

# Carry Skip Adder (5A)

---

- 
-

Copyright (c) 2021 – 2013 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

[https://en.wikipedia.org/wiki/AND\\_gate](https://en.wikipedia.org/wiki/AND_gate)  
[https://en.wikipedia.org/wiki/OR\\_gate](https://en.wikipedia.org/wiki/OR_gate)  
[https://en.wikipedia.org/wiki/XOR\\_gate](https://en.wikipedia.org/wiki/XOR_gate)  
[https://en.wikipedia.org/wiki/NAND\\_gate](https://en.wikipedia.org/wiki/NAND_gate)

Please send corrections (or suggestions) to [youngwlim@hotmail.com](mailto:youngwlim@hotmail.com).

This document was produced by using OpenOffice and Octave.

# Carry Lookahead Adder

## Carry Lookahead Adder

$$p_i = a_i \oplus b_i$$

$$g_i = a_i \wedge b_i$$

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

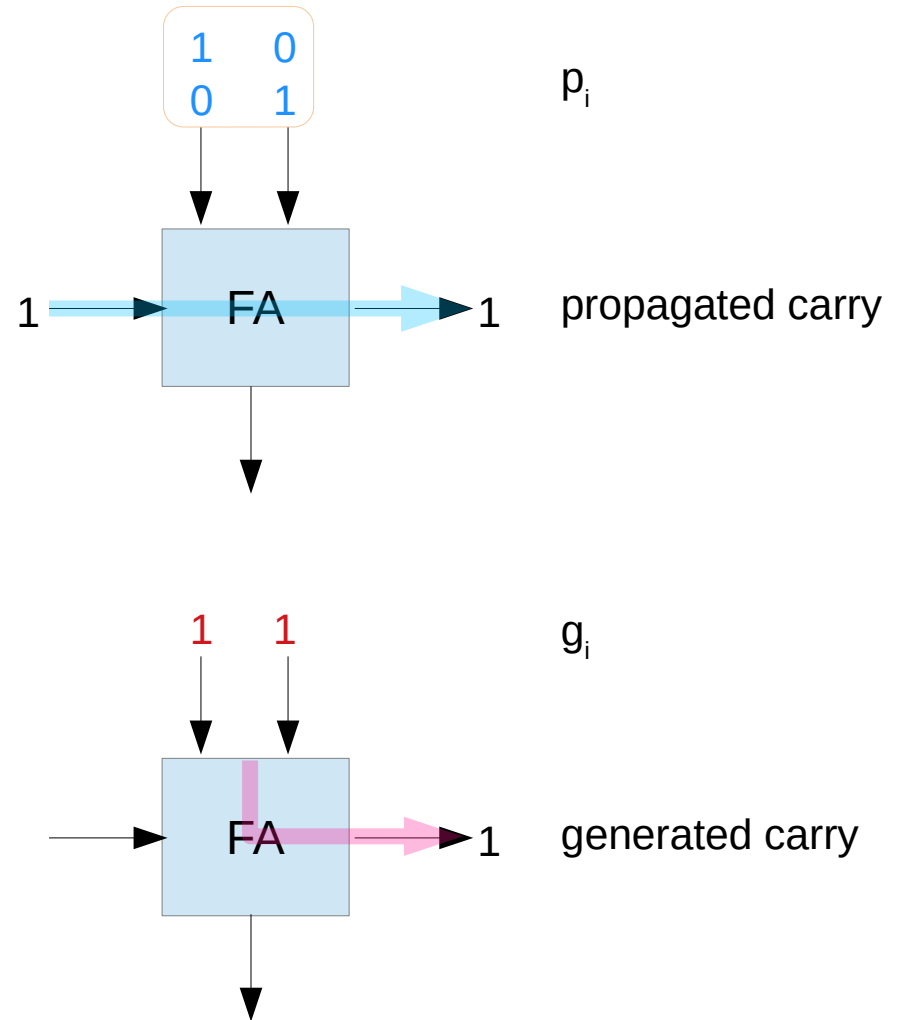
$$c_2 = g_1 + p_1 \wedge c_1$$

$$c_3 = g_2 + p_2 \wedge c_2$$

$$c_4 = g_3 + p_3 \wedge c_3$$

generated carry

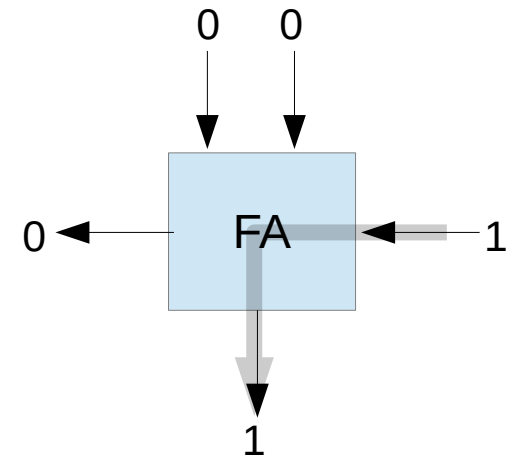
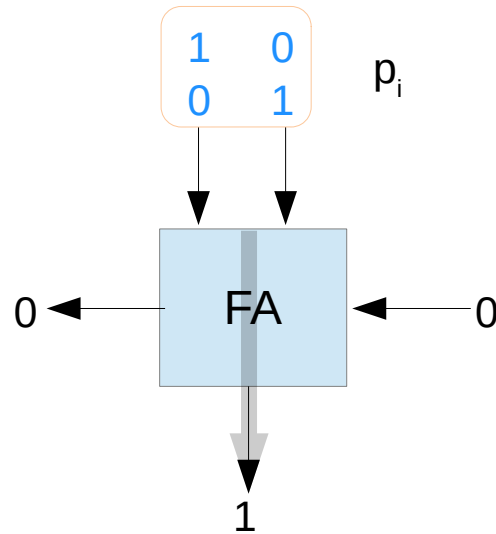
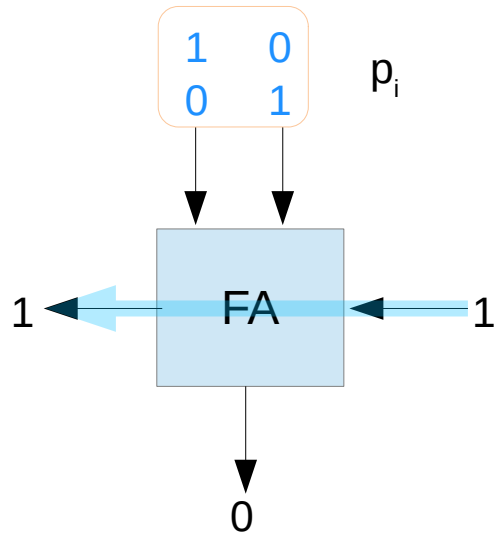
propagated carry



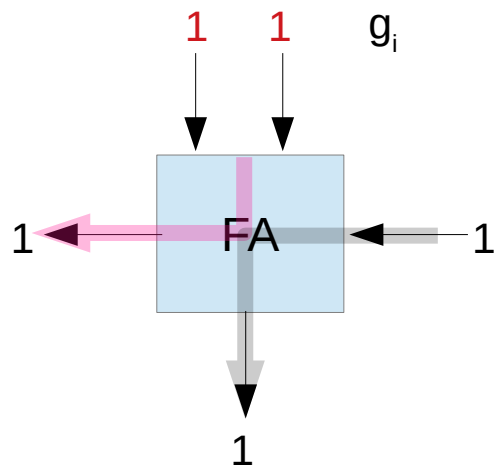
[https://en.wikipedia.org/wiki/Carry-skip\\_adder](https://en.wikipedia.org/wiki/Carry-skip_adder)

# Propagated and Generated Carries

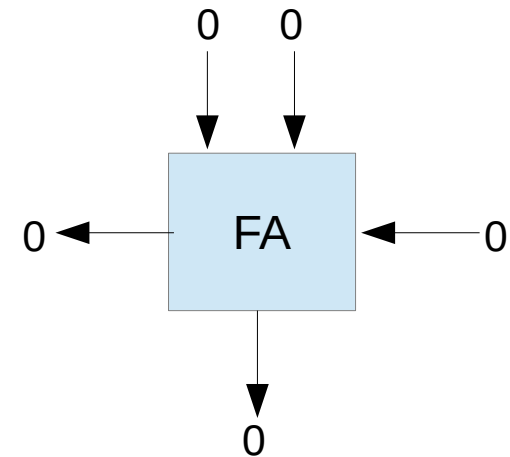
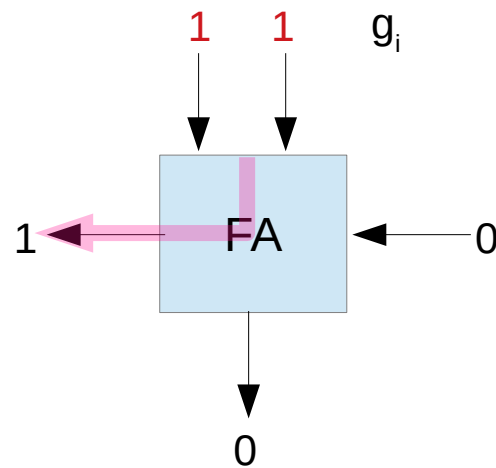
propagated carry



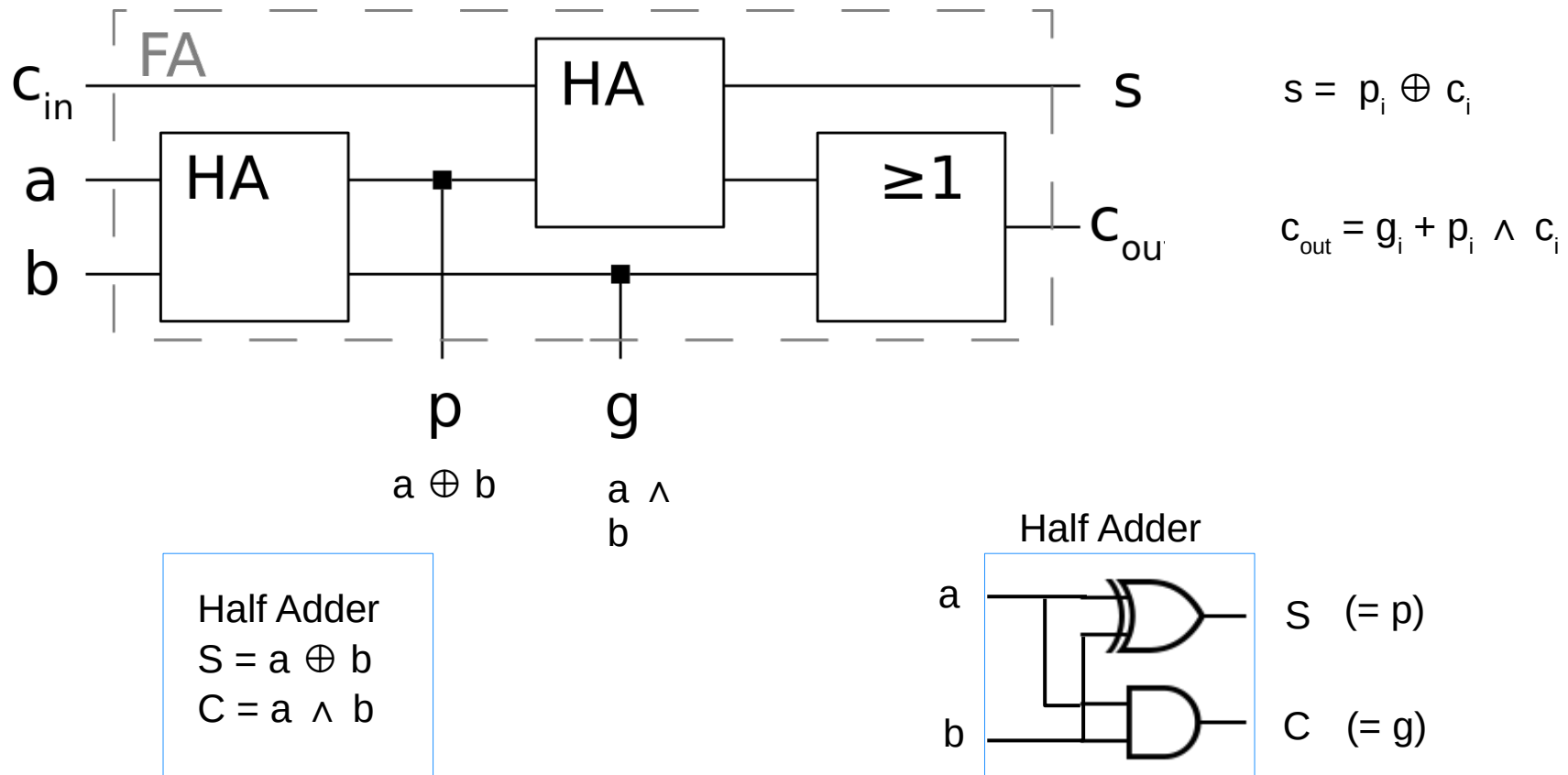
generated carry



generated carry



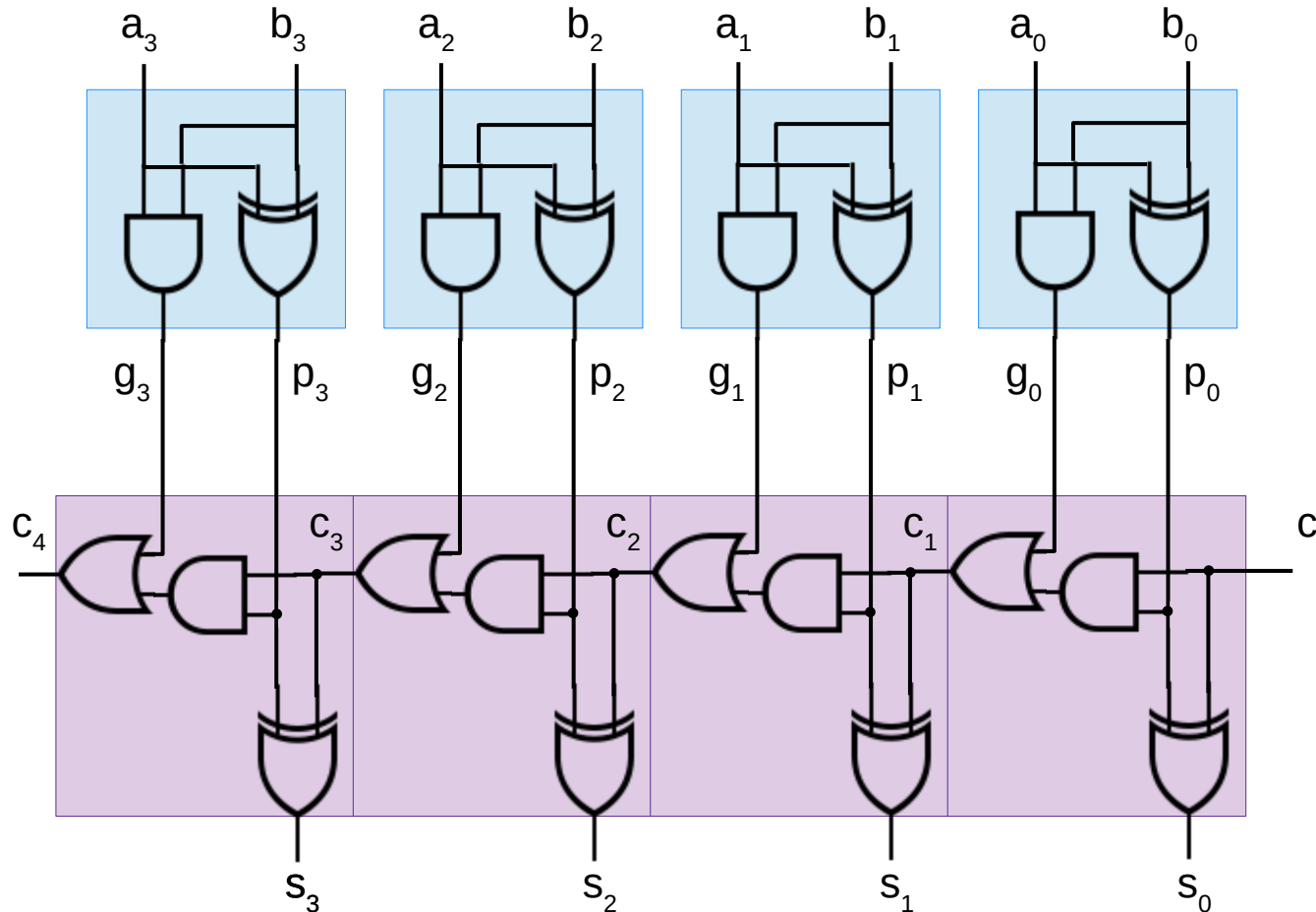
# FA with P & G



Full adder with additional generate and propagate signals.

[https://en.wikipedia.org/wiki/Carry-skip\\_adder](https://en.wikipedia.org/wiki/Carry-skip_adder)

# 4-bit Full Adder with P and G



## Half Adder

$$p_i = a_i \oplus b_i$$

$$g_i = a_i \wedge b_i$$

$$c_{i+1} = g_i + p_i c_i$$

$$s_i = p_i \oplus c_i$$

<https://upload.wikimedia.org/wikiversity/en/1/18/RCA.Note.H.1.20151215.pdf>

# FA with P & G

For each operand input bit pair  $(a_i, b_i)$

the propagate-conditions  $p_i = a_i \oplus b_i$  are determined using an XOR-Gate .

When all propagate-conditions are true,

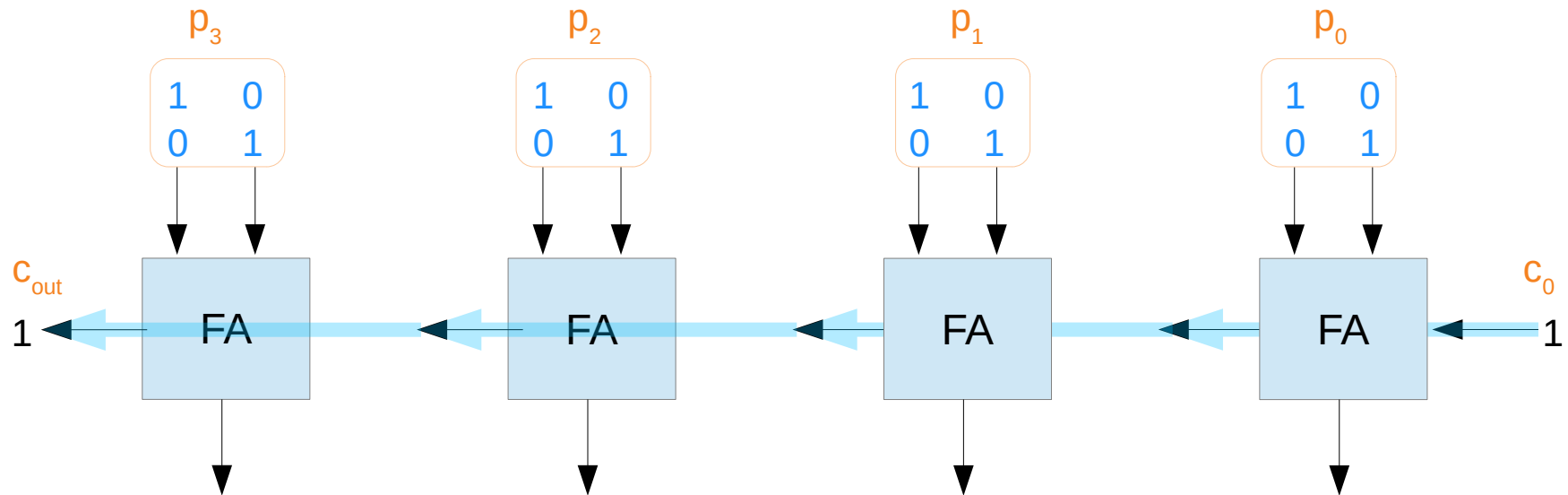
$$\begin{aligned} s &= p_{n-1} \wedge p_{n-2} \wedge \cdots \wedge p_1 \wedge p_0 = p_{[0:n-1]} \\ &= (a_{n-1} \oplus b_{n-1}) \wedge (a_{n-2} \oplus b_{n-2}) \wedge \cdots \wedge (a_1 \oplus b_1) \wedge (a_0 \oplus b_0) \end{aligned}$$

then the carry-in bit  $c_0$  determines the carry-out bit.

$c_0$  can be propagated to  $c_{out}$  only when  $s = 1$

[https://en.wikipedia.org/wiki/Carry-skip\\_adder](https://en.wikipedia.org/wiki/Carry-skip_adder)

# $C_0$ propagation condition



$c_0$  can be propagated to  $c_{out}$  only when  $s = 1$

$$s = p_{n-1} \wedge p_{n-2} \wedge \cdots \wedge p_1 \wedge p_0 = p_{[0:n-1]}$$
$$= (a_{n-1} \oplus b_{n-1}) \wedge (a_{n-2} \oplus b_{n-2}) \wedge \cdots \wedge (a_1 \oplus b_1) \wedge (a_0 \oplus b_0)$$

[https://en.wikipedia.org/wiki/Carry-skip\\_adder](https://en.wikipedia.org/wiki/Carry-skip_adder)



# FA with P & G

The n-bit-carry-skip adder consists of  
a n-bit **carry-ripple-chain**,  
a n-input **AND-gate** and  
one **multiplexer**.

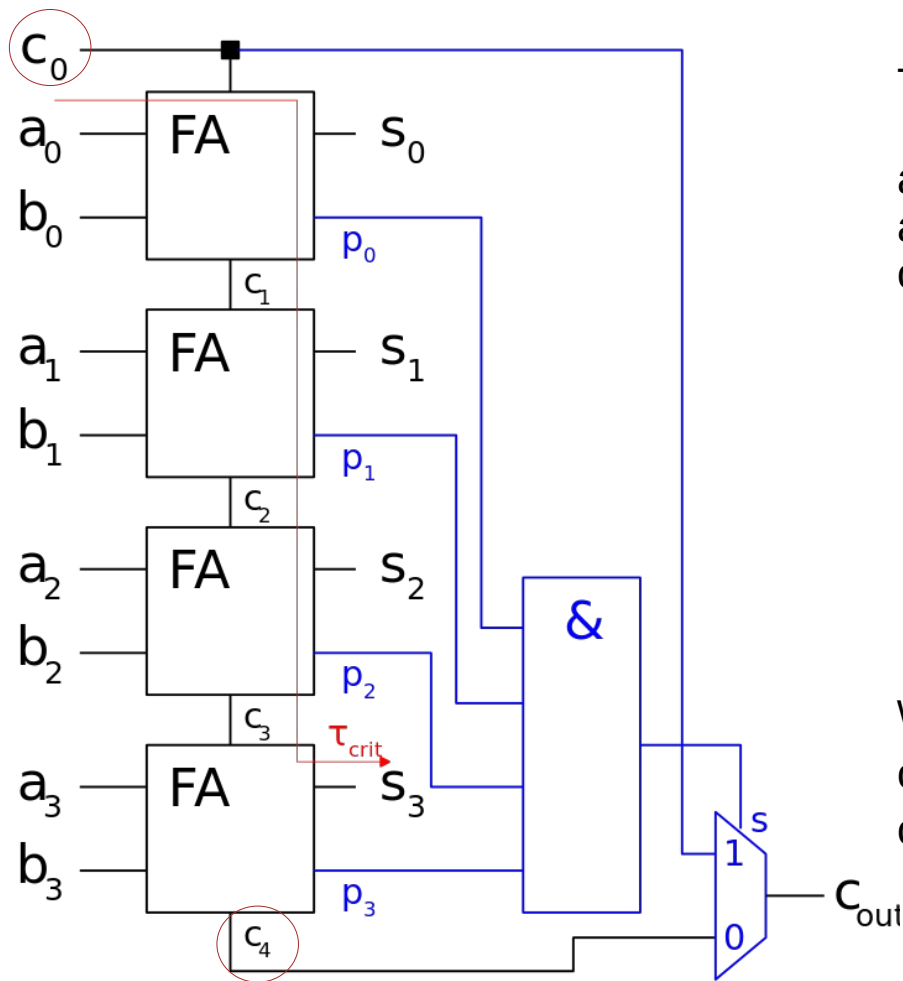
Each propagate bit  $p_i$  that is provided by the carry-ripple-chain  
is connected to the n-input AND-gate.

The resulting bit is used as the select bit of a multiplexer  
that switches either the last carry-bit  $c_n$  or the carry-in  $c_0$   
to the carry-out signal  $c_{out}$

$$s = p_{n-1} \wedge p_{n-2} \wedge \cdots \wedge p_1 \wedge p_0 = p_{[0:n-1]}$$

[https://en.wikipedia.org/wiki/Carry-skip\\_adder](https://en.wikipedia.org/wiki/Carry-skip_adder)

# 4-bit Carry Skip Adder



The n-bit-carry-skip adder consists of

a n-bit **carry-ripple-chain**,  
a n-input **AND-gate** and  
one **multiplexer**.

a multiplexer switches  
either the last carry-bit  $c_n$  or the carry-in  $c_0$   
to the carry-out signal  $c_{out}$

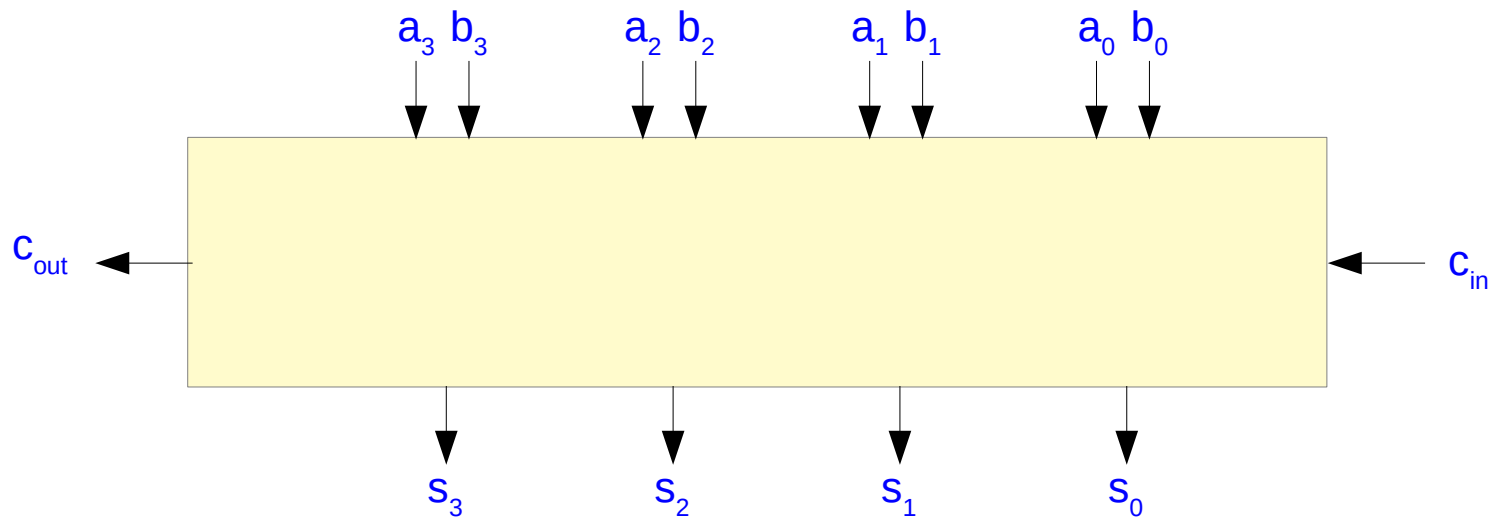
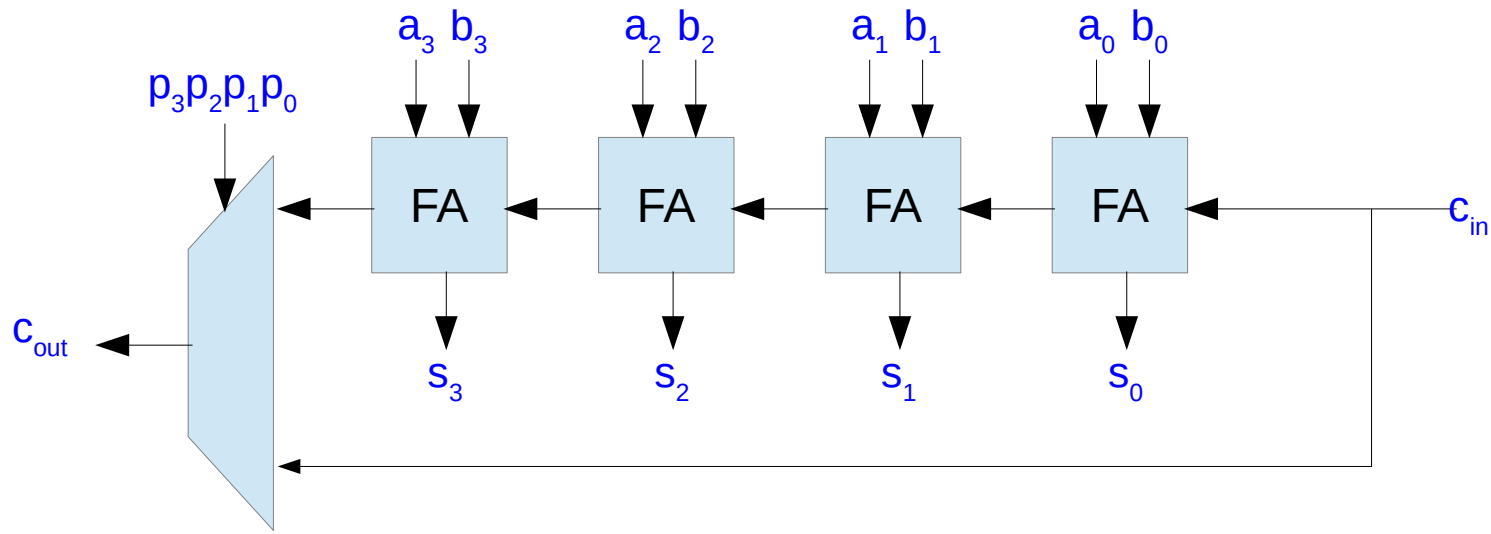
$$s = p_{n-1} \wedge p_{n-2} \wedge \cdots \wedge p_1 \wedge p_0 = p_{[0:n-1]}$$

when  $s = 1$ ,  $c_{out} \leftarrow c_0$

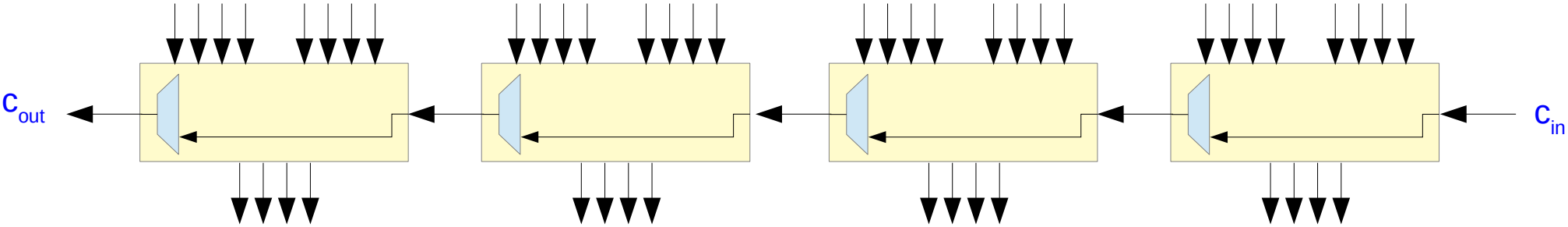
otherwise, internally generated carries  
can be propagated to  $c_{out}$

[https://en.wikipedia.org/wiki/Carry-skip\\_adder](https://en.wikipedia.org/wiki/Carry-skip_adder)

# Carry Skip Adder



# Carry Skip Adder



# Block Carry Skip Adder

Block-carry-skip adders are composed of a number of carry-skip adders. There are two types of block-carry-skip adders. The two operands

$A = (a_{n-1}, a_{n-2}, \dots, a_1, a_0)$  and  $B = (b_{n-1}, b_{n-2}, \dots, b_1, b_0)$  are split in  $k$  blocks of  $(m_k, m_{k-1}, \dots, m_2, m_1)$  bits.

- Why are block-carry-skip-adders used?
- Should the block-size be constant or variable?
- Fixed block width vs. variable block width

[https://en.wikipedia.org/wiki/Carry-skip\\_adder](https://en.wikipedia.org/wiki/Carry-skip_adder)



# Carry Skip Adder

Since the **Cin-to-Cout** represents the longest path in the ripple-carry-adder, an obvious attempt is to accelerate carry propagation through the adder.

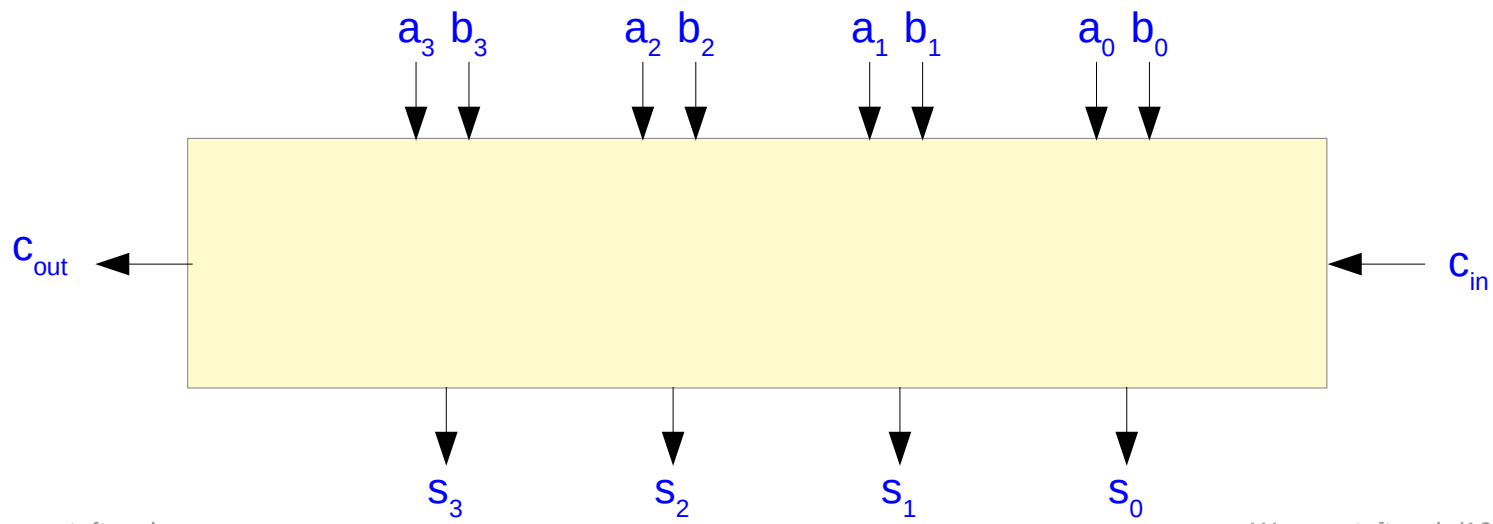
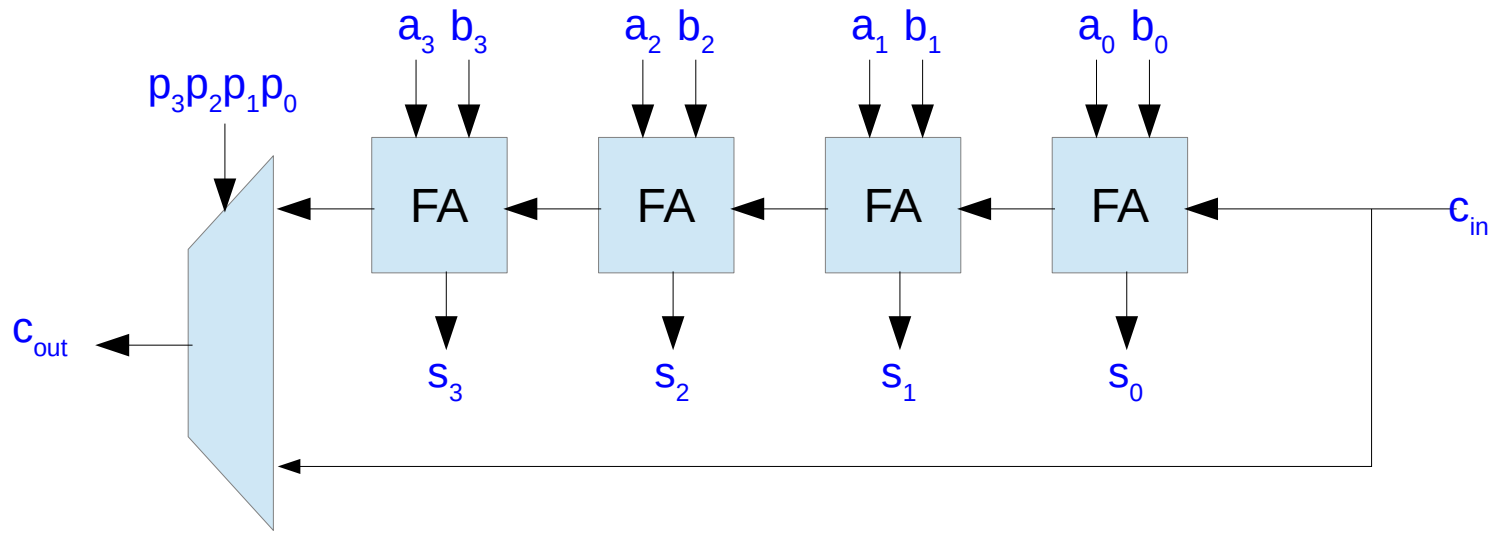
This is accomplished by using **Carry-Propagate**  $p_i$  signals within a group of bits.

If all the  $p_i$  signals within the group are  $p_i = 1$ , the condition exist for the carry to bypass the entire group:

$$P = p_i \cdot p_{i+1} \cdot p_{i+2} \cdot \dots \cdot p_{i+k-1}$$

Oklobdzija: High-Speed VLSI arithmetic units : adders and multipliers

# Carry Skip Adder





# Carry Skip Adder

The **Carry Skip Adder** (CSKA) divides the words to be added into groups of equal size of **k-bits**.

The basic structure of an **N-bit Carry Skip Adder**

Within the group, carry propagates in a ripple-carry fashion.

In addition, an AND gate is used to form the **group propagate** signal **P**.

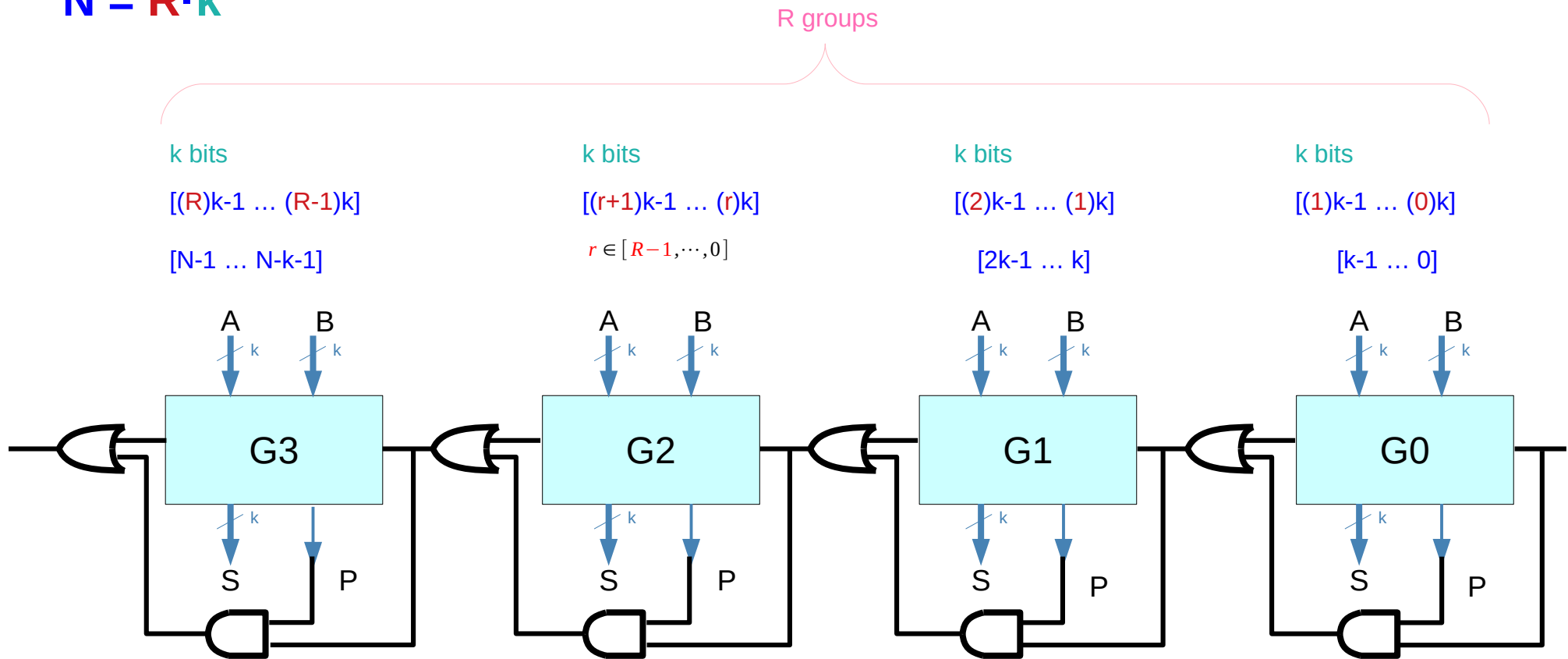
$$P = p_i \cdot p_{i+1} \cdot p_{i+2} \cdot \dots \cdot p_{i+k-1}$$

If  $P = 1$  the condition exists for carry to bypass (skip) over the group

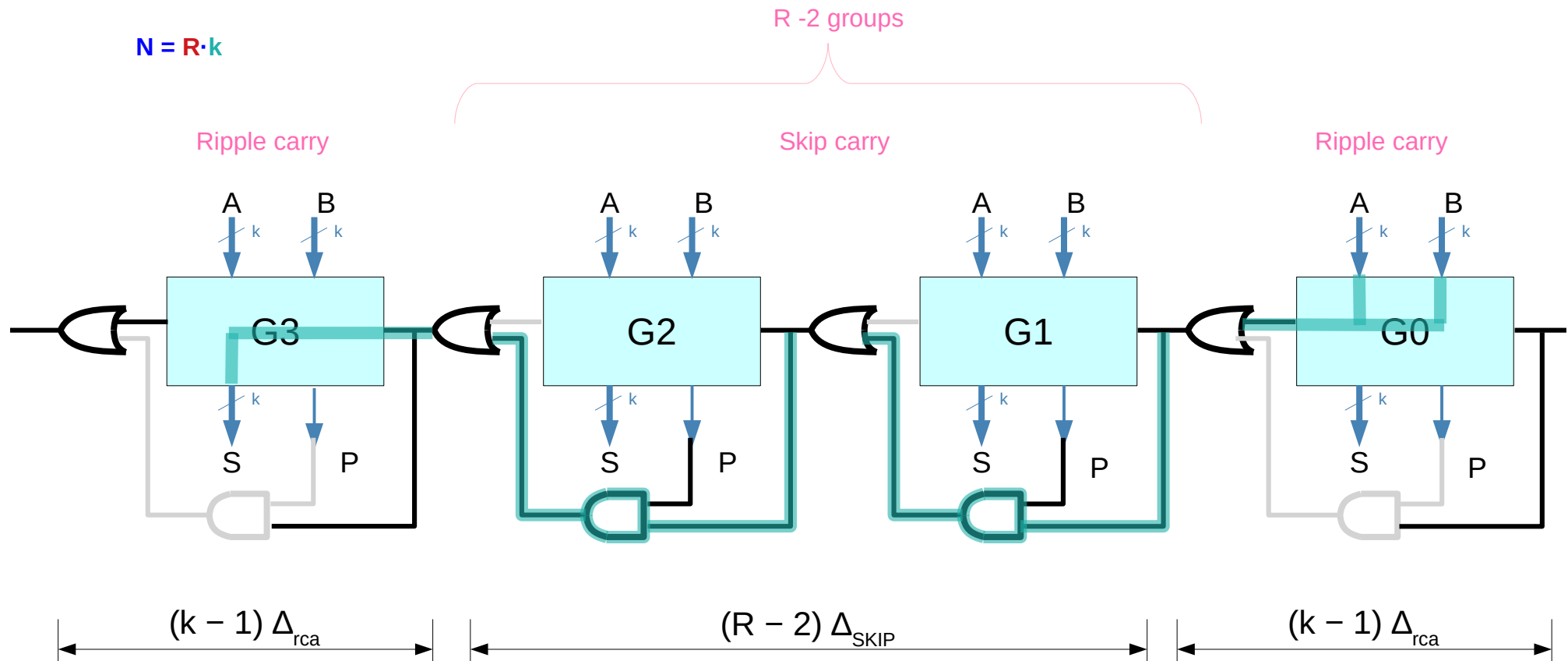
Oklobdzija: High-Speed VLSI arithmetic units : adders and multipliers

# Carry Skip Adder

$$N = R \cdot k$$



# Carry Skip Adder



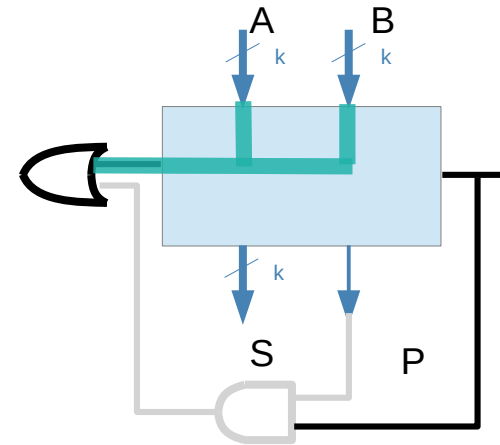
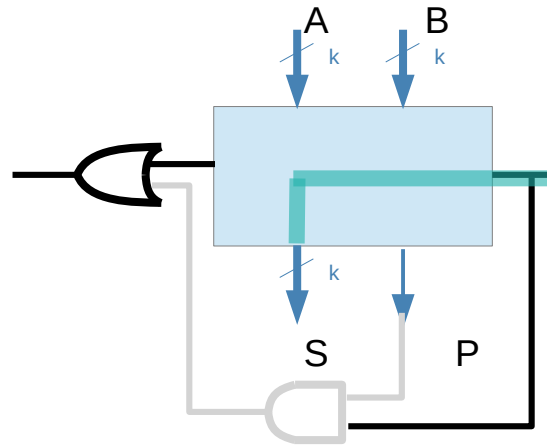
Any kill or generate condition results in divided (broken) critical paths

All FA's in R-2 groups must have the propagate condition

# Carry Skip Adder

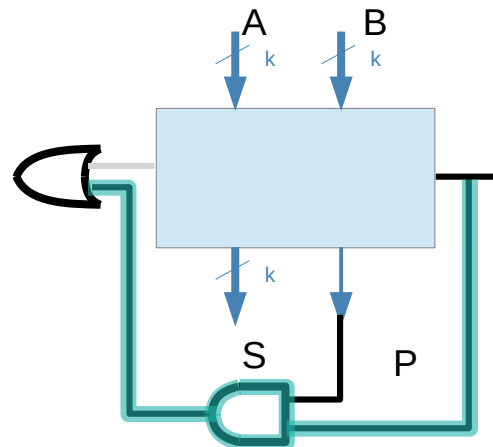
Ripple through  $k-1$  bits

$$(k-1) \Delta_{rca}$$



Skip carry

$$\Delta_{\text{SKIP}}$$



Oklobdzija: High-Speed VLSI arithmetic units : adders and multipliers

# Carry Skip Adder

The maximal delay  $\Delta$  of a Carry Skip Adder is encountered when **carry** is generated in the **least-significant bit** position,

- rippling through  $k-1$  bit positions,
- skipping over  $R-2 = N/k-2$  groups in the middle,
- rippling to the  $k-1$  bits of most significant group and
- being assimilated in the  $N$ -th bit position to produce the sum  $S_N$  :

$$\begin{aligned}\Delta_{\text{CSA}} &= (k - 1) \Delta_{\text{rca}} + (R - 2) \Delta_{\text{SKIP}} + (k - 1) \Delta_{\text{rca}} \\ &= 2 (k - 1) \Delta_{\text{rca}} + (R - 2) \Delta_{\text{SKIP}} \\ &= 2 (k - 1) \Delta_{\text{rca}} + (N/k - 2) \Delta_{\text{SKIP}}\end{aligned}$$

Oklobdzija: High-Speed VLSI arithmetic units : adders and multipliers

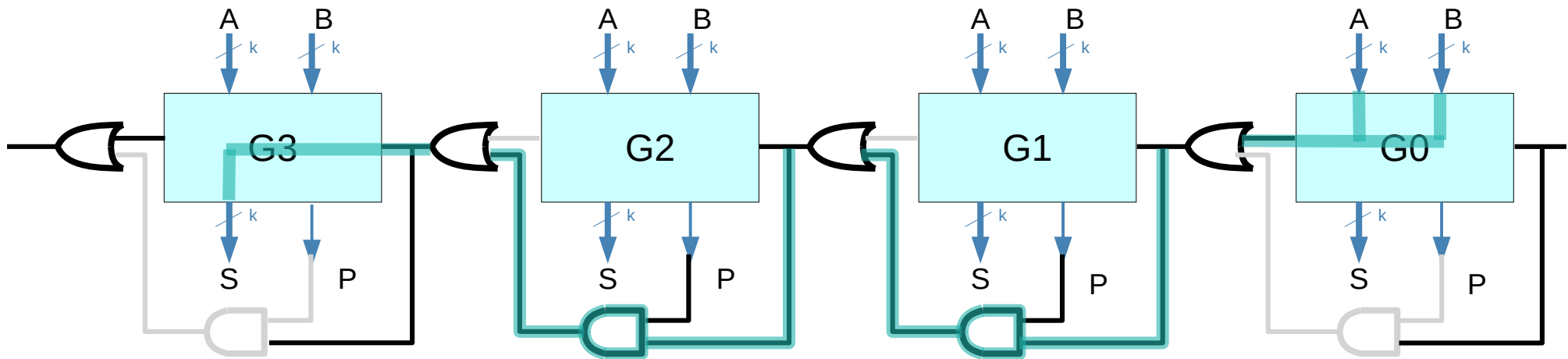
# Carry Skip Adder

$$\begin{aligned}\Delta_{\text{CSA}} &= (k - 1) \Delta_{\text{rca}} + (R - 2) \Delta_{\text{SKIP}} + (k - 1) \Delta_{\text{rca}} \\ &= 2(k - 1) \Delta_{\text{rca}} + (R - 2) \Delta_{\text{SKIP}} \\ &= 2(k - 1) \Delta_{\text{rca}} + (N/k - 2) \Delta_{\text{SKIP}}\end{aligned}$$

Carry Skip Adder is faster than RCA at the expense of a few relatively simple modifications.

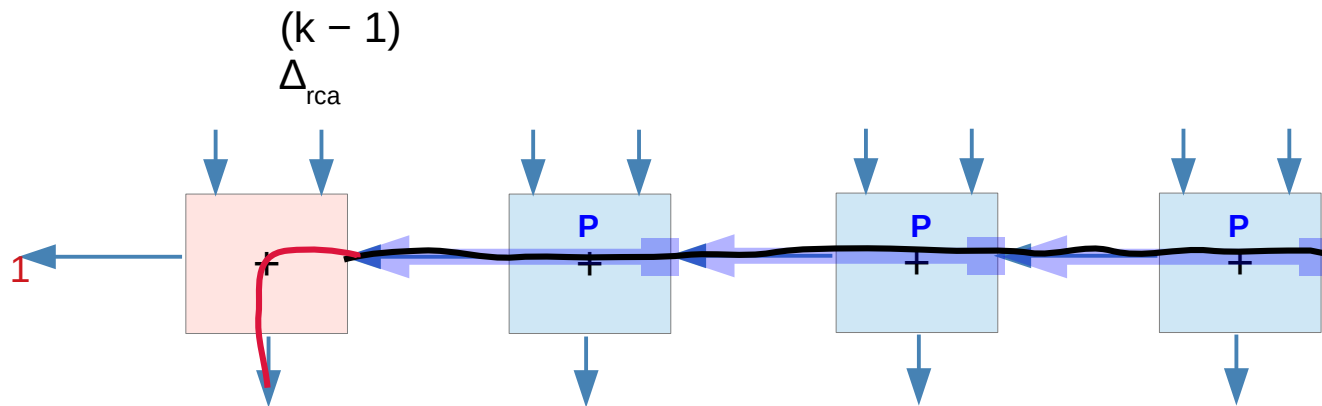
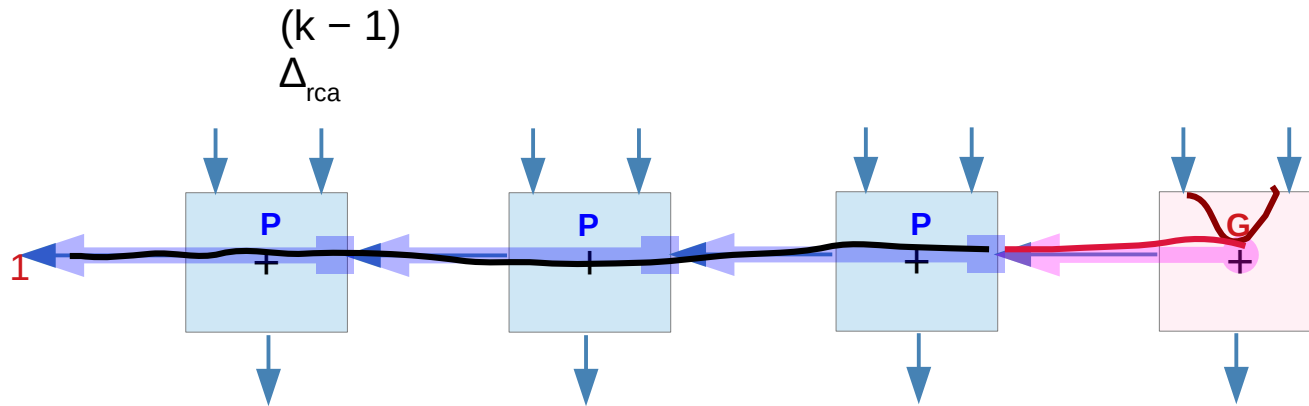
The delay is still linearly dependent on the size of the adder  $N$ , however this linear dependence is reduced by a factor of  $1/k$

$$N = R \cdot k$$



Oklobdzija: High-Speed VLSI arithmetic units : adders and multipliers

# Design C (9) – When Cout1 = 1



High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

# Carry Skip Adder

---

If an arbitrary block generated a carry by itself,  
The carry will always propagate to the next block  
However, if the second block generates a carry itself,  
Or kill the carry, then that is the end of the critical path

If the second block propagates the carry, then we see  
The advantage of the CSA architecture

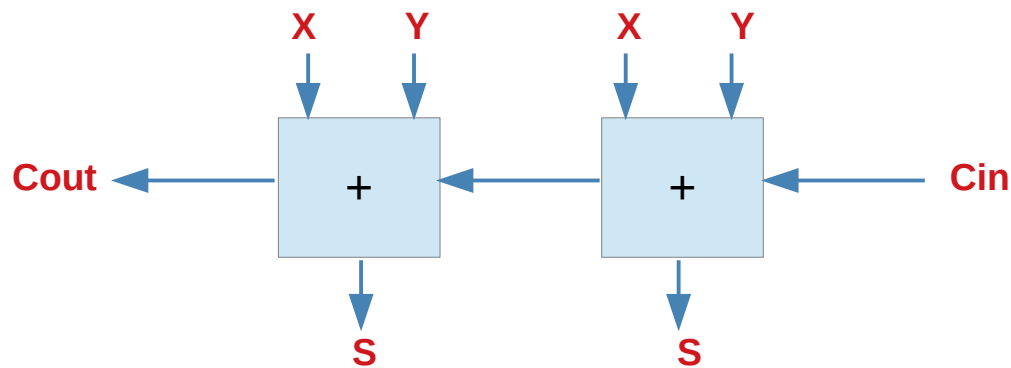
<https://electronics.stackexchange.com/questions/21251/critical-path-for-carry-skip-adder>

<https://electronics.stackexchange.com/questions/21251/critical-path-for-carry-skip-adder>



# Carry Skip Adder

X	Y		
0	0	K	Kill ( $=\bar{P}G$ )
0	1	P	Propagate
1	0	P	Propagate
1	1	G	Generate



Unless the two FA's are in **propagate** mode, the transition of **Cin** does not affect the transition of **Cout**

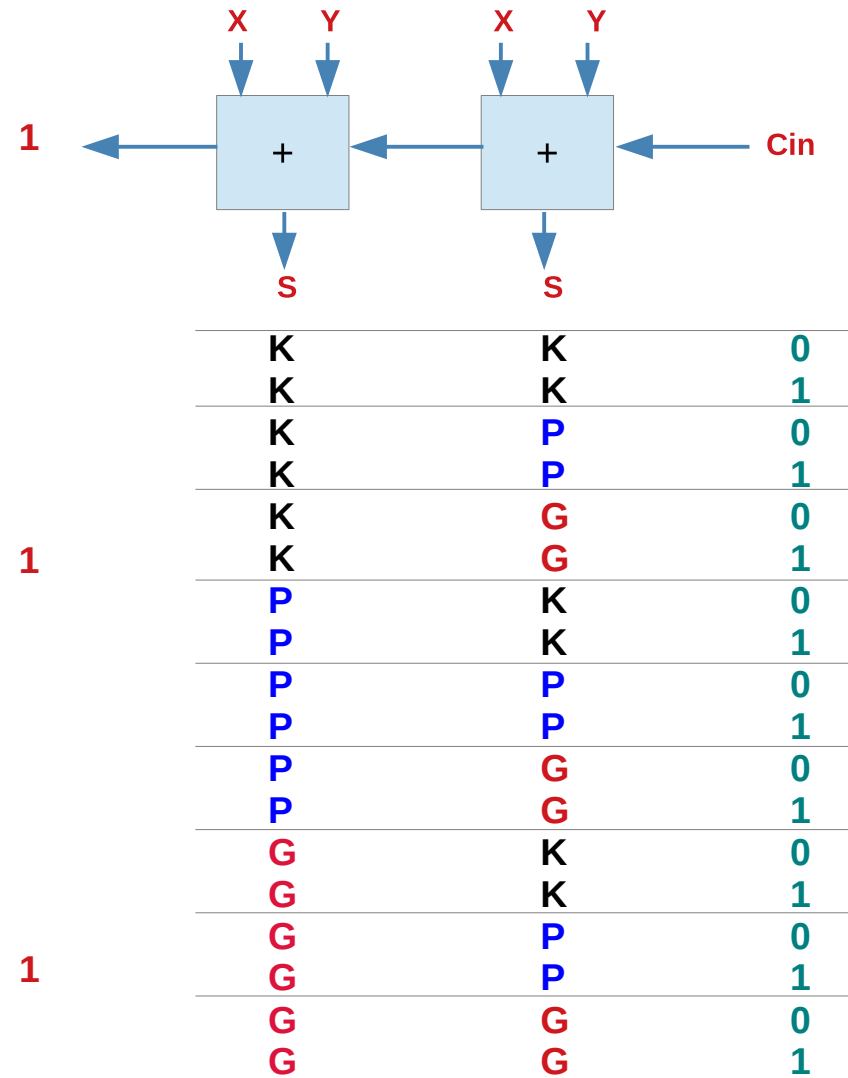
Critical path – all FA's in **propagate** mode

Broken paths for any FA in other mode  
- kill mode, **generate** mode

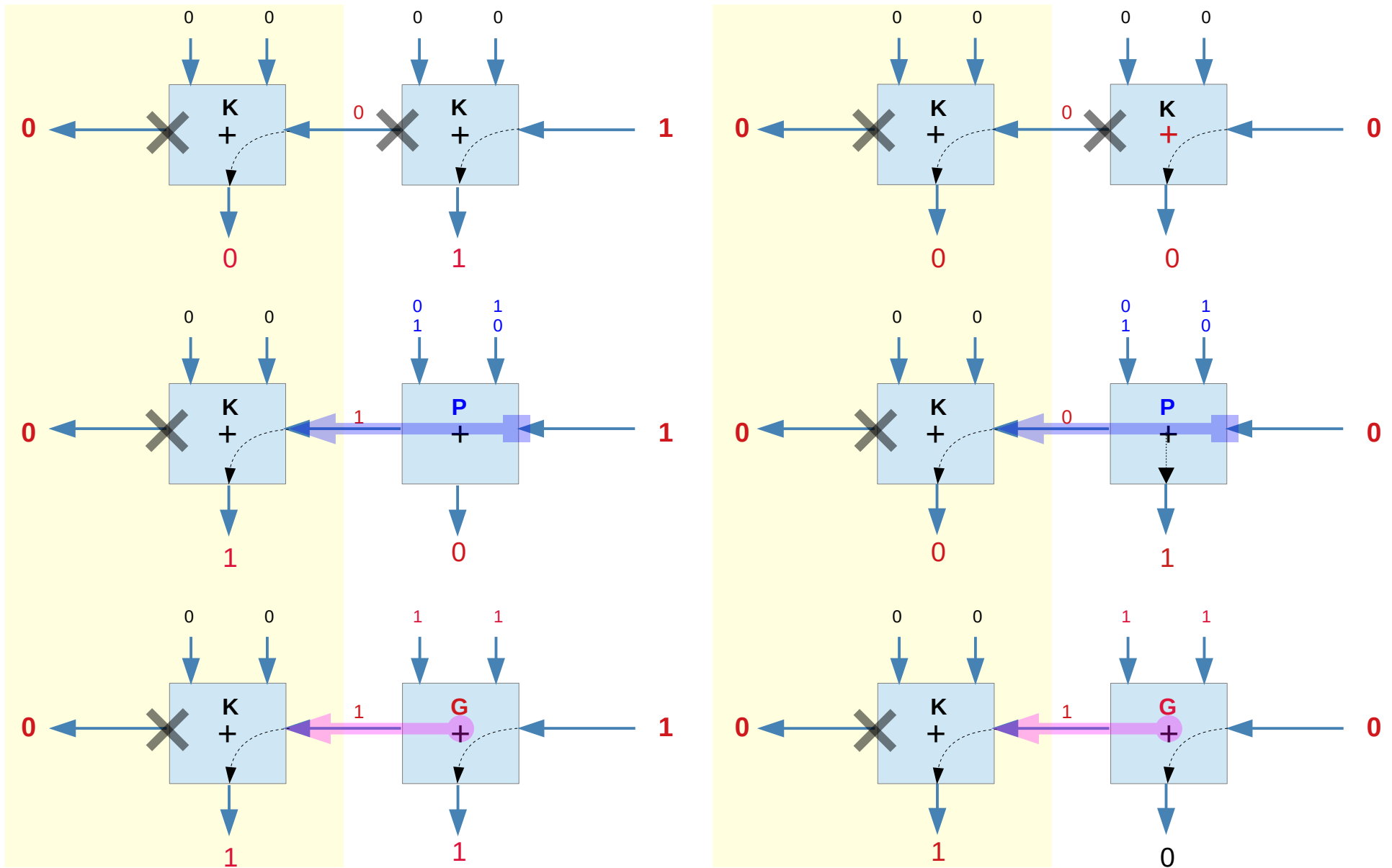
<https://electronics.stackexchange.com/questions/21251/critical-path-for-carry-skip-adder>

# Carry Skip Adder

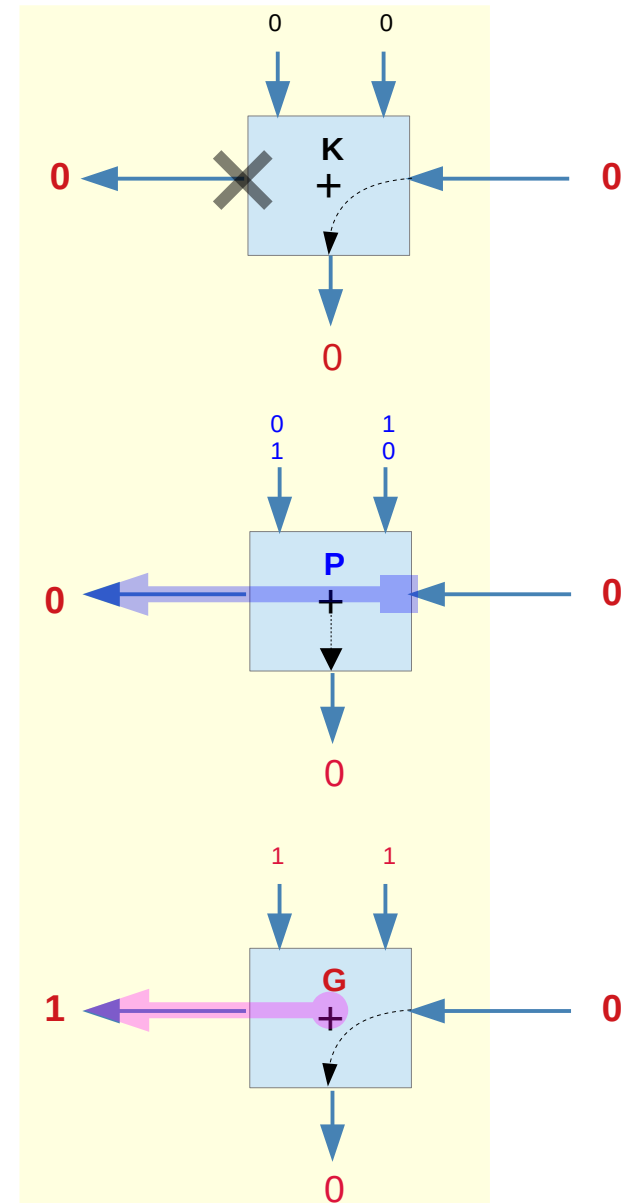
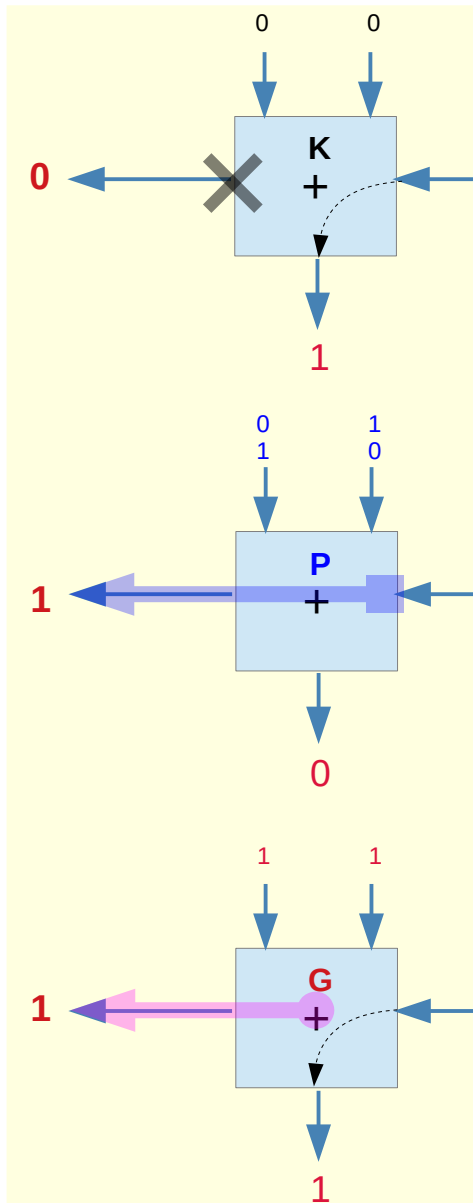
X	Y		
0	0	K	Kill ( $=\bar{P}\bar{G}$ )
0	1	P	Propagate
1	0	P	Propagate
1	1	G	Generate



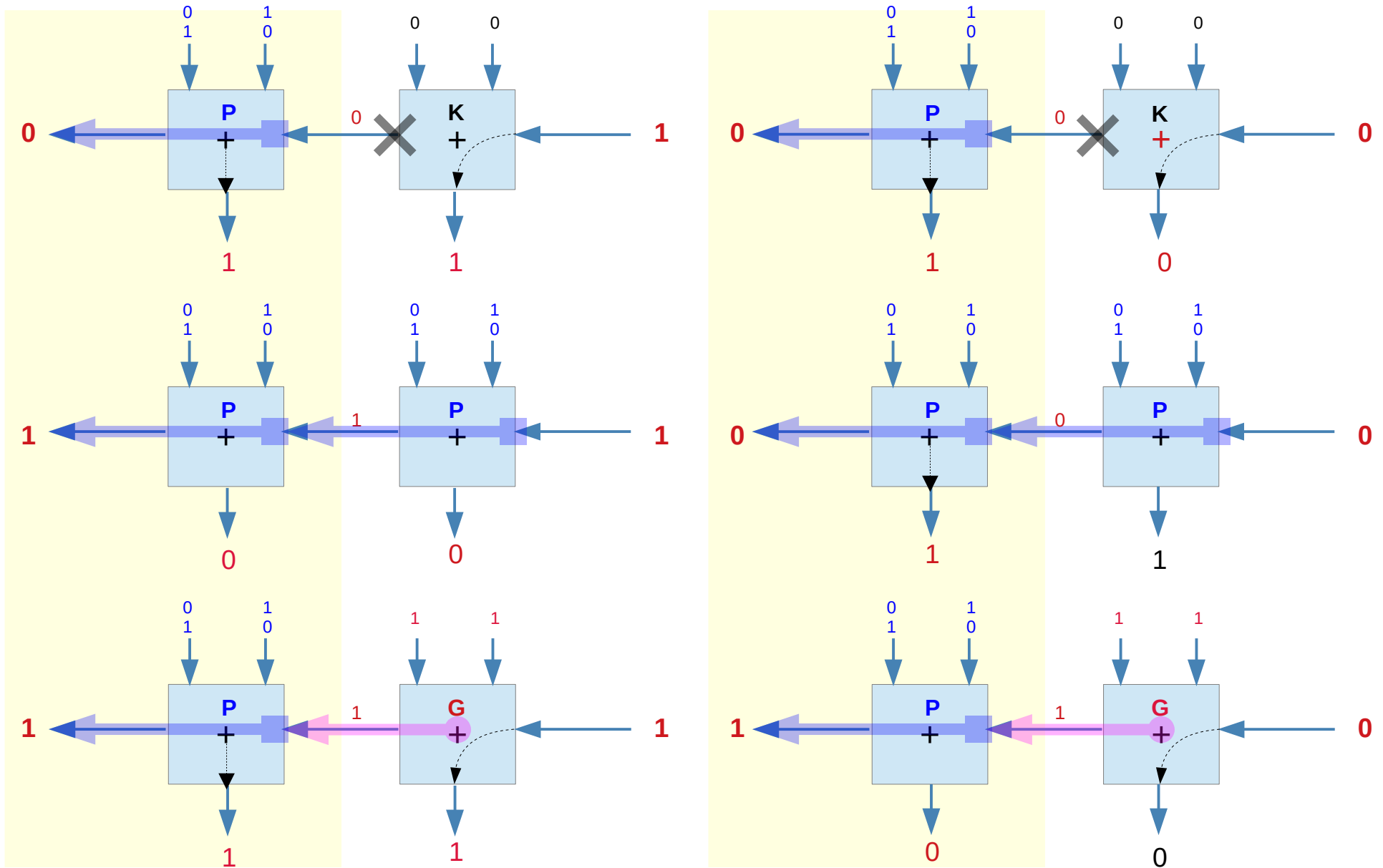
# Cases when FA1 in the Kill mode



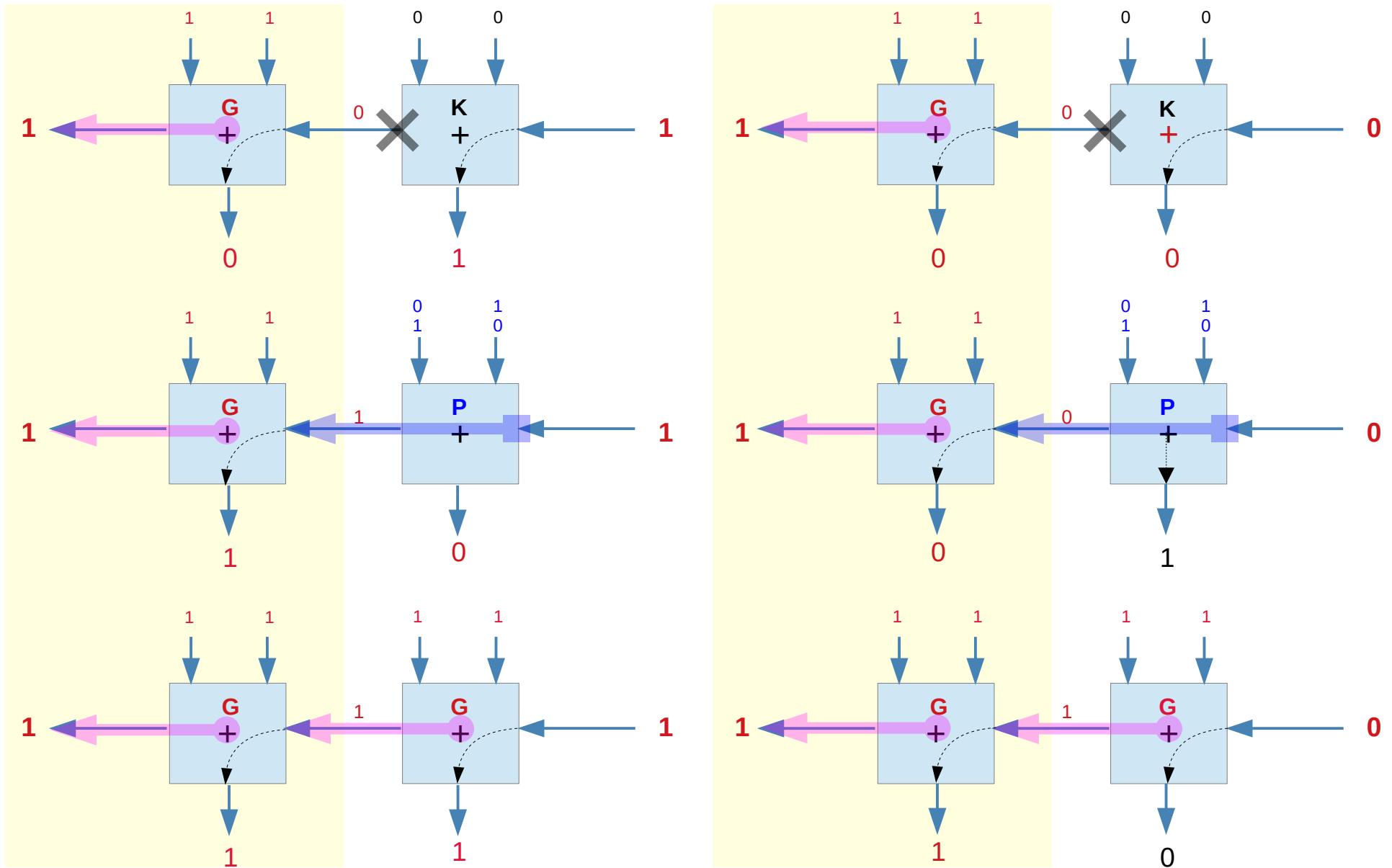
# Cases when FA1 in the Kill mode



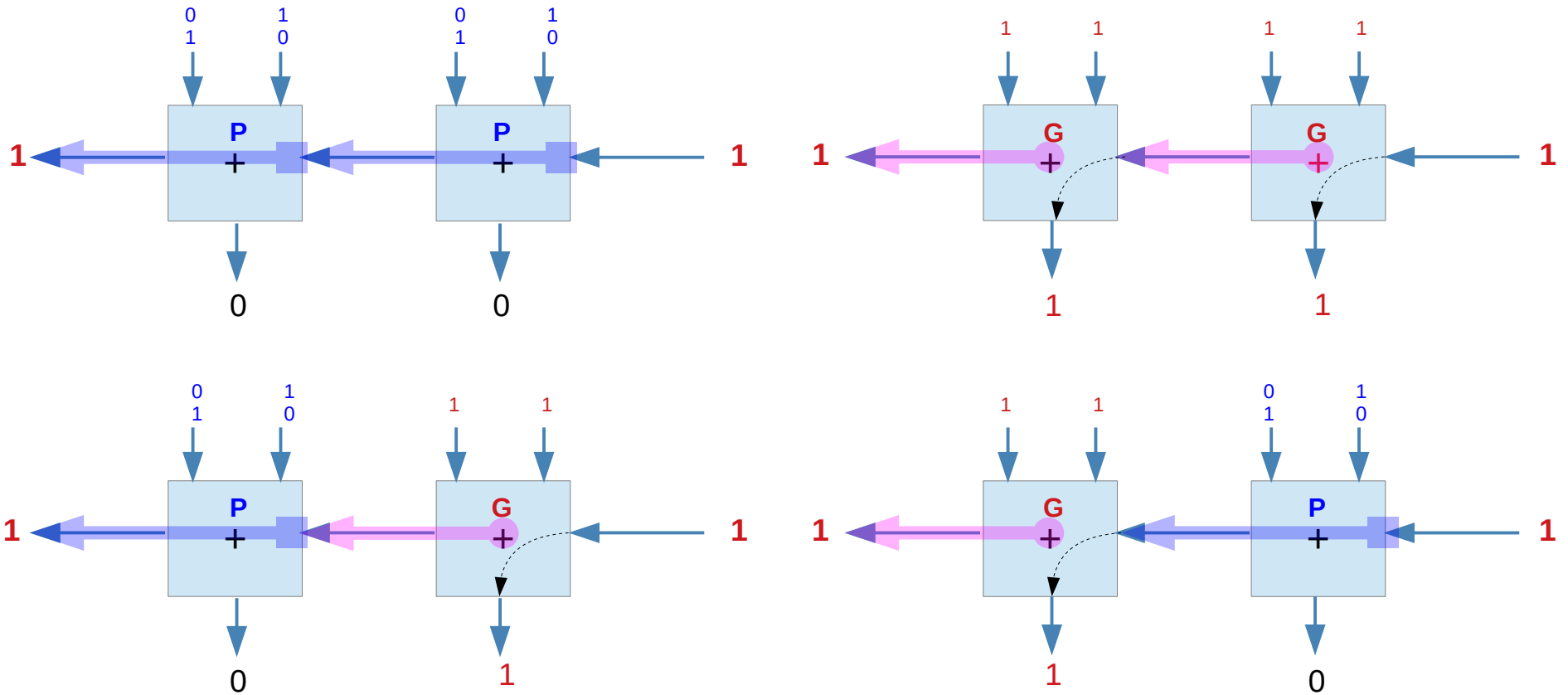
# Cases when FA1 in the Propagate mode



# Cases when FA1 in the Generate mode



# Cases for Cout1 = 1



X	Y		
0	0	K	Kill ( $=\overline{PG}$ )
0	1	P	Propagate
1	0	P	Propagate
1	1	G	Generate





---

## References

- [1] [en.wikipedia.org](https://en.wikipedia.org)
- [2] Parhami, “Computer Arithmetic Algorithms and Hardware Designs”