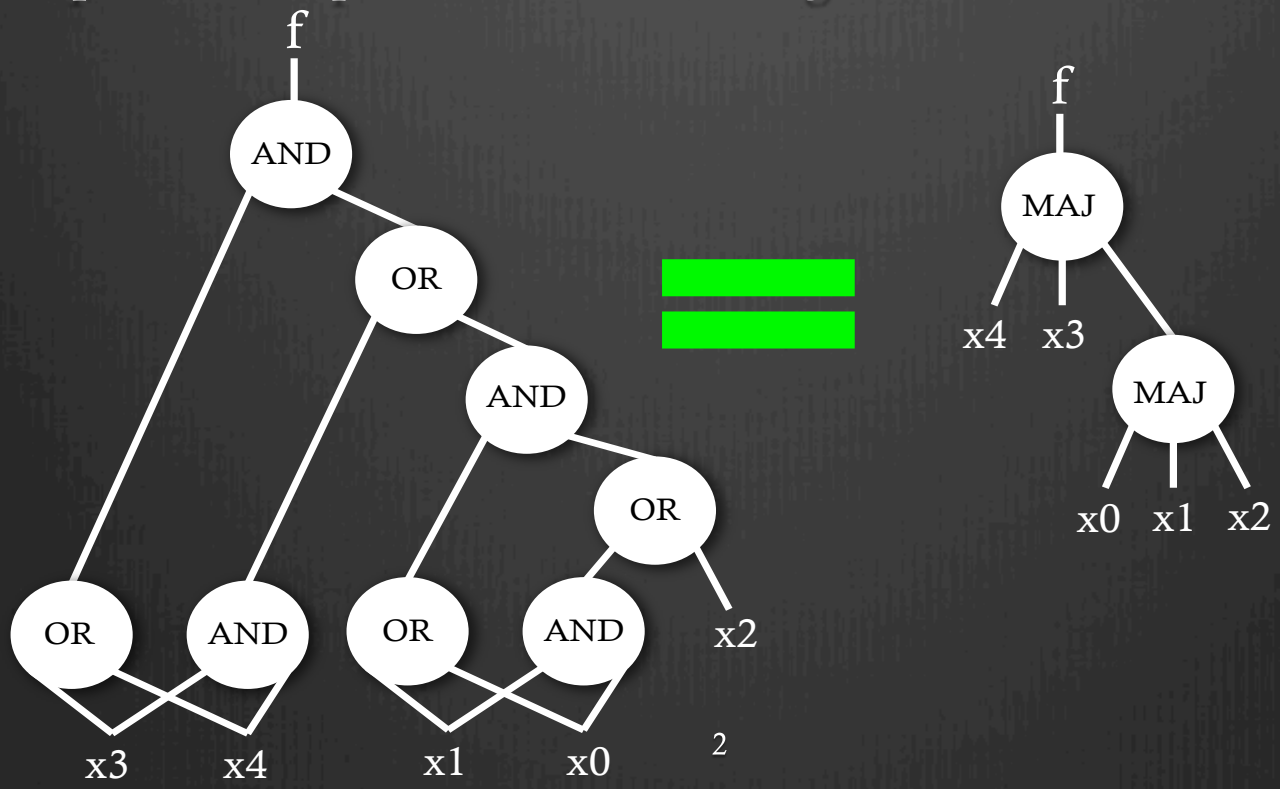
The Majority Logic Optimization Paradigm

Luca Amarù
Integrated Systems Laboratory (LSI), EPFL, Switzerland.

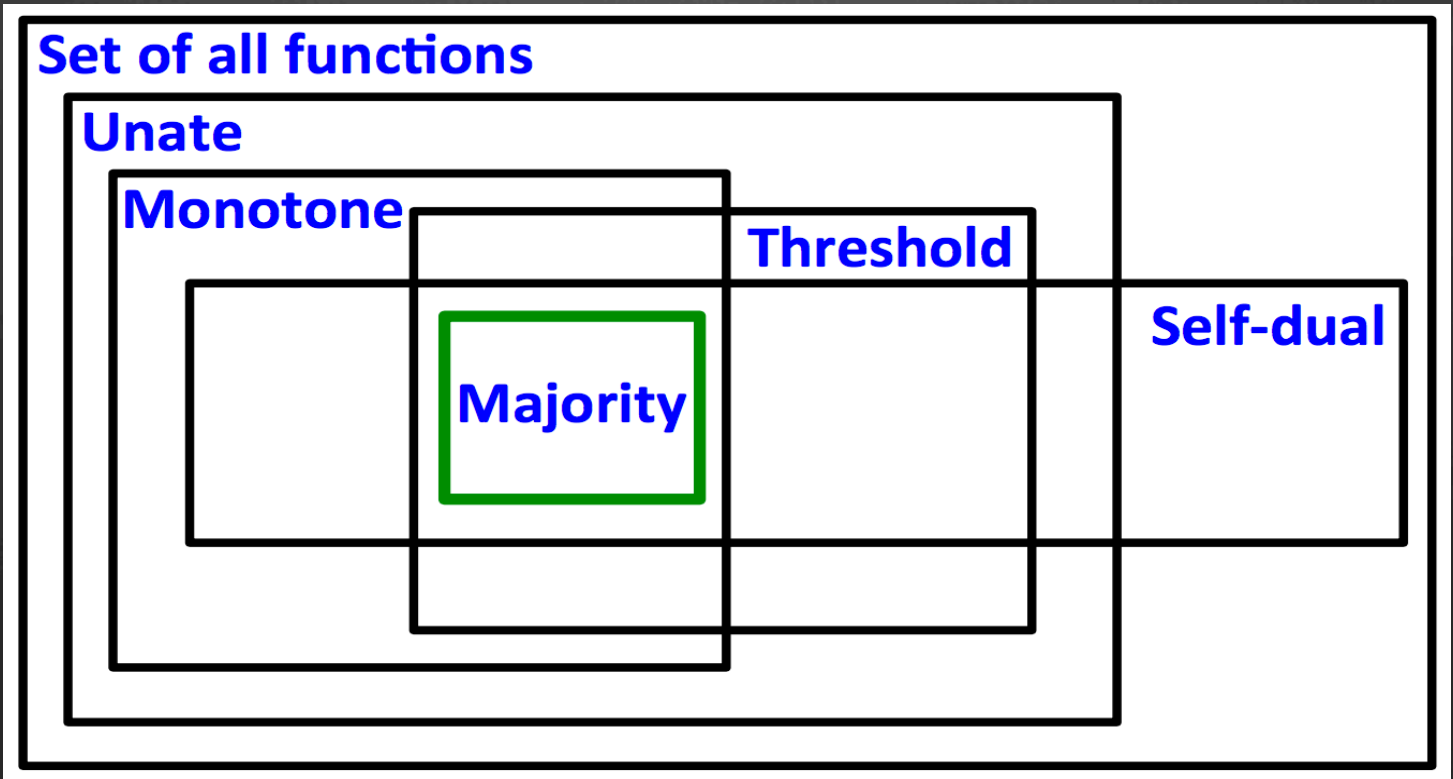
Why Majority Logic?

- Majority logic is a powerful generalization of AND/ORs.
- MAJ(x_1,x_2,x_3,…,x_n)=1 if more than *n/2* inputs are 1.
- MAJ(a,b,c)=ab+ac+bc. MAJ(a,b,1)=a+b. MAJ(a,b,0)=ab.
- More compact as compared to AND-OR logic:



How Powerful is Majority?

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



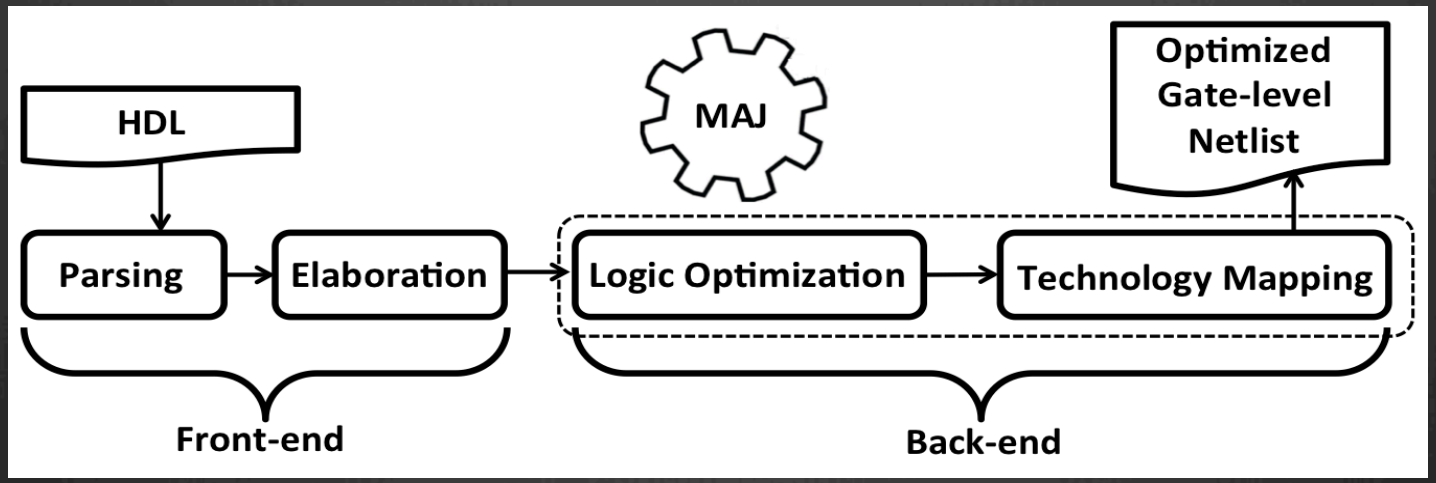
AA. Sherstov, Separating AC 0 from depth-2 majority circuits, Proc. STOC, 2007

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.

Exploiting Majority Logic

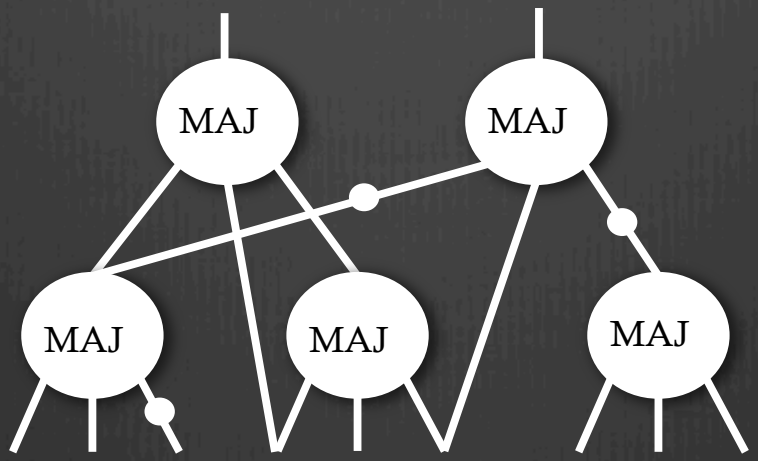
- There is an exponential gap between the expressive power of traditional AND/OR circuits and MAJ circuits when considering arithmetic.
- So, why not exploiting the majority logic representation expressiveness when synthesizing circuits?



- In order to manipulate majority logic we define a homogenous data structure.
- We call it Majority-Inverter Graph.

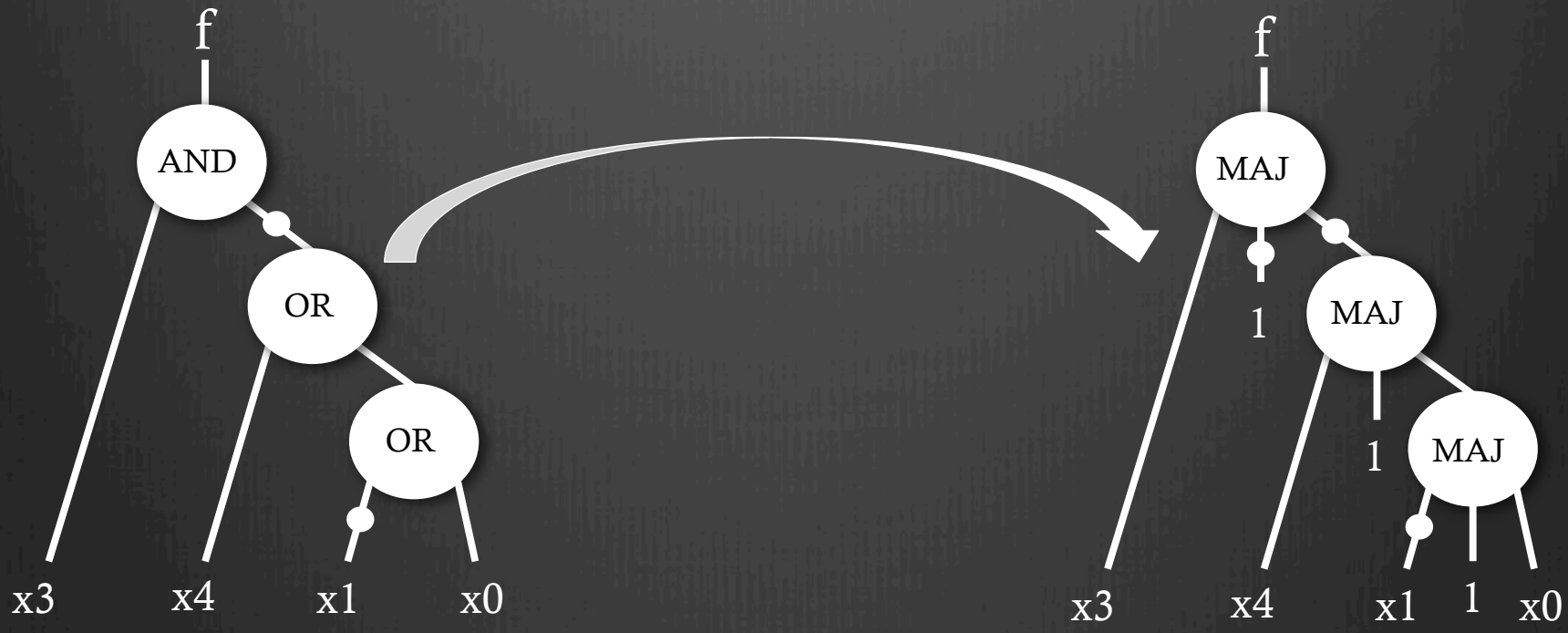
Majority-Inverter Graph

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



MIG Properties

AOIGs ➔ MIGs



MIGs **include** AOIGs **include** AIGs

Manipulating MIGs: MIG Boolean Algebra

Ω

1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$

2- Majority: $if(x = y), M(x, y, z) = x = y$

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

3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$

4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

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

Theorem: (B,M,',0,1) subject to axiom in Ω is a Boolean algebra

MIG Boolean Algebra

Ω

1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$

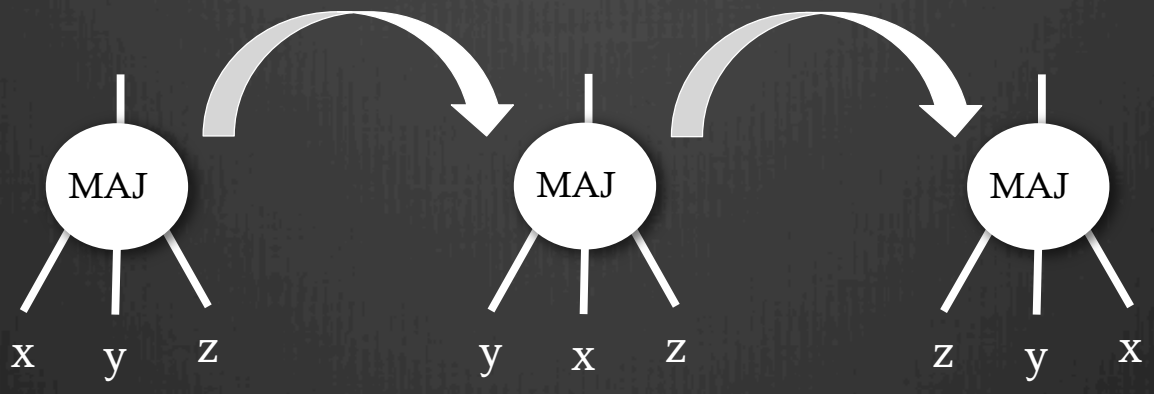
2- Majority: $if(x = y), M(x, y, z) = x = y$

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

3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$

4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

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



MIG Boolean Algebra

Ω

1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$

2- Majority: $if(x = y),\ M(x, y, z) = x = y$

$if(x = y'),\ M(x, y, z) = z$

3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$

4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

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

MIG Boolean Algebra

Ω

1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$

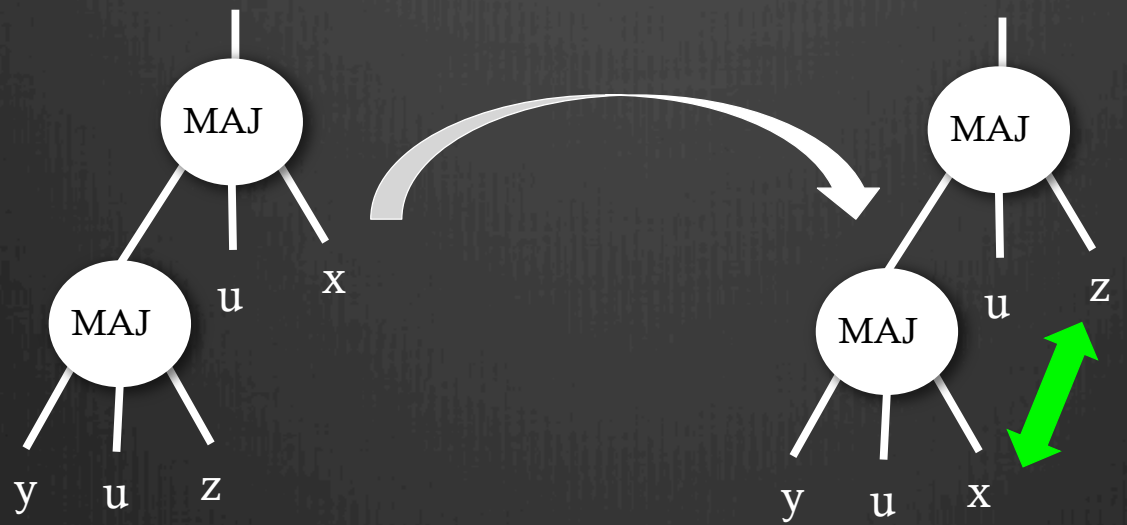
2- Majority: $if(x = y),\ M(x, y, z) = x = y$

$if(x = y'),\ M(x, y, z) = z$

3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$

4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

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



MIG Boolean Algebra

Ω

1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$

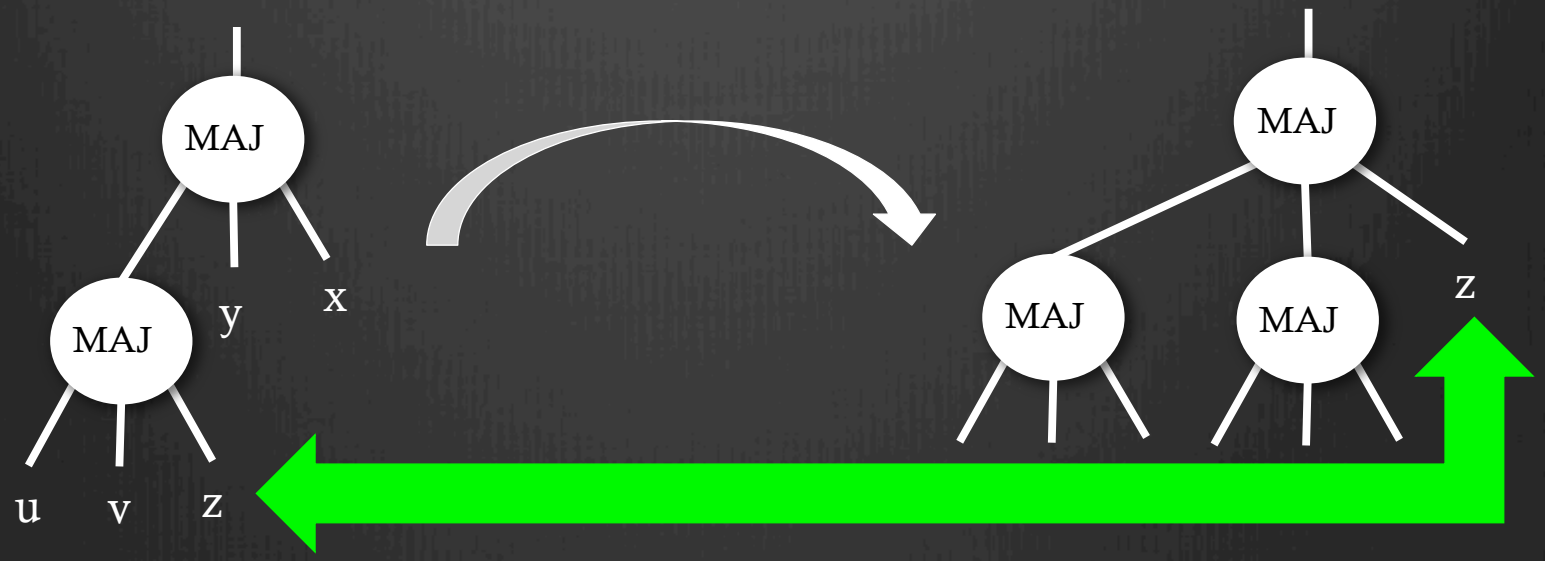
2- Majority: $if(x = y), M(x, y, z) = x = y$

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

3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$

4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

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



MIG Boolean Algebra

Ω

1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$

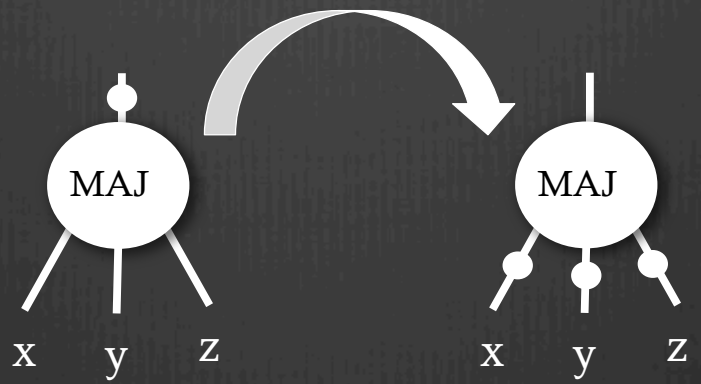
2- Majority: $if(x = y), M(x, y, z) = x = y$

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

3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$

4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

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



Optimizing MIGs

Ω
- 1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$
- 2- Majority: $if(x = y), M(x, y, z) = x = y$
  $if(x = y'), M(x, y, z) = z$
- 3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$
- 4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$
- 5- Inverter Propagation: $M'(x, y, z) = M(x', y', z')$

- Ω is the basis for more elaborated optimization transformations.
- For instance, it is possible to extend associativity:
  - Complementary Associativity:
  - $M(x, u, M(y, u', z)) = M(x, u, M(y, x, z))$

Theorem: MIG Boolean algebra is sound and complete

Optimizing MIGs

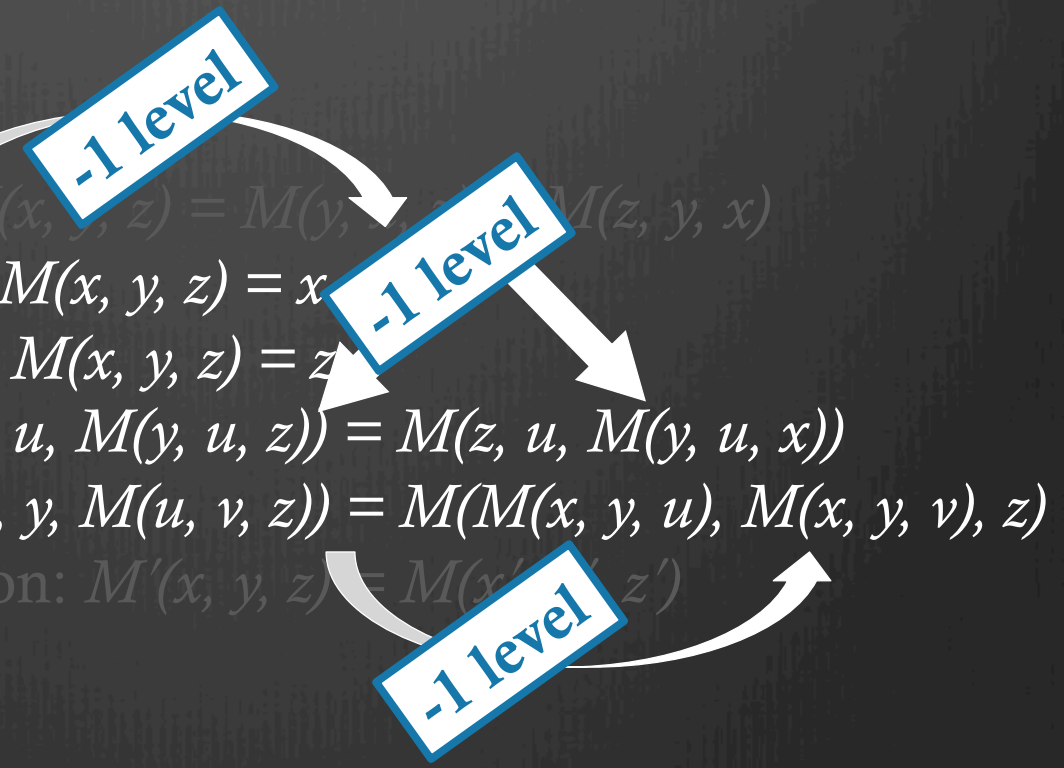
Ω
- 1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$
- 2- Majority: $if(x = y), M(x, y, z) = x = y$
  $if(x = y'), M(x, y, z) = z$
- 3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$
- 4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$
- 5- Inverter Propagation: $M'(x, y, z) = M(x', y', z')$

- By using Ω transformations we want to optimize an MIG
- What do we care about?
- **Area** → MIG size (details in TCAD'15)
- **Delay** → MIG depth – discussed in this presentation
- **Power** → MIG SW Activity (details in TCAD'15)

MIG Depth Optimization

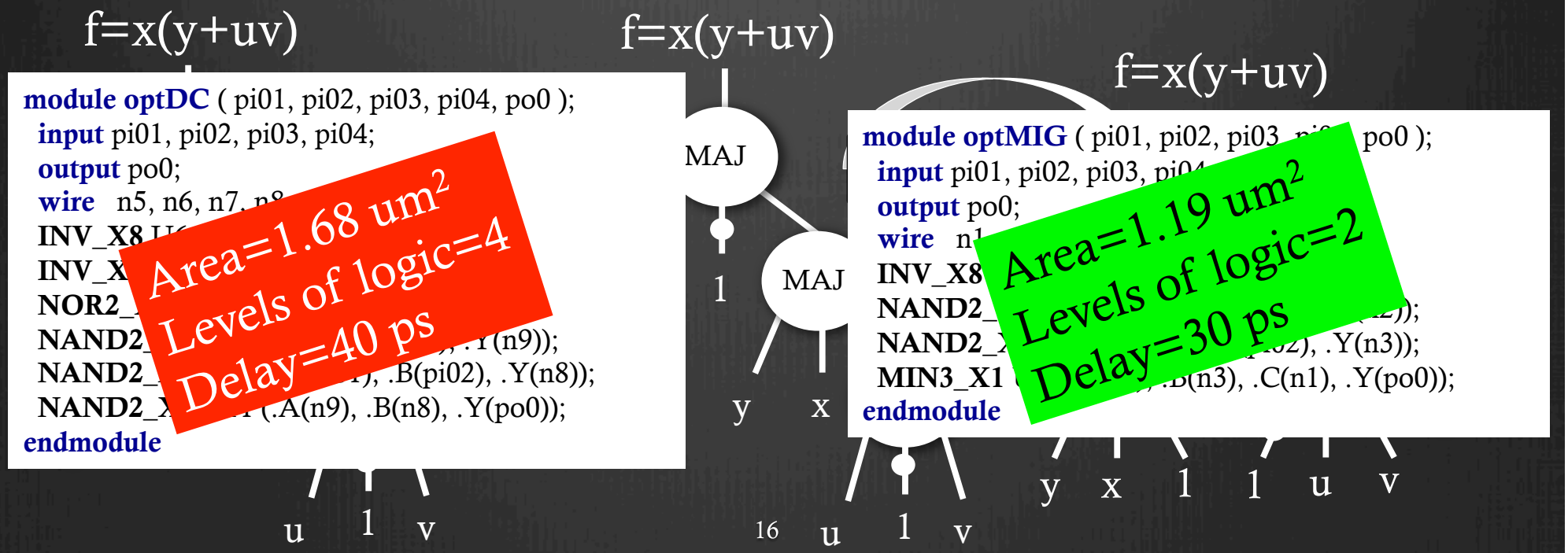
- How to reduce the depth of an MIG?
- Let's see what comes handy from Ω:

Ω
1- Commutativity: $M(x, y, z) = M(y, \ldots, z) = M(z, y, x)$
2- Majority: $if(x = y),\ M(x, y, z) = x$
$if(x = y'),\ M(x, y, z) = z$
3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$
4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$
5- Inverter Propagation: $M'(x, y, z) = M(x', \ldots, z')$



MIG Depth Optimization

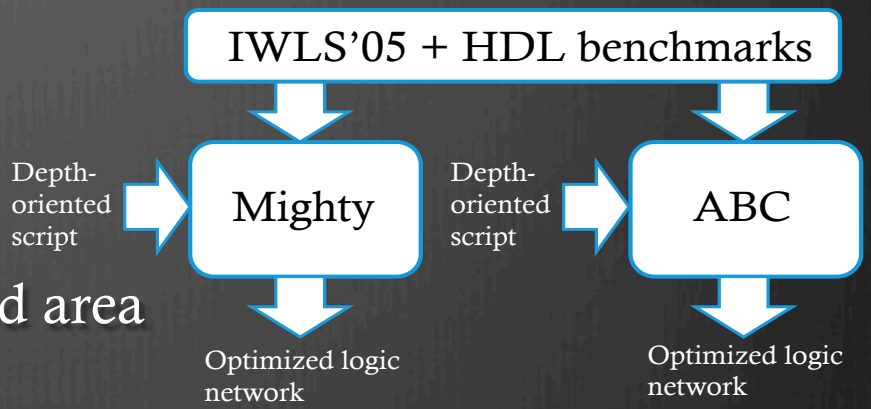
- Rationale: move critical variables closer to the outputs via associativity, distributivity and majority rules
- Reshaping the MIG with other Ω rules

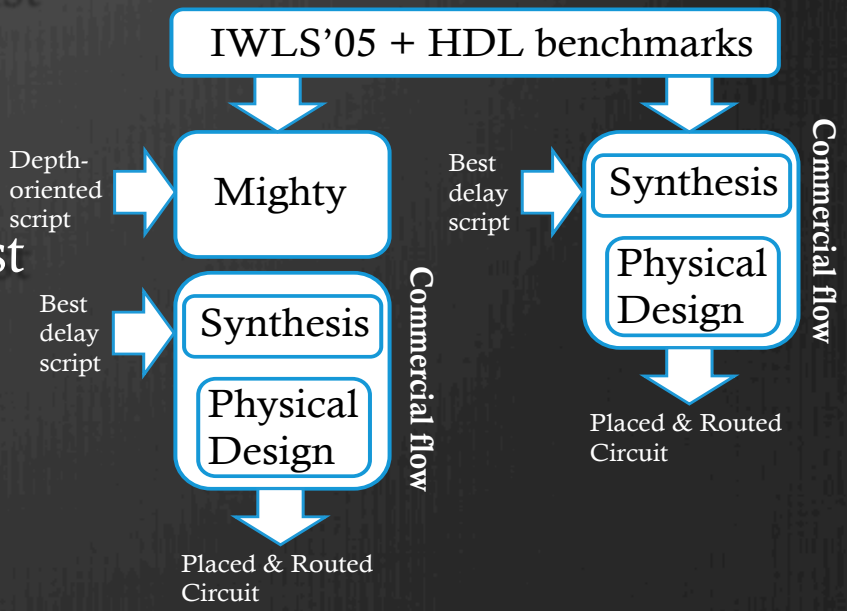


Experimental Results

Methodology

- We implemented in C language a MIG optimizer, called Mighty.
- We target high speed results with bounded area and power overhead.
- Pure logic optimization experiments against AIG-optimization.
  - Case study on adder optimization.
- Complete ASIC design experiments against commercial tools.
  - Results verified with Synopsys Formality.
- Nanotechnology design experiments.



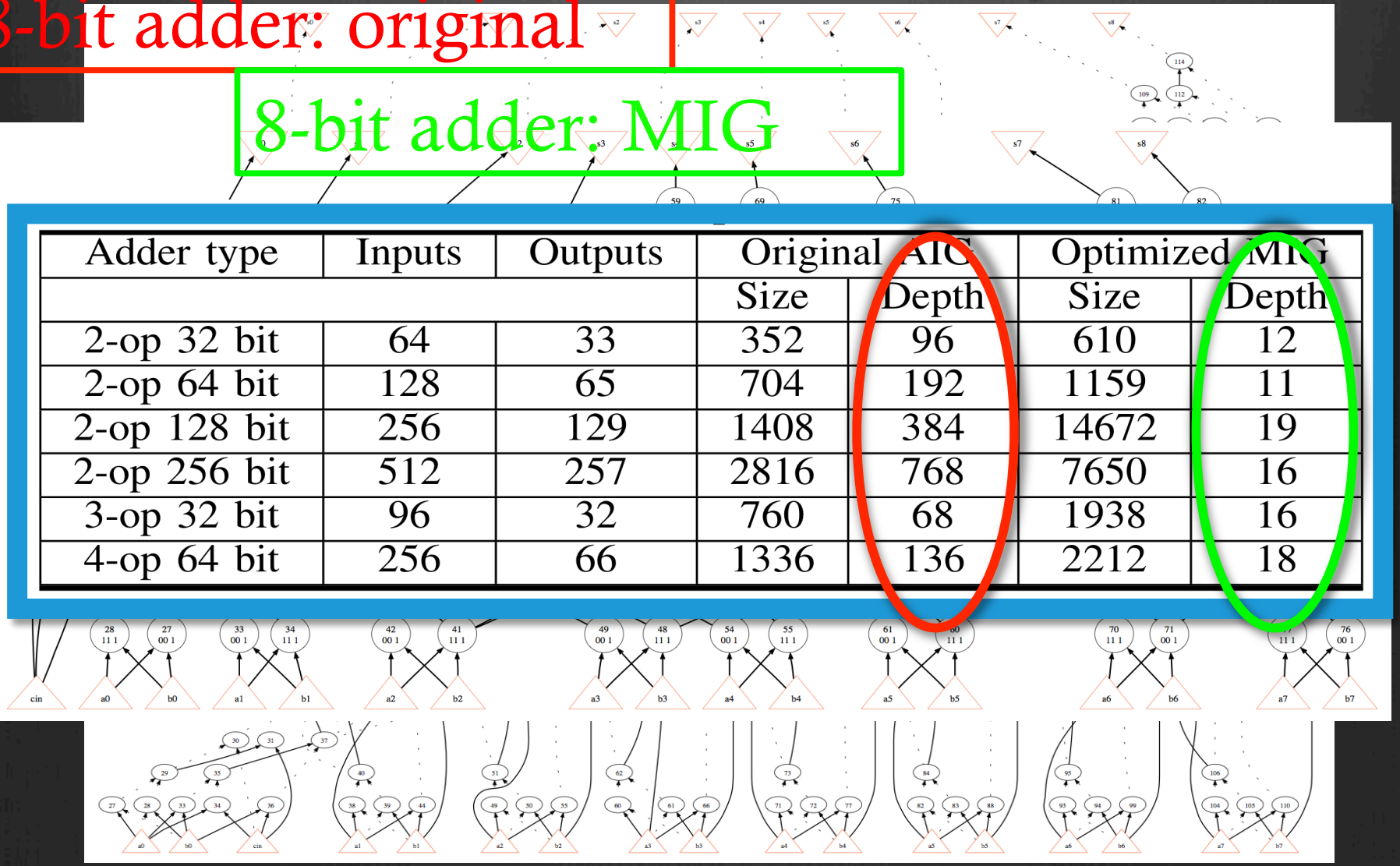


Logic Optimization Experiments: Adders Case Study

8-bit adder: original

8-bit adder: MIG

Adder type	Inputs	Outputs	Original AIG		Optimized MIG	
			Size	Depth	Size	Depth
2-op 32 bit	64	33	352	96	610	12
2-op 64 bit	128	65	704	192	1159	11
2-op 128 bit	256	129	1408	384	14672	19
2-op 256 bit	512	257	2816	768	7650	16
3-op 32 bit	96	32	760	68	1938	16
4-op 64 bit	256	66	1336	136	2212	18



ASIC Design Experiments

Advanced 22nm CMOS

Well-established 90nm CMOS

MIG as f… to L.S. & P.D.

MIG as f… to L.S. & P.D.

27 b…

and…

Behavioral

```
module div32 (a, b, quotient_uns, quotient_tc, remainder_uns,
 remainder_tc);
 parameter width = 32;
 input [width-1
 output [width-1
 output signed [w
 // operators for
 assign quotient_
 assign quotient_tc = $signed(a) / $signed(b);
 assign remainder_uns = a % b;
 assign remainder_tc = $signed(a) % $signed(b);
endmodule
```

Area: 0.21 mm²

Delay: 11.22 ns

GC: 37k

MIG

```
assign w0 = (w39266 & w27196) | (w39266 & w42866) | (w27196 & w42866);
assign w1 = w44100 & w4464;
assign w2 = ~w19966 & w27946;
assign w3 = ~w22044 & ~w30809;
assign w4 = w42722 & ~b[26];
assign w5 = (~w30032 & w19016) | (~w30032 & w47192) | (w19016 & w47192);
assign w6 = ~w26074 & w5179;
assign w7 = (w42822 & ~w46131) | (w42822 & w16107) | (~w46131 & w16107);
assign w8 = ~w3517
assign w9 = ~w4555
assign w10 = ~w144
assign w11 = (~w16
assign w12 = (w271
assign w13 = (w283
assign w14 = ~w252
assign w15 = (w437
assign w16 = ~w443
assign w17 = ~w229
assign w18 = ~w303
assign w19 = (~w23
assign w20 = w2680
assign w21 = (~w43
assign w22 = ~w36188 & w11587;
assign w23 = ~w33838 & ~w16188;
assign w24 = ~w38787 & w19348;
assign w25 = w42096 & w20651;
assign w26 = (w45543 & w39924) | (w45543 & w25446) | (w39924 & w25446);
assign w27 = ~w34177 & w38267;
assign w28 = ~w10074 & ~w29118;
assign w29 = w25495 & w32055;
assign w30 = (w48190 & w13546) | (w48190 & w19290) | (w13546 & w19290);
```

Area: 0.18 mm²

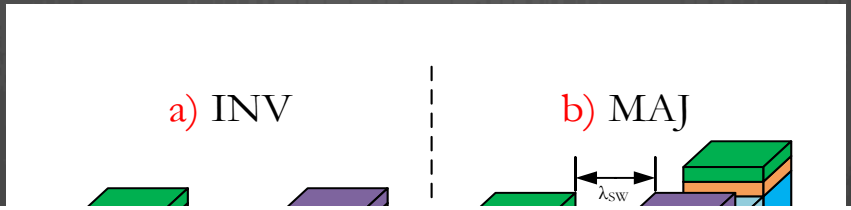
Delay: 10.10 ns

GC: 24k

Nanotechnology Design

Spin Wave Device

feature size 24 nm



		SWD technology - MIG			SWD technology - AIG		
Benchmarks	I/O	A (μm^2)	D (ns)	P (μW)	A (μm^2)	D (ns)	P (μW)
bigkey	487/421	152.50	3.14	2.11	170.99	3.14	2.34
my_adder	33/17	9.42	6.11	0.07	5.00	10.28	0.04
cla	129/65	36.57	7.60	0.21	32.21	11.77	0.19
dalu	75/16	50.47	6.71	0.31	39.17	9.39	0.25
b9	41/21	5.60	2.24	0.08	5.60	2.54	0.08
count	35/16	6.36	2.54	0.11	4.67	6.11	0.09
alu4	14/8	47.81	4.62	0.42	49.22	4.62	0.43
clma	416/115	433.59	12.96	1.37	450.15	14.15	1.42
mm30a	124/120	41.57	30.52	0.06	35.70	37.66	0.05
s38417	1494/1571	319.86	7.01	1.92	319.86	7.9	1.88
misex3	14/14	45.84	4.33	0.43	44.14	4.62	0.41
Average	212/176	90.02	9.07	0.53	89.60	11.02	0.53

Conclusions

- Majority-Inverter Graphs support optimization techniques.
  - The expressive power of MIG Boolean algebra axioms, such as distributivity and inverter propagation, permits more agile logic manipulation.

- MIG optimization show promising results.
  - Strong optimization of arithmetic, e.g., adders: automatic ripple-carry to carry-look-ahead transformation.
  - At the design level, we showed reductions in delay, area and power as compared to a modern commercial ASIC flow.
  - Efficient design of circuits in nanotechnologies where majority is the primitive gate for logic computation.

Questions?

Thank you for your attention!

Backup slides

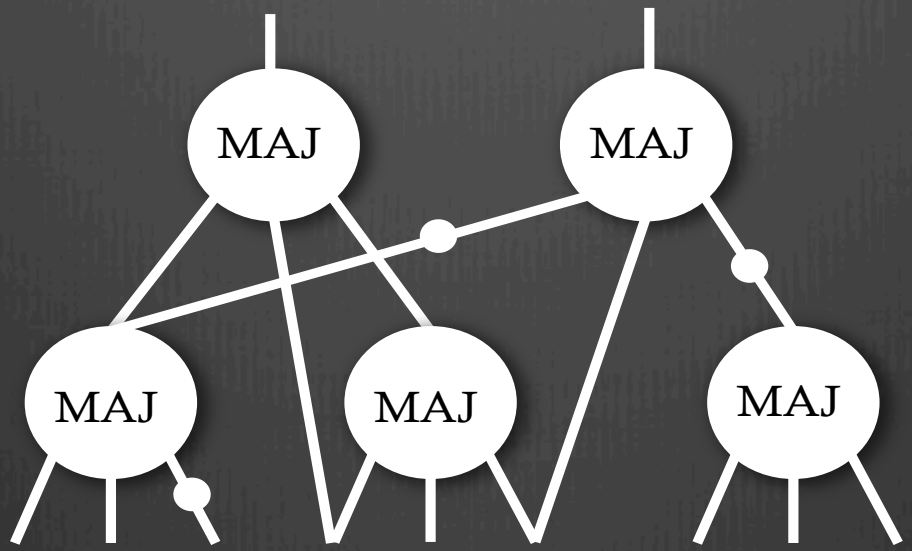
Pushing MIG Optimization to the Limits

- Good, we have an algebraic framework for MIG optimization.
- But…
- There exist practical cases where algebraic optimization heuristics fail to produce improved results.
- … so how to attain further optimization?
- Exploit the Boolean properties of MIG data structure.

MIG Boolean Optimization!

MIG Boolean Optimization

- Exploit the Boolean nature of MIGs.

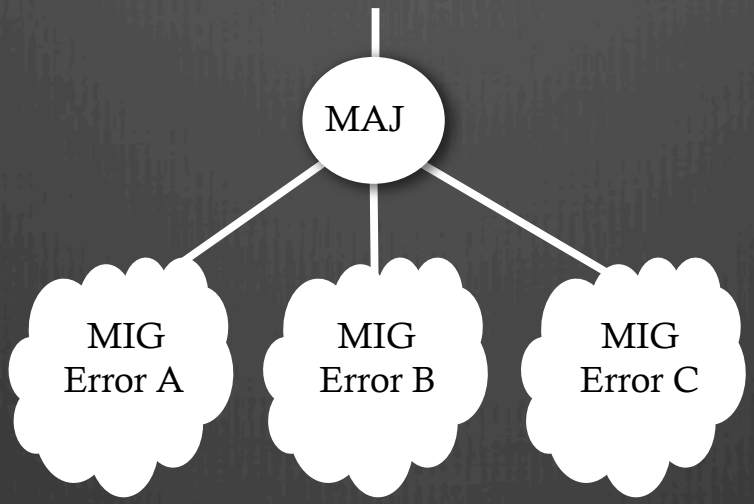


- An MIG is hierarchical majority voting system.
- Majority voting can correct various types of bit-errors.

MIG Boolean Optimization

- MIG voting resilience allows us to insert logic errors.

Original MIG functionality



- Logic errors smartly placed can heavily simplify a logic network.
- Not all types of logic errors can be inserted.
- **To be safe, errors f^A, f^B and f^C must be pairwise orthogonal**

IWLS'05+HDL Benchmarks

		MIGhty					ABC				
Benchmark	I/O	Opt. MIG		Map. MIG			Opt. AIG		Map. AIG		
Open Cores IWLS'05		Size	Depth	LUT6	Depth	Runtime (s)	Size	Depth	LUT6	Depth	Runtime (s)
DSP	4223/3953	40517	34	11077	11	7.98	39958	41	11309	12	5.39
ac97_ctrl	2255/2250	10745	8	2917	3	6.52	10497	9	2914	3	8.98
aes_core	789/668	20947	18	3902	4	11.78	20632	19	3754	5	8.22
des_area	368/72	4186	22	735	6	1.04	5043	24	1012	7	2.11
des_perf	9042/9038	67194	15	12796	3	34.22	75561	15	12814	3	25.43
ethernet	10672/10696	57959	15	18108	6	23.69	56882	22	18267	6	36.54
			8								
			19								
			16								
			11								
			6								
			8								
spi	274/276	3337	19	812	6	3.71	3430	24	854	7	2.28
ss_pcm	106/98	397	6	104	2	0.01	381	6	104	2	0.01
systemcaes	930/819	9547	25	1845	7	5.26	11014	31	2060	8	4.79
systemcdes	314/258	2453	19	515	5	2.21	2495	21	623	5	1.05
tv80	373/404	7397	30	1980	11	6.43	7838	35	2036	11	2.97
usb_funct	1860/1846	12995	19	3333	5	13.45	13914	20	3394	5	9.04
usb_phy	113/111	372		136	2	0.11	380		136	2	0.05
IWLS'05 total		266613	305	66791	96	120.38	277618	354	68201	103	109.52
Arithmetic HDL		Size		LUT6	Depth	Runtime (s)	Size		LUT6	Depth	Runtime (s)
MUL32	64/64	9096	36	1852	10	2.90	8903	40	1993	11	1.90
sqrt32	32/16	2171	164	544	54	1.02	1353	292	236	55	1.22
			219					235			
			102					132			
			61					73			
			41					54			
			77					95			
			143					162			
mul64	128/128	25773	109	6557	31	13.84	26024	186	6751	43	10.09
max	512/130	4210	29	1023	12	1.67	2964	113	818	20	2.23
square	64/127	17550	40	4393	13	18.66	17066	168	4278	35	12.24
log2	32/32	31326	201	8809	59	23.32	30701	272	8223	73	15.54
Arithmetic total		149727	1222	39432	355	155.62	154436	1822	39650	422	94.37

IWLS'05:

-17.98% depth

IWLS'05:

-12.65% size

IWLS'05:

-10.00% SW act.

Arith. HDL:

-26.69% depth

Arith. HDL:

-7.7% size

Arith. HDL:

-0.1% SW act.

MIG Optimization Tool

```
assign w1981 = w1491 & ~w1326;
assign w1982 = w1491 & ~w1327;
assign w1983 = (w225 & ~w1340) | (w225 & ~w1339) | (~w1340 & ~w1339);
assign w1984 = (w225 & ~w1340) | (w225 & ~w1338) | (~w1340 & ~w1338);
assign w1985 = (~w1342 & ~w1341) | (~w1342 & ~w1339) | (~w1341 & ~w1339);
assign w1986 = (~w1342 & ~w1341) | (~w1342 & ~w1338) | (~w1341 & ~w1338);
assign w1987 = w1495 & ~w1354;
assign w1988 = w1495 & ~w1353;
assign w1989 = (~w372 & ~w374) | (~w372 & w1987) | (~w374 & w1987);
assign w1990 = (~w372 & ~w374) | (~w372 & w1988) | (~w374 & w1988);
assign w1991 = (~w393 & w1806) | (~w393 & w2007) | (w1806 & w2007);
assign w1992 = (~w393 & w1806) | (~w393 & w2008) | (w1806 & w2008);
assign w1993 = (~w1364 & ~w1363) | (~w1364 & w1361) | (~w1363 & w1361);
assign w1994 = (~w1364 & ~w1363) | (~w1364 & w1360) | (~w1363 & w1360);
assign w1995 = (~w1366 & ~w1365) | (~w1366 & w1361) | (~w1365 & w1361);
assign w1996 = (~w1366 & ~w1365) | (~w1366 & w1360) | (~w1365 & w1360);
assign w1997 = w1499 & ~w1398;
assign w1998 = w1499 & ~w1397;
assign w1999 = (w1401 & w1400) | (w1401 & ~w1398) | (w1400 & ~w1398);
assign w2000 = (w1401 & w1400) | (w1401 & ~w1397) | (w1400 & ~w1397);
assign w2001 = (w1405 & w1404) | (w1405 & ~w1398) | (w1404 & ~w1398);
assign w2002 = (w1405 & w1404) | (w1405 & ~w1397) | (w1404 & ~w1397);
assign w2003 = w962 & ~w1563;
assign w2004 = w962 & ~w1564;
assign w2005 = w1492 & ~w167;
assign w2006 = (~w184 & ~w186) | (~w184 & w2005) | (~w186 & w2005);
assign w2007 = (~w372 & ~w374) | (~w372 & w1987) | (~w374 & w1987);
assign w2008 = (~w372 & ~w374) | (~w372 & w1988) | (~w374 & w1988);
assign w2009 = (~w1357 & ~w1356) | (~w1357 & w354) | (~w1356 & w354);
assign w2010 = (~w1359 & ~w1358) | (~w1359 & w354) | (~w1358 & w354);
assign w2011 = (w645 & ~w1385) | (w645 & ~w630) | (~w1385 & ~w630);
assign w2012 = (~w1387 & ~w1386) | (~w1387 & ~w630) | (~w1386 & ~w630);
assign w2013 = (w498 & ~w1811) | (w498 & w1995) | (~w1811 & w1995);
assign w2014 = (w498 & ~w1811) | (w498 & w1996) | (~w1811 & w1996);
assign one = 1;
assign po00 = w8;// level 6
assign po01 = w24;// level 9
assign po02 = w44;// level 11
assign po03 = w65;// level 12
assign po04 = w86;// level 13
assign po05 = w107;// level 14
assign po06 = w127;// level 15
assign po07 = w148;// level 16
assign po08 = w169;// level 16
assign po09 = w189;// level 15
assign po10 = w209;// level 17
assign po11 = w230;// level 18
assign po12 = w251;// level 19
assign po13 = w272;// level 19
assign po14 = w293;// level 19
```

```
[lsi1mac11:~/Desktop/MLP] amaru% ./MLP
EPFL, Integrated Systems Laboratory, CAD Group
Majority Logic Manipulation Package, MLP, Beta version
mig 0> read CSA464.aig
AIGER read, Inputs 256, Outputs 66, ANDs 1336
Depth: 136, Size: 1336
mlp 1> selective_depth
Depth: 136, Size: 1336
SD-pre -- Depth: 136, Size: 1336
SD-pre -- Depth: 27, Size: 1550
SD-pre -- Depth: 26, Size: 1576
SD-pre -- Depth: 25, Size: 1602
SD-pre -- Depth: 25, Size: 1640
SD-pre -- Depth: 25, Size: 1640
SD-aft -- Depth: 25, Size: 1640
Depth: 25, Size: 1640
mlp 2> aggressive_depth 1.2
Depth: 25, Size: 1640
AD-pre -- Depth: 25, Size: 1640
AD-aft -- Depth: 24, Size: 1970
Depth: 24, Size: 1970
mlp 3> selective_depth
Depth: 24, Size: 1970
SD-pre -- Depth: 24, Size: 1970
SD-pre -- Depth: 22, Size: 1978
SD-pre -- Depth: 20, Size: 2026
SD-pre -- Depth: 19, Size: 2038
SD-pre -- Depth: 19, Size: 2080
SD-pre -- Depth: 19, Size: 2080
SD-aft -- Depth: 19, Size: 2080
Depth: 19, Size: 2080
mlp 4> bounded_size
Depth: 19, Size: 2080
SR-pre -- Depth: 19, Size: 2080
SR-aft -- Depth: 19, Size: 2015
Depth: 19, Size: 2015
mlp 5> write_minimal
File vMIG.CSA464.aig.v written successfully
mlp 6> quit
[lsi1mac11:~/Desktop/MLP] amaru%
```