Memristive stateful logic

Eero Lehtonen[1], Jussi Poikonen[2]

[1]University of Turku, Finland

[2]Aalto University, Finland

January 22, 2014

Outline

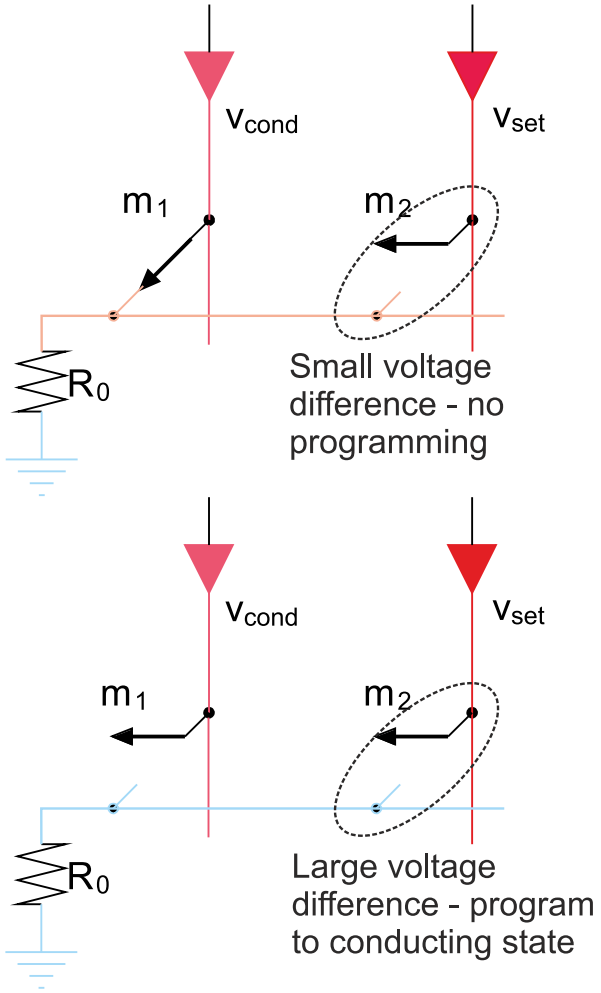
1. *Basic principle of memristive stateful logic*
2. *Generalized memristive stateful logic*
3. *Parallelization to a crossbar*
4. *Segmentation*

Introduction

- In the following, we discuss *memristive stateful logic*
- This means, in essence, logical operations between the persistent binary states of memristors
- This talk is based mainly on our chapter *Memristive stateful logic* in the forthcoming book *Memristor Networks* (Springer).

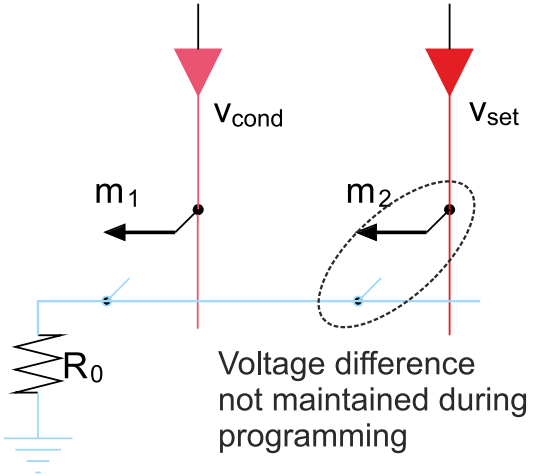
Principle of elementary stateful logic operations

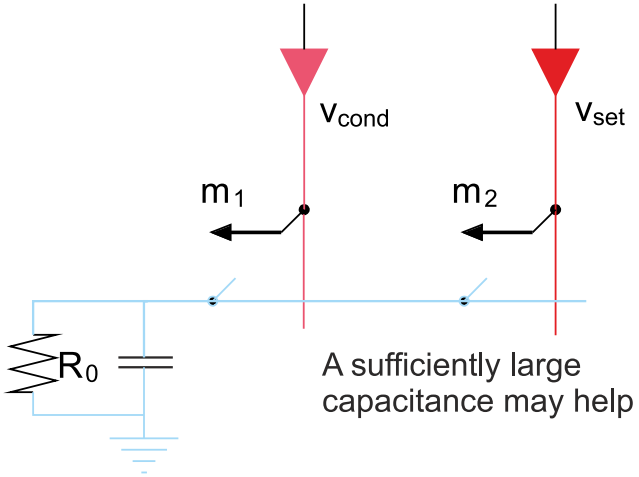
- Assume a circuit where two vertical (nano)wires are connected by memristors to a horizontal (nano)wire.
- The depicted voltages are chosen as follows:
  $0 < v_{\text{cond}} < V_{\text{TH}}$,
  $v_{\text{set}} > V_{\text{TH}}$,
  $v_{\text{set}} - v_{\text{cond}} < V_{\text{TH}}$, where V_{TH} is the programming threshold voltage of the memristors.



Practical considerations

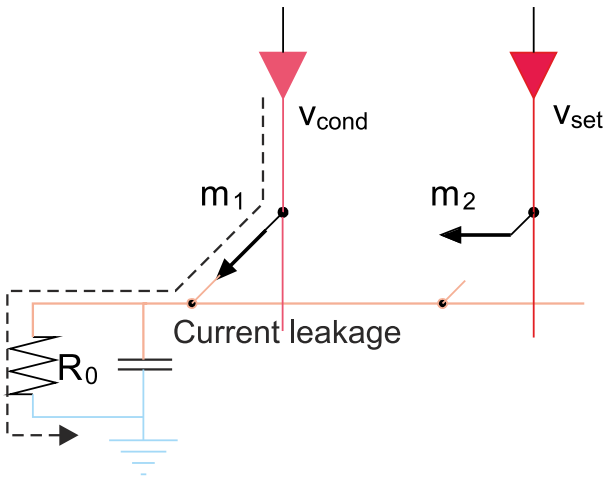
- The resistor R_0 enables voltage division, where the voltage at the horizontal wire varies according to the memristances of m_1 and m_2.
- This voltage will also change when m_2 is programmed, possibly interrupting the programming.
- Adding capacitance may help, but will reduce operation speed.

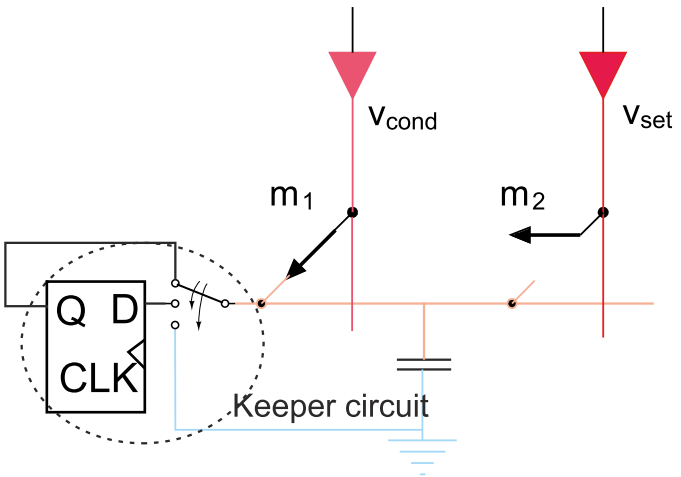




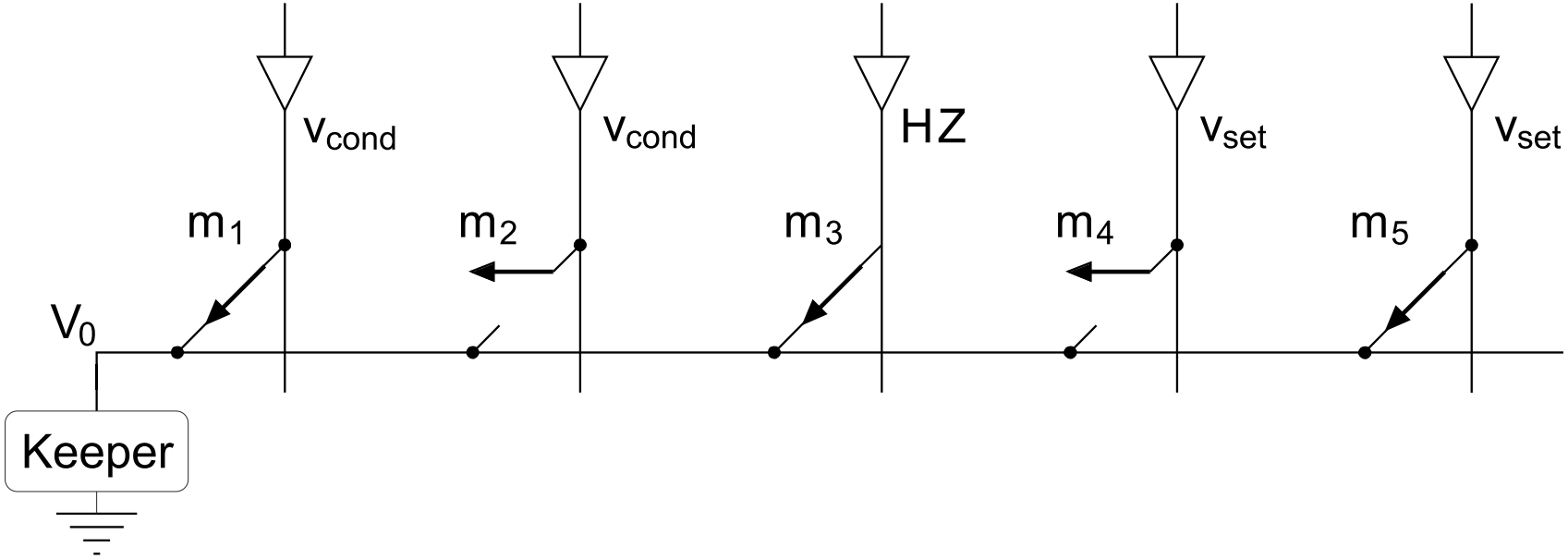
Practical considerations

- Another problem with passive voltage division is that there is a constant current path to ground.
- Using an active CMOS *keeper circuit* will reduce energy consumption, but also increase area overhead.
- With a keeper circuit, the operation is divided into a read phase and a programming phase.





Generalized stateful operations



- The figure shows a generalized stateful logic operation S yielding $m_4 = S(\mathrm{OR}(m_1, m_2), m_4)$ and $m_5 = S(\mathrm{OR}(m_1, m_2), m_5)$.
- The vertical wires of memristors not participating are connected to drivers in high impedance state

Obtainable logical operations

$p = m_{i1} \vee \ldots \vee m_{ik}$	$q = m_j$	$p \rightarrow q$	$p \not\leftarrow q$	$p \wedge q$	$p \vee q$
0	0	1	0	0	0
0	1	1	1	0	1
1	0	0	0	0	1
1	1	1	0	1	1

Table: Truth tables of the logical operations available with generalized memristive stateful logic. Note that $p \rightarrow q \equiv \mathrm{OR}(\neg p, q)$ and $p \not\leftarrow q \equiv \mathrm{AND}(\neg p, q)$.

- It can also be assumed that any memristor can be reset at will
- Any Boolean expression can be synthesized in many ways using combinations of these operations

Parallel memristive stateful logic in a crossbar

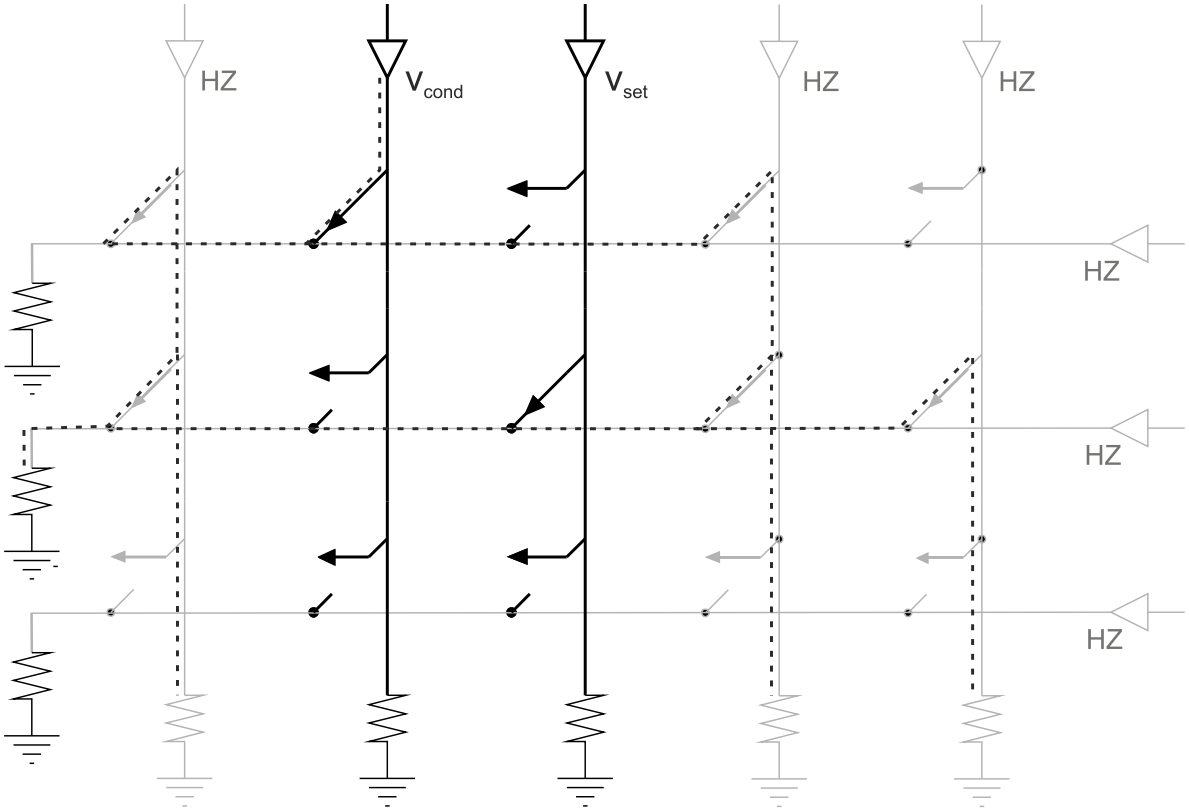
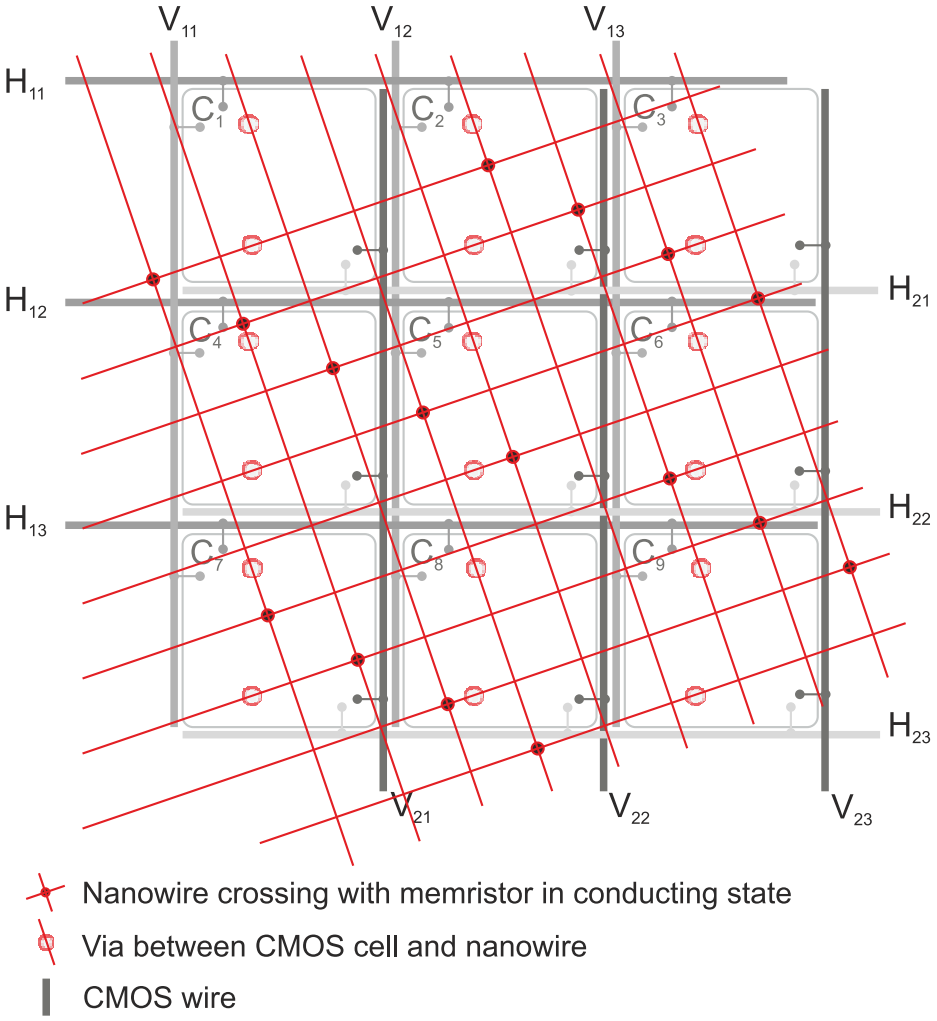


Figure: A stateful logic operation performed in parallel on all rows over the second and third memristors from the left.

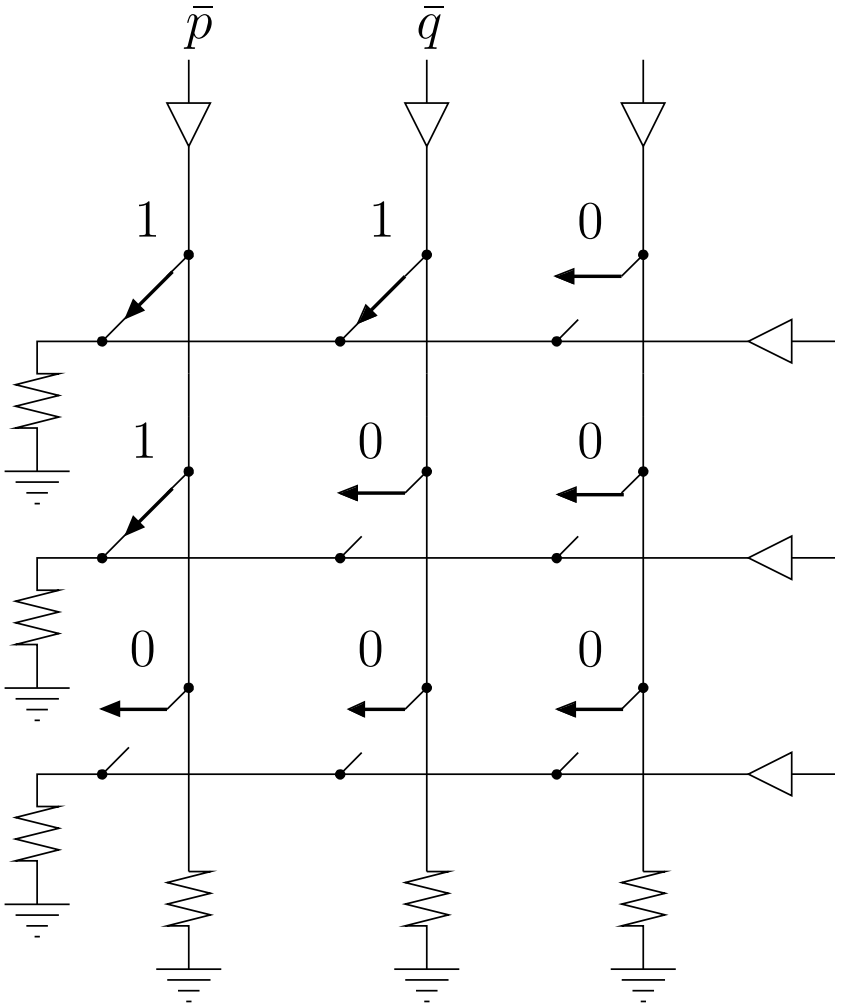
The CMOL solution to implementing parallel logic



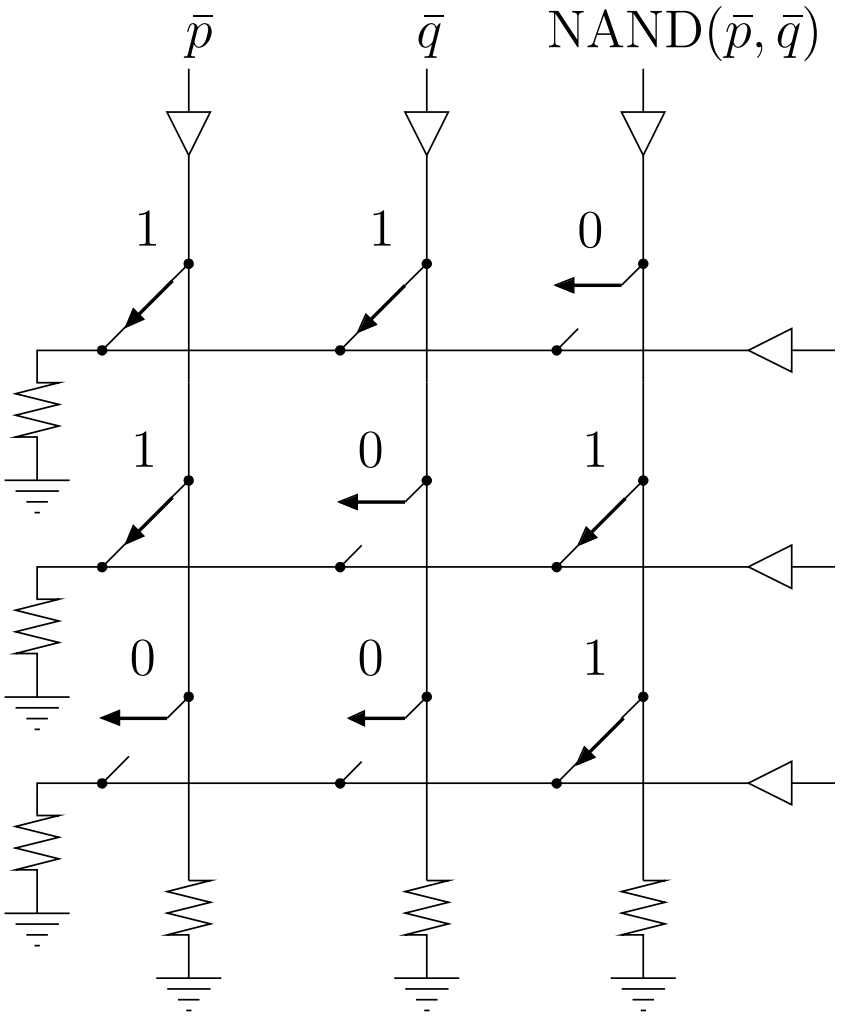
Parallel stateful logic

- In the following, parallel column operations are presented. Row operations are performed similarly, using reverse polarities of voltages
- To avoid sneak current paths, rectifying memristors are assumed (only positive current through memristors)
- This limits the availability of operations to implication and converse non-implication

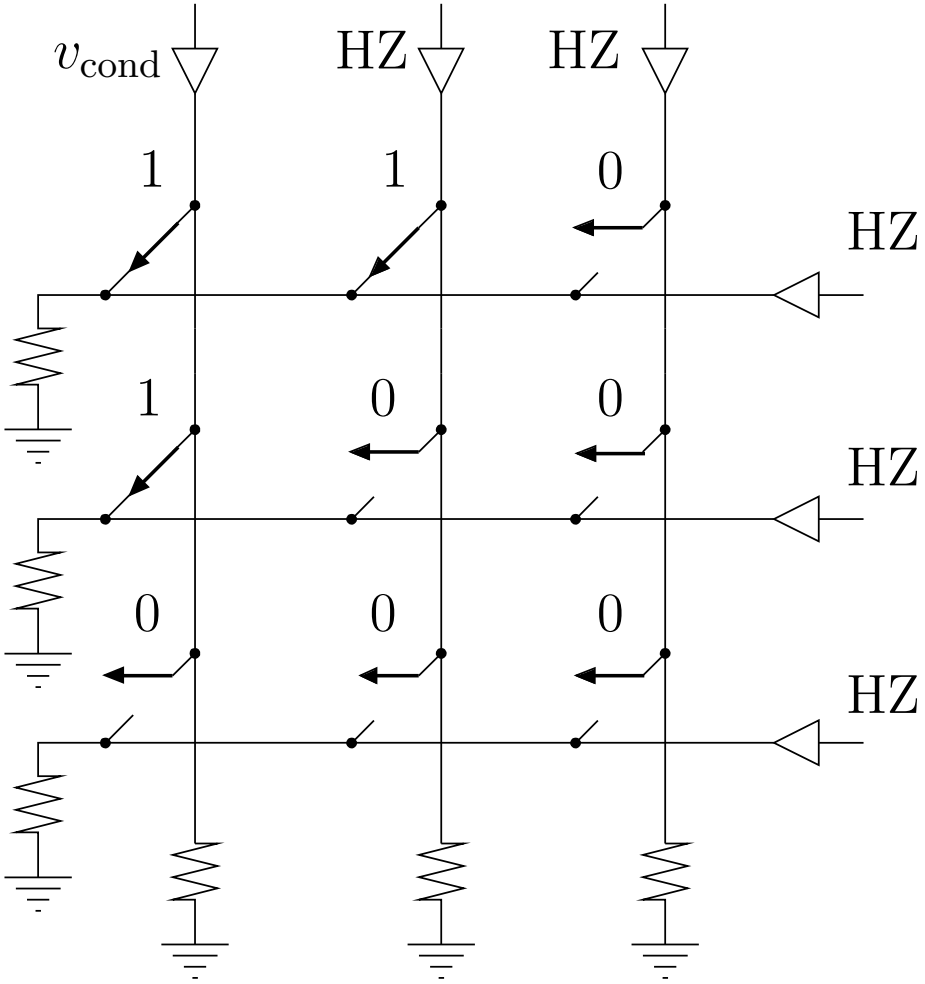
NAND of columns



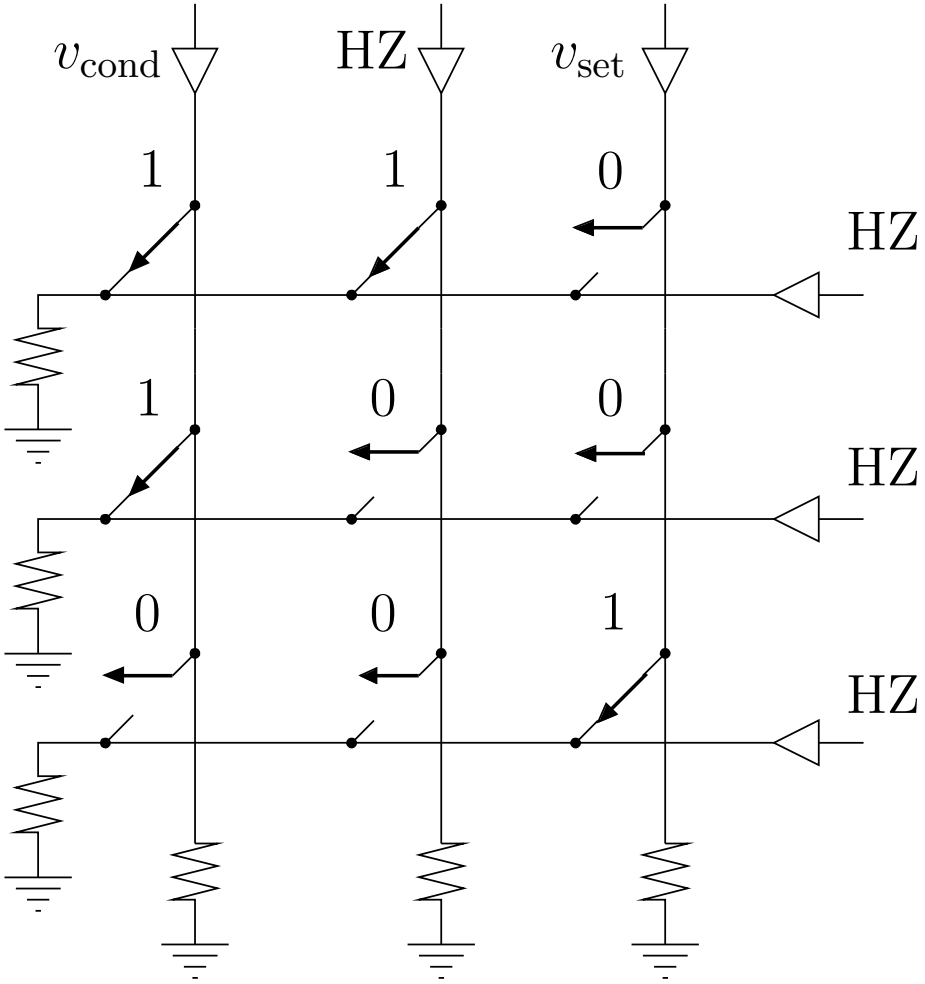
NAND of columns



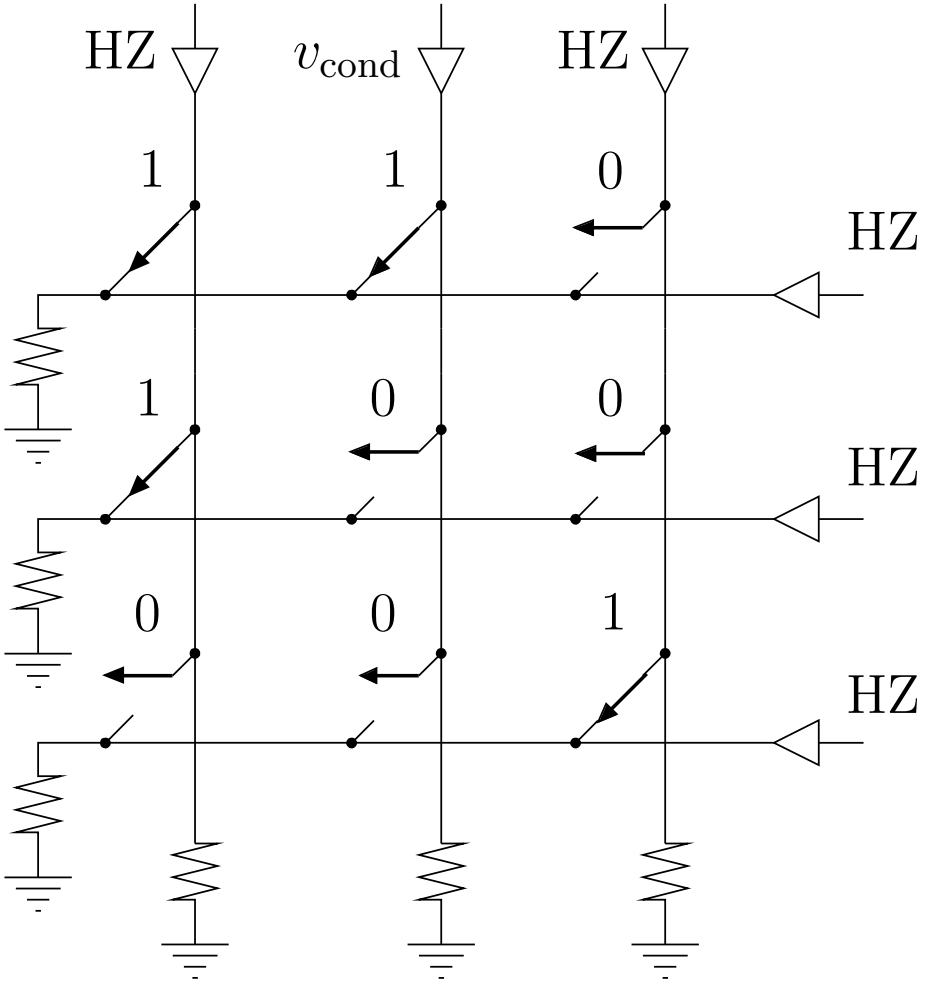
NAND (1st implication)



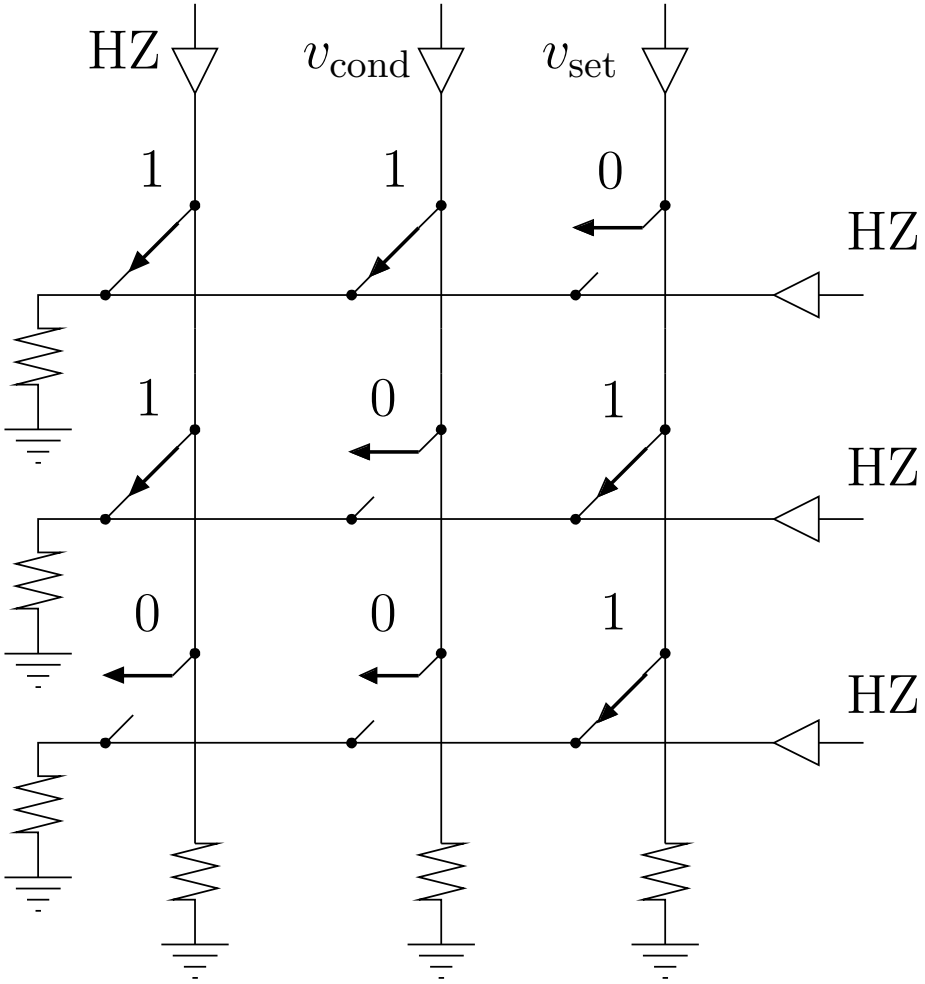
NAND (1st implication)



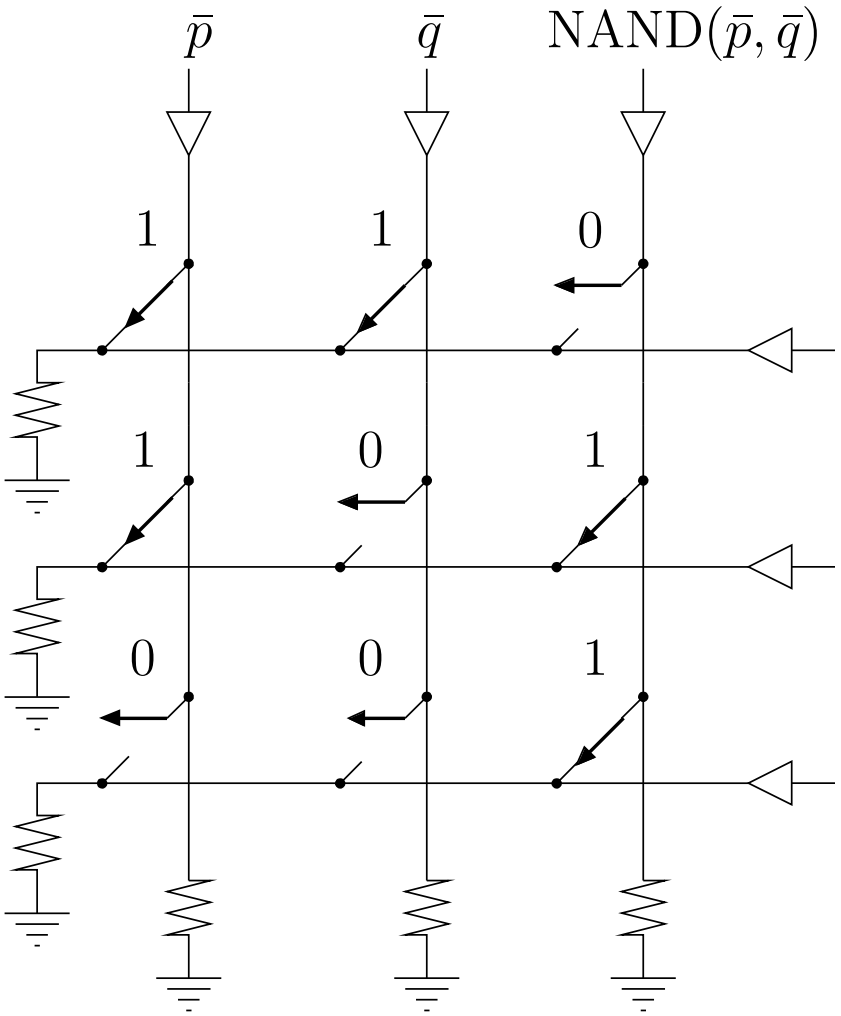
NAND (2nd implication)



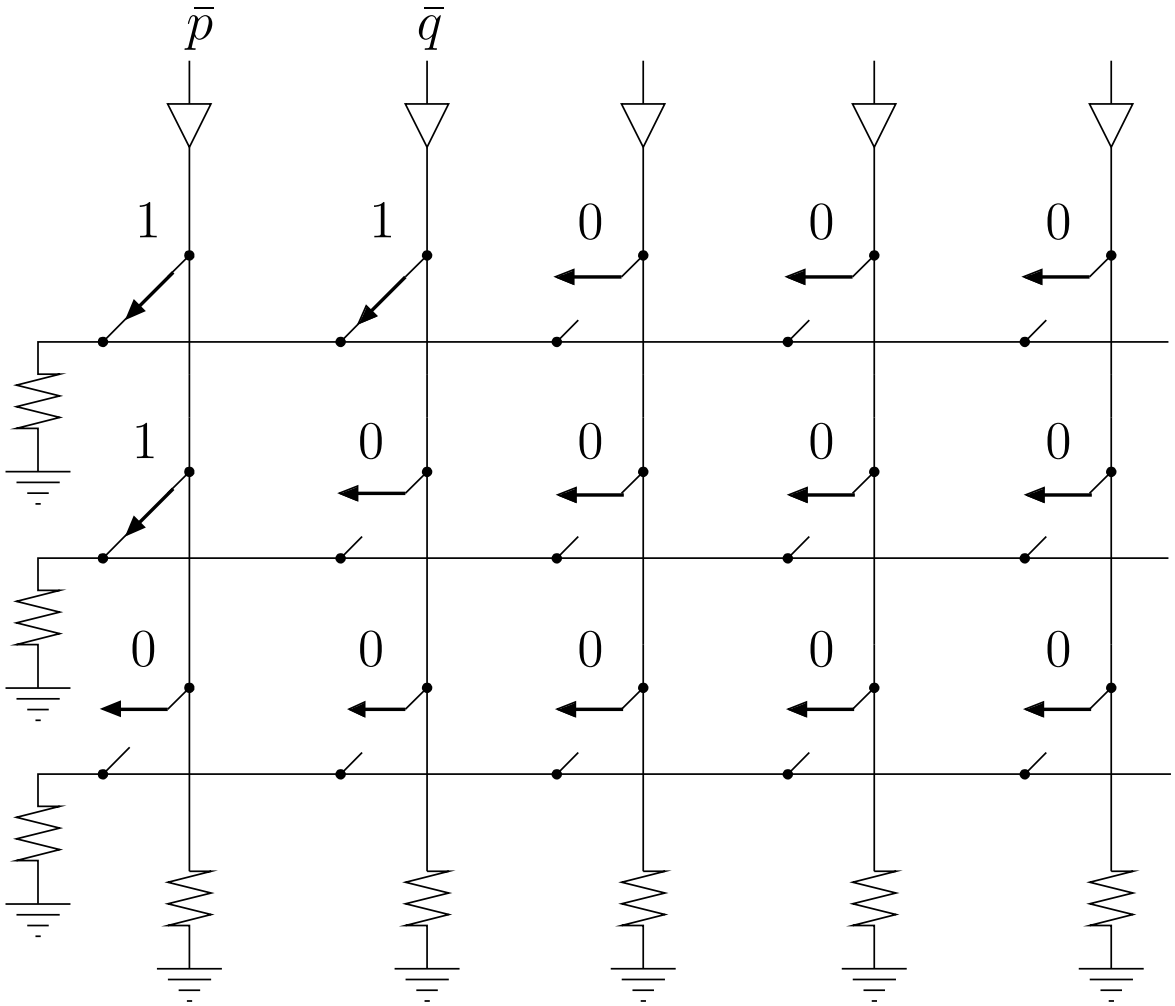
NAND (2nd implication)



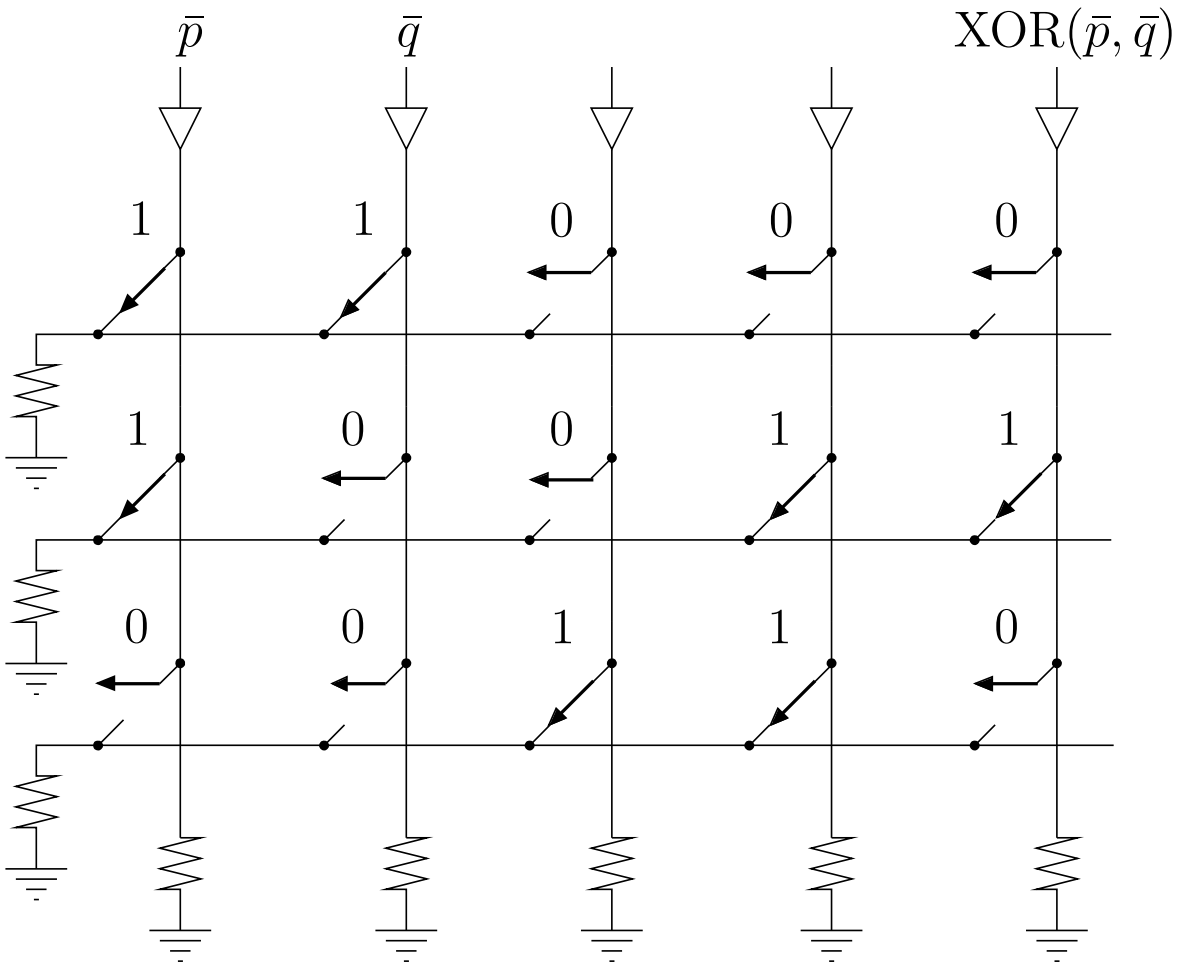
NAND of columns



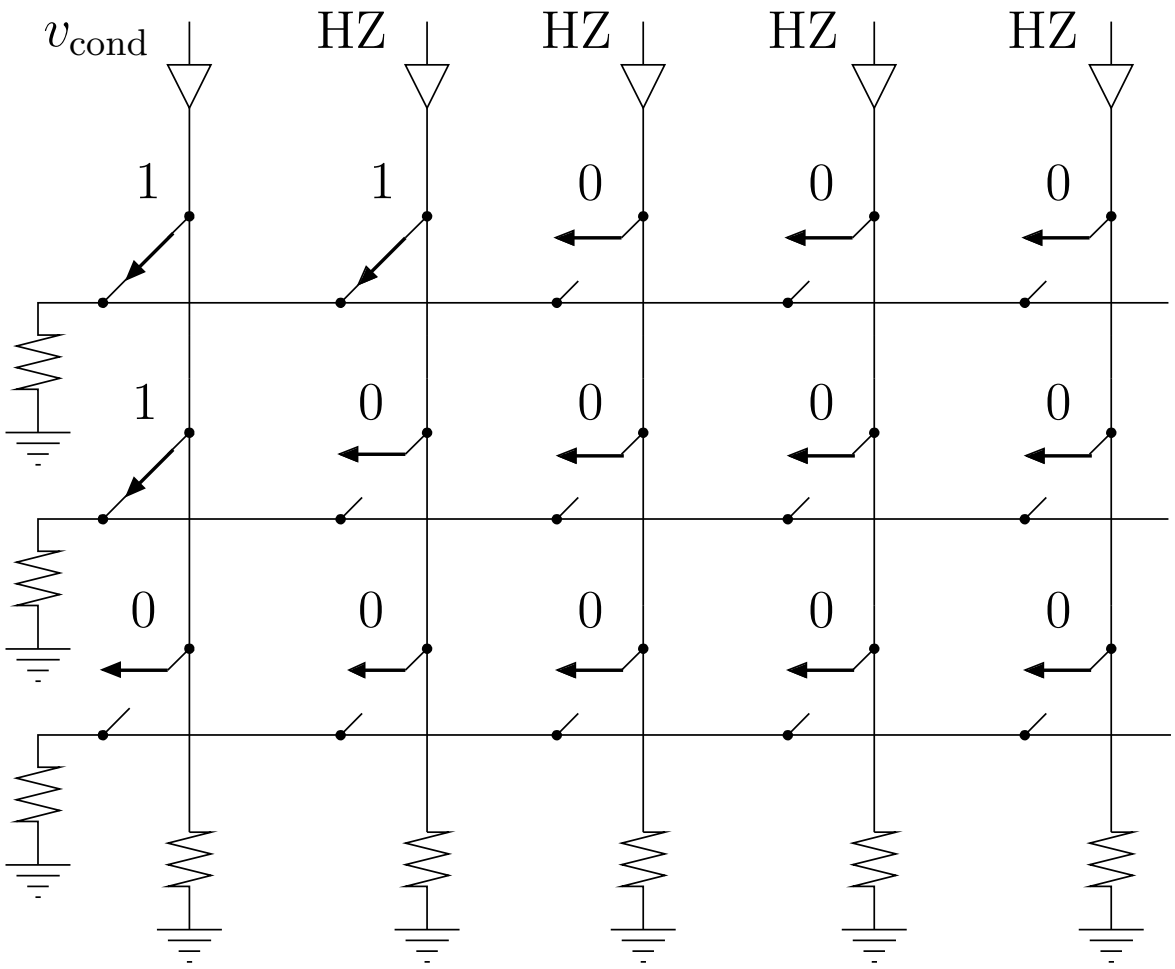
XOR of columns



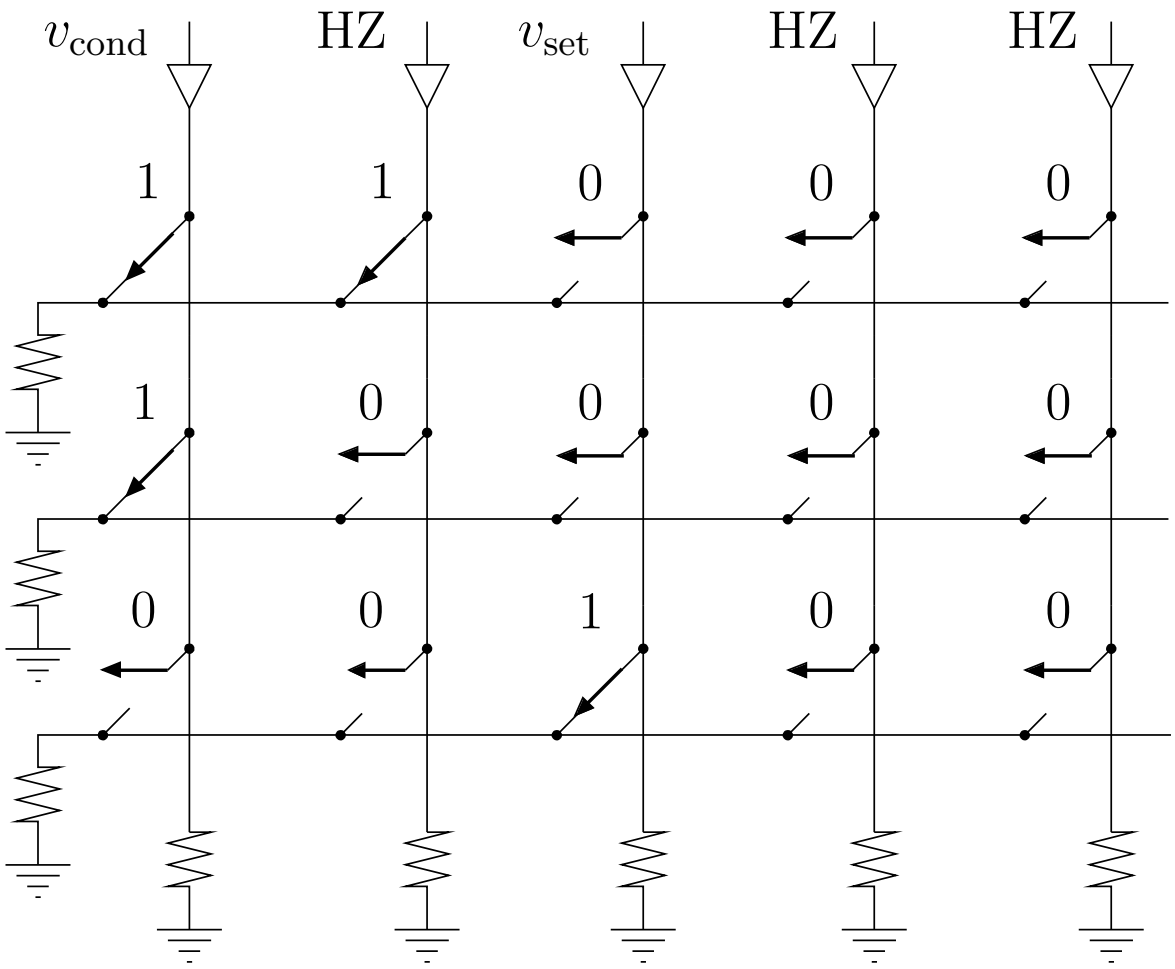
XOR of columns



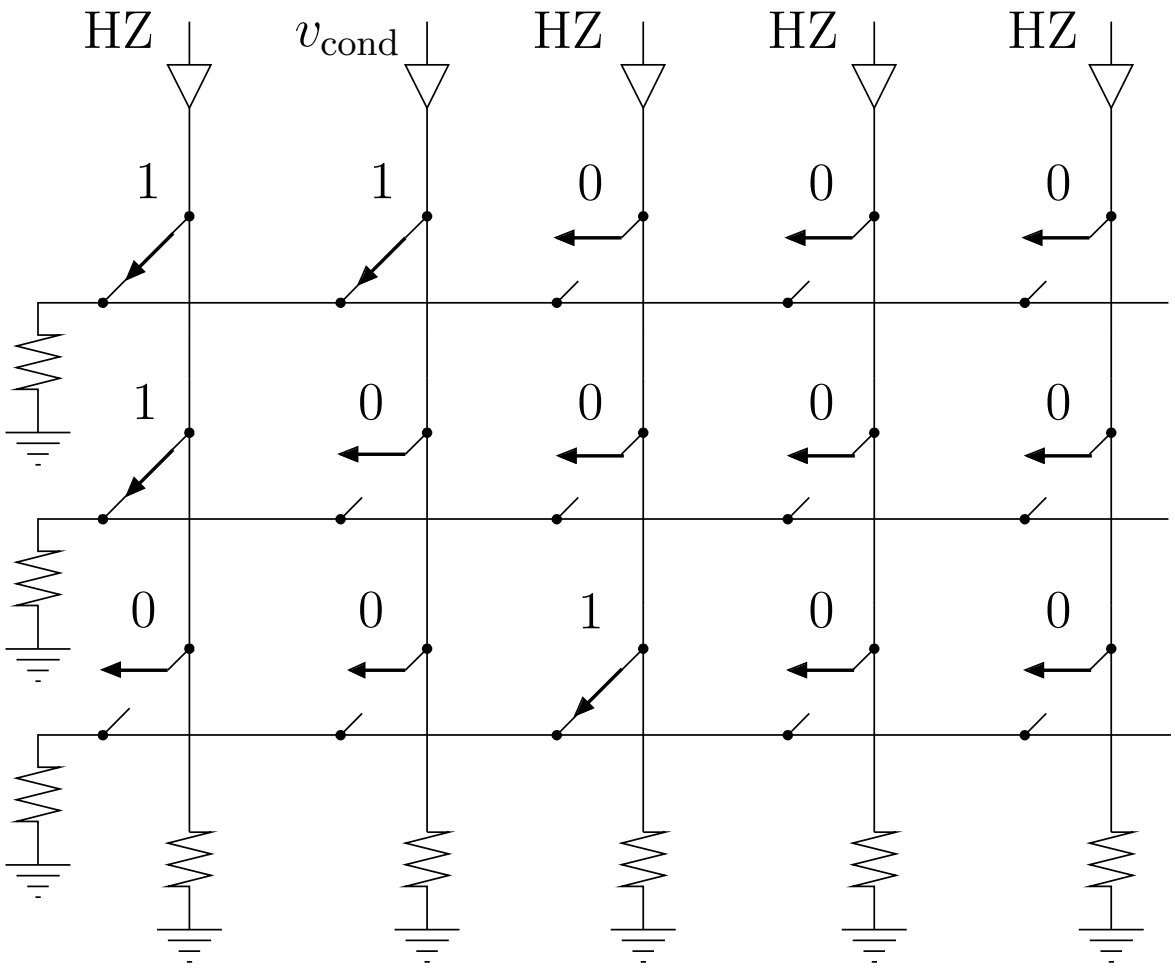
XOR (1st implication)



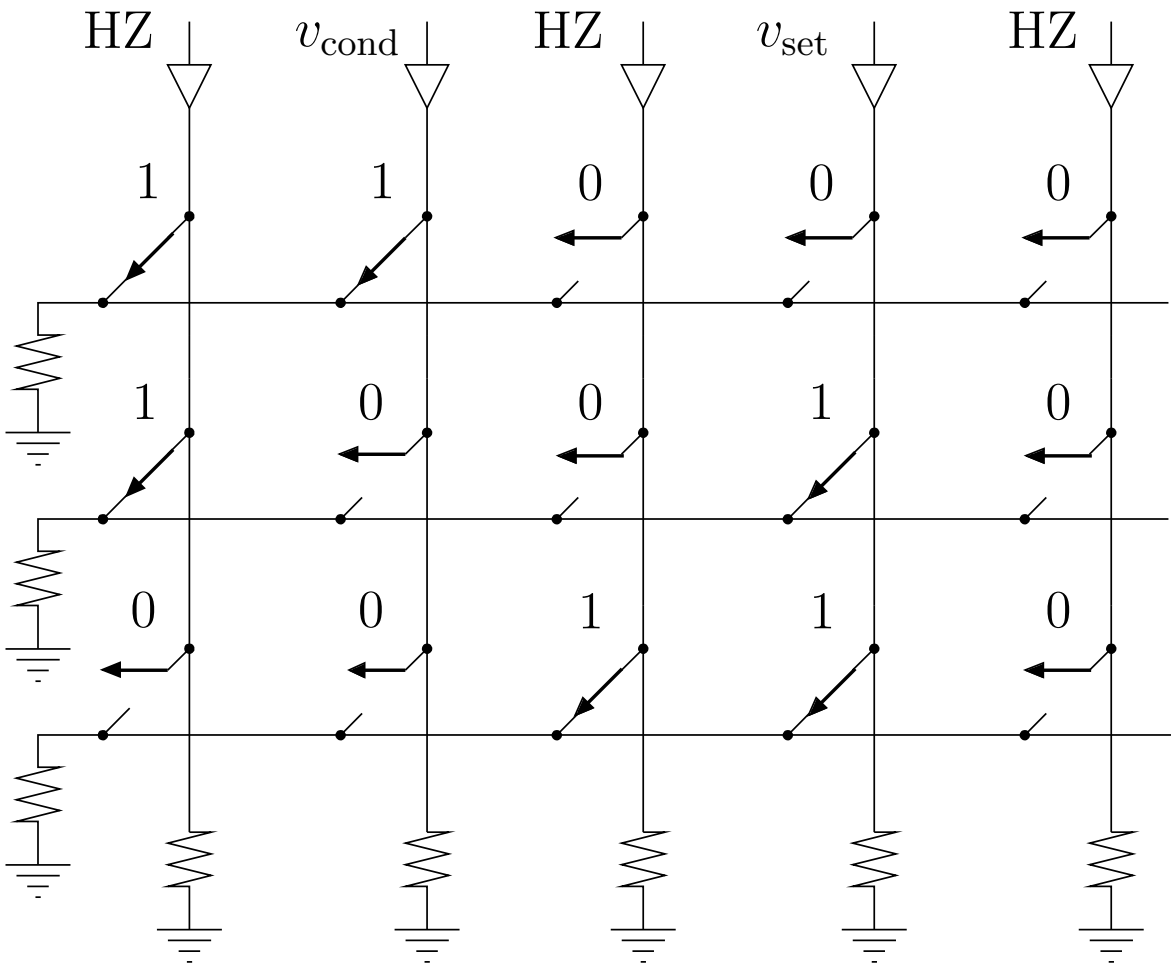
XOR (1st implication)



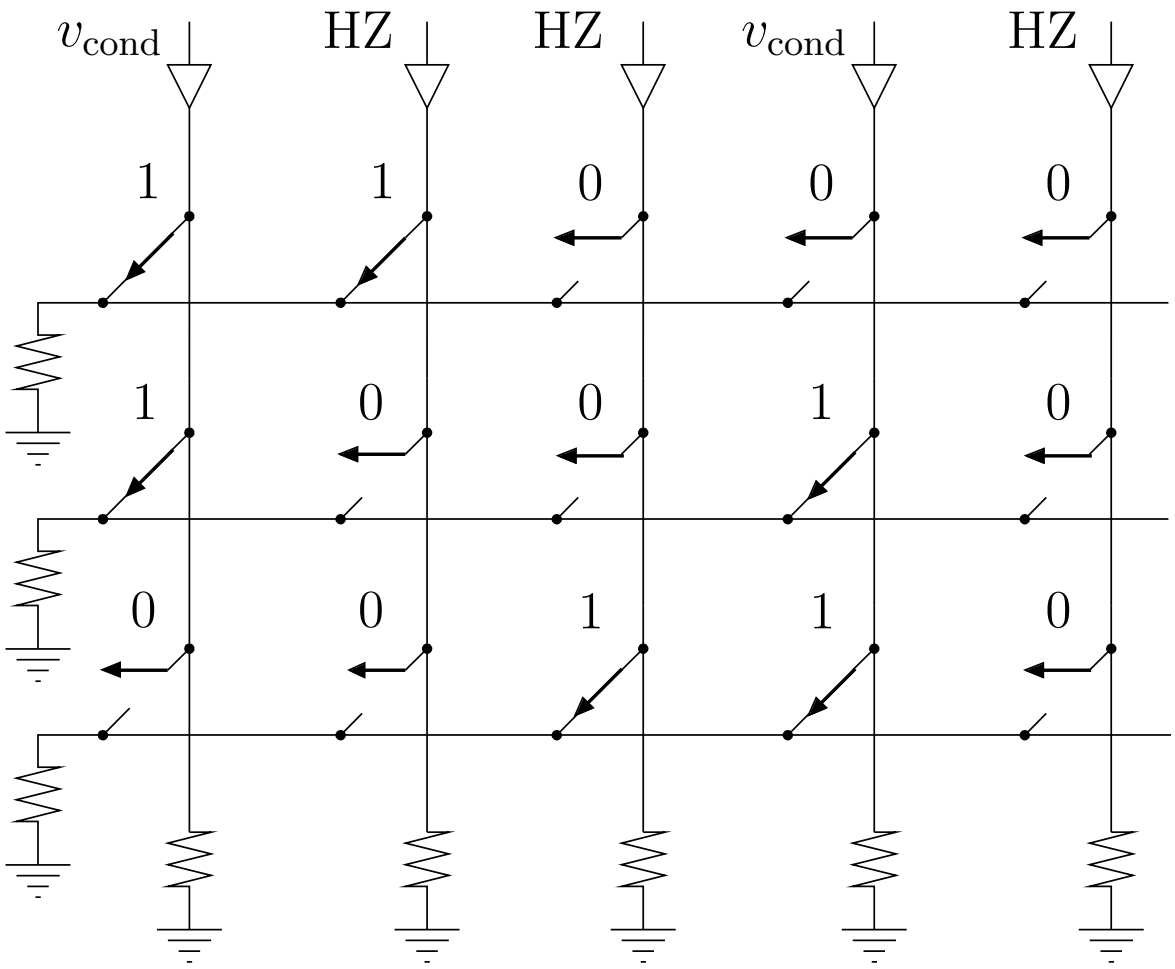
XOR (2nd implication)



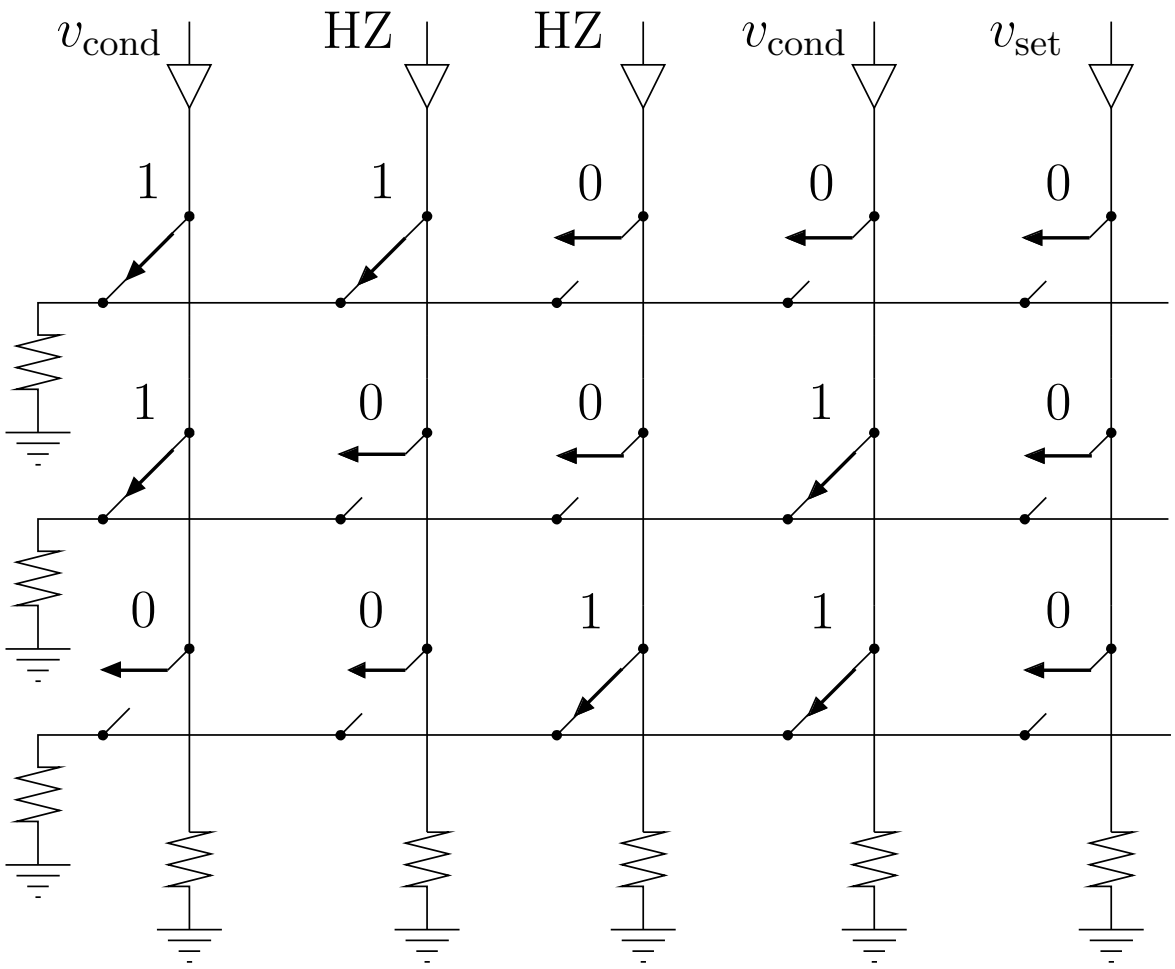
XOR (2nd implication)



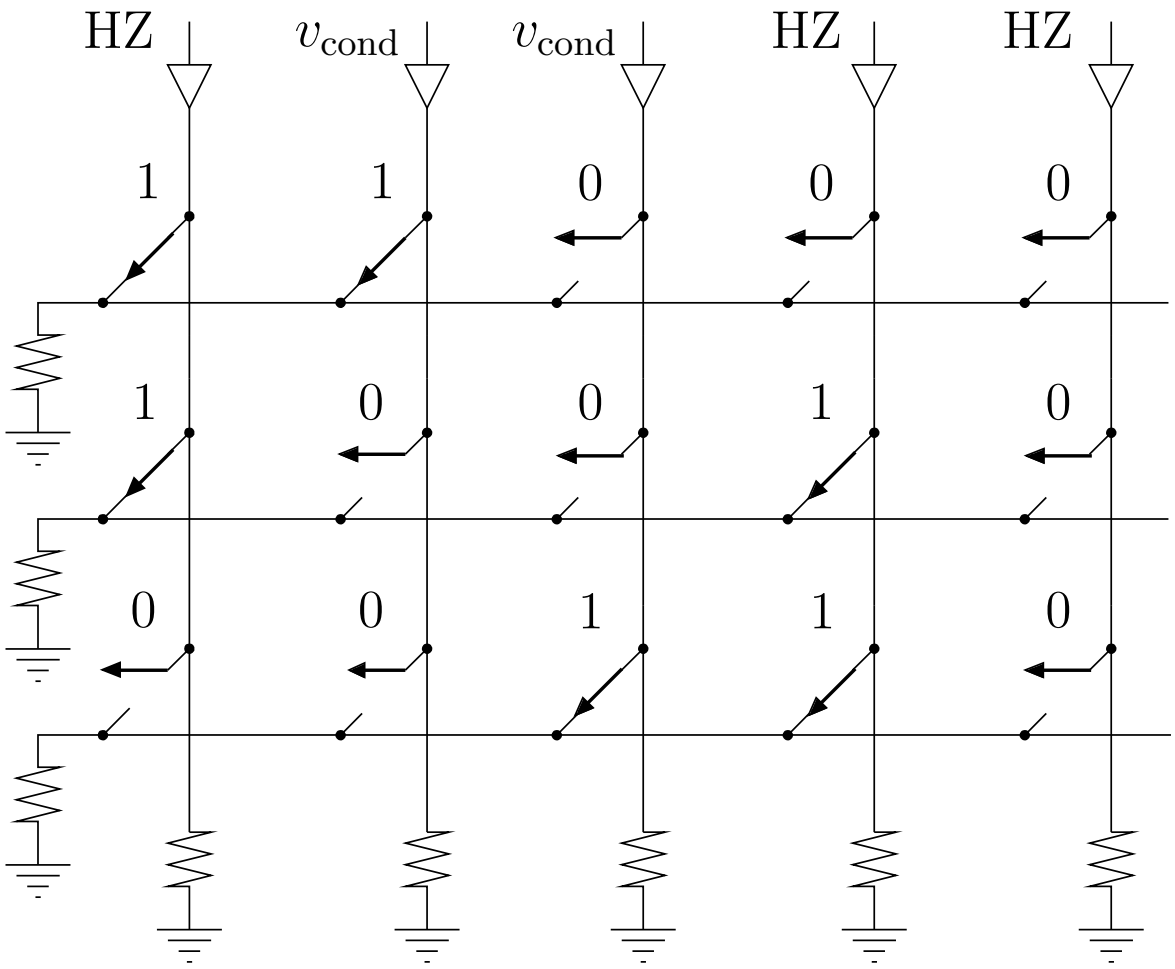
XOR (3rd implication)



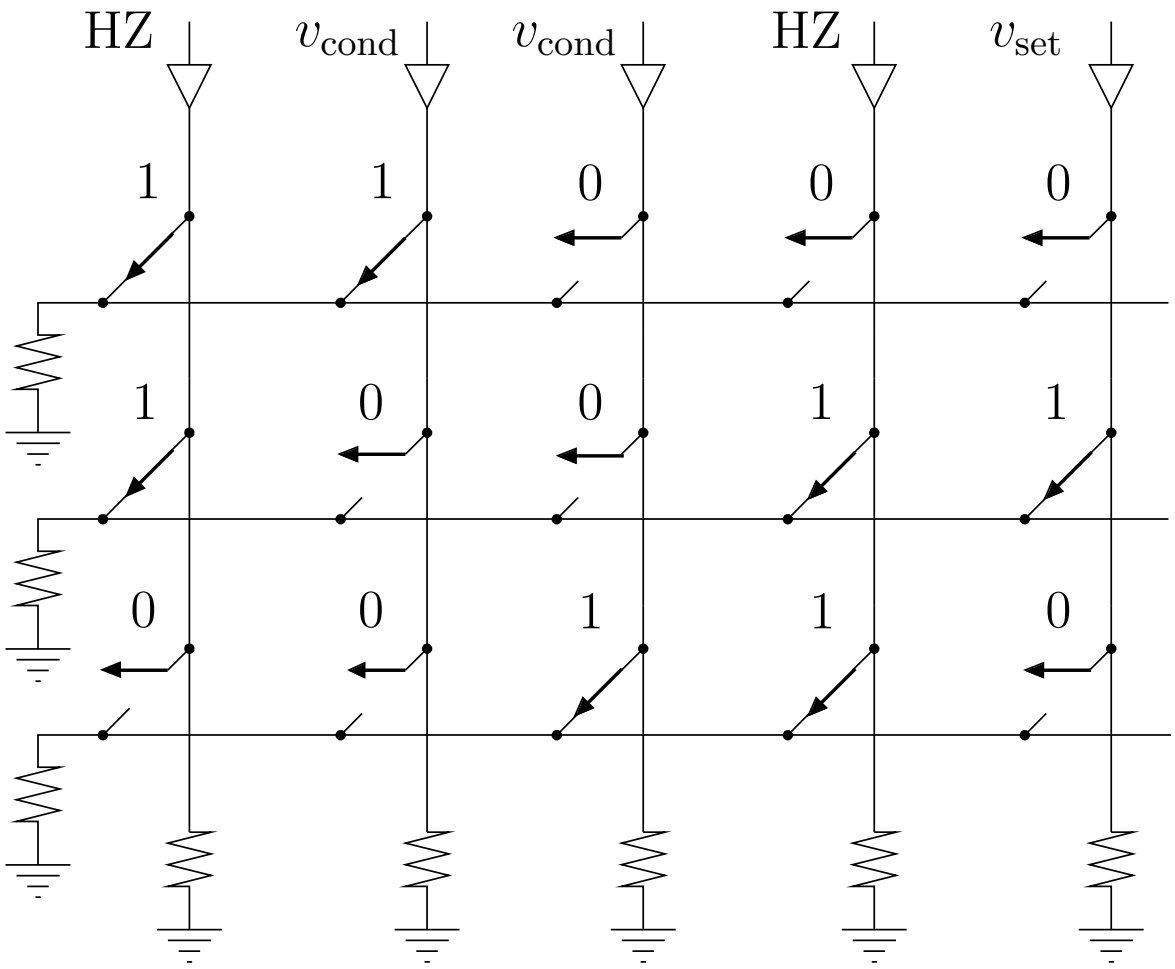
XOR (3rd implication)



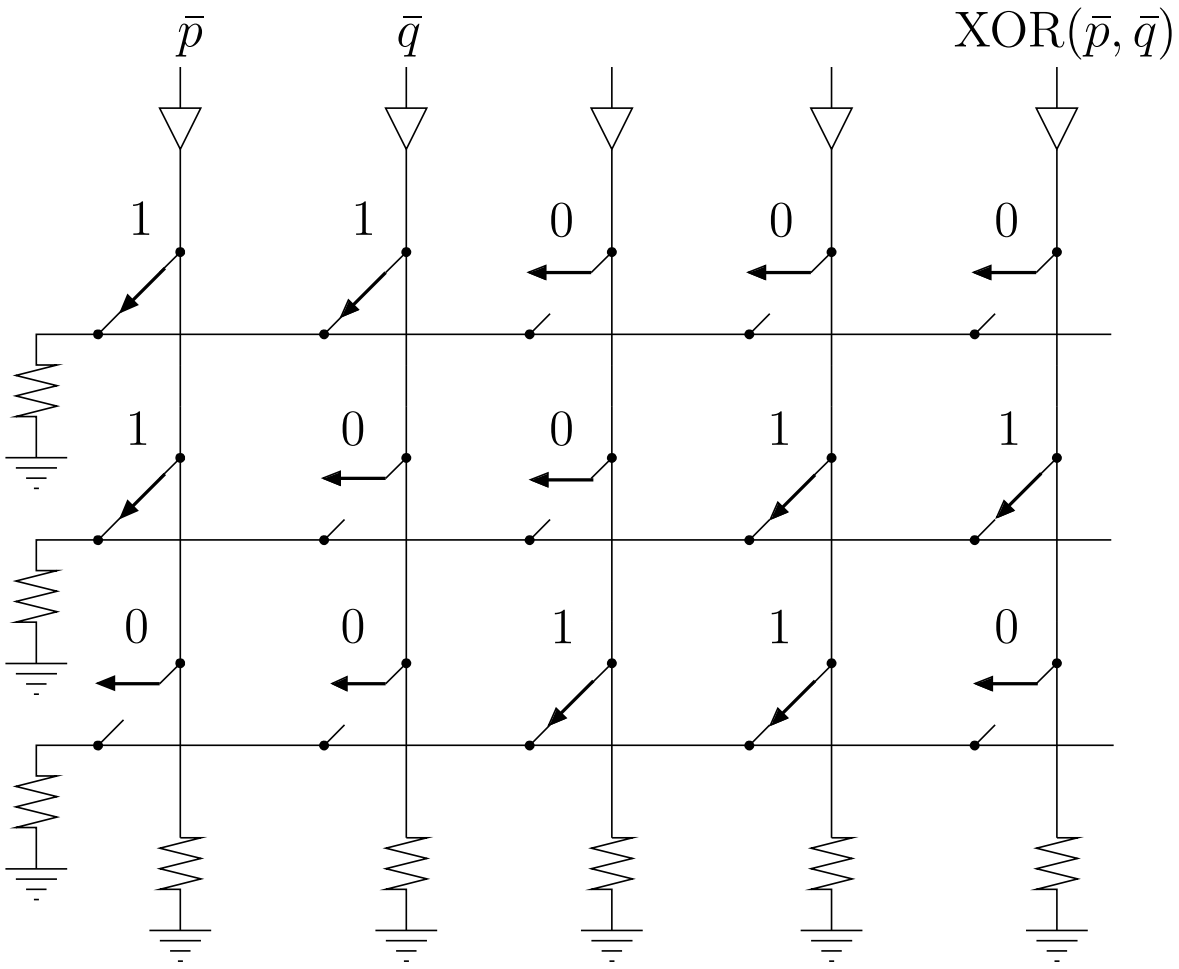
XOR (4th implication)



XOR (4th implication)



XOR of columns



Vector-parallel operations

Parallelization improves efficience. But...

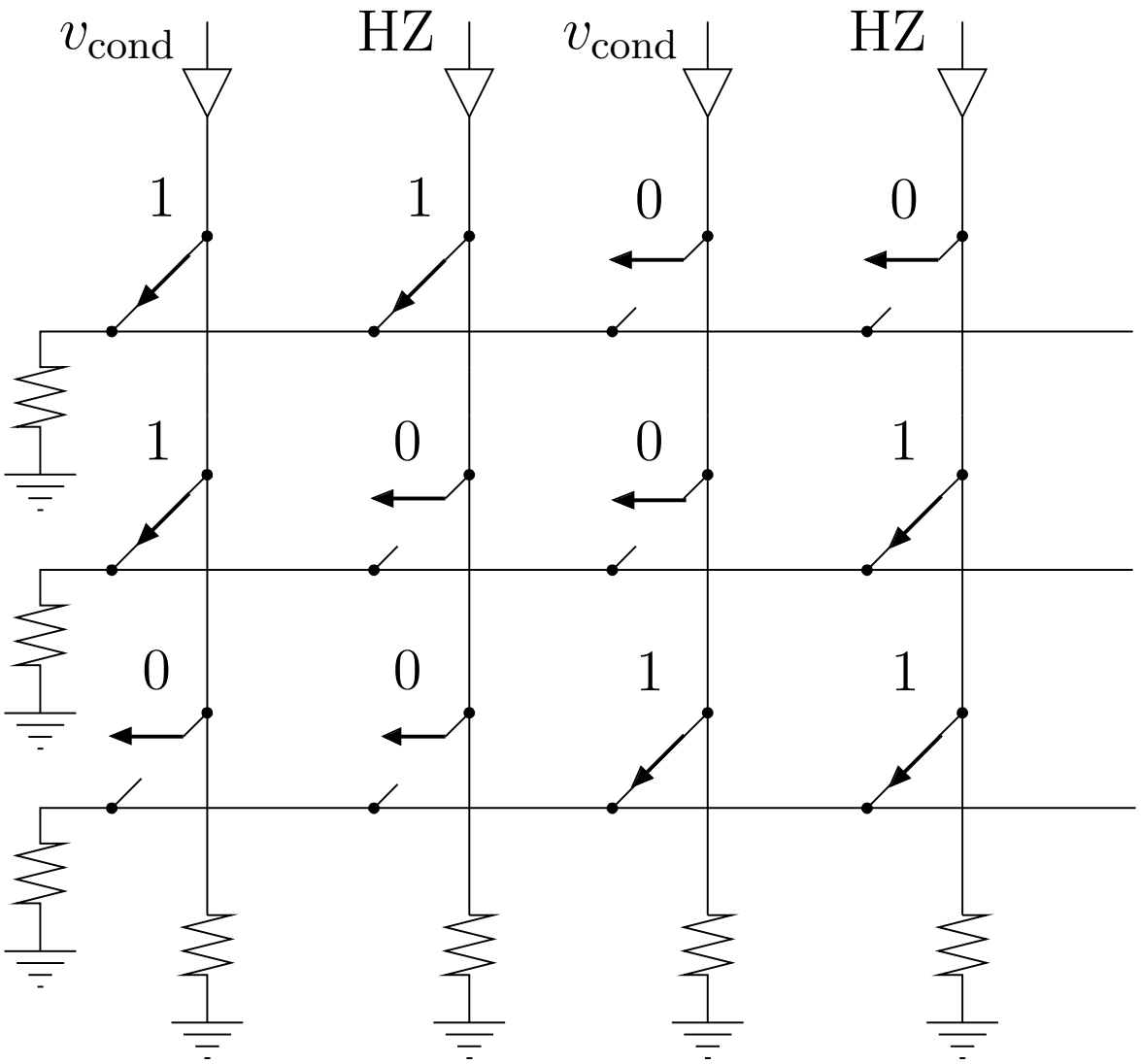
- Only one operation at a time
- Capacitance of a wire increases with the number of memristors
- Possible solution: segmenting of wires

Vector-parallel operations

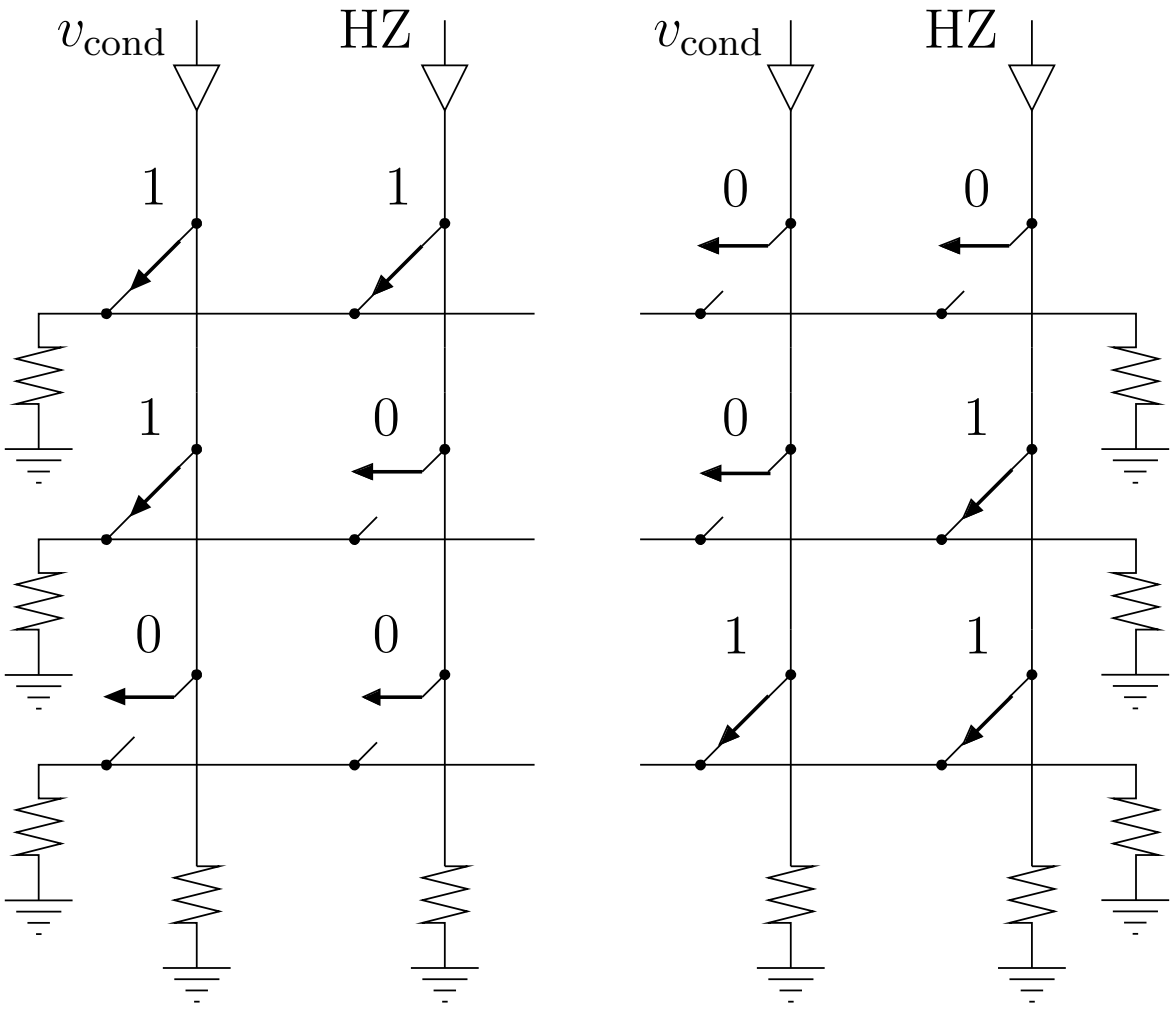
Parallelization improves efficience. But...

- Only one operation at a time
- Capacitance of a wire increases with the number of memristors
- Possible solution: segmenting of wires

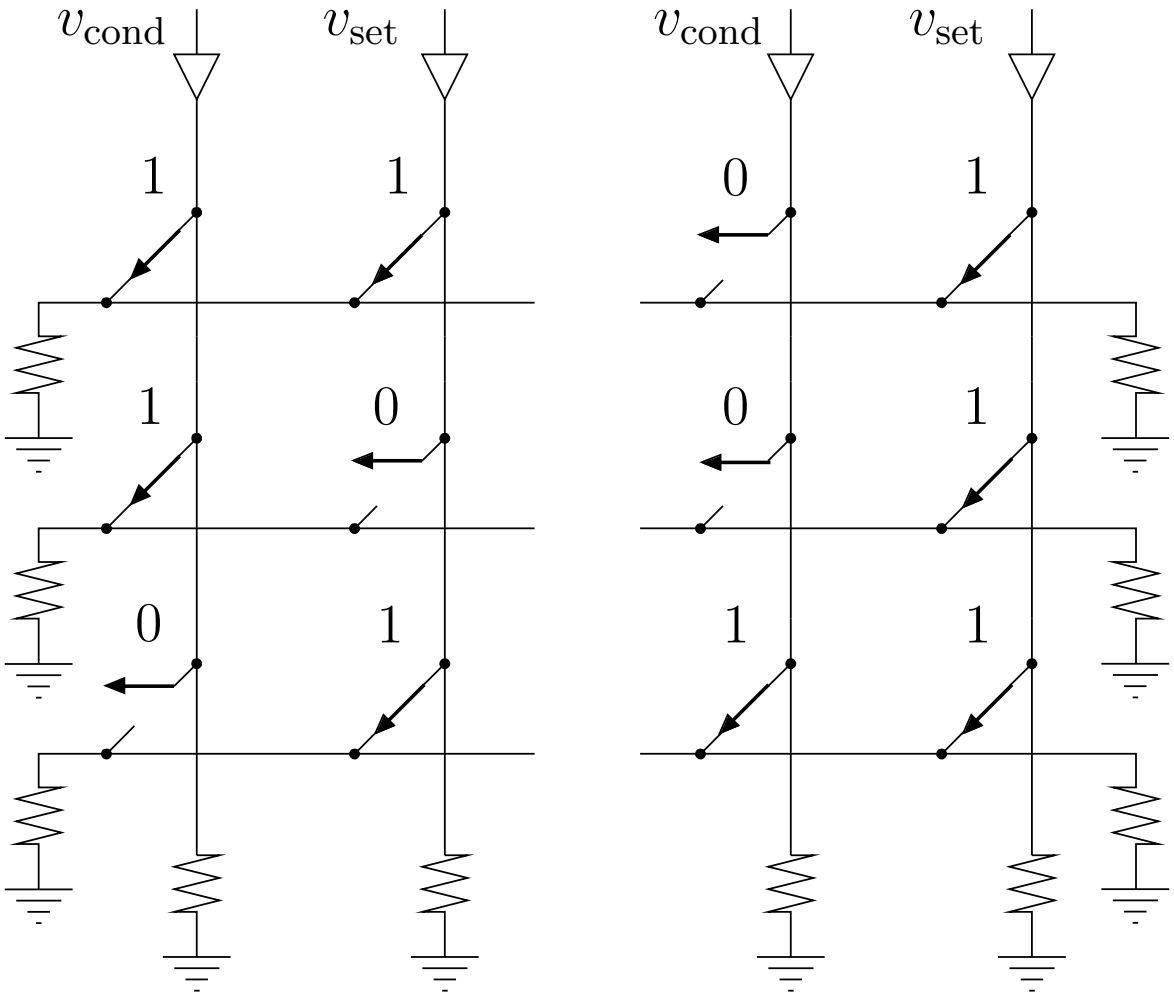
Segmenting wires



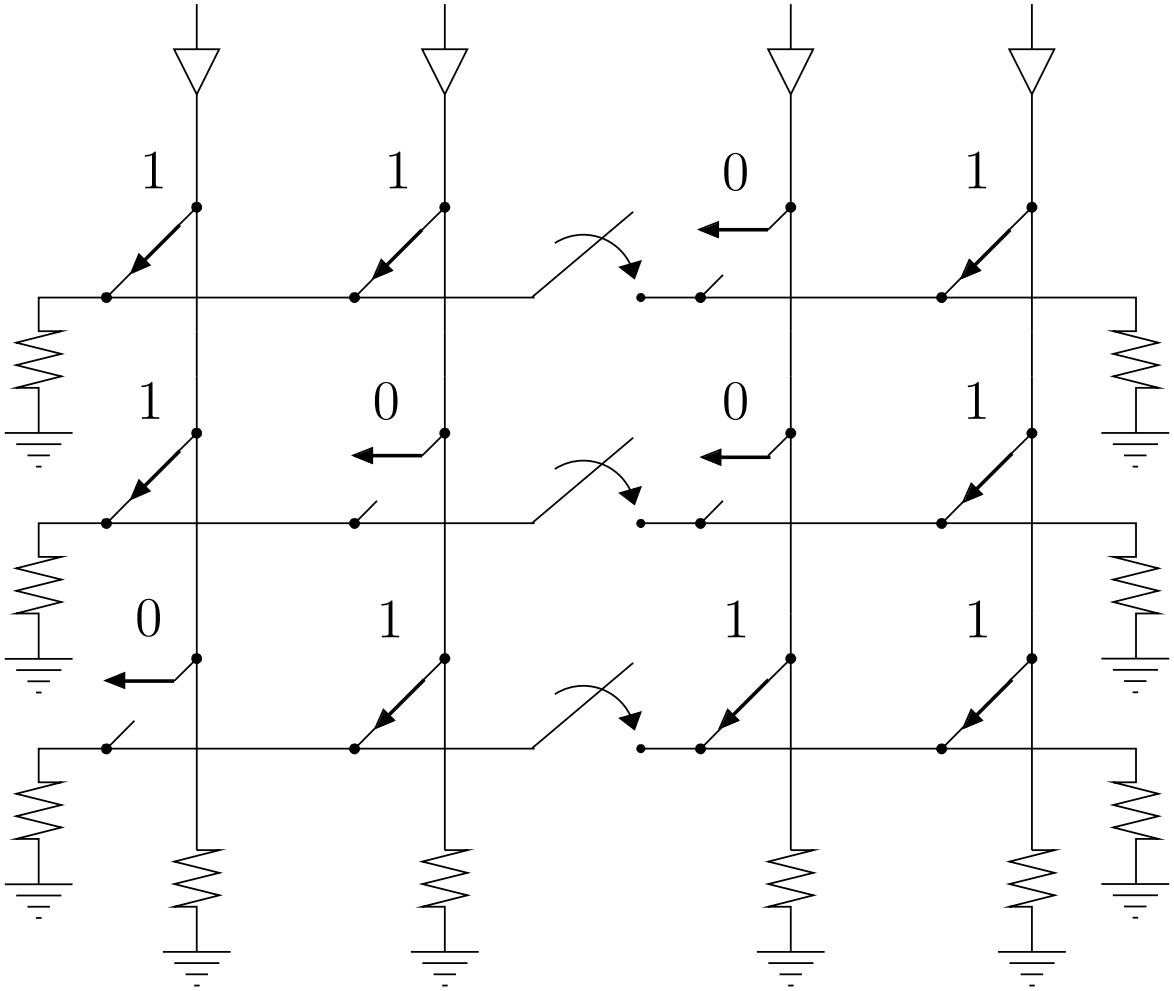
Segmenting wires



Segmenting wires



Segmenting wires



Vector-parallel operations

Parallelization improves efficience. But...

- Only one operation at a time
- Large capacitance when many memristors on a wire
- Possible solution: segmenting of wires
- Implementation: memristive, nanowire transistors, CMOS...?

Vector-parallel operations

Parallelization improves efficience. But...

- Only one operation at a time
- Large capacitance when many memristors on a wire
- Possible solution: segmenting of wires
- CMOS implementation: local operations should be fast (10 - 100 MHz)
- For example, 1000 rows x 1000 cols x 10e6 ops/s

Content-addressable memory

1	0	1	0	1	0	1	0	x
1	1	0	0	1	1	0	0	0
1	0	1	0	1	0	1	0	0
0	0	1	1	0	0	1	1	0

Content-addressable memory

1	0	1	0	1	0	1	0	x	
1	1	0	0	1	1	0	0	0	
1	0	1	0	1	0	1	0	1	$\longleftarrow$
0	0	1	1	0	0	1	1	0	

1	0	1	0	1	0	1	0	x
1	1	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	1	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

CAM (1st implication)

v_{cond}	1	0	1	0	1	0	1	0	x
	1	1	0	0	1	1	0	0	0
	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0
	1	0	1	0	1	0	1	0	0
	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0
	0	0	1	1	0	0	1	1	0
	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0

CAM (1st implication)

v_{cond}	1	0	1	0	1	0	1	0	x
	1	1	0	0	1	1	0	0	0
v_{set}	0	1	0	1	0	1	0	1	0
	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0
	1	0	1	0	1	0	1	0	0
v_{set}	0	1	0	1	0	1	0	1	0
	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0
	0	0	1	1	0	0	1	1	0
v_{set}	0	1	0	1	0	1	0	1	0
	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0

CAM (wire segmenting)

1	0	1	0	1	0	1	0	x
1	1	0	0	1	1	0	0	0
0	1	0	1	0	1	0	1	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0	0
0	1	0	1	0	1	0	1	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	1	0
0	1	0	1	0	1	0	1	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

CAM (XOR of search and memory vectors)

	1	0	1	0	1	0	1	0	x
	1	1	0	0	1	1	0	0	0
	0	1	0	1	0	1	0	1	0
	0	0	0	0	0	0	0	0	0
XOR	0	1	1	0	0	1	1	0	0
	1	0	1	0	1	0	1	0	0
	0	1	0	1	0	1	0	1	0
	0	0	0	0	0	0	0	0	0
XOR	0	0	0	0	0	0	0	0	0
	0	0	1	1	0	0	1	1	0
	0	1	0	1	0	1	0	1	0
	0	0	0	0	0	0	0	0	0
XOR	1	0	0	1	1	0	0	1	0

CAM (Multi-input column-wise implication)

	v_c	v_c	v_c	v_c	v_c	v_c	v_c	v_c	v_s	
	1	0	1	0	1	0	1	0	x	
	1	1	0	0	1	1	0	0	0	
	0	1	0	1	0	1	0	1	0	
	0	0	0	0	0	0	0	0	0	
XOR	0	1	1	0	0	1	1	0	0	$\longleftarrow$
	1	0	1	0	1	0	1	0	0	
	0	1	0	1	0	1	0	1	0	
	0	0	0	0	0	0	0	0	0	
XOR	0	0	0	0	0	0	0	0	1	$\longleftarrow$
	0	0	1	1	0	0	1	1	0	
	0	1	0	1	0	1	0	1	0	
	0	0	0	0	0	0	0	0	0	
XOR	1	0	0	1	1	0	0	1	0	$\longleftarrow$

Conclusion

In this presentation...

- Stateful logic operations
- Parallelization into a crossbar
- Wire segmenting: independent operations
- Future work: massively parallel stateful logic

Conclusion

In this presentation...

- Stateful logic operations
- Parallelization into a crossbar
- Wire segmenting: independent operations
- Future work: massively parallel stateful logic

Conclusion

In this presentation...

- Stateful logic operations
- Parallelization into a crossbar
- Wire segmenting: independent operations
- Future work: massively parallel stateful logic

Conclusion

In this presentation...

- Stateful logic operations
- Parallelization into a crossbar
- Wire segmenting: independent operations
- Future work: massively parallel stateful logic