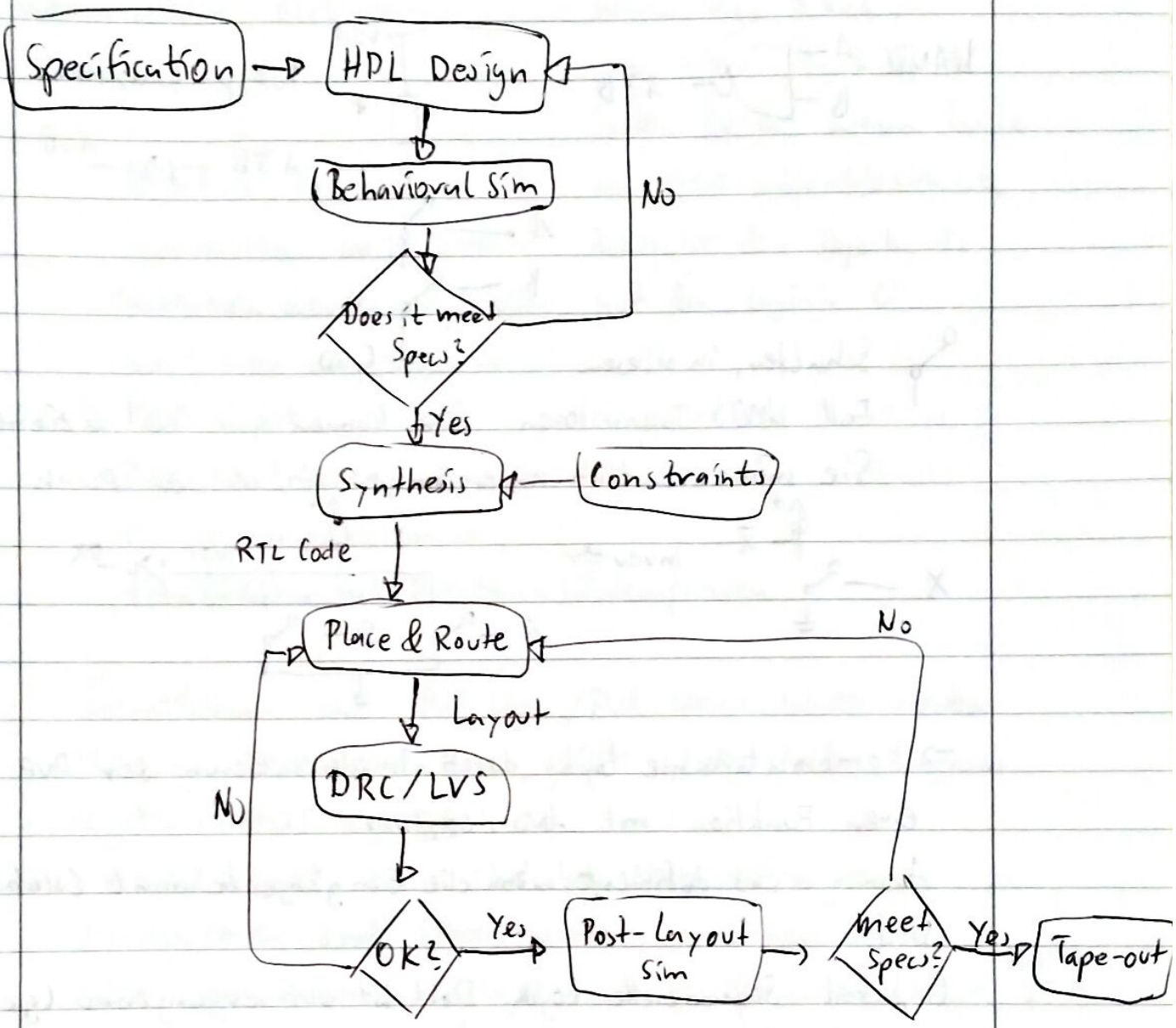
DDS

Digital Design Flow:



Warum binäre logik in der Digitaltechnik? Und nicht ternär?

Zahl N kodieren, mit X-stelligen Code

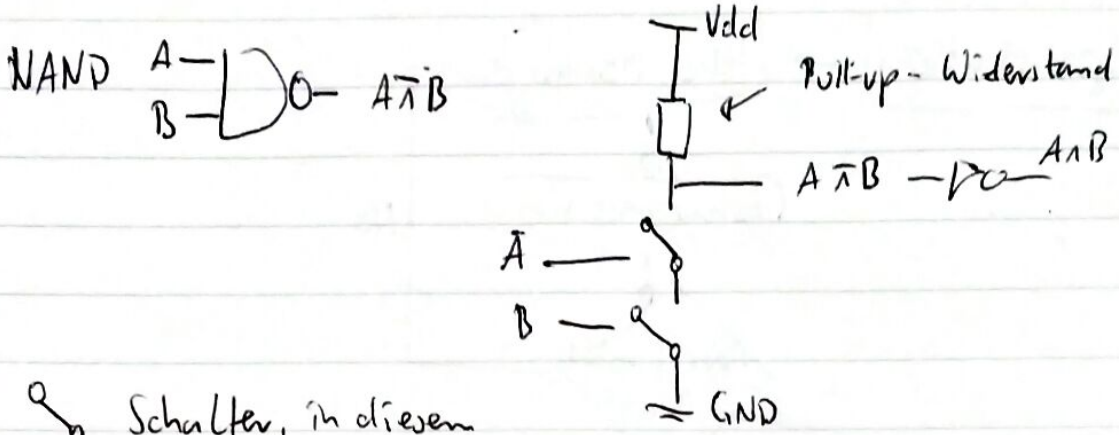
⇒ # bits: $\ln(N) / \ln(X)$ ..

Komplexität der Wahrheitstabellen / Operatoren: quadratisch

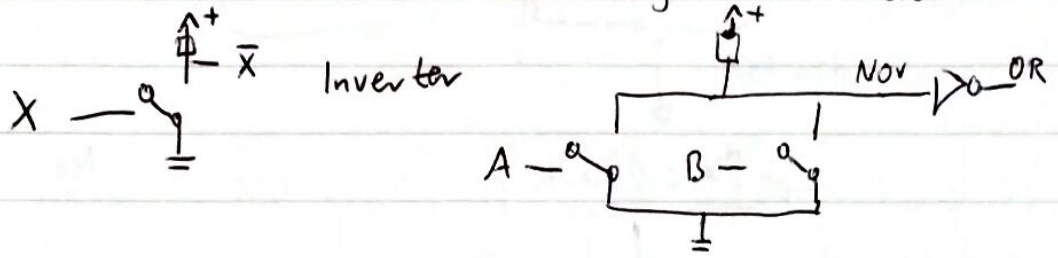
⇒ $C = X^2$

⇒ Komplexität der Schaltung: $(\ln(N) / \ln(X)) \cdot X^2$

Die Funktion hat ein Minimum in der Nähe von 2
⇒ binäre Kodierung am effizientesten

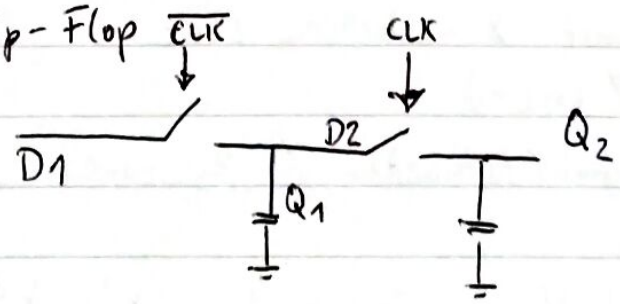


Schalter, in diesem Fall NMOS-Transistoren. Sie können gut "auf 0 ziehen"
Sie müssen deutlich niederohmiger sein, als der PU-W



⇒ Kombinatorische Logik durch Implementierung der DNF einer Funktion mit AND / OR / INV.
Ausgänge sind definiert, wenn die Eingänge bekannt (stabil) sind.
Gegenteil: Sequenzielle Logik. Dort ist ein vergangener (gespeicherter) Systemzustand ebenfalls Eingabe.
Diese Systeme brauchen: Takt / Reset / Speicher
Oft wird ein Zustandsautomat realisiert

Flip-Flop



2 DRAM Zellen Hintereinander

Die Schalter müssen genau in Gegenphase sein

CMOS

↳ PMOS & NMOS

	N	P
Dotierung:		
Ladungs-träger:	Löcher	Elektronen

Typinversion bei positiver Vgs

Wenn $V_{gs} > V_{th}$, auch wenn $V_g \leq V_d$

Leiten besser, wenn Source am GND angeschlossen ist

Schlecht für logisch 1, gut für logisch 0

Negative Vgs erzeugt Typinversion im N-Bereich

Elektronen werden abgestoßen und Löcher angezogen ⇒ PMOS leitet

PMOS leitet besser, wenn Source an VDD angeschlossen ist,

kombinierbar mit Pull-Down Widerständen

Schaltungen mit Pull-Up / Pull-Down Widerständen:

Statische Leistungsaufnahme, große Widerstände, langsam

⇒ PMOS+NMOS = CMOS

klein, schnell, kein statischer Stromverbrauch

Transistoren sind spannungsabhängige Widerstände

Strom am Drain = $I_{ds} = \mu \cdot C_{ox} \cdot \frac{W}{L} \cdot \left((V_{gs} - V_{th}) V_{ds} - \frac{V_{ds}^2}{2} \right)$

Für $V_{gs} > V_{ds}$ = $I_{ds} = \frac{1}{2} \mu \cdot C_{ox} \cdot \frac{W}{L} (V_{gs} - V_{th})^2$

Beste / Einfachste Parameter zum Dimensionieren: W, L

PMOS sollte etwa immer doppelt so groß sein, wie NMOS, da die Mobilität (μ) der Löcher etwa nur halb so groß ist, wie der der Elektronen

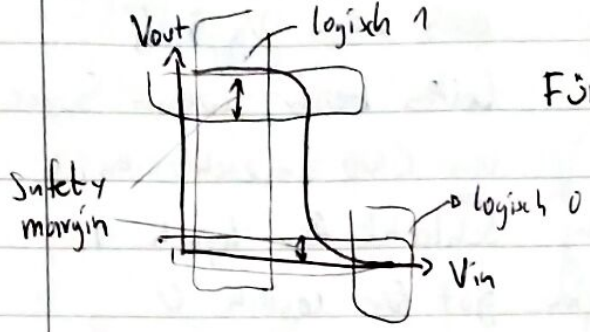
Inverter:

Im Bereich $V_{th} < V_{in} < V_{dd}$ leiten beide Transistoren

NMOS - Sättigung, wenn $V_{ds} > V_{gs} - V_{th}$

PMOS - Sättigung, wenn $|V_{ds}| > |V_{gs}| - |V_{th}|$

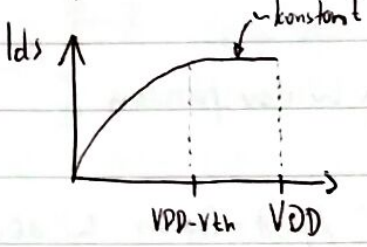
=> besonders steile Kennlinie



Für digitale Technik ist das in Ordnung

Geschwindigkeit eines Inverters

Im Ausgangsbereich zwischen VDD und VDD-Vth wird die Lastkapazität mit einem konstanten Strom ge-/entladen



Für $V_{out} < VDD - V_{th}$ hängt der Entladestrom von V_{ds} ab

$$\Rightarrow C \, dU/dt = -I_{ds}$$

$$\Rightarrow U(t) = 2 \cdot V_{gst} \cdot \frac{e^{-\frac{t}{\tau}}}{1+e^{\frac{-t}{\tau}}} \qquad \tau = C/K$$

Im Bereich $V_{ds} = 0$ verhält sich ein Transistor wie ein Widerstand mit $R_{on} = 1/K$

$$\Rightarrow U(t) = 2 \cdot V_{gst} \cdot \frac{e^{-\frac{t}{R_{on}C}}}{1+e^{\frac{-t}{R_{on}C}}}$$

sehr ähnlich zu:

$$U(t) = U(0) \, e^{\frac{-t}{R_{on}C}}$$

} => n Widerstand = Transistor (zu)

=> Geschwindigkeit des Inverters hängt von der Lastkapazität & von der W/L der Transistoren ab, sowie μ

Treiberstärke

Im Falle von großem C, bedarf es größeren Transistoren im Treibergatter

Alternative: Buffer (doppel-Inverter) mit großem W/L

Inv_2n meist ~~zwei~~ zwei parallelgeschaltete Inv_1, Inv_4 sind 2 Inv_2, ...

$$T_{gesamt} = \sum T_i = \alpha \cdot \frac{W(i+1) \cdot L(i+1)}{\frac{Wi}{Li}} = \alpha \cdot W(i+1)/Wi$$

$$\Rightarrow \prod T_i = \alpha^N \cdot \frac{WN}{WO}$$

Verfahren der Lagrange-Multiplikatoren

$$\Rightarrow T_i = T = \alpha \cdot K^{1/N}, \quad K = \frac{WN}{WO}$$

$$N = \ln(K)$$

$$\Rightarrow T = \alpha \cdot e \Rightarrow \frac{W(i+1)}{Wi} = e$$

$WN/WO = 1100$ — WN: Treiberlast

$N = \ln(1100) = 7$ — WO: start-Treiberstärke

$\Rightarrow$ 1x, 2.7x, 7,9x, 20x, 55x, 148x, 402x

N = # Buffer dazwischen. 1 Signal an 1100 FFs $\Rightarrow$ 7 Buffer nötig.

CMOS

Jede 0-Zeile wird als Serienschaltung von NMOS-Transistoren realisiert. Diese sind parallel an den Ausgang zu schließen

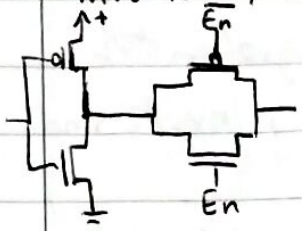
PMOS funktioniert analog dazu, allerdings sind die 1-Zeilen zu realisieren

Es folgt eine Reihe von Optimierungen

NAND & NOR skalieren gut mit der Anzahl ihrer Eingänge

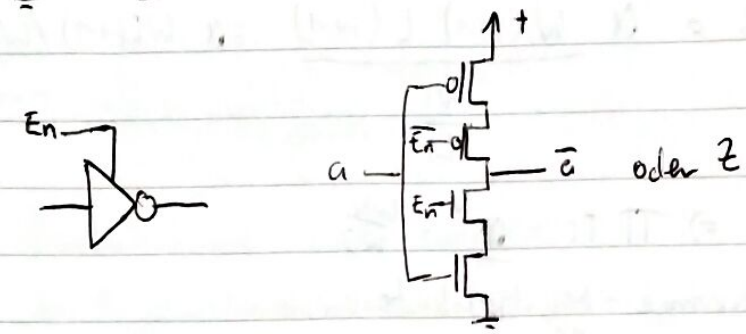
Hoch-Ω-Zustand (Z, tri-state)

Inverter / Buffer mit enable-signal

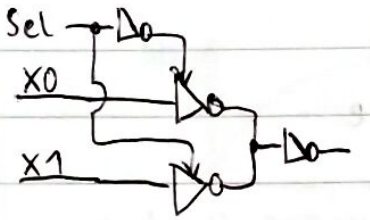


"Gated Inverter" ermöglicht das Kurzschließen von mehreren Gate-Ausgängen

Alternatives Schaltbild:



MUX mit Gated-Invertern:



12 Transistoren

256:1 MUX herkömmlich: ~ 6100 Transistoren

mit Gated-Invertern und Baumstruktur von 2:1 MUX: ~1000 T.

aber langsamer wegen höherer Stufenanzahl

Dekoder / DeMUX: ähnliche Vor- und Nachteile

Speicher mit Kondensator entlädt sich über die Zeit

=> Information kann nicht beliebig lange ohne Auffrischen gehalten werden (DRAM)

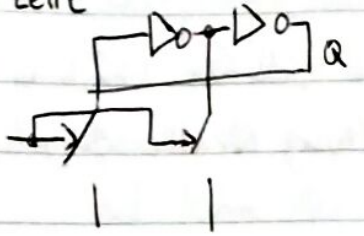
SRAM: 2 Inverter in Rückkopplung => Information so lange wie Versorgungsspannung erhalten

$V_{out} / V_{in} = 0$ => logische 0

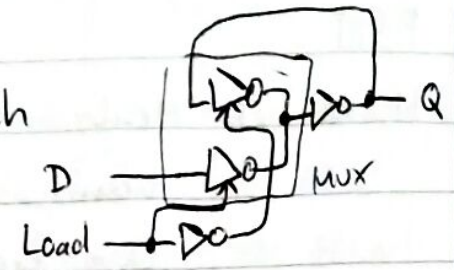
$V_{out} / V_{in} = VDD$ => logische 1

$V_{out} / V_{in} = \frac{VDD}{2}$ => undefiniert

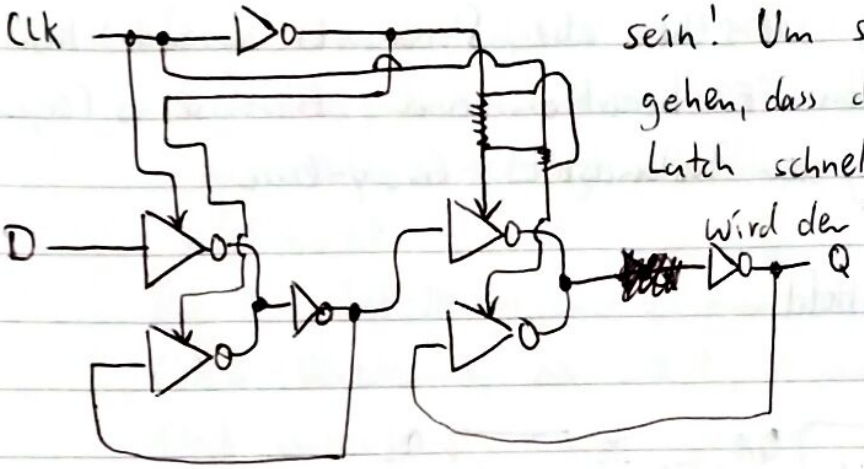
SRAM Zelle



Latch



Flip-Flop

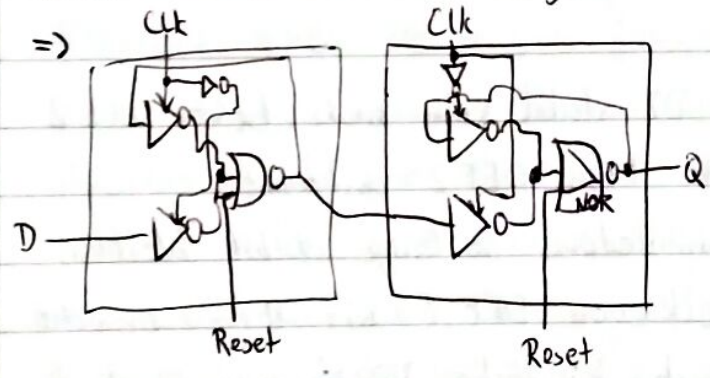


Wichtig: niemals sollen beide Latches transparent sein! Um sicher zu gehen, dass das erste Latch schneller schaltet, wird der Taktinverter oft im FF realisiert

Noch besser: auf für das 2. Latch!

Zur Vermeidung eines unbekannten Startzustandes nutzt man ein initiales Reset-Signal

=>



Das NOR-Gate sorgt für eine 0 im Latch, egal, welcher Zustand zuvor eingenommen wurde, insofern Reset = 1 gilt

Da immer mindestens 1 Latch im Speicherzustand (sel = 1) ist, kann immer resettet werden. Der asynchrone Reset ist stärker.

$$Slack = (clk1 + Delay) - (clk2 + Thold) > 0 \quad \text{ist gut!}$$

Für Holdzeit (nächste Seite)

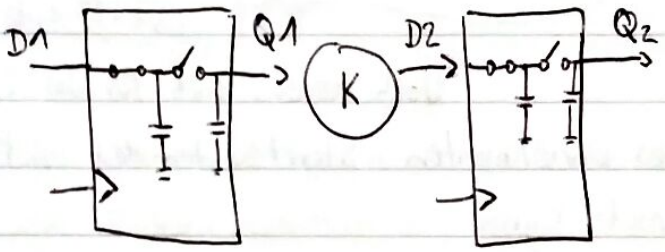
FSM

Moore vs. Mealy

Weniger Zustände, dafür kombinatorische (komplexere) Schaltung für die Ausgabe

Full synchron: FFs als Speicherelemente
mindestens 1 asynchrones Eingangssignal (außer reset): Latches
In HDL: Hierarchie des Kontrollfluss einfach durch Reihenfolge der if-then-else Statements auszudrücken.
Im Falle von FFs bedarf es keiner otherwise => (eigen Kante)
Kodierung der Zustandsbits: Gray-Code

Setup & Hold



Haltzeit: Zeit, die D2 stabil sein muss, bevor clk 2 am Latch 1 in DFF2 ankommt.
=> K muss mindestens so lange stabil bleiben, wie die clk von DFF1 bis DFF2 braucht.
Notfalls durch künstliche Verzögerung in K herbeiführen.
Bei Hold-time-violation wird ein Ergebnis, das eigentlich erst im Folgetakt sichtbar sein sollte, direkt in DDF2 gespeichert. Dabei wird das zuletzt gespeicherte Ergebnis überschrieben.

Setupzeit ≙ Zeitpunkt vor der aktiven Flanke, zu dem sich D2 geändert haben muss Latch 1 in DDF2 verlässt ansonsten den transparenten Modus und kann das Ergebnis von K sonst nicht mehr speichern

$Slack = (clk2_{(i+1)} - T_{setup}) - (clk1_{(i+1)} - T_{clk}) - Delay > 0$

Die Logik rechnet zu lange, sie braucht zu lange, um ihr Ergebnis an DFF2 zu geben.

⇒ Taktfrequenz mindern, was im Gegensatz zur Hold-Verletzung noch zur Laufzeit möglich ist.

Bei Hold-Zeit-Design erzeugt man eine künstliche clk 2 - Verzögerung zur clk 1, so genannte Clock-uncertainty (overconstraint)

Hold-Slack > $T_{unc.}$

Kodierer

Abbildung einer Menge von Eingabebelegungen auf eine Menge von Code-Wörtern

Tastatur: 'A' ↦ 00101101

'b' ↦ 00001101 ...

Was, wenn mehrere Tasten gleichzeitig gedrückt werden?

⇒ Bedarf an Prioritätsnetzwerk, das immer nur höchstens ein aktives Signal durchlässt

Priorisierung wie folgt (beispielhaft):

```
AP7 = A7
AP6 = !A7 && A6
AP5 = !A7 && !A6 && A5
  :
```

Kaskadierung von Prioritätskodierern

ermöglicht durch Prio_1 und prio_0 -Signale

Prio_0 ist die OR-Function von Prio_1 und allen Eingängen

Die Ausgänge sind nur dann aktiv, wenn Prio_1 = 0 ist.

Volladdierer

C_{out} $AB \vee BC_{in} \vee AC_{in}$

$= AB \vee ((A \vee B) C_{in})$

C_{in} \ A	0	0	1	0
0	0	1	1	1
1				

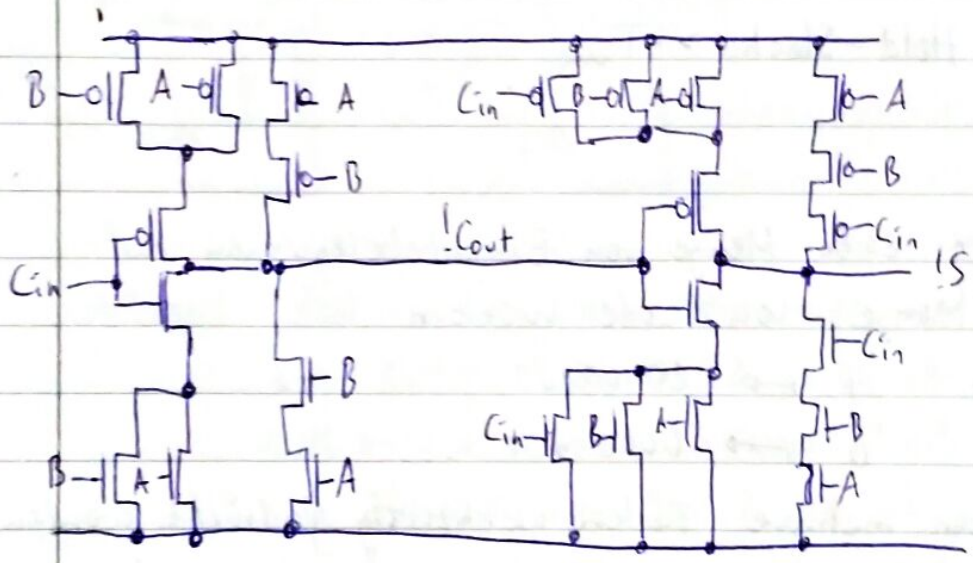
B

$S = A \dot{\vee} B \dot{\vee} C$

$= ABC_{in} \vee ((A \vee B \vee C) \wedge \overline{C_{out}})$

C_{in} \ A	0	1	0	1
1	1	0	1	0
0				

B



22 Transistoren

Schieberegister können schnell Haldzeitverletzungen hervorrufen

Werden oft genutzt für Verzögerung von Signalen, Zustandskodierung, Zähler

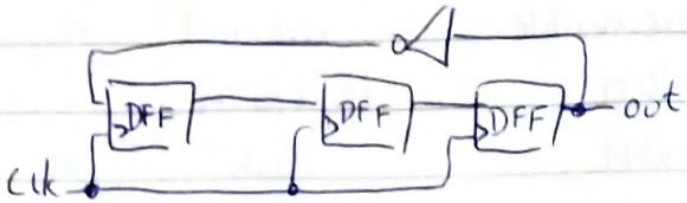
Zähler

LFSR (Linear feedback shift register)

Johnson-Zähler

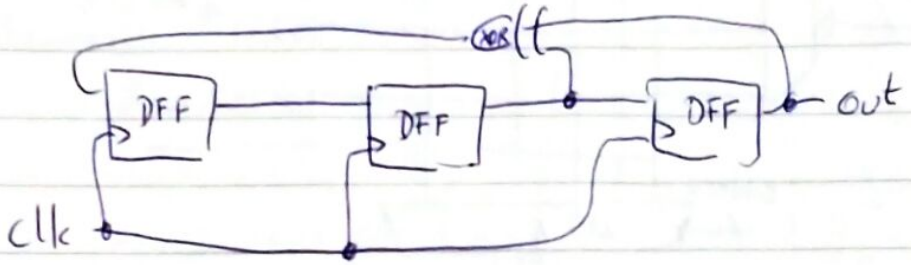
2N Zustände

010 → 101



PRBS (pseudo random bit sequence)

2N-1 Zustände (je nach Größe manchmal weniger)



000

Scrambler

Nutzen LFSRs oder fixe Tabellen

Kann zur Kodierung genutzt werden, der Empfänger muss die korrekte Phasenlage kennen

Vorteil: Fehler werden nicht multipliziert

Nachteil: „Schlüsseltausch" nötig

Selbstsynchronisierende (multiplikative) Scrambler benötigen keine Information über die Phasenlage. Zustand im LFSR darf beliebig sein

Nutzdatenfolge wirkt direkt auf das LFSR ein

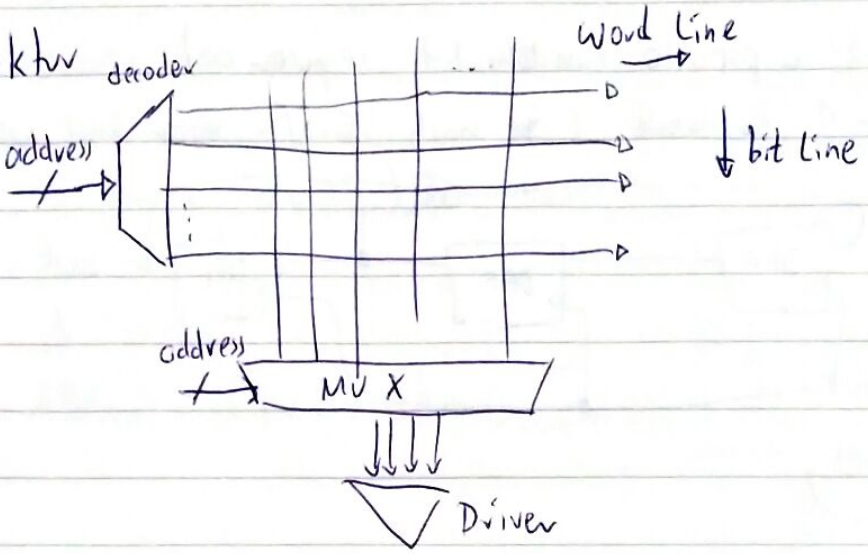
Übertragungsfehler pflanzen sich fort

Bestimmte Nutzdatenfolge kann das LFSR nullen

Speicher

Festwertspeicher		Schreib/Lese speicher
irreversible	reversibel	SRAM (DFFs)
ROM	(E)EPROM	DRAM (C)
PROM	Flash	

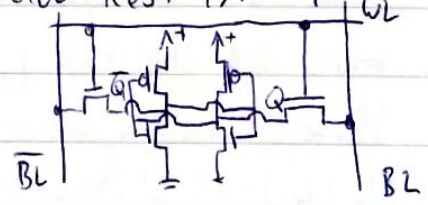
Struktur



ROM

Schottky-Dioden werden bei der Herstellung durchgebrannt
=> 0en werden geschrieben, der Rest ist 1

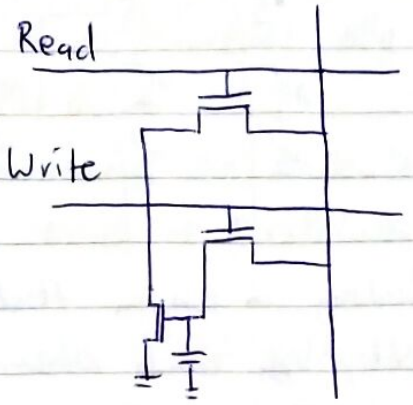
SRAM



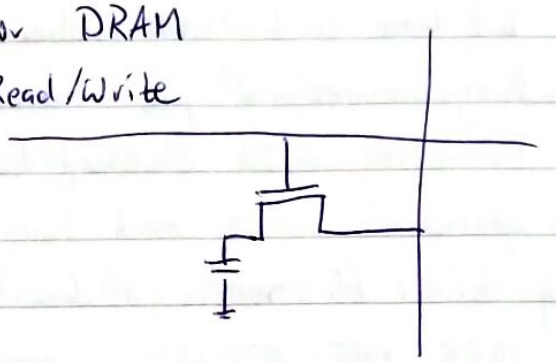
2 Inverter rückgekoppelt
Benötigt Leseverstärker
Lesen: 1. BL und $\overline{BL}$ auf $\frac{VDD}{2}$ aufladen
2. WL auf VDD setzen
Schreiben: 1. BL und $\overline{BL}$ auf den zu schreibenden Wert aufladen
2. WL auf VDD setzen
Hierbei muss gewährleistet sein, dass BL und $\overline{BL}$ genügen Stärke haben, um den zustand zwischen den Invertern eventuell zu flippen.

DRAM

2 - Transistor DRAM

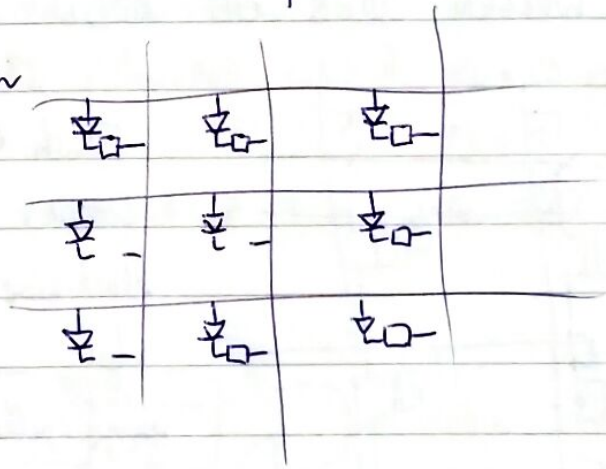


1 Transistor DRAM

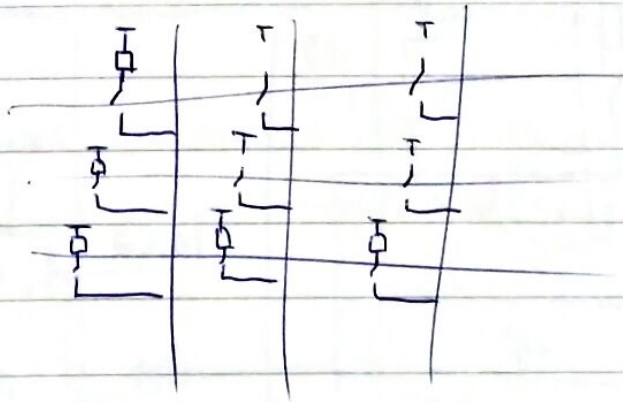


Permanentspeicher

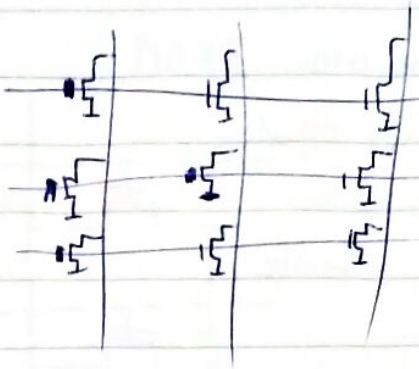
Variante mit Dioden



Schalter



Transistoren mit verschiedenem V_{th}

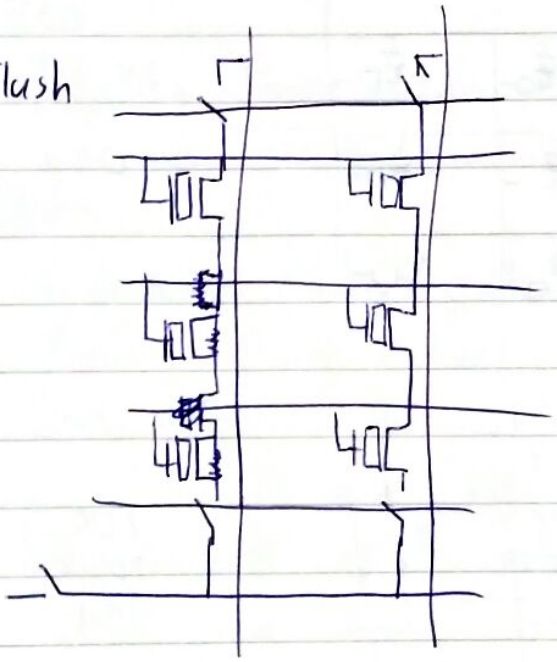


Dies wird durch eine Ladung in einem floating-gate erzeugt. Diese verschiebt V_{th} nach oben, sodass VDD nicht reicht, um den Transistor zu schalten. Dieser würde die bit lane auf GND ziehen.

Programmierung per Fowler-Nordheim Tunneling oder channel hot carrier injection.

FNT: Hohe Spannung bringt Elektronen in das Floating Gate. OB SIE WOLLEN, ODER NICHT XDDb!

Flash



Flash C EEPROM

NAND	NOR
block-wise	byte-wise
slow	fast
many cycles	few cycles

Phase - Locked - Loop (PLL)

zur Taktsynchronisation

Quarz liefert Grundtakt, alle anderen synchronisieren sich per PLL daran

Leitungsminimierung zur Datenübertragung per

- shift → serial → parallelisierer
- multiplexing

Dabei muss ein Takt mitgeliefert werden, damit der Empfänger die Daten korrekt empfangen kann

⇒ Synchronisation

Serialisierer und Parallelisierer werden mit Registern (Schieberegister) realisiert

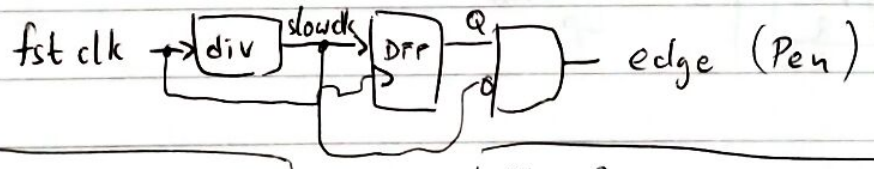
En-.Signal kann entscheiden, ob die Information des vorherigen Flip-Flops (en = 1) oder von sich selber übernommen wird (en = 0)

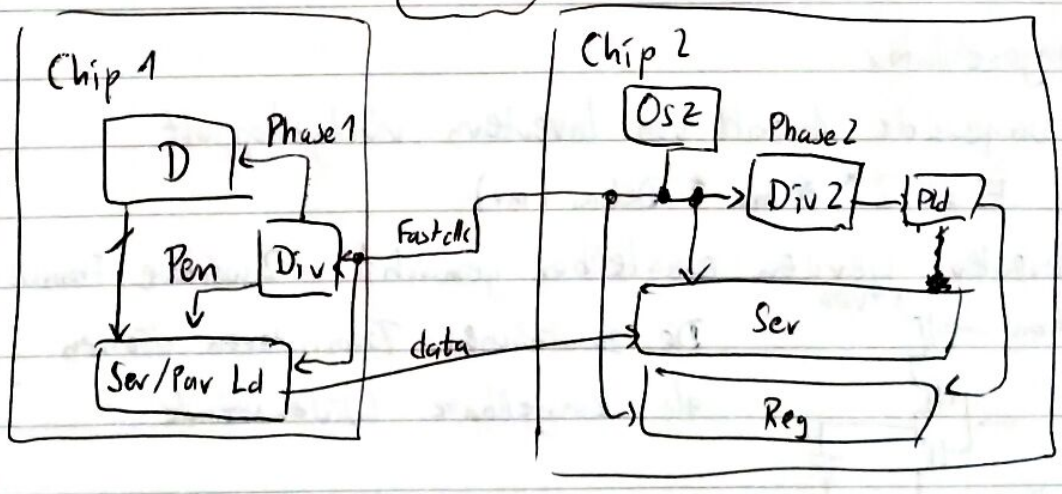
Ähnlich für PL (Parallel Load) PL = 0 ⇒ weiter shiften

PL = 1 ⇒ Parallel load

Clock divider: realisiert mit TFFs: Jedes TFF hat $\frac{clk}{2}$ auf Q

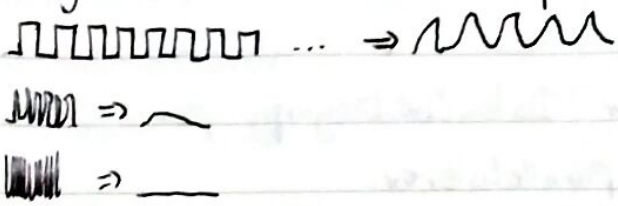
Flankendetektor:





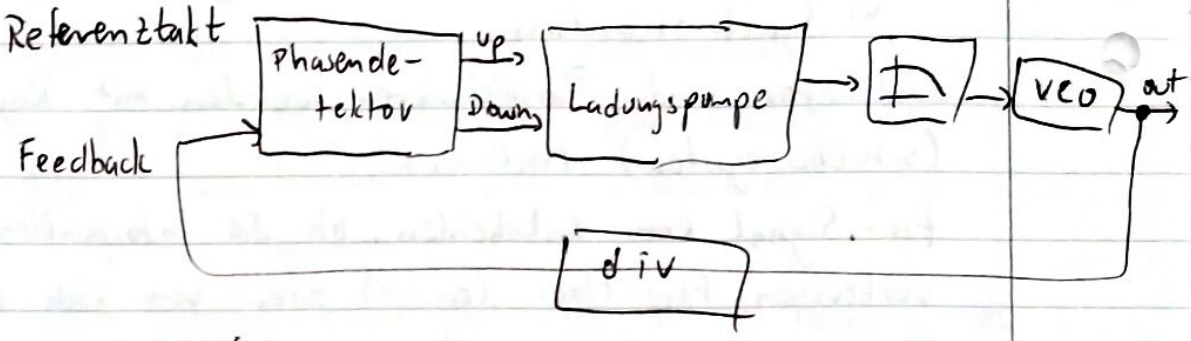
Clk dividers müssen beide Synchronisiert sein, da sie womöglich verschiedene Startzustände einnahmen ⇒ Phasenverschiebung

Schnelles Taktsignal ist ungünstig zum Senden über eine Leitung. Diese wirkt als Tiefpass.



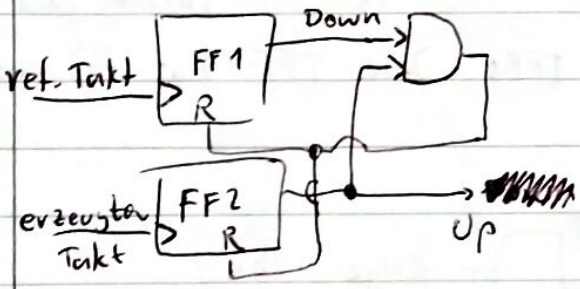
Daher wird lieber die slow-clk gesendet. Auf beiden Chips befindet sich ein PLL & Multiplizierer.

PLL:



⇒Phasenregelschleife

Phasendetektor:

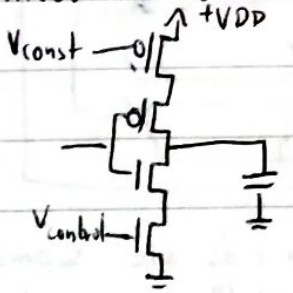


Ringoszillator

ungerade Anzahl von Invertern, rückgekoppelt

F = 2 * * Inv * Delay (inv)

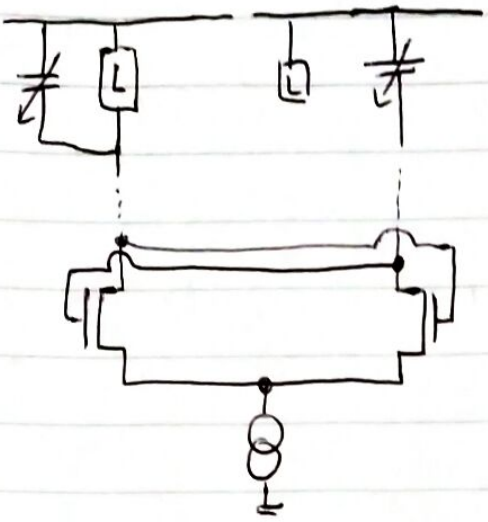
Inverter werden einstellbar gemacht: 2 weitere Transistoren:



Die zusätzlichen Transistoren dienen als einstellbare Widerstände

Oszillator mit weniger Jitter:

LC Oszillator



Ladungspumpe