Einführung in die Informatik

Bausteine und Schaltungen

Meik Teßmer

Inhalt der Veranstaltung

Lernziele:

- Rechnen mit logischen Formeln
- Übertragung auf Bausteine
- Entwicklung einer komplexen Schaltung

Wiederholung

- 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

Aussagenlogik

- Wahrheitstafel:

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 Schreibweise:
  - Konjunktion = Multiplikation $\rightarrow a \wedge b = a * b = ab$
  - Disjunktion = Addition $\rightarrow a \vee b = a + b$
  - Negation $\rightarrow \neg a = !a$

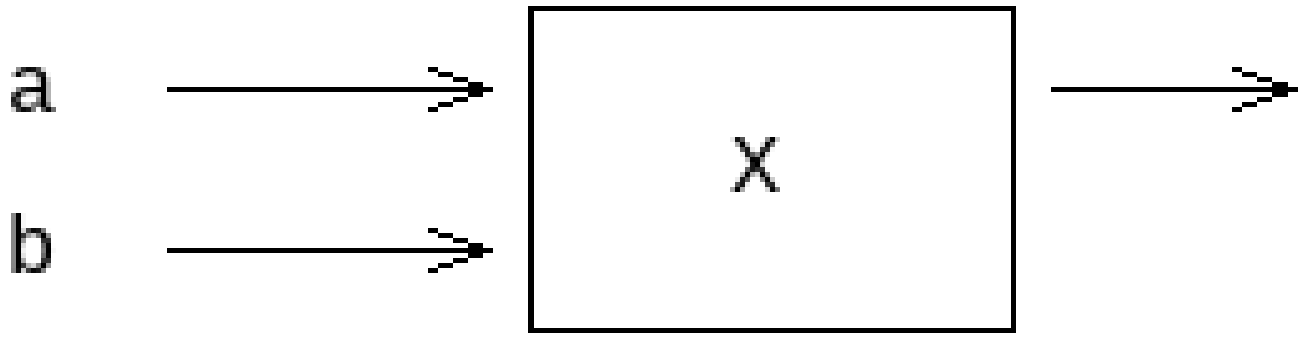
  Bsp.: $(a \wedge b) \vee c \rightarrow ab + c$

- weitere Alternative: Präfix-Notation

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

Bausteine

- Symbole für Grundbausteine:



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

- Anforderung: Reduktion der Anzahl nötiger Grundbausteine
  - ist einfacher zu fertigen
  - Produktion wird billiger
- Idee: alle drei Bausteine mit einem fiktiven `Nand` bauen

Zusammengesetzte Bausteine

Not

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

Zusammengesetzte Bausteine

Or

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

Zusammengesetzte Bausteine

And

$$
\begin{aligned}
And(a,b) &= !!And(a,b) \\
&= !!And(a,b) \vee !!And(a,b) \\
&= !(!And(a,b) \wedge !And(a,b)) \\
&= Nand(Nand(a,b), Nand(a,b))
\end{aligned}
$$

Umsetzung in Form eines Bausteins

Integrierte Schaltung

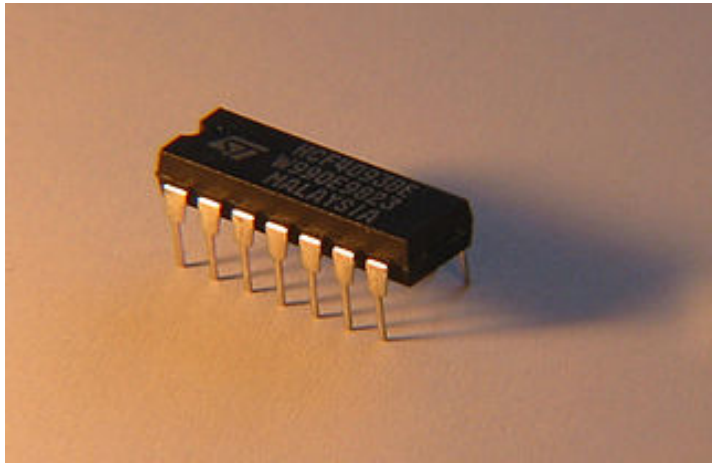


Figure 1: Integrated Circuit (IC)

Umsetzung in Form eines Bausteins

Schaltungsplan dazu (Mehrfach-Nand)

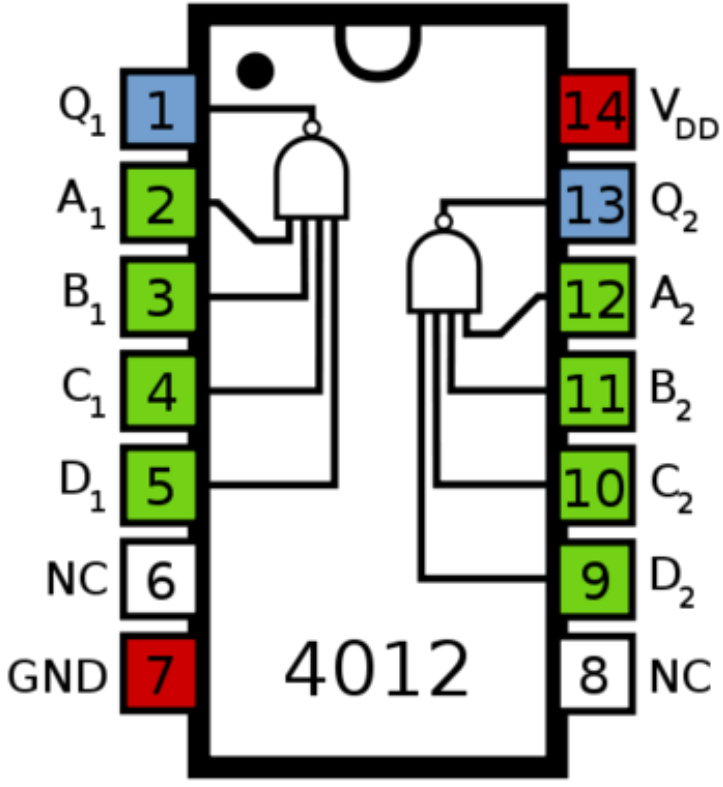


Figure 2: Schaltung dazu

Komplexere Bausteine

- weiterer praktischer Baustein: Xor (eXclusive Or)
- normales Or: $1 \vee 1 = 1$
- Xor: $1 \vee 1 = 0$

Aufbau

Wie könnte der aussehen?

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:

a	b	a $\underline{\vee}$ b
1	1	0
1	0	1
0	1	1
0	0	0

- Welche Bausteine werden dazu benötigt?

  Hilfsmittel: kanonische Repräsentation

Kanonische Repräsentation

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

Beispiel:

a	b	a $\underline{\vee}$ b
1	0	1
0	1	1

$\rightarrow$ 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

(a $\wedge$ $\neg$b) $\vee$ ($\neg$a $\wedge$ b): Schaltung?

Entwicklung von Xor

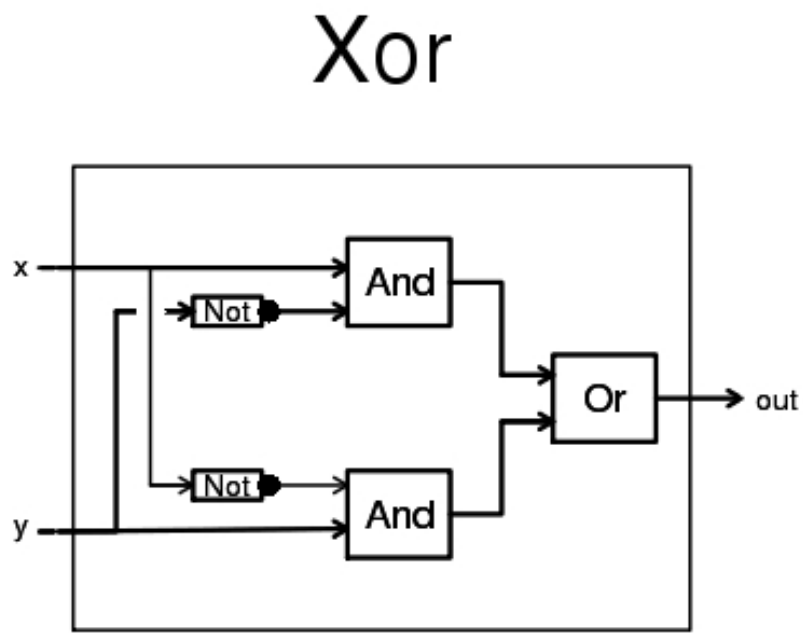


Figure 3: Xor-Schaltplan

Nand, Nor und Xnor

weitere praktische Bausteine:

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

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

Einsatzgebiete

- wo liegt der praktischer Nutzen der vorgestellten Bausteine?
- Rückblick: Addition im Dualsystem

$$\begin{array}{rr} & 1011_2 \\ + & 11_2 \\ \hline & 1110_2 \end{array}$$

Figure 4: Addition

```
1 + 1 = 10
1 + 0 =  1
0 + 1 =  1
0 + 0 =  0
```

Was fällt auf?

Umsetzung der Addition

Darstellung der Addition als Baustein:

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

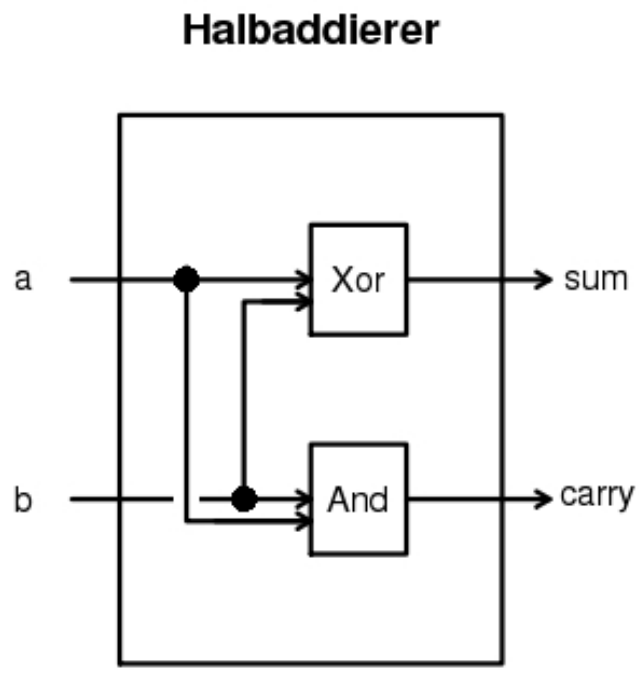
Welche geeigneten Bausteine kennen wir?

Umsetzung der Addition

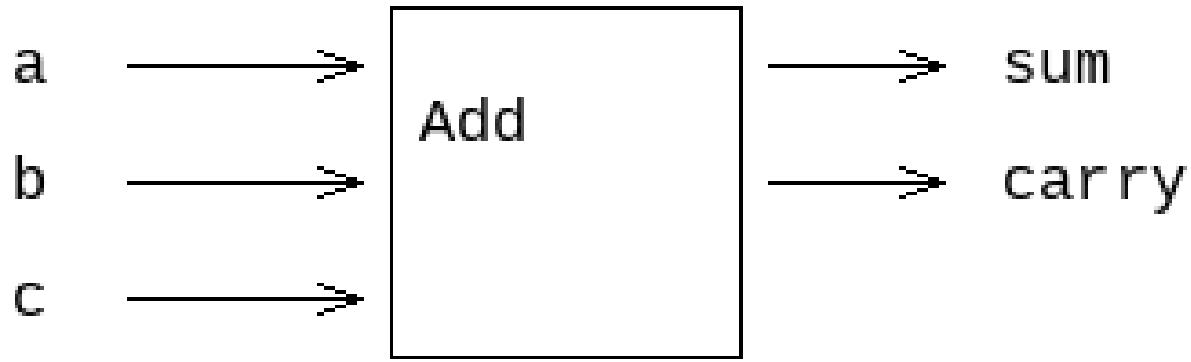
benötigte Bausteine:

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

Ergebnis ist ein *Halbaddierer*



Erweiterung



Erweiterung der Halbaddierer-Schaltung

- 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

$$
\begin{aligned}
(a \wedge b \wedge c) \quad &\vee \\
(a \wedge \neg b \wedge \neg c) \quad &\vee \\
(\neg a \wedge b \wedge \neg c) \quad &\vee \\
(\neg a \wedge \neg b \wedge c)
\end{aligned}
$$

Kanonische Repräsentationsform

Formel für carry:

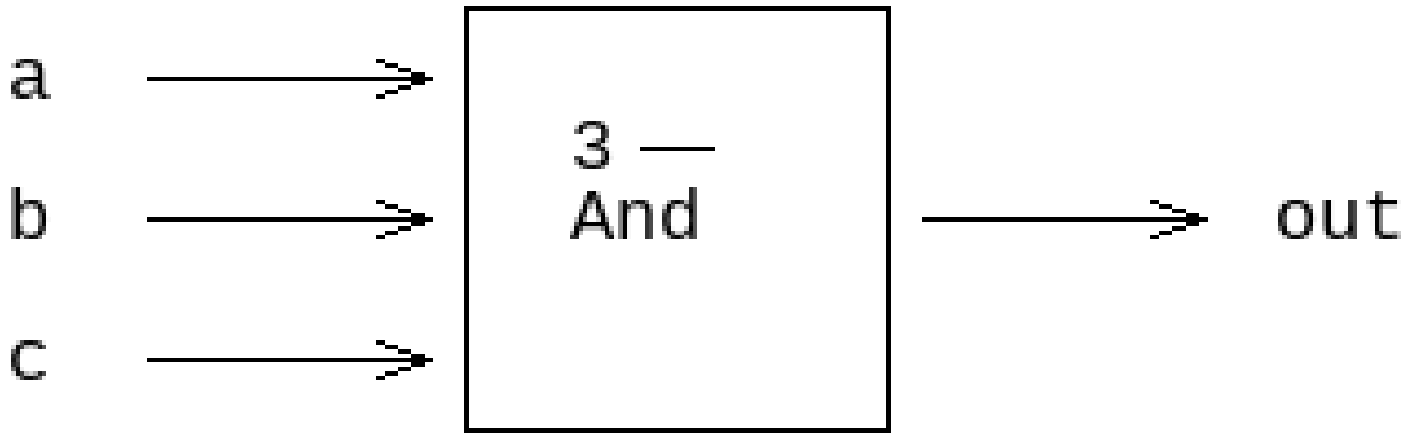
a	b	c	carry
1	1	1	1
1	1	0	1
1	0	1	1
0	1	1	1

$$
\begin{aligned}
(a \wedge b \wedge c) \quad & \vee \\
(a \wedge b \wedge \neg c) \quad & \vee \\
(a \wedge \neg b \wedge c) \quad & \vee \\
(\neg a \wedge b \wedge c) \quad &
\end{aligned}
$$

Problem: And mit drei Eingängen?

Umsetzung

And-Baustein mit drei Eingängen:



Umsetzung von 3-And?

Umsetzung

eine Möglichkeit: zwei And-Bausteine verknüpfen

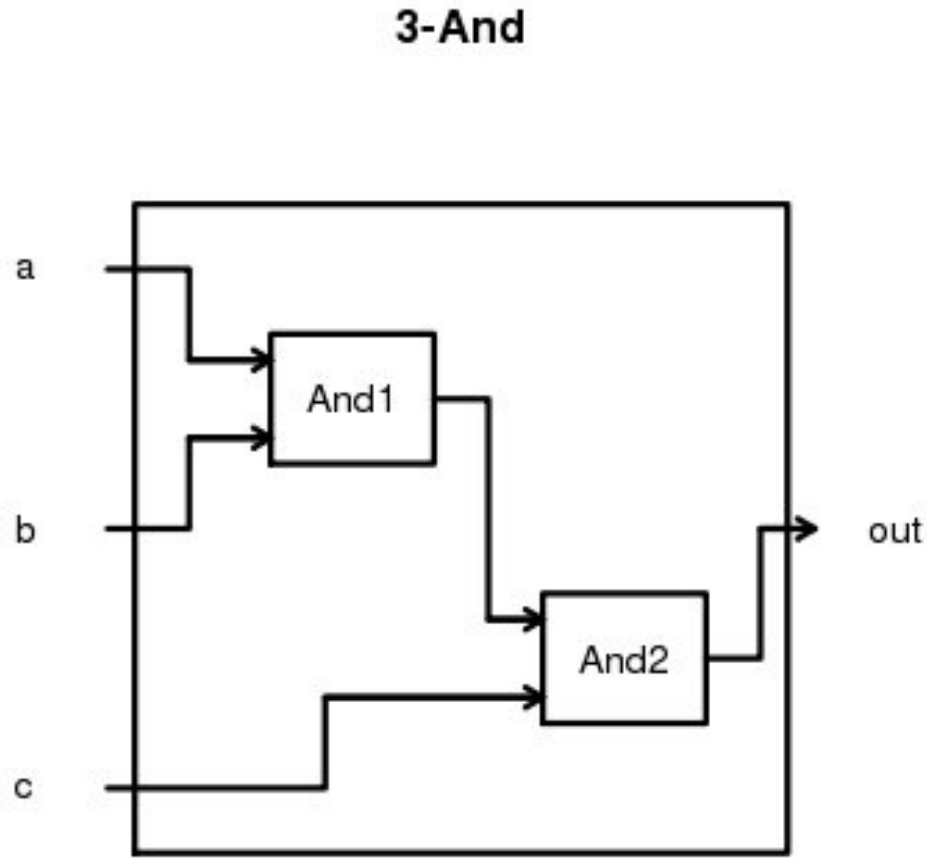


Figure 6: And-Addierer

Umsetzung des Volladdierers

- alle benötigen Teile liegen vor:
  - 3-And und Or-Bausteine
  - Eingänge ggf. mit Not-Baustein anpassen
- Problem: Für sum und carry brauchen wir dann jeweils 8 And + 3 Or + 3 Not = 14 Bausteine!

Verbesserungsmöglichkeiten?

Verbesserungen

Ausgangspunkte:

- Vereinfachen der logischen Formeln
- Xor als Baustein liegt vor

für carry:

```
carry =   abc + ab!c    | ab ausklammern, c + !c=1
        + a!bc + !abc
      = ab
        + a!bc + !abc   | c ausklammern
      = ab
        + c(a!b + !ab)  | hier steht ein Xor
```

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

benötigte Bausteine: 2 And, 1 Or, ein Xor

Verbesserungen

für sum:

```
sum = abc
      + a!b!c + !ab!c   | !c ausklammern
      + !a!bc
    = abc               | c ausklammern
      + !c(a!b + !ab)
      + !a!bc           |  "   "
    = c(ab + !a!b)
      + !c(a!b + !ab)   | Xor
    = c(ab + !a!b)      | Xnor (Not Xor)
      + !c(Xor(a, b))
    = c(¬Xor(a, b)
      + !c(Xor(a, b))
```

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

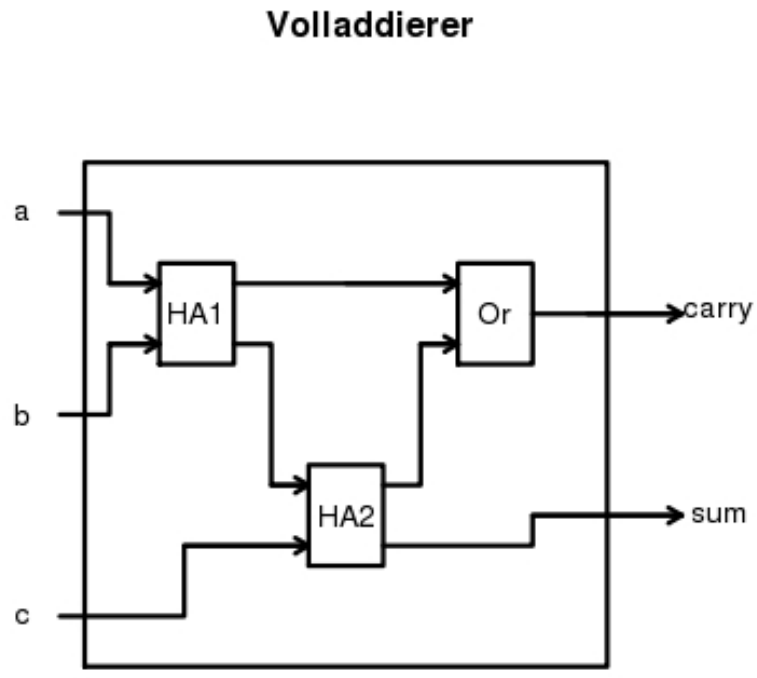
und: $x \wedge \neg y \vee \neg x \wedge y$ ist auch ein Xor!

$\rightarrow$ sum = Xor(Xor(a,b), c)

Umsetzung des Volladdierers

Genau betrachtet haben wir so nur zwei Halbaddierer kombiniert!

- Bausteine: Xor, And
- 2 Halbaddierer mit Or verknüpfen $\rightarrow$ Volladdierer



Umsetzung des Volladdierers: Details

- HA1 = a + b
- HA2 = sum + c

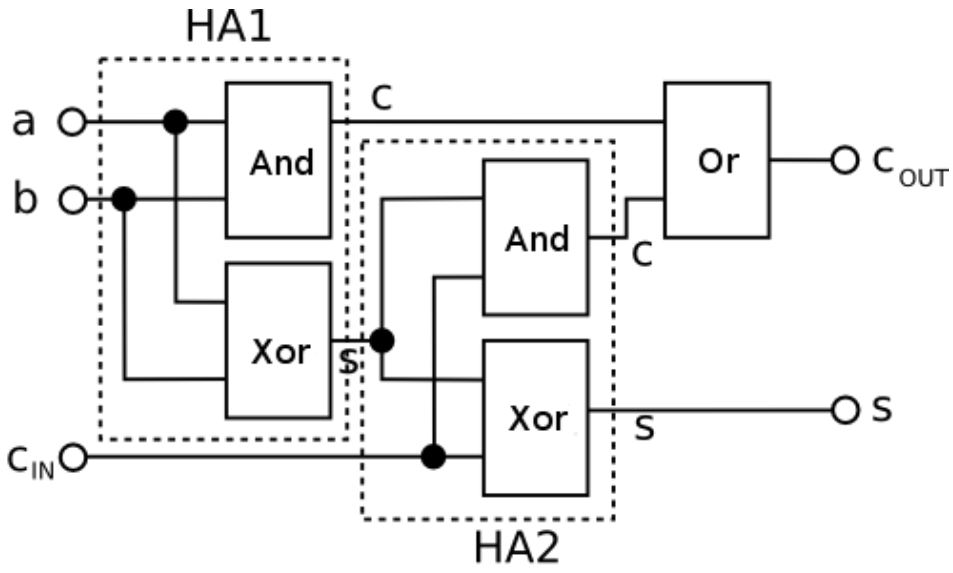


Figure 8: Volladdierer

4-Bit-Addierer

Idee: 4 Volladdierer zusammenschalten

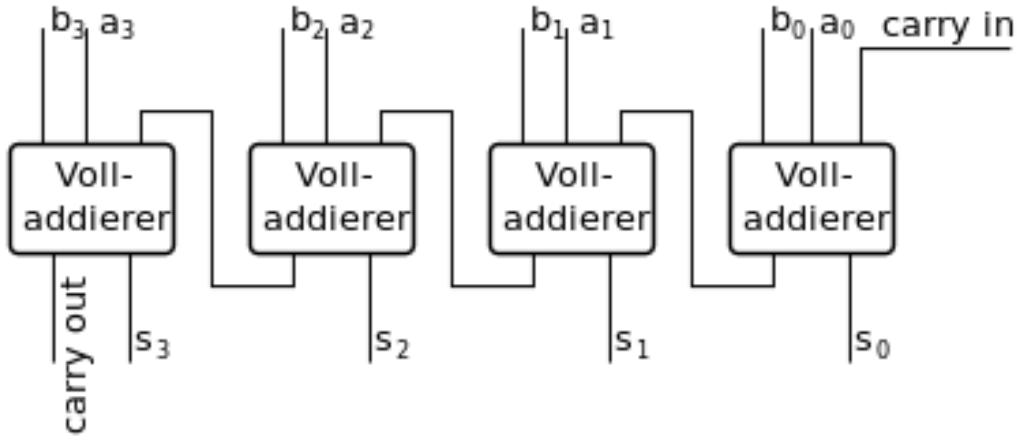


Figure 9: 4-Bit-Addierer

Ende