

Bereits erklärt: Stellenwertsysteme, insbesondere Binär- und Hexadezimalsystem (siehe pdf auf Homepage)

1.1. Bits and Bytes.

- **Bit** = binary digit = Binärziffer, also 0 oder 1.
- **Byte** = Folge von 8 Bit = 8-stellige Binärzahl, z. B. 0100 1101.

Weil man an jeder der 8 Positionen zwei mögliche Ziffern hat, kann ein Byte  $2^8 = 256$  verschiedene Werte annehmen, nämlich alle Binärzahlen von 00000000 bis 11111111.

1.1.1. Wenn eine Festplatte eine Speicherkapazität von 400 Gigabyte hat, kann man auf ihr  $400 \cdot 10^9$  8-stellige Binärzahlen abspeichern. Wie viel Information ist das?

Auf eine Seite DIN-A4-Papier passen ca. 2000 Zeichen, was 2000 Byte oder 2 Kilobyte entspricht, wenn man jedes Zeichen mit einem Byte abspeichert. Auf eine Festplatte von 400 Gigabyte passen also ca.  $\frac{400 \cdot 10^9}{2 \cdot 10^3} = 2 \cdot 10^8$  Seiten Text. Drucken wir diesen Text doppelseitig aus und nehmen an, dass eine Seite 0,1 Millimeter dünn ist, so ergibt dies einen Stapel der Höhe  $10^8 \cdot 0,1 \text{ mm} = 10^4 \text{ m} = 10 \text{ km}$ ; der Papierstapel ist also ungefähr so hoch wie der Mount Everest. Zum Vergleich: Überschlagsmässig gerechnet ist der Stapel aller Bücher der British Library, der grössten Bibliothek der Welt, „nur“ 70 mal so hoch.

1.2. **Logische Verknüpfungen und Gatter.** Bereits erklärt: AND, OR, NOT, Wahrheitstabellen (siehe pdf); elektrotechnische Realisierung der AND-, OR-, NOT-Gatter mit Transistoren (vgl. Youtube-Video)

1.2.1. In der Logik bzw. Mathematik schreibt man oft

- $a \vee b$  statt  $a$  OR  $b$ ; (Das Symbol  $\vee$  kommt von lateinisch *vel* oder.)
- $a \wedge b$  statt  $a$  AND  $b$ ; (Man mag  $\wedge$  als stilisierten Buchstaben A für AND oder als umgedrehtes  $\vee$  interpretieren.)
- $\neg a$  oder  $\bar{a}$  statt NOT  $a$ .

**Aufgabe 1.2.2.** Vervollständige die folgende Wahrheitstabelle! Was fällt dir auf?

$a$	$b$	$\overline{a \wedge b}$	$\overline{a \vee b}$	$\bar{a} \wedge \bar{b}$	$\bar{a} \vee \bar{b}$
0	0				
0	1				
1	0				
1	1				

1.2.3. Wenn du die Aufgabe korrekt gelöst hast, hast du die beiden **de Morganschen Gesetze** gezeigt:

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

**Aufgabe 1.2.4.** Zeige analog mindestens eines der beiden(!) **Distributivgesetze**

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c) \qquad a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c).$$

1.2.5. Für das Rechnen mit Booleschen Ausdrücken gibt es einige weitere Gesetze, siehe [Wikipedia: Boolesche Algebra; Definition](#), die man analog beweisen kann. Auf Grund der dort angegebenen Assoziativgesetze führt es zu keinen Missverständnissen, wenn man  $a \wedge b \wedge c$  statt  $a \wedge (b \wedge c)$  oder  $(a \wedge b) \wedge c$  schreibt, und analog für  $\vee$ .

**Aufgabe 1.2.6** (Binärzahlen auf digitalem Display darstellen). In der zweigeteilten Tabelle 1 kodieren die drei Bits  $a, b, c$  jeweils eine dreistellige Binärzahl, die ganz links als Dezimalzahl in digitaler Siebensegmentanzeige angegeben ist (vgl. [Wikipedia: Siebensegmentanzeige](#)). Schreibe in die freie Spalte eine 1, wenn das linke untere vertikale Segment leuchtet, und sonst eine 0. Finde einen logischen Ausdruck (also so etwas wie  $a \vee (b \wedge \bar{c})$ ) mit dieser Wahrheitstabelle.

	$a$	$b$	$c$			$a$	$b$	$c$	
0	0	0	0		4	1	0	0	
1	0	0	1		5	1	0	1	
2	0	1	0		6	1	1	0	
3	0	1	1		7	1	1	1	

TABELLE 1. Zur Siebensegmentanzeige

Hinweis: Um einen logischen Ausdruck mit den gesuchten Eigenschaften zu finden: Trage die Variablen  $a$ ,  $b$ ,  $c$  und ihre Verneinungen  $\bar{a}$ ,  $\bar{b}$ ,  $\bar{c}$  geschickt in die folgende „Lückenformel“ ein.

$$\left( \quad \wedge \quad \wedge \quad \right) \vee \left( \quad \wedge \quad \wedge \quad \right) \vee \left( \quad \wedge \quad \wedge \quad \right)$$

Wenn das noch nicht hilft: Jeder Klammerausdruck entspricht einer Zeile mit einer 1 ganz rechts.

**1.2.7.** Der Hinweis in der vorherigen Aufgabe liefert rasch: Jede Wahrheitstabelle kann man durch einen logischen Ausdruck, in dem nur die Verknüpfungen  $\wedge = \text{AND}$ ,  $\vee = \text{OR}$  und  $\bar{\quad} = \text{NOT}$  vorkommen, darstellen. Die im Hinweis angedeutete Standardform heisst „disjunktive Normalform“, denn es handelt sich um die „Ver-ODER-ung“ (= Disjunktion) von Ausdrücken zu denjenigen Zeilen der Tabelle, in denen eine 1 herauskommen soll (vgl. [Wikipedia: Disjunktive Normalform, Beispiel](#)).

Theoretisch zeigt dies bereits, dass man eine (riesigengrosse) logische Schaltung bauen kann, die zwei 8-stellige Binärzahlen addiert (die zugehörige Wahrheitstafel hat  $2^{16} = 65536$  Zeilen und der logische Ausdruck ist entsprechend lang). Mit der üblichen *schriftlichen Addition*, die in jedem Stellenwertsystem funktioniert, geht das aber deutlich einfacher, wie wir im Folgenden sehen werden.

**Aufgabe 1.2.8** (NAND ist ein universelles Gatter). (freiwillig) Definiere die logische Verknüpfung NAND (kurz für *not and*) durch

$$a \text{ NAND } b := \text{NOT } (a \text{ AND } b).$$

Stelle Negation NOT  $a$ , Konjunktion  $a$  AND  $b$  und Disjunktion  $a$  OR  $b$  nur mit Hilfe der NAND-Verknüpfung dar.

Dies zusammen mit Bemerkung 1.2.7 zeigt, dass man allein mit der NAND-Verknüpfung jede beliebige Wahrheitstabelle erzielen kann: Man nennt NAND deswegen eine *universelle* Verknüpfung oder ein *universelles Gatter*.

**1.2.9.** Die schriftliche Addition zweier vierstelliger Zahlen kann man durch das Schema in Abbildung 1 illustrieren. Man addiert die beiden Ziffern ganz rechts und erhält eine „Summe“ (hier als  $s$  markiert) und einen Übertrag (hier  $c_{\text{out}}$  für englisch *carry*). Die beiden Ziffern an der nächsten Stelle zusammen mit diesem Übertrag ergeben die zweite Ziffer des Ergebnisses und einen weiteren Übertrag undsoweiter.

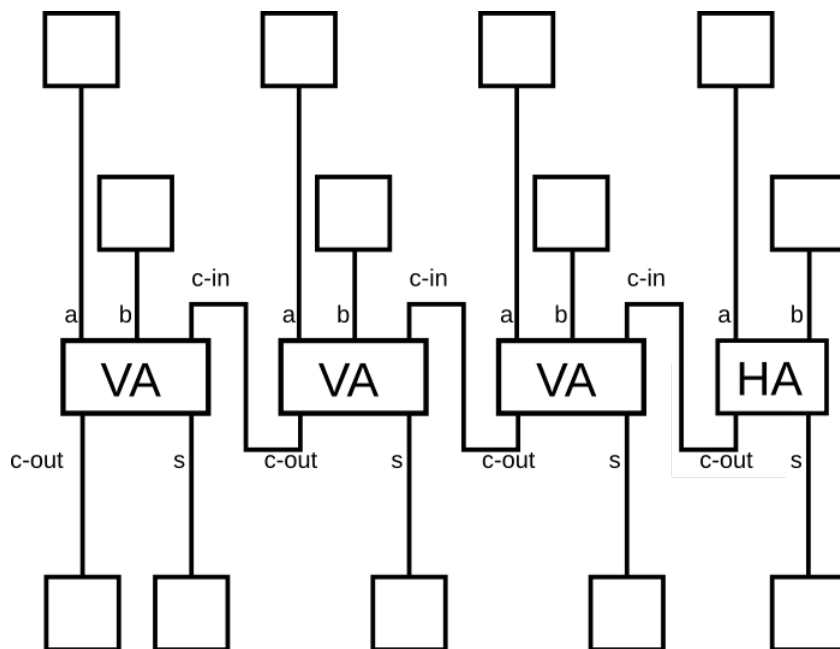


ABBILDUNG 1. 4-Bit-Addierer

Dies funktioniert in jedem Stellenwertsystem, wie aus der Primarschule für das Zehnersystem bekannt. Im Folgenden beschränken wir uns auf das Binärsystem.

**Aufgabe 1.2.10.** Die Wahrheitstafel für den Halbaddierer (= die Box/das Bauteil HA) sieht wie in Tabelle 2 dargestellt aus. Stelle  $c$  und  $s$  in Abhängigkeit von  $a$  und  $b$  durch einen logischen Ausdruck dar.

**Aufgabe 1.2.11.** Vervollständige Tabelle 3, die Wahrheitstabelle für den Volladdierer (= das Bauteil VA), und stelle  $c_{\text{out}}$  und  $s$  in Abhängigkeit von  $a$ ,  $b$  und  $c_{\text{in}}$  durch einen logischen Ausdruck dar.

$a$	$b$	carry/Übertrag $c$	sum/Summe $s$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

TABELLE 2. Halbaddierer

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

TABELLE 3. Volladdierer

**1.2.12.** In der Wahrheitstabelle des Halbaddierers (siehe Tabelle 2) gilt genau dann  $s = 1$ , wenn entweder  $a = 1$  oder  $b = 1$  gilt. Man schreibt deswegen auch  $s = a \text{ XOR } b$  und nennt diese Verknüpfung exklusives Oder bzw. englisch *exclusive or*.

**Aufgabe 1.2.13.** Installiere die Simulations-Software Logisim für logische Schaltungen auf deinem Rechner:

- Projektseite: <http://www.cburch.com/logisim/>
- Downloadseite: <https://sourceforge.net/projects/circuit/>

Spiele mit Logisim herum und baue den Halbaddierer, dann den Volladdierer und dann den 4-Bit-Addierer (mit Hexadezimal-Displays)!

Hinweis: Für diverse hilfreiche Videos siehe

<https://fginfo.ksbg.ch/dokuwiki/doku.php?id=lehrkraefte:blc:informatik:glf20:bitsundbytes-bitoperationen#addierer>

**1.2.14** (Verschlüsseln mit XOR). Dieses exklusive Oder XOR eignet sich vorzüglich zum Verschlüsseln, denn wendet man zweimal  $a \text{ XOR}$  auf ein Bit  $b$  an, so erhält man  $b$  zurück:

$$a \text{ XOR } (a \text{ XOR } b) = b$$

Zum Verschlüsseln längerer Nachrichten wendet man dies bitweise an: Ist etwa  $k = 011010101$  der sogenannte Schlüssel (englisch *key*), der nur Alice und Bob bekannt ist, so kann Alice eine Nachricht, etwa  $n = 110101100$  verschlüsseln, indem sie XOR bitweise auf  $k$  und  $n$  anwendet.

```

k = 011010101
n = 110101100
-----
e = 101111001

```

Das Ergebnis  $e$  sendet sie über eine möglicherweise abgehörte Verbindung an Bob; ohne Kenntnis des Schlüssels  $k$  kann ein Angreifer mit  $e$  nichts anfangen, denn  $e$  unterscheidet sich genau an den Stellen von  $n$ , an denen der Schlüssel  $k$  den Wert 1 hat. Zum Entschlüsseln wendet Bob XOR bitweise auf  $k$  und  $e$  an und erhält so die ursprüngliche Nachricht.