Modular Multi-ported SRAM-based Memories

Ameer M.S. Abdelhadi

Guy G.F. Lemieux

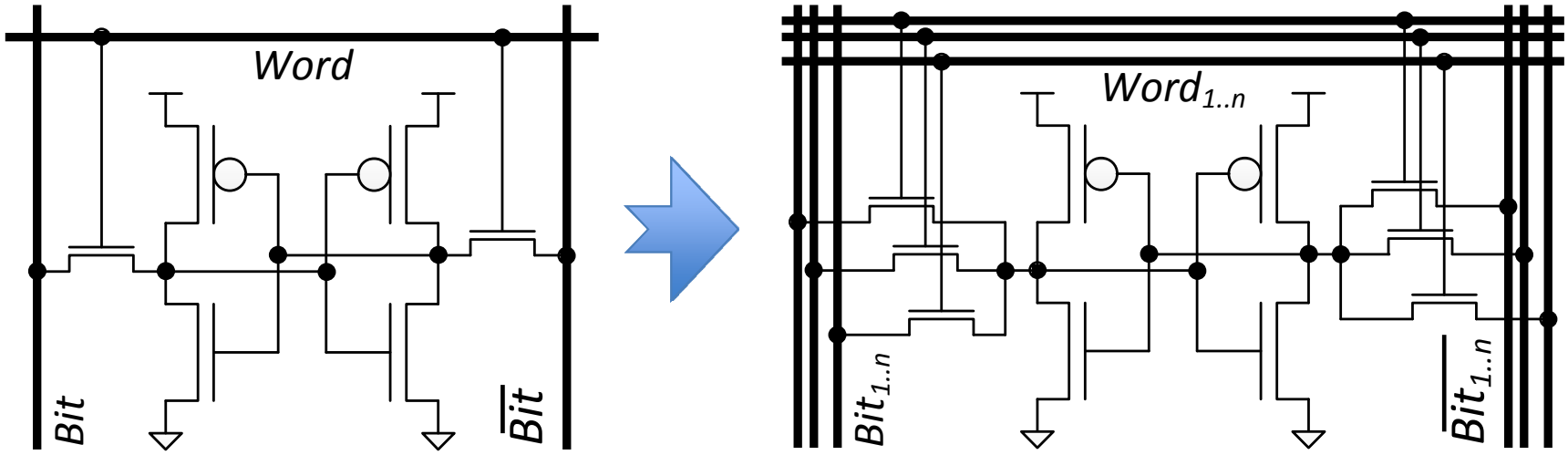
Multi-ported Memories: A Keystone for Parallel Computation!

- Enhance ILP for processors and accelerators, e.g.
  - VLIW Processors
  - CMPs
  - Vector Processors
  - CGRAs
- DSPs

✗ Major FPGA vendors provide dual-ported RAM only!

✗ ASIC RAM compilers provide limited ports!

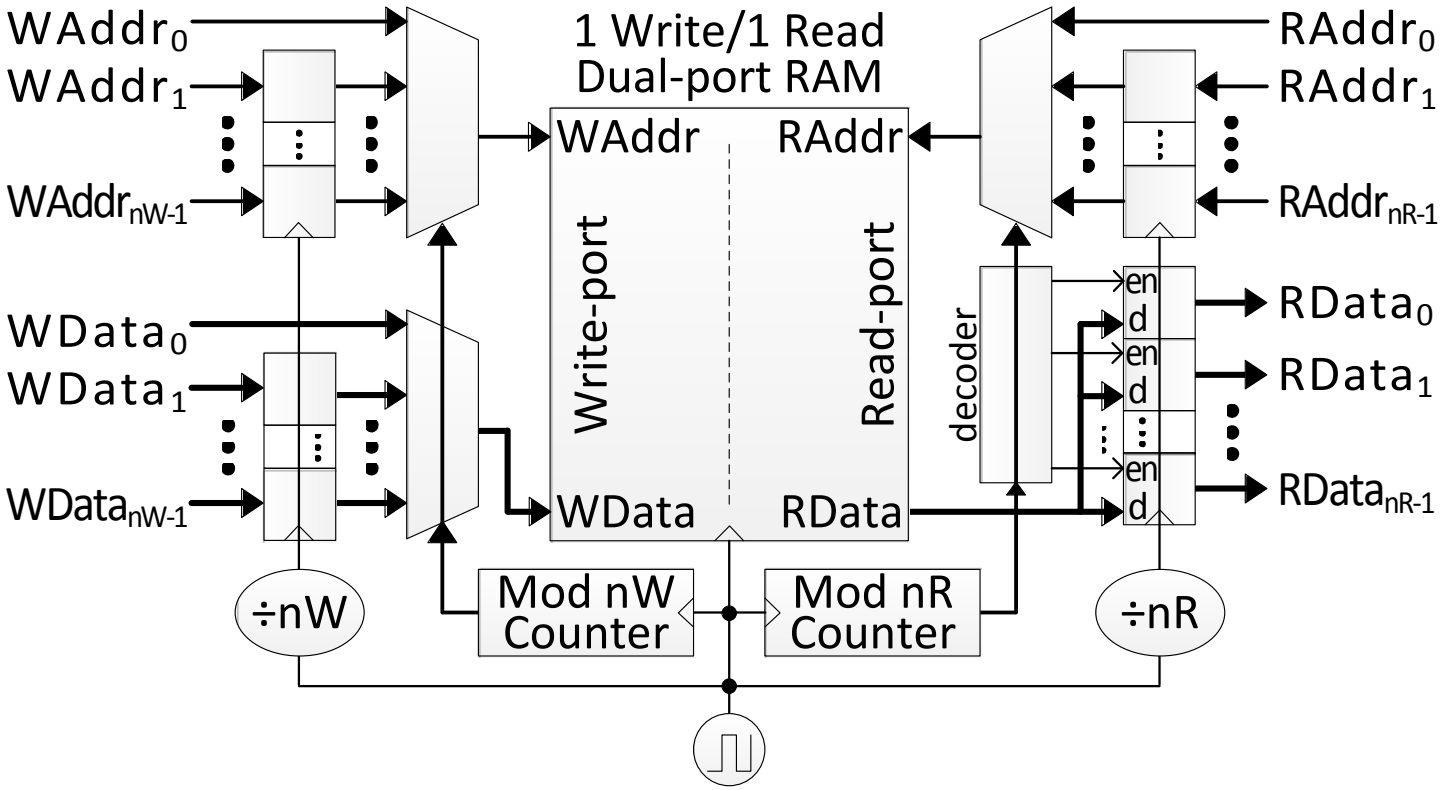
Multi-ported SRAM Cell



✗ ASICs / custom design only!

✗ Increasing ports incurs higher delays and area consumption

RAM Multi-pumping



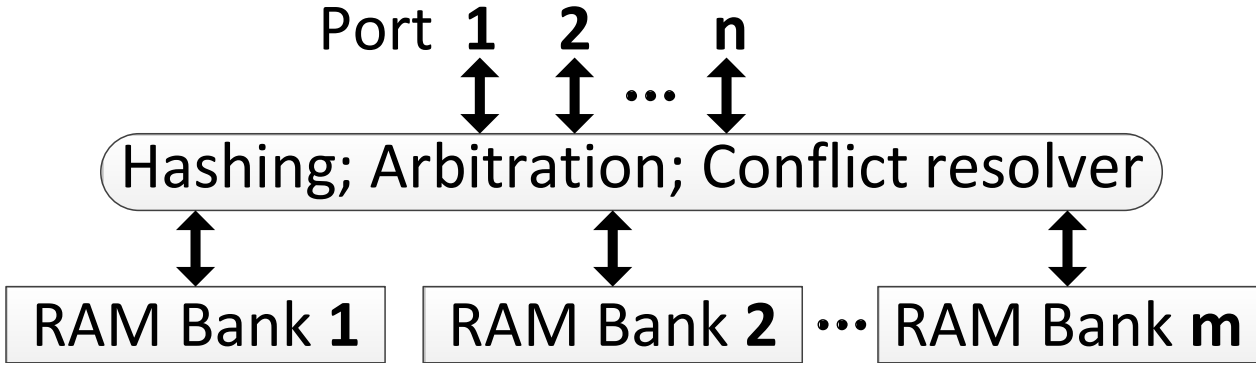
✓ Resources sharing

✓ Low area

✗ Performance degradation

✗ Data dependencies

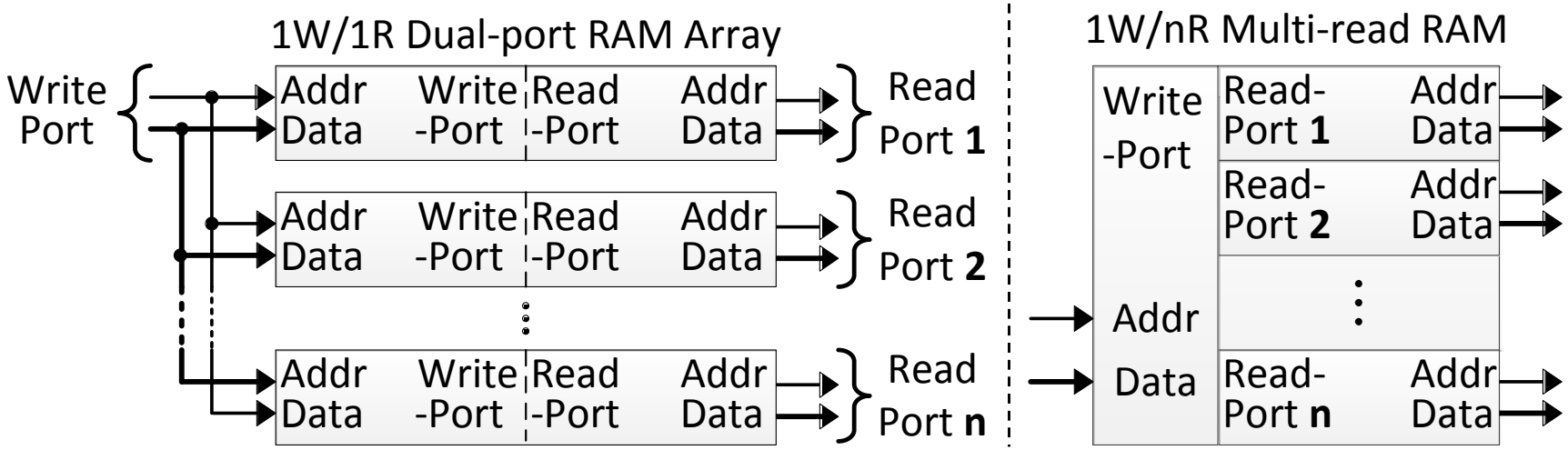
Multi-banking



- Divide memory into smaller banks
- Distribute data using fixed hashing scheme
- Access to same bank is resolved by multiple request
- The Pentium (P5) has 8-way two port interleaved cache*
- Area efficient
- Long arbitration delays
- Variable access latency

*[Alpert & Avnon, IEEE Micro, June 1993]

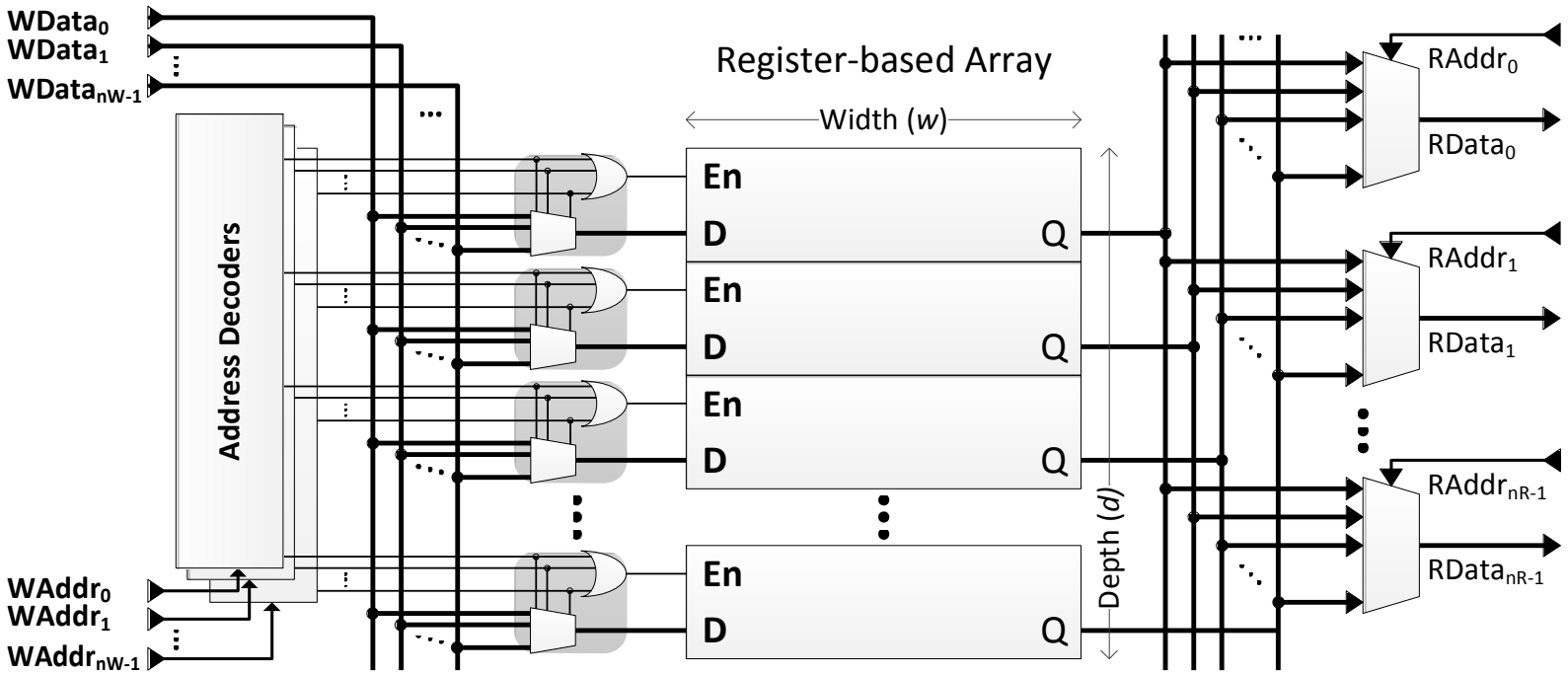
Multi-read by Bank Replication



- Example: Alpha 21264*
  - Each integer cluster has a replicated 80-entry register file
  - The 72-entry floating-point cluster register file is duplicated
  - number of read ports is doubled
  - Support two concurrent units each

*[Ditlow et al., IEEE ISSCC , Feb. 2011]

Register-based Multi-ported RAM

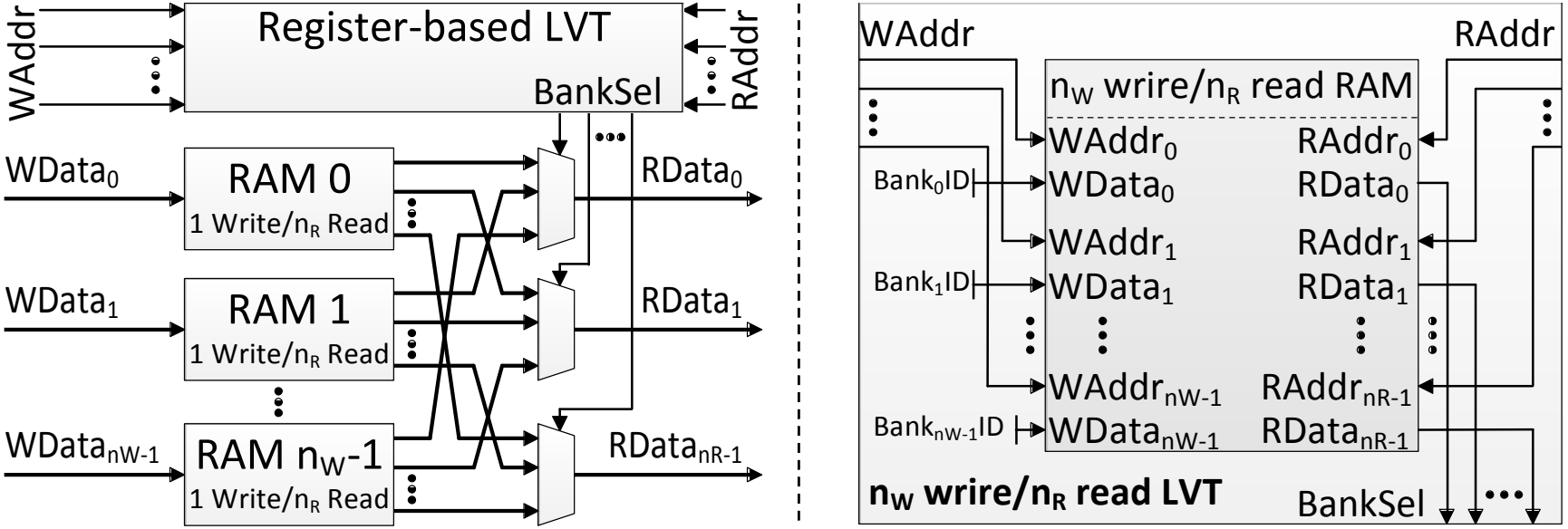


✓ High performance for small caches (<1k lines)

✗ High resources consumption for deep memories (scaling)

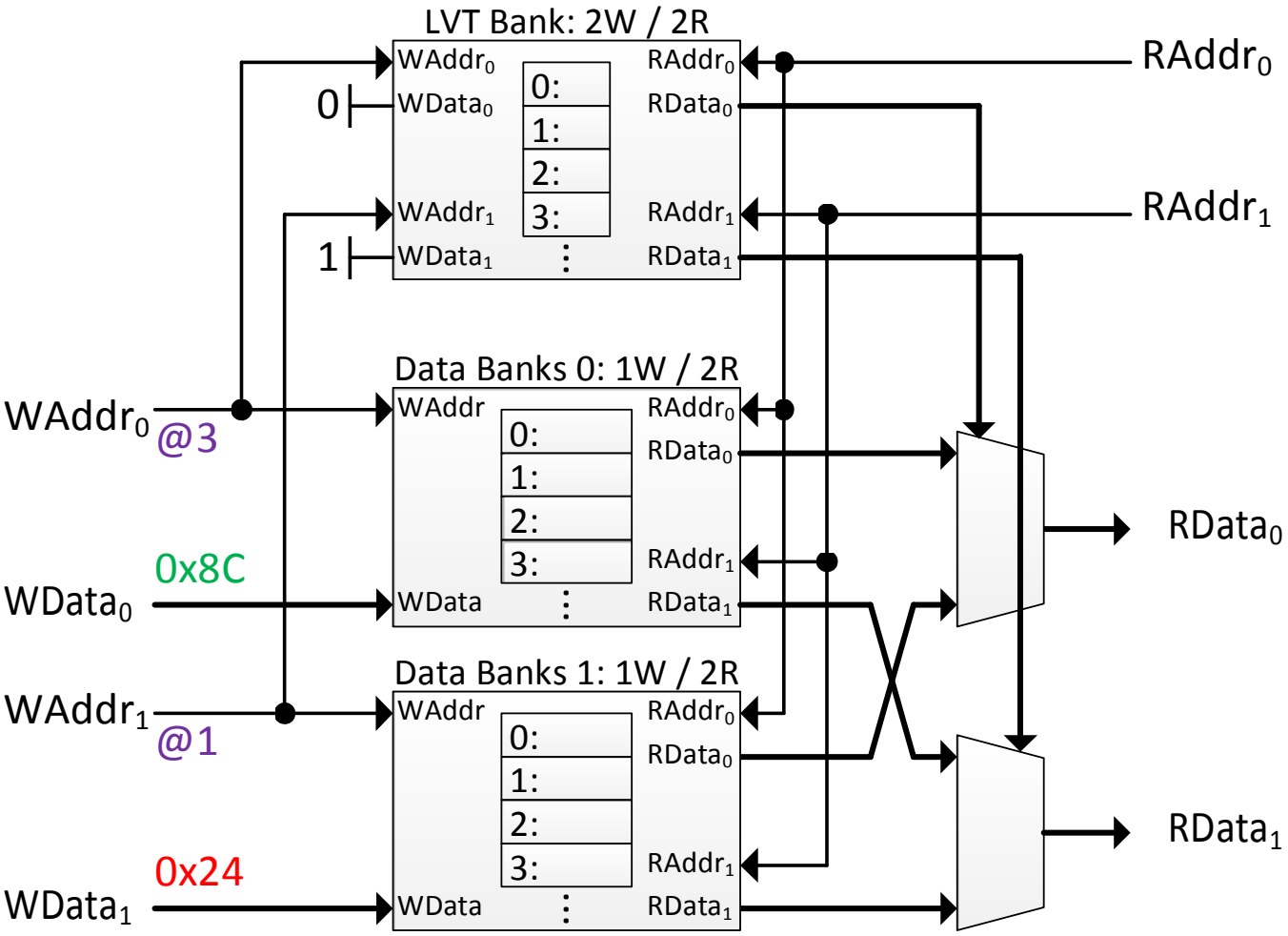
Infeasible on Altera's high-end Stratix V with our smallest test-case!

LVT-based Approach

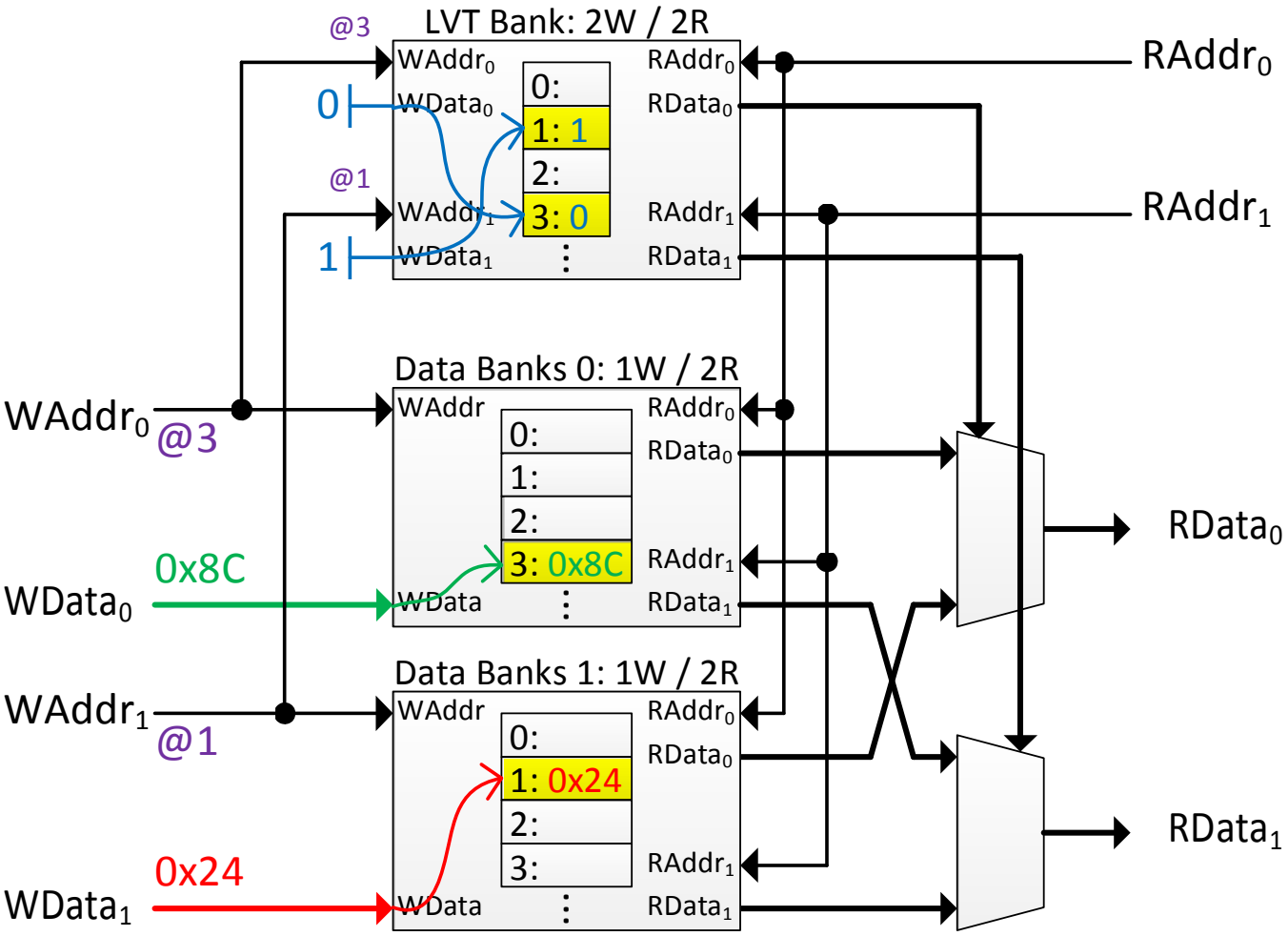


- Stores the ID of latest written bank
- LVT is a multi-ported RAM for banks IDs
  - Implemented with registers
  - Still has scaling issues: infeasible for deep memories!

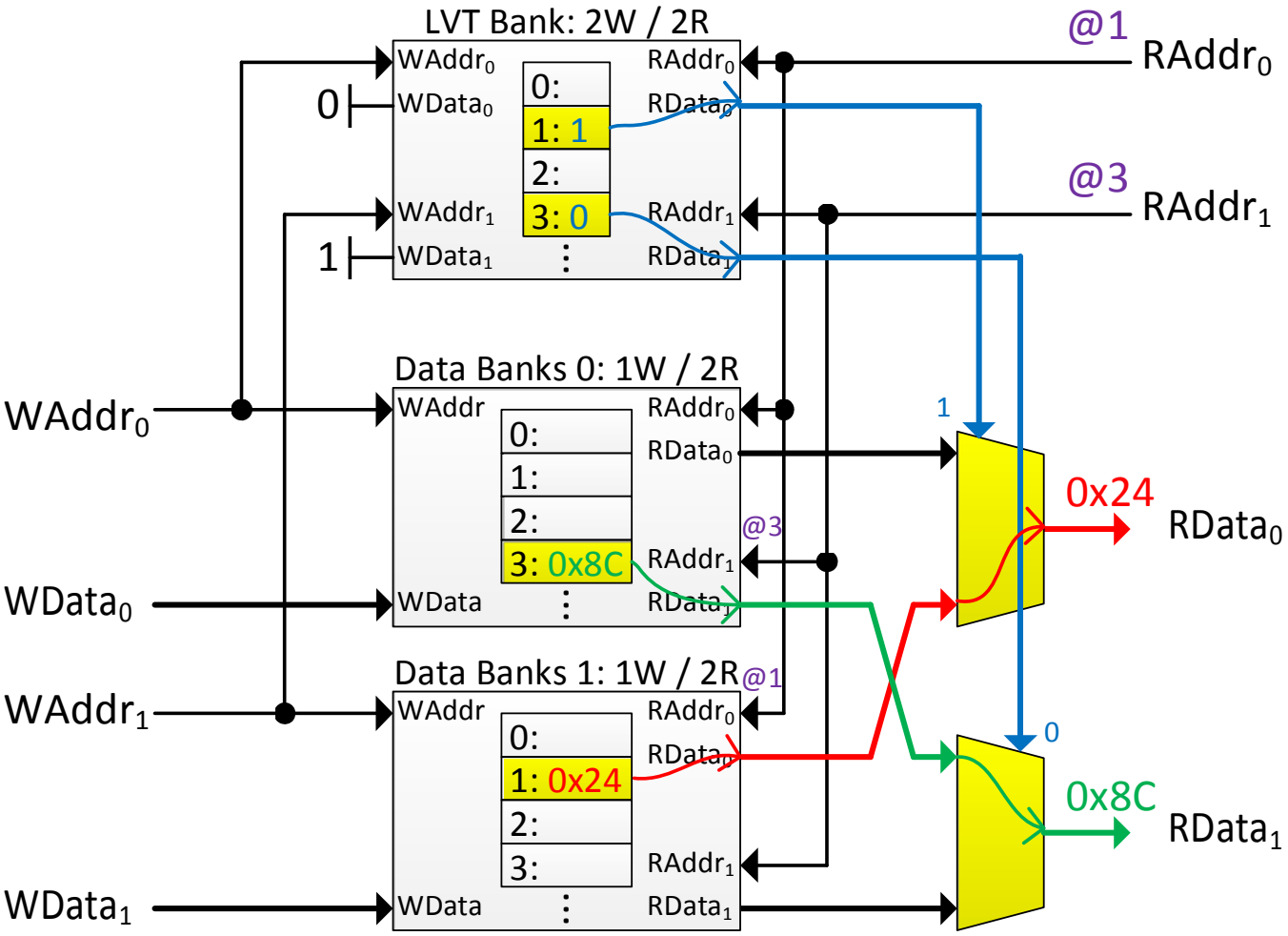
LVT-based Multi-ported RAM Example (1)



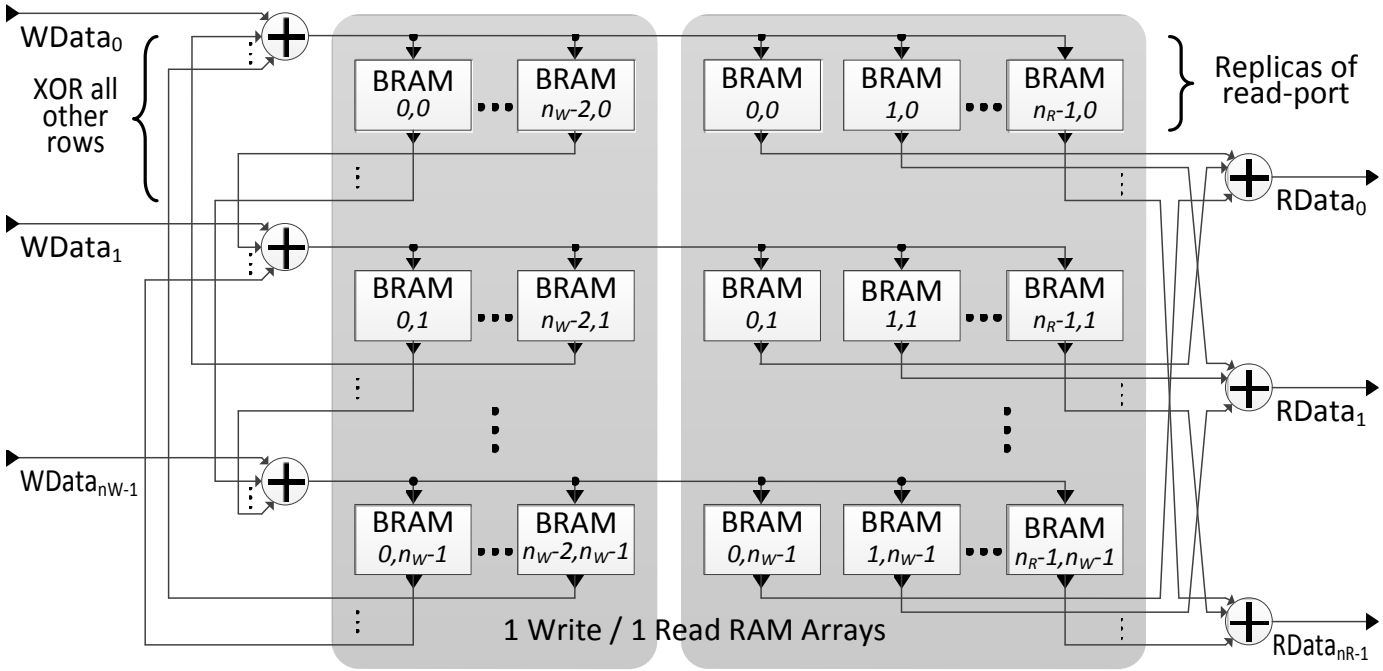
LVT-based Multi-ported RAM Example (2)



LVT-based Multi-ported RAM Example (3)



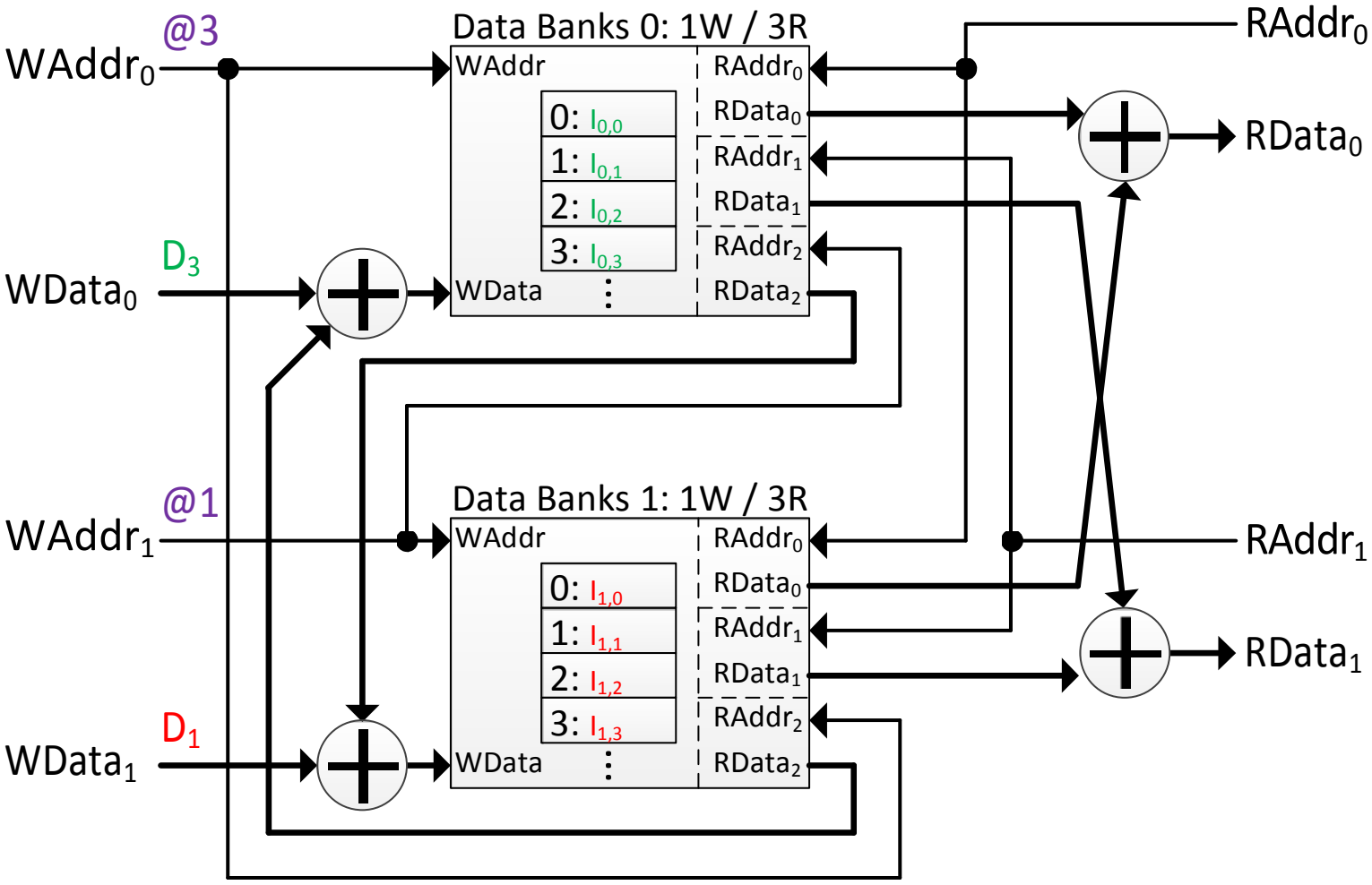
XOR-based Multi-ported RAM*



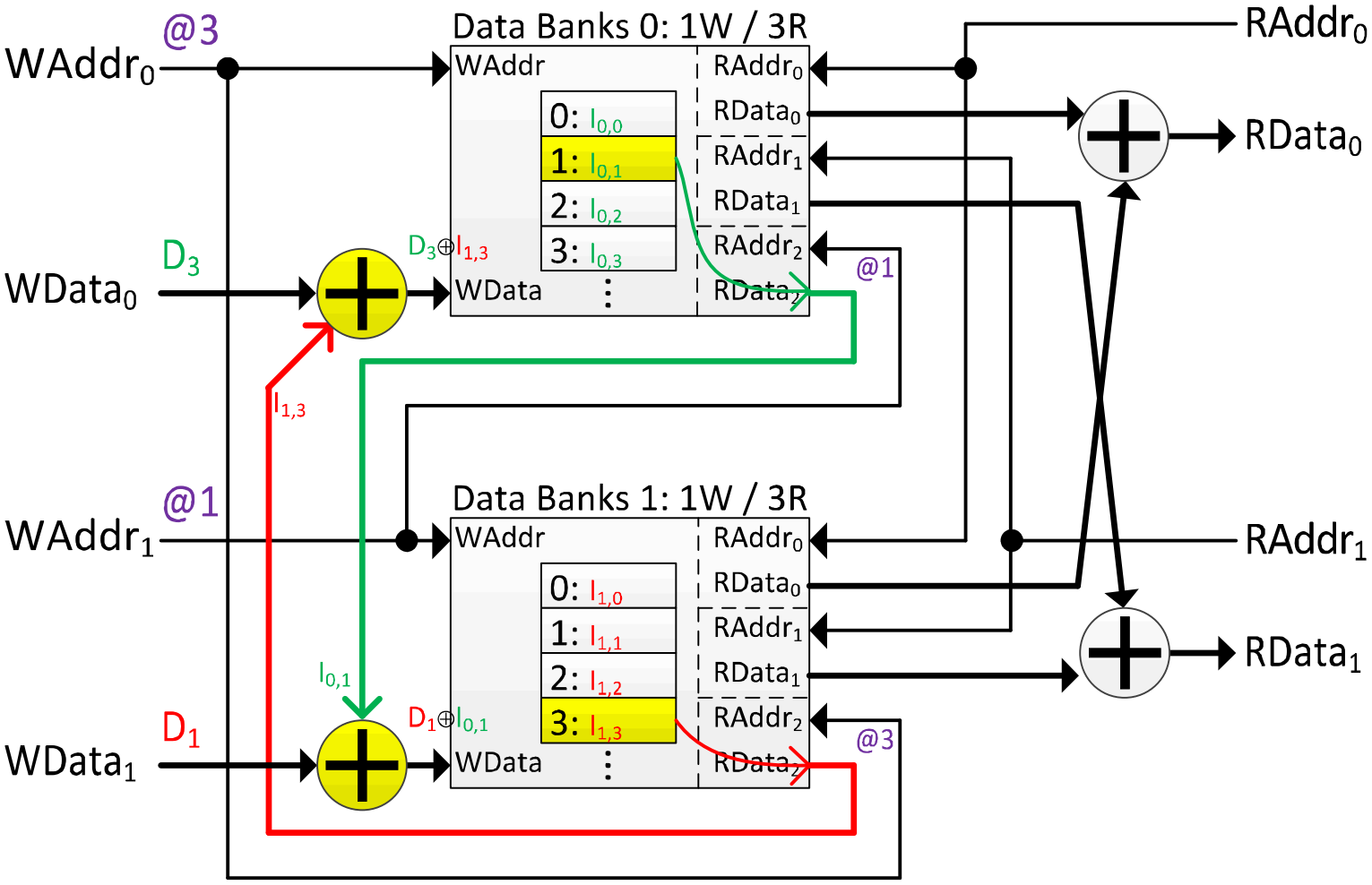
- SRAM-based
- XOR is used to embed and extract data back:
  Embed: *DATA=OLD⊕NEW*
  Extract: *DATA⊕OLD=OLD⊕NEW⊕OLD=NEW*

*[Laforest et al. ACM/SIGDA FPGA, Feb. 2012]

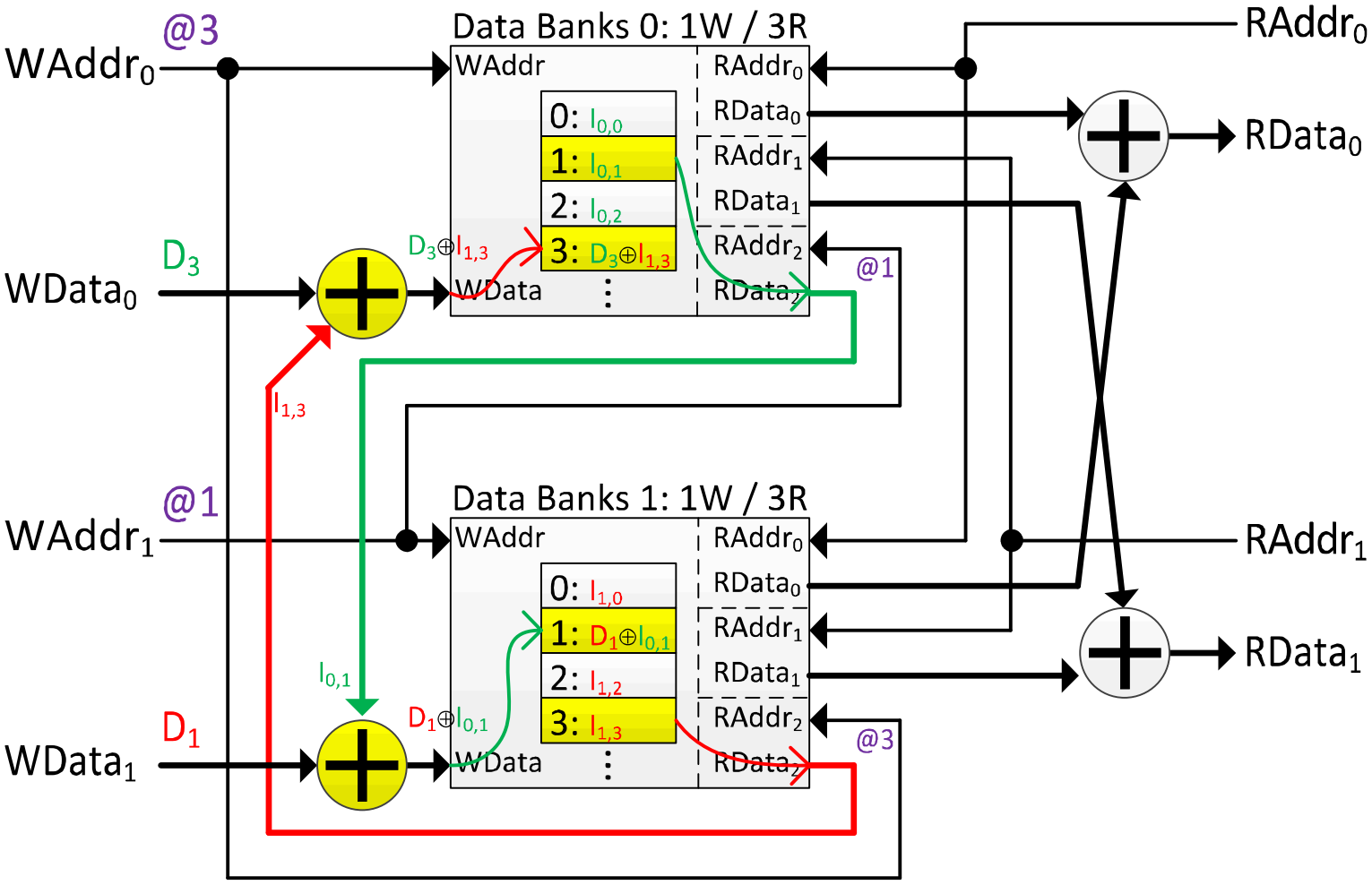
XOR-based Multi-ported RAM Example (1)



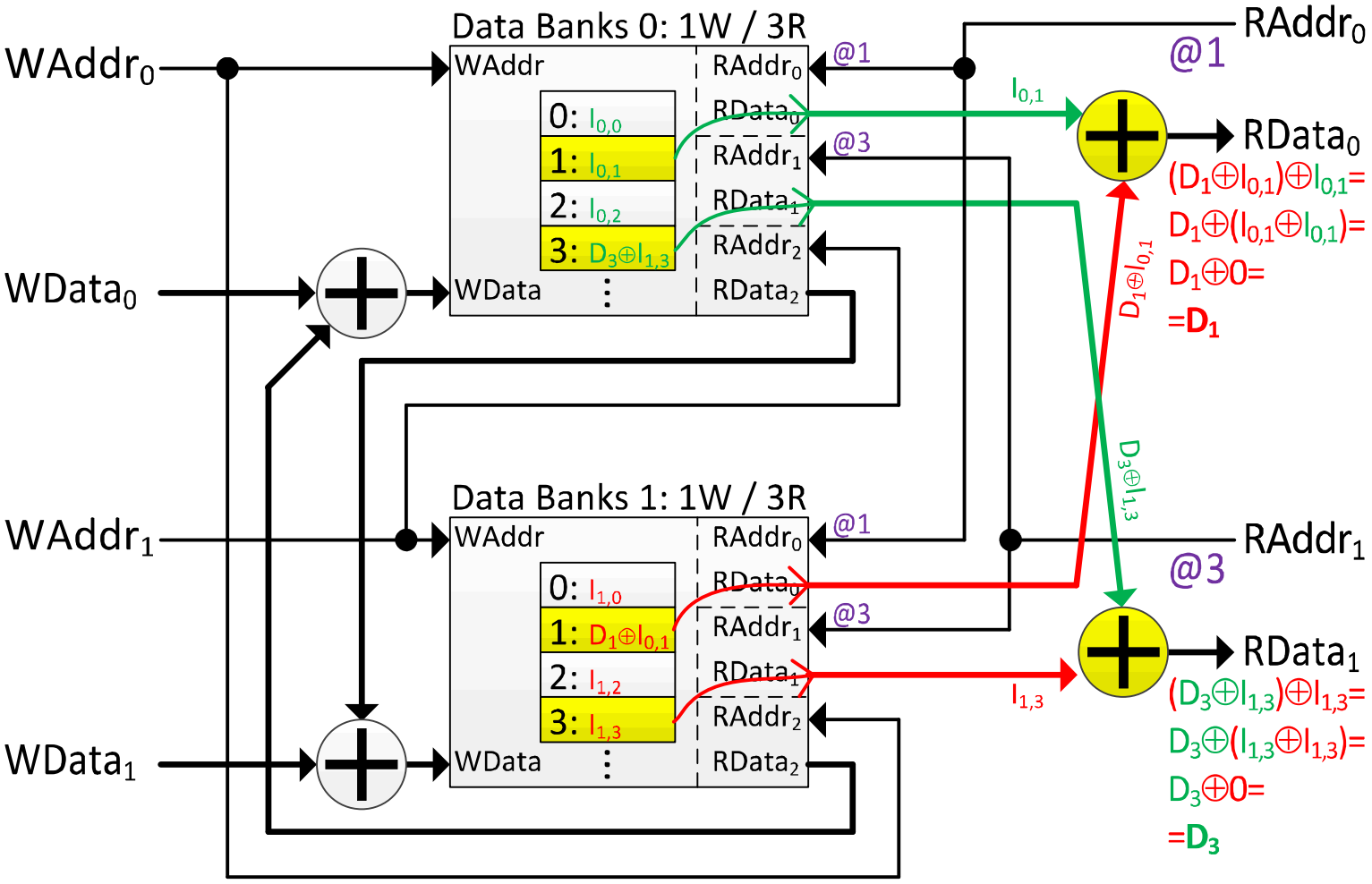
XOR-based Multi-ported RAM Example (2)



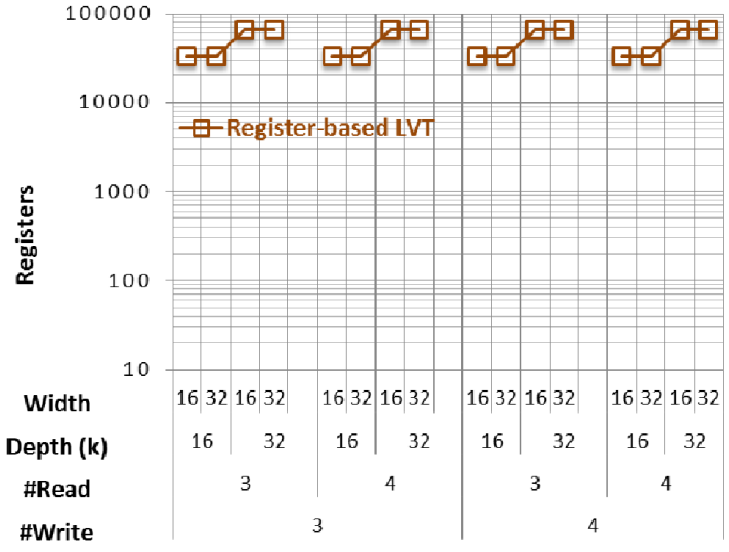
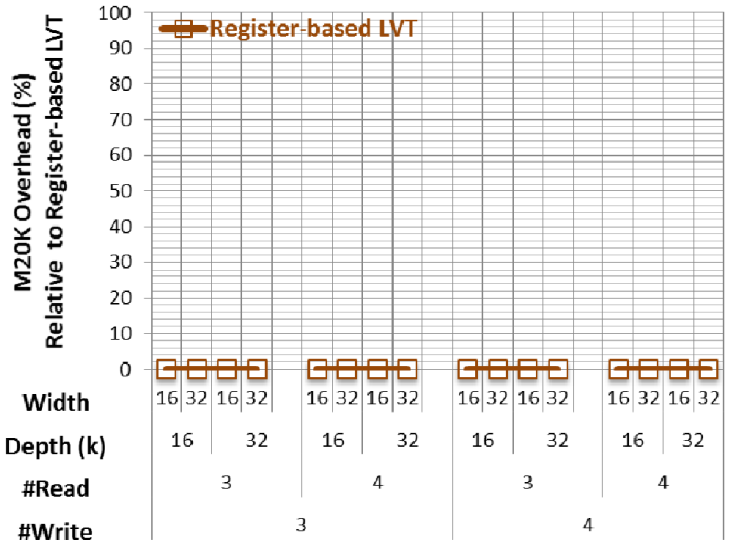
XOR-based Multi-ported RAM Example (3)



XOR-based Multi-ported RAM Example (4)

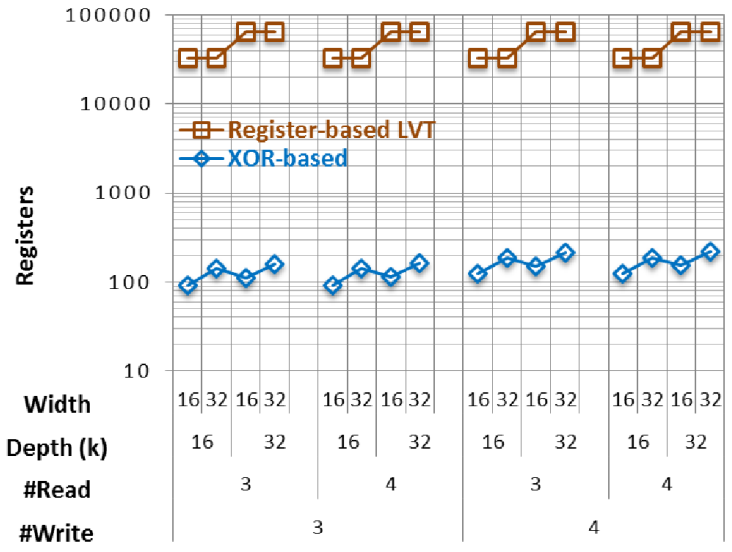


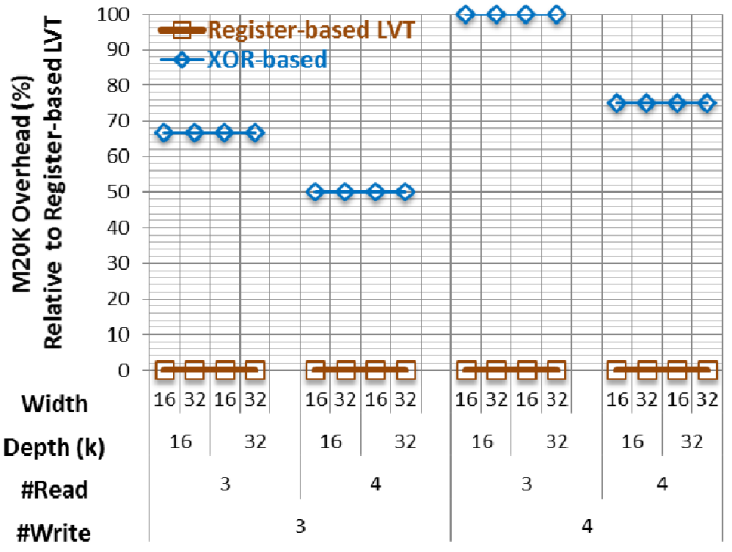
Motivation

	#Registers	#BRAMs
Register-based LVT		
	✗	✓

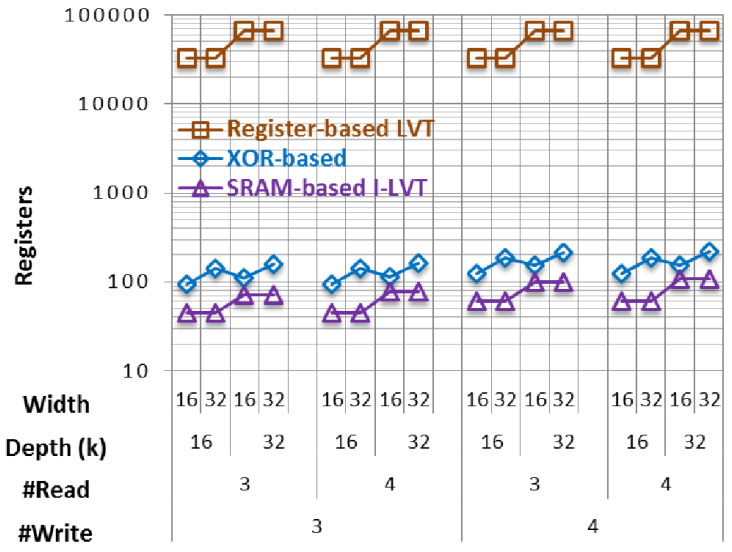
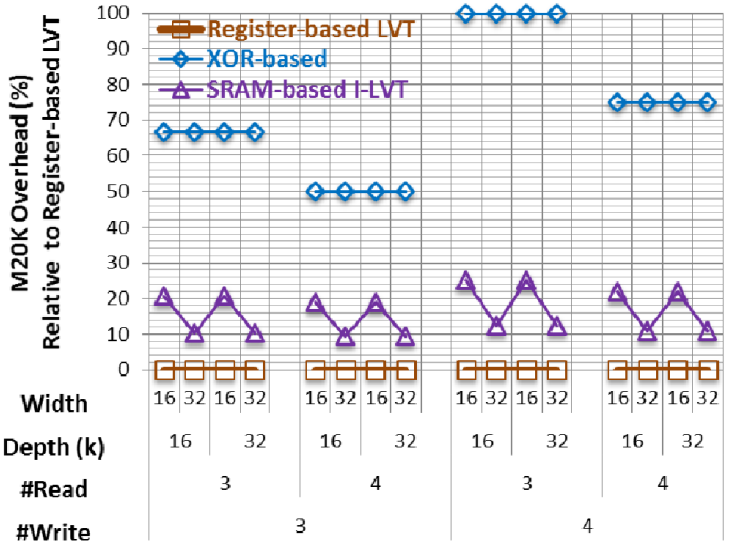
Motivation

	#Registers	#BRAMs
Register-based LVT	✗	✓
XOR-based	✓	✗



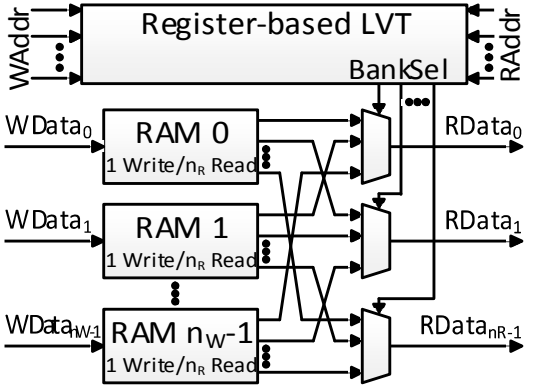


Motivation

	#Registers	#BRAMs
		
Register-based LVT	✗	✓
XOR-based	✓	✗
Proposed I-LVT	✓	✓

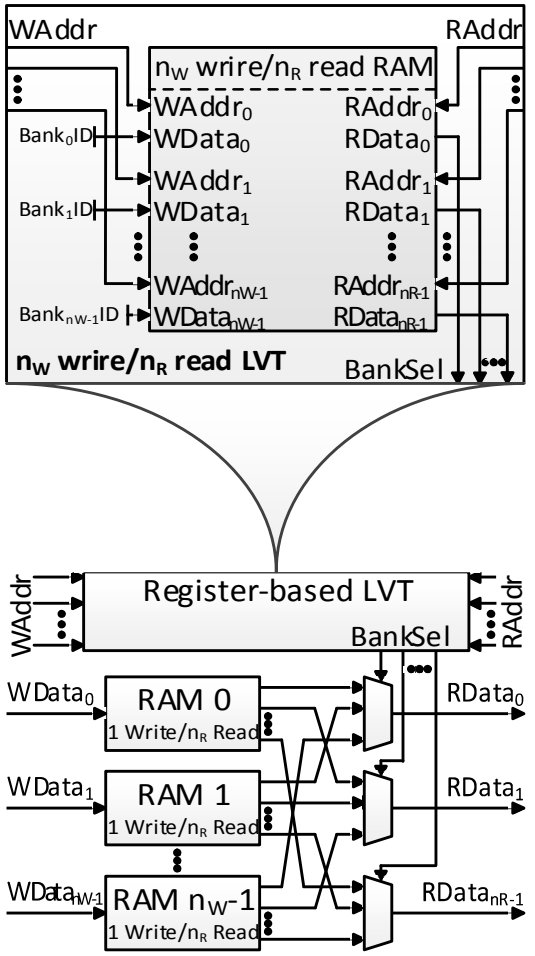
Method

- Based on LVT approach



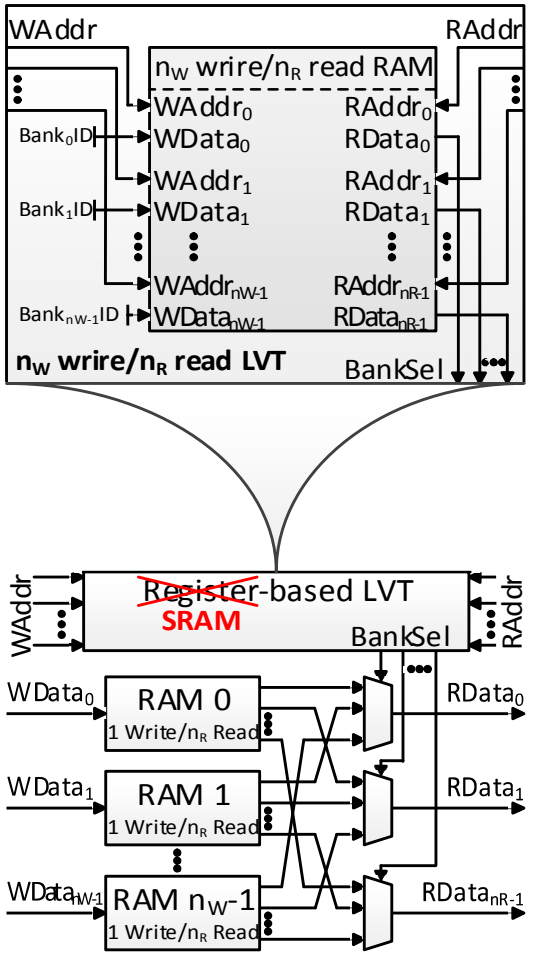
Method

- Based on LVT approach
- The LVT is a multi-ported RAM with constant inputs (bank IDs)



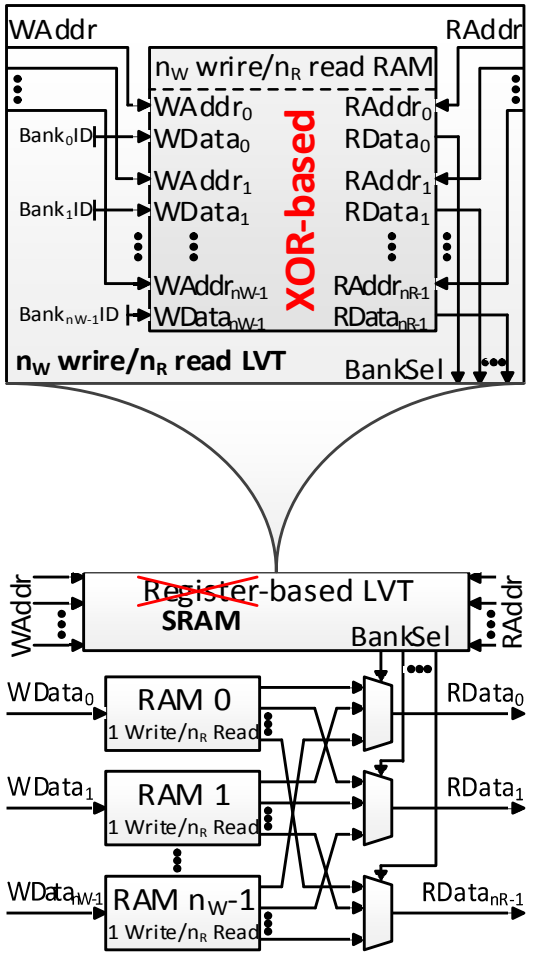
Method

- Based on LVT approach
- The LVT is a multi-ported RAM with constant inputs (bank IDs)
- SRAM-based LVT



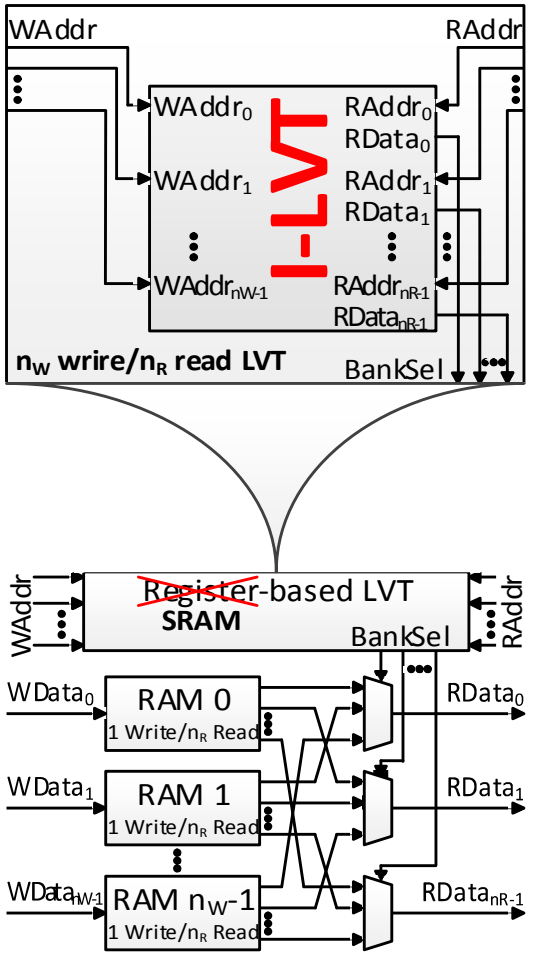
Method

- Based on LVT approach
- The LVT is a multi-ported RAM with constant inputs (bank IDs)
- SRAM-based LVT
  - Can be implemented with XOR-based multi-ported RAM



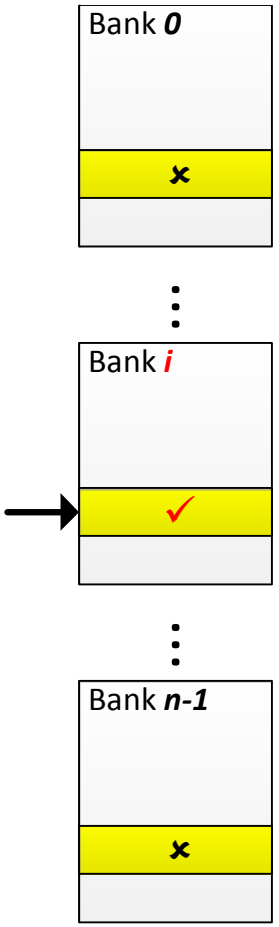
Method

- Based on LVT approach
- The LVT is a multi-ported RAM with constant inputs (bank IDs)
- SRAM-based LVT
  - Can be implemented with XOR-based multi-ported RAM
  - Is generalized by the proposed I-LVT approach
  - Two special cases are provided:
    - Binary-coded I-LVT
    - One-hot-coded I-LVT



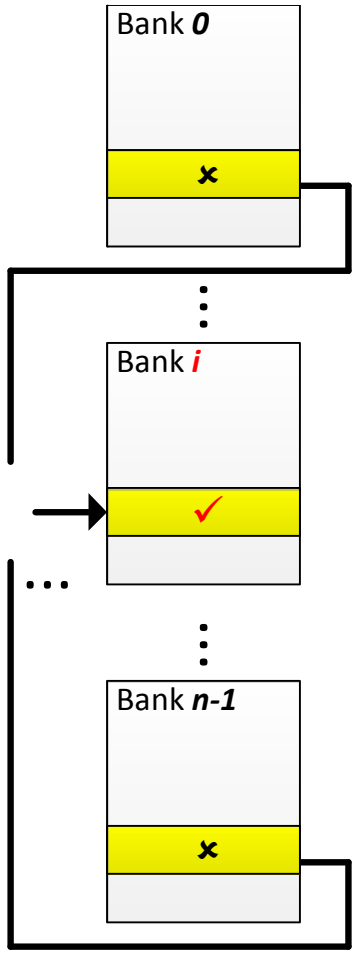
Invalidation Table Approach (I-LVT)

- A bank for each write
- A single write to a specific bank invalidates all the other banks



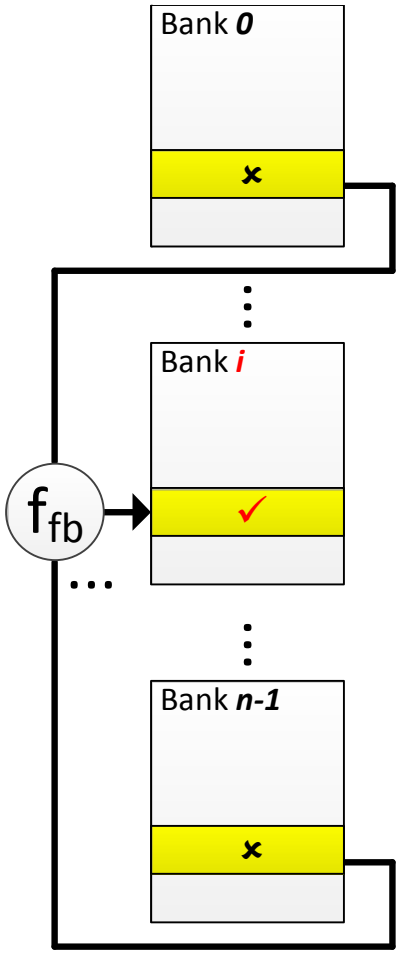
Invalidation Table Approach (I-LVT)

- A bank for each write
- A single write to a specific bank invalidates all the other banks
- Feedbacks are received from all other banks



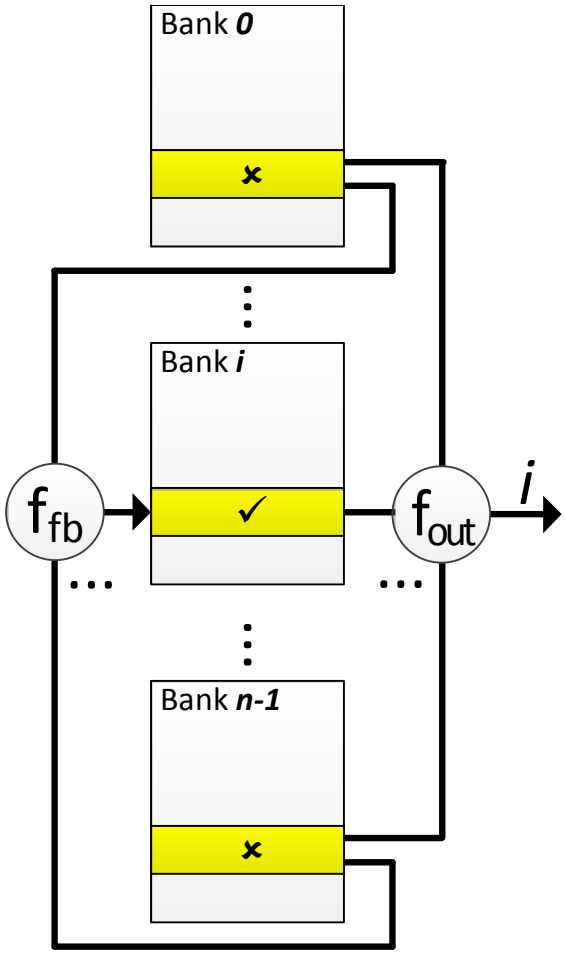
Invalidation Table Approach (I-LVT)

- A bank for each write
- A single write to a specific bank invalidates all the other banks
- Feedbacks are received from all other banks
- f_{fb} generates a new data that contradicts all the other banks



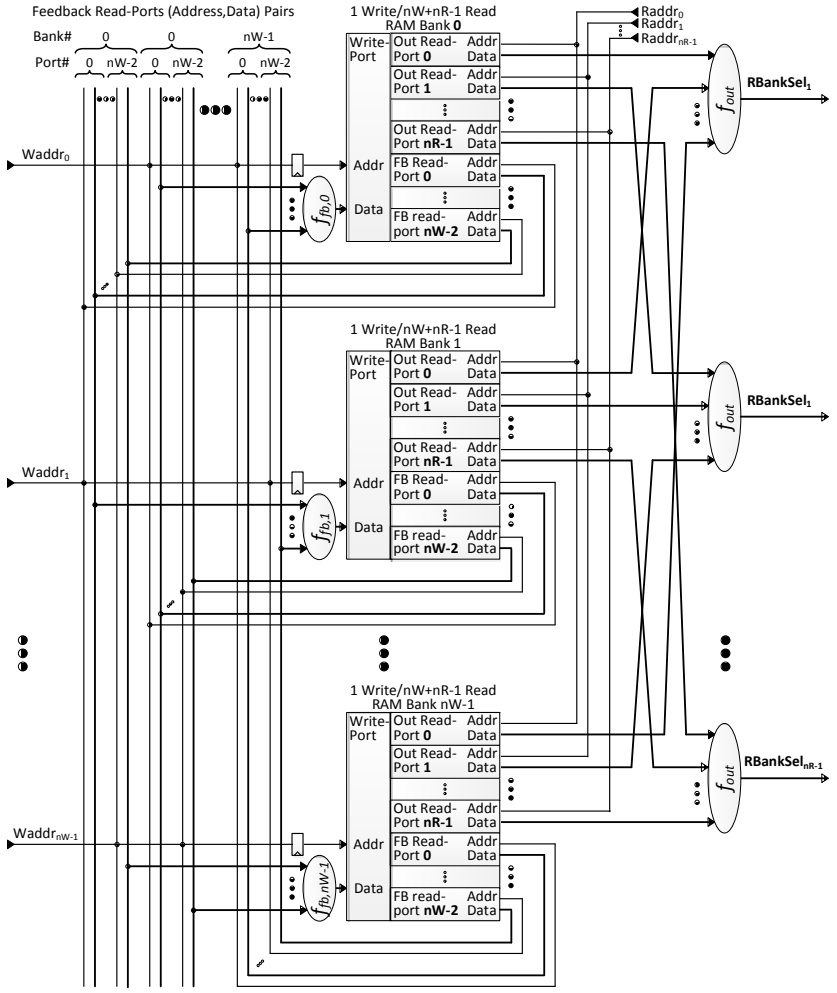
Invalidation Table Approach (I-LVT)

- A bank for each write
- A single write to a specific bank invalidates all the other banks
- Feedbacks are received from all other banks
- f_{fb} generates a new data that contradicts all the other banks
- f_{out} detects the uncontradicted bank ID



Invalidation Table Approach (I-LVT)

- A bank for each write
- A single write to a specific bank invalidates all the other banks
- Feedbacks are received from all other banks
- f_{fb} generates a new data that contradicts all the other banks
- f_{out} detects the uncontradicted bank ID



Bank ID Embedding: Binary-coded Bank Selectors

Feedback function for bank k:

$$f_{fb,k} = k \bigoplus_{\substack{0 \le i \le n_W \\ i \ne k}} Bank_i[WAddr_k]$$

Output function (all banks):

$$f_{out,k} = \bigoplus_{0 \le i \le n_W} Bank_i[RAddr_k]$$

Mutual-exclusive Conditions: One-hot-coded Bank Selectors

Feedback function for bank *k*:

$$\boldsymbol{f}_{\boldsymbol{fb,k}}\langle \boldsymbol{i}\rangle\big|_{\mathbf{0}\le \boldsymbol{i}<\boldsymbol{n_W}-\mathbf{1}} : Bank_{\mathrm{k}}[WAddr_k]\langle \mathrm{i}\rangle \leftarrow \begin{Bmatrix} i<k & \overline{Bank_{\iota}[WAddr_k]\langle k-1\rangle} \\ else & Bank_{i+1}[WAddr_k]\langle k\rangle \end{Bmatrix}$$

Output function (check if condition match):

$$\boldsymbol{f}_{\boldsymbol{out,k}}\langle \boldsymbol{i}\rangle\big|_{\mathbf{0}\le \boldsymbol{i}<\boldsymbol{n_W}-\mathbf{1}} : Bank_{\mathrm{k}}[WAddr_k]\langle \iota\rangle \oplus \overline{\begin{Bmatrix} \iota<k & \overline{Bank_{\iota}[WAddr_k]\langle k-1\rangle} \\ else & Bank_{\iota+1}[WAddr_k]\langle k\rangle \end{Bmatrix}}$$

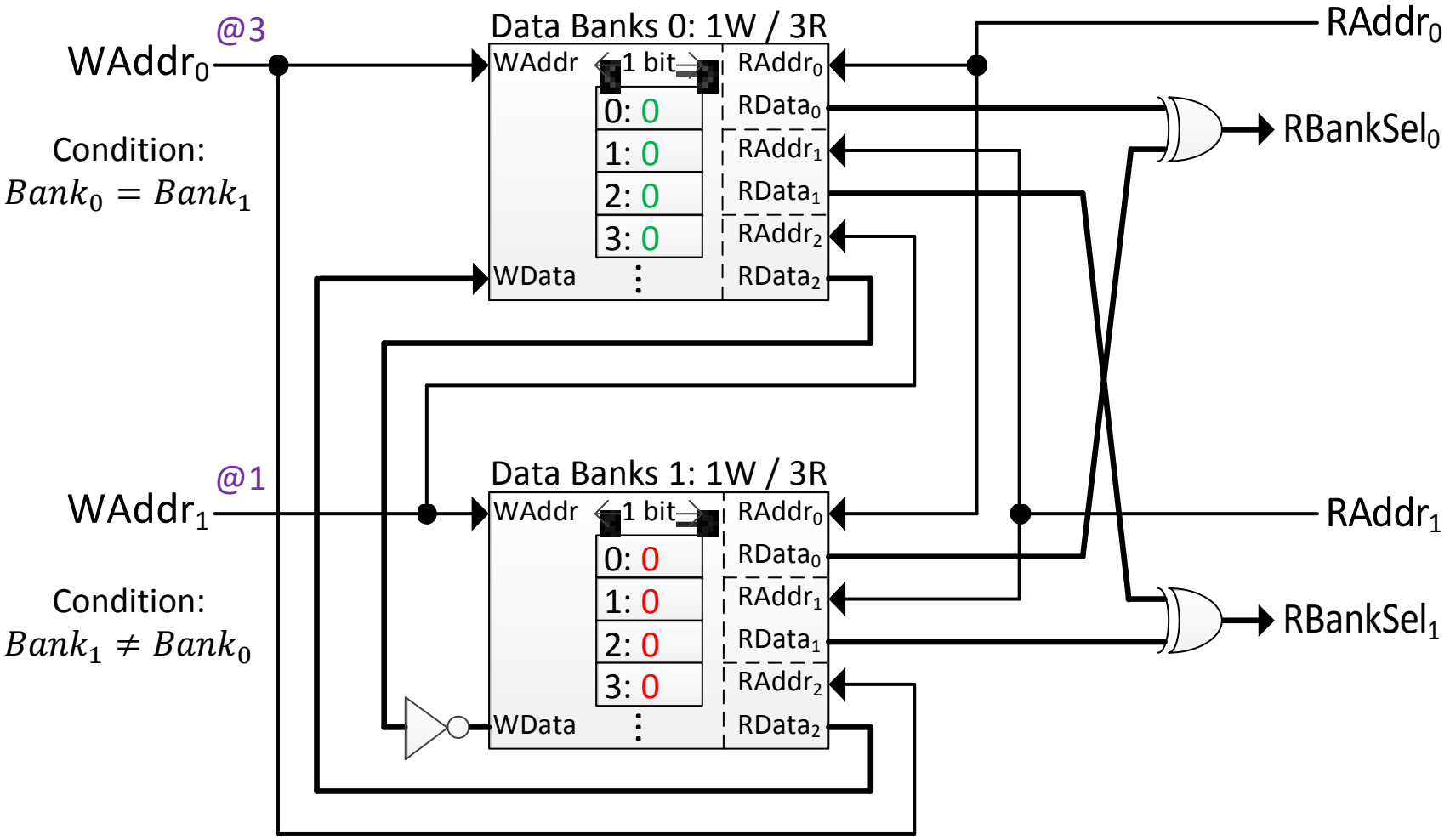
Mutual-exclusive Conditions Examples

- Each lines pair has a negated conditions
- One and only one line is logically true

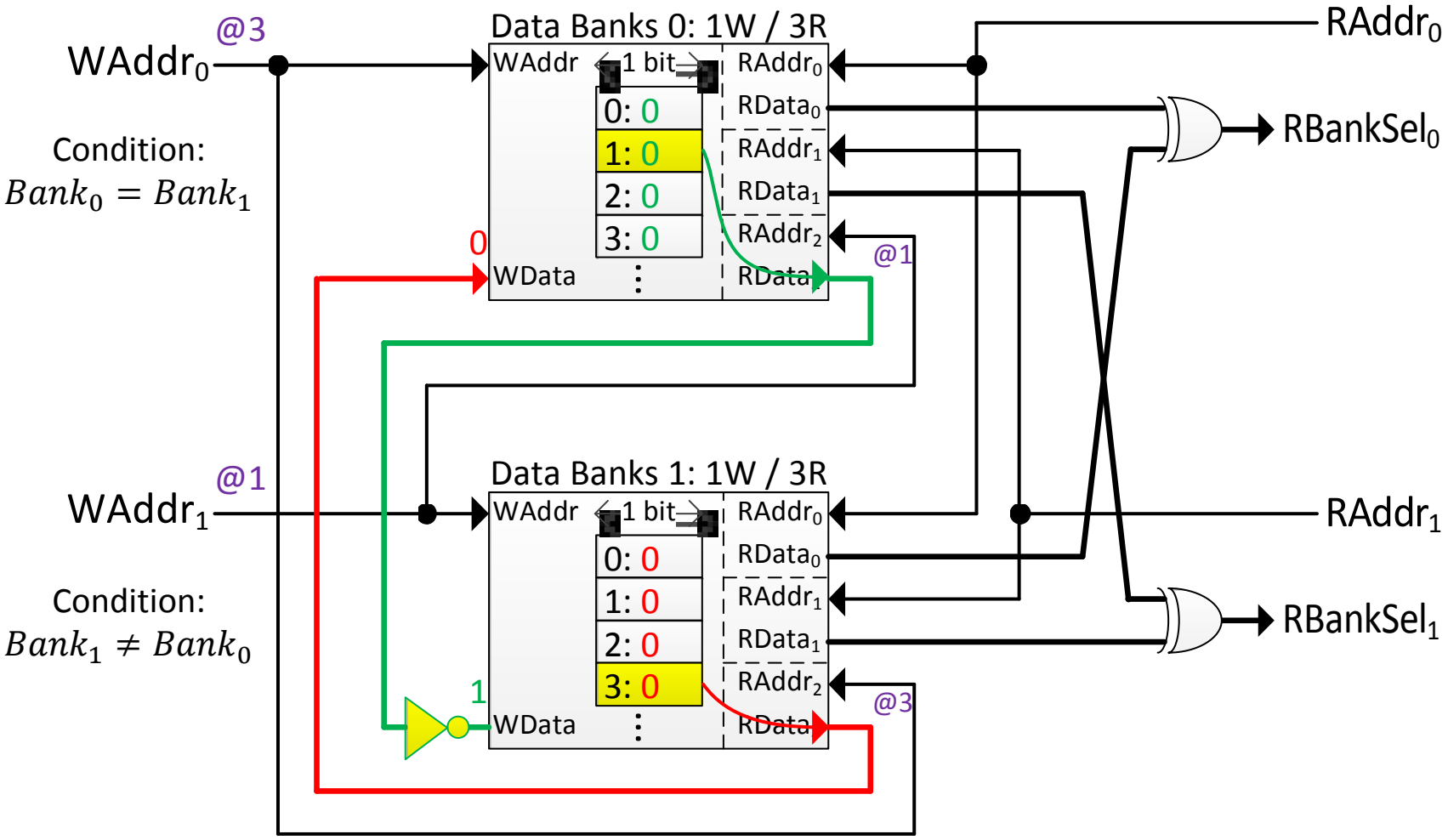
$$n_W = 2: \begin{cases} f_{fb,0}: Bank_0\langle 0\rangle \leftarrow Bank_1\langle 0\rangle \\ f_{fb,1}: Bank_1\langle 0\rangle \leftarrow \overline{Bank_0\langle 0\rangle} \end{cases}$$

$$n_W = 3: \begin{cases} f_{fb,0}: Bank_0\langle 1:0\rangle \leftarrow \langle Bank_2\langle 0\rangle, Bank_1\langle 0\rangle\rangle \\ f_{fb,1}: Bank_1\langle 1:0\rangle \leftarrow \langle Bank_2\langle 1\rangle, \overline{Bank_0\langle 0\rangle}\rangle \\ f_{fb,2}: Bank_2\langle 1:0\rangle \leftarrow \langle \overline{Bank_1\langle 1\rangle}, \overline{Bank_0\langle 1\rangle}\rangle \end{cases}$$

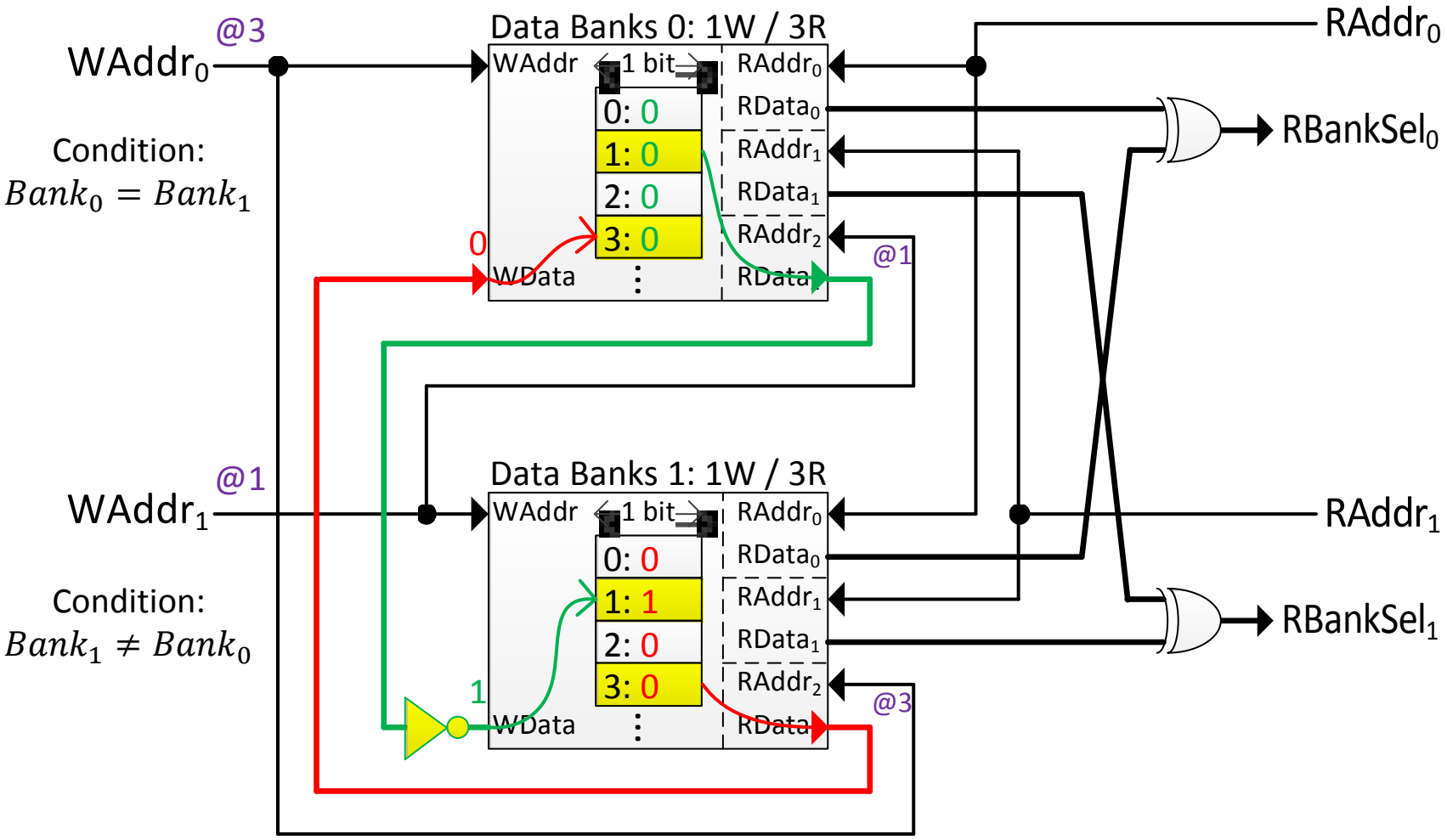
One-hot/Binary Coded 2W/2R Example (1)



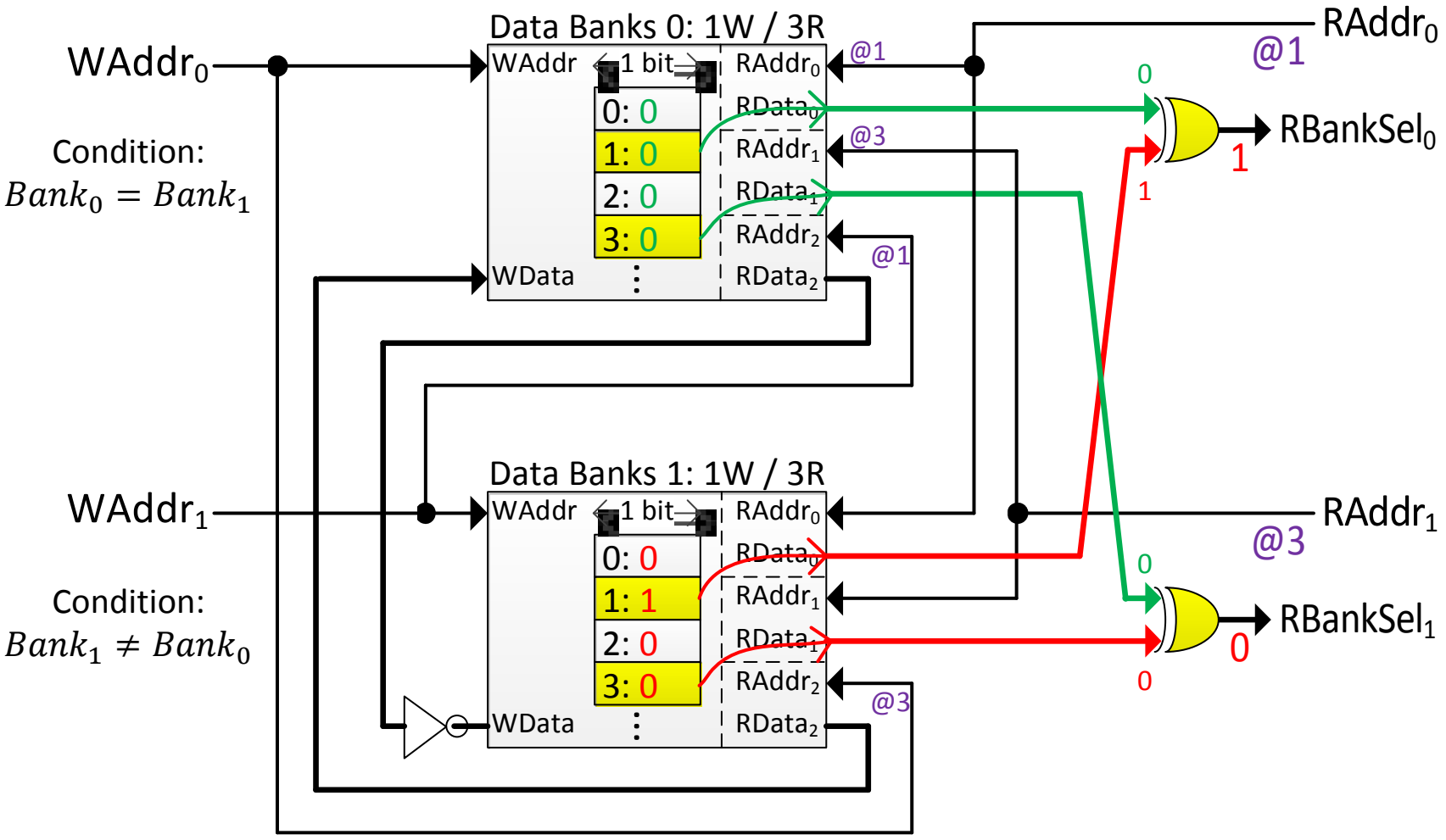
One-hot/Binary Coded 2W/2R Example (2)



One-hot/Binary Coded 2W/2R Example (3)

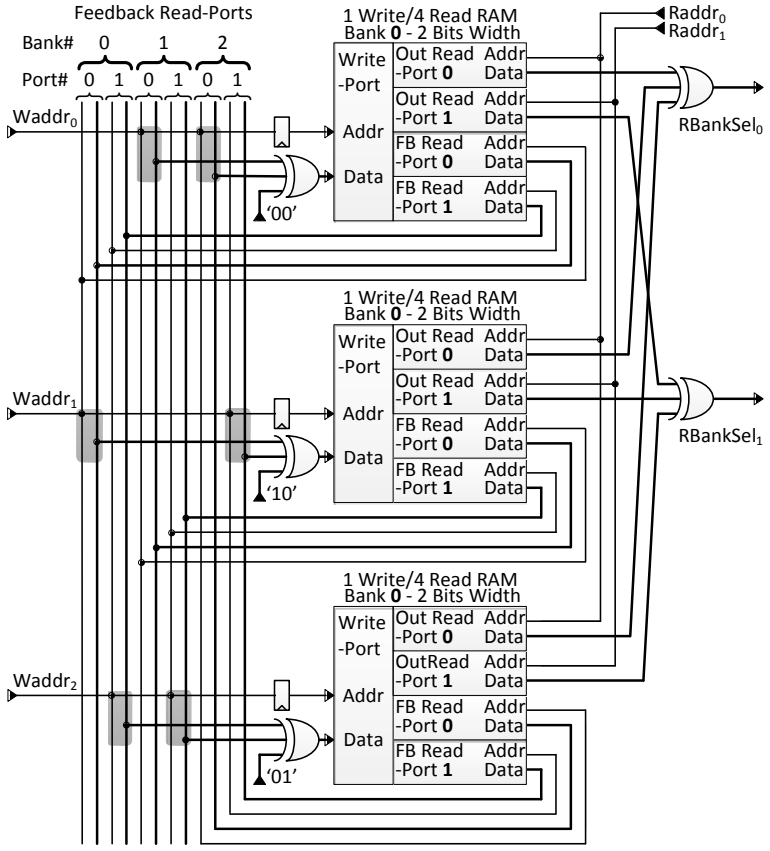


One-hot/Binary Coded 2W/2R Example (4)

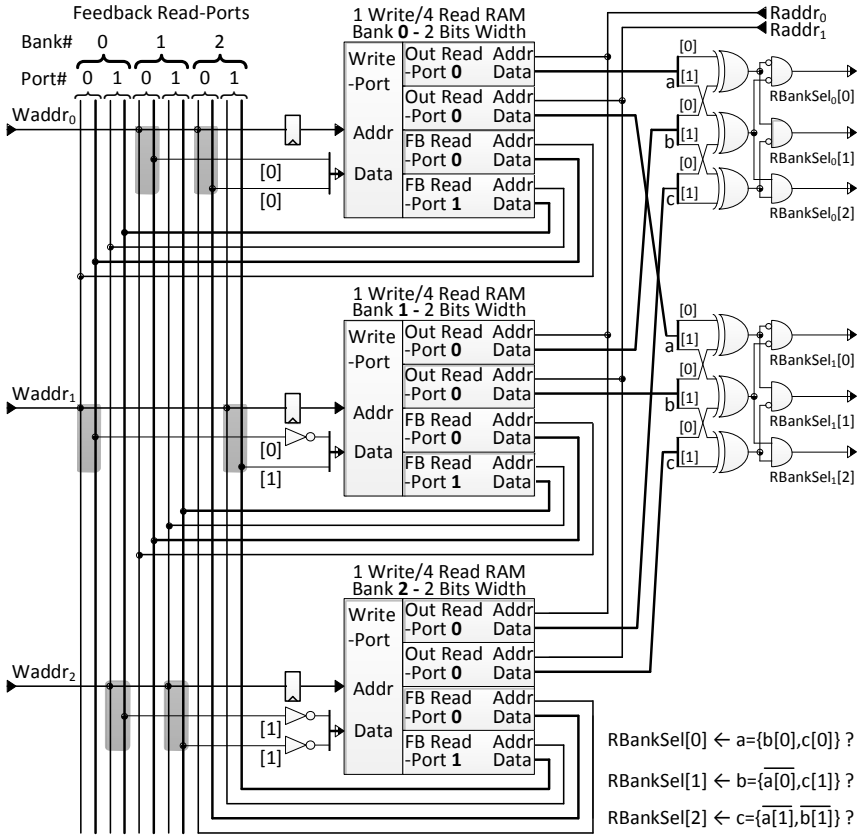


3W/2R I-LVT Implementation

Binary-coded I-LVT



One-hot-coded I-LVT



SRAM Consumption

Register-based LVT	$d \cdot w \cdot n_W \cdot n_R$
XOR-based	$d \cdot w \cdot n_W \cdot n_R + d \cdot w \cdot n_W \cdot (n_W - 1)$
Binary-coded I-LVT	$d \cdot w \cdot n_W \cdot n_R + d \cdot \lceil \log_2 n_W \rceil \cdot n_W \cdot (n_W - 1) + d \cdot \lceil \log_2 n_W \rceil \cdot n_W \cdot n_R$
One-hot-coded I-LVT	$d \cdot w \cdot n_W \cdot n_R + d \cdot (n_R + 1) \cdot n_W \cdot (n_W - 1)$

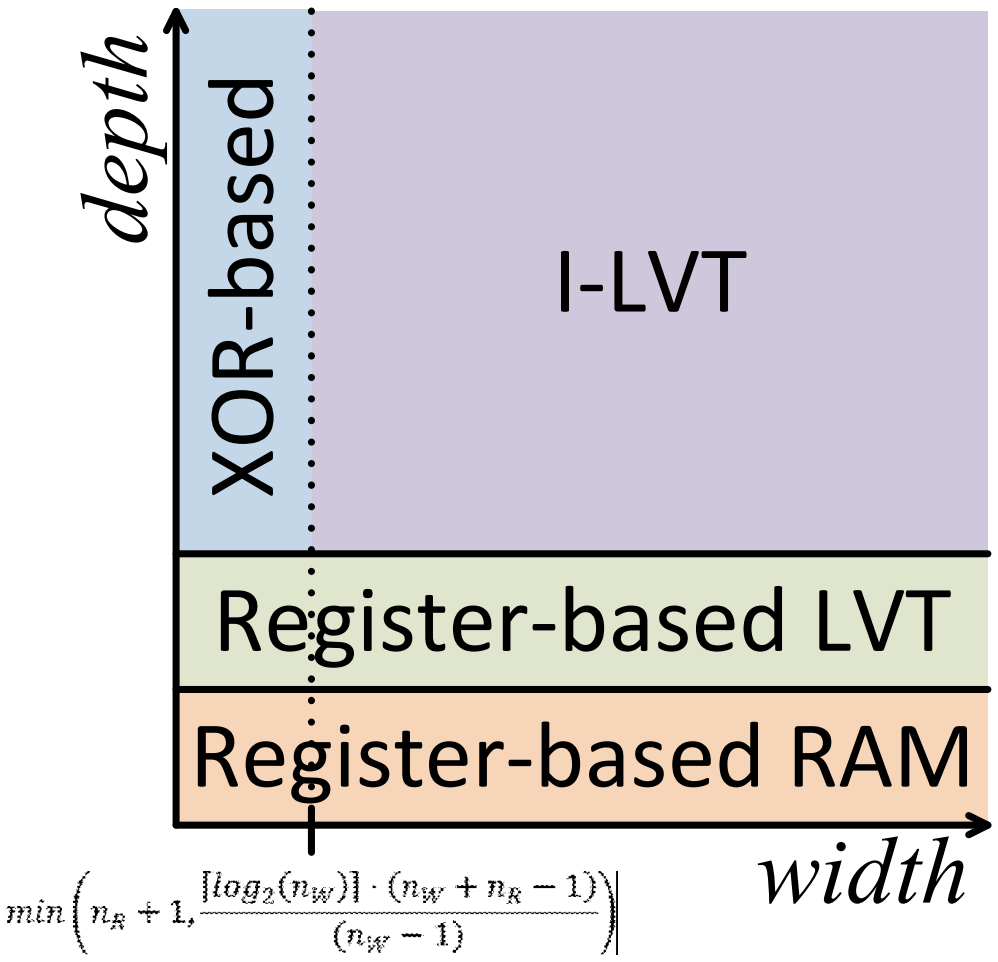
- XOR-based consumes fewer SRAM cells if:

  $w < min\left(n_R + 1, \frac{\lceil log_2(n_W) \rceil \cdot (n_W + n_R - 1)}{(n_W - 1)}\right)$ (Unlikely!!)

- Otherwise, one-hot consumes fewer SRAM cells than binary-coded if:

  $$1 < n_W \leq 3 \;\; OR \;\; n_R < \left. \frac{(n_W - 1) \cdot (\lceil log_2(n_W) \rceil - 1)}{(n_W - 1) - \lceil log_2(n_W) \rceil} \right|_{n_W > 3}$$

Usage Guideline



Experimental Environment

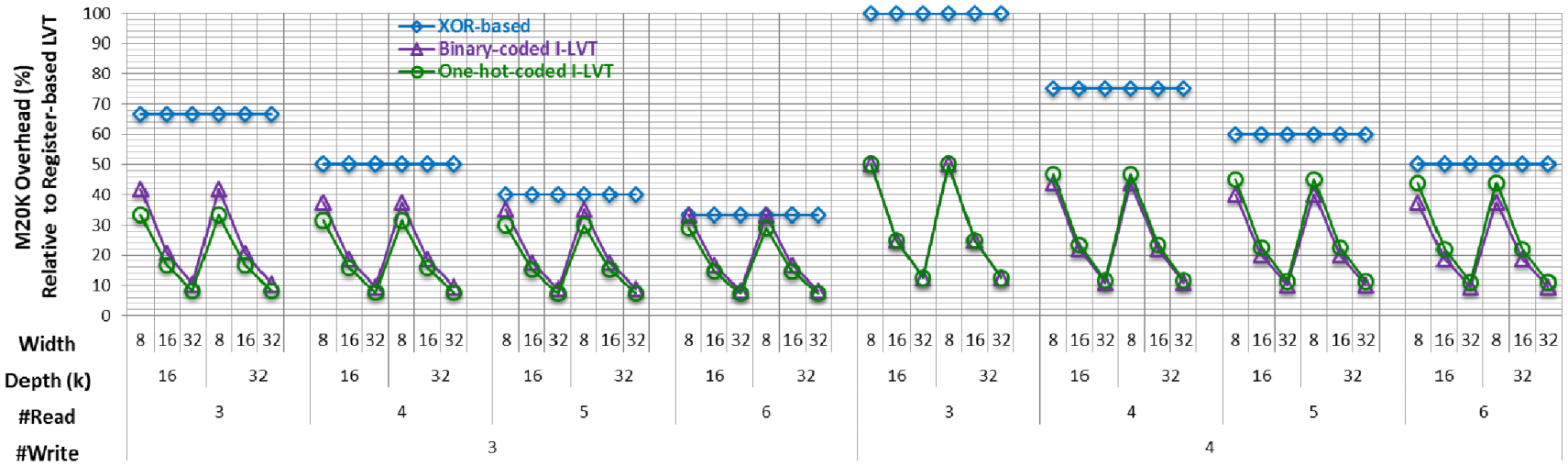
- Different ~1k designs have been synthesized with various parameters sweep
  - Altera's Quartus II with Altera's Stratix V device
- Verified with Altera's ModelSim
  - Over Million RAM cycles for each configuration
- Bypassing capability:
  - New data read-after-write (same as Altera's M20K)
  - New data read-during-write (same as a single register)
- Parameterized Verilog and simulation/synthesis run-in-batch manager are available online:

  https://code.google.com/p/multiported-ram/

Experimental Results

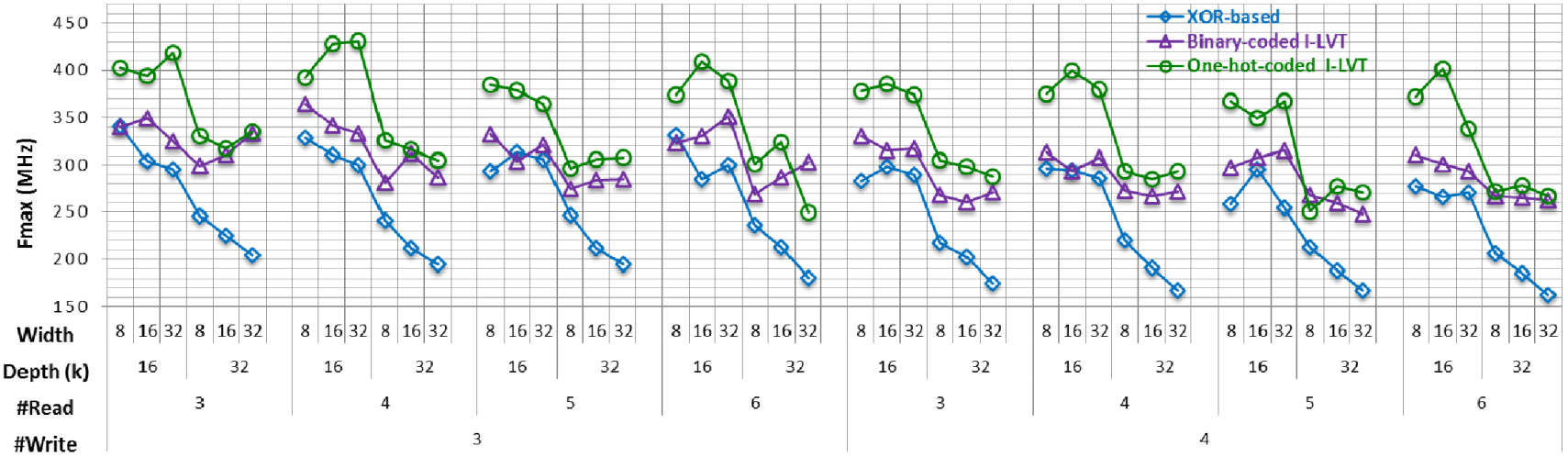
BRAM Consumption

- Compared to XOR-based approach:
  - ➢ Average of 19%; up to 44% BRAM reduction
- #BRAM compared to 32bit wide register-based LVT:
  - – Up to 200 % in XOR-based
  - – Up to 12.5% in I-LVT-based



Experimental Results
Fmax

- Compared to XOR-based approach:
  - ➢ Average of 38%; up to 76% Fmax increase
- One-hot-coded I-LVT exhibits the highest Fmax
  - – Due to fast feedback paths
  - – BRAM consumption still within 6% of the minimal



Conclusions

Modular multi-ported SRAM-based memories for embedded systems

- Based on dual-ported BRAMs
- Dramatically lower resources consumption and higher performance than previous approaches
  - Close to register-based LVT BRAM consumption; No further significant improvement can be done
- Additional features e.g. bypassing and initializing
- Ready to use open source parameterized Verilog and a run-in-batch manager are available online

https://code.google.com/p/multiported-ram/

Thank You