Technology-aware Logic Synthesis For ReRAM Based In-memory Computing

Debjyoti Bhattacharjee[1], **Luca Amaru**[2] and **Anupam Chattopadhyay**[1]

debjyoti001@ntu.edu.sg

1. School of Computer Science and Engineering, Nanyang Technological University, Singapore
2. Synopsys Inc, US

NANYANG TECHNOLOGICAL UNIVERSITY

Outline

Background

Technology Aware Logic Synthesis

Results

Logic-in-memory (LiM) Computing

Dedicated Arithmetic Circuits

Multi-valued Logic Circuits

Neuromorphic Circuits

General Purpose Programmable Architectures

PLiM

ReVAMP

MAGIC

1. The programmable logic-in-memory (PLiM) computer, Galliardon et al., DATE 2016
2. ReVAMP: ReRAM based VLIW Architecture for in-Memory computing, Bhattacharjee et al., DATE 2017
3. Logic design within memristive memories using memristor-aided loGIC (MAGIC), Talati et al., IEEE Trans 2016

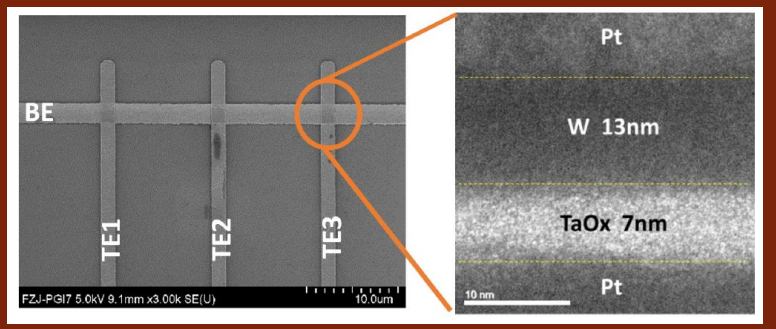
General Purpose Programmable Architectures

Primitive Boolean functions natively realized vary.

ReRAM devices arranged as a crossbar used for computation

Crossbar constraints differ across architectures

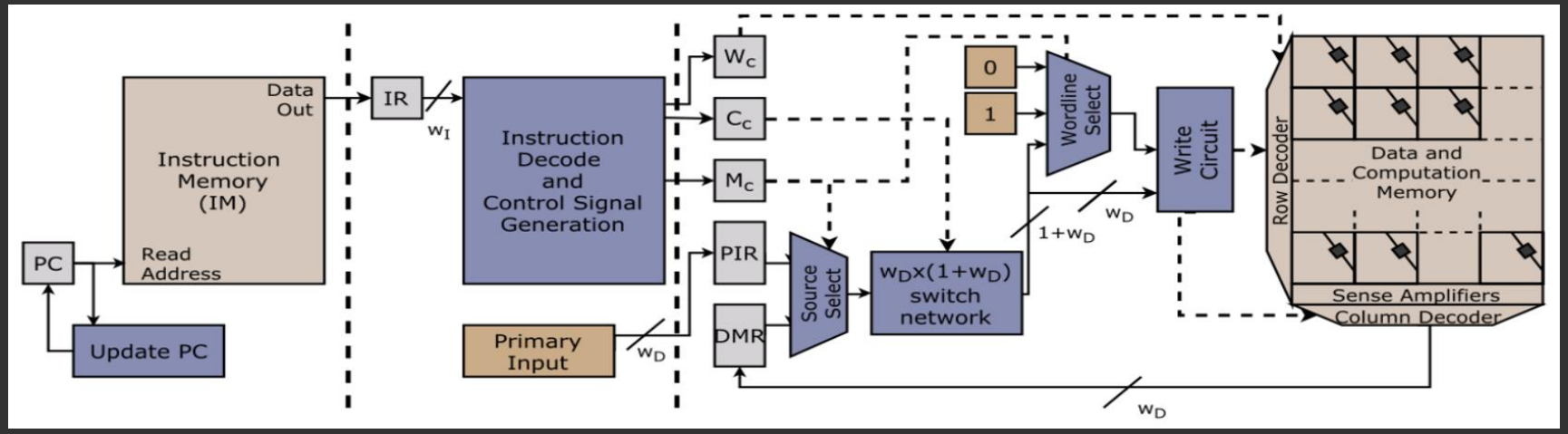
$1x3$ crossbar array



Multi-valued and Fuzzy Logic Realization using TaOx Memristive Devices, Bhattacharjee et al., Scientific Reports 2018.

ReVAMP

A VLIW-like Architecture using ReRAM crossbar memory



Instruction Fetch

- Conventional Instruction memory
- Two instructions : *Read* and *Apply*

Instruction Decode

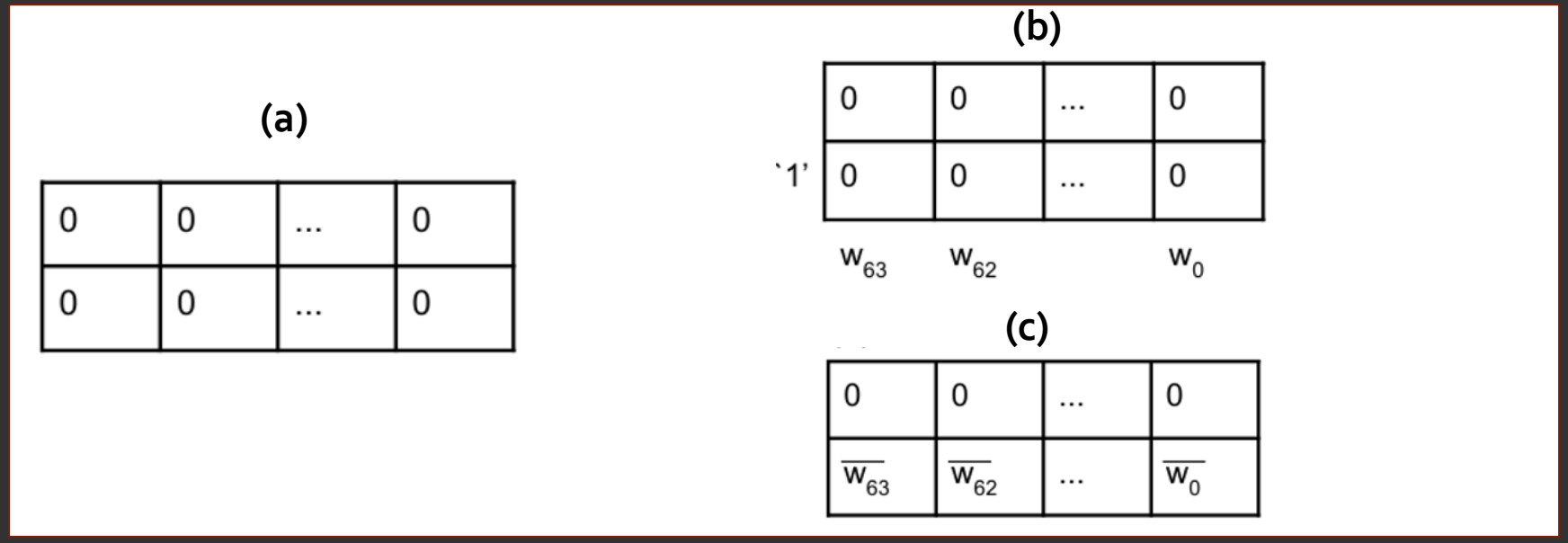
- Decodes instruction to populate control registers

Instruction Execute

1. ReVAMP: ReRAM based VLIW Architecture for in-Memory computing, Bhattacharjee et al., DATE 2017

ReVAMP

A VLIW-like Architecture using ReRAM crossbar memory

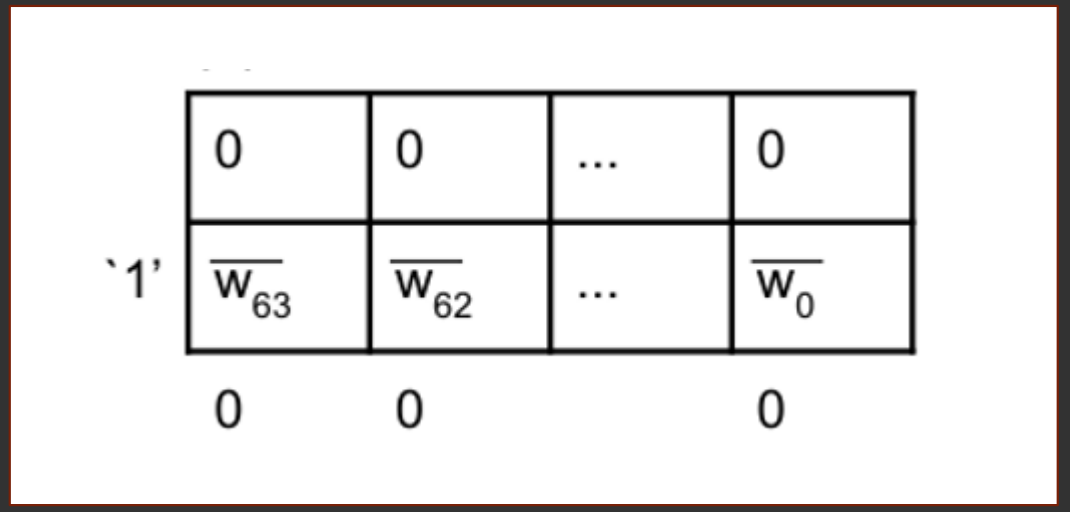


- All the devices are reset to 0 initially
- Values are applied via a wordline and the bitlines to perform computation
- This is specified using $Apply$ instruction

$$Apply\ w\ ws\ wb\ (v\ val_{63})(v\ val_{62})\ \ldots\ (v\ val_0)$$

ReVAMP

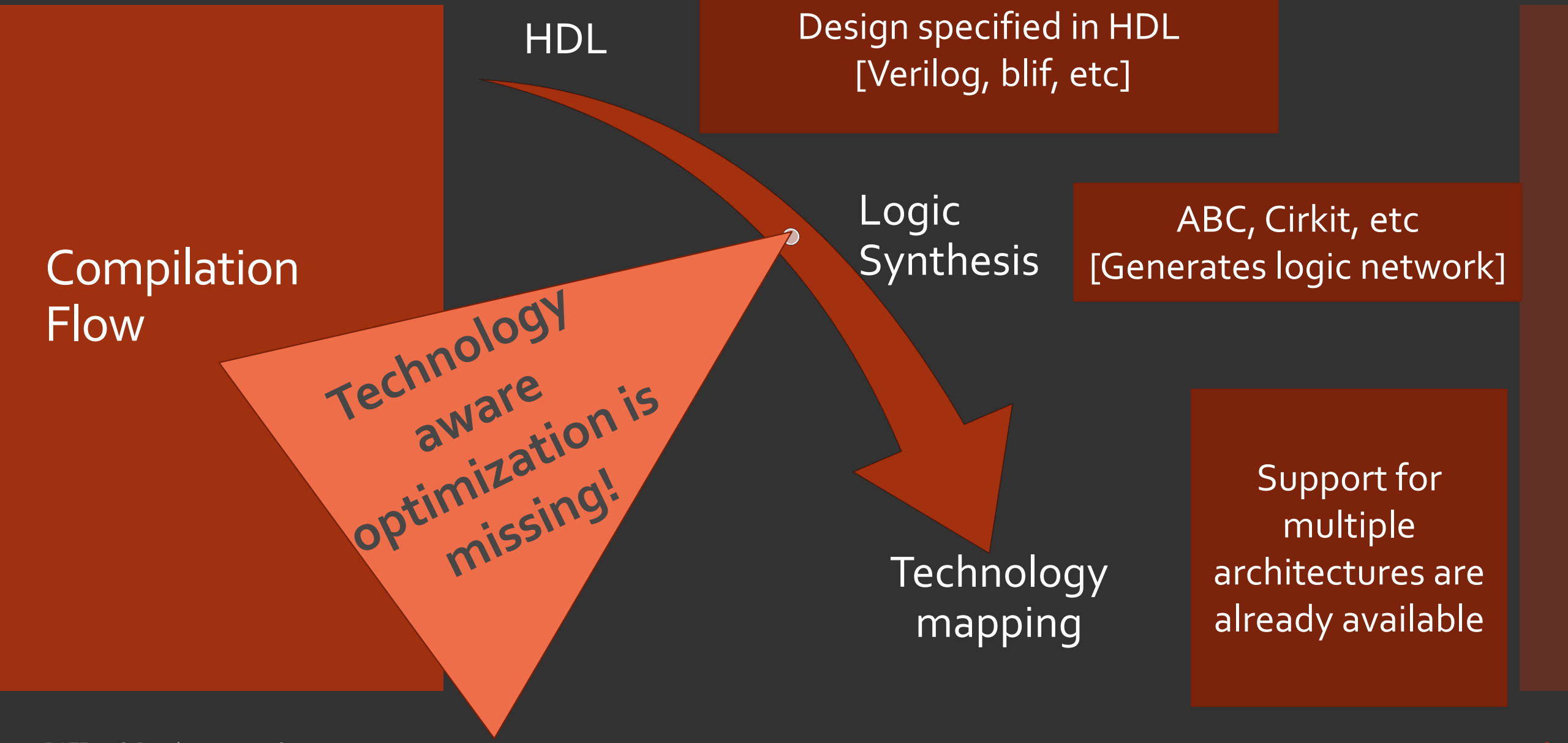
A VLIW-like Architecture using ReRAM crossbar memory



- The value stored in a word has to be read out for meaningful logical operations
- This is specified using $Read$ instruction ➔ Available in DMR

$Read\ w$

Compilation Flow



Outline

Background

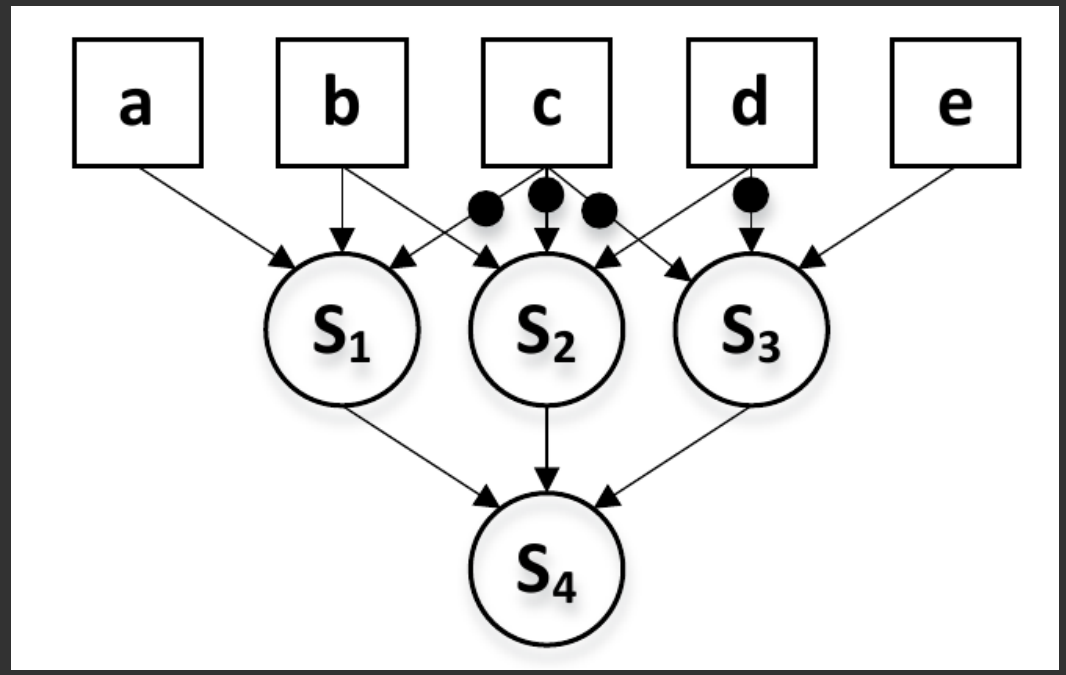
Technology Aware Logic Synthesis

Results

Technology aware Logic Synthesis

Technology aware Logic Synthesis

MIG → Algebraic Rewriting → Boolean IPM → Majority Refractoring



Technology aware Logic Synthesis

MIG → Algebraic Rewriting → Boolean IPM → Majority Refractoring

The axiomatic system for the MIG Boolean algebra, referred to as Ω

Commutativity — $\Omega.C$

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

Majority — $\Omega.M$

$$if(x = y) : M(x,y,z) = x = y$$
$$if(x = \overline{y}) : M(x,y,z) = z$$

Associativity — $\Omega.A$

$$M(x,u,M(y,u,z)) = M(z,u,M(y,u,x))$$

Distributivity — $\Omega.D$

$$M(x,y,M(u,v,z)) = M(M(x,y,u),M(x,y,v),z)$$

Inverter Propagation — $\Omega.I$

$$\overline{M}(x,y,z) = M(\overline{x},\overline{y},\overline{z})$$

"Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization," Amaru et al., DAC 2014

Technology aware Logic Synthesis

- MIGs ➔ hierarchical majority voting systems.
- Majority voting ➔ Capable of correcting different types of bit-errors.
  - Error masking property can be exploited for logic optimization.
  - Purposely introduce logic errors
- Choose inputs with the highest dictatorship for inserting errors
  - Dictatorship : Ratio of input patterns over the total (2^n) for which the output assumes the same value than the selected input.

$$f = (a + b)c$$
$$Dictatorship(a) = 5/8$$
$$Dictatorship(b) = 5/8$$
$$Dictatorship(c) = 7/8$$

Technology aware Logic Synthesis

MIG → Algebraic Rewriting → Boolean IPM → Majority Refractoring

- Select a subset of the primary inputs with highest dictatorship.
- For each selected input, we determine a condition that causes an error.
  - These errors have to be orthogonal.

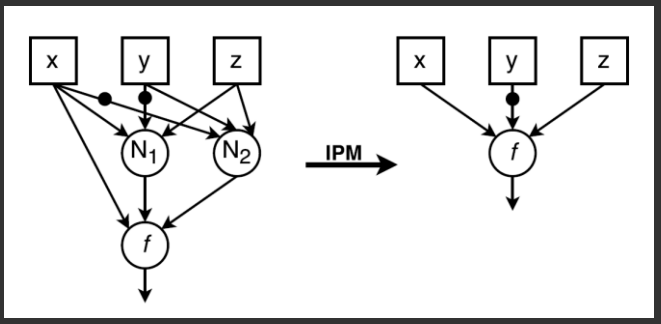
- x, y and z can be partitioned using the following partition
  ➔ $x\ !=\ y;\ x\ =\ y\ =\ z;\ x\ =\ y\ =\ z'$
- Corresponding errors are
  ➔ A : $x\ =\ y$; B : $z\ =\ y'$ when $x\ =\ y$; C : $z\ =\ y$

"Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization," Amaru et al., DAC 2014

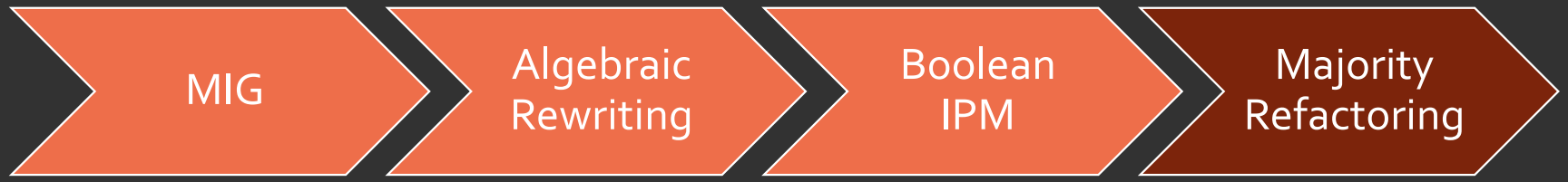
Input Partitioning Methods

MIG → Algebraic Rewriting → Boolean IPM → Majority Refactoring

- Consider $f = M(x, M(x, y', z), M(x', y, z))$
- For error A: $x = y$,
  - $fA = M(x, M(y, y', z), M(x', x, z)) = M(x, z, z) = z$
- For error B: $z = y'$,
  - $fB = M(x, M(x, y', y'), M(x', y, y')) = M(x, y', x') = y'$
- For error C: $z = y$,
  - $fC = M(x, M(x, z', z), M(x', z, z)) = M(x, x, z) = x$
- Thus, $f = M(fA, fB, fC) = M(z, y', x)$
  - Two levels collapse into a single level
  - Node count reduces by 2



Technology aware Logic Synthesis

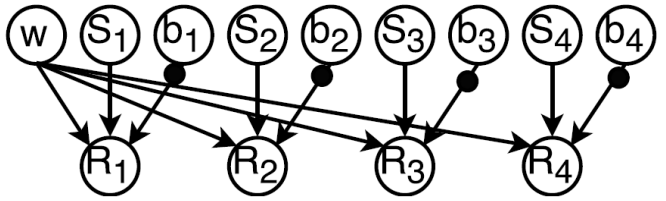


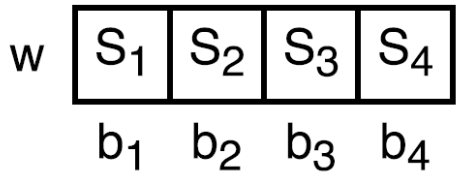
- Cone of logic ➔ canonical logic representation (BDD, Truth Table)
- Canonical logic representation ➔ a new local MIG (majority decomposition)
- Compute cost of the new local MIG
  - If $cost(local\ MIG) < cost\ (original\ logic\ cone)$: Local MIG is imported back to the network
- Repeat for every node of the MIG in topological order
  - Stop if no more improvement or a computation limit is reached

Technology aware Logic Synthesis

Optimization goals

- #nodes indicates the number of computations needed.
  - reduction of number nodes in MIG is primary objective
- For ReRAM devices arranged as a crossbar,
  - All the operations in a level of the logic networks might not be computed in parallel
  - Depth of MIG **does not** directly translate to delay
- Rationale for crossbar-aware optimization :
  - Enforce logic sharing in the MIG
  - Share non-inverted edges,
    - if multiple inverted-edges are shared, propagate the inverts above.





Outline

Background

Technology Aware Logic Synthesis

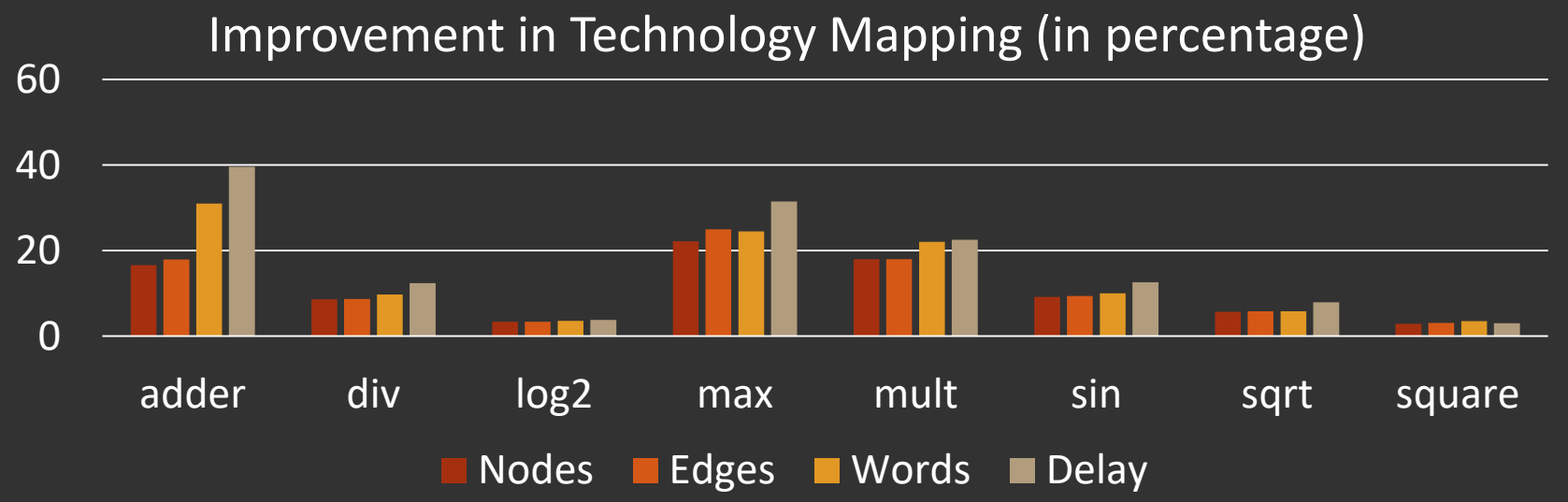
Results

Results

Performance on HARD EPFL benchmarks

- Depth-optimized hard EPFL benchmarks, available as MIG
- Word length of crossbar = 16
- Default technology mapping flow for ReVAMP used.

Benchmark	#Nodes	#Edges	#Words	Delay
adder	3369	9333	255	12597
adder(opti)	2811	7662	176	7603

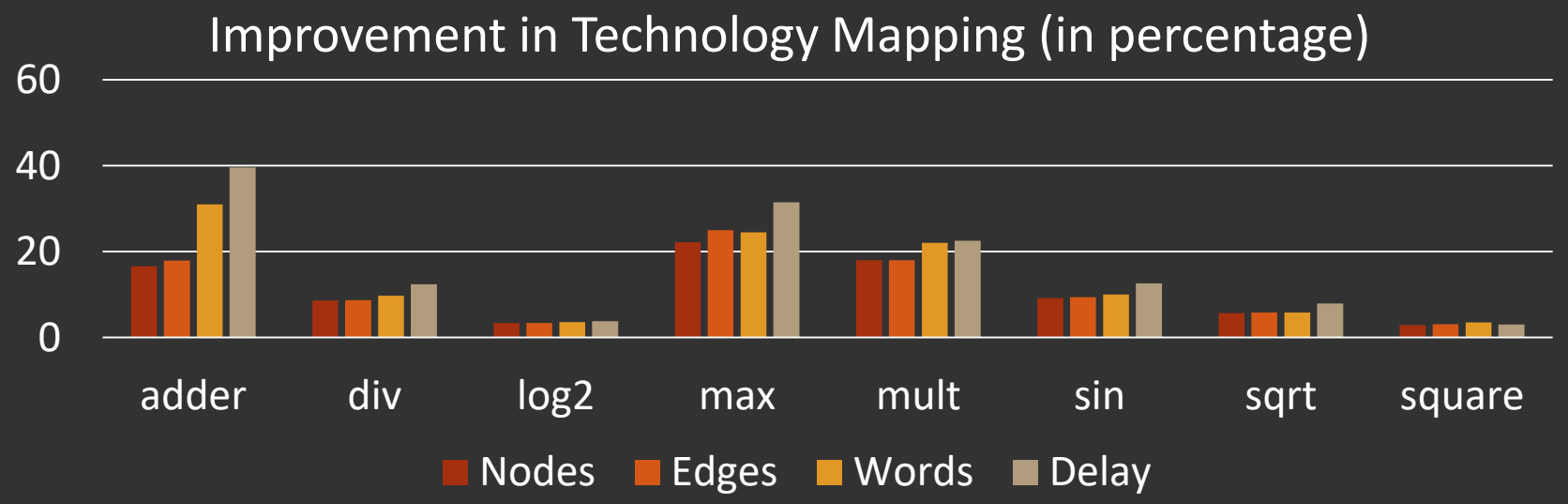


adder	div	log2	max	mult	sin	sqrt	square
3369	76005	37650	7846	42194	7946	52538	19395

Results

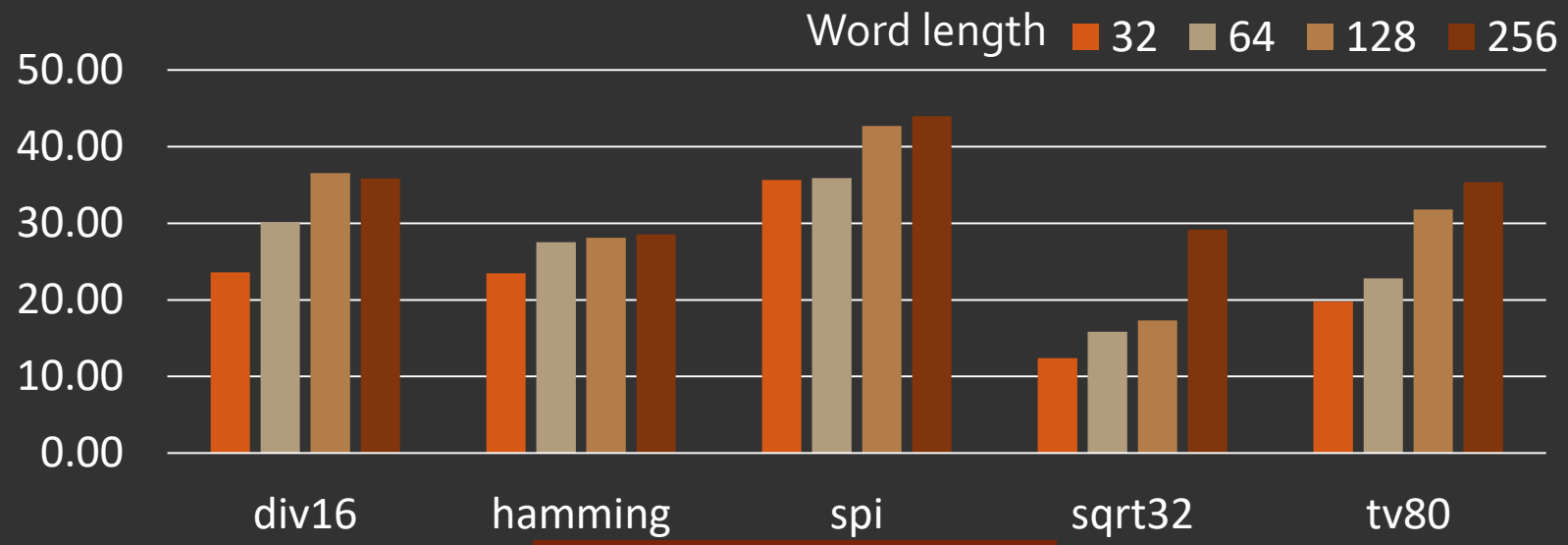
Performance on HARD EPFL benchmarks

- On average,
  - Reduction #nodes = 10.8% (maximum reduction $\mathbf{16.56\%}$)
  - Reduction in overall delay = 16.67% (maximum reduction of $\mathbf{39.64\%}$)
- For all the benchmarks, > 99% device utilization achieved by the technology mapping algorithm.

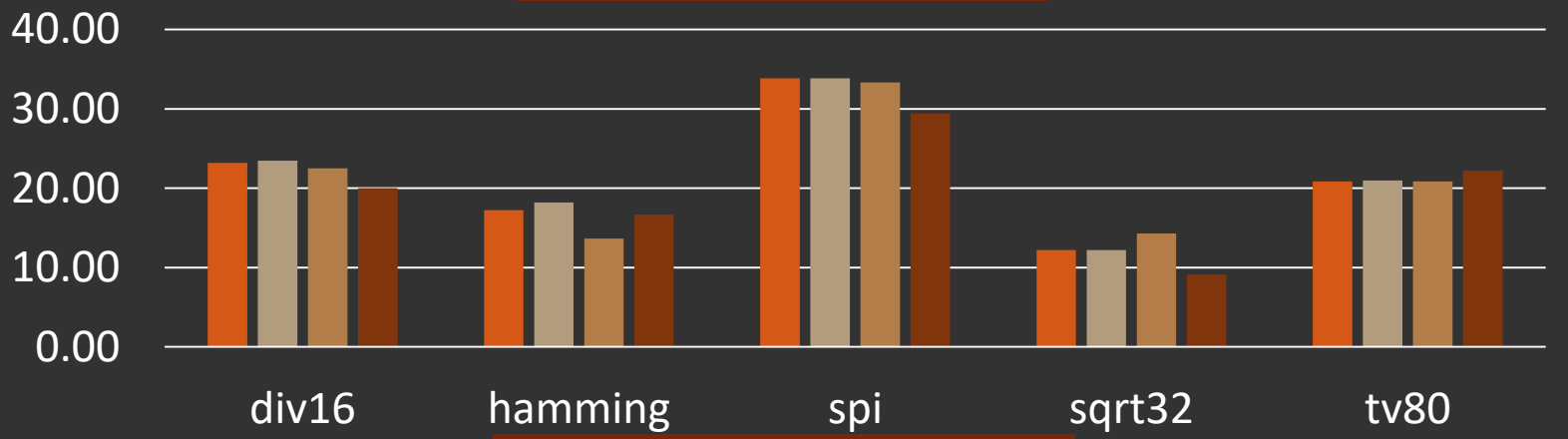


Results

Impact of word length on Synthesis optimizations



Improvement in Delay



Improvement in #Words

Conclusion

- Proposed a novel crossbar-aware MIG optimization automation flow
- Integrated into existing crossbar-aware technology mapping flow
- Demonstrated performance benefits over large benchmarks
  - Significant reduction in delay and number of words required for mapping
  - Enabled improved performance with increase in word length

Technology-aware Logic Synthesis For ReRAM Based In-memory Computing

Debjyoti Bhattacharjee[1], **Luca Amaru**[2] and **Anupam Chattopadhyay**[1]

debjyoti001@ntu.edu.sg

1. School of Computer Science and Engineering, Nanyang Technological University, Singapore
2. Synopsys Inc, US

NANYANG TECHNOLOGICAL UNIVERSITY