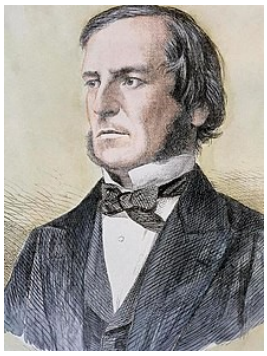


George Boole
nach Wikipedia:
"
(2.November 1815; † 8.Dezember 1864)
Irland; War ein englischer Mathematiker (Autodidakt), Logiker und Philosoph."

Allgemeine Boolesche Algebra

log. Rechenzeichen: ∧ **UND (AND)** oder kein Zeichen z.B. **a** ∧ **b = ab**
∨ **ODER (OR)** **z.B. a** ∨ **b**
¬ **Negation NEG oder** $\overline{\ }$ **z.B.** ¬**a oder** $\bar{a}$

(da sich ein Strich über einem Zeichen (bzw. über eine Formel) schlecht schreiben lässt , wird das Zeichen ¬ , geschrieben vor dem zu negierenden Zeichen, genutzt)

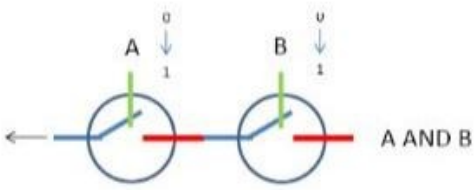
Wer erstmalig mit der Booleschen Algebra konfrontiert wird, kann durchaus Verständnis-Probleme haben, denn es gilt:

UND ≠ PLUS
∧ ≠ +

z.B.

1 + 0 = 1	1 **PLUS** 0 = 1	**aber**
1 ∧ **0 = 0**	1 **UND** 0 = 0	

Man kann sich die UND (AND) Funktion als eine Reihenschaltung von zwei Schaltern vorstellen:



Es kann nur Strom durch die Anordnung fließen, wenn:
Schalter A UND Schalter B
geschlossen sind -
das ist doch logisch!

Herr Boole hat diese logischen Funktionen systematisiert und Grundregeln zum Zusammenfügen dieser Funktionen erarbeitet.

Mit den oben genannten drei Rechenzeichen hat er folgende logische Gesetze formuliert:

Kommutativität
$a \wedge b = b \wedge a$ $\quad ab = ba$
$a \vee b = b \vee a$

Assoziativität
$(a \wedge b) \wedge c = a \wedge (b \wedge c)$ $\quad (ab)c = a(bc)$
$(a \vee b) \vee c = a \vee (b \vee c)$
$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ $\quad a(b \vee c) = ab \vee ac$
$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ $\quad a \vee bc = (a \vee b)(a \vee c)$

De Morgansche Regeln
$\neg(a \wedge b) = \neg a \vee \neg b$ $\quad \overline{ab} = \overline{a} \vee \overline{b}$
$\neg(a \vee b) = \neg a \wedge \neg b$ $\quad \overline{a \vee b} = \overline{a} \wedge \overline{b}$

Die folgenden Kürzungsregeln lassen sich bereits aus den eben genannten Gesetzen erarbeiten:

Kürzungsregeln:

$a \wedge (a \vee b) = a$	$a(a \vee b) = a$
$a \vee (a \wedge b) = a$	$a \vee ab = a$
$a \wedge 1 = a$	
$a \wedge \neg a = 0$	$a \wedge \overline{a} = 0$
$a \wedge a = a$	
$a \vee 0 = a$	
$a \vee \neg a = 1$	$a \vee \overline{a} = 1$
$a \vee a = a$	
$\neg(\neg a) = a$	$\overline{\overline{a}} = a$
$(a \vee b) \wedge (a \vee \neg b) = a$	$(a \vee b)(a \vee \overline{b}) = a$
$a \wedge (\neg a \vee b) = a \wedge b$	$a(\overline{a} \vee b) = ab$
$(a \wedge b) \vee (a \wedge \neg b) = a$	$ab \vee a\overline{b} = a$
$a \vee (\neg a \wedge b) = a \vee b$	$a \vee (\overline{a}\, b) = a \vee b$

Augustus De Morgan (* 27. Juni 1806; † 18. März 1871)
war ein englischer Mathematiker.

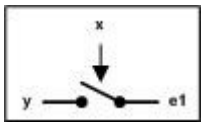
$\neg(a \wedge b) = \neg a \vee \neg b$
$\neg(a \vee b) = \neg a \wedge \neg b$

nach ihm benannte Regeln, die **De Morganschen Gesetze**

Diese besagen, dass jede Konjunktion durch eine Disjunktion ausgedrückt werden kann und umgekehrt. Sie wurden seither häufig bei mathematischen Beweisen und auch bei der Programmierung verwendet.
De Morgan gilt heute gemeinsam mit George Boole als **Begründer der formalen Logik**.

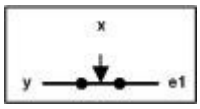
Darstellung mittels Schalter

y = x



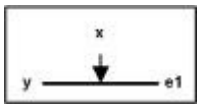
So wie sich x verändert folgt auch y, es ist ein **Folgesystem**
Wir wollen das Element als **Folger** bezeichnen.
In der Regelungstechnik hat es auch die Bezeichnung **Identität**

y = $\bar{x}$



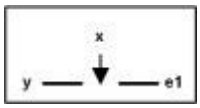
Der Ausgang von dem Teil verhält sich entgegengesetzt zum Steuersignal, man spricht bei dem Element von **Negation**.

y = 1



Egal was am Eingang x passiert, das Ergebnis ist immer 1. Es entspricht einem durchgehendem **Draht**

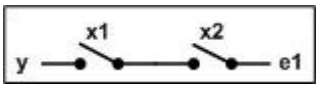
y = 0



Das Ergebnis ist immer 0. Es scheint wohl eine kaputte Verbindung zu sein, eine **Unterbrechung**

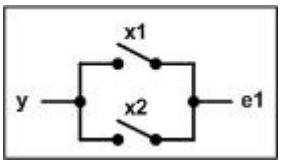
Jeweils eine spezielle Reihen- und Parallel-Schaltung wird gesondert betrachtet und auch zu den Grundelementen gezählt:

y = x1 ∧ x2



y = x1 AND x2 (das ∧ bedeutet AND (UND))
wenn x1 und x2 betätigt sind, dann ist y = e1, bei e1 = 1 folgt **y = 1**
Man bezeichnet diese als **AND-Schaltung** bzw. als **Konjunktion**.

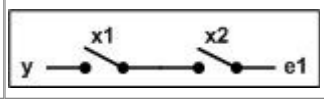
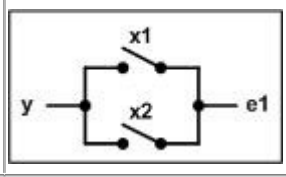
y = x1 ∨ x2



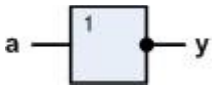
y = x1 OR x2 (das ∨ bedeutet OR (ODER))
wenn x1 oder x2 oder beide betätigt sind, dann ist y = e1, bei e1 = 1 folgt **y = 1**
Man bezeichnet diese als **OR-Schaltung** bzw. als **Disjunktion**.

Eine weitere Beschreibung wird mit der Schaltbelegungs-Tabelle erreicht:

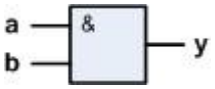
▼

AND	**y = x1 ∧ x2**		x1 x2 y 0 0 0 1 0 0 0 1 0 1 1 1	Wenn x1 und x2 geschlossen ist, dann ist y = 1 (e1 = 1)
OR	**y = x1 ∨ x2**		x1 x2 y 0 0 0 1 0 1 0 1 1 1 1 1	Wenn x1 oder x2 oder beide geschlossen sind, dann ist y = 1 (e1 = 1)

Negation – als Bauelement (Gatter): Negator

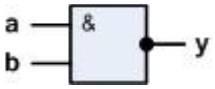


AND – als Bauelement (Gatter): AND-Gatter

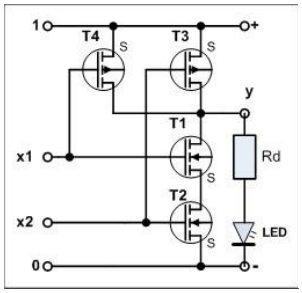


Die **N**egation von **AND** = **NAND**

¬(a ∧ b) = ¬a ∨ ¬b = $\overline{a \wedge b} = \overline{a} \vee \overline{b}$



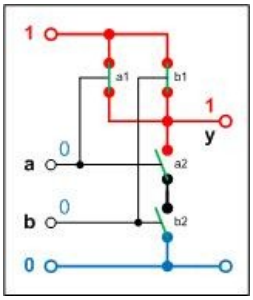
Genau für diese logische Funktion gibt es einfache technische Realisierungen, die sehr genau die logische Funktion NAND realisieren:

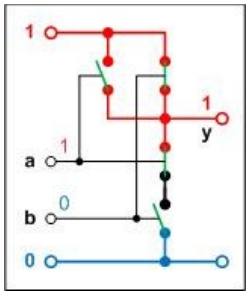


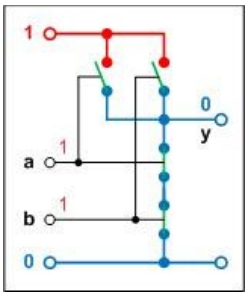
Wenn man ein gut funktionierendes NAND-Element hat, braucht man eigentlich kein weiteres Element, denn mit den Regeln der Booleschen Algebra kann man alle anderen Elemente daraus (damit) erzeugen.

Diese Elemente kann man tatsächlich als diskrete Bauelemente kaufen (**IC 4011;** 4 NANDs mit je 2 Eingängen), sie werden so auch in Rechnerschaltkreisen realisiert.

Diese Schaltung kann man sehr gut auch durch Schalter darstellen, die Funktion ist dann sehr gut verständlich:

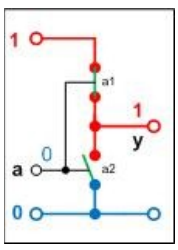


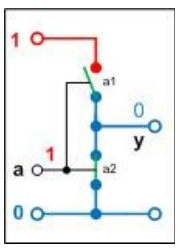




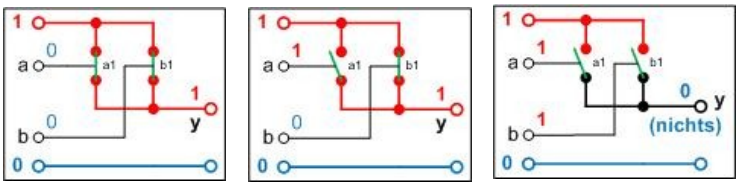
Die oberen beiden Schalter (a1/b1) sind bei der Ansteuerung 0/0 geschlossen, die unteren (a2/b2) dagegen offen. Bei dieser Ansteuerung, wie auch bei 1/0 oder 0/1, wird die 1 von oben an den Ausgang y gelegt, nur bei 1/1 erscheint am Ausgang eine 0.

Die Reduzierung der Schaltung nur auf einen Schalter (z.B. a1/a2) Realisiert einen Negator:





Das funktioniert so auch mit der Transistor-Schaltung, wird so auch angewendet
(siehe auch Digitale Schaltungen
NAND/NOR DTL - TTL - MOSFET - CMOS)

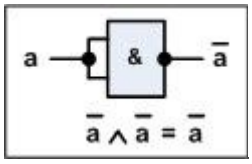


Mit Schaltern ist auch diese NAND-Variante möglich!. Ein offener Schalter stellt „Nichts“ (nicht 0) bereit. Sollte jedoch 0 beim Öffnen zugeschaltet werden
geht diese Variante nicht, dann funktioniert es nur so wie oben, bzw. die Zusammenschaltung der Schalter muss mittels Dioden zur richtigen ODER-Schaltung umfunktioniert werden!
Mit elektronischen Bauelementen geht es so aber nicht.

Negation aus NAND erzeugen
Die Boolesche Algebra bietet dazu zwei Möglichkeiten an:

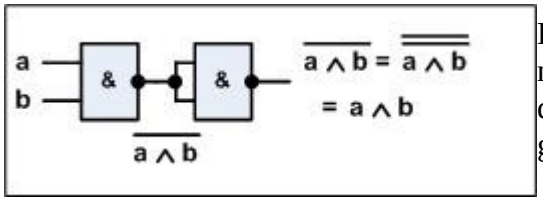
a ∧ 1 = a
a ∧ a = a

a ∧ 1 bedeutet, dass der eine Eingang auf 1 (Betriebsspannung) gelegt werden muss oder der eine Schalter ist immer zu, es ist eine Leitung (Draht). Damit hängt das Ergebnis nur noch von dem einen verbliebenen Schalter ab.
oder
a ∧ a beide Schalter bekommen das gleiche Signal, sie funktionieren immer gemeinsam und damit gleich – so wie einer!



2.Variante

AND aus NAND erzeugen

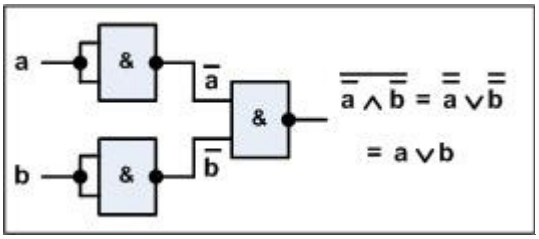


Das negierte Ausgangssignal wird noch einmal negiert (mit eben gezeigter Schaltung). Das bedeutet, dass das Ausgangssignal zwei mal negiert wird. Es gilt die Regel:

$$a = \overline{\overline{a}}$$

Die Negation fällt weg, das Ergebnis ist AND.

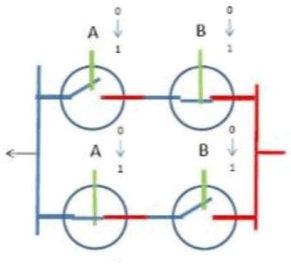
OR aus NAND erzeugen



Nach den **Morganschen Regeln** werden durch die Negation beide Eingangssignale negiert, aber mit getauschtem Rechenzeichen verknüpft:

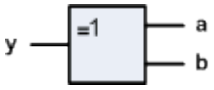
$$\overline{a \vee b} = \bar{a} \wedge \bar{b}$$

Negiert man also zuvor die Eingangssignale, so werden beide Signale noch einmal negiert und das fällt wieder auf Grund der Regeln weg (Negation der Negation → kann entfallen), wir haben ein klassisches OR!
So etwa kann man alle weiteren gebrauchten Funktionen mit der Booleschen Algebra erzeugen und auch als elektronisches Bauelement aufbauen.
XOR realisiert die **Antivalenz**,
d.b. wenn der Eingang **a ≠ b** ist, dann ist **y = 1** , also **a = 1 und b = 0 oder a = 0 und b = 1**.
Das kann man als Formel aufschreiben:



$$y = a\bar{b} \vee \bar{a}b$$

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

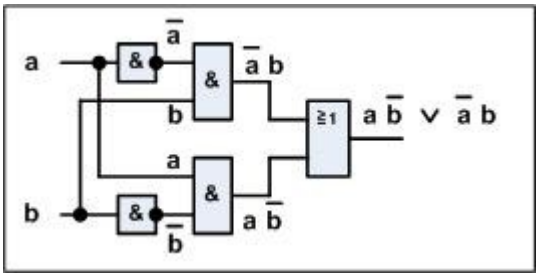


Das braucht man, um z.B. **aus logischen Funktionen arithmetische Funktionen herzustellen**. Und genau das hat **Leibniz** bereits vor mehr als 300 Jahren festgestellt, jedoch nicht technisch realisiert.

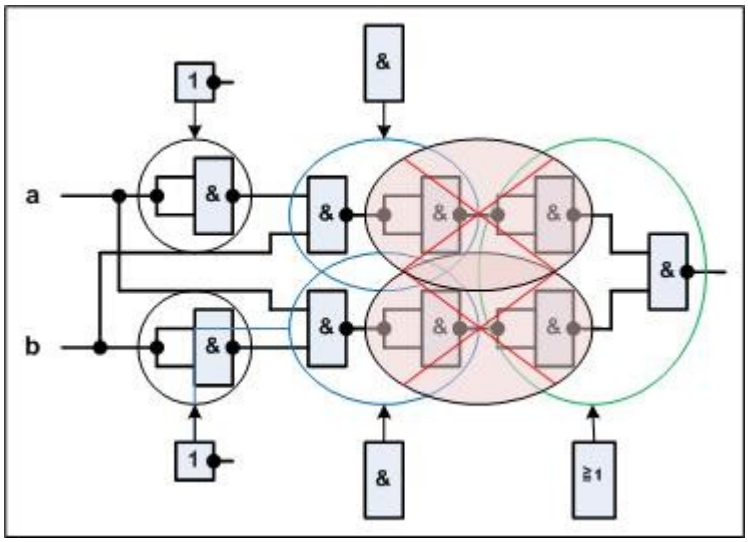
Auch dieses Gatter (logische Schaltfunktion) kann man aus den bisher genannten logischen Schaltungen herstellen. Basis ist die Schaltfunktion

$$y = a\bar{b} \vee \bar{a}b$$

Durch probieren kann man eine Lösung finden, besser ist es, diese mit den bekannten Mitteln zu konstruieren.
Dazu werden die in der Schaltfunktion genannten logischen Funktionen entsprechend der Formel zusammengeschaltet. Wir brauchen **2 Negatoren, 2 UND-Elemente und eine ODER-Funktion**. Das sieht dann so aus:

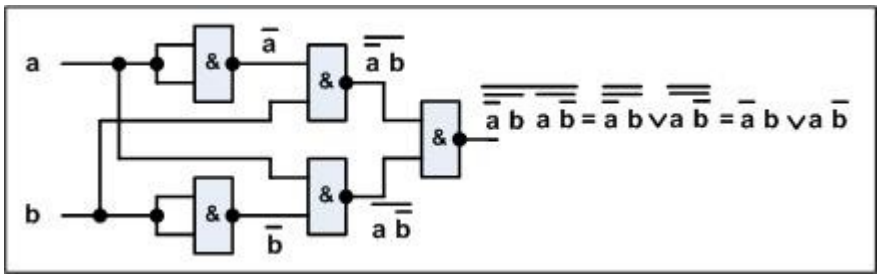


Im nächsten Schritt setzen wir die oben gezeigten Realisierungen von NEG, AND und OR aus NAND-Elementen in der Schaltung ein:



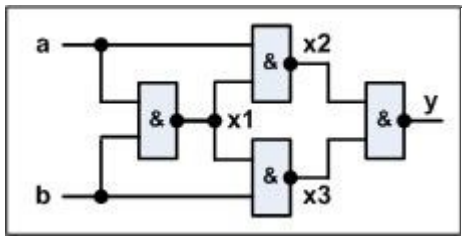
Die **schwarze Ellipse** realisiert die **NEG-Fkt**., die **blaue die AND-Fkt**. und die **grüne** die **OR-Fkt**.

Sofort sieht man dass der Negation vom Ausgang des AND wieder eine Negation als Eingang vom OR folgt. Auch hier in der Hardware gelten die Gesetze der Booleschen Algebra – **die Negation der Negation hebt sich auf** – können beide zusammen **weggelassen** werden (rote Ellipse)!
Damit wird die Schaltung wieder kleiner:



Es werden genau so viele NAND-Gatter wie beim Original verschiedene Gatter gebraucht.
Daraus kann man nun ein neues XOR-Gatter (Antivalenz) definieren.

Aber - es gibt nicht „**die**“ Schaltung zur Realisierung einer bestimmten Funktion. Für **XOR** findet man auch folgende Schaltung:



Ob die nun auch richtig funktioniert, kann man sofort nicht sagen?
Es gibt aber Möglichkeiten, das heraus zu bekommen:
- Schaltbelegungs-Tabelle
- Schaltfunktion berechnen
- ausprobieren (echt oder Simulator)

Die Schaltbelegungs-Tabelle

a	b	x1	x2	x3	y
0	0	1	1	1	0
1	1	0	1	1	0
1	0	1	0	1	1
0	1	1	1	0	1

Um die Funktion überprüfen zu können, wurden Zwischen-Ergebnis-Punkte eingeführt x1, x2 und x3.
z.B. x1= (a UND b) negiert = 0 UND 0 = 0 negiert 1
Danach kann man x2 und x3 berechnen,
z.B. x2 = (a UND x1) negiert
So entstehen die Ergebnisse für y.

Schaltfunktion berechnen

$$x1 = \overline{a\,b}$$

$$x2 = \overline{a\;\overline{a\,b}}$$

$$x3 = \overline{b\;\overline{a\,b}}$$

$$y = \overline{a\;\overline{a\,b}}\;\wedge\;\overline{b\;\overline{a\,b}} = \overline{a\;\overline{a\,b}} \vee \overline{b\;\overline{a\,b}} = a\;\overline{\overline{a\,b}} \vee b\;\overline{\overline{a\,b}}$$

$$= a(\bar{a} \vee \bar{b}) \vee b(\bar{a} \vee \bar{b}) = a\,\bar{a} \vee a\,\bar{b} \vee b\,\bar{a} \vee b\,\bar{b} = 0 \vee a\,\bar{b} \vee b\,\bar{a} \vee 0$$

$$y = a\,\bar{b} \vee b\,\bar{a}$$

Das geht am Besten auch über die Zwischenpunkte (Zwischenergebnisse), jetzt aber nicht mit Werten, sondern allgemein mit den Booleschen Gesetzen.

x1 = (a UND b) negiert	x1 = ¬(ab)
x2 = (a UND x1) negiert	x2 = (a UND (a UND b)negiert)negiert = ¬(a¬(ab))
x3 = (b UND x1) negiert	x3 = (b UND (a UND b)negiert)negiert = ¬(b¬(ab))
y = (x2 UND x3) negiert	**y = ¬(¬(a¬(ab))∧¬(b¬(ab)))**

Nun muss man die zur Vereinfachung die Gesetze anwenden und löst von außen nach innen die Negationen auf und kommt tatsächlich auf:

y = a¬b∨b¬a

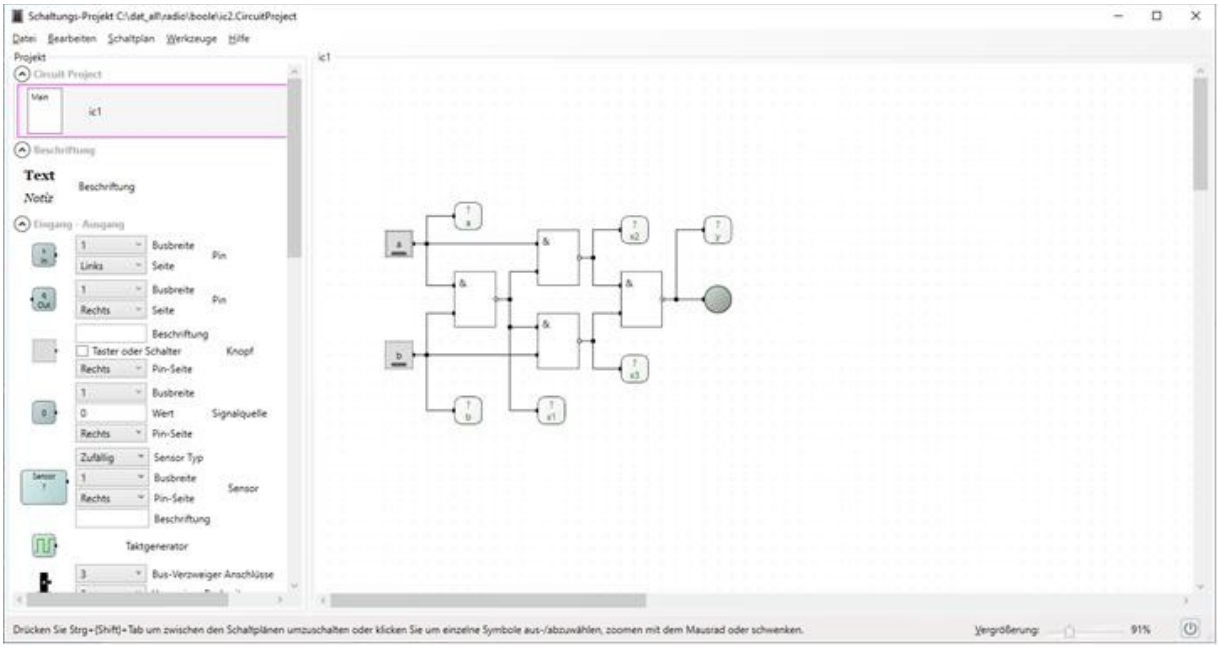
der Schaltfunktion von XOR (Antivalenz)

Simulator LogicCircuit

(https://logiccircuit.de.softonic.com/)
„LogicCircuit – is educational software for designing and simulating digital circuits.“ [http://logiccircuit.org]

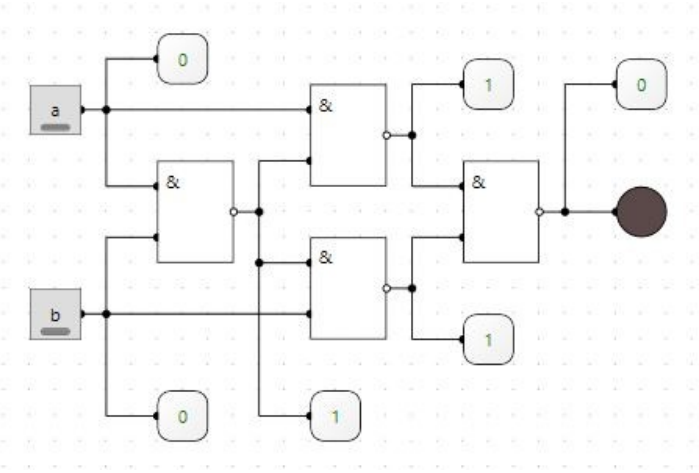
LogicCircuit – ist freie, Open-Source-Bildungssoftware zum Entwerfen und Simulieren digitaler Logikschaltungen. Intuitive grafische Benutzeroberfläche, ermöglicht es Ihnen, eine uneingeschränkte Schaltungshierarchie mit Multi-Bit-Bussen zu erstellen, Debug-Schaltungen Verhalten mit Oszilloskop und navigieren Laufschaltung Hierarchie.

Nach Installation und Start der Software erhält man beispielsweise folgende Bedien-Oberfläche:

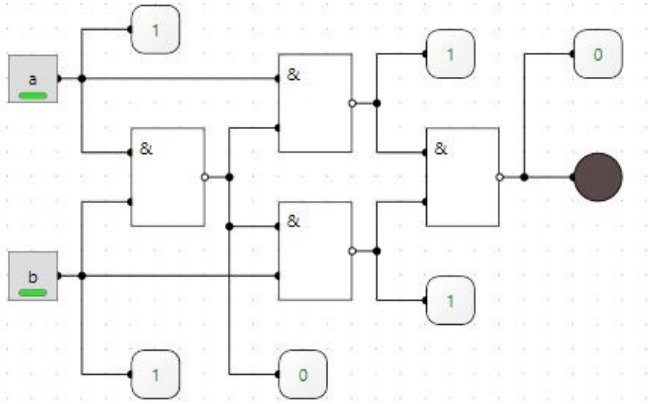


Links gibt es ein Menü mit sehr vielen Teilen der Digitaltechnik, meist kann man sie weiter konfigurieren, z.B. Anzahl der Eingänge am IC. Rechts gibt es einen Arbeitsbereich, in dem die

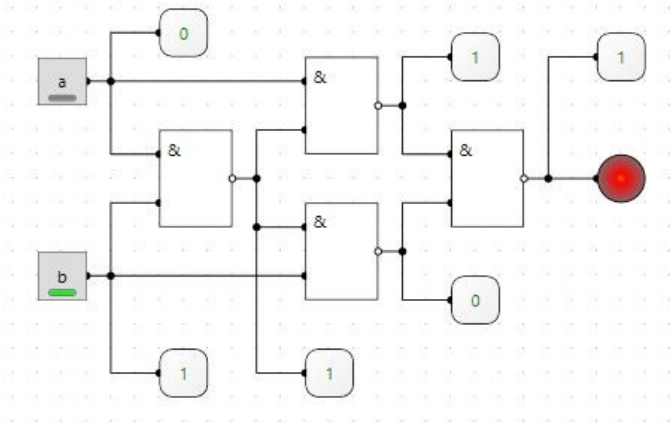
Schaltung zu generieren ist, das ist die oben zuletzt gezeigte Schaltung. Die Punkte x1, x2, x3 und y wurden als Messpunkte festgelegt, weiterhin auch die Eingaben a und b, die aber selbst bereits über eine Anzeige verfügen (grün – 1).
Unten rechts findet man das Symbol:
Damit wird die Simulation gestartet ⏻ und auch wieder beendet. Das sind dann die Ergebnisse:



An beiden Eingängen der Schaltung liegt der Wert 0.
Das Ergebnis wird durch eine LED (schwarzer gefüllter Kreis) nach dem letzten NAND angezeigt.
Die LED zeigt dunkle Farbe, also ist das Ergebnis 0. Das ist richtig, da XOR für gleiche Eingangssignale 0 zeigen soll.



Auch bei den Eingangssignalen 1 und 1 leuchtet die LED nicht – das ist auch richtig!



Anders dagegen bei ungleichen Eingangssignalen, die LED leuchtet **rot**, am Ausgang wird eine 1 bereitgestellt.
Die Schaltung funktioniert!

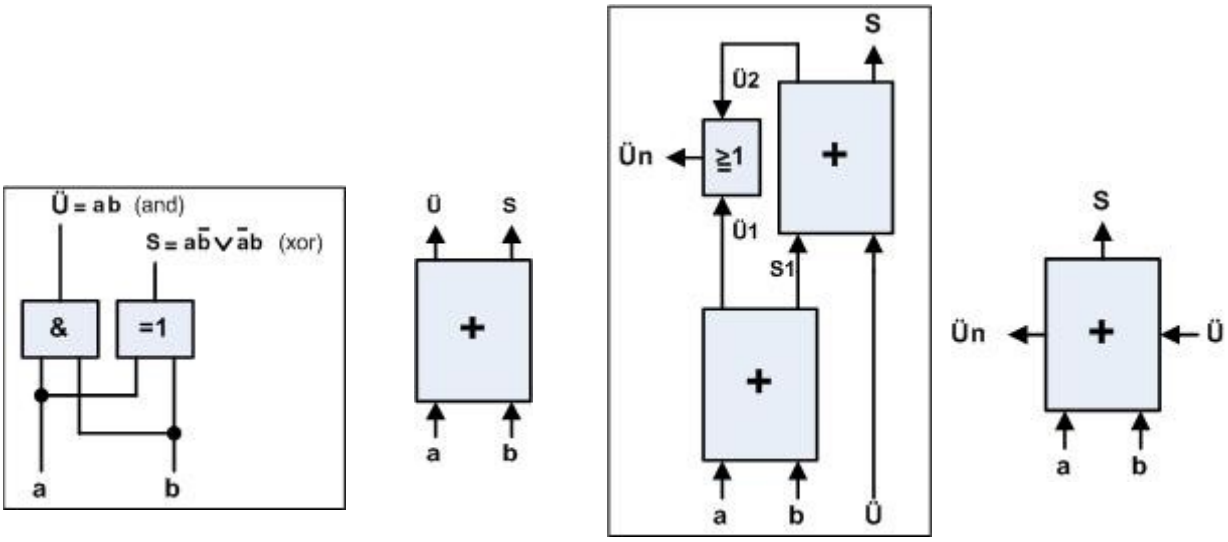
Im Folgenden wird die arithmetische Funktion **PLUS** mittels logischer Bauelemente erzeugt.
Es wird eine Bit-Stelle addiert
Addierer für eine Bit-Stelle

	Ü	b	a	S	Ün
	0	0	0	0	0
S1 →	0	0	1	1	0
	0	1	0	1	0
Ü1 →	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

Diese Schaltfunktionen kann man aus der Tebelle ermitteln, die Summe S → XOR, Übertrag Ü → AND

$$S = \bar{Ü}\,\bar{b}\,a \vee \bar{Ü}\,b\,\bar{a} \vee Ü\,\bar{b}\,\bar{a} \vee Ü\,b\,a$$

$$Ün = b\,a \vee Ü\,b \vee Ü\,a$$



Die arithmetische Operation Addition (PLUS; +) wird aus den logischen Funktionen AND, XOR und OR realisiert.
(weitere Informationen zur Erstellung eines **Addierers**)

" ... **Leibniz** ... Er entdeckte, dass sich Rechenprozesse viel einfacher mit einer binären Zahlencodierung durchführen lassen, und ferner, dass sich mittels des binären Zahlencodes die **Prinzipien der Arithmetik mit den Prinzipien der Logik verknüpfen** lassen"

Ebenfalls wie bei den logischen Funktionen lassen sich aus der **arithmetischen Funktion PLUS** alle weiteren arithmetischen realisieren!
Für die ganzzahlige Multiplikation braucht man mehrfach die Addition, für die Division die Subtraktion und die Addition. Was wir also noch brauchen ist die **Subtraktion**.
Die Mathematik bietet für dieses Problem schon eine Lösung an. Pascal hat wohl beim Bau seiner Rechenmaschine 1642 über die Subtraktion mittels Komplementbildung nachgedacht.
Basis dazu sind **negative Zahlen**, das sind **nicht** die positiven Zahlen mit Minuszeichen, sondern andere Zahlen, z.B.

5 – 3 daraus wird **5 + (-3)**

eine negative Zahl wird addiert, die müssen wir erzeugen.

1. Schritt:
Von 3 bis zum maximalen Stellenwert 9 = 6 - Teil wird **9-er Komplement** genannt

2. Schritt:
wir müssen eigentlich bis 10 rechnen, die gibt es aber nicht in der Stelle, die kann man aber erzeugen:
6 + 1 = 7 - Teil wird **10-er Komplement** genannt

7 ist die negative Zahl von 3!

Diese Zahl muss nun zu 5 addiert werden
5 + 7 = 1**2**
Da wir nur eine Stelle darstellen können, nutzen wir die Einer-Stelle also, **2 als Ergebnis**!
Die 1 ist der Übertrag in die nächste Stelle und zeigt mit 1 an, das das Ergebnis positiv ist, eine 0 würde auf negativ hinweisen, man müsste wieder das 10-er Komplement bilden.

Das Problem beim dezimalen Zahlensystem ist die Bildung des 9-er Komplements!

Der gleiche Mechanismus soll bei allen Stellenwert-Zahlensystemen funktionieren, als auch beim Dualsystem:
$5 - 3 = 2_{(10)}$ **$0101 - 0011 = 0010_{(2)}$**
Bildung des **1-er Komplement** (Differenz bis zum maximalen Wert).
Der maximale Wert ist: 1111

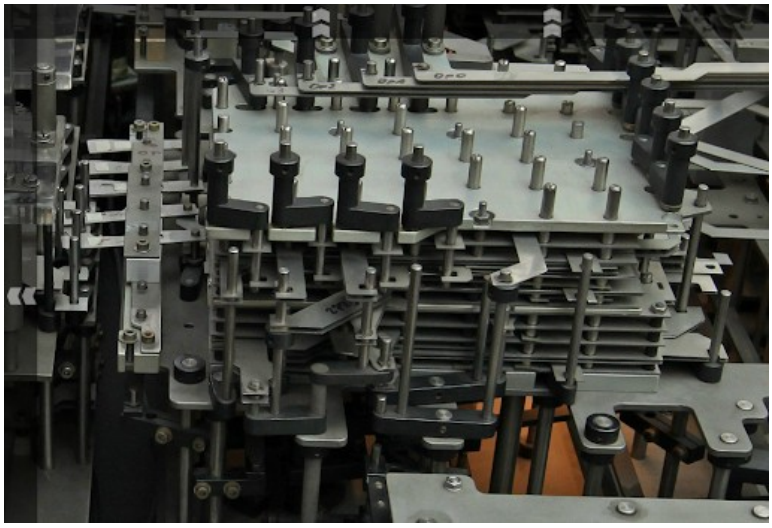
```
  1111
- 0011
------
  1100   ← 1-er Komplement

  1100
+ 0001
------
  1101   ←  2-er Komplement  - negative Zahl 3

   5   0101
+(-3)  1101
       ------
      10010
      ↑↑
 Übertrag|Ergebnis
```

Wie zu erwarten war, ist das Ergebnis **$0010_{(2)} = 2_{(10)}$**
(Übertrag: 1 – positive Zahl; 0 – negative Zahl, Erg. muss noch umgewandelt werden)

Der Vorteil im Dualsystem liegt bei der Bildung des 1-er Komplements (der Differenzbildung), es ist einfach nur die Negation der positiven Zahl, das lässt sich wieder leicht mit einem Negator erzeugen. Die folgende Addition ist ja schon bekannt.
Das ganze kann man als Subtraktions-**Programm** realisieren (Software) oder auch durch logische Bauelemente (Hardware – **Arithmetik-Prozessor**)!

So weit hat es **Herr Boole** nicht realisiert, es fehlten ja die technischen Mittel, aber durch ihn wurde die theoretische Basis, die von **Leibniz** angedacht war, erarbeitet.

Letztere **Aussage ist aber nicht ganz richtig**, denn **Herr Zuse** hat mit seiner programmgesteuerten Rechenmaschine Z1 gezeigt, dass auch mechanisch eine Rechenmaschine im Dualsystem realisierbar ist.

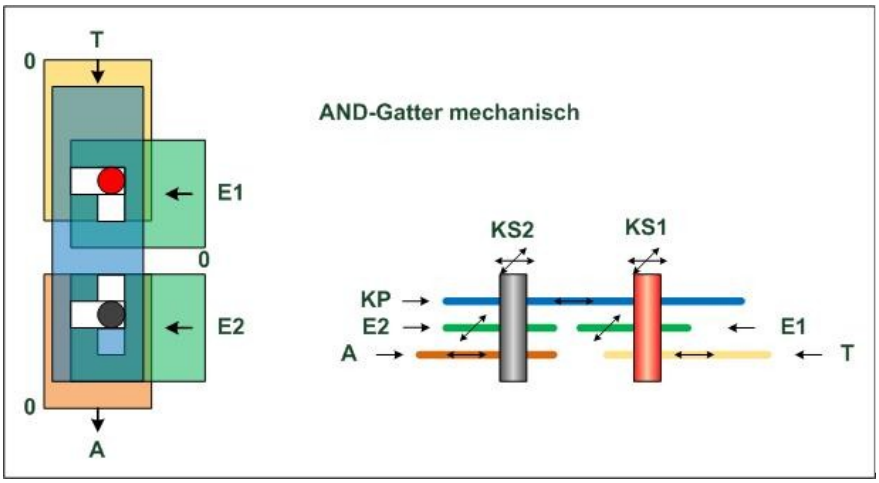


Die Rechnerei, wie auch die Speicherung der Daten erfolgt rein mechanisch.
Die Eingabe der Daten erfolgt durch einen Lochstreifen, die Abarbeitung der Befehle wird durch einen Kurbelantrieb, bzw. durch einen Elektromotor, realisiert. Dazu wird wie in einem modernen Rechner ein Mikroprogramm abgearbeitet

Die erste Maschine wurde durch den Krieg vernichtet. Der Nachbau steht nun auch schon sehr lange im Museum, zwar durch Glas und Plastik geschützt, aber die Materialien der Maschine sind gealtert, so dass Führungen der Bleche nicht mehr die Richtige Position garantieren, so dass dann bei Bewegung mit ausreichender Kraft diese verbiegen und die gesamte Maschine nicht mehr funktioniert.

Z1-Addierermodelle: https://www2.hs-fulda.de/~grams/mathehilft/Zuse/Z1-Addierermodelle.pdf

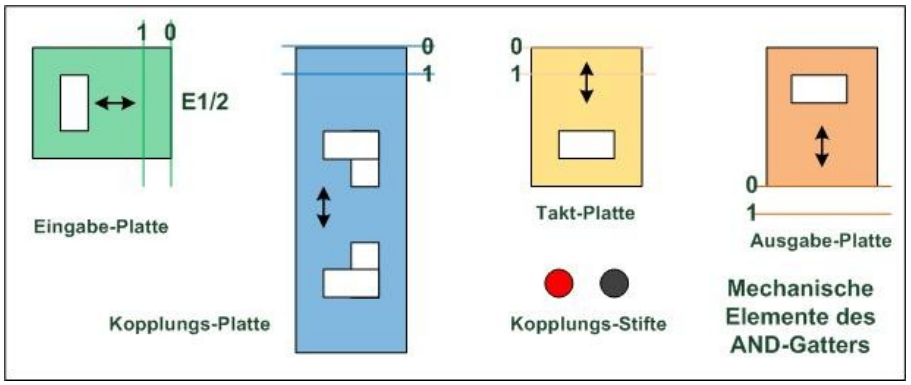
Es erfordert schon eine Menge Vorstellungskraft, die Funktion der Maschine zu verstehen.
Es wird alles auf die Grundelemente der Boolesche Algebra zurück geführt, folglich muss es möglich sein diese Elemente zu separieren.
Ich möchte deshalb kurz ein mechanisches **AND-Gatter** vorstellen und auch eine Simulation zeigen.



Es besteht aus drei Ebenen:
Takt- und Ausgabeebene, Eingabe-Ebene und einer **Kopplungs-Ebene**.

Die Kopplungsebene realisiert die Funktion des Gatters.
Wichtig sind auch die **Kopplungsstifte**, sie verbinden die einzelnen Elemente der Ebenen und bewegen sich in speziellen Ausschnitten in den Elementen. Das Auslösen der Bewegung erfolgt

durch das Takt-Element. Die Stifte sind völlig frei, natürlich in den Ausschnitten, beweglich und verändern zum Teil die Position der Elemente in den Ebenen (Ausgabeelement, Kopplungselement).
Darstellung der Elemente:



Die Eingabeplatten bewegen sich waagerecht, die anderen Platten nur senkrecht. Die Kopplungsstifte hingegen können sich sowohl waagerecht wie senkrecht bewegen und übertragen ihre Bewegung auf alle anderen Platten.

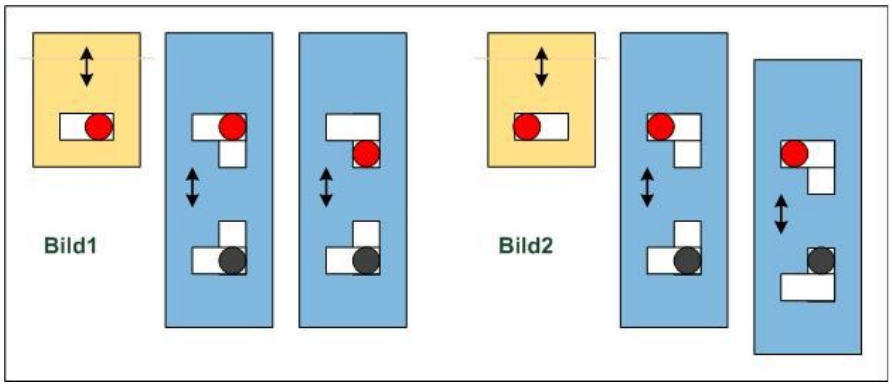
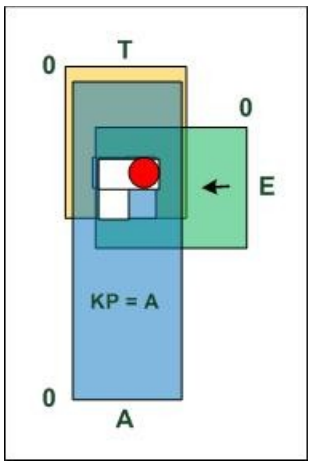


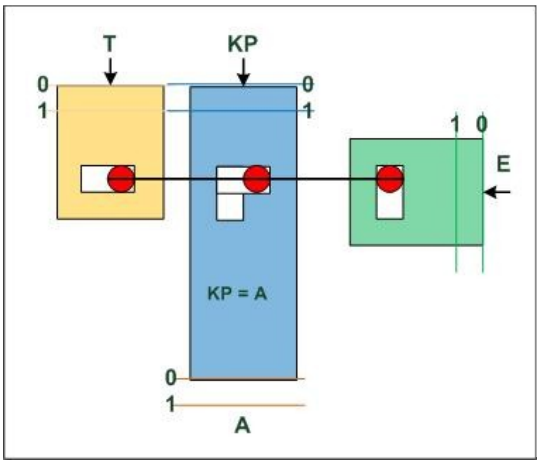
Bild1:
Die Eingabeplatte steht auf 0, somit steht der rote Kopplungsstift rechts in der Kopplungsplatte (auch der schwarze).Wird nun der Takt betätigt (auf 1 nach unten geschoben) rutscht der rote Kopplungsstift im Schlitz der Kopplungsplatte nach unten – die Kopplungsplatte wird nicht bewegt! Der schwarze Stift bleibt deshalb an seiner alten Position.

Bild2:
Die Eingabeplatte steht auf 1, also wurde der rote Stift nach links geschoben. Jetzt bewegt der Takt 1 die Kopplungsplatte nach unten, da es an dieser Stelle keinen Schlitz nach unten gibt. Aber der schwarze Stift kann in dem Schlitz nach oben rutschen – er bewegt sich nicht von der Stelle, d.h. die Ausgabe bleibt unverändert.
Nur wenn die Eingabe2 den schwarzen Stift nach links geschoben hatte, würde auch der schwarze Stift nach unten geschoben (andere Ausgabe).
Genau so soll ein AND-Gatter funktionieren, nur wenn beide Eingänge auf 1 stehen wird auch die Ausgabe auf 1 gesetzt!

In ähnlicher Weise kann auch ein Negator gestaltet werden.

Es gibt natürlich nur einen Eingang, eine Takt und Kopplungsplatte, die in diesem Fall zugleich Ausgabe ist:

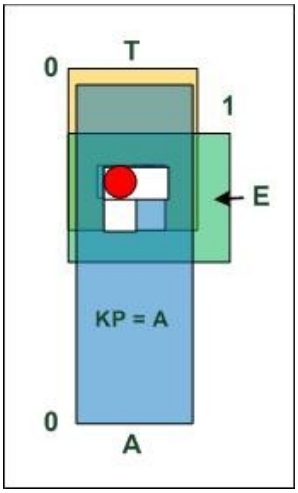


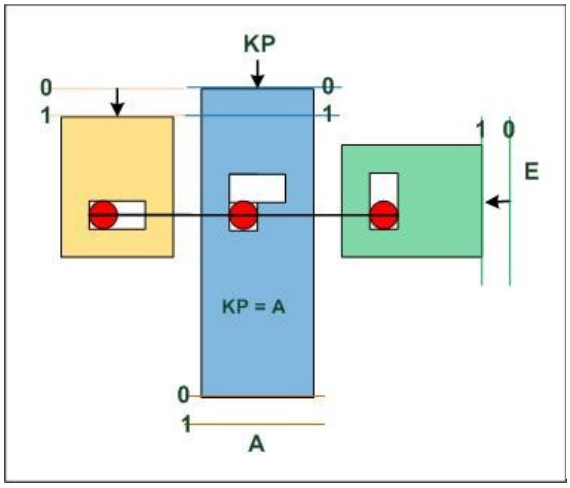


Ist der Eingang 0 steht der Kopplungsstift rechts in dem Ausschnitt der Kopplungsplatte.
Wird nun der Takt auf 1 gesetzt, wird auch die Kopplungsplatte über den Kopplungsstift nach unten geschoben, da keine andere Möglichkeit besteht.
Am Ausgang erscheint eine 1, das ist richtig weil für den Negator gilt:

A = ¬E (1 = ¬0)

Anders hingegen die Situation E = 1:





Der Kopplungsstift steht nun links.
Wird nun der Takt auf 1 gesetzt, rutscht der Stift in den Schlitz in der Kopplungsplatte nach unten, die Platte bewegt sich nicht, am Ausgang bleibt 0. Das ist richtig, denn es gilt:

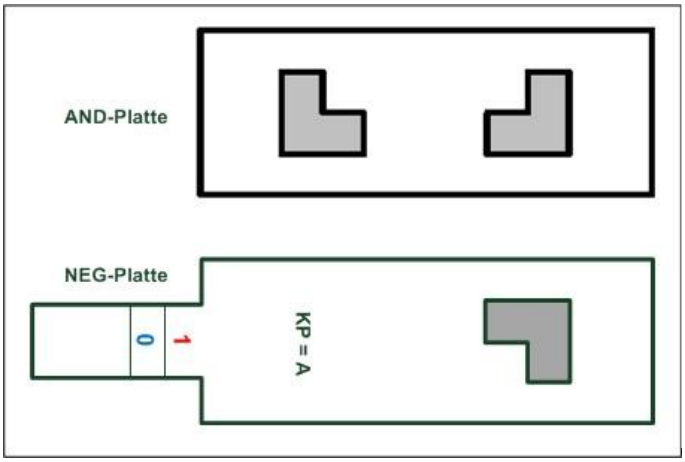
A = ¬E (0 = ¬1)

Im Internet kann man zu der Problematik mehrere Beiträge finden, sehr interessant ist der Betrag:

Bastelbogen für ein mechanisches Schaltglied-modell

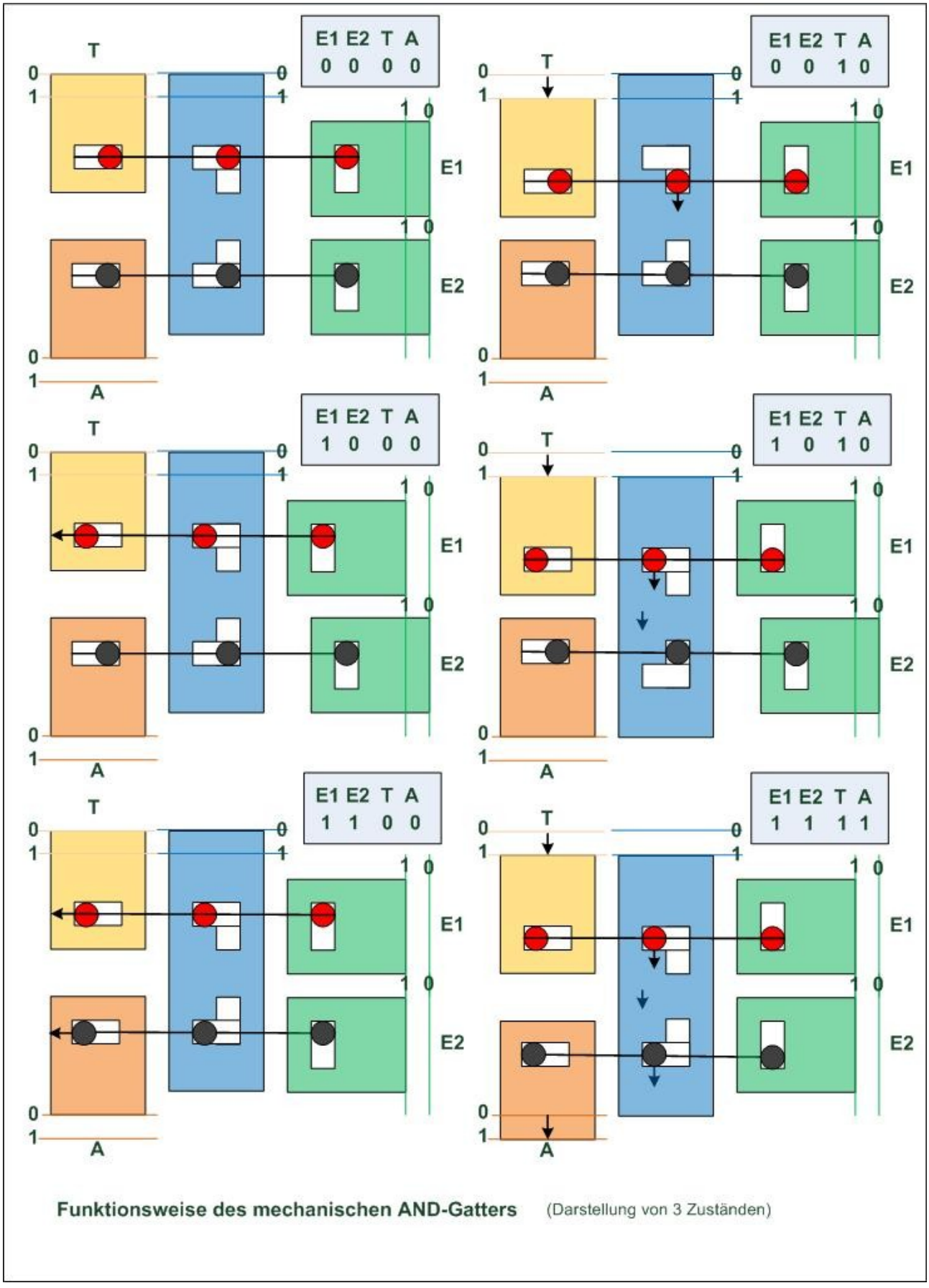
Timm Grams, Fulda, 17. März 2010 (aktualisiert: 26.07.10)

Neben einer kleinen Beschreibung wird für ein **AND-Gatter** eine Möglichkeit angeboten, dieses Teil aus Papier zu gestalten. Ich habe es nachgebaut und es funktioniert gut.
Weiterhin habe ich auch einen Negator mit diesem Teil realisiert, es wird nur ein Eingang genutzt und die Kopplungsplatte so gestaltet, dass sie die Funktion NEG realisiert und zugleich das Ergebnis ausgibt:



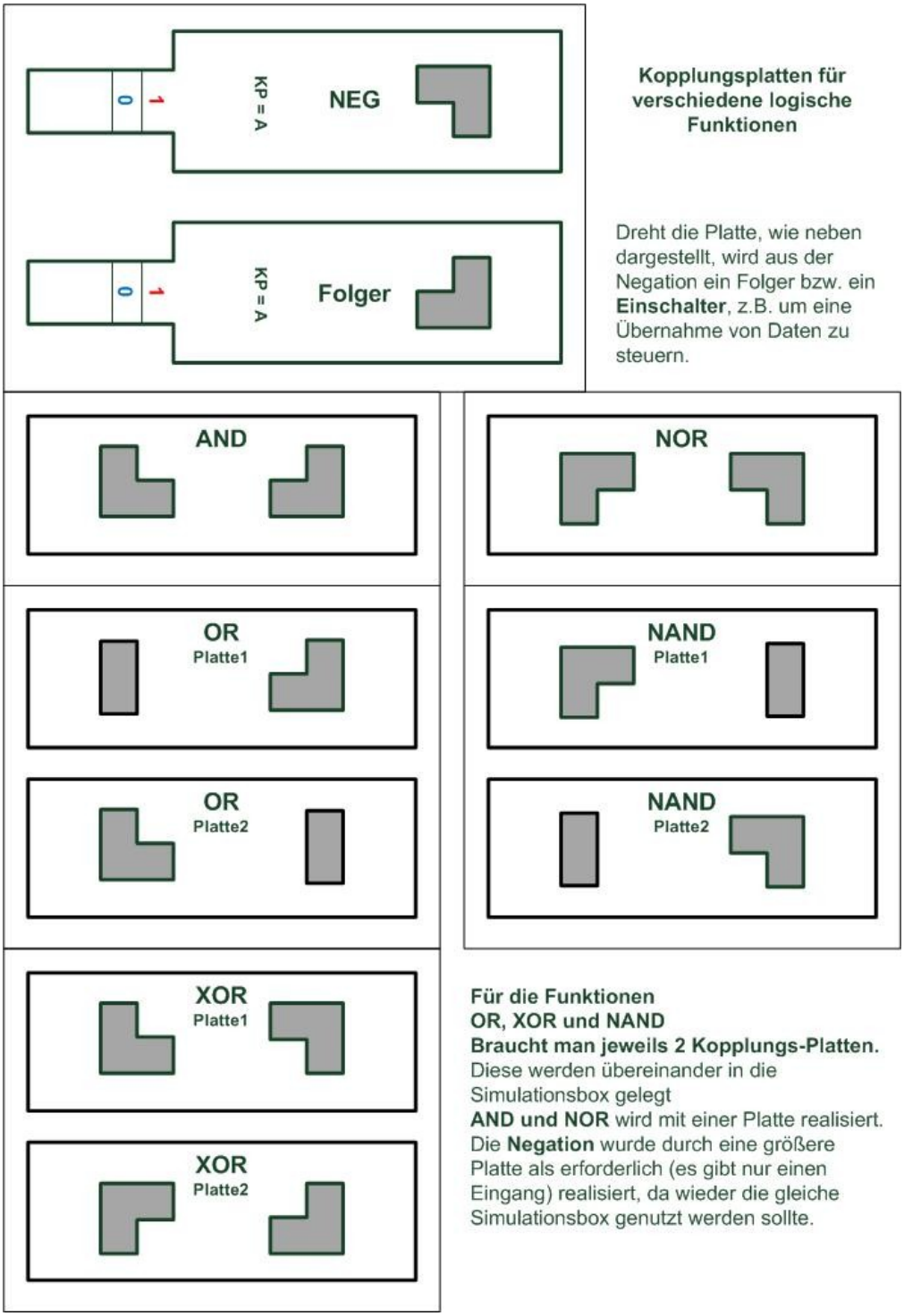
Die Grundplatte kann man natürlich auch halbieren, ich wollte jedoch das gleiche Teil benutzen und nur die Kopplungsplatte austauschen.

Ich habe diesen **Bastelbogen** ebenfalls hier per Link verbunden, **es ist nicht mein Eigentum, es bleibt natürlich Eigentum von Herrn Grams!**



Das sind so etwa die logischen Schalt-Elemente im Zuse-Rechner Z1.
(hier gibt es ein Beispiel der **AND-Simulation**)

Weitere logisch, mechanische Grundelemente

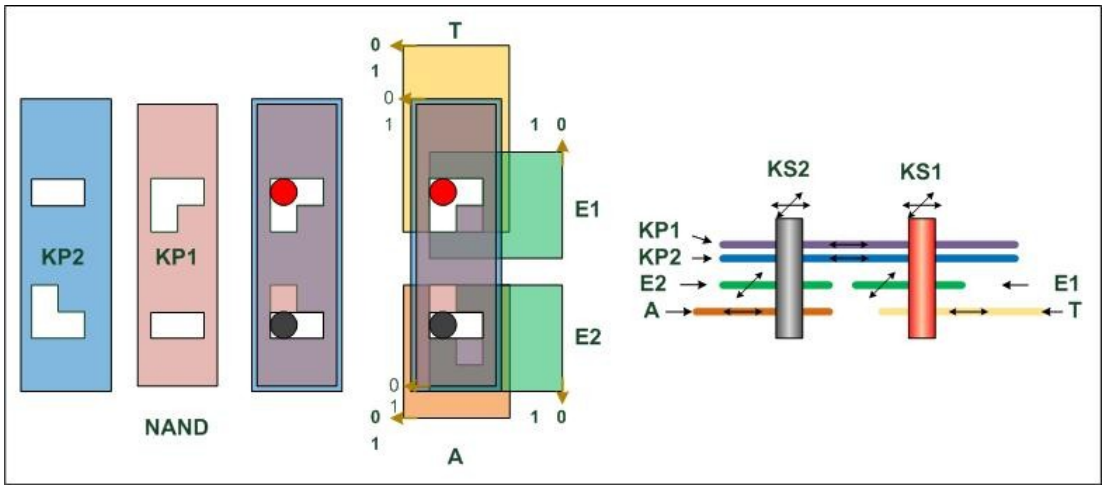


Kopplungsplatten für verschiedene logische Funktionen

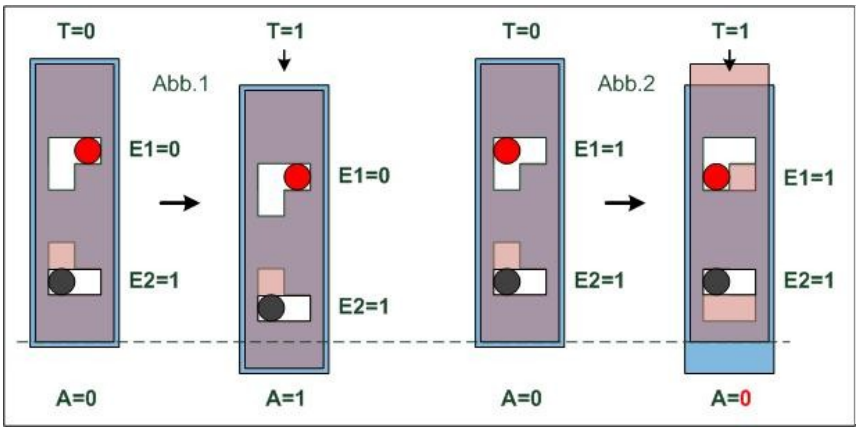
Dreht die Platte, wie neben dargestellt, wird aus der Negation ein Folger bzw. ein **Einschalter**, z.B. um eine Übernahme von Daten zu steuern.

Für die Funktionen
OR, XOR und NAND
Braucht man jeweils 2 Kopplungs-Platten.
Diese werden übereinander in die Simulationsbox gelegt
AND und NOR wird mit einer Platte realisiert.
Die **Negation** wurde durch eine größere Platte als erforderlich (es gibt nur einen Eingang) realisiert, da wieder die gleiche Simulationsbox genutzt werden sollte.

Unter Einsatz dieser Koppelplatten sollte es nun möglich sein alle notwendigen mechanische logische Elemente zu schaffen, z.B. ein **NAND**-Element



Es werden die oben gezeigten Kopplungsplatten KP1 und KP2 gebraucht. Die Reihenfolge der Platten spielt keine Rolle, jedoch müssen sie wie dargestellt, zu den Eingabeplatten platziert werden. Die Kopplung aller Platten erfolgt ohnehin nur über die Kopplungsstifte.

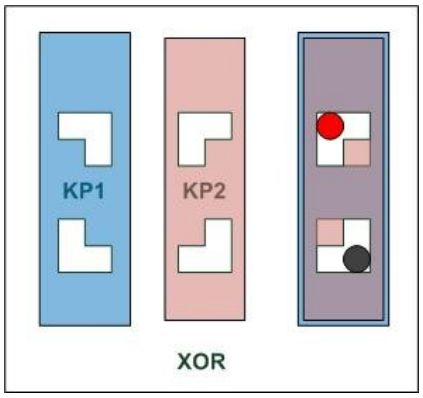


Das nebenstehende Bild zeigt die Kopplungsplatten und die Kopplungsstifte. Der schwarze Stift ist zuständig für die Ausgabe.
Die Abb.1 zeigt die Situation, dass nur ein Stift nach links geschoben ist. Bei T=1 wird der Stift herausgeschoben, A ist 1. Das trifft auch für die anderen beiden Fälle (E1=0, E2=1; E1=0, E2=0) zu. Nur für den in Abb.2 dargestellten Fall (E1=1, E2=1 und T=1) wird der schwarze Stift nicht bewegt, der Ausgang ist 0. Das entspricht den Regeln für **NAND**:

E1	**E2**	**A**
0	0	1
1	0	1
0	1	1
1	1	**0**

(NAND-Simulation **zwei Fälle)**

In ähnlicher Weise lässt sich auch die logische Funktion **XOR** (bzw. Antivalenz) realisieren. Oben sind bereits die Kopplungsplatten, es sind zwei, dargestellt:



Beide werden übereinander gelegt, die Reihenfolge spielt keine Rolle.

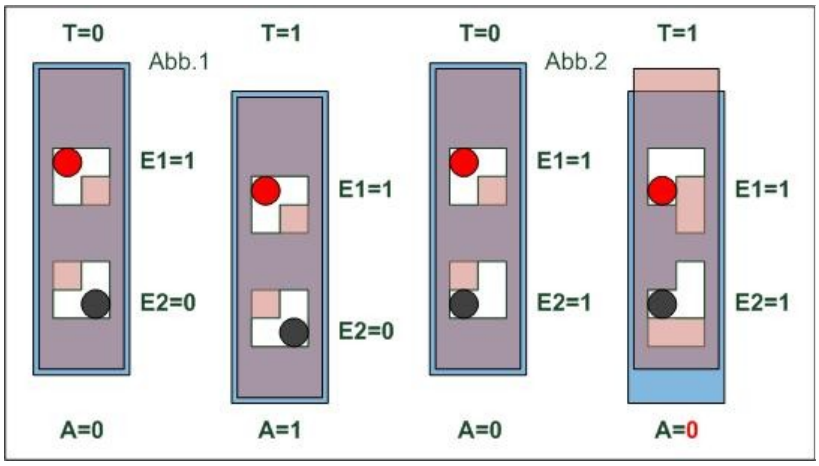
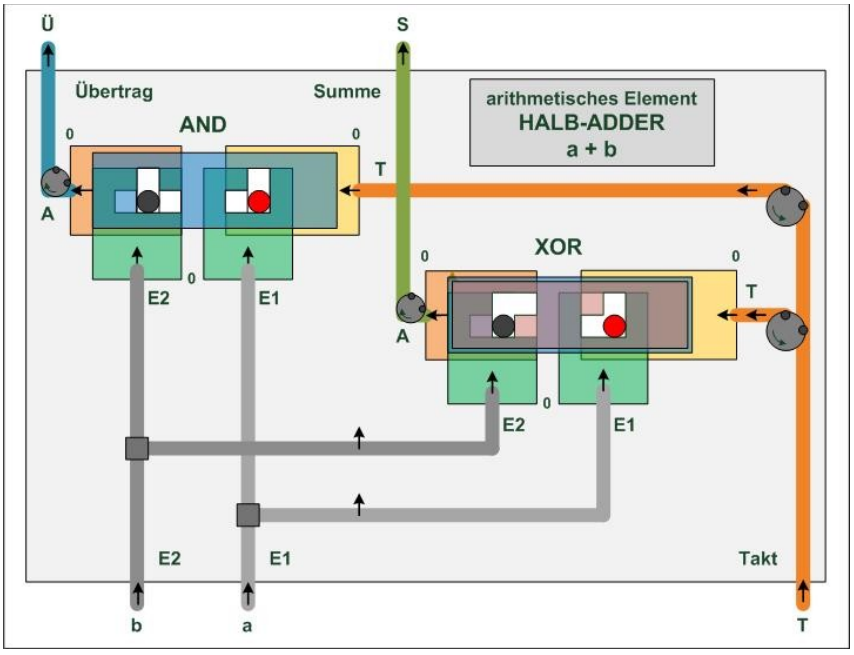


Abb.1 zeigt in dem Bild die Ungleichheit der Eingänge (E1=1, E2=0). Als Ergebnis stellt sich am Ausgang 1 ein. Für den anderen nicht dargestellten Fall (E1=0, E2=1) entsteht das gleiche Ergebnis.
Abb.2 zeigt den Fall der Gleichheit (E1=1, E2=1). Dadurch das eine Kopplungsplatte heraugeschoben wird, die andere aber nicht, bleibt der schwarze Kopplungsstift an der Stelle, das Ergebnis ist (wie auch bei E1=0, E2=0), gleich 0.
(hier kann man eine Simulation der **XOR-Funktion** sehen, nur die eben beschriebenen 2 Fälle))

Arithmetisches Grundelement PLUS (+)

Nun ist es auch möglich mit den logischen Elementen **XOR** und **AND** ein **HALB-ADDER** zu erstellen (siehe oben), ein arithmetisches Element:

Genau diese Erkenntnis hatte schon **Leibniz**, er hätte es auch realisieren können!



Weitere Information zur Simulation und Test logischer Funktionen, einschließlich der Bauanleitung des Simulators kann man unter den oben bereits genannten Beitrag lesen:

Timm Grams, Fulda; **Bastelbogen für ein mechanisches Schaltgliedmodell**

Wir haben nun ein **AND-Gatter** und ein **XOR-Gatter**, somit ist es möglich das arithmetische **PLUS** zu realisieren.

Und ich wollte nur einfach wissen, wie es geht – **und es geht!**