Einführung in die Technische Informatik

Prof. Dr.-Ing. Stefan Kowalewski

WS 22/23

Kapitel 9: Rechenstrukturen

Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

Abschnitt 9.1

Addiernetze

- Halbaddierer
- Volladdierer
- Ripple-Carry-Adder
- Carry-Bypass-Addiernetz
- Carry-Save-Addiernetz
- Wallace-Tree

Recap: Schriftliche Addition

Dezimal

$$
\begin{array}{rr}
183 & x \\
+\ 997 & y \\
1110 & U \\
\hline
1180 & R
\end{array}
$$

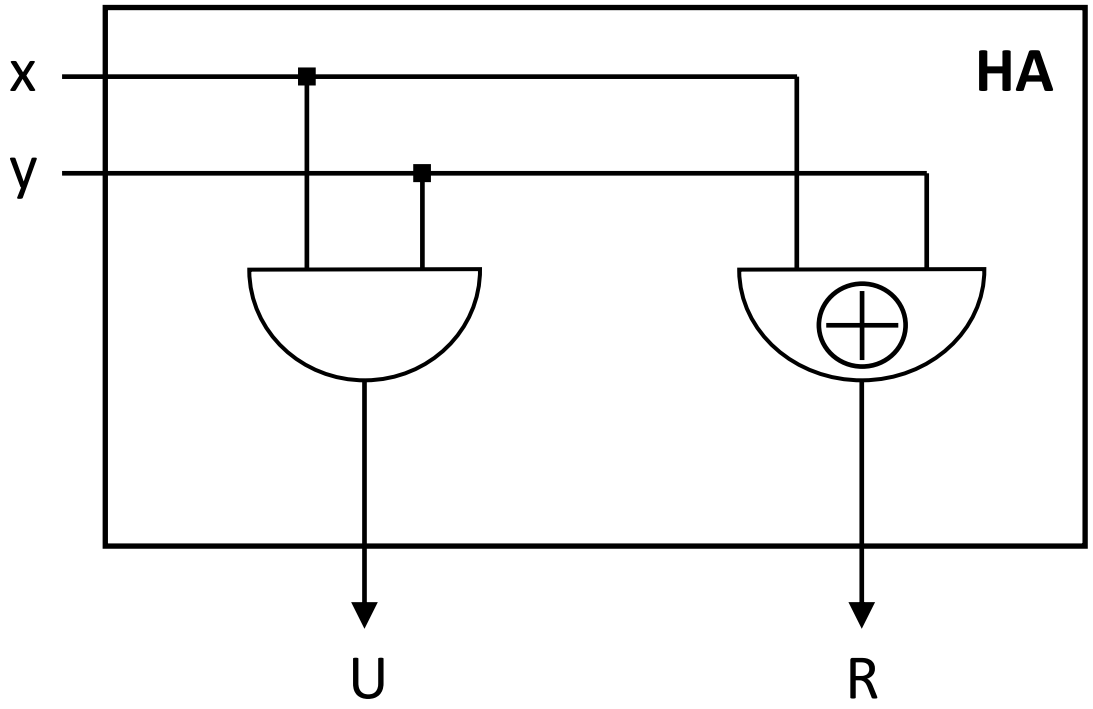
Binär

$$
\begin{array}{rr}
0111 & x \\
+\ 1101 & y \\
11110 & U \\
\hline
10100 & R
\end{array}
$$

2 Bits addiert

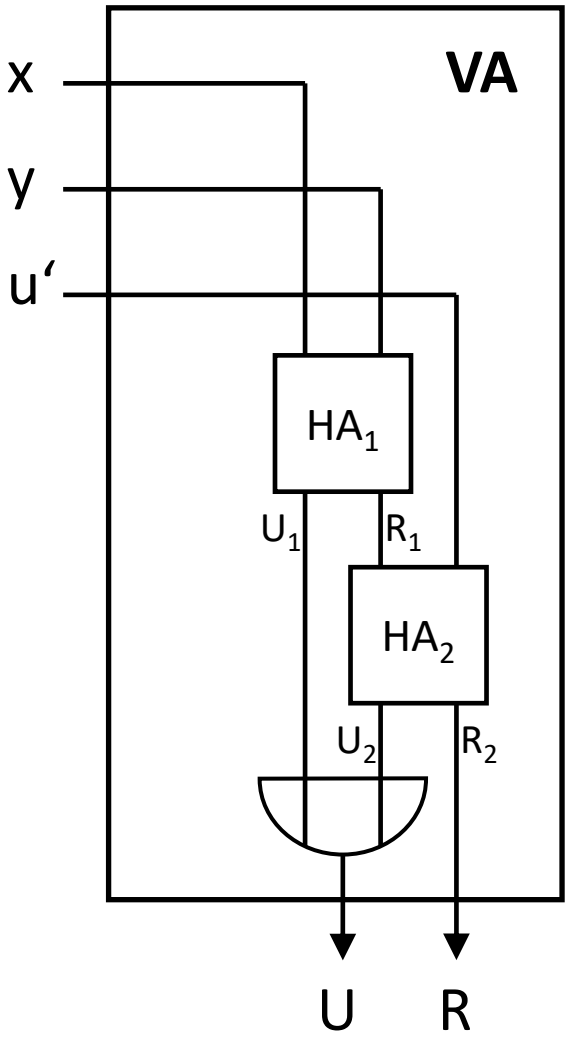
3 Bits addiert

Halbaddierer



x	y	Σ_{10}	U	R
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	2	1	0

Volladdierer

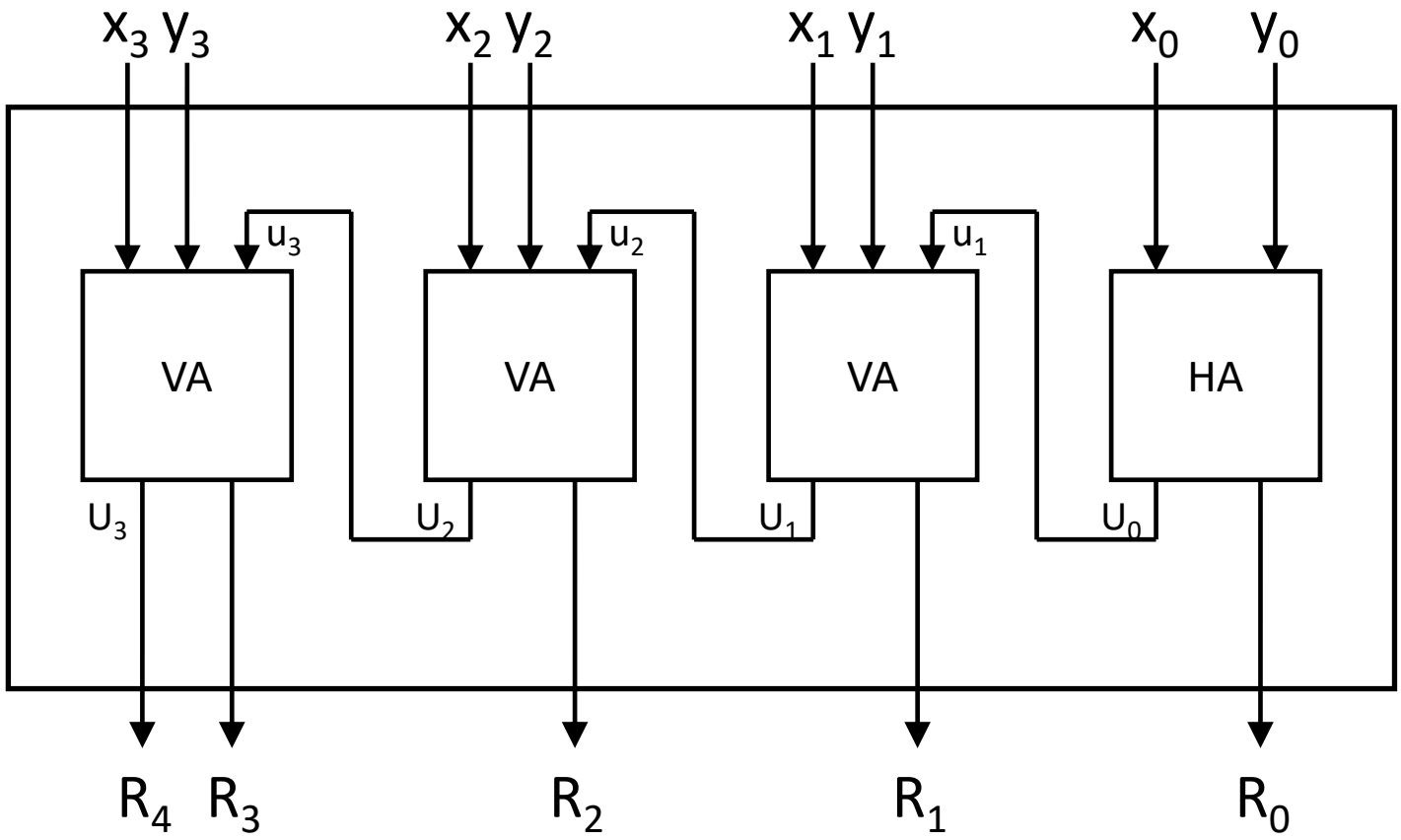


x	y	u’	Σ_{10}	U	R
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	2	1	0
1	0	0	1	0	1
1	0	1	2	1	0
1	1	0	2	1	0
1	1	1	3	1	1

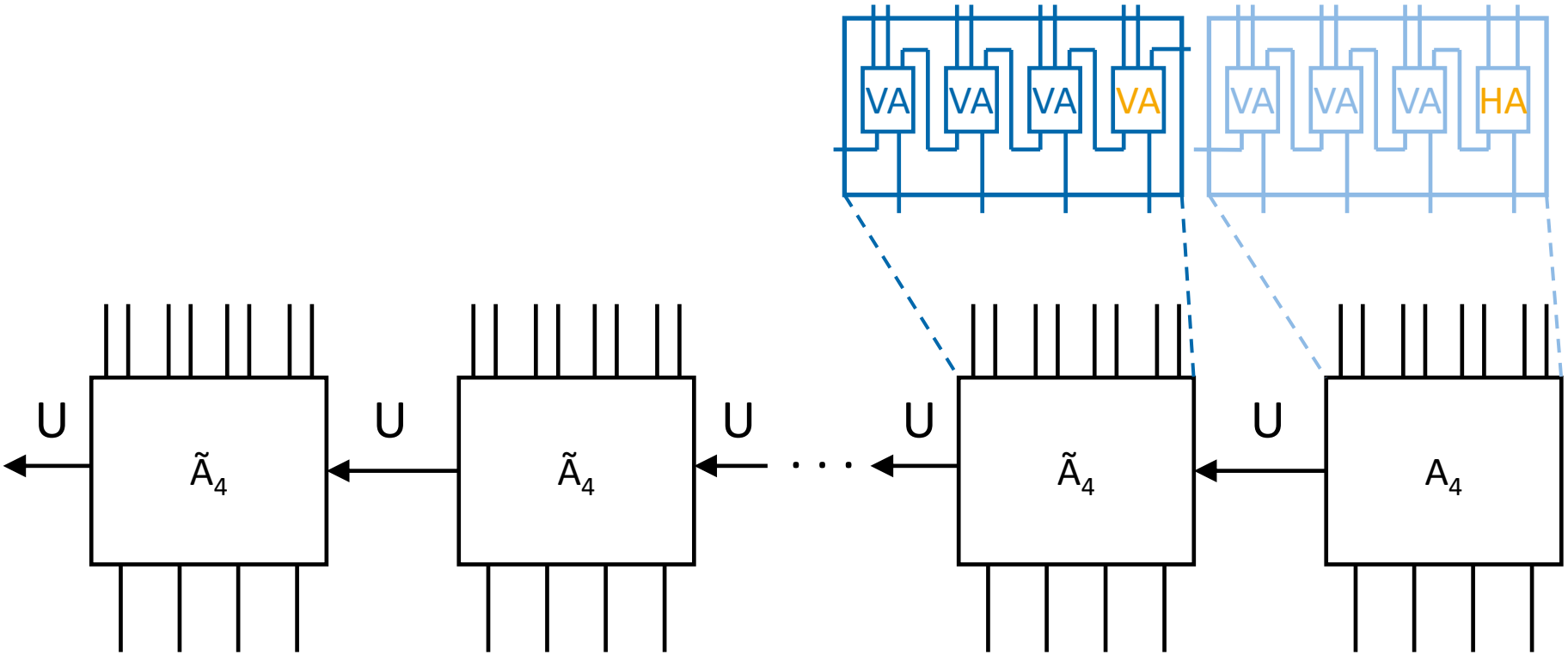
$$U_1 = x \cdot y \qquad R_1 = x \oplus y$$
$$U_2 = (x \oplus y) \cdot u' \qquad R_2 = (x \oplus y) \oplus u'$$
$$U = x \cdot y + (x \oplus y) \cdot u' \qquad R = (x \oplus y) \oplus u'$$

Addiernetz für zwei 4-stellige Dualzahlen

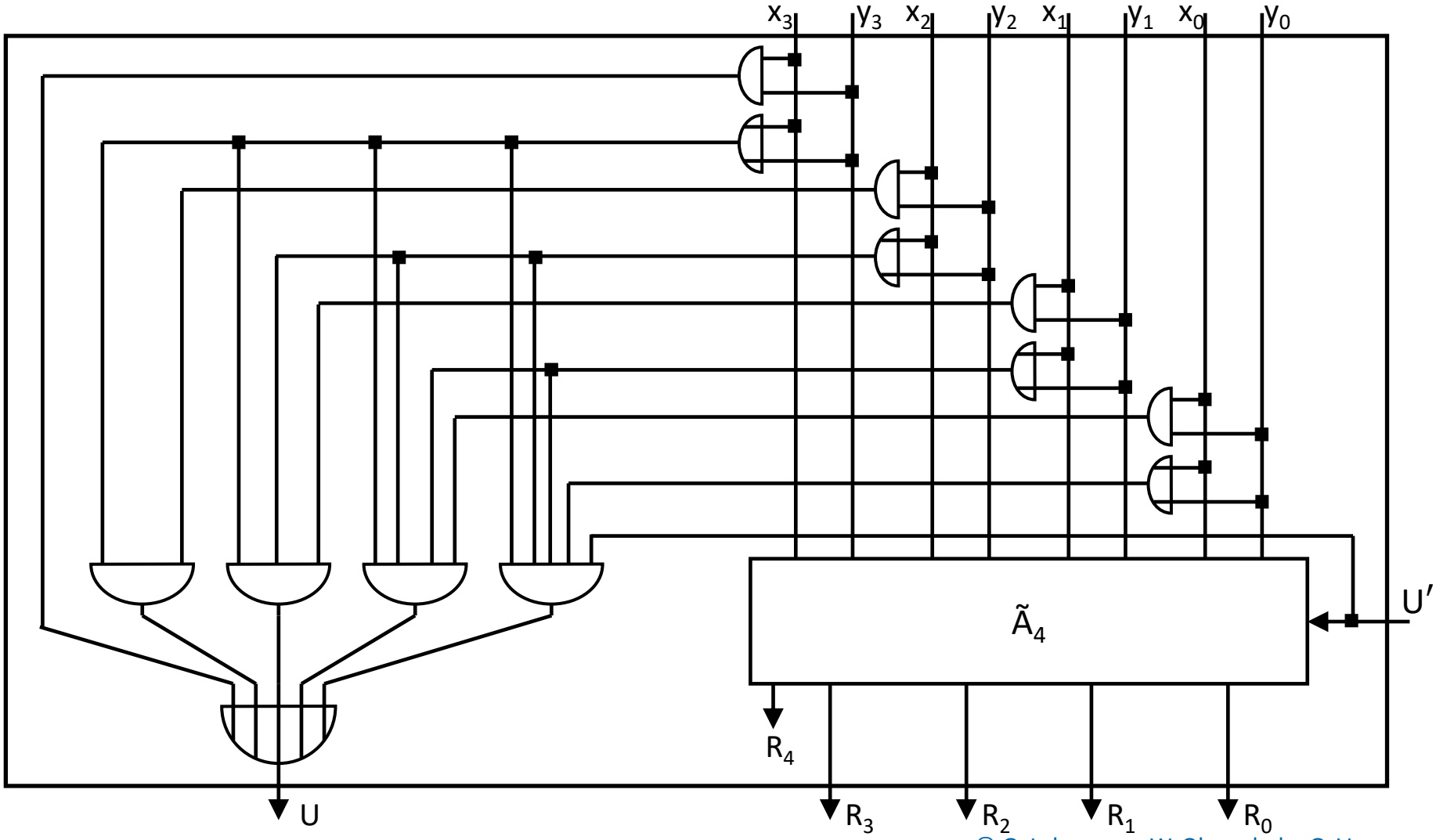
asynchrones (Parallel-) Addiernetz, Ripple-Carry-Addierer:



n-stelliges Addiernetz



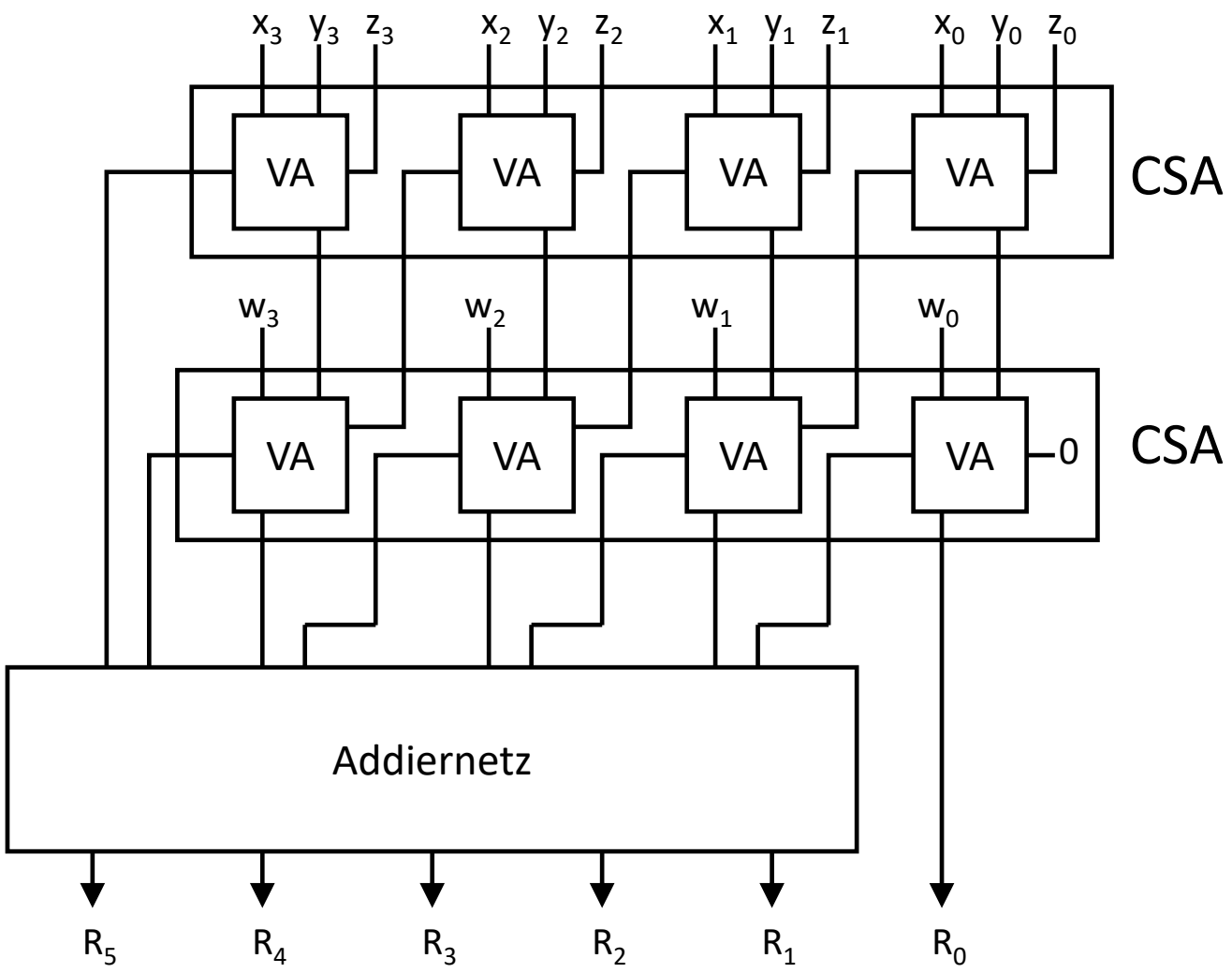
Carry-Bypass-Addiernetz



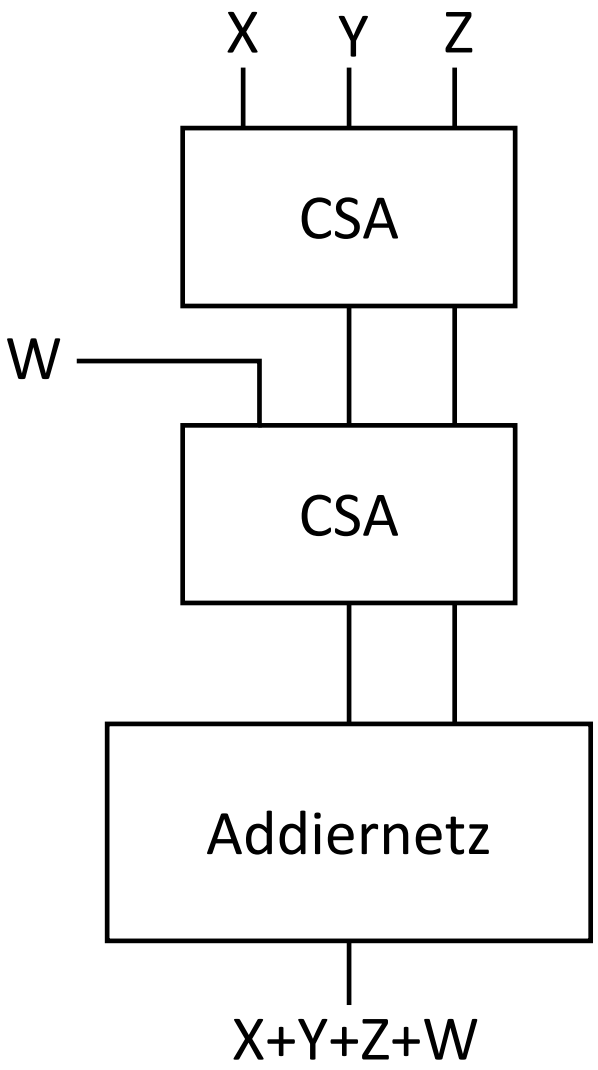
Carry-Bypass-Addiernetz

Carry-Save-Addiernetz

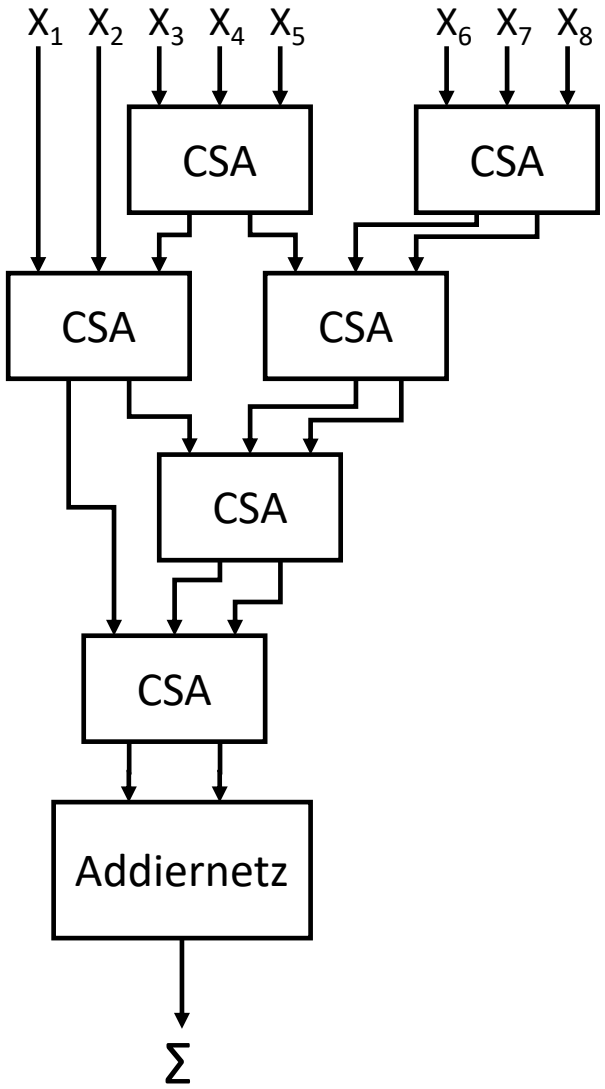
Carry-Save-Addiernetz



Prinzip der Carry-Save-Addition



Wallace-Tree



Abschnitt 9.2

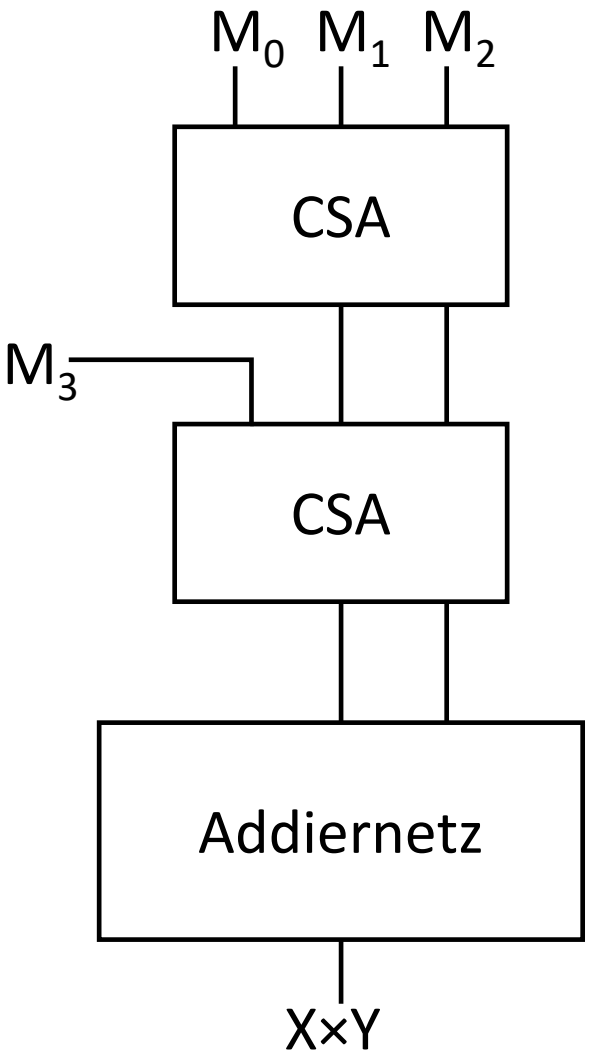
Multiplikation

- Carry-Save-Multiplikation
- Schaltung zur Multiplikation

Carry-Save-Multiplikation

			x_3	x_2	x_1	x_0	x
	×		y_3	y_2	y_1	y_0	y
0	0	0	$y_3 \cdot x_0$	$y_2 \cdot x_0$	$y_1 \cdot x_0$	$y_0 \cdot x_0$	M_0
0	0	$y_3 \cdot x_1$	$y_2 \cdot x_1$	$y_1 \cdot x_1$	$y_0 \cdot x_1$	0	M_1
0	$y_3 \cdot x_2$	$y_2 \cdot x_2$	$y_1 \cdot x_2$	$y_0 \cdot x_2$	0	0	M_2
$y_3 \cdot x_3$	$y_2 \cdot x_3$	$y_1 \cdot x_3$	$y_0 \cdot x_3$	0	0	0	M_3

Carry-Save-Multiplikation



Multiplikation (Wdh.)

Schulmethode:

Sei *x* der Multiplikand, $y = (y_{n-1}, \ldots, y_0)$ der Multiplikator, dann ist

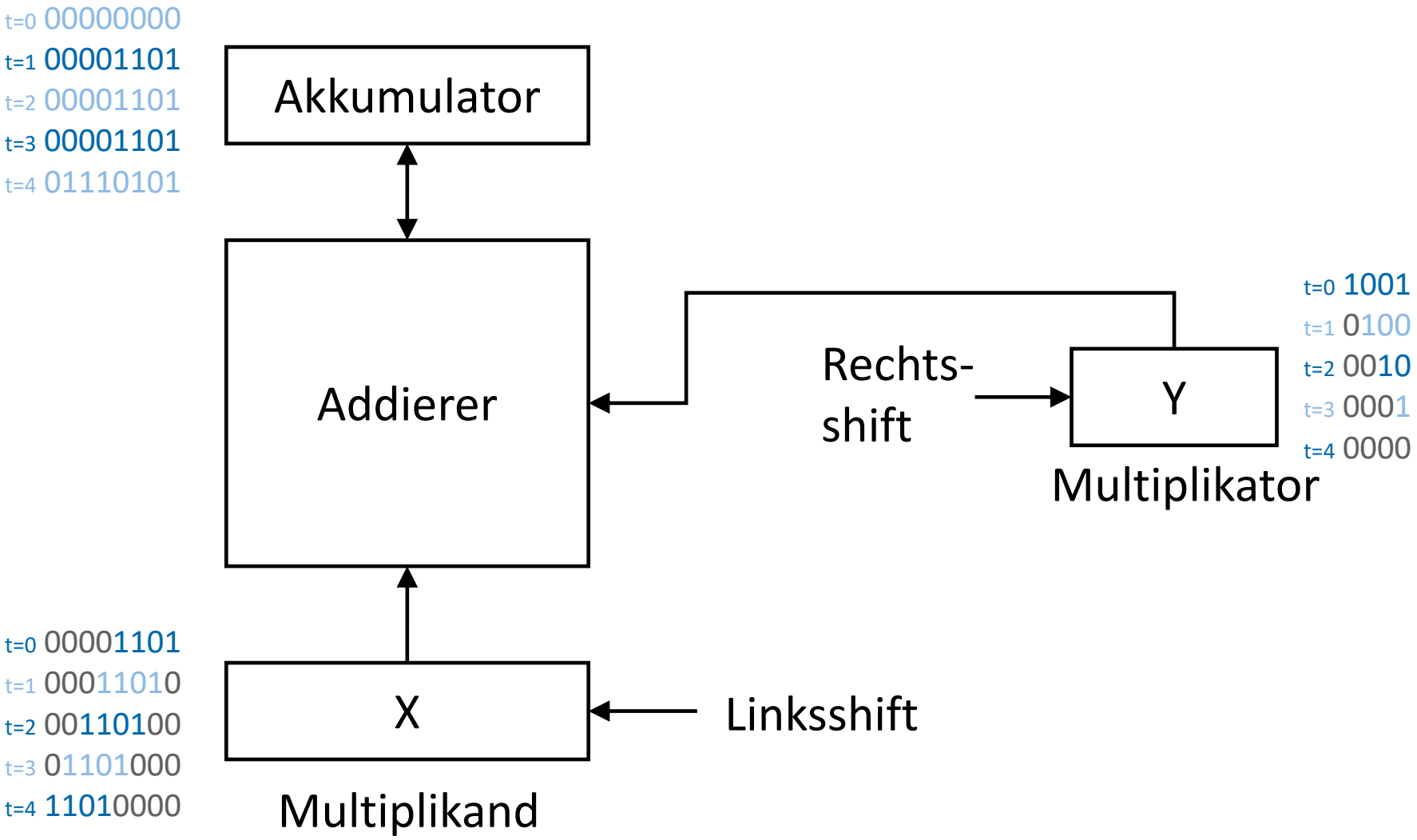
$$x \cdot y = x \cdot y_0 + x \cdot y_1 \cdot 2 + x \cdot y_2 \cdot 2^2 + \cdots + x \cdot y_{n-1} \cdot 2^{n-1}$$

$$= \sum_{i=0}^{n-1} x \cdot y_i \cdot 2^i$$

In der Praxis ist es sinnvoll, jeden Term der Form $x \cdot y_i \cdot 2^i$ zu addieren, sobald er generiert wurde:

$$x \cdot y = \left(\ldots\left(\left(x \cdot y_0 + x \cdot y_1 \cdot 2\right) + x \cdot y_2 \cdot 2^2\right) + \cdots\right) + x \cdot y_{n-1} \cdot 2^{n-1}$$

Schaltung zur Multiplikation

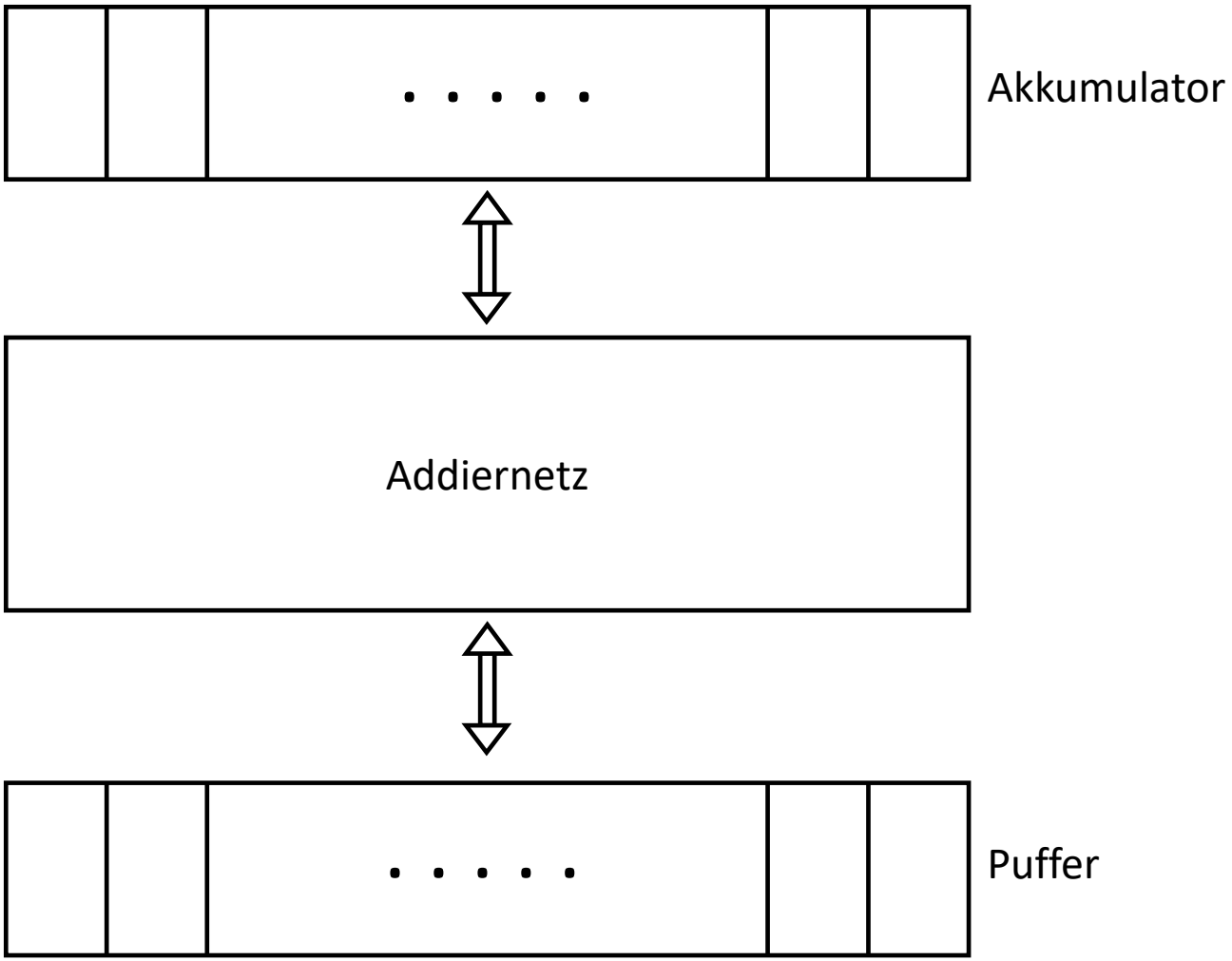


Abschnitt 9.3

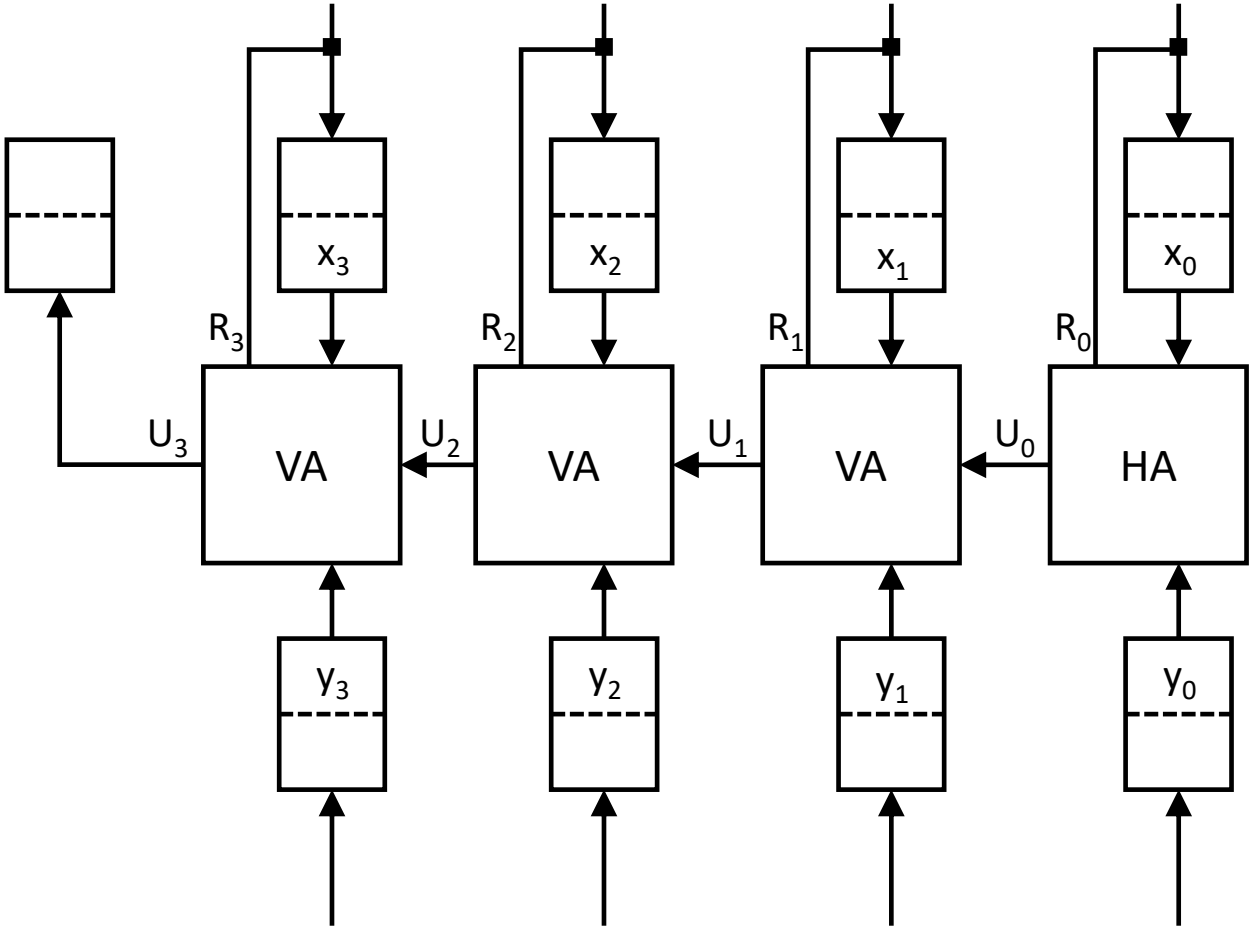
Addierwerke

- Parallel-Addierwerk
- Serien-Addierwerk
- von Neumann-Addierwerk

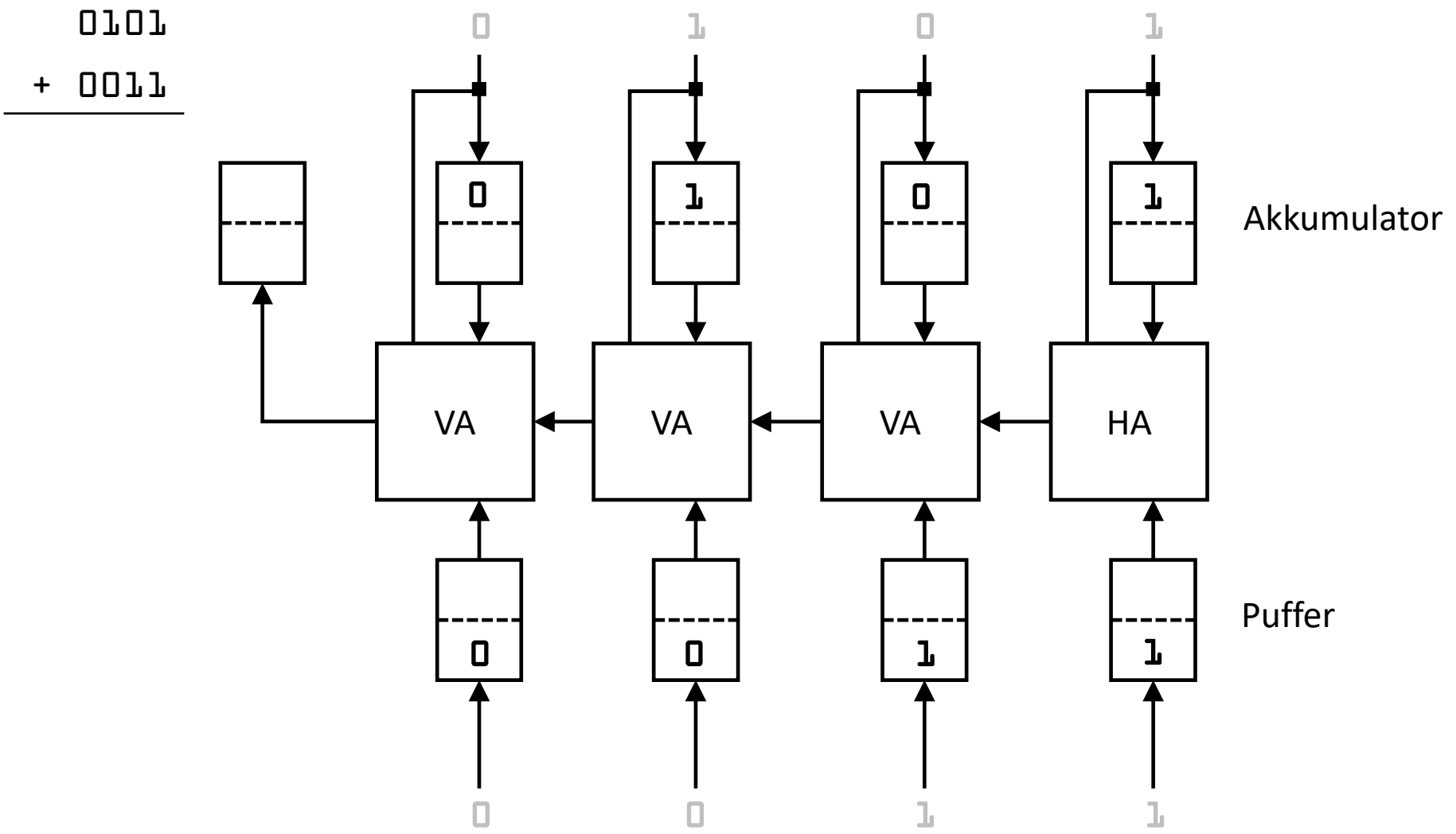
Addierwerk (Organisationsplan)



4-Bit-Parallel-Addierwerk (Ripple-Carry-Addierer mit D-Flipflops)

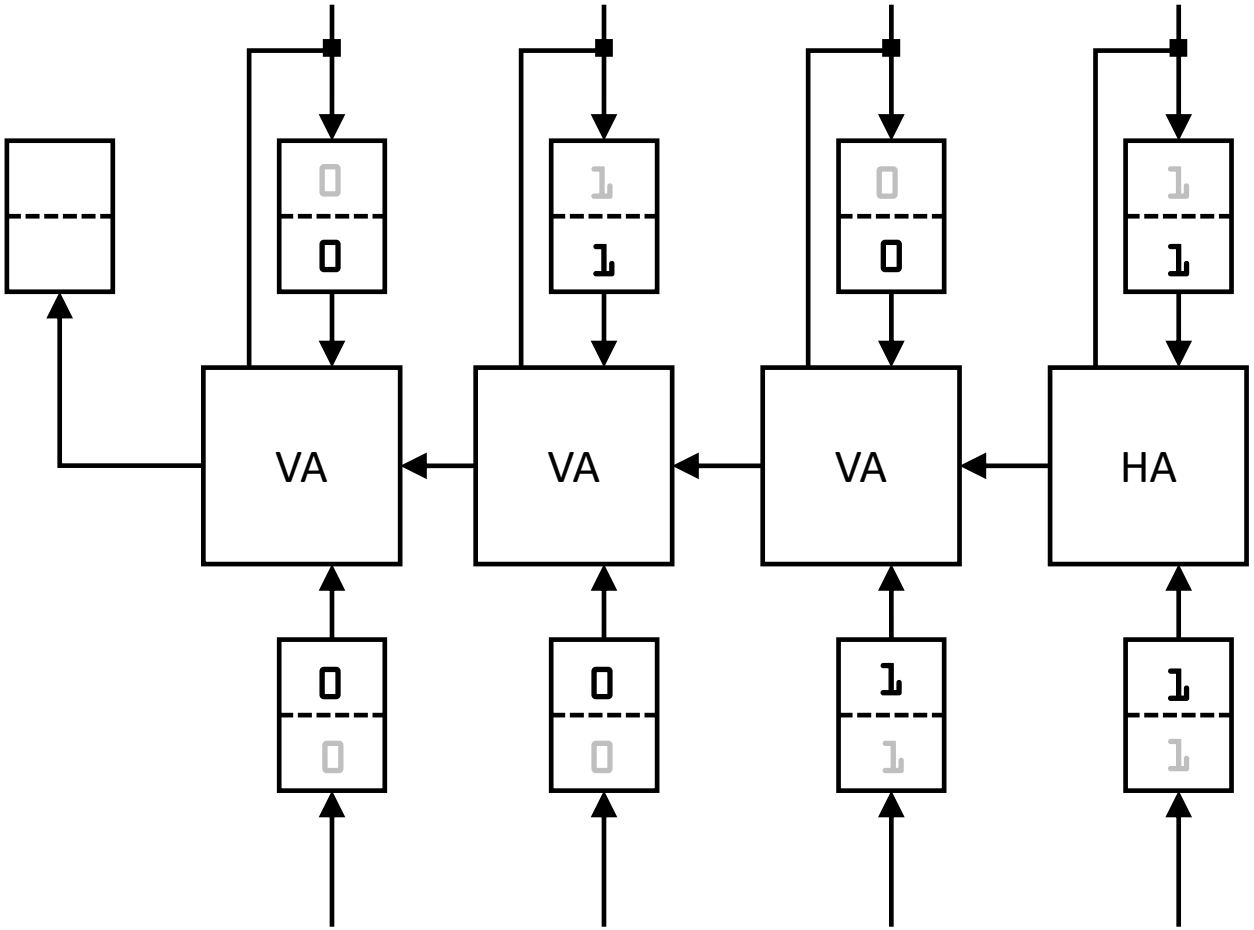


4-Bit-Parallel-Addierwerk (Ripple-Carry-Addierer mit D-Flipflops)



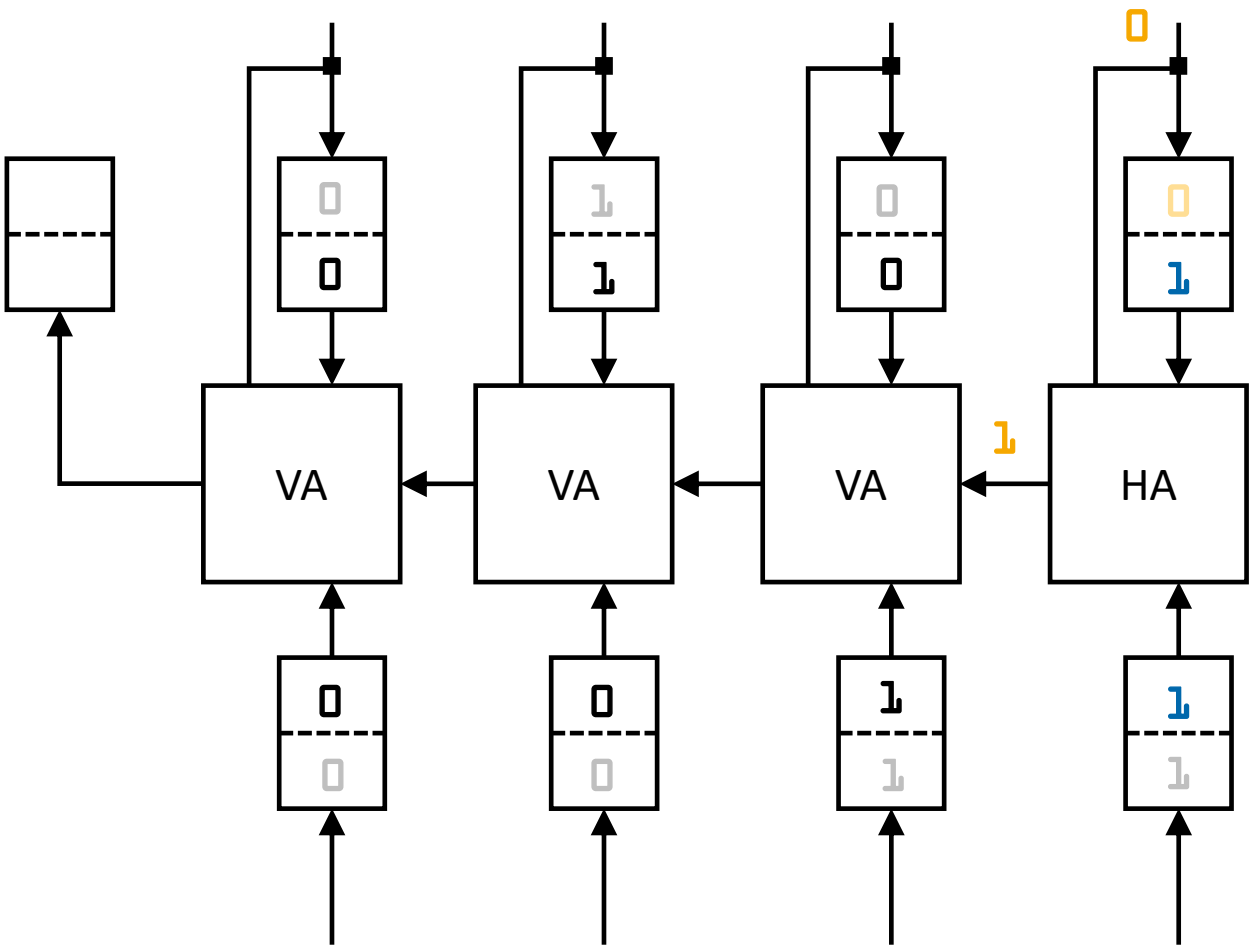
4-Bit-Parallel-Addierwerk (Ripple-Carry-Addierer mit D-Flipflops)

```
  0101
+ 0011
```



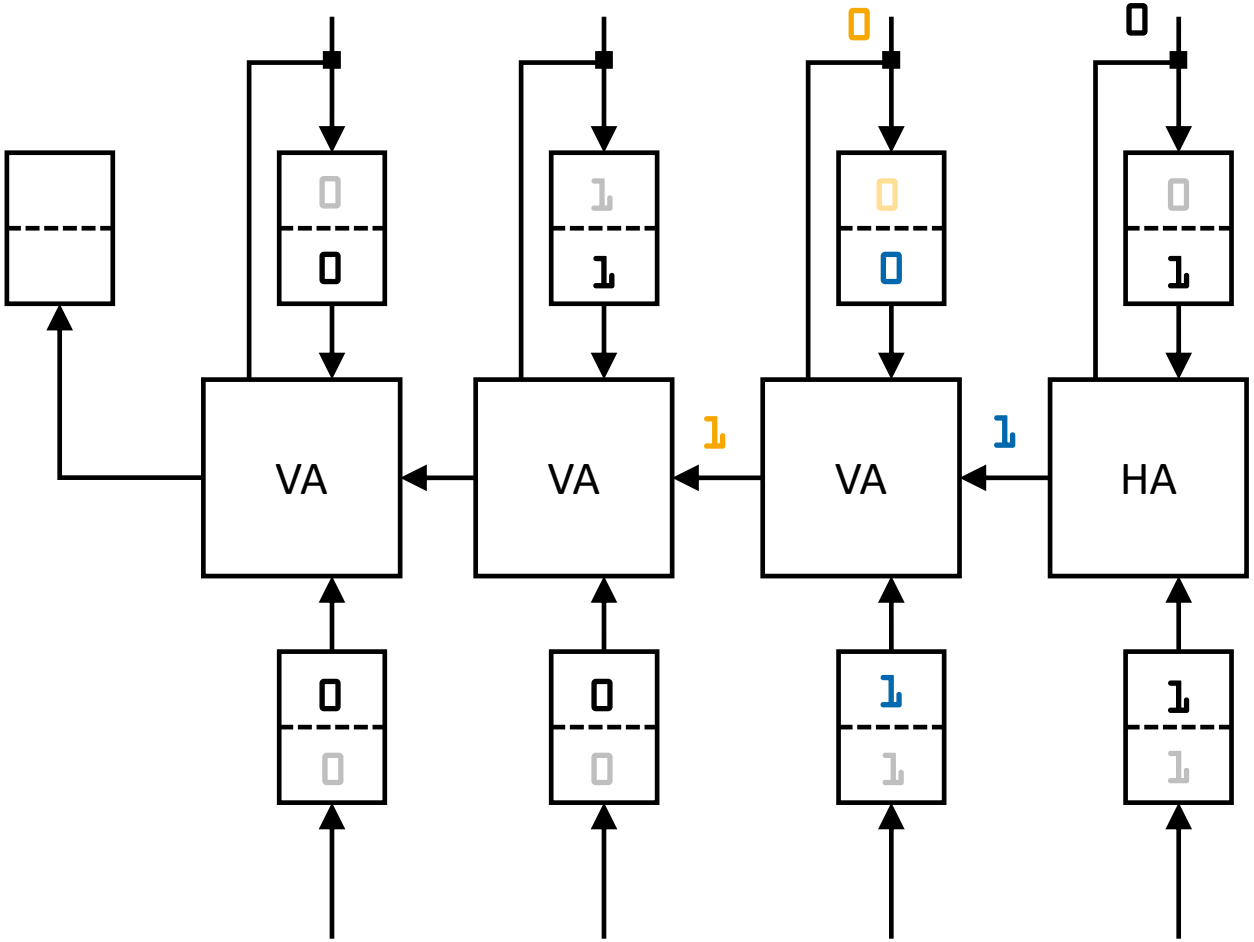
4-Bit-Parallel-Addierwerk (Ripple-Carry-Addierer mit D-Flipflops)

```
  0101
+ 0011
```



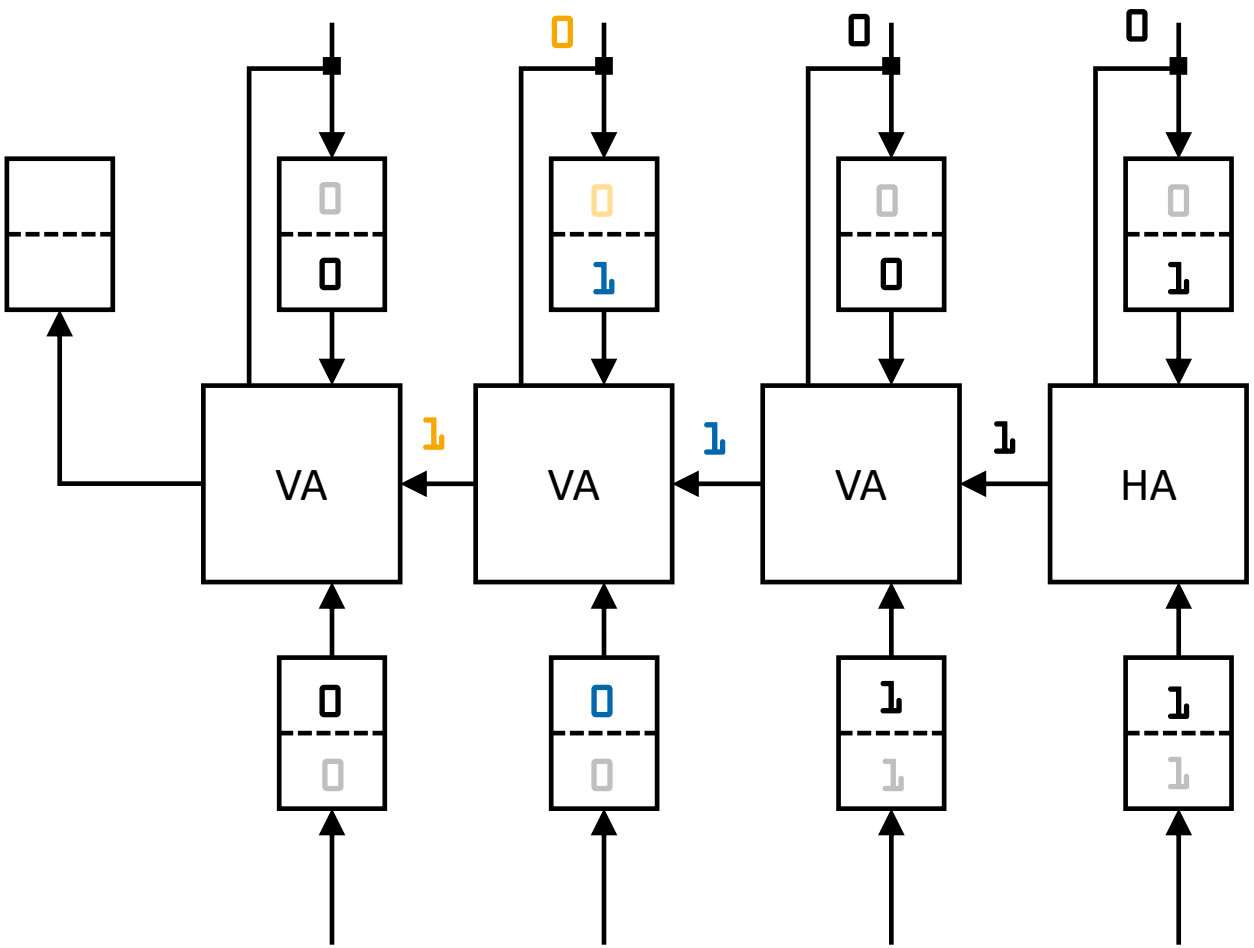
4-Bit-Parallel-Addierwerk (Ripple-Carry-Addierer mit D-Flipflops)

```
  0101
+ 0011
```



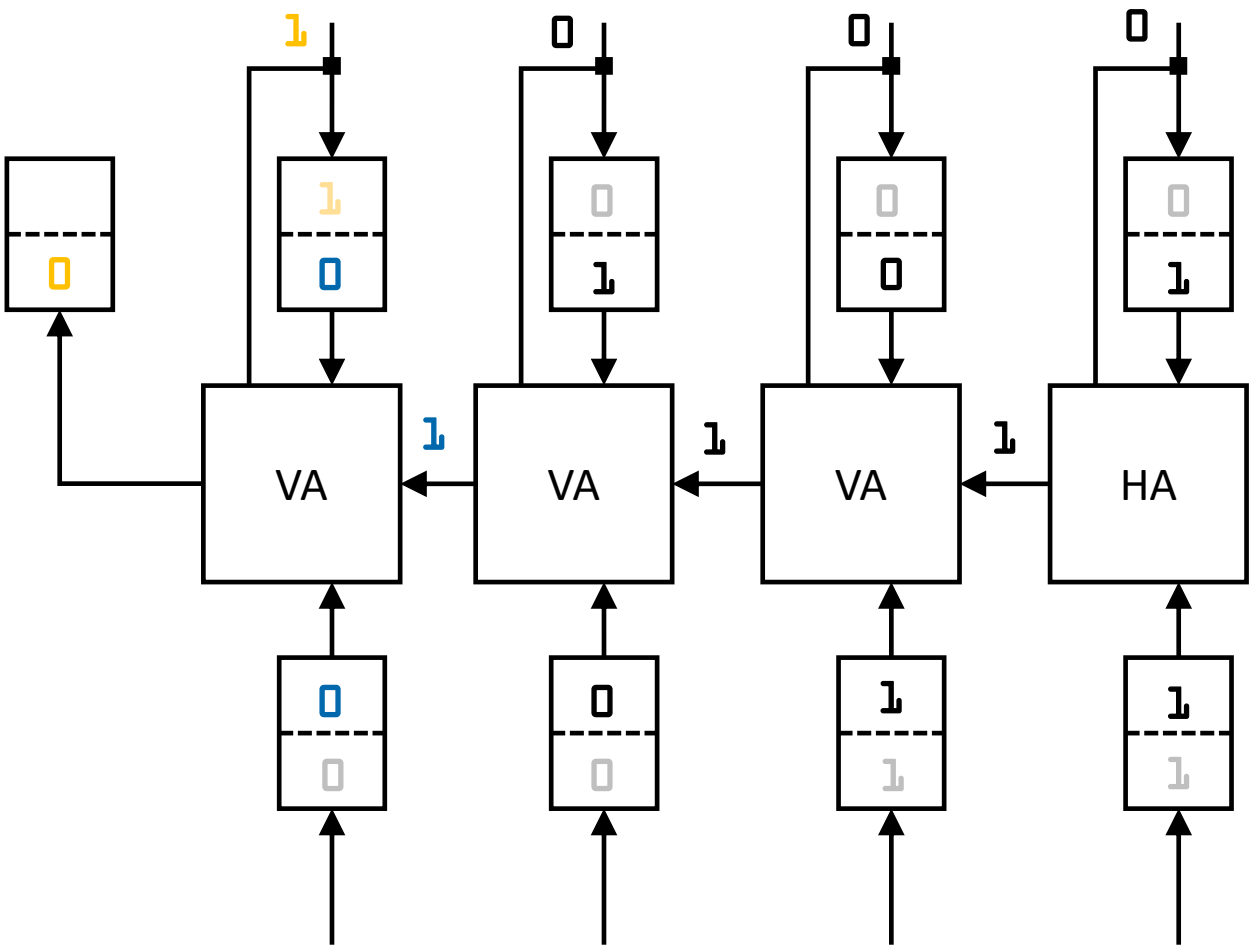
4-Bit-Parallel-Addierwerk (Ripple-Carry-Addierer mit D-Flipflops)

```
  0101
+ 0011
```



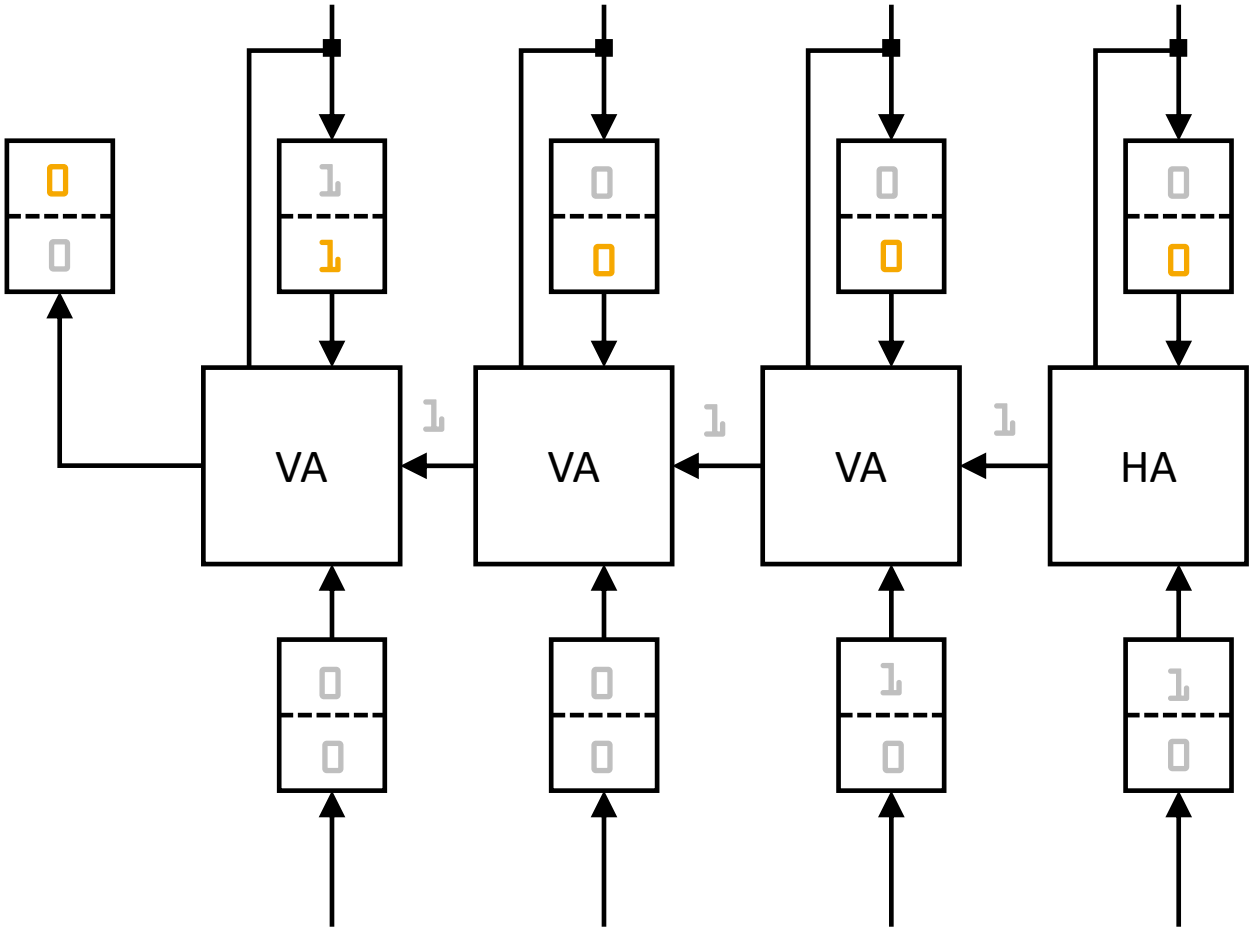
4-Bit-Parallel-Addierwerk (Ripple-Carry-Addierer mit D-Flipflops)

```
  0101
+ 0011
------
```

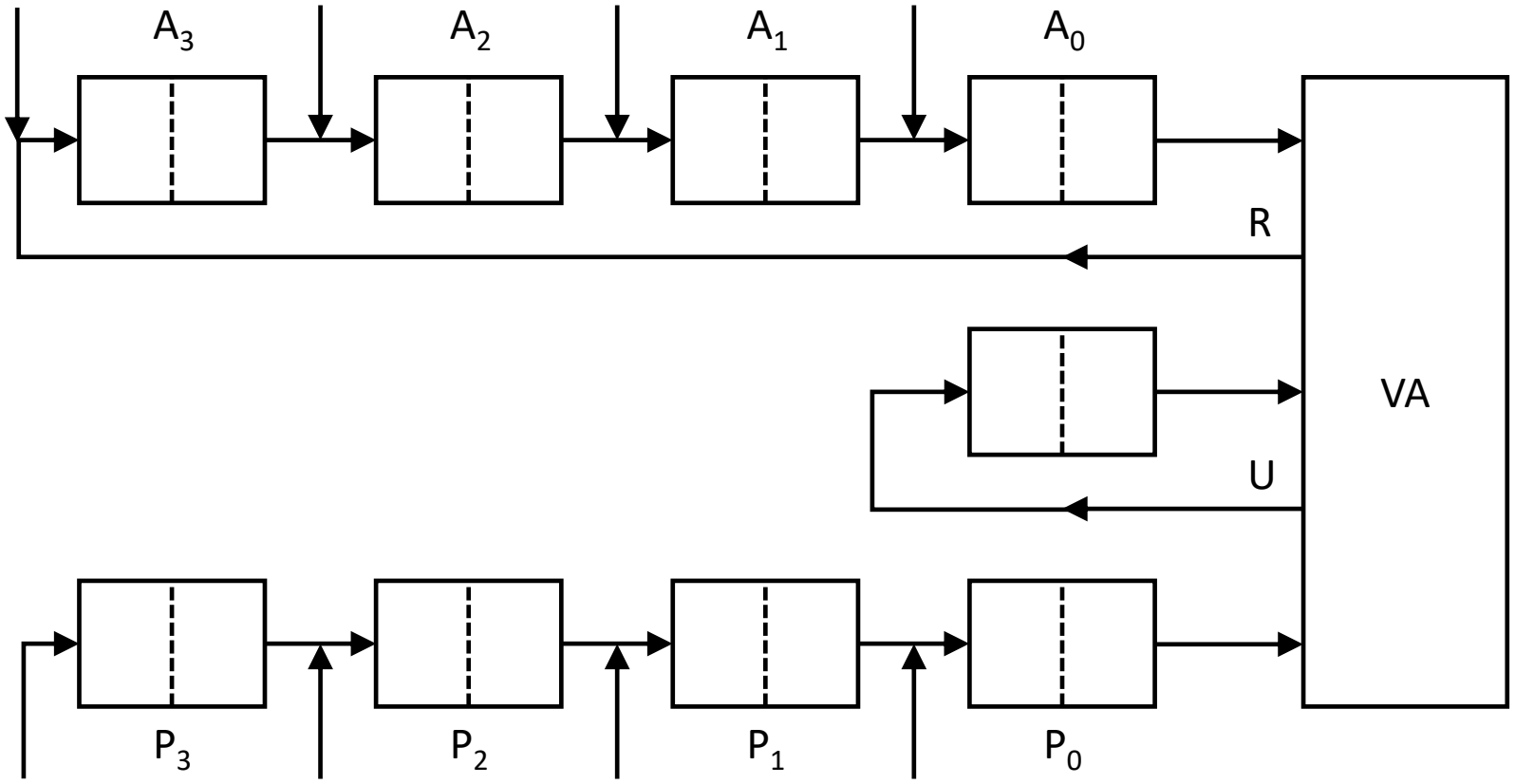


4-Bit-Parallel-Addierwerk (Ripple-Carry-Addierer mit D-Flipflops)

```
  0101
+ 0011
------
  1000
```

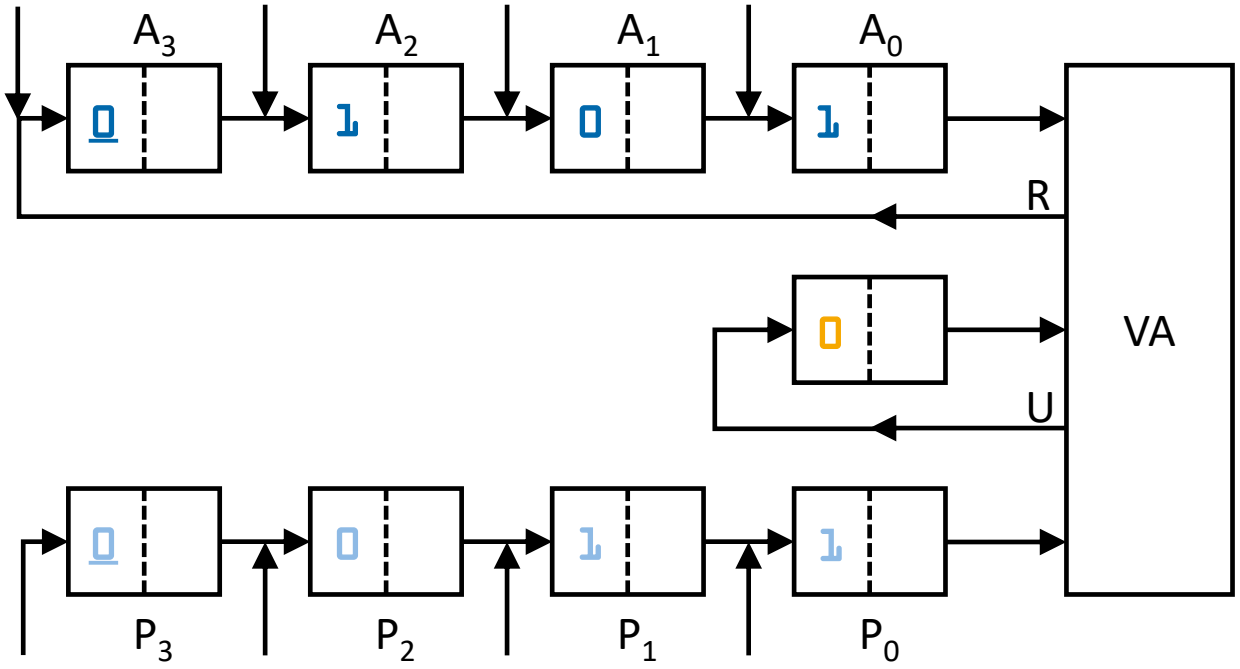


4-Bit-Serien-Addierwerk



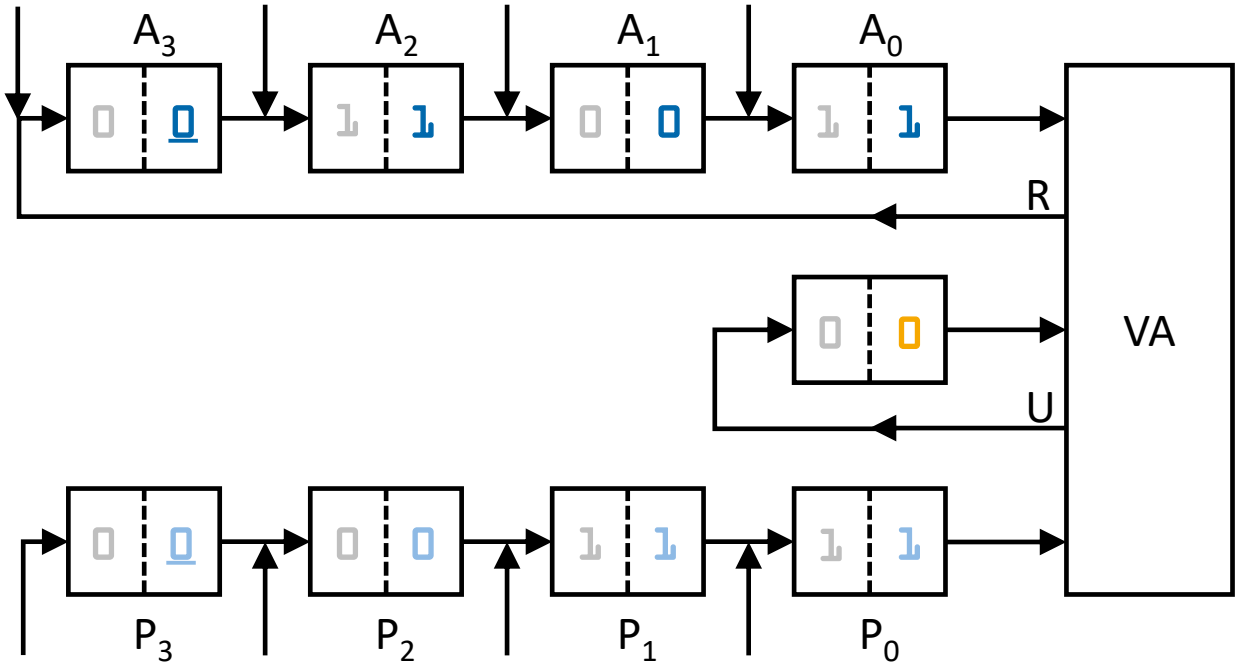
4-Bit-Serien-Addierwerk: Beispiel

```
  0101
+ 0011
------
```



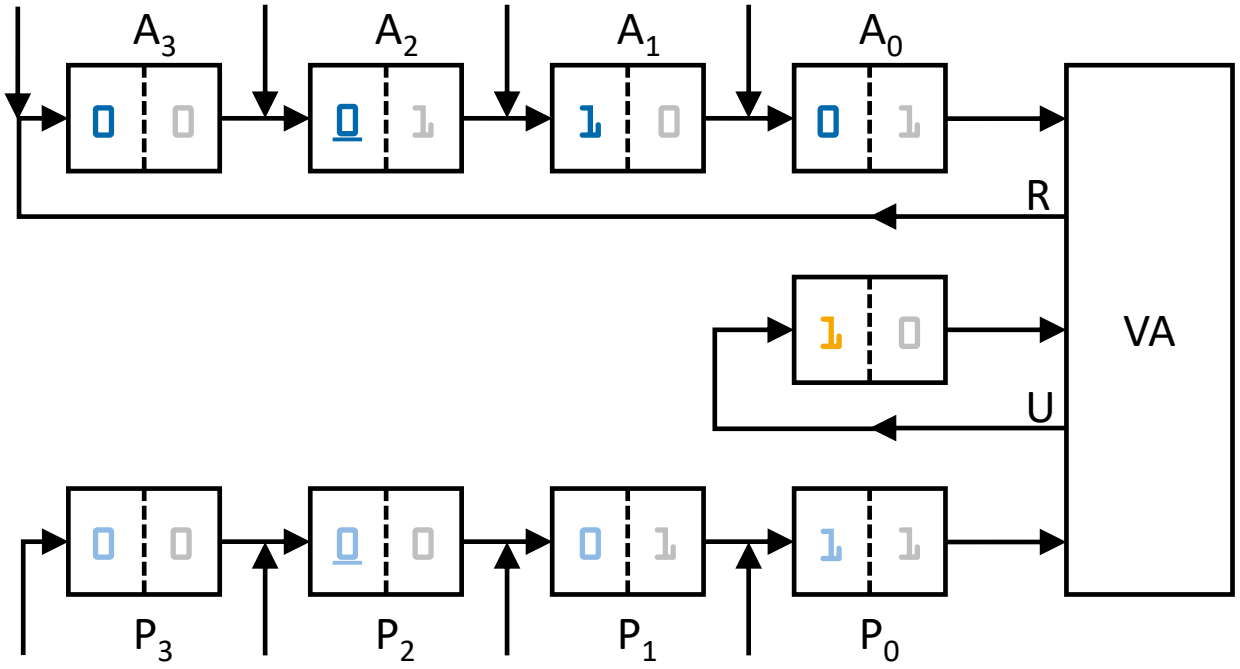
4-Bit-Serien-Addierwerk: Beispiel

```
  0101
+ 0011
------
```



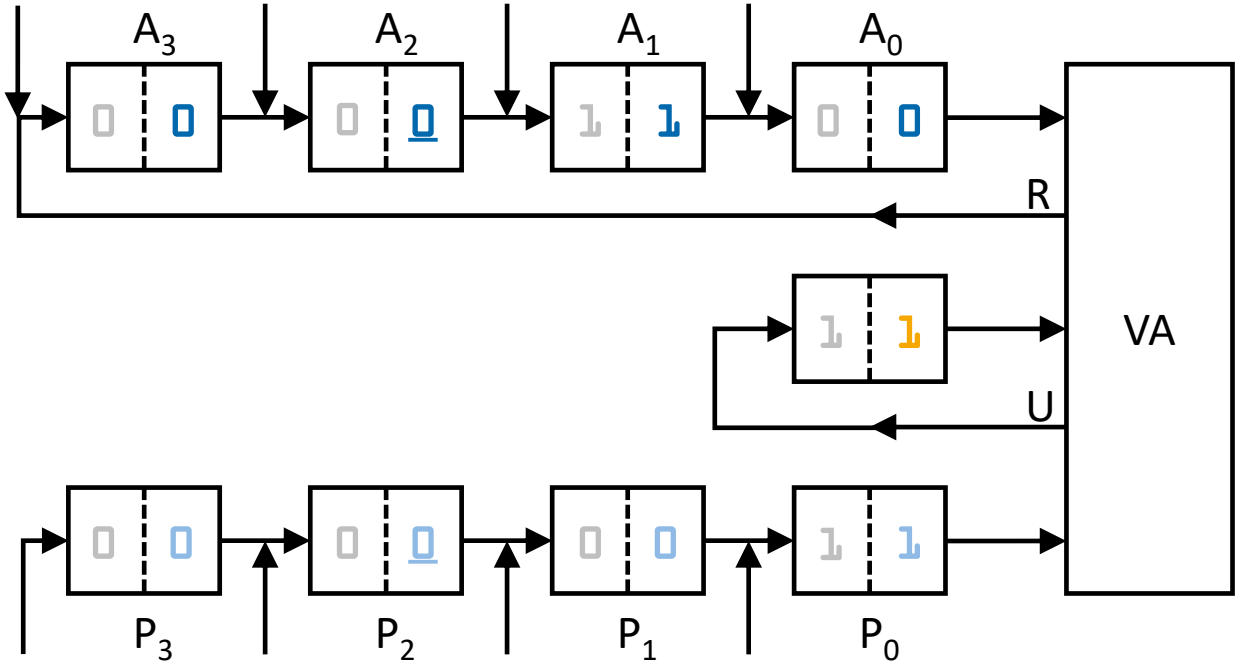
4-Bit-Serien-Addierwerk: Beispiel

```
   0101
 + 0011
 ------
   0010
 + 0001
 +    1
 ------
```



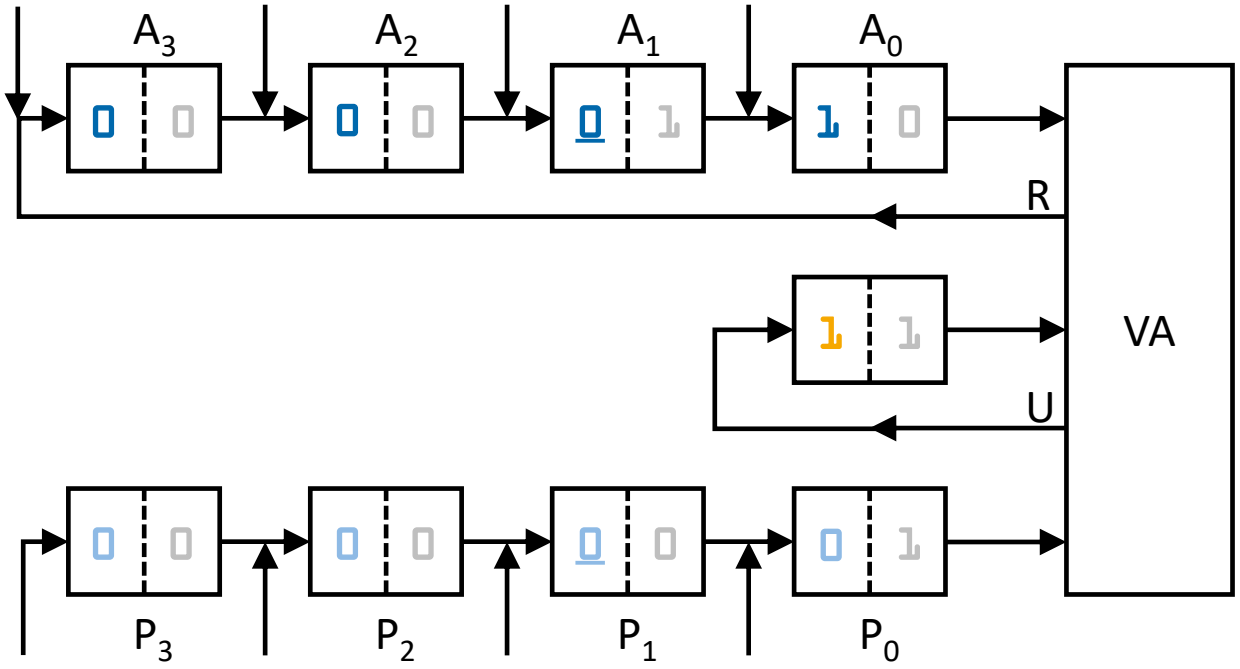
4-Bit-Serien-Addierwerk: Beispiel

```
   0101
 + 0011
 ------
   0010
 + 0001
 +    1
 ------
```



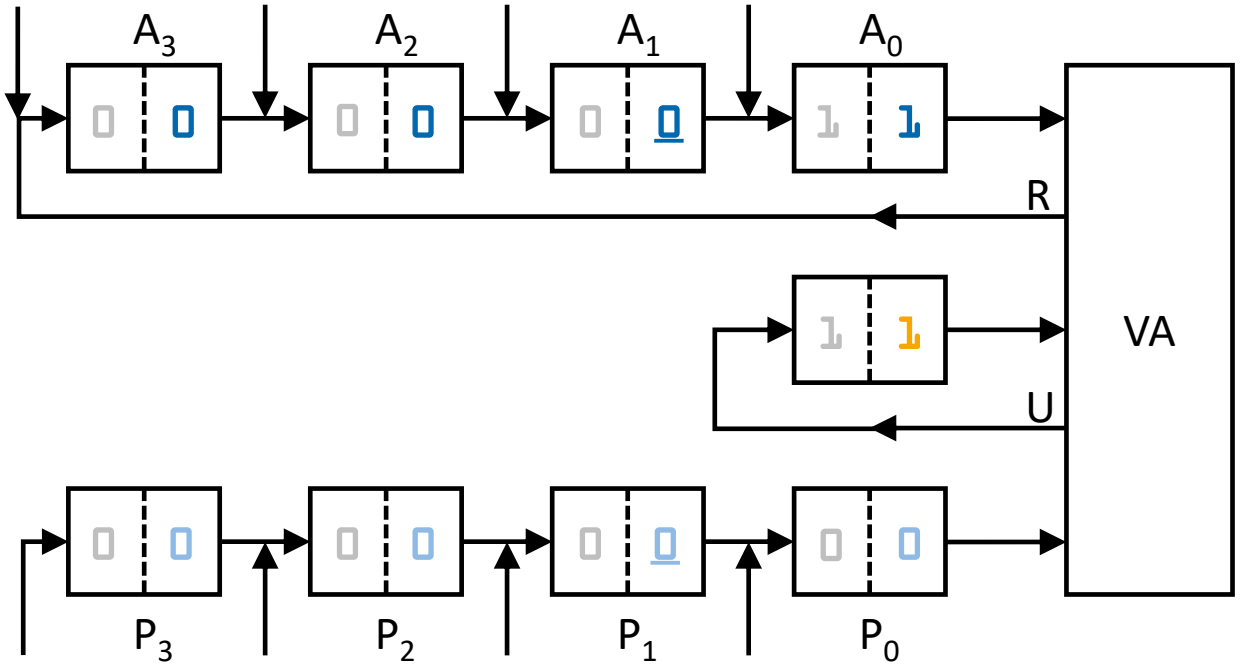
4-Bit-Serien-Addierwerk: Beispiel

```
   0101
 + 0011
------
   0010
 + 0001
 +    1
------
   0011
 + 0000
 +    1
------
```



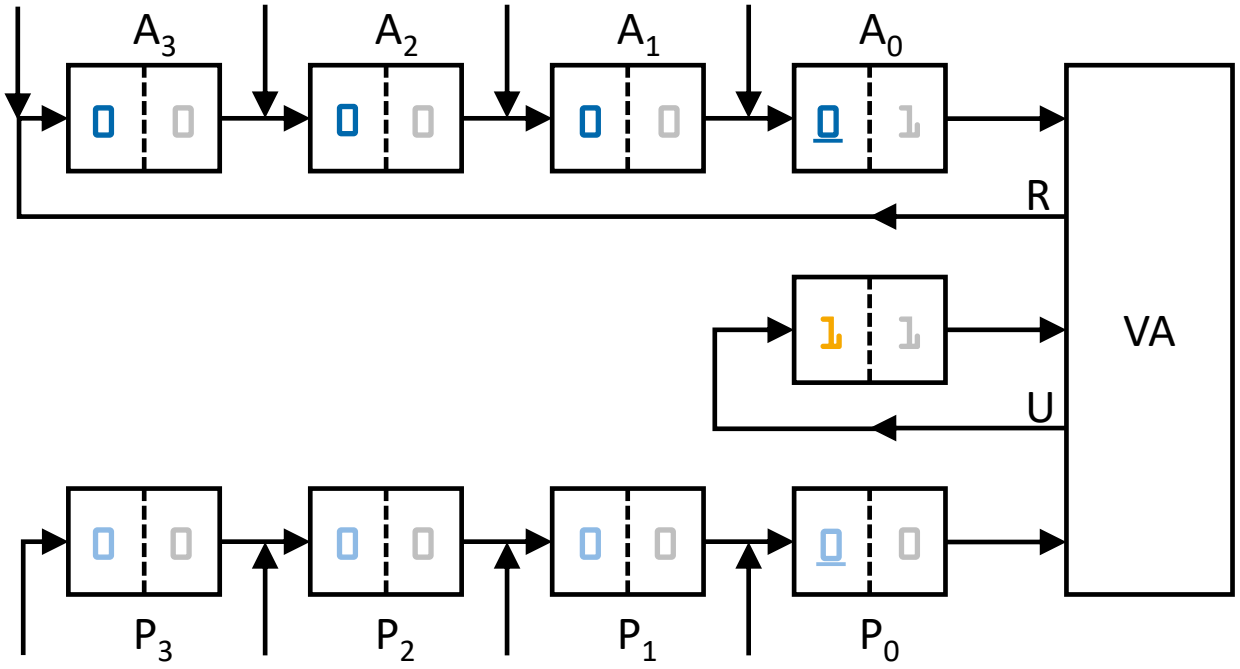
4-Bit-Serien-Addierwerk: Beispiel

```
   0101
 + 0011
------
   0010
 + 0001
 +    1
------
   0011
 + 0000
 +    1
------
```



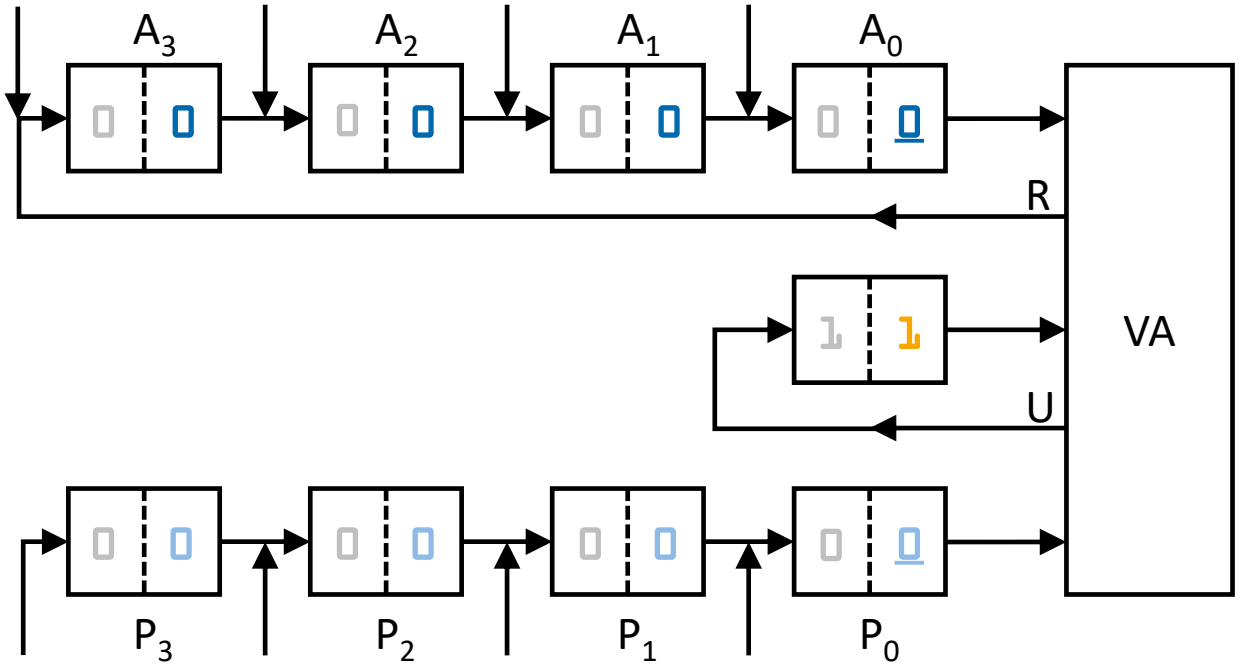
4-Bit-Serien-Addierwerk: Beispiel

```
   0101
 + 0011
------
   0010
 + 0001
 +    1
------
   0001
 + 0000
 +    1
------
   0000
 + 0000
 +    1
------
```



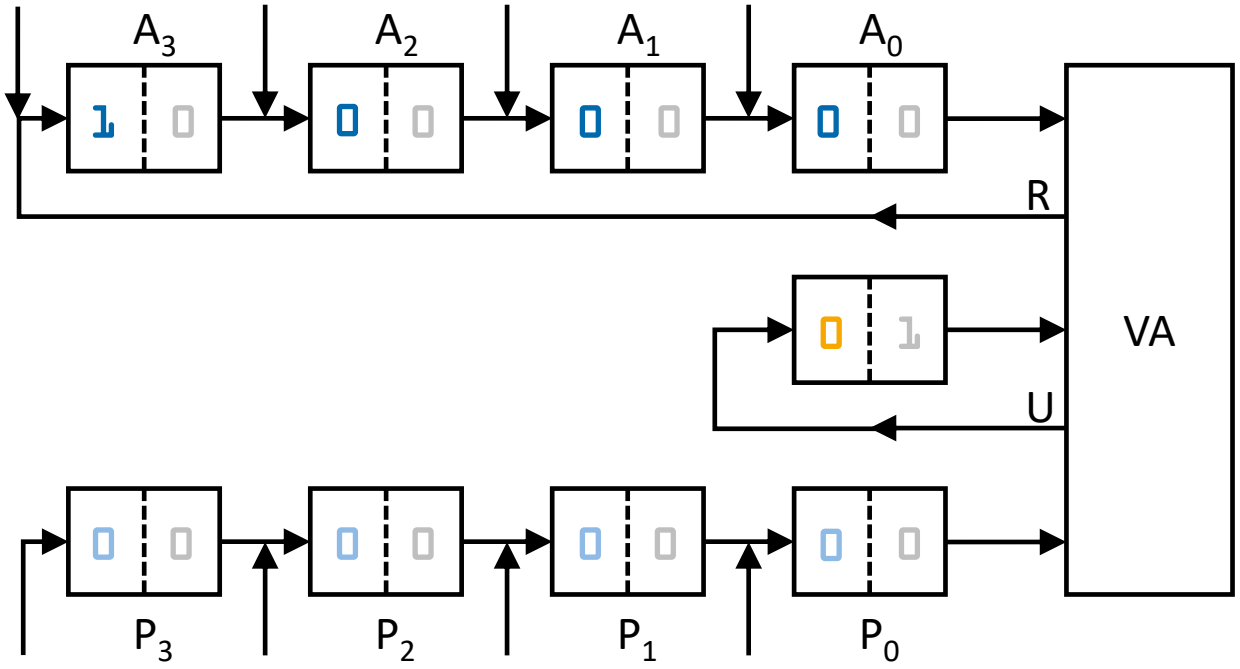
4-Bit-Serien-Addierwerk: Beispiel

```
   0101
 + 0011
------
   0010
 + 0001
 +    1
------
   0001
 + 0000
 +    1
------
   0000
 + 0000
 +    1
------
```



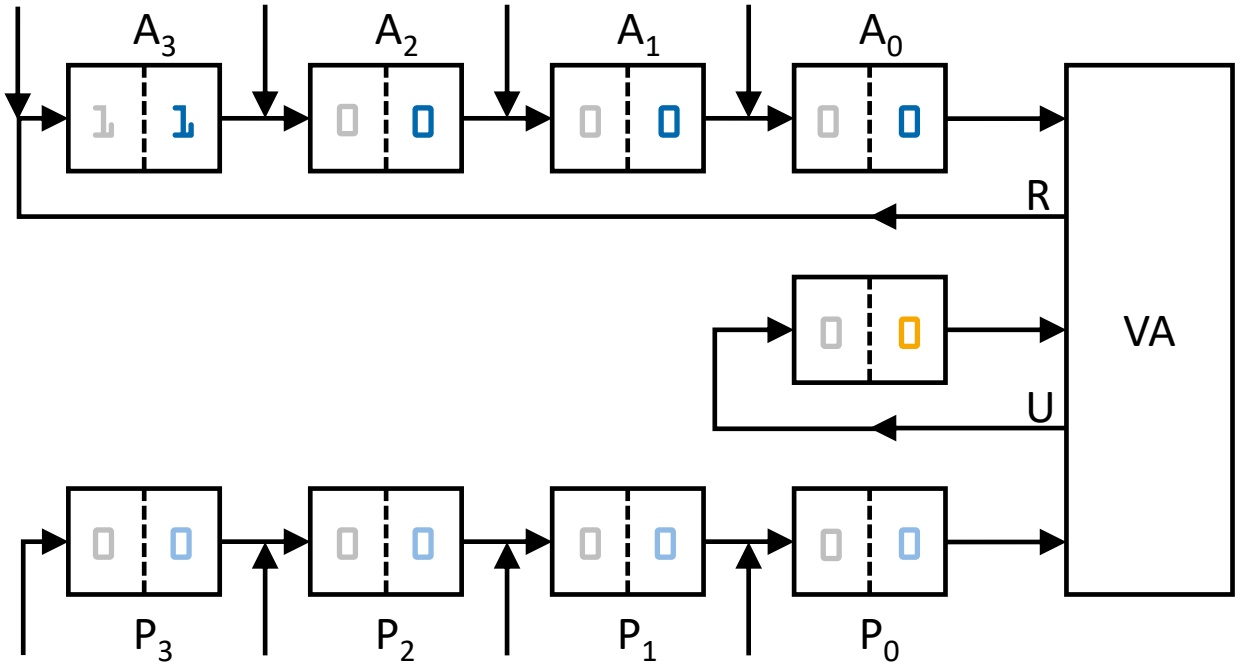
4-Bit-Serien-Addierwerk: Beispiel

```
   0101
 + 0011
------
   0010
 + 0001
 +    1
------
   0001
 + 0000
 +    1
------
   0000
 + 0000
 +    1
------
   1000
```

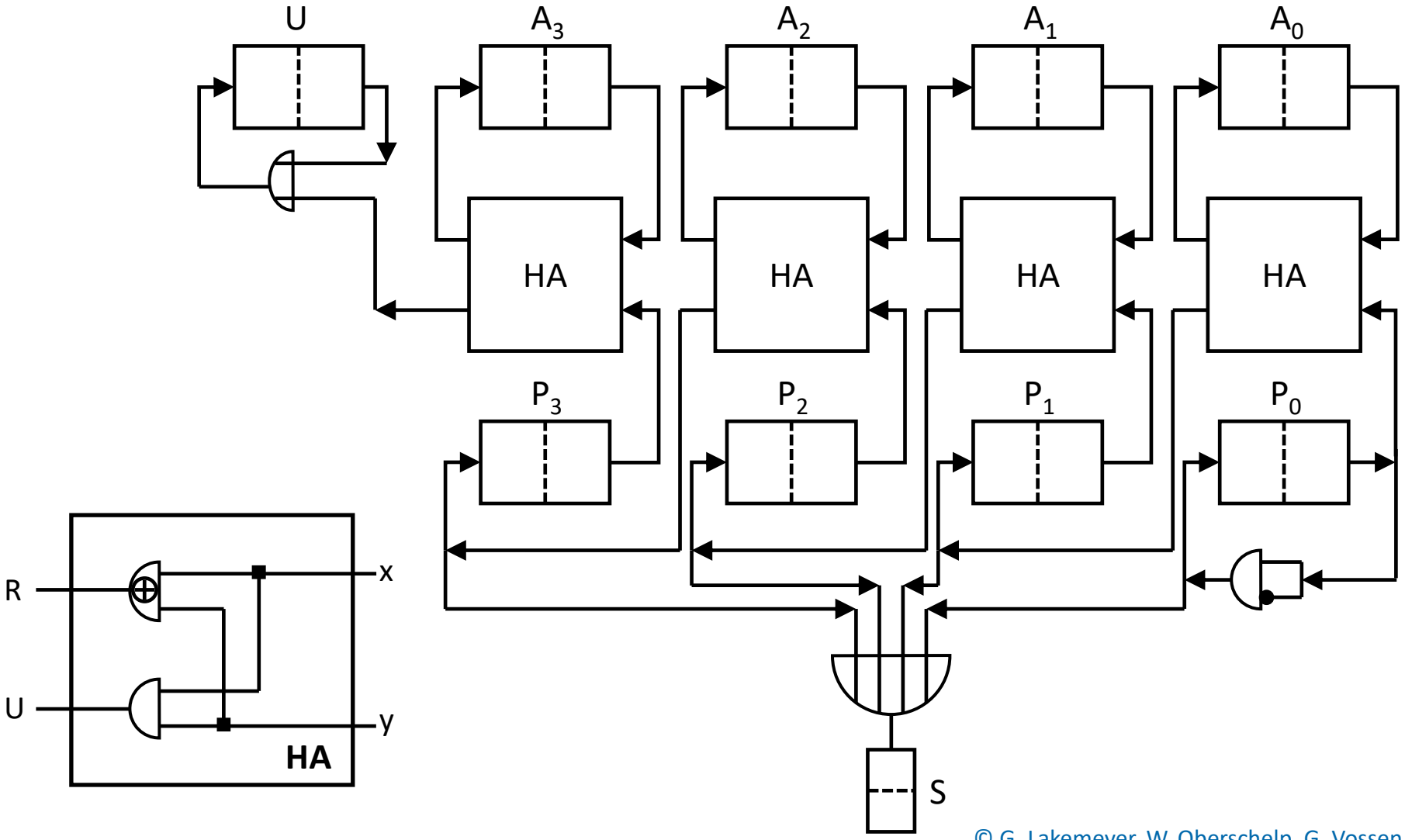


4-Bit-Serien-Addierwerk: Beispiel

```
   0101
 + 0011
------
   0010
 + 0001
 +    1
------
   0001
 + 0000
 +    1
------
   0000
 + 0000
 +    1
------
   1000
```

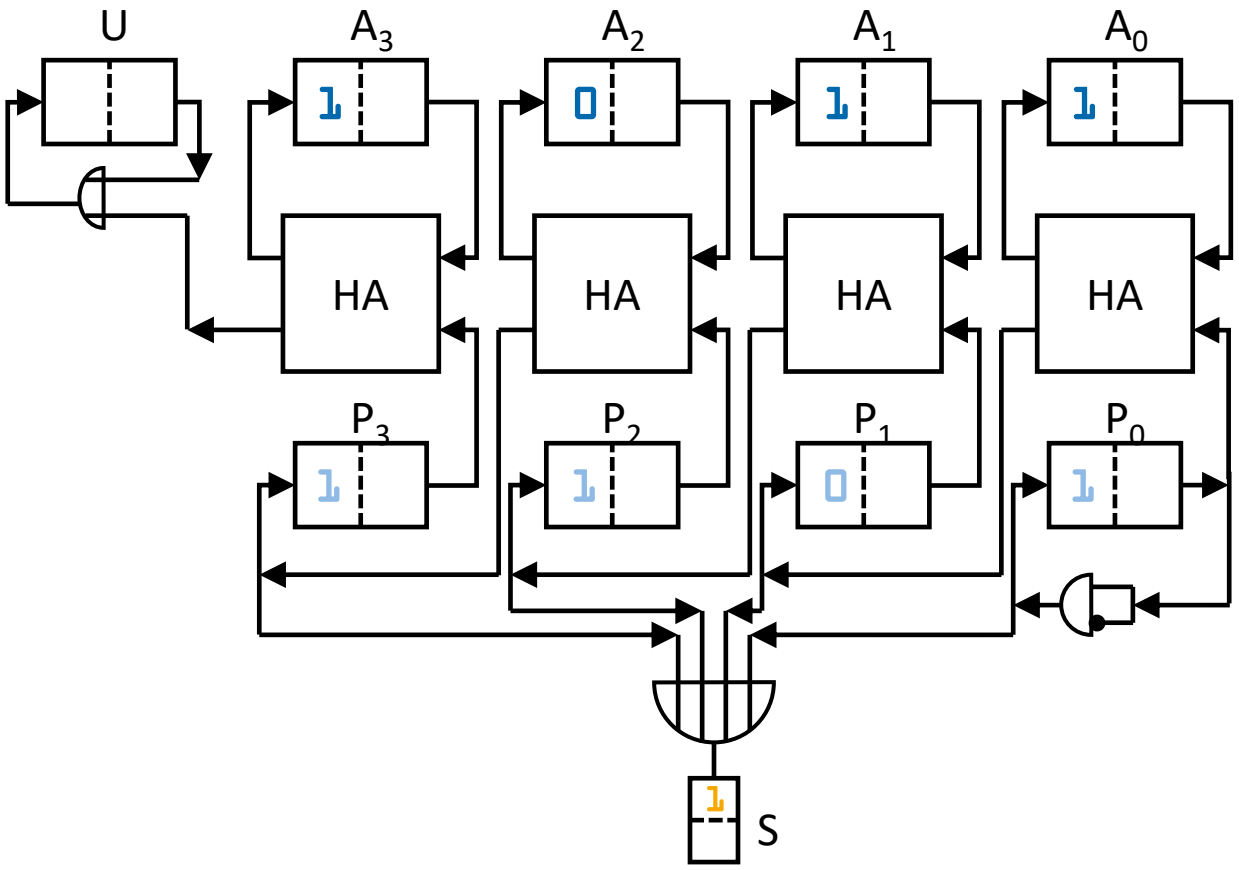


4-Bit-von Neumann-Addierwerk



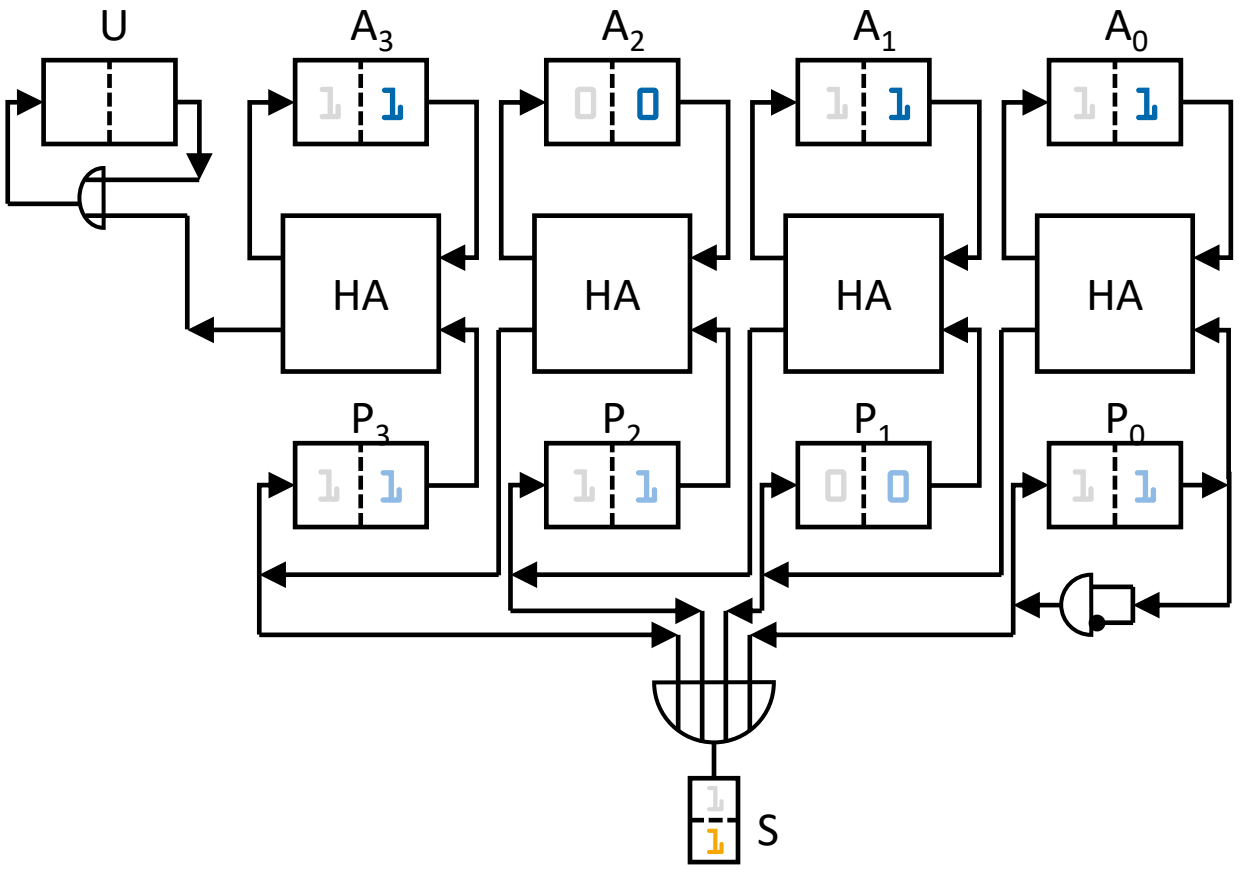
4-Bit-von Neumann-Addierwerk: Beispiel

```
  1011
+ 1101
------
```



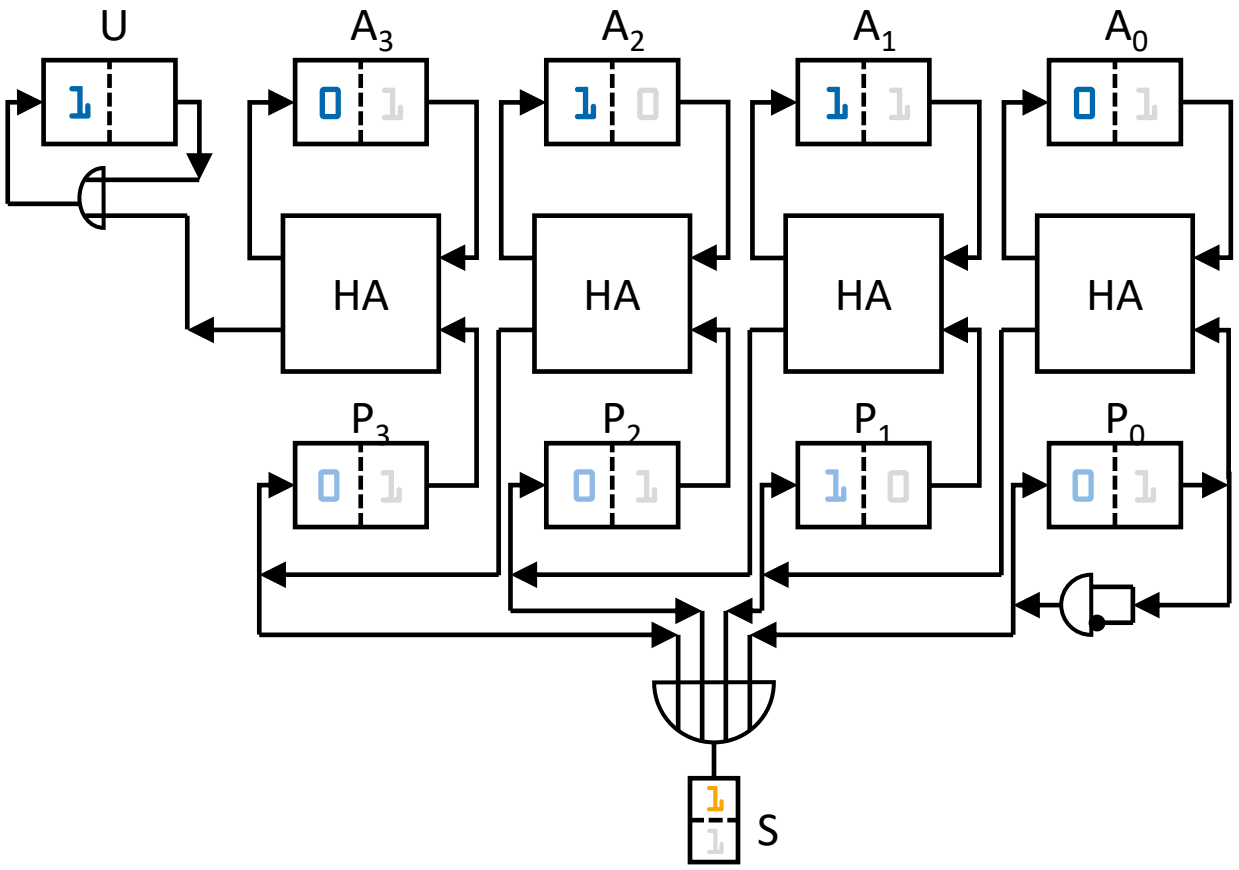
4-Bit-von Neumann-Addierwerk: Beispiel

```
  1011
+ 1101
------
```



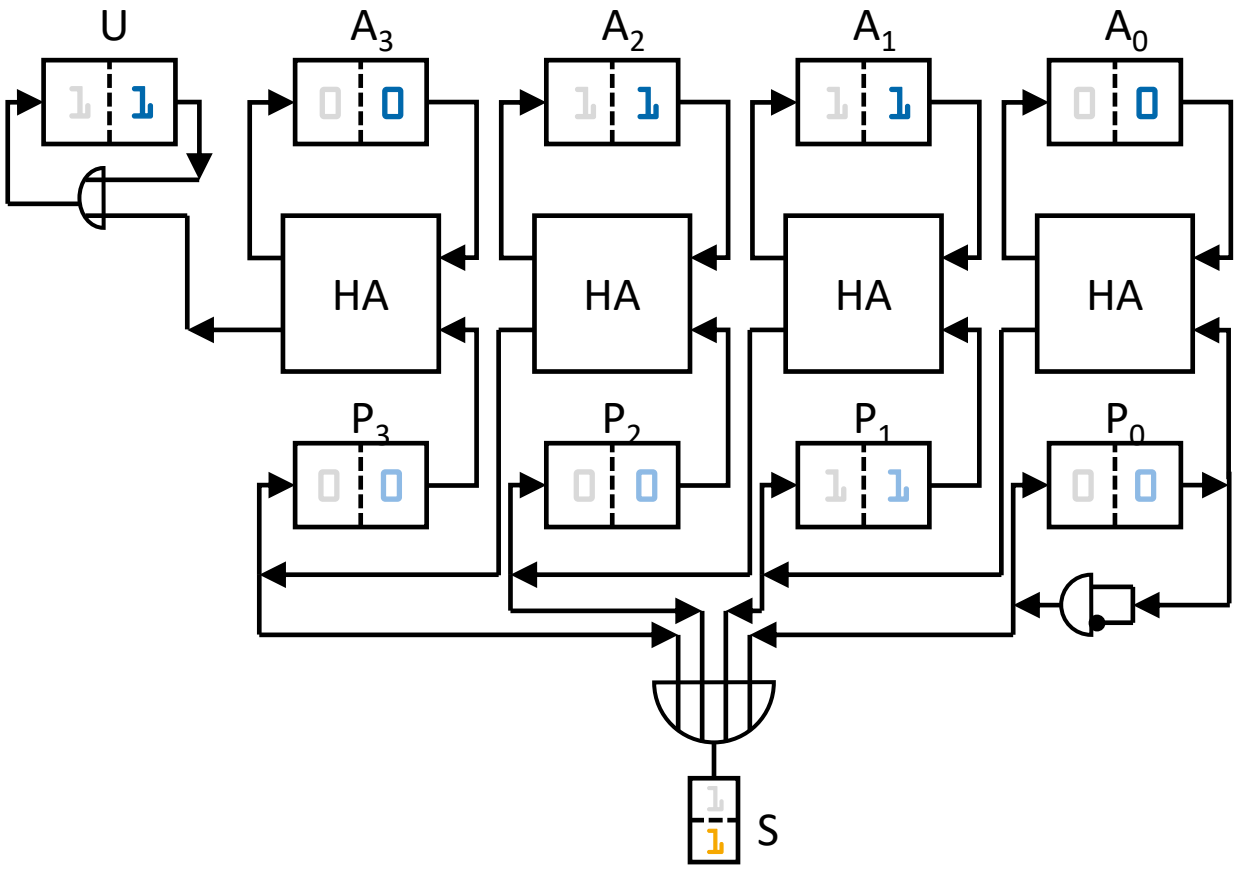
4-Bit-von Neumann-Addierwerk: Beispiel

```
   1011
 + 1101
-------
 1 0110
 + 0010
-------
```



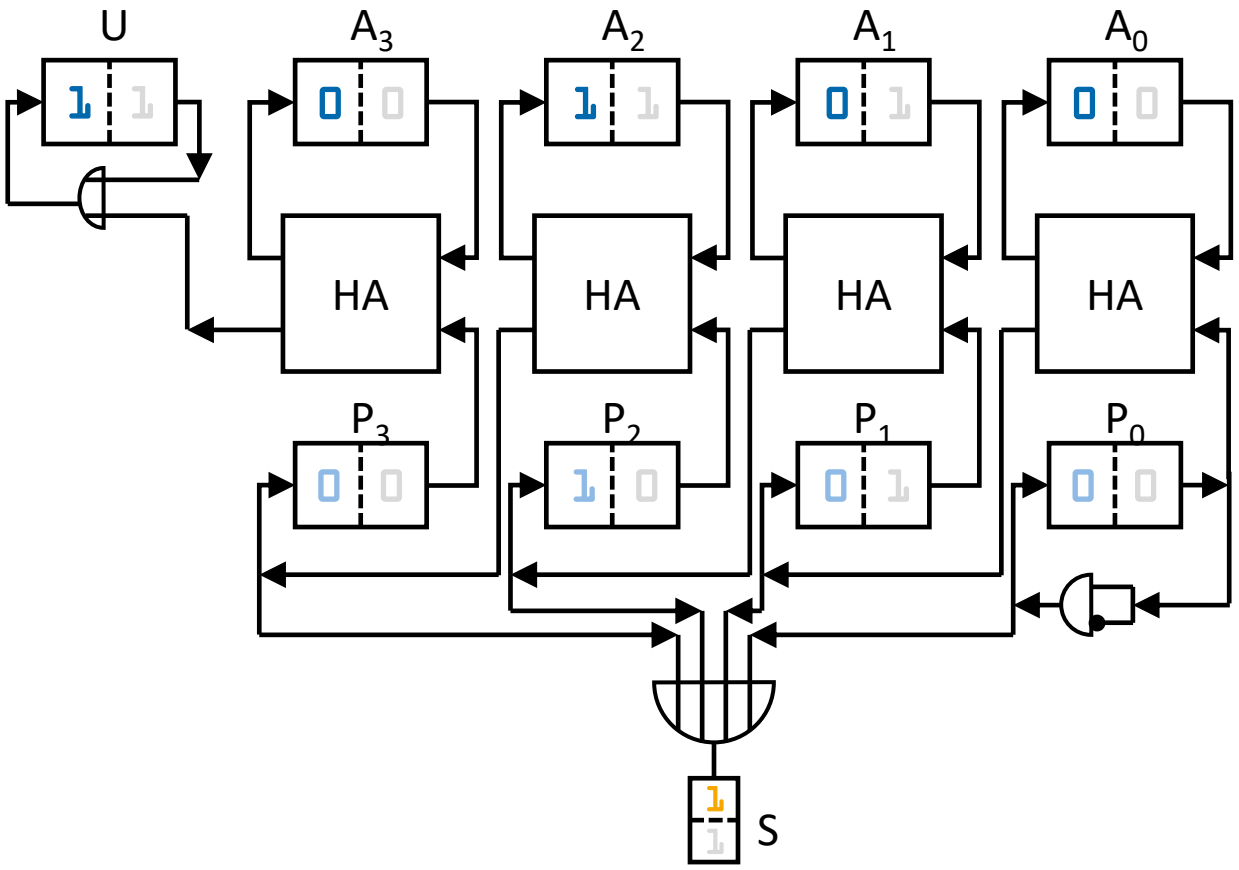
4-Bit-von Neumann-Addierwerk: Beispiel

```
   1011
 + 1101
-------
 1 0110
 + 0010
-------
```



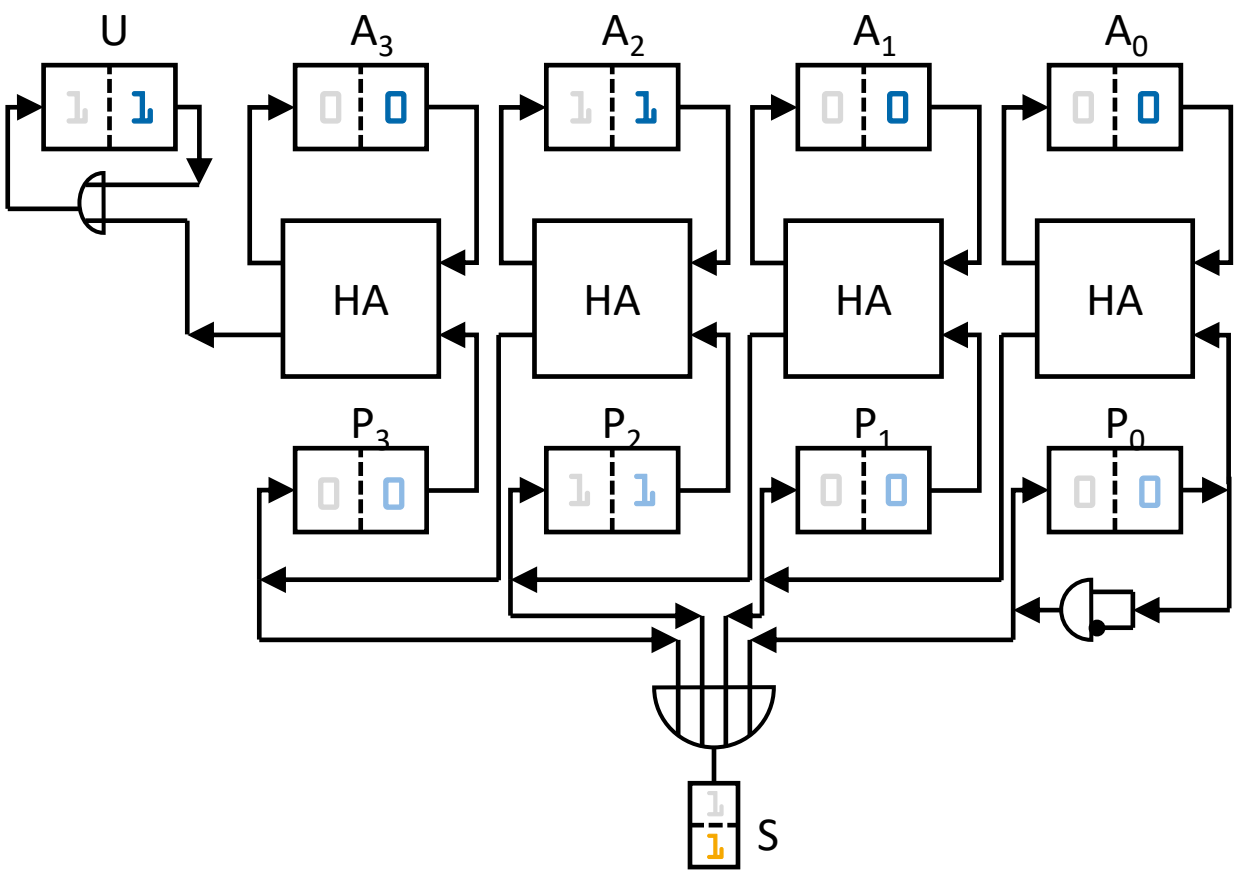
4-Bit-von Neumann-Addierwerk: Beispiel

```
   1011
 + 1101
-------
 1 0110
 + 0010
-------
 1 0100
 + 0100
-------
```



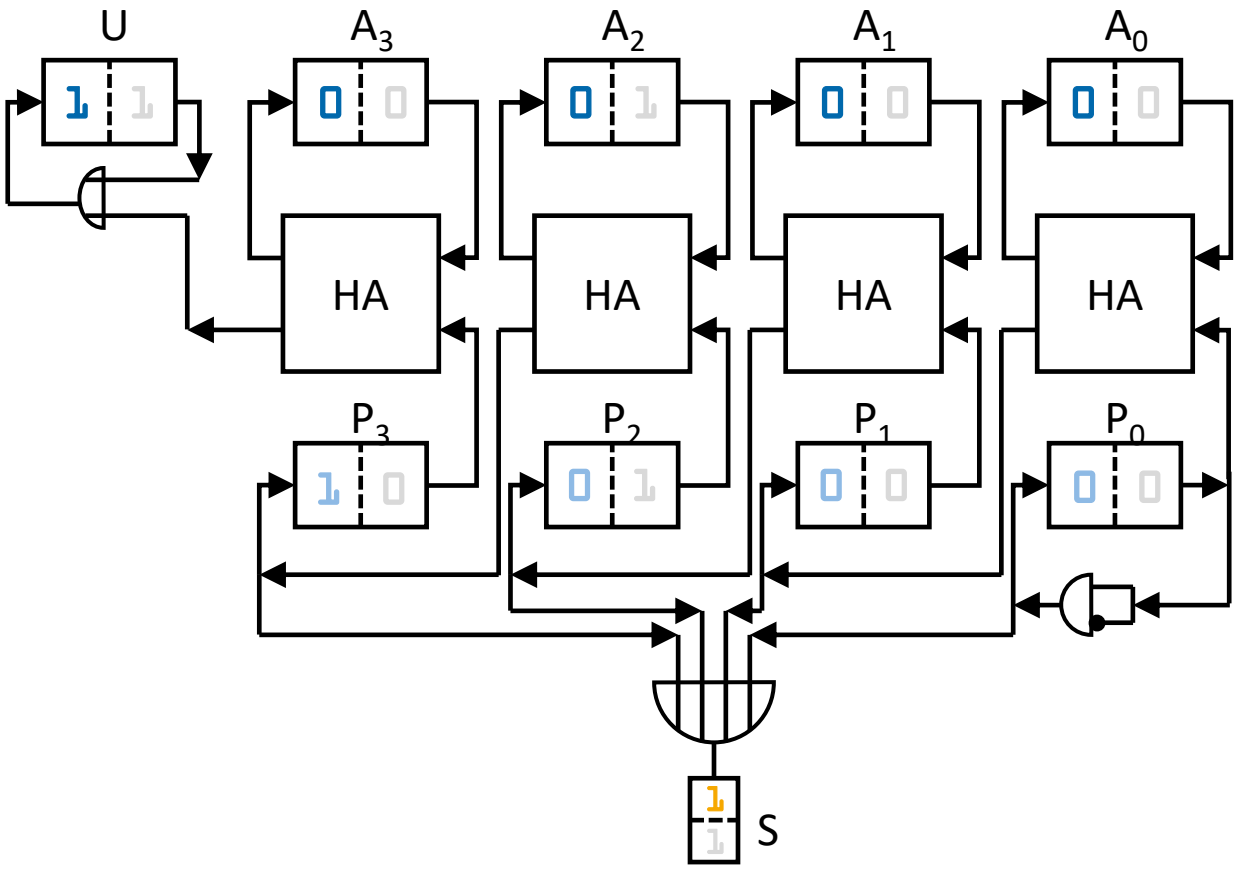
4-Bit-von Neumann-Addierwerk: Beispiel

```
   1011
 + 1101
-------
 1 0110
 + 0010
-------
 1 0100
 + 0100
-------
```



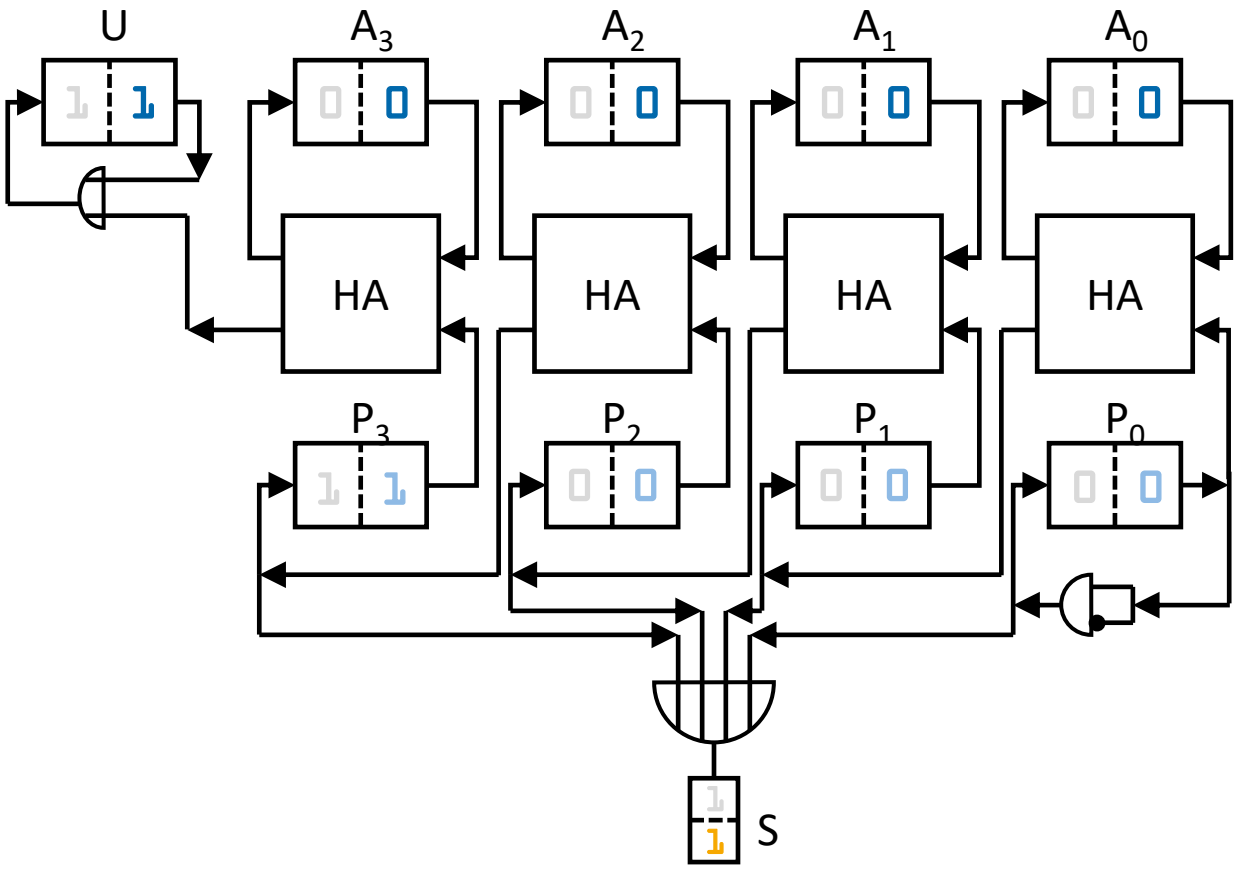
4-Bit-von Neumann-Addierwerk: Beispiel

```
   1011
 + 1101
-------
 1 0110
 + 0010
-------
 1 0100
 + 0100
-------
 1 0000
 + 1000
-------
```



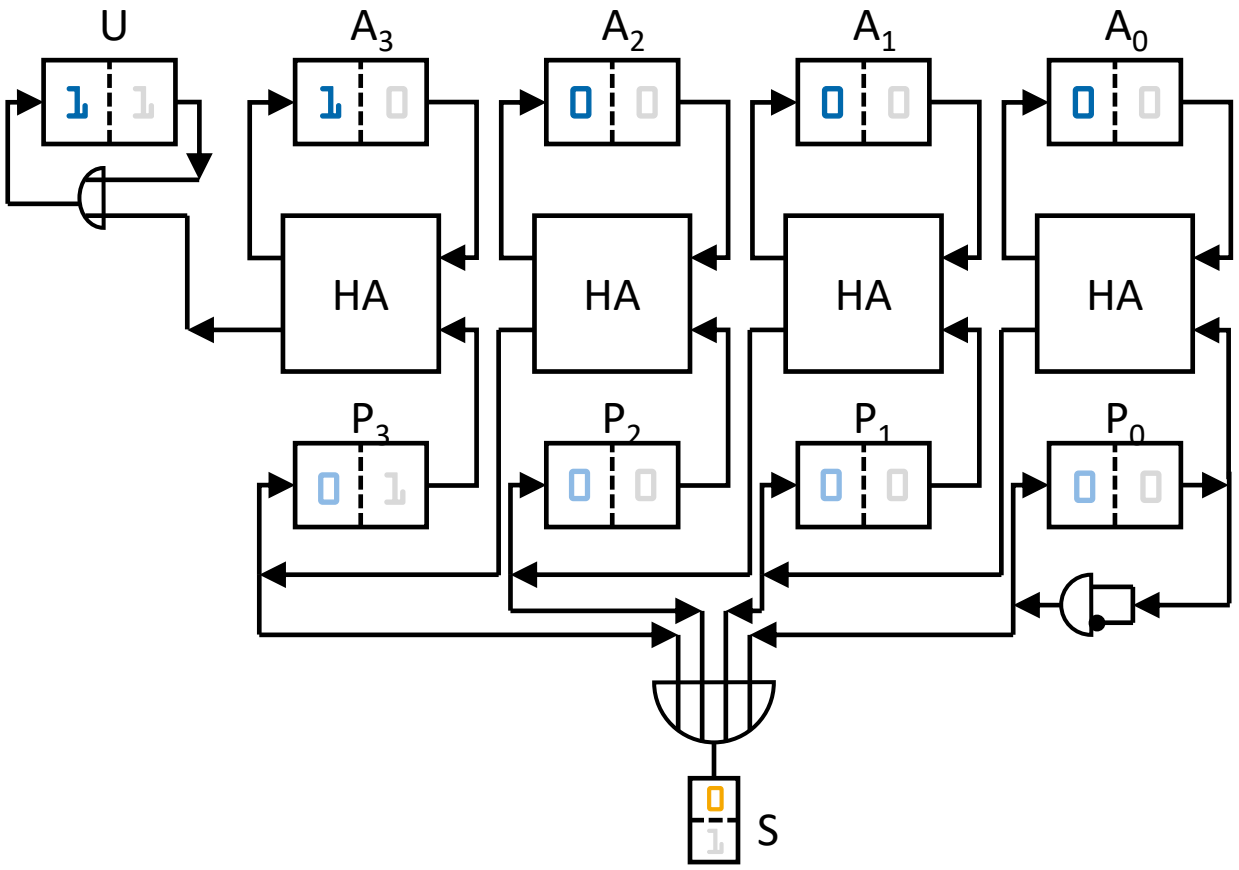
4-Bit-von Neumann-Addierwerk: Beispiel

```
   1011
 + 1101
-------
 1 0110
 + 0010
-------
 1 0100
 + 0100
-------
 1 0000
 + 1000
-------
```



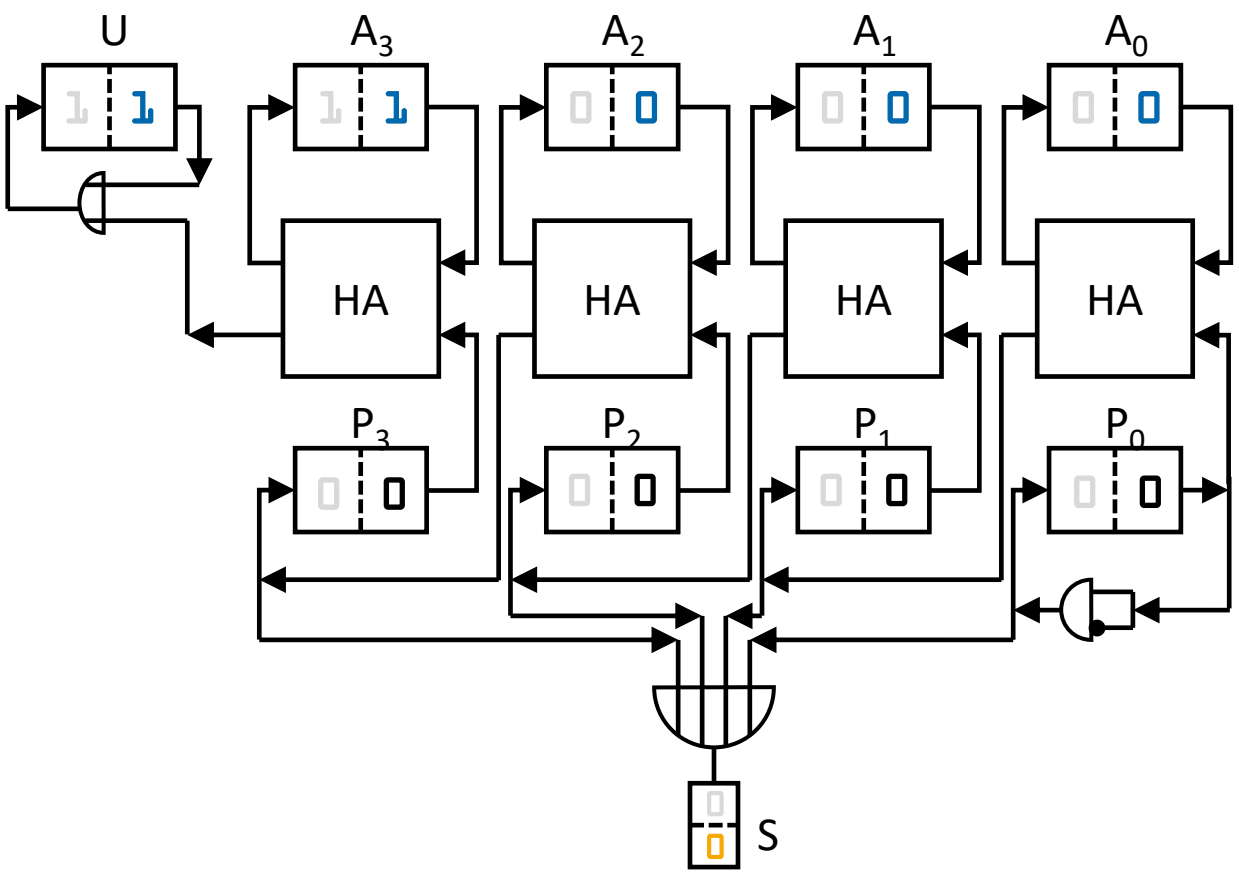
4-Bit-von Neumann-Addierwerk: Beispiel

```
   1011
 + 1101
-------
 1 0110
 + 0010
-------
 1 0100
 + 0100
-------
 1 0000
 + 1000
-------
 1 1000
 + 0000
-------
```



4-Bit-von Neumann-Addierwerk: Beispiel

```
   1011
 + 1101
-------
 1 0110
 + 0010
-------
 1 0100
 + 0100
-------
 1 0000
 + 1000
-------
 1 1000
 + 0000
-------
 1 1000
```



Arbeitsweise (Beispiel)

Aufgaben: 13+11, 10+12, 15+15, 9+10, 0+0

Zeile	U	Akku-Inhalt dual $(U)A_3A_2A_1A_0$	Akku-Inhalt dezimal	Puffer-Inhalt dual $P_3P_2P_1P_0$	Puffer-Inhalt dezimal	S
1	0	(0)0000	0	0000	0	0
2	0	(0)1011	11	1101	13	1
3	1	(1)0110	22	0010	2	1
4	1	(1)0100	20	0100	4	1
5	1	(1)0000	16	1000	8	1
6	1	(1)1000	24	0000	0	0
7	0	(0)1100	12	1010	10	1
8	1	(1)0110	22	0000	0	0
9	0	(0)1111	15	1111	15	1
10	1	(1)0000	16	1110	14	1
11	1	(1)1110	30	0000	0	0
12	0	(0)1010	10	1001	9	1
13	1	(1)0011	19	0000	0	0