheidung, Aufdampfen, Sputtern, Simulation von Aufwachsraten und Schichtprofilen
- Lithographie: Photo-, Elektronen-, Röntgenlith., Simulation von Belichtungszeiten und Lackprofilen
- Ätztechnik: Naßchem. Ätzen, Plasmaunterstütztes Ätzen, Simulation von Ätzprofilen
- Dotiertechnik: Diffusion, Implantation, Simulation von Dotierstoffverteilungen

## CMOS - Prozess (LOCOS-Technik = Local Oxidation of Silicon):

1.) Ausgangsmaterial: n+ dot. Si mit n dot einkrist. Si Schicht
2.) Oxidation der Si Scheibe => $Si_3N_4$ zur lokalen Oxidation
3.) Photolith. für P-Wanne => Ätz=> Ionenimplant. mit Bor
4.) Photoresist Ätz (zur Beseitigung der Lackreste)=> Lokale Oxidation nur in Bereichen die nicht mit $Si_3N_4$ bedeckt waren (=LOCOS)
5.) $Si_3N_4$ Ätz => Ionenimplant. mit Phosphor für N-Wanne
6.) Diffusion der N-Wanne => Ätz=> Oxidation => Ätz
7.) Photolith. mit Maske zur Defintion der Feldoxidbereiche => Ätz
8.) Photoresist Ätz => Photolith zur N-Kanalfeldimplantation => Implantation von Bor
9.) Photoresist Ätz => Lokale Oxidation zur Erzeugung des Feldoxids
10.) Ätz => Ionenimplantation von Bor zur Einstellung der Einsatzspannung der N- und P-Kanal Transistoren
11.) PolySi-Abscheidung und Phosphor Diffusion zur Gateerzeugung => Photolith => PolySi Ätz zur Gateerzeugung => Photoresist Ätz
12.) Photolith zur Dotierung Source/Drain Gebiete der N-Kanal-Trans
13.) Photoresist Ätz => Reoxidation => Photolith zur Dotierung von Source/Drain Gebiete der P-Kanal Trans => Ionenimpl. von Bor zur Source/Drain Erzeugung der P-Kanal-Transistoren
14.) Photoresist Ätz => Photolith für Kontaktlöcher => Kontaktloch Ätzung => Photoresist Ätz
15.) Al Sputtern für Leiterbahnen => Photolith zur Def. der Leiterbahnen => Al Ätzung => Passivierungsschicht Abscheidung (Schaltungsschutz) => Photolith für Kontaktlöcher der Anschlußpads => Kontaktlochätz durch Pass.Schicht zum Freilegen und Bonden

## Layout Entwurf

Bestimmende Größen (wegen Bauelemente Physik, Technologie):
- Auflösungsvermögen der Lithographie
  Strukturfeinheit / Auflösung Δx prop. λ Wellenlänge Licht
- Justiergenauigkeit der Lithographie
  Justierung zwischen zwei Lithographie Ebenen
- Maßänderungen bei der Strukturübertragung
  Maske => Lack; Photolack => geätzte Struktur
- Toleranzen bei technologischen Prozessen
  Photolack-Belichtung, Photolack-Entwicklung, Unterätzung

Designregeln:
- innerhalb Maske: Mindestgröße & -Abstand der Strukturen
- zwischen 2 Masken: Mindesabstände & - Überlappungen

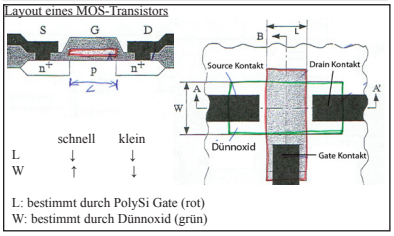
Grundregel: Bei größtmöglicher Dejustierung der Masken und bei max. Versatz irgendwelcher Strukturen soll die Fkt der Schaltung erhalten bleiben und keine gravierende Redukation der Güte auftreten.

Elektr. Designregeln:
- Stromdichte in Al-Leiterbahn (max. $10^5$ A/cm²) - Elektromigration
- Verlustleistungsdichte (punktuell) (max. 100µW/µm²) - Wärmeabfuhr

## Layout Überprüfung:

- Überprüfung Entwurfsregeln (Design Rule Check, DRC)
- Schaltungsextraktion (Circuit Extraction)
- Schaltungsverfikation (Circuit Verification)
  = Simulation der extrahierten Schaltung und Vergleich mit der gewünschten Funktion

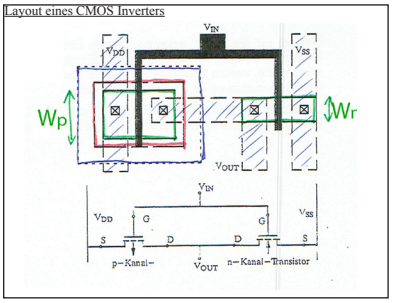
## Layout eines MOS-Transistors



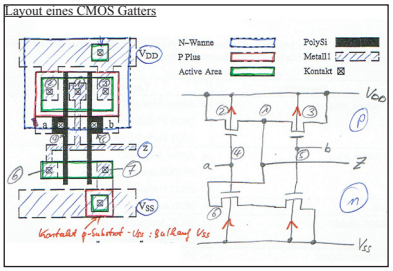

schnell klein
L ↓ ↓
W ↑ ↓

L: bestimmt durch PolySi Gate (rot)
W: bestimmt durch Dünnoxid (grün)

## Layout eines CMOS Inverters




## Layout eines CMOS Gatters




# 8. Zellenbasierter Entwurf

## Digitale Simulation:

- Logiksimulaton - Gatter - durch Boolsche Variablen
- Register-Transfer-Simulation - Register, Kombinatorik - durch Signabündel (Busse), abstrahierte Datentypen
- Verhaltensimulation - Blöcke (funktional, algorithmisch) - abstrahierte Datentypen

## Spezifikation:

= detailierte Beschreibung der zu entwickelnden Schaltung in Form von Texten, Tabellen und Diagrammen
enthält: funktionale Beschreibung der Schalutng & funktionale Angabe (Technologie, Versorgungsspannung, Verlustleistung etc.)

## Logiksynthese und Technology Mapping

Logiksynthese = automatische umwandlung einer Digitalschaltung auf Register-Transfer-Ebene in technologieunabhängige Strukturbeschreibung auf Logikebene
Technologie Mapping = Einbindung der Zieltechnologie in Strukturbeschreibung => technologiespezifische Gatternetzliste

## Vorgehen bei Schaltplaneingabe (mittels grafischer Eingabe)

Schaltungshierarchie festlegen => Grafikeditor aufrufen => Symbole der Schaltungszellen aus Zellenbibliothek aufrufen => Verdrahtung der Zellen zur gewünschten Schaltung innerhalb einer Hierachieebene => Signalbündel zu Bussen zusammenfassen => Beschriftung der Signale und Schaltungszellen => Teilschalten überprüfen => Verbindung der Teilschaltungen in den Hierachieebenen zur Gesamtschaltung => Überprüfen & Abspeichern der Gesamtschaltung

## Testsynthese

- automatisches Einfügen von Testhilfen
  Scan Path, Boundary Scan
- automatische Generation der Testvektoren
  ATPG (Automatic Test Pattern Generation), Anzahl Testvektoren je nach Fehlerüberdeckungsgrad(Testgüte) = Anzahl der testbaren Fehler (mit vorgegebenen Testmustersatz) / Anzahl der möglichen Fehler (nach verwendetem Fehlermodell)
- automatisches Erstellen eines Testprogramms zum zuverlässigen Test der gefertigten Schaltung

## Logiksimulation

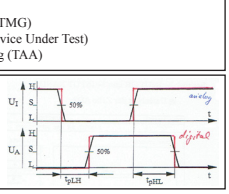
1.) Netzliste der Schaltung (HDL) und Eingangsbitmuster
2.) Logiksimulator
- Schaltungseintrag (Systembibliothek mit Logikmodellen der Zellen, Projektbibliothek mit Logikmodell der gesamten Schaltung)
- Simulation (Layout der Leitungslaufzeiten zur Postsimulation)
- Auswertung (Interaktiv am Bildschirm, Steuerdatei)

3.) Überprüfung ob Schaltungslogikkorrekt, wenn nein zurück zu 1.)

Testbench = HDL Modell der Schaltung wird in eine HDL-Testumgebung integriert; Bestandteile:
- Testmuster erzeugen (TMG)
- Testobjekt (DUT = Device Under Test)
- Testantwortauswertung (TAA)

## Verzögerungszeit Zelle:

$U_I$ = Input
$U_A$ = Ausgang
$t_{pLH}$ = 50% fallend $U_I$ bis 50% steigend $U_A$
$t_{pHL}$ = 50% steigend $U_I$ bis 50% fallend $U_A$




## Verzögerungszeit & Laufzeitberechnung

Zelle:
$t_{pLH} = T_{UP} + K_{UP} * C_L$ $t_{pLH} = T_{DN} + K_{DN} * C_L$
T = Signalverzögerungen ohne angeschlossen Last (Datenblatt)
K = Lastfaktoren (Datenblatt)
$C_{ZELLE} + C_{LEITERBAHNEN} = C_L$ = zu treibende Kapazitive Last = Anzahl an Ausgang angeschlossene Gatter * Last pro Gatter (Load Unit)

## Einflußfaktoren auf Verzögerungszeit der Zelle

$t_D = K * t_{DTYP}$ $K = K_V + K_T + K_P$
$t_D$ = Verzögerungszeit Zelle
$t_{DTYP}$ = Typische Verzögerungszeit der Zelle
K = Korrekturfaktoren (V = für Versorungsspannung; T = für Temperatur; P = für Herstellungsprozeß)
VDD ↑ => I ↑ => $t_D$ ↓ T ↑ => µ ↓ => I ↓ => $t_D$ ↑

## Leiterbahnlaufzeit (R-, C-Belag pro Längeneinheit)

Schichten Leiter + $SiO_2$ + Si = pro LE $U_I$ über C // (R + $U_A$ + R)
$T_{dLH} = k * t_{LH} * l$ (VerzZeit ab Sprung $U_I$ => $U_A$ 10% auf 90%)
$T_{dHL} = k * t_{LH} * l$ (VerzZeit ab Sprung $U_I$ => $U_A$ 90% auf 10%)
l = Leiterbahnlänge
$t_{LL}$ = Verzögerungszeiten pro Längeneinheit
K = Korrekturfaktor = f ($U_B$, T, Technologie)

# 9. VHDL

## Allgemeines

HDL = Hardware Description Language
- Erlauben Beschreibung der Struktur oder des Verhaltens von Digitalschaltungen auf den logischen Ebenen in unterschiedlichen Abstraktionsstufen (Verhalten & Struktur auf System-, Algorithmischer-, Register-Transfer- & Logikebene vgl Y-Diagramm)
- Entspricht einem Schaltplan bzw. dessen Aufbau aus Komponenten
- VHDL = VHSIC Hardware Description Language
- VHSIC = Very High Speed Integrated Circuits
- Gehört zur ALGOL-PASCAL-Sprachfamilie
- Nicht case sensitiv
- Programmteile die in mehreren Beschreibung benötigt werden können in einem Package zusammengefaßt werden

## Programmstruktur

Schaltungsbeschreibung:
- entity:
  Schaltungseinheit, Name, Schnittstellen zur Umgebung
- architecture:
  „Inhalt“, bestimmt Relation zwischen Eingangs- und Ausgangssignalen, beschreibbar als:
  - Struktur (structural)
  - Datenfluß (behavior, concurrent, dataflow)
  - Verhalten (behavior, sequential)
- configuration: Verbindung zwischen Entity und Architecture

Unterprogramme:
- procedure (mit sequentiellen Anweisungen)
- function (mit sequentiellen Anweisungen)

## Entity

- Beschreibung als Schaltungseinheit
- Entspricht im Schaltplan dem Symbol für die Gesamtschaltung
- Enthält:
  - Name zur Identifikation in Schaltung = Entity Name
  - Schnittstellen der Schaltung = Ports
  - Konstanten zur Parametrisierung = Generics

## Architecture

- Beschreibung der Schaltung selbst
- Entspricht Schaltplan zum zugehörigen Symbol
- Identifikation über Architecture Namen

## Strukturelle (structural) Beschreibung:

- Entspricht Beschreibung durch Schaltplan
- Beschreibung durch Komponenten und deren Verschaltung

## Parallele (concurrent) Verhaltenbeschreibung:

- Entspricht der tatsächlichen Funktion der Digitalen Schaltung
- Wert des Signals wird als Funktion der Werte seines Vorgängersignals beschrieben (=sensitiv)
- Auswertung unabhängig von Reihenfolge im Programm
- Zeitliches Verhalten durch Zeitangaben beschrieben
- Bezeichnungen: nebenläufig oder datenflußorientiert

## Sequentielle (sequential) Verhaltensbeschreibung:

- Entspricht Beschreibung durch Algorithmus wie in prozeduraler Programmiersprache
- Anweisungen werden in Reihenfolge wie im Programm abgearbeitet (sequentiell)
- Verwendung innerhalb einer Prozeßanweisung
- Syncronisation durch geeignete Konstrukte nötig

## Notation:

::= Ersetzungszeichen
| trennt Alternativen
[ ] schließen optionale Teile ein
{ } schließen optionale Teile ein, die wiederholt werden können
-- Kommentare

## Grundsätzlicher Programmaufbau:

Mindestens ein Entity und eine Architecture

```
entity entity is                        -- Schnittstellen der Schaltung
    -- port declaration:                -- Signale, keine Variablen
    port (port:mode type { , port : mode type } ) ;
end;

architecture architecture of entity is  -- „Inhalt“ der Schaltung
    -- signal_declaration:              -- Signale, keine Variablen
    signal signal : type := initial_value;
    -- component declarations           -- für Struktur
    component component port (port_list) ; end component ;
begin
    -- concurrent_statements
    -- concurrent_signal_assignments :
    signal <= waveform ;
    -- conditional_signal_assigment :
    signal <= {waveform when cond else} waveform;
    -- selected_signal_assigment
    with expre select signal <= waveform when choices
                              { ,  waveform when choices};
    -- process_statement                -- Verhalten sequentiell
    -- component_instantiation_statement -- Struktur
    label : component port map (port_association_list) ;
end ;

Sequentielle Beschreibungen in einem Prozess
process(signal { , signal } )           -- Sensitivity-Liste (Datenfluß)
    -- variable_declaration:            -- Variable, keine Signale
    variable variable : type := initial_value;
begin
    -- sequential_statements:
    -- wait:
    wait on signal ;                    -- Schnittstelle Datenfluß
    wait until condition;
    wait for time;
    -- signal_assignment:
    signal <= waveform ;                -- Schnittstelle Datenfluß
    -- variable_assigment:
    variable := expr;
    -- if:
    if cond then stmt { elsif cond then stmt } { else stmt } end if;
    -- case:
    case expr is { when choices => stmt } end case;
    -- loop
    -- ...
end process;
```

## Datentyp Standard Logic

Datentyp zur Beschreibung von Schaltungen.
Im Package ieee.std_logic_1164 enhalten
Zustände (Auswahl):
- ‘U‘ = uninitialized (um Zustand bis zum ersten Setzen / Rücksetzen zu definieren)
- ‘Z‘ = high impedance
- ‘0‘ = Binär 0
- ‘1‘ = Binär 1

Möglichkeiten zur Signalwertbestimmung bei mehreren Quellen

## Inverter (Parallele Verhaltensbeschreibung)

```
entity INV_E is
    generic (DELAY_T:time:=1 ns);
    port (I: in bit; Q: out bit);                   -- einzelne Ports
end INV_E;

architecture INV_A of INV_E is
begin
Q <= not I after DELAY_T;
end INV_A;
```

## Testbench

- Testbench zur Schaltungssimulation
- Eingabedaten: Schaltung selbst
- Simulationsmodelle der Komponenten & zeitlicher Verlauf der Eingangsignale (Stimuli)
- Ausgabedaten: Zeitlicher Verlauf der Ausgangssignale
- Zweck: Vergleich des Verhaltens der beschriebenen Schaltung mit der Spezifikation
- Testumgebung (Testbench) enhält:
  - Erzeugung der Stimuli
  - Einbindung der zu testenden Schaltung (= dut)
  - Auswertung der Ausgangssignale

## Testbench für Inverter (strukturelle Beschreibung; Inverter siehe oben)

```
entity INV_TB_E is
    generic (clock_T:time:=5 ns)
end INV_TB_E;

architecture INV_TB_A of INV_TB_E
    component INV_E                 -- component declaration
        port (I: in bit ;
              Q: out bit);
    end component;
    signal I_S, Q_S, bit;           -- signal declaration
begin
    DUT: INV_E
        port map (I=>I_S, Q->Q_S); --port_map(I_S, Q_S)
          I_S <='0' after 0*clock_T;
                '1' after 1*clock_T;
                '0' after 2*clock_T;
end INV_TB_A;
```

Einbindung des Inverters durch Bezug in der component declaration

## NOR Gatter (Parallele Verhaltensbeschreibung)

```
entity NOR_E ist
    generic (DELAY_T: time:= 1 ns);
    port (I, J : in bit;
          Q: out bit;
end NOR_E;

architecture NOR_A of NOR_E is
begin
    Q<=I nor J after DELAY_T;
end NOR_A;
```

## Testbench für NOR Gatter (Sequentielle Verhaltensbeschreibung)

(Erzeugung aller möglichen Eingangswertkombinationen durch 2 verschachtelte for Schleifen)

```
library work;
use work.my_pack.all;

entity NOR_TB_E is
    generic (CLOCK_T: time:= CLOCK_TC);
end NOR_TB_E;

architecture NOR_TB_A of NOR_TB_E is
    component NOR_E
        generic (DELAY_T: time);
        port (I, J : in bit;
              Q  : out bit);
    end component;
    signal I_S, J_S, Q_S: bit;
begin
    DUT: NOR_E
        generic map(DELAY_T => DELAY_TC)
        port map(I=>I_S, J=> J_S, Q=>Q_S);
STIMULI: process
begin
    for II in 0 to 1 loop
        for JJ in 0 to 1 loop
            I_s <= INT_TO_BIT (II)
            J_s <= INT_TO_BIT (JJ)
            wait for CLOCK_T;
        end loop;
    end loop;
    wait;
end process STIMULI;
end NOR_TB_A;
```

## Package MY_PACK (Umwandlungsfunktionen)

```
library ieee;
use ieee.std_logic_1164.all;

package MY_PACK is
    constant DELAY_TC: time:= 1 ns;
    constant CLOCK_TC: time:= 10 ns;
    function INT_TO_BIT(II: integer) return bit;
    function INT_TO_STD(II: integer) return std_ulogic;
    function BIT_TO_STD(B:bit) return std_ulogic;
    function STD_TO_BIT(S:std_ulogic) return bit;
end MY_PACK;

package body MY_PACK is

function INT_TO_BIT(II: integer) return bit
is
begin
    if II=0 then
        return '0';
    else
        return '1';
    end if;
end INT_TO_BIT;
--restliche Funktionen analog
```

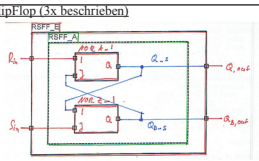
## OR Gatter (als NOR mit Inverter; Strukturelle Beschreibung)

```
entity OR_E is
    generic (DELAY_T: time:=1 ns);
    port (I, J : in bit;
          Q  : out bit);
end OR_E;

architecture OR_A of OR_E is
    component NOR_E
        port (I,J : in bit;
              Q : out bit)
    end component;
    component INV_E
        port(I: in bit;
             Q: out bit);
    end component;

    signal NOR_S: bit;
begin
NOR_I : NOR _E
    port map (I => I, J=> J, Q=>NOR_S);
INV_I : INV_E
    port map (I =>NOR_S, Q=>Q);
end OR_A
```

## Testbench für das OR-GATTER (inkl. Auswertung EVALUATE)

```
library work;
use work.my_pack.all;

entity OR_TB_E is
    generic (CLOCK_T: time:= CLOCK_TC);
end OR_TB_E;

architecture OR_TB_A of OR_TB_E is
    component OR_E
        generic (DELAY_T: time);
        port (I,J: in bit; Q: in bit);
    end component;
    signal I_S, J_S, Q_S: bit;
    signal SUCC_S, FAIL_S: bit:= '0';
begin
    DUT: OR_E
        generic map (DELAY_T => DELAY_TC)
        port map(I_S, J_S, Q_S);
    STIMULI: process
    begin
        for II in 0 to 1 loop
            for JJ in 0 to 1 loop
                I_S <= INT_TO_BIT(II);
                J_S <= INT_TO_BIT(JJ);
                wait for CLOCK_T;
            end loop;
        end loop;
        wait;
    end process STIMULI;
    EVALUATE: process
    begin
        wait for CLOCK_T/2;
        if Q_S /= (I_S or J_S) then
            FAIL_S <= '1';
        else
            SUCC_S <= '1';
        end if
        assert Q_S=(I_S or J_S)
        report "Fehler im ODER-Gatter"
        severity error;
        assert Q_S/=(I_S or J_S)
        report "Testfall ok";
        severety note;
        wait for Clock_T/2;
        FAIL_S <= '0';
        SUCC_S <= '0';
    end process EVALUATE;
end OR_TB_A;
```

## RS-FlipFlop (3x beschrieben)




RSFF_STRUCT_A ist strukturelle Beschreibung (aus 2x NOR)
RSFF_CONCUR_A ist parallele Verhaltensbeschreibung
RSFF_SEQU_A ist sequentielle Verhaltensbeschreibung

```
entity RSFF_E is
    generic (DELAY_T: time:= 1 ns);
    port (R, S: in bit; Q, QB: out bit);
end RSFF_E

architecture RSFF_STRUCT_A of RSFF_E is          -- strukturell
    component NOR_E
        port (I, J: in bit; Q: out bit);
    end component;
signal Q_S, QB_S : bit;
begin
    NOR1_I: NOR_E
        portmap(R , QB_S , Q_S);
    NOR2_I: NOR_E
        portmap(Q_S, S_ QB_S);
    Q<=Q_S;
    QB<= QB_S;
end RSFF_STRUCT_A;

architecture RSFF_CONCUR_A of RSFF_E is          -- parallele
    signal Q_S, QB_S : bit;
begin
    Q_S <= R nor QB_S after DELAY_T;
    QB_S <= Q_S nor S after DELAY_T;
    Q <= Q_S;
    QB <= QB_S
end RSFF_CONCUR_A

architecture RS_FF_SEQU_A of RSFF_E is           -- sequentiell
begin
    process (R,S)
    begin
        if S = '1' and R = '0' then              -- setzen
            Q  <= '1' after 2*DELAY_T;
            QB<= '0' after 1*DELAY_T;
        else if S = '0' and R = '1' then         -- rücksetzen
            Q  <= '0' after 1*DELAY_T;
            QB<= '1' after 2*DELAY_T;
        else if S = '1' and R = '1' then         -- ungültig
            Q  <= '0' after 1*DELAY_T;
            QB<= '0' after 1*DELAY_T;
        end if
    end process;
end RSFF_SEQU_A;
```

## Testbench für RS Flipflop

```
library work;
use work.my_pack.all

entity RSFF_TB_E is
    generic (CLOCK_T: time:= 10ns);
end RSFF_TB_E

architecture RSFF_TB_A of RSFF_TB_E is
    component RSFF_E
        port(R,S: in bit; Q, QB : out bit);
    end component;
    signal R_S, S_S, Q_S, QB_S: bit;
begin
    DUT: RSFF_E
        port map (R => R_S, S=>S_S, Q=>Q_S, QB=>QB_S);
    R_S <=      '0' after 0*CLOCK_T,
                '1' after 3*CLOCK_T,
                '0' after 4*CLOCK_T,
                '1' after 5*CLOCK_T,
                '0' after 6*CLOCK_T;
    S_S <=      '0' after 0*CLOCK_T,
                '1' after 1*CLOCK_T,
                '0' after 2*CLOCK_T;
    end RSFF_TB_A;
configuration RSFF_TB_STRUCT_C of RSFF_TB_E is
    for RSFF_TB_A
        for DUT : RSFF_E use
            entity work.RSFF_E(RSFF_STRUCT_A);
        end for;
    end for;
end RSFF_TB_STRUCT_C;
configuration RSFF_TB_CONCUR_C of RSFF_TB_E is
    for RSFF_TB_A
        for DUT : RSFF_E use
```

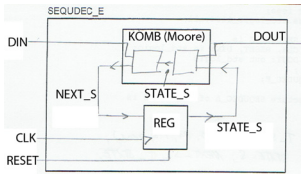
## Fortsetzung Testbench RSFF

```
            entity work.RSFF_E(RSFF_CONCUR_A);
        end for;
    end for;
end RSFF_TB_CONCUR_C;
configuration RSFF_TB_SEQU_C of RSFF_TB_E is
    for RSFF_TB_A
        for DUT : RSFF_E use
            entity work.RSFF_E(RSFF_SEQU_A);
        end for;
    end for;
end RSFF_TB_SEQU_C;
```

## Register (aus flankengesteuerten D-FFs mit asynchronem Reset)

```
library ieee;
use ieee.std_logic_1164.all;

entity Register_E is
    generic (DELAY_T: time := 1 ns;
             NBITS: natural := 8);
    port (CLK, RESET: in std_logic;
          D: in std_logic_vector(NBITS-1 downto 0);
          Q: out std_logic_vector(NBITS-1 downto 0));
end Register_E;

architecture REGISTER_A of REGISTER_E is

begin
process (CLK, RESET)
    if RESET= '1' then
        Q <= (others => '0')                     -- Kurzform
        -- for I NBITS-1 downto 0 loop           -- Langform
            -- Q(I) <= '0';
        -- end loop;
        elsif CLK 'event' and CLK = '1' and CLK 'lastvalue' = '0'
        then
            Q<=D;
        end if;
end process;
end REGISTER_A;
```

## Finate State Machine (4 Zustände, aus Register & Kombinatorik zur Sequenzdetektion 101)

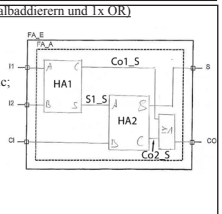



```
library ieee;
use iee.std_logic_1164.all;

entity SEQUDEC_E is
    port (CLK, RESET, DIN: in std_logic;
          DOUT: out std_logic);
end SEQUDEC_E

architecture SEQUDEC_A of SEQUDEC_E is

type T_STATE is (P0, P1, P10, P101);             -- Zustände
signal STATE_S, NEXT_S, T_STATE;

begin
    COMB: process(DIN,STATE_S)
    begin
        DOUT <= '0';
        NEXT_S <= P0;
        case STATE_S is
            when P0 =>
                if DIN = '1'
                    then NEXT_S <= P1;
                    else NEXT_S <= P0;
                endif;
            when P1 =>
                if DIN = '0'
                    then NEXT_S <= P10;
                    else NEXT_S <= P1;
                endif;
            when P10 =>
                if DIN = '1'
                    then NEXT_S <= P101;
                    else NEXT_S <= P0;
                endif;
            when P101 =>
                if DIN = '1'
                    then NEXT_S <= P0;
                    else NEXT_S <= P1;
                endif;
            DOUT <= '1';
            when others => NEXT_S <= P0;
        end case;
    end process COMB;

    REG: process (CLK, RESET)
        begin
            if CLK 'event' and CLK = '1' then
                if RESET = '1' then
                    STATE_S <= P0;
                else
                    STATE_S <= NEXT_S;
                end if;
            end if;
        end process REG;
end SEQUDEC_A;
```

## Halbaddierer (Parallele)

```
library ieee;
use ieee.std_logic_1164.all;

entity HA_E is
    port (I1, I2: in std_logic;
          S, Co: out std_logic);
end HA_E;

architecture HA_A of HA_E is
begin
    S <= I1 xor I2;
    Co <= I1 and I2;
end HA_A;
```

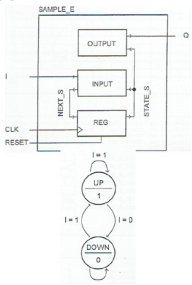
## Volladdierer (Parallele aus 2 Halbaddierern und 1x OR)




```
library ieee;
use ieee.std_logic_1164.all;

entitity FA_E is
    port (I1, I2, Ci : in std_logic;
          S, Co: out std_logic);
end FA_E;

architecture FA_A of FA_E is

component HA_E
    port (I1, I2: in std_logic;
          S, Co: out std_logic);
end component;

signal Co1_S, S1_S, Co2_S : std_logic;

begin
HA_1 : HA_E
    port map (I1 => I1, I2 => I2, Co => Co1_S, S => S1_S);
HA_2 : HA_E
    port map (I1 => S1_S, I2 => Ci, Co => Co2_S, S => S);

    Co <= Co1_S or Co2_S;
end FA_A;
```

Weitere Aspekte von Finite State Machines

```
library ieee;
use ieee.std_logic_1164.all;

entity sample_e ist
      port (i: in std_logic; clk,reset :
      in std_logic; q : out std_logic);
end sample_e;

architecture sample_a of sample_e is
      type state_t is (up,down);
      signal state_s, next_s: state_t;
begin
input: process(i, state_s)
begin
      case state_s is
         when down =>
            if i='1' then
               next_s <= up;
            else
               next_s <= down;
            end if;
         when up =>
            if i='1' then
               next_s <= up;
            else
               next_s <= down;
            end if;
      end case
end process input;
```







zu weitere Aspekte von Finite State Machines

```
output: process(state_s)
begin
      case state_s is
         when down =>
            q <= '0';
         when up =>
            q <= '1';
      end case;
end process output;

reg: process(clk, reset)
begin
      if reset = '1' then
         state_s <= down;
      elsif rising_edge(clk) then
         state_s <= next_s;
      end if;
end process reg;

end architecture sample_a;
```

Short (Bei Doppelzuweisung wird x als Wert gesetzt)

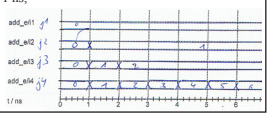
```
library ieee;
use ieee.std_logic_1164.all;

entity SHORT_E is
end SHORT_E;

architecture SHORT_A of SHORT_E is
      signal S : STD_LOGIC;
begin
      S<= '1';
      S<= '0';
end SHORT_A;
```

Taktsignale erzeugen (2 Möglichkeiten)

```
library ieee;
use ieee.std_logic_1164.all;

entity CLK_E is
end CLK_E;

architecture CLK_A of CLK_E is
      signal CLK1, CLK2 : std_logic := '0';

begin
      CLK1 <= not CLK1 after 1 ns;   -- Möglichkeit 1
end process
      begin                          -- Möglichkeit 2
         CLK2 <= '0';
         wait for 1 ns;
         CLK2 <= '1';
         wait for 1 ns;
      end process;
end clk_a;
```

Add (Prozessauslösungen)

```
library ieee;
use ieee.std_logic_1164.all;

entity ADD_E is
end ADD_E

architecture ADD_A of ADD_E is
      signal I1, I2, I3, I4, J1, J2, J3, J4, K1, K2, K3, K4, L: integer :=0;

begin
      I2 <= I1+1 after 1 ns;
      I3 <= I2+1 after 1 ns;
      I4 <= I4+1 after 1 ns;
process (J1, J2, J3, J4)
begin
      J2 <= J1+1 after 1 ns;
      J3 <= J2+1 after 1 ns;
      J4 <= J4+1 after 1 ns;
end process;
process (K1)
begin
      K2 <= K1+1 after 1 ns;
      K3 <= K2+1 after 1 ns;
      K4 <= K4+1 after 1 ns;
end process;
end ADD_A;
```




# 10. Test von ICs

Motivation zum Testen

Fehlerbeseitigung umso teurer, je später der Fehler erkannt und behoben wird (exponentielle Funktion)

Prinzip

Testmuster (Testdaten) in

Testobjekt — Referenzobjekt

=> Ist-Testantwort — Soll-Testantwort (Testdaten)

Vergleich: Ist = Soll Testantwort?

=> Gut oder Schlecht Fall

Tests während der Lebensdauer einer Schaltung

| Phase | Test | Testart | Fehlerart | Charakteristik |
|---|---|---|---|---|
| Entwurf | Entwurfstest | Simulation | Entwurfs-fehler | systematisch |
| Fertigung | Fertigungstest | elektr. logisch | Fertigungs-fehler | statistisch |
| Betrieb | Betriebstest | log. Verhalten | Verschleiß Betriebs-störungen | zeitabhängig statistisch |

Fehler durch Alterung & Verschleiß ergeben „Badewannenkurve“ (Anfangs viele Ausfälle durch falsche Parameter, Ende viele Ausfälle durch Alterung und Verschleiß)

Maß für Fertigungsfehler: Fertigungsausbeute

= Anzahl fehlerfreie Chips / Anzahl gefertigter Chips

Spezielle Probleme beim Test von ICs

Kompletter Test aller Bitmuster kann sehr aufwendig sein

=> Umwandlung in Fehlermodell mit Aussage über Testgüte

=> Messung nur an Schaltungsknoten auf korrekten Pegel

=> Komplexität sinkt; Aber: Durch Miniaturisierung Einstellbarkeit und Beobachtbarkeit der internen Knoten schlecht (Strukturverkleinerung um 1/2 => Anzahl an zu testenden Knoten * 2)

=> Anstieg der Testmusterzahl => Anstieg der Testkosten

=> Tests müssen schon beim Entwurf berücksichtigt werden = Design for Testability (DFT)

Fertigungstests

Entwurfsfehler = systemmatisch d.h.Fehler in allen Schaltung gleich

Fehlerklassen:

| Fehlerart | Beispiele | Abdeckung Fehlermodelle |
|---|---|---|
| statisch | Unterbrechung, Kurzschluss | groß |
| dynamisch | zu kurze oder zu lange Signallaufzeiten | gering |
| parametrisch | Strom-&Spannungsabweichungen | gering |

Statischer Test

= In einem Testzyklus wird an alle Eingänge der Schaltung gleichzeitig ein Testmuster angelegt. Zu einem festgelegten Zeitpunkt danach, wenn Ausgänge stabil sind, werden Ausgangssignale mit Abfrage-Strobe ausgelesen und Ist-Testantwort ausgewertet. Danach neuer Testzyklus.

Dynamischer Test

z.B. Messungen: Laufzeit, Phase (Bestimmung des Zeitlichen Versatzes zwischen Signalen z.B. Setup Zeit), Clockrate Test (Bestimmung Minimal- und Maximalfrequenzen)

Parametertest

z.B. Messungen: statische und dynamische Stromaufnahme, Treiberfähigkeit der Ein- und Ausgangszellen, Ströme, Spannungen und Kennlinien ausgewählter Bauelemente (z.B. Leckströme)

Fehlersimulation

Fehlerüberdeckungsgrad = Anzahl (mit Testdatensatz) erkennbarer Fehler / Anzahl (mit Fehlermodell) modellierten Fehler

Stuck-at-Fehlermodell

2 Fehlerarten:

- Stuck-at-0: Knoten liegt ständig auf logisch 0
- Stuck-at-1: Knoten liegt ständig auf logisch 1

=> Anzahl möglicher Fehler einer Schaltung: Schaltungsknoten * 2

Prinzip der Fehlersimulation

= Bestimmen des Fehlerüberdeckungsgrades eines Testmusters

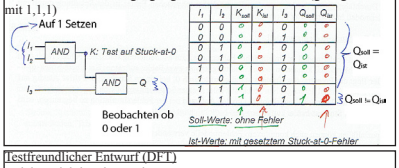
1.) Simulation der fehlerfreien Schaltung -> Soll-Testantwort

2.) Wiederholen für alle (nach Fehlermodell)

- Setzen eines Stuck-at-Fehlers
- Simulation der fehlerbehafteten Schaltung -> Ist-Testantwort
- Bei Abweichung: Bitmuster zum finden des Fehlers in der gefertigten Schaltung

3.) Aus Ergebnissen ergibt sich Fehlerüberdeckungsgrad, zum erreichen dieses kann Anzahl Testmuster erhöht werden oder Schaltung testfreundlicher gemacht werden

Bsp: Um Stuck-at-Fehler zu finden (hier Test auf 0 bei K) muss der Knoten auf den entgegengesetzten Pegel eingestellt werden (d.h. K auf 1) und über die Ausgangssignale beobachtet werden. (gelingt hier mit 1,1,1)




| $I_1$ | $I_2$ | $K_{ist}$ | $K_{st}$ | $I_3$ | $Q_{ist}$ | $Q_{st}$ |
|---|---|---|---|---|---|---|
| 0 | 0 | o | o | 0 | o | o |
| 0 | 0 | o | o | 1 | o | o |
| 0 | 1 | o | o | 0 | o | o |
| 0 | 1 | o | o | 1 | o | o |
| 1 | 0 | o | o | 0 | o | o |
| 1 | 0 | o | o | 1 | o | o |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 |

$Q_{soll} = Q_{ist}$

$Q_{soll} \neq Q_{ist}$

Soll-Werte: ohne Fehler

Ist-Werte: mit gesetztem Stuck-at-0-Fehler

Testfreundlicher Entwurf (DFT)

Ziel: Reduktion der Testkosten

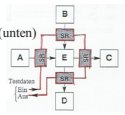
- zusätzlicher Hardware-Aufwand nötig
- 2 Betriesbmodi: Normalbetrieb und Testbetrieb
- Ansätze zu DFT: Passive Testhilfen ↔ aktiver Selbsttest

Passive Testhilfen

- Ad-Hoc-Methoden
  - Hinzufügen von Testpunkten
  - Aufteilung der Schaltung
  - Auftrennung von langen Zählerketten
  - Unterbrechung von Rückkopplungspfaden
  - Hinzufügen von Testpunkten
  - = Seperate Testleitungen für schlecht testbare Knoten
    => Signale einstellen oder beobachten

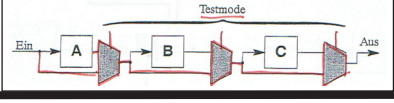
Aufteilung der Schaltung

mit Schieberegistern (rechts) oder Multiplexern (unten)




Gesichtspunkte:

- Überschaubere gut testbare Blöcke
- Funktionsmodule
- Systematisch generierte Blöcke
- Blöcke gemäß ihrer logischen Struktur




zu DFT

Aufteilung in überschaubare Blöcke:

Test jedes Ausgangs einzelnd in Abhängigkeit seiner Eingangsignale

=> Reduktion der Testvektoren

Auftrennung von langen Zählerketten:

MUX zwischenschalten

vorderer Schaltungsteil => Testdaten Ausgangs & Eingang MUX

Testendateneingang => 2. Eingang MUX

Testmode => Steuerender MUX Eingang

Unterbrechung von Rückkopplungspfaden

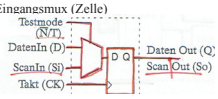
Zwischenschalten von MUX um sequentielles Verhalten in rein kombinatorische Blöcke umwandeln => leichter testbar

Prüfbus (Scan Path)

Prinzip: Trennung in sequentielle und kombinatorische Teile

=> alle FFs benötigen einen Eingangesmux (Zelle)

Testablauf:




1.) Test der Schieberegister

- Scan-Path FFs in Testmode schalten (T=1)
- Ein-&Ausschieben der Testmuster

2.) Test der Kombinatorik

- Scan-Path FFs in Testmode schalten
- Serielles Laden der Register mit Testmuster
- Anlegen der Testmuster an primäre Eingängeder Schaltung
- Umschalten der Scan-Path FFs in Betriebsmodus
- Übernahme der Testantworten der Kombinatorischen Schaltungsteile in die Scan-Path FFs mit einem Takt
- Testantworten der Primärausgänge mit Sollwerten vergleichen
- Scan-Path FFs wieder in Testmode schalten
- Testantwort aus dem Scan-Path rausschieben und bitweise mit Sollwerten vergleichen

Verfahrensarten:

- LSSD (level sensitive scan design) = pegelgesteuerte FFs
- ESSD (edge sensitive scan design) = flankengesteuerte FFs

Selbsttest (BIST = Build - in self test)

= einige Funktionen des Testautomaten auf IC ausgelagert z.B.

- Testmustergenerierung (TMG)
  - aus linear rückgekoppelten Schieberegistern
  - erzeugt Testmuster und legt dieses an Schaltung an
- Testantwortungauswertung (TAA)
  - aus parallelen Signatur-/Schieberegistern (MISR = multiple input shift register)
  - kompaktiert Testmusterantworten und vergleicht Ist ↔ Soll
- Teststeuerung (TS)
  - koordiniert Testablauf vom Start bis Auswertung
- Testkonfiguration (TK)
  - trennt Testmode vom Rest der Schaltung und stellt Datenpfade für Testdatentransfer zur Verfügung (Signatur)

kompletter ↔ teilweiser Selbsttest

Vorteile: Vereinfachung der Testvorbereitung, Verbesserung der Testdurchführung, Verbesserung der Betriebssicherheit

Nachteile: zusätzlicher Schaltungsaufwand, nur für Off-Line Tests geeignet

BILBO (Built in Logic Block Observer)

= Schieberegister mit 2 Testmode Signalen und 4 Funktionen (Register, Schieberegister, LSFR/TMG, MISR/TAA)

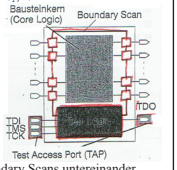
Boundary Scan (= standardisierte Testhilfe für Baugruppentest)

Vorteile: -genormte Testhilfe auf die jeder Schaltungsentwickler aufsetzen kann

- Bindeglied zwischen Baustein- und Baugruppentest
- Beitrag zu durchgängigen Selbsttest-Konzept
- Ersatz für In-Circuit-Test mit Nadelbrettadapter,
- getrennter Test von Bausteinen und Verbindungen zwischen den Bausteinen möglich
- Einbindungen anderer Testhilfen möglich

Prinzip:

besteht aus ladbarem Schieberegister das ringförmig um die Bausteine zwischen den Pads und der internen Schaltung (Core Logic) angeordenet ist und die Eingänge und Ausgänge dieser Bausteine einbindet.




Test Access Point (TAP):

- Einstellen der Betriebsarten
- Verbindung und Steuerung von Boundary Scans untereinander
- Schieben von Testmustern und Ausschieben der Testantworten
- Besteht aus 4 zusätzlichen Pins / Anschlüssen zur Integration des BS
  - TDI (Test Data Input) = Serieller Einschiebeeingang
  - TDO (Test Data Output) = Serieller Ausgabeausgang
  - TMS (Test Mode Select) = Steuerung des Testablauf
  - TCK (Test Clock) = Testtakt (unabhängig vom Systemtakt)

Testmodi:

- Externer Testmode (EXTEST)
  Zum Test der Verbindung zwischen den Bausteinen (Auffinden von Kurzschlüssen und Unterbrechungen)
- Sample Testmode (SAMPLE / PRELOAD)
  Test der Betriebsdaten während dem Normalbetrieb (SAMPLE)
  Vorbelegung von Anfangswerten der BS-Zellen (PRELOAD)
- Interner Testmode (INTEST)
  Zum Test der internen Bausteinlogik

Test von ICs

Test im Labor:

- Spitzenmeßplatz (Waferprober)
  Ermittelung der elektr. Eigenschaften, Verzögerungszeiten, Treiberfähigkeiten, Temperaturverhalten („hot-chuck“)
- Logikanalysator
  Überprüfen der Logikfunktion, Verzögerungszeiten

Test in der Produktion:

- Scheibentest (Erkennung defekter Chips vor Montage)
  Gleichstromaufnahme, wichtige Prüfmuster, mit Spitzenmeßplatz, Treiberfähigkeit, Markierung defekter Chips
- Bausteintest (Überprüfung Bausteinspezifikation)
  im Testautomaten mit Prüfbitmustern

Test mit Elektronenstrahlmessgerät:

= Messung von zeitlichen Potentialverläufen an Punkten innerhalb des Chips (Potentialkontrastverfahren), belastungs-&zerstörungsfrei, nur Messung (kein Einprägen von Spannung möglich)

Aufbau (Röhre): Elektronenkanone, Austast-Kondensator, Primärelektronenstrahl, Ablenkspulen, Signalsauswertung

IC-Tester:

Steuerrechner ↔ Teststeuerung ↔ Pinelektronik ↔ Testobjekt

Pin Versorung und Meßsystem

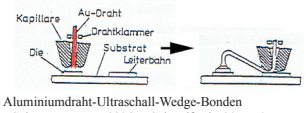
Timing Generator

# 11. Systemintegration

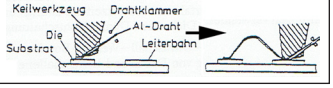
Chip Montagetechnik

klassische Prozessfolge:

- Aufspannen des Wafers
- Chip-Trennung (Sägen)
- Chipmontage (Diebonden)
- Kontaktierung (Drahtbonden)
  - Golddraht-Thermosonic-Ball-Bonden
    Substrattemp: <150°C  Schweißzeit: 40 ... 60ms



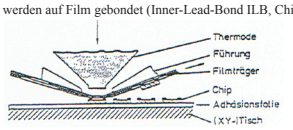

  - Aluminiumdraht-Ultraschall-Wedge-Bonden
    Substrattemp: ca. 23°C  Schweißzeit: 30 ... 50ms



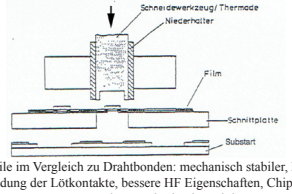

Film-Bonden (Tape-Automated-Bonden TAB)

Alternative zu Drahtbonden,

Chips werden auf Film gebondet (Inner-Lead-Bond ILB, Chip on Tape)




Chips werden vom Film heraus auf die Gehäuse gebondet (Out-Lead-Bond, OLB)




Vorteile im Vergleich zu Drahtbonden: mechanisch stabiler, keine Ermüdung der Lötkontakte, bessere HF Eigenschaften, Chips können vor Montage getestet werden, Rückseitenkontaktierung möglich

Chip-Gehäuse

Anforderungen: gute elektr. Verbindung, gute Wärmeabfuhr, gute mechan. Befestigung, guter Schutz gegen Umwelteinflüsse, gute Handhabbarkeit, gute Testbarkeit, hohe Zuverlässigkeit, geringes Gewicht, geringe Baugröße, geringe Kosten

Gehäusearten & Bezeichnungen:

| Kurzzeichen | Bedeutung | Layout |
|---|---|---|
| SIL | Single In Line | |
| DIL<br>PDIL<br>CDIL<br>DIP | Dual In Line<br>Plastik DIL<br>Ceramic DIL<br>Dual In Line Package | |
| CC<br>LCC<br>PLCC<br>LCCC | Chip Carrier<br>Leaded CC<br>Plastik LCC<br>Leaded Ceramic CC | |
| SO<br>SOIC<br>SOJ | Small Outline<br>SO IC<br>SO J Leaded IC | |
| PGA<br>PPGA<br>CPGA | Pin Grid Array<br>Plastic PGA<br>Ceramic PGA | |
| TAB | Tape Automated Bonding | |
| QFP<br>PQFP<br>GQFP<br>TQFP | Quad Flat Pack<br>Plastic QFP<br>Guardring QFP<br>Thin QFP | |
| BGA | Ball Grid Arrays | |

Leiterplattentechnik:

Einteilung:

- Einseitige Leiterplatte
- Doppelseitige Leiterplatte (ohne Durchkontaktierung)
- Doppelseitige Leiterplatte (mit Durchkontaktierung)
- Multilayer (Verklebung mehrerer Einzellagen und Vermaschung über Durchlkontaktierung)
- Flexible Schaltungen (Metall auf dünnen biegsamen Folien)

Basismaterialien:

- starre Substrate
  - Harz (Phenol, Epoxyd, Polyester, Polyimid)
  - Trägermaterial (Papier, Glasgewebe, Glasmatte)
- flexible Substrate
  - Rohstoffbasis (Polyester, Polymid, Teflon)

Herstellungsprozess:

1.) Bildübertragung

- Fotodruck (Belichtung von fotopolymeren Trockenfilmen, bis 0.1 mm)
- Siebdruck (bis 0.3 ... 0.4 mm)

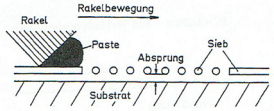
2.) Ätzen (z.b. Eisenchlorid, Kupferchlorid)

3.) Entschichten

- Bei Fotodruck durch chem. und mech. Reinigen
- Bei Siebdruck durch Lösungsmittel

Montage von Bauelementen:

- Einsteckmontage
- Hybridtechnik
  - Hybridschaltung = integrierte Schichtschaltung
  - Dickschichttechnik = ICs mit Schichten im Siebdruck

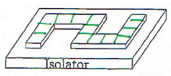



  - Dünnschichttechnik = ICs mit Vakuumbeschichtungsverfahren
- Oberflächenmontage (Surface Mounted Technology SMT)
  - Festkleben des Bauteils und kontaktierung durch Tauchlöten
  - Vorteile: Flächenreduktion, niedrige Bestückungskosten, hohe Zuverlässigkeit
  - Lötprozesse: Reflowlöten oder Schwallöten
  - Für Bauteile: C, R, IC
  - Gehäuse: SO, PLCC, TAB, QFP

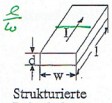
Elektrische Systemkomponenten

Widerstand:

Ausführung:




Berechnung:




$n = \frac{l}{w}$

$R = \frac{\rho l}{dw} = R_{\square}\frac{l}{w} = R_{\square} n$ (Anzahl Quadrate)

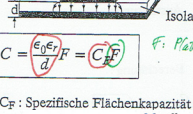
$R_{\square}$ : Schichtwiderstand [Ω/ ] □

Strukturierte Leiterbahnen:

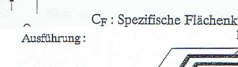
$R = R_{\square} n$

Kondensator:

Ausführung:




Berechnung:

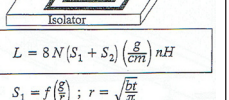



$C = \frac{\varepsilon_0 \varepsilon_r}{d} F = C_F F$

$C_F$ : Spezifische Flächenkapazität

Spule

Ausführung:




Berechnung:


$L = 8N(S_1 + S_2)\left(\frac{g}{cm}\right) nH$

Leiterquerschnitt

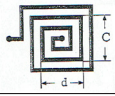
$S_1 = f\left(\frac{g}{r}\right)$ ; $r = \sqrt{\frac{bt}{\pi}}$

$S_2 = f\left(\frac{g}{d}; N\right)$ ; $a = \frac{c+d}{2}$


c mittlere Breite der Windungen

d mittlere Länge der Windungen

g Ganghöhe der Windungen




$S_1, S_2$ aus einschlägigen Diagrammen entnehmen