# Digitaltechnik

**Einheitspräfixe**

| Deka | Hekto | Kilo | Mega | Giga | Tera | Peta | Exa | Zetta | Yotta |
|---|---|---|---|---|---|---|---|---|---|
| da | h | k | M | G | T | P | E | Z | Y |
| 1 | 2 | 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 |
| d | c | m | $\mu$ | n | p | f | a | z | y |
| Dezi | Zenti | Milli | Mikro | Nano | Piko | Femto | Atto | Zepto | Yokto |

$Bit \xrightarrow{\cdot 8} Byte \xrightarrow{\cdot 1024} kByte \xrightarrow{\cdot 1024} MByte$

**Moores Law:** alle 18-24 verdoppelt sich die Anzahl der Chips pro Fläche. Probleme: hohe Verlusleistungsdichte, Komplex bei der Herstellung (von kleinen Strukturen), höhere Störanfälligkeit da weniger e- an Operation beteiligt sind (heisenbergsche Unschärfe etc)

**RISC-Prozessor**: Speicher, Ein- und Ausgangsystem, Rechenwerk, Bussystem

## Zahlensysteme

**Größte darstellbare Zahl nach Anzahl der Ziffern**

b: Basis; p: Anz. Vorkommast.; n: Anz. Nachkommast.

$$Z_{max} = b^p - b^{-n}$$

### 3.1 Umrechnung

| | $Z \geq 1$ | $Z < 1$ |
|---|---|---|
| $r \to 10$ | $Z_{10} = \sum r^i \cdot d_i$<br>$101_2 \to 1 \cdot 1 + 0 \cdot 2 + 1 \cdot 4$ | $Z_{10} = \sum r^{-i} \cdot d_{-i}$<br>$0.11_2 \to 1 \cdot 0.5 + 1 \cdot 0.25$ |
| $10 \to r$ | $d_i = Z_{10} \% r^i$<br>$58/8 = 7\text{Rest } 2 (LSB)$<br>$7/8 = 0\text{ Rest } 7 (MSB)$ | $0.4 \cdot 2 = 0.8$ Übertrag $0 (MSB)$<br>$0.8 \cdot 2 = 1.6$ Übertrag 1 |

Anzahl der benötigten Ziffern n bezüglich der Basis r, um $Z_{10}$ darstellen zu können:

$$n = \lfloor log_r(Z) \rfloor + 1$$

### 3.2 Zweierkomplement

**Wertebereich:** $-2^{n-1} \leq Z \leq 2^{n-1} - 1$

Wandle 2 in -2 um:

1. Invertieren aller Bits
2. Addition von 1
3. Ignoriere Überträge beim MSB

$0010 \Rightarrow 1101$
$1101 + 1 = 1110$
$\Rightarrow -2_{10} = 1110_2$

Radix-Komplement
Allgemein:
K(Z)=$r^n$-Z=($r^n$-1)-Z+1; $r^n$ : Basis des Zahlensystems

**binäre Rechenoperaionen (max. Stellen)**

Addition: $n_E = max(n_1, n2)+1$; Multiplikation: $n_E = n_1+n_2$

**IEEE 754 (Gleitkommadarstellung)**

| $s(1)$ | $e(8/11)$ | $f(23/52)$ |
|---|---|---|

Sonderdarstellung: e=0 ist 0 und e=255 ist ∞

Dezimal-->IEEE 754

$$e_{10} = \lfloor \log_2 |Z_{10}| \rfloor + 127 \qquad m_{10} = \lfloor (\frac{|Z_{10}|}{2^{e_{10}-127}} - 1) \cdot 2^{23} \rfloor$$

in die Form $1,m_2 * 2^x$ bringen und dann entsprechend anpassen: e=x+127, martisse mit 0er am Ende auffüllen

IEEE 754-->Dezimal

$$Z_{10} = (-1)^v \cdot (\frac{m_{10}}{2^{23}} + 1) \cdot 2^{e_{10}-127}$$

manuelles umwandeln geht schneller

$$Z_2 = (-1)^v \cdot (1, m_2) \cdot 2^{e_{10}-127}$$

Exponent umwandel, komma um die entsprechenden Stellen verschieben und dann in Dezimalumwandeln

| Hex. | Dualsystem | Dez. |
|---|---|---|
| 0 | 0 0 0 0 | 00 |
| 1 | 0 0 0 1 | 01 |
| 2 | 0 0 1 0 | 02 |
| 3 | 0 0 1 1 | 03 |
| 4 | 0 1 0 0 | 04 |
| 5 | 0 1 0 1 | 05 |
| 6 | 0 1 1 0 | 06 |
| 7 | 0 1 1 1 | 07 |
| 8 | 1 0 0 0 | 08 |
| 9 | 1 0 0 1 | 09 |
| A | 1 0 1 0 | 10 |
| B | 1 0 1 1 | 11 |
| C | 1 1 0 0 | 12 |
| D | 1 1 0 1 | 13 |
| E | 1 1 1 0 | 14 |
| F | 1 1 1 1 | 15 |

**boolsche Algebra**

| Boolesche Algebra $(0, 1; \cdot, +, \bar{x})$ | Kommutativ | Assoziativ | Distributiv | Indempotenz | Absorbtion | Neutral | Dominant | Komplement | De Morgan |
|---|---|---|---|---|---|---|---|---|---|
| | $x \cdot y = y \cdot x$ | $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ | $x \cdot (y + z) = x \cdot y + x \cdot z$ | $x \cdot x = x$ | $x \cdot (x + y) = x$ | $x \cdot 1 = x$ | $x \cdot 0 = 0$ | $x \cdot \bar{x} = 0$ | $\overline{x \cdot y} = \bar{x} + \bar{y}$ |
| | $x + y = y + x$ | $x + (y + z) = (x + y) + z$ | $x + (y \cdot z) = (x + y) \cdot (x + z)$ | $x + x = x$ | $x + (x \cdot y) = x$ | $x + 0 = x$ | $x + 1 = 1$ | $x + \bar{x} = 1$ | $\overline{x + y} = \bar{x} \cdot \bar{y}$ |
| | | | | | | | | $\bar{\bar{x}} = x$ | |

**Resolutionsgesetz:** $x \cdot a + \bar{x} \cdot b = x \cdot a + \bar{x} \cdot b + a \cdot b$

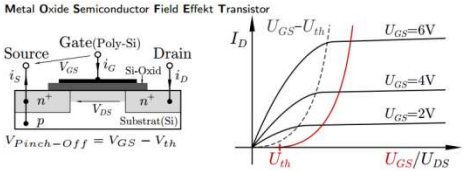
$$\boldsymbol{a\ XOR\ b = a\bar{b} + b\bar{a}}$$

**Gatter:**

| | Schaltungssymbol amerikanisch | Schaltungssymbol deutsch | Schaltungssymbol DIN-Norm 40900 | Verknüpfung, Abbildungsvorschrift | Funktion, Eins-Menge |
|---|---|---|---|---|---|
| UND AND | | | & | $y = f(x_1, x_2) = x_1 \cdot x_2$ | $f = \{11\}$ |
| ODER OR | | | ≥1 | $y = f(x_1, x_2) = x_1 + x_2$ | $f = \{01, 10, 11\}$ |
| NICHT NOT | | | 1 | $y = f(x) = \bar{x}$ | $f = \{0\}$ |
| NAND | | | & | $y = f(x_1, x_2) = \overline{x_1 \cdot x_2}$ | $f = \{00, 01, 10\}$ |
| NOR | | | ≥1 | $y = f(x_1, x_2) = \overline{x_1 + x_2}$ | $f = \{00\}$ |
| XOR (exkl. ODER) | | | =1 | $y = f(x_1, x_2) = x_1 \oplus x_2$ | $f = \{01, 10\}$ |
| XNOR (Äquivalenz) | | | = | $y = f(x_1, x_2) = \overline{x_1 \oplus x_2}$ | $f = \{00, 11\}$ |
| Subjunktion (Implikation) | | | ≥1 | $y = f(x_1, x_2) = x_1 \to x_2 = \bar{x}_1 + x_2$ | $f = \{00, 01, 11\}$ |
| MUX | | $x$ : Selektoreingang<br>$a, b$ : Dateneingänge | | $y = \beta(x, a, b) = x \cdot a + \bar{x} \cdot b$ | $\beta = \{001, 011, 110, 111\} = \{0*1, 11*\}$ |

**MOSFET-Transitor**

Metal Oxide Semiconductor Field Effekt Transistor




### 5.1 Bauteilparameter

Verstärkung: $\beta = K' \frac{W}{L}$ mit $K' = \frac{\mu \epsilon_{0x} \epsilon_0}{t_{0x}}$

| | |
|---|---|
| Kanalweite | W |
| Kanallänge | L |
| Elektronenbeweglichkeit | $\mu \quad \mu_n \approx 250 \cdot 10^{-4} \frac{m^2}{Vs}$, $\mu_p \approx 100 \cdot 10^{-4} \frac{m^2}{Vs}$ |
| rel. Dielektrizität des Gateoxyds | $\epsilon_{ox} \approx 3,9$ |
| Dielektrizitätskonstante | $\epsilon_0 = 8.8541878 \cdot 10^{-12} \frac{As}{Vm}$ |
| Gateoxyddicke | $t_{ox}$ |
| Verstärkung | $\beta = \frac{\mu_n \varepsilon_{ox} \varepsilon_0}{t_{ox}} \cdot \frac{W}{L} = K' \frac{W}{L} = \frac{\mu_n C_G}{L^2}$ |
| Kapazität | $C_G = \varepsilon_{ox} \varepsilon_0 \frac{WL}{t_{ox}}$ |
| Verzögerungszeit | $t_{pHL} \propto \frac{C_L t_{ox} L_p}{W_p \mu_p \varepsilon_{ox} (V_{DD} - \lvert V_{th} \rvert)}$ |

- große Kanalweite ⇒ große Drain-Störme
  ⇒ schnelle Schaltgeschwindigkeit (da $i_d \propto \beta \propto \frac{W}{L}$)
  Aber: große Fläche.
- nMos schaltet schneller als pMOS

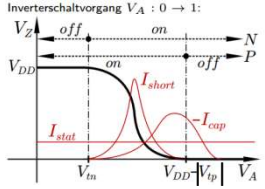
Kanalweite W zur Kompensation der niedrigeren Löcherbeweglichkeit der p-mos: $K'_p * (W_p/L_p) = K'_n * (W_n/L_n)$

**$t_{pHL}$: steigend mit:** Schalungskomplexität $C_L$ , Oxiddicke, Kanallänge, Schwellspannung $U_{th}$; **fallend mit:** Kanalweite W, Ladungsträgerbeweglichkeit, Oxiddielektrizität, Versorgungsspannung

**CMOS-Verlustleistung:**

| Dynamische (50%) | Statische |
|---|---|
| -Kapazitiver Anteil (~80%) -Kurzschluss Anteil ->mit Schaltfunktion verknüft ->abh. von Signalflanken | -Leckstrom (Diodensperrstrom) -Gate Strom -->parasitäre Effekte -->abhängig von $V_{dd}$ und $U_{th}$ -wird größer, wegen immer kleineren Strukturen (Isolierschichten) |

**CMOS-Inverter**




**Dynamische Verlustleistung**

Kapazitive Verluste

Kurzschlussstrom

Schalthäufigkeit

$P_{dyn} = P_{cap} + P_{short}$

$P_{cap} = \alpha_{01} f C_L V_{DD}^2$

$P_{short} = \alpha_{01} f \beta_n \tau (V_{DD} - 2V_{tn})^3$

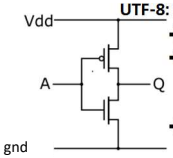
$\tau$: Anstiegs-/Abfallszeit von A

$\alpha_{0 \to 1} = \frac{\text{Schaltvorgänge(pos. Flanke)}}{\#\text{ Betrachtete Takte}}$

Abhängig von den Signalflanken, mit Schaltfunktionen verknüpft

$\approx V_{DD} 1/ \propto$ Schaltzeit: $\frac{V_{DD2}}{V_{DD1}} = \frac{t_{D1}}{t_{D2}}$ (bei Schaltnetzen $t_{log}$)

Verzögerungszeit $\propto \frac{1}{V_{DD} - V_{th}}$

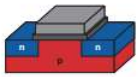



**UTF-8:** Codewörter mit Länge 8- ,16- , 24- , 32-Bit (1-4 Byte)

- ➔ MSB = 0 -> 1 Byte (restliche 7 Bits: ASCII)
- ➔ MSB = 1 -> 2-4 Byte -> ersten 3/4/5 Bit geben Länge an(110, 1110, 11110). Bytes 2-4 beginnen mit 10 um nicht als neues 1Byte Zeichen erkannt zu werden
- ➔ bsp: "110xxxxx 10xxxxxx"

## nMOS

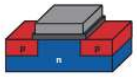
Guter Pull-Down
Source am niedrigeren Potential ($U_{DS} > 0$)



$$I_D = \begin{cases} 0 & U_{GS} < U_t (aus) \\ & \wedge\, U_{DS} \geq 0 \\ \beta\left(U_{GS} - U_t - \frac{U_{DS}}{2}\right) U_{DS} & U_{GS} > U_t \text{ (linear)} \\ & \wedge\, 0 < U_{DS} < U_{GS} - U_t \\ \frac{\beta}{2}\left(U_{GS} - U_t\right)^2 & U_{GS} > U_t \text{ (Sättigung)} \\ & \wedge\, 0 < U_{GS} - U_t < U_{DS} \end{cases}$$

## pMOS

Guter Pull-Up
Source am höheren Potential ($U_{DS} < 0$)



$$I_D = \begin{cases} 0 & U_{GS} > U_t (aus) \\ & \wedge\, U_{DS} \leq 0 \\ -\beta\left(U_{GS} - U_t - \frac{U_{DS}}{2}\right) U_{DS} & U_{GS} < U_t \text{ (linear)} \\ & \wedge\, 0 > U_{DS} > U_{GS} - U_t \\ \frac{-\beta}{2}\left(U_{GS} - U_t\right)^2 & U_{GS} < U_t \text{ (Sättigung)} \\ & \wedge\, 0 > U_{GS} - U_t > U_{DS} \end{cases}$$

**CMOS-Gatterentwurf**

**CMOS-Logik ist invertierend!! INVERTER NICHT VERGESSEN!!!**

**Vorteil:** (Fast) nur bei Schaltvorgängen Verlustleistung - wenig statische Verluste

| Netzwerk | Pull-Down | Pull-Up |
|---|---|---|
| Transistoren | nMos | pMos |
| AND | Serienschaltung | Parallelschaltung |
| OR | Parallelschaltung | Serienschaltung |

1. Möglichkeit: Direkt; ggf. Inverter vor die Eingänge und Ausgänge schalten.
2. Möglichkeit: Mit bullshit Algebra die Funktion nur mit NAND und NOR darstellen.

**Logikminimierung:**

Effizienz des Terms: L(z)= Summe der Literale in Teilterme+ Anzahl der Teilterme

**Karnaugh Tabelle:**

Zyklische Gray-Codierung: 2dim:00, 01, 11, 10 3dim:000, 001, 011, 010, 110, 111, 101, 100

| $z \backslash xy$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |
| 1 | X | 1 | 1 | 0 |

Gleiche Zellen zusammenfassen: z.B. $\overline{x}\overline{y} + y \cdot z$

Don't Care Werte ausnutzen!

Gray Kodierung: benachbarte Matrixen Felder unterscheiden sich in nur einer Binärstelle

möglichst viele Felder zusammenfassen, auch mehrere mehrmals benutzen, wenn dann besser zusammengefasst werden kann

Don't cares ausnützen!!!

**Quines Methode:**

Gegeben: DNF, gesucht: MinSUP

1. DNF unf KDNF erweitern: $abc + \overline{a}\overline{b} = abc + \overline{a}\overline{b}(c + \overline{c})$

2. Bestimmung der Primimplikanten durch spezielles Resolulutionsgesetz ($a\overline{x} + xa = a$) und Absobtionsgesetz (a+ab=a)

Nachteil: man braucht VollSOP! 1. schritt liefert teils sehr viele Mintterme (Worst Case $2^n$)

**Beispiel:**

| | 0-Kubus | A | 1-Kubus | R | A | 2-Kubus | A |
|---|---|---|---|---|---|---|---|
| $m_0$ | | | | | | | |
| $m_1$ | $\overline{x}_1\overline{x}_2x_3$ | ✓ | $\overline{x}_2x_3$ | $m_1\&m_5$ | $p_1$ | | |
| $m_4$ | $x_1\overline{x}_2\overline{x}_3$ | ✓ | $x_1\overline{x}_2$ | $m_4\&m_5$ | ✓ | $x_1$ | $p_2$ |
| $m_5$ | $x_1\overline{x}_2x_3$ | ✓ | $x_1\overline{x}_3$ | $m_4\&m_6$ | ✓ | | |
| $m_6$ | $x_1x_2\overline{x}_3$ | ✓ | $x_1x_3$ | $m_5\&m_7$ | ✓ | | |
| $m_7$ | $x_1x_2x_3$ | ✓ | $x_1x_2$ | $m_6\&m_7$ | ✓ | | |

**Resolventenmathode:**

Gesetze: allgemeines Resolutionsgesetz($ax + b\overline{x} = ax + b\overline{x} + ab$) und Absobtionsgesetz:

| $f$ | Schicht |
|---|---|
| ~~$\overline{x} \cdot y + x \cdot y \cdot z + \overline{x} \cdot \overline{y} \cdot \overline{z}$~~ | 0 |
| $+y \cdot z + \overline{x} \cdot \overline{z}$ | 1 |

**Überdeckungstabelle (quine-McCluskey):**

Problem: welche Primimplikanten werden minimal benötigt um f(x) vollständig darzustellen

Methode:

1. Markieren der Überdeckungen
2. Auswertung der Dominazrelationen

Zeilen und Spalten, die dominiert werden streichen

| $p \backslash m$ | $m_0$ | $m_2$ | $m_3$ | $m_7$ | Kosten |
|---|---|---|---|---|---|
| $p_1$ | 1 | 1 | 0 | 0 | $L(p_1)$ |
| $p_2$ | 0 | 1 | 1 | 0 | $L(p_2)$ |
| $p_3$ | 0 | 0 | 1 | 1 | $L(p_3)$ |

L(p1):= Länge des Primimplikanten (Anzahl der Literale)

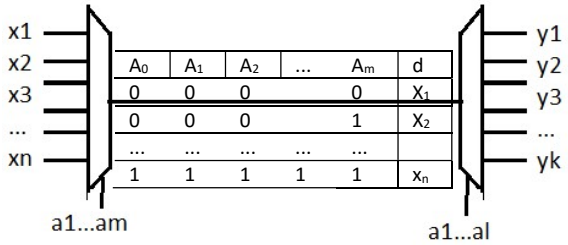
$\Rightarrow$ MinSOP : $f = p_1 + p_3 = \overline{x} \cdot \overline{z} + y \cdot z$

**Kombinatorisches Schaltnetz: (ohne Gedächtnis)**

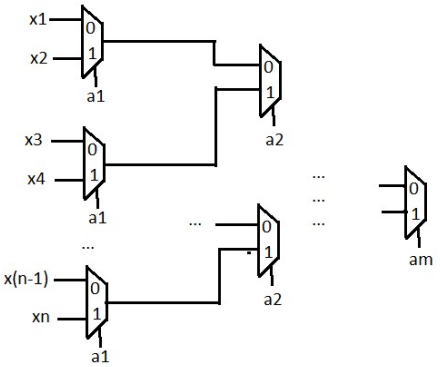
**MUX und DeMUX**

- Serielle Übertragung besser als parallele Übertragung, da zu kostenintensiv

-Mehrfachverwenung von Gleisen




DNF: $x_1 * \overline{a0} * \ldots * \overline{aM} + \ldots + x_n * a_0 * \ldots * a_m = 1$




Wenn Logik-Gleichung fertig optimiert wurde:

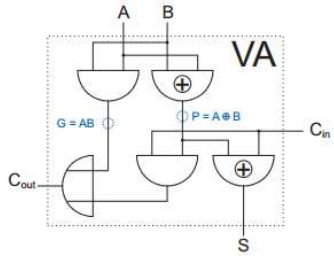
DNF:

$$z = a_2 * (a_1 * (a_0 * x_7 + \overline{a_0} * x_6) + \overline{a_1} * (a_0 * x_5 + \overline{a_0} * x_4)) + \overline{a_2} * (a_1 * (a_0 * x_3 + \overline{a_0} * x_2) + \overline{a_1} * (a_0 * x_1 + \overline{a_0} * x_0))$$

**Ripple-Carry-Adder: (28 Transistoren pro VA)**

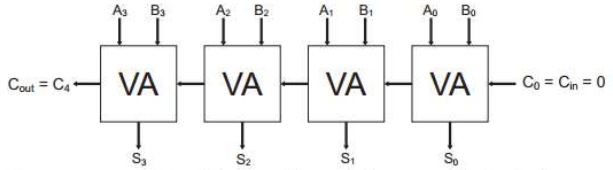
**Volladdierer**

| A | B | $C_{in}$ | $C_{out}$ | S |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |




$S = P \oplus C_{in}\ ,\ \ C_{out} = G + P \cdot C_{in}$

**Ripple-Carry-Adder**




Verzögerungszeit wird vom Carry-Übertrag dominiert! Maximale Verzögerungszeit, wenn beim LSB das Signal von G wechselt und bei allen anderen Gattern gilt: $P = 1$.

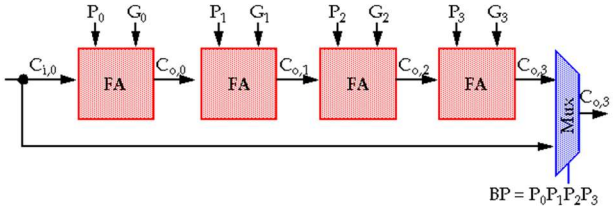
**Propergate**: P=A XOR B: falls 1, kann der Carry weitergeleitet weden zum Carry-Ausgang

**Generate:** G=AB=1: generiert eine 1 am Carryausgang

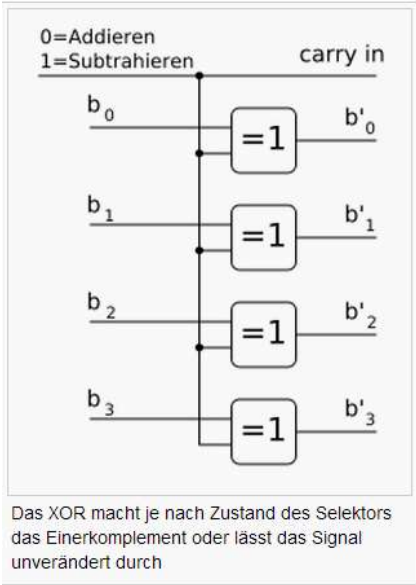
**Problem:** Es dauert seine Zeit bis der Carry vom ersten VA stabil ist und der zweite VA beginnen kann zu rechnen ($t_{max}$ wenn der Carry durch alle VAs geht)

**Lösung: Carry-Bypass-Adder**

Einfügen von DeMUX, die eine Verundung der Propergates der einzelnen $A_i$ und $B_i$ sind--> wenn diese Propergatebedingung auf 1 ist, dann kann der Carry vom ersten VA zum letzten weitergereicht werden




**Realisierung von Subtrahierer:**

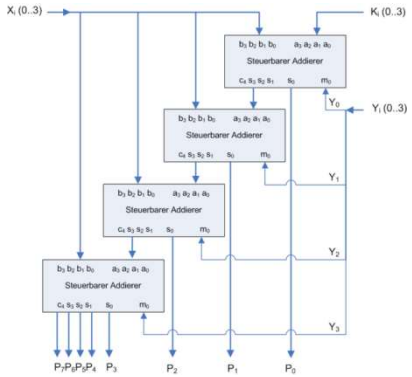



Das XOR macht je nach Zustand des Selektors das Einerkomplement oder lässt das Signal unverändert durch

**Muliplizierer:**

Multiplikation als Summe einer Summe

Array Multiplizierer: Parallele Generierung der patiellen Proukte und shift-Operation durch geschickte Verschaltung der Stufen, Nachteil: Kostenintensiv

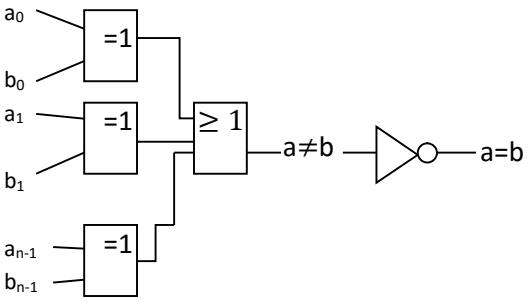



"Kasten" entspricht steuerbaren Addierer, der die Werte addiert, wenn y=1

Multiplikation mit $2^n$; kanonisch: einfache shift Operation

Logarithischer Barrel Shifter: k MUX Reihen/Zeilen (k: Wortbreite) mit $\lceil \log(n) \rceil$ Stufen/Spalten, wenn n die maximale Verschiebung ist. Nachteil: große Fläche und kostenintensiv

**Komparatoren/Vergleichoperationen:**




**Zeitanalyse kombinatorischer Schaltnetze:**

UND: 0 dominant: wenn ein Eingang 0, dann ist der Ausgang auch 0

OR: 1 dominant: wenn ein Eingang 1, dann auch der Ausgang ein

XOR: sensitiv auf beiden Eingängen, d.h. Ausgang hängt von beiden Eingängen ab

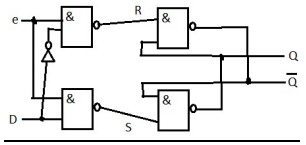
Allgemein: schauen wann der gewünschte Ausgang spätestens stabil ist.

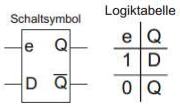
**Sequentielle Logik: (mit Gedächtnis)**

**Basic Speicherzelle:** Ring aus 2 Invertern um den Wert stabil zu halten. Problem: Änderung des Wertes welcher in dem Register gespeichert ist

Lösung: Set-Reset Latch/ Enable Lat

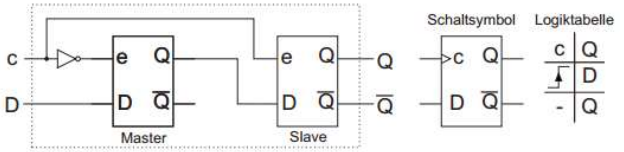
Level-Controled




übernimmt den Eingangswert auf D solange e=1

Taktflanken gesteuerter Regster: **Flip-Flop**




## Timing

$t_{setup}$: Zeit, in der der Eingangswert vor aktiver Taktflanke stabil sein muss

$t_{hold}$: Zeit, in der der Eingangswert nach aktiver Taktflanke stabil bleiben muss

$t_{c2q}$: Zeit, nach der der Eingangswert nach der Taktflanke stabil an Q anliegt (=Ausgangslatenz des Registers)

$t_{clk}$: Clock

$t_{längsterPfad}$: (=kritischer Pfad) Längste Verzögerungszeit zwischen zwei Registerstufen

$$t_{clk} \geq t_{c2q} + t_{längsterPfad} + t_{setup}$$

$$t_{hold} \leq t_{c2q} + t_{kürzesterPfad}$$

Kosten: zusätzliche Register, Dummy-Gatter, Latenz

Dummy-Gatter: 2 not Gatter, AND mit sich

$$f_{max} = \frac{1}{t_{setup}+t_{c2q}+t_{längsterPfad}}$$

$$Gesamtlatenz = (\text{Maximale Anzahl hintereinander geschalteter Register} - 1) \cdot t_{clk}$$

## Pipelining

Aufteilen langer kombinatorischer Pfade durch Einfügen zusätzlicher Registerstufen, um die Taktfrequenz erhöhen zu können (Gesamtlatenz wird allerdings nicht kleiner).

⇒ Möglichst Halbierung des längsten Pfades!

⇒ Evtl. müssen sog. „Dummy-Gatter“ eingefügt werden!

Gesamtlatenz wird bei Pipelining größer!! Durchsatz: $\frac{1\,sample}{tclk}$

$t_{clk,pipe}$= max ($t_{clk,Stufe}$), Latenz=#Stufen*$t_{clk,pipe}$

Möglichkeiten ohne Pipelining Registerzeiten nicht zu verletzen: tclk↓, schnellere Register&Gatter, Logikoptimierung

## Parallele Verarbeitungseinheiten

- Paralleles, gleichzeitiges Verwenden mehrere identischer Schaltnetze

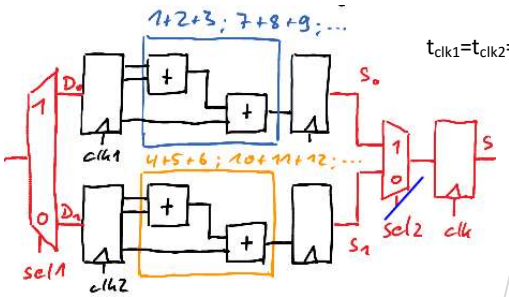
---

- Zusätzliche Kontrolllogik nötig (Multiplexer)
- Taktfrequenz und Latenz bleiben konstant
- Durchsatz steigt mit der Zahl der Verarbeitungseinheiten
- ABER: deutlich höherer Ressourcenverbrauch

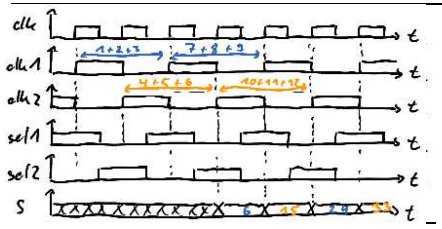
---

$T_{clk}$ von Register am Ende: $t_{clk,parallel}=\frac{tclk,Modul}{\#Module}$

Kosten= #Modul*(Kosten Logik)+Steuerlogik




$t_{clk1}=t_{clk2}=2t_{clk}$




**Tests:**

**Fehlermodell: Stuck-at-x**

Gatteranschluss ständig auf gnd/$V_{dd}$; Es ist nur ein Fehler s-a-x

2+r mögliche Fehler

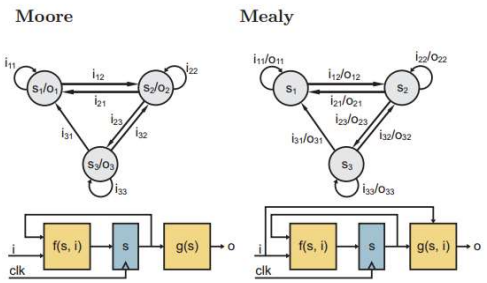
(r=#Gatteranschlüsse=#Eingänge+#Ausgäne+#Verbindungen im Gatter≈#Anzahl Gatter*#mittlerer Fan-out)

Fehlergruppe: Alle Fehler, die mit einen Test $t_v$ erkennbar sind bilden die Fehlergruppe des Test $t_v$

Testgruppe: Alle Tests die den Fehler $e_x$ und $e_y$ erkennen bilden Testgruppe

**Automaten:**

**Finite state machine**




Ausgang nur vom Zustand abhängig.

Ausgang vom Zustand und von Eingabe abhängig.

**Vorteile:**

| | |
|---|---|
| Kein Kombinatorischer Pfad von Eingängen zu Ausgängen<br>⇒ Begrenzung der Logik-Tiefe | Weniger Zustände, Übersichtlicher, Allgemeiner<br>Ausgang reagiert schneller |

**Nachteile:**

| | |
|---|---|
| Hohe Anzahl von Zuständen | Lange kombinatorische Pfade bei Vekettung mehrerer FSMs |

| ... | $S_1$ | $S_0$ | A | $S_1'$ | $S_0'$ |
|---|---|---|---|---|---|
| ... | 0 | 0 | 0 | 0 | 0 |
| ... | 0 | 0 | 1 | 0 | 0 |
| ... | 1 | 0 | 0 | 0 | 0 |
| ... | 1 | 0 | 1 | 0 | 0 |

Logikgleichung erstellen: Zustände kodieren-->für jeden Zustand aufstellen in welchen Zustand es unter dem einem Eingangswert A wechselt

**Test-Fehler-Relationstabelle:**

jede Eingangsbelegung testen welches y idealerweise rauskommt, dann für jeden Ausgang testen was rauskommt wenn er **s**tuck-**a**t-0/1 ist.

Gleiche Spalten: Fehler sind nicht zu unterscheiden, gleiche Zeilen: Tests erkennen gleichen Fehler

Fehlerüberdeckungstabelle:

-gleiche Spalten gruppieren

-Fehler suchen, die nur von einem Test erkannt werden--> Aufnahme des Tests in die Testauswahl, dann die Tabelle vervollständigen

**Einzeltestgenierieung (D-Algoithmus):**

Fehler annehmen und dann die Eingangsvariablen so geschickt wählen (durch Vorwärts- und Rückwärtsimplikation), dass sich der Fehler bis zum Ende durchpropagiert

Sensitivität eines Pfades: Ein Pfad ist bei $t_v$ sensitiv auf einen Fehler ey, wenn alle Leitungen sensitiv sind

Fan-out-freie Schaltungen: Eingangsvariablen die zu einem gemeinsamen Signalwert beitragen sind verschieden; Jedes Einstellungsproblem kann unabhängig voneinander gelöst werden

**Allgemein:**

**Persistente Fehler:** Designfehler; Fehler im Fertigungsprozess

**Dynamische Fehler**: Alterung

**DNF:** z=$a\overline{b}c + abc + \overline{a}b\overline{c}$=1; **KNF:** z=(a+b+c)*(...)=0

Umwandlung: durch Wahrheitstabelle oder alles doppelt negieren und deMorgan anwenden

**Mintterm:** alle n Variablen kommen in einer UND-Verknüpfung vor

**Maxterm:** alle n Variablen in einer ODER-Verknüpfung

**gebundene Variable:** $x_1+x_2x_3$: $x_1$ ist freie Variable, $x_2$und $x_3$sind gebunden

**Implikant:** Produkt bzw. UND-Term: überdeckt mindestens einen Mintterm

**Primimplikant**: Implikant, der nicht mehr weiter vereinfacht werden kann

**Fan-out:** Anzahl der nachfolgenden Gatter. Ist der Fan-out größer, so wird die Verzögerungszeit größer

**Fan-in:** Anzahl der Eingänge eines Logikgatters

**Parity-Prüfsumme:** Ergänzung der Bytes, so, dass geradzahlig viele 1en drin vorkommen

| | | |
|---|---|---|
| SOP (DNF) | eine Summe von Produkttermen | Terme sind ODER-verknüpft |
| POS (KNF) | ein Produkt von Summentermen | Terme sind UND-verknüpft |
| CSOP (nur 1) | Menge aller Minterme | analog CPOS |
| VollSOP (nur 1) | Menge aller Primimplikanten | Bestimmung siehe Quine Methode oder Schichtenalgorithmus |
| MinSOP (min. 1) | Minimale Summe v. Primimplikanten | durch Überdeckungstabelle |

| x | y | AND $x \cdot y$ | OR $x+y$ | XOR $x \oplus y$ | NAND $\overline{x \cdot y}$ | NOR $\overline{x+y}$ | EQV $\overline{x \oplus y}$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

**Boolsche Funktionen:**

**Kofaktor**

- $f|_{x_i=1} = f_{x_i}$ , $f|_{x_i=0} = f_{\overline{x}_i}$
- $(f_{x_i})_{x_j} = (f_{x_j})_{x_i} = f_{x_i x_j}$
- $x_i \cdot f = x_i \cdot f_{x_j}$ , $\overline{x}_i \cdot f = \overline{x}_i \cdot f_{\overline{x}_j}$
- $x_i + f = x_i + f_{\overline{x}_j}$ , $\overline{x}_i + f = \overline{x}_i + f_{x_j}$

**Entwicklungssätze**

- $f = x_i \cdot f_{x_i} + \overline{x}_i \cdot f_{\overline{x}_i} = \beta(x_i, f_{x_i}, f_{\overline{x}_i})$
- $\overline{f} = x_i \cdot \overline{f_{x_i}} + \overline{x}_i \cdot \overline{f_{\overline{x}_i}}$

**Sonstiges**

- $f$ unabhängig von $x_i \Leftrightarrow f_{x_i} = f_{\overline{x}_i} \Leftrightarrow f_{x_i} \oplus f_{\overline{x}_i} = 0$
- $f$ abhängig von $x_i \Leftrightarrow f_{x_i} \neq f_{\overline{x}_i} \Leftrightarrow f_{x_i} \oplus f_{\overline{x}_i} = 1$
- $f$ positiv symmetrisch in $x_i$ und $x_j \Leftrightarrow f_{x_i\overline{x}_j} = f_{\overline{x}_i x_j}$
- $f$ negativ symmetrisch in $x_i$ und $x_j \Leftrightarrow f_{x_i x_j} = f_{\overline{x}_i \overline{x}_j}$