Delay-Optimal Technology Mapping for In-Memory Computing using ReRAM Devices

Debjyoti Bhattacharjee and Anupam Chattopadhyay

School of Computer Science and Engineering
Nanyang Technological University,
Singapore

Email : {debjyoti, anupam}@ntu.edu.sg

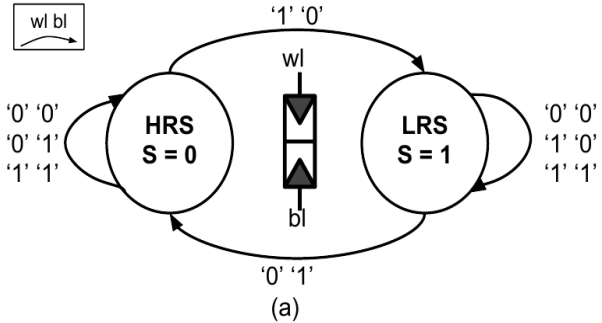
NANYANG TECHNOLOGICAL UNIVERSITY

Outline

- ReRAM fundamentals
- Problem statement
- Logic Representation and Technology mapping
- Delay Optimal Technology Mapping
- Device Reduction
- Dispatch parallelism constrained scheduling
- Experimental Results
- Conclusion

ReRAM fundamentals

- Two input terminals - *wl* (wordline) and *bl* (bitline)
- Internal resistive state *S* acts as third input.



wl	bl	NS
'0'	'0'	S
'0'	'1'	'0'
'1'	'0'	'1'
'1'	'1'	S

(b)

- Next state NS is Boolean majority of the three inputs (M_3) with the bitline input inverted.

$$NS = M_3(S, wl, \neg bl)$$

NANYANG TECHNOLOGICAL UNIVERSITY

ReRAM fundamentals

- In a clock cycle/step, a device can be read out
  The *readout* value can be applied as input to the wordline or bitline of any other device.

- The *primary inputs (PI)* and the *inverted values* of the *PI*, constants `1' and `0', can be applied directly to the wordlines and bitlines of the devices.

Outline

- ReRAM fundamentals
- Problem statement
- Logic Representation and Technology mapping
- Delay Optimal Technology Mapping
- Device Reduction
- Dispatch parallelism constrained scheduling
- Experimental Results
- Conclusion

Problem statement

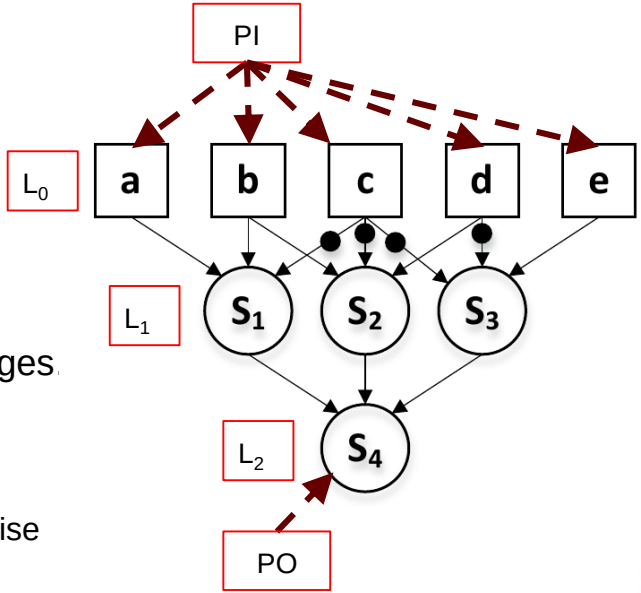
- Determine a sequence of inputs, to be applied, to the *wordlines* and the *bitlines* of ReRAM 1S1R devices for computation of a Boolean function
- The *delay* of the obtained mapping is equal to the number of *steps* that the mapping contains
- The mapping is ***delay − optimal*** if the number of *steps* is ***minimum***.

Outline

- ReRAM fundamentals
- Problem statement
- Logic Representation and Technology mapping
- Delay Optimal Technology Mapping
- Device Reduction
- Dispatch parallelism constrained scheduling
- Experimental Results
- Conclusion

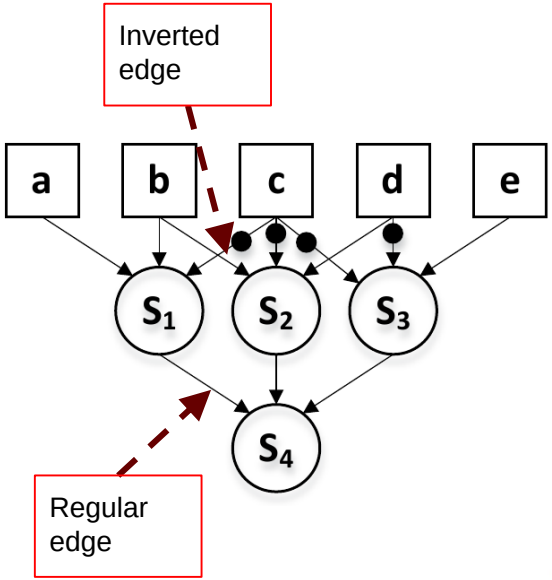
Logic representation

- Boolean functions are represented using
  - *Majority Inverter Graph (MIG)*
- Primary Input (PI)
  - Nodes without incoming edges
- Internal node
  - Node with incoming edges.
- Primary Output (PO)
  - Internal node without any outgoing edges
- Level of a node:
  - level(node) = 0, if node is PI
  - level(*node*) = max(level(succ))+1, otherwise



Logic representation

- Each internal node represents a Boolean majority with 3 inputs (M_3).
- Directed edge $i \rightarrow j$ indicates that output of node i is input to node j
- Edges can be *regular* or *inverted*.



A Majority Inverter Graph (MIG)

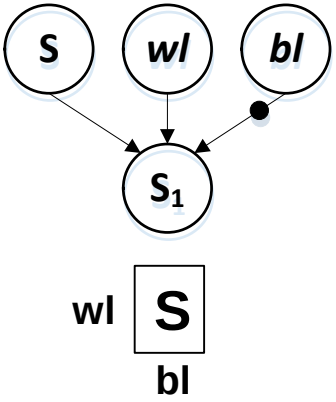
Technology mapping

- Native function realized by ReRAM devices

$$S_1 = M_3(S, wl, \neg bl)$$

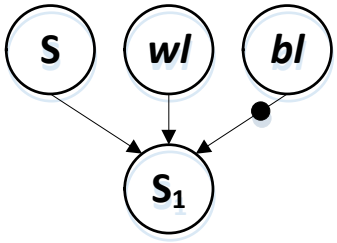
- A device holds the state S
  - This is the “host” for computation
- *wl* and *bl* are applied to the wordline and bitline of the “host” to compute S_1.

- An internal node with a single inverted node can be directly realized using a single operation.



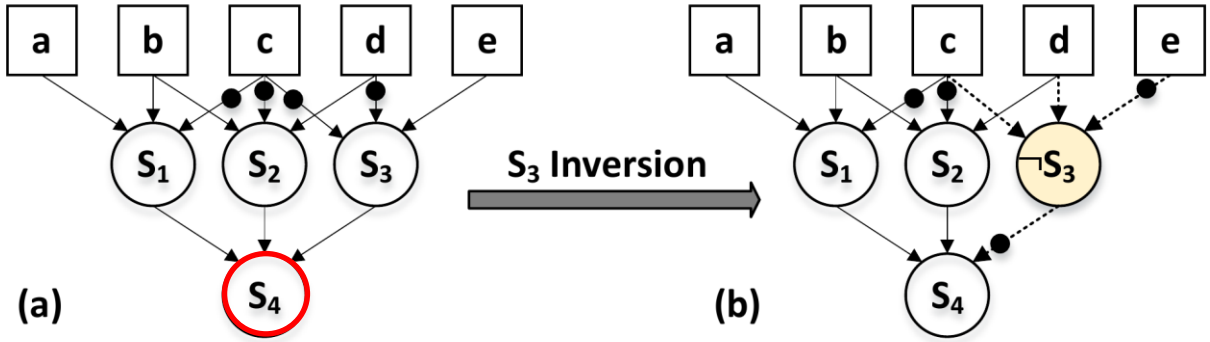
Technology mapping

- To enable computation of an internal node in a **single operation**
  - The node must have a *single inverted* input
  - A *host* has to be selected for computation



- We propose a set of transformations to the MIG which will ensure that these conditions are always true.

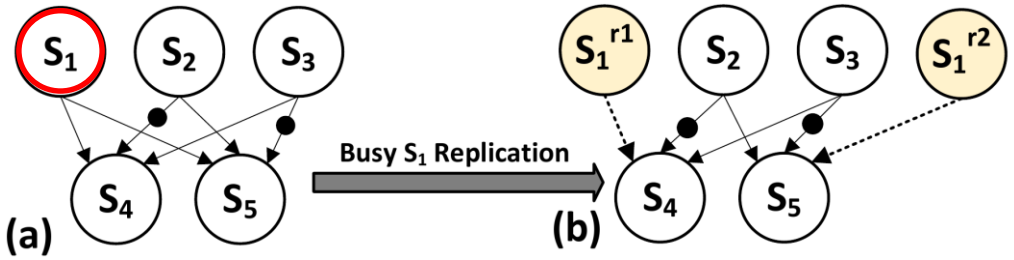
Inversion Transformation



- Node S_4 does not have any inverted inputs.
- Node S_3 is inverted and used as *bitline* input.
- This inversion is propagated bottom up.

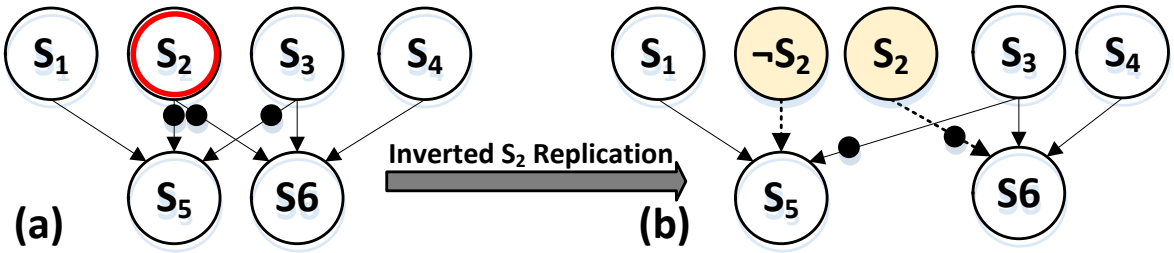
$$\neg M_3(S1, S2, S3) = M_3(\neg S1, \neg S2, \neg S3)$$

Busy Replication



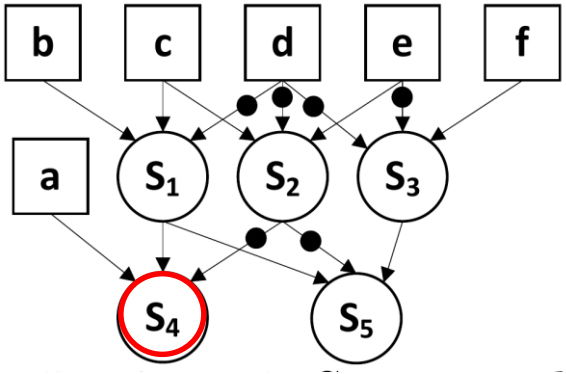
- To choose a predecessor of a node as “host”
  - It should not be “busy” ➔ multiple successors of a node implies it is “busy”.
- For nodes S_4 and S_5, S_1 is chosen as “host” but it is “busy”.
  - Replicas of the “host” node are therefore created.

Negation Replication



- Node S_2 is required in negated form by S_5 and regular form by node S_6
  - Therefore two copies of S_2 are required – one inverted and the other non-inverted.
- This is negation replication.

Preloading PI



- All the internal node predecessors of S_4 are busy
  - We can use “busy” replication, but that might lead to further replication of nodes at higher levels
- Instead, “a” can be pre-loaded in a new device and used for computation of S_4

Outline

- ReRAM fundamentals
- Problem statement
- Logic Representation and Technology mapping
- Delay Optimal Technology Mapping
- Device Reduction
- Dispatch parallelism constrained scheduling
- Experimental Results
- Conclusion

Delay-Optimal Technology mapping

```
Data: G, pi, po
Result: n2DMap,opList
/* prio(n_i)>prio(n_j), if level(n_i)>level(n_j)                    */
1 nodeHeap = maxHeap ();
2 n2DMap = HashMap ();
3 for node ∈ po do
4     nodeHeap.add (node);
5 while nodeHeap ≠ ∅ do
6     node = nodeHeap.pop();
7     ProcessNode(node);
8 PropagateDevice();
```

- ProcessNode determines the "host", wordline and bitline inputs of the node
  - To do so, it uses the transformations that were discussed earlier.
- ProcessNode adds the predecessors of the *node* to *nodeHeap* for processing.

NANYANG TECHNOLOGICAL UNIVERSITY

The proposed algorithm maps any MIG with k-levels using ReRAM devices using k +1 steps.

Outline

- ReRAM fundamentals
- Problem statement
- Logic Representation and Technology mapping
- Delay Optimal Technology Mapping
- Device Reduction
- Dispatch parallelism constrained scheduling
- Experimental Results
- Conclusion

Device Reduction by reuse

- We define start time and end time of a device
  - t_{start} : The first cycle in which a device is used.
  - t_{end} : The last cycle in which a device is used.
- One cycle is needed to reset a device
- Therefore a device d1 can be *reused* for operations of device d2 if
  - $d2.\ t_{start} > d1.\ t_{end} + 1$

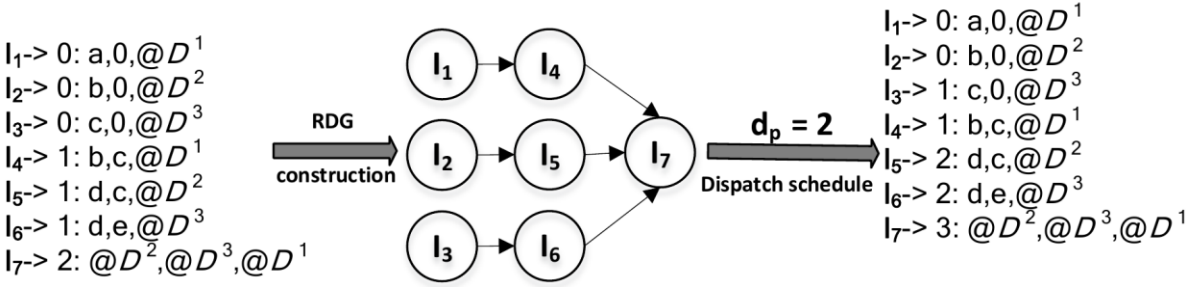
Outline

- ReRAM fundamentals
- Problem statement
- Logic Representation and Technology mapping
- Delay Optimal Technology Mapping
- Device Reduction
- Dispatch parallelism constrained scheduling
- Experimental Results
- Conclusion

Dispatch Parallelism

- The dispatch parallelism d_p of a controller is equal to the maximum number of operations that it can issue in a clock cycle.
- Dependencies among instructions can be shown by a ReRAM Dependency Graph (RDG).

RDG example with scheduling



- The initial set of operations that map a MIG is obtained using the delay-optimal mapping algorithm.
- From these instructions, the RDG is constructed.
- With dispatch parallelism 2, the RDG is scheduled using As-Soon-As-Possible scheduling heuristic.

Outline

- ReRAM fundamentals
- Problem statement
- Logic Representation and Technology mapping
- Delay Optimal Technology Mapping
- Device Reduction
- Dispatch parallelism constrained scheduling
- Experimental Results
- Conclusion

Experimental Results

- EPFL benchmarks were used for evaluation
- Implementation was done in Python3
- Execution system specifications
  - Ubuntu 14.04 with 16 cores and 64 GB RAM.

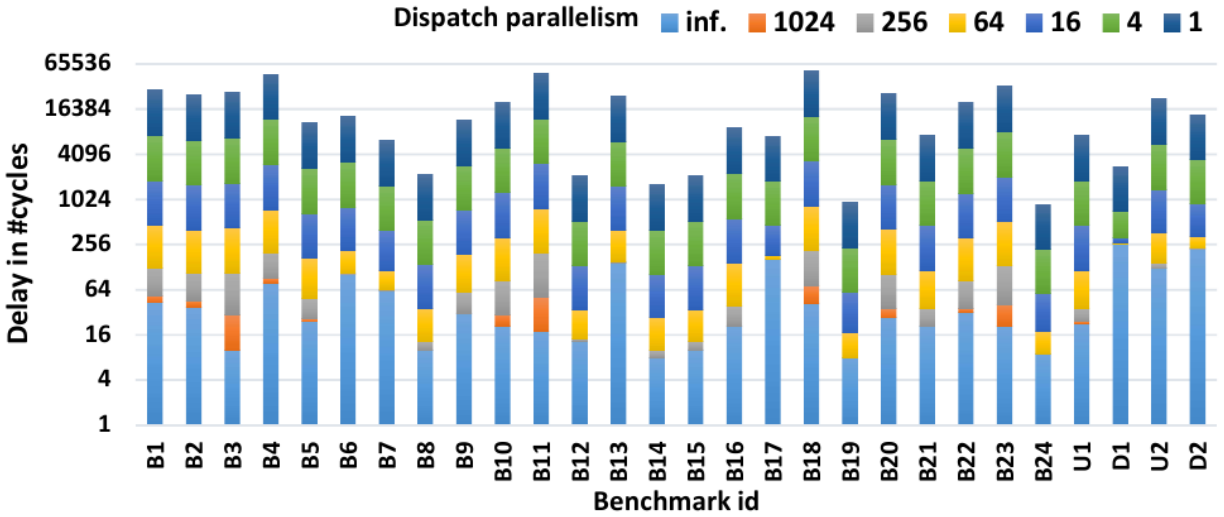
Experimental Results

	Benchmark	#L	#PI#PO	#N	#Nr	Inc.%	#D	#D_r	sav.%	#Ins.	#C
B_1	MAC32	42	96/65	9391	14347	52.77	10631	3488	67.19	29214	43
B_2	MUL32	37	64/64	9160	12981	41.71	8929	2968	66.76	25365	38
B_3	ac97_ctrl	9	2255/ 2250	12995	15932	22.60	11516	9080	21.15	27127	10
B_4	comp	78	279/193	18686	25710	37.00	15233	5042	66.90	47938	79
B_5	des_area	23	368/72	4258	5930	39.27	3825	1913	49.99	10686	24
B_6	div16	103	32/32	4406	6798	54.29	3776	440	88.34	12909	104
B_7	hamming	62	200/7	2078	3089	48.65	2117	569	73.12	6178	63
B_8	i2c	9	147/142	1113	1323	18.87	850	661	22.24	2200	10
B_9	max	30	512/130	4340	6604	52.17	3851	1422	63.07	11670	31
B_{10}	mem_ctrl	20	1198/1225	8368	10588	26.53	7148	2959	58.60	20141	21
B_{11}	pci_bridge32	17	3519/3528	22131	27368	23.66	17242	76.48	55.64	49307	18
B_{12}	pci_spoci_ctrl	12	85/76	1008	1280	26.98	757	477	36.99	2127	13
B_{13}	revx	144	20/25	7541	11434	51.62	7697	567	92.63	24388	145
B_{14}	sasc	7	133/132	753	941	24.97	669	501	25.11	1605	8
B_{15}	simple_spi	9	148/147	984	1220	23.98	813	531	34.69	2109	10
B_{16}	spi	20	274/276	3613	4865	34.65	3242	1261	61.10	9188	21
B_{17}	sqrt32	165	32/16	2172	3627	66.99	2163	231	89.32	7147	166
B_{18}	square	41	64/127	18014	27311	51.61	19514	7332	62.43	52767	42
B_{19}	ss_pcm	7	106/98	495	553	11.71	368	305	17.11	926	8
B_{20}	systemcaes	26	930/819	10366	13244	27.76	9100	2983	67.22	25970	27
B_{21}	systemcdes	20	314/258	2711	3652	34.71	2557	917	64.13	7321	21
B_{22}	tv80aig	31	373/404	7801	10287	31.86	6702	2181	67.46	19853	32
B_{23}	usb_funct	20	1860/1846	14841	17689	19.19	11763	6207	47.23	32819	21
B_{24}	usb_phy	8	113/111	483	537	11.18	337	289	14.24	883	9
U_1	adder	256	256/129	1405	1406	0.07	1020	389	61.86	2801	257
D_1	adder.dep	22	256/129	2647	4440	67.73	2387	1639	31.34	7319	23
U_2	sin	225	24/25	5459	6124	12.18	4259	712	83.28	13906	226
D_2	sin.dep	122	24/25	7558	10077	33.32	6895	1480	78.53	22363	123

NANYANG TECHNOLOGICAL UNIVERSITY

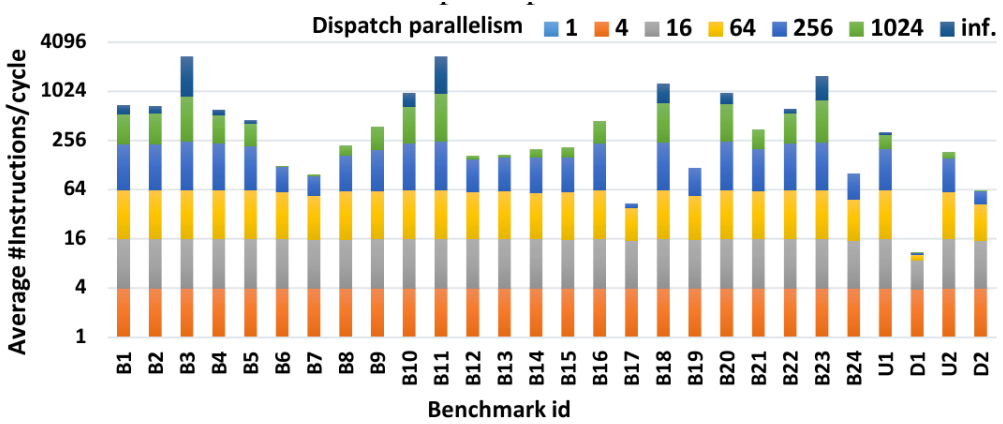
Experimental Results

Dispatch Parallelism vs Delay in #cycles



Experimental Results

Dispatch Parallelism vs Avg. #Ins/cycle



NANYANG TECHNOLOGICAL UNIVERSITY

Outline

- ReRAM fundamentals
- Problem statement
- Logic Representation and Technology mapping
- Delay Optimal Technology Mapping
- Device Reduction
- Dispatch parallelism constrained scheduling
- Experimental Results
- Conclusion

Conclusion

- We presented a delay-optimal technology mapping algorithm for mapping MIG to ReRAMs
- Further, we proposed an effective device reuse algorithm
  - Reduction in number of devices used by up to 92.63%, compared to the basic solution.
- Heuristic for dispatching d_p -instructions in parallel was proposed
  - Enables controls over the complexity of the controller
  - A smaller d_p makes the controller simpler, but potentially causing the MIG computation to be slower.