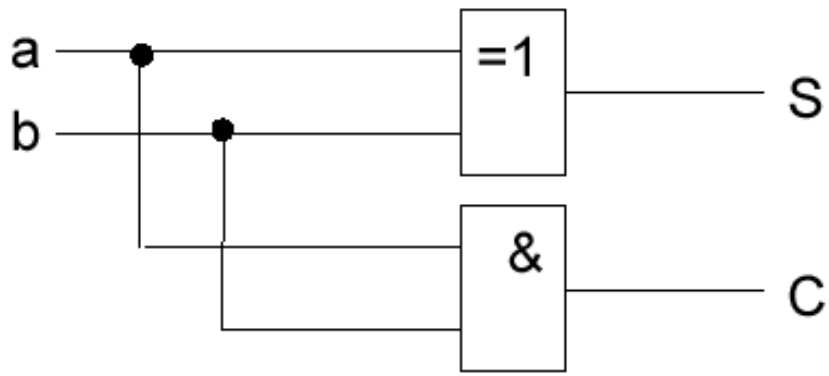
# Addierer

Signalverarbeitung 1

# Verschiedene Addierer

- Halb-, Volladdierer
- Ripple-Carry-Addierer
- Carry-Lookahead-Addierer
- Carry-Select-Addierer

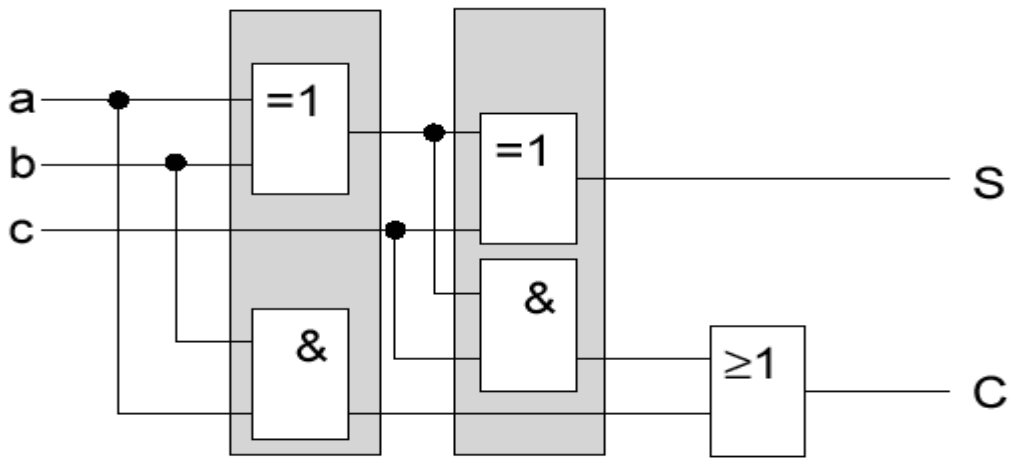
# Halbaddierer




S = a XOR b
C = a AND b

$$S = a \oplus b$$
$$C = a \cdot b$$

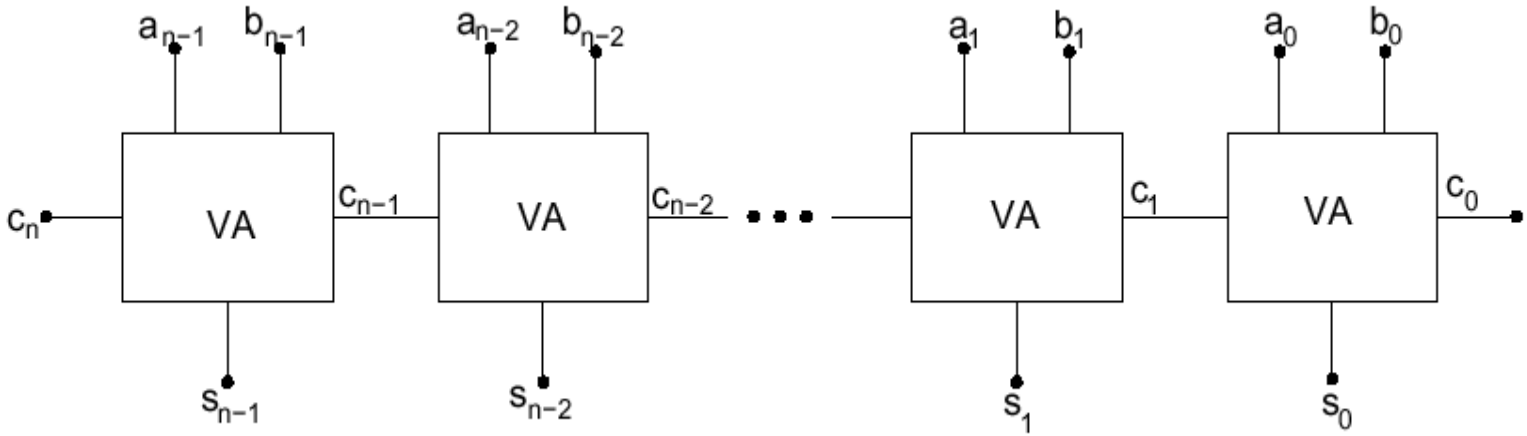
# Volladdierer




$$S = a \oplus b \oplus c$$

$$C = a \cdot b + (a \oplus b) \cdot c$$

# Ripple-Carry-Addierer




Carry – Übertrag
Ripple - rieseln

# Carry-Lookahead-Addierer

Grund für die schlechte Laufzeit des Ripple-Carry-Addierers ist die Berechnung des Übertrags c.

Lösung: Vorrausschauende (lookahead) Berechnung des Übertrags unter Ausnutzung folgender Grundsätze:

-Jedes Bitpaar „generiert“ einen Übertrag

$$a_i \cdot b_i = 1$$

-oder „propagiert“ (leitet durch) einen Übertrag

$$(a_i \oplus b_i) \cdot c_i = 1$$

# Carry-Lookahead-Addierer

Zur Berechnung des Übertrags c kann man folgende rekursive Formel aufstellen:

$$c_i = g_{i-1} + p_{i-1} \cdot c_{i-1}$$

mit „generate“

$$g_i = a_i \cdot b_i$$

und „propagate“

$$p_i = a_i \oplus b_i$$

# Carry-Lookahead-Addierer

Für einen 4-Bit-Carry-Lookahead-Addierer (CLA-Addierer) ergeben sich die folgenden Gleichungen für die Berechnung der Überträge:

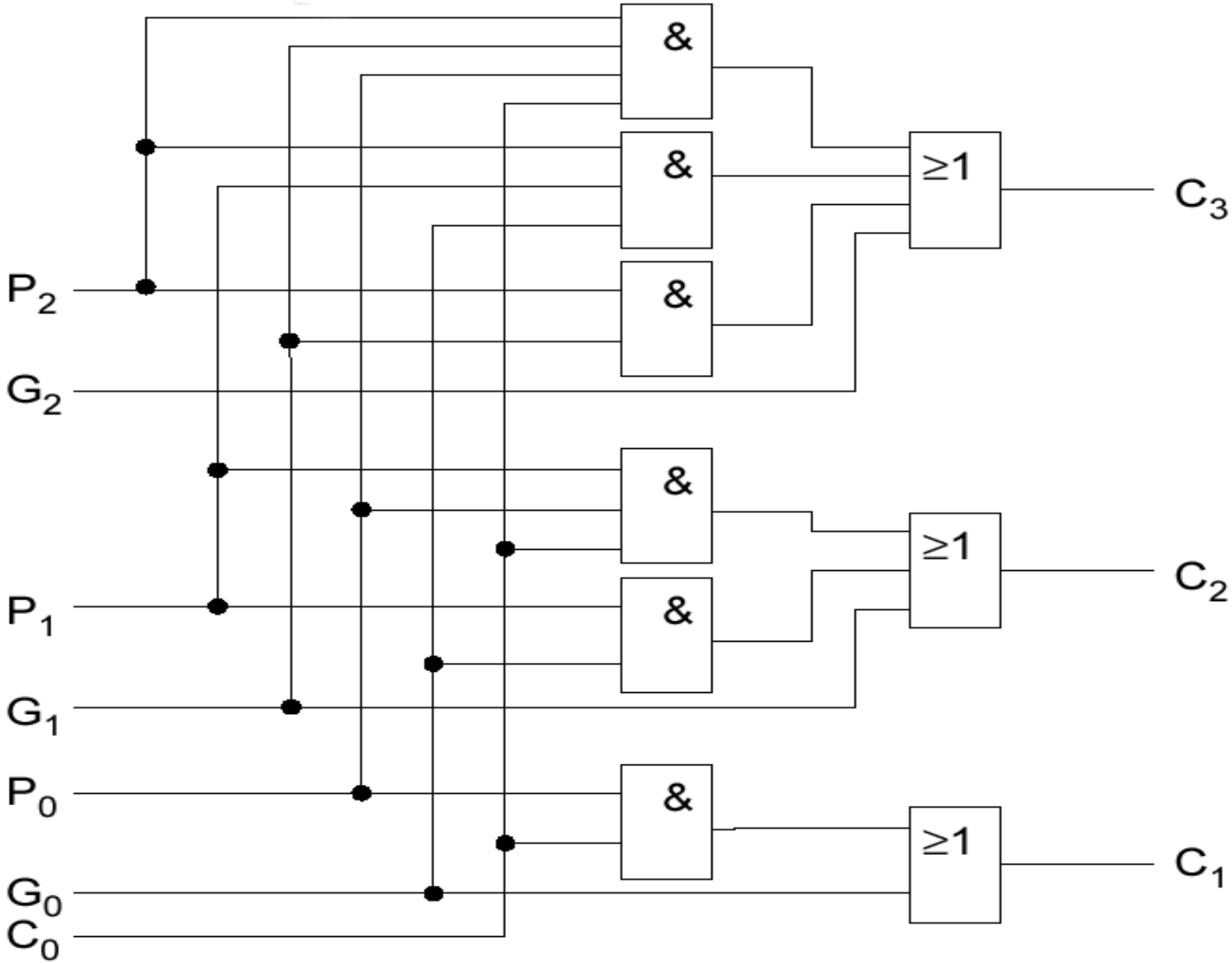
$$c_1 = g_0 + p_0 \cdot c_0$$

$$c_2 = g_1 + p_1 \cdot c_1 = g_1 + p_1(g_0 + p_0 \cdot c_0) = g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_0$$

$$c_3 = g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_0$$

$$c_4 = g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 + p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_0$$

# Diese Gleichungen können einfach realisiert werden:

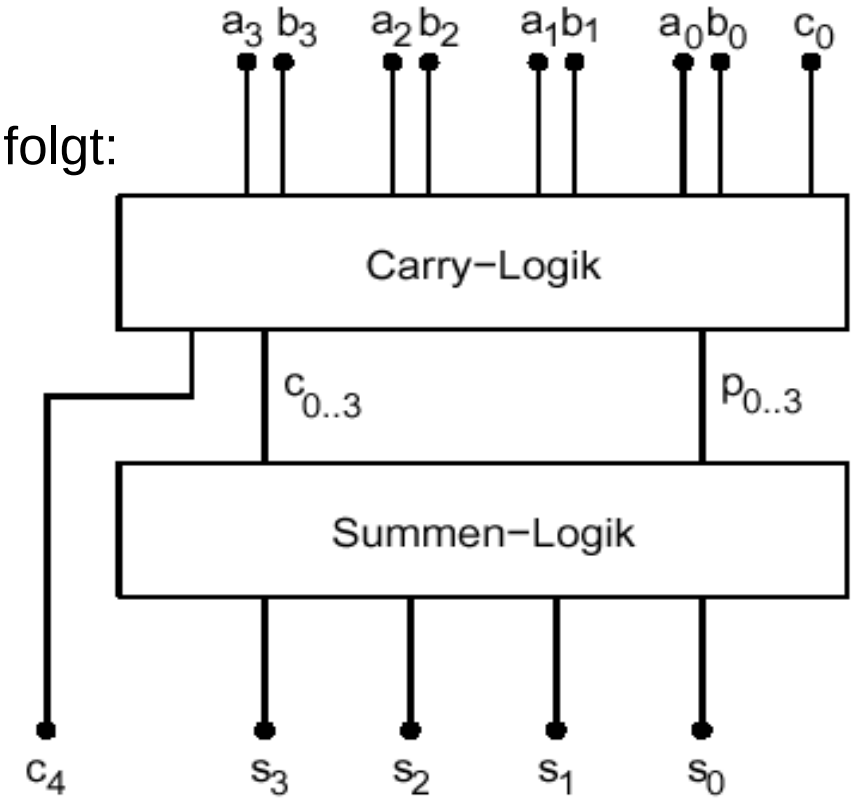


# Carry-Lookahead-Addierer

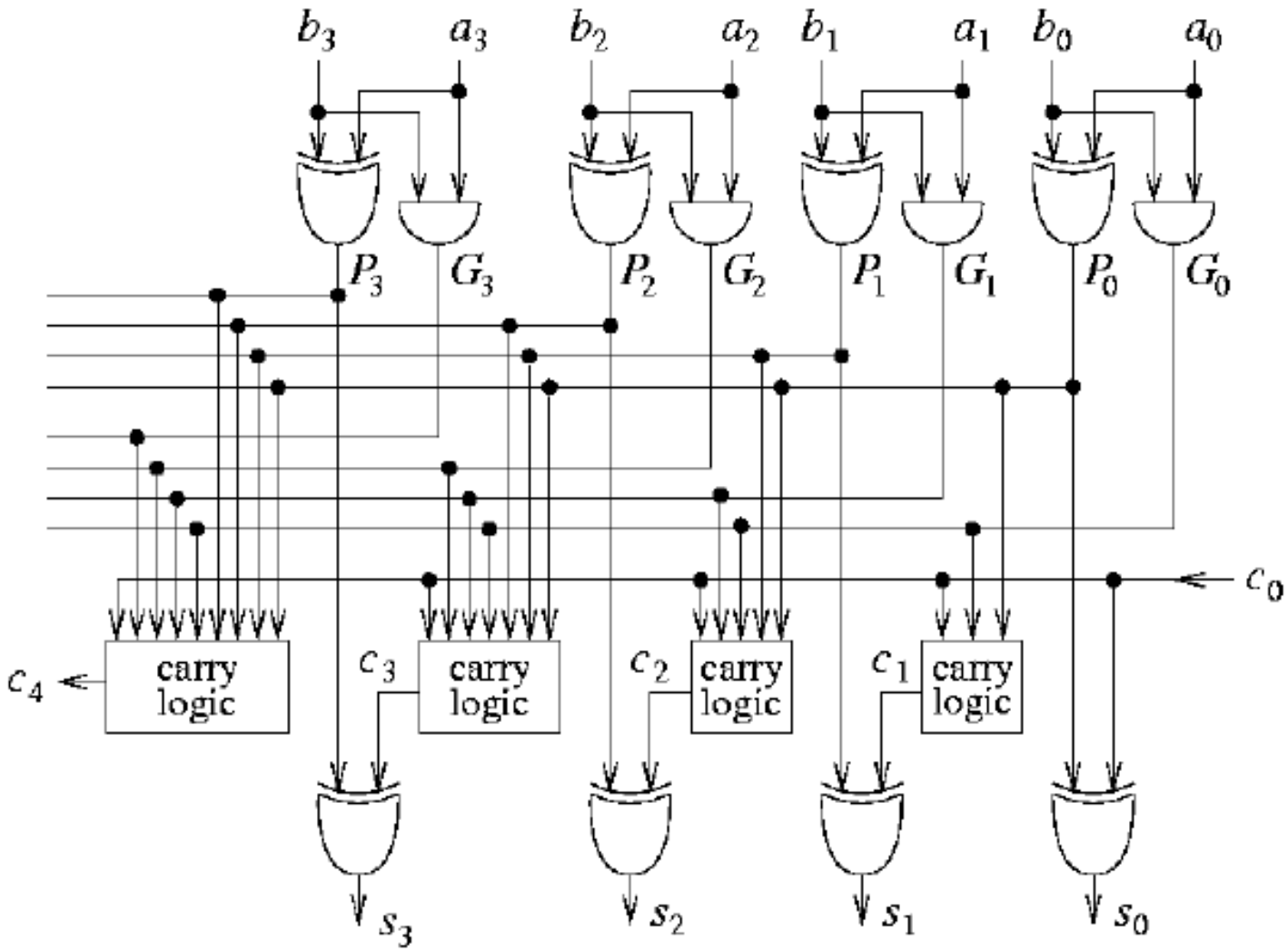
Die Summe berechnet sich wie folgt:

$$s_i = p_i \oplus c_i = a_i \oplus b_i \oplus c_i$$

Aus Carry-Logik und Summen-Logik wird der Carry-Lookahead-Addierer



# Bsp.: 4-Bit CLA

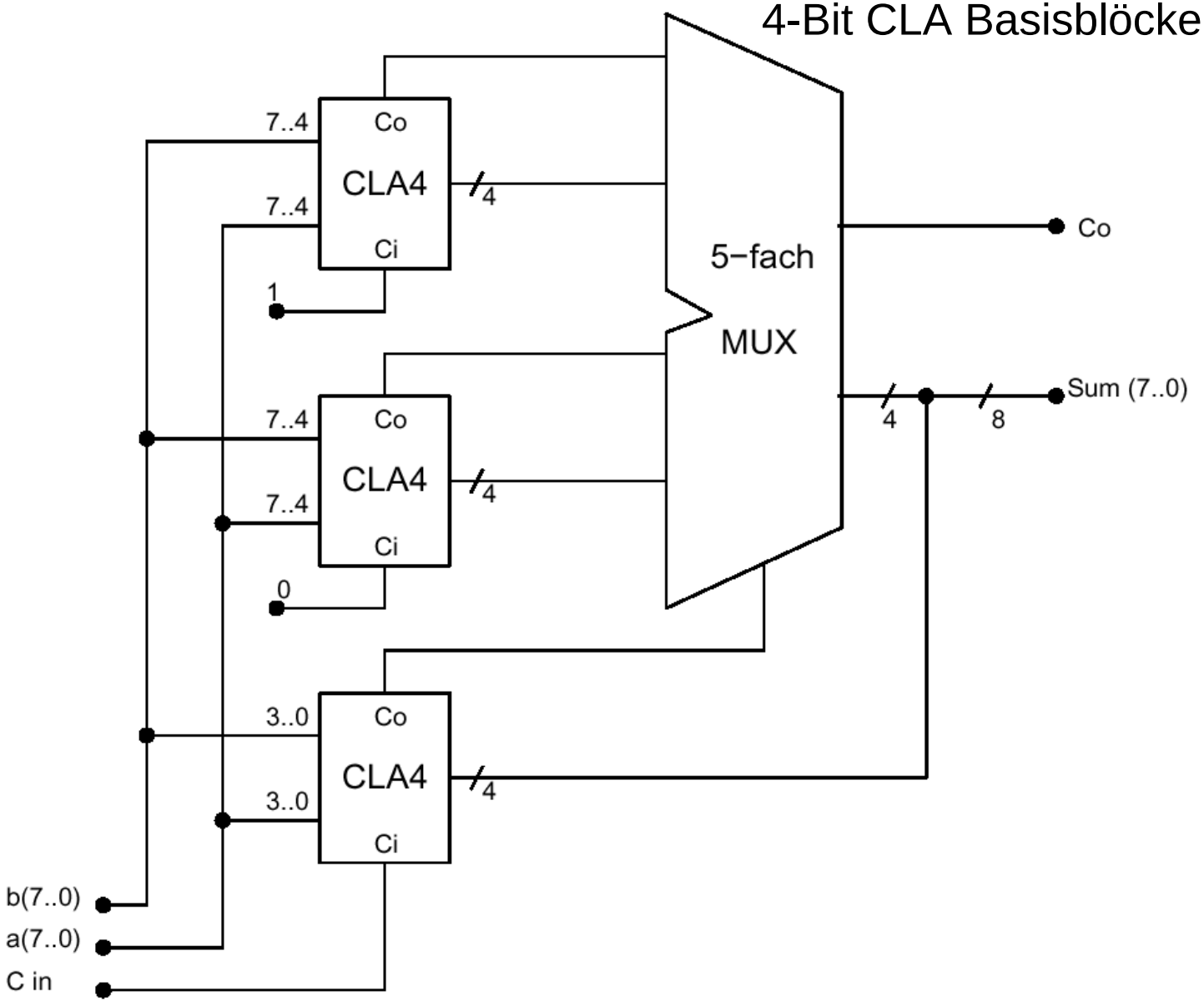

$b_3$
$a_3$
$b_2$
$a_2$
$b_1$
$a_1$
$b_0$
$a_0$
$P_3$
$G_3$
$P_2$
$G_2$
$P_1$
$G_1$
$P_0$
$G_0$
$c_0$
$c_4$
carry logic
$c_3$
carry logic
$c_2$
carry logic
$c_1$
carry logic
$s_3$
$s_2$
$s_1$
$s_0$

# Carry-Select-Addierer

Strategie:

- Wort zweiteilen, zweiten Teil zweimal berechnen
- Einmal mit Cin=1, einmal mit Cin=0
- Mittels Multiplexer „richtigen“ zweiten Teil auswählen

# 8-Bit Carry-Select-Addierer mit 4-Bit CLA Basisblöcken



# Vergleich

| | Ripple-Carry | CLA | Carry-Select<br>mit CLA Blöcken |
|---|---|---|---|
| Zeit | **3*n** Gatterlaufzeiten<br>(n = Wortbreite) | immer **4**<br>Gatterlaufzeiten | immer **6**<br>Gatterlaufzeiten |
| Aufwand | gering<br>(bitslice) | hoch<br>(für grosse Wortbreite enorm) | mittel<br>(ab 16-Bit besser als CLA) |

Quellen: Wikipedia, Skripte der Uni Ulm, Duisburg und der FH Karlsruhe