Logic Design with Memristors

Shahar Kvatinsky

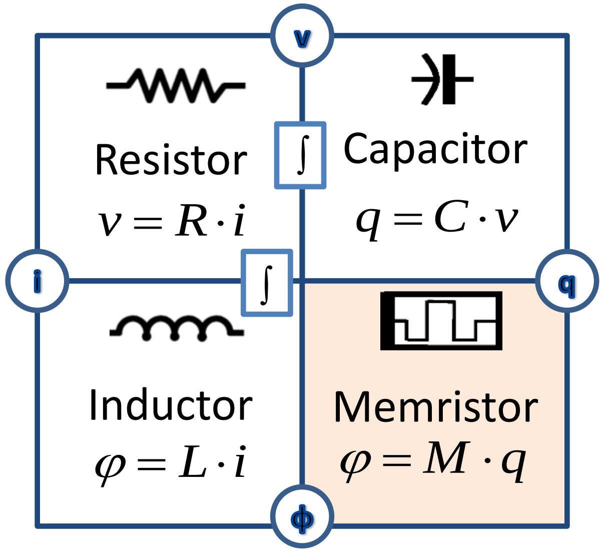
Technion – Israel Institute of Technology

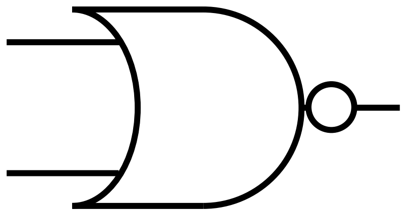
ACRC Workshop March 2012

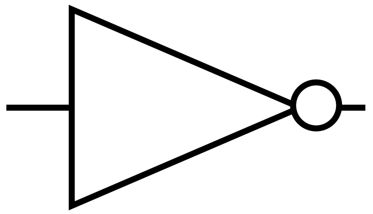
Memristors are Useful for Logic

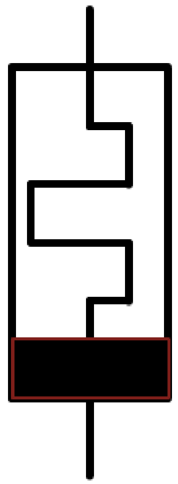
- Two terminal resistive device
- Not only memory
- This talk is about

memristor-based logic circuits





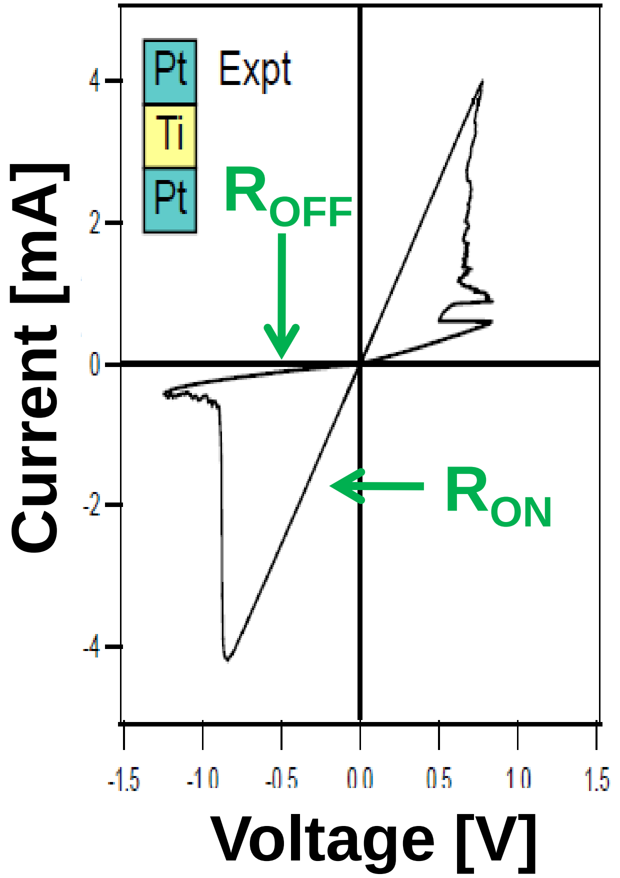




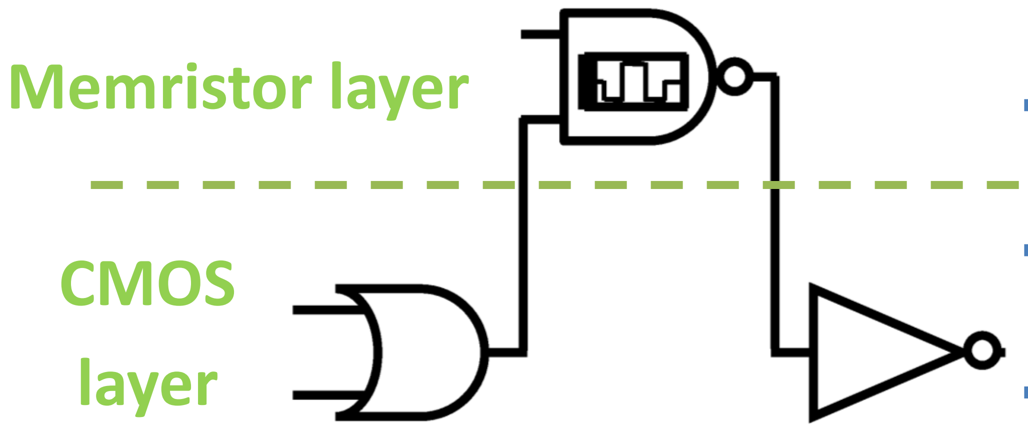
Chua, "Memristor – The Missing Circuit Element," *IEEE Trans.*, 1971
Chua and Kang, "Memristive Devices and Systems," *Proceedings of the IEEE*, 1976

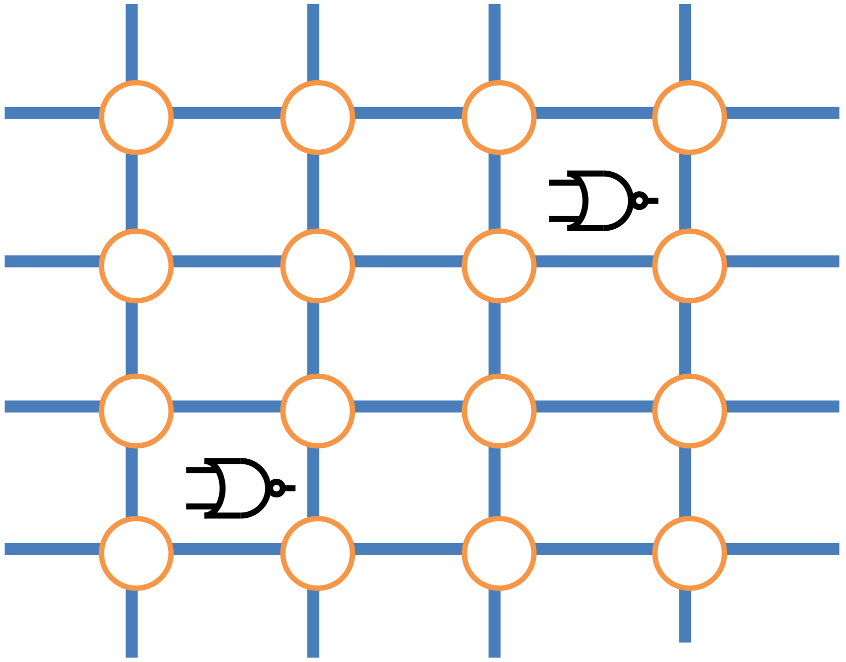
Logic with Memristors

- Memristors as a building block
- Memristor can:
  - Store an **input** value
  - Store an **output** value
  - Perform logic operation
  - Act as a state register (**latch, Flip-Flop**)
  - Act as a **configurable switch**



Why Use Memristors in Logic?





Integrating memristors with standard logic

Save die area

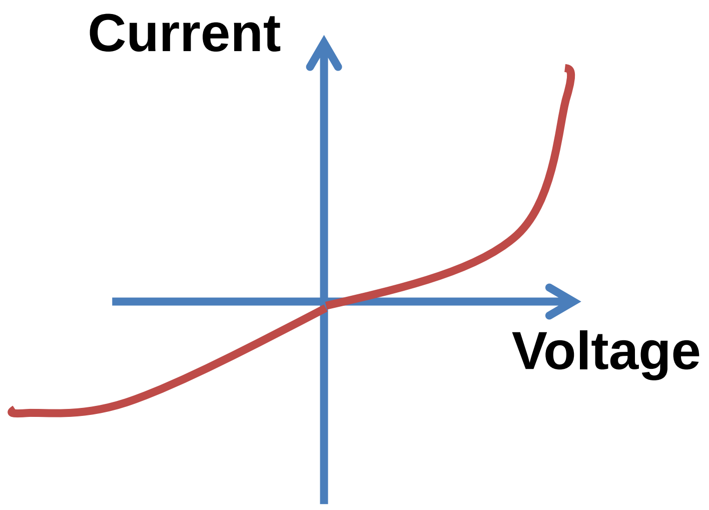
More logic on die

Logic inside the memory

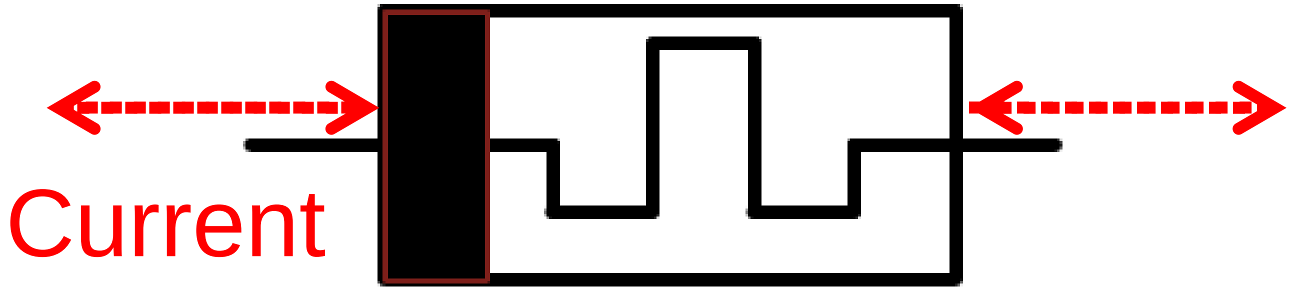
Beyond Von-Neumann

Flexible

Memristor Polarity



Decrease resistance

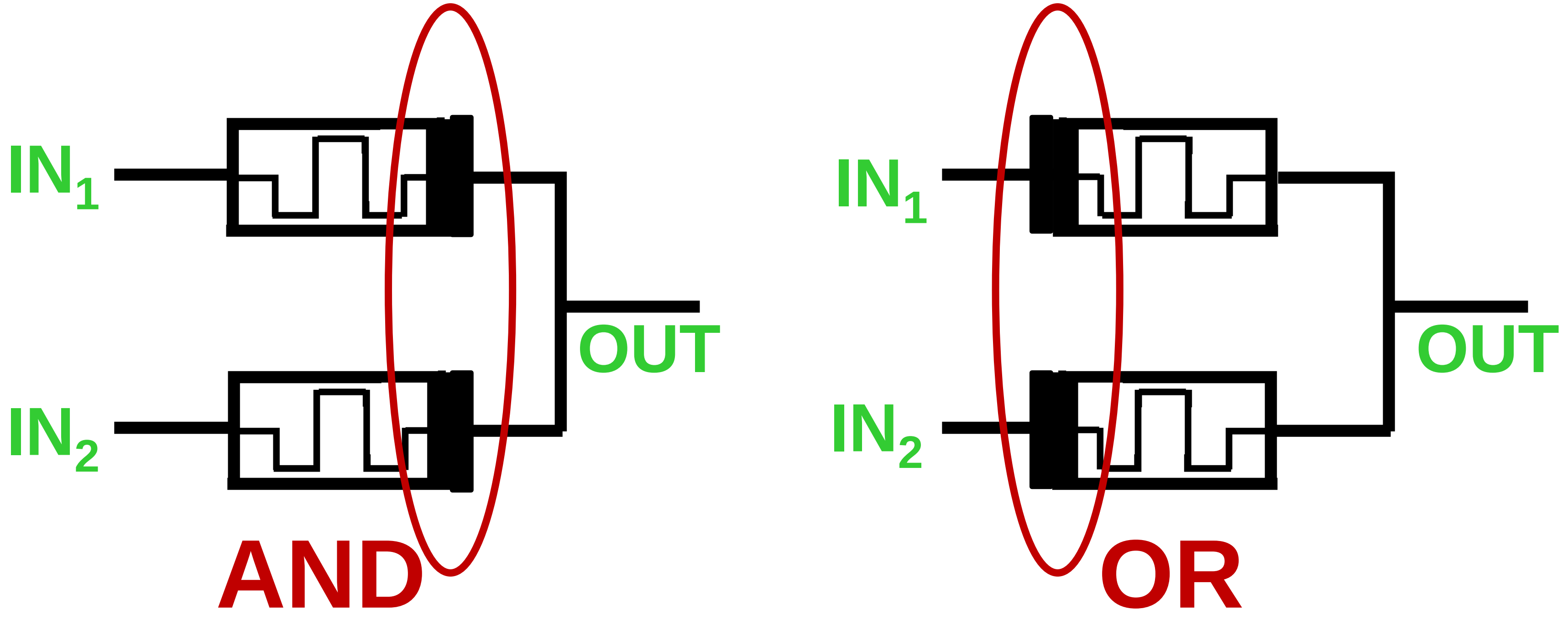


Outline

- Motivation / Why logic with memristors?
- **Integrating memristor with standard logic**
- Memristor-based logic inside the memory:
  - IMPLY logic Gate
  - Memristor Aided LoGIC (MAGIC)
- Design methodology
- Conclusions

AND and OR (Eshraghian 2011)

- Voltage as logic state
- Memristors only as computational elements

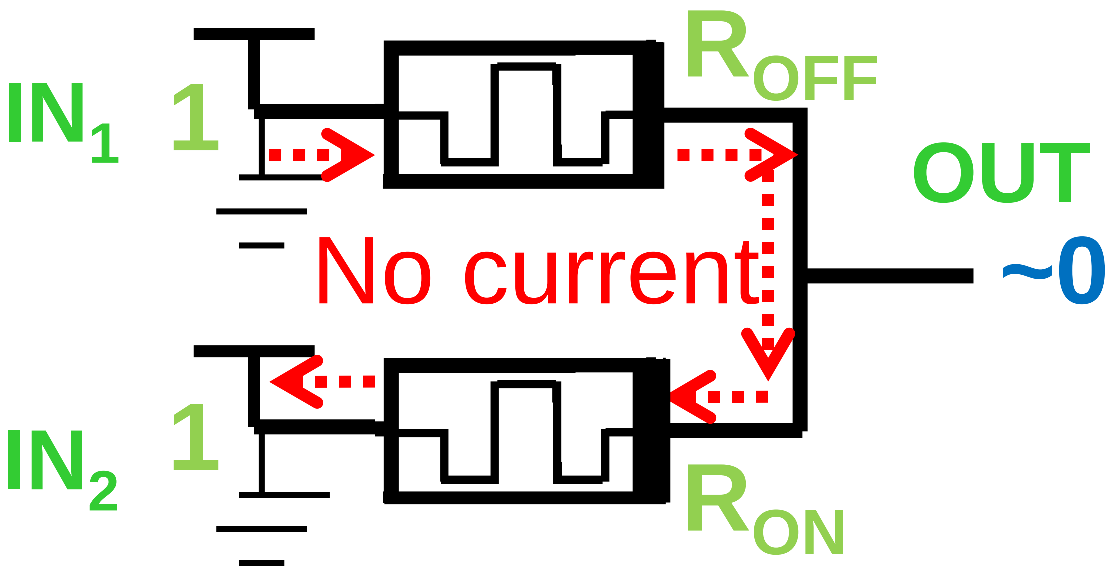


AND Operation

$$V_{OUT} = V_{CC} \cdot \frac{R_{ON}}{R_{ON} + R_{OFF}} \approx V_{CC} \cdot \frac{R_{ON}}{R_{OFF}} << V_{CC}$$

IN_1	IN_2	AND
0	0	0
0	1	0
1	0	0
1	1	1

Increase resistance

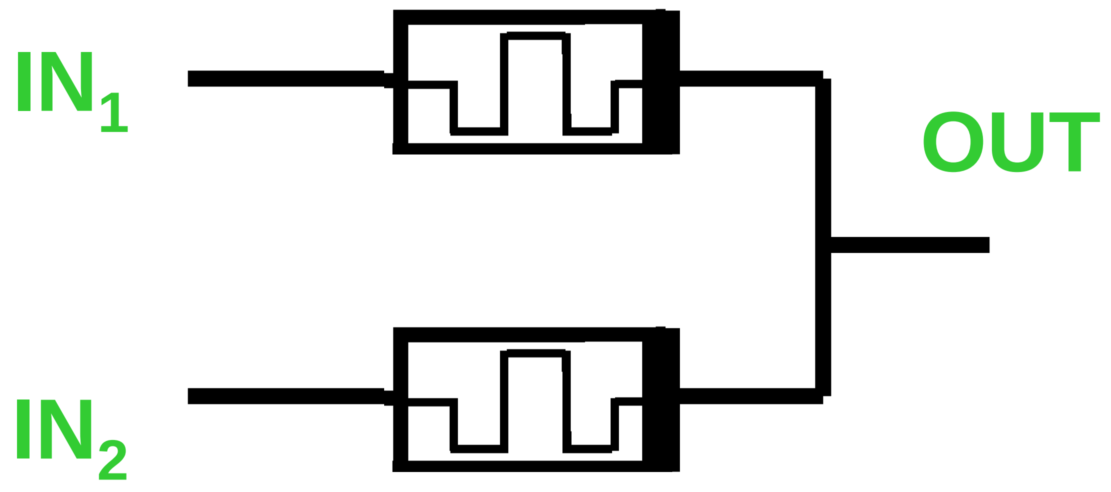


Decrease resistance

$R_{OFF} >> R_{ON}$

AND to OR

IN_1	IN_2	OR
0	0	0
0	1	1
1	0	1
1	1	1



AND OR

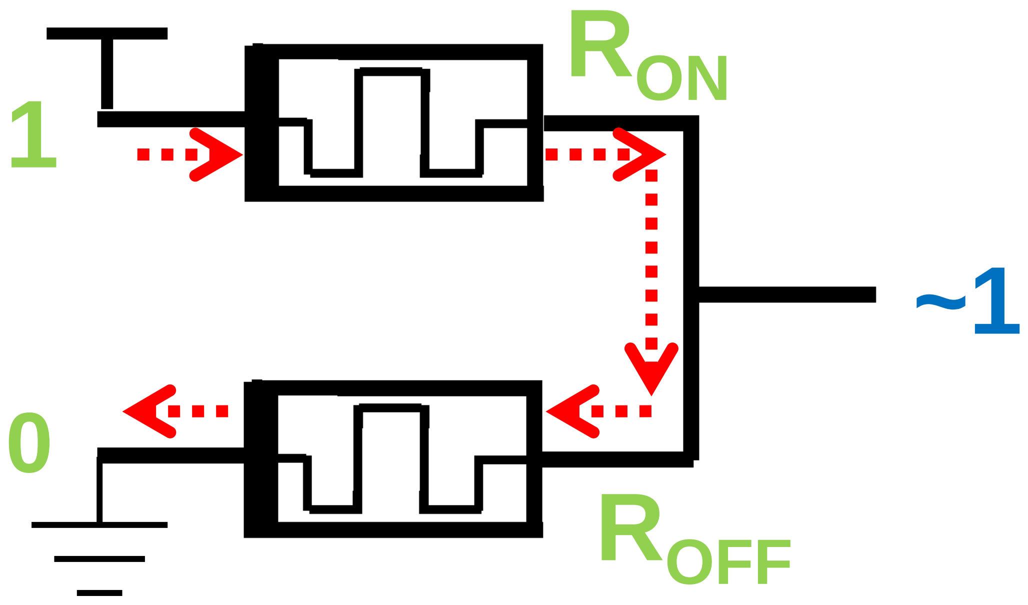
OR Operation

$$V_{OUT} = V_{CC} \cdot \frac{R_{OFF}}{R_{ON} + R_{OFF}} \approx V_{CC}$$

$R_{OFF} >> R_{ON}$

IN$_1$	IN$_2$	OR
0	0	0
0	1	1
1	0	1
1	1	1

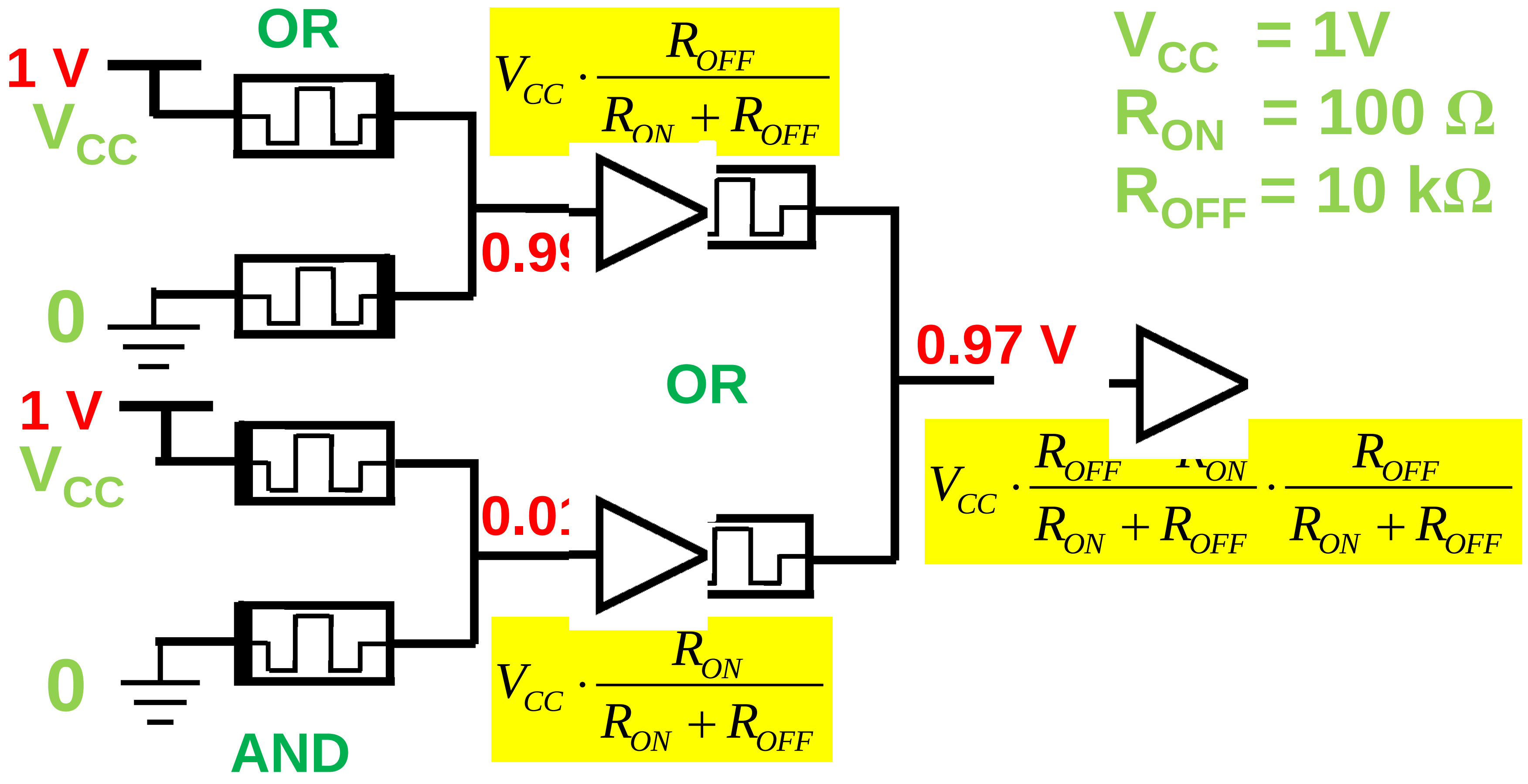
Decrease resistance



Increase resistance

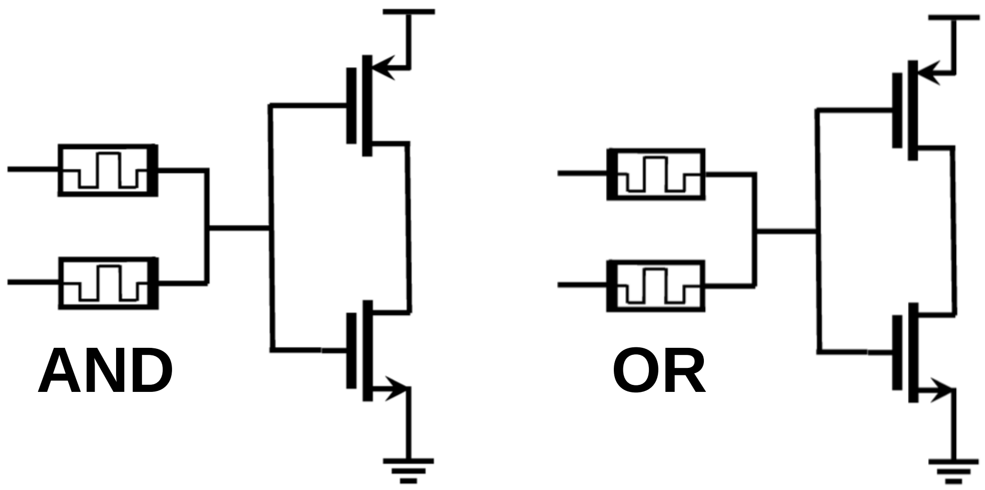
Need for Amplification

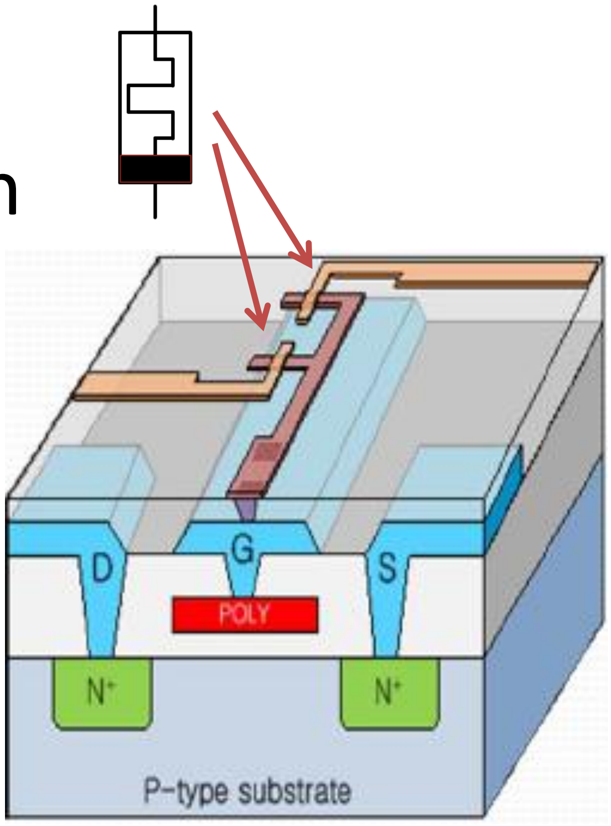
- Chain of memristor-based logic gates


OR
1 V
V_{CC}
0
$V_{CC} \cdot \frac{R_{OFF}}{R_{ON}+R_{OFF}}$
0.9
OR
1 V
V_{CC}
0.0
0
AND
$V_{CC} \cdot \frac{R_{ON}}{R_{ON}+R_{OFF}}$
0.97 V
$V_{CC} \cdot \frac{R_{OFF}}{R_{ON}+R_{OFF}} \cdot \frac{R_{OFF}}{R_{ON}+R_{OFF}}$
$V_{CC} = 1V$
$R_{ON} = 100\ \Omega$
$R_{OFF} = 10\ k\Omega$

CMOS Compatibility

- Memristors can be fabricated with CMOS
- Input/output are voltages – as in standard CMOS logic
- Amplify signal – signal restoration



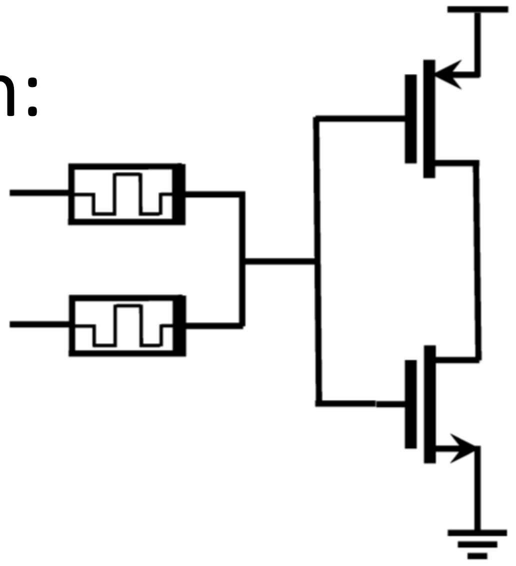


K. Eshraghian, course notes on "Memristive Circuits and Systems," Technion, June 2011
J. Borghetti etl al, "A Hybrid Nanomemristor/Transistor Logic Circuit Capable of Self-Programming," PNAS 2008

Integrating Memristor with Standard Logic - Summary

- Good for integration with standard logic
- Signal restoration through CMOS
- Save die area: 2 transistor - 2 memristor
- CMOS - memristor layer transition:

  vias, power and area overhead

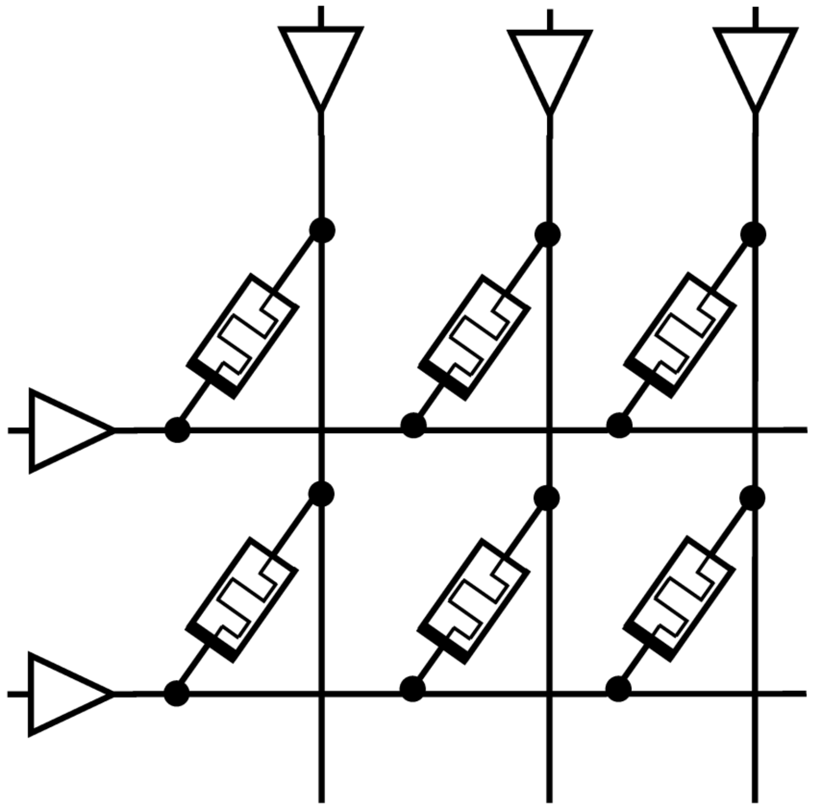


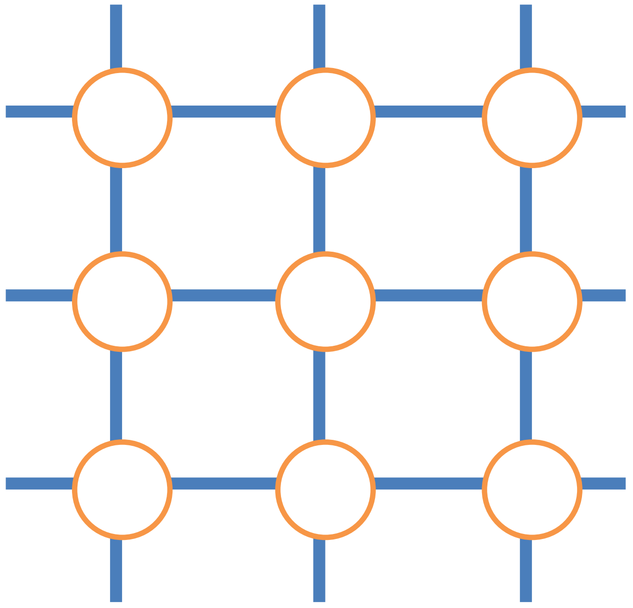
Outline

- Motivation / Why logic with memristors?
- Integrating memristor with standard logic
- **Memristor-based logic inside the memory:**
  - **IMPLY logic Gate**
  - Memristor Aided LoGIC (MAGIC)
- Design methodology
- Conclusions

Logic Inside the Memory

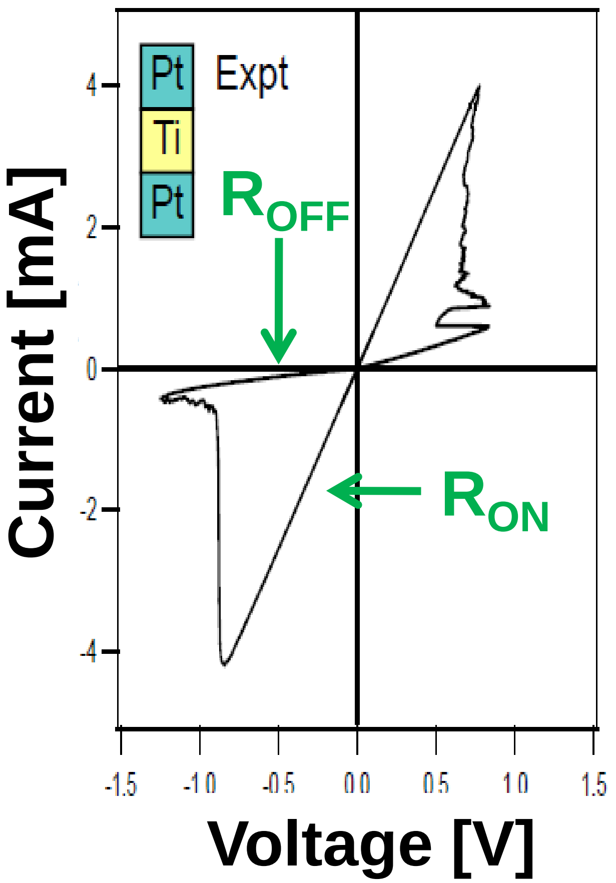
- Based on memristor-based crossbar memory





Logical State as Resistance

- R_{ON} → logical ‘1’, R_{OFF} → logical ‘0’
- The **input** of the logic gate is the memristor-based cells value
- The **result** is stored into the memory



IMPLY Function

- One of the elementary 2 input Boolean functions

Truth Table

p	q	*p IMP q*
0	0	1
0	1	1
1	0	0
1	1	1

$\mathbf{p} \rightarrow \mathbf{q}$

If p then q

IMPLY + FALSE $\Rightarrow$ **Complete logic**

IMPLY Logic with Memristors

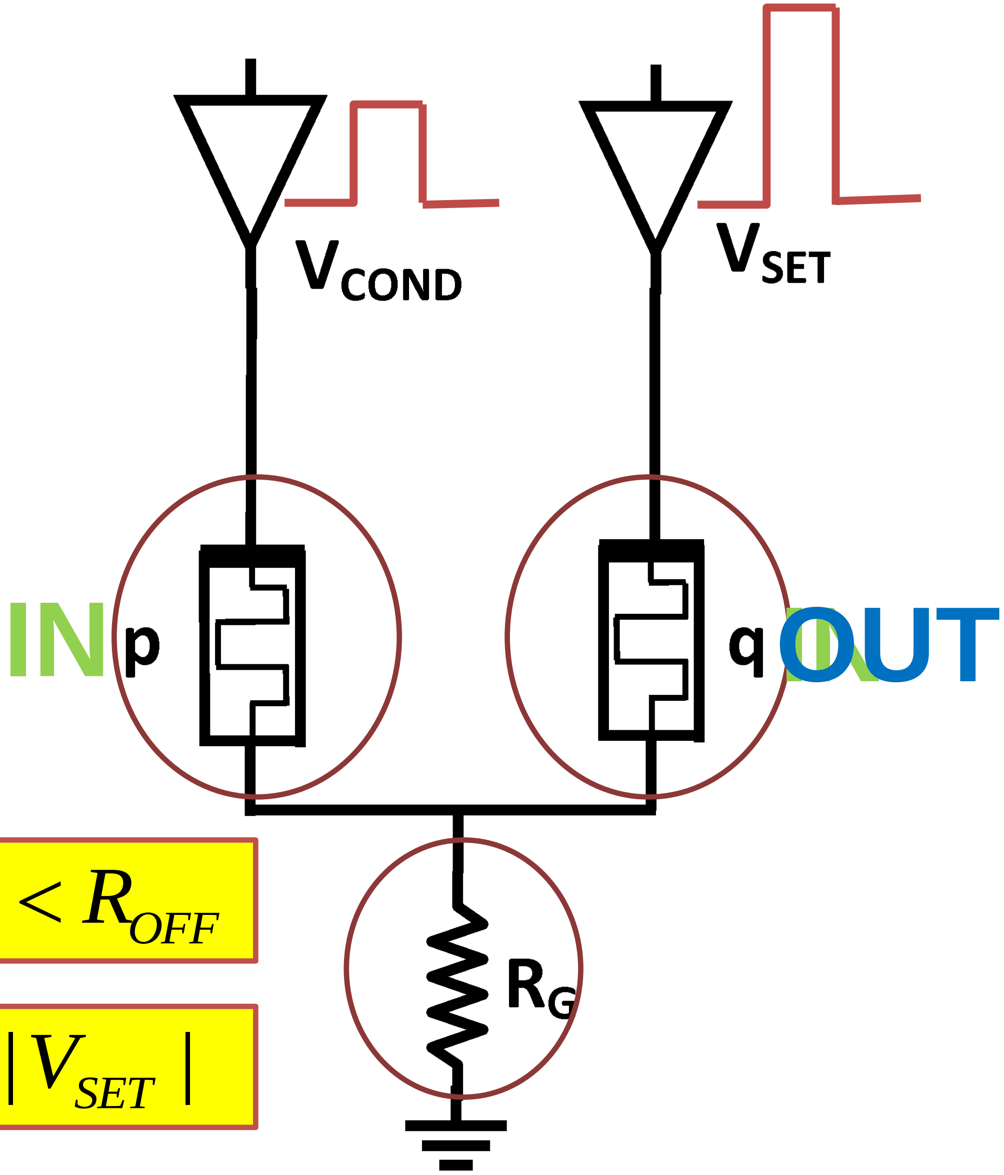
Logic 0 $\rightarrow R_{OFF}$

Logic 1 $\rightarrow R_{ON}$

p	q	*p IMP q*
0	0	1
0	1	1
1	0	0
1	1	1

$$R_{ON} < R_G < R_{OFF}$$

$$|V_{COND}| < |V_{SET}|$$



J. Borghetti et al, “Memristive Switches Enable ‘Stateful’ Logic Operation via Material Implication,” Nature, 2010

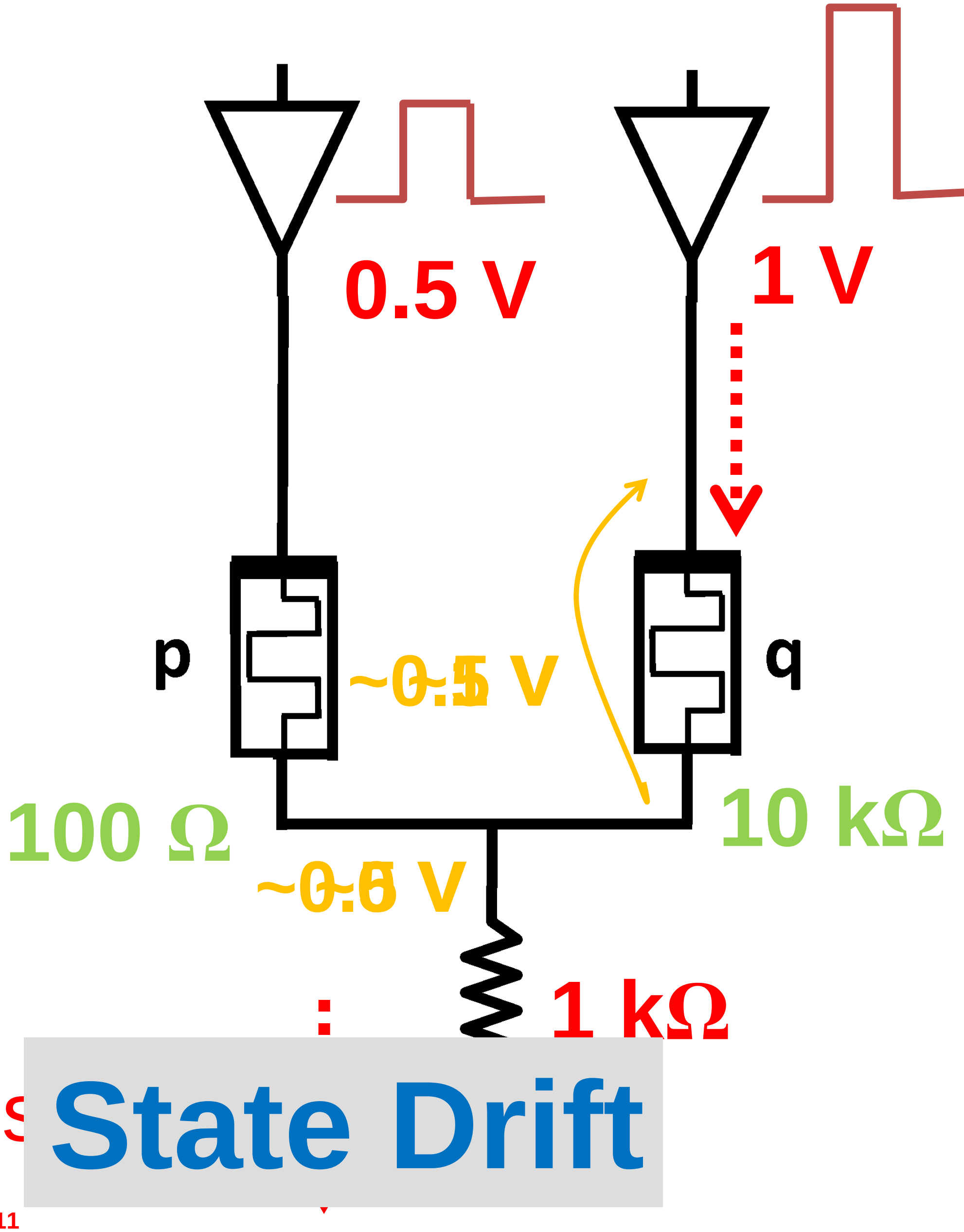
Behavior for Different Cases

R_{ON} = 100 Ω

R_{OFF} = 10 kΩ

Case	p	q	*p IMP q → q*
1	0	0	1
2	0	1	1
3	1	0	0
4	1	1	1

IN OUT



Decrease resis

State Drift

Performance and Robustness Tradeoff

Case	p	q	*p IMP q → q*
1	0	0	1
2	0	1	1
3	1	0	0
4	1	1	1

IN OUT

Write time

↕ TRADEOFF

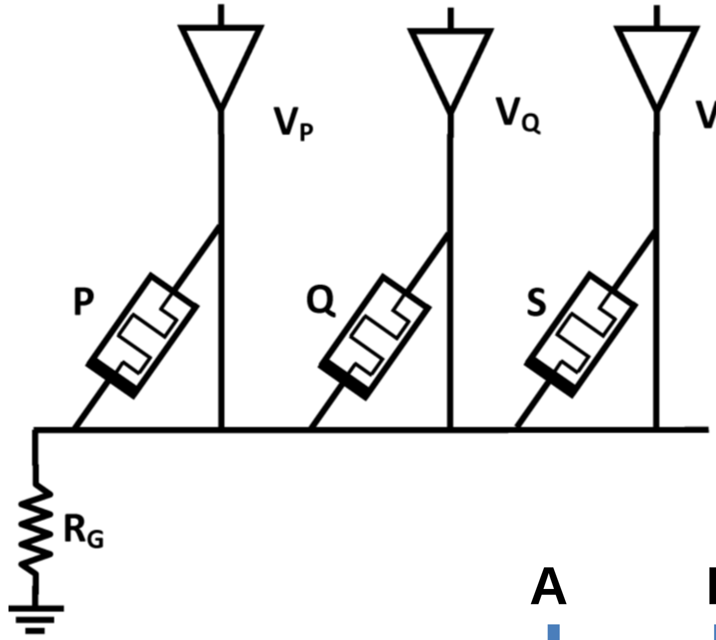
State drift

↓

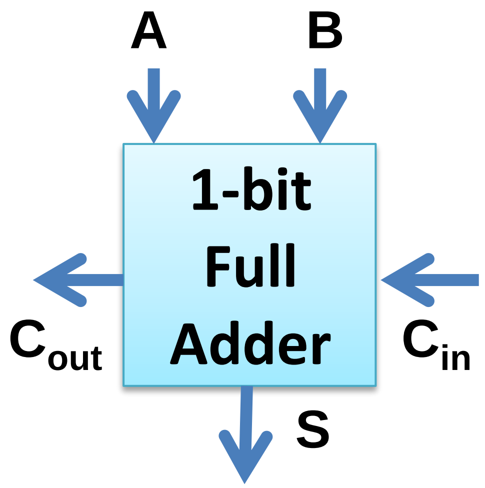
Refreshing the gate

General Functions with IMPLY

- **Sequential operation** of IMPLY and FALSE
- NAND:
  - Step 1 – FALSE(S)
  - Step 2 – P IMPLY S
  - Step 3 – Q IMPLY S

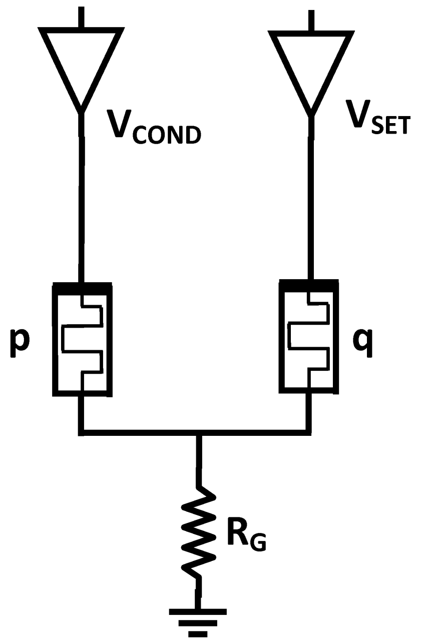


- 1- bit Full Adder
  - Naive approach: 89 computation steps
  - Parallel approach: 5 computation steps



IMPLY Summary

- Performance and robustness tradeoff
- Need for refresh because of state drift
- For general Boolean function needs many computation stages:
  - Slow
  - Complex controller
  - Power consumption

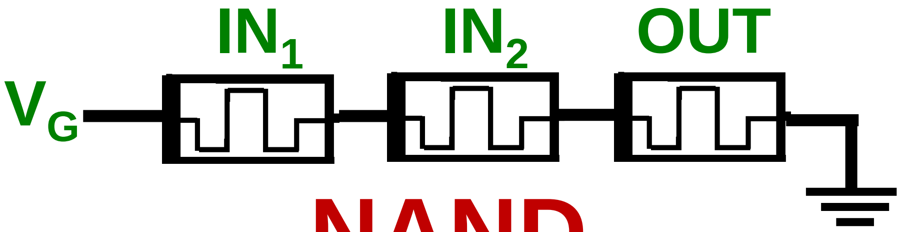


Outline

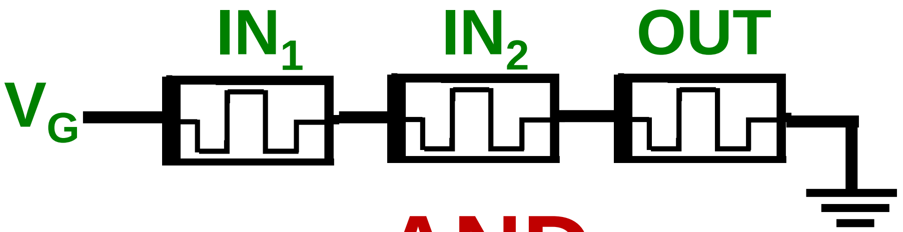
- Motivation / Why logic with memristors?
- Integrating memristor with standard logic
- Memristor-based logic inside the memory:
  - IMPLY logic Gate
  - **Memristor Aided LoGIC (MAGIC)**
- Design methodology
- Conclusions

MAGIC – Memristor Aided LoGIC

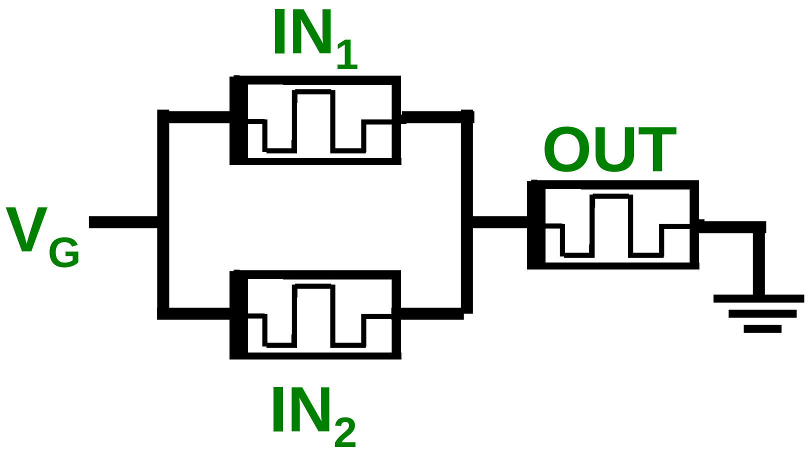
- One applied voltage V_G
- Separate input and output memristors



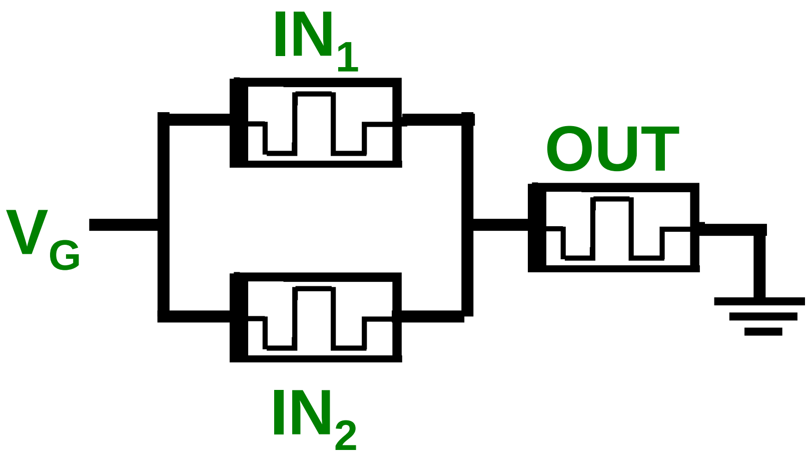
NAND



AND



NOR



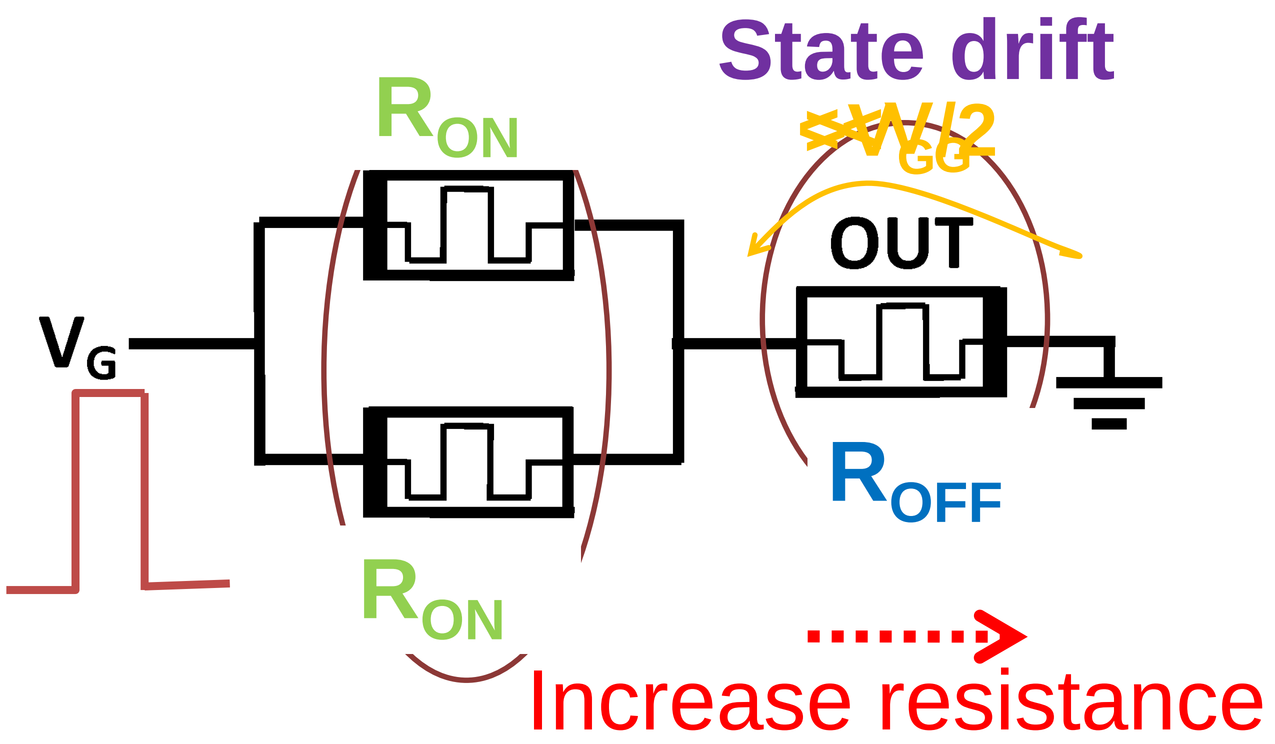
OR

MAGIC NOR

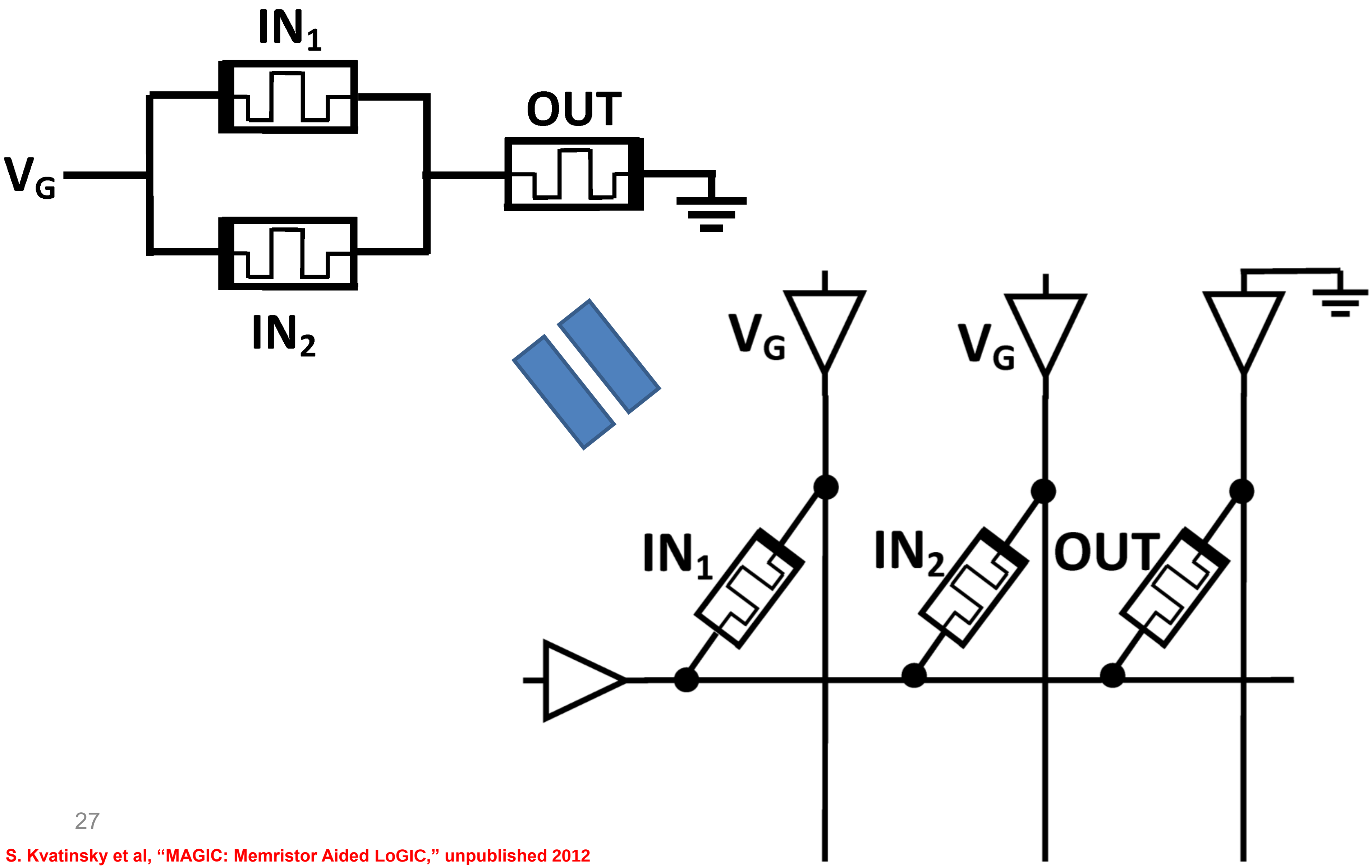
Initialize OUT to R_{ON}

$R_{OFF} >> R_{ON}$

IN_1	IN_2	*NOR*
0	0	1
0	1	0
1	0	0
1	1	0

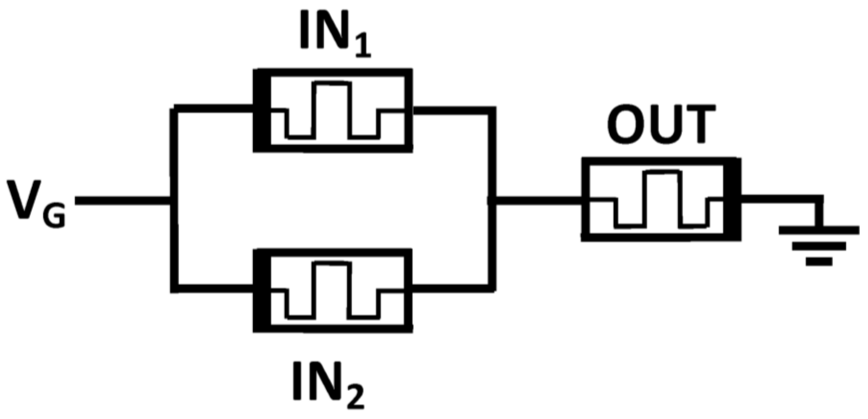


MAGIC NOR in Crossbar



MAGIC - Summary

- Good for logic inside the memory
- Separate input and output memristors
- Easy and intuitive
- State drift phenomenon - noise margin issues

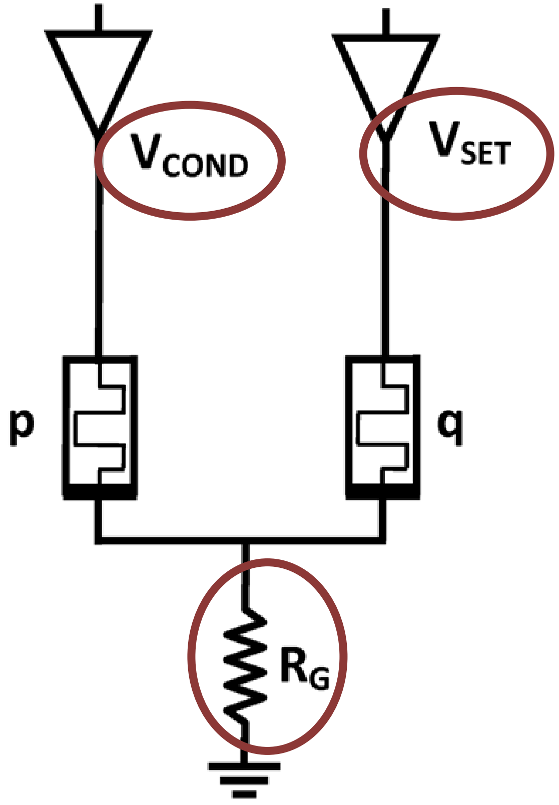


Outline

- Motivation / Why logic with memristors?
- Integrating memristor with standard logic
- Memristor-based logic inside the memory:
  - IMPLY logic Gate
  - Memristor Aided LoGIC (MAGIC)
- **Design methodology**
- Conclusions

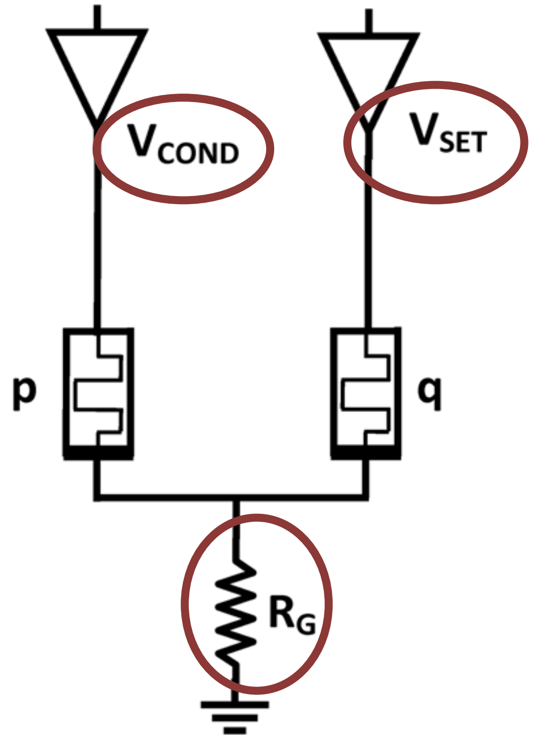
Need Design Methodology

- Decide which family to use
- Determine proper circuit parameters
  - R_G?
  - Voltage levels? V_{COND}? V_{SET}?
  - Logic gate delay?



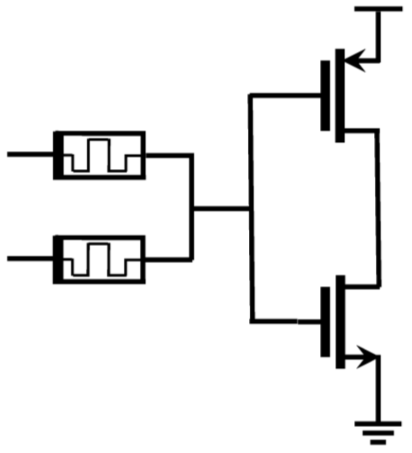
Developing Design Methodology

- IMPLY logic gate design methodology (ICCD 2011)
- Developing a complete design methodology
- General design constraints:
  - Power
  - Area
  - Performance
- Specific design constraints

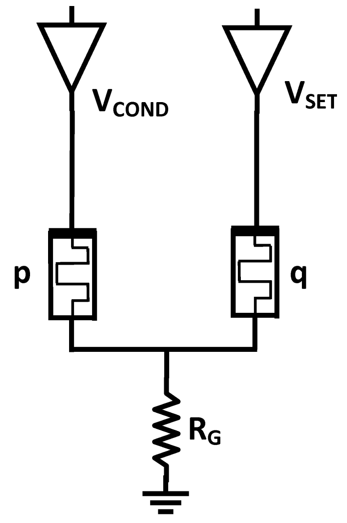


Outline

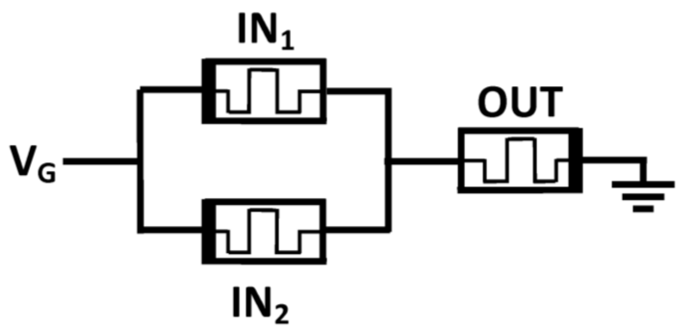
Logic with Memristors



Hybrid



IMPLY



MAGIC

Many issues – huge opportunities!

Reduce die area

More computation on die

Beyond Von-Neumann architecture

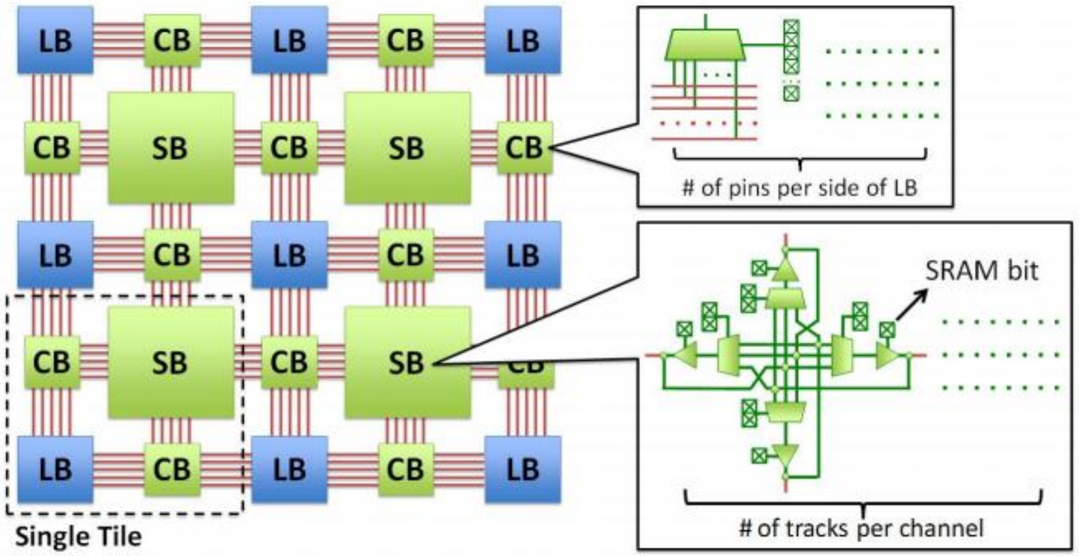
Thanks!

http://memristor.shorturl.com

BACKUP

Conventional FPGA

- FPGA power consumption:
  - 90% - SRAM (routing)
  - 10% - computing

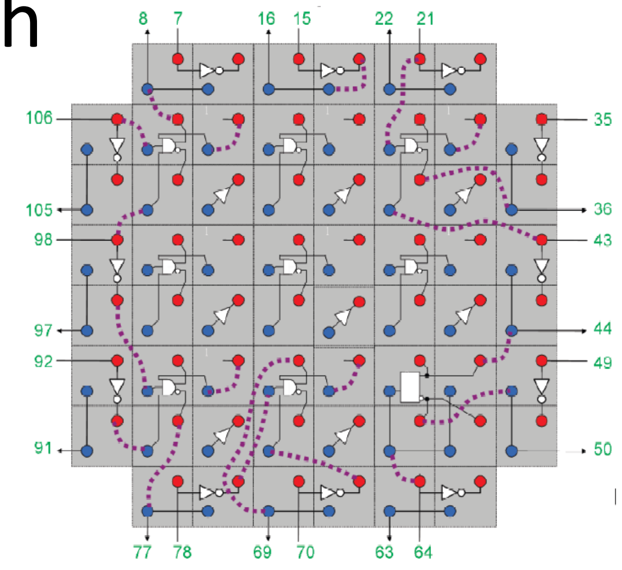


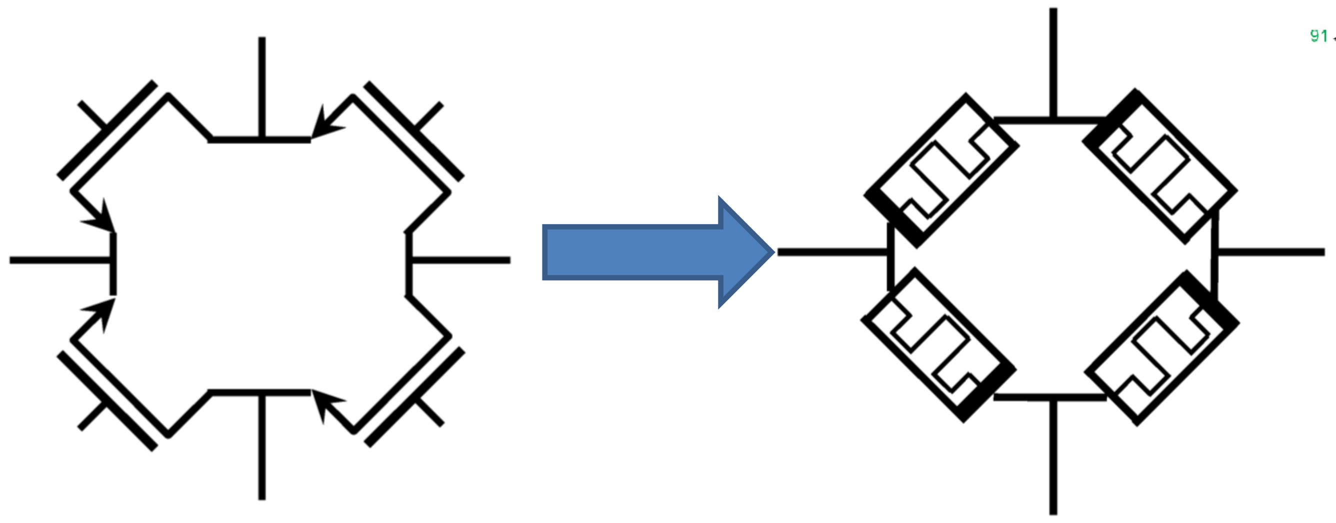
- LB – logic blocks
- CB – connection blocks
- SB – switching blocks

Cong and Xiao, “mrFPGA: A Novel FPGA Architecture with Memristor-Based Reconfiguration,” NANOARCH 2011

Switching and Connection Blocks

- Memristor as configurable switch
- 1.6X better power consumption
- 2.28X better critical path delay

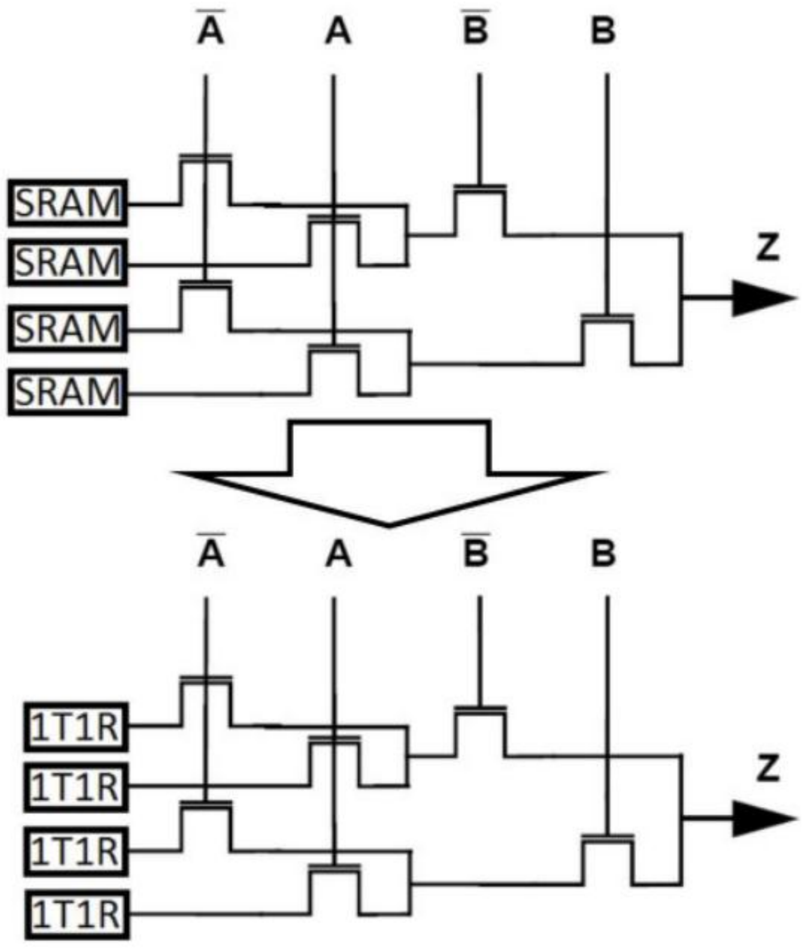




Xia et al, "Memristor-CMOS Hybrid Integrated Circuits for Reconfigurable Logic," Nano Letters 2009

Logic Block

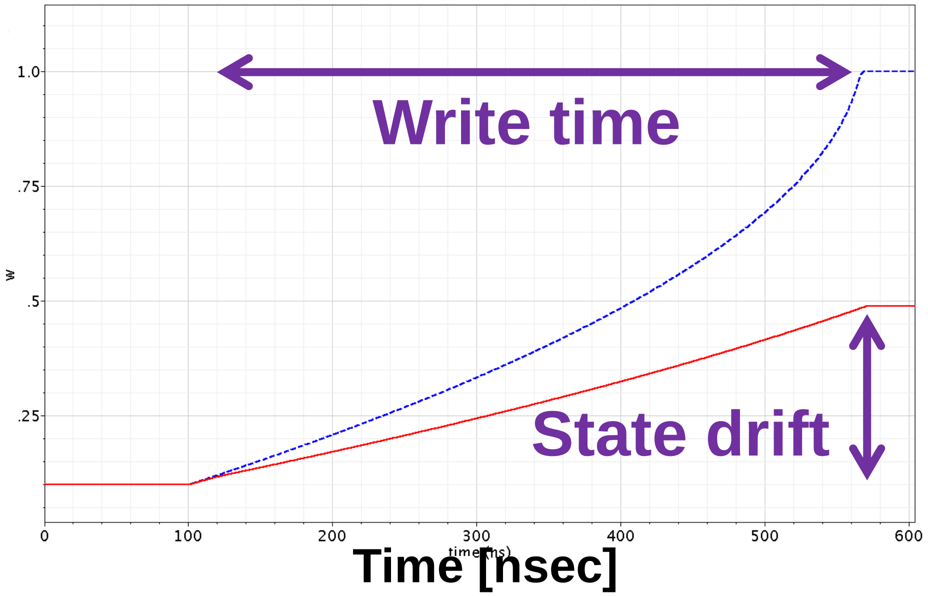
- LUT with 1T1M instead of SRAM



Liu and Wang, “rFPGA: Exploring RRAM Application in FPGA,” NANOARCH 2008

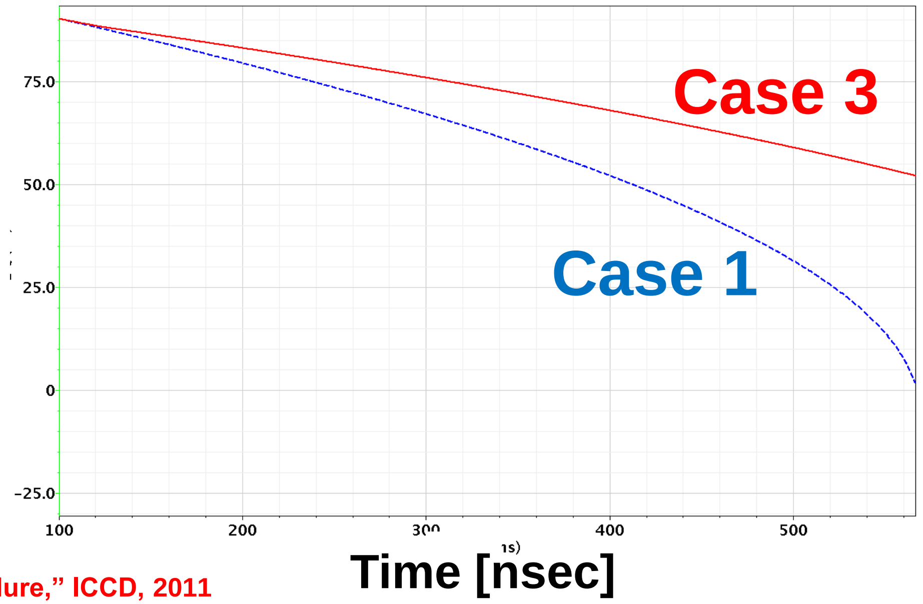
Linear Ion Drift Model

State variable - w

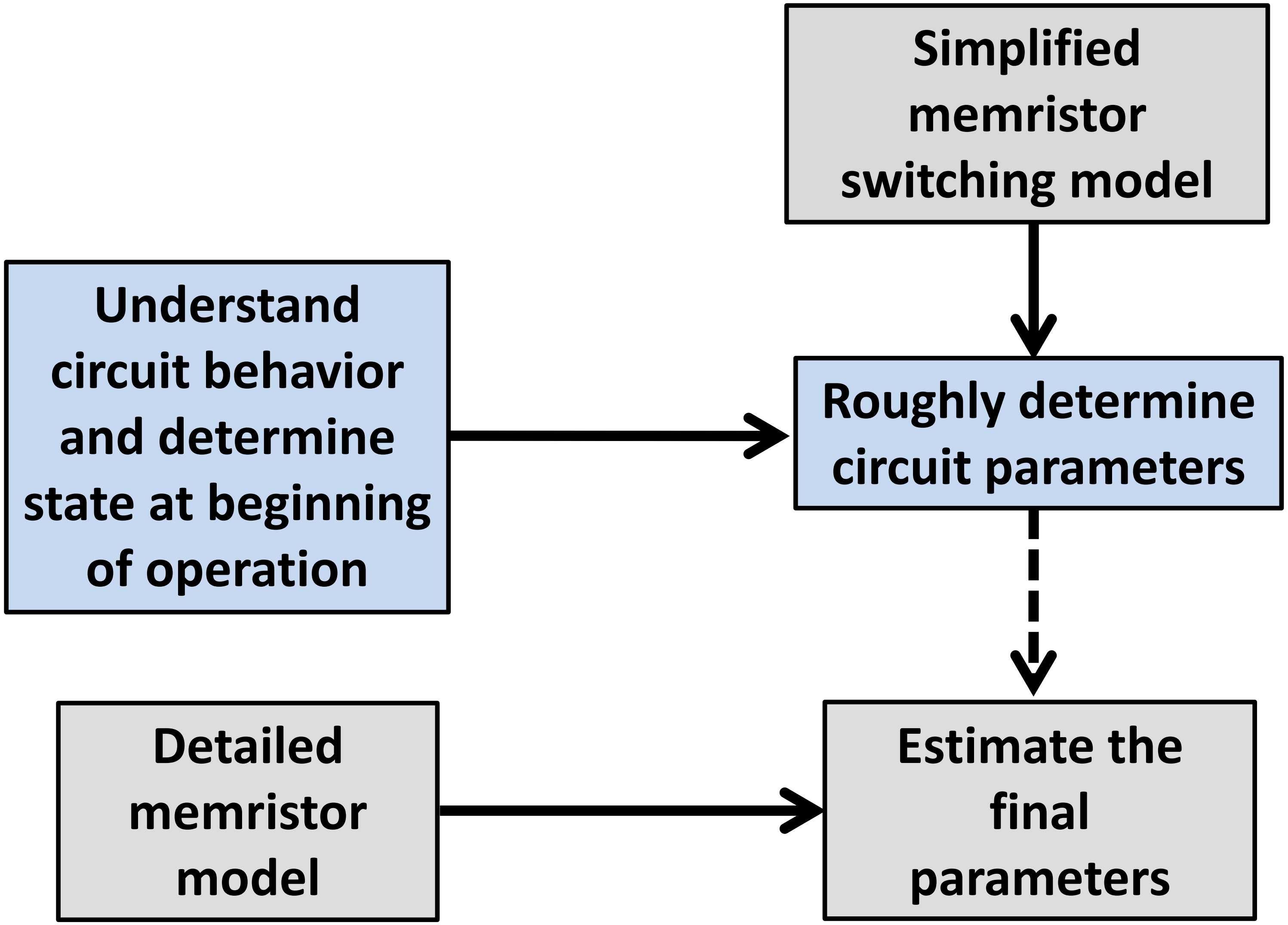


Case	p	q	*p IMP q*
1	0	0	1
3	1	0	0

Memristance M_Q

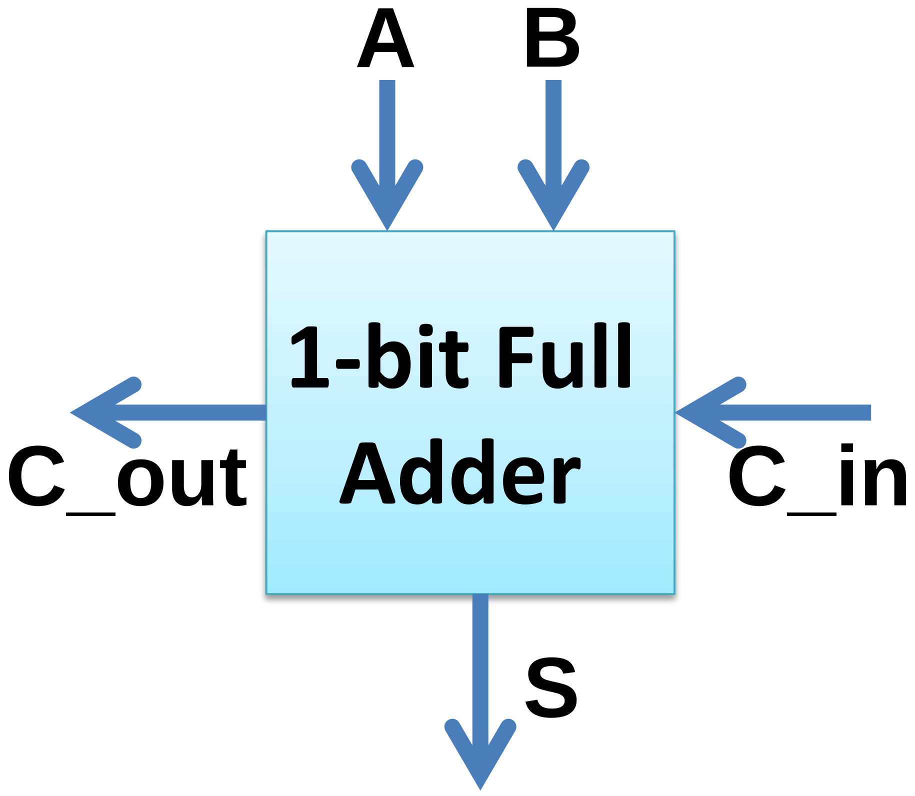


Design Flow


Simplified memristor switching model
Understand circuit behavior and determine state at beginning of operation
Roughly determine circuit parameters
Detailed memristor model
Estimate the final parameters

8-bit Full Adder Example

A	B	C_in	S	C_out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

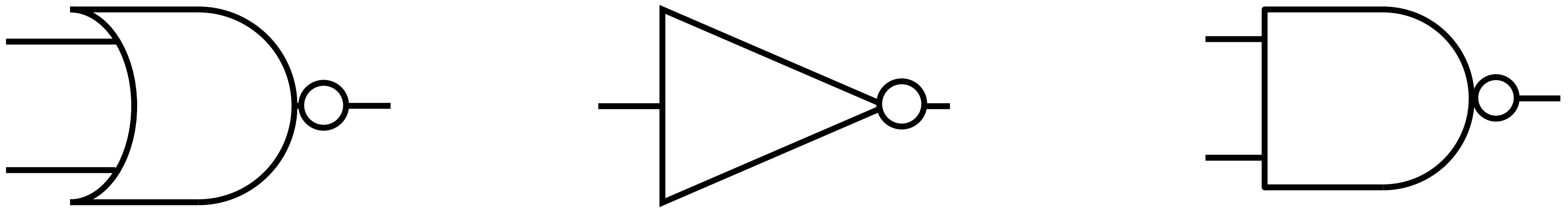


$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = A \cdot B + C_{in} \cdot \left(A \oplus B \right)$$

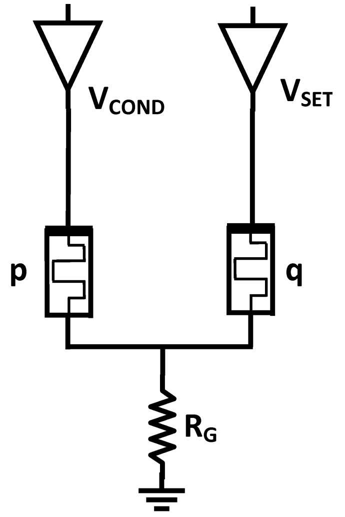
General Design Constraints

- Power consumption
- Performance – gate delay time
- Area – number of memristors (and transistors)



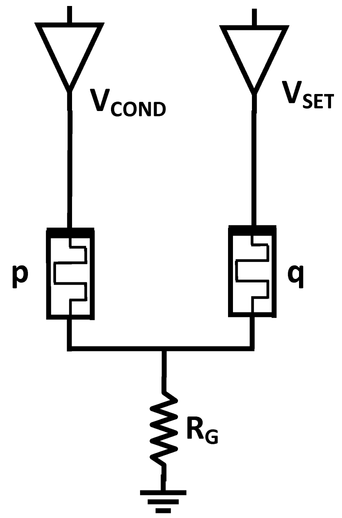
IMPLY Design Constraints

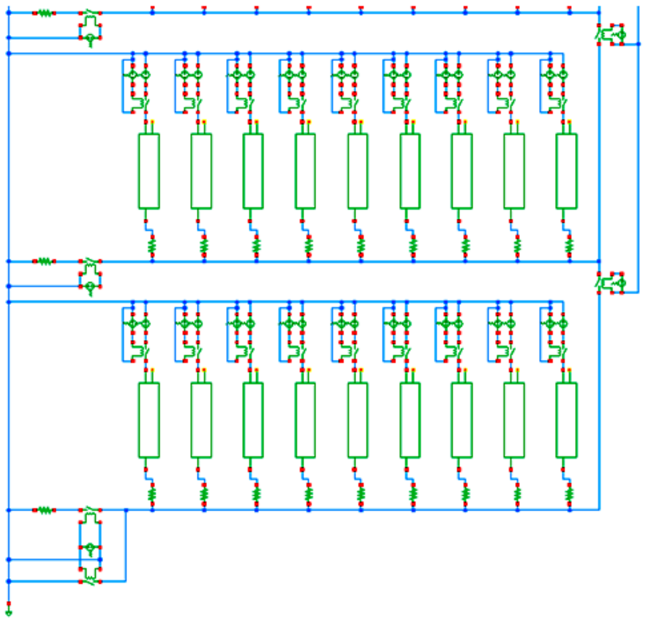
- Power – determine V_{SET} and V_{COND}
- Performance - minimize computation steps
- Area - minimize number of memristors
- **Solution – parallel computing**



8-bit IMPLY Full Adder

- Naive approach:
  - 89 computation steps per bit
  - 3 memristors per bit + 5 memristors
- Improved approach:
  - Parallel computing, scheduling
  - 5 computation steps per bit (+18)
  - 9 memristors per bit (72 total)

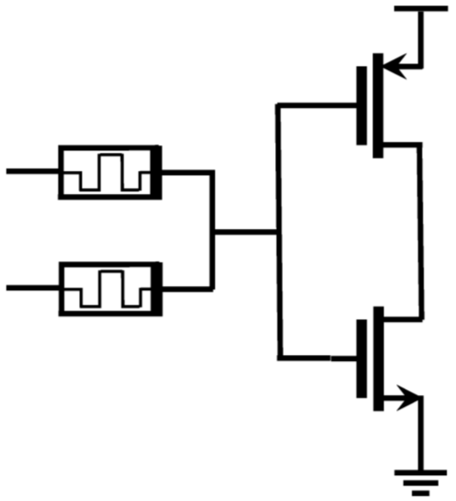




Wald and Satat B.Sc. Project, Technion 2012

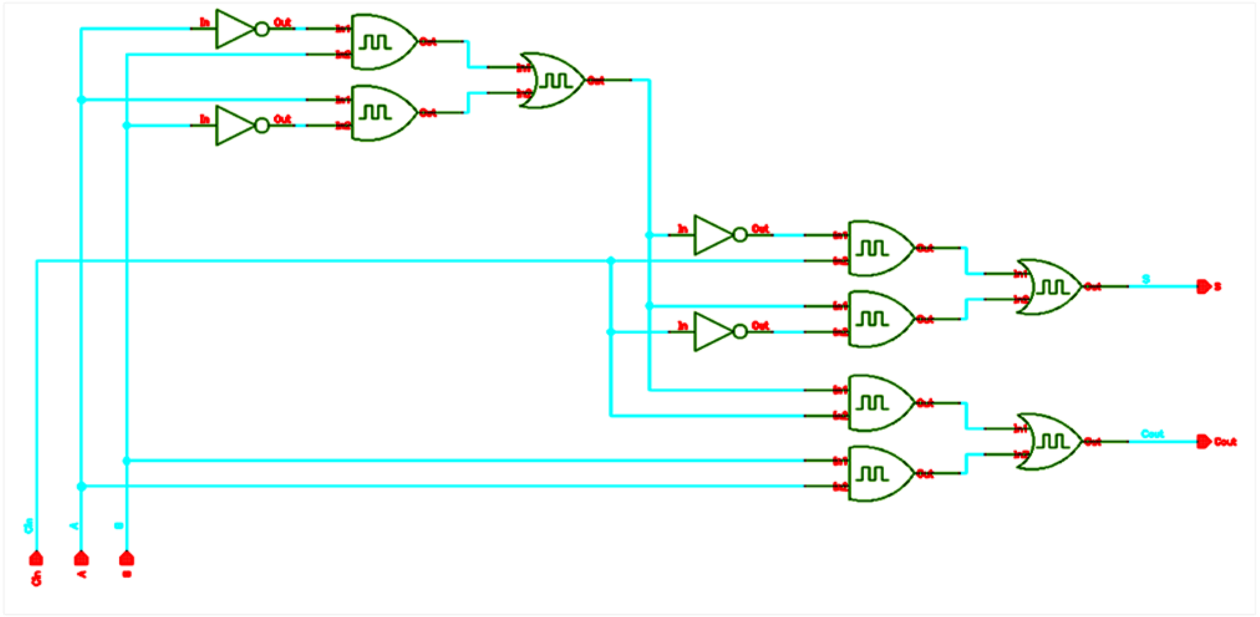
Hybrid CMOS-Memristor Design Constraints

- Minimize number of CMOS-memristor transitions (number of vias)
- **Solution: use inverter (or buffers) only when necessary**



8-bit Hybrid CMOS-Memristor Full Adder

- 144 memristors
- Area and vias is depended on memristor behavior:
  - Linear memristor – 160 transistors, 80 vias
  - Nonlinear memristor- 256 memristors, 96 vias



Design Summary

	IMPLY	Hybrid CMOS-Memristor
Performance	Sequential 58 steps	6X faster 1 step
Area – memristor layer	72 memristors 2X smaller	144 memristors
Area – CMOS layer	Controller	Memristor behavior 80-96 transistors
Power	Dynamic power	Static and dynamic More power