MI-ARI

(Computer arithmetics)

winter semester 2017/18

PR. **Problems with carry and its accelerating**

© **Alois Pluháček Pavel Kubalík**, 2017

Department of digital design

Faculty of Information technology

Czech Technical University in Prague

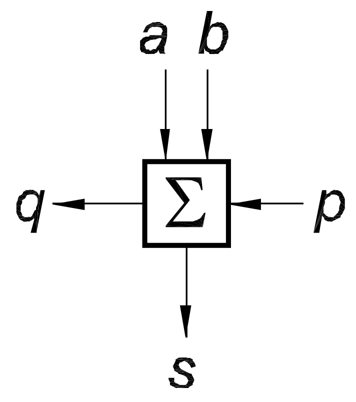
https://edux.fit.cvut.cz/courses/MIE-ARI/lectures

PR. Problems with carry and its accelerating

- **Binary adder**
- **Ripple carry adder**
- **Carry skip adder**
- **Carry look-ahead adder**
- **Conditional sum adder**
- **Asynchronous adder**

Binary adder

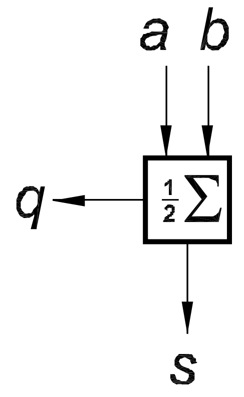
full adder (1 bit binary adder)



$$\begin{aligned} s &= a \oplus b \oplus p = \\ &= \overline{a}\overline{b}p + \overline{a}b\overline{p} + a\overline{b}\overline{p} + abp \end{aligned}$$

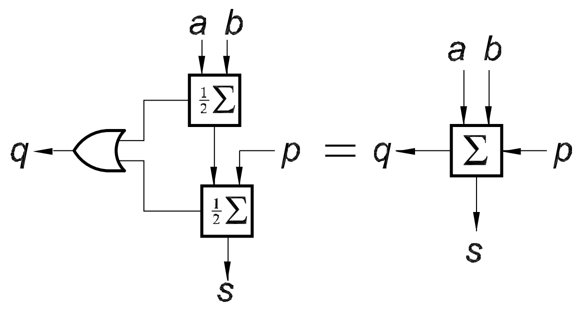
$$\begin{aligned} q &= \mathsf{M}_3(a,b,p) = \\ &= ab + ap + bp = \\ &= ab \oplus ap \oplus bp = \\ &= ab + (ap \oplus bp) \end{aligned}$$

half adder



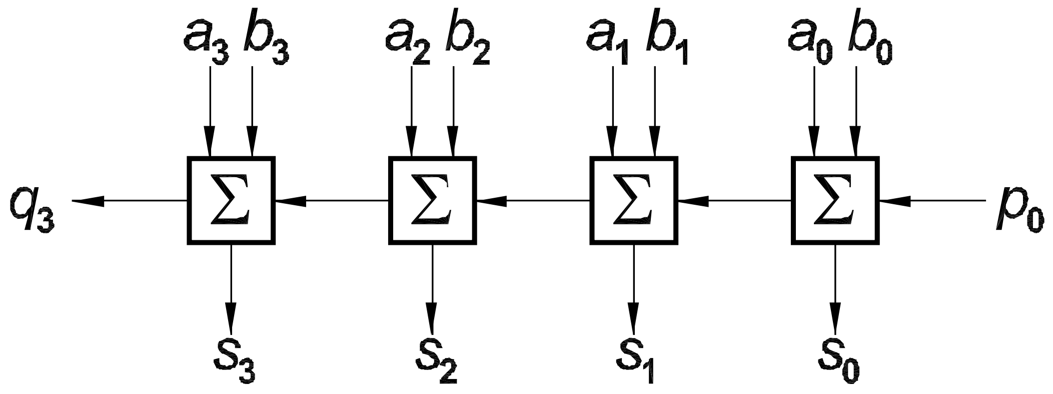
$$\begin{aligned} s &= a \oplus b \\ &= \overline{a}b + a\overline{b} \end{aligned}$$

$$q = a \cdot b$$

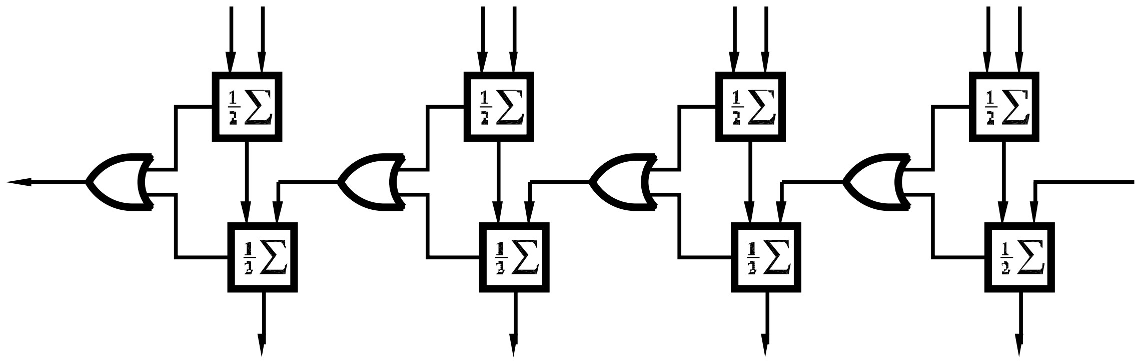


Ripple carry adder

Parallel ripple carry adder:



the same using half-adder:



Ripple carry adder ii

$q_i = a_i b_i + a_i p_i + b_i p_i$ ... **carry for the next order**

$q_i = a_i b_i + (a_i p_i \oplus b_i p_i) = a_i b_i + (a_i \oplus b_i) \cdot p_i$

$a_i = 0$ **a** $b_i = 0 \Rightarrow q_i = 0$ ... **carry in order i is terminated**

$a_i = 1$ **a** $b_i = 1 \Rightarrow q_i = 1$ ... **carry in order i is generated**

else (if $a_i \neq b_i$) $\Rightarrow q_i = p_i$ ... **carry on order i is propagated**

$p_{i+1} = q_i = p_i$

! however in steady state !

p_{i+1} **is delayed towards to p_i**

the delays are accumulated ➠ clock frequency

max. delay must be respected

$q_i = G_i + P_i \cdot p_i$, **where**

$G_i = a_i \cdot b_i$ (carry in order i is generated)

$P_i = a_i \oplus b_i$ (carry is „propagated" in order i)

$$P_i = 1,\ P_{i+1} = 1,\ \ldots,\ P_{i+k} = 1 \quad \Longrightarrow \quad q_{i+k} = p_i$$

$$p_i \cdot P_i \cdot P_{i+1} = 1, \cdot \ldots \cdot P_{i+k} = 1 \quad \Longrightarrow \quad q_{i+k} = 1$$

Carry skip adder

carry skip adder

Adder is divided into section of k positions. The carry which would be propagated throw k positions is bypassed around the section.

■

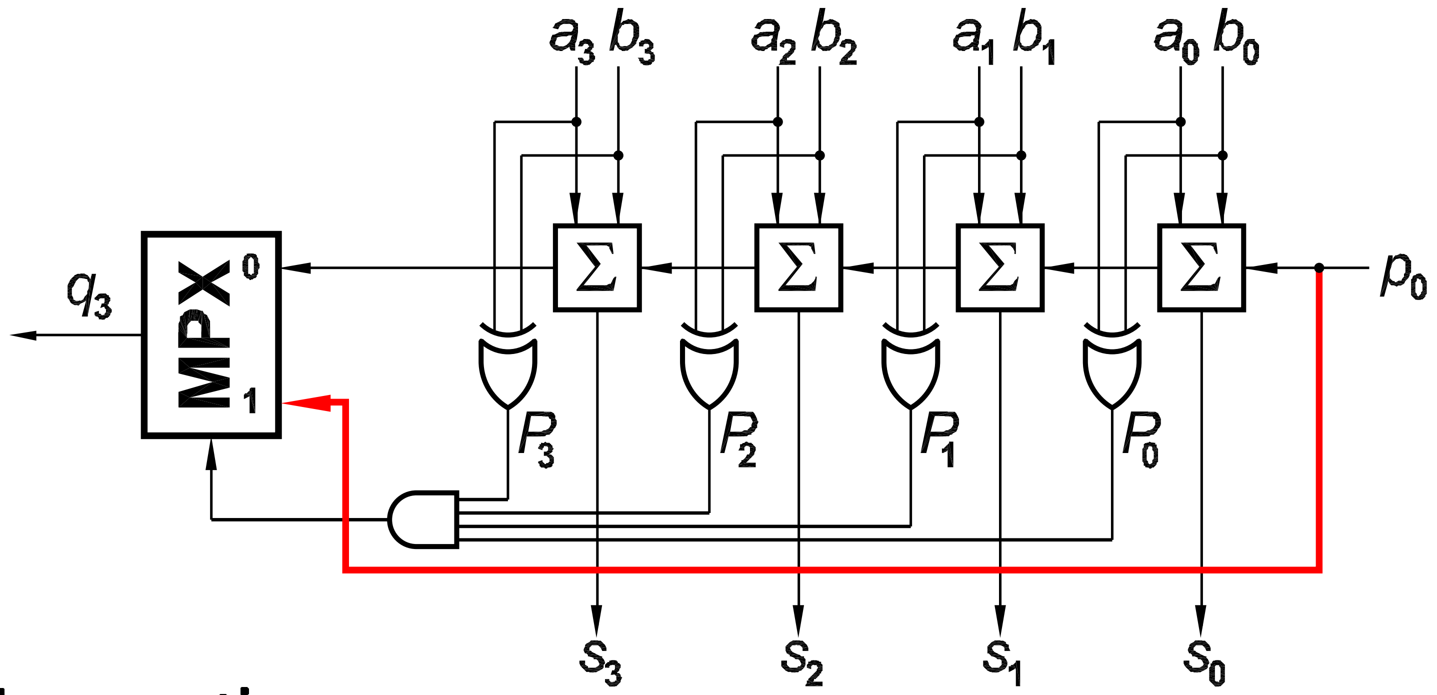
Notes:

- At least 3 sections are needed to reduce the latency.
- Instead of $P_i = a_i \oplus b_i$ this equation $P'_i = P_i + G_i$ can be used.
$$P'_i = a_i + b_i \, .$$
- If the full-adder is composed using two half-adders the output of first half-adder can be used as $P_i = a_i \oplus b_i$.
- The sections can be associated into large sections, which can be called super-sections and next again to bigger super-sections.

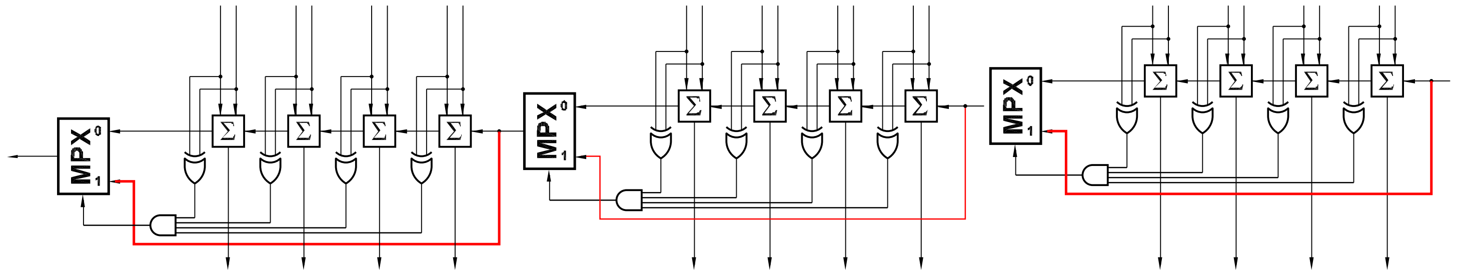
Carry skip adder ii

Ex.: 3 section with 4 positions $\sim$ Total 12 bits

one section:



three sections:



Notes:

- **At least 3 sections are needed to reduce the latency.**
- **Instead of $P_i = a_i \oplus b_i$ this equation $P'_i = P_i + G_i$ can be used.**
$$P'_i = a_i + b_i \,.$$
- **If the full-adder is composed using two half-adders the output of first half-adder can be used as $P_i = a_i \oplus b_i$.**
- **The sections can be associated into large sections, which can be called super-sections and next again to bigger super-sections.**

Carry look-ahead adder

sčítačka s predikcí přenosů [carry look-ahead adder]

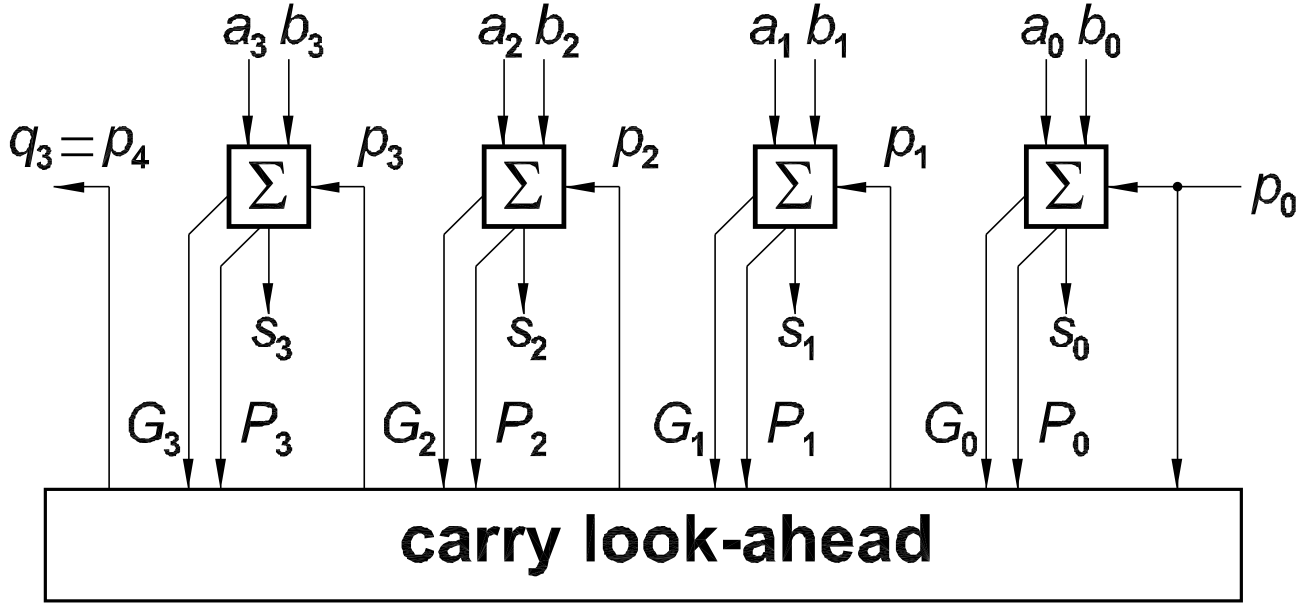
$$\begin{aligned} q &= ab + (ap \oplus bp) = ab + (a \oplus b) \cdot p = G + P \cdot p \\ &= ab + ap + bp \quad = ab + (a + b) \cdot p = G + P' \cdot p \end{aligned}$$

$G_i = a_i \cdot b_i$ **carry is generated in order** i
$P_i = a_i \oplus b_i$ **carry is propagated in order** i
$P'_i = a_i + b_i$ **carry is generated or propageted in order** i

$q_0 = p_1 = G_0 + P_0 \cdot p_0$ $\qquad q_1 = p_2 = G_1 + P_1 \cdot p_1$
$q_1 = p_2 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot p_0$
$q_2 = p_3 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot p_0$
etc.

The P'_i can be used instead of P_i.

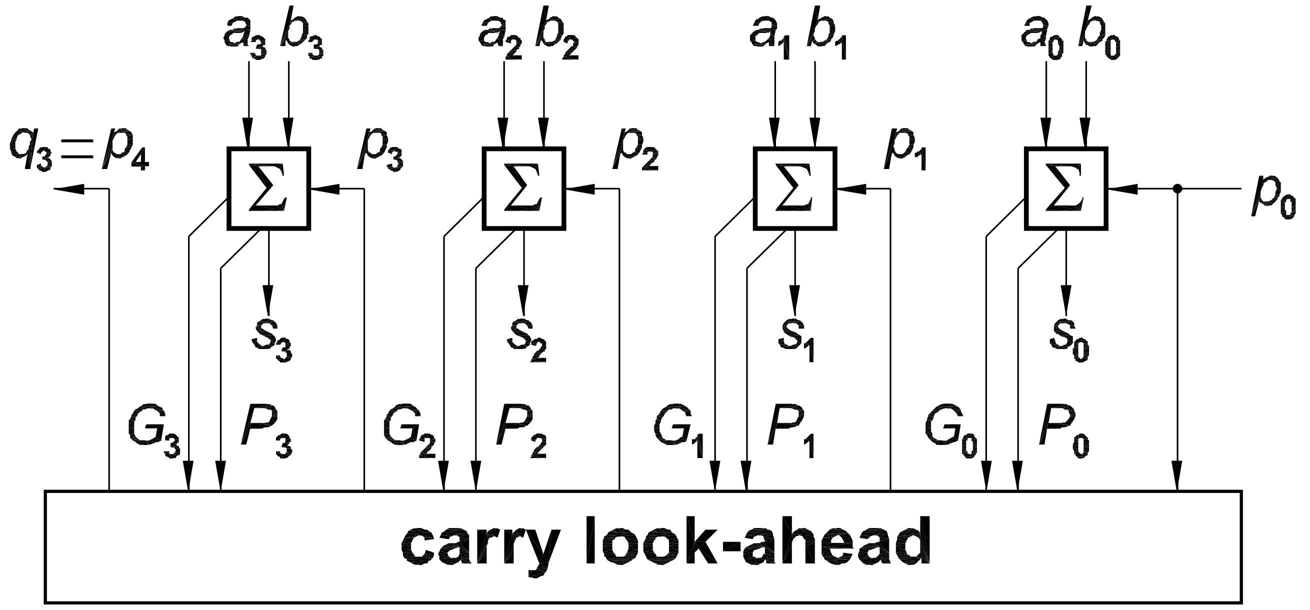
Carry look-ahead adder ii



carry look-ahead:

$$p_1 = G_0 + P_0 \cdot p_0$$
$$p_2 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot p_0$$
$$p_3 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot p_0$$
$$p_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + \\ + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot p_0$$

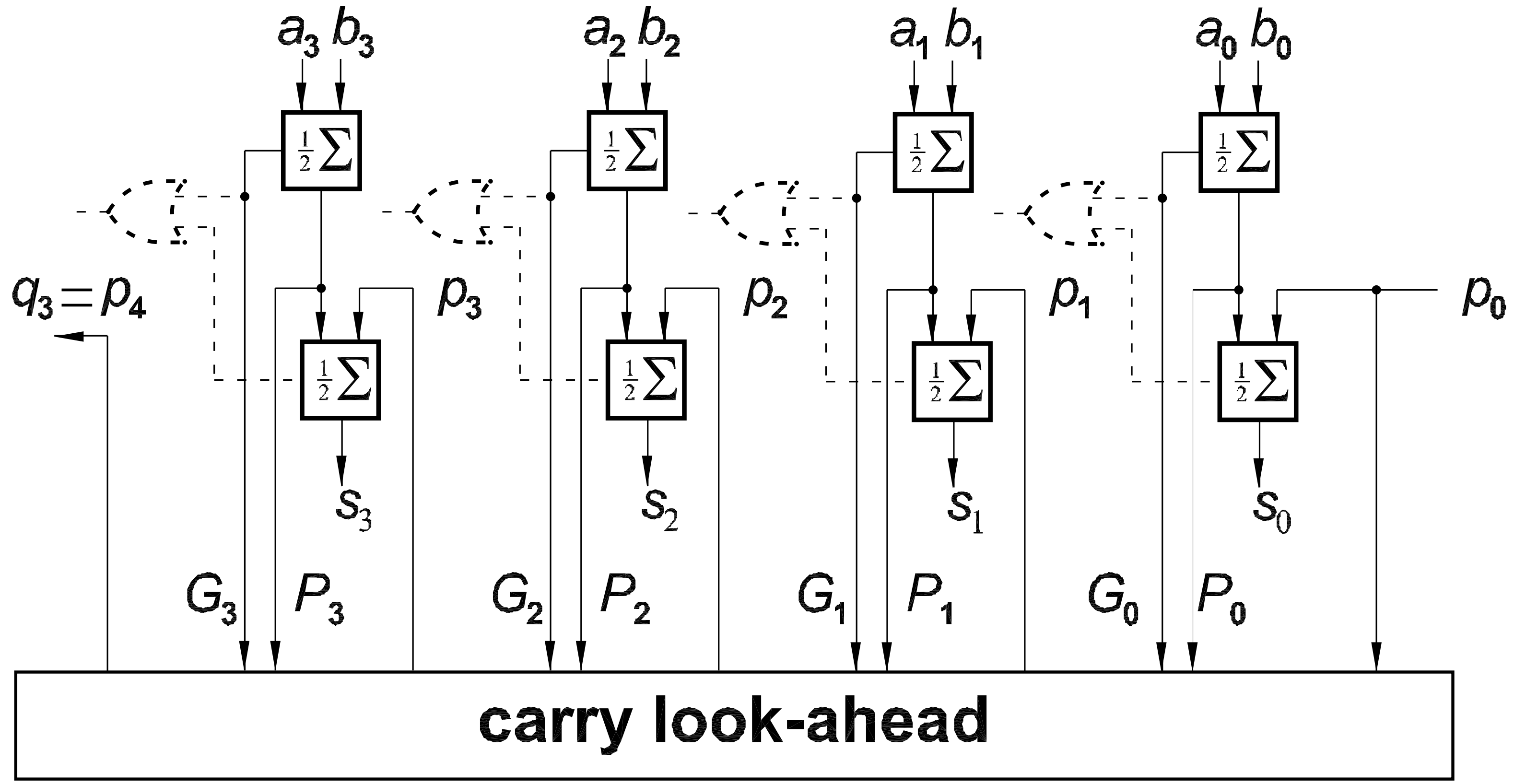
Carry look-ahead adder iii



carry look-ahead:

$$p_1 = G_0 + P_0 \cdot p_0$$
$$p_2 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot p_0$$
$$p_3 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot p_0$$
$$p_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + \\ + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot p_0$$

carry look-ahead adder using half-adders



Carry look-ahead adder — cascade scheme

4bit section (as an example):

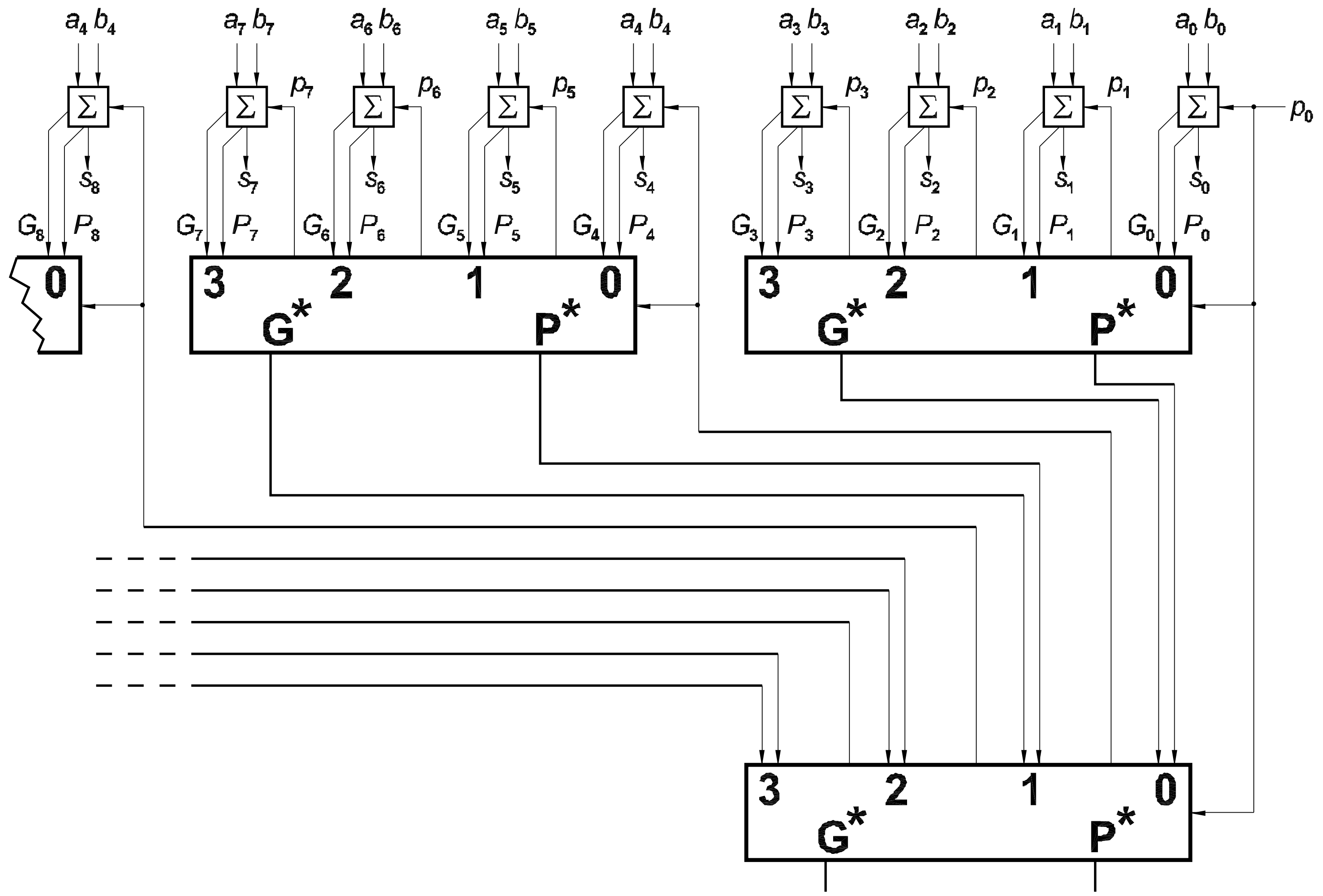
$$\begin{aligned}
p_1 &= G_0 + P_0 \cdot p_0 \\
p_2 &= G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot p_0 \\
p_3 &= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot p_0 \\
p_4 &= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + \\
&\qquad + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot p_0
\end{aligned}$$

$$\begin{aligned}
G^* &= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 \\
P^* &= P_3 \cdot P_2 \cdot P_1 \cdot P_0
\end{aligned}$$

G^* ... **carry-out is generated from its higher order**
P^* ... **carry is propagated in this section**

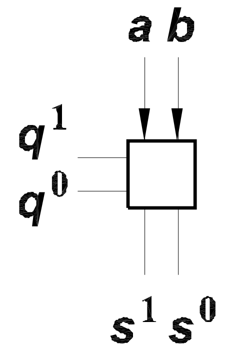
$$p_4 = G^* + P^* \cdot p_0$$

Carry look-ahead adder — cascade scheme ii



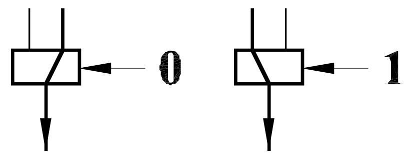
conditional sum adder

basic logic element:

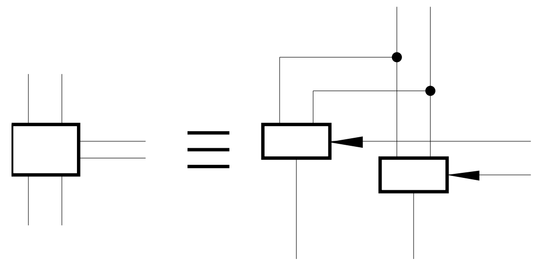


$$\begin{array}{llll} s^0 & = & a \oplus b & \ldots \text{ bit of sum for } p = 0 \\ s^1 & = & a \oplus b \oplus 1 & \ldots \text{ bit of sum for } p = 1 \\ q^0 & = & a \cdot b & \ldots \text{ next carry for } p = 0 \\ q^1 & = & a + b & \ldots \text{ next carry for } p = 1 \end{array}$$

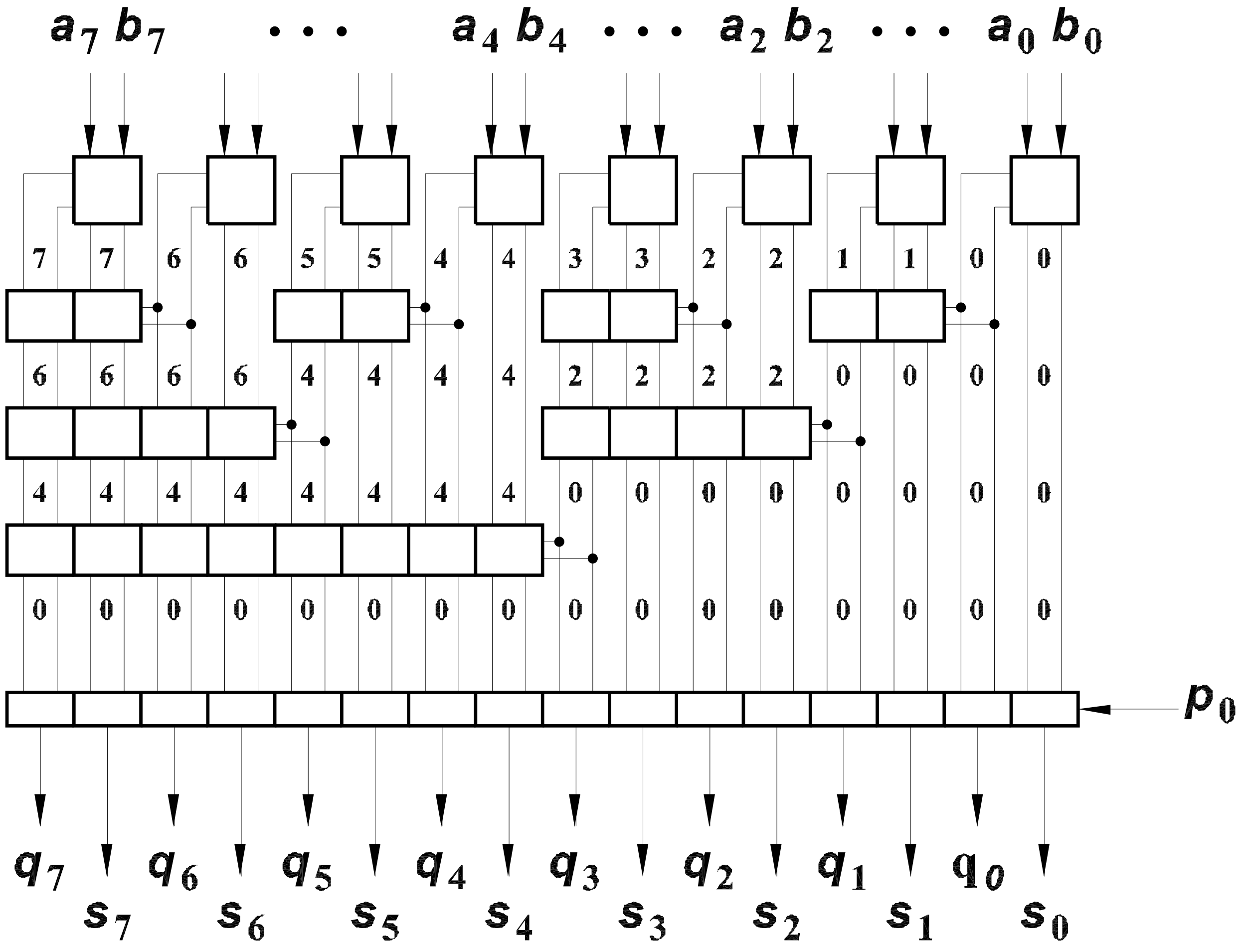
multiplexor:



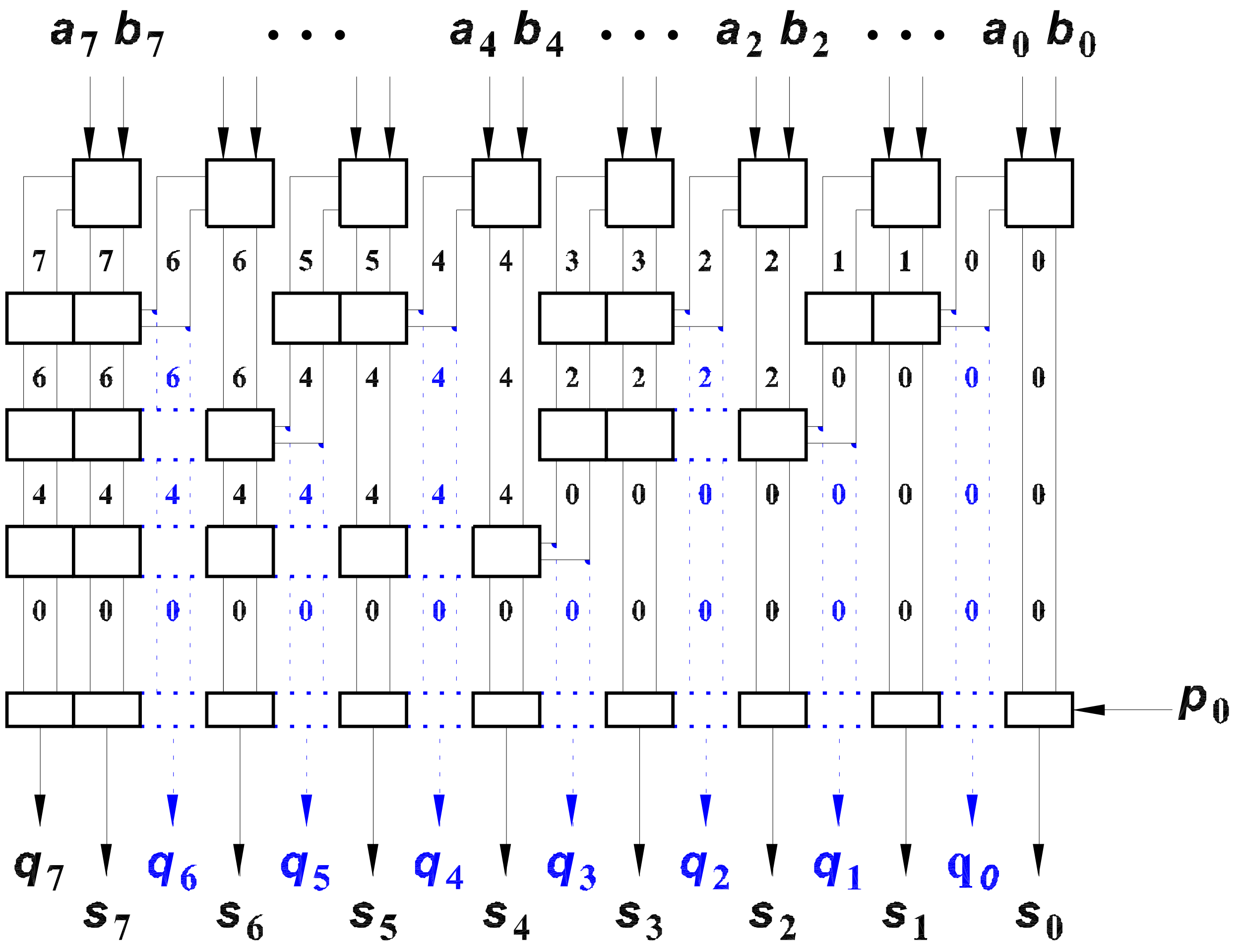
double of multiplexor:



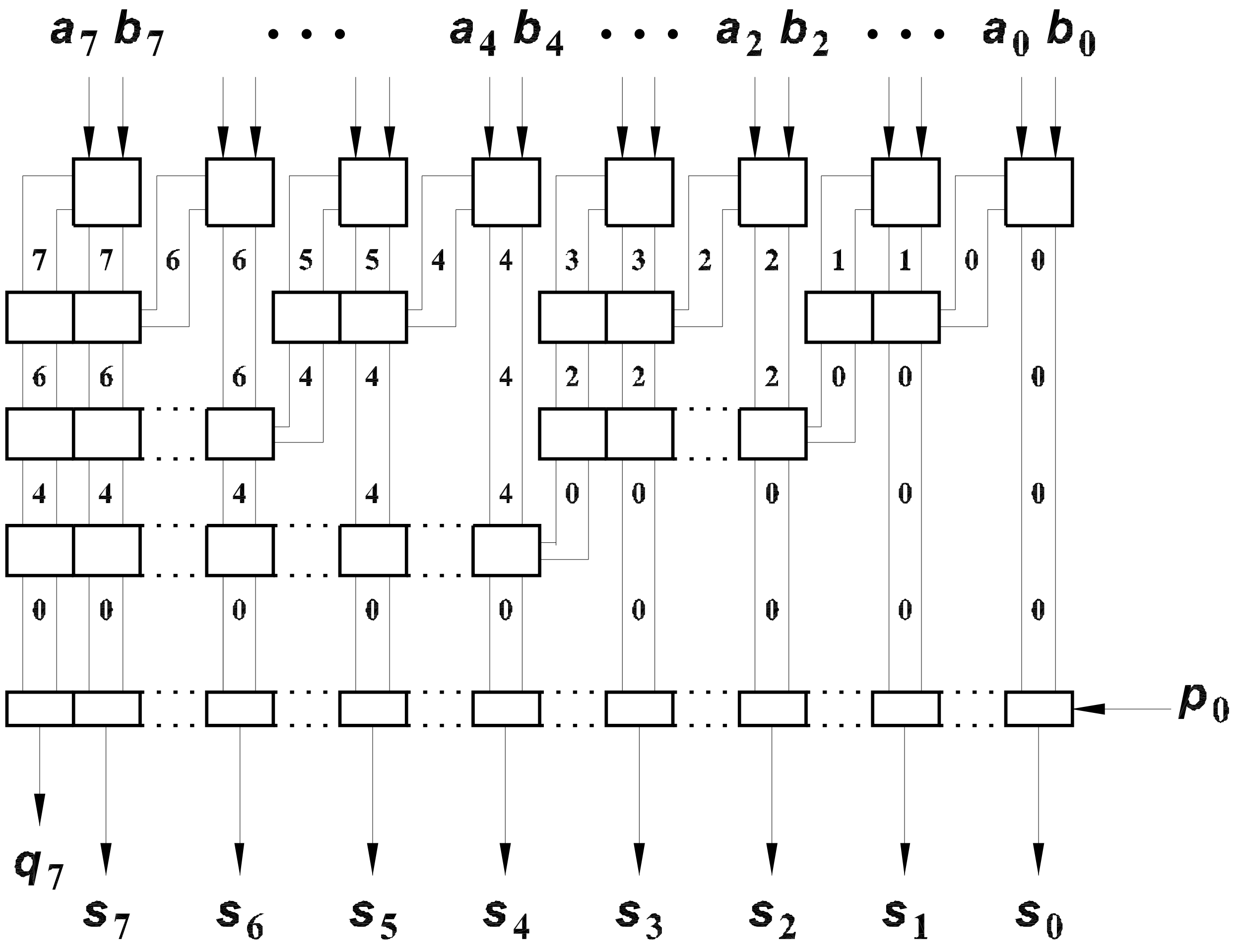
8bits adder scheme


a_7 b_7 . . . a_4 b_4 . . . a_2 b_2 . . . a_0 b_0
7 7 6 6 5 5 4 4 3 3 2 2 1 1 0 0
6 6 6 6 4 4 4 4 2 2 2 2 0 0 0 0
4 4 4 4 4 4 4 4 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
p_0
q_7 s_7 q_6 s_6 q_5 s_5 q_4 s_4 q_3 s_3 q_2 s_2 q_1 s_1 q_0 s_0

Conditional sum adder iii


$a_7\ b_7\ \dots\ a_4\ b_4\ \dots\ a_2\ b_2\ \dots\ a_0\ b_0$
7 7 6 6 5 5 4 4 3 3 2 2 1 1 0 0
6 6 6 6 4 4 4 4 2 2 2 2 0 0 0 0
4 4 4 4 4 4 4 4 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
p_0
$q_7\ q_6\ q_5\ q_4\ q_3\ q_2\ q_1\ q_0$
$s_7\ s_6\ s_5\ s_4\ s_3\ s_2\ s_1\ s_0$

If it is not necessary determine carry from order 0, 1, . . . , 6, then the given multiplexers can be released:



Asynchronous adder

We take into account ripple carry adder:

length L of critical path: max. number of neighboring orders, over them the carry is propagated
(it means, that carry is not generated or deleted in this orders)

!!! length L depends on specific values of addends !!!

N **. . . number of bits of both addends A and B**

$$0 \le L \le N$$

The worst case is used to determine frequency of clock cycle, it means:

$$L = N.$$

But (statistically) the mean value is equal only to:

$$\boxed{L = \lceil \log_2 \tfrac{4}{5} N \rceil}\ ;$$

e.g. Mean value for $N = 64$ is $L = 7 \ll 64.$

Asynchronous adder *ii*

The idea is, that it is possible to wait for minimal required time before the next clock pulse start. But this require:

1. **detect** (or somehow determine)**, that the transition process in combinational circuit is ended,**
2. **to use generator driven by some condition** (that these processes are ended)**.**

ad 1: **Adder can be even most problematic circuit due to the big delay.**

ad 2: **The driven clock cycle generator is used in asynchronous computers.**

Asynchronous computer **is based on** ***synchronous sequential circuit*****, where exist different delays between clock pulses.**

$\boxed{\textbf{asynchronous adder} \ \in \ \textbf{asynchronous computer}}$

One-digit asynchronous adder

one-digit asynchronous adder (JAS)

- **„replacement“ of full adder,**
- **output signalizing valid output carry ($h \sim$ „done“).**

basic principle:

The carry is determine independently

$$q = a\,b + a\,p + b\,p$$

and its negation

$$\overline{q} = \overline{a}\,\overline{b} + \overline{a}\,\overline{p} + \overline{b}\,\overline{p}$$

In steady state must be satisfied

$$q = 1 \quad \text{or} \quad \overline{q} = 1,$$

it must be

$$q + \overline{q} = 1.$$

One-digit asynchronous adder ii

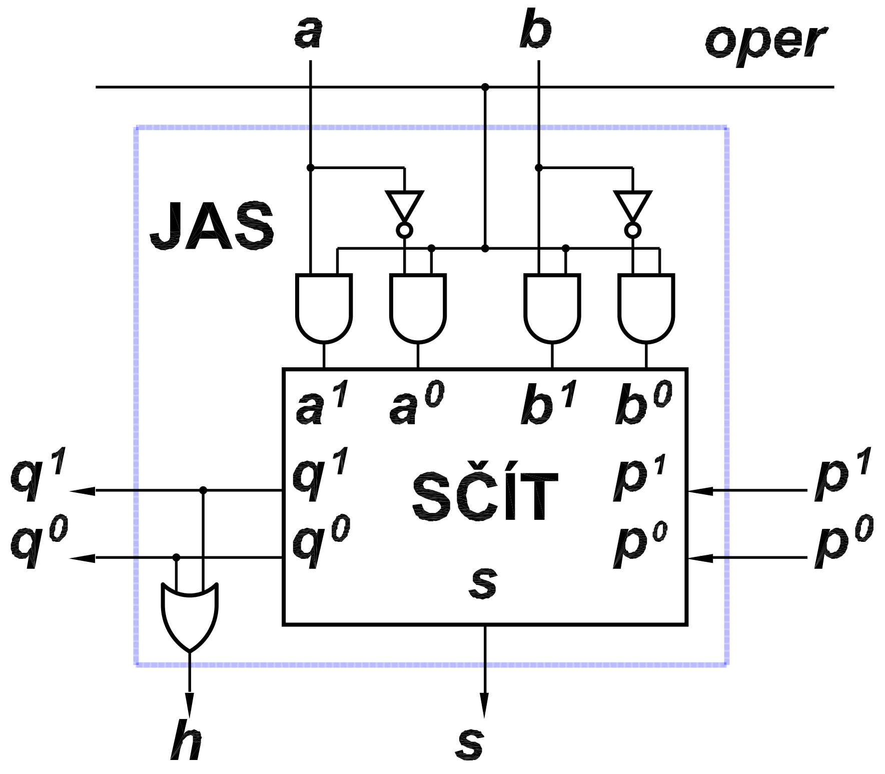
$$\boxed{\begin{array}{l} q^0 = a^0 b^0 + a^0 p^0 + b^0 p^0 \\ q^1 = a^1 b^1 + a^1 p^1 + b^1 p^1 \end{array}}$$

$$\left.\begin{array}{l} a^0 = 0 \\ b^0 = 0 \\ \\ a^1 = 0 \\ b^1 = 0 \end{array}\right\} \Rightarrow \begin{cases} q^0 = 0 \\ q^1 = 0 \end{cases} \qquad \left.\begin{array}{l} a^0 = \overline{a} \\ b^0 = \overline{b} \\ p^0 = \overline{p} \\ a^1 = a \\ b^1 = b \\ p^1 = p \end{array}\right\} \Rightarrow \begin{cases} q^0 = \overline{q} \\ q^1 = q \end{cases}$$

$$a^0 = \overline{a}, \quad b^0 = \overline{b}, \quad a^1 = a, \quad b^1 = b$$

$$\left.\begin{array}{lll} a = b = 0 & \Rightarrow & q^0 = 1 \\ a = b = 1 & \Rightarrow & q^1 = 1 \\ a \neq b \text{ a } p^0 = 1 & \Rightarrow & q^0 = 1 \\ a \neq b \text{ a } p^1 = 1 & \Rightarrow & q^1 = 1 \end{array}\right\} \qquad h = q^0 + q^1 = 1$$

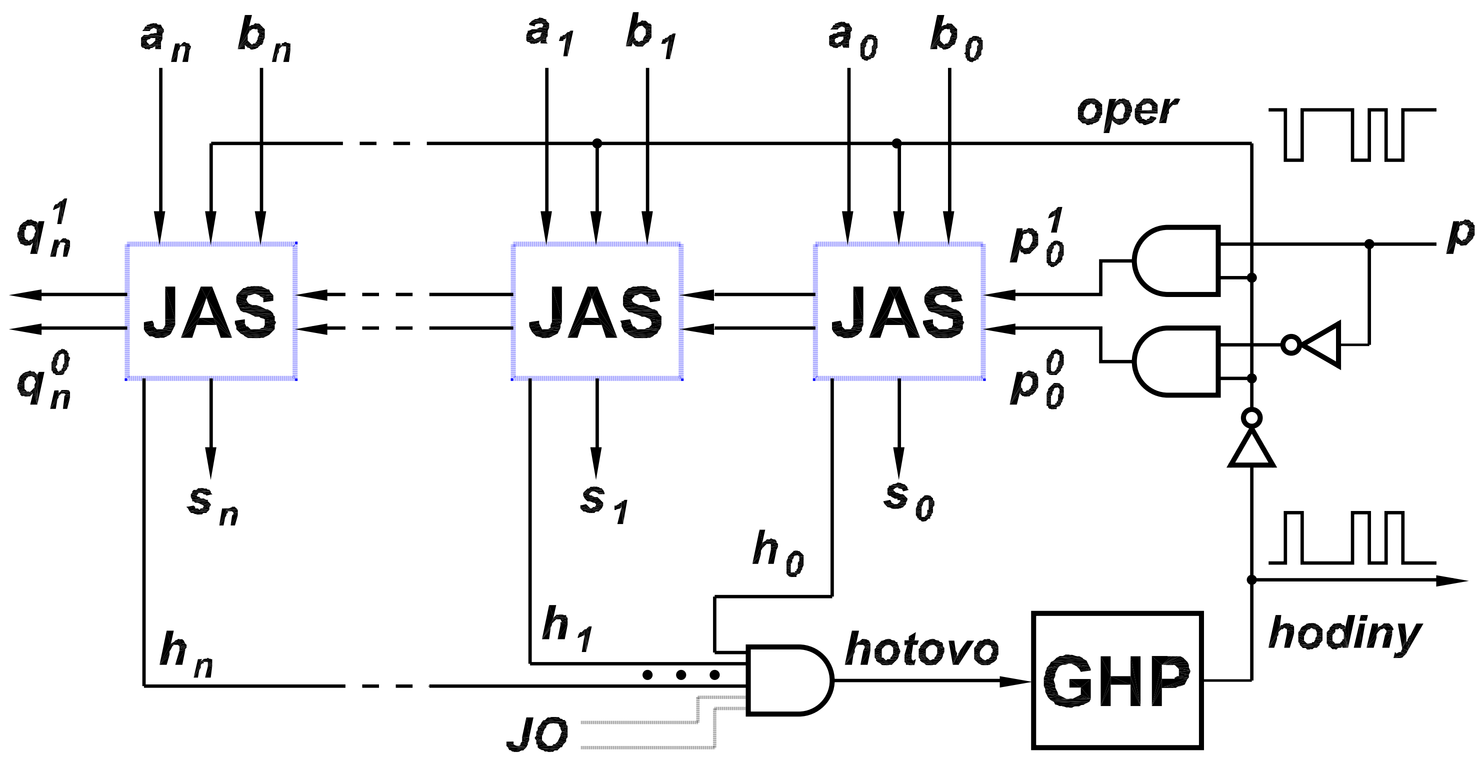
One-digit asynchronous adder *iii*



$$q^0 = a^0 b^0 + a^0 p^0 + b^0 p^0$$
$$q^1 = a^1 b^1 + a^1 p^1 + b^1 p^1$$
$$s = a^1 \oplus b^1 \oplus p^1$$

(n+1) bit asynchronous adder

asynchronous adder in asynchronous computer



GHP . . . generator of clock pulse

JO . . . other circuit are in steady state