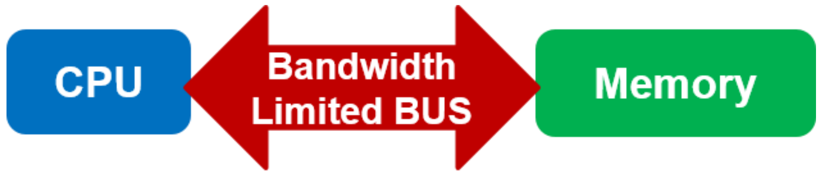
LOSSS - Logic Synthesis based on Several Stateful logic gates for high time-efficient computing

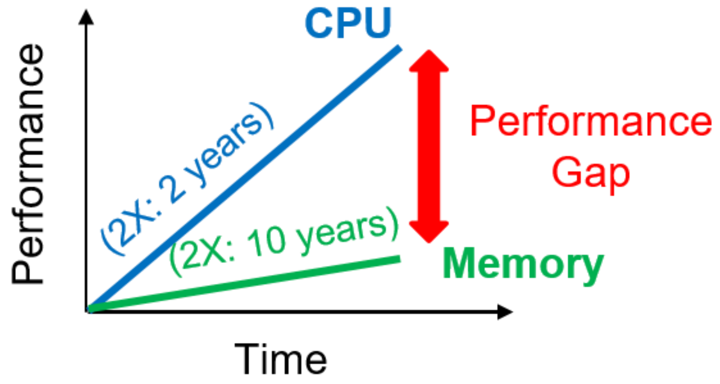
Yihong Hu[1] , Nuo Xu[1,2,*] , Chaochao Feng[1,2] , Wei Tong[3] , Kang Liu[1] , Liang Fang [1]

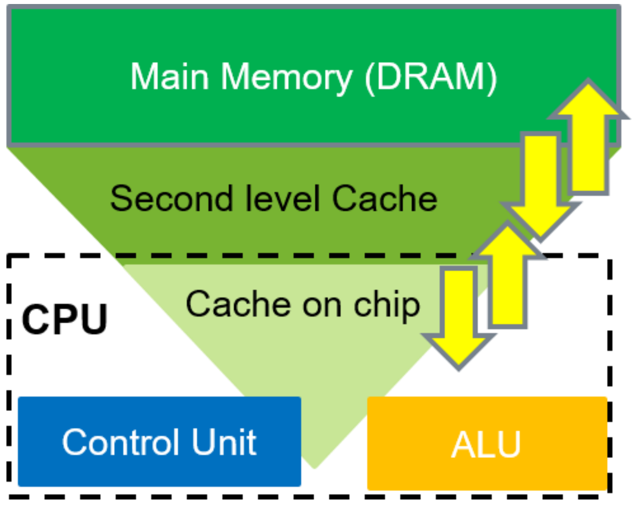
[1] College of Computer, National University of Defense Technology, Changsha 410073, China
[2] Key Laboratory of Advanced Microprocessor Chips and Systems, Changsha 410073, China
[3] Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074, China

Email：xunuo@nudt.edu.cn

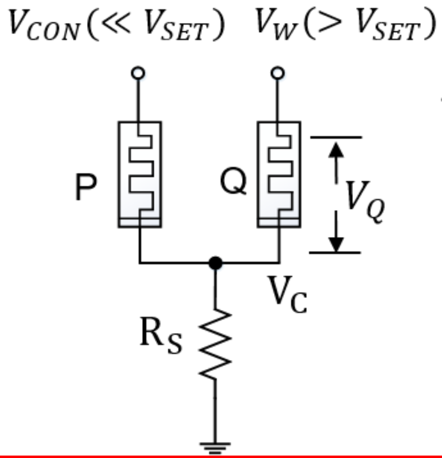
Introduction



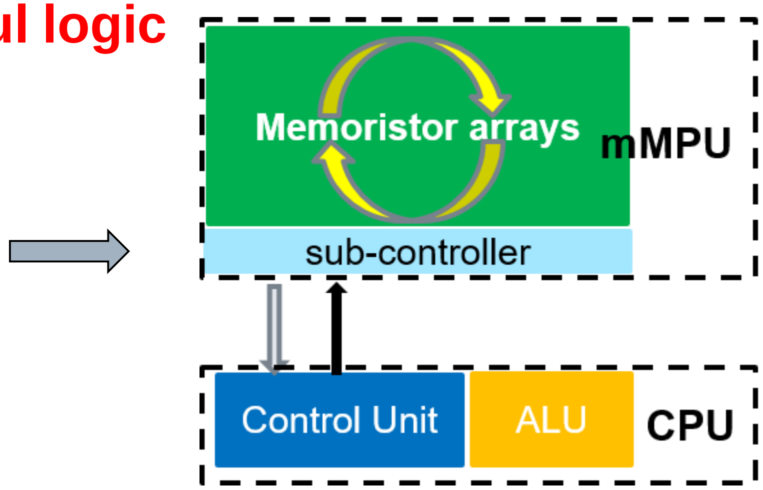




In-memory stateful logic



p	q	V_Q	q'
0	0	V_W	1
0	1	V_W	1
1	0	$V_W - V_{COND}$	0
1	1	$\frac{V_W - V_{COND}}{2}$	1

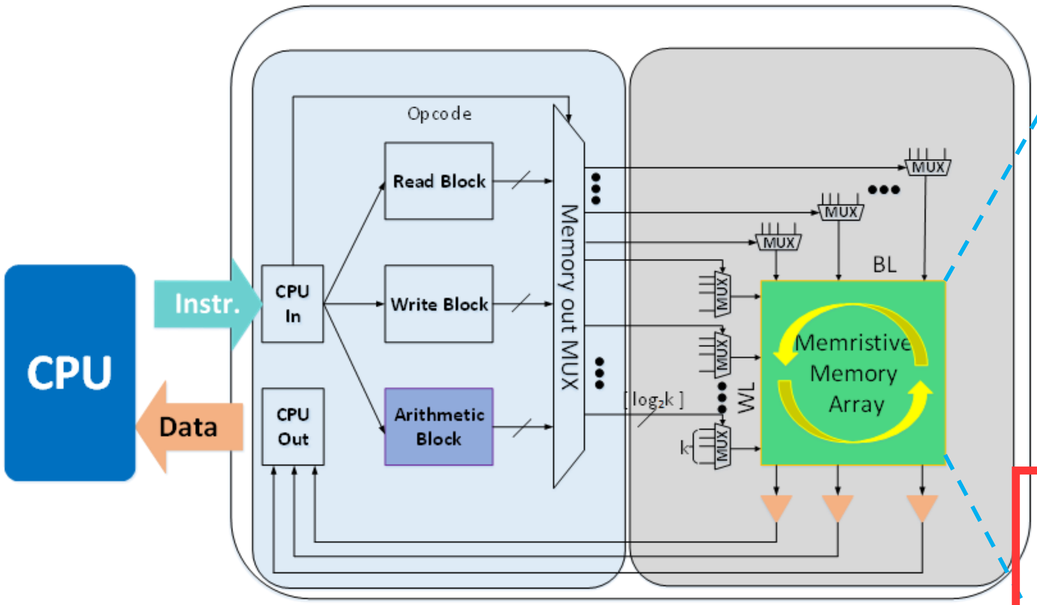


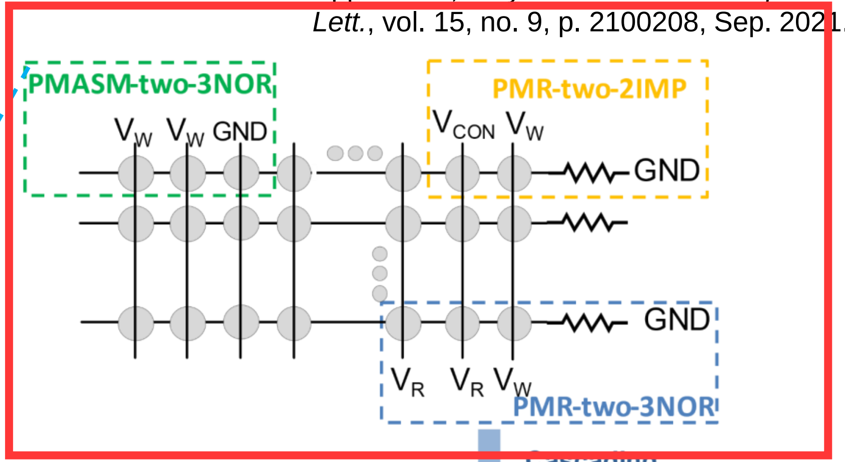
N. Xu, T. Park, K. J. Yoon, and C. S. Hwang, “In-Memory Stateful Logic Computing Using Memristors: Gate, Calculation, and Application,” *Phys. status solidi – Rapid Res. Lett.*, vol. 15, no. 9, p. 2100208, Sep. 2021.

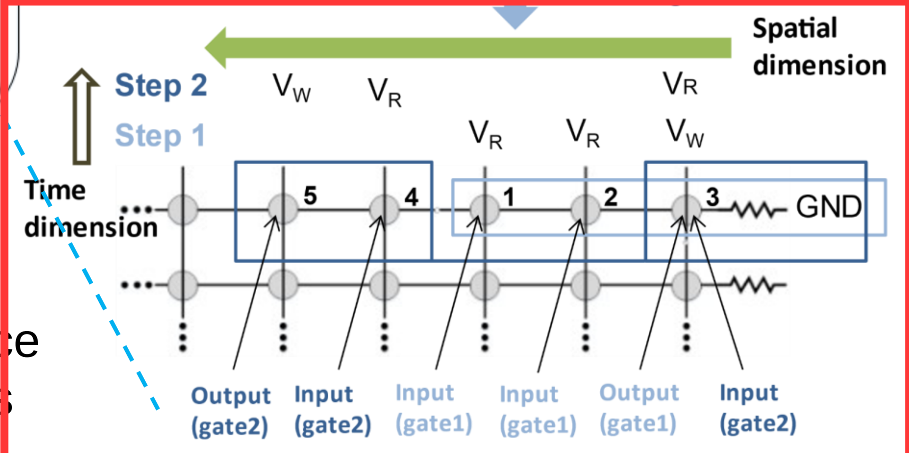
Introduction

- **In-memory stateful logic computing**

N. Xu, T. Park, K. J. Yoon, and C. S. Hwang, "In-Memory Stateful Logic Computing Using Memristors: Gate, Calculation, and Application," *Phys. status solidi – Rapid Res. Lett.*, vol. 15, no. 9, p. 2100208, Sep. 2021.





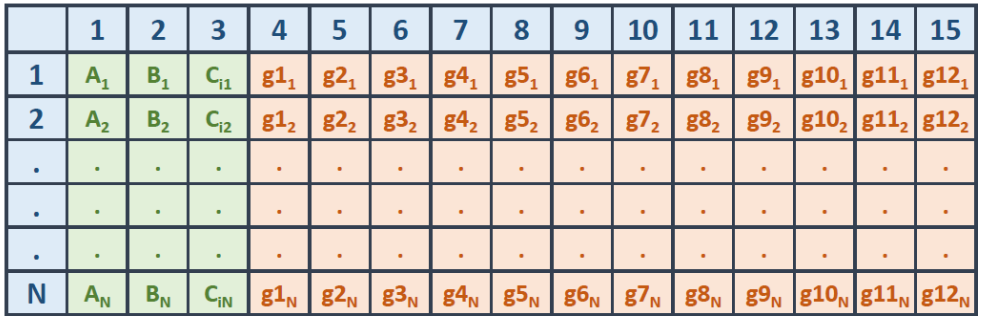


Complete complex computing in stateful logic paradigm

- **Synthesis and mapping of stateful logic**
  - ✓ Automatically get the cascade sequence for a given complex computing process
  - ✓ Optimization for reducing the number of the gates (or steps)

Introduction

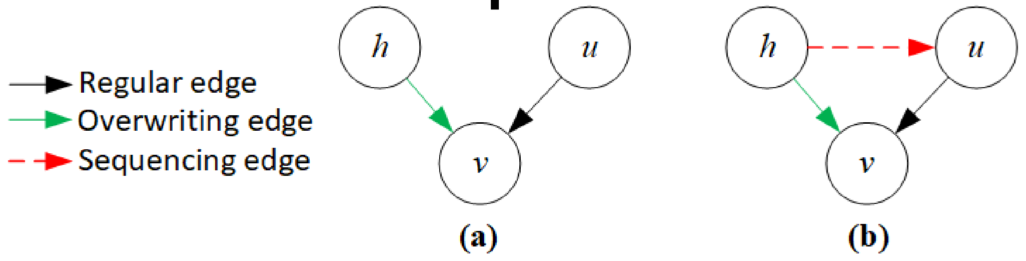
- **SIMPLER MAGIC**
  - NOR, NOT(reset)



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	A_1	B_1	C_{i1}	$g1_1$	$g2_1$	$g3_1$	$g4_1$	$g5_1$	$g6_1$	$g7_1$	$g8_1$	$g9_1$	$g10_1$	$g11_1$	$g12_1$
2	A_2	B_2	C_{i2}	$g1_2$	$g2_2$	$g3_2$	$g4_2$	$g5_2$	$g6_2$	$g7_2$	$g8_2$	$g9_2$	$g10_2$	$g11_2$	$g12_2$
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
N	A_N	B_N	C_{iN}	$g1_N$	$g2_N$	$g3_N$	$g4_N$	$g5_N$	$g6_N$	$g7_N$	$g8_N$	$g9_N$	$g10_N$	$g11_N$	$g12_N$

R. Ben-Hur *et al.*, "SIMPLER MAGIC: Synthesis and Mapping of In-Memory Logic Executed in a Single Row to Improve Throughput," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2434–2447, Oct. 2020

- **X-MAGIC**
  - NOR, NOT, $\overline{A+B}\cdot C$, $\overline{A}\cdot B$ (reset)
  - **Deal with the issue of input overwritten**



N. Peled, R. Ben-Hur, R. Ronen, and S. Kvatinsky, "X-MAGIC: Enhancing PIM Using Input Overwriting Capabilities," in *2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC)*, 2020, pp. 64–69.

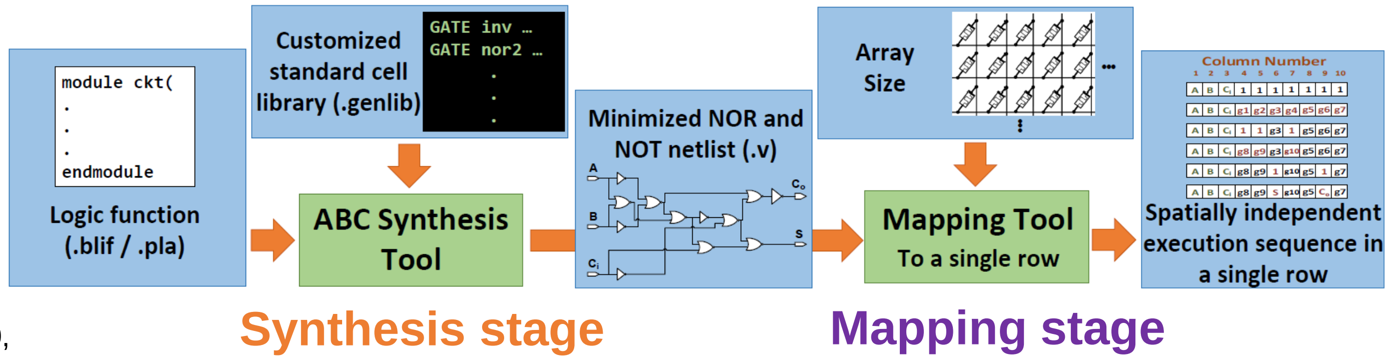
- **LOSSS (this work)**
  - COPY, NOT, NOR, OR, **IMP, ONOR**
  - **Avoid input overwritten through smart merge strategies**
  - Cell allocation for **set** and **reset** gates respectively

Methods and Procedures

- **Synthesis and mapping flow of LOSSS**

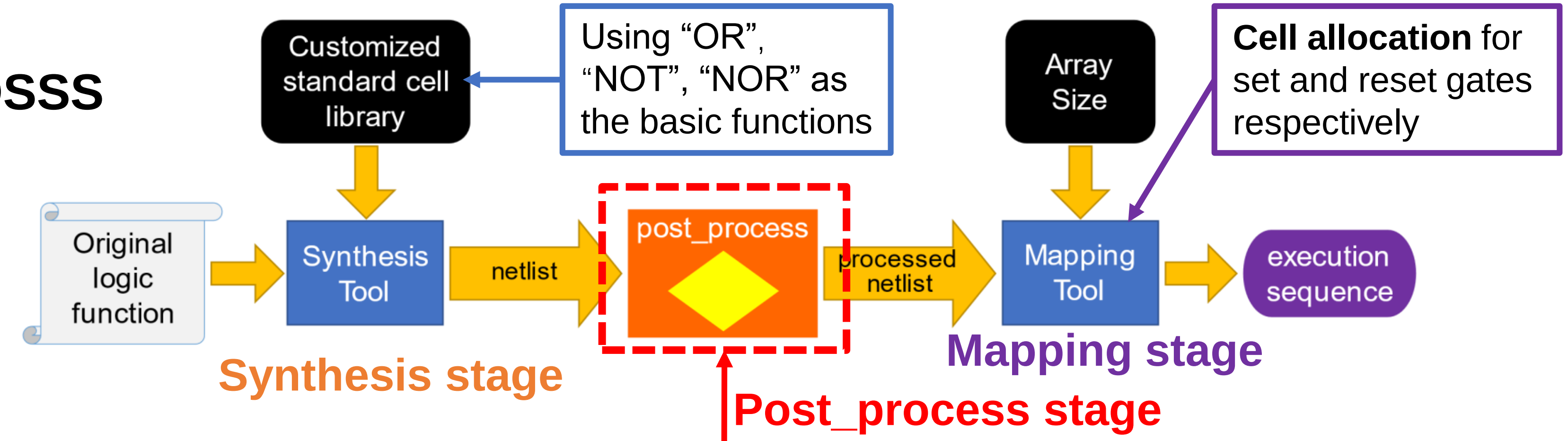
SIMPLER

R. Ben-Hur *et al.*, “SIMPLER MAGIC: Synthesis and Mapping of In-Memory Logic Executed in a Single Row to Improve Throughput,” *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2434–2447, Oct. 2020.



module ckt(
endmodule
Logic function (.blif / .pla)
Customized standard cell library (.genlib)
GATE inv …
GATE nor2 …
ABC Synthesis Tool
Minimized NOR and NOT netlist (.v)
Array Size
Mapping Tool To a single row
Column Number
Spatially independent execution sequence in a single row
Synthesis stage
Mapping stage

LOSSS



Netlist Optimization

NOR (NOT) + OR → ONOR (IMP)

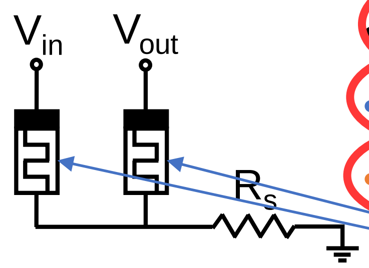
NOR (NOT) + NOR → ONOR (IMP)+NOT

NOT + NOT → NOT

Deal with Cyclic dependency

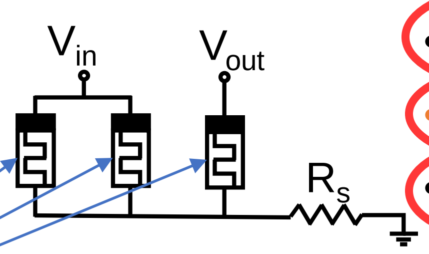
Bring in “COPY”

Methods and Procedures

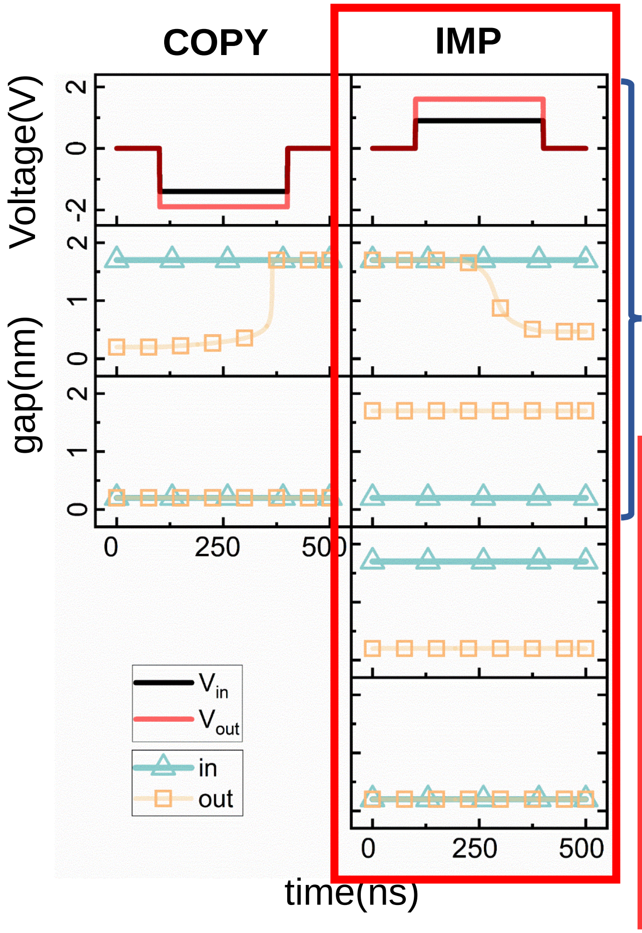


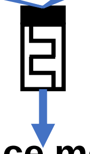
- NOT
- COPY
- IMP

- **Compatibility Verification for Stateful Logic Gates**



- NOR
- ONOR
- OR



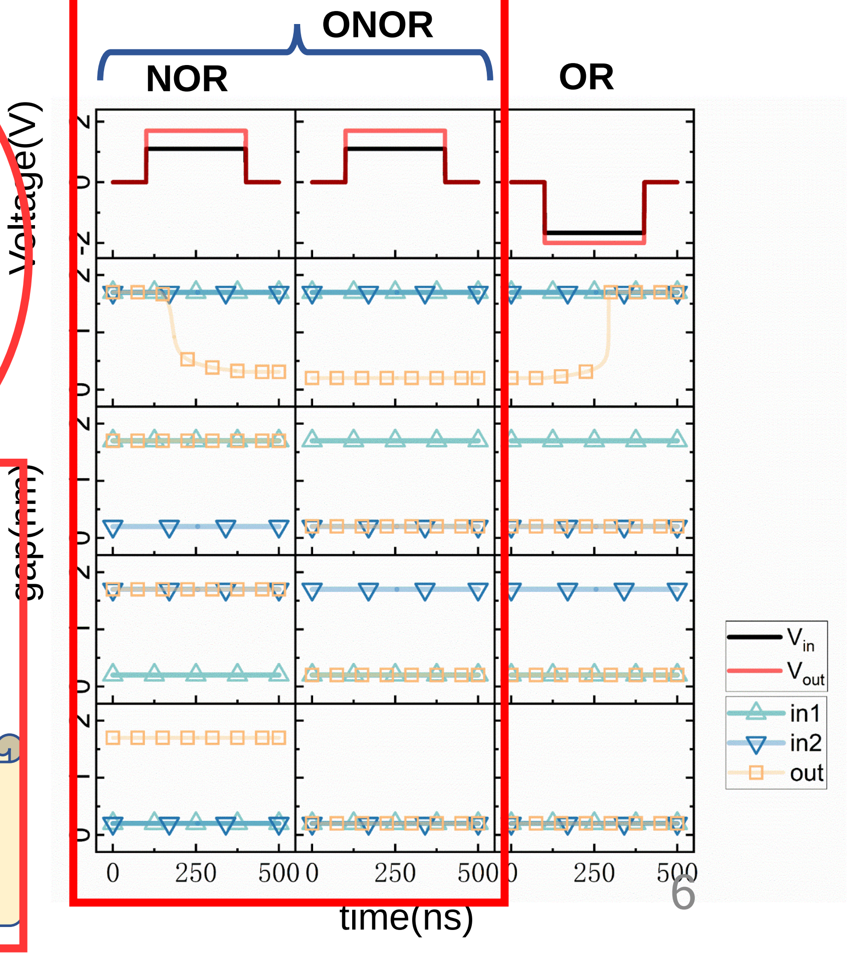


Device models with the same switching parameters (from a crossbar array)

Original logic function

CMOS synthesis with constrain on area

Netlist with logic functions of NOT, NOR, and OR

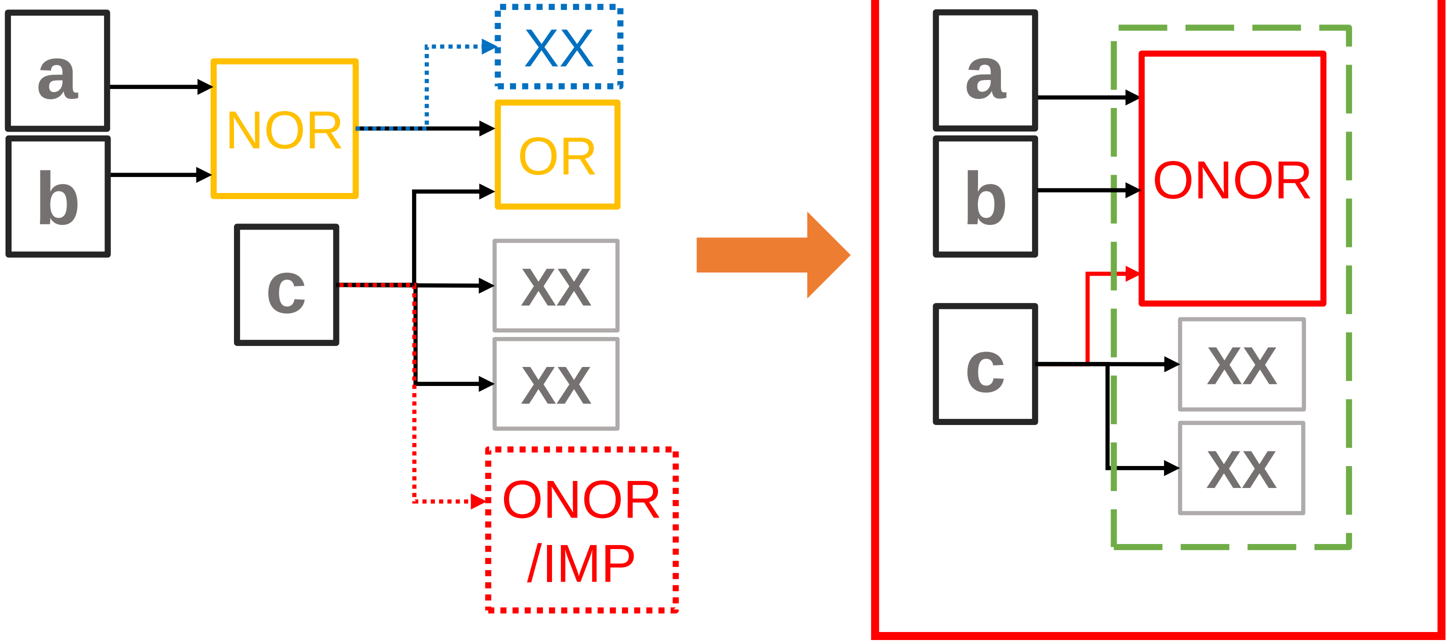


Methods and Procedures

- **Optimization algorithms in Post_process stage**

NOR(NOT)+OR → ONOR(IMP)

First case



Netlist with logic functions of NOT, NOR, and OR ⇒ **Netlist with logic functions of NOT, NOR, OR, ONOR, and IMP**

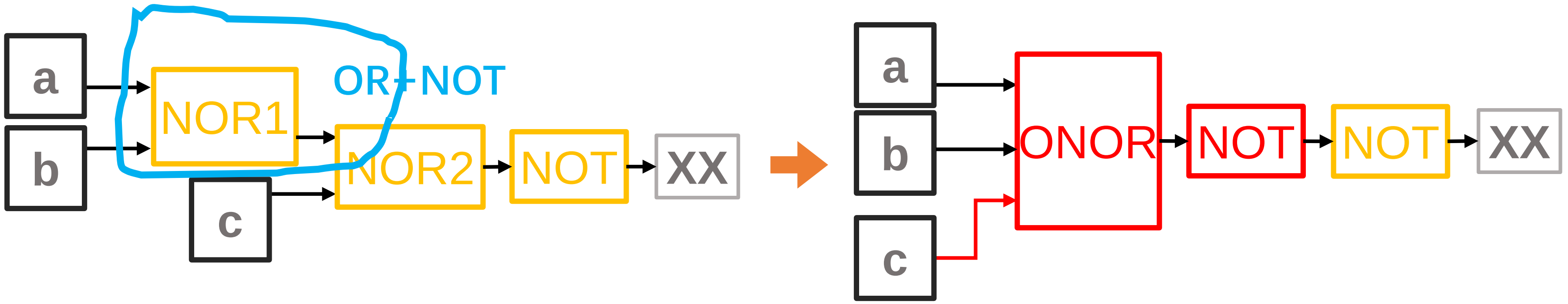
The number of the gates is reduced

Methods and Procedures

- **Optimization algorithms in Post_process stage**

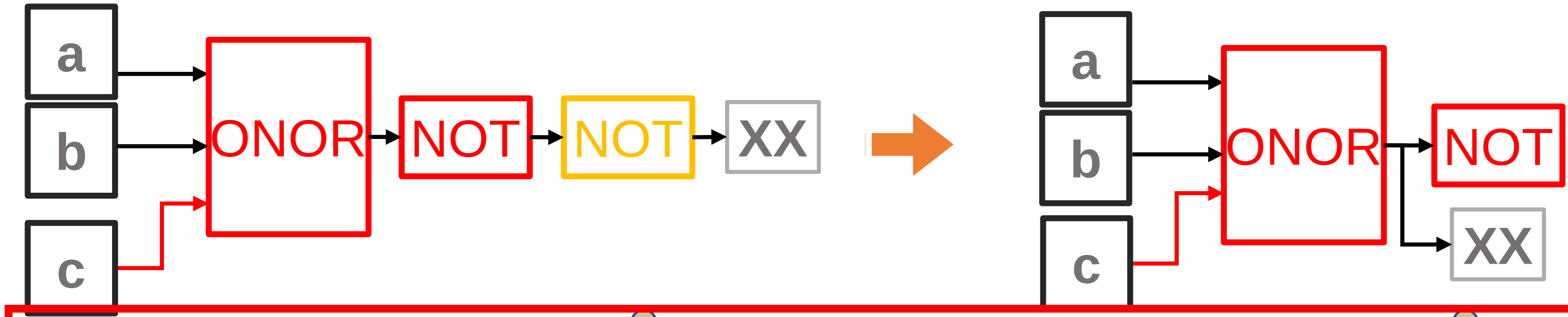
NOR(NOT)+NOR → ONOR(IMP)+NOT

Second case



NOT + NOT→NOT

Third case

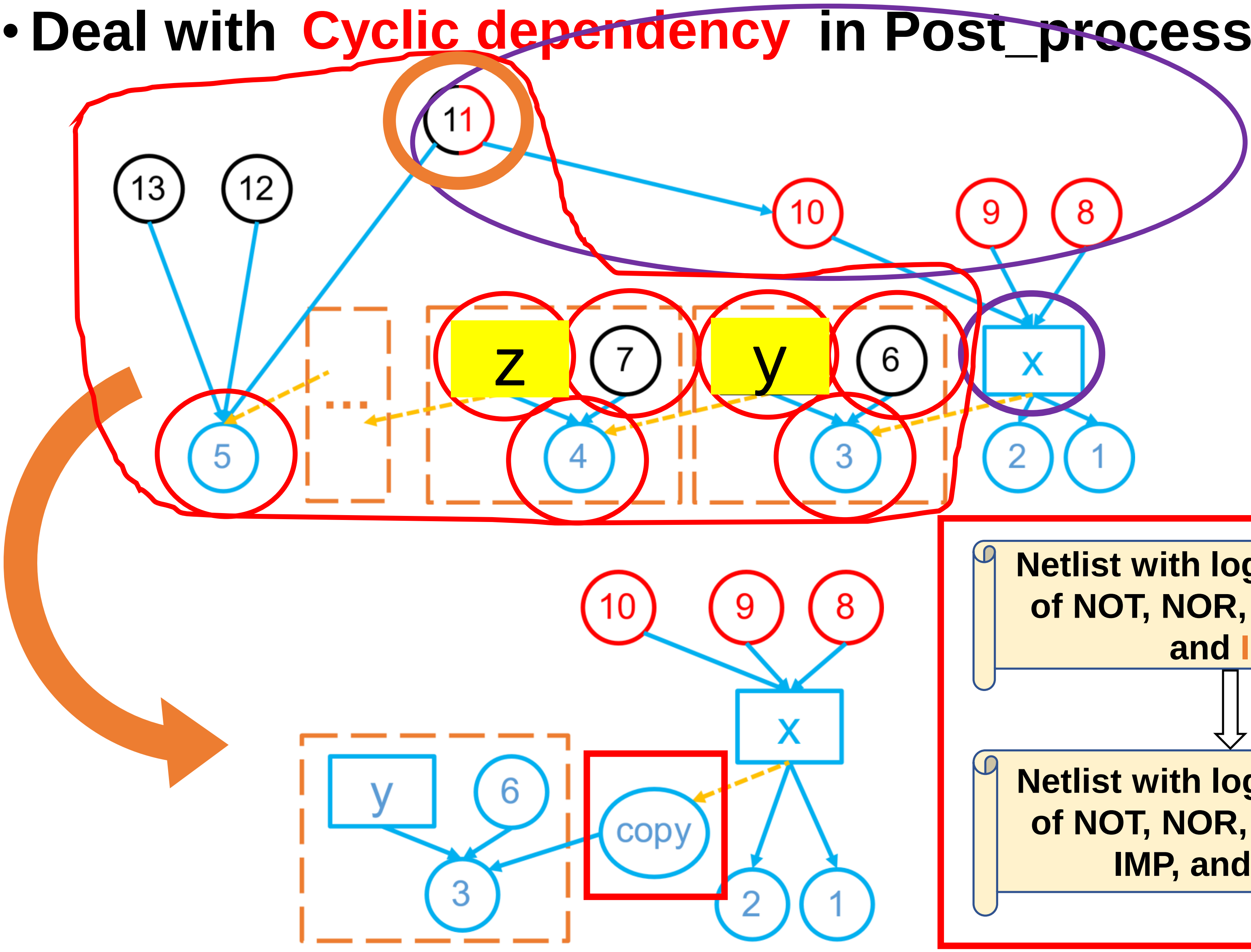


Netlist with logic functions of NOT, NOR, and OR ⇒ **Netlist with logic functions of NOT, NOR, OR, ONOR, and IMP**

The number of the gates is reduced

Methods and Procedures

- Deal with **Cyclic dependency** in Post_process stage



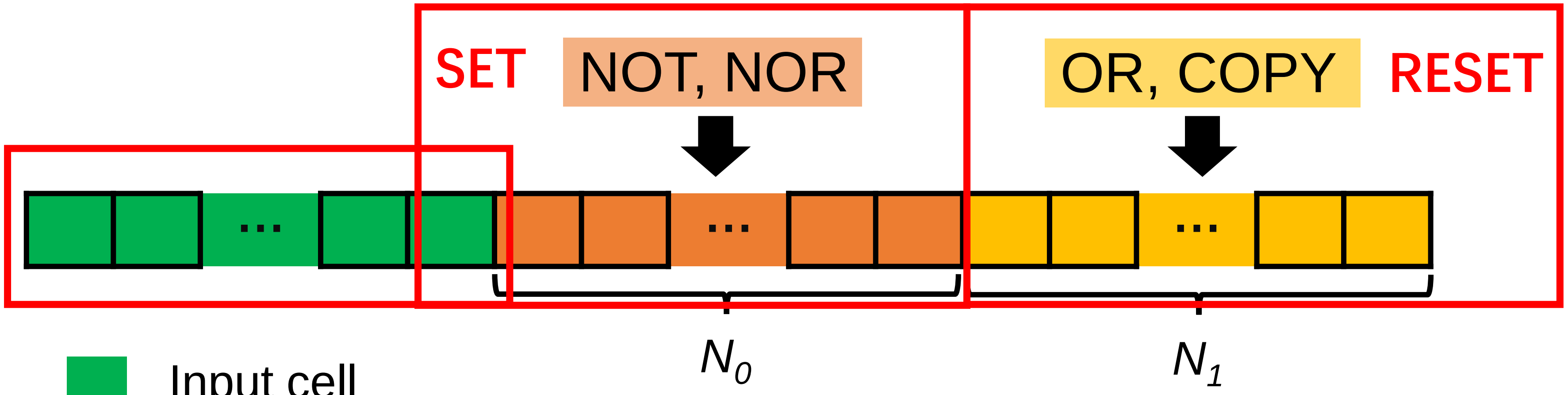
Netlist with logic functions of NOT, NOR, OR, ONOR, and IMP

⇩

Netlist with logic functions of NOT, NOR, OR, ONOR, IMP, and COPY

Methods and Procedures

- **Cell allocation strategies in mapping stage**



🟩 Input cell

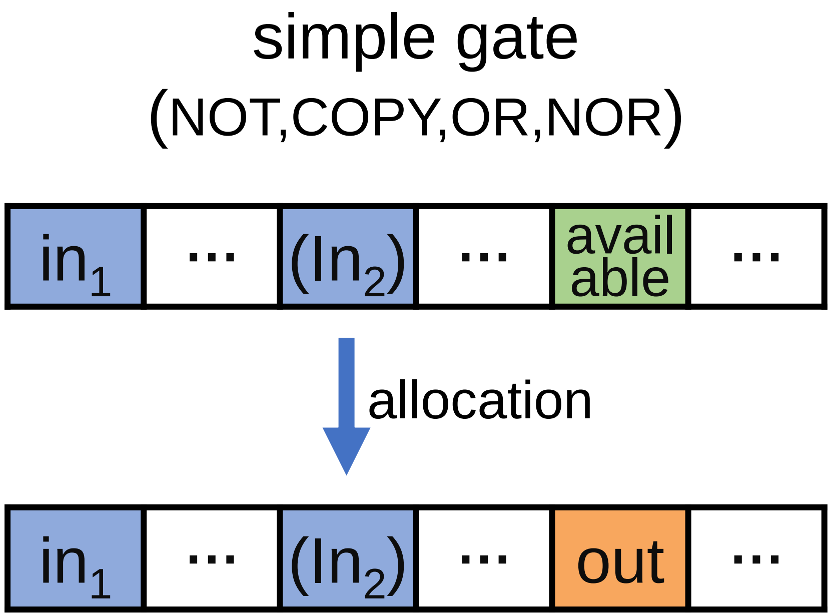
🟧 Initialized to 0 ⎫ Intermediate and output cell

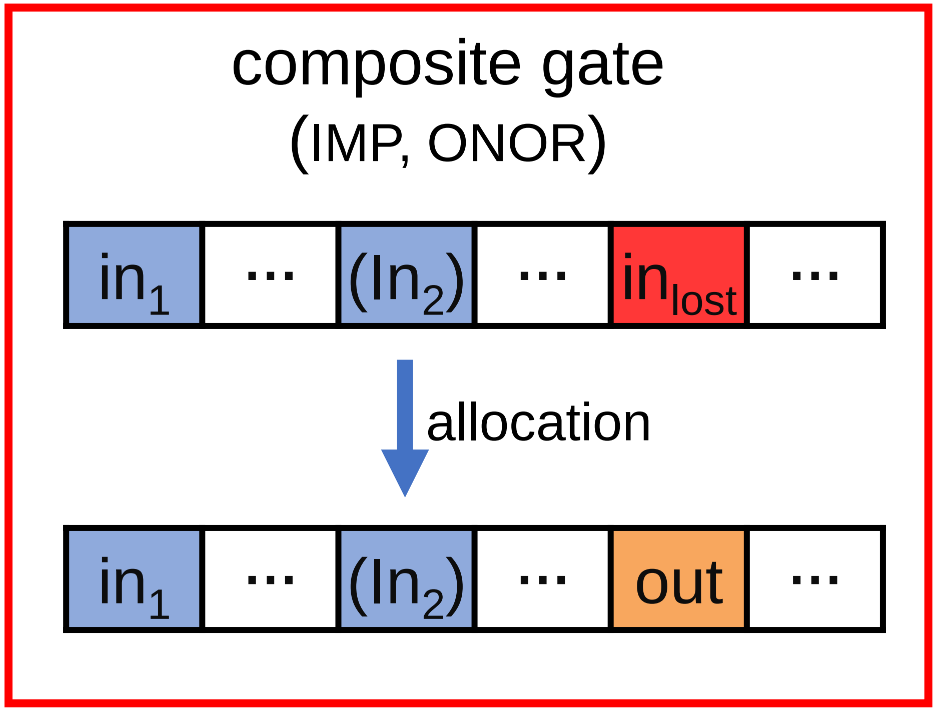
🟨 Initialized to 1 ⎭

$$\frac{N_0}{N_0+N_1}=\left(\frac{N_{\text{set\_gate}}}{N_{\text{gate}}}+\frac{\max\{CU\_of\_set\_gate\}}{\max\{CU\_of\_set\_gate\}+\max\{CU\_of\_reset\_gate\}}\right)/2$$

Methods and Procedures

- **Cell allocation strategies in mapping stage**





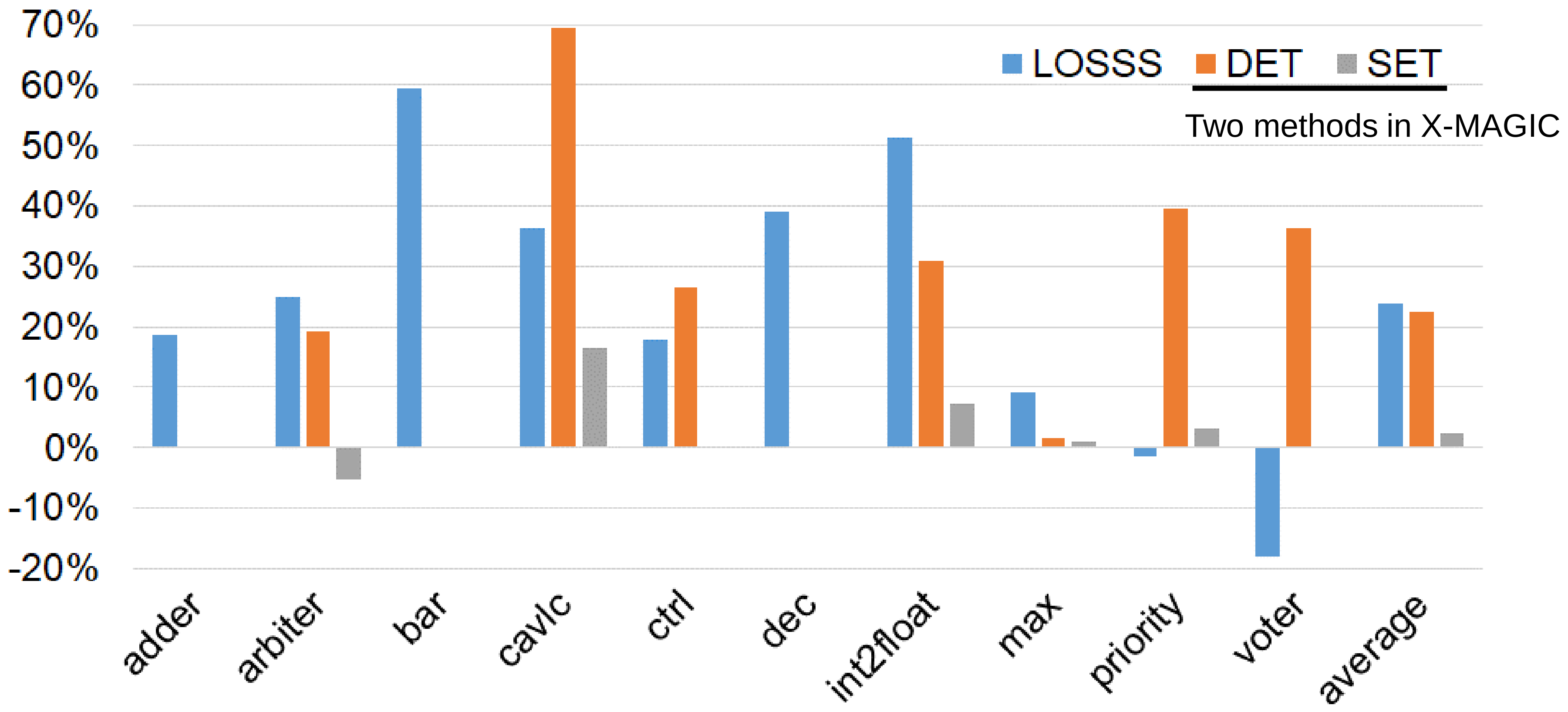
Cell reuse is similar to SIMPLER
(BUT divided into two cases of being initialized to 0 and 1)

Evaluation and Results

Name	inputs	outputs	LUT-6 count	levels	Row size		Max value of row size	SIMPLER MAGIC		LOSSS	
					SIMPLER MAGIC	LOSSS		cycles	operation	cycles	operation
adder	256	129	254	51	390	463	463	1542	3060	1225	1969
arbiter	256	129	2722	18	1719	2147	2147	7659	15296	5599	9819
bar	135	128	512	4	399	636	636	5301	10568	2897	4726
cavlc	10	11	122	4	124	169	169	896	1768	681	1115
ctrl	7	26	29	2	45	53	53	163	308	132	218
dec	8	256	287	2	267	371	371	361	720	314	624
int2float	11	7	49	3	41	62	62	269	520	194	312
max	512	130	842	56	783	854	854	3803	7554	3043	4962
priority	128	8	210	31	194	191	194	821	1552	655	1117
voter	1001	1	2691	16	1354	1110	1354	13648	27100	11532	18393

Evaluation and Results

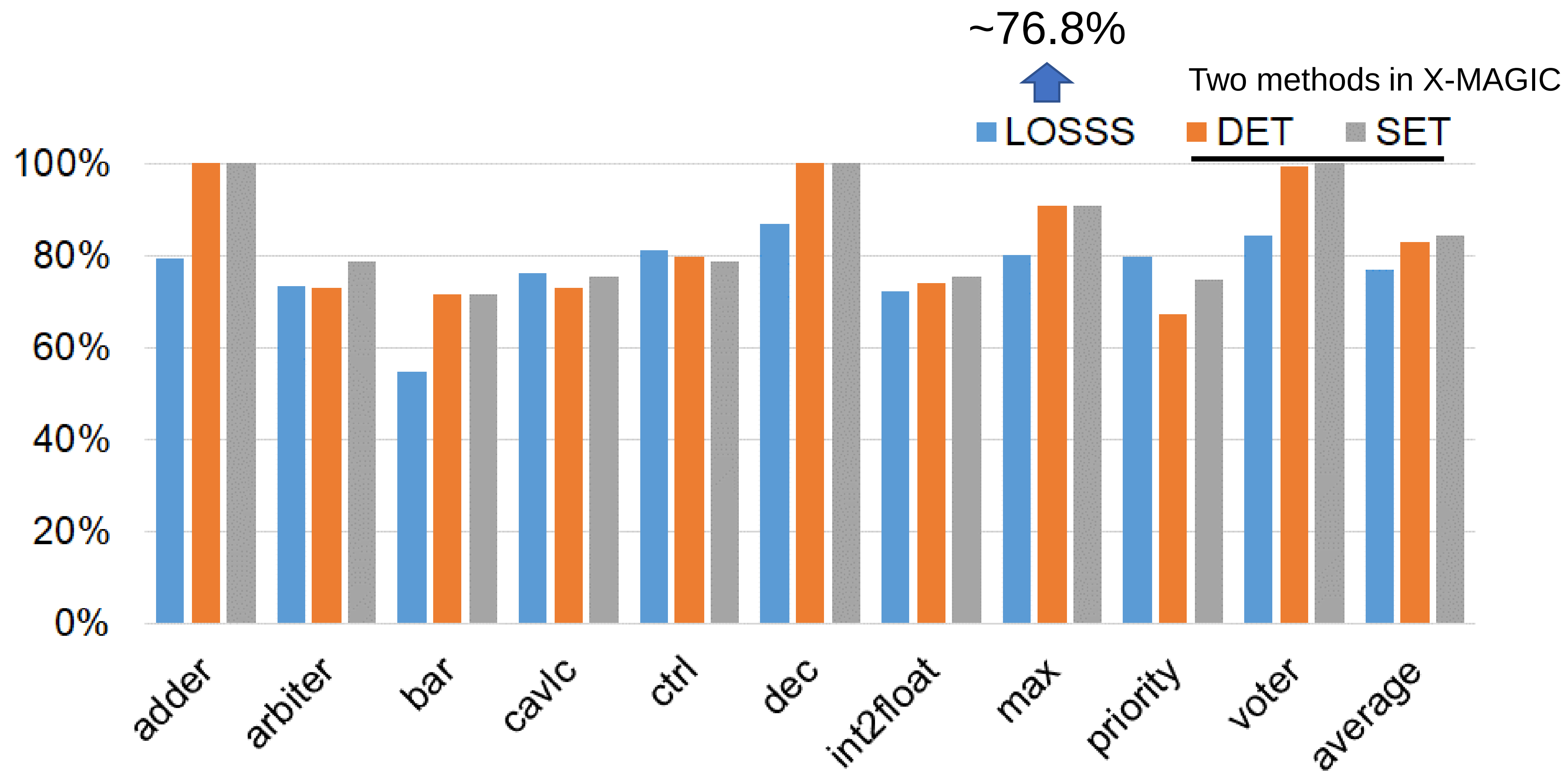
- **Area increase relative to SIMPLER**



(Lower is better)

Evaluation and Results

- **Latency relative to SIMPLER**

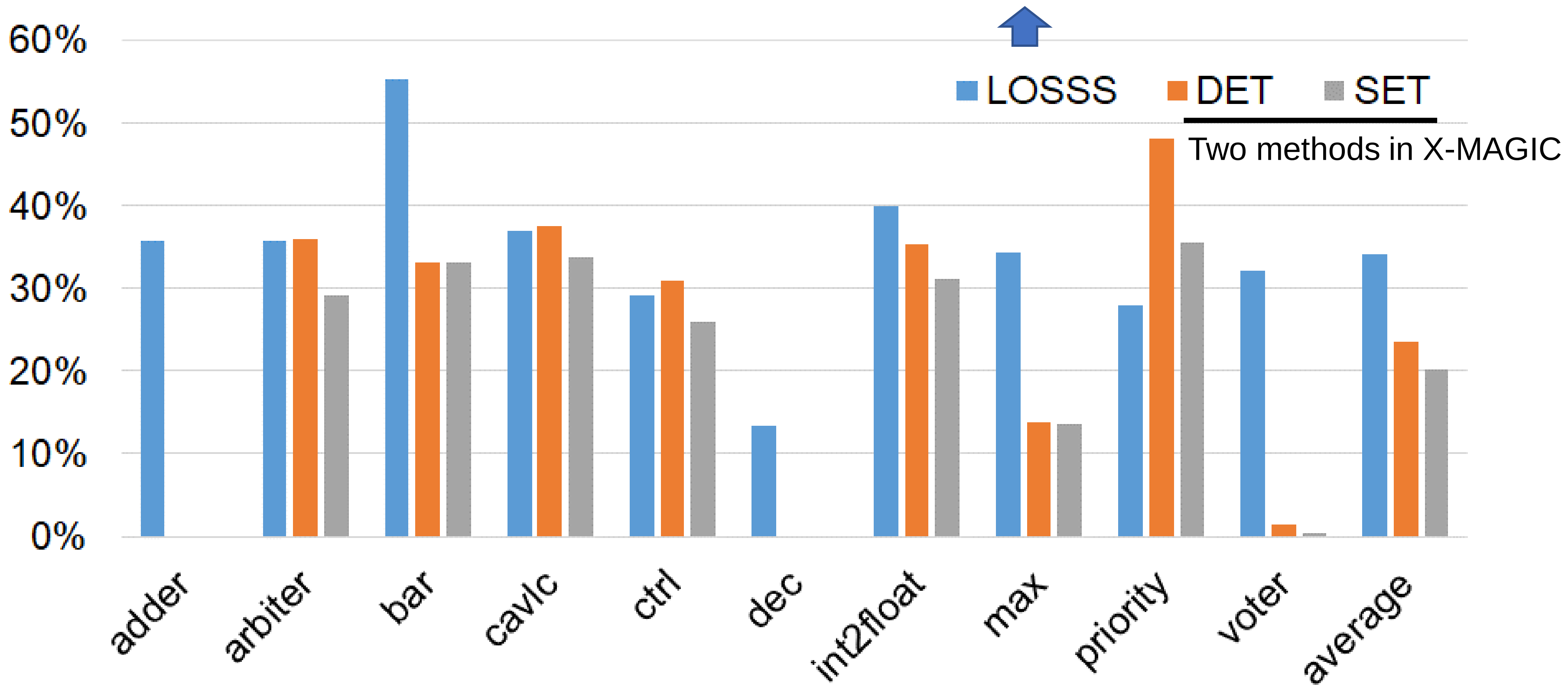


(Lower is better)

Evaluation and Results

- **Lifetime increase relative to SIMPLER**

~34% improvement



(Higher is better)

THANKS

Q&A

xunuo@nudt.edu.cn