
Einführung in die Informatik

Bausteine und Schaltungen

Meik Teßmer

Inhalte

Lernziele

- Rechnen mit logischen Formeln
- Übertragen auf Bausteine
- Entwickeln einer komplexeren Schaltung

Rückblick

Umgang mit Zahlen und Operationen im Dualsystem:

- Umrechnung Dezimalsystem \rightarrow Dualsystem: Division mit Rest
- Rechnen im Dualsystem: 4 Grundrechenarten
 - Trick bei der Subtraktion: Zweierkomplement
- Aussagenlogik: And (\wedge), Or (\vee), Not (\neg)
- Gesetze der Booleschen Algebra, DeMorgan

Inhalte
○○○

Aussagenlogik
●○○

Übertragung auf Bausteine
○○○○○○○○

Komplexere Bausteine
○○○○○○○○

Verwendungen
○○○○○○○○○○○○

Optimierungen
○○○○○○○○

Aussagenlogik

Wahrheitstafel

- schrittweises *Berechnen* einer Formel für alle möglichen Kombinationen von Eingangswerten
- Anzahl Zeilen korrespondiert zur Menge der Variablen: 2^n mit n =Anzahl
- Beispiel: Ergebnisse für $\neg a$, $a \wedge b$ und $a \vee b$

2 Variablen a und $b \rightarrow 4$ Zeilen

a	b	$\neg a$	$a \wedge b$	$a \vee b$
1	1	0	1	1
1	0	0	0	1
0	1	1	0	1
0	0	1	0	0

Alternative Schreibweisen

- angelehnt an das normale schriftliche Rechnen
- Konjunktion = Multiplikation $\rightarrow a \wedge b = a \cdot b = ab$
- Disjunktion = Addition $\rightarrow a \vee b = a + b$
- Negation $\rightarrow \neg a = !a$

Beispiel: $(a \wedge b) \vee c \rightarrow ab + c$

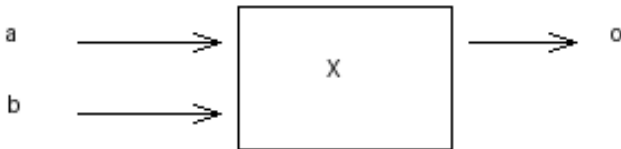
- weitere Alternative: Präfix-Notation für die Übertragung auf Bausteine

And(a, b), Or(a, b), Not(a)

Übertragung auf Bausteine

Bausteine

- allgemeine Darstellung von Bausteinen: Eingänge, Ausgänge, ggf. Interna



mit $X = \{And, Or, Not\}$

Anforderung bei der industriellen Umsetzung

- effiziente Produktion mit nur wenigen unterschiedlichen Bauteilen
- Ziel: Reduktion der Anzahl benötigter Grundbausteine

Könnten wir alle drei Grundbausteine mit einem fiktiven Nand-Element bauen?

Not mit Nand-Bausteinen

$$\begin{aligned} \text{Not}(a) &= !(a \wedge a) && \leftarrow \text{Nand-Entsprechung} \\ &= !a \vee !a \\ &= !a \end{aligned}$$

Or mit Nand-Bausteinen

$$\begin{aligned} Or(a, b) &= And(a, a) \vee And(b, b) \\ &= !(And(a, a) \vee And(b, b)) \\ &= !(!And(a, a) \wedge !And(b, b)) \quad \leftarrow \text{Nand-Entsprechung} \\ &= Nand(Nand(a, a), Nand(b, b)) \end{aligned}$$

And mit Nand-Bausteinen

$$\begin{aligned} \text{And}(a, b) &= \neg\neg\text{And}(a, b) \\ &= \neg\neg\text{And}(a, b) \vee \neg\neg\text{And}(a, b) \\ &= \neg(\neg\text{And}(a, b) \wedge \neg\text{And}(a, b)) \quad \leftarrow \text{Nand-Entsprechung} \\ &= \text{Nand}(\text{Nand}(a, b), \text{Nand}(a, b)) \end{aligned}$$

Umsetzung in Form eines Bausteins

- Ergebnis: die drei Grundbausteine können durch Nand aufgebaut werden → wir können And, Or und Not für den Schaltungsentwurf nutzen
- Nand-Bausteine gibt es als Integrierte Schaltung (kostet €0,5)

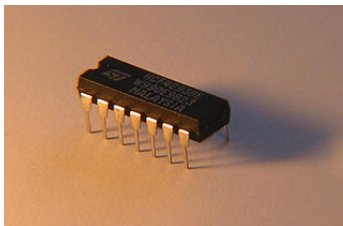


Abbildung 1: Integrated Circuit (IC)

Umsetzung in Form eines Bausteins

Schaltungsplan dazu (verwendet ein Mehrfach-Nand)

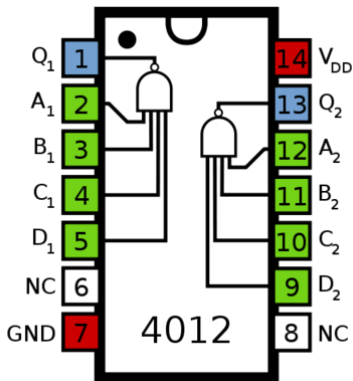


Abbildung 2: Schaltung dazu

Komplexere Bausteine

Beispiel: Xor

- eXclusive Or
- normales Or: $1 \vee 1 = 1$
- im Unterschied dazu Xor: $1 \vee 1 = 0$

Wie könnte dieser Baustein aus den Grundelementen And, Or und Not aufgebaut werden?

Entwicklung von Xor

- Funktionsweise: Ergebnis x ist nur dann 1, wenn entweder a oder b 1 ist, aber nicht beide
- Symbol: $\underline{\vee}$
- Wahrheitstafel für Xor:

<u>a</u>	<u>b</u>	<u>a $\underline{\vee}$ b</u>
1	1	0
1	0	1
0	1	1
0	0	0

- Welche Bausteine werden dazu benötigt?
→ Hilfsmittel: *kanonische Repräsentation*

Kanonische Repräsentation

- 1 Wahrheitstafel aufbauen
- 2 Zeilen auswählen, deren Ergebnis 1 ist
- 3 Eingänge mittels Konjunktion verbinden
- 4 diese Zeilen mittels Disjunktion verbinden

Beispiel:

a	b	$a \vee b$
1	0	1
0	1	1

→ zwei Konjunktionen: $a \wedge \neg b$ und $\neg a \wedge b$

zusammen: $(a \wedge \neg b) \vee (\neg a \wedge b)$

Entwicklung von Xor

Überprüfung mit Hilfe einer Wahrheitstafel:

a	b	$\neg a$	$\neg b$	$a \wedge \neg b$	$\neg a \wedge b$	$(a \wedge \neg b) \vee (\neg a \wedge b)$
1	1	0	0	0	0	0
1	0	0	1	1	0	1
0	1	1	0	0	1	1
0	0	1	1	0	0	0

Wie sieht $(a \wedge \neg b) \vee (\neg a \wedge b)$ als Schaltung aus?

Entwicklung von Xor

- 1 Kasten mit Ein- und Ausgängen zeichnen
- 2 Bausteine als Kästen einzeichnen
- 3 von den Eingängen Leitungen ziehen, über die Bausteine bis hin zu den Ausgängen

von *innen* nach *außen* arbeiten

→ erst Negation, dann Konjunktion, zuletzt Disjunktion

Entwicklung von Xor

Xor

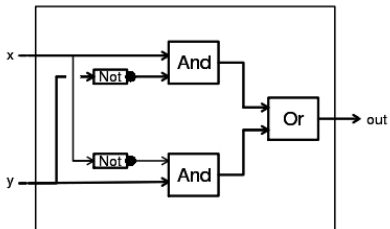


Abbildung 3: Xor-Schaltplan

Weitere praktische Bausteine

Nand, Nor und Xnor:

- Nand: $\neg(a \wedge b) = !(ab)$
- Nor: $\neg(a \vee b) = !(a + b)$
- Xnor: $\neg(a \underline{\vee} b) = !Xor(a, b)$
- Xor: $(a \wedge \neg b) \vee (\neg a \wedge b) = a \underline{\vee} b = Xor(a, b)$

a	b	Nand(a,b)	Nor(a,b)	Xnor(a,b)	Xor(a,b)
1	1	0	0	1	0
1	0	1	0	0	1
0	1	1	0	0	1
0	0	1	1	1	0

Inhalte
○○○

Aussagenlogik
○○○

Übertragung auf Bausteine
○○○○○○○○

Komplexere Bausteine
○○○○○○○○

Verwendungen
●○○○○○○○○○○

Optimierungen
○○○○○○○○

Verwendungen

Mögliche Einsatzgebiete

- wo liegt der praktischer Nutzen der vorgestellten Bausteine?
- Rückblick: Addition im Dualsystem als Basis aller Rechenoperationen

$$\begin{array}{r} 1011_2 \\ + \quad 11_2 \\ \hline 1110_2 \end{array}$$

Abbildung 4: Addition

$$1 + 1 = 10$$

$$1 + 0 = 1$$

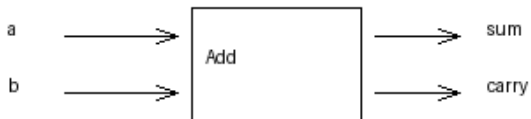
$$0 + 1 = 1$$

$$0 + 0 = 0$$

Was fällt auf?

Umsetzung der Addition

Als Baustein sähe eine Addition wie folgt aus:



Umsetzung der Addition

zugehörige Wahrheitstafel:

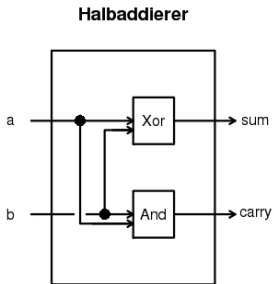
a	b	sum	carry
1	1	0	1
1	0	1	0
0	1	1	0
0	0	0	0

Welche geeigneten Bausteine kennen wir?

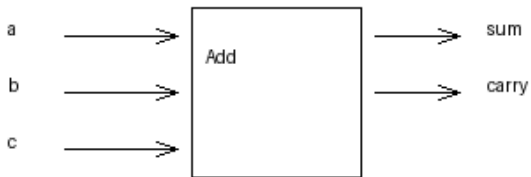
Umsetzung der Addition

- $\text{sum} \rightarrow \text{Xor}(a, b)$
- $\text{carry} \rightarrow \text{And}(a, b)$

Ergebnis ist ein sog. *Halbaddierer*



Erweiterung auf Zwei-Bit-Addition



Erweiterung auf 2-Bit-Addition

- 3 Eingänge: 2 Zahlen und Übertrag
- 2 Ausgänge: Ergebnis und Übertrag

Wahrheitstafel (a, b: Zahlen; c: Übertrag):

a	b	c	sum	carry
1	1	1	1	1
1	1	0	0	1
1	0	1	0	1
1	0	0	1	0
0	1	1	0	1
0	1	0	1	0
0	0	1	1	0
0	0	0	0	0

Kanonische Repräsentationsform

Formel für sum:

a	b	c	sum
1	1	1	1
1	0	0	1
0	1	0	1
0	0	1	1

$$(a \wedge b \wedge c) \vee$$

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

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

$$(\neg a \wedge \neg b \wedge c)$$

Kanonische Repräsentationsform

Formel für carry:

a	b	c	carry
1	1	1	1
1	1	0	1
1	0	1	1
0	1	1	1

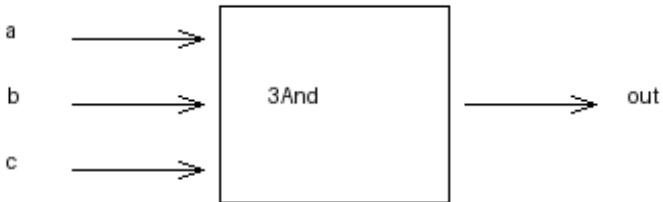
$$(a \wedge b \wedge c) \vee$$

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

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

$$(\neg a \wedge b \wedge c)$$

And-Baustein mit drei Eingängen



Umsetzung von 3-And?

Umsetzung

eine Möglichkeit: zwei And-Bausteine verknüpfen

3-And

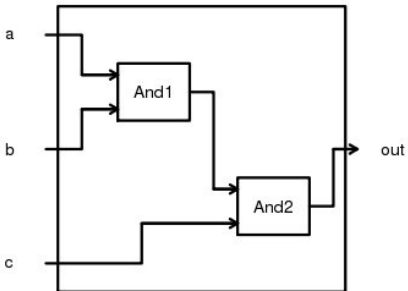


Abbildung 6: 3-And.

Umsetzung des Volladdierers

- alle benötigten Teile liegen vor:
 - 3-And und Or-Bausteine
 - Eingänge ggf. mit Not-Baustein anpassen
- Problem: Für sum und carry brauchen wir dann jeweils $8 \text{ And} + 3 \text{ Or} + 3 \text{ Not} = 14$ Bausteine!
- Verbesserungsmöglichkeiten für eine kostengünstigere Produktion?

Inhalte
○○○

Aussagenlogik
○○○

Übertragung auf Bausteine
○○○○○○○○

Komplexere Bausteine
○○○○○○○○

Verwendungen
○○○○○○○○○○○○

Optimierungen
●○○○○○○○

Optimierungen

Mögliche Verbesserungen

- Annahme: Xor als Baustein liegt vor
- Idee: Vereinfachen der logischen Formeln

carry und sum so ändern, dass durch Xor Bausteine gespart werden können

$$\text{carry} = (a \wedge b \wedge c) \vee (a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c)$$

$$\text{sum} = (a \wedge b \wedge c) \vee (a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c)$$

Mögliche Verbesserungen

$$\begin{aligned} \text{carry} &= abc + ab!c \quad | \text{ ab ausklammern, } c + !c=1 \\ &\quad + a!bc + !abc \\ &= ab \\ &\quad + a!bc + !abc \quad | \text{ c ausklammern} \\ &= ab \\ &\quad + c(a!b + !ab) \quad | \text{ hier steht ein Xor} \end{aligned}$$

$$\rightarrow \text{carry} = (a \wedge b) \vee (c \wedge \text{Xor}(a, b))$$

benötigte Bausteine: 2 And, 1 Or, ein Xor

Mögliche Verbesserungen

$$\begin{aligned}
 \text{sum} &= abc + a!b!c + !ab!c + !a!bc && | \text{!c ausklammern} \\
 &= abc && | \text{c ausklammern} \\
 &\quad + !c(a!b + !ab) \\
 &\quad + !a!bc && | \text{" " } \\
 &= c(ab + !a!b) \\
 &\quad + !c(a!b + !ab) && | \text{Xor} \\
 &= c(ab + !a!b) && | \text{Xnor (Not Xor)} \\
 &\quad + !c(\text{Xor}(a, b)) \\
 &= c(\neg\text{Xor}(a, b) + !c(\text{Xor}(a, b)))
 \end{aligned}$$

$$c \wedge \neg\text{Xor}(a, b) \vee \neg c(\text{Xor}(a, b))$$

und: $x \wedge \neg y \vee \neg x \wedge y$ ist auch ein Xor!

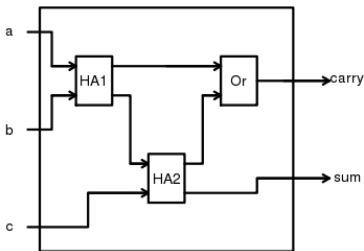
→ $\text{sum} = \text{Xor}(\text{Xor}(a, b), c)$

Umsetzung des Volladdierers

Genau betrachtet haben wir so nur zwei Halbaddierer kombiniert!

- Bausteine: Xor, And
- 2 Halbaddierer mit Or verknüpfen → Volladdierer

Volladdierer



Umsetzung des Volladdierers: Details

- $HA1 = a + b$
- $HA2 = \text{sum} + c$

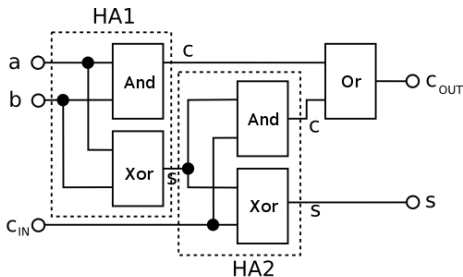


Abbildung 8: Volladdierer

4-Bit-Addierer

- Idee: 4 Volladdierer zusammenschalten
- zu addierende Zahlen $a = (a_0, \dots, a_3)$ und $b = (b_0, \dots, b_3)$
- Summe $s = (s_0, \dots, s_3)$ und carry

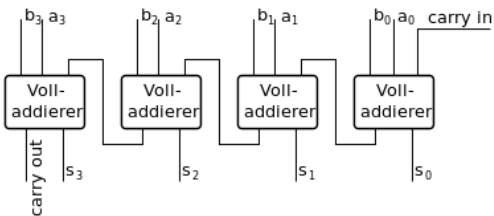


Abbildung 9: 4-Bit-Addierer

n-Bit-Addierer?

Inhalte
○○○

Aussagenlogik
○○○

Übertragung auf Bausteine
○○○○○○○○

Komplexere Bausteine
○○○○○○○○

Verwendungen
○○○○○○○○○○○○

Optimierungen
○○○○○○●

Ende