

Digitaltechnik (Technische Informatik)

Schichtenmodell

Programme

Anwendungssoftware

Betriebssysteme

Architektur

Mikroarchitektur

Logik

DigitalSchaltungen

Analogschaltungen

Bauteile

Physik

z.B.

Programme

Gerätetreiber

Befehle

Register

Datenpfade

Stackframe

Addierer

Speicher

Gatter

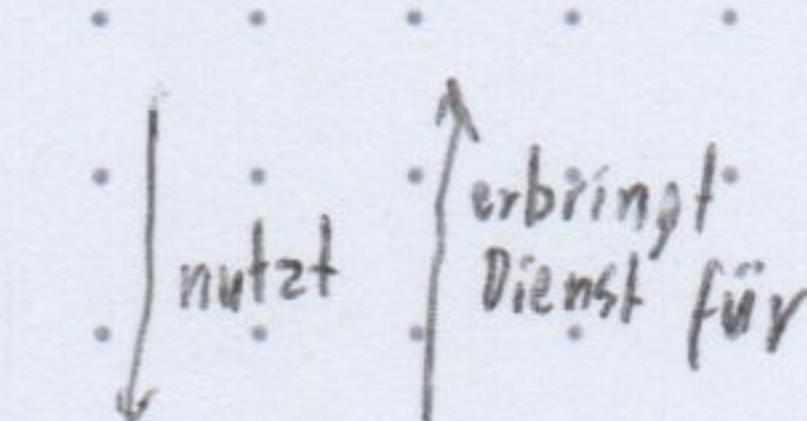
Vorstärker

Filte

Transistoren

Diode

Elektroden



- geringere Komplexität

- ggf. geringere Leistung

Hierarchie (Knotenteilen in Module)

Modularität (Schnittstellen)

Regularity (bevorzugt einheitliche Lösung)

Austauschbarkeit

$$2^0 = 1$$

$$2^1 = 2$$

$$2^2 = 4 = 1 \text{ Nibble}$$

$$2^3 = 8 = 1 \text{ Byte (B)}$$

$$2^4 = 16 = \text{Halbwort}$$

$$2^5 = 32 = \text{Wort}$$

$$2^6 = 64$$

$$2^7 = 128$$

$$2^8 = 256$$

$$2^9 = 512$$

$$2^{10} = 1024 = 1 \text{ KiB (KiBi)}$$

$$2^{11} = 2048$$

$$2^{12} = 4096$$

$$2^{13} = 8192$$

$$2^{14} = 16384$$

$$2^{15} = 32768$$

$$2^{16} = 65536$$

$$2^{17} = 131072$$

$$2^{18} = 262144$$

$$2^{19} = 524288$$

$$2^{20} = 1048576 = 1 \text{ Mi (MiBi)}$$

$$2^{30} = 1073741824 = 1 \text{ Gi (GiBi)}$$

$$2^{40} = 1099511627776 = 1 \text{ Ti (TiBi)}$$

Achtung

1 GB

$2^{30} \text{ B} > 10^9 \text{ B}$

von Plattform abhängig

(meist Registerbreite)

Digitale Abstraktion: Berücksichtigung nur endlich vieler Werte.

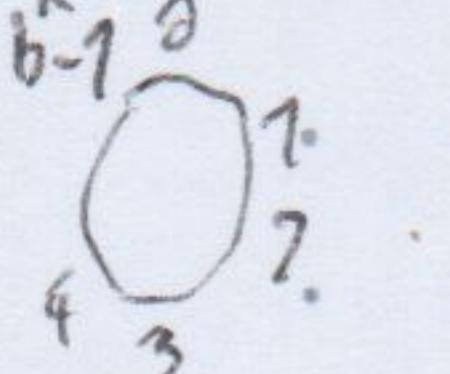
Digitale Disziplin: zwei Werte: 1 z.B. etw. repräsentiert versch. elektrisch, mechanisch, hydraulisch, logische Zustände

wissenschaftliche Beschränkung

Bit (Binary digit) kleinste mögliche Informationseinheit

Bitfolgen $2^{\text{Bitanzahl}} = \text{Zustände MSB}$ LSD Least significant Digit Zahlenfolgen MSD Most " "

IV. vorzeichenloses Stellsystem



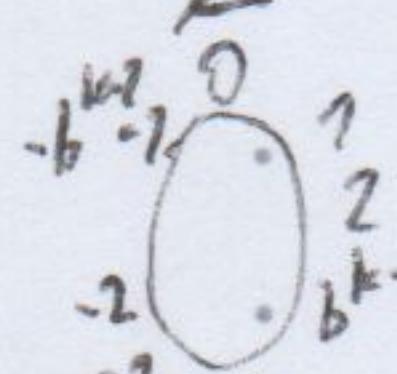
$$u_{b,k} : (a_{k-1} \dots a_0) \in Z_b^k \mapsto \sum_{i=0}^{k-1} a_i \cdot b^i \text{ (W. } 0 \text{ bis } b^k - 1\text{) (bijektiv)}$$

zero extension

$$2^{30} = 1073741824 = 1 \text{ Gi (GiBi)}$$

$$2^{40} = 1099511627776 = 1 \text{ Ti (TiBi)}$$

Z Vorzeichen und Betrag



$$v_{b,k} : (a_{k-1} \dots a_0) \in Z_b^k \mapsto (-1)^{a_{k-1}} \sum_{i=0}^{k-2} a_i \cdot b^i + 2^{-(b^{k-1}-1)} \cdot a_{k-1} \cdot (b^{k-1}-1) \text{ (bijektiv)}$$

0 doppelt

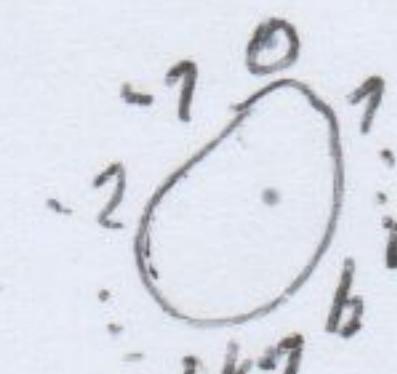
b^{k-1} Zahlen

$$2^{30} = 1073741824 = 1 \text{ Gi (GiBi)}$$

$$2^{40} = 1099511627776 = 1 \text{ Ti (TiBi)}$$

hündige Busen

Zweierkomplement (primär für Bitfolgen)



$$s_k : (a_{k-1} \dots a_0) \in B^k \mapsto a_{k-1} \cdot (-2^{k-1}) + \sum_{i=0}^{k-2} a_i \cdot 2^i \in Z \cdot (-b^{k-1}) \text{ bis } (b^{k-1}-1) \text{ oktal (bijektiv)}$$

sign extension

$$\text{dual/binär: } B = \{0,1\} \in Z_b^k \text{ } 0b \dots 00$$

$$1b \dots 10 \in Z_b^k \text{ } 1b \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

$$10 \dots 10 \in Z_b^k \text{ } 10 \dots 10$$

reelle Zahlen

1.01.

Festkomma: letzte l. Stellen Nachkommaz. z.B. Euro
 Addition: normal $n \cdot 2^{-l}$
 Mult: 2l Nachkommastellen (Random)

Gleitkomma:

relativ geringe Rundungsfehler

IEEE 754 1bit 8bits 23 bits
 VZ s e₇ e₆ e₅ e₄ e₃ e₂ e₁ e₀ f e₂₃ ... f₀ $\Rightarrow (-1)^s \cdot 1.f \cdot 2^{e-\text{bias}}$

Addieren Teile extrahieren (führen ergänzen)

kleineren Exponenten um Diff. nach links verschieben

Addieren/Subtrahieren

Resultierende Mantisse normalisieren, runden

Teile zusammenfassen

Sonderformate

x 0...0 0...0

x 0...0 x...x

x 1...1 0...0

x 1...1 1x...x

x 1...1 0x...0

± 0

denormalisierte Zahl $(0.x \dots x) \cdot 2^{1-\text{bias}}$

$\pm \infty$

silent NaN

signaling NaN

Achtung: oft Rundungsfehler $0.1 + 0.2 \neq 0.3$

dez>bin z.B. 13.53

Vk Nk

1 ← carry 55
 0 2⁵
 0 4
 0 8
 1 6
 2 2
 2

Hälfte
 Exponentenzustände
 -1 1

13.53₁₀ = 1101.10
 0011₂

Logikgatter

Buffer: $A \rightarrow D \rightarrow Y = A$

Not: $A \rightarrow D \rightarrow Y = \bar{A}$

And: $A \rightarrow D \rightarrow Y = AB$

OR: $A \rightarrow D \rightarrow Y = A+B$

XOR: $A \rightarrow D \rightarrow Y = A \oplus B$

XOR3: $A \rightarrow D \rightarrow Y = A \oplus B \oplus C$
(Paritätsfunktion)

Leistungsaufnahme $\frac{\text{Energie}}{\text{Zeit}} (\text{Watt})$

$$\begin{aligned} &\text{z.B. } V_{DD} = 1,2V \\ &f = 1 \text{ GHz} \\ &C_{\text{Gitterkapazität}} = 20 \text{ nF} \\ &t_{DD} = 20 \text{ ns} \\ &P = 24 \text{ mW} + 14,4 \text{ W} \end{aligned}$$

statisch: wenn kein Gatter schaltet
durch Leckstrom I_{DD}
kleiner Transistor schalten nicht mehr möglich
Pausenzeit, ...

$$P_{\text{static}} = I_{DD} \cdot V_{DD}$$

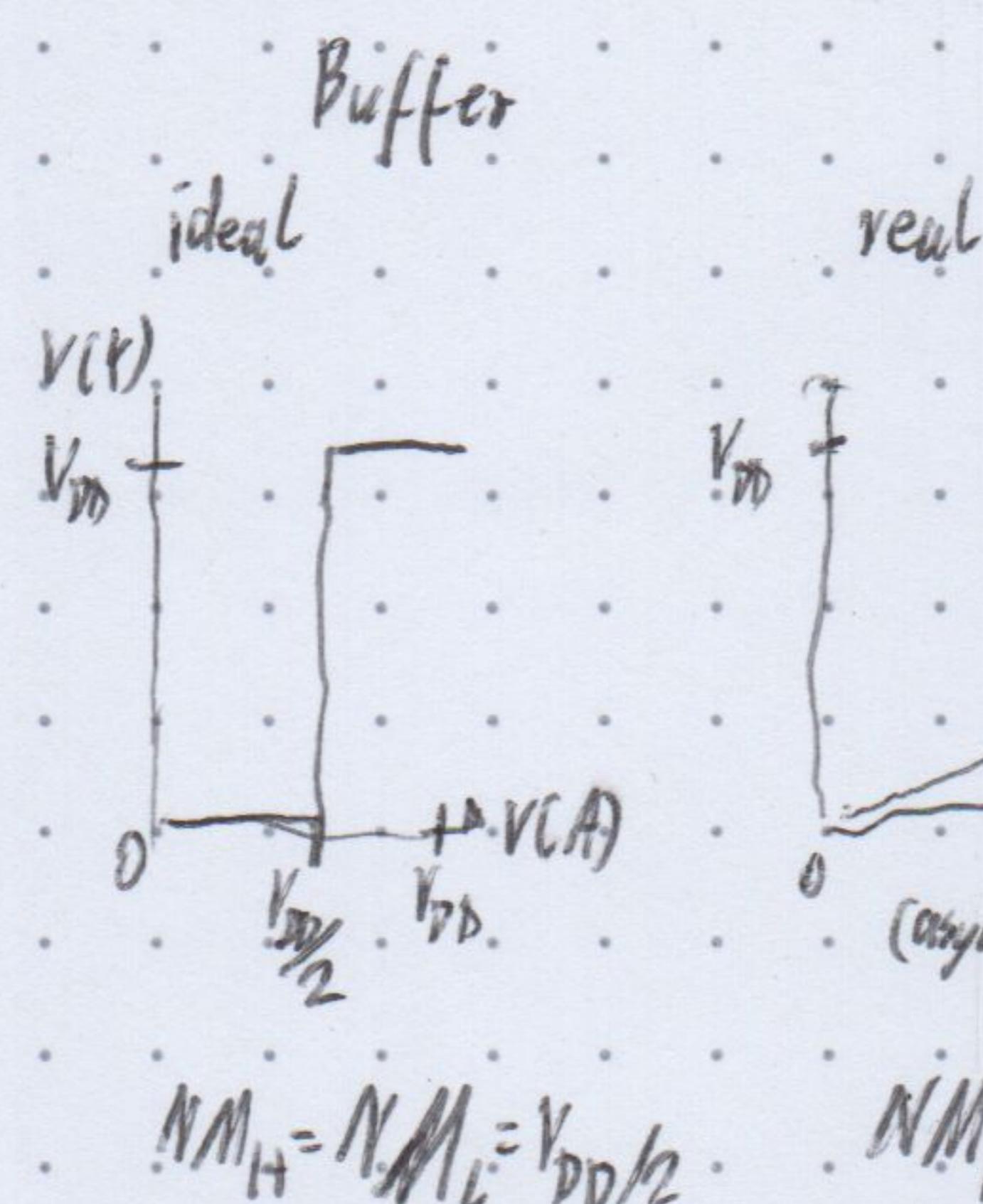
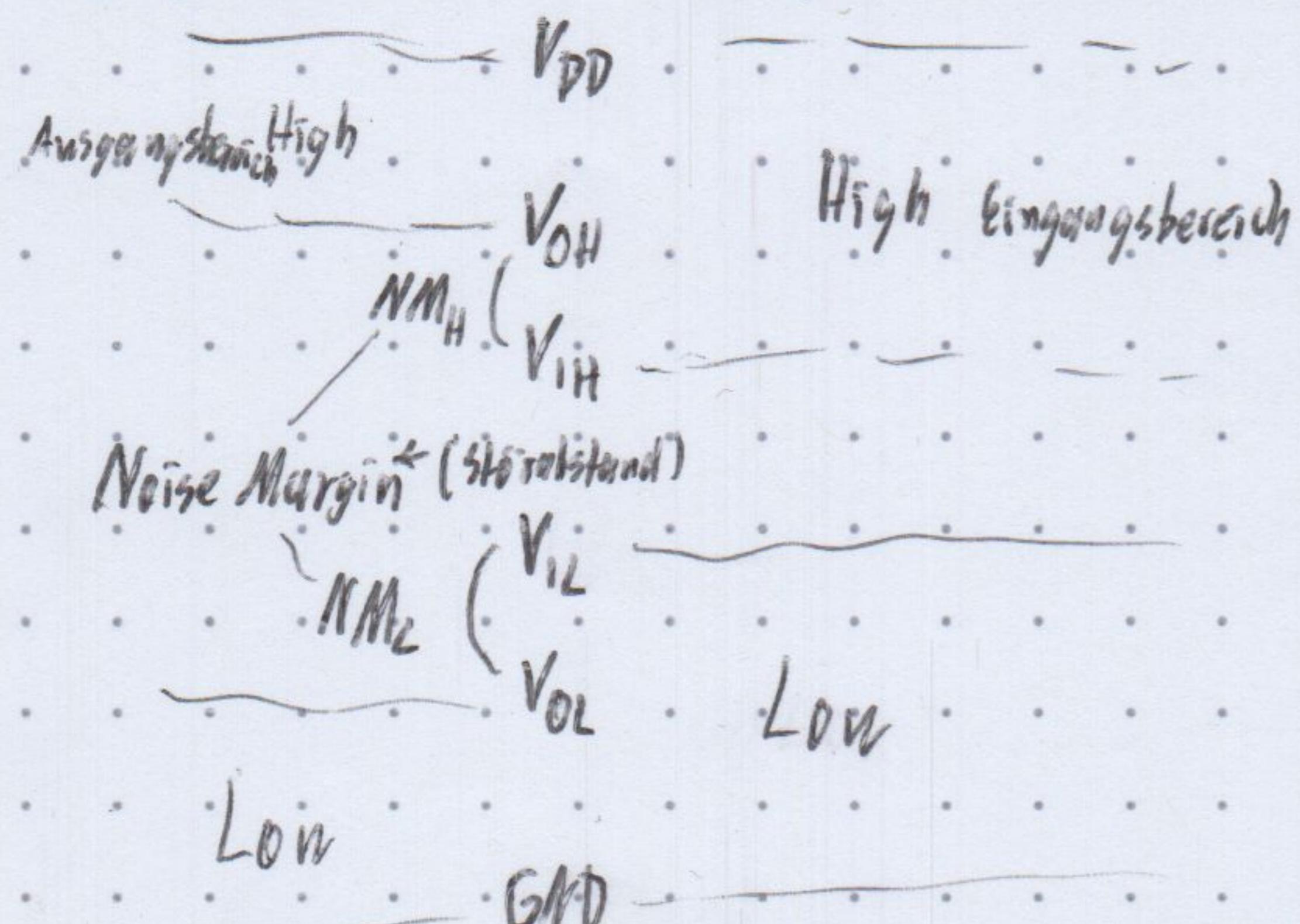
dynamisch: Aufladen Gate-Kapazität
Frequenzf., hälftet Aufladungen
 $I = \frac{Q}{t} = Q \cdot \frac{f}{2} = C \cdot V_{DD} \cdot \frac{f}{2}$

$$P_{\text{dynamic}} = I \cdot V = (C \cdot V_{DD} \cdot \frac{f}{2})(I_{DD}) = \frac{1}{2} C \cdot V_{DD}^2 \cdot f$$

$$P = P_{\text{static}} + P_{\text{dynamic}}$$

physikalische Realisierung

Ausgangscharakteristik Eingangscharakteristik



Transistoren: $\text{Feldeffekttransistoren}$

meist Silizium-basierte Halbleiter

Typ: freie Ladungsträger dotierte Element

n -> (Elektronen) Arsen (As), Phosphor (P)

p + (Loch) Bor (B), Gallium (Ga)

Gate Drain

Source

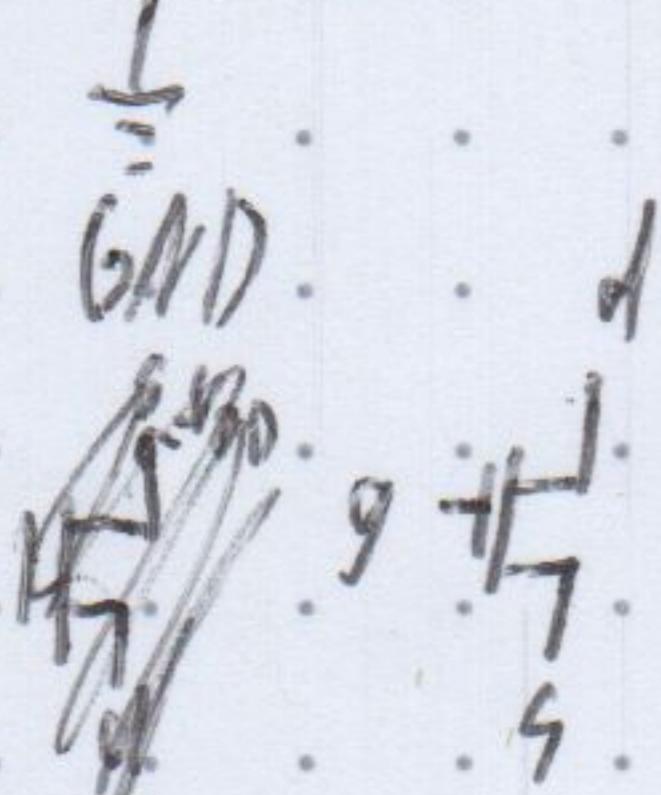
MOSFETs (Metall-Oxid-Halbleiter-Feldeffekttransistoren)

n-Mos

Gate / Polysilizium

source p-Drain

Oxid SiO₂ (Isolator)

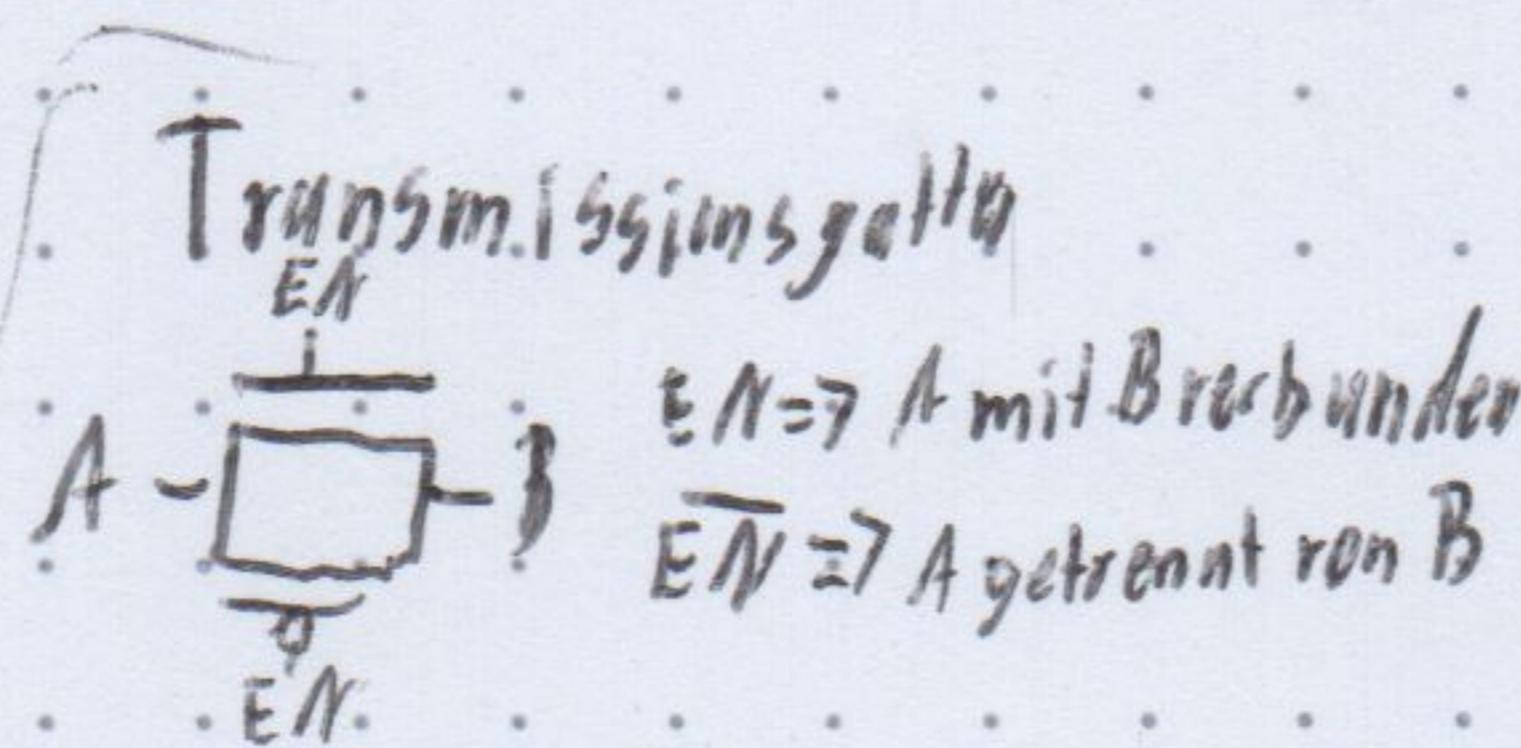
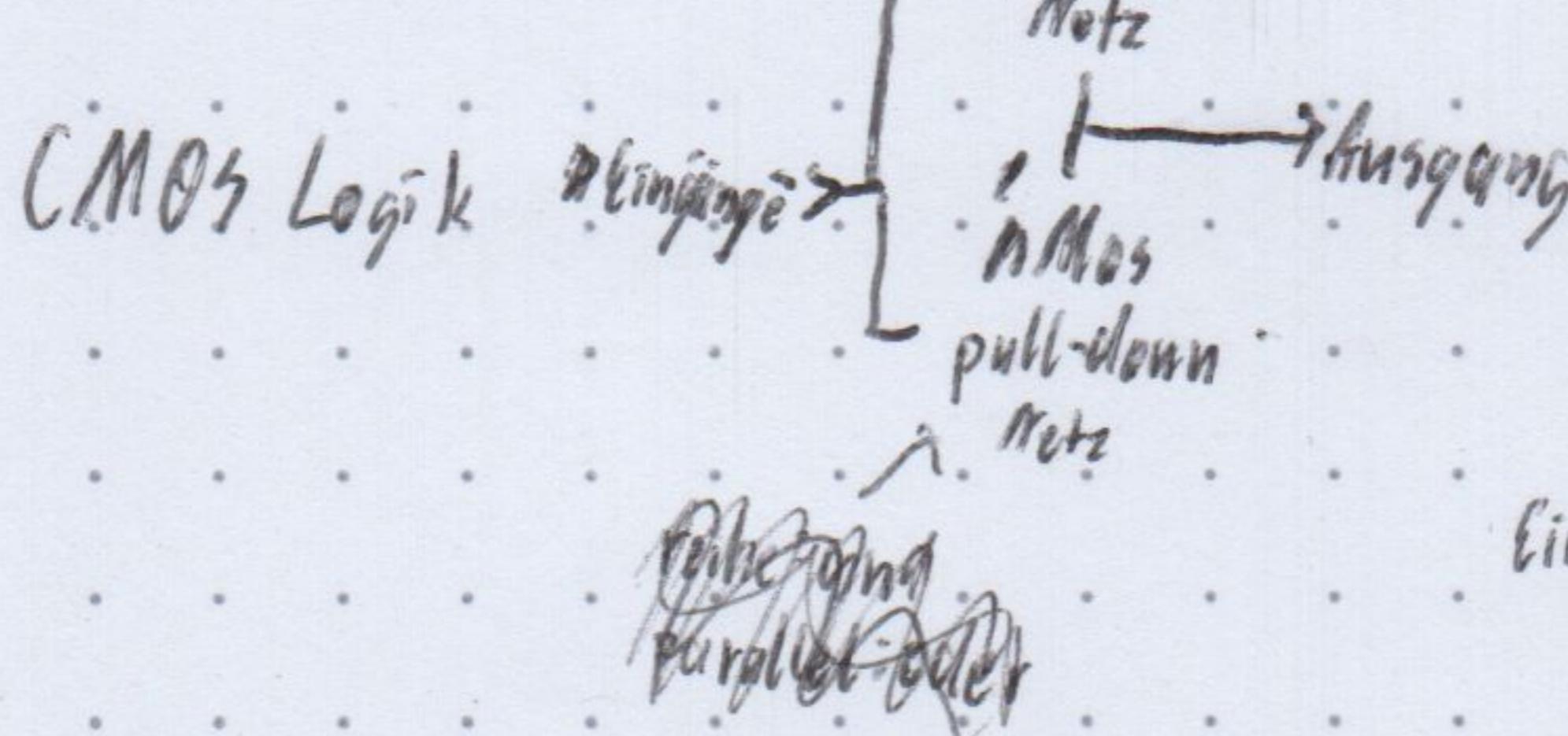


leiten gut 0-en
von source zu drain
deswegen GND an source

BRUNNEN

p-Mos analog n->p getauscht

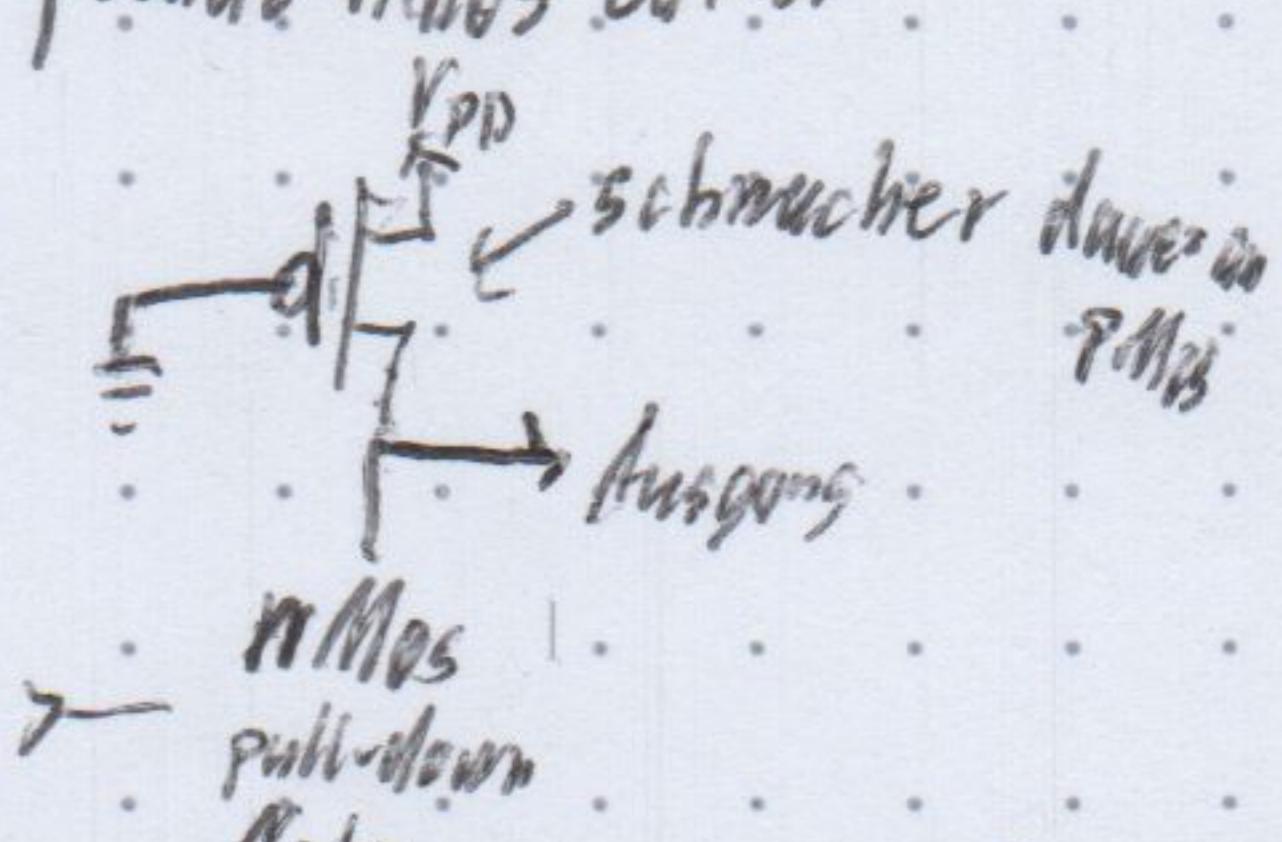
bei N...
reihe: oder
parallel: und



Logikfamilien Spannungspiegel
Transistor Transistor Logic Complementary Metal Oxide Semiconductor

	TTL	CMOS	LV-TTL	LVC-MOS
min V_{DD}	4,75	4,5	3	3
typ. I_{DD}	5	5	3,3	3,3
max V_{DD}	5,25	6	3,6	3,6
V_{DC}	0,4	0,33	0,4	0,3
V_{IL}	0,8	2,35	0,8	0,9
V_{IH}	2,0	3,75	2,0	1,8
V_{OH}	2,4	3,86	2,4	2,7

pseudo-n-Mos Gatter



Eingang \rightarrow n-Mos pull-down Netz
+ keine lange Transistorreihe
- mehr Energie schneller Danach Kurzschluss

B. NOR5

Abstraktion logische Schaltung

Eingänge, Ausgänge, Spezifikation funktionaler Verhalten und Zeitverhalten

Komponenten:

Verbindungsnoten
 { Eingangs-terminale
 Ausgangs-terminale
 Interne Knoten

Schaltungselemente
 { selbst eine Schaltung (Hierarchie)

algebraisch &

algorithmisch

exact: Quine-McCluskey-Methode

exponentiel:

heuristisch: Espresso

ESPRESSA

espresso -> exact Input.esp

optional:

#Kommentar

:i

ilb Namen Eingänge

:ob

ob

:D

:p Anzahl Tabellenspalten

Spalten:

0 Komplement

0 false (optional)

1 Variable

1 true

- nicht implementiert

- Dont care

optional: -e

Ausgabe: Funktion oder minimierte Implikanten

(Espresso kann auch mehrwertige Logik)

Zustandsautomaten

Boolsche Gleichung (beschreibt funktionales Verhalten Schaltnetz)

ohne Operatorpräzedenz

NOT, AND, XOR, OR

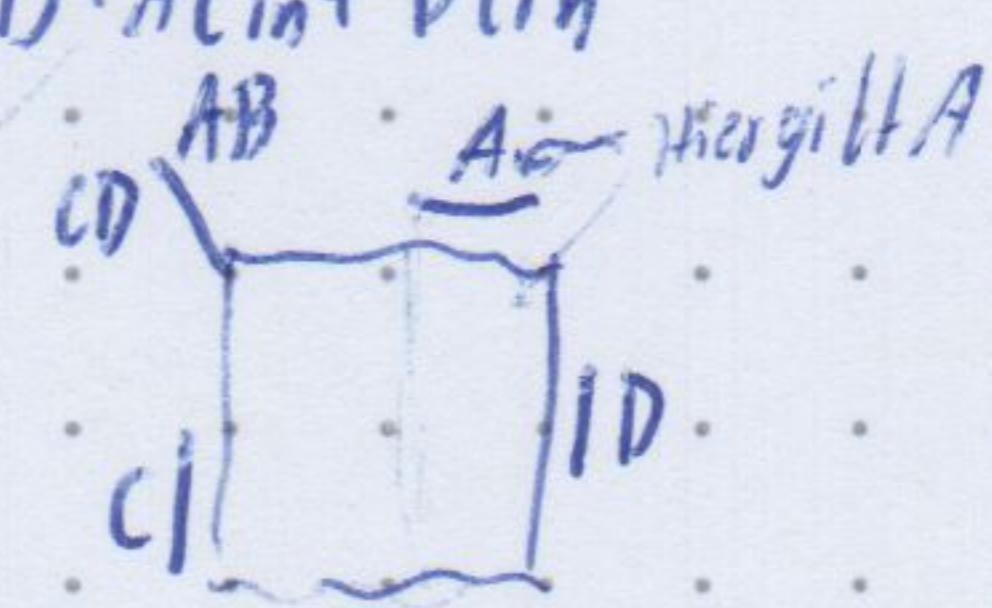
zuerst

als letzten

jeder Output durch Gleichung beschreibbar.

$$\begin{array}{l} A \rightarrow \\ B \rightarrow \\ \text{Lin} \rightarrow \end{array} \xrightarrow{\text{vollständigen}} S = A \oplus B \oplus \text{Lin}$$

$$B^3 \rightarrow B^2 \xrightarrow{\text{Lini}} C_{\text{out}} = AB + AC_{\text{in}} + BC_{\text{in}}$$



* Dont Cares

Minimierung:
 (Bubble Pushing)

graphisch: Karnaugh Diagramme

basiert auf Graycode Optimierung SOP zu Primimplikanten

wenn ein Input als v. Mar input

mit Min(2 N-1)

lookup table

bis auf Kommutation eindeutig
 direkt mit Zweistufige Logik umsetzbar

Logikrealisierung mit Multiplexern
 mit Decodern und OR. (wie DNF)

DNF (Disjunktive Normalform) / SOP (Sum of Products)

KNF (konjunktive Normalform) / POS (Product of sums)

Bool'sche Algebra

Axiome

Duales Axiom

Dualität $\rightarrow A1: B+1 = B=0$

$A1': B \neq 0 \Rightarrow B=1$

Negieren $\rightarrow A2: \bar{0}=1$

$A2': \bar{1}=0$

$A3: 0 \cdot 0=0$

$A3': 1+1=1$

$A4: 1 \cdot 1=1$

$A4': 0+0=0$

$A5: 0 \cdot 1=1 \cdot 0=0$

$A5': 1+0=0+1=1$

Beweisstrategien

Brute Force

auf Axiomen und Theoreme zurückführen

Theorem

T1: $A \cdot 1 = A$

T2: $A \cdot 0 = 0$

T3: $A \cdot A = A$

T4: $\bar{\bar{A}} = A$

T5: $A \cdot \bar{A} = 0$

T6: $A \cdot B = B \cdot A$

T7: $A \cdot (B \cdot C) = (A \cdot B) \cdot C$

T8: $A \cdot (B+C) = A \cdot B + A \cdot C$

T9: $A \cdot (A+B) = A$

T10: $A \cdot B + A \cdot \bar{B} = A$

T11: $AB + \bar{A}C \Leftrightarrow B(C = AB + \bar{A}C)$

T12: $\overline{A \cdot B \cdot C \dots} = \bar{A} + \bar{B} + \bar{C} \dots$

Faz

Duales Theorem

T1': $A+0=A$

T2': $A+1=1$

T3': $A+A=A$

T4': $\bar{\bar{A}}=A$

T5': $A+\bar{A}=1$

T6': $A+B=B+A$

T7': $A+(B+C)=(A+B)+C$

T8': $A+(BC)=(A+B)\cdot(A+C)$

T9': $A+(AB)=A$

T10': $(A+B)\cdot(A+\bar{B})=A$

T11': $(A+B)\cdot(\bar{A}+1)\cdot(B+C)=(AB)\cdot(\bar{A}+1)\cdot(B+C)$

T12': $\overline{A+B+C\dots}=\bar{A}\cdot\bar{B}\cdot\bar{C}\dots$

Bedeutung

Neutralität

Extremum

Idempotenz

Involution

Komplement

Kommutativität

Assoziativität

Distributivität

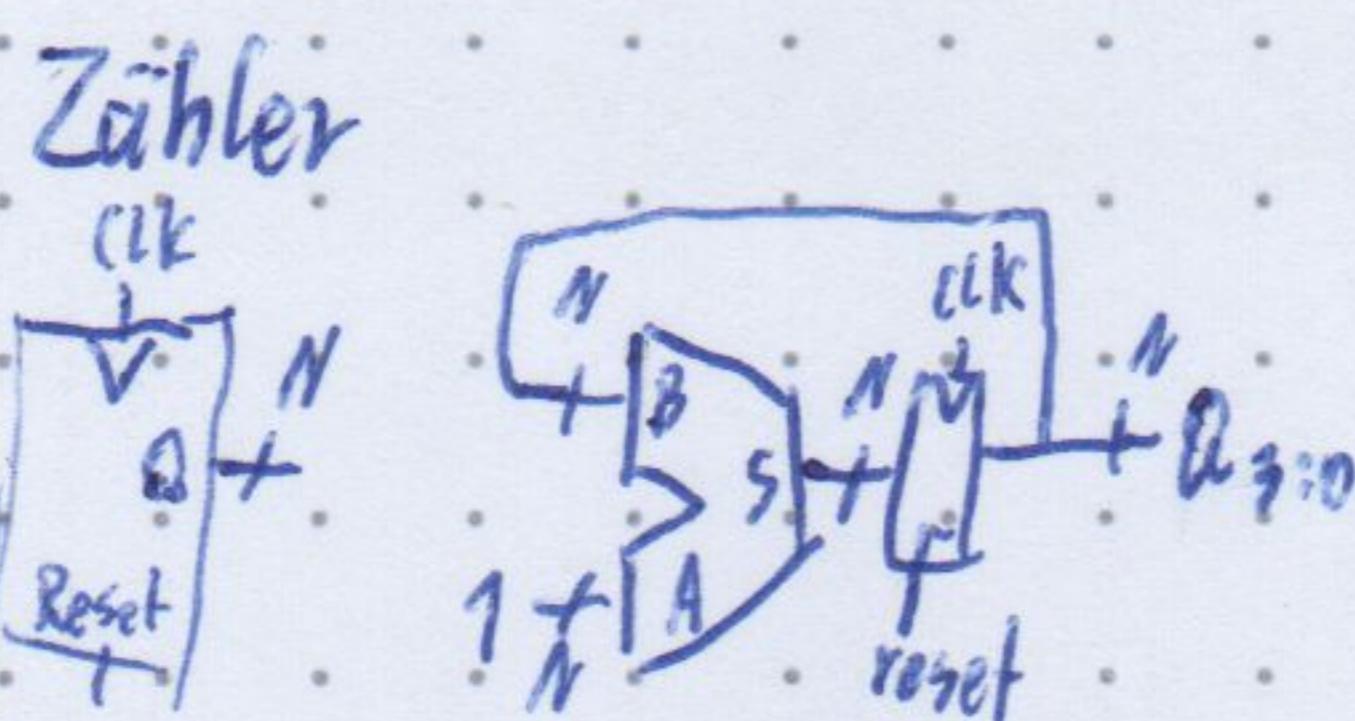
Absorption

Zusammenfassen

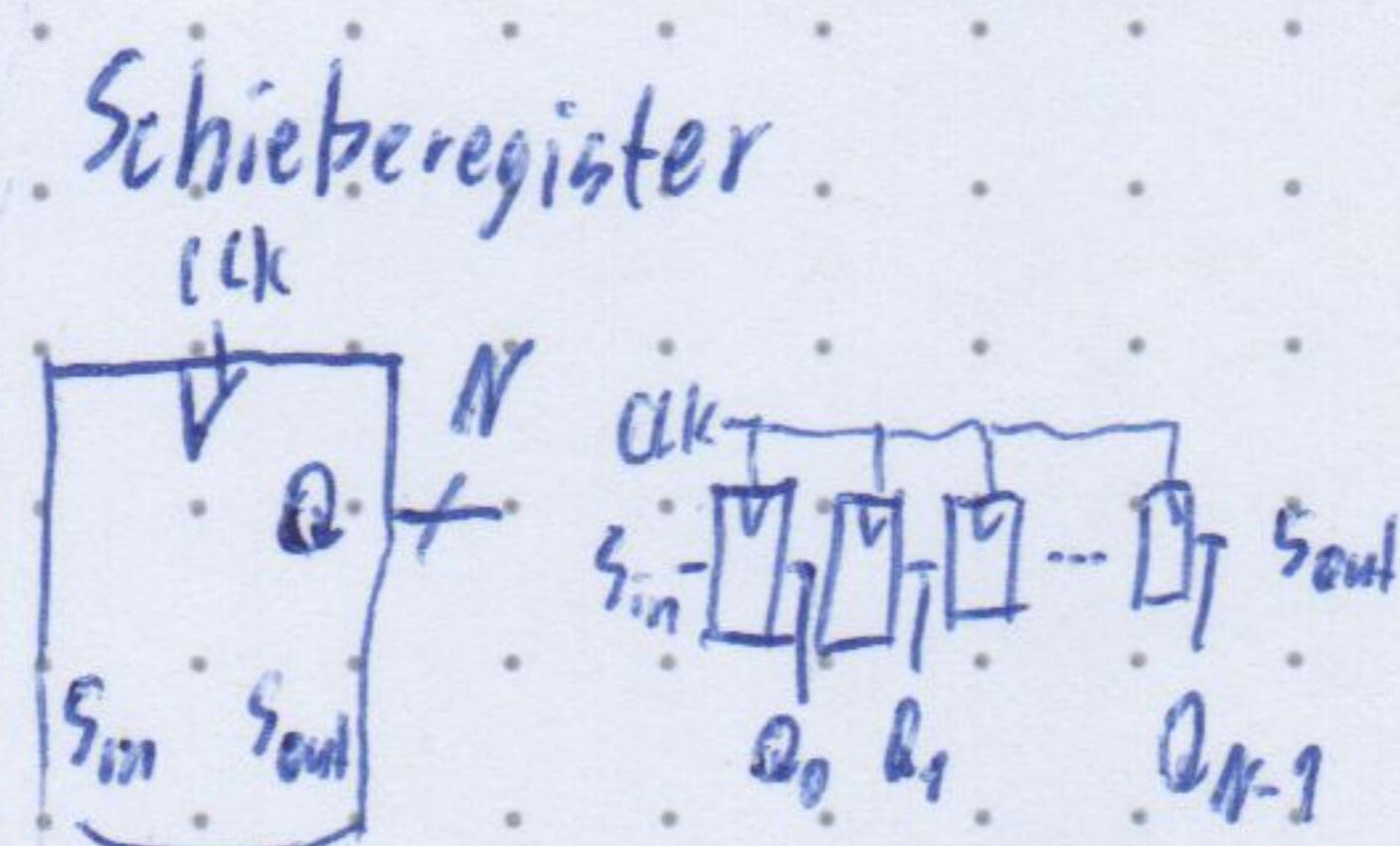
Konsensus

De Morgan

Multiplexer
 $A_0 \rightarrow 0$
 $A_1 \rightarrow 1$
 steuerungsinputs
 $MUX_n: B^{n \times 2^n} \rightarrow B$
 Anzahl inputs

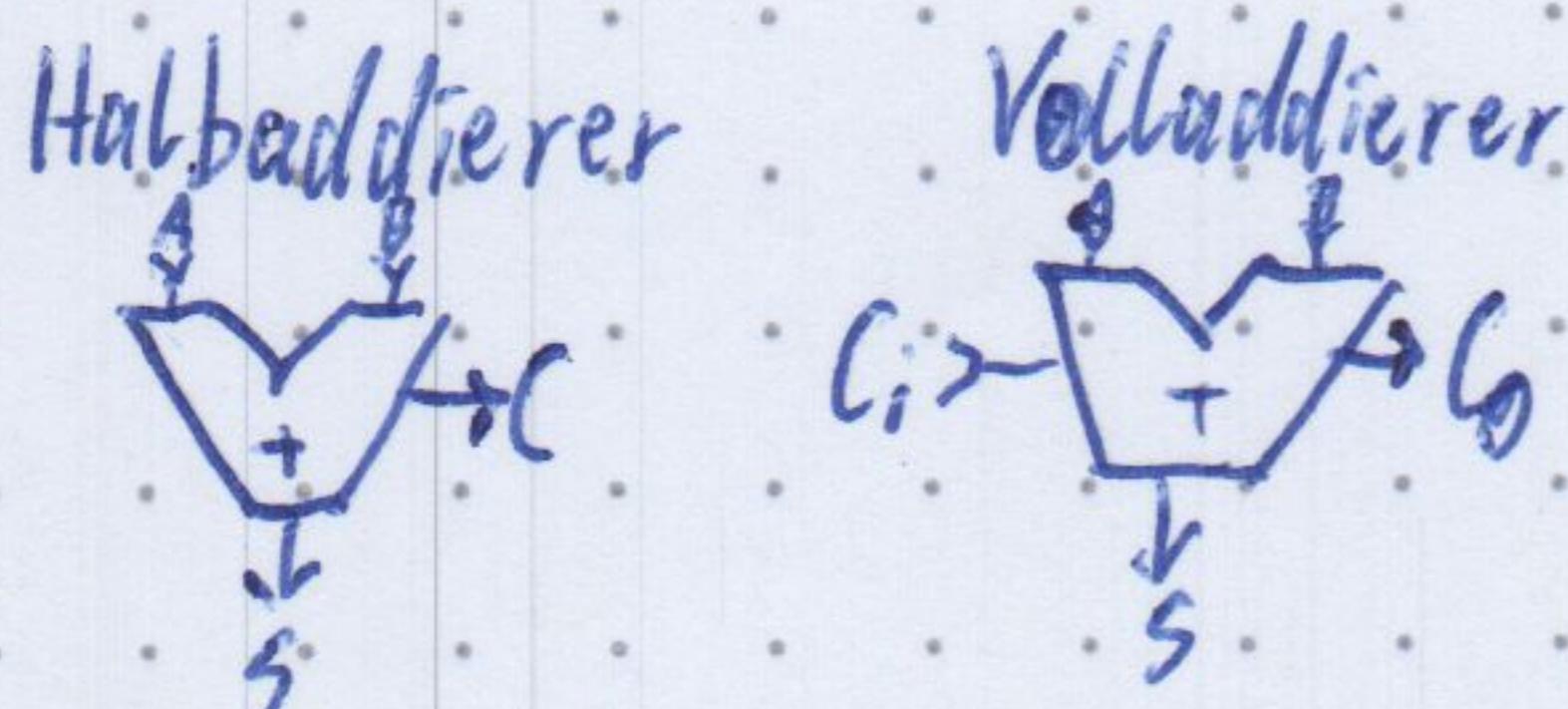


Dekodierer
 $A_2 \rightarrow$ DECODE2
 $0 \rightarrow Y_0$
 $1 \rightarrow Y_1$
 $2 \rightarrow Y_2$
 $3 \rightarrow Y_3$
 $A_0 \rightarrow$ DECODEn
 $B^n \rightarrow B^{(2^n)}$

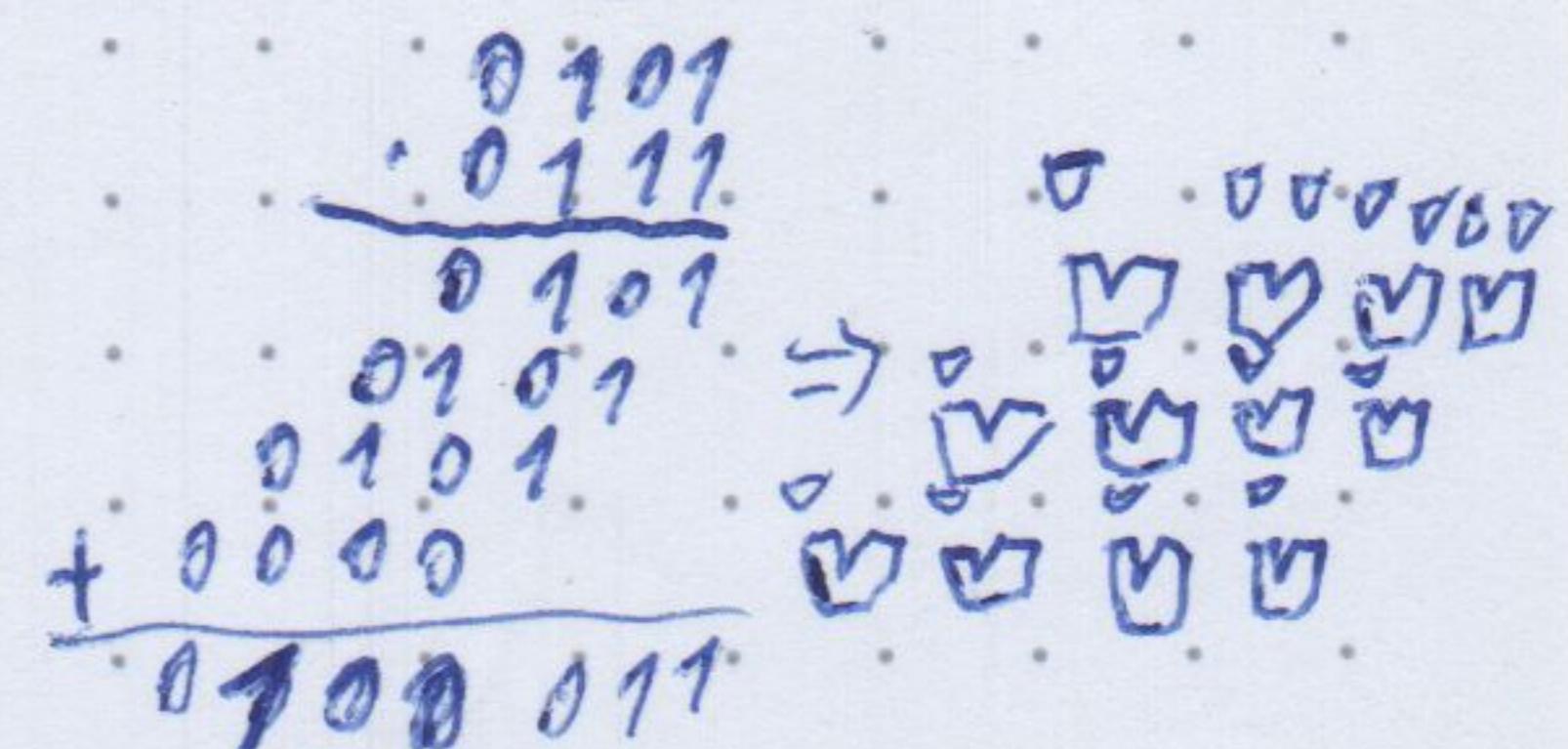


FIFO-Prinzip
 Seriell-Parallel Wandler
 Schieberegister mit parallelem Laden
 zusätzliche Inputs: load, D[0:1:0]
 nach Parallel-Seriell Wandler

Shifter
 logisch: 0-auffüllen (<<>)
 arithmetisch: MSB, bzw. 0-auffüllen (Entspricht Multiplikation)
 rotate (<<< >>>n)



Multiplizierer
 Produktbreite n+m



RCA: Ripple-Carry-Adder (rekursiver Aufbau: hintereinanderreihen)

CSA: Conditional-Sum-Adder
 vorberechnen mit und ohne Carry
 krit. Pfad nur noch erster Adder und Multiplexer

Weiteres Division
 Festkomma / Gleitkomma-Arithmetik

CLA: Carry Lookahead Adder
 Generate-Flag: $G_i = A_i B_i \Rightarrow G_i = G_{i-1} + P_i G_{i-1}$
 Propagate-Flag: $P_i = A_i + B_i$

für Block

$$P_{k,0} = P_{k-1,0} P_{k-2,0} \dots P_0$$

$$G_{k-1,0} = G_{k-1} + P_{k-1} G_{k-2} + \dots + P_{k-1} \dots P_0 G_0 = (G_{k-1} + P_{k-1} (G_{k-2} + P_{k-2} (\dots (P_1 G_0))))$$

Laufzeit für größte Bitbreiten N : $\frac{N}{k} \cdot (t_{pu, AND} + t_{pu, OR})$ \Rightarrow möglichst große Blöcke
 + erster Block aber mehr Ressourcen

somitige: Parallel Prefix Adder (mit Generate Propagate schnell bestimmen)

logarithmischer krit. Pfad

Carry-Safe Adder (parallelere Volladdierer für sum of more than two values)

Subtrahierer: Addition Komplement
 $C_0=1$ und Δ nur B inputs

Vergleich: kleiner als: $A - B$ und dann MEB

Gleichheit: XNOR jeweiliger Bit-Pairs A, B; dann AND-Baum

~~unwirksam~~ sequentielle Logik erlaubt auch instabile (oszillierende) Schaltungen.

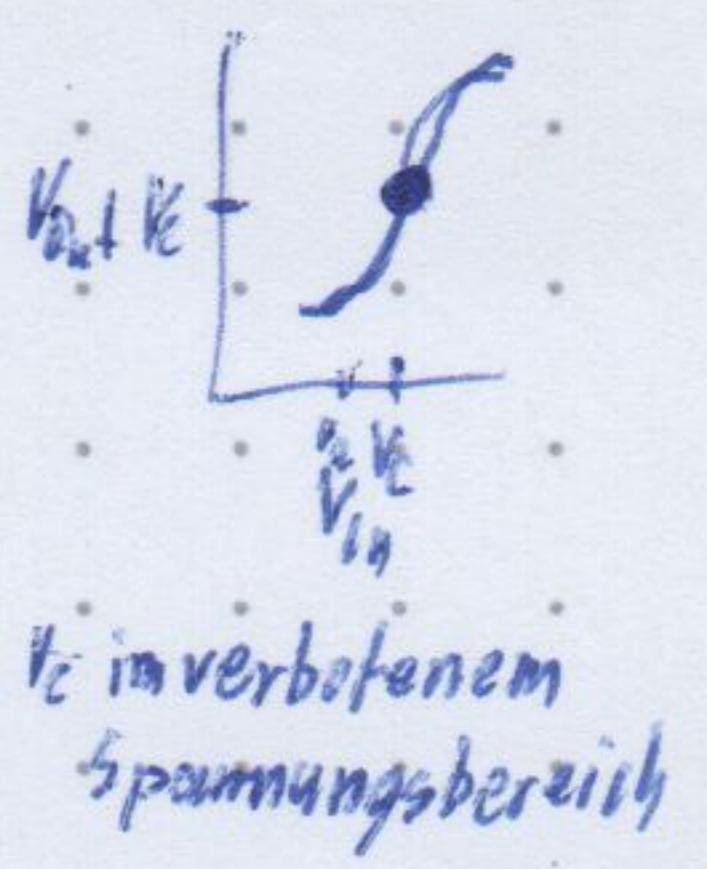
↳ synchrone sequentielle Schaltung

jedes Schaltungselement kombinatorisch oder Register
jeder zyklische Pfad min. 1 Register

min 1!!

alle gleiches
Taktsignal

Metastabilität
zeitlich begrenzter
undefinierter Zustand



V im verbotenen
Spannungsbereich

Zeitanforderung D-Flip-Flop Eingangssignal

t_{setup} vor Taktflanke

t_{hold} nach Taktflanke

$t_a = t_{\text{comp}} + t_{\text{hold}}$ (Abtestzeitfenster) Aperture time

Ausgangssignal:

t_{ccq} contamination delay clock-to-Q

t_{pcq} propagation delay clock-to-Q

Timing Bed.

$$t_{\text{ccq},1} + t_{\text{cd}} \geq t_{\text{hold},2} + t_{\text{skew}}$$

$$t_{\text{pcq},1} + t_{\text{pd}} + t_{\text{setup},2} \leq t_{\text{clk}}$$

schaltet CLK_2 sicher nach CLK_1 kann Taktfrequenz auch steigen

Synchronizer

z.B. Benutzereingaben
, Kommunikationsprotokoll

Timing Bedingungen können bei asynchronen Eingängen
nicht garantiert werden

⇒ zwei D-Flip-Flops hintereinander
erstes kann metastabil werden
zweites höchstwahrscheinlich nicht

Begriffe:

Datensatz

(erster von Input)

räumlich

Latenz (Zeit für einen
Datensatz)

zeitlich

Durchsatz

Pipelining (zeitliche Parallelität)

Ergänzung: Register (Pipelinstufe) \nearrow Durchsatz
nur wenn keine Abhängigkeit von Ergebnis der vorherigen Aufgabe \nearrow Latenz = # Pipelinestufen / Taktfrequenz
Möglichst gleich lang

Mehrwertige Logik

- X mehrfach getrieben → instabil destruktiv: Kurzschluss fast immer Entwurfssfehler
- Z ungetrieben/hochohmig z.B. Bus mit Tristate Buffern (erlaubt Wechsel Kommunikationsrichtung)

Resolutionstabellen (mehr konventionell)

zusammengefaßt: $A \rightarrow Y$

$A \rightarrow D \rightarrow Y$

$B \rightarrow D \rightarrow Y$

$B \rightarrow D \rightarrow Y$

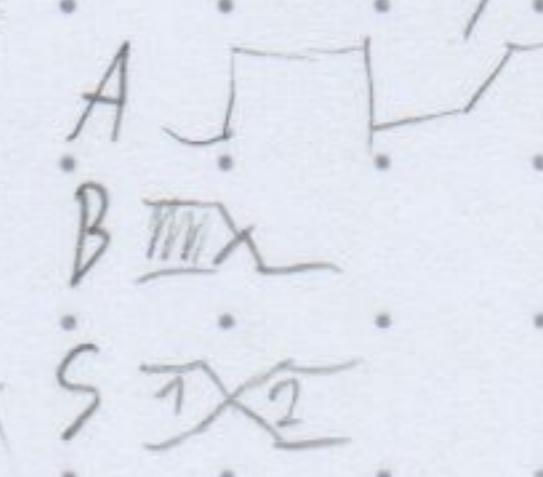
$A \setminus B$	X	0	1	Z
X	X	X	X	X
0	X	0	0	X
1	X	1	1	X
Z	X	0	1	Z

$A \setminus B$	X	Y
X	X	X
0	0	1
1	1	0

$A \setminus B$	X	0	1	Z
X	X	0	X	X
0	0	0	0	X
1	X	0	1	X
Z	X	0	X	X

$A \setminus B$	X	0	1	Z
X	X	X	1	X
0	X	0	1	X
1	1	1	1	1
Z	X	X	1	X

Timing Diagramm



Zeitverhalten (kombinatorische Schaltung)

Einflüsse: Lichtgeschwindigkeit, Kapazitäten, Induktivität, Widerstände, Temperatur, Pfad, t_{pd} , LH , $t_{pd,LH}$

t_{pd} : propagation delay (Ausbreitungsverzögerung) max Summe t_{pd} kritischer Pfad

t_{cd} : contamination delay (Kontaminationsverzögerung) min Summe t_{cd} kurzer Pfad

Glitch:

Input Änderung verursacht mehrere Output Änderungen

- schwer kontrollierbar

sequentielle Schaltung

$\overline{D}_{in} \rightarrow D \rightarrow Q$

Bistabile Grundschaltung

$D_{in} \rightarrow D \rightarrow \overline{Q}$

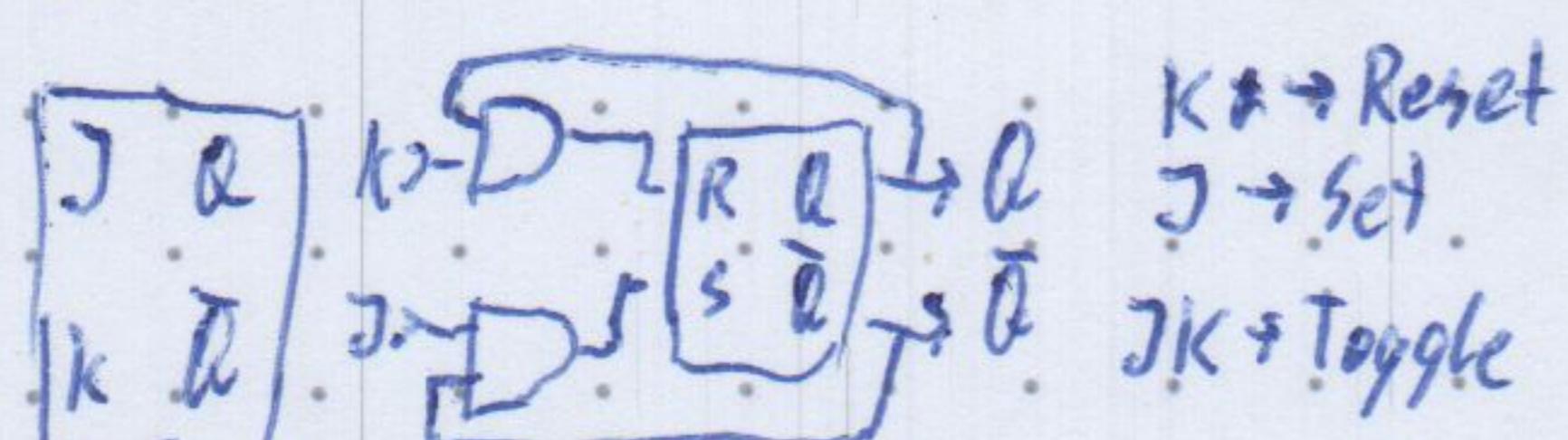
SR-Latch

Test Wertetabelle
Inputs

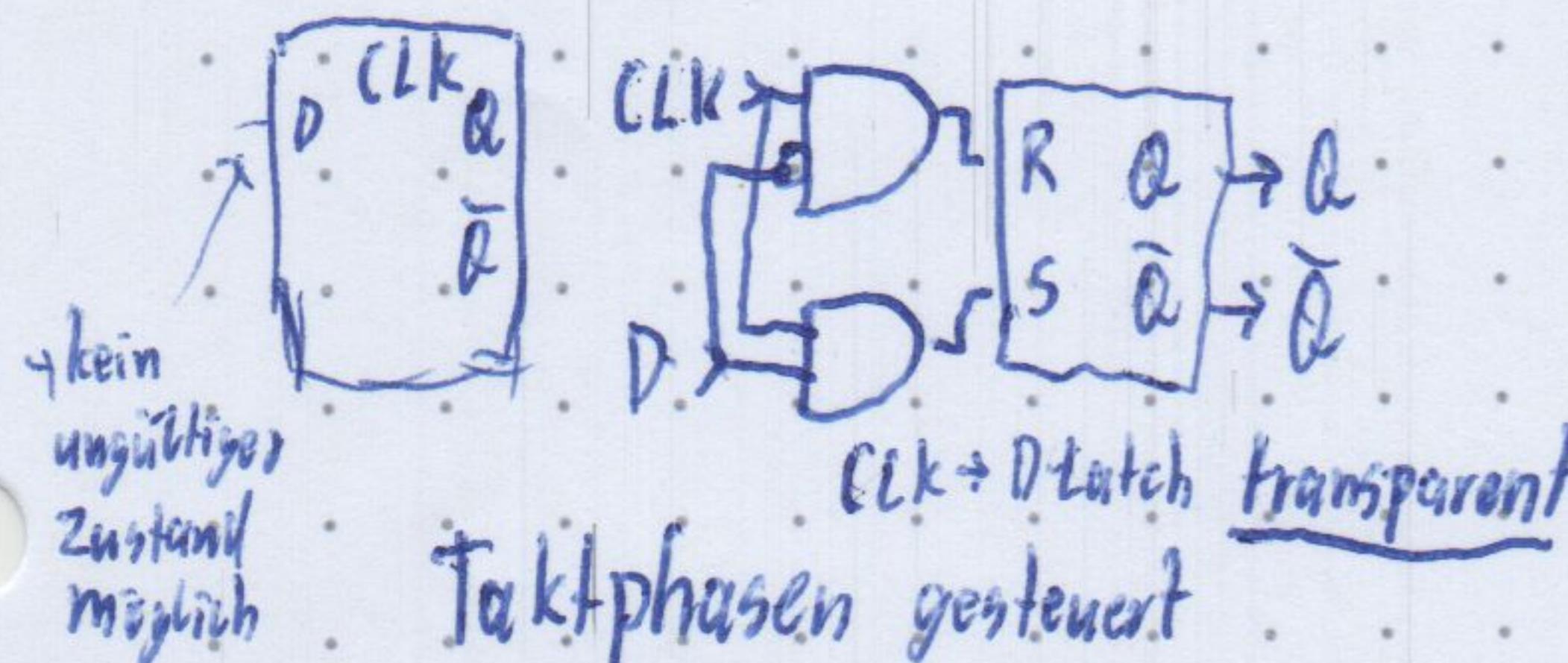


$SR \Rightarrow Q=0, \overline{Q}=0 \Rightarrow$ ungültiger Zustand

JK-Latch

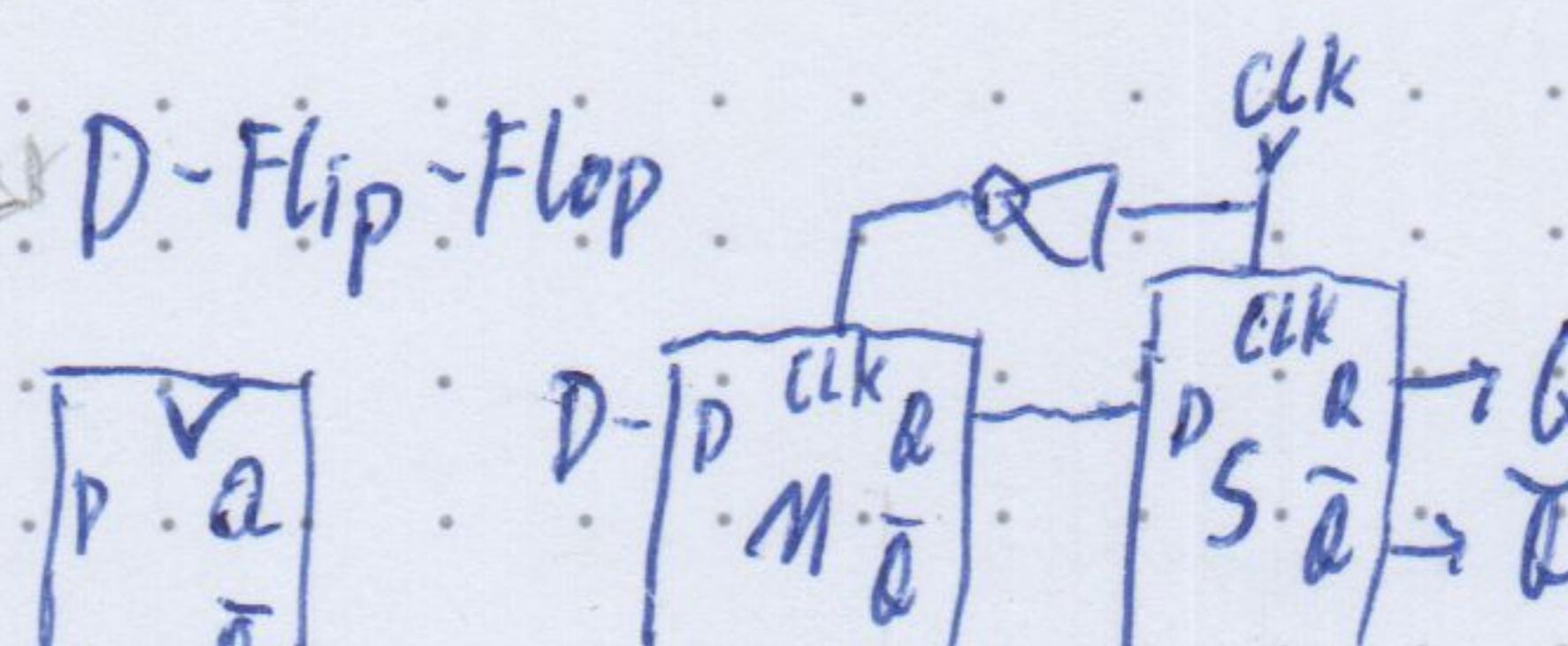


D-Latch



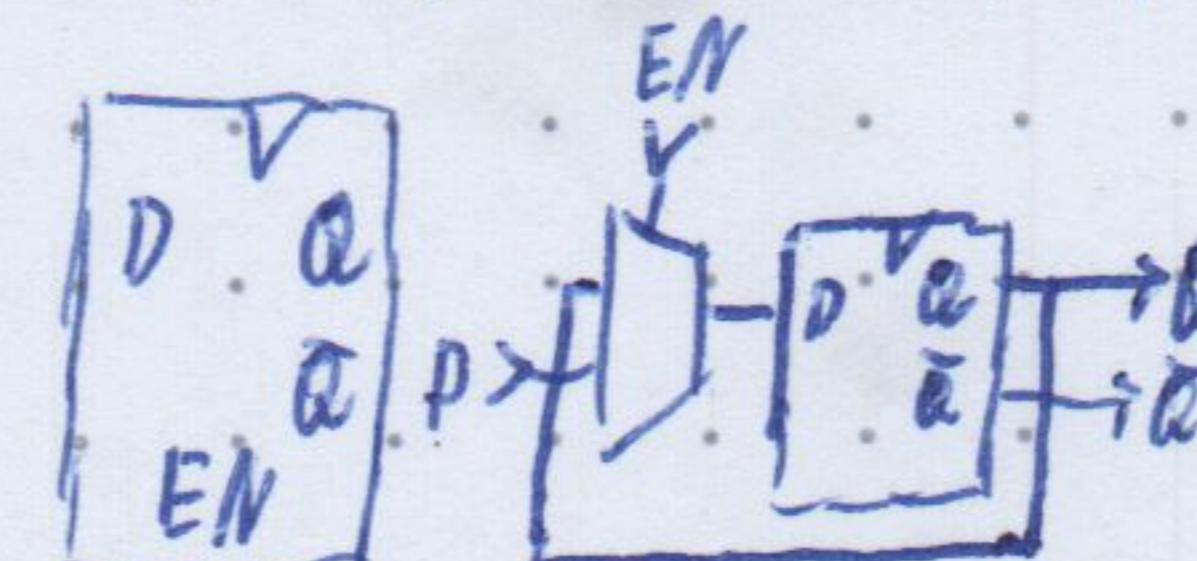
Taktphasen gesteuert

BRUNNEN breites Abtastfenster ⇒ Unschärfer

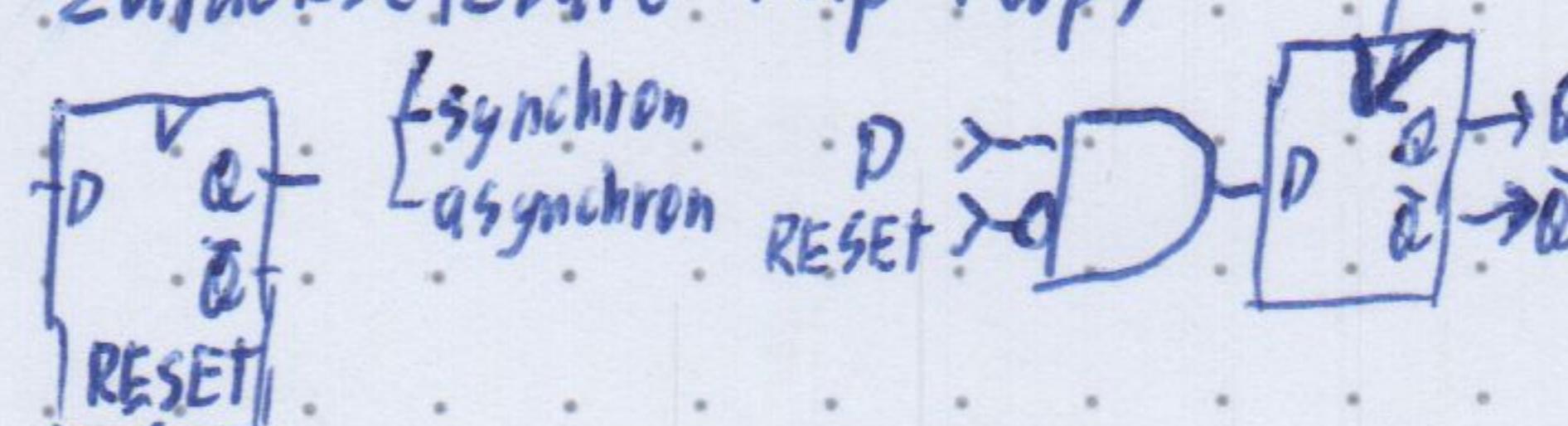


Taktflanken gesteuert
übernimmt Wert kurz vor Flanke

Flip-Flops mit Taktfreigabe



zurücksetzbare Flip-Flops



Register: parallele D-Flip-Flops

Shiftregister: $D_i = Q_{i-1}$

Endliche Zustandsautomaten (Finite State Machines)

synchron sequentielle Schaltung mit:

n Eingabebits

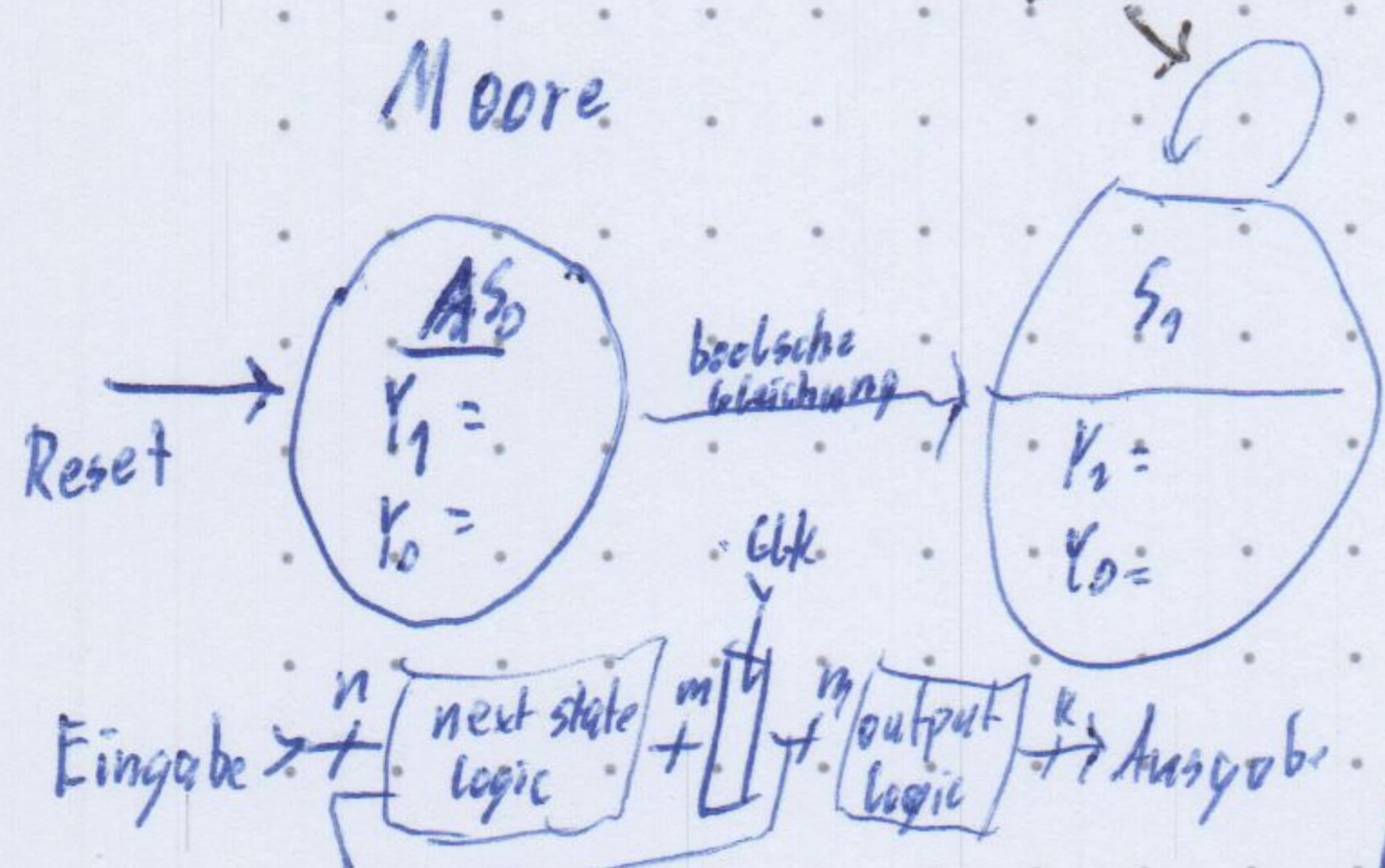
k Ausgabebits

interner Zustand ($m \geq 1$ Bits)

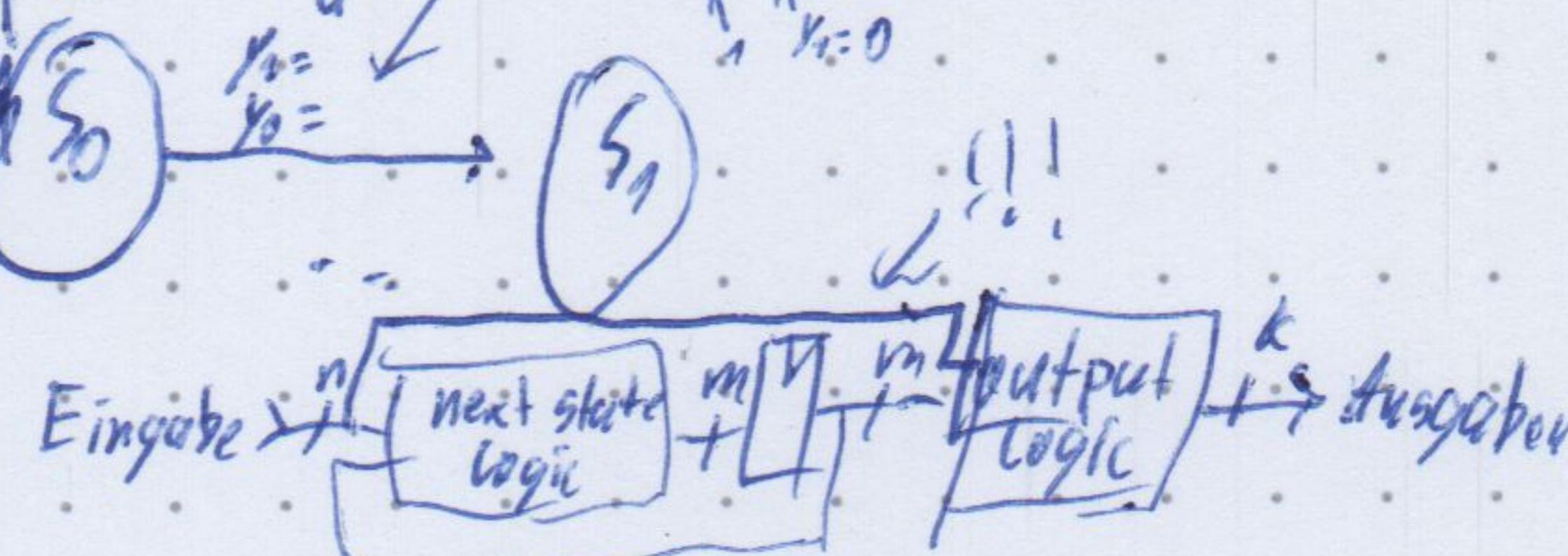
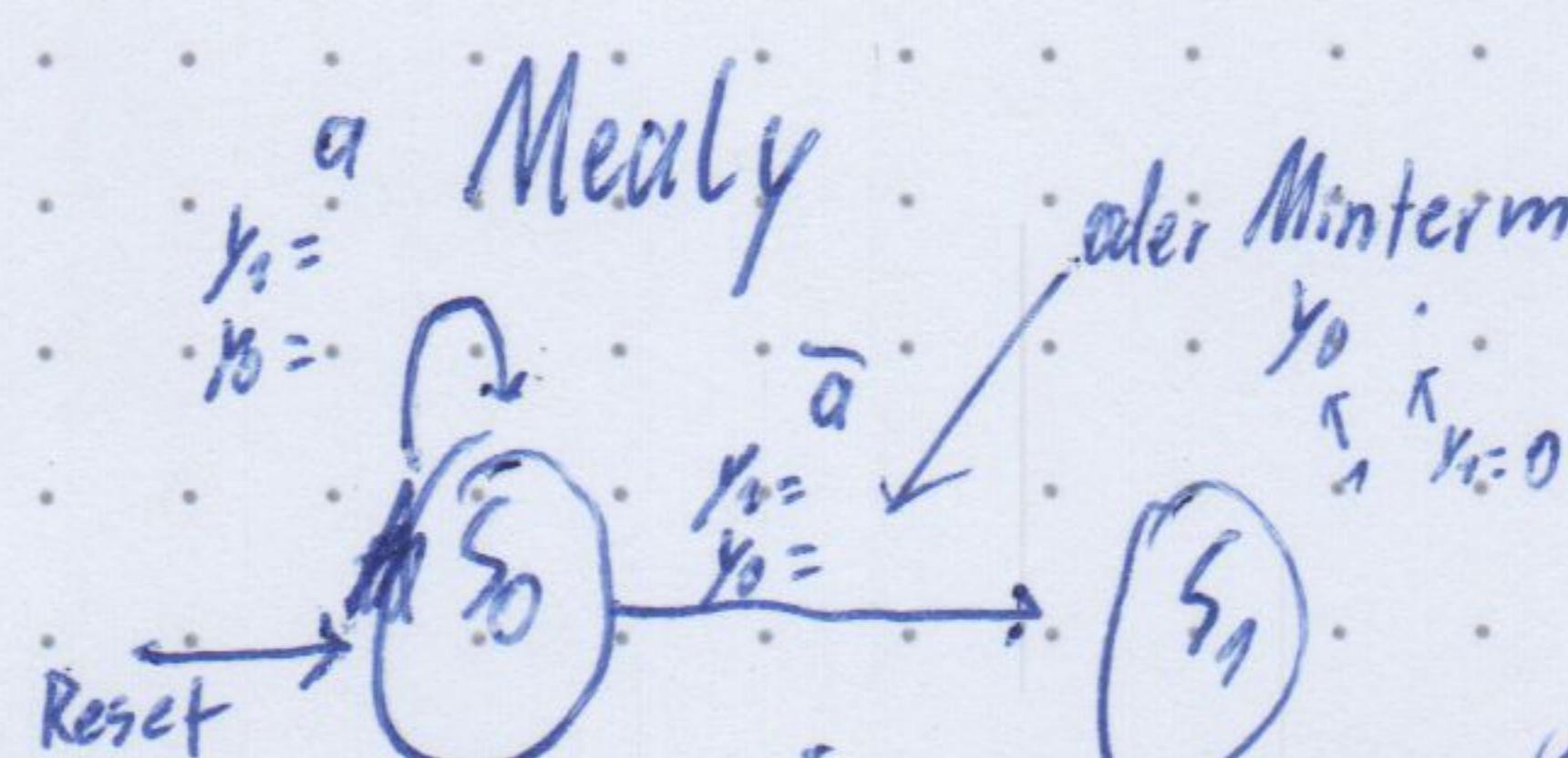
FSM Diagramme

kennen weggelassen werden

Moore



Realisieren in Hardware



Entwurfsverfahren: (definiere Ausgang) \rightarrow Wähle Moore \rightarrow Beschreibung \rightarrow Gleichung \rightarrow Hardware

s := aktueller Zustand textuell, FFSM

s' = nächster Zustand Wahrheitstabelle, kann auch zusammen bei Mealy realisiert abhängig.

Zustandsübergangstabelle $\xrightarrow{\text{bei Mealy}} \text{Ausgangstabelle}$

s	Inputs...	s'	lautet mit Dual Cares	s	Inputs	Outputs

nach außen nicht sichtbar

Zustandskodierung

$s | s_1 s_2$

Eingangs/
Ausgangskodierung

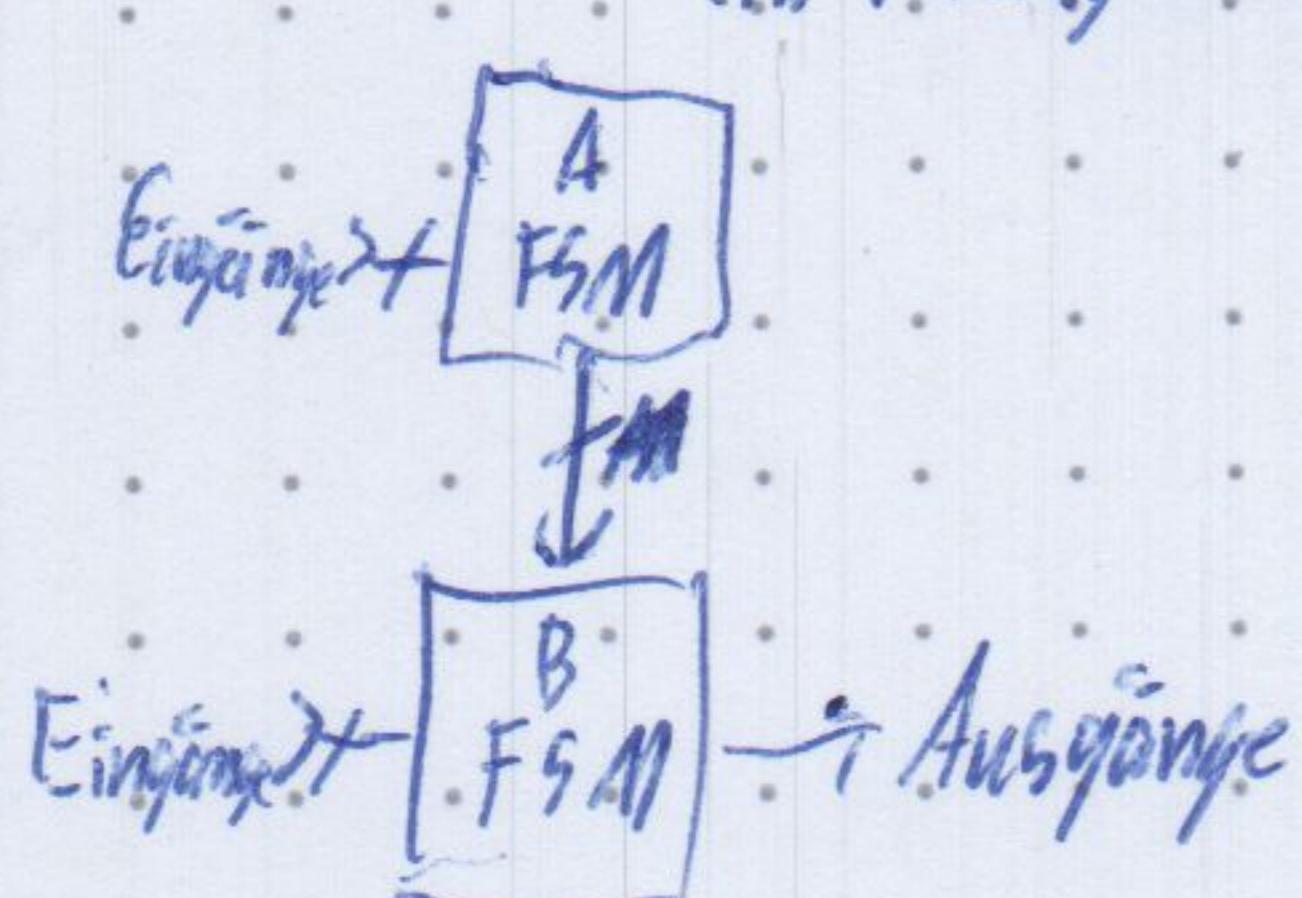
teilweise z.B. One-Hot
für Logik vorteilhaft

tendenziell bei statischen Ausgaben

tendenziell wenn Ausgaben kurzfristige Aktionen auslösen
auf Timing Diagramm! \rightarrow treuert schneller
sichtbar

Zerlegung von Zustandsautomaten (FSM Dekomposition)

(in interagierende/kommunizierende FSMS)



Hardwarebeschreibungssprachen (zur Beherrschung Komplexität)

zunächst → 1960/70 Register-Hochsprache

→ **ConLAN** (Consensus Language)

↳ sollte vereinen

Tendenz zu höheres Abstraktion
→ Highlevel Synthese: z.B. Verilog

VHDL (Very High-speed Integrated Circuits Hardware Description Language)

↳ von U.S. Department of Defence gefördert
IEEE 1076

Verilog HDL

↳ von Gateway Design Automation (Cadence) zur Simulation
IEEE 1364

Simulation

zur Fehlersuche

Synthese

Übersetzung in Netzliste

↳ entspricht Registertransfer-Ebene

beschreibt Schaltungselemente (Logikgatter) und Verbindungsknoten.

Zielarchitektur wichtig! (z.B. ASIC)

Technology mapping (zu FPGA)

System Verilog Weiterentwicklung Verilog (für Verifikation)

Verilog immer noch weit verbreitet

Unterschiede statt logic wire ← für zeitige Zuweisungen
reg ← für always Blöcke

Weiterus in System Verilog

Tasks, Funktionen, Programme

Klassen, Vererbung

Verifikationsunterstützung

fork und join

Präprozessor

...

System Verilog

[`timescale <base>/<precision>] Standardwert

Identifier: Güte sensitive, nicht mit Ziffern beginnen.
Whitespaces irrelevant
// Kommentar
/* multiline comment */

module [H (parameter <NAME> = <value>, parameter ...)] wie Prüfprogramme Anweisungen endmodule

datentyp lokales Signal, Vektor/Bus /* nur Zugriff auf einzelnen Wert keine Reduktion, bitweise Operatoren

assign /* delay */ <signal> = <ausdruck>; z.B. für Speicher
nicht synthetisierbar in LHS Variable oder Port
zuweisung bei Blockierung für einfache Komb Logik

always (sequentielle Anweisung)

always..comb (kombinatorische Anweisung) + nach initial und always Blöcken, immer wenn RHS sich ändert
= blockierende Zuweisung LHS darf nicht in anderen Blöcken geschrieben werden

case (<signals>) / casez in (A) bzw. always..comb
(value); (Anweisung) erlaubt Dont care mit ?

default:
endcase

always..latch (<latch>)

initial (Anweisung)

<submodulename> /* (parameter-map) */ (instanz_name) (port-map); oder, <name> (<value>,...)

Localparam <NAME> = <value>;

genvar <name>; vor generierte Block

Lies zu Bauch Submodule definieren

generate (Generierte Anweisungen) endgenerate.

Anweisungsblock: begin <Anweisungen> end

Anweisungen: case/casez

##> <signal> = <value>; innerhalb eines Blocks ein Signal nur entweder einer

<signal> (= <value>); vorgemerkt erfolgt erst bei fortschreiten

überallin → @(<expr>) #<alle> (pos/neg-edge <expr>) Systemzeit

(cerent? or cerent?)

nicht synthetisierbar (Hazard), aber Sensitivitätsliste

if (<Anweisung> /* else Anweisung */)

else if (<Anweisung> /* else Anweisung */)

Testumgebung

HDL-Programm zum Testen HDL-Programm (in Hardware sehr lange üblich)

getestet Modul Device under test (DUT), Unit under test (UUT)

nicht synthetisierbar

einfach Achtung: meist CLK erstellen

selbstprüfend

selbstprüfend mit Testrektoren & variable Testdaten

\$dumpfile("modulename.vcd"); Timing beachten

\$dumpvars; assert(\$==> craluer) else genor("msg");

\$display("Finished");

\$finish Platzhalter \$d,\$db,\$ch dann, craluer)

%in malutename Stimereformat(s, n, suffix, f) 10

%t für Zeit & Stimetime stimetime & int

aus-mitglied \$readtime & real

Skalierung zählen

→ Anzahl Nachkommaziffern

String vorzeichen

Zeichen nach Zeichen

BRUNNEN

mild normal niedrig

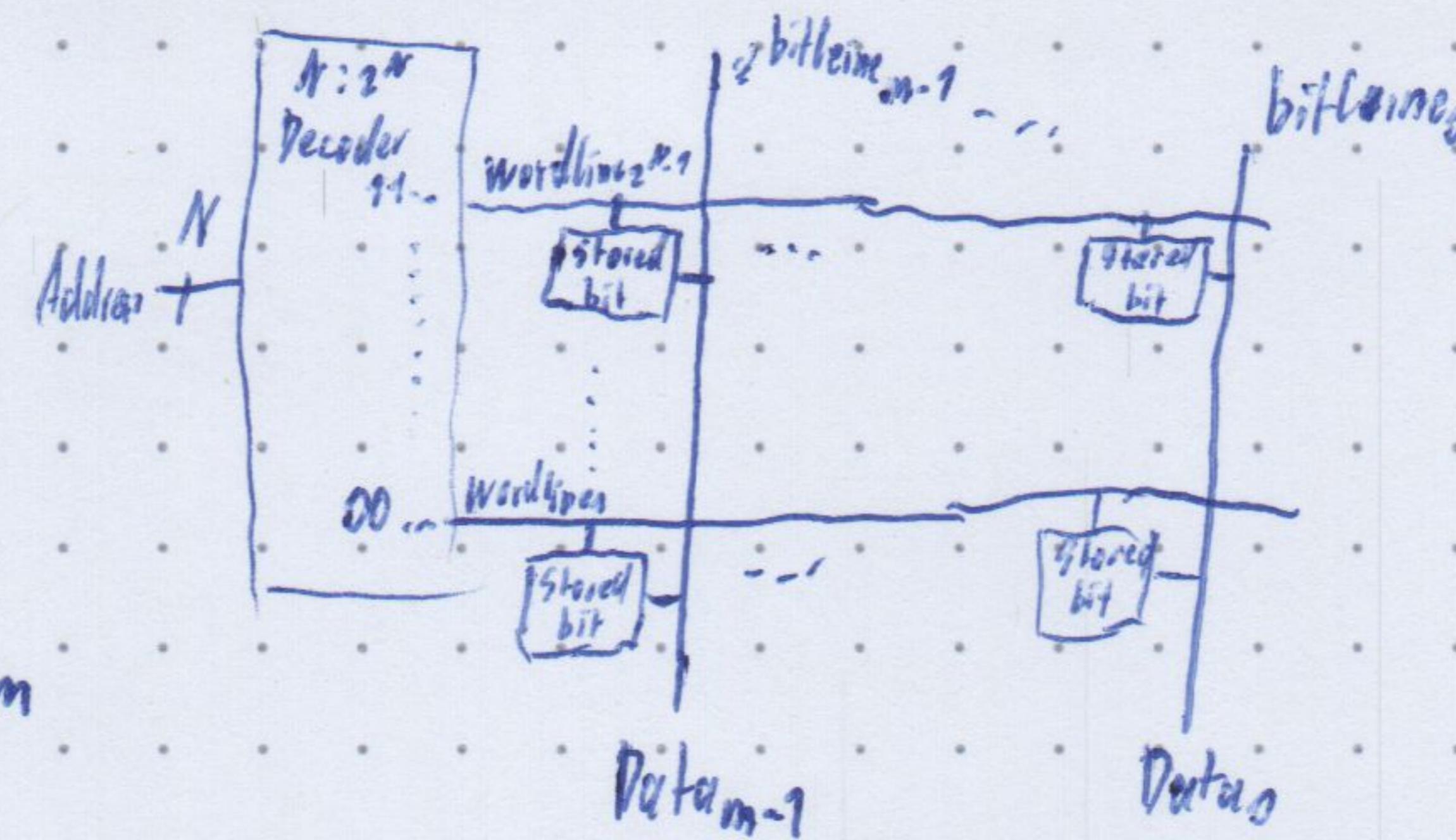
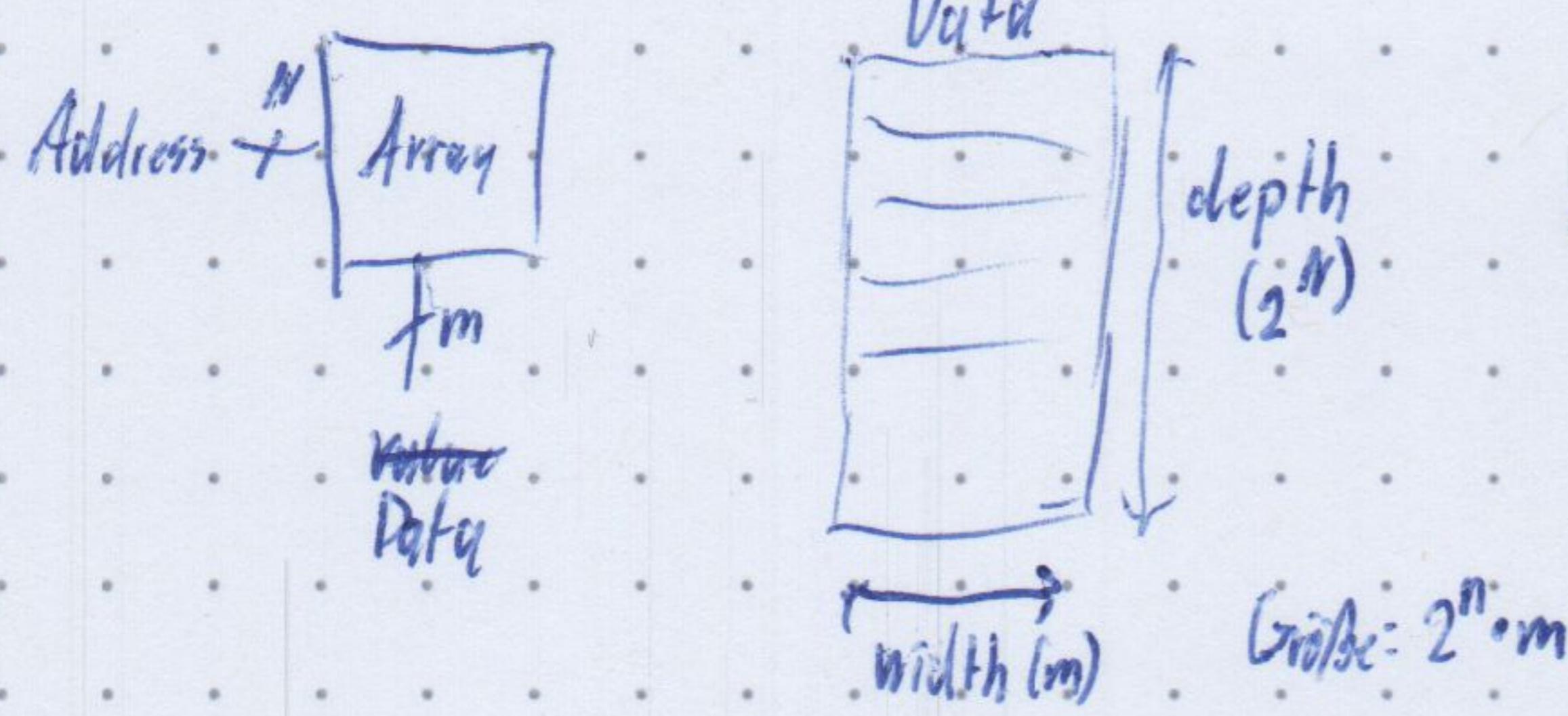
BRUNNEN

FSM-Modellierung in System Verilog

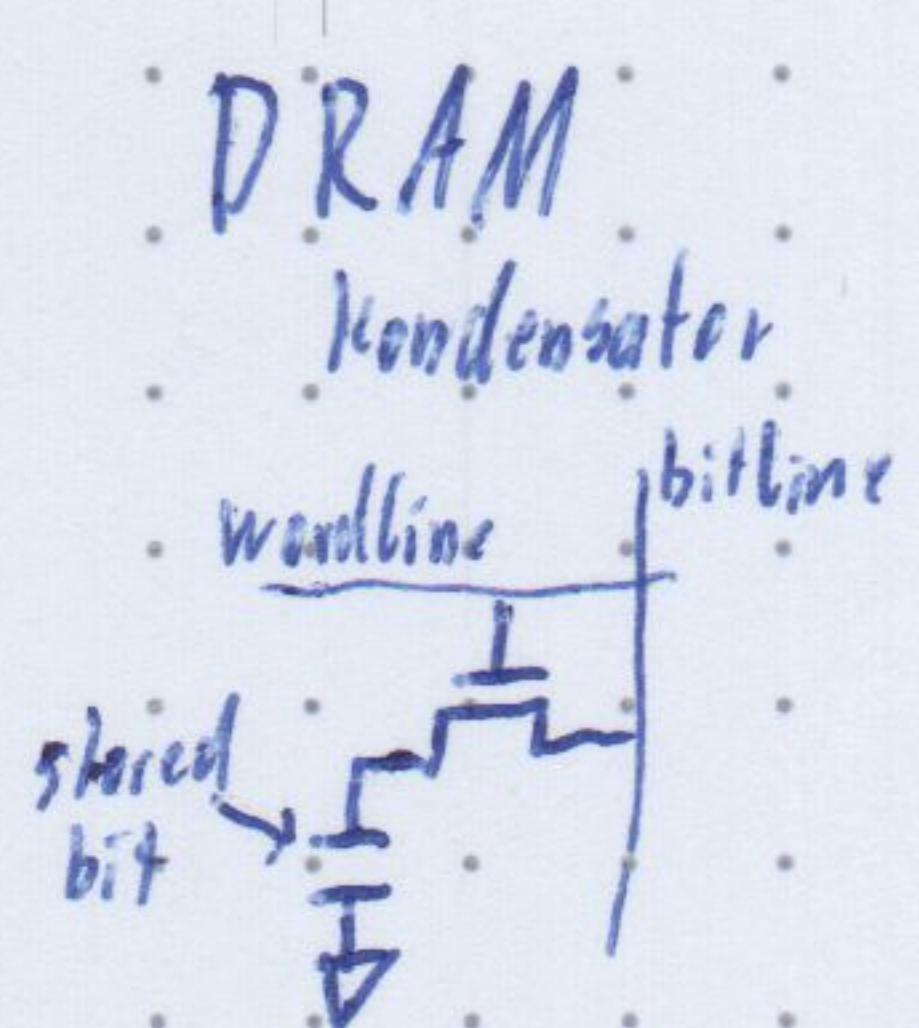
- Logikrektor oder enum für Zustände
- rücksetzbare Flip-Flops als Zustandsspeicher
- kombinatorische next-state Logik durch case in always-comb
- Ausgabe-Logik durch nebenläufige Zuweisung

Speicherfeld

2-dimensionales Bitzellenarray

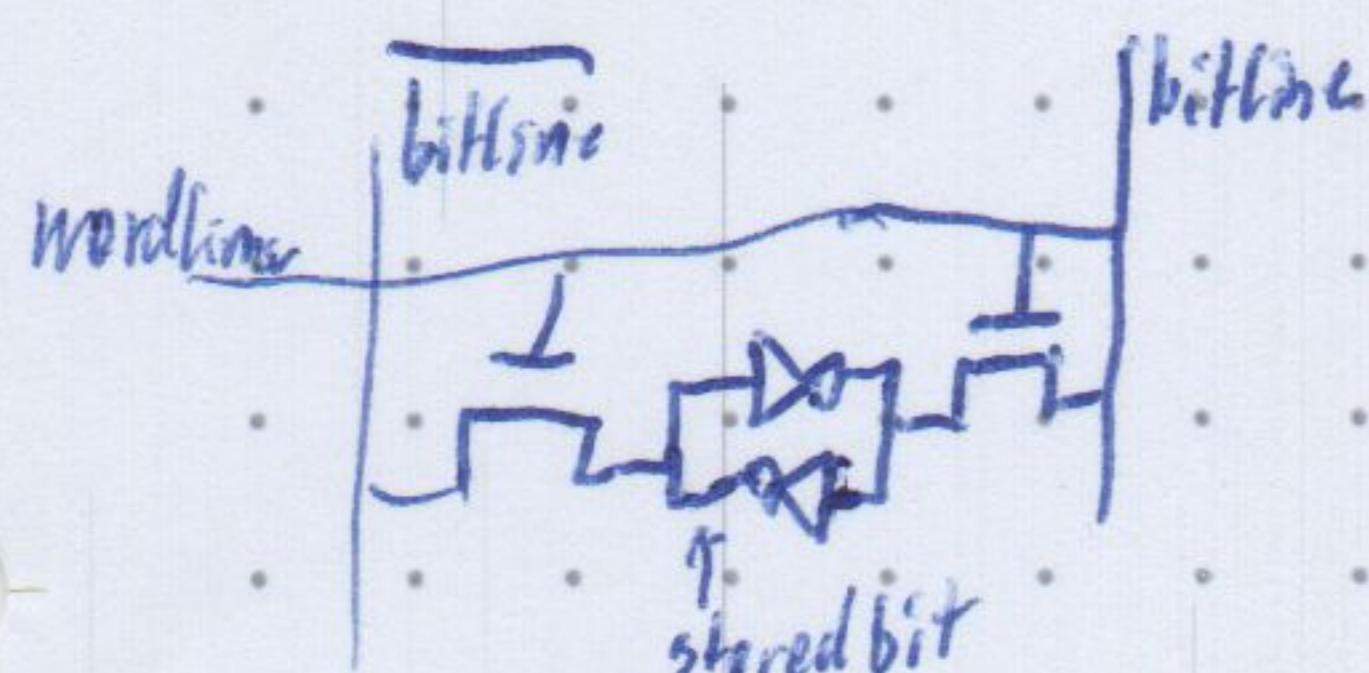


flüchtig d.h. verliert Data bei ausstehen hist. wegen Kontext spalten	Speicher RAM	ROM read only memory
DRAM dynamisch	SRAM static taktkontrolliert	Flash T flüssigkristall

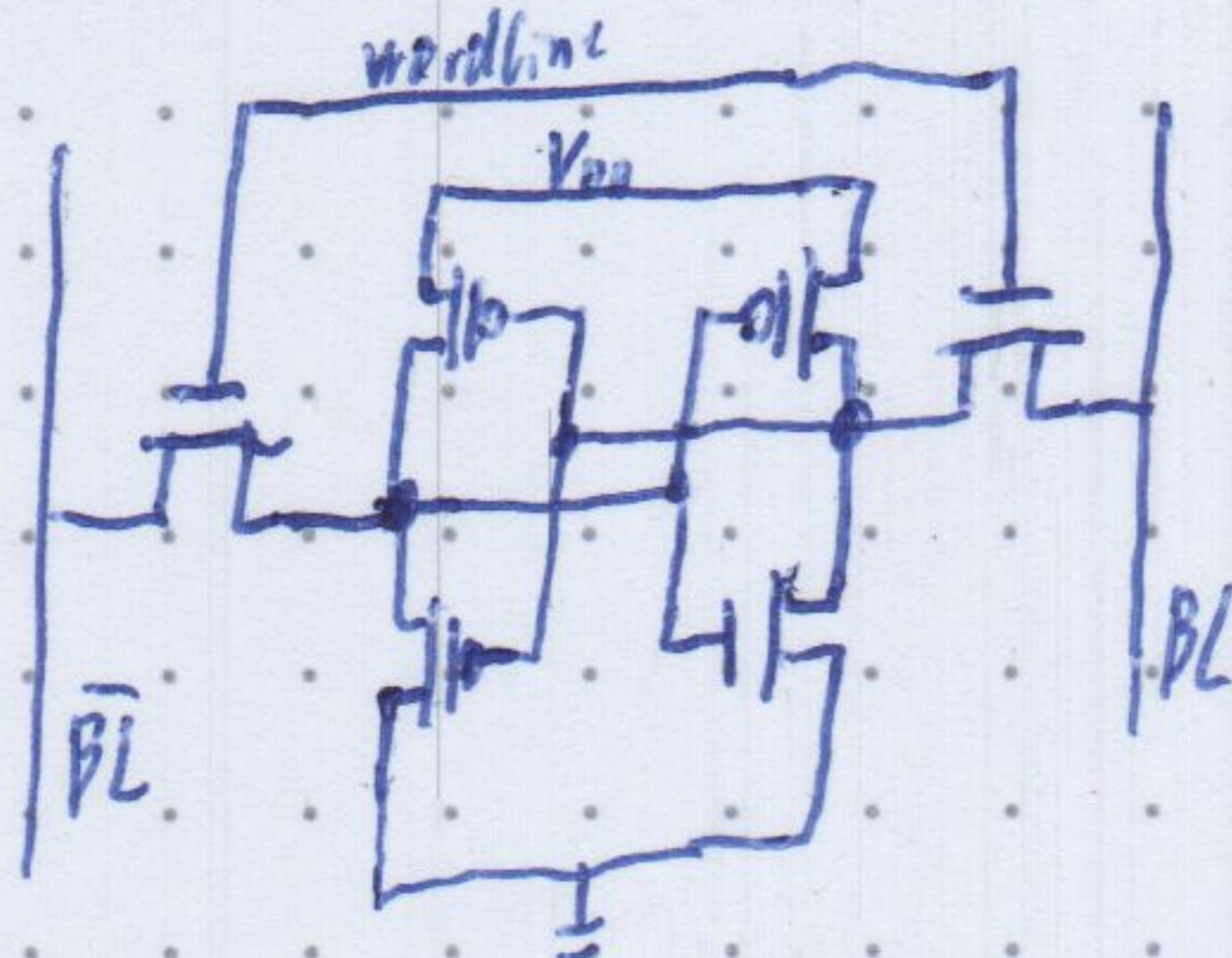


- muss regelmäßig und nachlegen
- neu geschrieben werden

SRAM

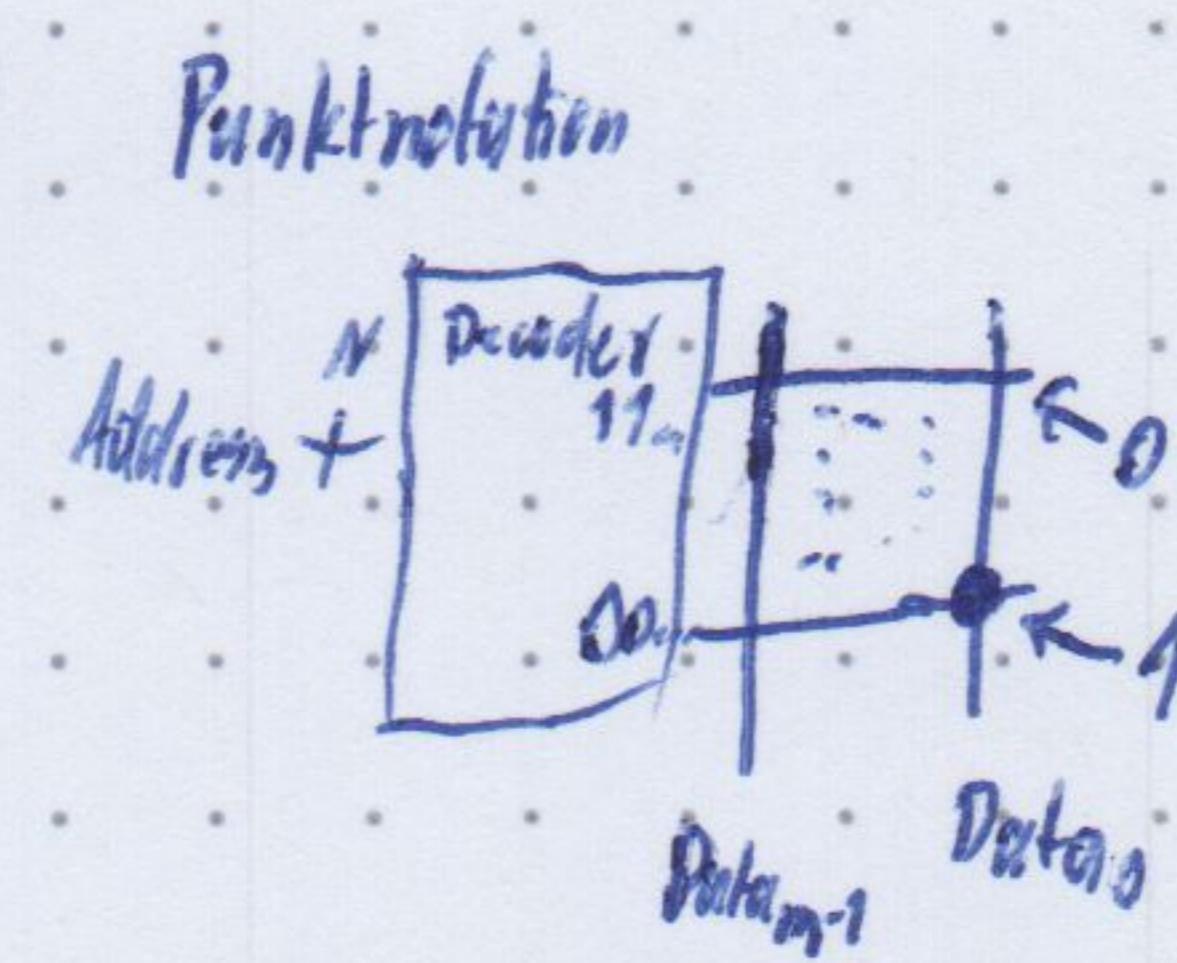
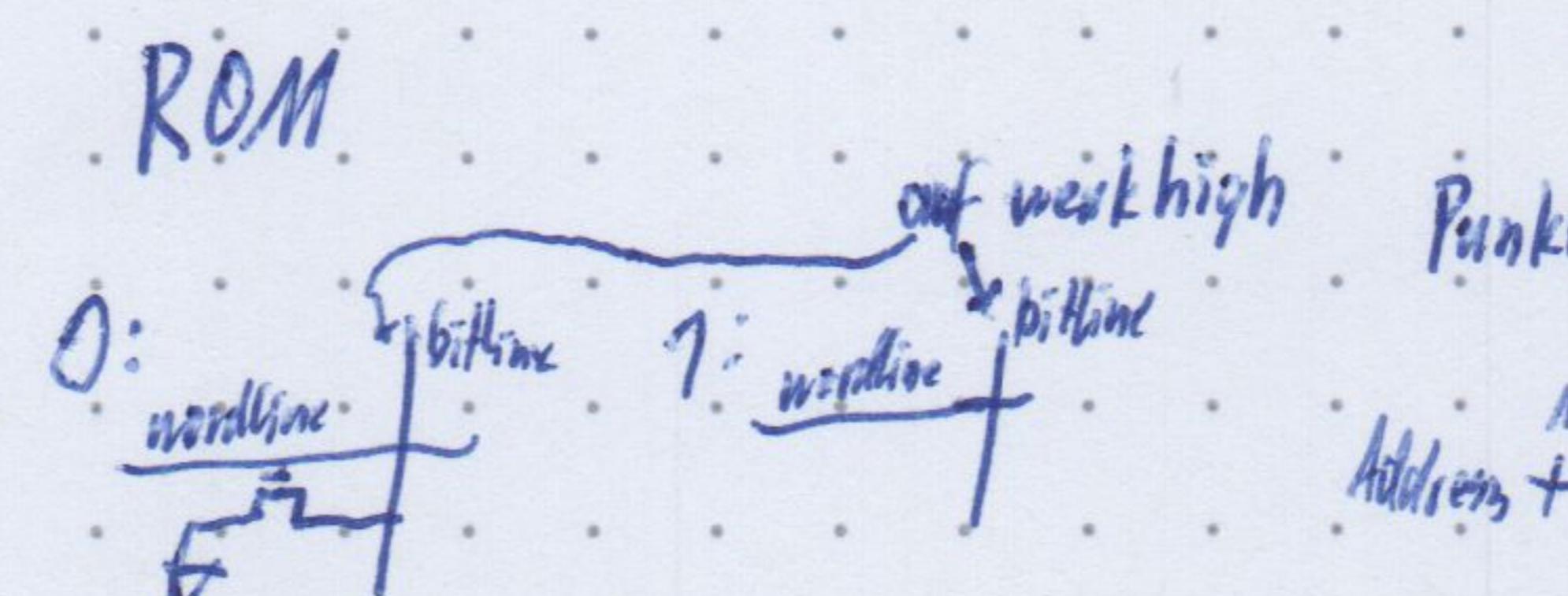


Schreiben erfordert Kurzschluss

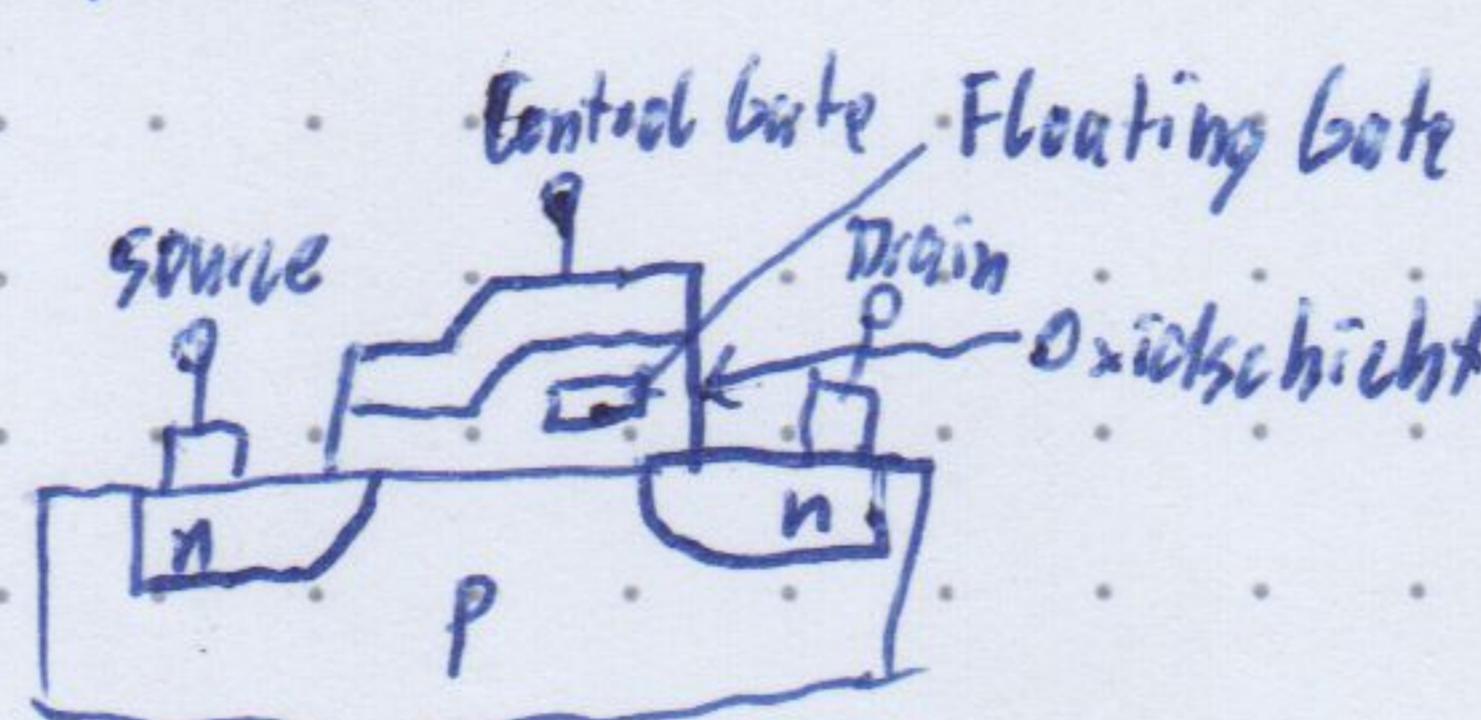


Logik via Speicherfeld
Lookup-table (LUT)

System Verilog
(ROM aus mit Fixwerten
Literalketten)



Flash



Laden: Source = 0V
Control Gate = Drain = 12V

durchfahren

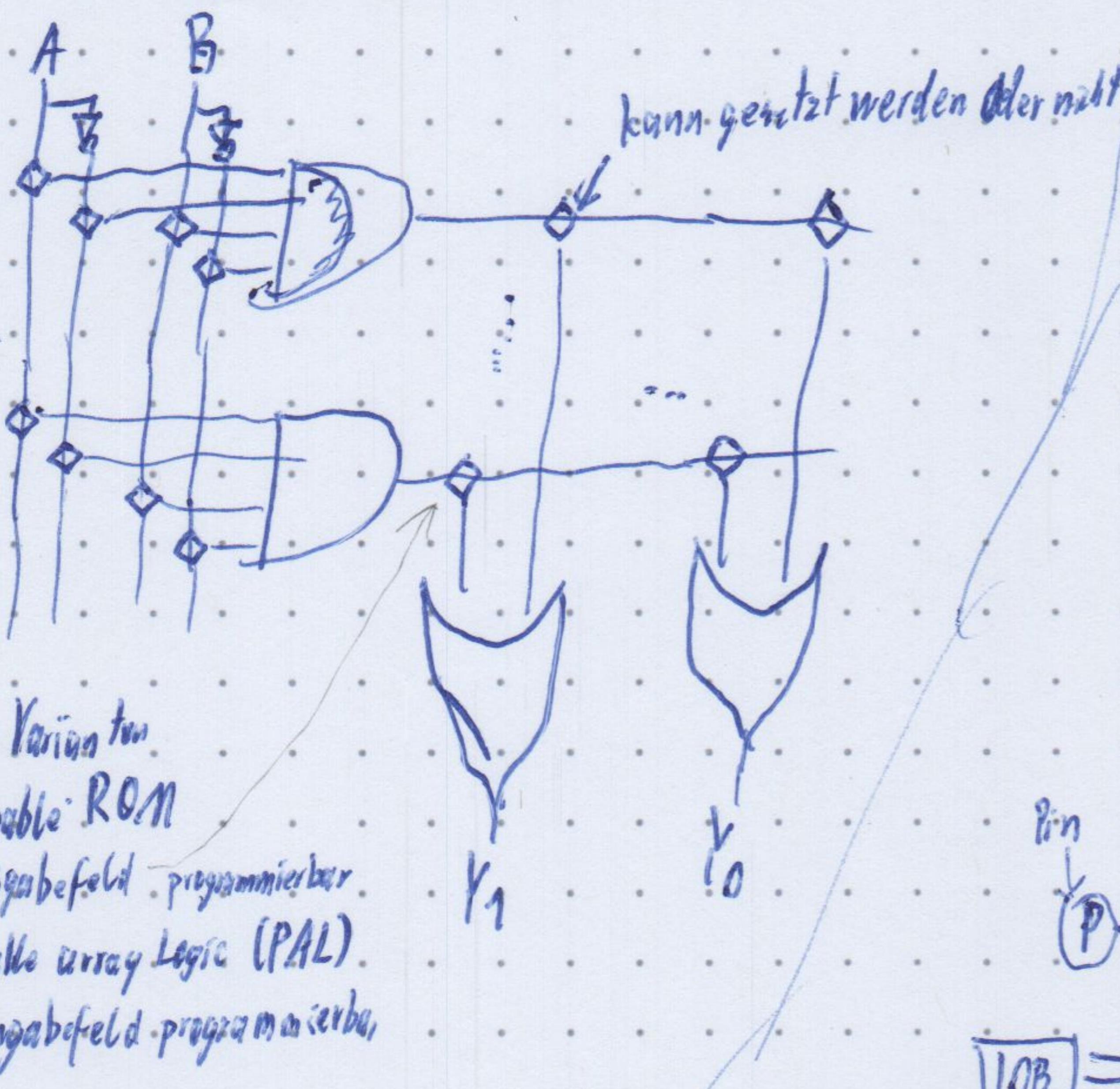
Entladen: Source = offen
Control Gate = 0V
Drain = 12V

mitreißen

⇒ zerstört Oxidschicht auf Dauer

Programmierbares Logikfeld Programmable Logic Array (PLA)

einfache Logik in Sum-of-Products



Performance vs. Flexibilität

ASIC application specific integrated circuit

optimisierte Datenpfade

z.B. CMOS-Basisgatter durch optisch/chemische Prozesse auf Silizium-Wafer

Software-Prozessor

führt generische Instruktionen sequentiell aus
nur generische (Mikro-)Architektur in Hardware

FPGA Field Programmable Gate Array

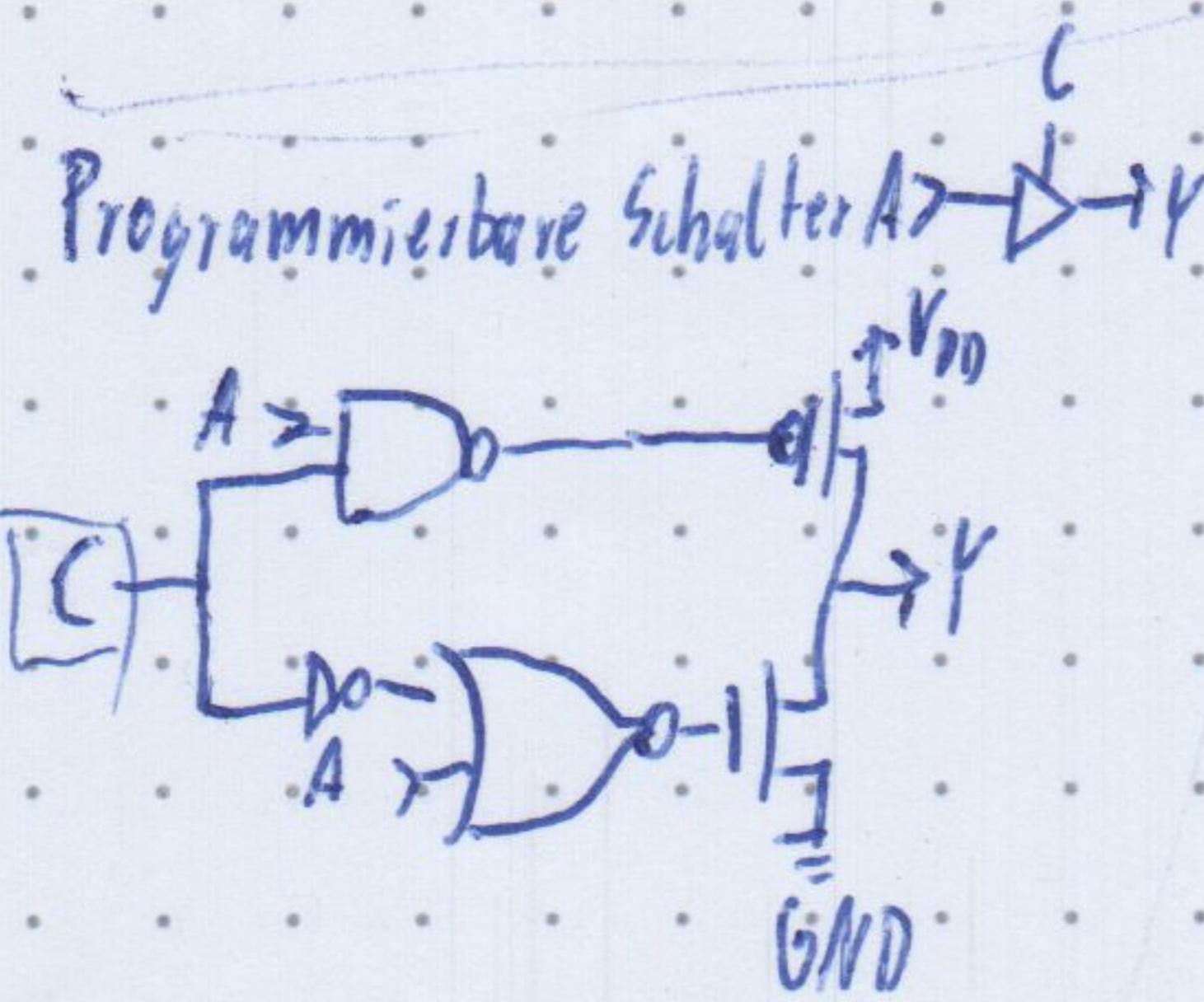
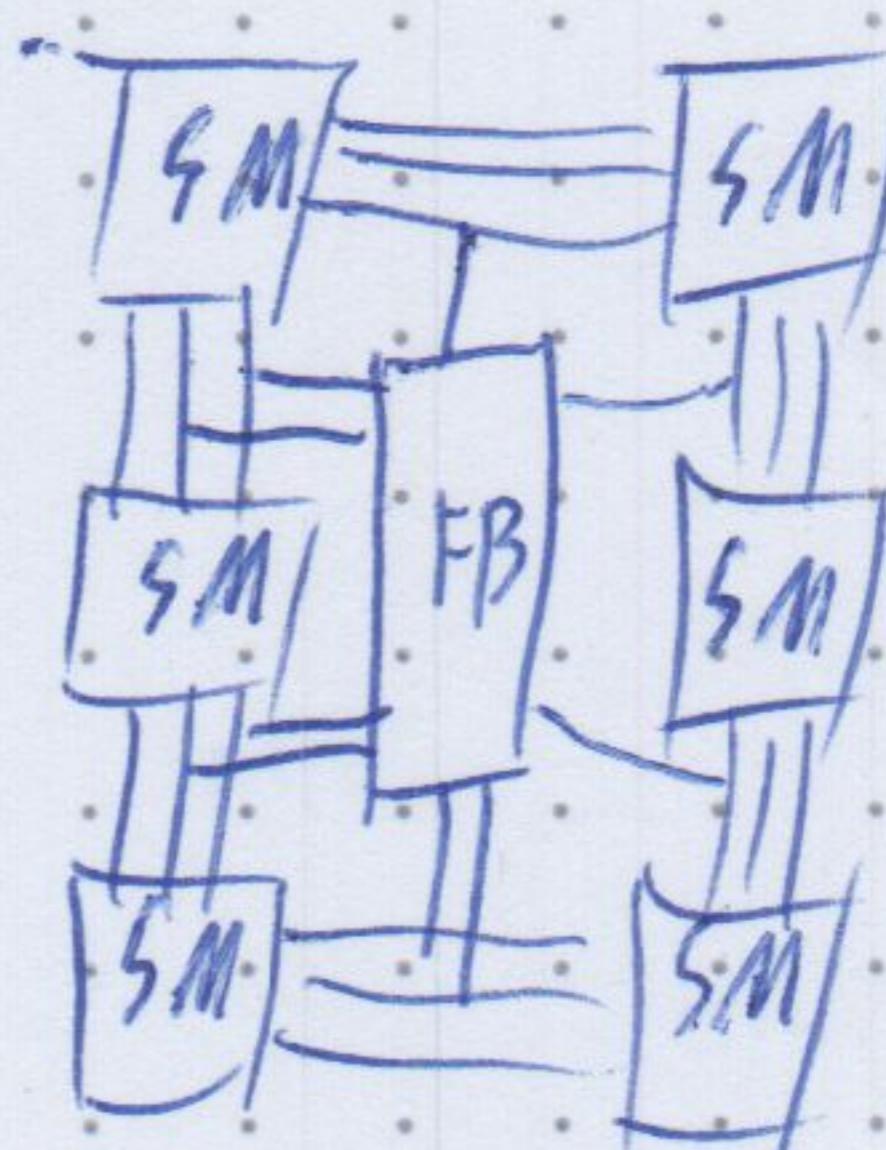
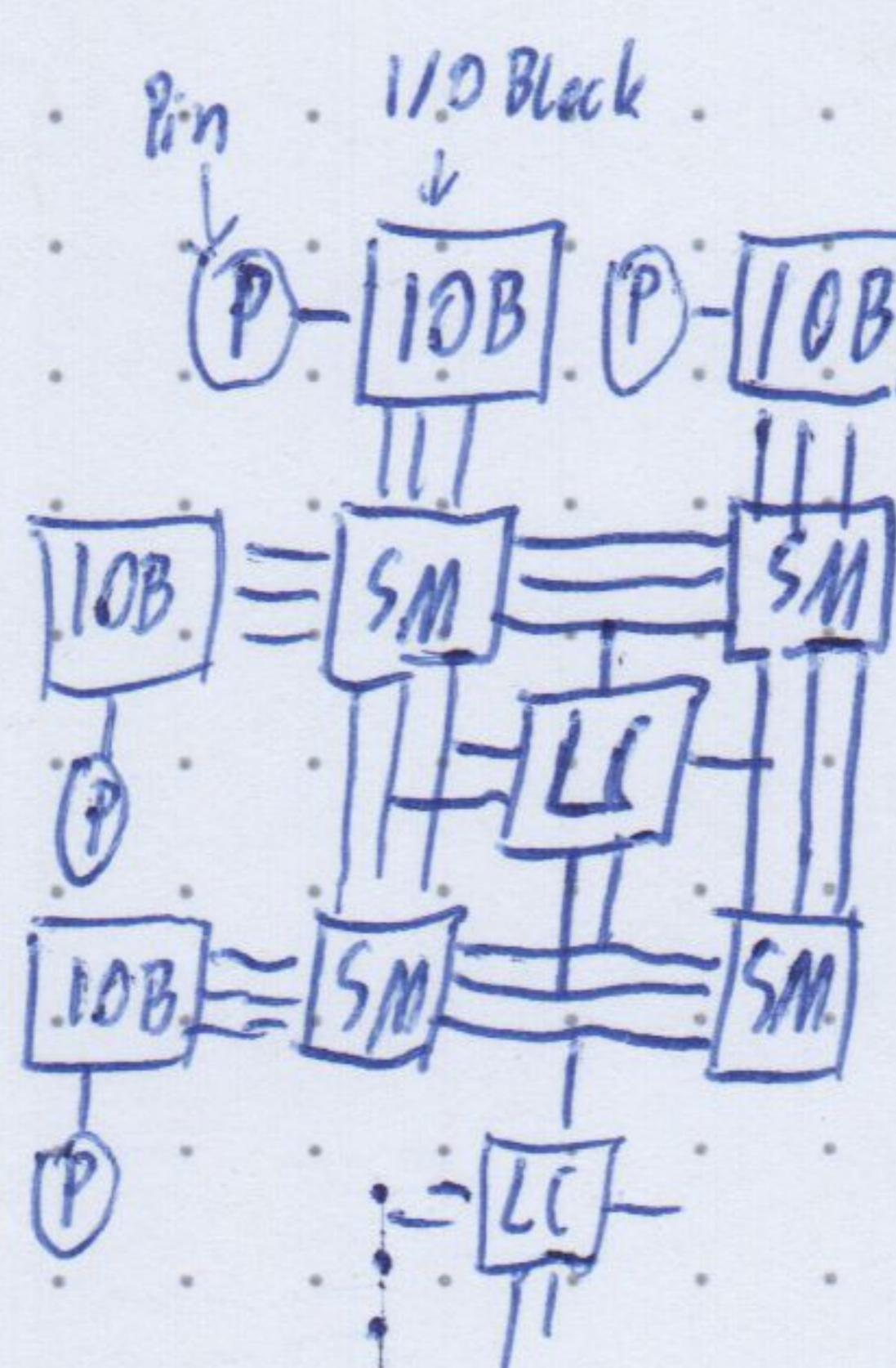
Kompromiss

Konfigurationspeicher (bitweise statt Wertweise)

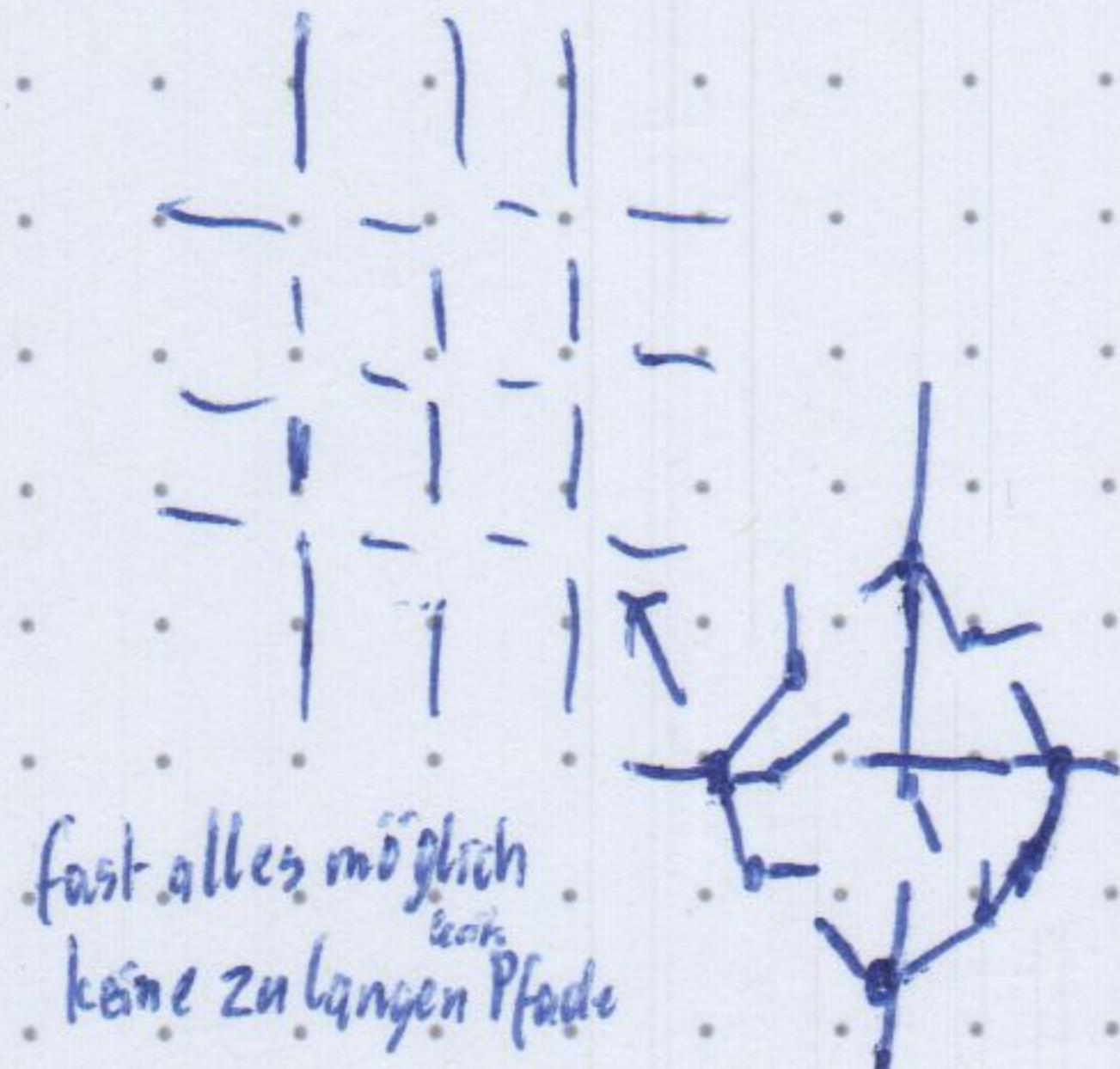
realisiert mit ROM oder SRAM benötigt permanent Strom (Pstatic)

Marktrelevante Hersteller:

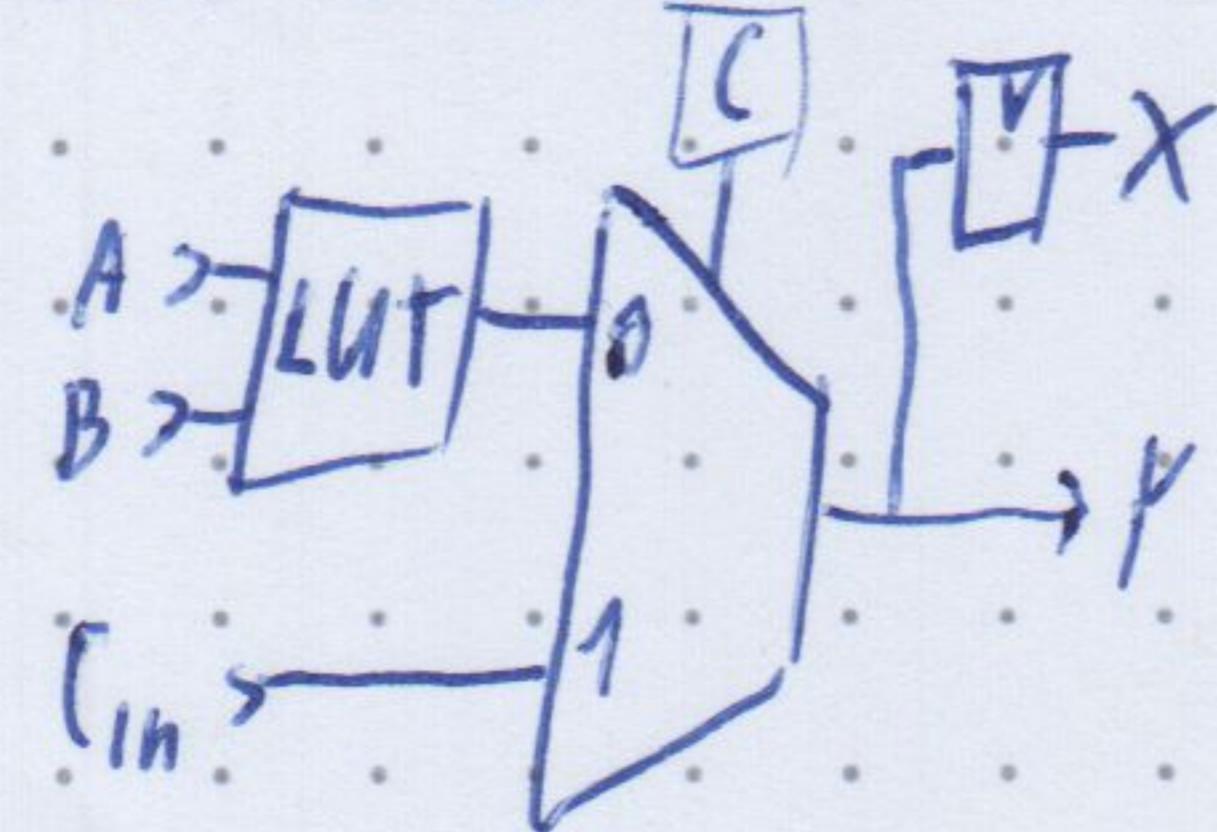
Xilinx: Zynq, Virtex, Kintex; Z-series, UltraScale+
Intel (Altera aufgekauft): Cyclone, Arria, Stratix
Microsemi: iML00, smartFusion, Polarfire, ProASIC
Lattice: iCE, Mach



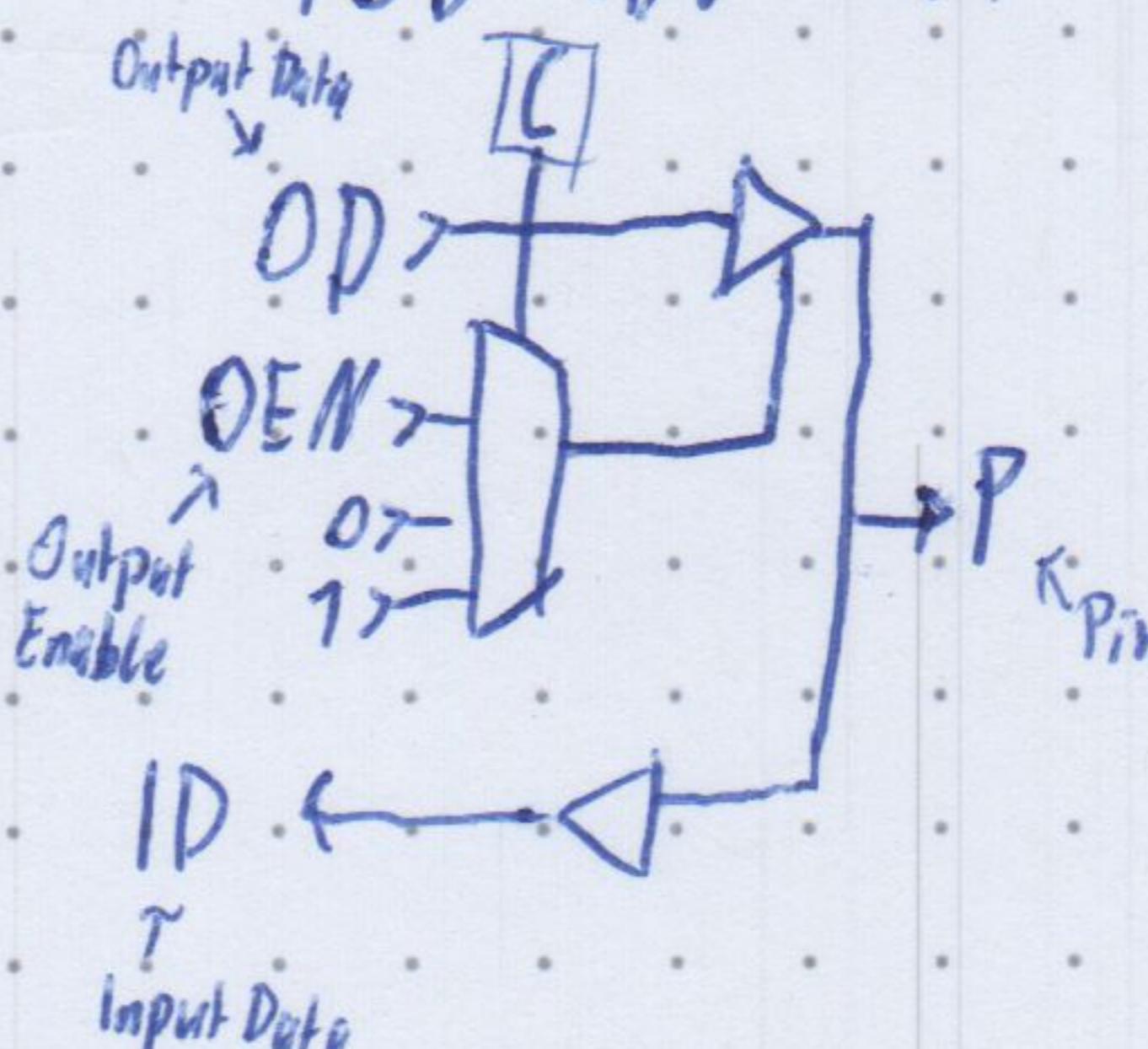
SM (Switch Matrix)



LC (Logic Cell)



I/OB (I/O-Block)



FB (Funktionsblöcke)

häufig verwendete Logikbausteine (begrenzte Ressourcen)

z.B. Block RAM (BRAM): kleiner SRAM

Digitale Signalverarbeitung (DSP): Multiplizierung, MAC

Phase-Locked Loop (PLL): Taktmultiplikation

Kommunikationssteuerer: USART, USB, Ethernet

kleine Prozessoren

FPGA Tool flow

