6 NORMAL FORMEN ALLG.

DNF
Disjunktiv

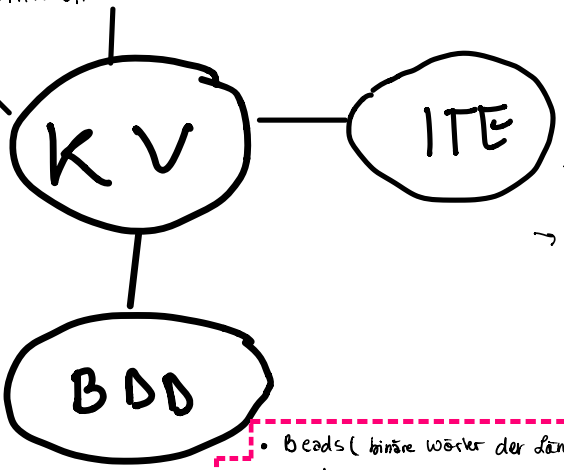
1er Blöcke
Var direkt auslesen
Konj. d. disj. verknüpft sind

KNF
Konjunktiv

0er Blöcke
Var invertiert
Disj. d. Konj. verknüpft sind

- Irrelevante Vars werden eliminiert
- Durch Umformungen möglich
  ↳ Res. not always minimal
- Minimierungsverfahren
  → Karnaugh - Veitch - Diagramm
- Für k o. weniger Vars sinnroll

If a then
  if b then c
  else d



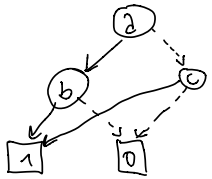
$a \wedge b \equiv ITE(a,b,0)$
$\neg a \equiv ITE(a,0,1)$
$a \vee b \equiv ITE(a,1,b)$

→ Umwandlung von Dis./Konj. NF in ITE
↳ Basiert auf Shannon Zerlegung

Reduction

- Umwandlung eines Decision Trees in min. DAG
- Anwenden von 2 Regeln
  - Delete Rule – Löschen unnötiger Zwischenk.
  - Merge Rule – Zusammenfassen von identen Teilgraph

Ergebnis: BDD

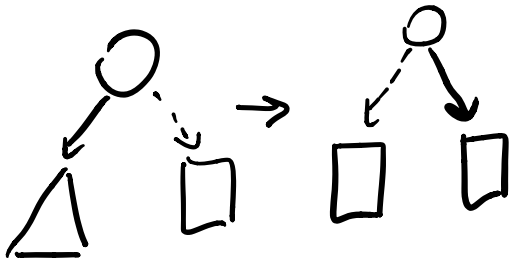


Beads

- Beads (binäre Wörter der Länge 2^n)
  - Form ww
    (w ein Wort der Länge 2^{n-1})
  - ≠ Squares
- Variablenreihenfolge
  - Für n vars n! Permutations
    ↳ Min. Form schwer zu finden
- Größe
  - gibt Boolesche Fkt, sodass unabhängig von d. Variablenreihenfolge alle BDDs exponentielle Größe haben
  - meisten relevanten Fkt. klein
  - Mult.: schwer zu vereinfachen

- Eigenschaften
  - ∀ Beads ∈ Wahrheitstabelle & nur diese
  - Darst. einer bel. Boole'schen Fkt ist gegebener Variablenreihenfolge ISOMORPH
  - 2 isomorphe BDDs beschreiben äquivalente Boole'sche Fkt.
  - Typ. Fkt kann man mittels BDD's realisieren
  - Ops. können direkt auf kompakten Darstellung durchgeführt werden

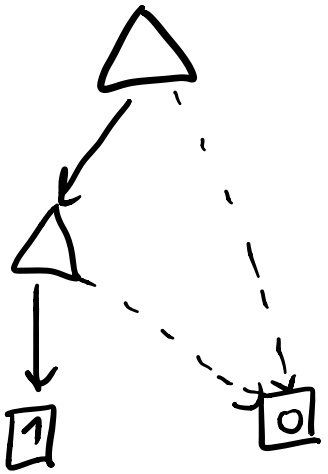
6| NEG.



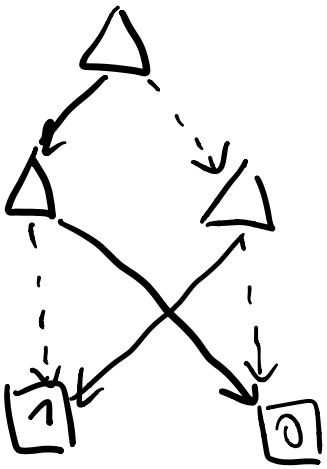
BDD

Fkts

UND



XOR



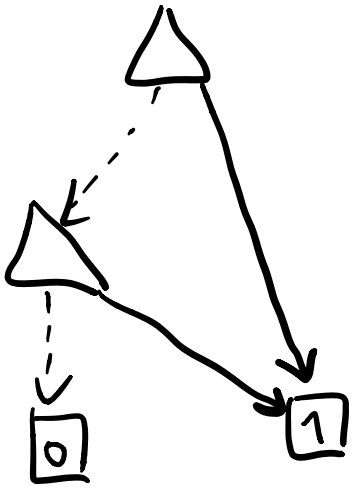
Greedy:

X so schnell als möglich durch 0 o. 1 ersetzen

Geduldig:

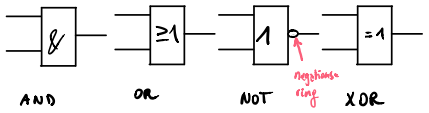
X so spät als möglich durch 0 o. 1 ersetzen

ODER



7 | Gatter

= Elementare Bausteine
- Impl. durch elekt. Bausteine
- Realisierung d. Booleschen Algebra:
  - Grundoperationen:



Mehrere Eingänge möglich

Basic Info

= Abbildung von Boolesch. Algebra auf physikalische Systeme
- Hier: elektrische Systeme
=> Egal welches System, weil:
  - möchten sammenhänge Werkbereich auf Booleschen Werk abbilden
  - → Variable → Signal
  - Operator → Gatter

(Verknüpfung d. phy. Signale entspr. d. Boole'schen Akt durch Gatter)

Diskret. von kontinu. Werkmenge (Kontinuum)
→ Abbild. von 0/1 auf werkbereich physikalischen größe

0/1 können beliebig zu H/L zugeord. werden
+/- Logik

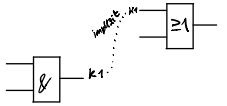
Abb. analoger Werte auf Boole. Zustände

NRZ – elekt. Signale
- 0/1 in physikalischen System beliebig zuordnungsbar
- HIGH ↑ Spannungspegel interrel
- LOW ↓ Spannungspegel interrel
- Deutliche Abstände zw. gült. Bereichen, um Störung auszugleichen

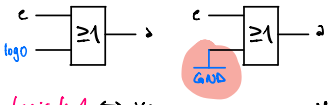
VERBOTENE ZONE

Schaltplan

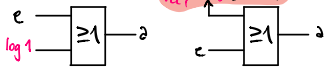
Namensgleiche Signale sind impl. verbunden
↳ Gilt auch für Versorgungsanschlüsse
(VCC/VDD: + Vorsorgespannung, GND: Masse/0V werden d. Übersichtlichkeit weggelassen)



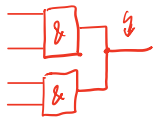
Fixe log. Zustd können auch durch Spannungspegel definiert werden
- logisch 0 ↔ Masse, Ground, GND, 0V



- logisch 1 ↔ Versorgungsspannung, V_{CC}, V_{DD}



→ Eingänge bleiben NIE offen (immer def. Signal)



Kombinatorik

Gatter, Standardbausteine

Warum NAND & NOR?

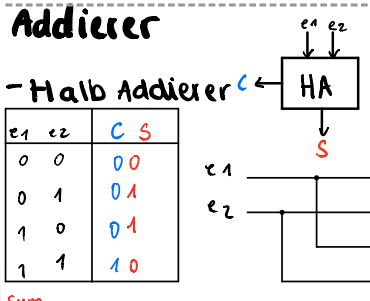
- Weniger Transistoren
  → weniger Chipfläche
  → weniger Kosten

Gatter	#Eingänge	Transistoren
NAND/NOR	2	4
XOR	2	5
AND	2	6
OR	2	6

Funktional vollständig: = Menge von Booleschen Transistoren, wenn damit alle anderen Boole'schen Operatoren ausgedrückt werden können
Substitution: Umform von Gatterfkt. mit geringeren Konstruktionsaufwand zu erreichen } Häufig NAND/NOR

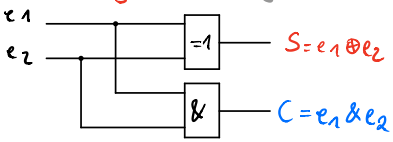
Addierer

Halb Addierer



Kein vorgehender Übertrag berücksichtigt

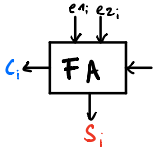
e_1	e_2	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Sum

Carry

Full Adder



Vorgehender Übertrag berücksichtigt

e_{1i}	e_{2i}	C_{i-1}	C_i	S_i
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

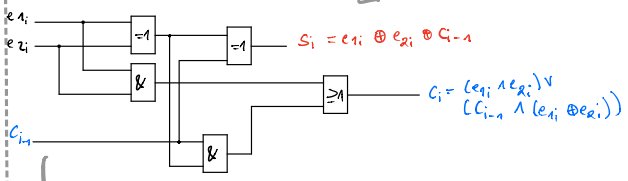
DNF:

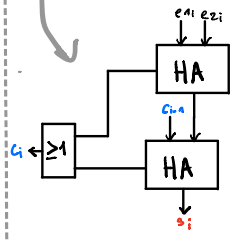
$S_i = (\overline{e_{1i}} \wedge \overline{e_{2i}} \wedge C_{i-1}) \vee (\overline{e_{1i}} \wedge e_{2i} \wedge \overline{C_{i-1}}) \vee (e_{1i} \wedge \overline{e_{2i}} \wedge \overline{C_{i-1}}) \vee (e_{1i} \wedge e_{2i} \wedge C_{i-1})$

KNF:

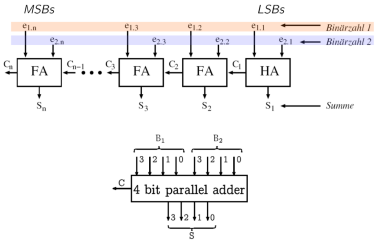
$C_i = (\overline{e_{1i}} \wedge e_{2i} \wedge C_{i-1}) \vee (e_{1i} \wedge \overline{e_{2i}} \wedge C_{i-1}) \vee (e_{1i} \wedge e_{2i} \wedge \overline{C_{i-1}}) \vee (e_{1i} \wedge e_{2i} \wedge C_{i-1})$

vereinfachen





Parallel addierer



Codierer vs Decodierer

Eingabelänge | Ausgabelänge

Codierer

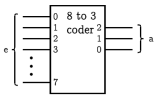
- Dienen d. Umwandlung von Codes
- = Encoder
- → verdichtet Infor.
- $n \geq m$
- n zu m Encoder
- Umwandlung in dichteren / dichten Code
- Opt.: zusätzl. Ausgänge zum Anzeigen d. Gültigkeit d. Works

Decodierer

- $m \geq n$
- n zu m Decoder
- Umwandlung in redundanten / redundanteren Code
- Opt.: zusätzliche Eingänge zum Aktivieren des Decoders

Encoder

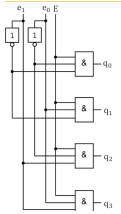
- Ein „1 aus 2^n" Code wird zu einer n-Bit Binärzahl (Binär-Encoder)
- Bitfolge am Eingang → Binärzahl am Ausgang



e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0	a_2	a_1	a_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

- Ausgangscodierung benötigt weniger Bits als Eingangscodierung

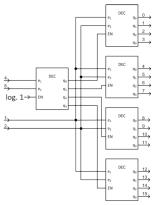
Decoder



- E . . . Enable-Eingang
- UND-Gatter verarbeiten Eingangskombinationen
- Genau ein Ausgang wird auf log. 1 geschaltet – Auswahl mittels Eingangsbits
- Ausgangscodierung benötigt mehr Bits als Eingangscodierung

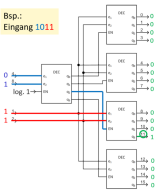
e_1	e_0	E	q_0	q_1	q_2	q_3
X	X	0	0	0	0	0
0	0	1	1	0	0	0
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	1

Kaskadierung



- Aufgebaut aus 5 Stk. (2 zu 4)-Decodern
- Enable-Eingang aktiviert die verschiedenen Decoder

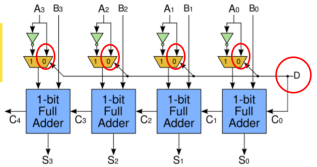
Bsp.:
Eingang 1011



- Aufgebaut aus 5 Stk. (2 zu 4)-Decodern
- Enable-Eingang aktiviert die verschiedenen Decoder

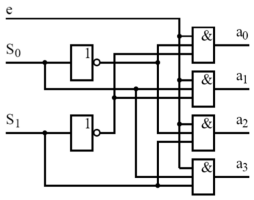
- Subtraktion
  - Einerkomplement (ungebräuchlich)
  - Zweierkomplement
    - Summand invertieren (= bitweise Negation) und Carry setzen (= 1 addieren)
  - Overflow-Erkennung (darstellbarer Zahlenbereich verlassen)

Subtraktion wird auf Addition zurückgeführt



Demultiplexer

- Für Digitalsignale äquivalent zu Decoder mit Enable-Eingang

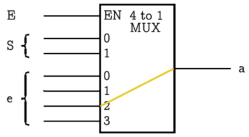


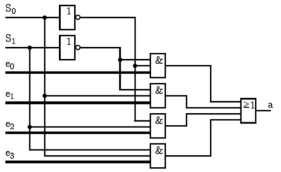
(1 zu 4)-Demultiplexer

Multiplexer

- Durchschalten eines gewählten Eingangs auf den Ausgang
- Steuersignal bestimmt welcher Eingang durchgeschaltet wird

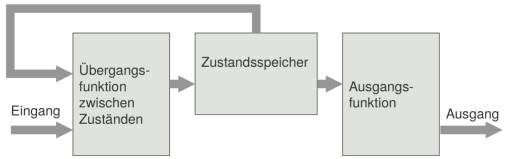
S_1	S_0	a
0	0	e_0
0	1	e_1
1	0	e_2
1	1	e_3



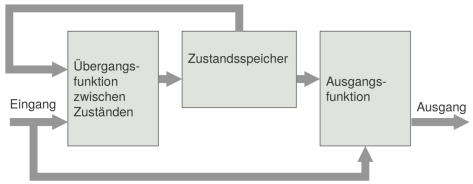


7 Schaltwerke

- Ausgangsfunktion hängt vom Zustand ab
  → Moore-Schaltwerk



- Ausgangsfunktion hängt vom Zustand und Eingang ab
  → Mealy-Schaltwerk



- Systeme mit Zustand (Schaltwerke) können mithilfe von Automaten beschrieben werden
- **Endliche Automaten:**
  - Bestehen aus
    - endlicher Zustandsmenge Q,
    - einem Startzustand $s \in Q$,
    - einem endlichen Eingabealphabet Σ,
    - einer Endzustandsmenge $F \subseteq Q$ und
    - einer Übergangsfunktion δ
  - Endlicher Automat: endliche Zustandsmenge
  - **Moore-Automat:** zusätzlich Ausgabealphabet Ω und Ausgabefunktion $\lambda : Q \to \Omega$, die beschreibt, welches Ausgabesymbol bei gegebenem aktuellen Zustand ausgegeben wird
  - **Mealy-Automat:** zusätzlich Ausgabealphabet Ω und Ausgabefunktion $\lambda : Q \times \Sigma \to \Omega$, die beschreibt, welches Ausgabesymbol bei gegebenem aktuellen Zustand und Eingabesymbol ausgegeben wird
  - Graphische Darstellung als Zustandsgraph

Moore- & Mealy Automat

Eigenschaften

- **Deterministischer Automat:** Nachfolgezustand ist für jeden Zustand und jedes Eingabesymbol eindeutig definiert. Übergangsfunktion: $\delta : Q \times \Sigma \to Q$
- **Nichtdeterministischer Automat:** es gibt ein Zustand-Eingabesymbol-Paar für das der Nachfolgezustand nicht eindeutig ist. Übergangsrelation: $\delta \subseteq Q \times \{\Sigma \cup \{\epsilon\}\} \times Q$
  (ϵ ist das Leerwort - es kann also auch ohne Eingaben zu Zustandswechsel kommen)
- **Vollständiger Automat:** für jeden Zustand sind Übergänge für alle Eingabesymbole vorhanden
  (zu jedem $(z, x) \in Q \times \Sigma$ existiert ein z' mit $\delta(z, x) = z'$)
- Wenn nicht anders angegeben, nehmen wir an: Automaten sind vollständig und deterministisch

  - Moore Schaltwerk
    - Ausgänge hängen nur vom Zustand ab
    - Eingänge beeinflussen Ausgänge nicht direkt
    - Für Ausgangsänderung ist ein Zustandswechsel notwendig
  - Mealy Schaltwerk
    - für jeden Zustand sind – gesteuert von den Eingängen – verschiedene Ausgänge möglich
    - sofortige Reaktion der Ausgänge auf die Eingänge
    - typischerweise weniger Zustände

7.1

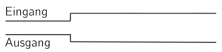
Basic info

- Bisher:
  Kombinatorische Logik: Ausgänge nur vom momentanen Zustand der Eingänge bestimmt
  - Keine Speicherung von Information, kein interner Zustand
  - Schaltnetz, „combinational logic“
- Nun:
  Sequenzielle Logik: Ausgänge vom momentanen Zustand der Eingänge **und** vom bisherigen Verlauf der Eingänge (Historie) bestimmt
  - aktueller Zustand wird in Speicherelementen gehalten
  - Schaltwerk, „sequential logic“

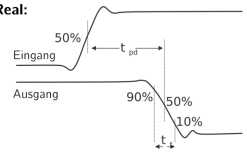
Zeitverhalten

- Z.B.: NOT-Gatter
- Ladungsträger benötigen Zeit zum Fließen
- **Durchlaufzeit** t_{pd}
  (propagation delay)
  „Zeitspanne zwischen dem Anliegen eines stabilen und gültigen Signals am Eingang bis zum Vorliegen eines stabilen und gültigen Signals am Ausgang.“
- **Flankensteilheit** t_f
  - Anstiegs-/Abfallzeit (rise/fall time)

Idealisiert:



Real:

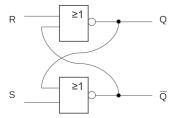


Sequenzielle Logik

Speicherelemente

- Schaltungen, die 1 Bit über einen gewissen Zeitraum speichern können
- Besitzen 2 stabile Zustände („bistabile Kippstufe“)
  - Set („gesetzt“)
  - Reset („rückgesetzt“)
- Diese Zustände können gespeichert werden
  - Solange Spannung anliegt
- Zustandswechsel durch Signale von außen

- Prinzip der Rückkoppelung

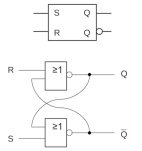


R ... Reset
S ... Set
Q ... Ausgang
$\overline{Q}$ ... Neg. Ausgang

RS-Latch

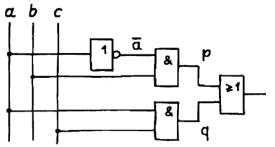
S	R	Q_{next}	
0	0	Q	Hold
0	1	0	Reset
1	0	1	Set
1	1	X	nicht erlaubt

Warum ist S=R=1 nicht erlaubt?



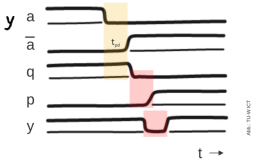
Hazards

- Temporäre Falschaussage einer Boole'schen Funktion $f(a, b, c)$ am Ausgang y



$f(a, b, c) = (\neg a \wedge b) \vee (a \wedge c)$
Annahme:
$a = b = c = 1 \rightarrow f(a, b, c) = 1$

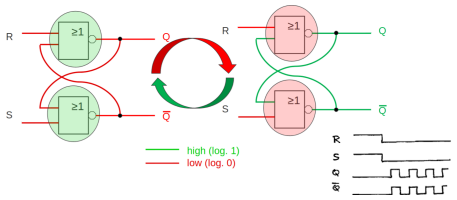
$a = 1$ geht über auf $a = 0$



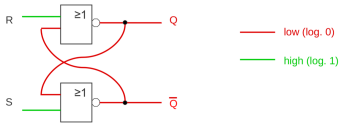
Abhilfe: Signale nur in festgelegten Intervallen betrachten (Takt)

Metastabilität

- Gleichzeitig von $R = S = 1$ auf $R = S = 0 \rightarrow Q = \overline{Q} = 1 = 0 = 1 = \ldots$
- Ausgänge pendeln undefiniert zwischen 0 und 1 (die Ausgänge sind metastabil)

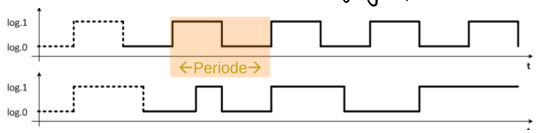


- $R = S = 1 \rightarrow Q = \overline{Q} = 0$ (Widerspruch zu $Q = \neg\overline{Q}$)
  - Was passiert nun wenn R und S gleichzeitig auf 0 wechseln?



Taktsignal

- Binäres Signal zur Koordination von Schaltung
  ↳ oszilliert zw. 2 versch. Spannungspegel
- Synchronisierung von Schaltungen
  → Gleichzeitig Ausführen von Berechnungen
- Takt kann periodisch (üblich; oben)
  aperiodisch (unten)
  sein
- Maßzahl für Geschwindigkeit d. Datenverarb.
  ↳ Taktfrequenz [Hz]
  1 Hz = eine Schwingung pro Sek



Schaltung

- **Asynchrone Schaltung**

→ Zstd kann jederzeit variieren

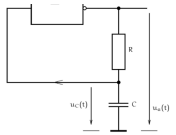
→ Bauteile agieren unabhängig voneinander

- **Synchrone Schaltung**

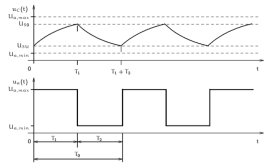
→ Zstd.wechsel nur zu diskreten Zeitpkt
  ↳ Einheitlich für alle (Speicher-) Elem
  ↳ Einfacher zu analysieren (diskret, schrittweise)
  ↳ Eindämmung von Hazards
    (Ausgl. Unterschied. Gatter-Durchlaufzeit)
  ↳ Nahezu alle auswendigeren Schaltungen
  ↳ Problem bei hohen Taktfrequenz
    → Signallaufzeit (Verzögerungen) in den Leitungen

Wie erzeugt man einen Takt?

Taktgeber



T_0: Periodendauer in Sekunden
$\frac{1}{T_0}$: (Takt-)Frequenz in Hertz (Hz)



Stabiler: Quarzoszillatoren

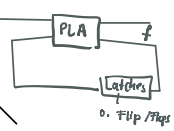
Tabellenspeicher

- Abrufen (Gedächtnis)
- wenn als binäre Zeichenkette gespeichert → Wort
- je nachdem welche Eingangsf. anliegt → wird d. an der Stelle gespeicherte Bitfolge ausgegeben

Funktionsspeicher

- Berechnen (kein Gedächtnis)
- Umsetzung einer Gatterschaltung (Fkt)
- Ausgabe wird jedes Mal neu ermittelt
  - ↳ Abhängigkeit von Eingangssignalen wird Schaltung durchlaufen & Ausgangssignale werden berechnet

können als einander realisiert werden } Auswahl d. Technologie abhängig von Effizienz



PLD

Programmable Logic Device

PLA

= Programmable logic Array
- One time programmable (OTP)
- Für Prototypen /Kleinserien
- Realis. von Fkt in DNF
- UND – ODER Matrix
- Alle Ausgänge Open-Collector
- Diode (lässt Strom in eine Richtung durch & blockiert in andere) } hier Sicherung

PAL

= Programmable Array logic
- Wie PLA
- ODER Matrix vordefiniert

RAM

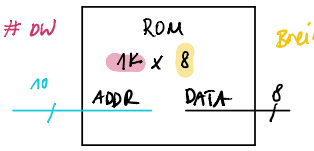
- beliebiger Zugriff
- Basis: Binary cells

statisch
SRAM
nicht-fl
- Aufbau wie FlipFlop
- Leitungsbreite sehr gering (Strahlbarkeit)

dynamisch
DRAM
flüchtig
- Speicherung im Kondensator
- Schaltkreis zum Speichern/Lesen
- Auffrischung periodisch nach einigen Millisekunden

ROM

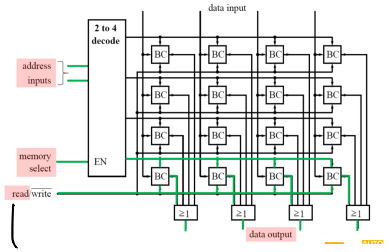
(M, P, EP, EEP, Flash-EP)
- permanente Speicherung } nicht flüchtig
- vollständige Wahrheitstabelle für alle Eingangskombis & jeden Ausgang
- maskenprogrammiert (ein- & mehrfach anwendungsprog. bar)



Busleitung mit 10 Signalleitungen

Anfang

■ Matrix aus „binary cells"



Speichern oder Lesen

FPGA

(= Field Programmable Gate Array)
- mehr & kleinere log. Blöcke
- Konfig erstellt aus:
  - → graphisch (Schaltplan)
  - → VHDL (very high Speed Integrated circuit HW Dex. lang.)
  - → Verilog / HDL

Prozessor

= Schaltwerk zur Bearbeitung von Daten

- Besteht aus (= ist einheiten innerhalb dig. Rechensystems, d.

Rechenwerk
Hier: ALU
Führt R.-operationen aus wie carry Ripple Adder

Leitwerk
Hier: M/R
- Steuert RF, in der Befehle (e.P.) ausgeführt werden
- Entschlüsselt/ modifiziert Befehle
- gibt erforderliche dig. Signale für Ausführung eines digl. Rechensystems

micro 16

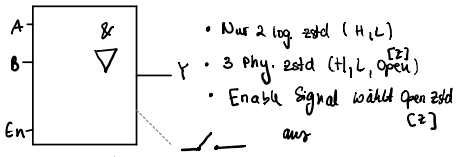
ASICs

= Application Specific Integrated Circuit
- Anwendungsspezifisch, speziell entwickelt für Anwendungsgebiet
  ↳ Schnell & effizient
- Hohe Kosten
- Design-Prägung (full custom → standard
- Ausgangspkt → standard cells cell)
  ↳ Einzelne Verdrahtungen werden Kunden spezifisch vorgenommen

Speicher

Zusammenschalten von Ausgängen

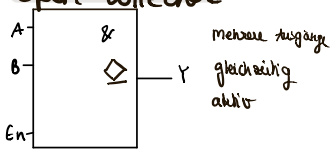
Tri-State



- Nur 2 log. zstd (H, L)
- 3 Phy. zstd (H, L, Open [Z])
- Enable Signal wählt Open zstd [Z] aus

A	B	EN	Y	
L	L	H	L	Schalter geschlossen
L	H	H	L	
H	L	H	H	
H	H	H	L	
X	X	L	Z	Schalter offen

Open Collector



Mehrere Ausgänge gleichzeitig aktiv

Nur High, wenn alle Ausgänge High sind
→ Hier 0