Universität Bamberg

GRABS: Grundlagen der Rechnerarchitektur und Betriebssysteme

Vorlesung 1:
Alles ist eins – ausser der Null

Michael Engel (michael.engel@uni-bamberg.de)
Lehrstuhl für Praktische Informatik, insbes. Systemnahe Programmierung
https://www.uni-bamberg.de/sysnap

Syllabus (geplant)

Organisatorisches (13.4.)

1. **Alles ist eins – ausser der Null (16.4.):**
   Bits, kombinatorische Logik, Schaltungen, einfache Arithmetik
2. **Motivation und Überblick (20.4.):**
   Struktur von von Computersystemen und HW/SW-Schnittstellen
3. **Minimierung und EAs (23.4.):**
   Minimierung von Schaltungen, Sequentielle Logik, endliche Automaten
4. **Rechnerarithmetik (27.4.):**
   Ganzzahlen, negative Zahlen und Komplementdarstellung, **Multiplikation, Beschleuniger**
5. **Rechnerarithmetik (30.4.):**
   Fix/Gleitkommazahlen **(+denormalisierte Darstellung)**, Zeichendarstellung
6. **Speicher (4.5.):**
   Halbleiterspeicher, Speicherhierarchie, **Caches**
7. **Von Bits zu Prozessoren (7.5.):**
   CPU als endlicher Automat, Komponenten der CPU, Assembler & Maschinensprache, Übersetzungsvorgang und Äquivalenzen zu Hochsprachencode
8. **Von Pipelines und Kontrollpfaden (11.5.):**
   RISC-V-Architektur, Registertransfer-Beschreibung, Pipelines, Konflikte
9. **Ein- und Ausgabe (18.5.):**
   Schnittstellen, IRQs, Softwarebehandlung

Syllabus (geplant)

10. **Hardware/Software-Schnittstellen (21.5.):**
    Systemaufrufe, Privilegienmodi
11. **Vom Programm zum Prozess (28.5.)**
    Programmrepräsentation, Prozess vs. Programm, Zustand
12. **Prozesse und das Betriebssystem (1.6.)**
    Definition Betriebssystem, Ressourcen, Prozesse, Scheduling, Asynchronität und Signale
13. **Ressourcenzugriff und -multiplexing (8.6.)**
    Systemaufrufe, Privilegienmodi, Isolation, Zugriffsmatrix, Speicher, **Capabilities und CHERI**
14. **Nebenläufigkeit und Parallelität (11.6.)**
    Kritische Abschnitte, Koordination/Synchronisation, Konflikte
15. **Persistente Speicherung – Dateisysteme (15.6.)**
    Grundlagen, Unix-Dateisysteme, Erweiterungen: Logging, Journaling, Flash
16. **Virtueller Speicher (18.6.)**
    Hardware für virtuellen Speicher, Funktionsweise auf Betriebssystem-Seite, Nutzung, Isolation
17. **Kommunikation (22.6.)**
    Shared memory, IPC, synchron/asynchron

Syllabus (geplant)

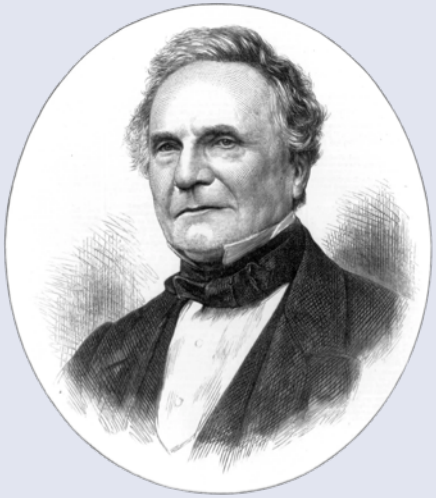
17. **Multicore-Systeme (25.6.)**
    Shared memory, IPC, Cachekohärenz
18. **Speicherkonsistenzmodelle (29.6.)**
    Sequential consistency, relaxed models (TSO…)
19. **GPUs (2.7.)**
    Aufbau, Parallelitäts- und Speichermodelle, Programmierung
20. *Reservetermin oder Gastvorlesung (6.7.)*
21. Ausblick und Zusammenfassung (9.7.)

Am Anfang…

Antikythera
Antikes Griechenland
ca. 60–70 v. Chr.

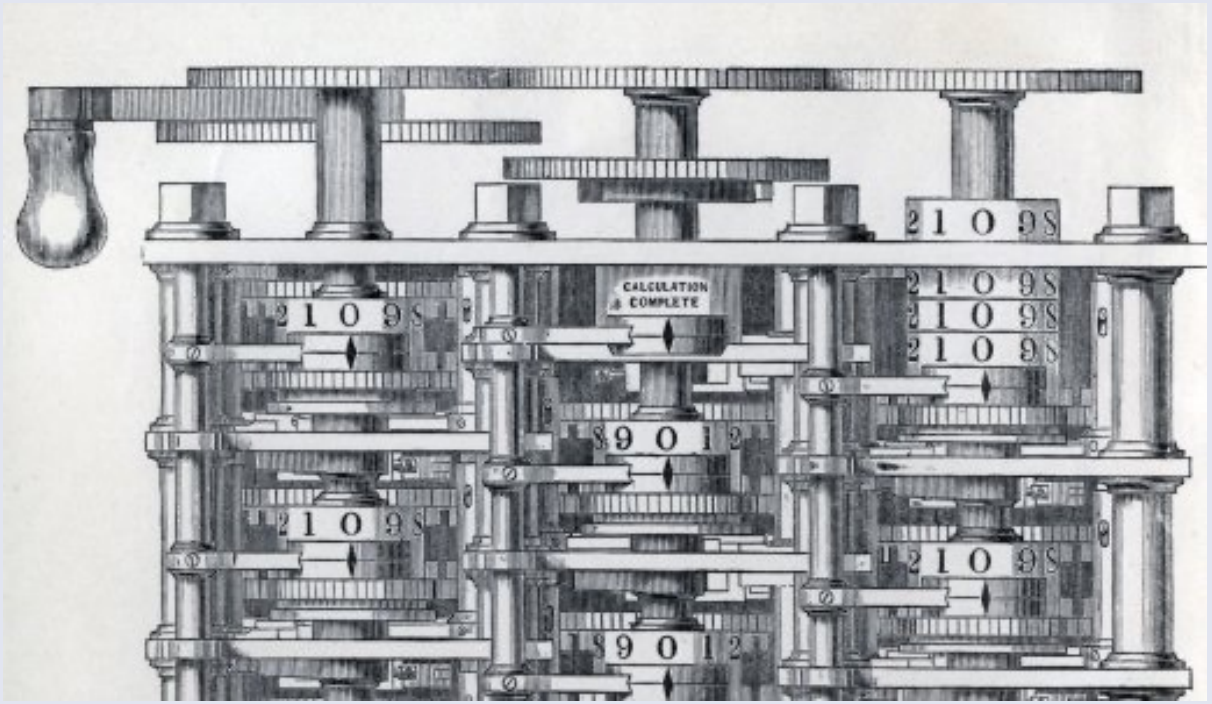
Analoger Computer zur Bestimmung von Kalendern und Himmelsereignissen (z.B. Sonnenfinsternisse)

Differential und Analytical Engines
ab 1820 von Charles Babbage entwickelt
– Difference Engine:
  Automatischer ***mechanischer*** Rechner
– Analytical Engine: ***Programmierbarer***
  Rechner – der erste Computer



Analytical Engine

Mechanische Berechnungen im **Dezimalsystem**
– **10 Positionen pro Zahnrad**
– Übertrag auf nächste Dezimalstelle durch Mitnehmer



Analytical Engine

Das erste Computerprogramm

– 1840 entwickelt für die Analytical Engine von Augusta Ada King, Countess of Lovelace

– Algorithmus zur Berechnung von Bernoulli-Zahlen

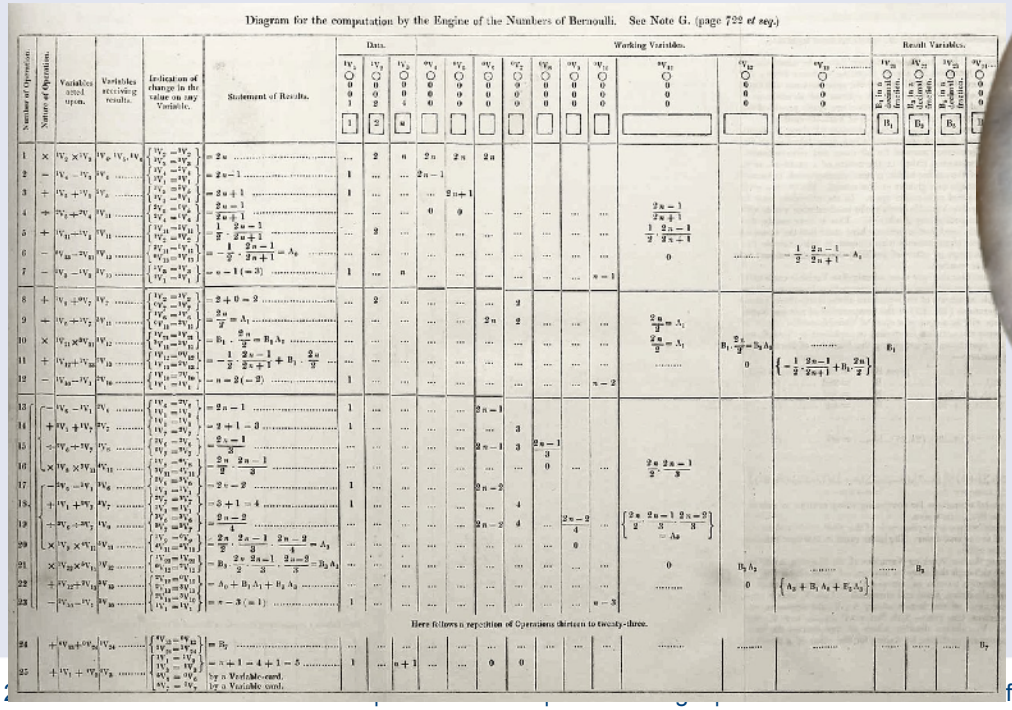


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Zahlensysteme

Brauchen wir wirklich zehn verschiedene Ziffern, um rechnen zu können?

– **Unäres System:** Basis 1
"Strichliste"

– **Römische Zahlen**
seeeehr unpraktisch!

– **Babylonische Zahlen:** Basis 60
Keilschrift, vermutlich Basis für unsere 60 Sekunden/Minuten-Rechnung

Stellenwertsysteme

Universität Bamberg

Probleme mit Zahlensystemen

– **Unäres System**
Striche zählen, um Wert der Zahl zu ermitteln…

– **Römische Zahlen**
Wert einer Ziffer abhängig von Position zur vorherigen/nachfolgenden Ziffer
z.B. CX = 110, XC = 90

– **Unsere gewohnten Dezimalzahlen und auch babylonische Zahlen sind *Stellenwertsysteme***

Stellenwertsysteme

In ***Stellenwertsystemen*** wird der Wert einer Ziffer nur von der Ziffer selbst und deren ***absoluter Position*** innerhalb der Zahl bestimmt!

Zerlegung einer Zahl in ***Exponentialdarstellung:***

$$12305 = 1 \cdot 10000 + 2 \cdot 1000 + 3 \cdot 100 + 0 \cdot 10 + 5 \cdot 1$$

(In der Informatik verwendet man oft "*" anstelle von "·" als Symbol für Multiplikation ⇒ leichter erkennbar, ab jetzt auch hier…)

oder auch

"Ziffer mit Wert 2"

"an Stelle 3" (von rechts, wir fangen bei 0 an)

$$12305 = 1 * 10^4 + 2 * 10^3 + 3 * 10^2 + 0 * 10^1 + 5 * 10^0$$

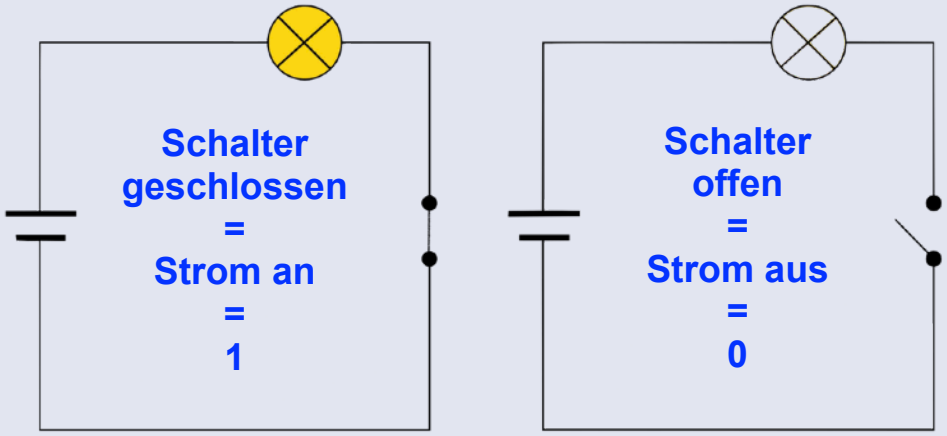
Das Duale Zahlensystem

Universität Bamberg

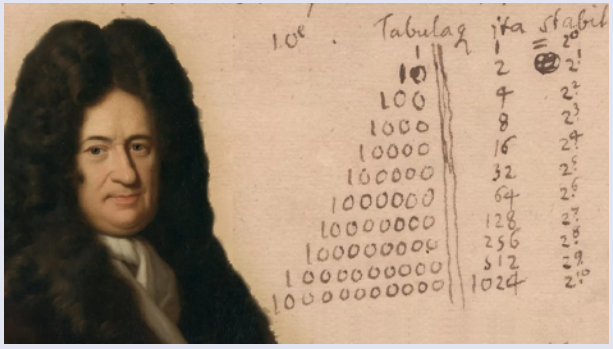
Brauchen wir zehn verschiedene Ziffern? ***Nein!***

Zwei verschiedene Ziffern reichen aus!
(Dual- oder Binärsystem)

Binary digit = Bit!



Einfach durch technische Systeme realisierbar –
z.B. Strom **an = 1**, **aus = 0**



Gottfried Wilhelm Leibniz (Hannover 1696)
mit Vorarbeiten aus Großbritannien und
Spanien aus dem 16. Jahrhundert

Null, eins, mehr?

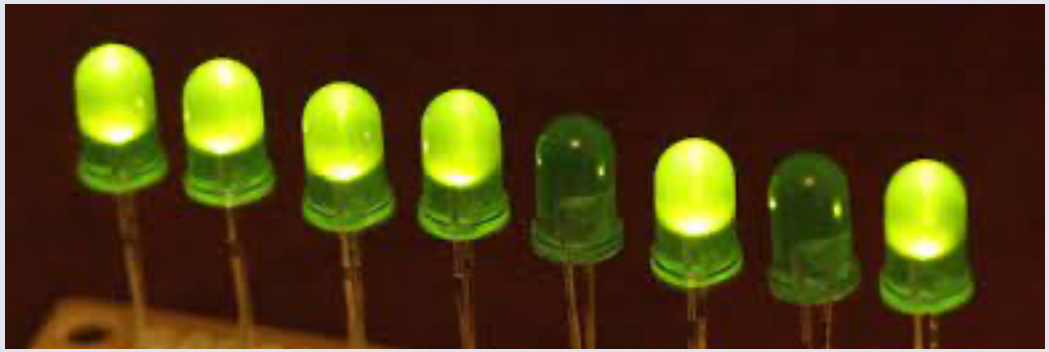
Stellenwertsystem mit nur zwei verschiedenen Ziffern:

$$\mathbf{10110_{(2)} = 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0}$$

$$= \quad 16 + \qquad\qquad 4 + \quad 2 \qquad\qquad = 22$$

Gibt die Basis 2 an

Jede natürliche Zahl ist also durch Kombination einer Menge von "0"- und "1"-Bits darstellbar!



Welche Zahl ist hier dargestellt?

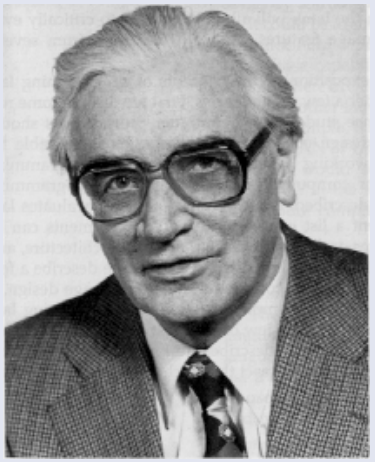
Binär macht’s einfacher!

Die ersten Computer (ab 1939) in den USA rechneten noch im Dezimalsystem und waren sehr komplex und groß!

Konrad Zuse hat seit 1936 **Computer entwickelt, die das Binärsystem und Gleitkomma nutzten** – zuerst komplett **mechanisch** (Z1), danach mit **Elektr(on)ik** (Z3 und ff.) [3]

Z1 (Nachbau, Technikmuseum Berlin)

Z3 (Nachbau, Deutsches Museum München)



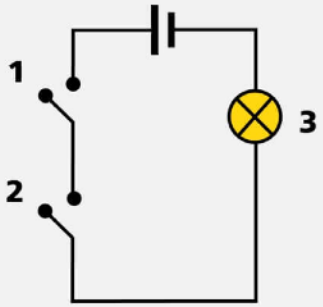
"Rechnen" mit Bits

Auf der Ebene der Elektronik gibt es keine einfache (digitale) Schaltung, die Zahlen addiert!

Wir berechnen Funktionen durch **Kombinieren von Zuständen:**

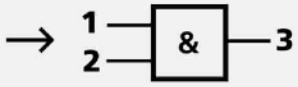
ODER: **mindestens ein** Schalter "1"
UND: **beide** "1"

UND (*AND*)

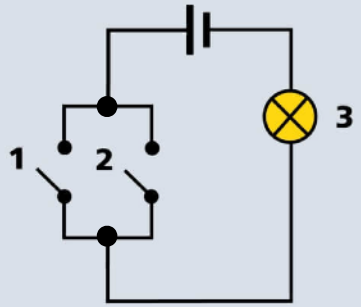


∧ =

1	2	3
0	0	0
0	1	0
1	0	0
1	1	1

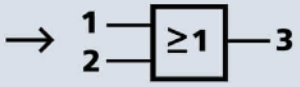


ODER (*OR*)



∨ =

1	2	3
0	0	0
0	1	1
1	0	1
1	1	1



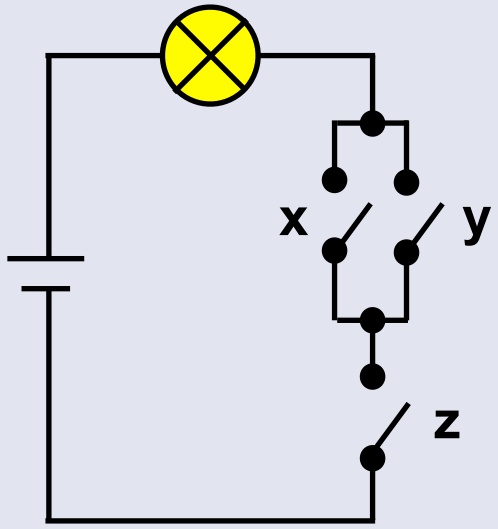
Schaltsymbol = "Gatter" (*gate*)

Schaltungen und Wertetabellen

Wir können die ***Funktion***, die durch eine Kombination von Logikfunktionen berechnet wird, durch eine **Wertetabelle** darstellen:

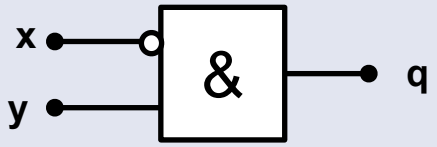
x	y	z	q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Beispiel:
q = (x ODER y) UND z

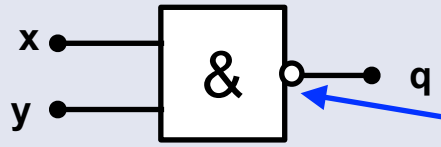


Aus der Eins wird die Null…

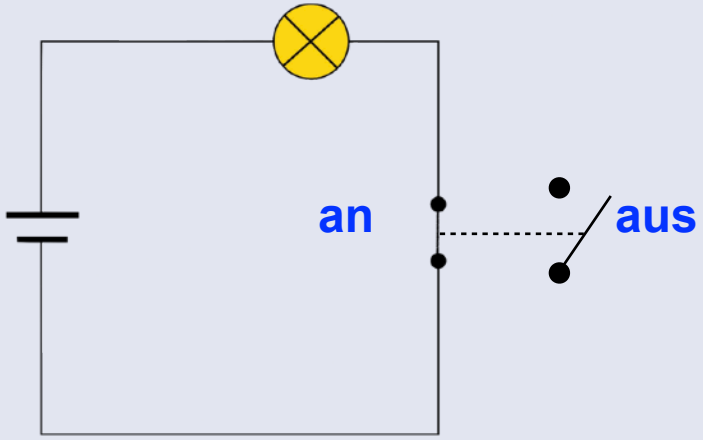
Wir benötigen noch eine weitere Funktion
Invertieren "INV" eines Binärwertes:
aus "0" wird "1" und umgekehrt –
auch als "**NICHT**" (engl. *NOT*) bezeichnet



q = NICHT(x) UND y



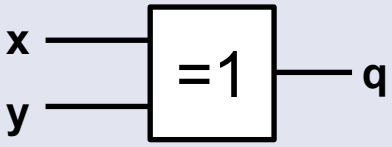
NICHT(q) = x UND y



Hier wird das **Ergebnis**
von UND **invertiert**!

Weitere nützliche Gatter

XOR



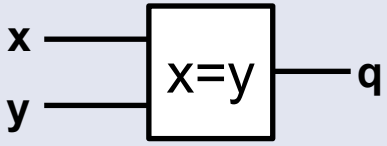
Exklusiv-Oder: "x ≠ y"

x	y	q = x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

x

XOR = NICHT(EQUIV)

EQUIV



Äquivalenz: "x = y"

x	y	q = x EQUIV y
0	0	1
0	1	0
1	0	0
1	1	1

Brauchen wir alle verschiedenen Gatter?

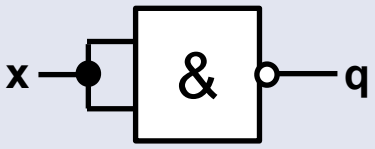
Wir haben gesehen:

- EQUIV lässt sich als NICHT(XOR) schreiben

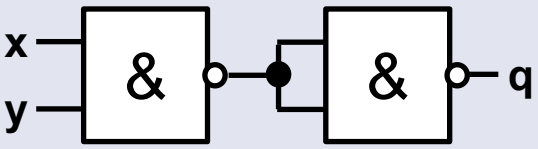
Funktioniert das auch für andere Gatter?

- **JA ⇒ Funktionale Vollständigkeit** durch NAND-Gatter
- Alle möglichen Logikgatter sind durch Menge von NAND-Gattern realisierbar
- **Nachweis** z.B. über Wertetabelle

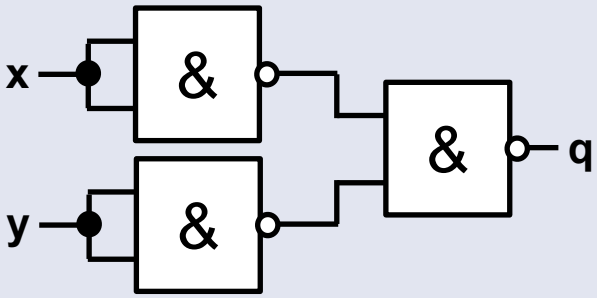
Inverter
q = NICHT(x):



Und
q = x UND y:

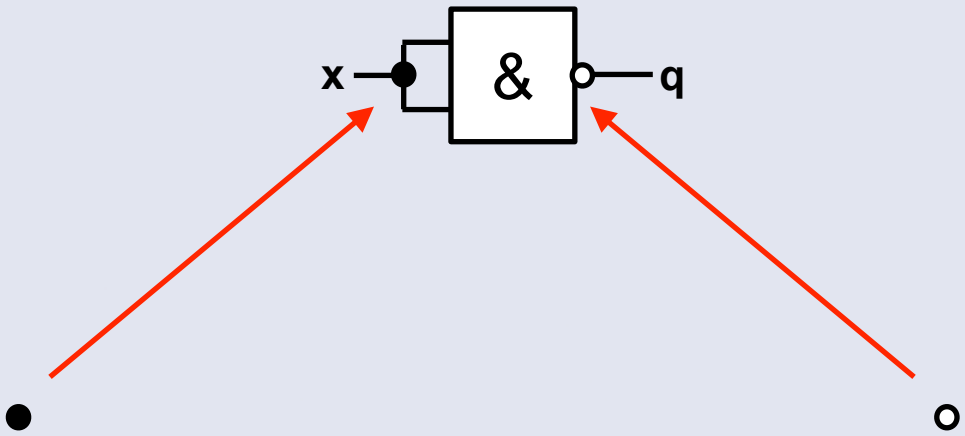


Oder
q = x ODER y:



Hinweis: Punkte…

Achtung:
Wir verwenden zwei Arten von "Punkten" in Schaltungen:

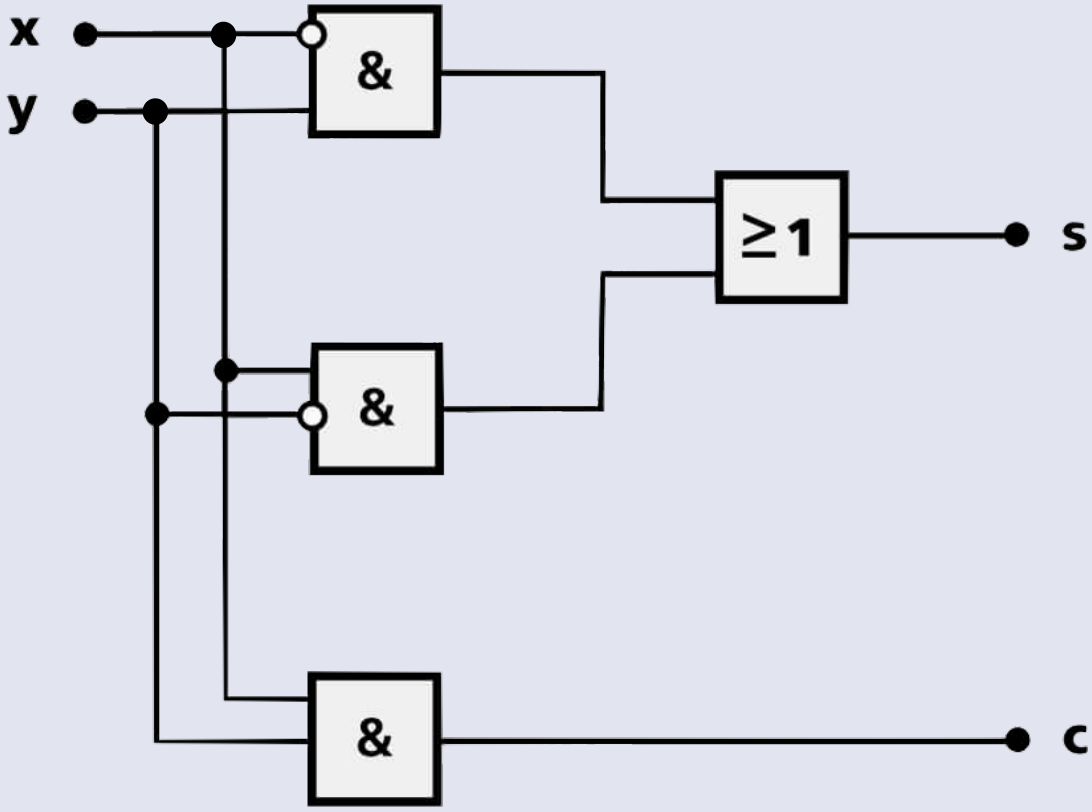


Geschlossener Kreis:
"Lötstelle", eine leitende Verbindung in der Schaltung

Offener Kreis:
Inverter

Addieren von Binärzahlen

"Halbaddierer"-Schaltung:



x	y	Summe s	Über-trag c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

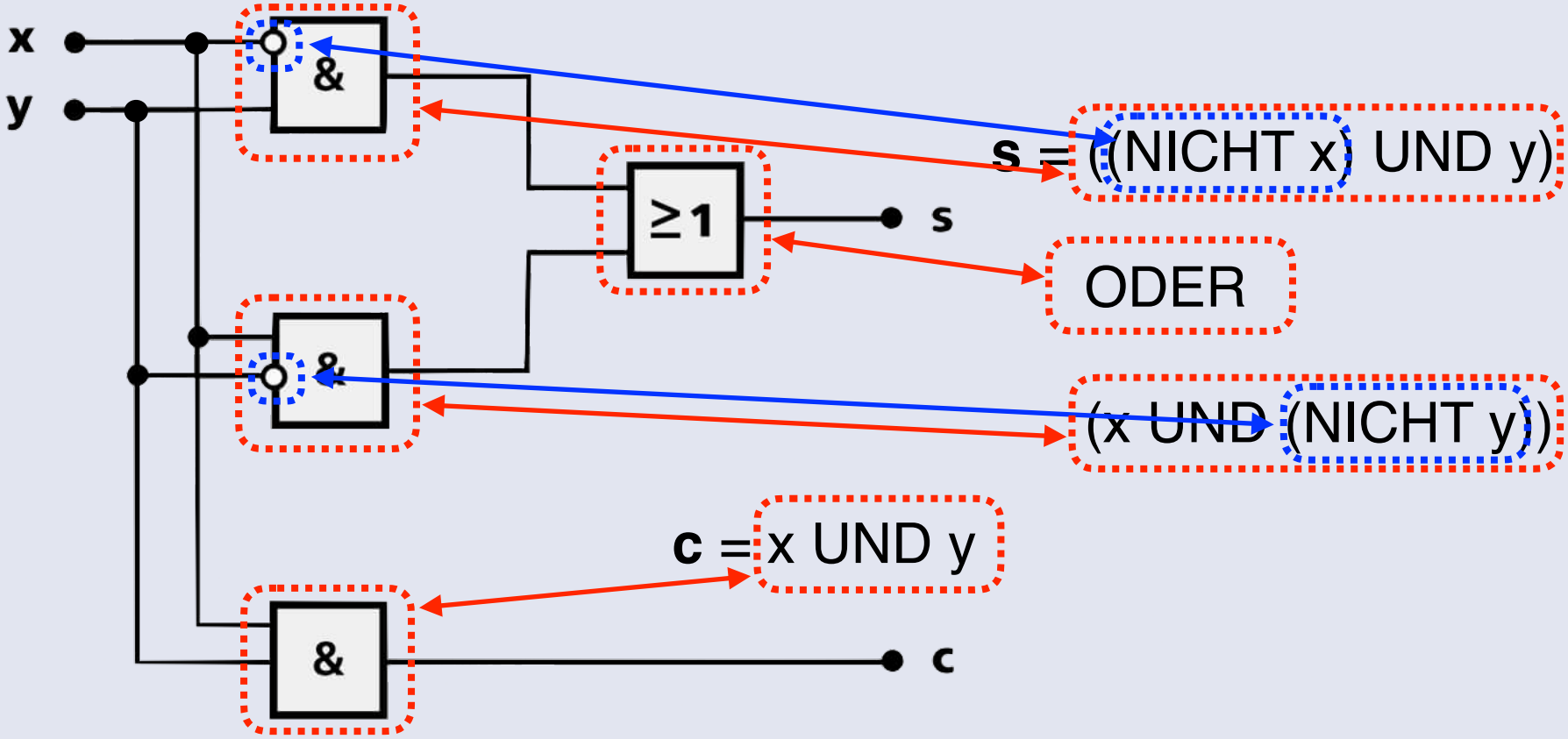
s = ((NICHT x) UND y)
ODER
(x UND (NICHT y))

c = x UND y

"c" von Engl. ***carry*** = **Übertrag**

…im Detail

"Halbaddierer"-Schaltung:



Logische Formelschreibweise

Universität Bamberg

Lästig…

s = ((NICHT x) UND y)
ODER
(y UND (NICHT x))

c = x UND y

Wir verwenden in GRABS diese Variante!

Kürzer: **Logikschreibweise**

s = ((¬x) ∧ y) ∨ (y ∧ (¬x))
c = x ∧ y

UND = ∧
ODER = ∨
NICHT = ¬

Hardwarebeschreibungssprache

s = ((/x) * y) + (y * (/x))
c = x * y

UND = *
ODER = +
NICHT = /

C-Schreibweise

s = ((~x) & y) | (y & (~ x))
c = x & y

UND = &
ODER = |
NICHT = ~

Logische Formelschreibweise

s = ((/x) * y) + (y * (/x))
c = x * y

Wieso wird in dieser Schreibweise "" für UND und "+" für ODER verwendet?*

⇒ **Theorie:**

Die Menge der Zahlen {0,1} mit den Operatoren UND, ODER und NOT stellt eine **boolesche Algebra** dar

Wertetabelle für UND und *

x	y	x UND y	x * y
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Wertetabelle für ODER und *arithmetischen Operator* +

x	y	x ODER y	x + y
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	2

Addieren mehrstelliger Binärzahlen

Wie können wir **mehrstellige Binärzahlen addieren?**

…(fast) wie in der Grundschule gelernt!

$5 + 7 = 0101_{(2)} + 0111_{(2)}$:

$$
\begin{array}{r}
 0\ 1\ 0\ 1 \\
+\ 0\ 1\ 1\ 1\ 1 \\
\hline
 1\ 1\ 0\ 0
\end{array}
$$

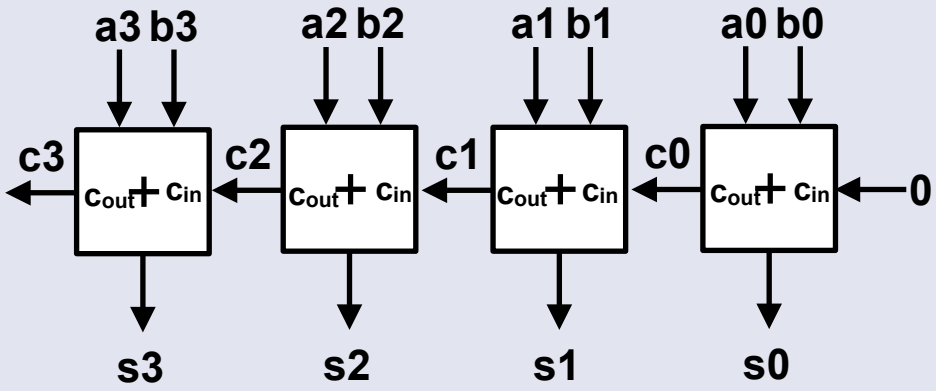
Übertrag

Übertrag berücksichtigen – Volladdierer

Abstraktion – Schaltungen auf "niederer" Ebene zu einem Symbol zusammenfassen

Beispiel: Vier-Bit-Addierer aus Einzelbit***voll***addierern:

a	b	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Eingänge des Volladdierers:

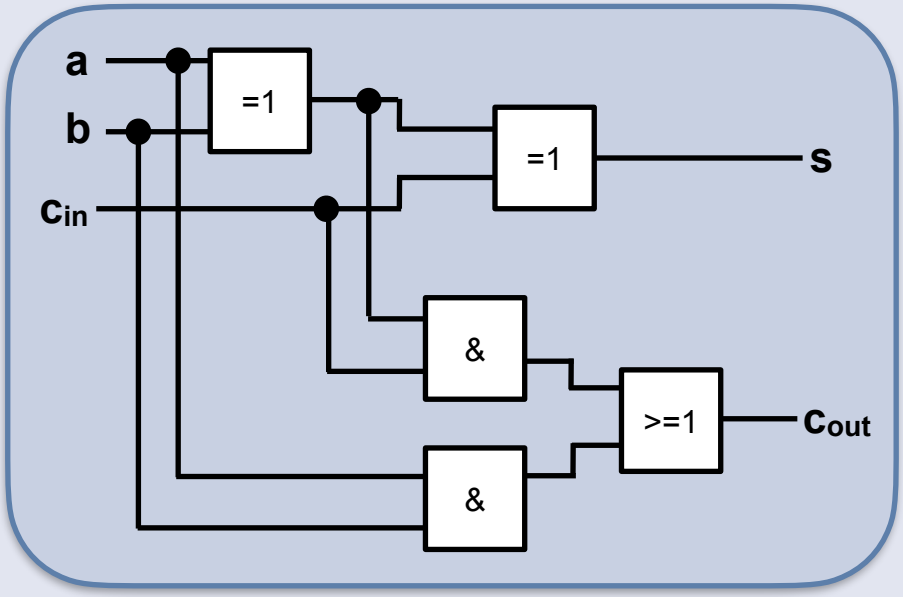
- Zu addierende Bits a_i, b_i
- Eingehender Übertrag c_{in}

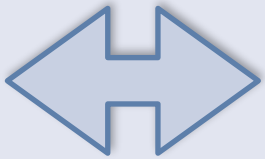
Ausgänge des Volladdierers:

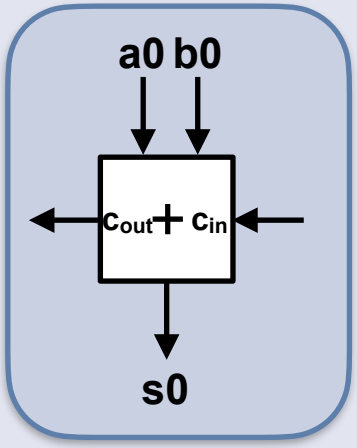
- Summenbit s_i
- Ausgehender Übertrag c_{out}

Übertrag berücksichtigen – Volladdierer

Was fehlt noch? **Carry in**!



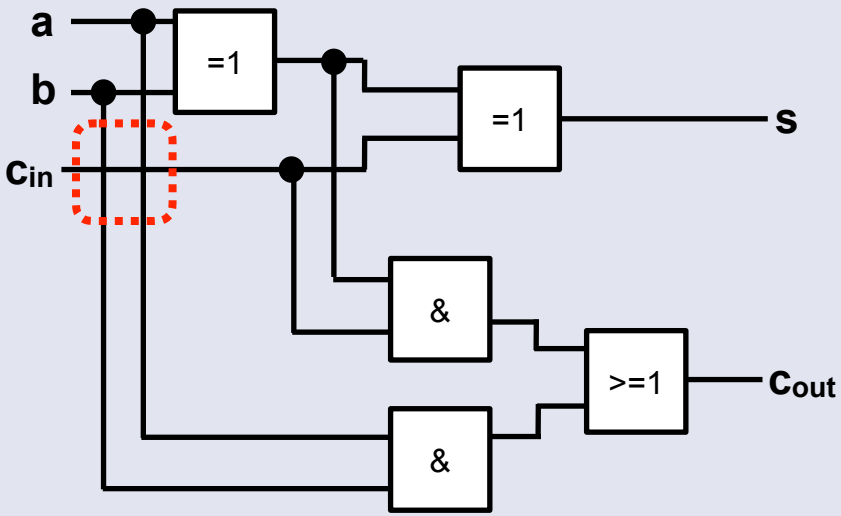




a	b	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Leitende und nicht leitende Verbindungen

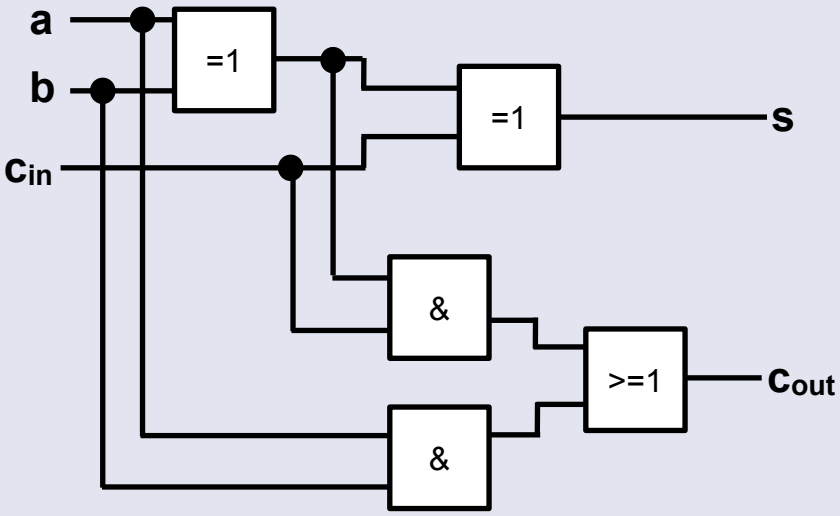
Kreuzende Leitungen ***ohne*** Punkt sind ***nicht*** verbunden:



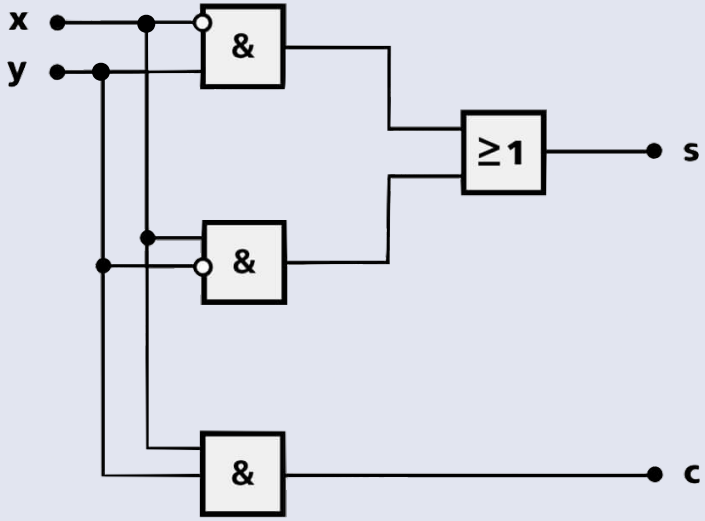
Vergleich Volladdierer – Halbaddierer

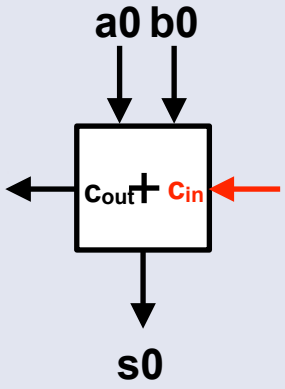
Universität Bamberg

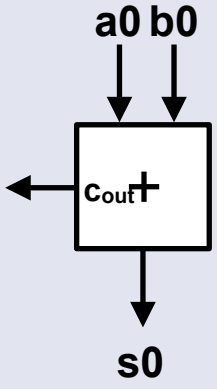
Volladdierer



Halbaddierer

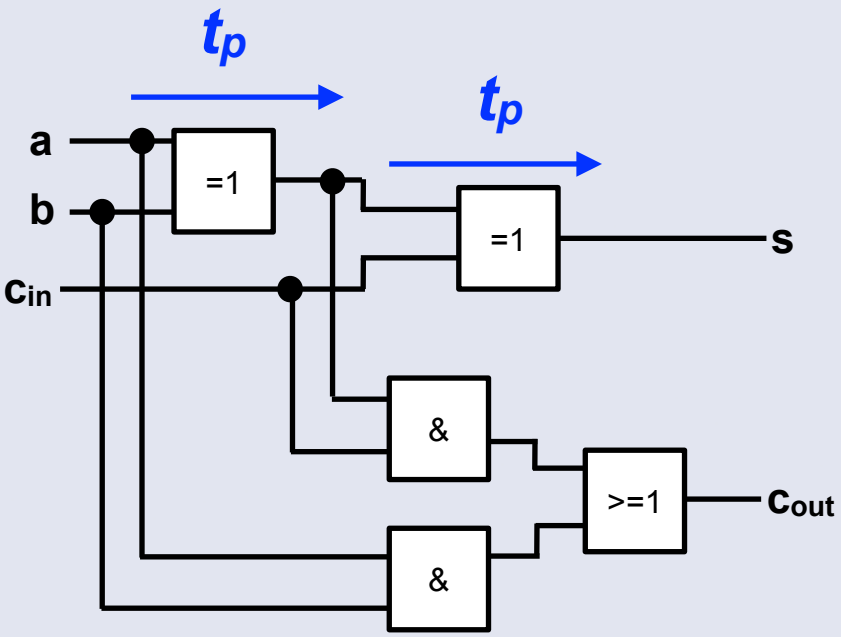






Nicht-funktionale Eigenschaften von Volladdierern

Nichtfunktionale Eigenschaften: Zeit, elektr. Leistungs- und Energieaufnahme, Zuverlässigkeit, Sicherheit, …

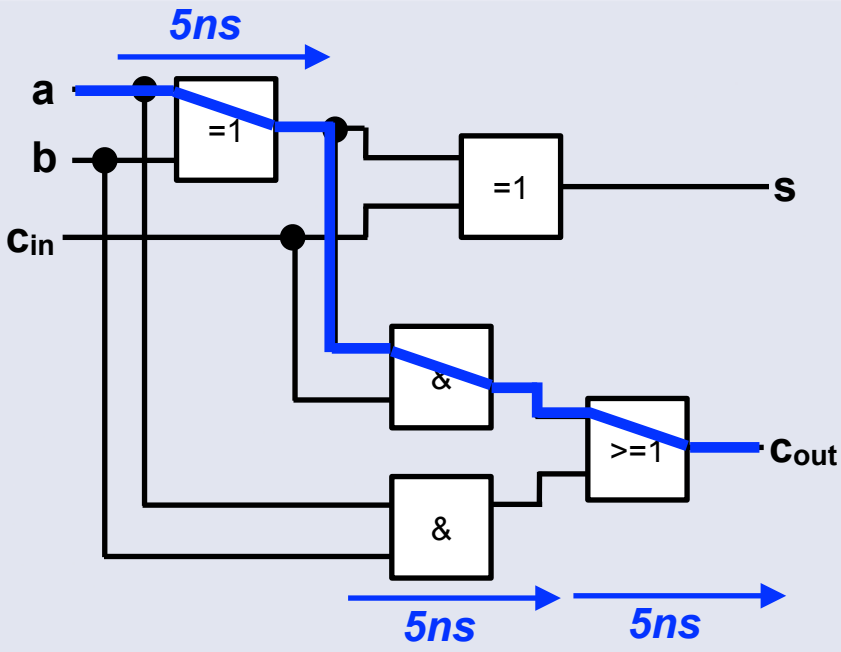


- Jedes Gatter besitzt eine ***Gatterverzögerung*** t_p***:*** Zeit, die vergeht, bis Ergebnis bei Änderung eines Eingangs am Ausgang bereitsteht

Nicht-funktionale Eigenschaften von Volladdierern

- Gatterverzögerung eines Logikgatters abhängig von Technologie, üblicherweise im Nanosekundenbereich, z.B. $t_p = 5$ ns ($5 * 10^{-9}$ s)

- Gatterverzögerung eines 1-Bit-Volladdierers:
  - **Längster Pfad** von Eingang zu Ausgang
  - Leitungsverzögerungen werden ignoriert
- Hier: $3 * t_p = 3 * 5$ ns $= 15$ ns

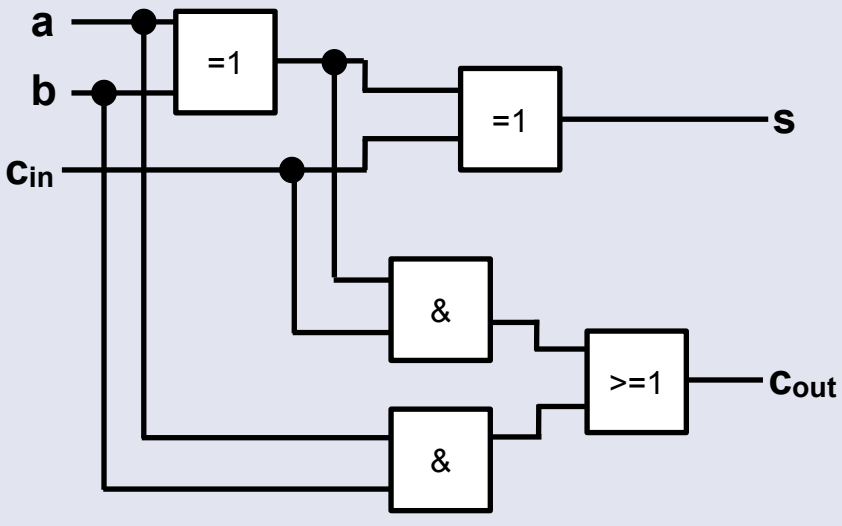


Gatterverzögerung eines Inverters

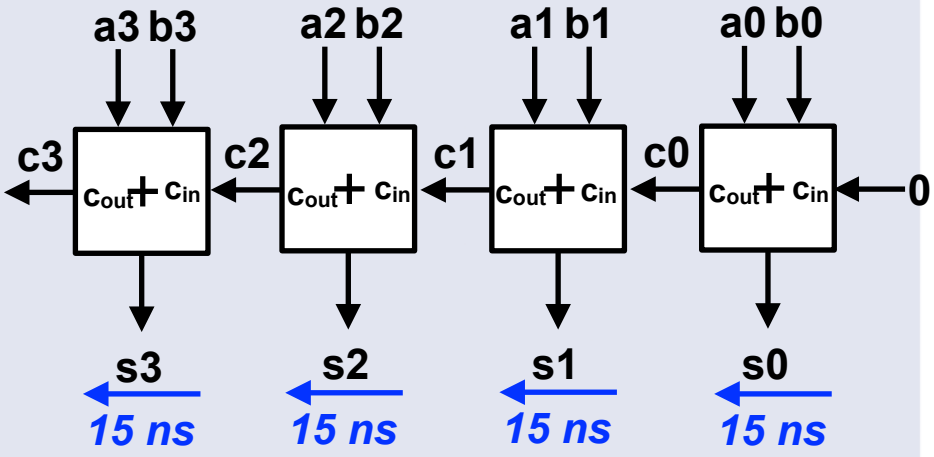
Nicht-funktionale Eigenschaften von Volladdierern

- Gatterverzögerung eines Vier-Bit-Volladdierers
- Carry-Bit muss von Bit 0 nach Bit 3 durchpropagiert werden:

  = 4 * ***t_p(Volladdierer)***
  = 4 * 15 ns = 60 ns
- Maximale Anzahl Additionen pro Sekunde:
  = 1.000.000.000 ns / 60 ns
  = 16.666.666



t_p(Volladdierer) = 15 ns

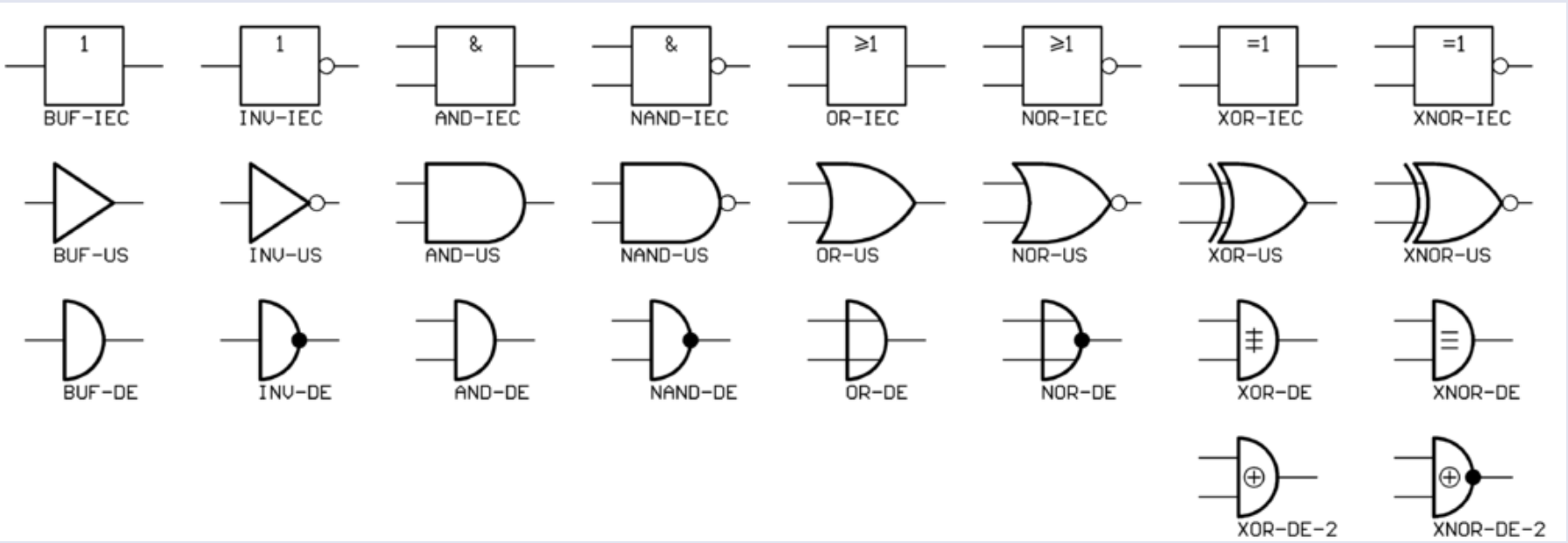


Standards für Gattersymbole

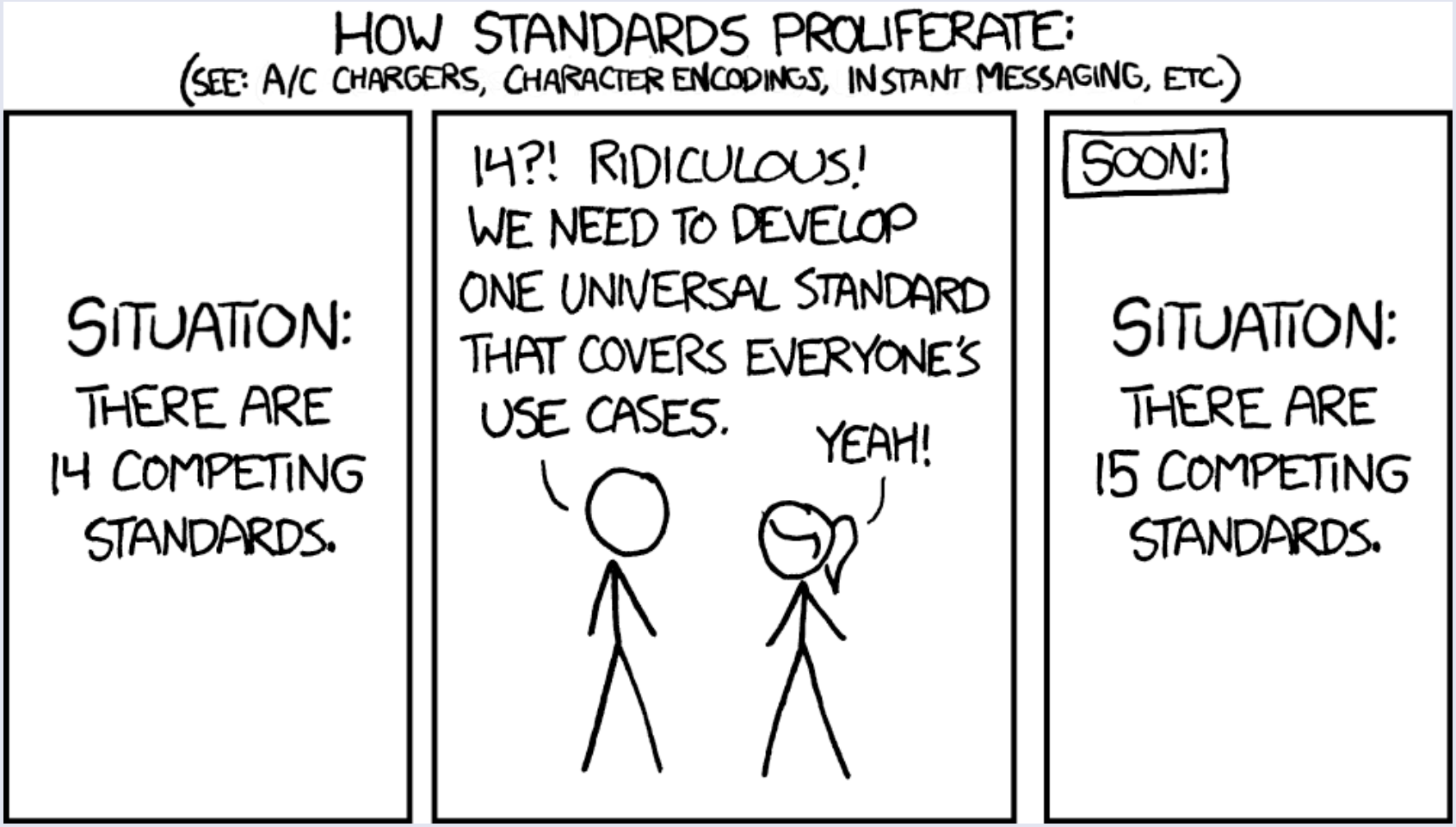
Universität Bamberg

Verschiedene Standards existieren:

- IEC-Standard: Quadrate
- US-Standard
- DIN-Standard: Halbkreise



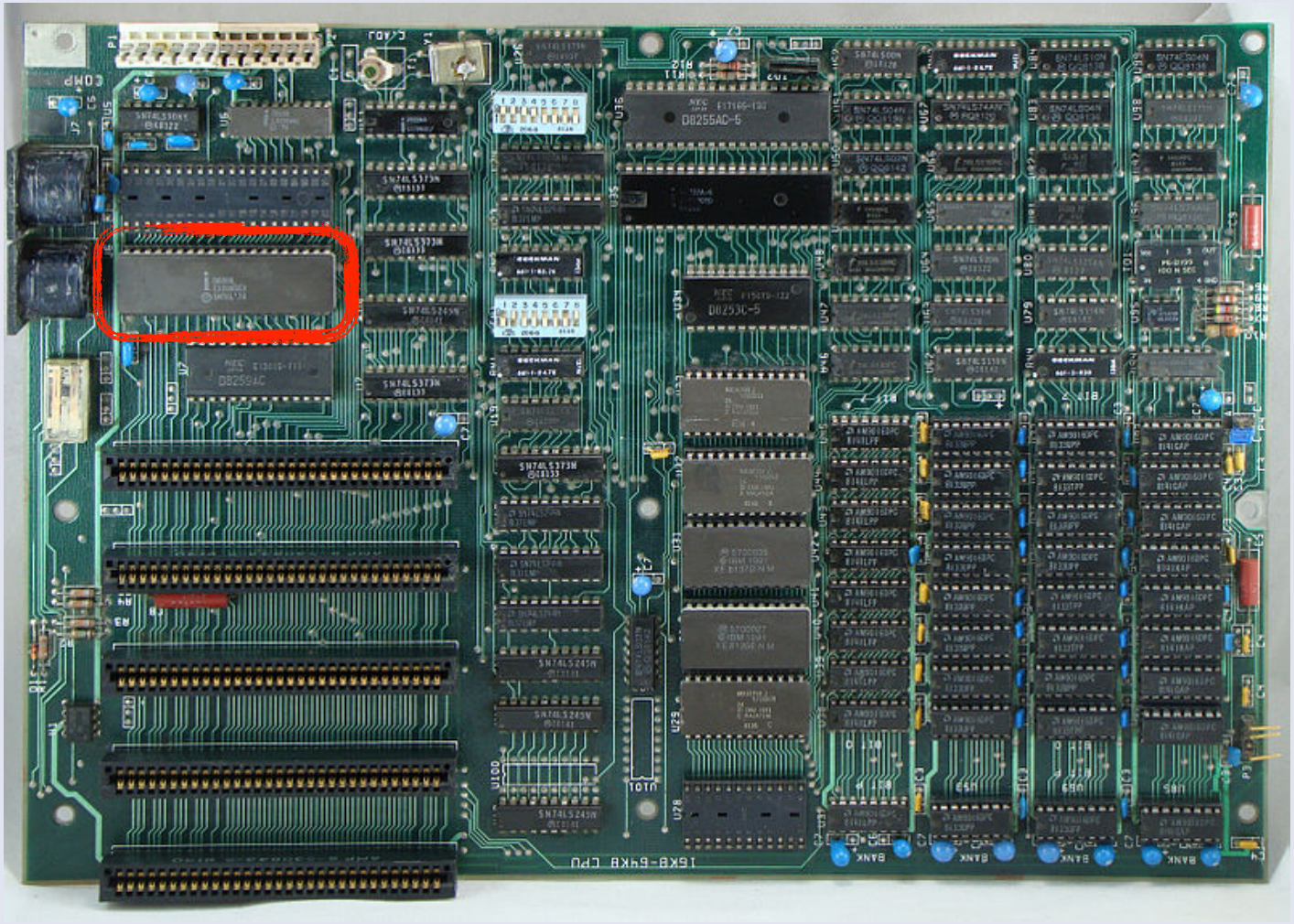
Obligatorischer xkcd…



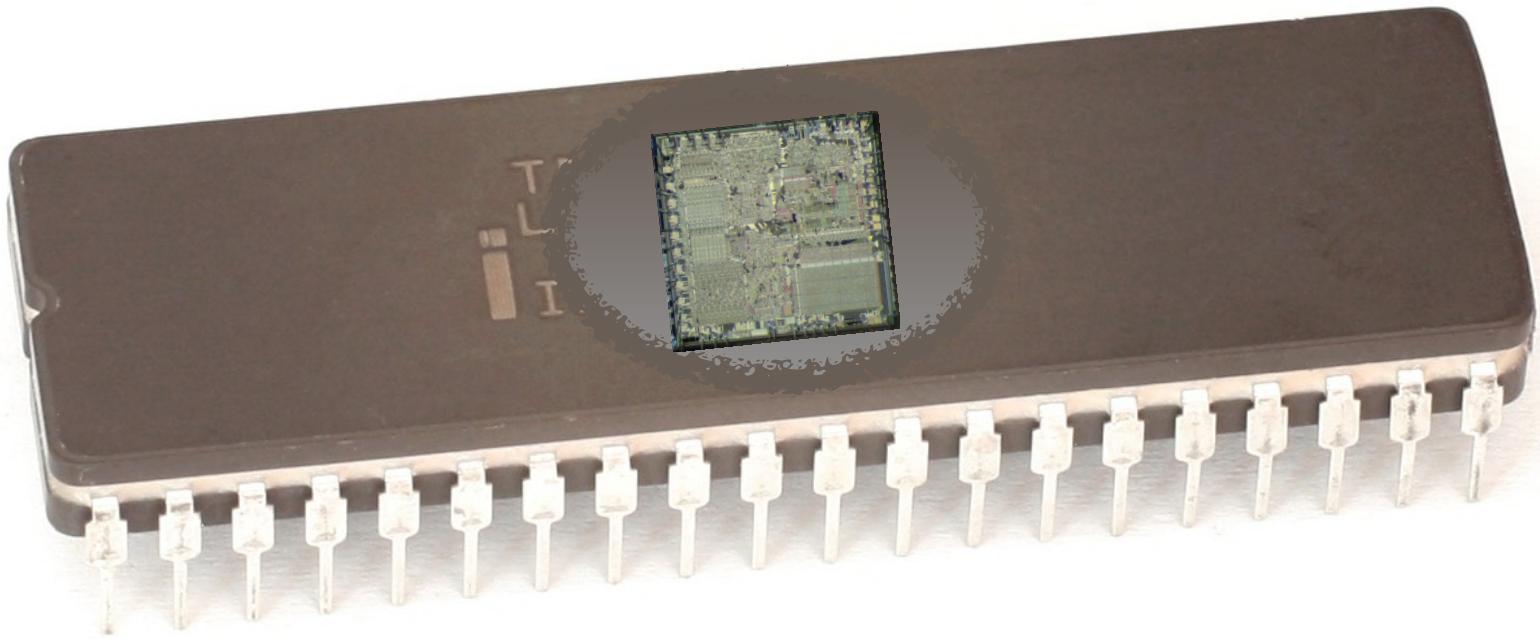
https://xkcd.com/927/

Kann ich das mal sehn?

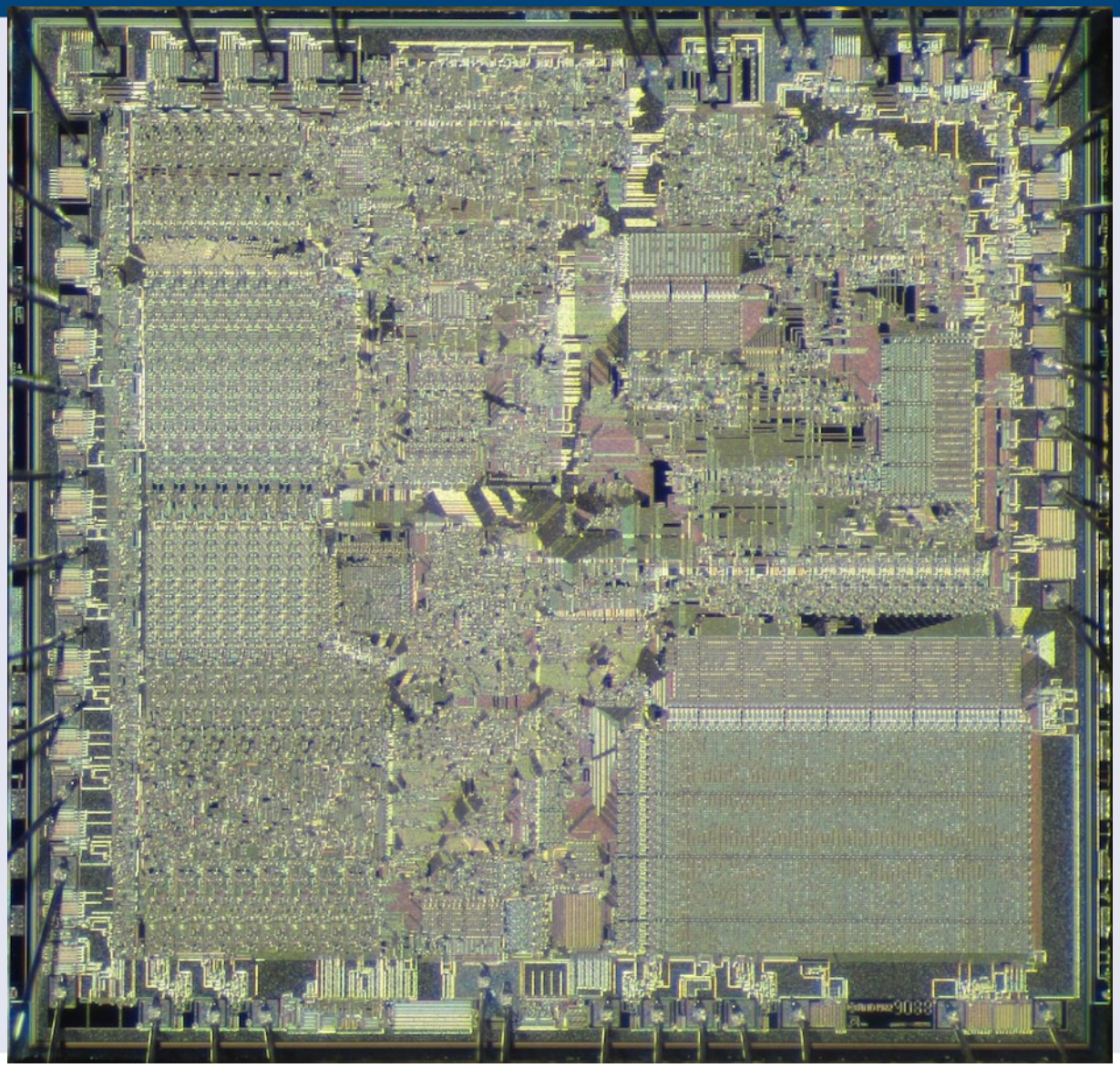
Prozessor (CPU):
Intel 8088
4,77 MHz



Der 8088-Prozessor



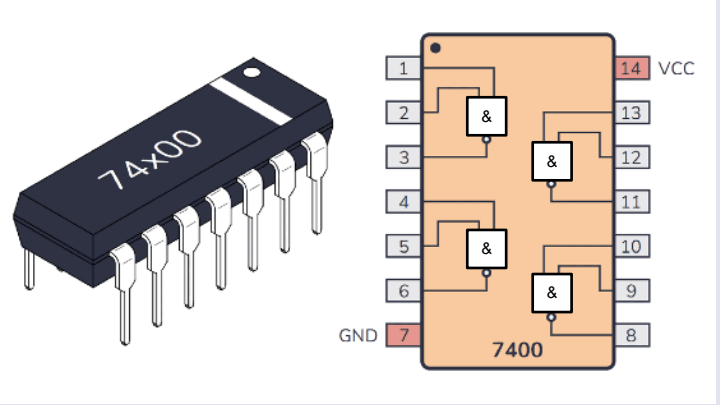
…das Silizium-Die des 8088



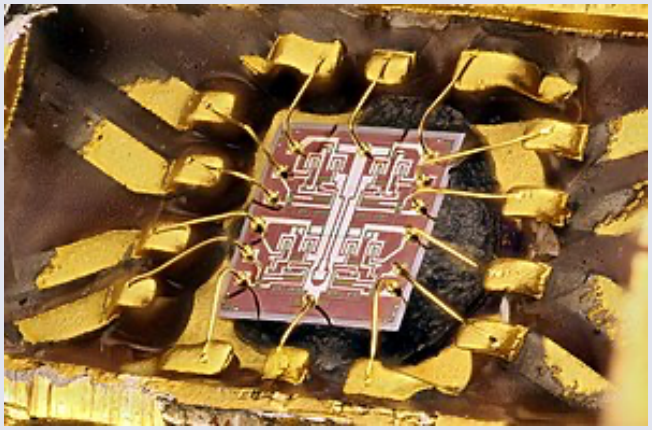
ca. 40.000 Transistoren

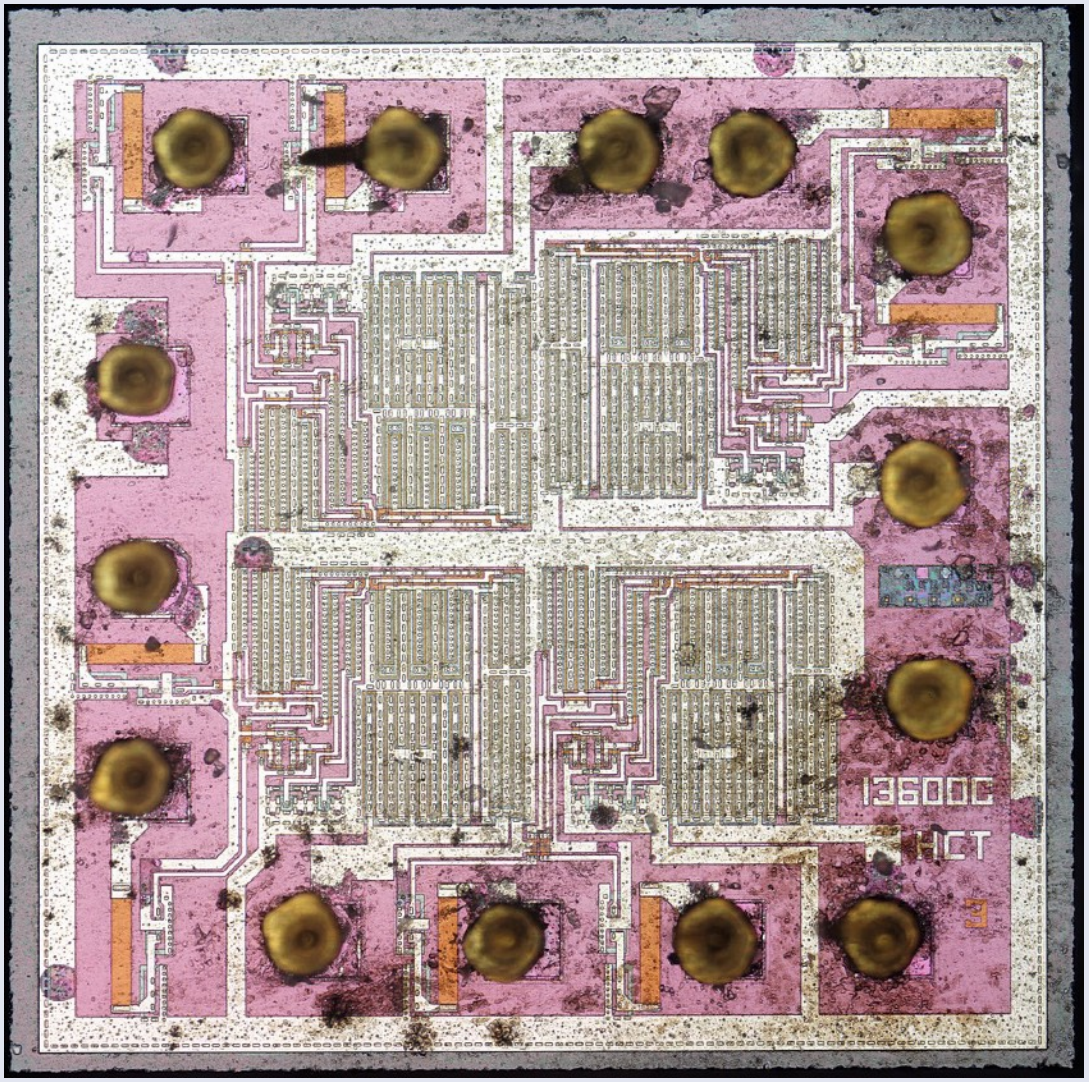
Eine Nummer kleiner: 74HCT00 NAND

Universität Bamberg



74HCT00:
Vier **NAND**-Gatter in
CMOS-Technologie





Wie baut man Logikgatter?

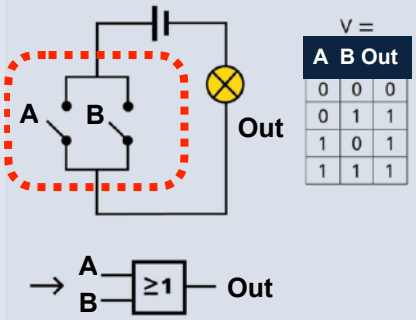
Universität Bamberg

Grundelement von Halbleiter-Schaltungen sind **Transistoren**

Aktuell: **CMOS** (**C**omplementary **M**etal **O**xide **S**ilicon)

"MOSFET"-**Transistoren** werden **als Schalter** genutzt

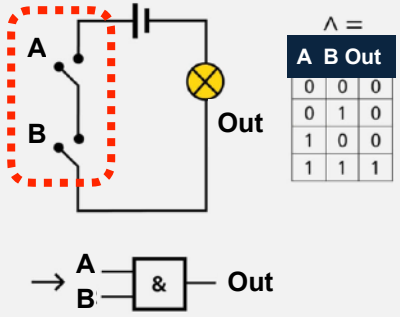
ODER



Schalte "**1**" auf "Out",

wenn **A = 0** **oder B = 0**

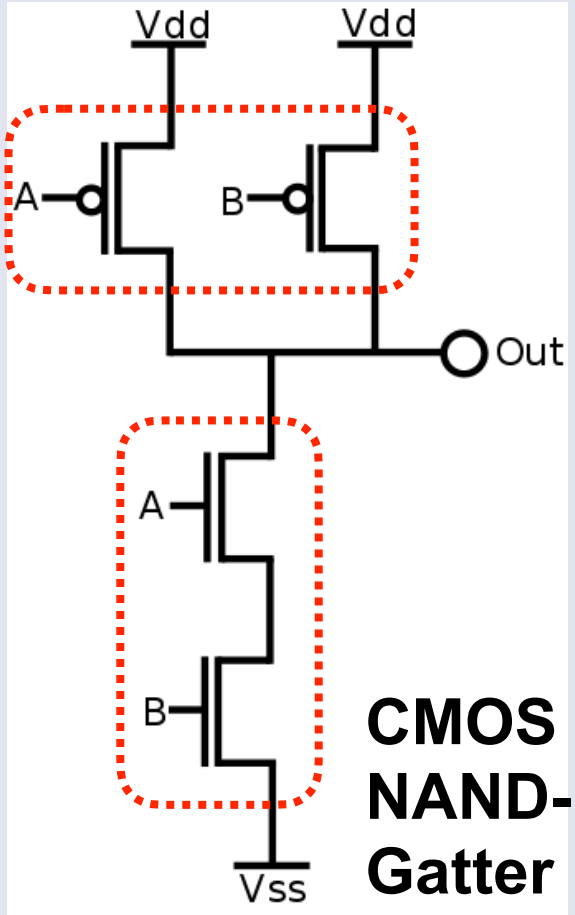
UND



Schalte "**0**" auf "Out",

wenn **A = 1** **und B = 1**

Vdd = logische "**1**"

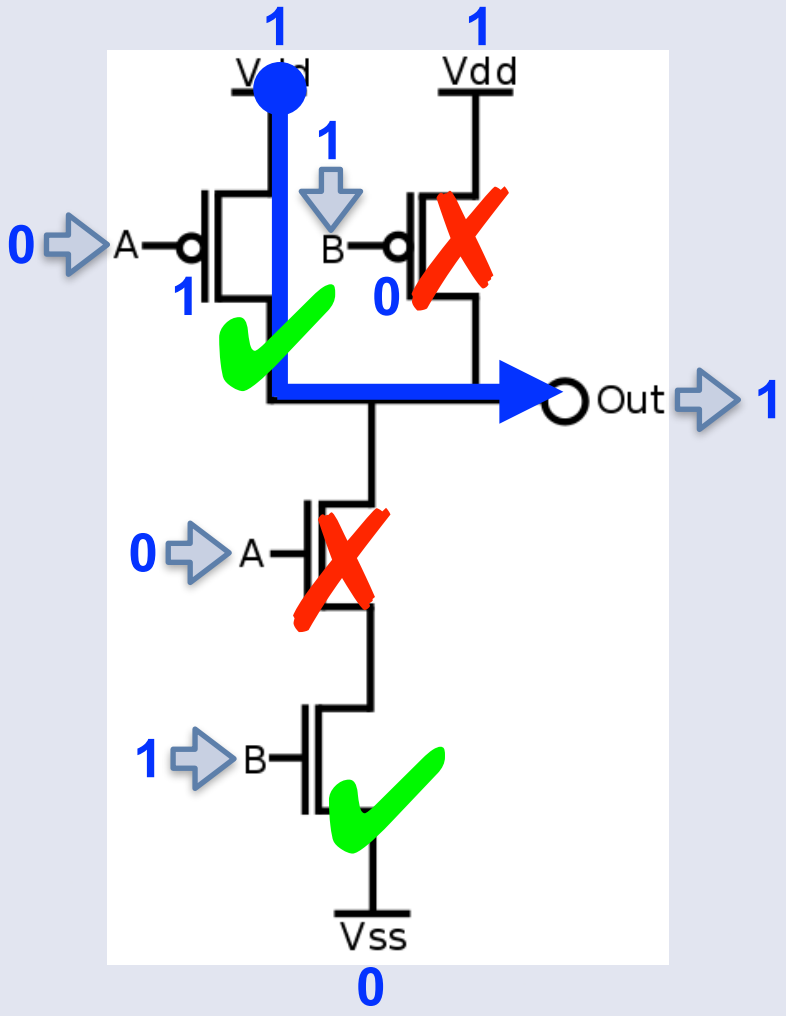


CMOS NAND-Gatter

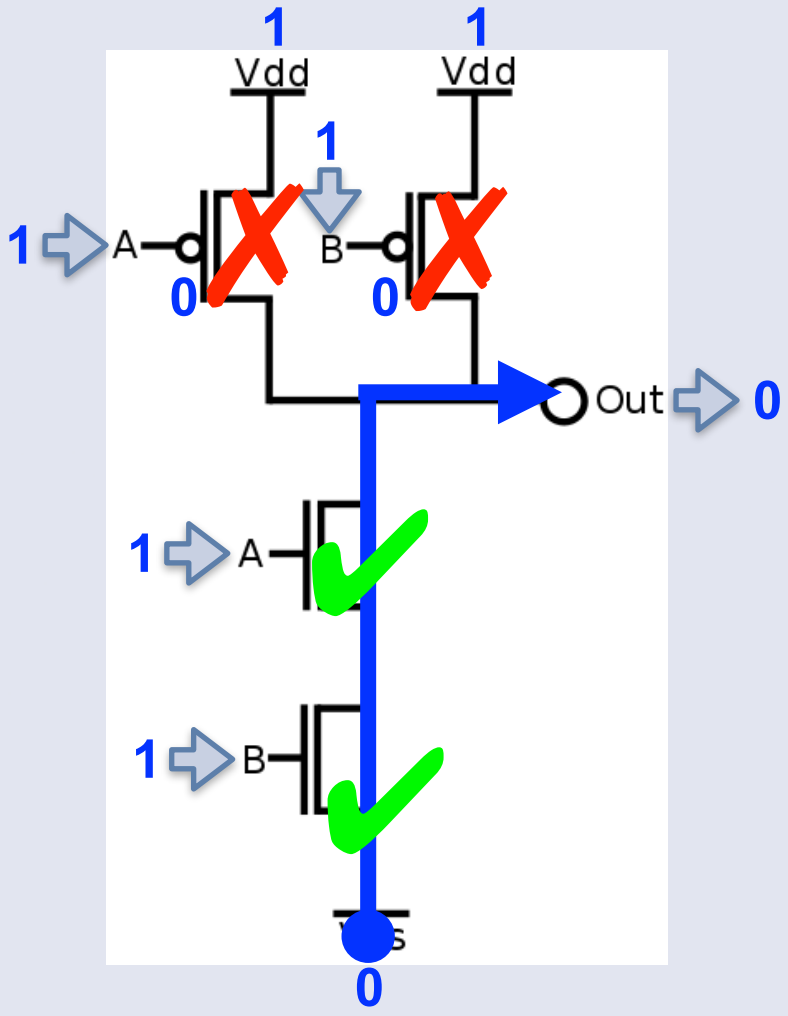
Vss = logische "**0**"

NAND-Logikgatter im Einsatz

A = 0, B = 1 ⇒ /(A * B) = 1

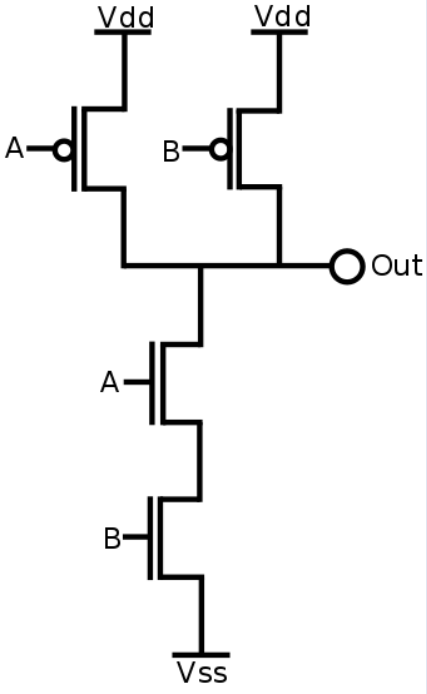


A = 1, B = 1 ⇒ /(A * B) = 0

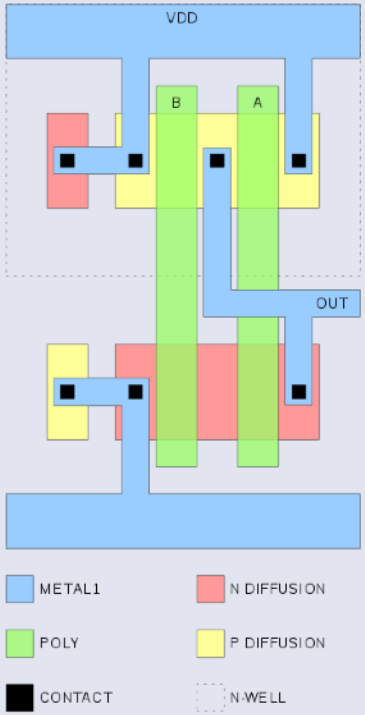


Logikgatter: Transistoren und Silizium

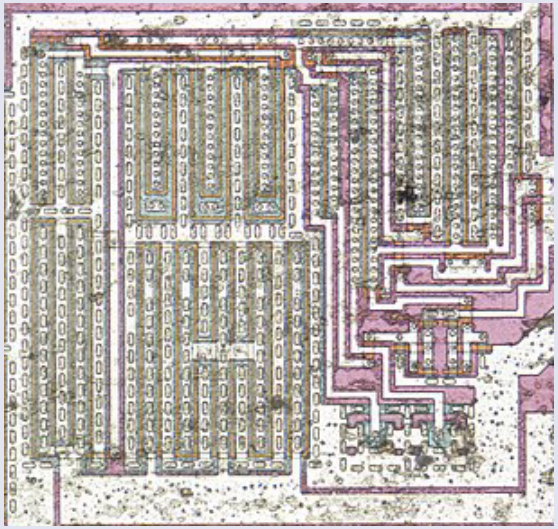
Universität Bamberg



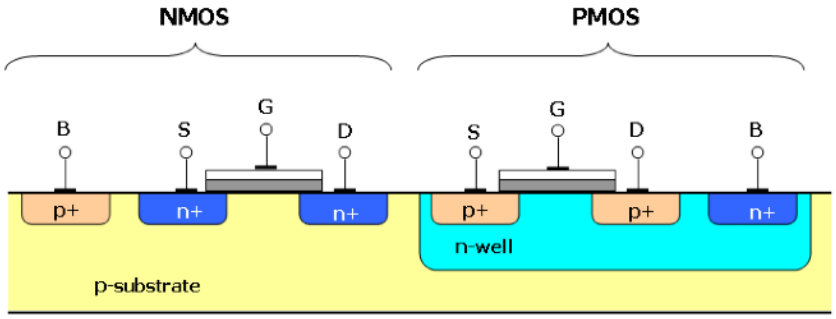
Transistor-schaltbild



Chip-layout



Siliziumchip



vertikaler Schnitt durch das Die

Siliziumchips sind dreidimensionale Strukturen!

…nee, ich will das live sehen!

Universität Bamberg

Visual 6502 Demo

https://www.visual6502.org

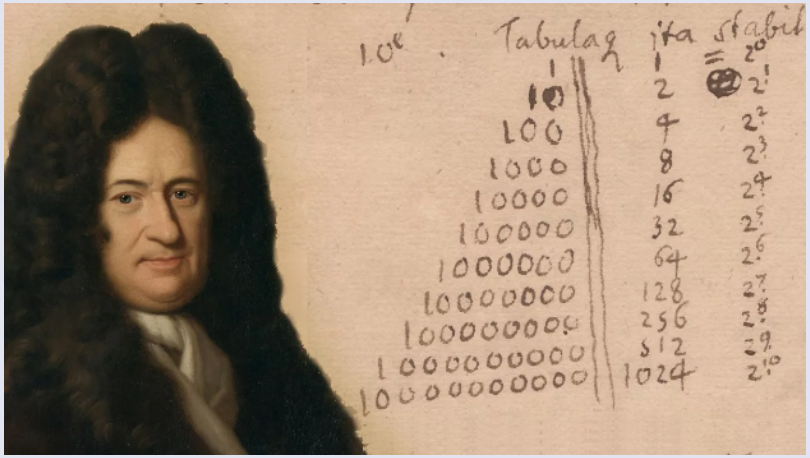
Prozessor handgemalt!

Prozessoren wurden in den 1970er Jahren von den Entwerfern **von Hand** auf transparente Folien ("rubylith") gezeichnet – jeder Transistor einzeln! – und dann fotografisch verkleinert!

Hier sind Mike Janes (links), Harry Bawcom (rechts) und Rod Orgill (sitzend) von MOS zu sehen

Photo courtesy of Phyllis Orgill, widow of Rod Orgill
https://www.team6502.org/

Butterkekse!



…wurden nach Hannovers prominentem Universalgelehrten Gottfried Wilhelm Leibniz (1646–1716) benannt!

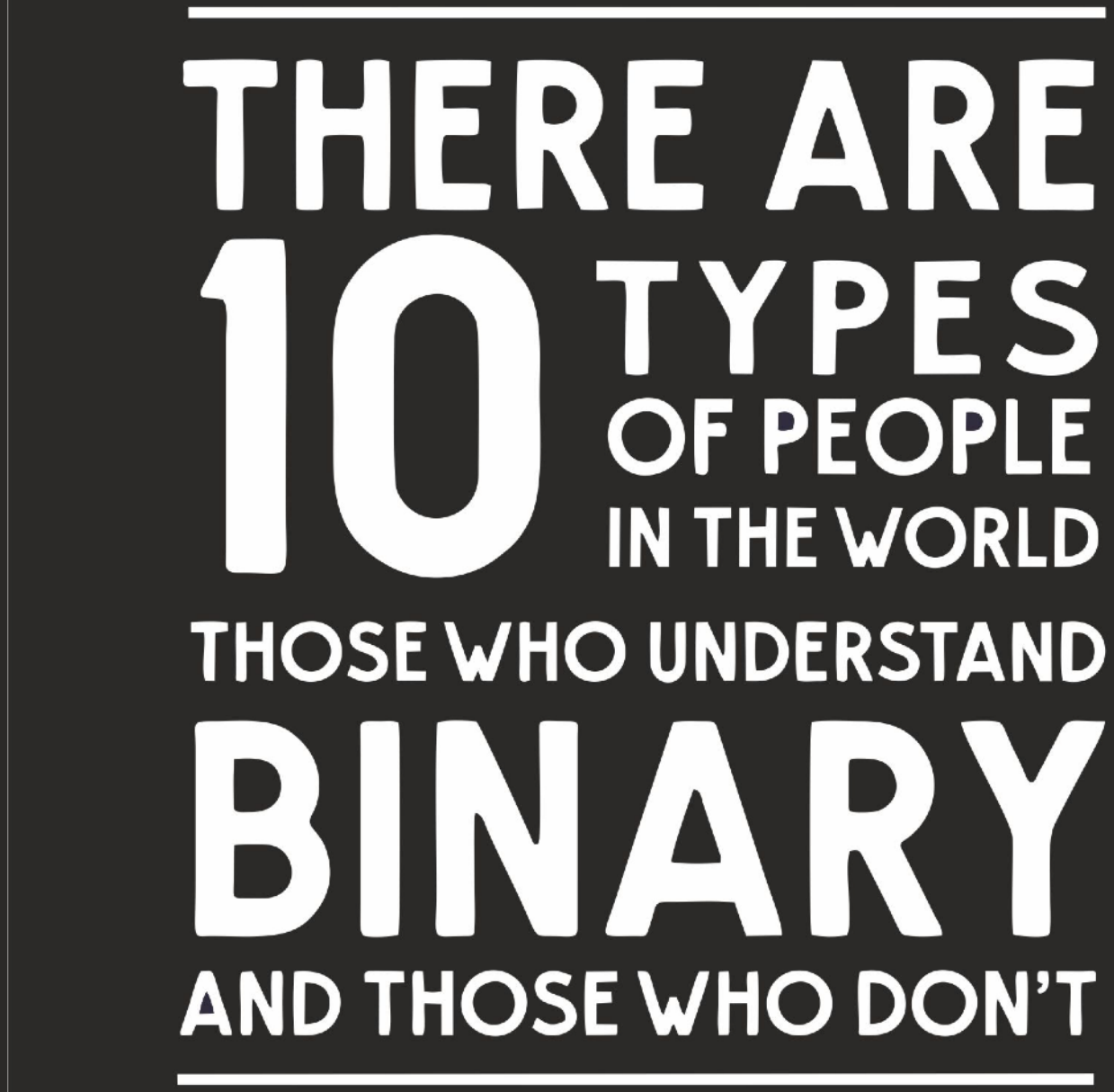
Referenzen

[1] Frank Slomka, Michael Glaß
Grundlagen der Rechnerarchitektur – Von der Schaltung zum Prozessor
Springer 2023, ISBN 978-3-658-36659-9

[2] David Harris, Sarah Harris
Digital Design and Computer Architecture, RISC-V Edition
Morgan-Kaufman 2021, ISBN 978-0128200643

[3] Raul Rojas
Konrad Zuse's Early Computers – The Quest for the Computer in Germany
Springer 2023, ISBN 978-3-031-39876-6
https://link.springer.com/book/10.1007/978-3-031-39876-6

[4] Niklaus Wirth
Digital Circuit Design for Computer Science Students
Springer 2012, ISBN 978-3-642-57780-2


THERE ARE
10 TYPES
OF PEOPLE
IN THE WORLD
THOSE WHO UNDERSTAND
BINARY
AND THOSE WHO DON'T