MI-ARI

(Computer arithmetics)

winter semester 2017/18

N2. **Multiplication II.**

Department of digital design

Faculty of Information technology

Czech Technical University in Prague

https://edux.fit.cvut.cz/courses/MIE-ARI/lectures

N2. **Multiplication II.**

- **Principle of skip over 0's and 1's**
- **Parallel multipliers**
  - **Principle of parallel multiplier**
  - **Carry-save parallel multiplier**
  - **Wallaces multiplier**
  - **Daddas multiplier**
  - **Luks and Vuillemins multiplier**
  - **Parallel counter**
- **Multiplication in decimal system**

Principle of skip over 0's

ex.: $10111101 \cdot 10000101$ ☞

$$
\begin{array}{r}
1\,0111\,101 \\
101\,1110\,1 \\
1011\,1101 \\
\hline
1100\,0100\,0110\,001
\end{array}
$$

only **3 clock cycles** (instead of 8 „standard“ ones)

possibility to skip over l zeros:

- **adder width and data paths must be greater of l**
- **circuit capable to shift from 0 to $l-1$ positions is needed**

Principle of skip over 1's

Principle: **The sequence of j 1's is converted to signed-digit number systems, e.g. $1111 = 1\ 000\widehat{1}$**
Than $j-1$ 0's can be skipped
(this is suitable for $\boldsymbol{j > 2}$) only, of course.

ex.: $\mathbf{1011\ 1101 \cdot 0111\ 0111}$

$$\mathbf{0111\ 0111 = 100\widehat{1}\ 100\widehat{1} = 1000\ \widehat{1}00\widehat{1}}$$

☞

$$\begin{array}{lr}
- & \mathbf{1011\ 1101} \\
- & \mathbf{101\ 1110\ 1000} \\
+ & \mathbf{101\ 1110\ 1000\ 0000} \\
\hline
 & \mathbf{101\ 0111\ 1101\ 1011}
\end{array}$$

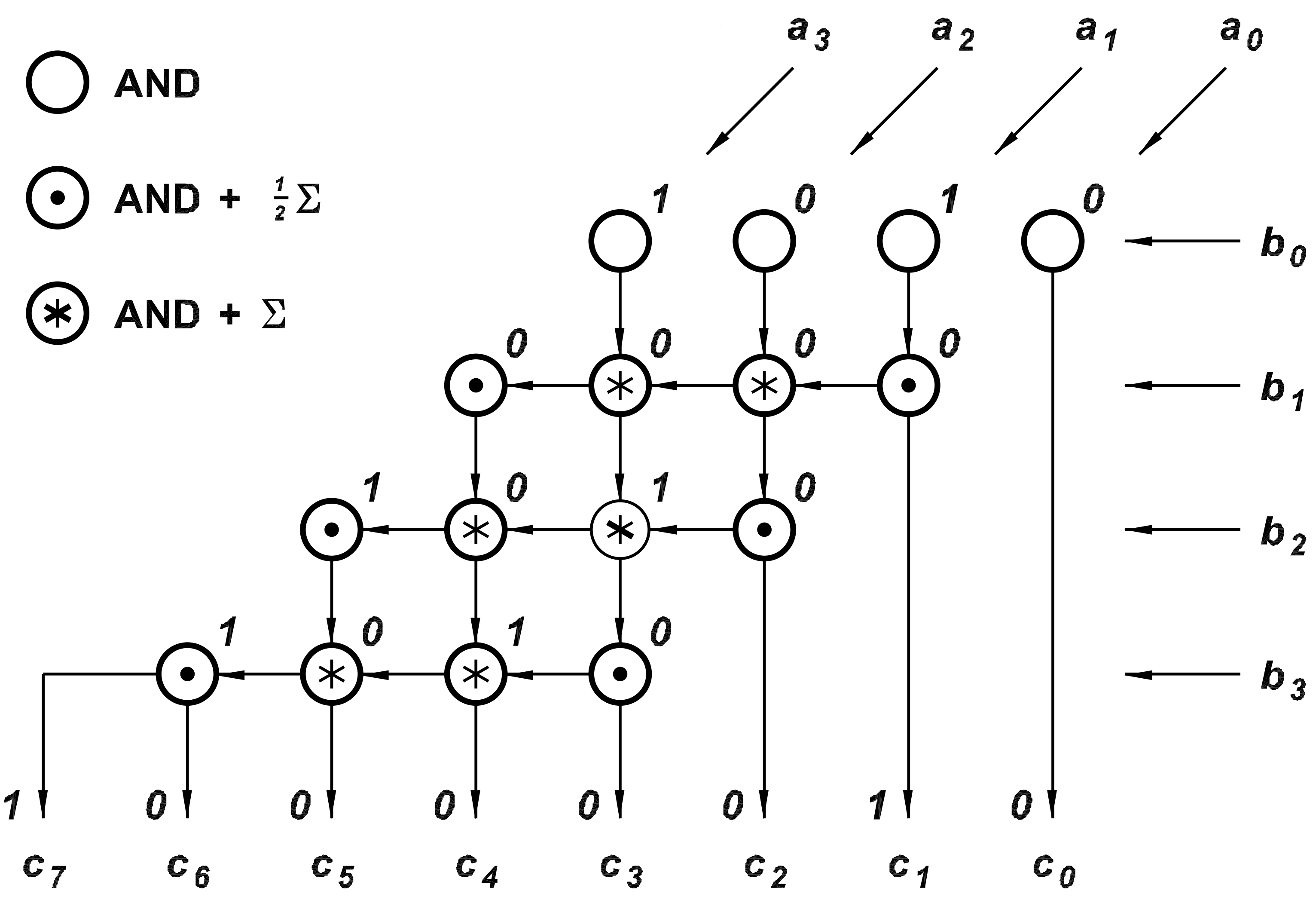
Principle of parallel multiplier

Example ($10 \times 13 = 130$):

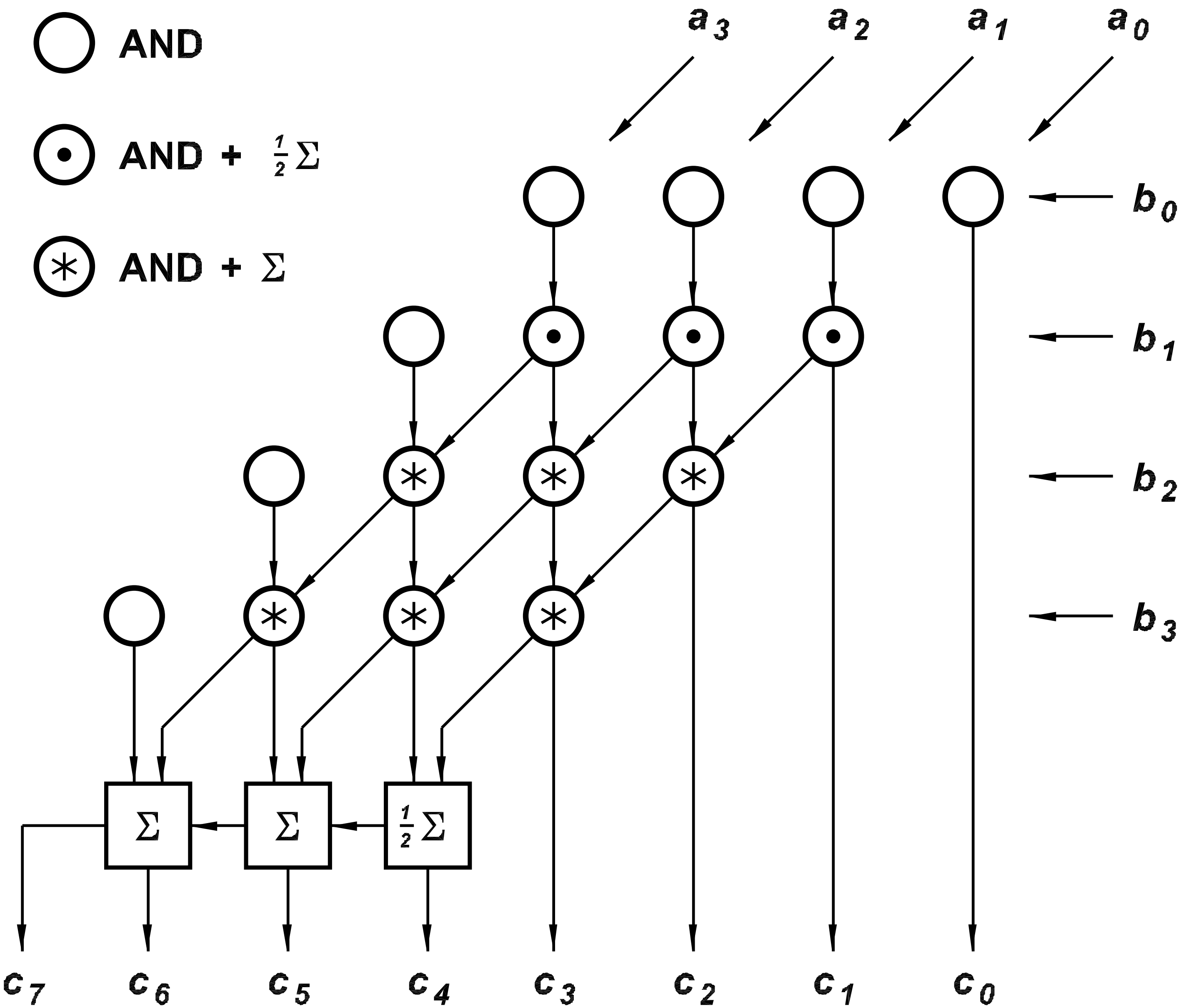
$$\mathbf{1\ 0\ 1\ 0} \bullet \mathbf{1\ 1\ 0\ 1} = \mathbf{1\ 0\ 0\ 0\ \ 0\ 0\ 1\ 0}$$

$$\begin{array}{cccccccccccccc}
 & & & & & & & & & & \downarrow & \downarrow & \downarrow & \downarrow \\
 & & & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & & & & & & \text{☜} & \mathbf{1} \\
 & & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & & & & & & \text{☜} & \mathbf{0} & \\
 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & & & & & & \text{☜} & \mathbf{1} & & \\
\mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & & & & & & \text{☜} & \mathbf{1} & & & \\
\hline
\mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & & & & & &
\end{array}$$

Principle of parallel multiplier



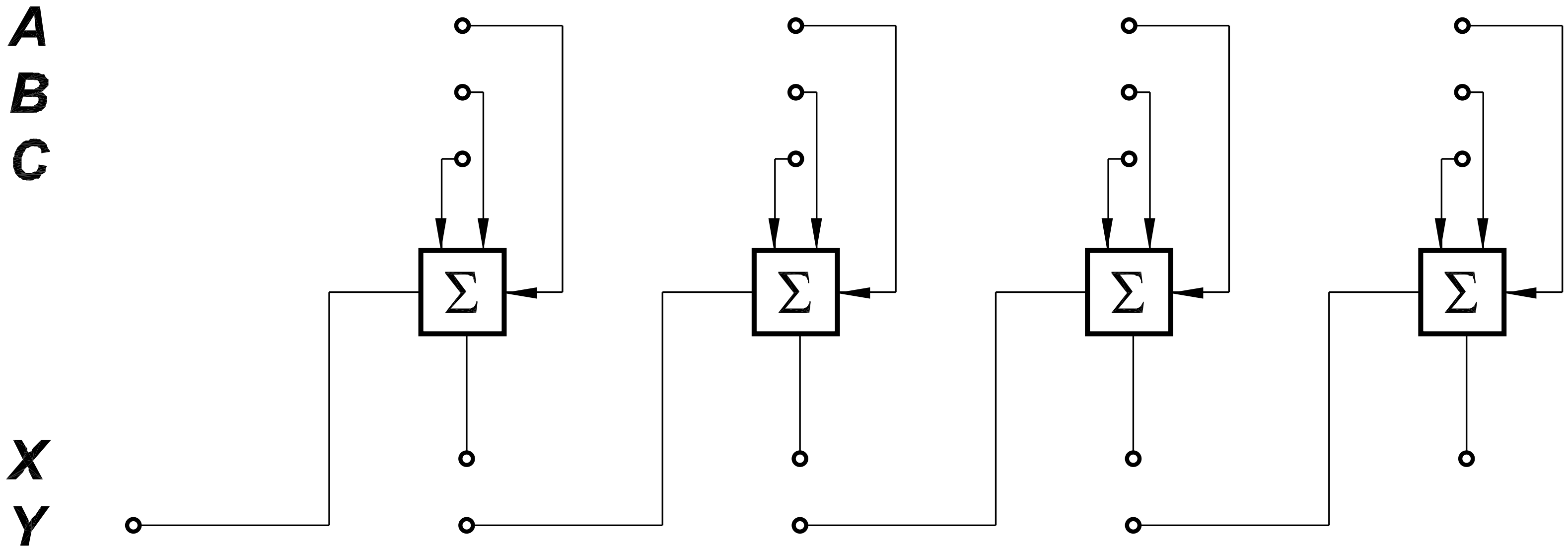
Carry-save parallel multiplier



Wallaces multiplier

pseudo-adder: **3 inputs** A**,** B **a** C
2 outputs X **and** Y

$$\boxed{X + Y = A + B + C}$$



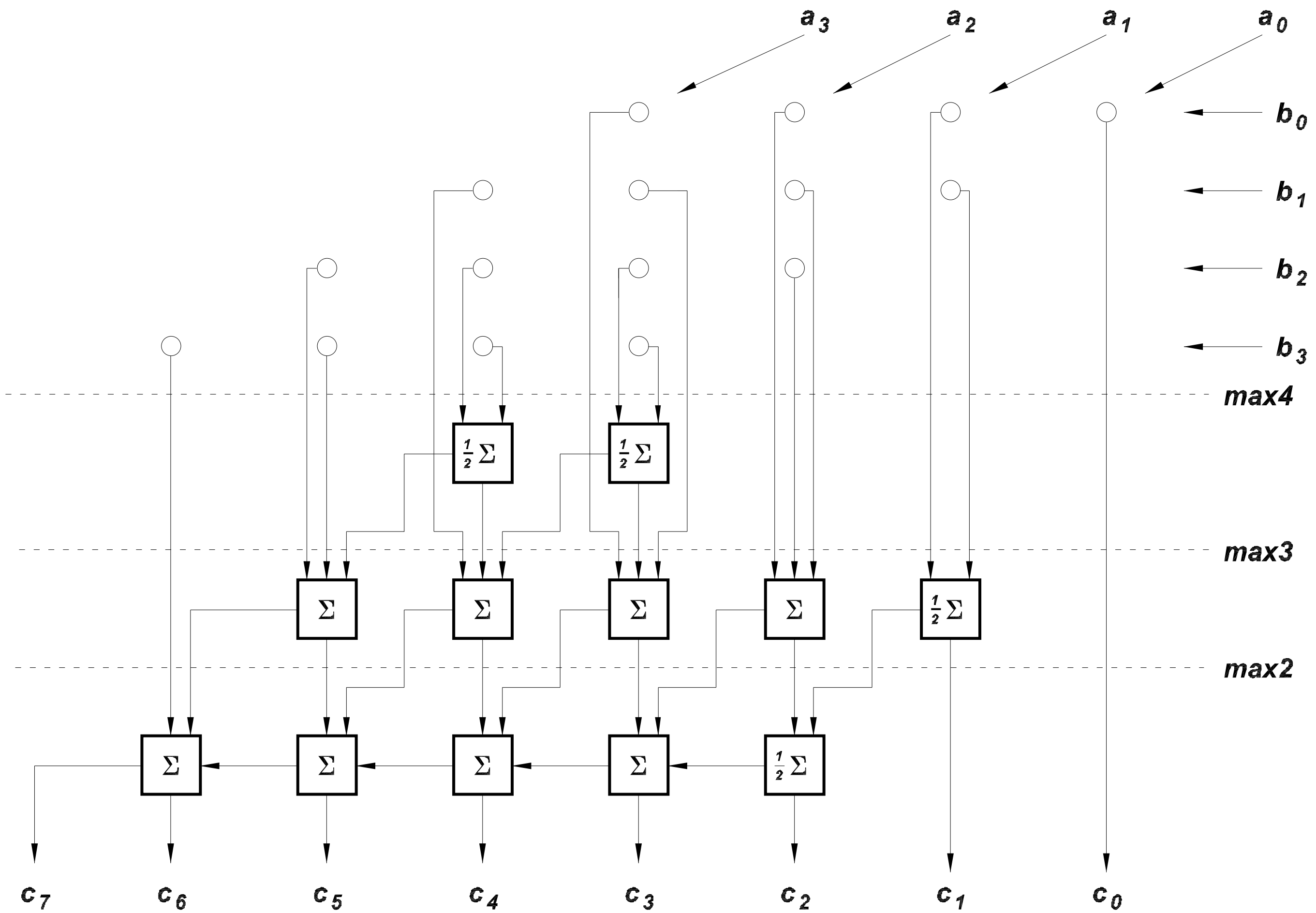
Latency is equal to latency of one full adder.

Wallaces multiplier

repeated reductions: triplets of addends → pair of addends

ex.: $\mathbf{32 \rightarrow 22 \rightarrow 15 \rightarrow 10 \rightarrow 7 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2}$

ex.:



Dadds multiplier is modification of Wallaces multiplier. Algorithm differs in first step, where the number of addend are reduced on the number which is element of given set

$$\mathbf{X} = \{\, 2, 3, 4, 6, 9, 13, 19, 28, 42, 63, 94, 141, \ldots \},$$

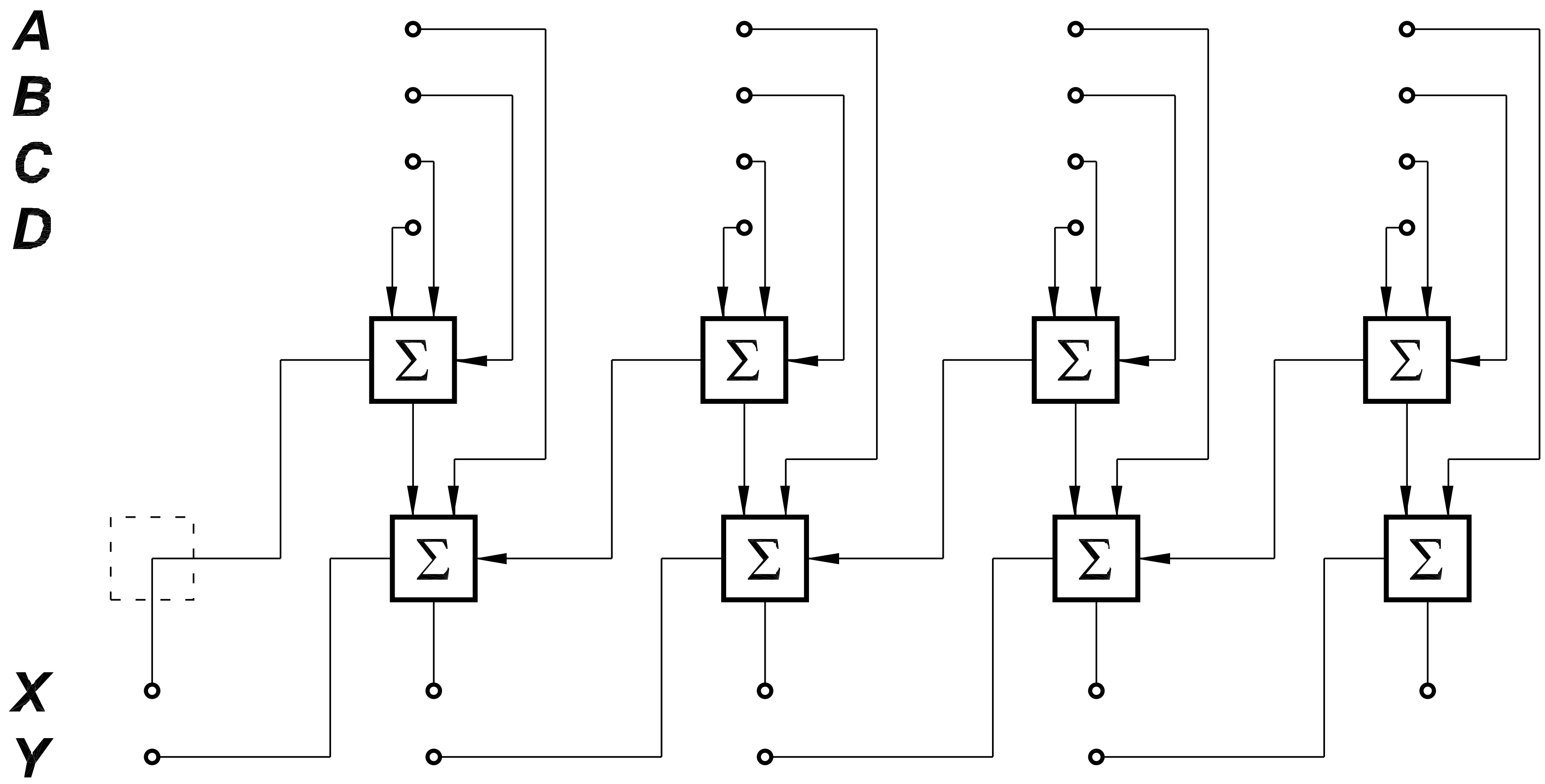
whose elements are numbers: $x_1 = 2$ **and**
$x_{i+1} = \lfloor x_i / 2 \cdot 3 \rfloor$ **for** $i = 1, 2, \ldots$

Dadds multiplier is simpler or same complexity as Wallaces multiplier.

ex.: **32 → 28 → 19 → 13 → 9 → 6 → 4 → 3 → 2**
compare Wallace:
32 → 22 → 15 → 10 → 7 → 5 → 4 → 3 → 2

Luks and Vuillemins multiplier

reduction of 4 addend on 2 addend



$$\boxed{X + Y = A + B + C + D}$$

Latence is same as in 2 full-adders.

Parallel counter

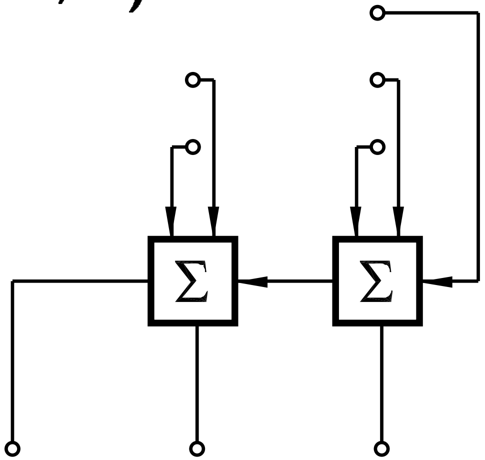
parallel counter $(p_r, p_{r-1}, \ldots, p_0;\ q)$

- **combination circuit**
- q **outputs**
- $p_r + p_{r-1} + \ldots + p_0$ **inputs**
- p_i **outputs has weight** 2^i

ex.: full-adder = paralel counter (3; 2)

ex.: half-adder = paralel counter (2; 2)

ex.: paralel counter (2, 3; 3)



Multiplication in decimal system

Multiplication can be converted on repeated addition and shifts.

ex.: $67 \times 19 = 1273$

$$
\begin{array}{ccccc}
 & & 0 & 0 & 0 \\
+ & & 0 & 6 & 7 \\
+ & & 0 & 6 & 7 \\
+ & & 0 & 6 & 7 \\
+ & & 0 & 6 & 7 \\
+ & & 0 & 6 & 7 \\
+ & & 0 & 6 & 7 \\
+ & & 0 & 6 & 7 \\
+ & & 0 & 6 & 7 \\
+ & & 0 & 6 & 7 \\
+ & 0 & 6 & 7 & \\
\hline
 & 1 & 2 & 7 & 3
\end{array}
$$

Secound operand can be converted to the signed-digit number system.

ex.: $67 \times 19 = 1273$

$19 = 02\widehat{1}$

$$
\begin{array}{cccccc}
 & & & 0 & 0 & 0 \\
- & & & 0 & 6 & 7 \\
+ & & 0 & 6 & 7 & \\
+ & & 0 & 6 & 7 & \\
+ & 0 & 0 & 0 & & \\
\hline
 & & 1 & 2 & 7 & 3
\end{array}
$$

Digits of second operand can be continuously converted e.g. to the weight code 6,3,2,1 with maximum of two 1's in each word.

The one digit of second operand is „processed" in one clock period.
(Otherwise, up to 9 clk's is necessary for 1 digit.
In addition 3 adders are necessary,
(some of them can be „reduced"):
$\mathbf{2x = x+x}$
$\mathbf{3x = 2x+x}$
$\mathbf{6x = 3x+3x}$
In addition, the next adder is needed to create multiple of 4-, 5-, 7-, 8- and 9- and also relevant multiplexers must be added.

	6	3	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	1	0	0
4	0	1	0	1
5	0	1	1	0
6	1	0	0	0
7	1	0	0	1
8	1	0	1	0
9	1	1	0	0

Note.: Analogous, the weight code can be also used 7,4,2,1 (and containing maximally two 1's in each word).
But there is also need to add next adder!

Multiplication in sign and magnitude code:

trivial: **determine sign of result and multiplication of magnitude**

Multiplication in radix complement code:

conversion of second operand to the signed-digit number system
— analogous such as in 2's complement code
(„second fast solution“ cannot be used)

