Majority Logic Circuits Optimization by Node Merging

Chun-Che Chung, Yung-Chih Chen, Chun-Yao Wang, Chia-Cheng Wu
National Tsing Hua University, Yuan Ze University
Taiwan

Outline

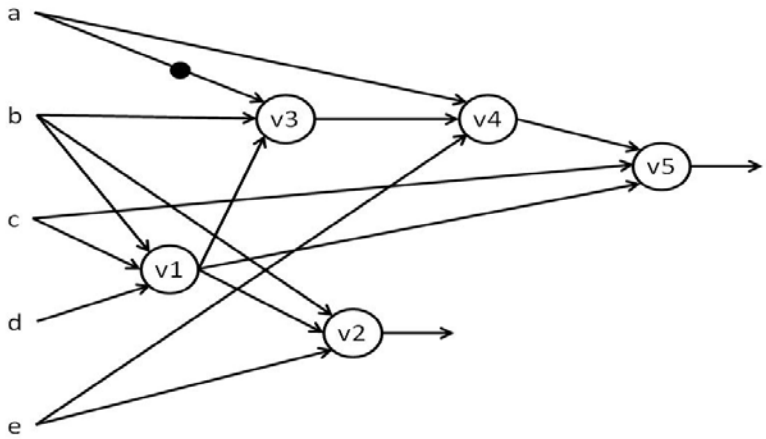
- Introduction
  - Majority function
  - Quantum-dot Cellular Automata
- Preliminaries
- Problem formulation
- Node merging approach
  - Noncontrolling pair
  - MA computation
- Experimental results
- Conclusions

Outline

- **Introduction**
  - Majority function
  - Quantum-dot Cellular Automata
- Preliminaries
- Problem formulation
- Node merging approach
  - Noncontrolling pair
  - MA computation
- Experimental results
- Conclusions

Majority function

- A majority function is an odd-input function that has the output value of *v* if and only if more than half of the inputs are assigned the value of *v*
- Majority-Inverter-Graph (MIG) is a directed, acyclic graph that represents a logic network with three-input majority gates and inverters [1]

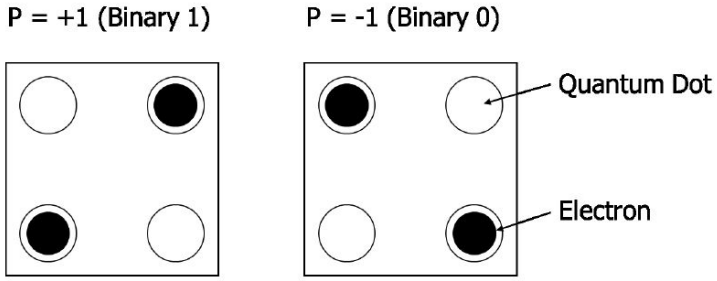


Node: Majority gate, Dot: Inverter

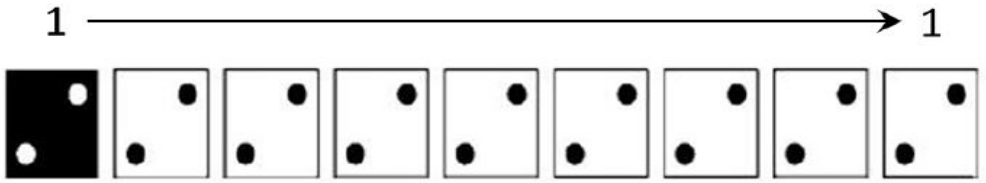
[1] L. Amarú, P.-E Gaillardon, and G. De. Micheli, "Majority-Inverter Graph: A Novel Data-Structure and Algorithm for Efficient Logic Optimization," *in Proc. DAC*, 2014

Quantum-dot Cellular Automata

- Quantum-dot Cellular Automata (QCA) is the lower-power nanotechnology that is considered as a replacement candidate for CMOS



(a) Binary information of a QCA cell.

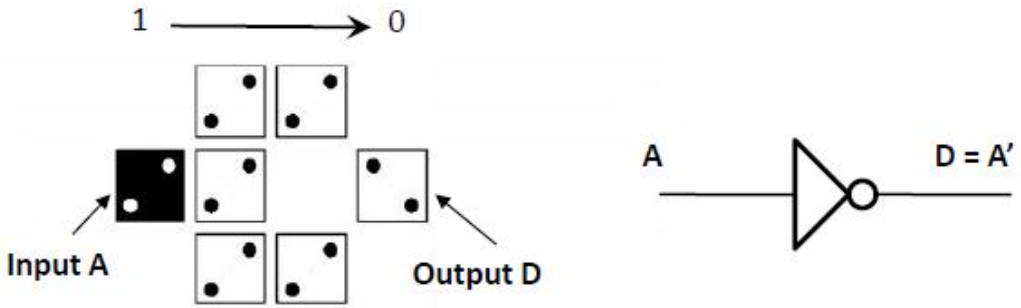


(b) QCA wire

Quantum-dot Cellular Automata

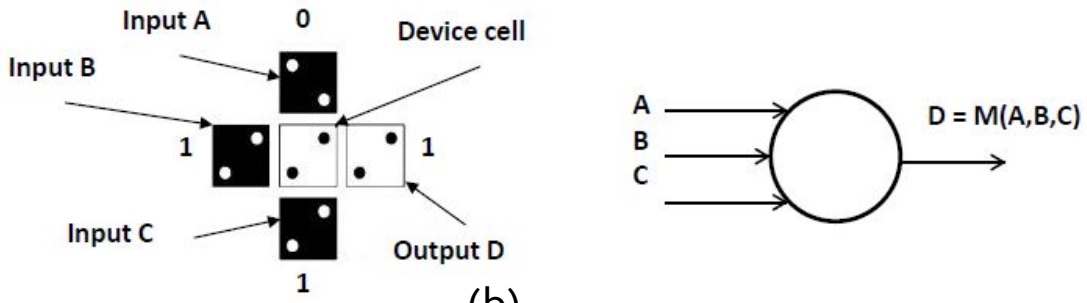
- The underlying QCA logic devices include QCA inverter, and QCA majority gate.

- QCA inverter



(a)

- QCA majority gate:



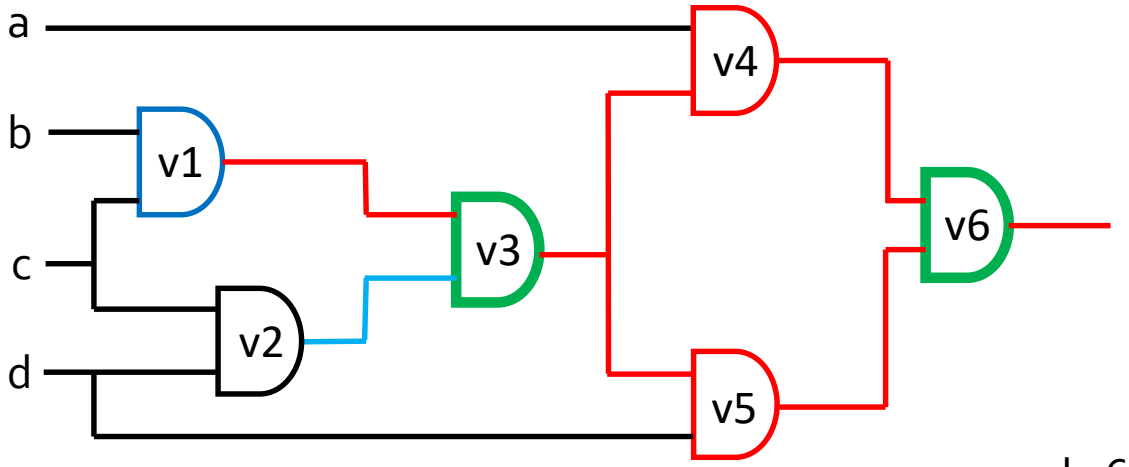
(b)

Outline

- Introduction
  - Majority function
  - Quantum-dot Cellular Automata
- **Preliminaries**
- Problem formulation
- Node merging approach
  - Noncontrolling pair
  - MA computation
- Experimental results
- Conclusions

Preliminaries

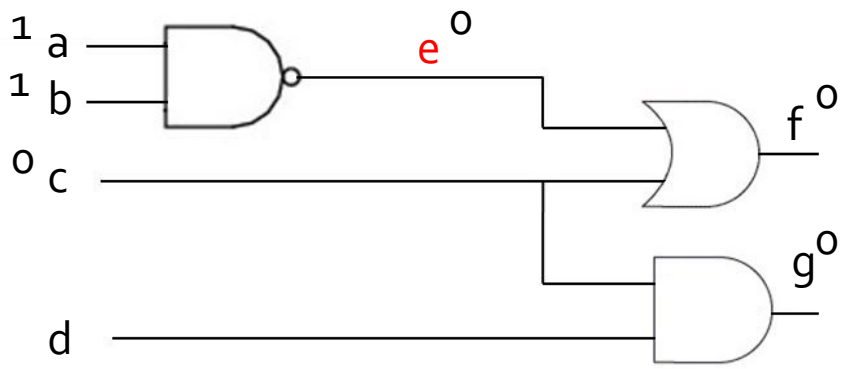
- The dominators of a gate g are a set of gates G such that all paths from g to any PO have to pass through all gates in G



v3 and v6 are dominators of v1

Preliminaries

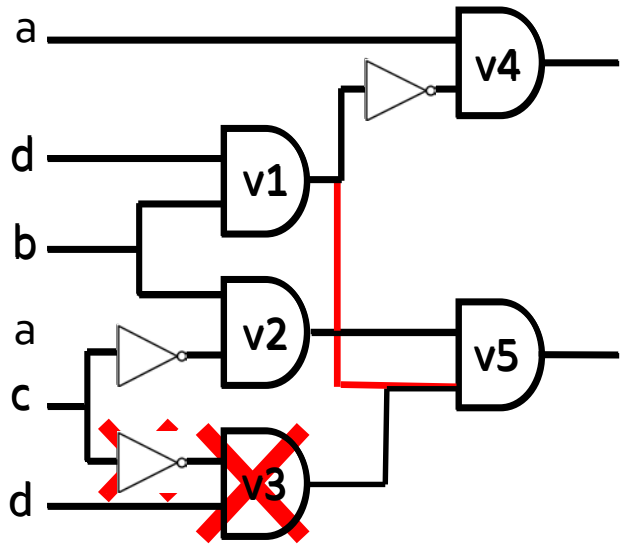
- The mandatory assignments (MAs) of a fault test are the unique value assignments to nodes necessary for the test to exist



For stuck-at 1 fault at e:
Mandatory assignments (MAs) =
{ a=1, b=1, c=0, e=0, f=0, g=0 }

Preliminaries

- Node Merging on And-Inverter-Graph [2]



v1 and v3 only differ when d=1 and b=c. However, b=c Implies v2=0

Because v2=0, the value of v3 cannot be observed at v5

Replacing v3 with v1 does not change the overall functionality

[2] Yung-Chih Chen and Chun-Yao Wang, "Fast detection of node mergers using logic implications," in *Proc. ICCAD*, 2009.

Preliminaries

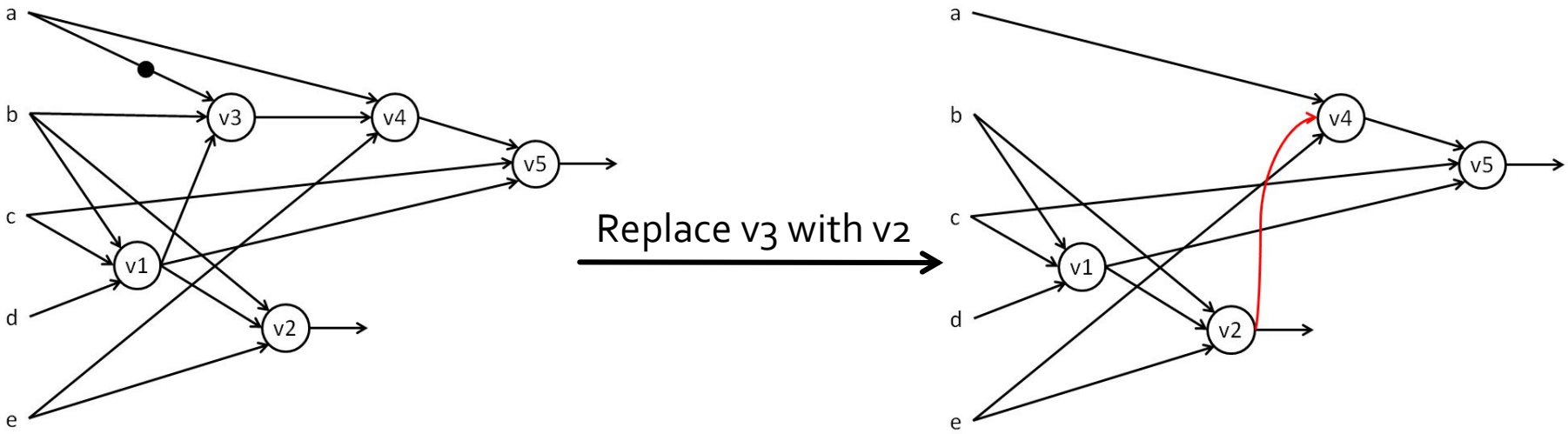
- A sufficient condition to identify n_s for n_t:
  - Condition: Let f denote an error of replacing n_t with n_s. If n_s=1 and n_s=0 are MAs for stuck-at 0 and stuck-at 1 fault tests on n_t, respectively, f is undetectable

Outline

- Introduction
  - Majority function
  - Quantum-dot Cellular Automata
- Preliminaries
- **Problem formulation**
- Node merging approach
  - Noncontrolling pair
  - MA computation
- Experimental results
- Conclusions

Problem formulation

- Given: a Majority-Inverter Graph
- Objective: a simplified Majority-Inverter Graph
- Cost function: the number of Majority gates

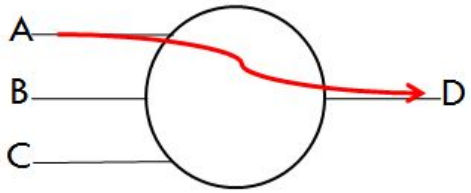


Outline

- Introduction
  - Majority function
  - Quantum-dot Cellular Automata
- Preliminaries
- Problem formulation
- **Node merging approach**
  - Noncontrolling pair
  - MA computation
- Experimental results
- Conclusions

Noncontrolling pair

- Unlike two-input AND/OR gates, a three-input majority gate has two side-inputs in the fault propagation path, named as a side-input pair
- Does not have noncontrolling value for fault propagation
- The input value can be propagated to the output if and only if the side-input pair are assigned to different values, and these different values are named as a noncontrolling pair



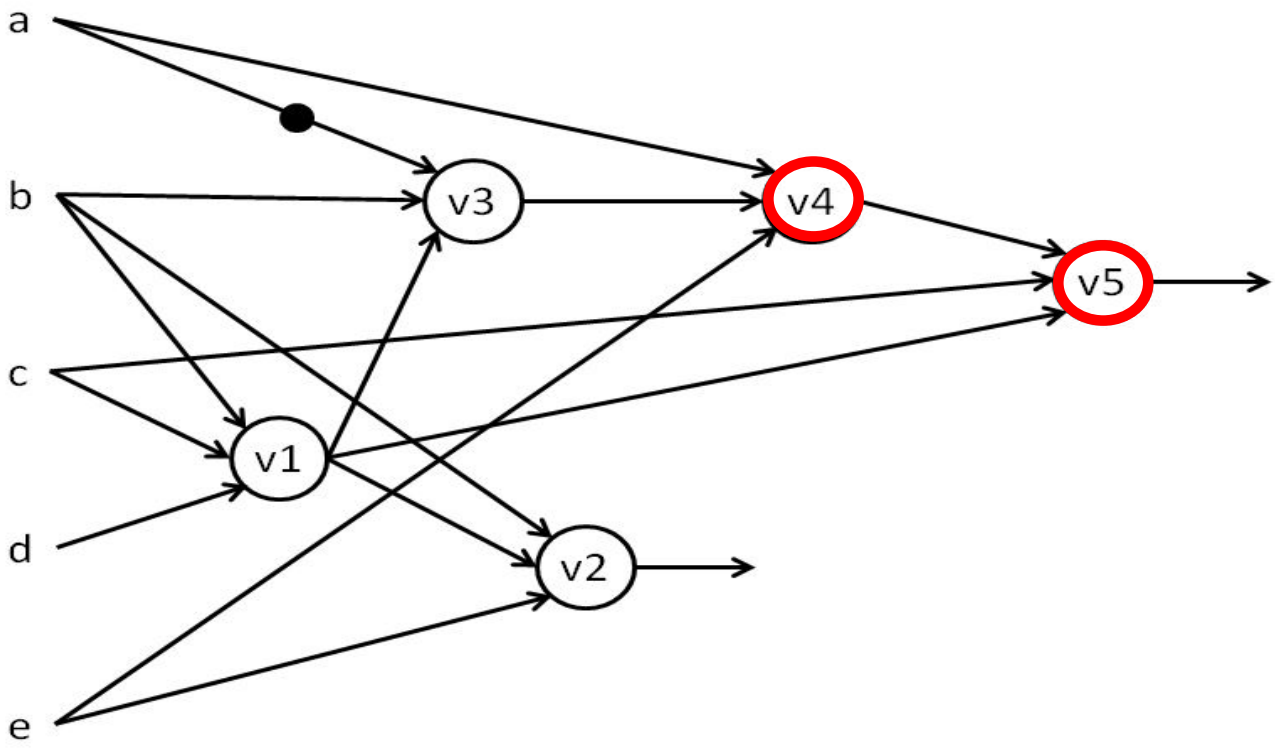
For propagating the value of the input A to the output D, side-input pair (B, C) have to be assigned (0, 1) or (1, 0)

MA computation

- In the MIG, we use the side-input pair and noncontrolling pair to propagate the fault-effect to any POs
- Since the noncontrolling pairs have two value assignments to the side-input pair, the resultant MA set are the intersection of the sets of value assignments which are consistent
- Dominator-based MA computation
- The fanouts of a target node can be either single or multiple, the processes of finding MAs are different

MA computation (Single-fanout)

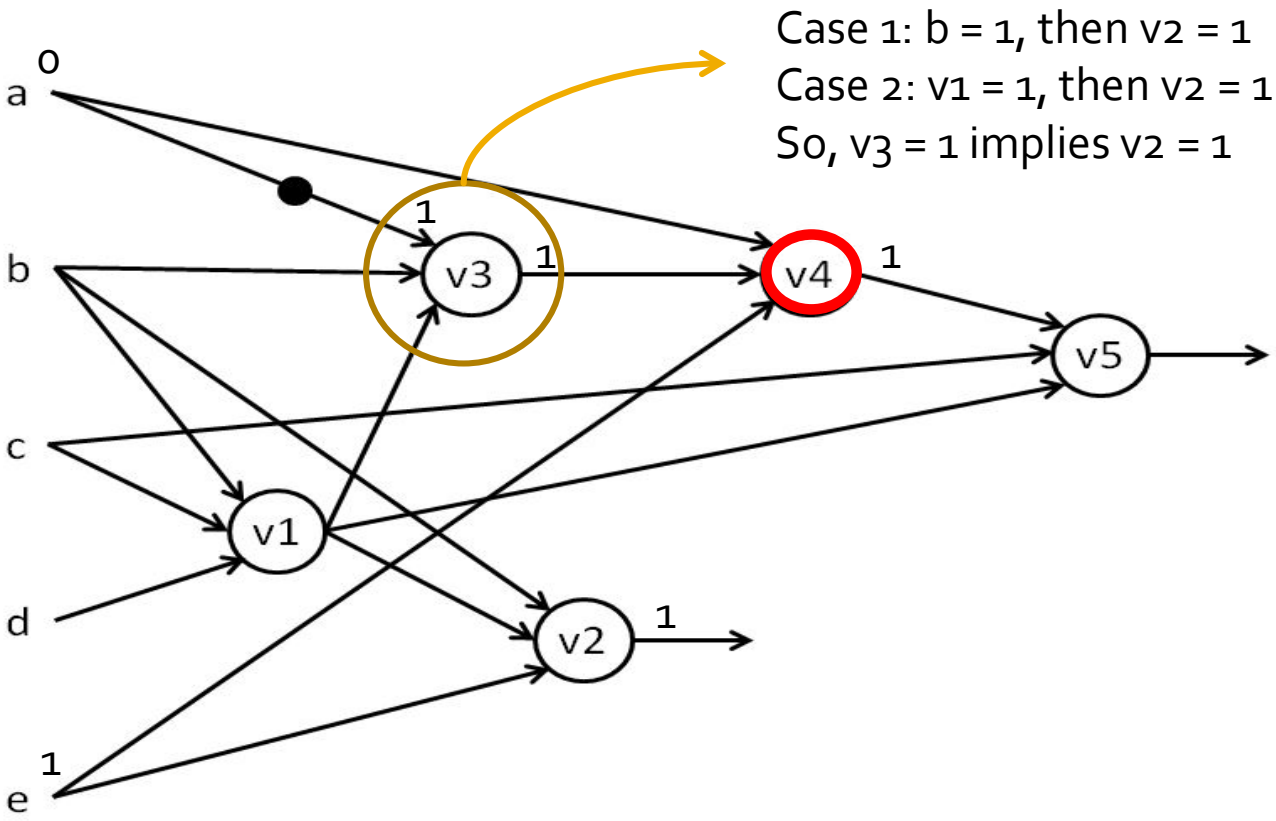
Target node: v3
Dominators: v4, v5
Side-input pairs: (a, e), (c, v1)



MA computation (Single-fanout)

Stuck-at 0 fault on v3

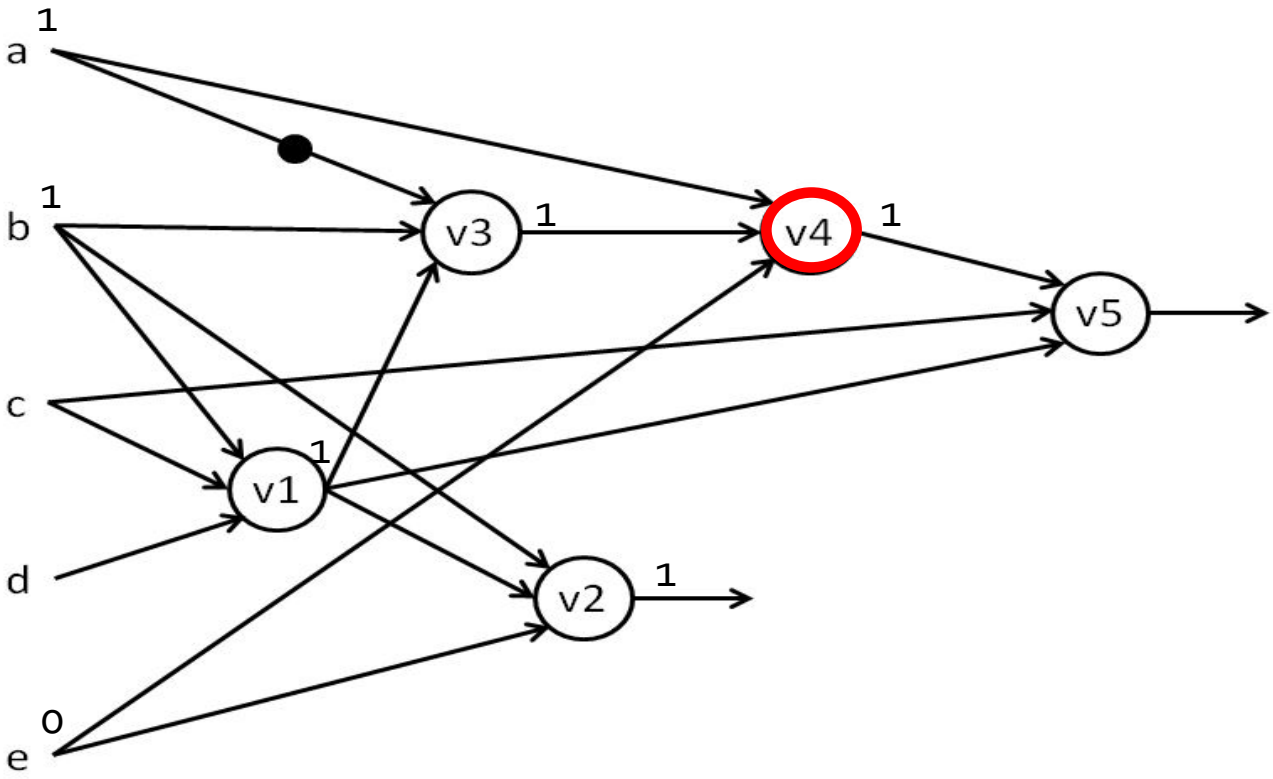
(a, e) = (0, 1) assignments1 = { v3=1, v4=1, a=0, e=1, v2=1 }



MA computation (Single-fanout)

Stuck-at 0 fault on v3

(a, e) = (0, 1) assignments1 = { v3=1, v4=1, a=0, e=1, v2=1 }



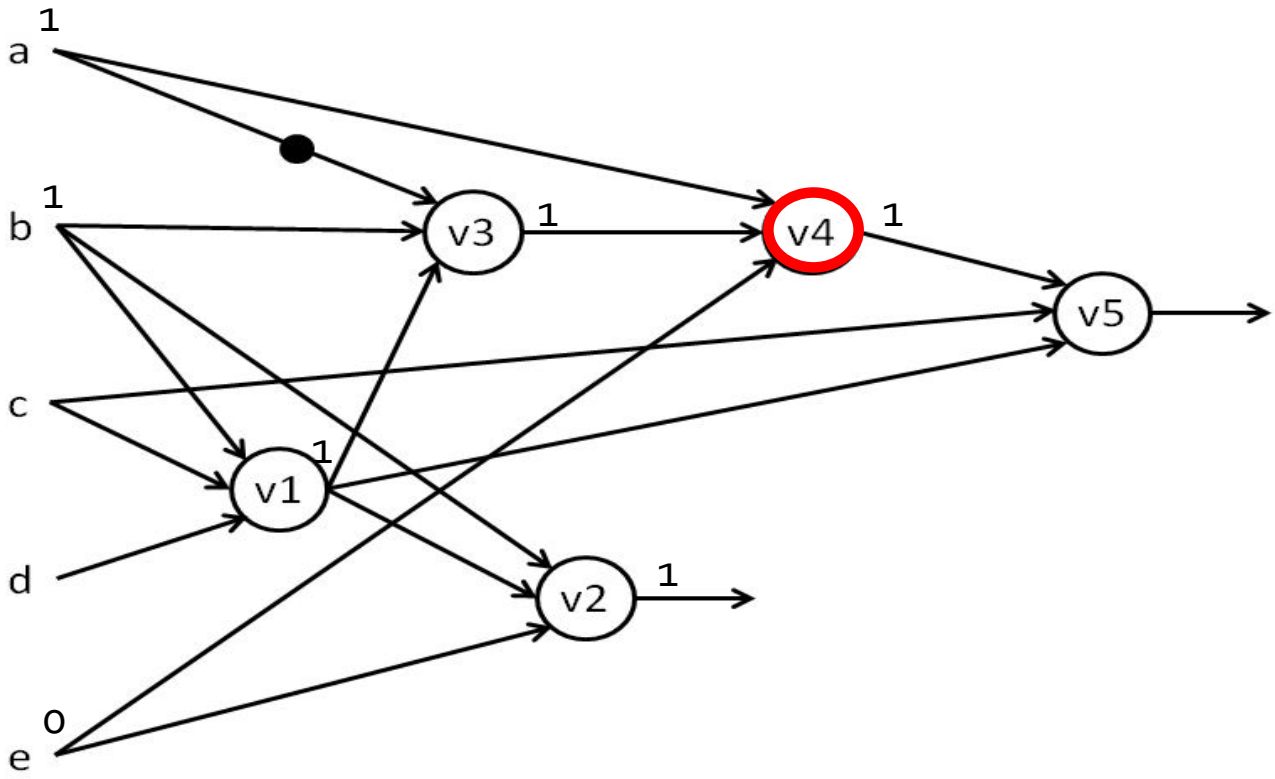
MA computation (Single-fanout)

Stuck-at 0 fault on v3

(a, e) = (0, 1) assignments1 = { v3=1, v4=1, a=0, e=1, v2=1 }

(a, e) = (1, 0) assignments2 = { v3=1, v4=1, a=1, e=0, v2=1, b=1, v1=1 }

MAs = { v3=1, v4=1, v2=1 }

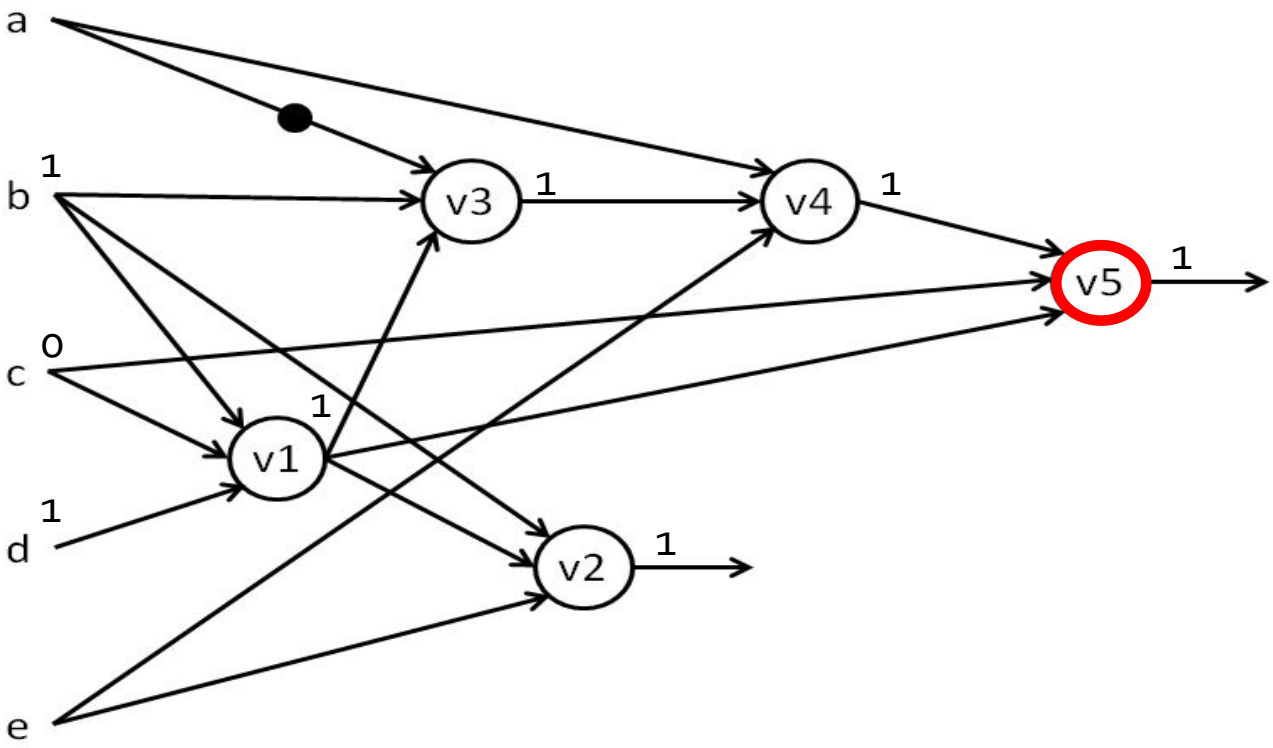


MA computation (Single-fanout)

Stuck-at 0 fault on v3

Previous MAs = { v3=1, v4=1, v2=1 }

(c, v1) = (0, 1) assignments1 = { v3=1, v4=1, v5=1, v2=1, c=0, v1=1, b=1, d=1 }



MA computation (Single-fanout)

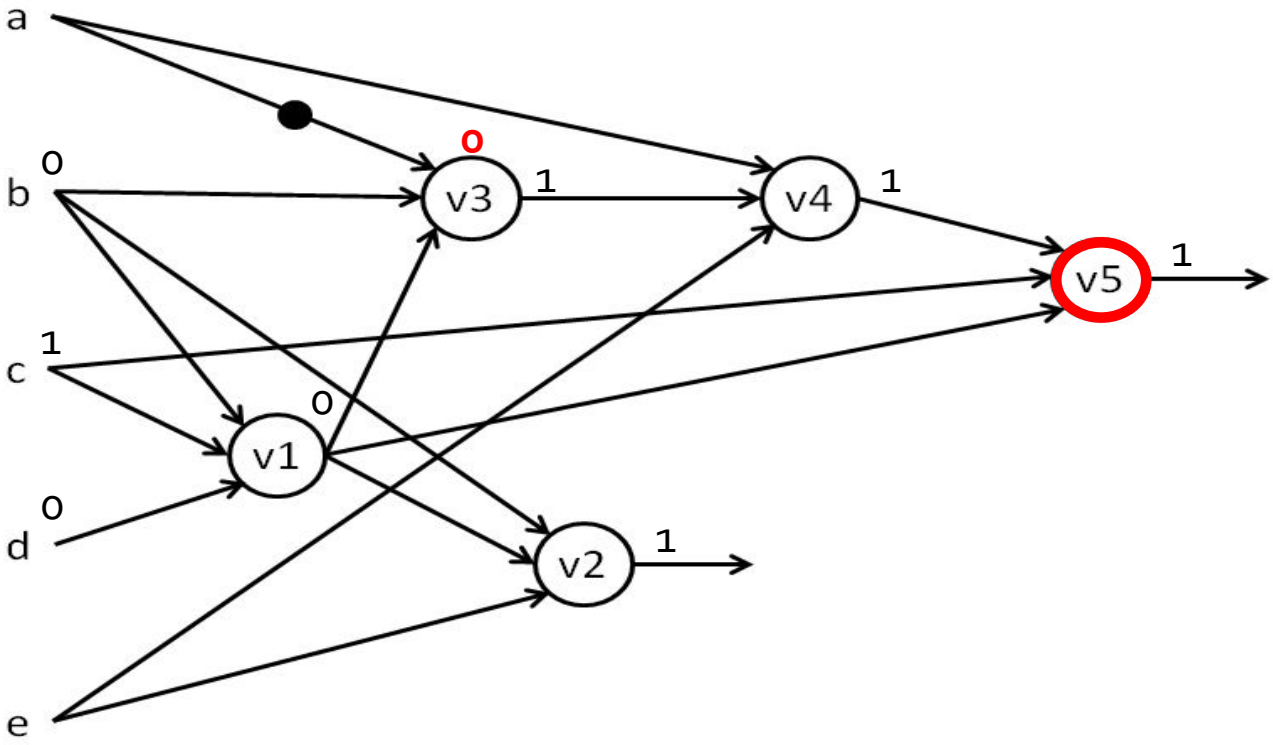
Stuck-at 0 fault on v3

Previous MAs = { v3=1, v4=1, v2=1 }

(c, v1) = (0, 1) assignments1 = { v3=1, v4=1, v5=1, v2=1, c=0, v1=1, b=1, d=1 }

(c, v1) = (1, 0) assignments2 = { **v3=1**, v4=1, v5=1, v2=1, c=1, v1=0, b=0, d=0, **v3=0** } (inconsistent)

Resultant MAs = { v3=1, v4=1, v5=1, v2=1, c=0, v1=1, b=1, d=1 }

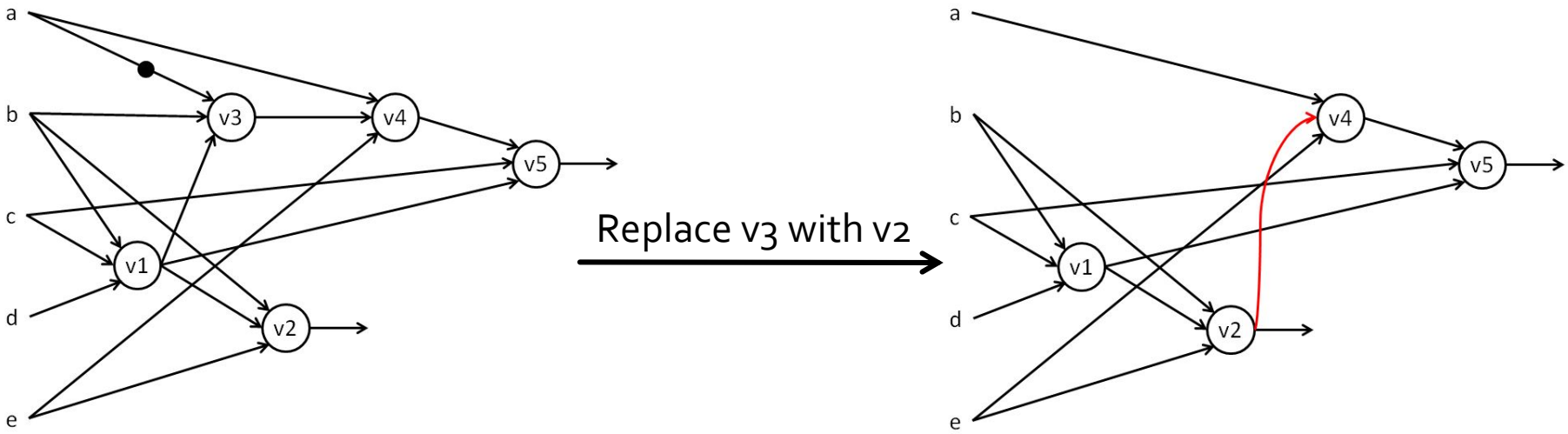


MA computation (Single-fanout)

MAs(v3=sa0) = { v3=1, v4=1, v5=1, v2=1, c=0, v1=1, b=1, d=1 }

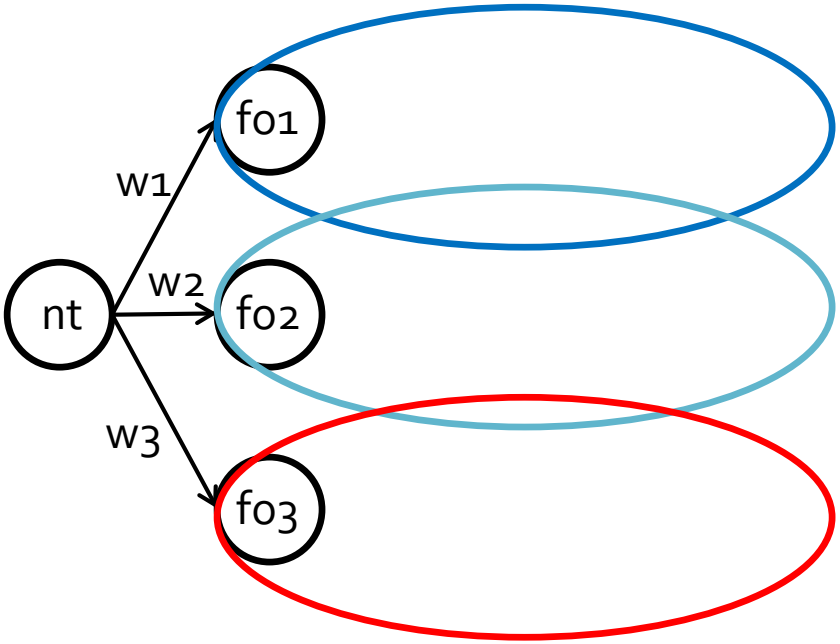
MAs(v3=sa1) = { v3=0, v4=0, v5=0, v2=0, c=1, v1=0, b=0, d=0 }

Substitute nodes: v2, c, v1, b, d



MA computation (Multiple-fanout)

- Perform the stuck-at fault test on each fanout wire of n_t
- The MA computation of each fanout wire is similar to single-fanout method
- MAs(n_t=*sav*) is the intersection of all consistent MAs(w_i=*sav*)



Resultant MAs =
MAs(w_1=*sav*) ∩ MAs(w_2=*sav*) ∩ MAs(w_3=*sav*)

Outline

- Introduction
  - Majority function
  - Quantum-dot Cellular Automata
- Preliminaries
- Problem formulation
- Node merging approach
  - Noncontrolling pair
  - MA computation
- **Experimental results**
- Conclusions

Experimental Environment & Benchmarks

- The approach was implemented in C++ language
- The experiments were conducted on an Intel $Xeon^{®}$ X5570 2.93GHz CentOS 5.11 platform with 48 GBytes memory
- Benchmarks were from *http://lsi.epfl.ch/MIG*
- Two experiments were performed in this paper:
  - Perform on well–optimised benchmarks
  - Perform on the original benchmarks with the MIG online synthesis system *MIGhty*

Experimental Results

				Depth Increase					Depth Preservation				
Benchmark	I/O	Size	Depth	Size	(%)	Depth	(%)	Time	Size	(%)	Depth	(%)	Time
usb_phy	113/111	484	8	469	3.10	8	0.00	0.51	470	2.89	8	0.00	0.51
ss_pcm	106/98	496	7	493	0.60	7	0.00	0.82	494	0.40	7	0.00	0.81
sasc	133/132	754	7	749	0.66	9	-28.57	1.15	751	0.40	7	0.00	1.14
simple_spi	148/147	985	9	962	2.34	11	-22.22	2.56	976	0.91	9	0.00	2.54
pci_spoci_ctrl	85/76	1009	12	863	14.47	15	-25.00	7.50	872	13.58	12	0.00	7.67
i2c	147/142	1114	9	1086	2.51	12	-33.33	2.68	1093	1.89	9	0.00	2.70
hamming	200/7	2709	62	1933	7.02	65	-4.83	11.88	1952	6.11	62	0.00	12.10
sqrt32	32/16	2173	165	2072	4.65	187	-13.33	42.46	2088	3.91	165	0.00	41.71
systemcdes	314/258	2712	20	2577	4.98	25	-25.00	34.40	2630	3.02	20	0.00	34.42
spi	274/276	3614	20	3361	7.00	20	0.00	75.81	3363	6.95	30	0.00	76.53
des_area	368/72	4259	23	4006	5.94	24	-4.34	136.53	4008	5.89	23	0.00	134.89
max	512/130	4341	30	4300	0.94	30	0.00	204.56	4300	0.94	30	0.00	204.80
div16	32/32	4407	103	3966	10.00	129	-25.24	141.08	4060	7.87	103	0.00	145.52
revx	20/25	7542	144	6989	7.33	174	-20.83	238.48	7180	4.80	144	0.00	252.21
tv80	373/404	7802	31	7553	3.19	38	-22.58	305.69	7592	2.69	31	0.00	306.78
mem_ctrl	1198/1225	8369	20	8256	1.35	30	-50.00	132.98	8278	1.09	20	0.00	134.49
MUL32	64/64	9161	37	8497	7.25	44	-18.97	117.07	8703	5.00	37	0.00	120.60
MAC32	96/65	9392	42	9144	2.64	58	-38.09	79.10	9182	2.24	42	0.00	78.95
systemcaes	930/819	10367	26	10229	1.33	29	-11.53	284.82	10272	0.93	26	0.00	282.89
ac97_ctrl	2255/2250	12996	9	12834	1.25	9	0.00	97.68	12979	0.13	9	0.00	100.35
usb_funct	1860/1846	14842	20	14636	1.39	25	-25.00	258.98	14704	0.93	20	0.00	259.17
square	64/127	18015	41	17562	2.51	81	-97.56	259.66	17937	0.43	41	0.00	263.63
diffeq1	354/289	18015	220	17162	4.73	244	-10.90	436.08	17358	3.65	220	0.00	446.67
comp	279/193	18687	78	18285	2.15	106	-26.31	1841.55	18388	1.60	78	0.00	1883.61
aes_core	789/668	21616	19	20402	5.62	24	-26.31	831.81	20555	4.91	19	0.00	841.73
pci_bridge32	3519/3528	22132	17	21955	0.80	21	-23.52	501.10	22022	0.50	17	0.00	503.10
mult64	128/128	25901	110	25403	1.92	113	-2.72	637.64	25410	1.90	110	0.00	636.07
log2	32/32	31359	202	30659	2.23	216	-6.93	5001.42	30715	2.05	202	0.00	5000.27
DSP	4223/3953	44166	35	43444	1.63	45	-28.57	1723.69	43614	1.25	35	0.00	1730.58
Average					3.85		-20.73	462.40		3.06		0.00	465.74

Experimental Results

				Optimization			
Benchmark	I/O	Size	Depth	Size	(%)	Depth	(%)
ss_pcm	106/98	413	7	413	0.00	7	0.00
usb_phy	113/111	498	10	464	6.83	9	10.00
sasc	133/132	782	9	686	12.28	8	11.11
simple_spi	148/147	1068	12	877	17.88	16	-33.33
sqrt32	32/16	1113	495	1109	0.36	493	0.40
i2c	147/142	1181	14	1006	14.82	11	21.43
pci_spoci_ctrl	85/76	1402	18	749	46.58	13	27.78
max	512/130	2865	287	2865	0.00	287	0.00
systemcdes	314/258	3131	27	2640	15.69	24	11.11
hamming	200/7	3612	75	1790	50.44	69	8.00
spi	274/276	3847	32	3330	13.44	23	28.13
des_area	368/72	4865	34	4538	6.72	26	23.53
div16	32/32	7175	136	2467	65.62	134	1.47
tv80	373/404	9691	52	7098	26.76	34	34.62
MUL32	64/64	11613	43	8182	29.54	47	-9.30
MAC32	96/65	12063	70	8993	25.45	64	8.57
systemcaes	930/819	12533	46	9098	27.41	30	34.78
ac97_ctrl	2255/2250	14382	12	12444	13.48	9	25.00
mem_ctrl	1198/1225	15604	36	8145	47.80	25	30.56
usb_funct	1860/1846	16048	27	14681	8.52	25	7.41
revx	20/25	16164	192	6486	59.88	171	10.94
square	64/127	18485	250	17897	3.18	54	78.40
aes_core	789/668	21658	26	19645	9.29	27	-3.85
pci_bridge32	3519/3528	23215	30	20019	13.77	22	26.67
mult64	128/128	27062	274	26858	0.75	112	59.12
log2	32/32	32060	444	30639	4.43	229	48.42
diffeq1	354/289	33632	303	17962	46.59	247	18.48
Total		296162	2961	231081		2216	
Improvement					21.98		25.16

Conclusions

- We propose a node merging algorithm targeting at gate count minimization for majority logic circuits
- Use logic implications to identify substitute nodes directly
- Our approach can effectively optimize the majority logic circuit, and the implementation cost of the corresponding QCA circuit is also reduced

Thank You

Q & A