In-memory Computing with Majority RRAM Operations

Pierre-Emmanuel Gaillardon
Department of Electrical and Computer Engineering – University of Utah

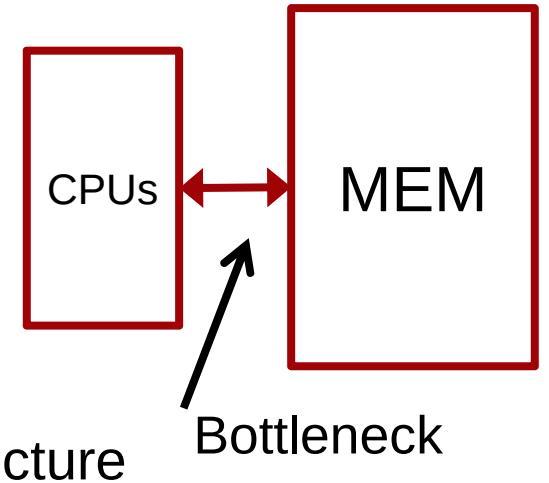


D43D Workshop
June 27, 2017 – Grenoble, France

How to Break the Memory Bottleneck?

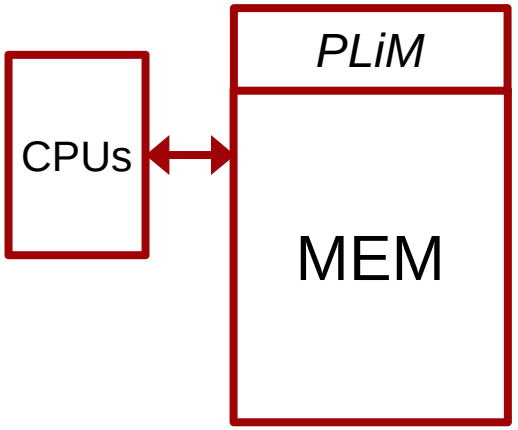
Emerging memories are highly promising devices:

- Non-volatile storage (1-bit or multi-bit)
- Energy improvements – Higher performance
- High density of integration – Low fabrication cost



No matter how good the memories are, the architecture is at the root of the problem

Explored Solution:
Offload some processing tasks
to the memory
→ **Programmable Logic-in-Memory (*PLiM*)**



PLiM = A low-cost distributed memory system capable of operating on its own content

© LNIS-UofU 2017. All rights reserved.

Resistive-MAJ In-memory Computing

- **Resistive-majority operations and PLiM computer**
- Majority-based logic manipulation
- Logic manipulation for PLiM
- Conclusion

© LNIS-UofU 2017. All rights reserved.

Our Objectives

- **Our problem:**

  How to realistically exploit the native Boolean computation property?

- **Our solution:**

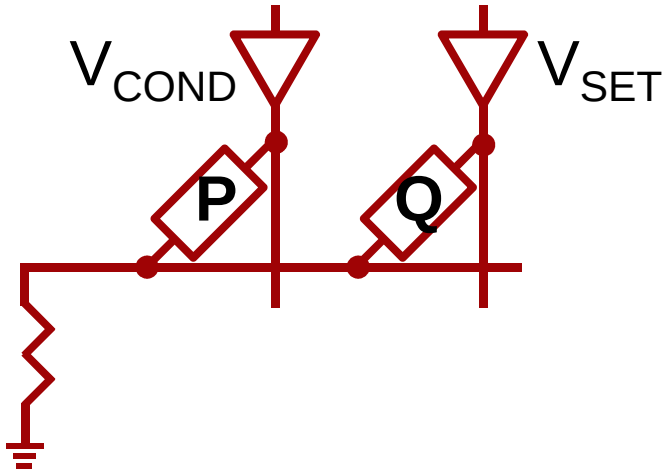
  A low-cost distributed memory system capable of operating at low-power on its own content

  – Use the native logic capabilities of RRAM to perform in-memory computation
  – Address the question of control for the different computing elements at low cost

© LNIS-UofU 2017. All rights reserved.

Material Implication

There exists a major body of work on material implication



p	q	q_n
0	0	1
0	1	1
1	0	0
1	1	1

Theoretically interesting
but practical aspects are commonly overlooked

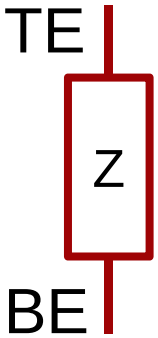
Open questions:

- How to deploy the complex signals involved with the operation?
- How to implement a dual-memory select with common ground?
- How to schedule operations?

Can we go beyond material implication?

© LNIS-UofU 2017. All rights reserved.

In-memory Majority-oriented Logic



$V_{TE,BE} > V_{th}$

→ SET Operations

→ Z = 1

TE	BE	Z	Z_n
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	0

$Z_n = TE \,.\, BE'$

$V_{TE,BE} < -V_{th}$

→ RESET Operations

→ Z = 0

TE	BE	Z	Z_n
0	0	1	1
0	1	1	0
1	0	1	1
1	1	1	1

$Z_n = TE + BE'$

$Z_n =$ $(TE \,.\, BE') \,.\, Z' + (TE + BE') \,.\, Z =$ **MAJ (TE, BE', Z)**

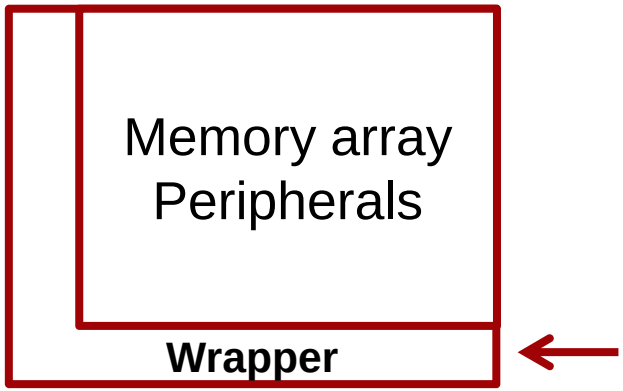
RRAM devices act as MAJ operators!

(In-memory computing with a functionally complete operator RM_3)

© LNIS-UofU 2017. All rights reserved.

The PLiM Paradigm

How can we provide additional computing capabilities to regular memory arrays?



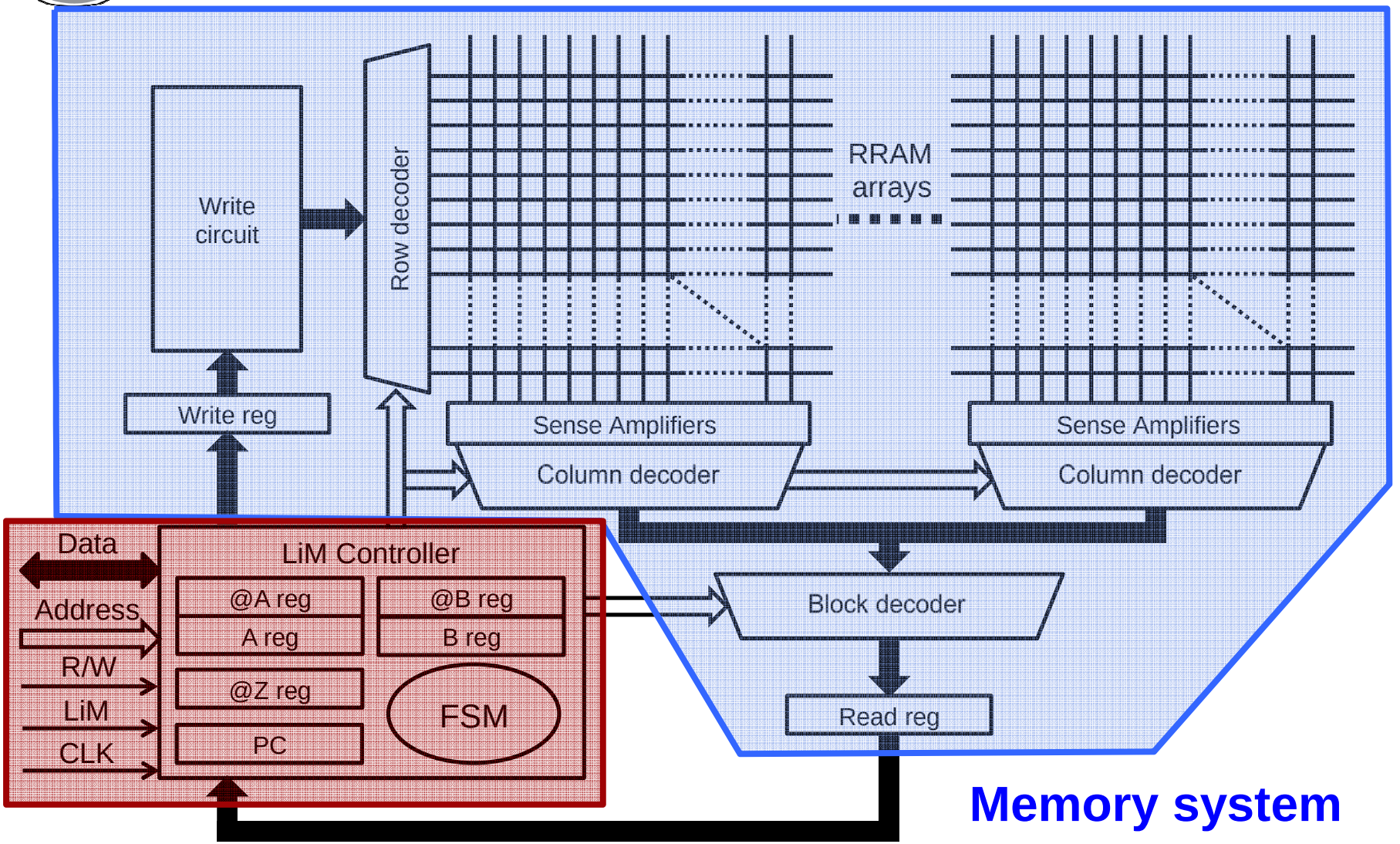
Exploits RM_3 instructions

1- Provide minimal controller overhead at a cost of a lower performance → *Architectural aspects*
2- Address the question of controlling the operations and scheduling → *CAD aspects*

P.-E. Gaillardon *et al.*, *DATE'16*

© LNIS-UofU 2017. All rights reserved.

The PLiM Architecture



PLiM Controller

Memory system (banks, peripherals, …)

P.-E. Gaillardon *et al.*, *DATE'16*

© LNIS-UofU 2017. All rights reserved.

The RM_3 Instruction in details

RM_3 encoding: @A, @B, @Z

Let's consider the following toy example:

Initial state		Final state
0x00: 1 1 0 0	Instruction memory	0x00: 1 1 0 0
0x01: 1 1 1 1		0x01: 1 1 1 1
0x02: 1 1 0 1		0x02: 1 1 0 1
0x03: 0 1 0 1	Data memory	0x03: 0 1 1 1

Step 1: RM_3(A, B, Z) is dumped from the memory by reading the operand addresses – @A = 1100, @B=1111, @Z=1101

The operands are using a bit level addressing:

@A = [Address of the word containing A (2bits here); Position of A within the word]

Step 2: The operands A and B are read.

As @A=1100 – A is read from LSB of bit 0x03 – A=1; B=0

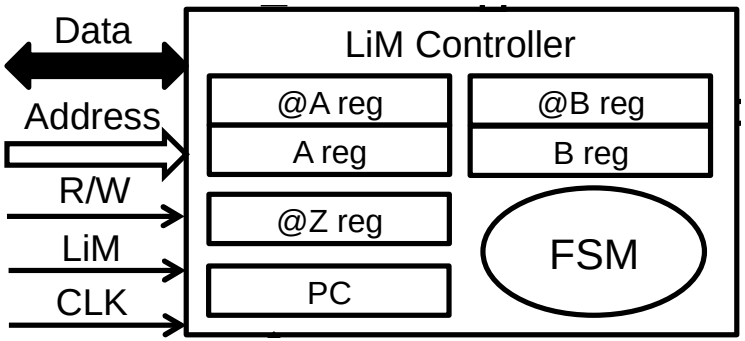
Step 3: The final destination Z is written by fixing A = 1 and B = 0.

Z = RM_3(1,0,0) = MAJ (1,1,0)=1.

© LNIS-UofU 2017. All rights reserved.

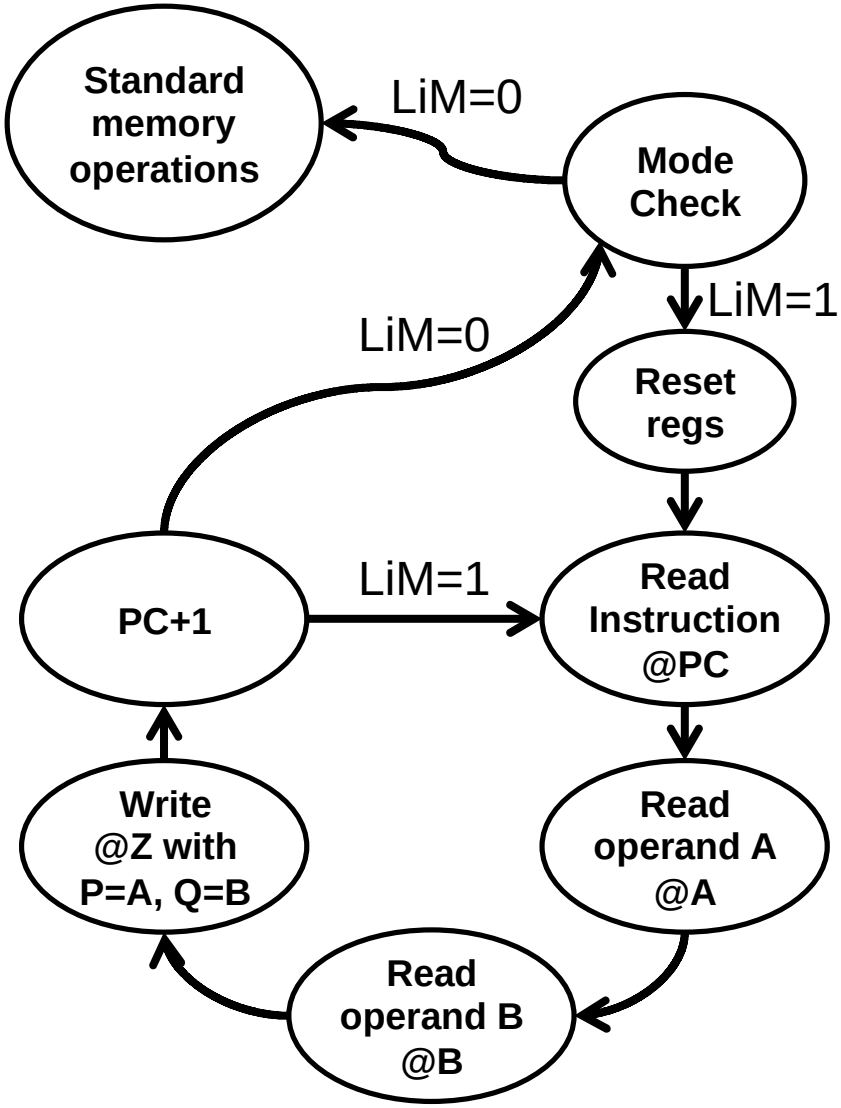
The PLiM Controller

Lightweight control



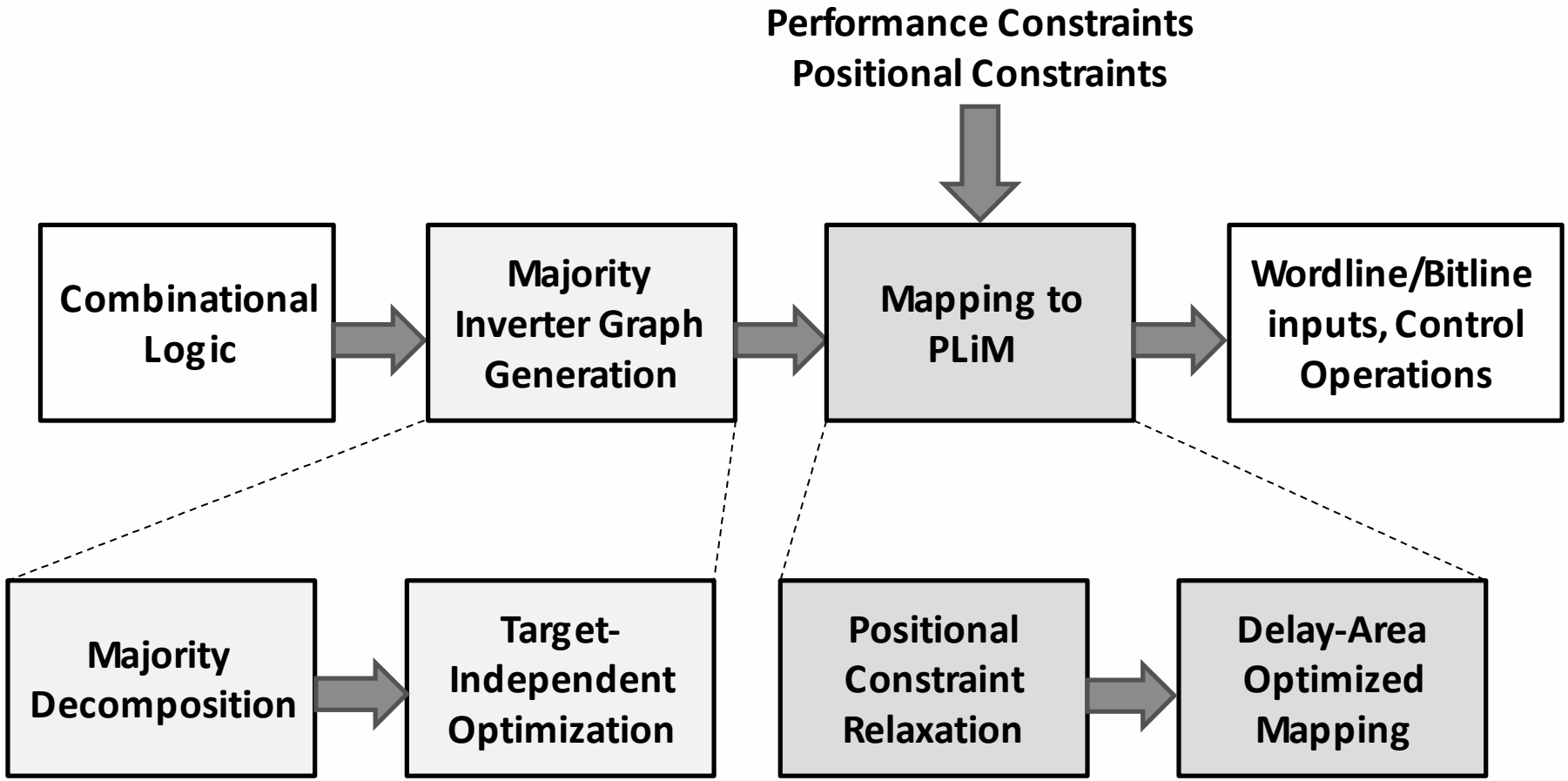
- PC for the operation
- 3× bit-wise address registers
- 2× 1-bit operand FFs
- Control FSM

Only process a succession of RM_3 operations



© LNIS-UofU 2017. All rights reserved.

PLiM Synthesis Flow for Larger Functions



© LNIS-UofU 2017. All rights reserved.

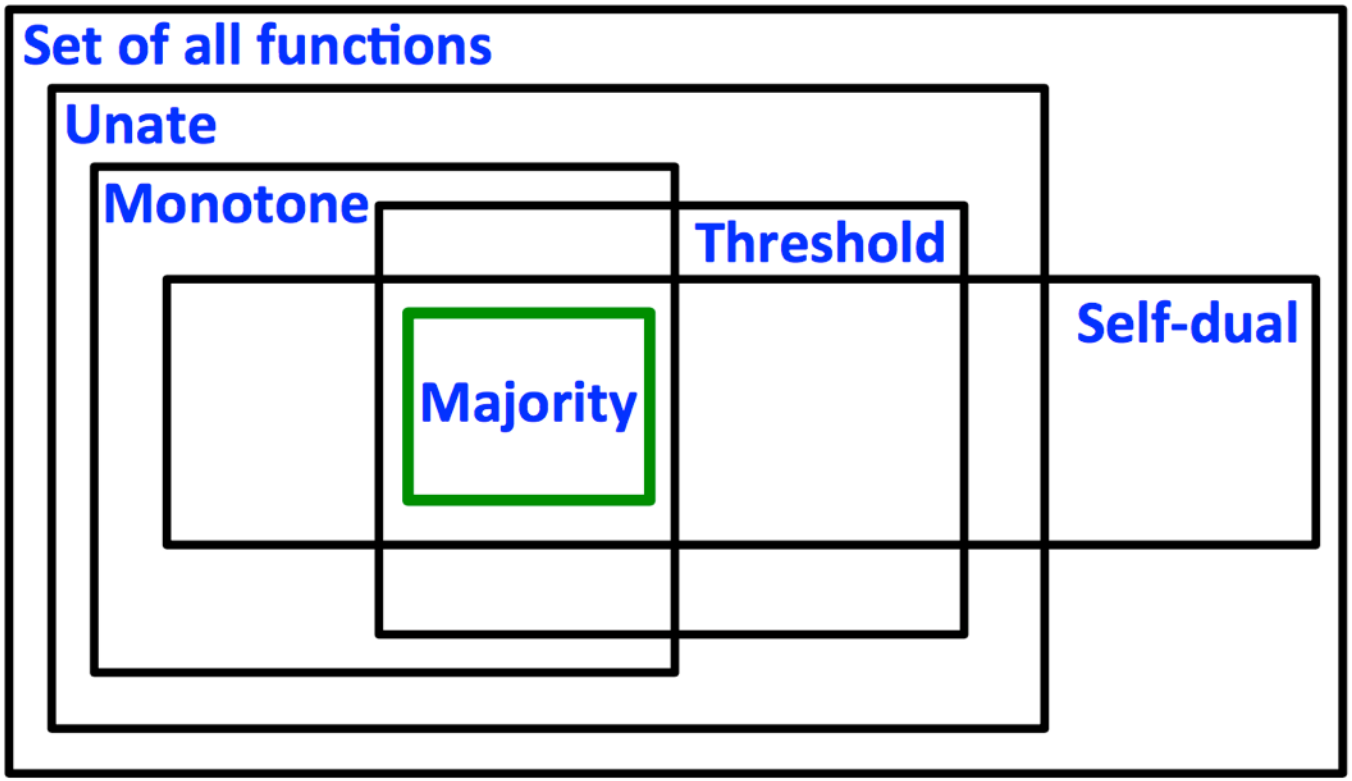
Resistive-MAJ Near-memory Computing

- Resistive-majority operations and PLiM computer
- **Majority-based logic manipulation**
- Logic manipulation for PLiM
- Conclusion

© LNIS-UofU 2017. All rights reserved.

How Powerful is Majority?

- Majority logic vs. AND/OR logic in representing arithmetic circuits.
- Consider small depth representations, target 4/5 logic levels.



AA. Sherstov, Separating AC 0 from depth-2 majority circuits, Proc. STOC, 2007

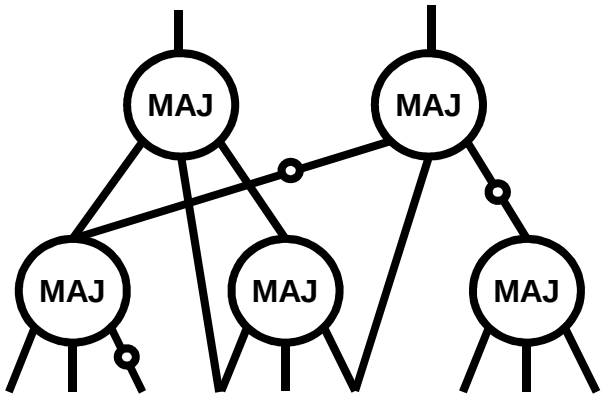
Matthias Krause and Pavel Pudlak, On the computational power of depth-2 circuits with threshold and modulo gates, Theor. Comput. Sci., 174 (1997), pp. 137–156.

Kai-Yeung Siu and Vwani P. Roychowdhury, On optimal depth threshold circuits for multiplication and related problems, SIAM J. Discrete Math., 7 (1994), pp. 284–292.

© LNIS-UofU 2017. All rights reserved.

How to manipulate MAJ logic?: Majority-Inverter Graph

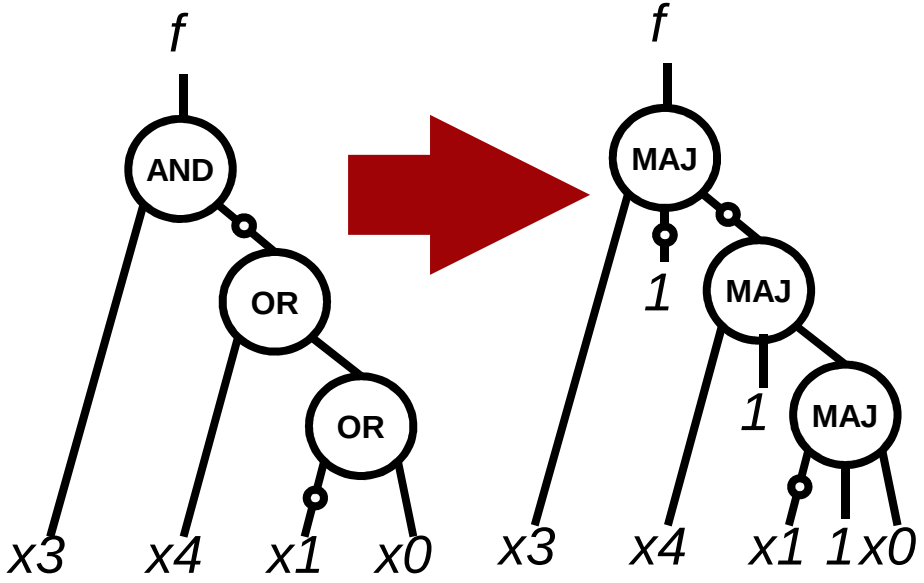
Definition: An MIG is a logic network consisting of 3-input majority nodes and regular/complemented edges.



From AOIG to MIG by direct transposition

Theorem: MIGs include AOIGs
MIGs include AIGs

MIGs are at least as compact as AOIGs
Exploiting the MAJ functionality unlock better representations



L. Amarù *et al.*, *DAC'14, TCAD'16*

© LNIS-UofU 2017. All rights reserved.

MIG Manipulation through an efficient Boolean Algebra

- MIG Axiomatic System Ω:
  - Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$
  - Majority: $if(x = y), M(x, y, z) = x = y \quad if(x = y'), M(x, y, z) = z$
  - Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$
  - Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$
  - Inverter Propagation: $M'(x, y, z) = M(x', y', z')$
- **Theorem:** (B,M,',0,1) subject to axiom in Ω is a Boolean algebra.
- **Theorem:** It is possible to transform any MIG α into any other logically equivalent MIG β, by a sequence of transformations in Ω.

In practice, it is always possible to reach a desired MIG, i.e., an **optimized** MIG, starting from an initial MIG.

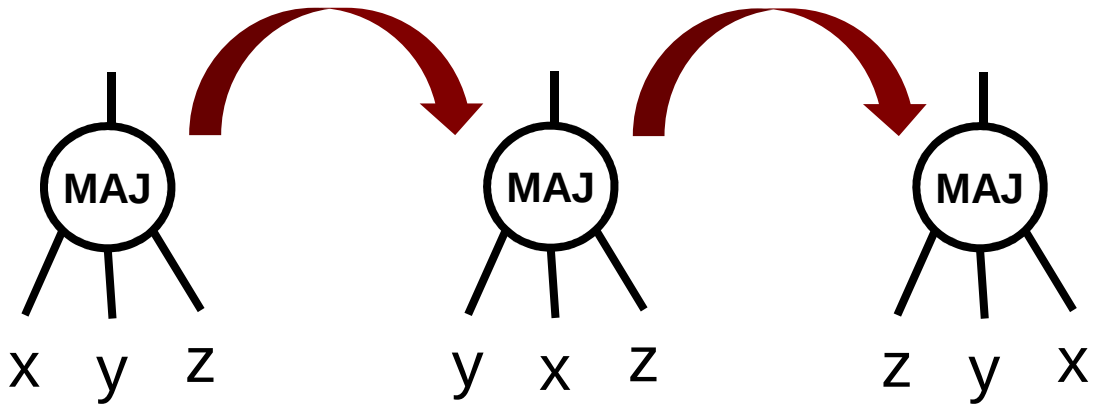
Area = MIG size – Delay = MIG depth – Power = MIG SWactivity

L. Amarù *et al.*, *DAC'14, TCAD'16*

© LNIS-UofU 2017. All rights reserved.

MIG Algebra: Commutativity Rule

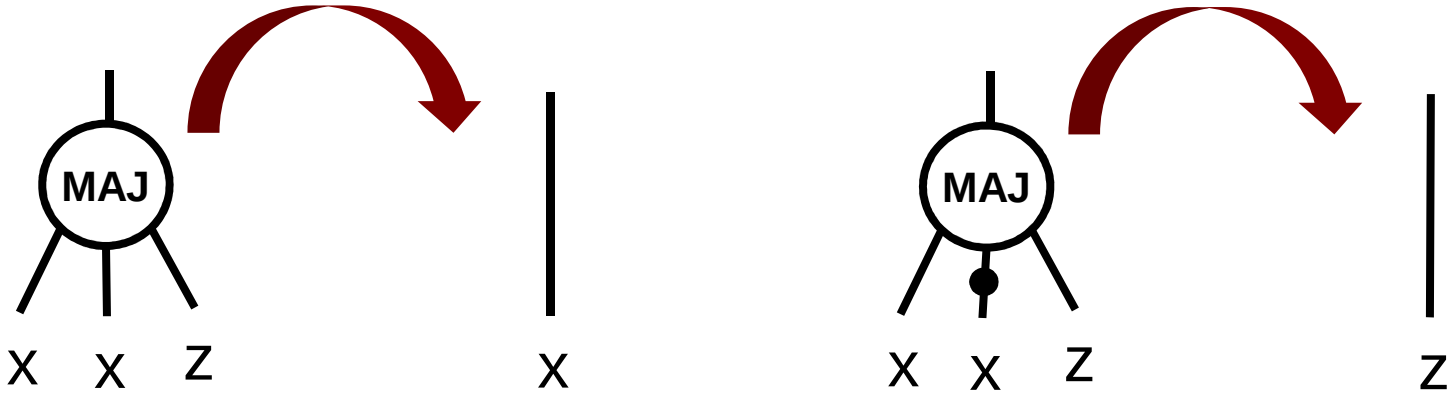
- **Commutativity: *M(x, y, z) = M(y, x, z) = M(z, y, x)***
- Majority: *if(x = y), M(x, y, z) = x = y if(x = y'), M(x, y, z) = z*
- Associativity: *M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))*
- Distributivity: *M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)*
- Inverter Propagation: *M'(x, y, z) = M(x', y', z')*



© LNIS-UofU 2017. All rights reserved.

MIG Algebra: Majority Rule

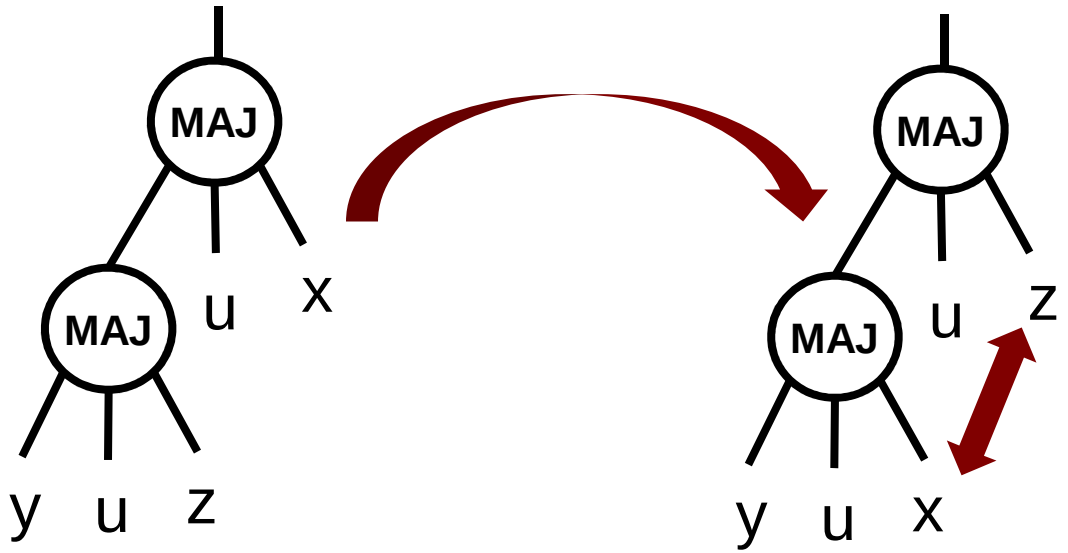
- Commutativity: *M(x, y, z) = M(y, x, z) = M(z, y, x)*
- **Majority:** ***if(x = y), M(x, y, z) = x = y if(x = y'), M(x, y, z) = z***
- Associativity: *M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))*
- Distributivity: *M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)*
- Inverter Propagation: *M'(x, y, z) = M(x', y', z')*



© LNIS-UofU 2017. All rights reserved.

MIG Algebra: Associativity Rule

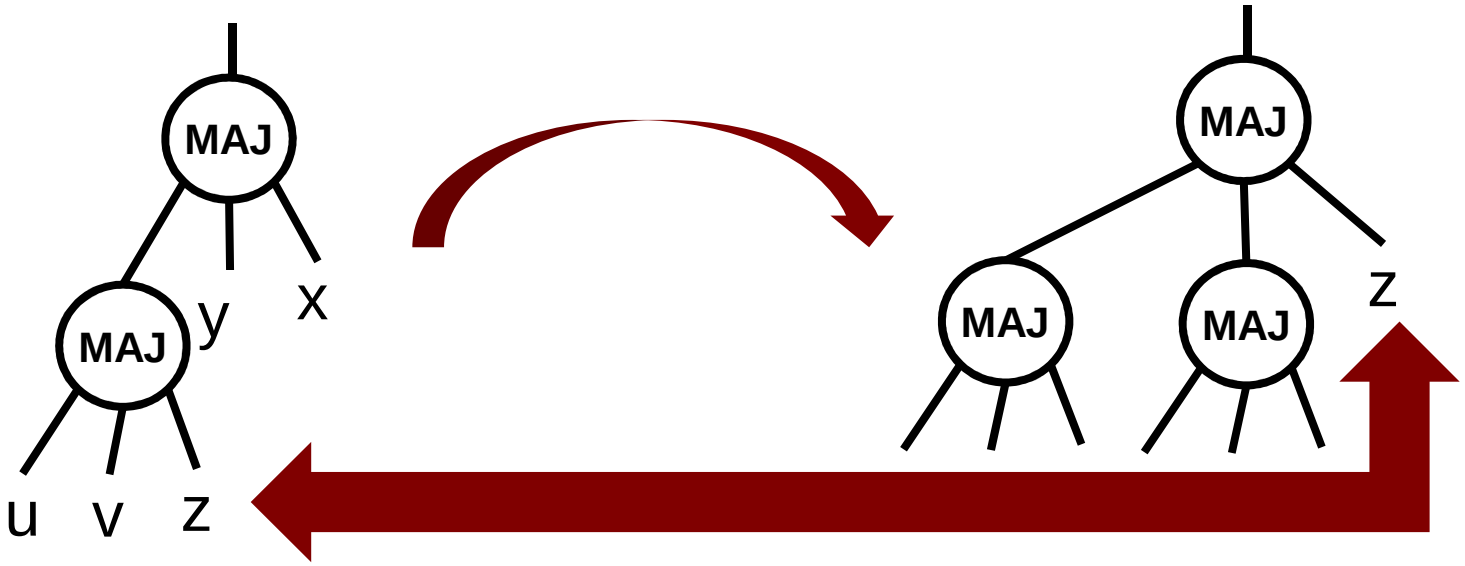
- Commutativity: *M(x, y, z) = M(y, x, z) = M(z, y, x)*
- Majority: *if(x = y), M(x, y, z) = x = y if(x = y'), M(x, y, z) = z*
- **Associativity: *M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))***
- Distributivity: *M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)*
- Inverter Propagation: *M'(x, y, z) = M(x', y', z')*



© LNIS-UofU 2017. All rights reserved.

MIG Algebra: Distributivity Rule

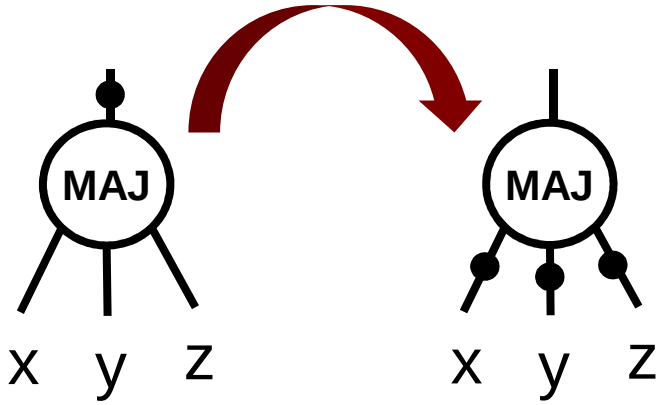
- Commutativity: *M(x, y, z) = M(y, x, z) = M(z, y, x)*
- Majority: *if(x = y), M(x, y, z) = x = y if(x = y'), M(x, y, z) = z*
- Associativity: *M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))*
- **Distributivity: *M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)***
- Inverter Propagation: *M'(x, y, z) = M(x', y', z')*



© LNIS-UofU 2017. All rights reserved.

MIG Algebra: Inverter Propagation Rule

- Commutativity: *M(x, y, z) = M(y, x, z) = M(z, y, x)*
- Majority: *if(x = y), M(x, y, z) = x = y if(x = y'), M(x, y, z) = z*
- Associativity: *M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))*
- Distributivity: *M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)*
- **Inverter Propagation: *M'(x, y, z) = M(x', y', z')***

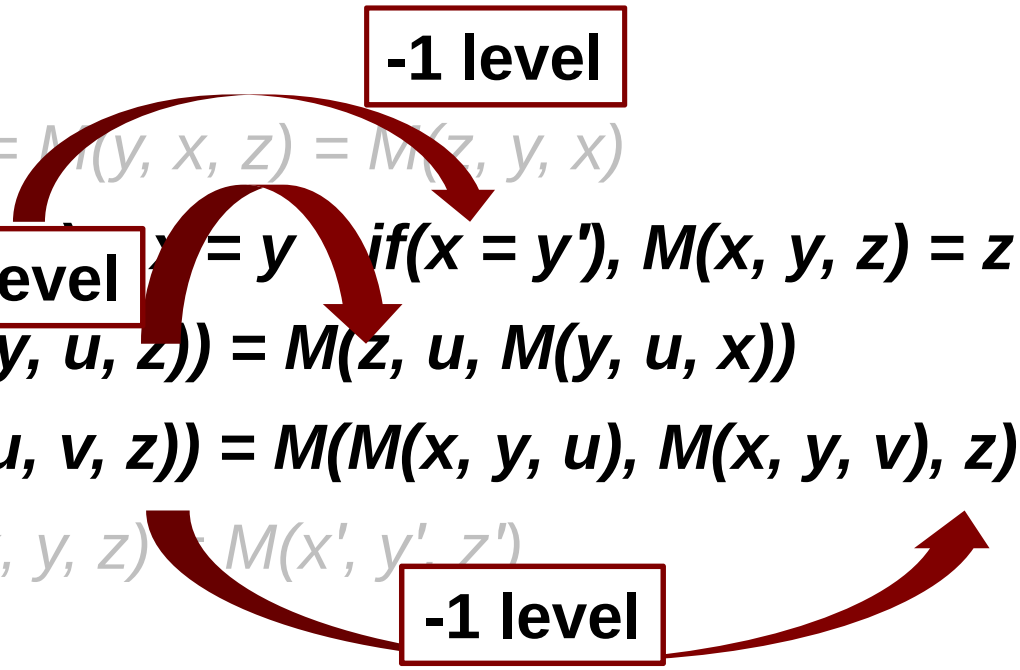


© LNIS-UofU 2017. All rights reserved.

MIG Depth Optimization

- How to reduce the depth of an MIG?
- What are the useful transformations for depth reduction?

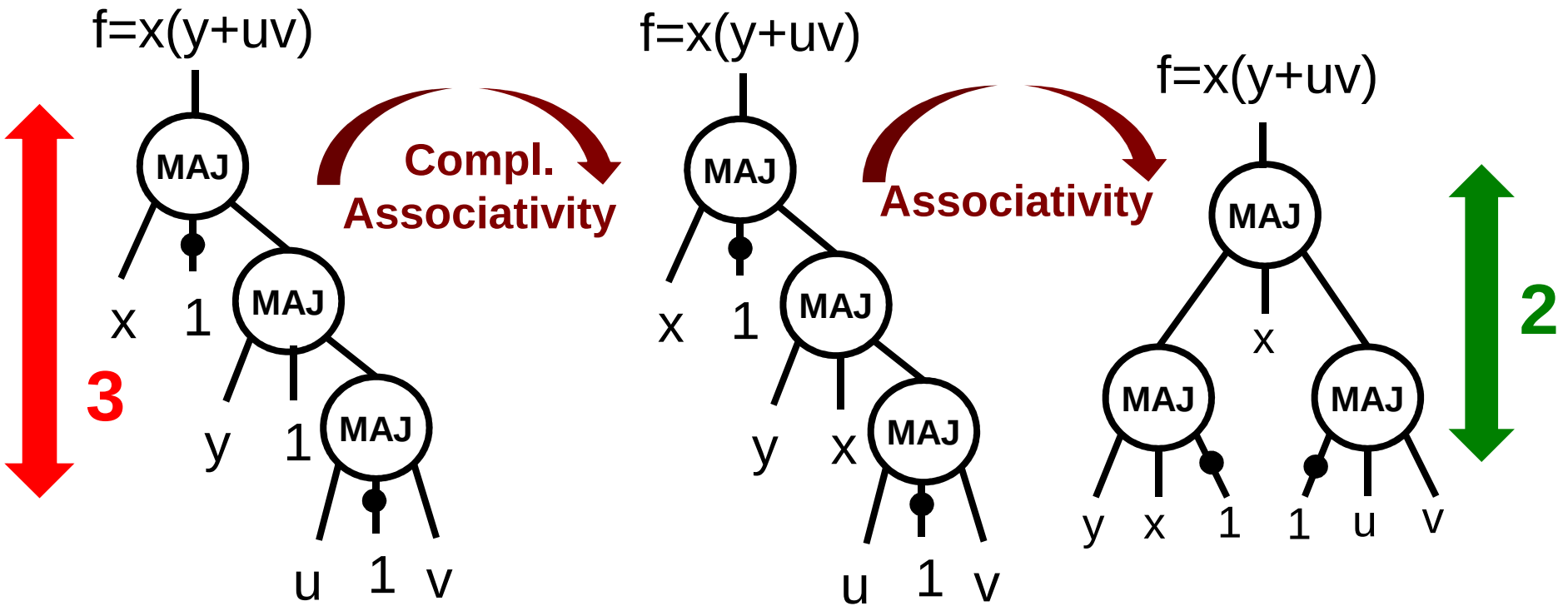
- Commutativity: *M(x, y, z) = M(y, x, z) = M(z, y, x)*
- **Majority:** ***if(x = y), M( ) = y if(x = y'), M(x, y, z) = z***
- **Associativity:** ***M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))***
- **Distributivity:** ***M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)***
- Inverter Propagation: *M'(x, y, z) = M(x', y', z')*



Optimization Rationale: Move critical variables closer to the outputs via associativity, distributivity and majority rules. Reshape the MIG with the other rules

© LNIS-UofU 2017. All rights reserved.

MIG Depth Optimization - Example

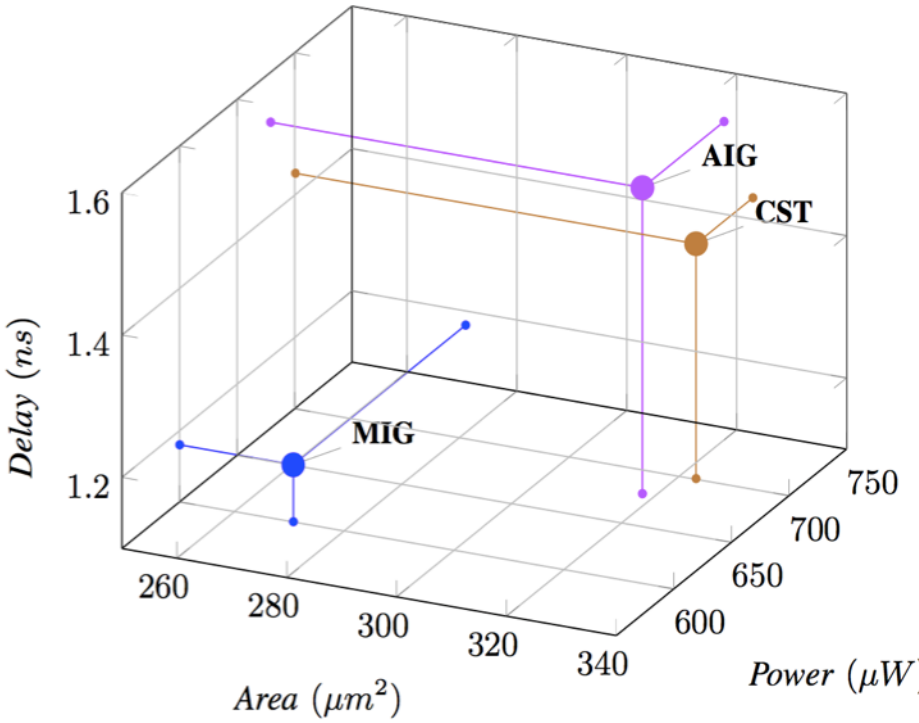


© LNIS-UofU 2017. All rights reserved.

Superiority of MIG w.r.t. Standard Techniques

Logic Synthesis results in 22-nm
CMOS – MCNC suite

Cells = {MIN, XOR, XNOR, NAND, NOR, INV}



ASIC

Post Tech. Mapping
-(22%, 14%, 11%)
delay, area, power
w.r.t. AOIG-based synthesis

Post P&R
-(14%, 4%, 3%)
delay, area, power
w.r.t. commercial flow

FPGA

Post Tech. Mapping
-16% depth LUT-6 reduction
w.r.t. AOIG-based synthesis

Post P&R
-(10%, 10%, 5%)
delay, area, power
w.r.t. commercial FPGA flow

L. Amarù *et al.*, *TCAD'16*

© LNIS-UofU 2017. All rights reserved.

Resistive-MAJ Near-memory Computing

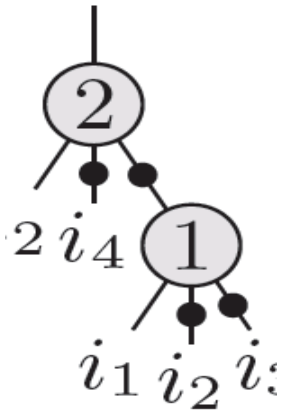
- Resistive-majority operations and PLiM computer
- Majority-based logic manipulation
- **Logic manipulation for PLiM**
- Conclusion

© | NIS-UofU 2017 - All rights reserved

MIG Techniques for PLiM

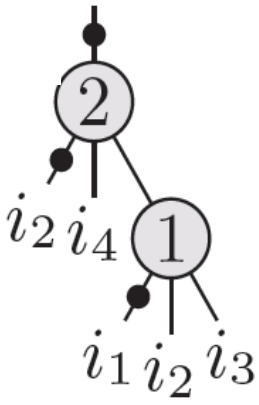
MIGs are native supports to optimize logic for PLiM (and more generally for RRAM in-memory computing)

Need to develop custom techniques to deal with inverters



Before MIG optimization

01: 0, 1, @X_1	$X_1 \leftarrow 0$
02: 1, i_3, @X_1	$X_1 \leftarrow \overline{i_3}$
03: i_1, i_2, @X_1	$X_1 \leftarrow N_1$
04: 0, 1, @X_2	$X_2 \leftarrow 0$
05: 1, @X_1, @X_2	$X_2 \leftarrow \overline{N_1}$
06: i_2, i_4, @X_2	$X_2 \leftarrow N_2$



After MIG optimization

01: 0, 1, @X_1	$X_1 \leftarrow 0$
02: i_3, 0, @X_1	$X_1 \leftarrow i_3$
03: i_2, i_1, @X_1	$X_1 \leftarrow N_1$
04: i_4, i_2, @X_1	$X_1 \leftarrow N_2$

S. Shirinzadeh *et al.*, *DATE'16*

© LNIS-UofU 2017. All rights reserved.

MIG Rewriting for INV optimization

Inverter Propagation: $M'(x, y, z) = M(x', y', z')$

MIGs allow natural flow of inverters within the graph while keeping homogeneity

1-level rules

4 INVs

M'(x', y', z') = M(x, y, z)

3 INVs

M'(x, y, z) = M(x', y', z')

2 INVs

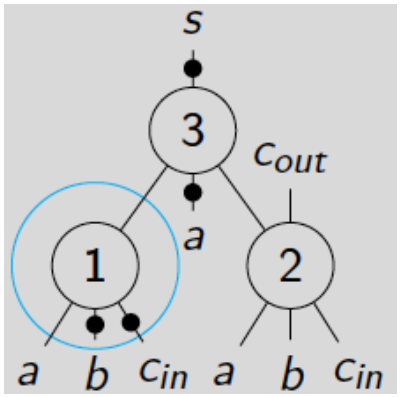
M'(x, y', z') = M(x', y, z)

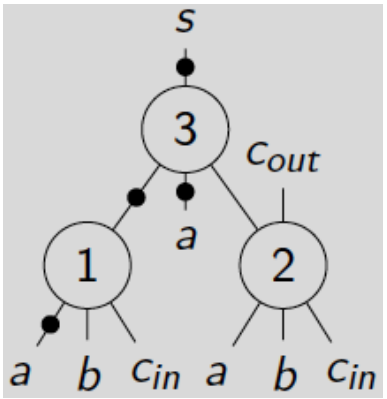
0 INVs

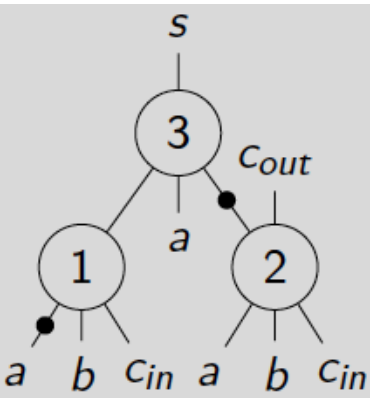
M(x', y', z) = M'(x, y, z')

…

2-level rules







Need for a general procedure

S. Shirinzadeh *et al.*, *DATE'16 – E. Testa* et al., *Nanoarch'16*

© LNIS-UofU 2017. All rights reserved.

MIG Rewriting for PLiM

Eliminate

Majority
M(x, x, z) = x M(x, x', z) = z
Distributivity
M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)

Use of common MIG optimization recipes

L. Amarù *et al.*, *TCAD'16*

Reshape

Associativity
M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))
Complementary Associativity
M(x, u, M(y, u', z)) = M(x, u, M(y, x, z)

MIG rewriting for PLiM

```
for (cycles = 0; cycles < effort; cycles++)
  Eliminate;
  Reshape;
  Eliminate;
  Inverter_Propagation(all_rules);
  Inverter_Propagation(3/4 INV rules);
end for
```

Objective:
Min (RM_3 instructions)
Max (nodes with a unique edge)

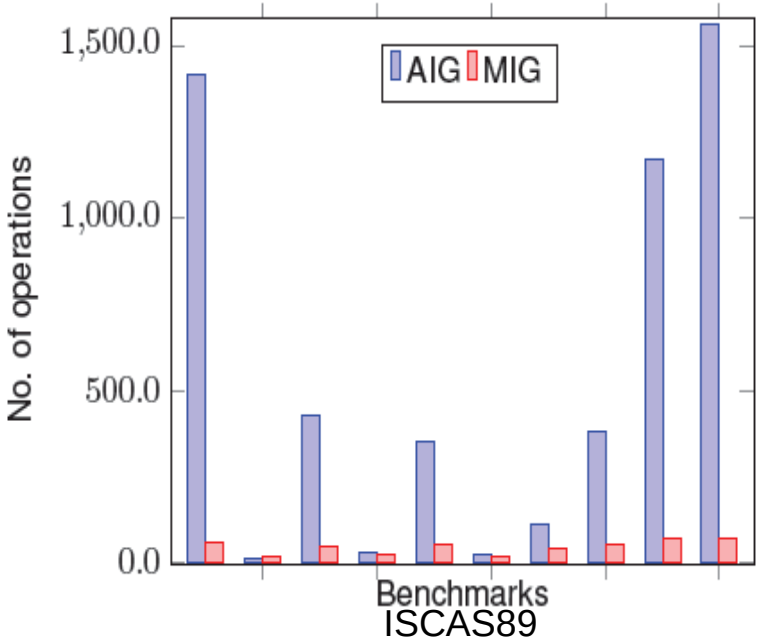
S. Shirinzadeh *et al.*, *DATE'16*

© LNIS-UofU 2017. All rights reserved.

Experimental Evaluation

Native use of Majority primitives leads to a **7.1× reduction** in the number of RM_3 instructions

Complemented edges optimization leads to **additional 14.4% reduction** in used memories and **12.4% in RM_3 instructions**



S. Shirinzadeh *et al.*, *DATE'16*

Benchmark	#N	MIG			PLiM Optimal				
		#CE	#R	#I	#CE	#R	Impr.	#I	Impr.
adder	2978	2905	263	6583	2620	261	0.8%	6586	-0.1%
log2	37582	38585	1851	85000	28259	1748	5.6%	73891	13.1%
max	7202	6543	567	13462	4348	562	0.9%	11385	15.4%
multiplier	41885	39589	613	100202	33606	551	10.1%	83078	17.1%
sin	7890	7625	671	16899	6305	571	14.9%	15807	6.5%
sqrt	52344	53327	501	111321	44213	507	-1.2%	101748	8.6%
square	19200	23390	941	52494	13650	287	69.5%	38776	26.1%
Average							14.4%		12.4%

EPFL suite

E. Testa *et al., Nanoarch'16*

© LNIS-UofU 2017. All rights reserved.

Resistive-MAJ Near-memory Computing

- Resistive-majority operations and PLiM computer
- Majority-based logic manipulation
- Logic manipulation for PLiM
- **Conclusion**

© | NIS-UofU 2017 - All rights reserved

Take-Away Messages about PLiM

- Computing with RRAMs goes be done with many different operators, e.g., Logic Implication, MAJ.
- MAJ operation done with a unique memory and simpler control signals
- The *PLiM* approach allows for:
  - Large memory arrays to natively perform small footprint processing
  - Memory manufacturers to add this functionality at a very low area overhead
- The *PLiM* architecture is a holistic approach that considers:
  - Operations natively made within RRAMs
  - The question of control
  - The question of mapping and synthesis
- Low-cost distributed memory systems capable of operating on their own content may be the basis for new computing paradigms

© LNIS-UofU 2017. All rights reserved.

Why EDA must follow Emerging Techs?

- Logic Synthesis (and EDA) is a **CMOS supporter**
  - LS techniques derive from CMOS abilities - NAND/NOR/MUX
  - Novel technologies and computing paradigms disrupt this model
- Logic Synthesis may evolve towards **more expressive primitives**, i.e., **MAJ-based Logic** and be **more technology-aware**

More expressive CMOS	Nanotechnologies (RRAMs, SWD,…)	Neuromorphic computing (threshold)
L. Amarù *et al.*, *TCAD'16*	***E. Testa et al., Nanoarch'16* - S. Shirinzadeh *et al.*, *DATE'16* - P.-E. Gaillardon et al., *DATE'16***	

Donald E. Knuth (The Art of Computer Programming – 4A): "**MAJ(x,y,z) is probably the most important ternary operation in the entire universe**, because it has amazing properties that are continually being discovered and rediscovered."

© LNIS-UofU 2017. All rights reserved.

Acknowledgments

Funding

NSF

FNSNF

Pr. Nanni De Micheli



Architecture Design

Dr. Somayyeh Rahimian
Dr. Hassan Ghasemzadeh
Mr. Xifan Tang
Mr. Gain Kim
Mr. Edouard Giacomin

EDA Tools

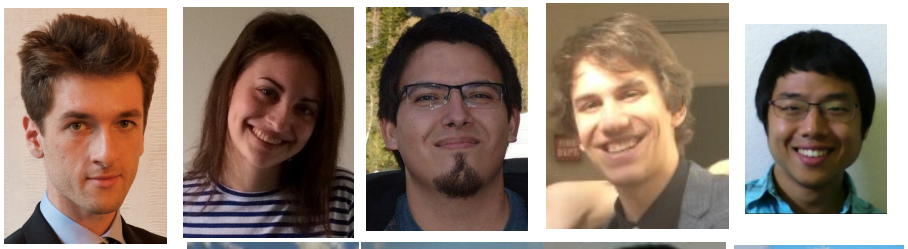
Dr. Luca Amarù
Mr. Winston Haaswijk
Ms. Eleonora Testa



Physical Design



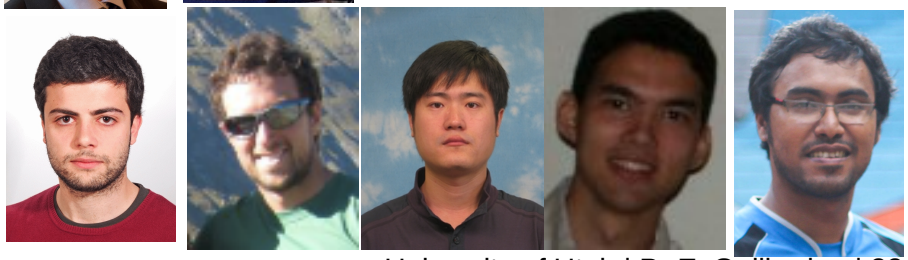
Modeling



Technology

Dr. Michele De Marchi
Dr. Jian Zhang
Mr. Maxime Thammasack

Mr. Giovanni Resta
Mr. Jorge Romero
Mr. Tanvir Ahmad



© LNIS-UofU 2017. All rights reserved.

Thank you for your attention

Questions?



Laboratory for NanoIntegrated Systems (LNIS)
Department of Electrical and Computer Engineering
MEB building – University of Utah – Salt Lake City – UT – USA