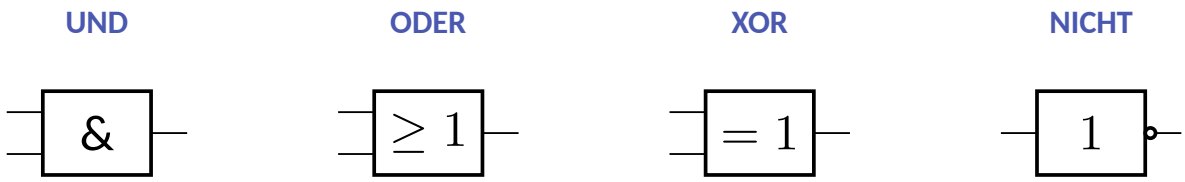
2.9 Kapitel 2 kompakt: der Lernzettel

Leitidee

Dieser Lernzettel fasst das Kapitel auf wenigen Seiten zusammen. Nutze ihn zum Wiederholen vor einem Test. Ausführliche Erklärungen, Beispiele und Übungen findest du in den einzelnen Abschnitten.

2.9.1 Grundgatter, Belegungstabellen, Schaltungsentwurf (Abschnitte 2.1 und 2.2)



Die vier Grundgatter als IEC-Symbole. CircuitVerse zeigt sie als US-Symbole

Gatter	Zeichen	Term	Ausgang 1, wenn …
UND (AND)	`&`	$A \wedge B$	**alle** Eingänge 1 sind
ODER (OR)	`≥1`	$A \vee B$	**mindestens ein** Eingang 1 ist
XOR	`=1`	$A \oplus B$	die Eingänge **unterschiedlich** sind
NICHT (NOT)	`1` mit Kreis	$\neg A$	der Eingang 0 ist (Gegenteil)

A	B	$A \wedge B$	$A \vee B$	$A \oplus B$	$\neg A$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

- **Belegungstabelle:** für jede Kombination der Eingangswerte der Ausgangswert. Bei n Eingängen 2^n Zeilen (3 Eingänge: 8 Zeilen). Reihenfolge: Die letzte Spalte wechselt in jeder Zeile, die davor alle 2 Zeilen, die davor alle 4 Zeilen.
- **Schaltung → Tabelle:** von links nach rechts, für jeden Gatterausgang eine **Hilfsspalte**.
- **Entwurf in vier Schritten:** Eingänge und Ausgang benennen (was bedeuten 1 und 0?), Belegungstabelle aufstellen, Gatter auswählen und verschalten (Term hilft), mit **allen** Belegungen testen.
- Beispiel Alarmanlage: Alarm $=$ (Tür $\wedge$ Scharf) $\vee$ Rauch. Was in der Klammer steht, rechnet ein eigenes Gatter zuerst.
- Kontrolle mit Python: `and`, `or`, `not`, XOR als `a != b`; alle Belegungen mit geschachtelten Schleifen `for a in [0, 1]:` usw. Ergebnis mit `int(...)` als 0 oder 1 ausgeben.

Typische Fehler: ODER und XOR verwechseln (Unterschied nur bei 1 1); eine Zeile der Tabelle vergessen; die Bedeutung von 0 und 1 nicht festlegen; Klammern vergessen ($(T \wedge S) \vee R$ ist nicht $T \wedge (S \vee R)$).

2.9.2 Binäre Addition, Halb- und Volladdierer (Abschnitte 2.3 und 2.4)

- **Binäre Addition** wie schriftlich im Dezimalsystem, von rechts nach links: $0 + 1 = 1$, $1 + 1 = 10$ (0 schreiben, 1 übertragen), $1 + 1 + 1 = 11$ (1 schreiben, 1 übertragen). Überträge in eine eigene Ü-Zeile.

```
      0 1 0 1     (= 5)
    + 0 0 1 1     (= 3)
Ü     1 1 1
    ---------
      1 0 0 0     (= 8)
```

- **Halbaddierer:** addiert **zwei** Bits. Summe $S = A \oplus B$ (XOR-Gatter), Übertrag $C = A \wedge B$ (UND-Gatter). „Halb“, weil er keinen eingehenden Übertrag verarbeiten kann.
- **Volladdierer:** addiert **drei** Bits A, B, C_{ein} und liefert S und C_{aus}. Aufbau: zwei Halbaddierer und ein ODER-Gatter: HA 1 rechnet $A + B$ (ergibt S_1, C_1), HA 2 rechnet $S_1 + C_{\text{ein}}$ (ergibt S, C_2), dann $C_{\text{aus}} = C_1 \vee C_2$. Kurz: $C_{\text{aus}}S$ ist die Summe der drei Bits als Binärzahl.
- **Carry-Ripple-Addierer:** ein Volladdierer pro Stelle; der ausgehende Übertrag einer Stelle ist der eingehende Übertrag der nächsthöheren Stelle. Ganz rechts ist der Übertrag 0. Der Übertrag „rieselt“ von rechts nach links; jede Stufe wartet auf die vorige. Vier Stellen ergeben bis zu fünf Ergebnisbits.
- Kontrolle mit Python: `int("0101", 2)` ergibt 5, `bin(8)` ergibt `'0b1000'`.

Typische Fehler: „1 + 1 = 2“ schreiben; Übertrag vergessen oder unter die falsche Stelle schreiben; den letzten Übertrag ganz links weglassen; beim Volladdierer ein UND statt des ODER-Gatters am Ende.

2.9.3 Rückkopplung und Flipflop (Abschnitt 2.5)

- **Kombinatorische Schaltung:** Ausgang hängt nur von den **aktuellen** Eingängen ab (Gatter, Addierer, Alarmanlage).
- **Flipflop:** merkt sich ein Bit; der Ausgang hängt auch von der **Vorgeschichte** ab. Grundlage schneller Speicher, z. B. Register.
- **Rückkopplung:** Ein Ausgang wird auf einen Eingang der eigenen Schaltung zurückgeführt.
- **SR-Flipflop:** zwei NOR-Gatter (ODER mit Kreis am Ausgang), **über Kreuz** zurückgekoppelt. Ausgänge Q und $\overline{Q}$ (auch `Q'`).
- $S = 1$, $R = 0$: **Setzen** ($Q = 1$). $S = 0$, $R = 1$: **Rücksetzen** ($Q = 0$). $S = 0$, $R = 0$: **Speichern** (Q bleibt wie vorher). $S = R = 1$: **verboten**.
- In Python ahmt eine Variable die Rückkopplung nach: `q = sr_flipflop(s, r, q)`.

Typische Fehler: bei $S = R = 0$ einfach 0 eintragen statt den alten Wert; S und R vertauschen.

2.9.4 Aussagenlogik und Termbäume (Abschnitt 2.6)

Name	Zeichen	Gatter	Python
Negation	$\neg A$	NICHT	`not a`
Konjunktion	$A \wedge B$	UND	`a and b`
Disjunktion	$A \vee B$	ODER	`a or b`

- **Induktive Definition:** (1) Jede Aussagenvariable ist ein Term. (2) Ist t ein Term, dann auch $\neg t$. (3) Sind t_1, t_2 Terme, dann auch $(t_1 \wedge t_2)$ und $(t_1 \vee t_2)$. (4) Nichts anderes ist ein Term.

- **Termbaum:** Blätter = Variablen, innere Knoten = Operatoren, Wurzel = Operator, der **zuletzt** ausgewertet wird. Beispiel $(A \wedge B) \vee \neg C$: Wurzel $\vee$, darunter $\wedge$ (mit A, B) und $\neg$ (mit C).
- **Term ↔ Schaltung:** Jeder innere Knoten ist ein Gatter ($\wedge$ UND, $\vee$ ODER, $\neg$ NICHT), die Blätter sind die Eingänge, die Wurzel ist das letzte Gatter. Was in einer Klammer steht, rechnet ein eigenes Gatter zuerst.

Typische Fehler: $\neg(A \vee B)$ und $\neg A \vee B$ verwechseln (das $\neg$ gilt nur für das, was direkt dahinter steht); die falsche Wurzel wählen; $\oplus$ in einen Term schreiben (gehört nicht zu den Regeln).

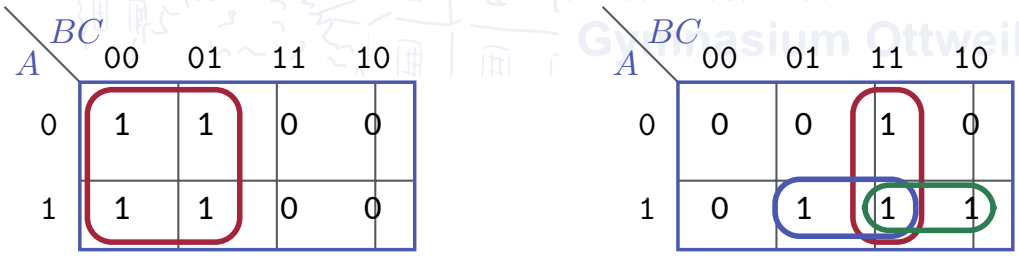
2.9.5 Äquivalenz und DNF (Abschnitt 2.7)

- **Äquivalent:** Zwei Terme liefern für **jede** Belegung denselben Wert, haben also dieselbe Belegungstabelle. Äquivalenz ist ein Begriff **über** Terme, kein Operator. Für „nicht äquivalent“ genügt **eine** Belegung, in der sie sich unterscheiden.
- Beispiel (De Morgan): $\neg(A \wedge B)$ ist äquivalent zu $\neg A \vee \neg B$, aber **nicht** zu $\neg A \wedge \neg B$.
- **Minterm:** Konjunktion **aller** Variablen; eine Variable mit dem Wert 0 bekommt ein $\neg$. Zur Zeile $A = 1$, $B = 0$, $C = 1$ gehört $A \wedge \neg B \wedge C$.
- **DNF:** Minterme aller Zeilen mit Ausgang 1, verbunden mit $\vee$. Beispiel XOR: Einsen bei $(0, 1)$ und $(1, 0)$, DNF $(\neg A \wedge B) \vee (A \wedge \neg B)$.
- Äquivalenz mit Python prüfen: beide Terme als Funktionen schreiben, `aequivalent = True` setzen, in geschachtelten Schleifen alle Belegungen vergleichen und bei `!=` auf `False` setzen.

Typische Fehler: $\neg$ an die Variablen mit Wert 1 setzen; eine Variable im Minterm vergessen; Zeilen mit Ausgang 0 verwenden; $\wedge$ und $\vee$ vertauschen.

2.9.6 KV-Diagramme (Abschnitt 2.8)

- **KV-Diagramm:** alle Belegungen in einem Raster, Spalten (und bei vier Variablen auch Zeilen) in der Reihenfolge **00, 01, 11, 10**. Benachbarte Felder unterscheiden sich in **genau einer** Variablen.
- Benachbarte Einsen zu **Blöcken** zusammenfassen: Rechtecke aus 1, 2, 4 oder 8 Feldern, nur Einsen, so **groß** und so **wenige** wie möglich; Überlappen ist erlaubt; jede 1 liegt in einem Block.
- **Term eines Blocks:** nur die Variablen, die im ganzen Block **konstant** sind (mit $\neg$ bei 0). Alle Blockterme mit $\vee$ verbinden. Ergebnis: **minimierte DNF**, äquivalent zur DNF, aber mit weniger Gattern.



Links: ein Viererblock mit $B = 0$, Ergebnis $\neg B$. Rechts: Mehrheit, drei Zweierblöcke, Ergebnis $(B \wedge C) \vee (A \wedge C) \vee (A \wedge B)$

Typische Fehler: Spalten in der Reihenfolge 00, 01, 10, 11 anlegen; diagonale Felder zusammenfassen; Dreierblöcke bilden; zu kleine Blöcke wählen (richtig, aber nicht minimal); eine 1 in keinem Block vergessen.