Application of Feynman-like notation to synthesis of circuits from memristors

Marek Perkowski

November 5, 2012

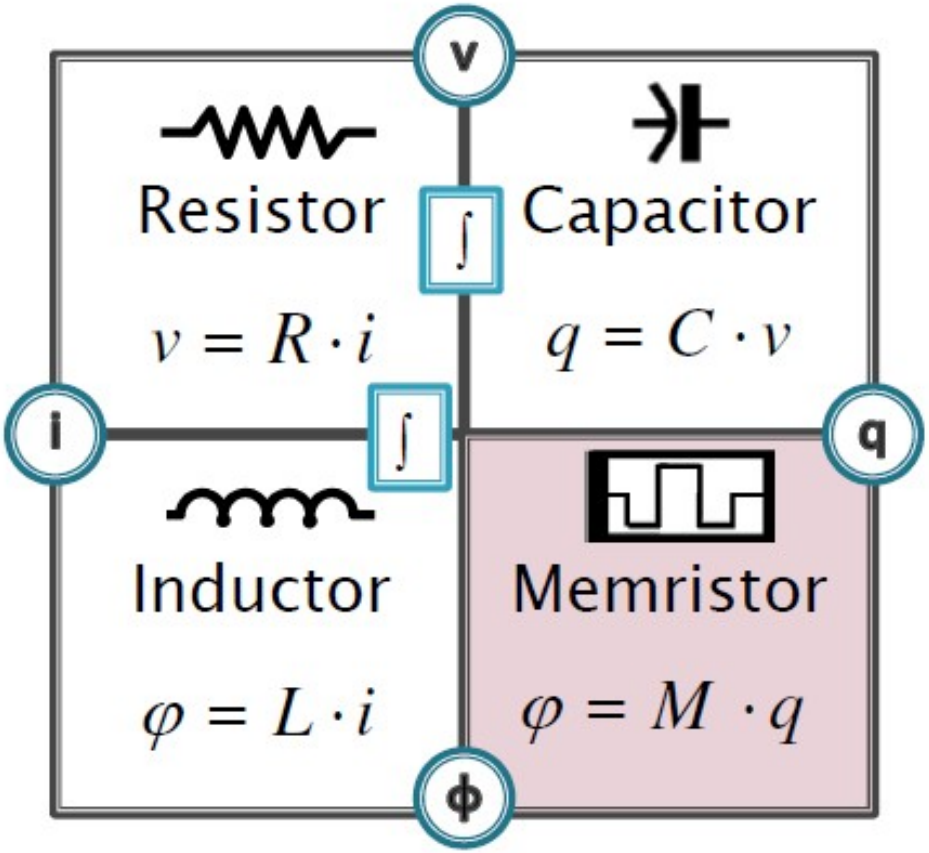
Using the Memristor to build basic logic gates

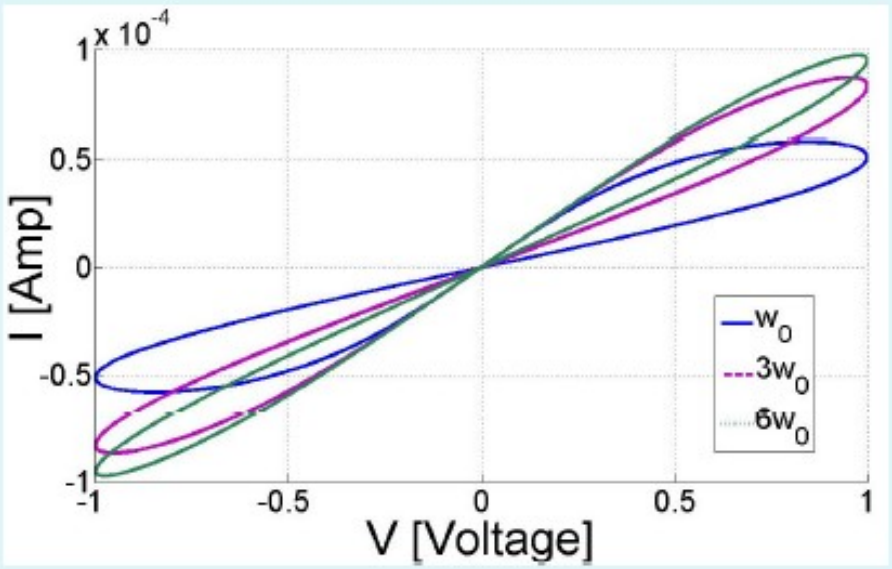
Logic Design with Memristors

1. Wiliams Team at HP
    1. Basic concepts
    2. Demonstration of IMPLY
    3. Demonstration of NAND
2. Israeli Team at Technion - Kvatinsky
    1. Circuit design methodology
    2. Performance robustness tradeoff
    3. Demonstration that the widely used memristor model is not pracrtical
3. Finnish team - Lehtonen
    1. Show theory with minimum ancilla bits
    2. No algorithm
    3. No results
    4. Restriction of assumption
4. Our work at PSU
    1. Practical model
    2. Practical assumptions
    3. More general
       Optimal algorithm

Leon Chua Historic Discovery

- Leon Chua, The Missing Circuit Element, IEEE





$$v = M(x,i) \cdot i$$

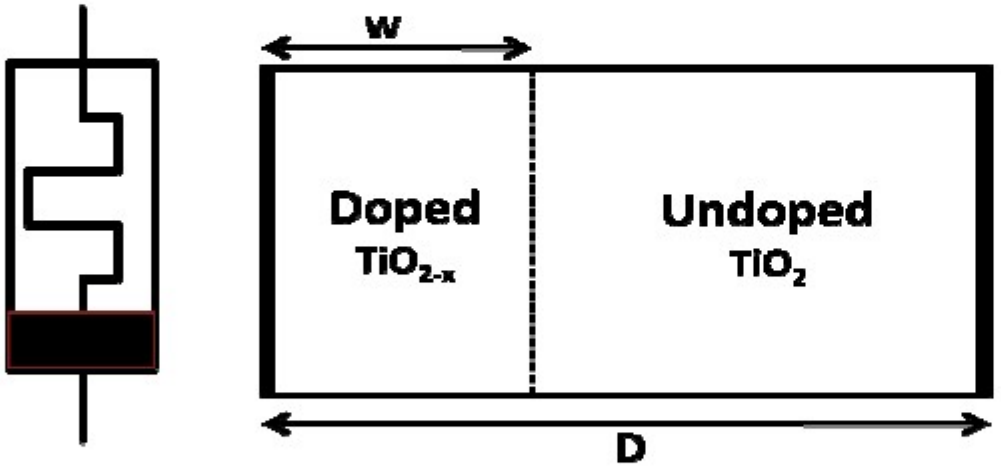
$$\frac{dx}{dt} = f(x,i)$$

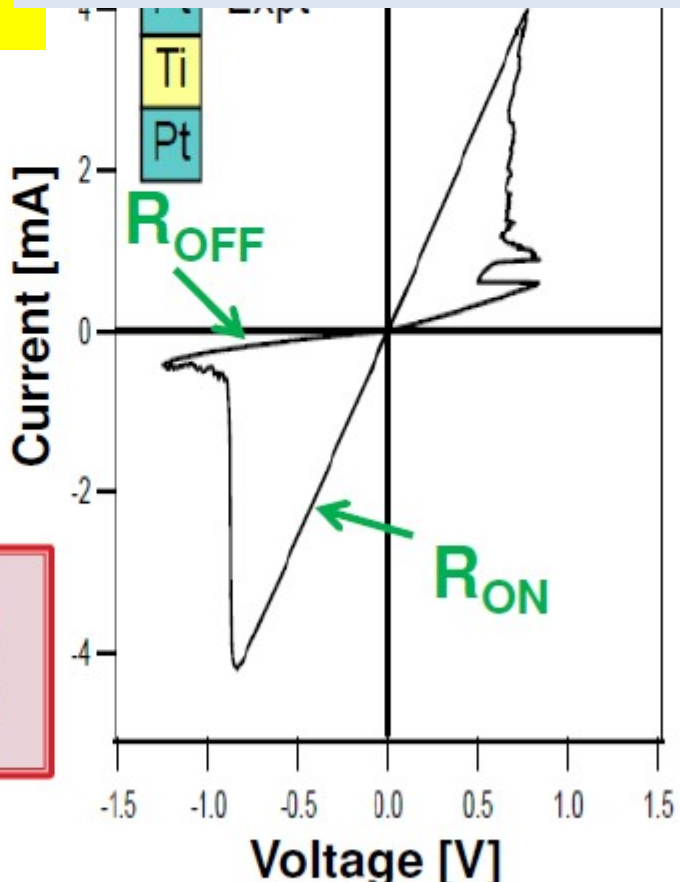
Invention based on search for general principles of physics and Nature = he referst to Aristotle etc

Memristors becoming practical

- Hewlett Packard 2008
- D Strukov et al The Missing Memristor found, Nature 2008

R off = high resistance
R ON = low resistance

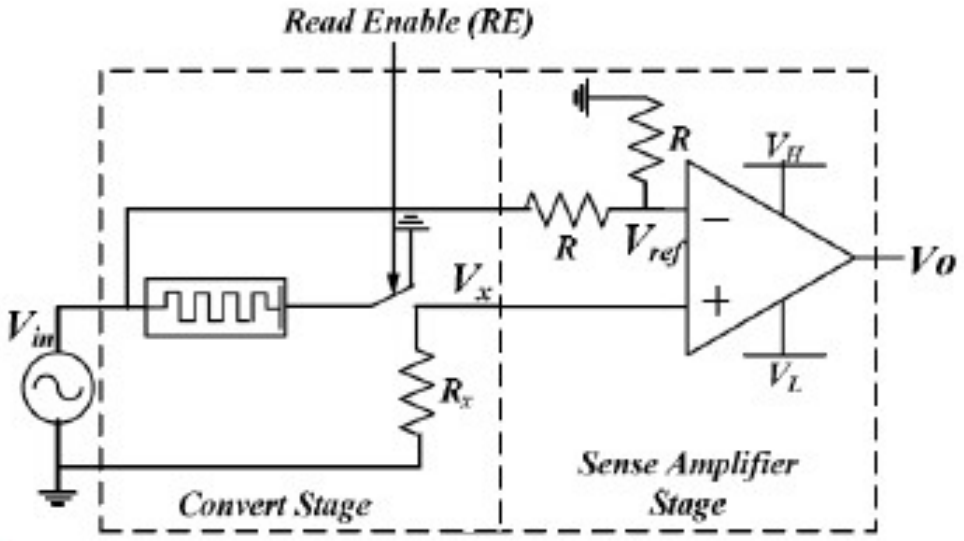




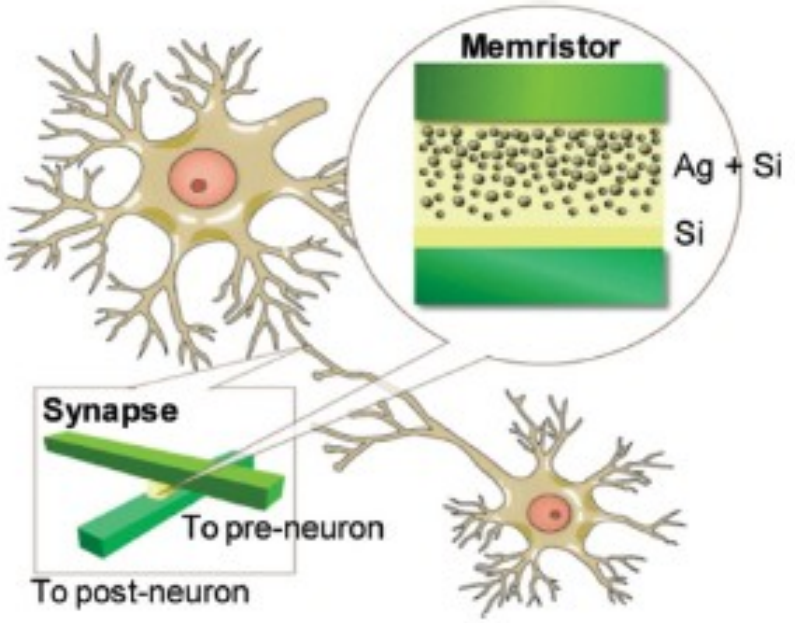
$$M(q) = R_{OFF}\left(1 - \frac{\mu_v R_{ON}}{D^2} q(t)\right)$$

Very simplified model of

So far, most of the applications of memristor are in analog design



1. Dr. Teuscher PSU – evolving CAM circuits
2. Memory Digital
3. Memory analog
4. Fuzzy Circuits
5. Analog Circuit
6. Neuromorphic Systems



Y. Ho et al, Nonvolatile Memristor Memory, Device Characteristics and Design Implications, ICCAD, 2009

A. Afifi et al, Implementation of Biologically Plausible Spiking Neural Network Models on the Memristor Crossbar-based CMOS/Nano Circuits.ECCTD 2009

We can do standard binary logic with memristors!

1. Logic values represented as resistances.
2. RON = logic 1, ROFF = logic 0.
3. Circuit uses primarily resistors

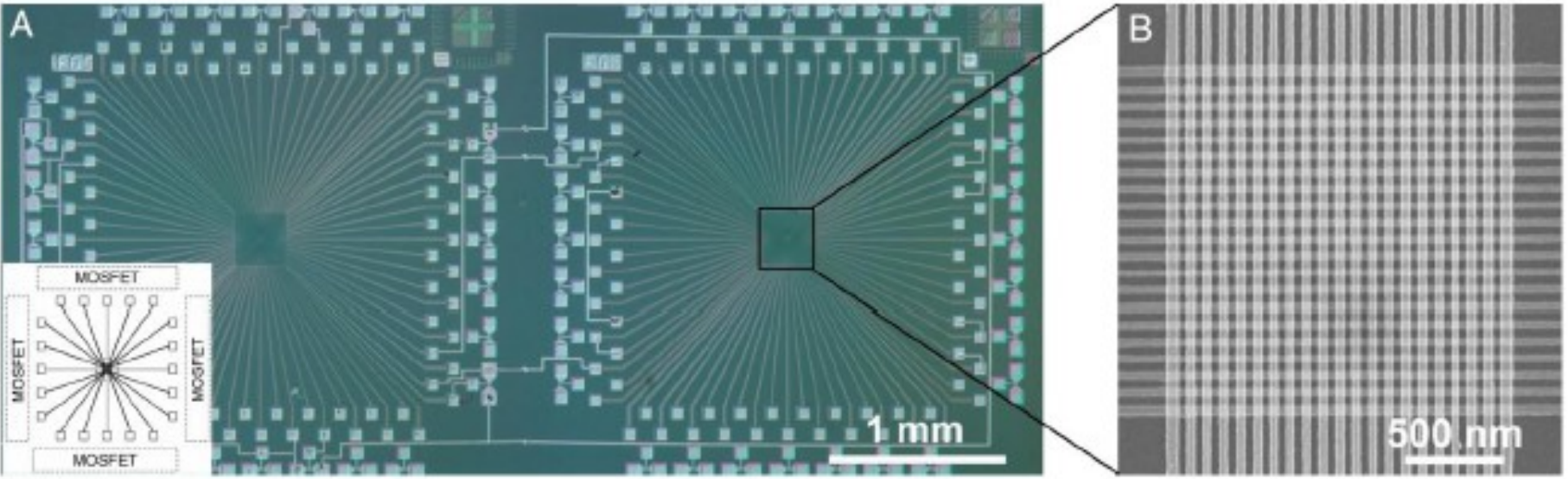
4. Memristor can be used as:
    1. Input storage/processing
    2. Output storage/processing
    3. Computational logic element
    4. Latch or memory circuit

Computing with Memristive Devices

1. Memristive Devices allow for reconfigurable logic
2. Will allow for reconfigurable analog circuis
3. They allow high density logic
4. They will be used in high density DSP and image processing, Neural, etc. applications.
5. Memristive devices will change computing paradigm.
    1. CPU embedded memory
    2. Reboot-less power down at any time
    3. Extending Moore's Law beyond CMOS Limit.

- J Borghetti et al PNAS, Vol 106, No 6, 2009.

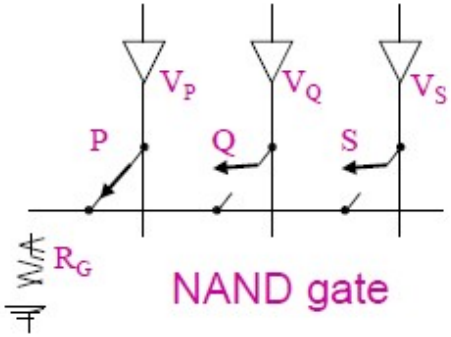
Islands of Memristors for parallel calculations, SIMD mode inside normal MOSFET logic



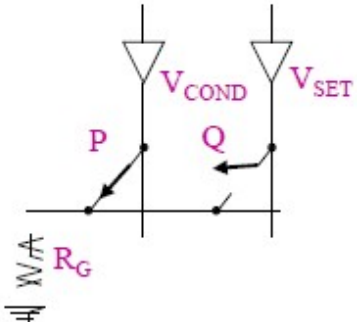
Memristive Stateful Logic Gate

1. This gate realizes material implication p ⊏ q
2. Device q can be SET conditionally depending on the status of the device p.
3. “Stateful Logic”, functioning complete NAND and status storing in device s

Needs two pulses



NAND gate



p	q	q'
0	0	1
0	1	1
1	0	0
1	1	1

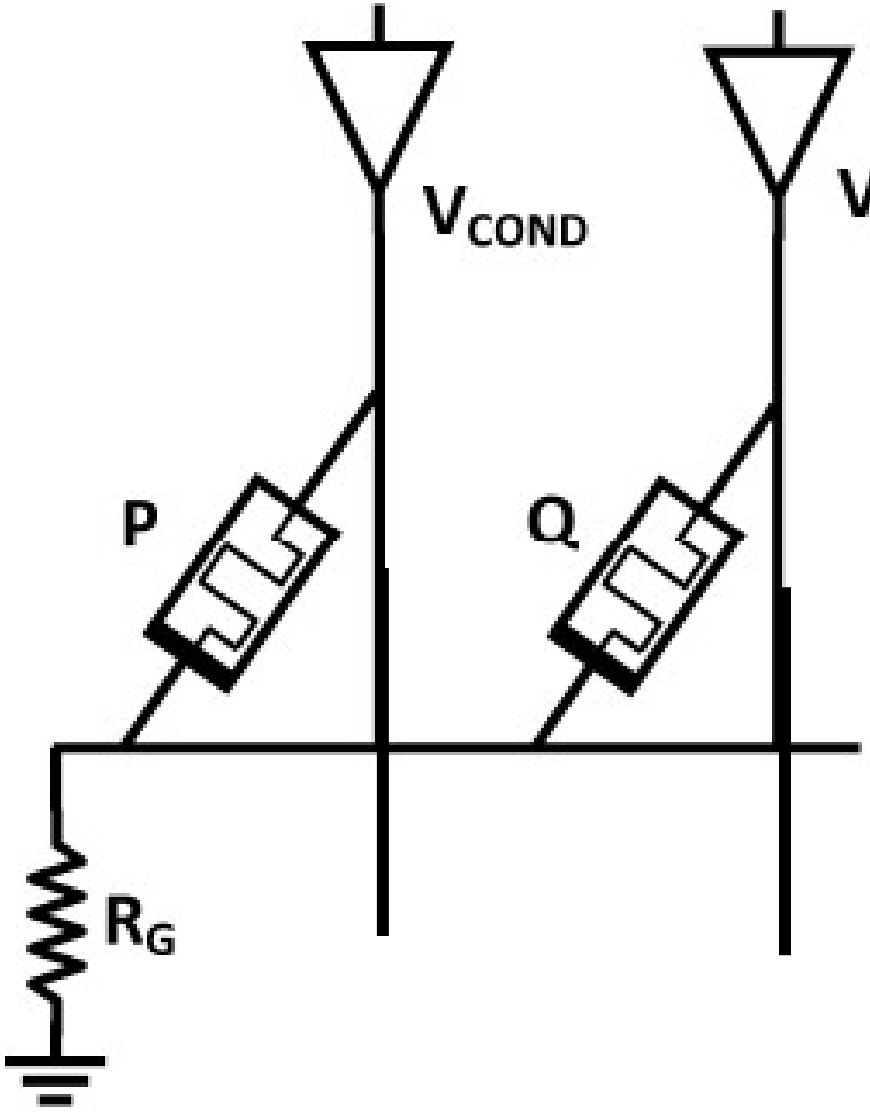
	V_P	V_Q	V_S	$s,\ s',\ s''$
Step-1			V_{CLEAR}	0
Step-2	V_{COND}		V_{SET}	IMP(p, s)
Step-3		V_{COND}	V_{SET}	IMP(q, s′)

Needs three

- J Burghes et al, Nature Vol 404, 2010

Memristive Implication Logic

- DR Stewart, GS Snider, PJ Kuekes, JJ Yang, DR Stewart, RS Williams, Nature 464, 873 (2010)

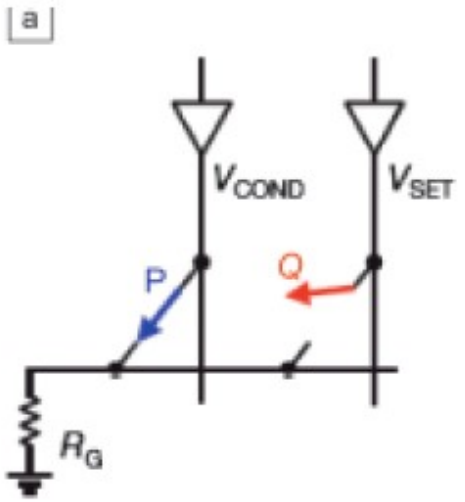


Needs two pulses for executing IMPLY

Figure 1. Schematic of a memristor-based IMPLY gate. Two memristors P and Q are connected to a resistor R_G. The logic state of the memristors P and Q are, respectively, p and q.

Memristive Implication

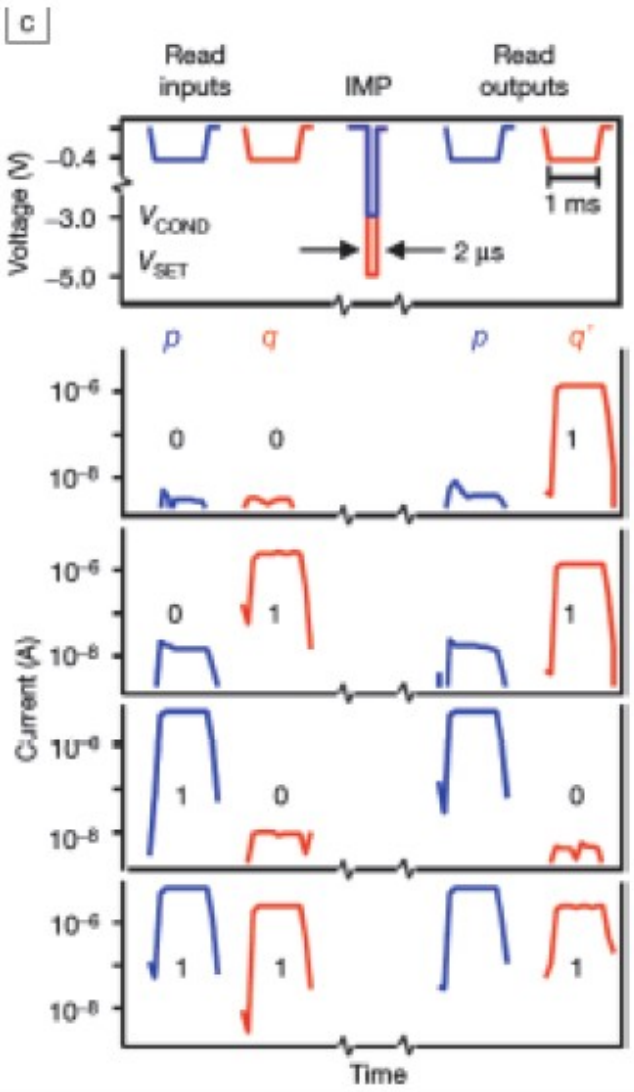
- DR Stewart, GS Snider, PJ Kuekes, JJ Yang, DR Stewart, RS Williams, Nature 464, 873 (2010)



b

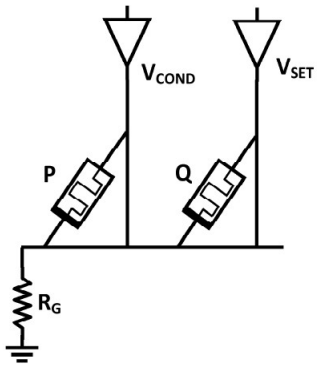
$q' \leftarrow p\text{IMP}q$

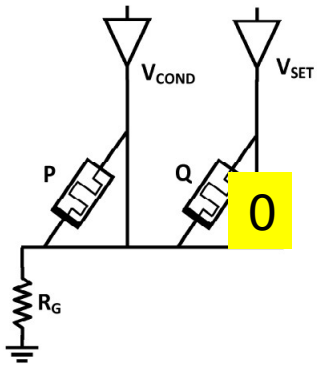
In P	In q	Out q'
0	0	1
0	1	1
1	0	0
1	1	1



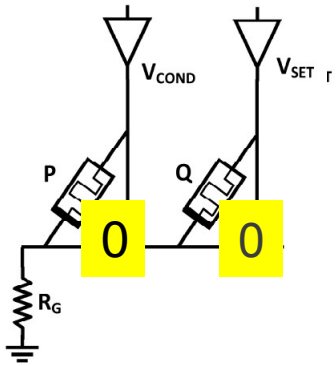
Needs two pulses

Set to 0

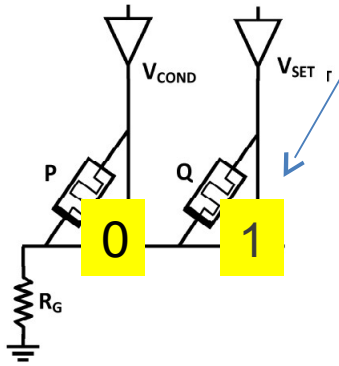




P⊟ q

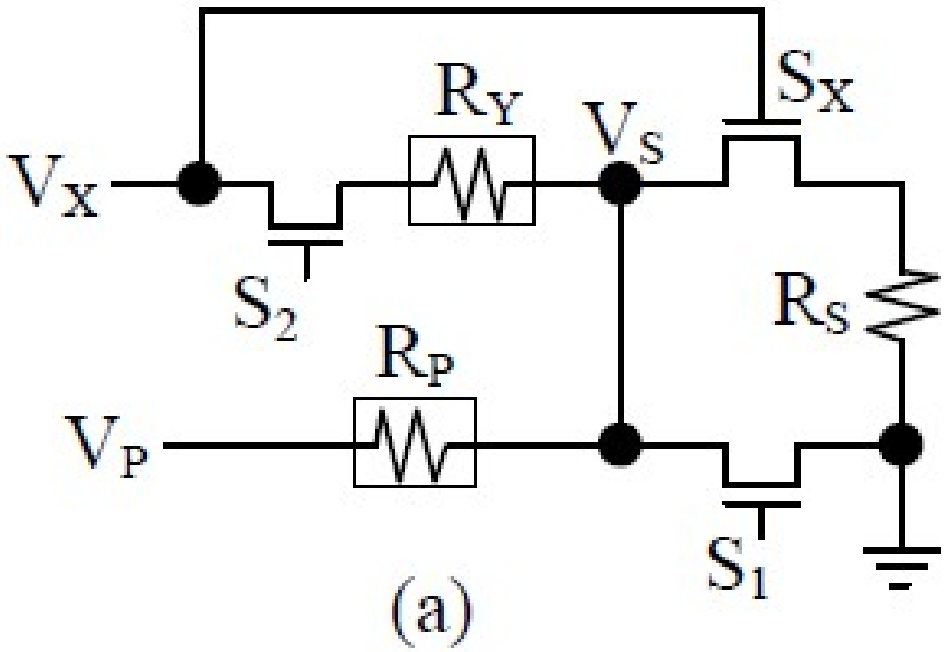


New valu e



EXOR realized with memristors

Memristor Based XOR gate



(a)

(b)

	V_P	S_1	S_2
Step-I	V_{RST}	1	0
Step-II	V_{SET}	0	1

(c)

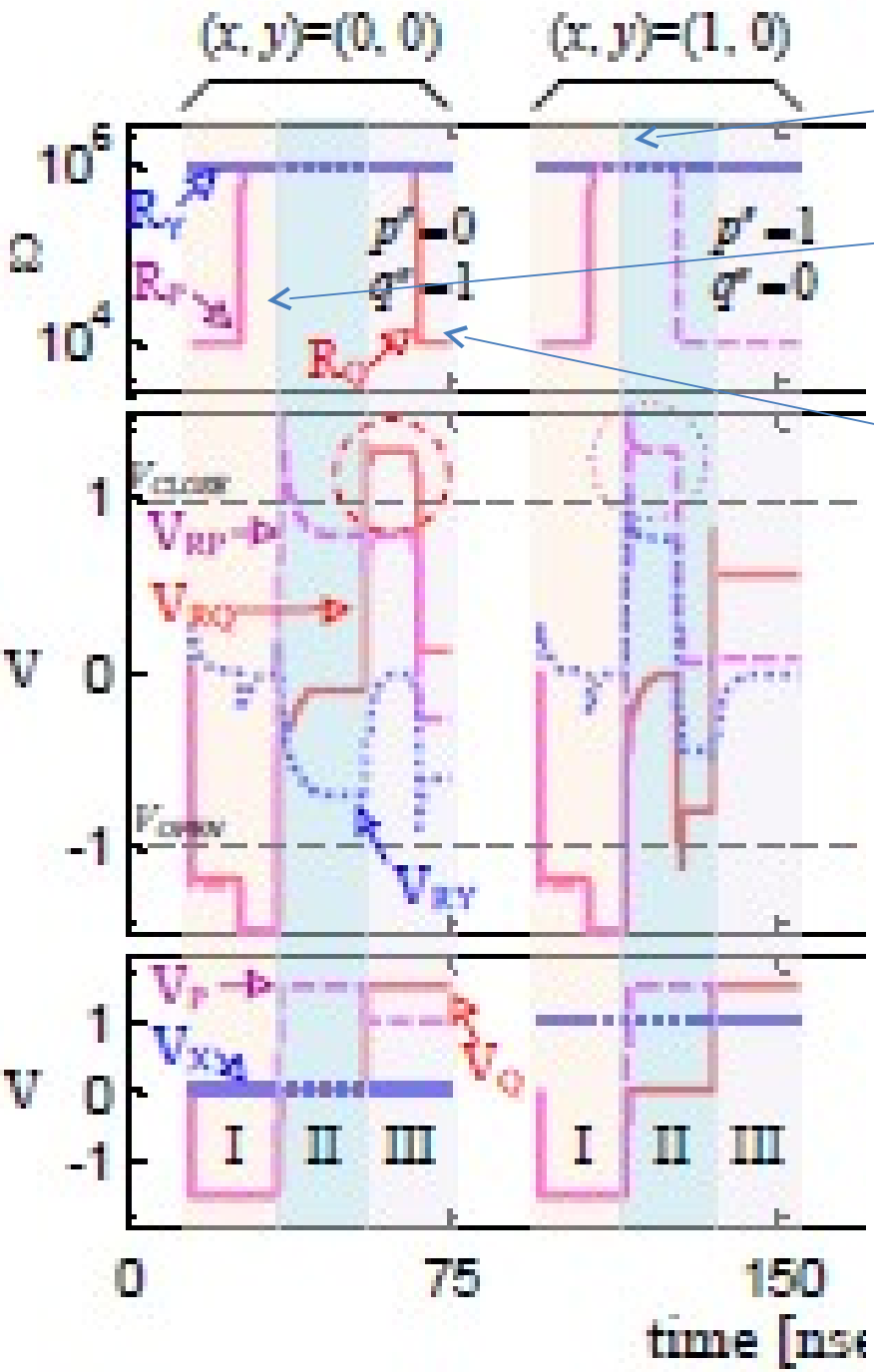
x	y	p	p'	Mode
0	0	0	0	$p' = y$
0	1	0	1	
1	0	0	1	$p' = \bar{y}$
1	1	0	0	

output

Input and output data are in memristors

- **First create VRST by S1=1, S2 =0**
- **Next create VSET by S1=0**

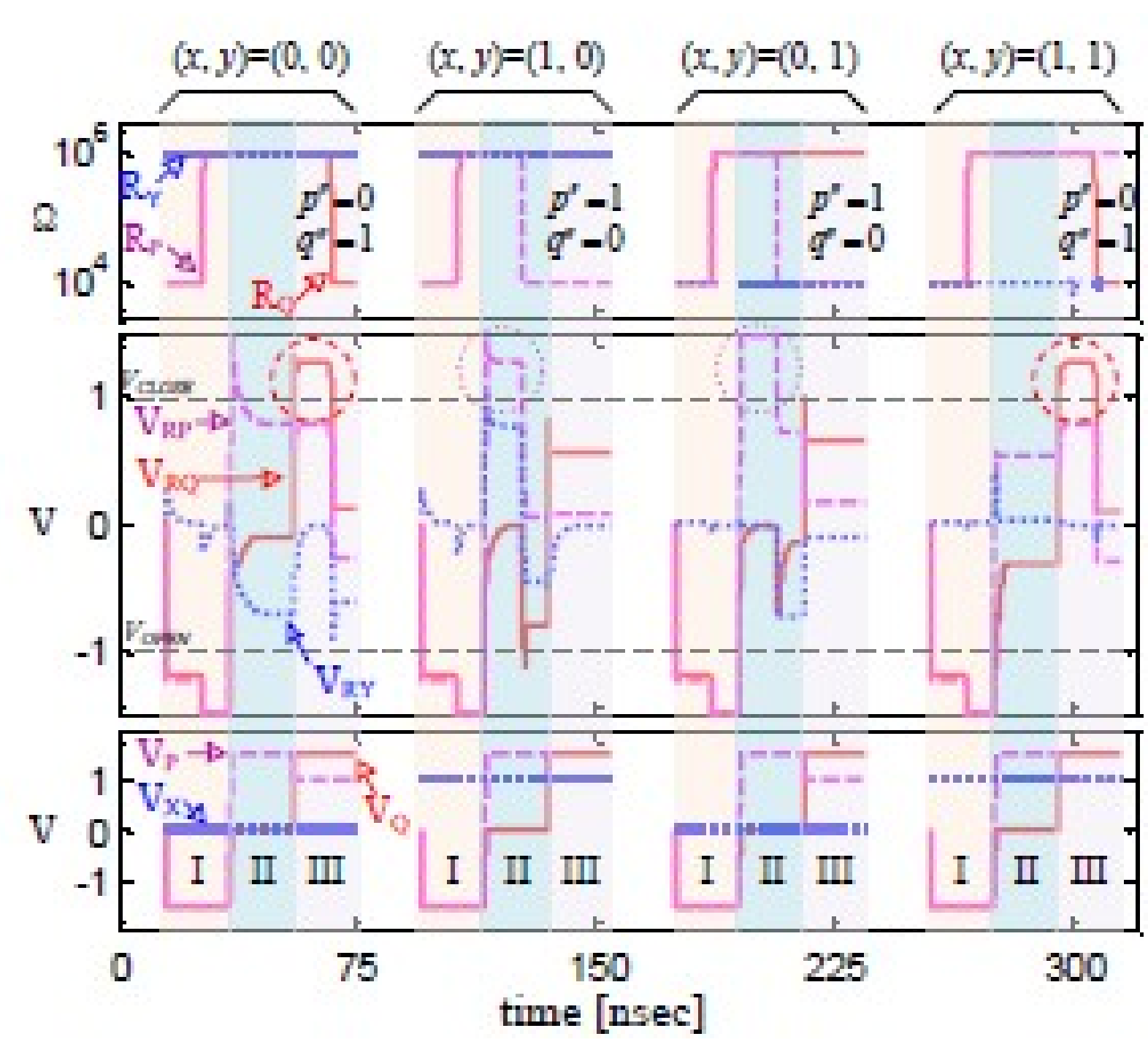
- 2 memristors
- 3 FETs
- 1 resistor
- 2 steps


(x, y)=(0, 0)
(x, y)=(1, 0)
10^6
10^4
Ω
R_Y
R_P
R_Q
p' =0
q' =1
p' =1
q' =0
1
V 0
-1
V_{CLOCK}
V_{RP}
V_{RQ}
V_{COMMON}
V_{RY}
V_P
V_{XY}
V_Q
I
II
III
I
II
III
0
75
150
time [ns

R
Y

R
P

R
Q


(x, y)=(0, 0)
(x, y)=(1, 0)
(x, y)=(0, 1)
(x, y)=(1, 1)
Ω
p' =0
q' =1
p' =1
q' =0
p' =1
q' =0
p' =0
q' =1
V
V
I
II
III
0
75
150
225
300
time [nsec]

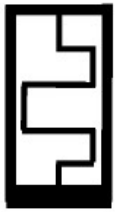


Figure 1. Memristor symbol.

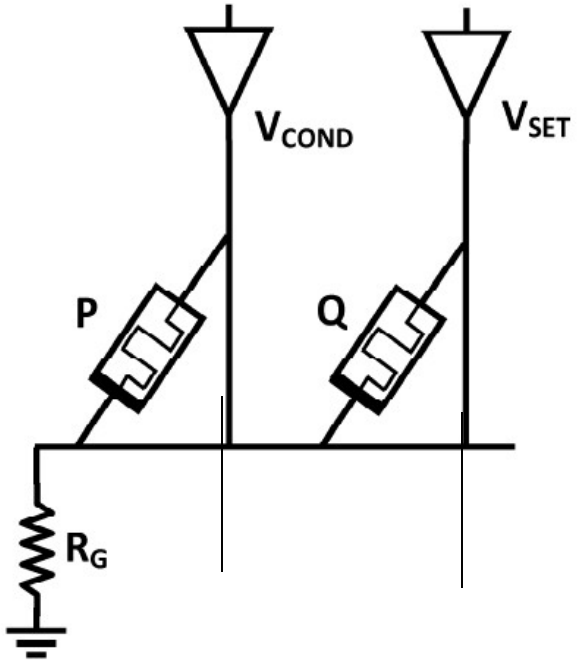


Figure 2. Memristor-based IMPLY logic gate.

- In this approach, the logic state is represented as resistance, where a high resistance is logical state zero, and a low resistance is logical state one.
- An example of this approach, the IMPLY logic gate, is shown in Figure 2.

Using the Memristor to build basic logic gates

- NOT
- OR
- AND
- NAND
- NOR
- EXOR
- MUX

- Memristor

A	B	X
0	0	1
0	1	1
1	0	0
1	1	1

Costs of gates with

	Realized with memristors	Numbers of levels	
IMPLY	**1**	**1**	**+**
AND	**3**	**3**	
OR	**2**	**2**	**+**
NAND	**2**	**2**	**+**
NOR	**3**	**3**	
EXOR	**4**	**3**	
XNOR			
INHIBIT	**2**	**2**	**+**
NOT	**1**	**1**	**+**
MUX	**5**	**3**	

CHANGE

Good for TANT, Negative gate methods

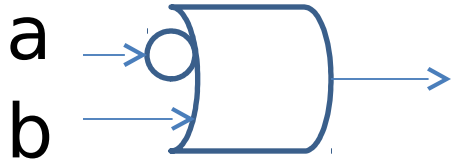
examples

- ((abc)’+ (ab))’ + (bcd) =
- ((a’ + b’ + c’) + ab)’ + bcd =
- (abc) * (ab)’ + bcd =
- (abc) * (a’ + b’) + bcd =
- (b’c) + bcd

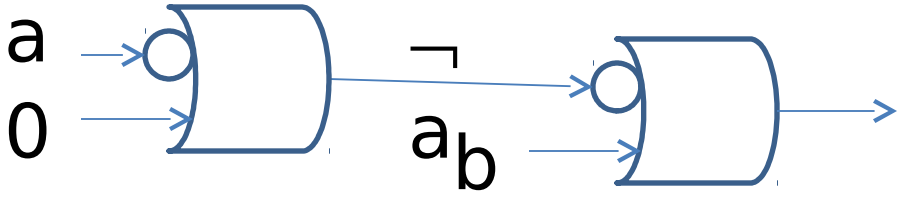
example

- ((ab) ⨽ 0) = (a’+b’)
- ((ab) ⨽ (ab)) = (a’+b’) + ab
- (0 ⨽ ab) = (ab)
- (a ⨽ b) = (a’+b)

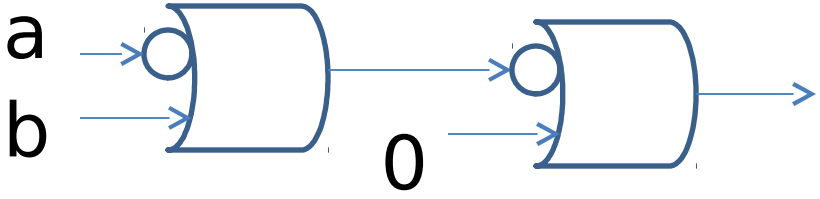
All these circuits assume that value of b already exists.
If it does not exist, we need two inverters (from IMPLY) to create it.



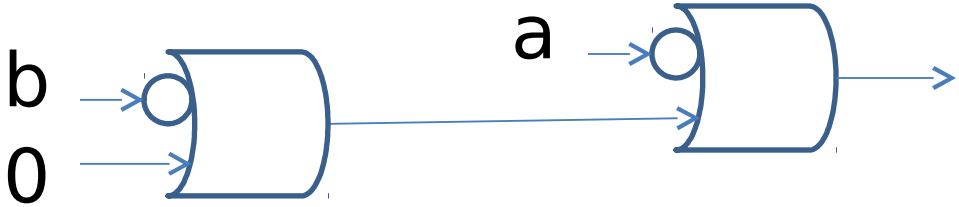
$\neg a + b$



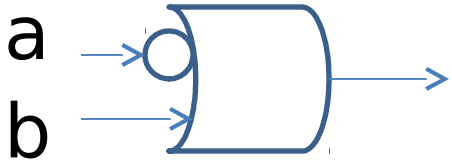
$a + b$



$\neg (\neg a + b) = a * \neg b$

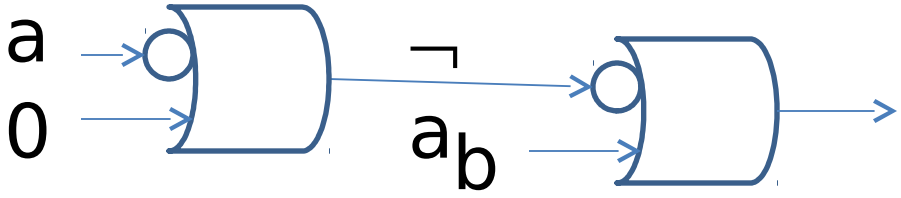


$\neg a + \neg b = \neg(a * b)$

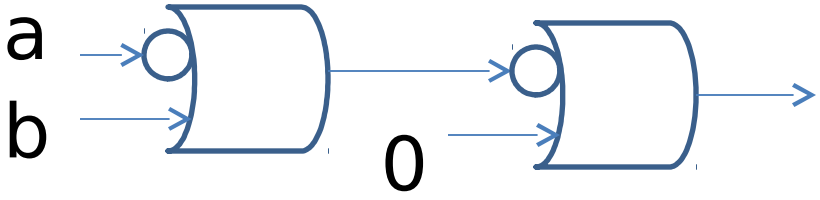


$\neg a + b$

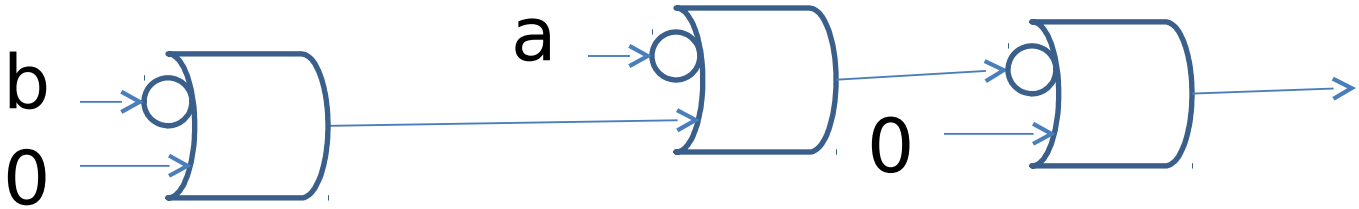
All these circuits assume that value of b already exists.
If it does not exist, we need two inverters (from IMPLY) to create it.



$a + b$



$\neg (\neg a + b) = a * \neg b$



$\neg \neg(a * b)$

Now we assume that all inputs must be created with Stateful IMPLY technology from scratch

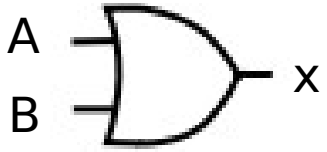
NOT

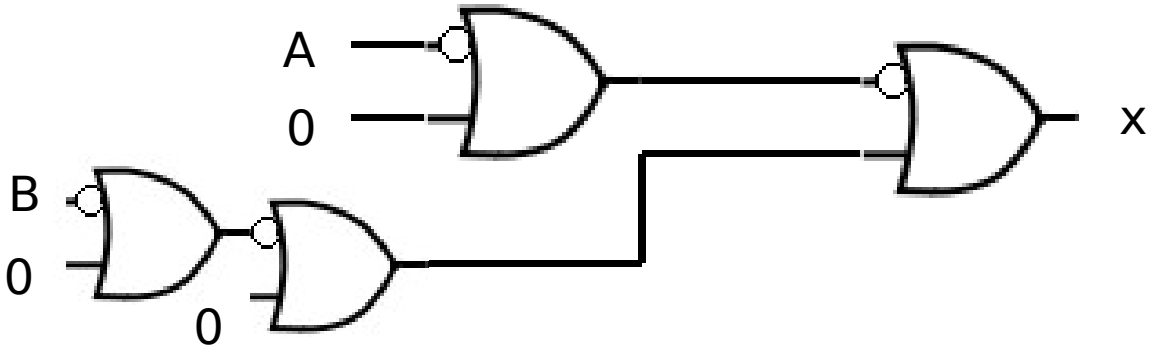
& OR

NOT & OR

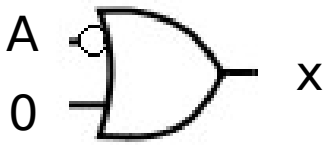
NOT

OR with two inputs



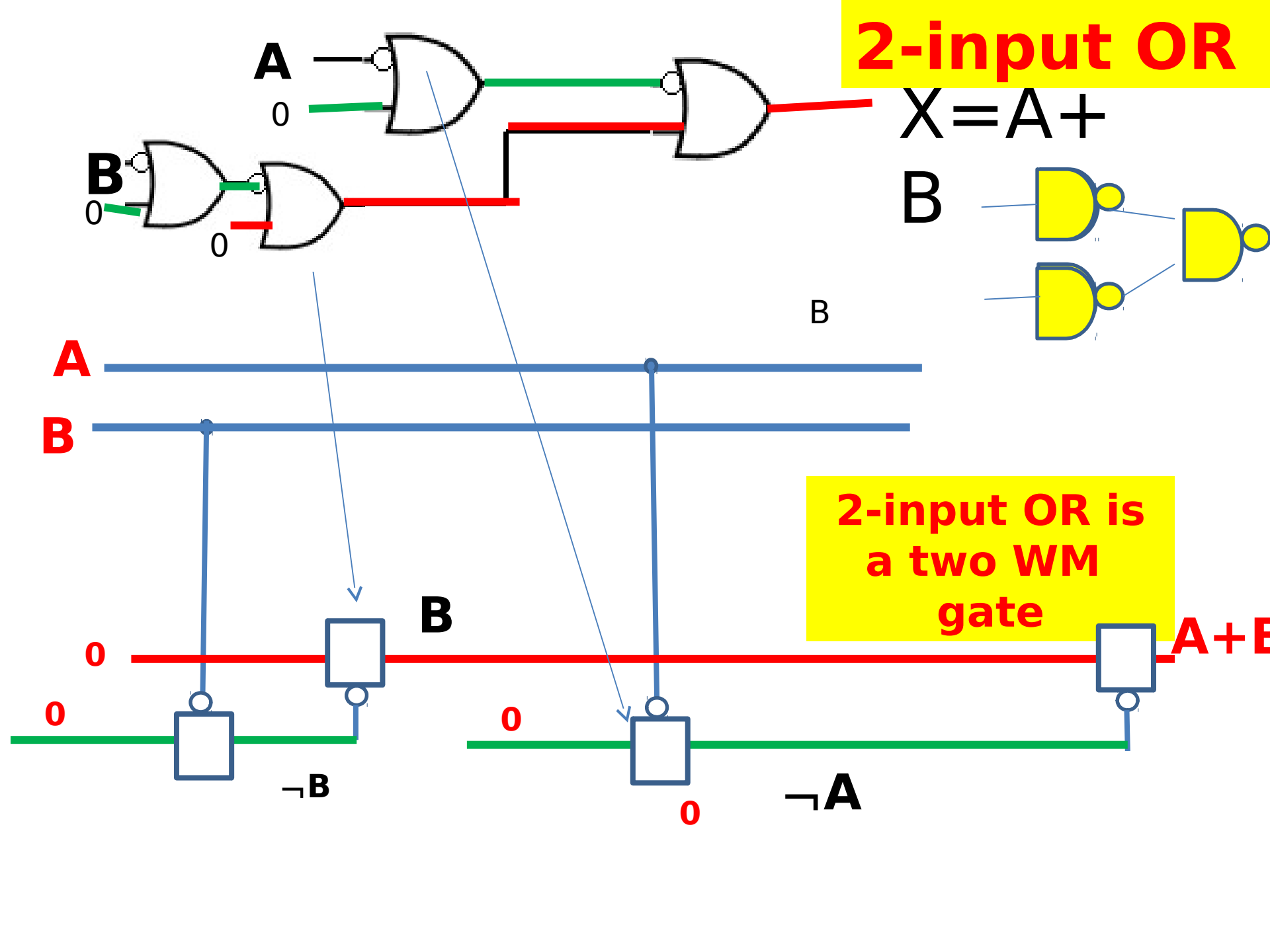


NOT A x



NOT is a one WM gate

2-input OR



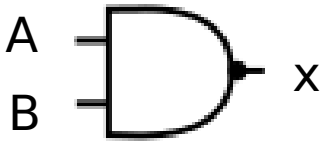
NAN

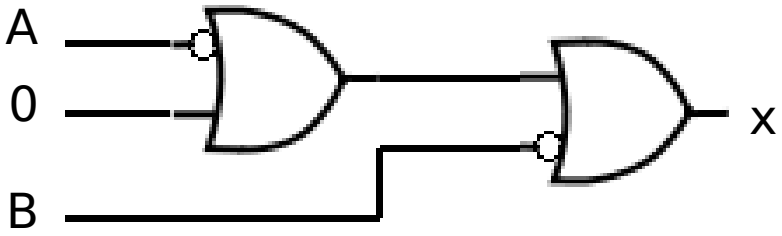
D &

AND

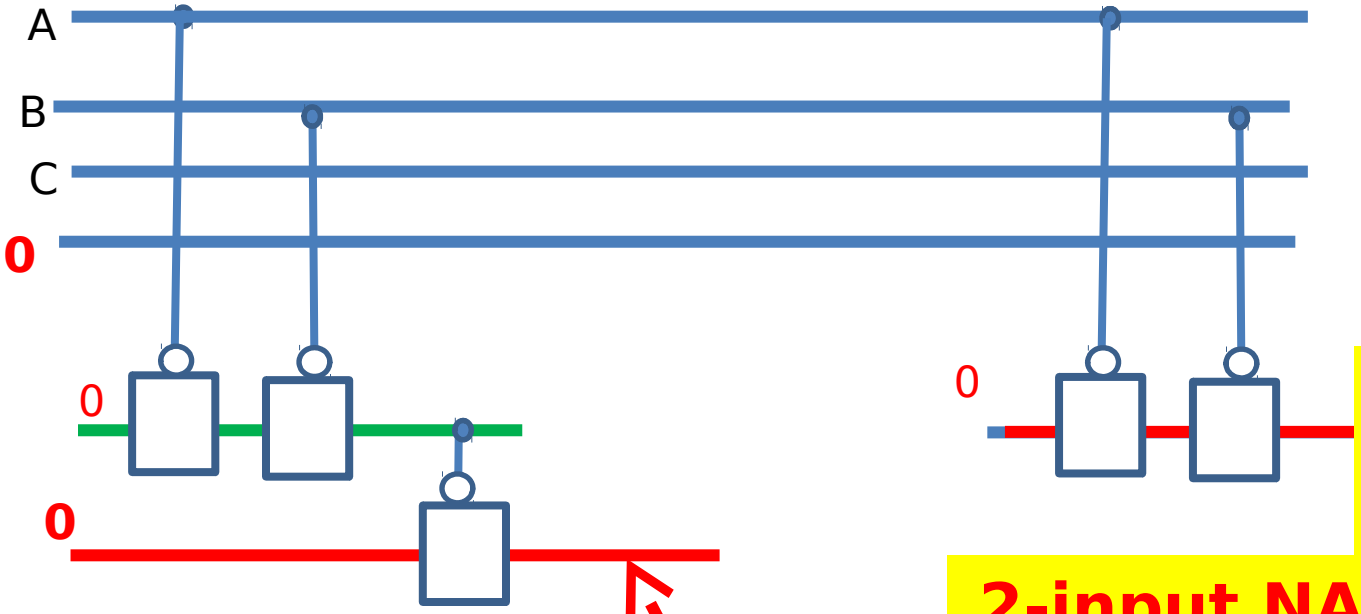
NAND & AND

NAND





NAND & AND

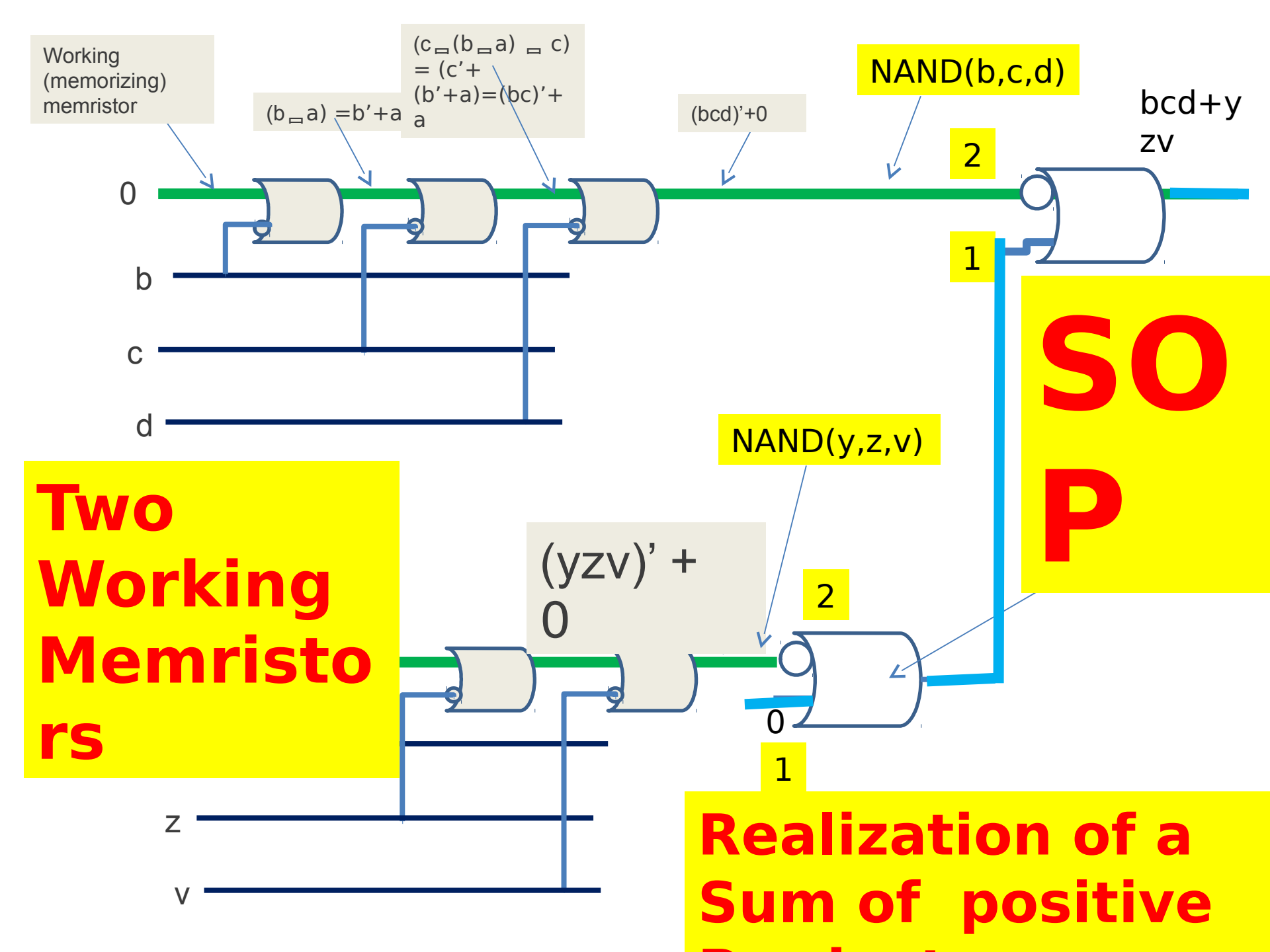


NAND(a, b)

2-input NAND is a one ancilla gate

AND(a, b)

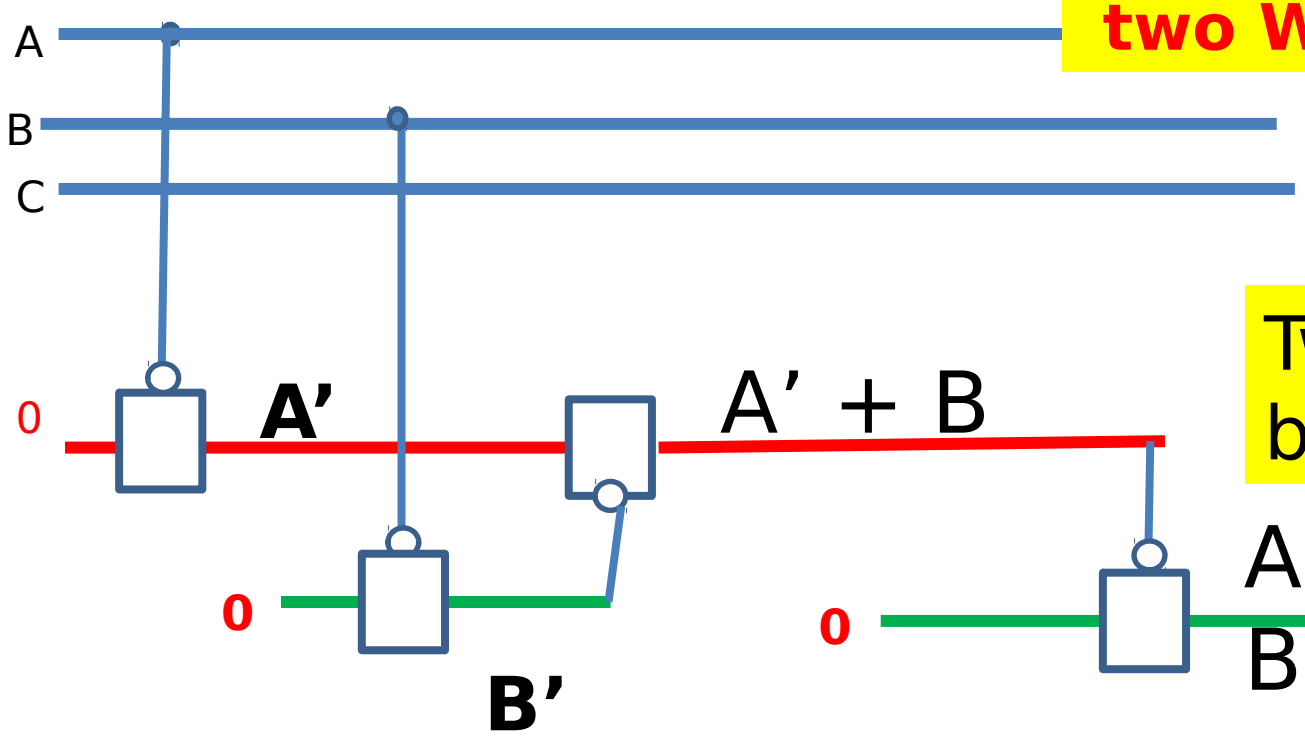
2-input AND is a two


Working (memorizing) memristor
(b ␣ a) =b'+a
(c ␣ (b ␣ a) ␣ c) = (c'+ (b'+a)=(bc)'+ a
(bcd)'+0
NAND(b,c,d)
bcd+yzv
2
1
0
b
c
d
SOP
NAND(y,z,v)
(yzv)' + 0
2
0
1
z
v
Two Working Memristors
Realization of a Sum of positive

Inhibit gate

A * B’ = (A’ + B)’ 2 gates

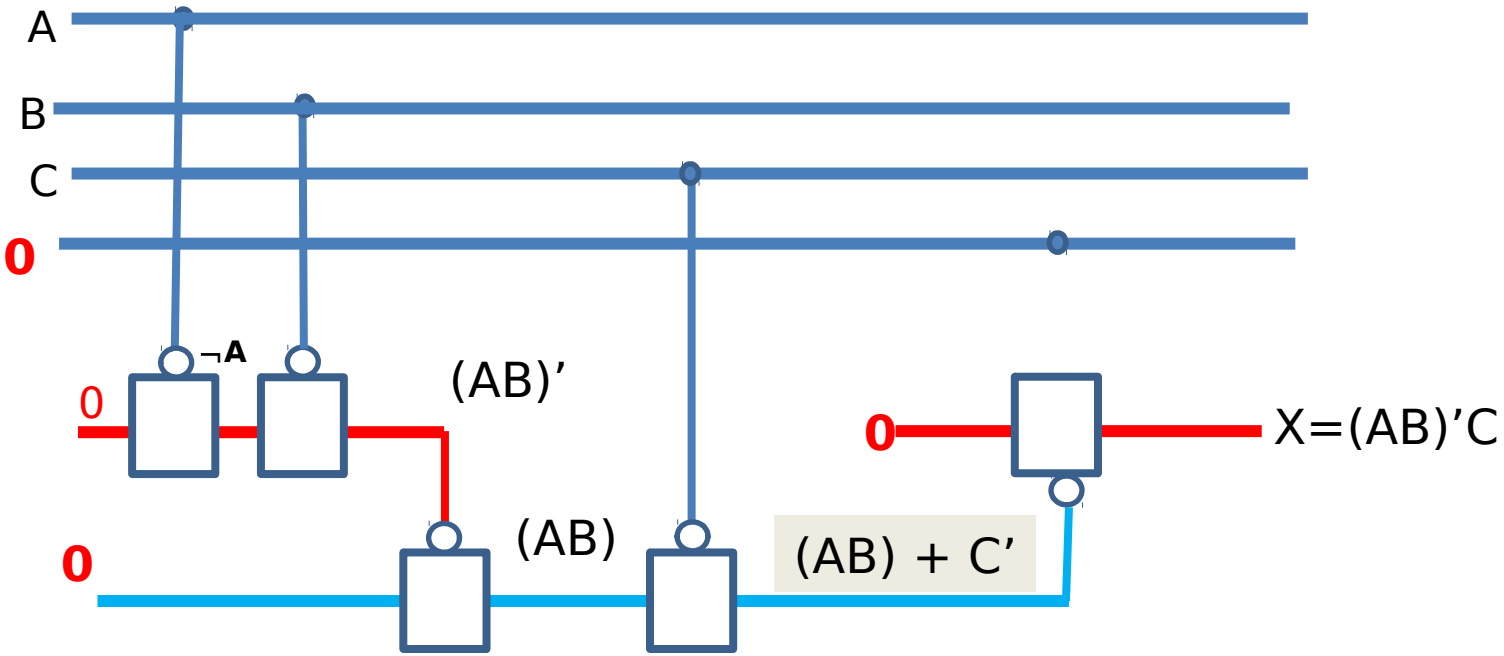
2-input INHIBIT is a two WM gate



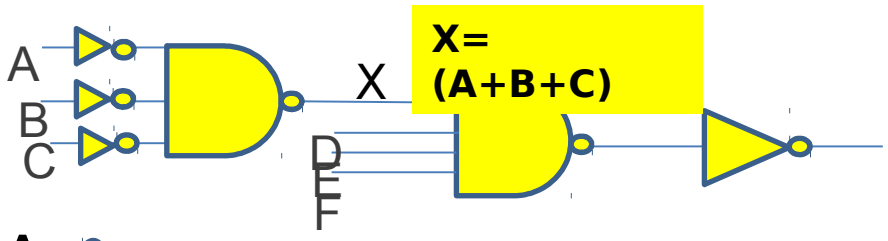
Two working bits

A * B’ = (A’ + B)’

PSE gates



PS is a two WM gate



X= (A+B+C)

PSE gate has 2 WM

A
B
C
D
E
F

¬C

0 0 0 0

¬A

¬B

X'+D'+E' +F'=(XD EF)'

XDE F

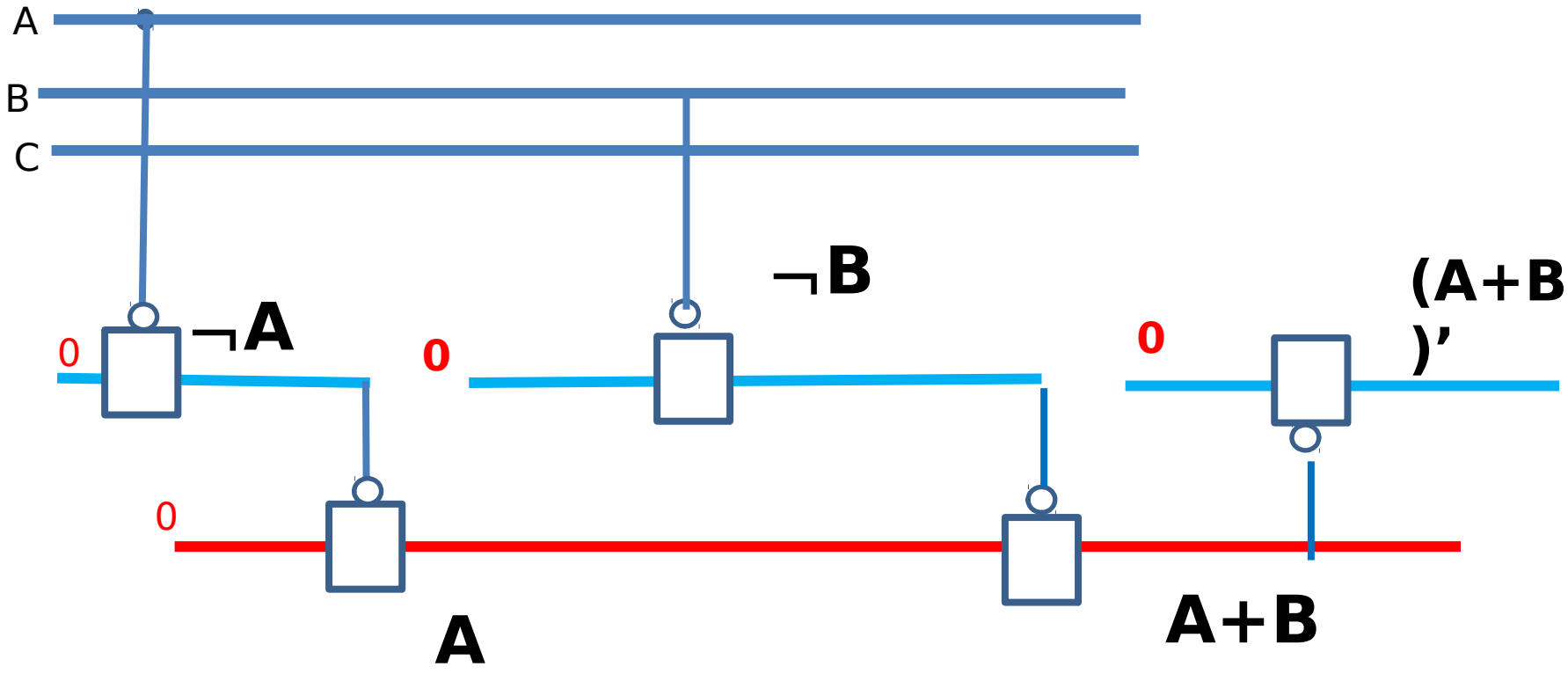
0 0

(A+B+C)=

NOR Gates

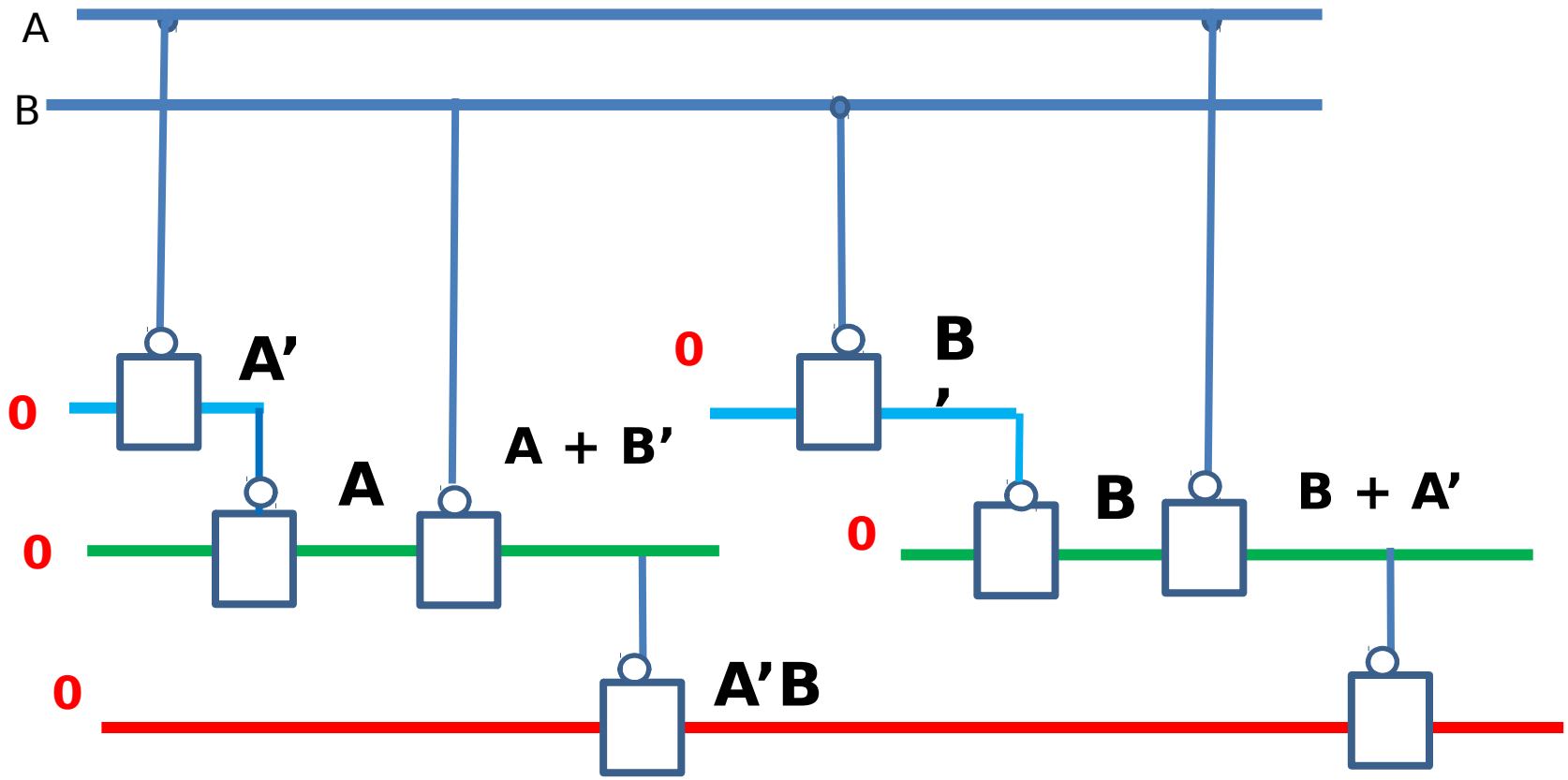
NOR

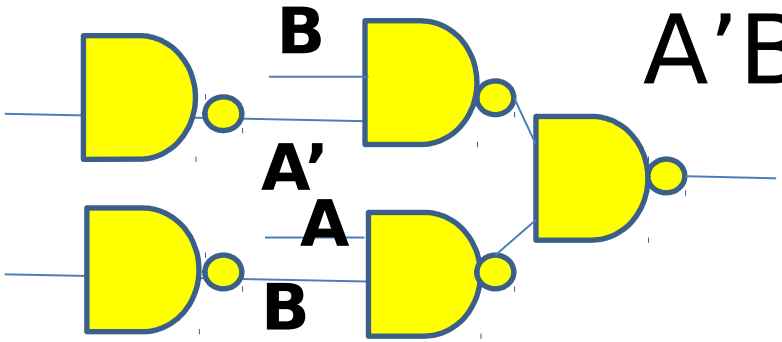
NOR is a two WM bit gate



EXOR Gates

EXOR = 8 literals in NAND = 8 IMPLY

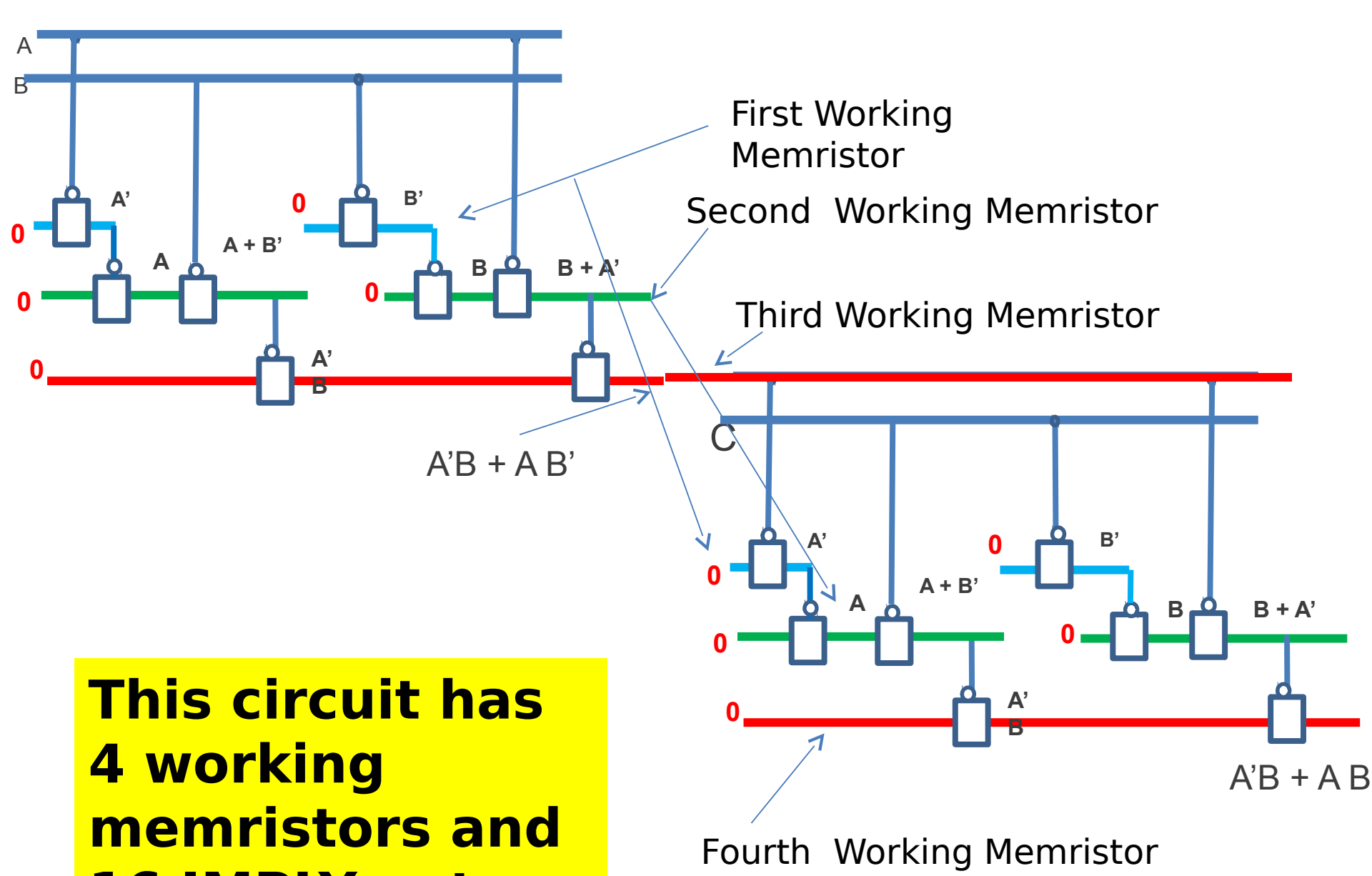




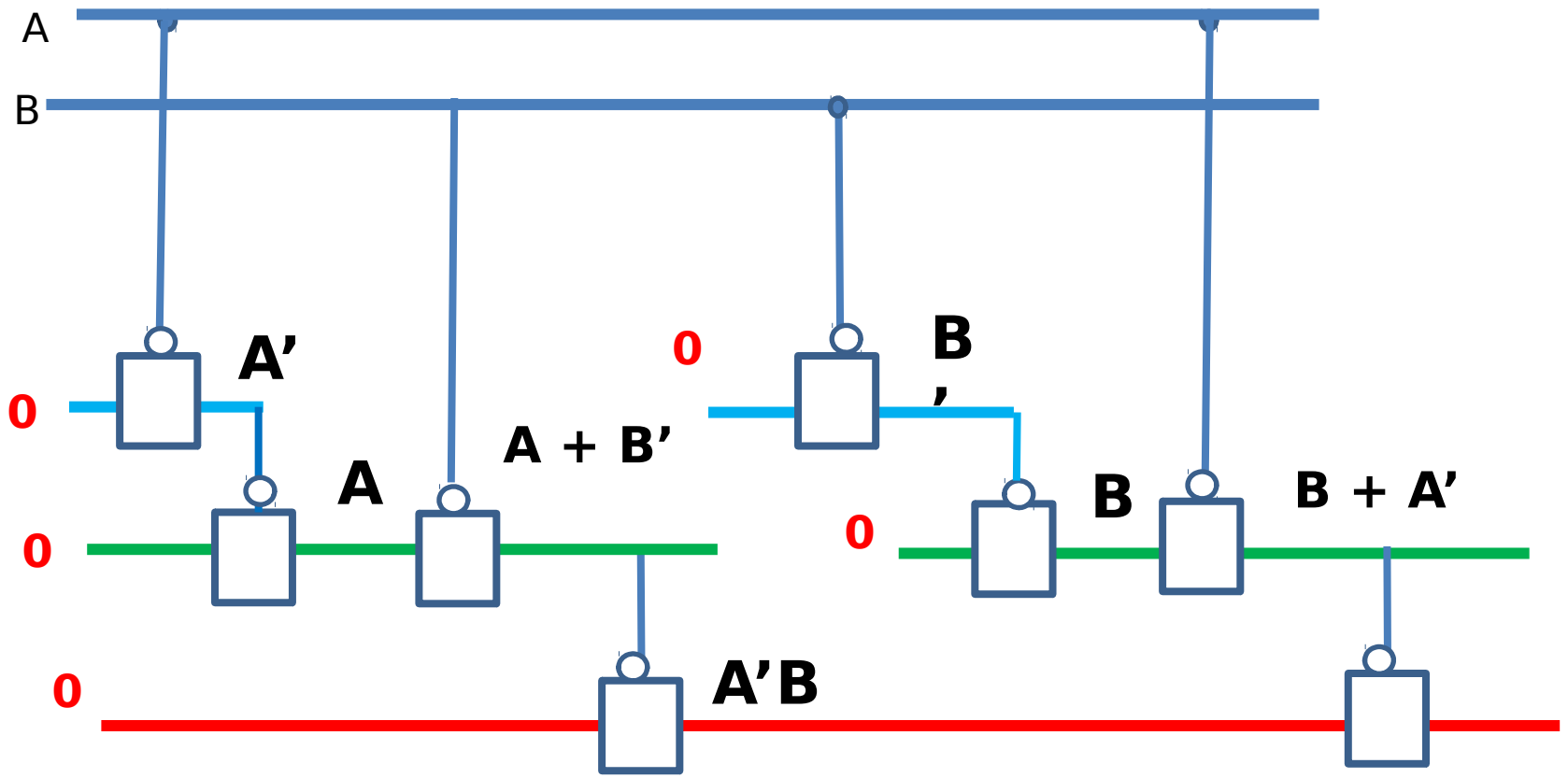
A'B + A B'

EXOR is a three WM

SYNTHESIS WITH EXORS WITH NO LIMIT ON NUMBER OF ANCILLA



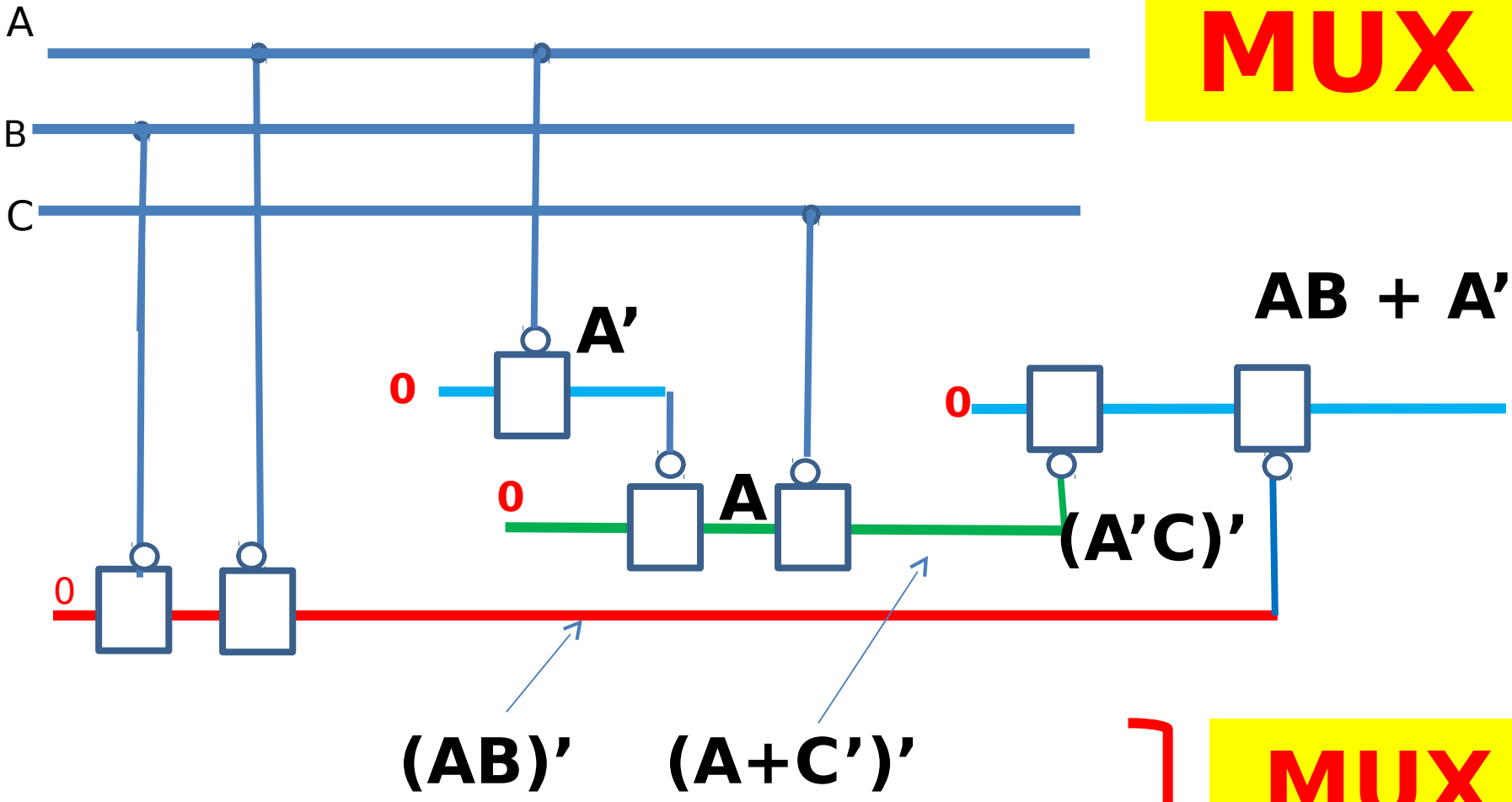
This circuit has 4 working memristors and 16 IMPLY gates

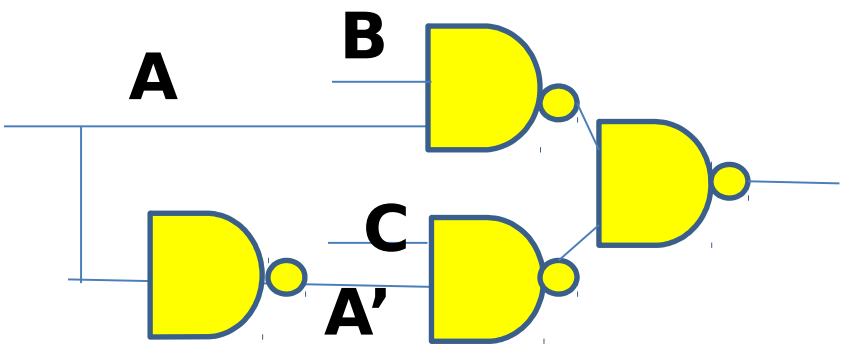


A’B + A B’

MUX

MUX





7 WM expected

MUX is a three ancill

Circuits from reversible gates versus circuits from memristor material implications

Similarities

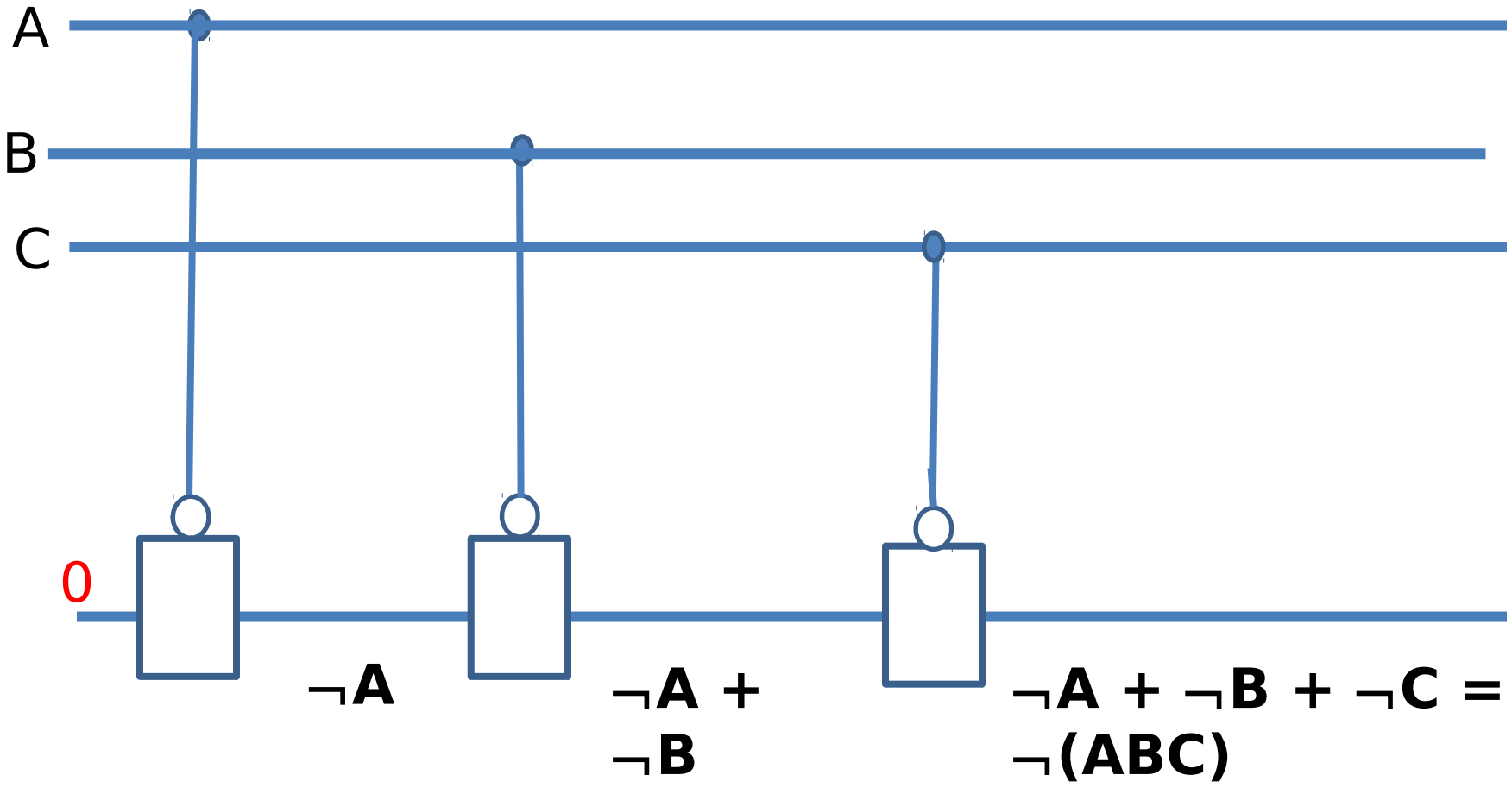
- No fanout
- In-gate memory exists

Differences

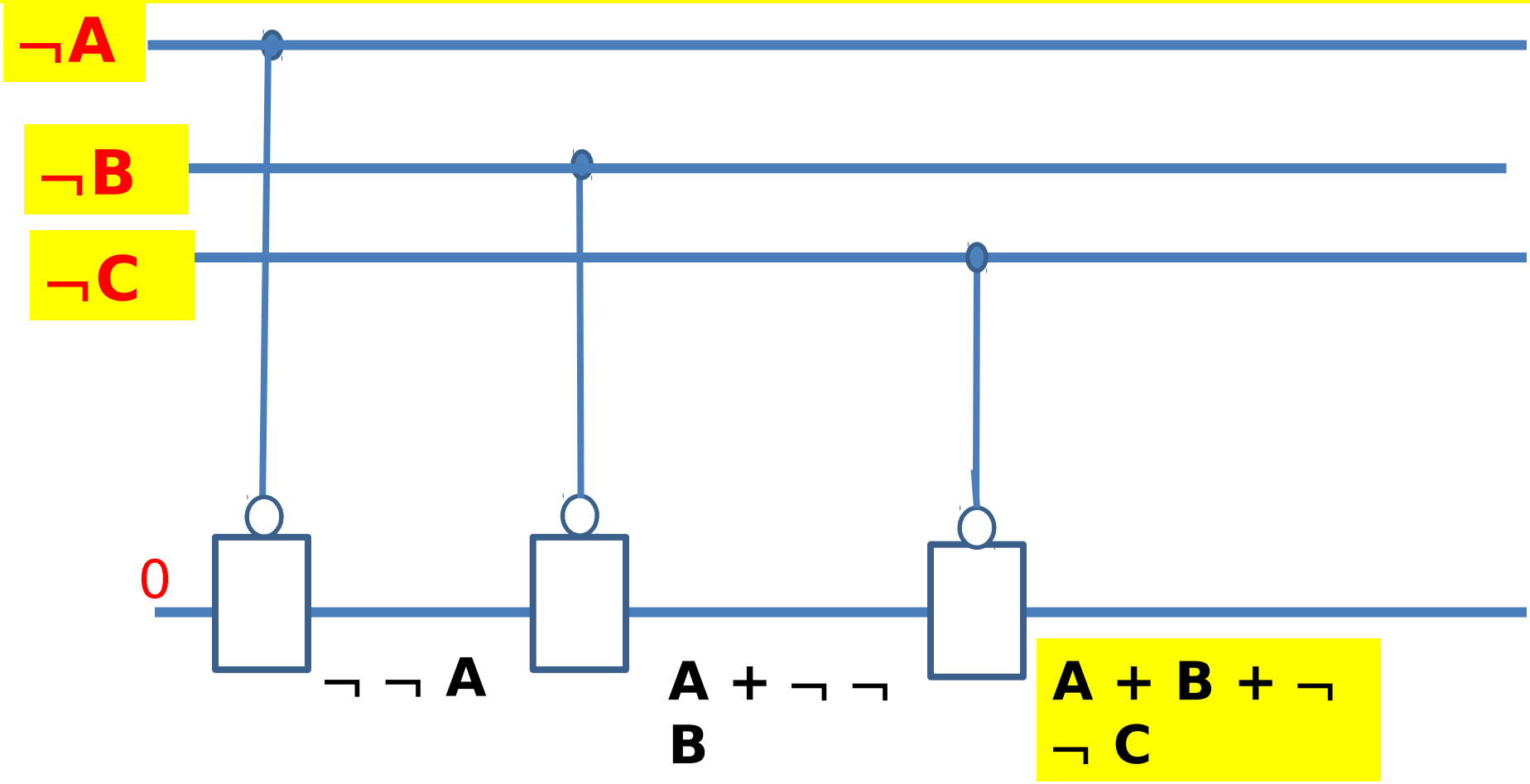
- No inverter
- Different gates

Examples of typical multi-input gates

Realization of positive product (negated) which is multi-input NAND

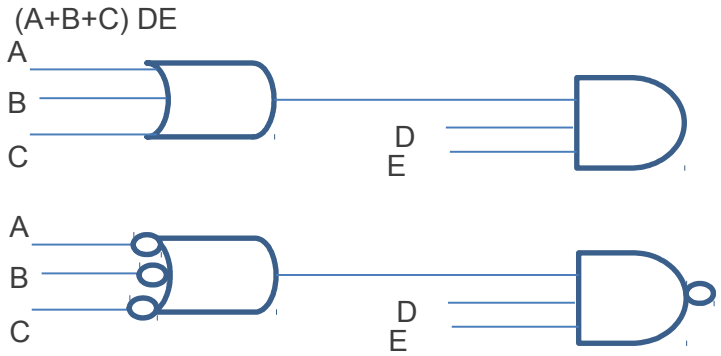


Realization of positive product (negated) which is multi-input OR



Gates from AND and OR similar to PS Gate

Various gates similar to PSE



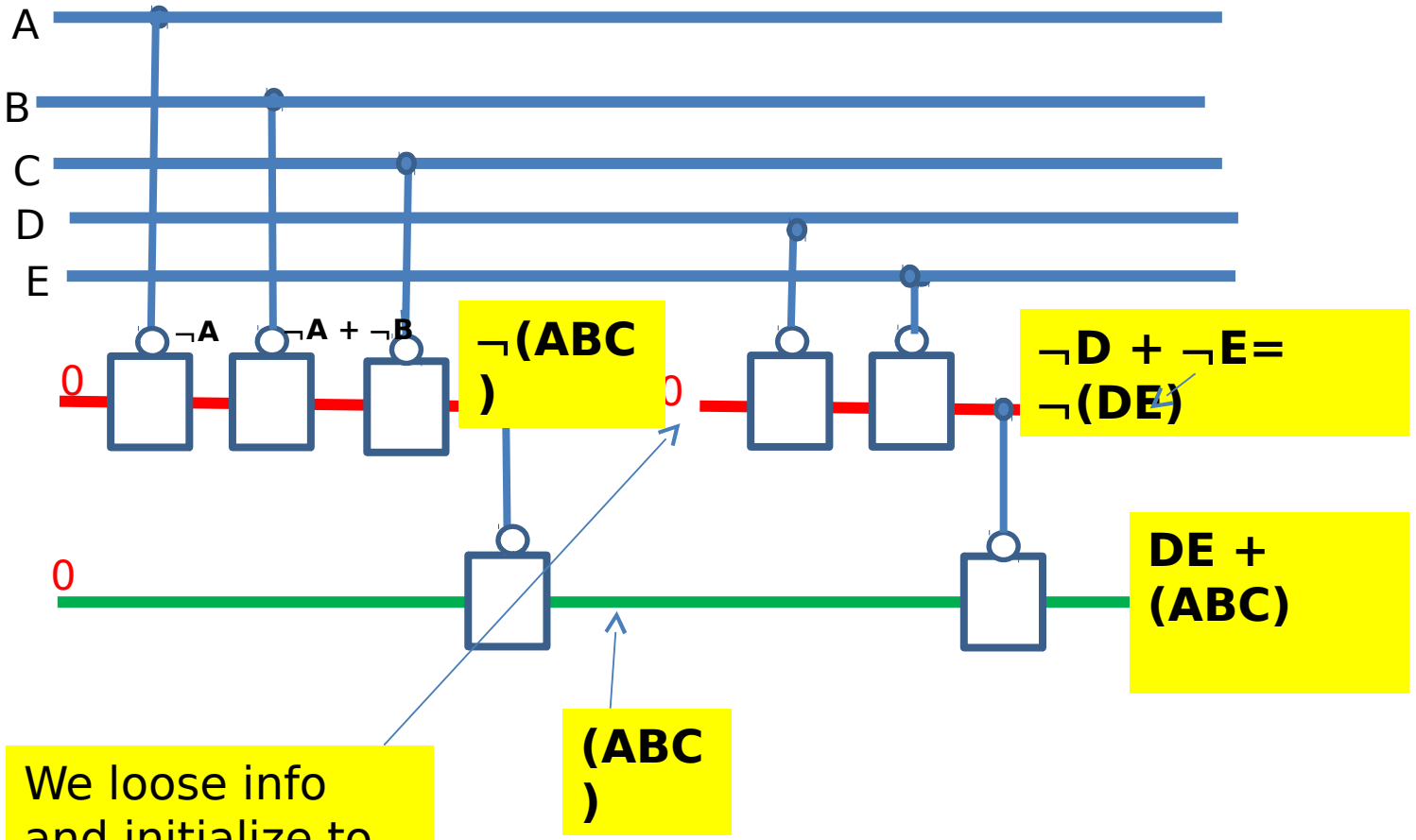
DE * (A+B+C)

PSE gate with not negated inputs

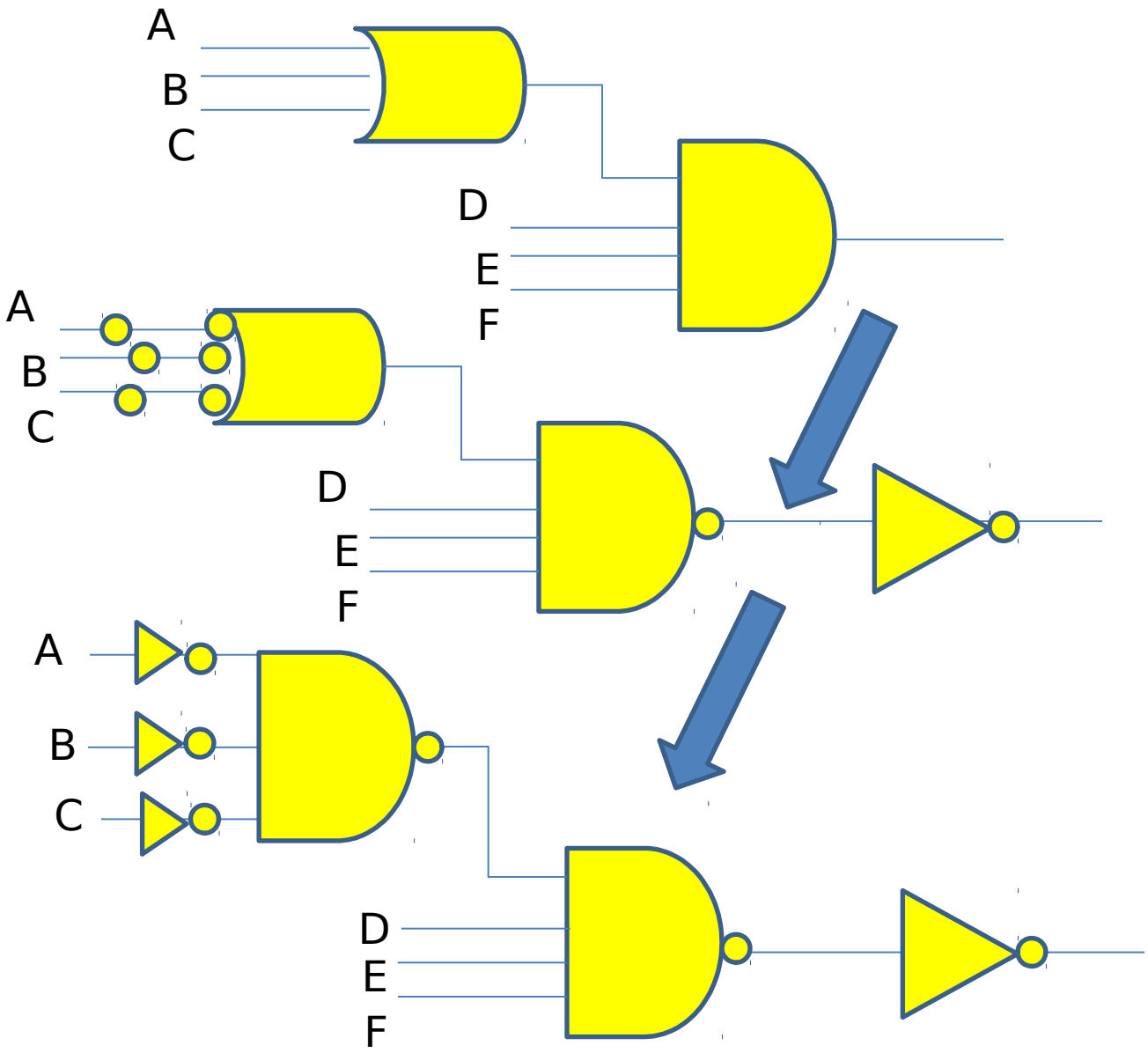
D’ + E’ + (ABC)

TANT IMPLICANT

Sum of any number of positive products is a two-ancilla gate



AND/OR gate (PS gate)



A+B+

METHOD 1:
Example of synthesis using the step-by-step realization method with KMaps

Using the Memristor to build OTHER gates

Example of synthesis based on mapping

Mapping from an existing circuit network

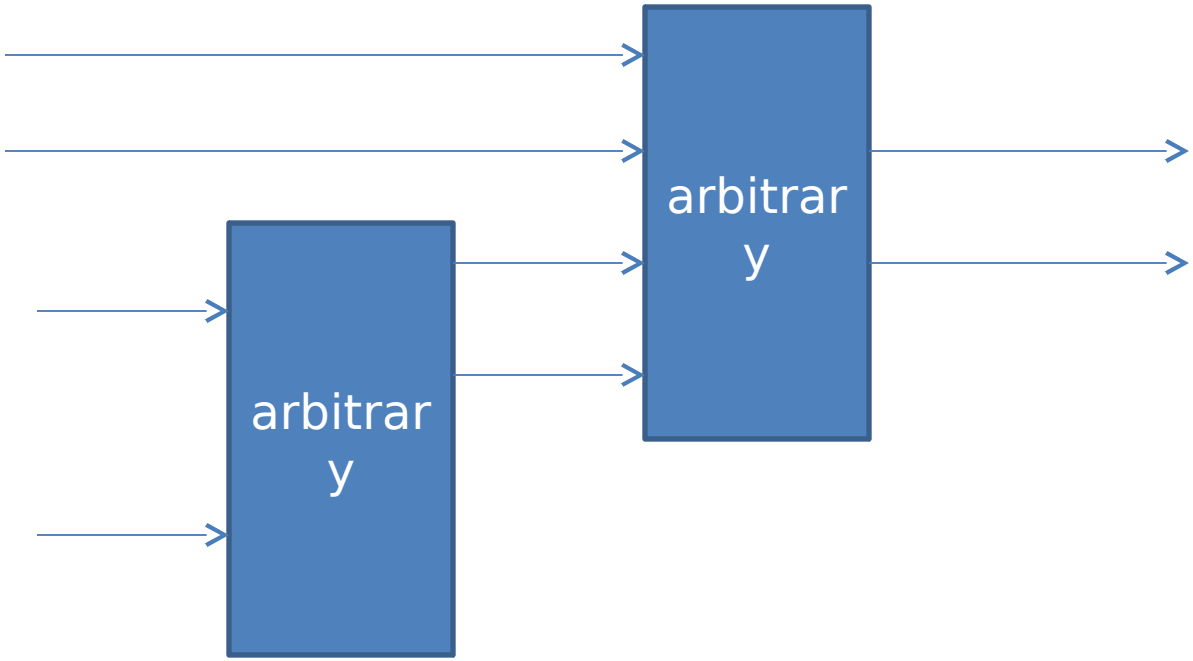
Can be solved using dynamic programming

MATERIAL NOT FOR NOW

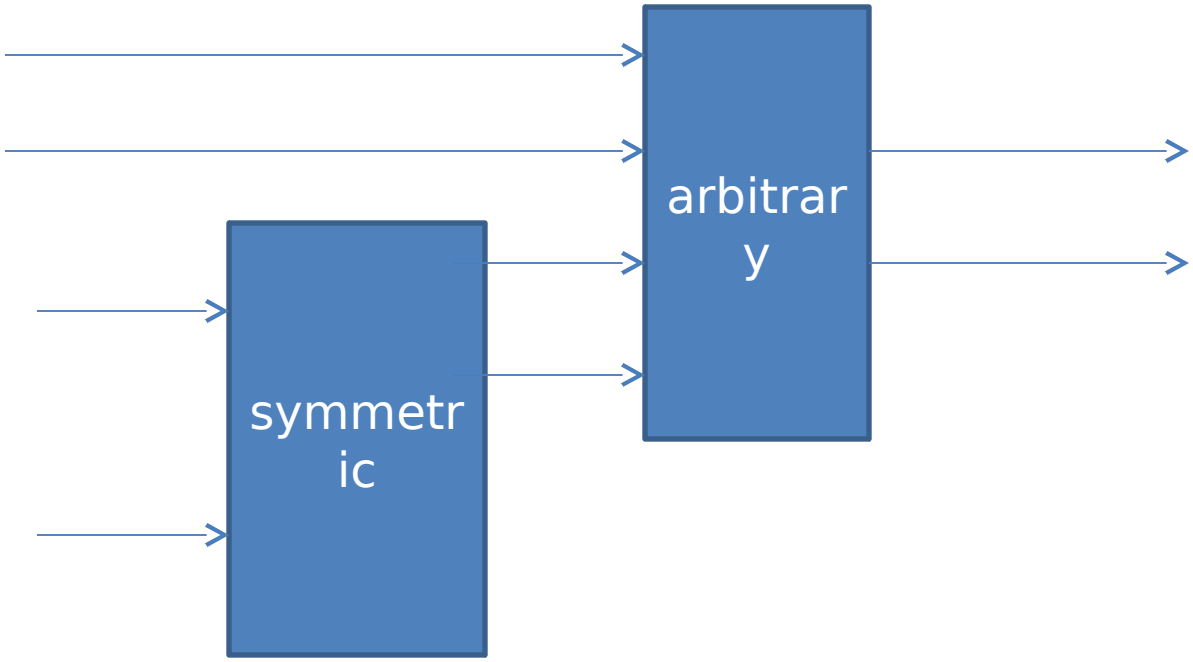
Slides below have

Ashenhurst Curtis Decomposition

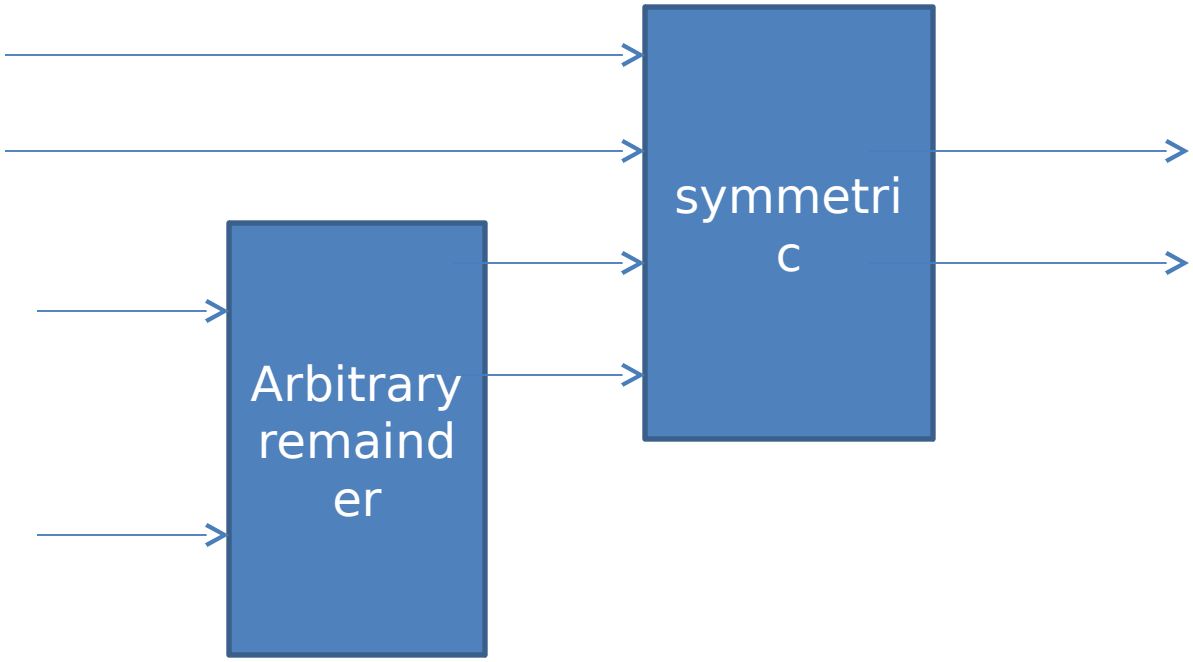
Ashenhurst-Curtis Decomposition



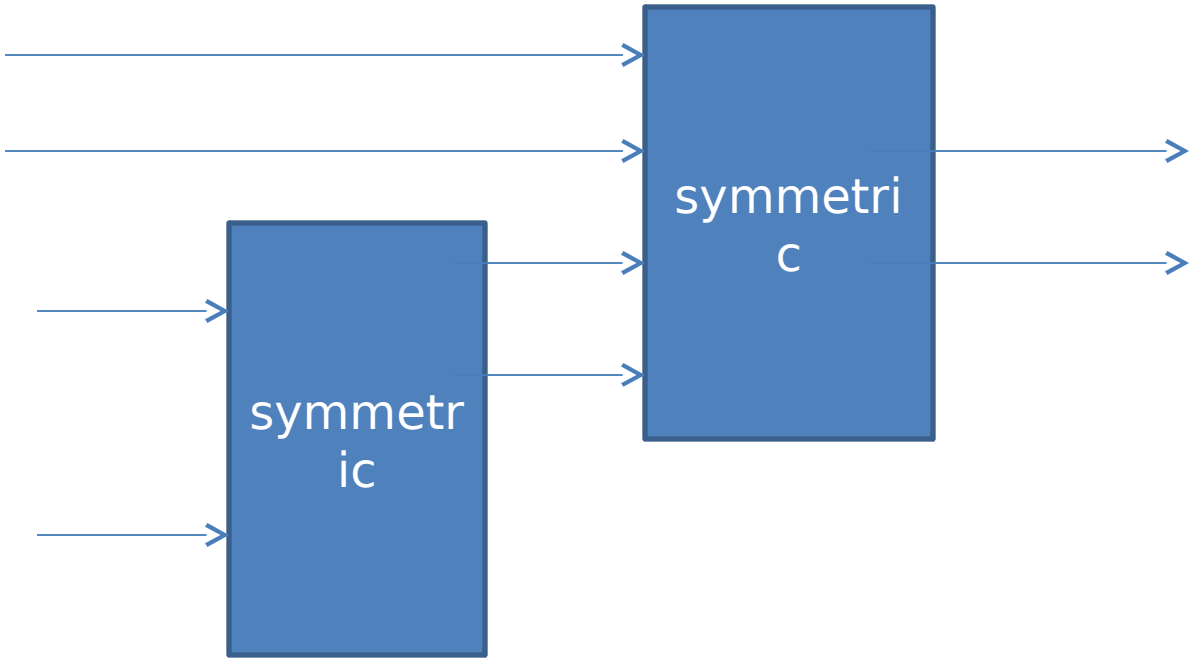
Ashenhurst-Curtis Decomposition with symmetric predecessor



Ashenhurst-Curtis Decomposition with symmetric predecessor



Ashenhurst-Curtis Decomposition with symmetric predecessor and successor



METHOD 2:

Bi-Decomposition of Binary Single-Output Circuits

Boolean Operations

1. Derivative:

$$\frac{\partial f(a,b,c)}{\partial a} = f(a=0,b,c) \oplus f(a=1,b,c)$$

2. Minimum

$$\min_a f(a,b,c) = f(a=0,b,c) \cdot f(a=1,b,c)$$

3. Maximum

$$\max_a f(a,b,c) = f(a=0,b,c) + f(a=1,b,c)$$

3. We find function g on variables a and b only

ab \ cd	00	01	11	10
00	0	0	1	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

f(A, B)

a)

0	0	0	0
1	1	1	1
0	0	0	0
1	1	1	1

g(A)

b)

0	0	1	0
-	-	-	-
0	0	0	0
-	-	-	-

h(A, B)

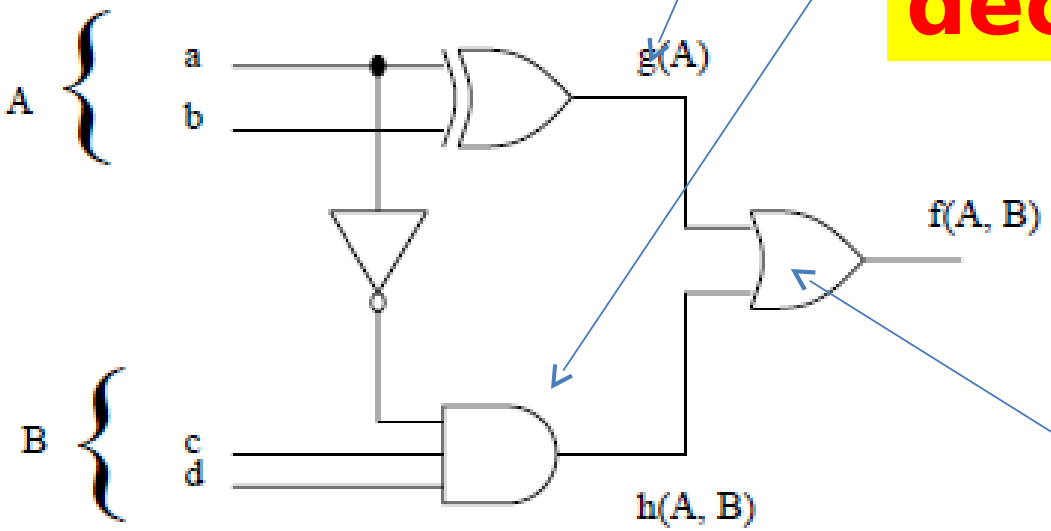
c)

4. We calculate the reminder function g on variables c and d and may be more

Graphical OR decomposition

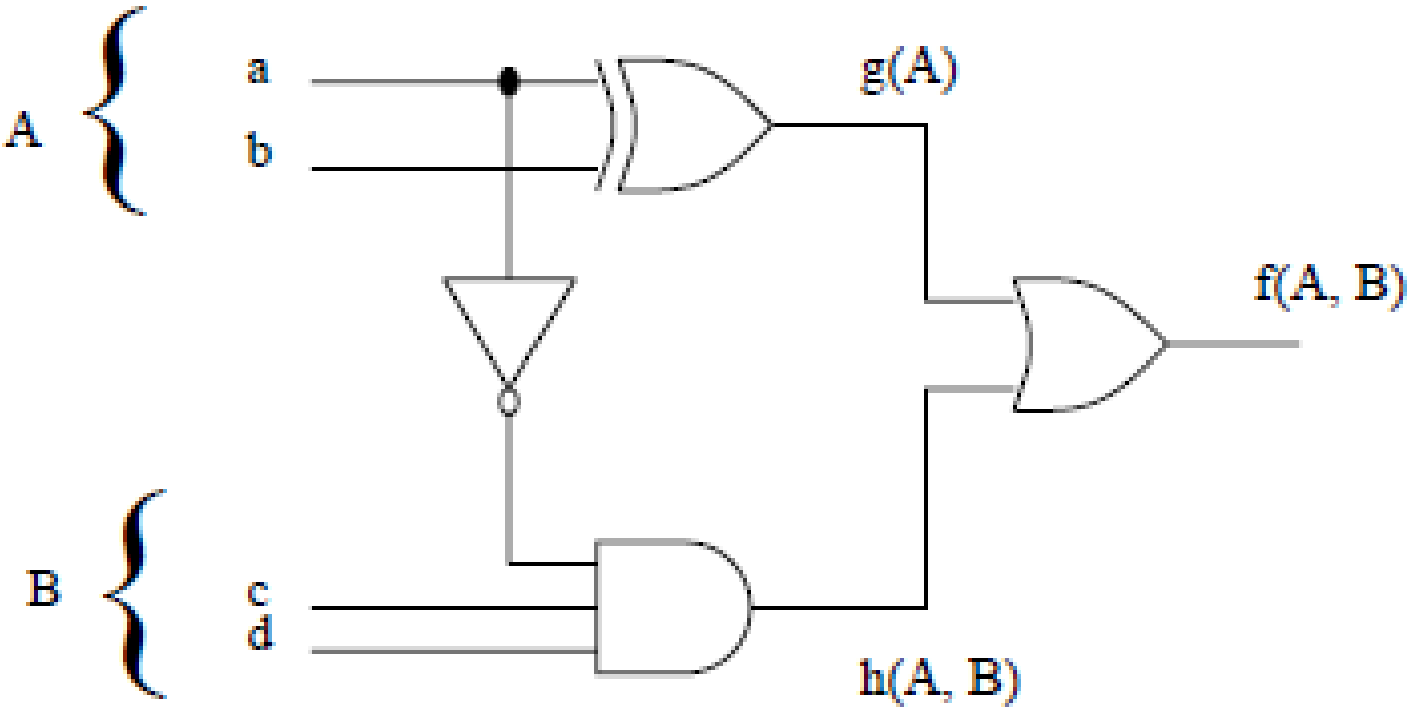
A = ab

B = cd



1. We arbitrarily partitionate all input variables to sets {a,b} and {c,d}
2. We assume

First variant of synthesis



Student should show how this function is realized on memristors to minimize the number of working transistors (Ancilla)

Second variant of synthesis

We assume AND Decomposition

ab \ cd	00	01	11	10
00	1	1	0	1
01	0	0	-	0
11	1	-	1	1
10	0	-	0	0

f(A, B)

a)

1	1	1	1
0	0	0	0
1	-	1	1
0	0	0	0

g(A)

b)

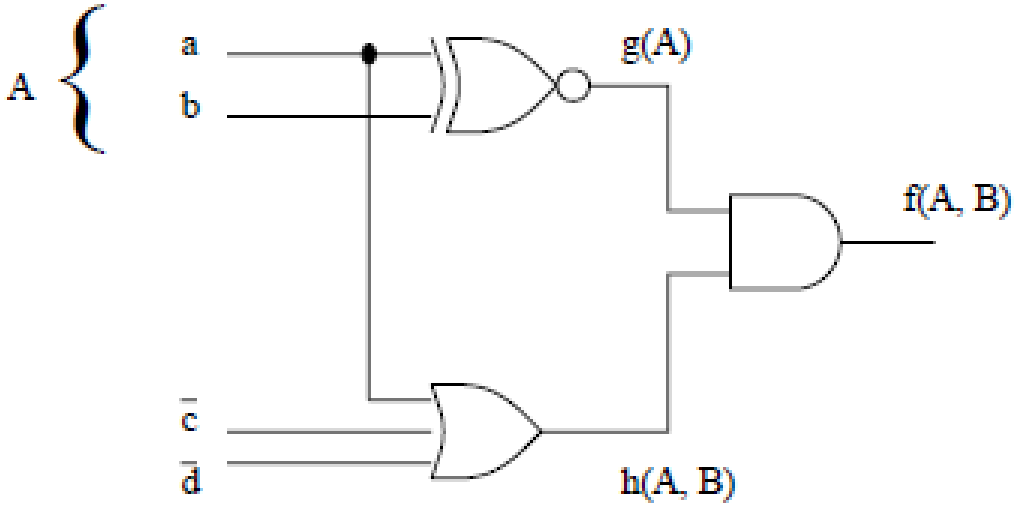
1	1	0	1
-	-	-	-
1	-	1	1
-	-	-	-

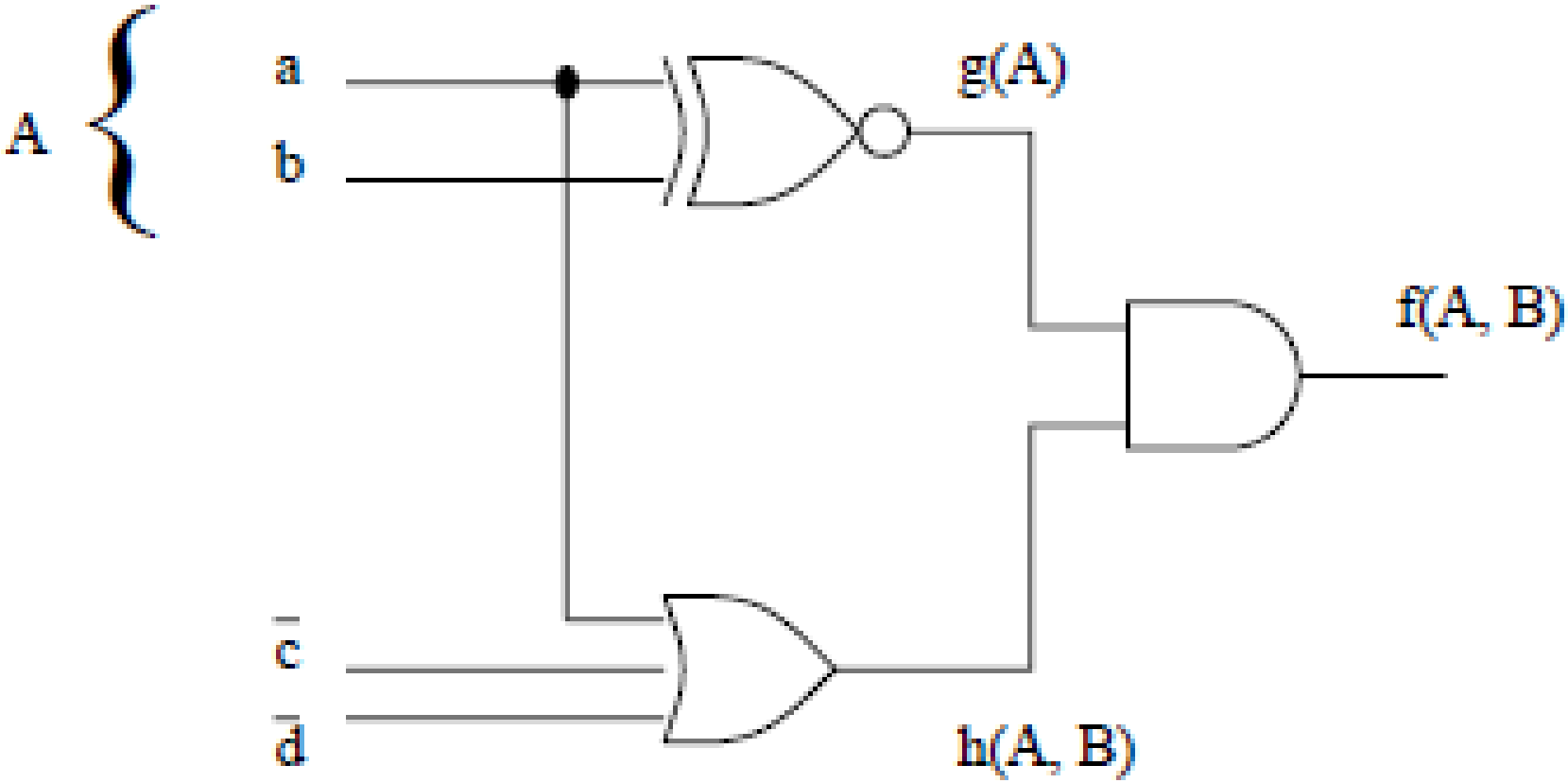
h(A, B)

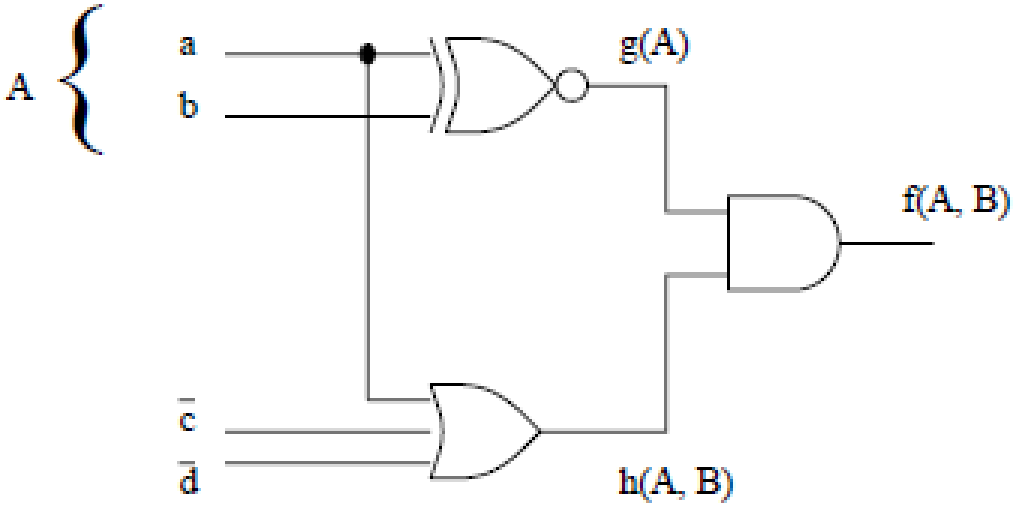
c)

A = ab

B = cd




A
a
b
g(A)
f(A, B)
c
d
h(A, B)


A
a
b
g(A)
f(A, B)
$\bar{c}$
$\bar{d}$
h(A, B)

ab \ cd	00	01	11	10
00	0	1	0	0
01	0	1	1	0
11	1	0	1	1
10	0	1	0	0

f(A, B)

a)

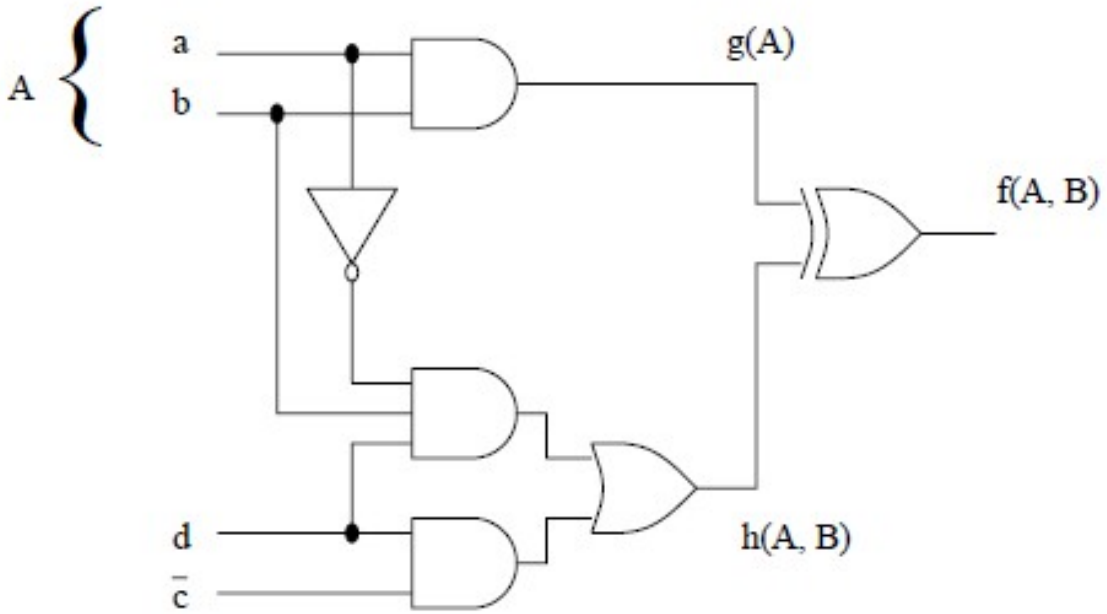
0	1	0	0
0	1	1	0
0	1	0	0
0	1	0	0

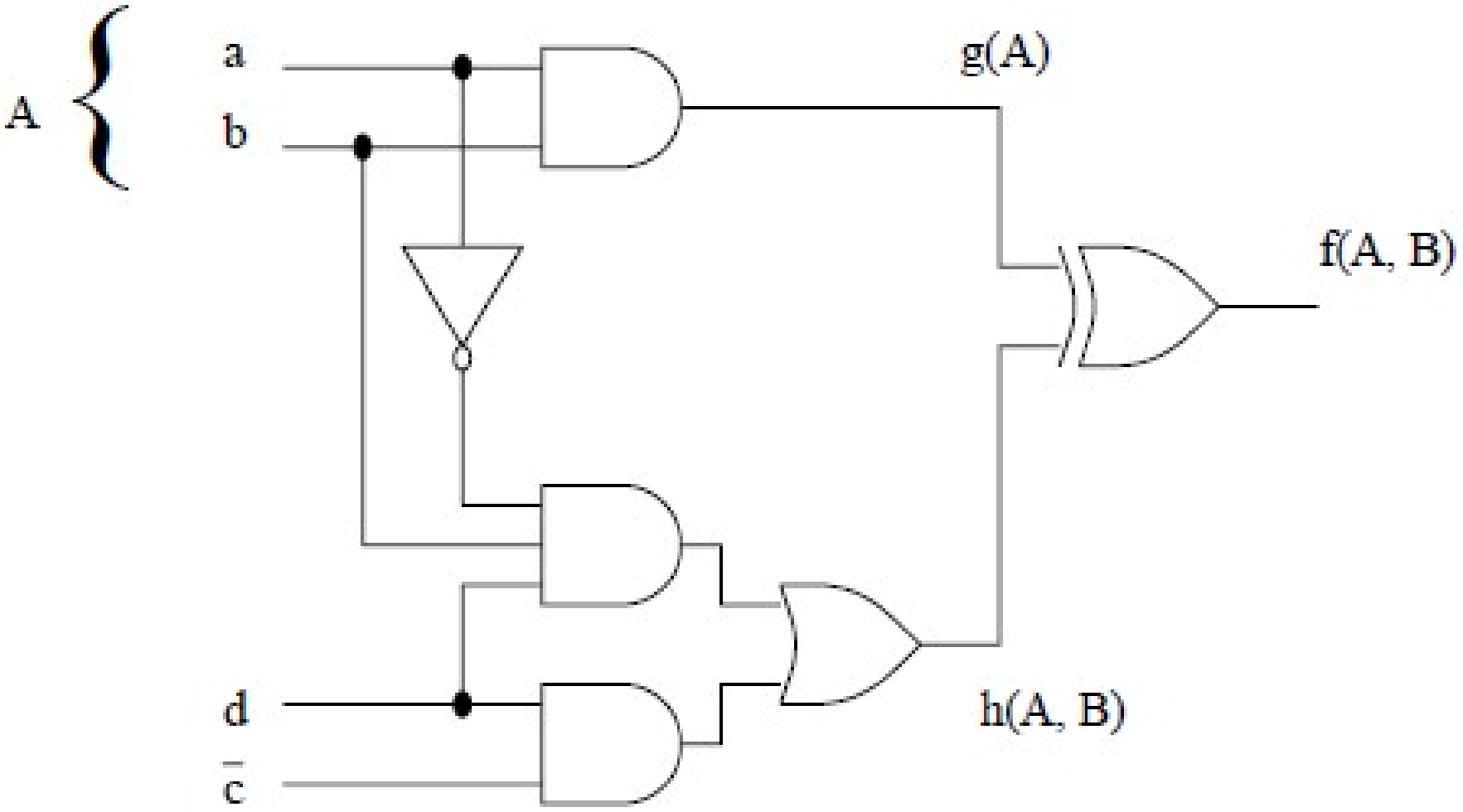
g(A)

b)

A = ab

B = cd




A
a
b
d
$\bar{c}$
g(A)
f(A, B)
h(A, B)

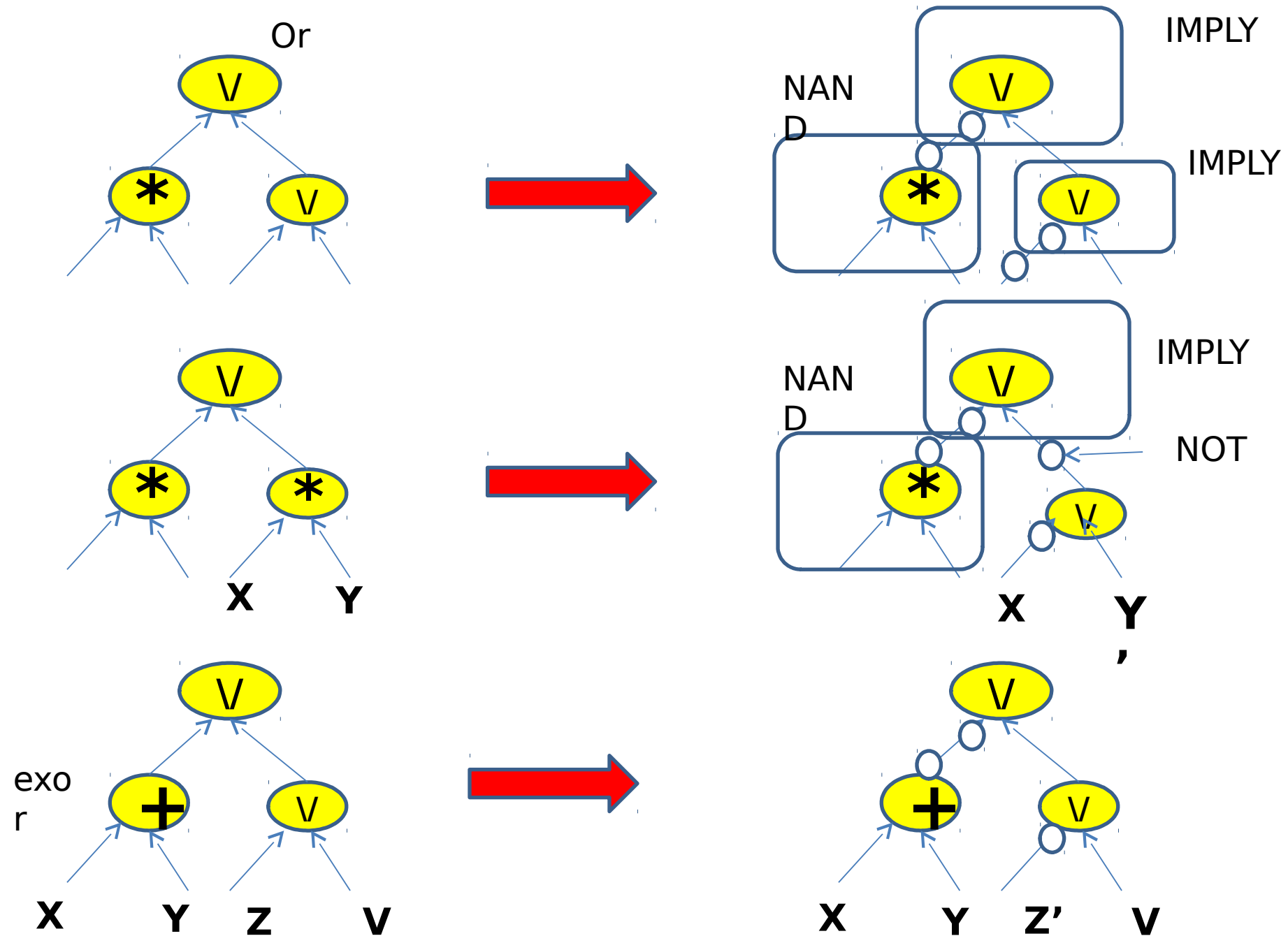
Our final algorithm (initial ideas)

1. Use METHOD 2 - bi-decomposition, first try OR decomposition, next AND, next EXOR.
2. Count width of circuit at every stage.
3. If the desired width is exceeded continue function using METHOD 1.

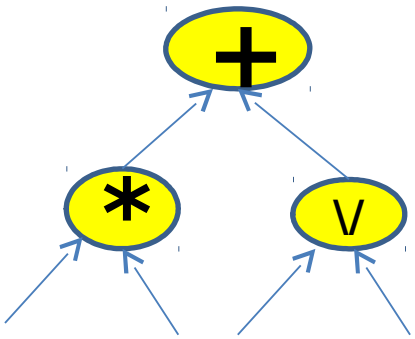
1. Anika, give examples of combined method on function of 5 variables or 6
2. variables.

   Write the pseudo-code of the final algorithm before programming.

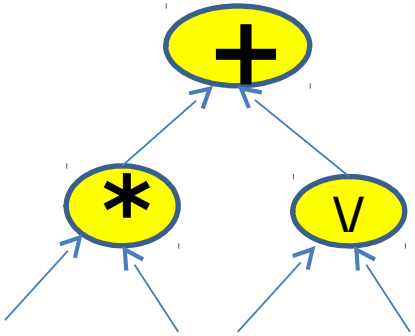
The method to analyze all circuits in bi-decomposition, and how to realize them with memristors

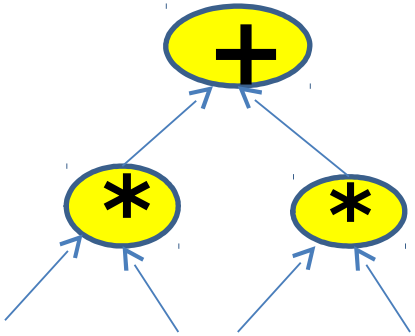

Or
V
*
V
IMPLY
NAN
D
IMPLY
X
Y
NOT
Y
'
exo
r
+
Z
Z'

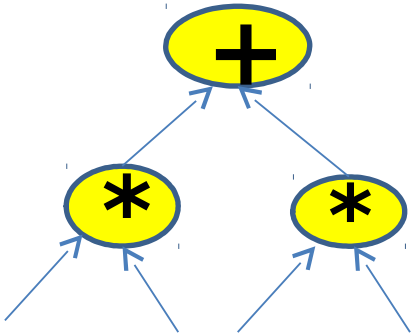
exor

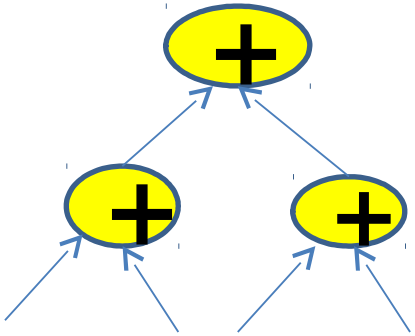


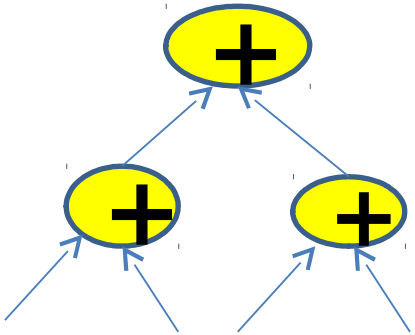
Please complete



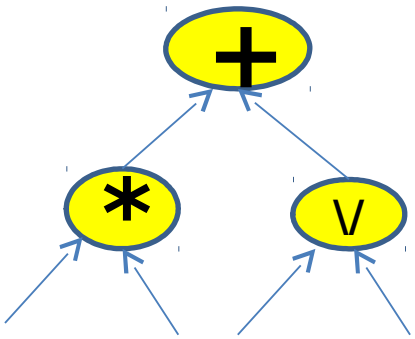




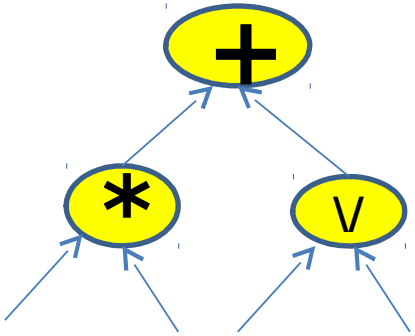


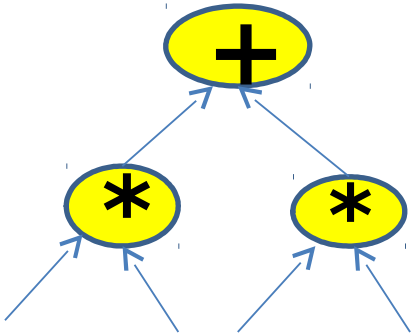


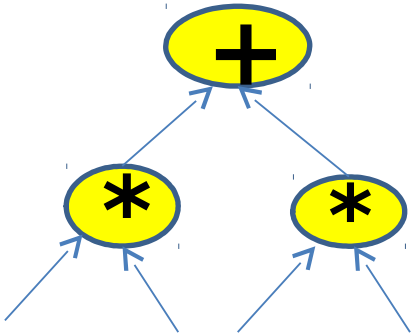
exor

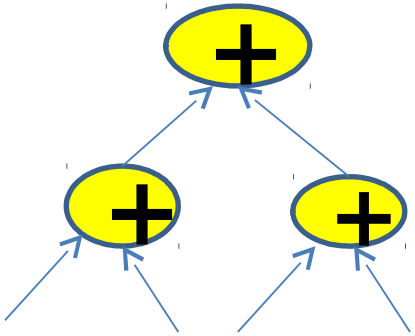


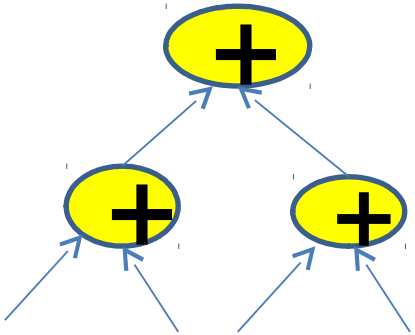
Please complete

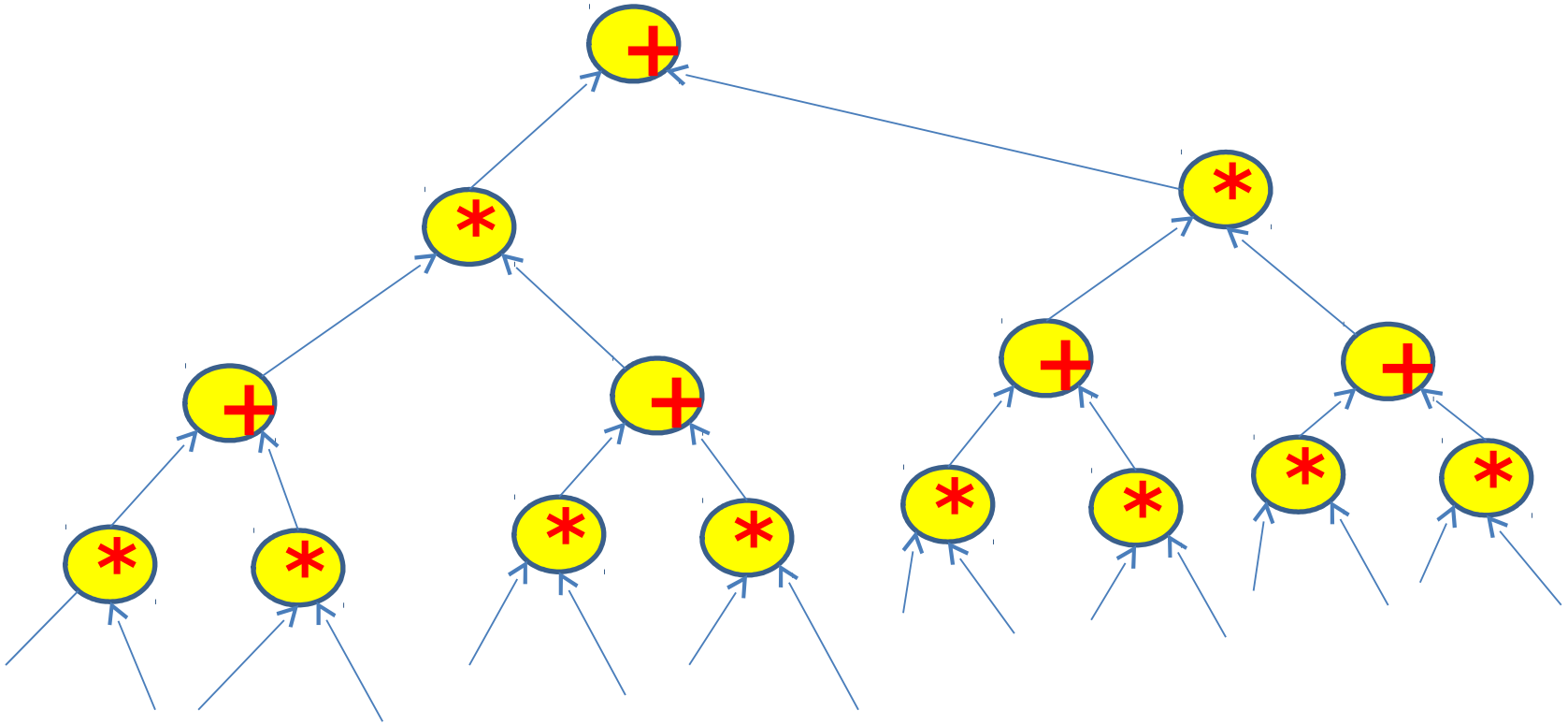


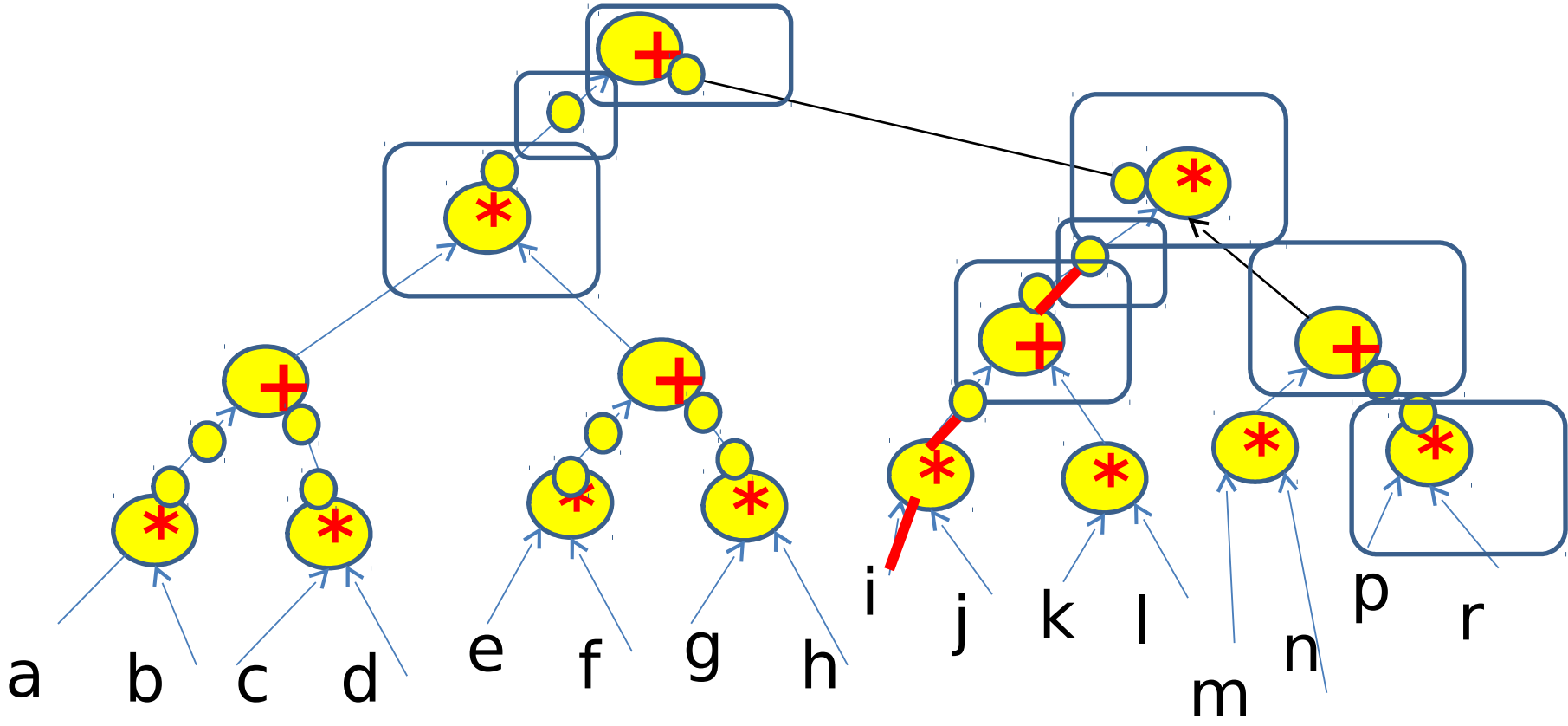




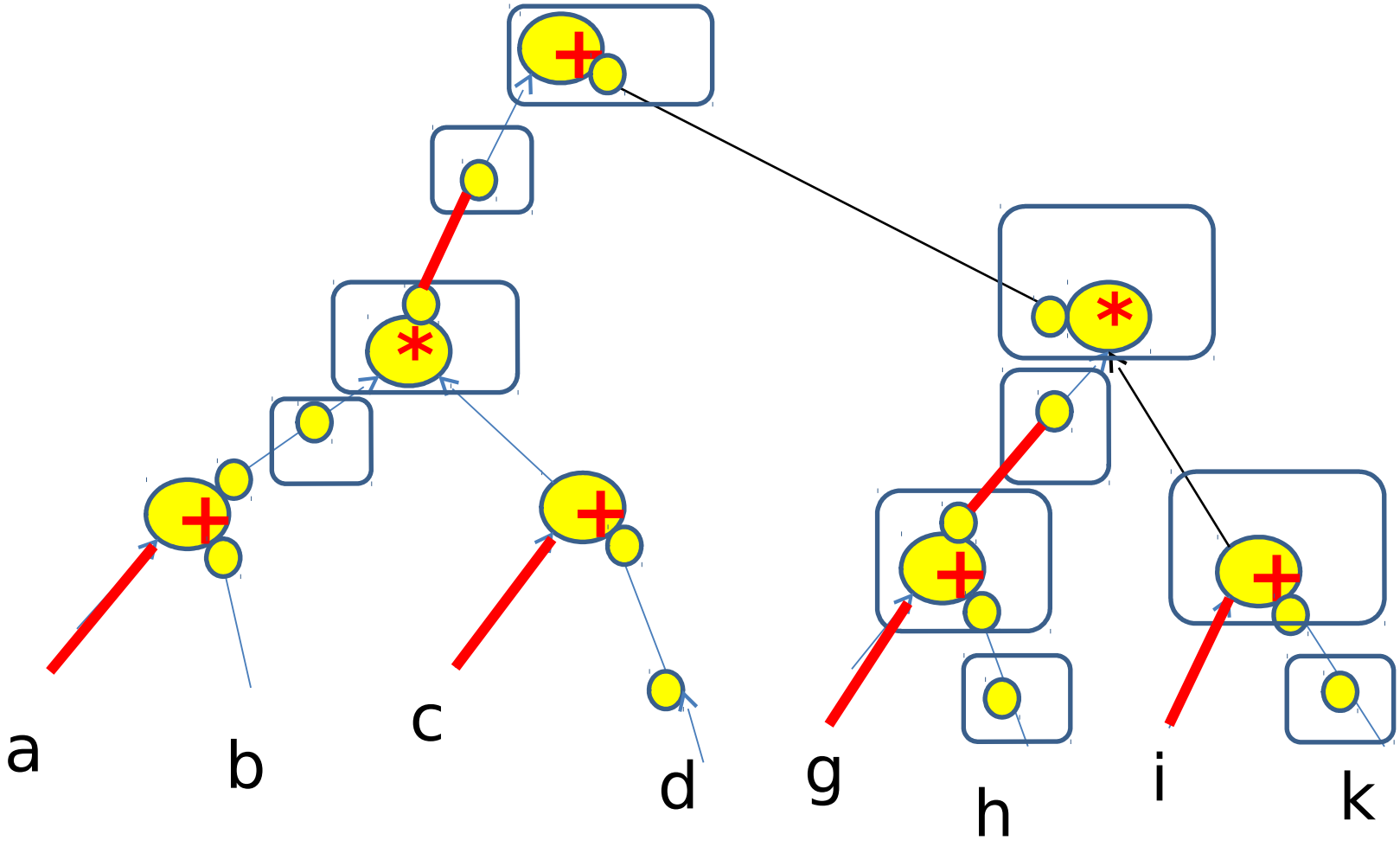


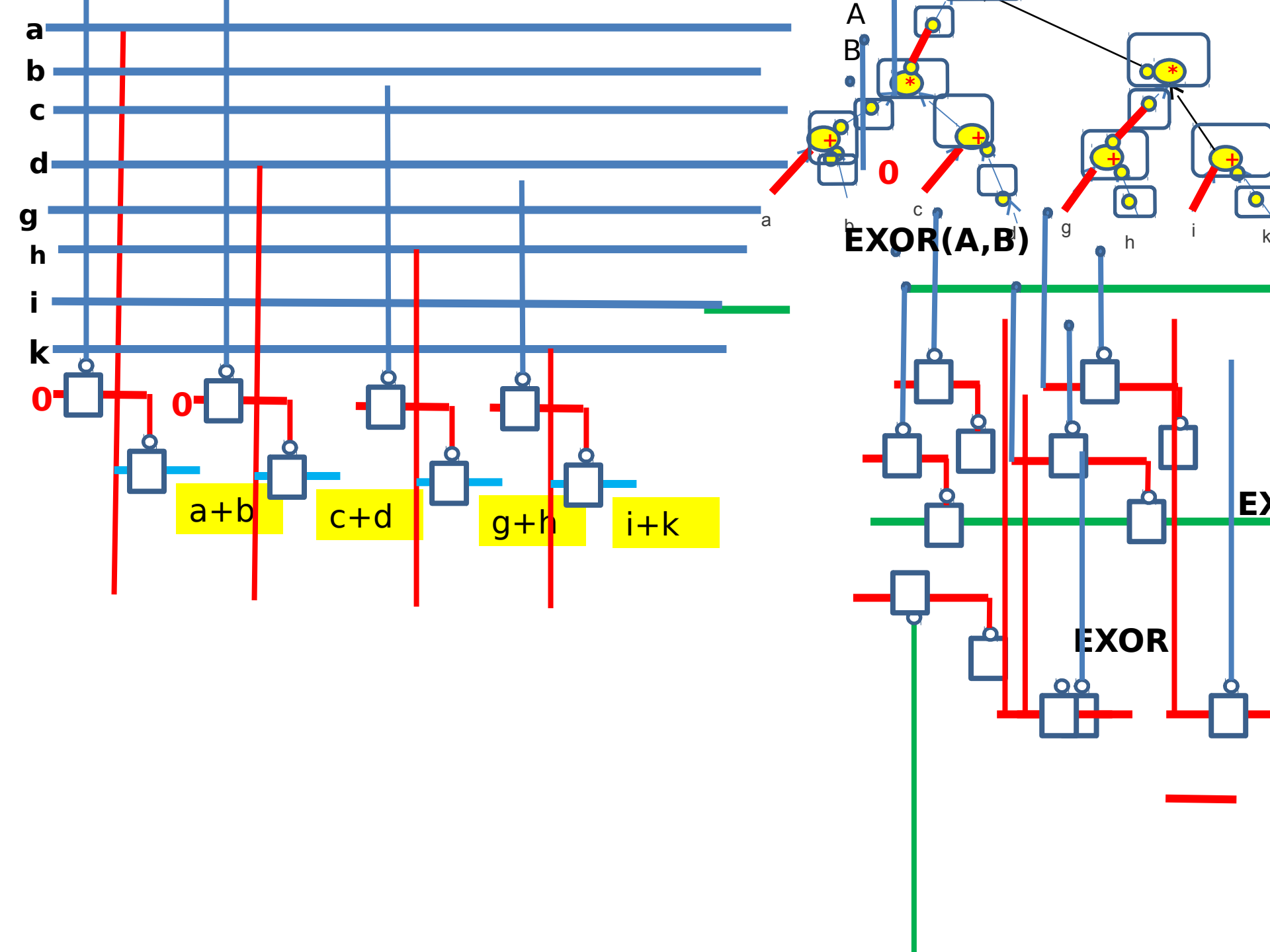


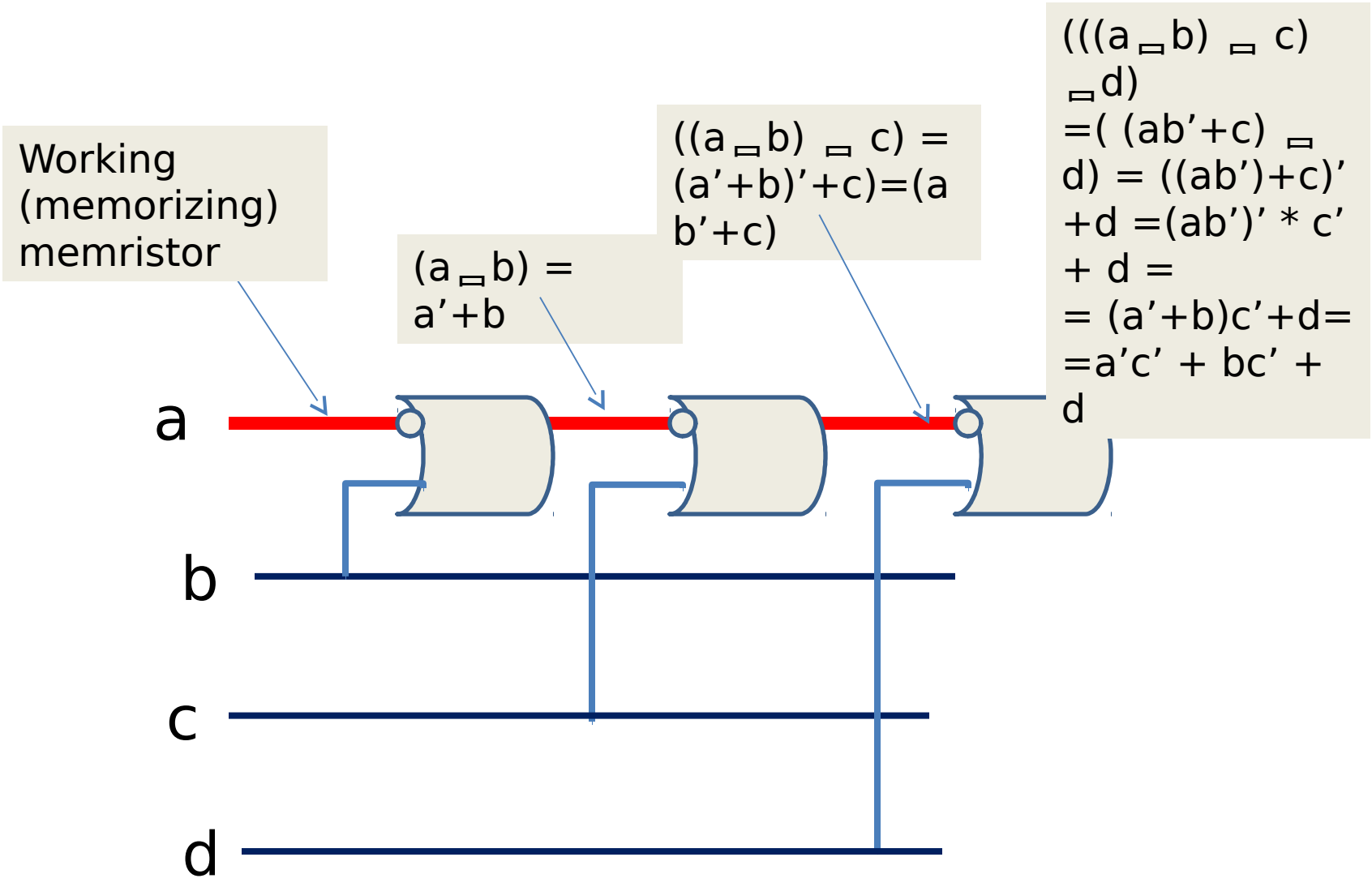




Can we evaluate quickly how many two-input NAND and IMPLY gates and how many ancilla we need directly from a tree?

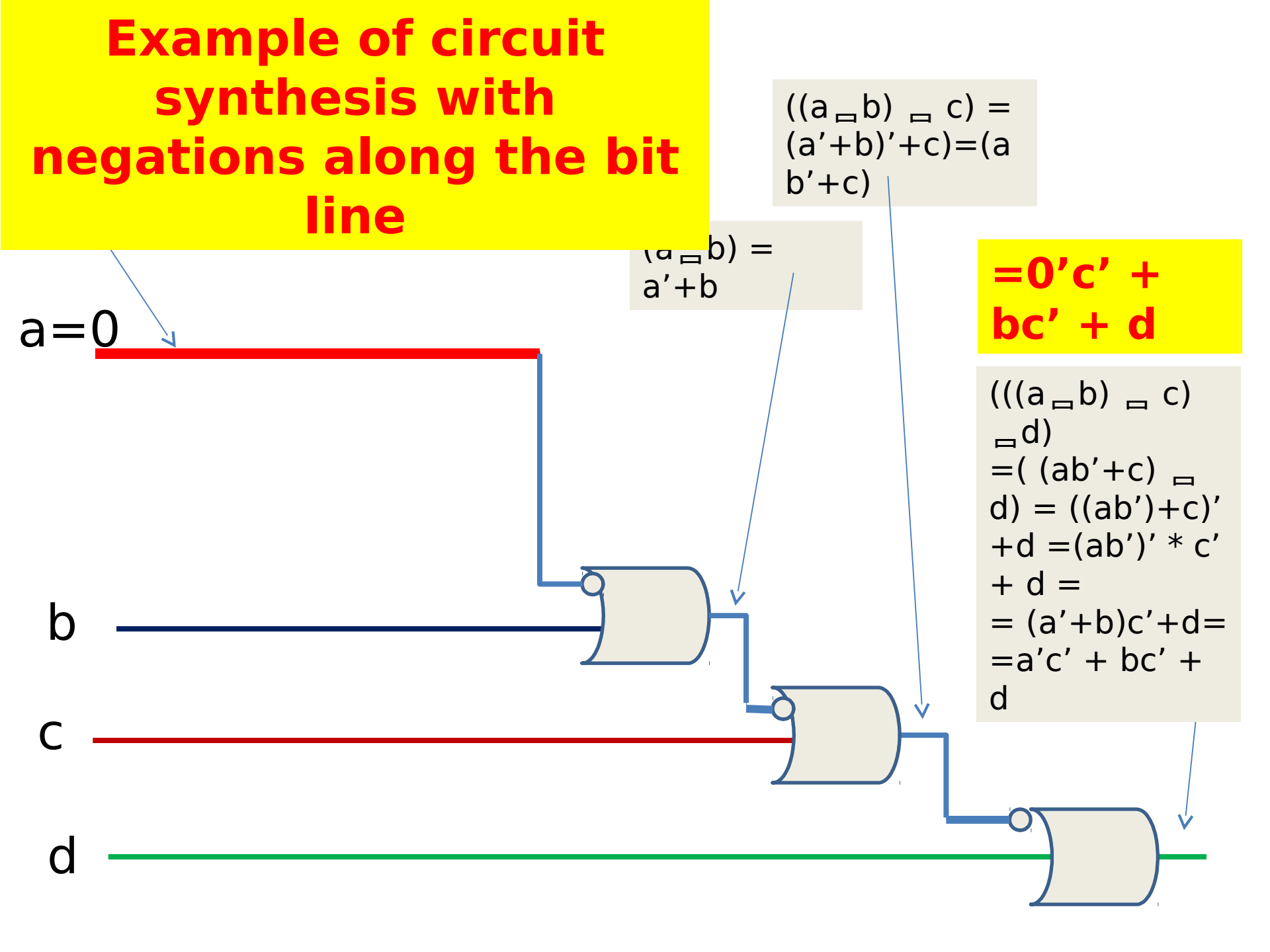

a
b
c
d
g
h
i
k


a
b
c
d
g
h
i
k
0
0
a+b
c+d
g+h
i+k
A
B
0
a
b
c
d
g
h
i
k
EXOR(A,B)
EXOR

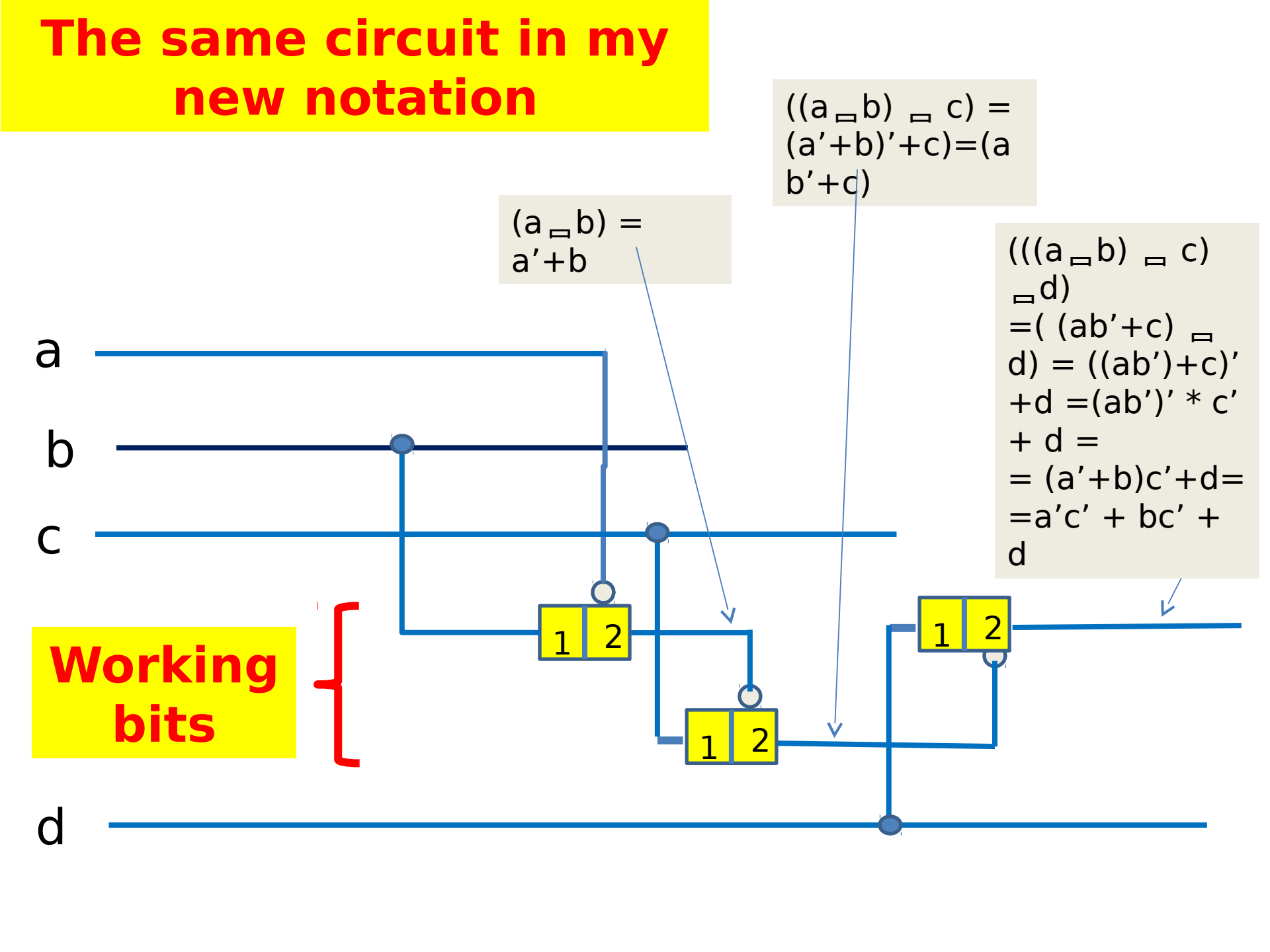


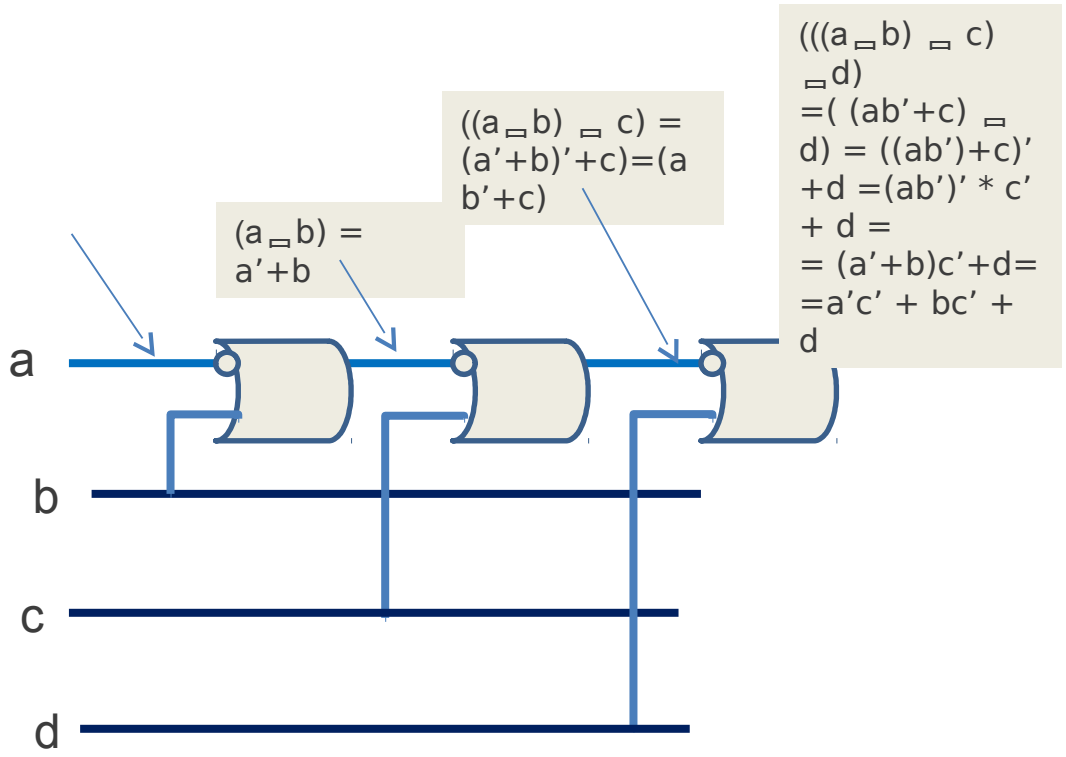
Memristors vs reversible

Example of circuit synthesis with negations along the bit line



The same circuit in my new notation





1	1	1	0
1	1	1	0
1	1	1	0
0	1	1	0

d

1	-	-	0
1	-	-	0
1	-	-	0
0	-	-	0

0	-	-	1
0	-	-	1
0	-	-	1
1	-	-	1

c

1
1
1
0

b

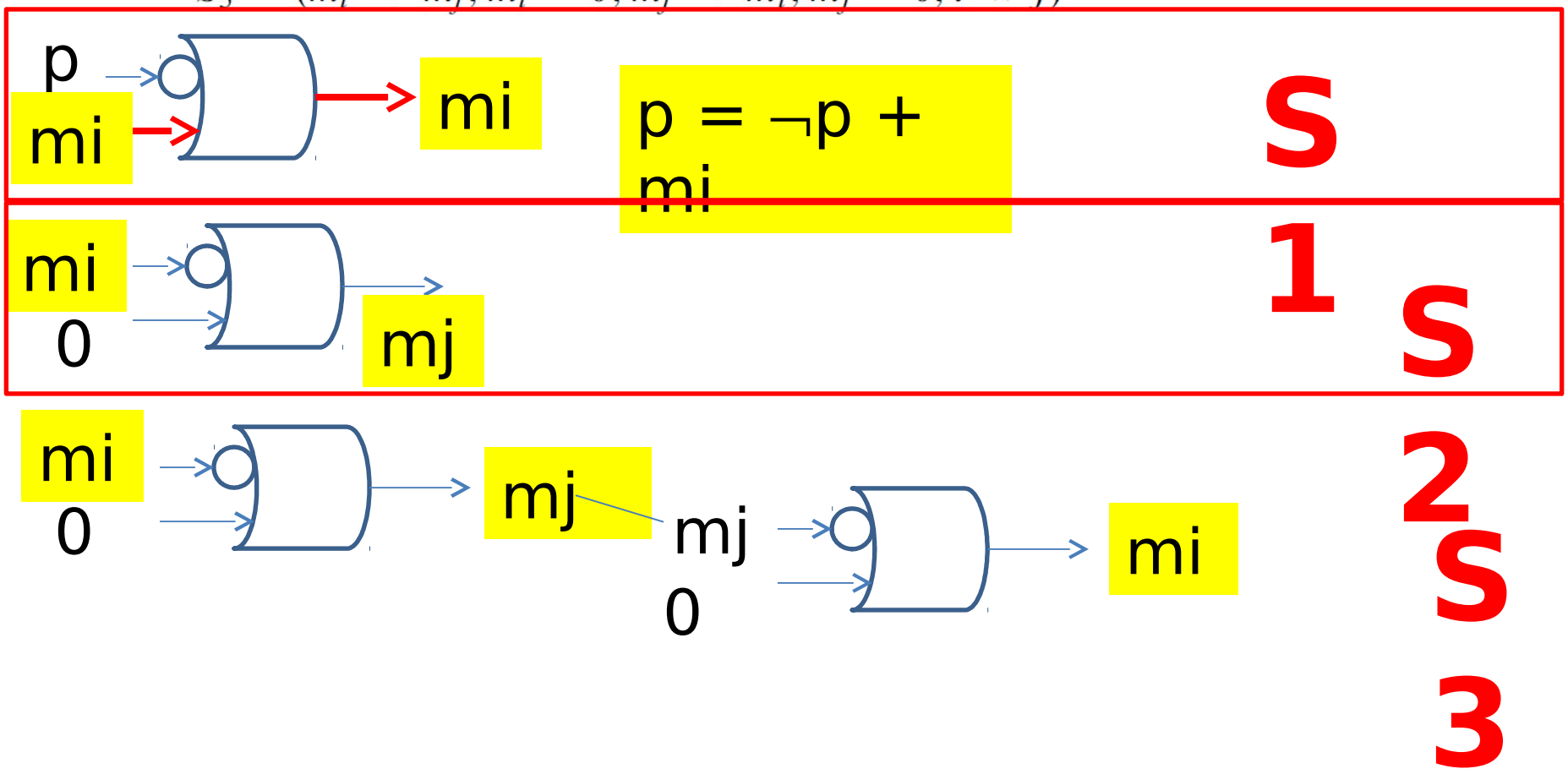
Graphical Illustration of the method with

Let $i, j \in \{1, 2\}, i \neq j$. In the following, implications from the inputs are allowed only into m_i, while m_j is used to maintain the result of the computational sequence calculated so far. We define three sequences of operations realisable at any stage of computation:

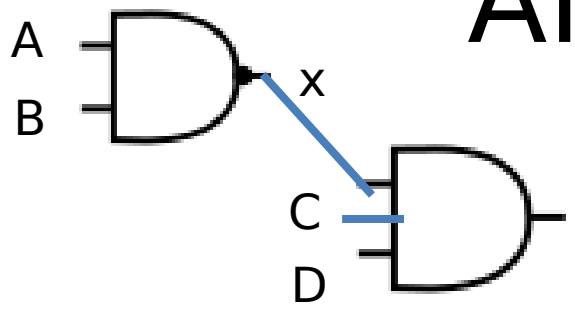
$$S_1 = (p \rightarrow m_i) \text{ for any } p \in P$$

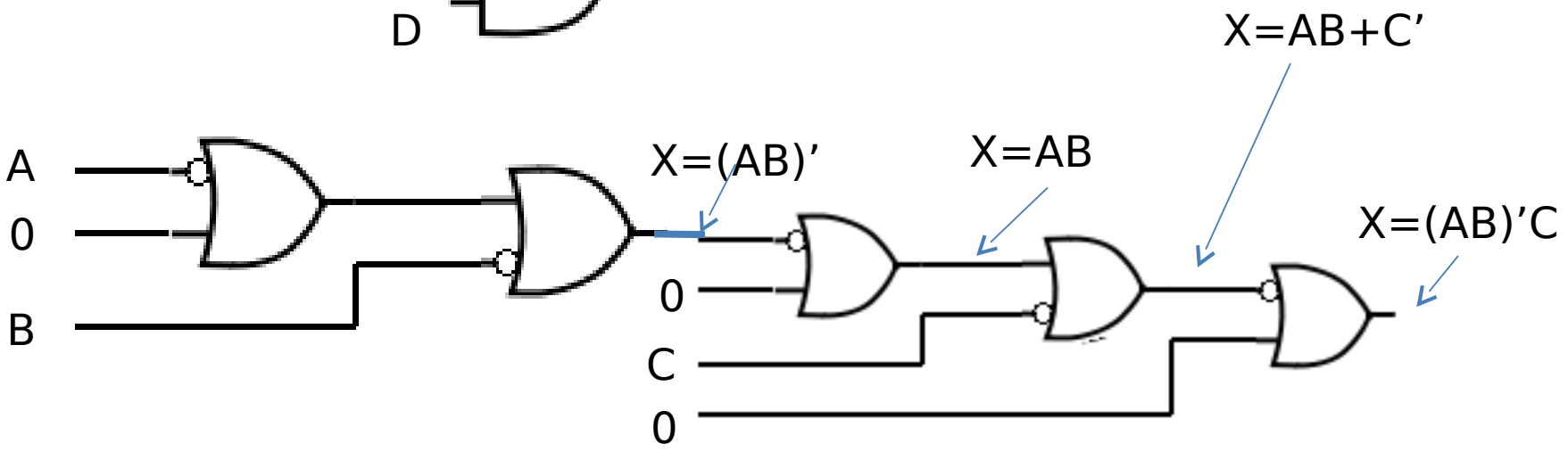
$$S_2 = (m_i \rightarrow m_j, m_i = 0)$$

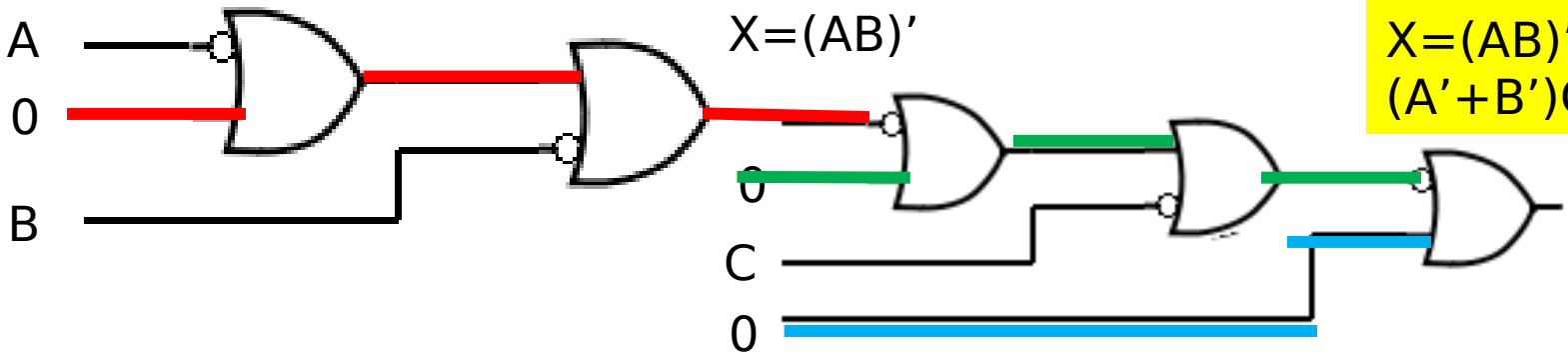
$$S_3 = (m_i \rightarrow m_j, m_i = 0, m_j \rightarrow m_i, m_j = 0, i \leftrightarrow j)$$



AND/OR

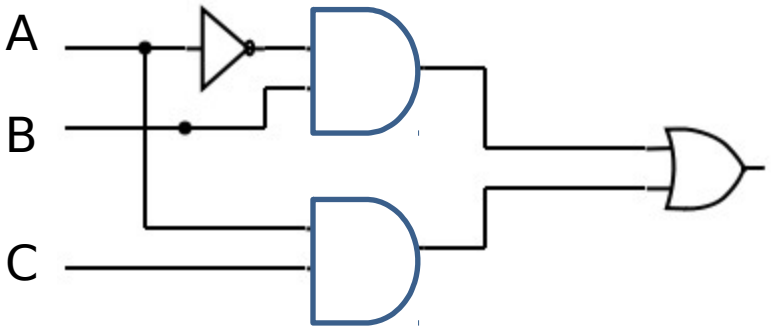




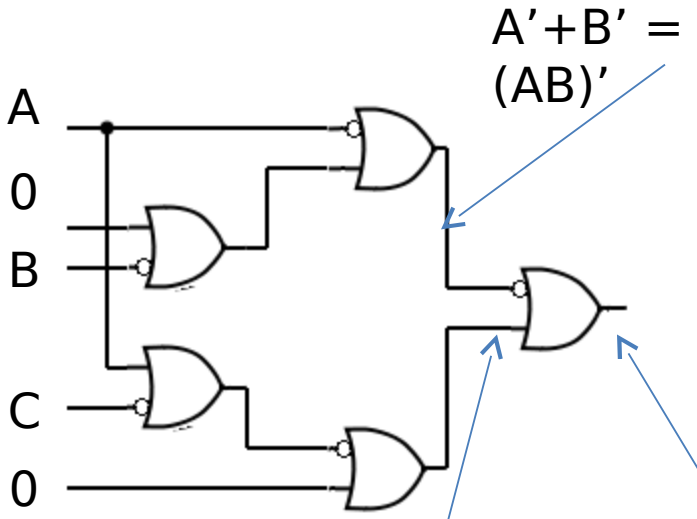


X=(AB)'C =
(A'+B')C

MUX



(A’B)+(AC)



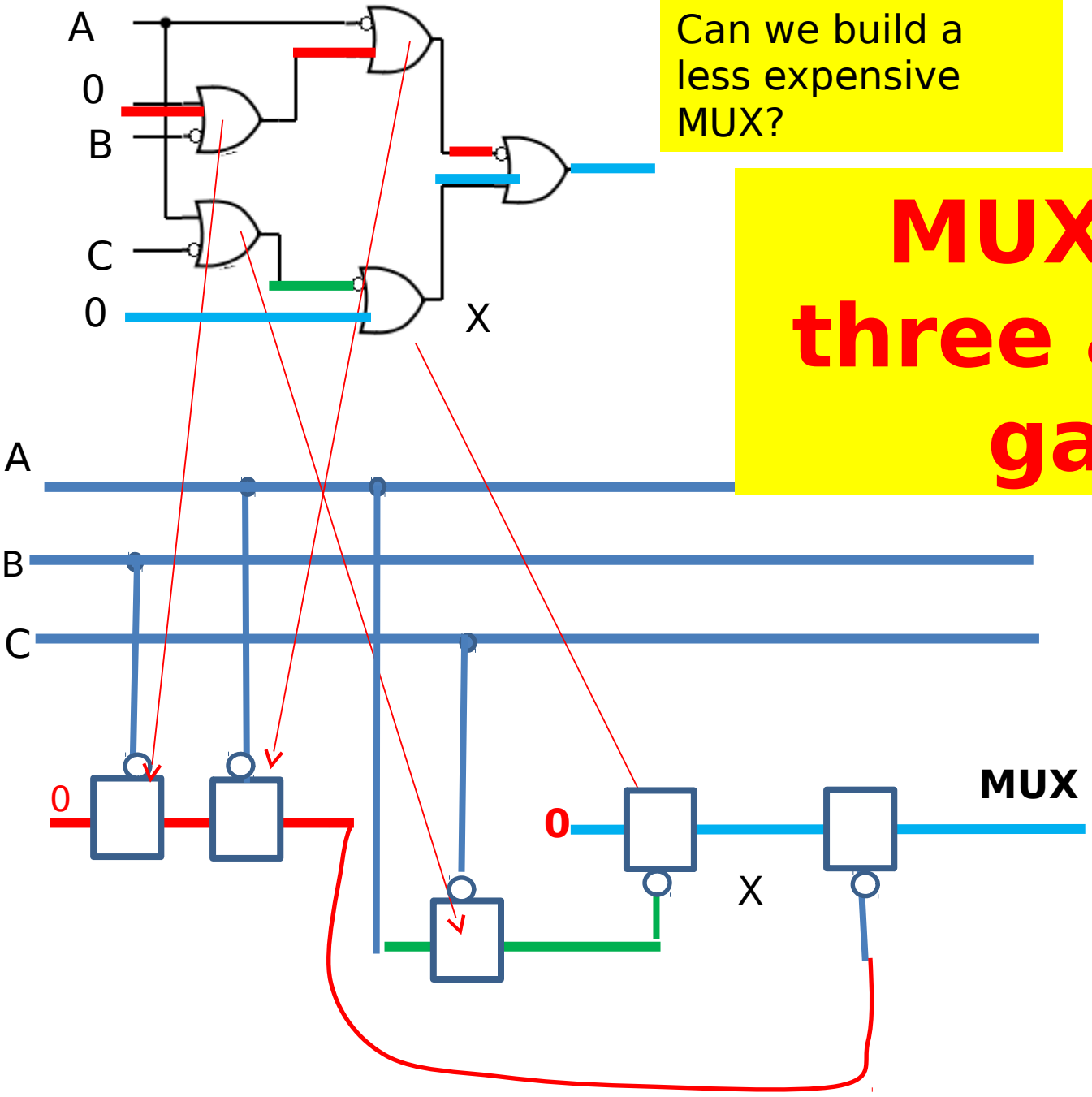
AB
+A’C

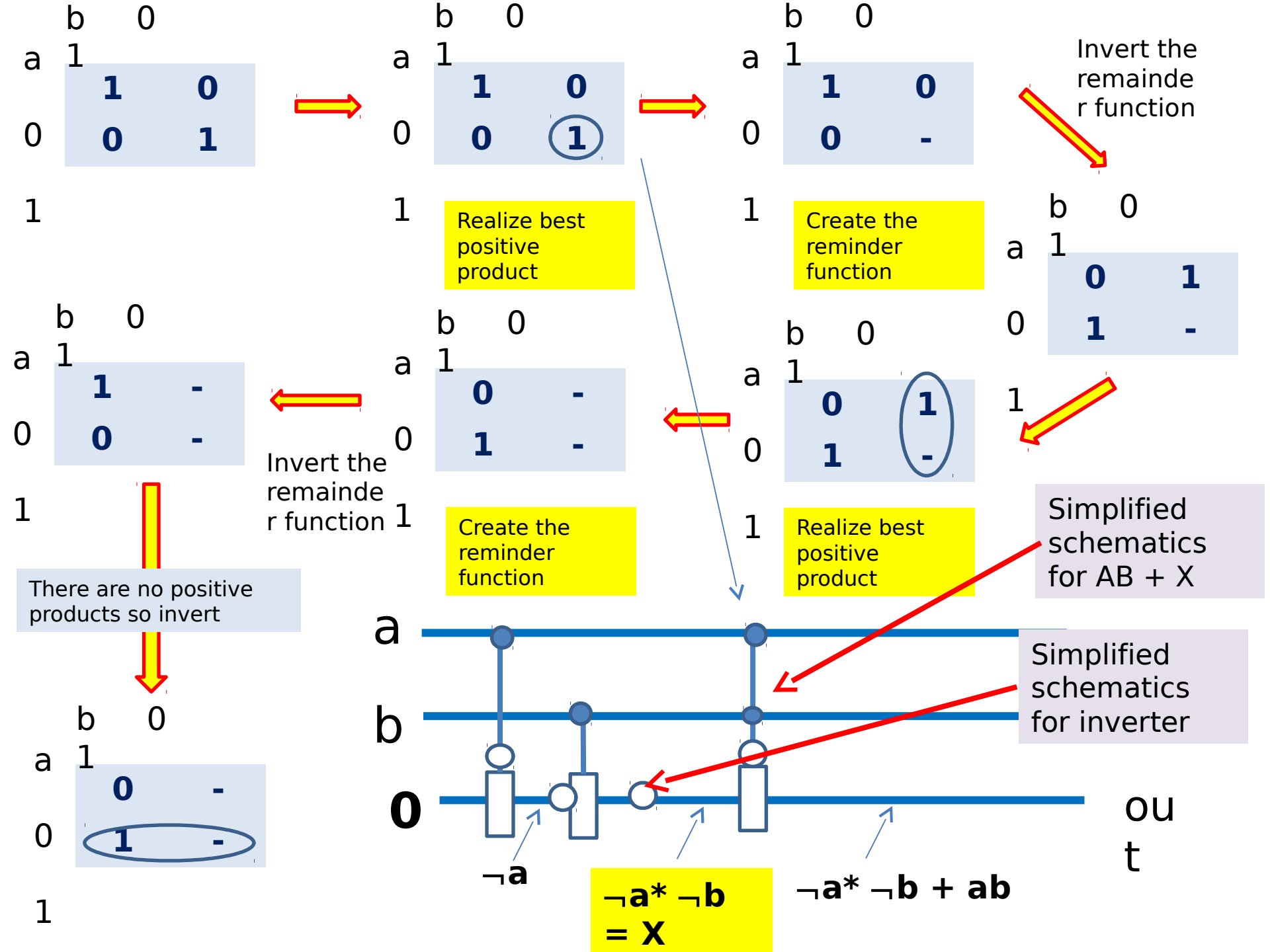
(A+C’)’= A’C

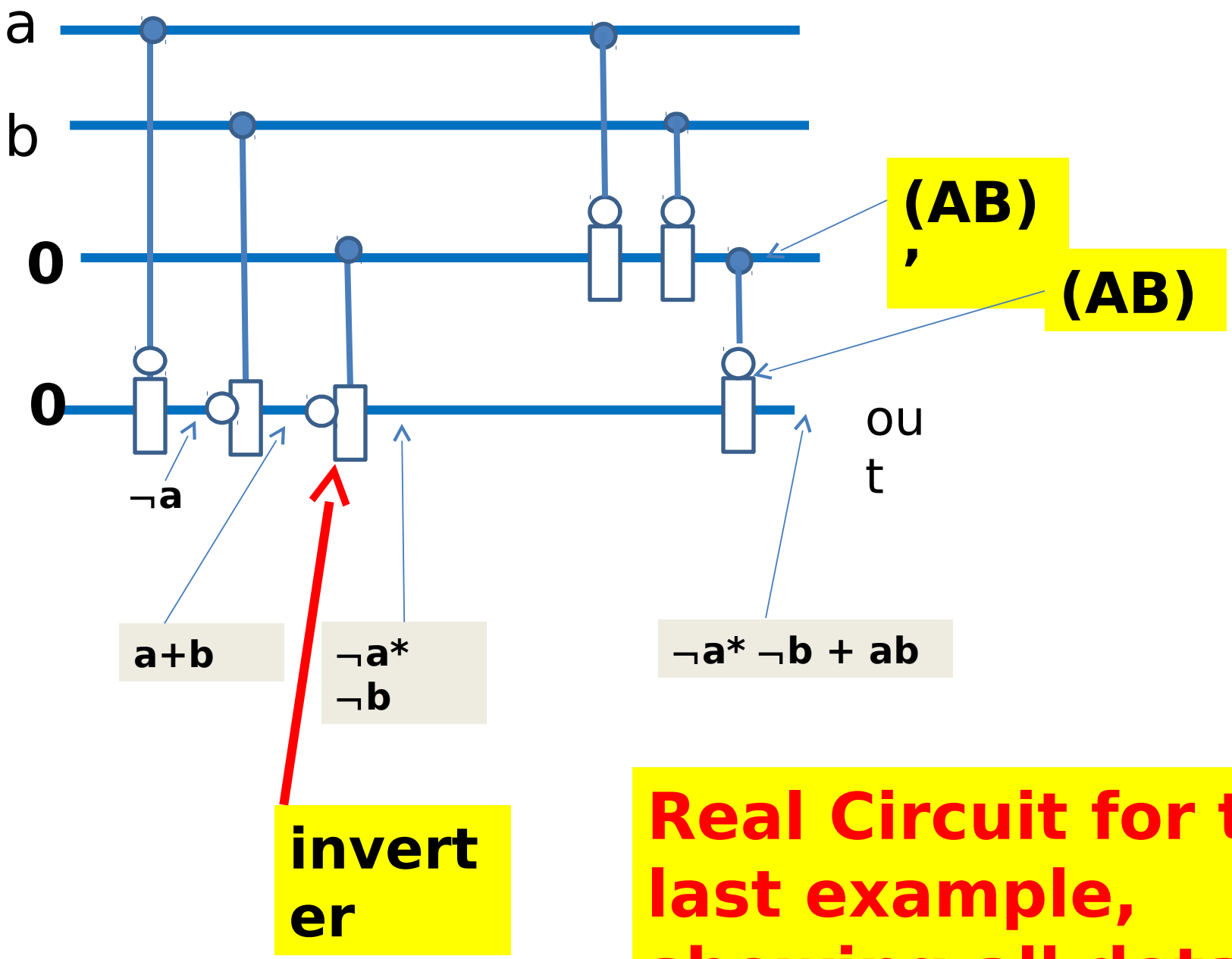
MUX

Can we build a less expensive MUX?

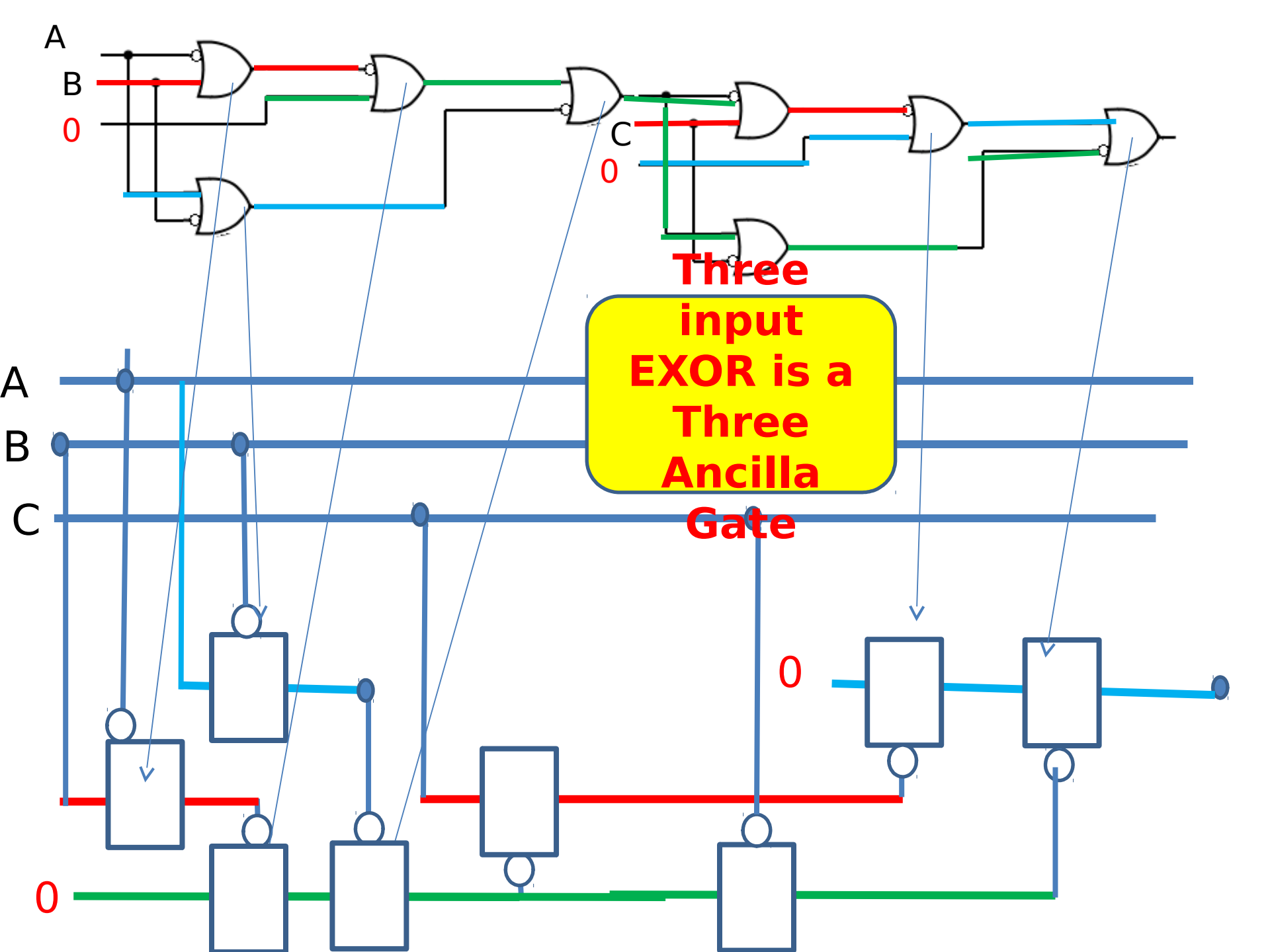
MUX is a three ancilla gate



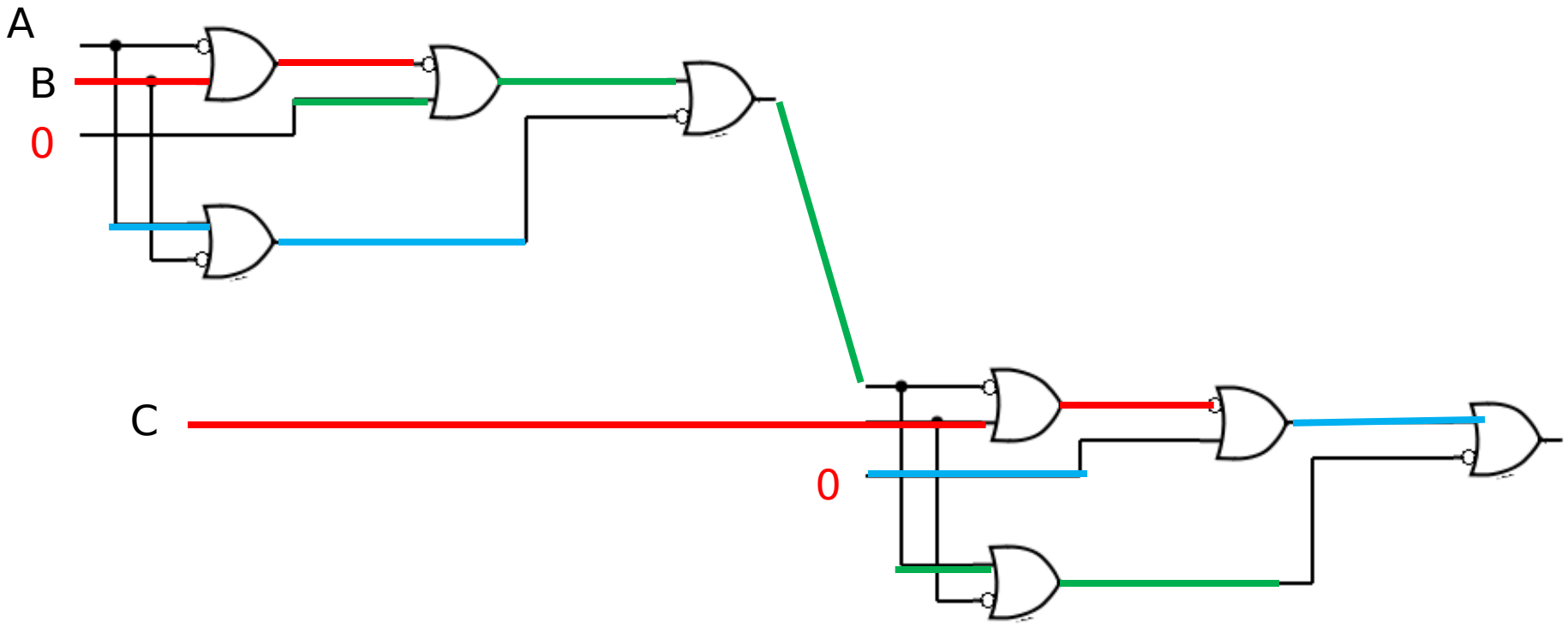

b 0
a 1
1 0
0 0 1
1
b 0
a 1
1 0
0 0 1
1
Realize best positive product
b 0
a 1
1 0
0 0 -
1
Create the reminder function
Invert the remainde r function
b 0
a 1
0 1
0 1 -
1
b 0
a 1
1 -
0 0 -
1
Invert the remainde r function
b 0
a 1
0 -
0 1 -
1
Create the reminder function
b 0
a 1
0 1
0 1 -
1
Realize best positive product
There are no positive products so invert
b 0
a 1
0 -
0 1 -
1
Simplified schematics for AB + X
Simplified schematics for inverter
a
b
0
out
¬a
¬a* ¬b = X
¬a* ¬b + ab



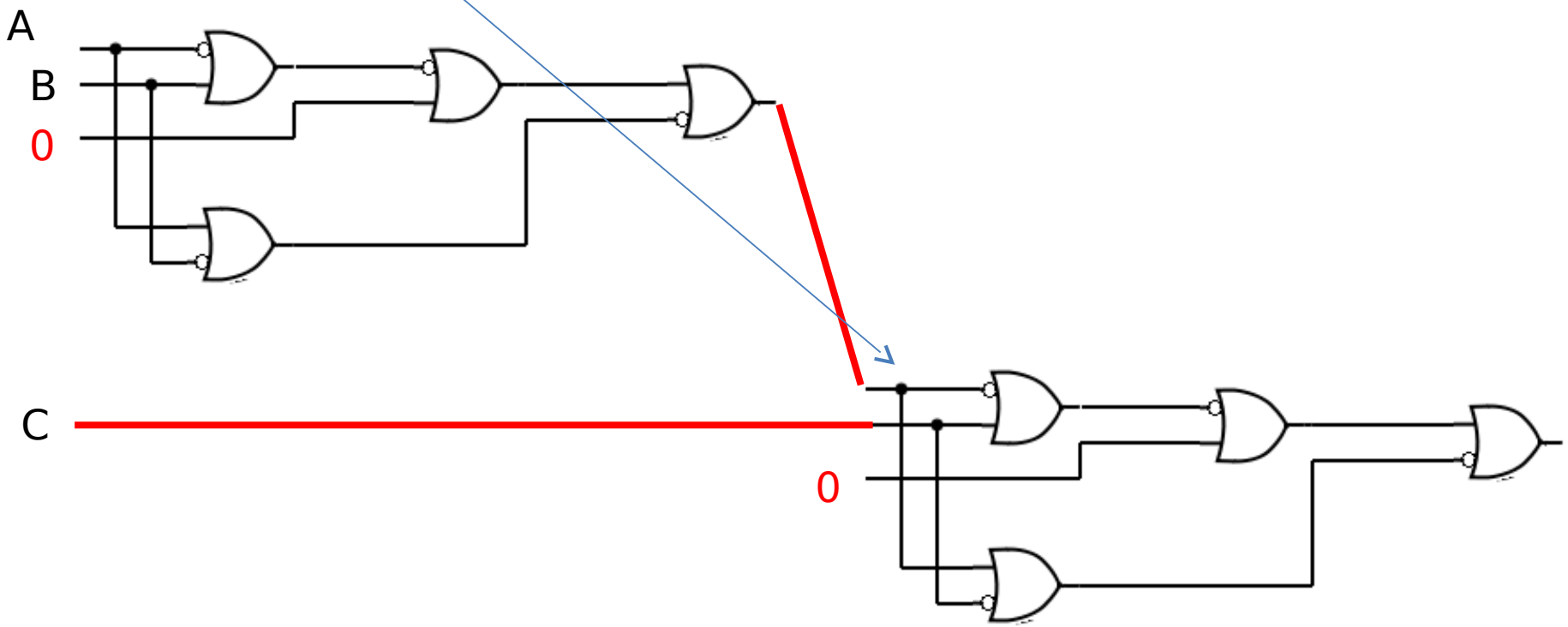
Real Circuit for the last example, showing all details


A
B
0
C
0
Three input EXOR is a Three Ancilla Gate
A
B
C
0
0

SYNTHESIS WITH EXORS WITH NO LIMIT ON NUMBER OF ANCILLA BITS



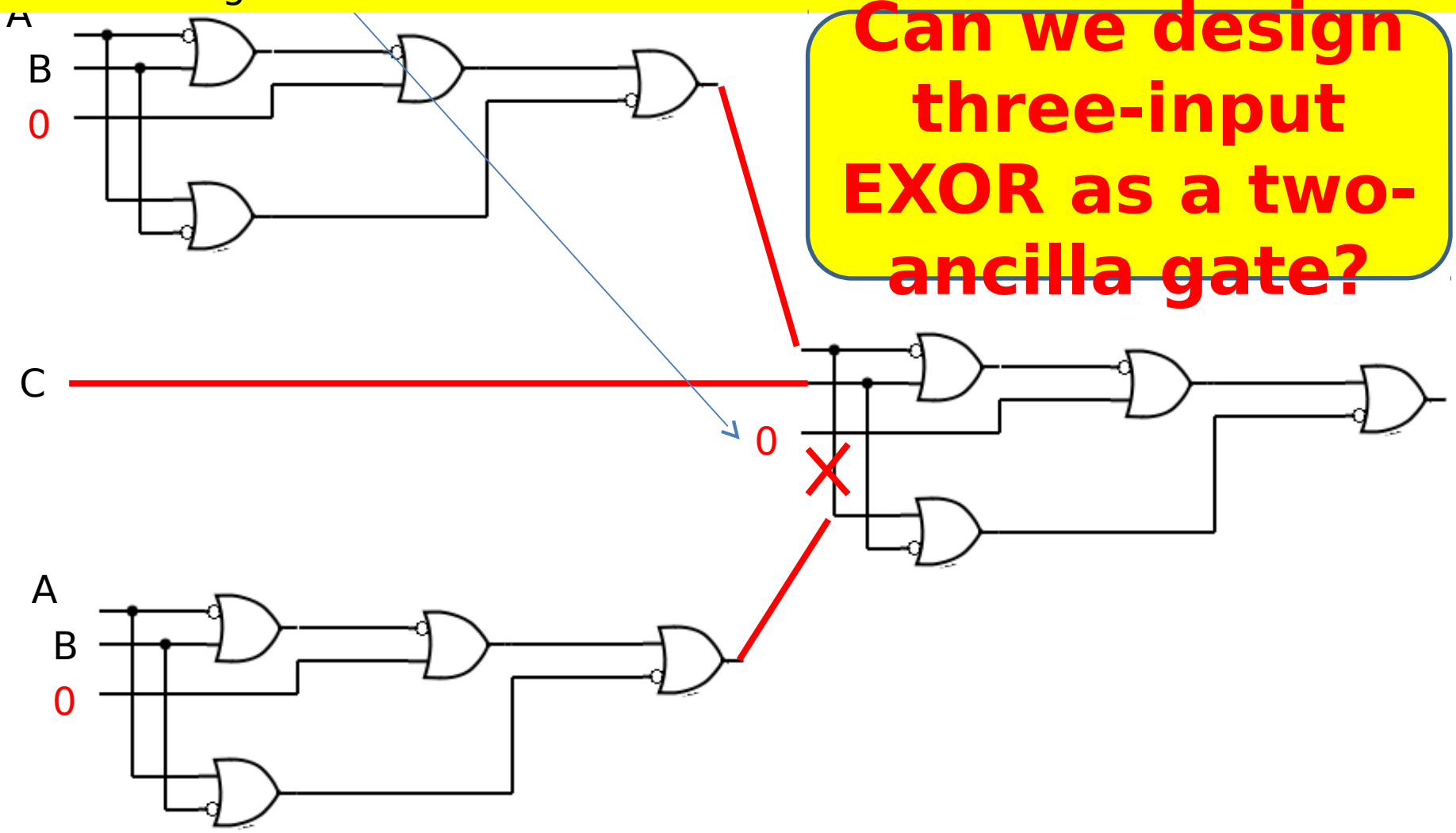
1. In their design there are 21 memristors
2. We need 2 two-input exors. Exor has 4 gates.
3. 2*4 = 8. Our design has 8 gates not 21.
4. But we have fan-out of two



- **Conclusion. When we want to decrease the number of ancilla bits we need:**

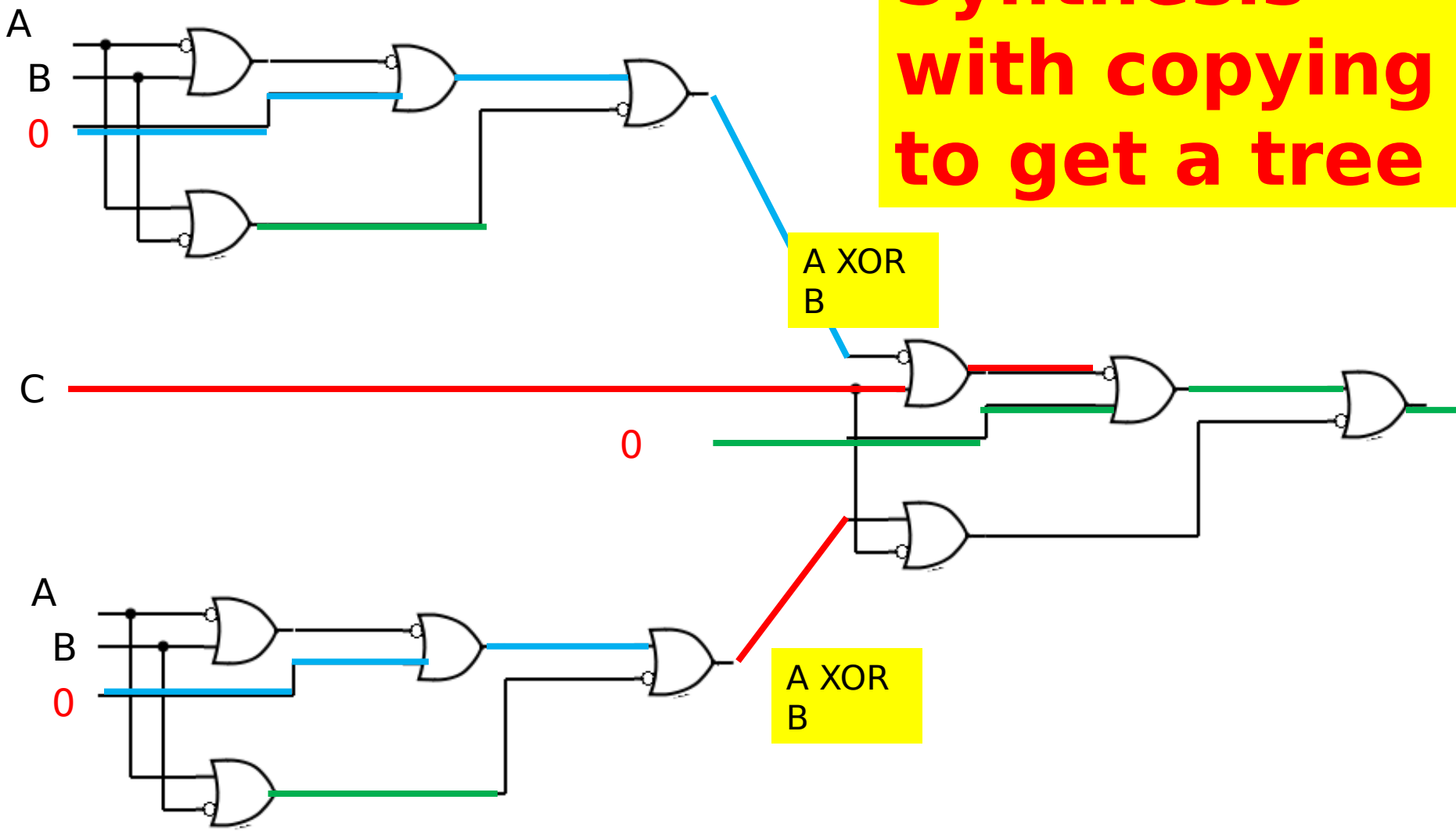
1. Methods with many exors are not good as they would require copying elements.
2. Methods are good if they

1. But we have fan-out of two
2. So we replicate the first EXOR , and we have only 12 gates, not 21 as in their design

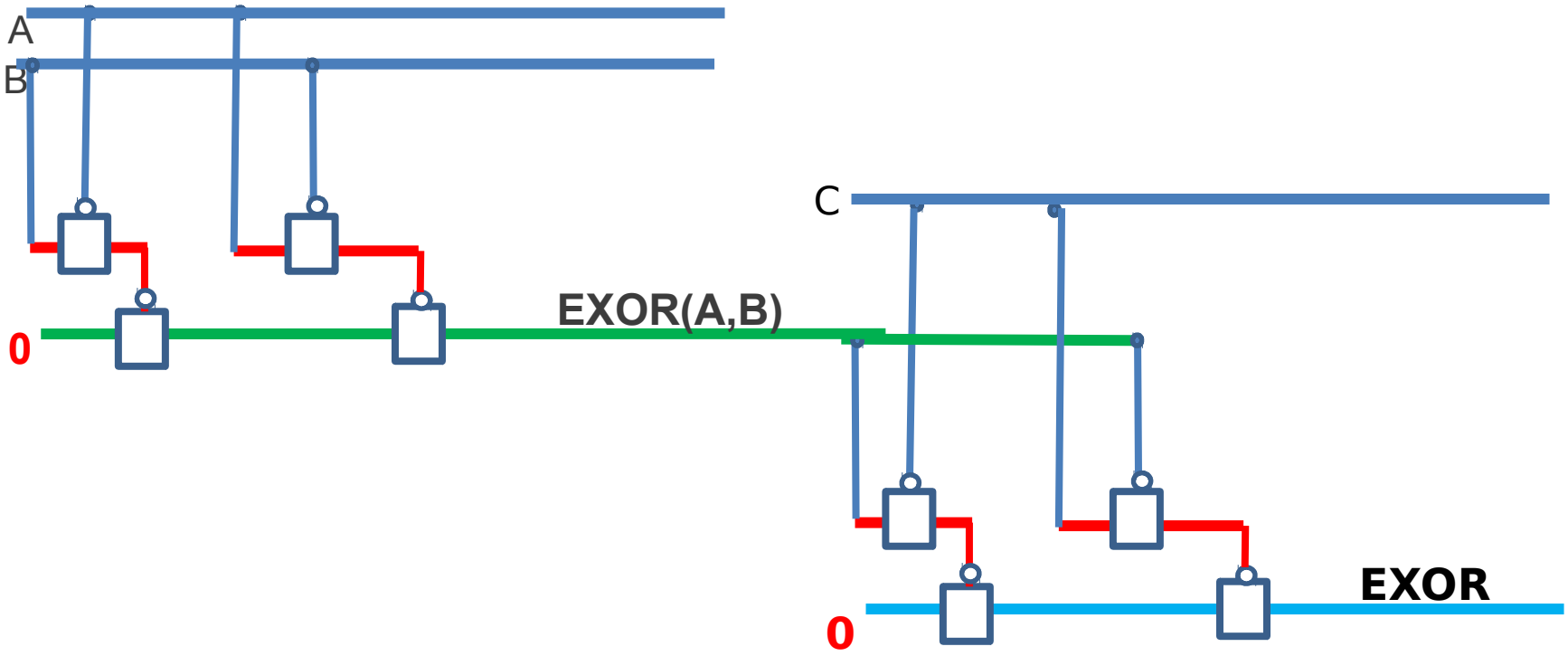


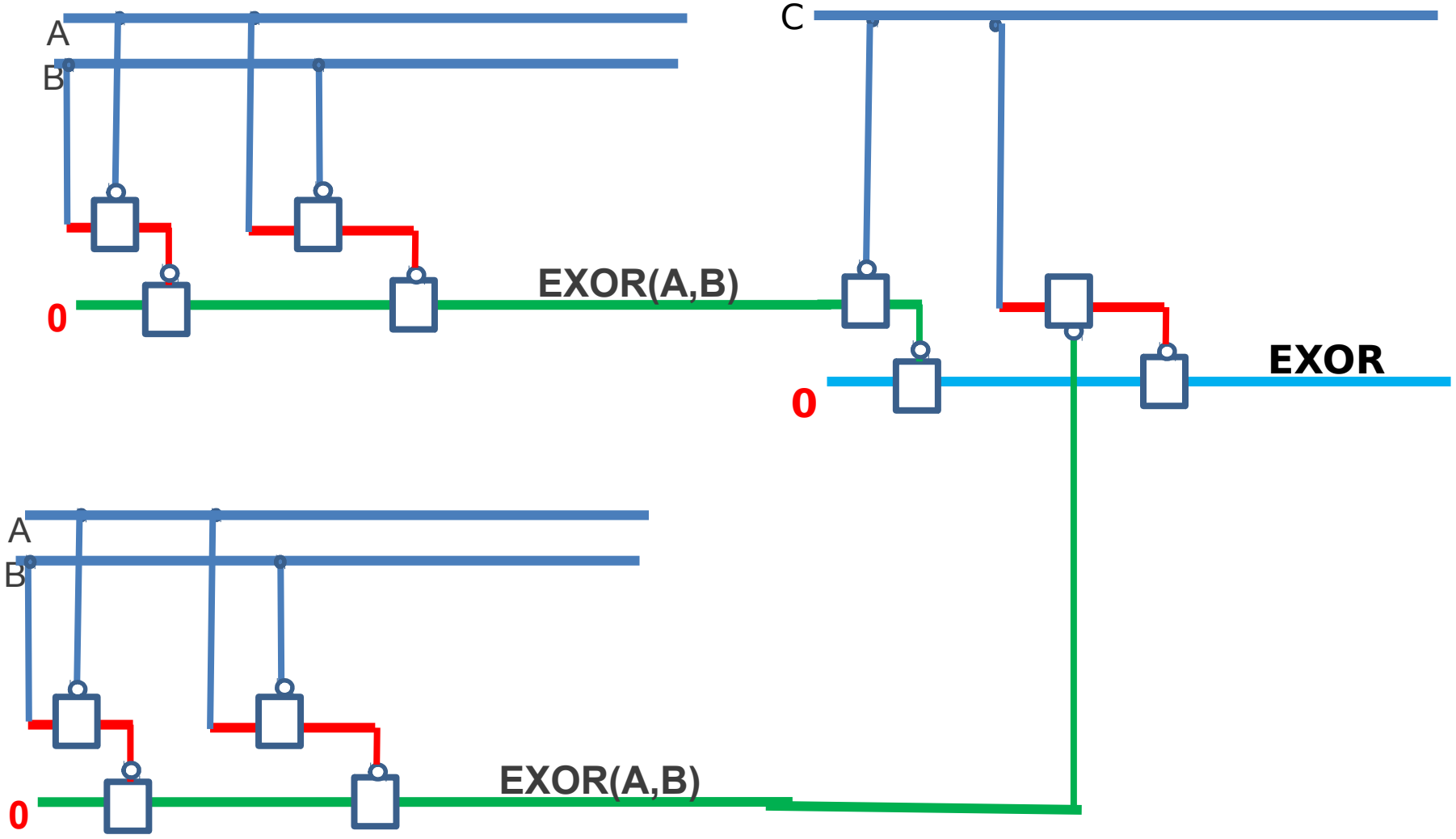
Can we design three-input EXOR as a two-ancilla gate?

Synthesis with copying to get a tree



Example of coloring to show that exor of three inputs must have three ancilla,
all in red are one ancilla but blue and green cannot be combined




A
B
0
EXOR(A,B)
C
0
EXOR
A
B
0
EXOR(A,B)