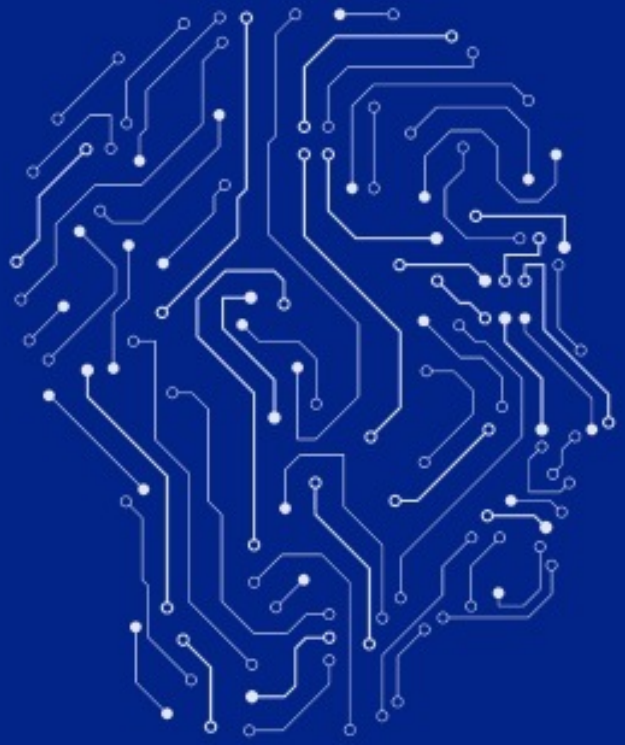


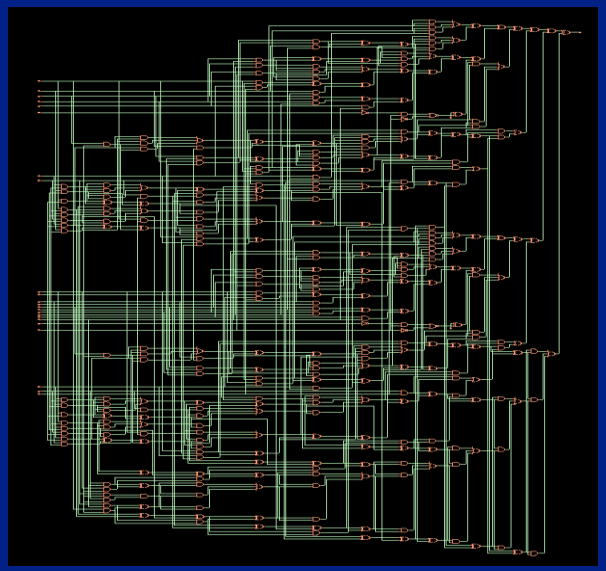
Coupling Reverse Engineering and SAT to Tackle NP-Complete Arithmetic Circuitry Verification in O(# of gates)

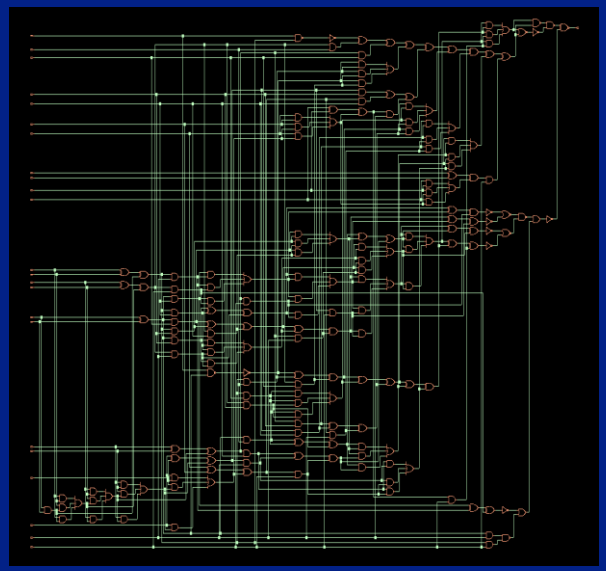
Easy-Logic Technology Ltd.

Outline

- Difficulty of SAT in Comparing Arithmetic Logic
- Formal Verification by Macro Level Function Checking
- Experimental Results
- Conclusion

Formal Verification





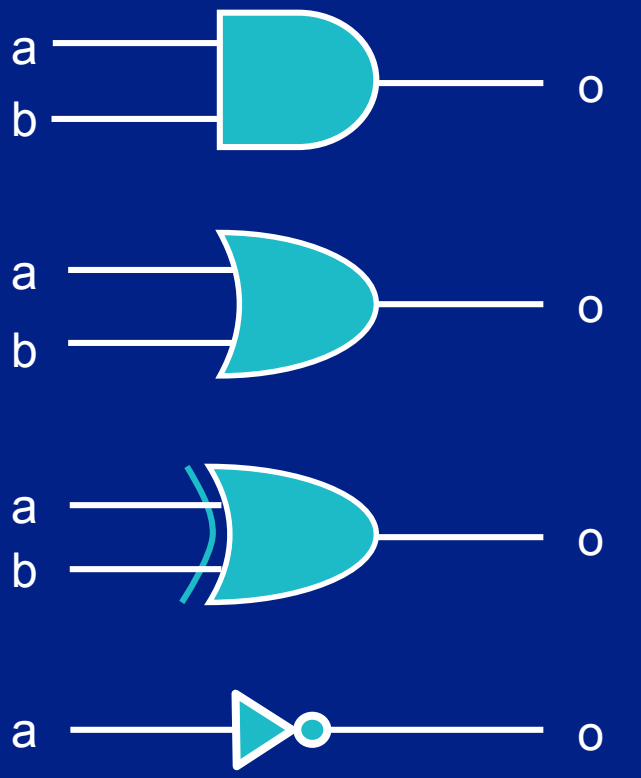
Golden circuit

Use SAT!

Revised circuit

SAT Solver

Translate the circuit into a set of CNF clauses

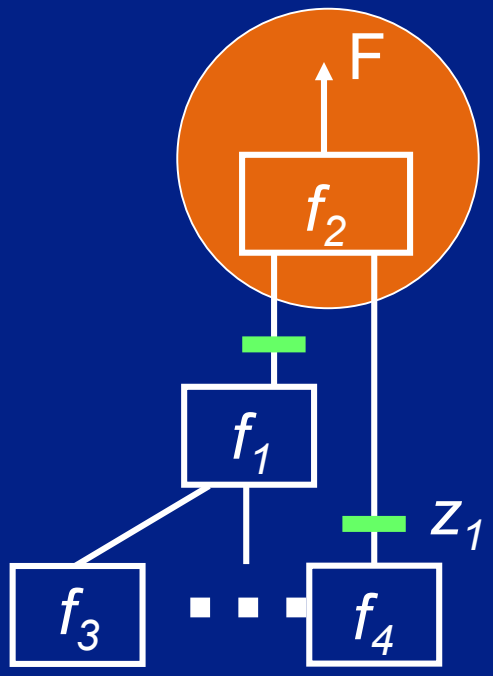


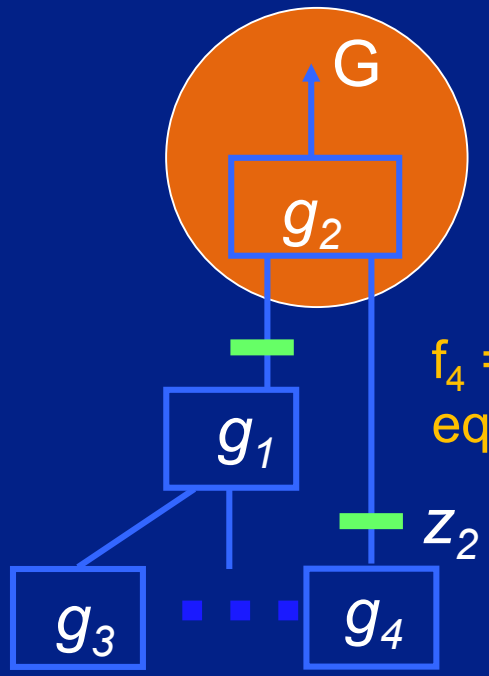
Performance of SAT is poor when circuits contain arithmetic macros, especially multipliers.

Solve the CNF by SAT solver (MiniSat, Glucose, Lingeling, etc)

Verification & Internal equivalence

- Equivalence checkers can perform extremely well if the two designs to be compared contain a high degree of structural similarity, which means that the two circuits contain a lot of internal equivalences.





$f_4 = g_4 \Rightarrow (z_1, z_2)$ are internal equivalences

Verification & Internal equivalence

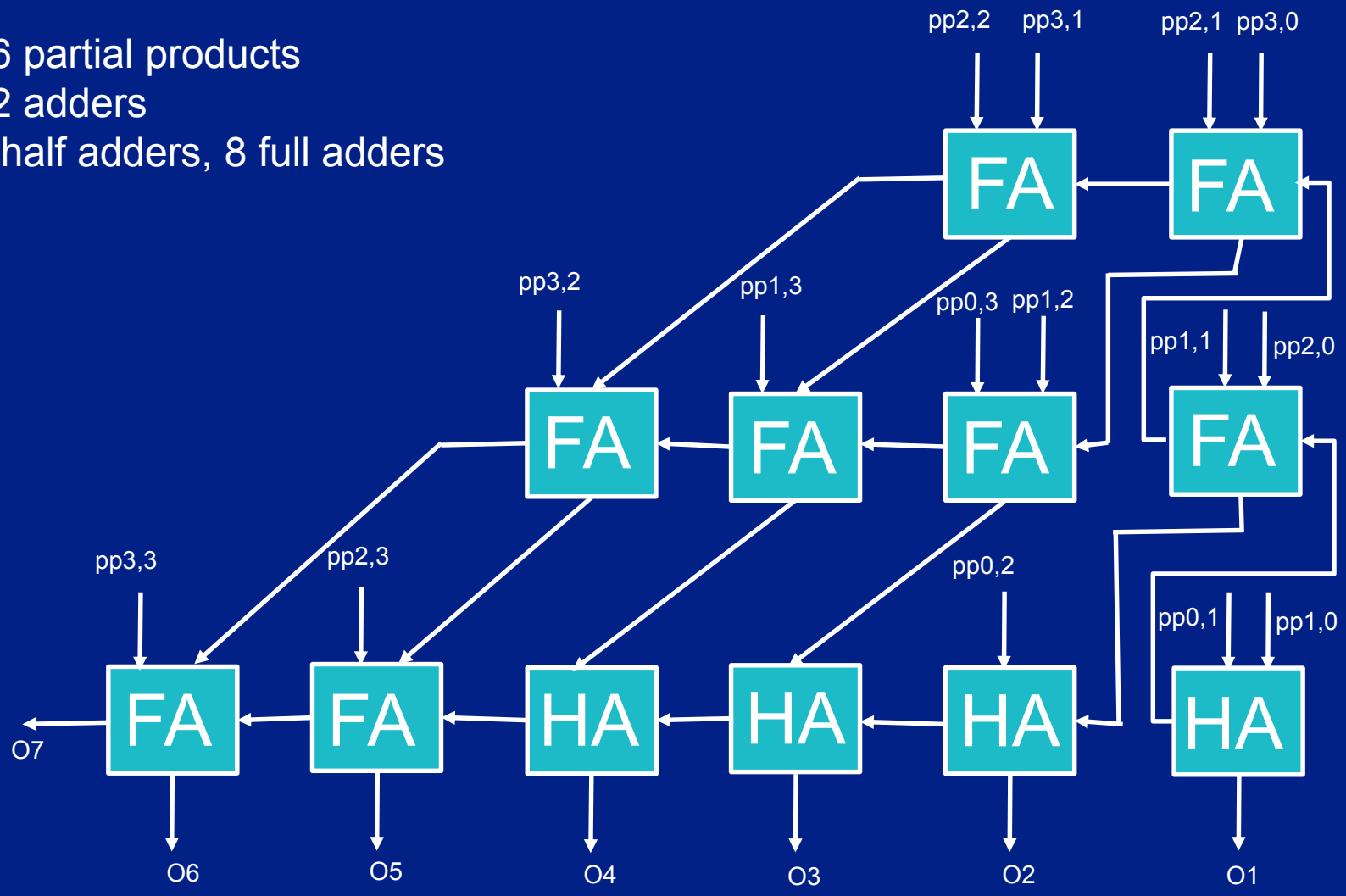
- check equivalence within functional blocks separated by (internal) equivalent points
  - Assign CNF variables one by one
  - Propagate new assignment and see if there is any conflict

Verification & Internal equivalence

- In other words, if no internal equivalences exist, verification can become impossible even for small cases.

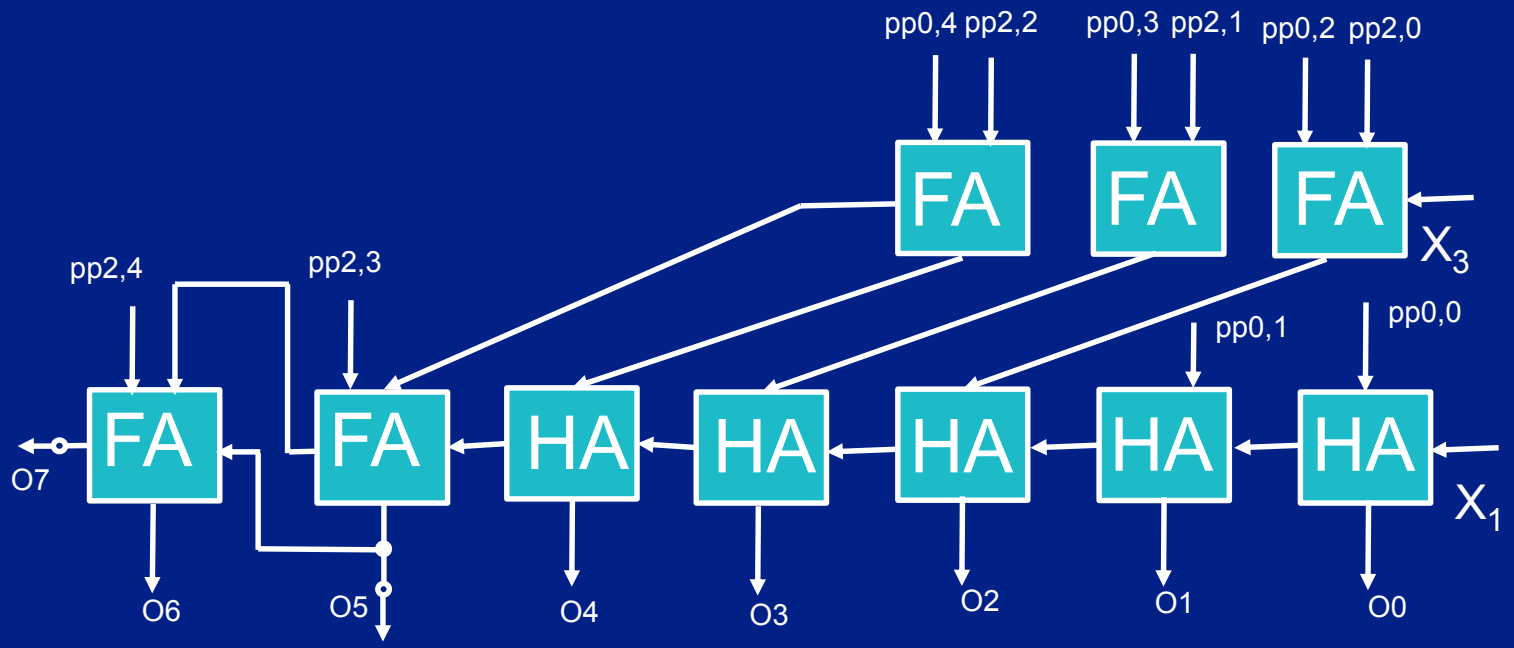
A 4×4 Wallace Tree Multiplier

16 partial products
12 adders
4 half adders, 8 full adders

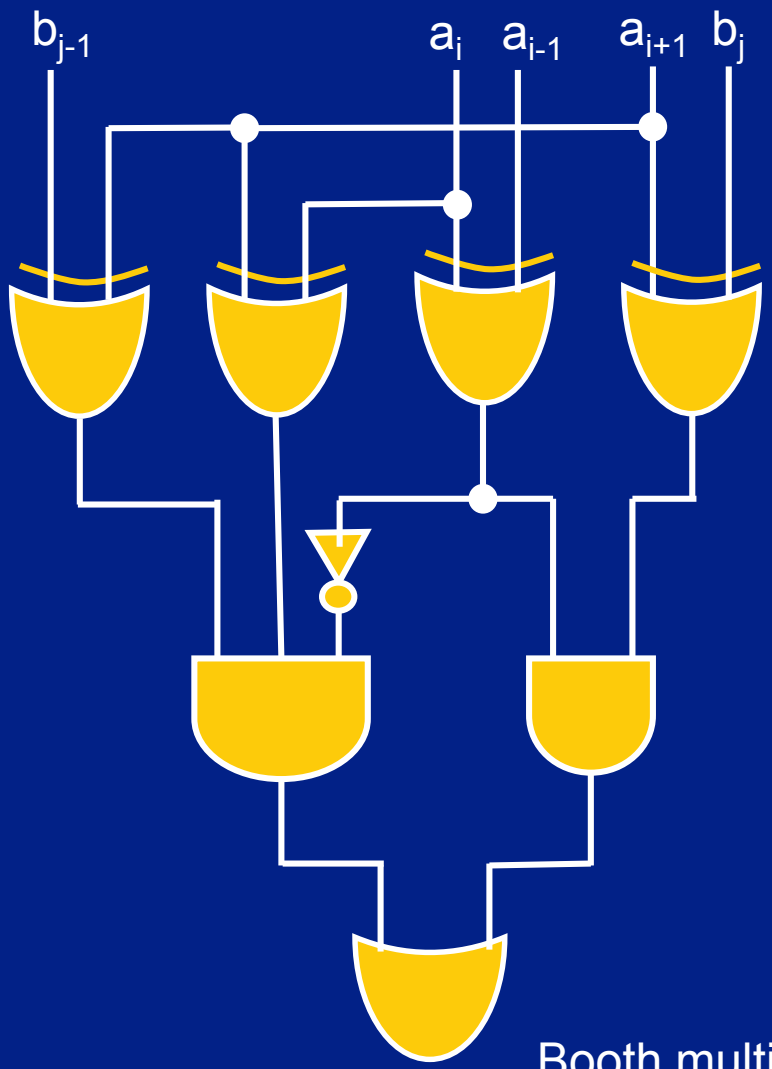


A 4×4 Booth Multiplier

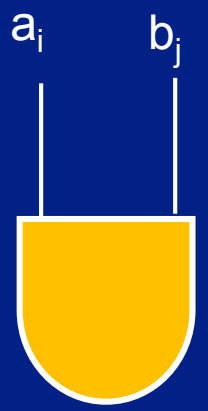
10 partial products, 2 constants
10 adders
5 half adders, 5 full adders



Partial Products Difference

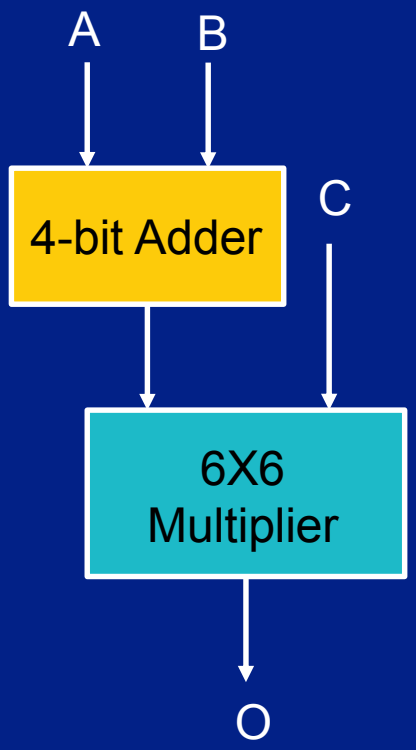


Booth multiplier partial product

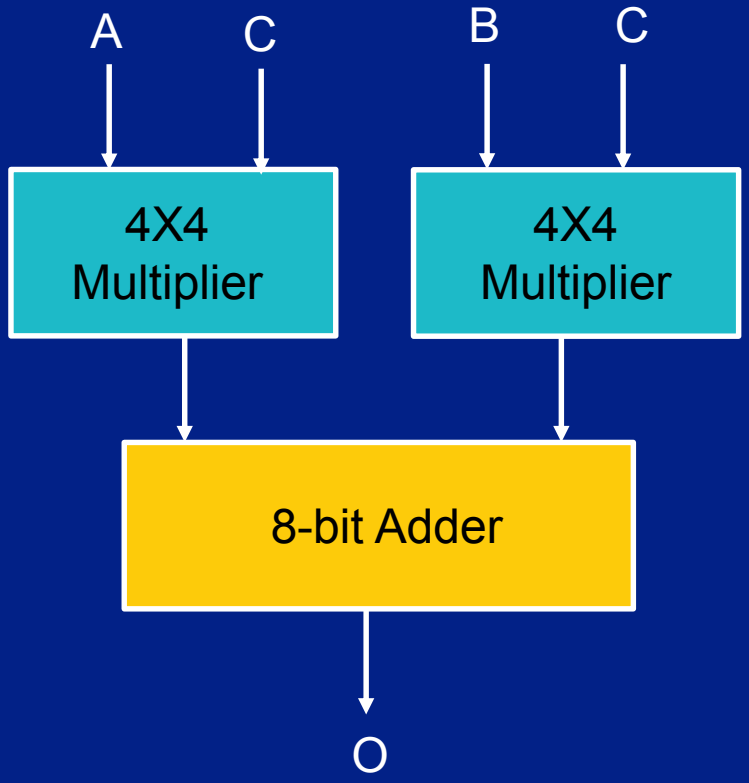


Wallace tree multiplier partial product

Multiplier and Adder



(A + B) X C



A X C + B X C

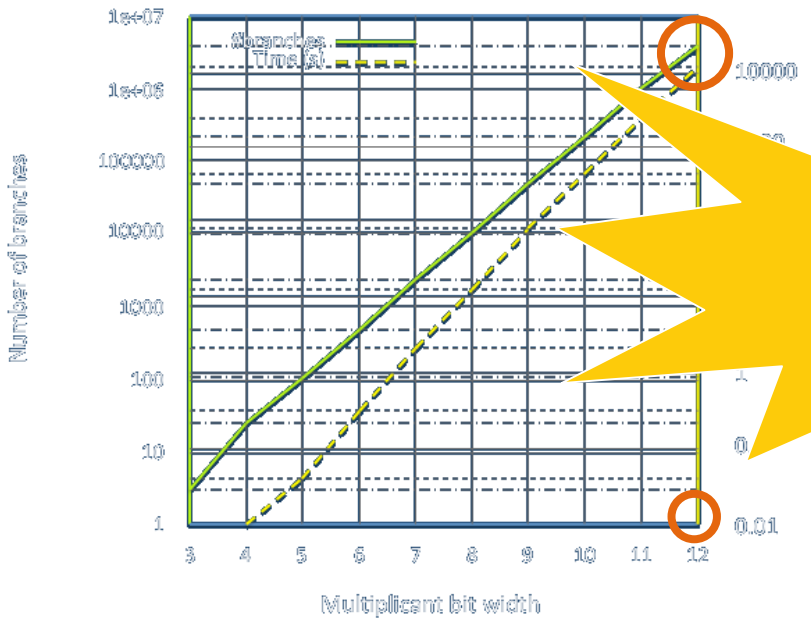
Difficulties of Verify Multipliers

- BDD: requires $O(2^n)$ memory to represent an n-bit multiplier [1]
- SAT: requires $O(2^n)$ branches or decisions [2]

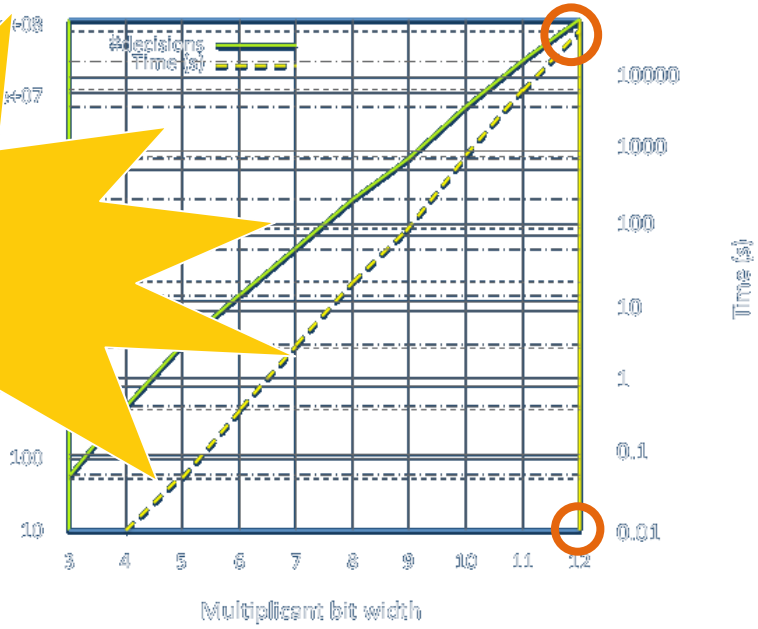
[1] Bryant, Randal E. "Graph-based algorithms for Boolean function manipulation.“ *Computers, IEEE Transactions on* 100.8 (1986): 677-691.
[2] Järvisalo, Matti. "Equivalence checking multiplier designs (2007) SAT Competition 2007 benchmark description."

SAT Performance of Verifying Two Different Multipliers*

Solving time for a 64-bit multiplier?



Over 10^{20} centuries!



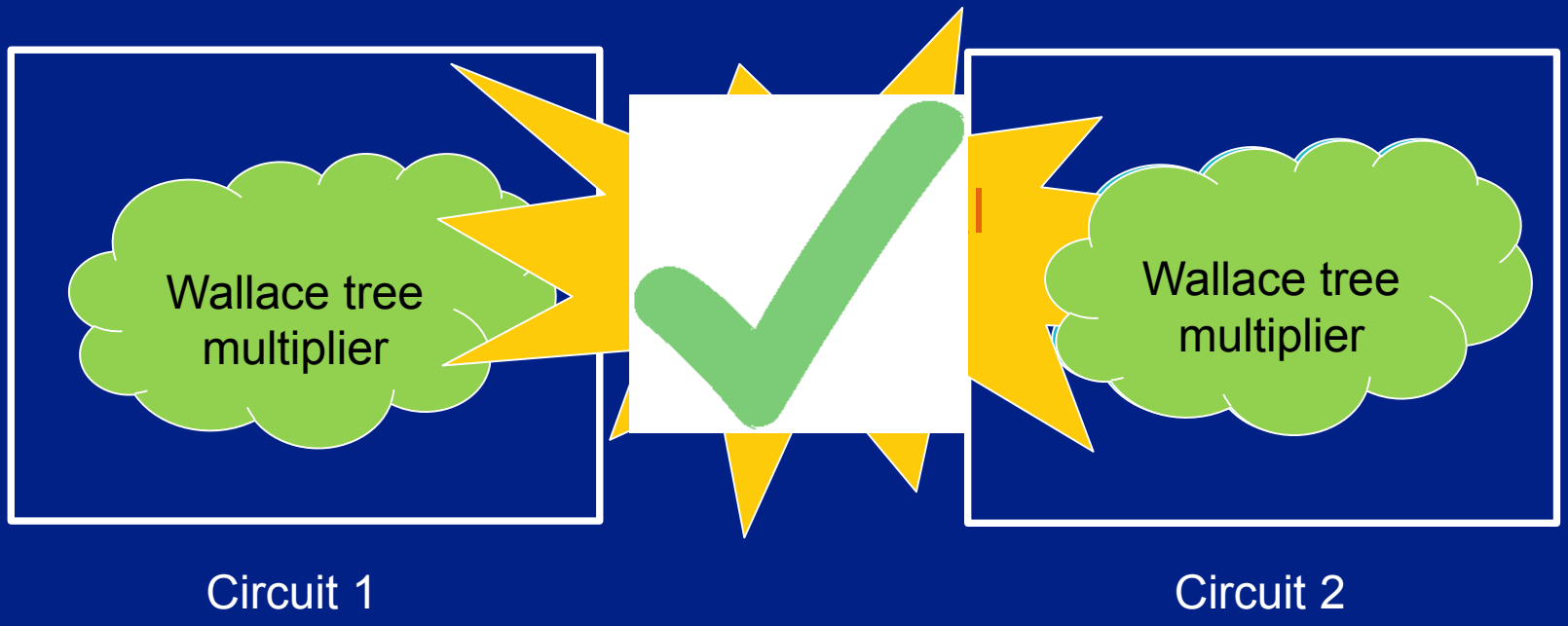
Results for Satz 2.15

Results for Minisat 2.0 with preprocessing

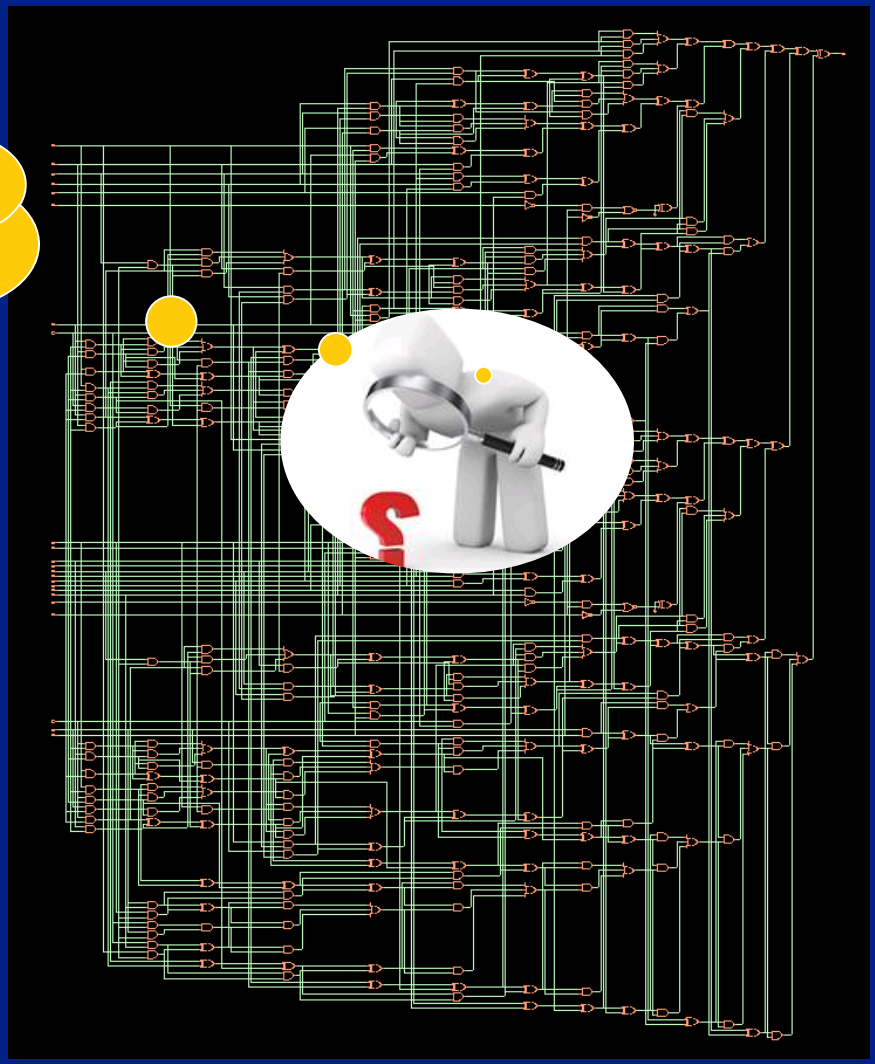
* Järvisalo, Matti. "Equivalence checking multiplier designs (2007) SAT Competition 2007 benchmark description."

Outline

- Difficulty of SAT in Comparing Arithmetic Logic
- Formal Verification by Macro Level Function Checking
- Experimental Results
- Conclusion


Wallace tree multiplier
Wallace tree multiplier
Circuit 1
Circuit 2

Where are the macros?



- Sizes are very large
- Buried in a flatten design

Reverse Engineering

- Though it's difficult to verify a multiplier from pure logic view, we can recognize it from some intrinsic features.
- We propose a reverse engineering scheme which can map1-bit adder arithmetic macros including adders, multipliers (Wallace tree or Booth), and multiplexers.

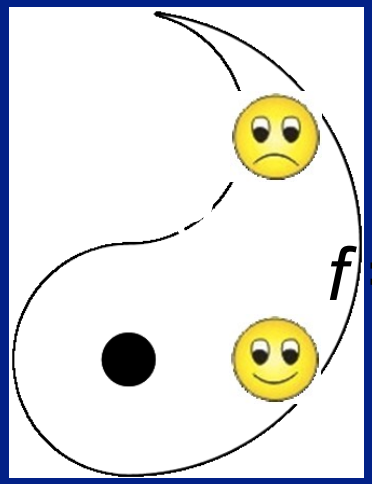
Solve $f = g$?

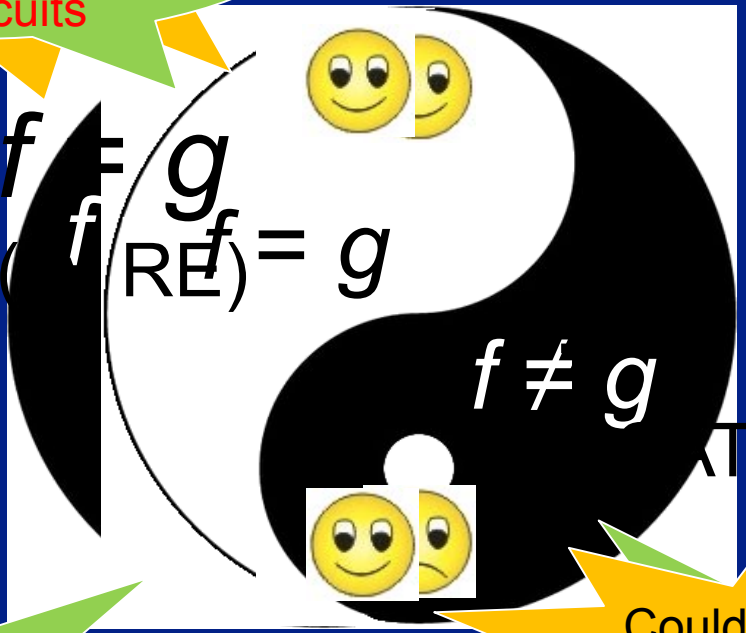
by Reverse Engineering + Solver

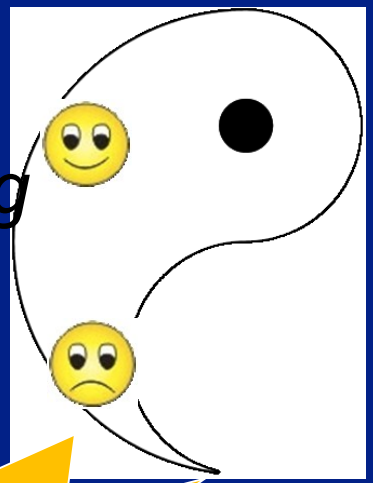
by SAT

("structural DNA tracing" method)

Can be solved in P time for "practical" industrial circuits



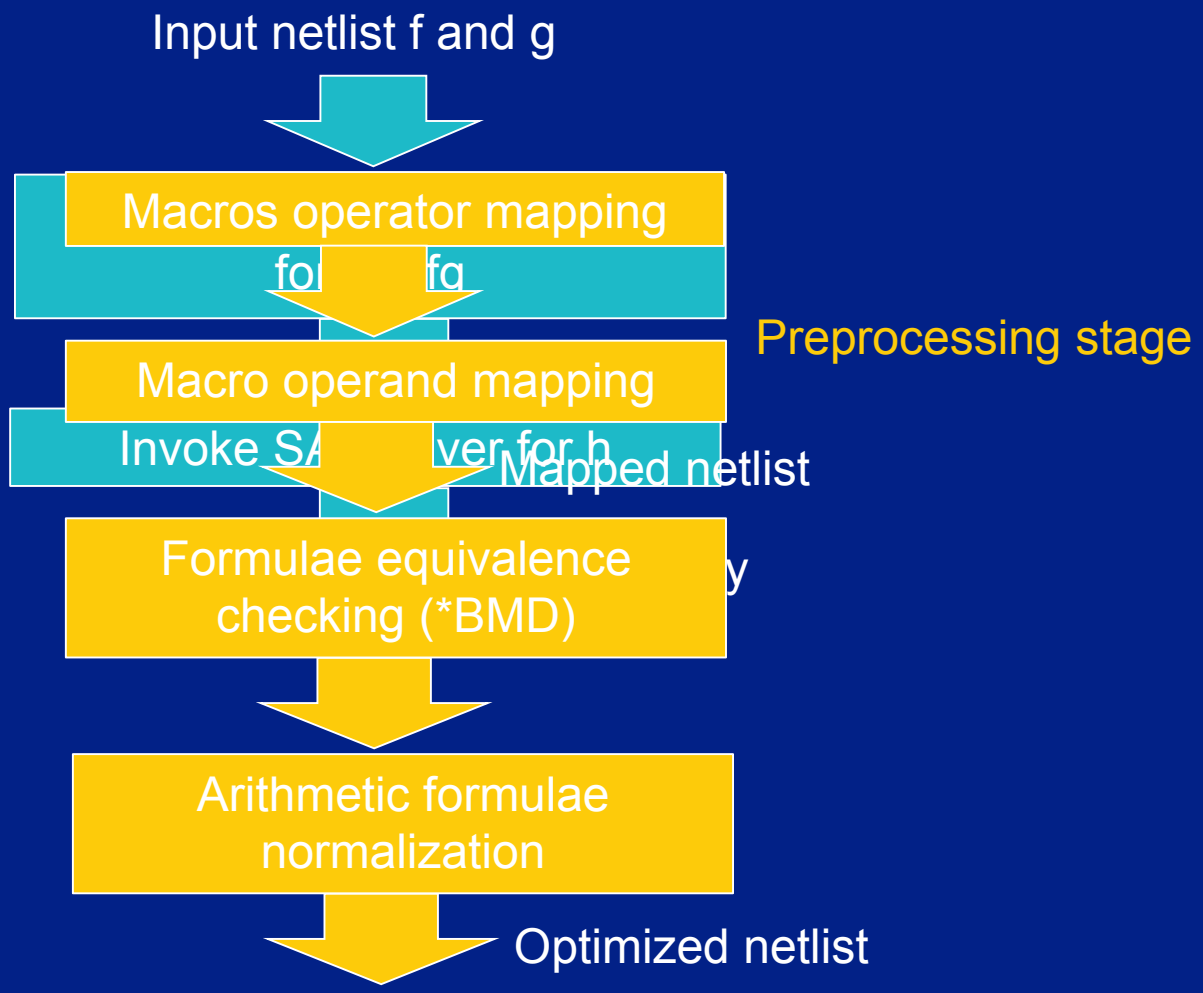




Efficiently solved in P time in any condition

Could be exponential

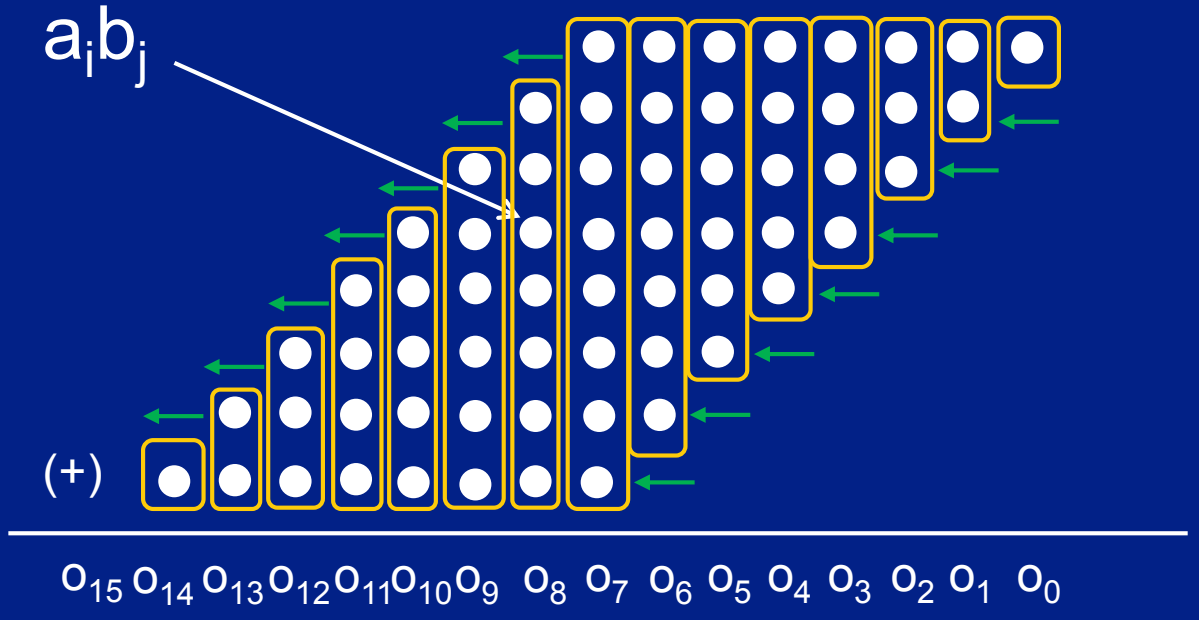
Our Flow


Input netlist f and g
Macros operator mapping
Macro operand mapping
Invoke SA ver for h
Mapped netlist
Formulae equivalence checking (*BMD)
Arithmetic formulae normalization
Optimized netlist
Preprocessing stage

8-bit Multiplier Structure

Steps of identify a multiplier macro:

1. Construct adder-trees
2. Construct adder-forest by connecting trees Use carries
3. Determine multiplier boundary

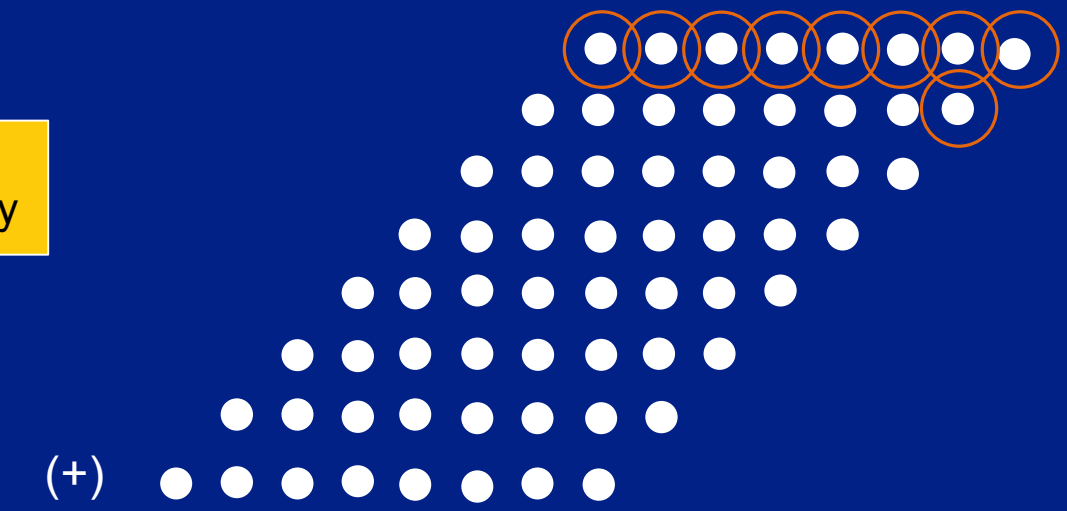


Determine Multiplier Boundary

A: a_7 …… a_1 a_0

B: b_7 …… b_1 b_0

Input boundary

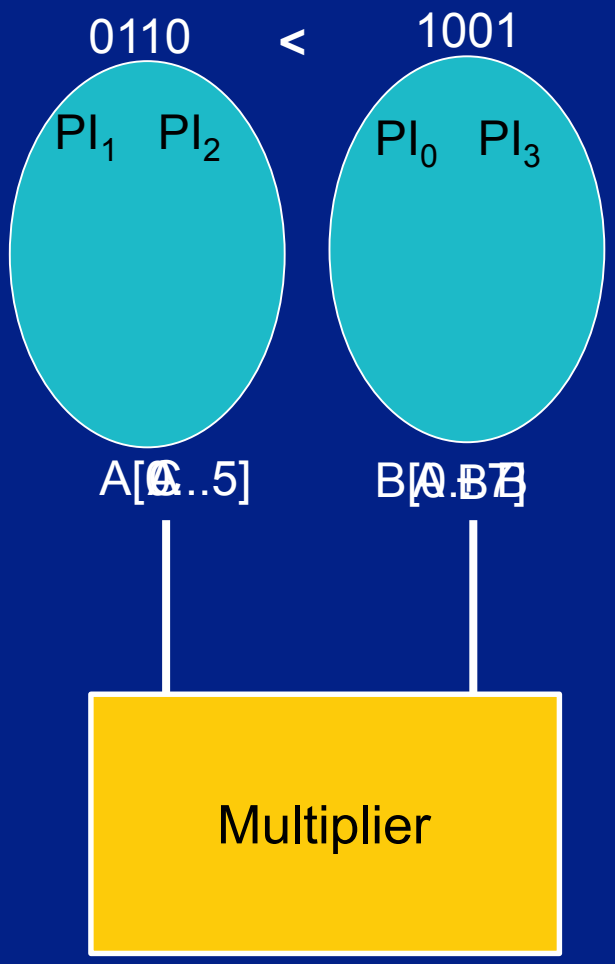


o_{15} o_{14} o_{13} o_{12} o_{11} o_{10} o_9 o_8 o_7 o_6 o_5 o_4 o_3 o_2 o_1 o_0

Output boundary

Operand Mapping

A×B or B×A?



PI: PI_0 PI_1 PI_2 PI_3

Operand with more leading zeros placed on left

Operand ordered by variable placement in fanin cone

Booth Multiplier

- Also 1-bit adder based macro
- Different at partial products, adder tree and adder forest structure
- Mapping process similar to Wallace tree multiplier

Complexity of Mapping Multiplier

- Construct adder trees and forest: linear to circuit size
- Determine multiplier boundary: $O(n^2)$ to n-bit multiplier

Formula Checking & Normalization

- Choose some most common formula patterns.
  - e.g. $a + b$, $a \times b$, $(a + b) \times c$, $a \times b + \mathrm{c} \times d + e \times d$, $a + b + c \times d - e \times f$, $a \times b + c \ldots$
- Create a standard structure form for every chosen formula pattern
  - e.g. use Wallace tree structure as the canonical form of $a \times b$
- Replace every macro by its pre-defined form

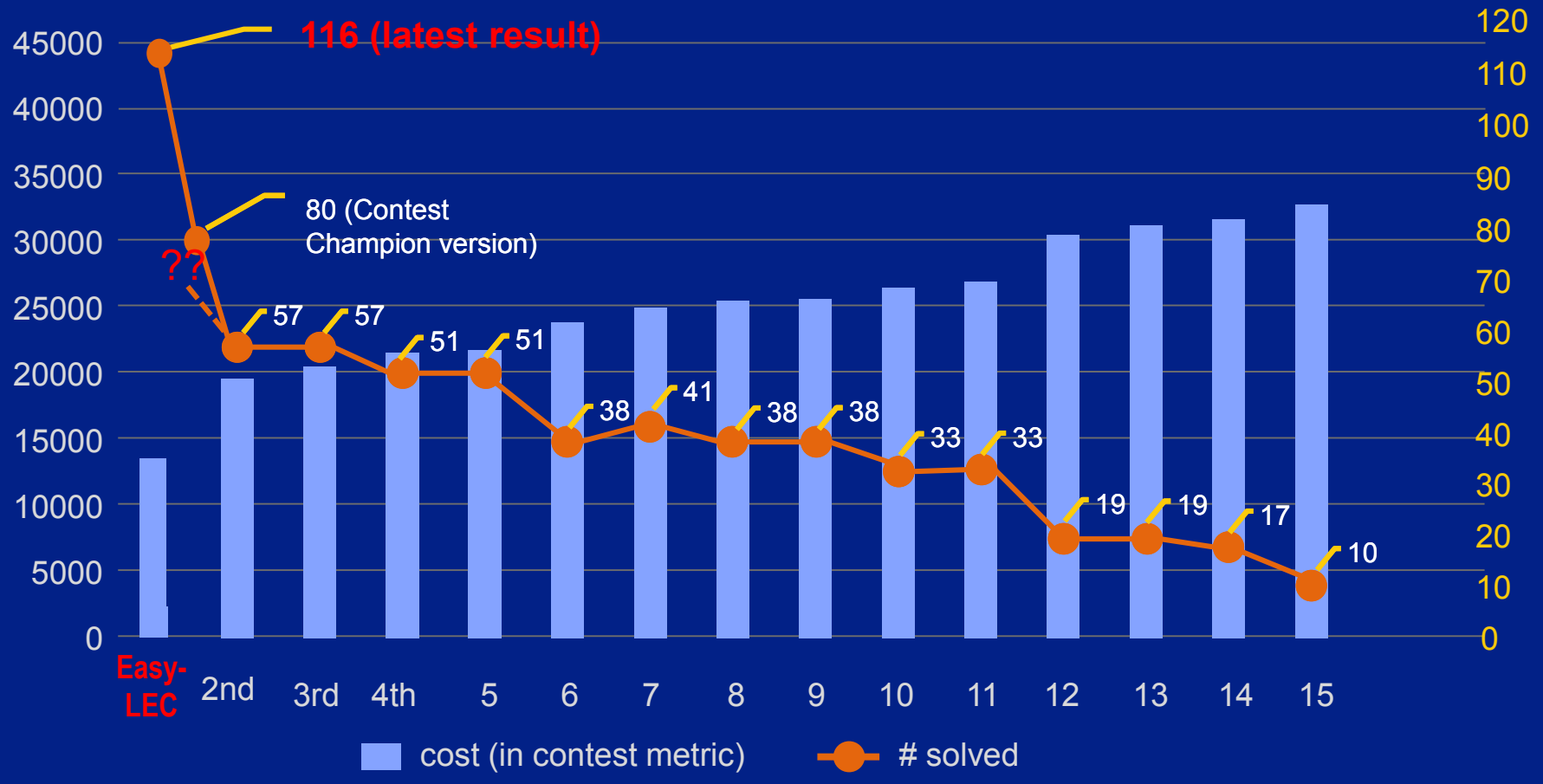
Outline

- Difficulty of SAT in Comparing Arithmetic Logic
- Formal Verification by Macro Level Function Checking
- Experimental Results
- Conclusion

Benchmark Information

Case	#primitive gates	Contained arithmetic macros	Multiplier type	Multiplier size
ut1	280-1261	$(a+b)\times c$; $a\times \mathrm{c}+\mathrm{b}\times \mathrm{c}$	Wallace tree	(6 * 6) – (8 * 7)
ut2	1197-1994	$a\times b$	Booth	(16 * 16) – (16 * 16)
ut3	2727-4226	$a\times b$	Booth	(32 * 32) – (48 * 48)
ut5	1025-2261	$a\times b$; MUX	Wallace tree	(12 * 12) – (12 * 12)
ut7	474-2301	(signed) $a\times b$	Booth; Wallace tree	(9 * 9) – (24 * 24)
ut8	1061-2308	(signed) $a\times b$	Booth; Wallace tree	(23 * 23) – (24 * 24)
ut13	697-2385	$a\times b$	Booth; Wallace tree	(11 * 11) – (17 * 17)
ut14	1402-3402	$a\times b$	Booth; Wallace tree	(17 * 17) – (19 * 17)
ut15	851-3023	$a\times b$	Booth; Wallace tree	(12 * 12) – (17 * 17)
ut20	584-22600	(signed) $a\times b$; $a+b+c\times d-e\times f$	Booth; Wallace tree	(10 * 10) – (45 * 45)
ut26	564-10383	$a\times b$; $a\times b+\mathrm{c}$; $a+b+c\times d+e\times f$	Booth; Wallace tree	(9 * 9) – (28 * 28)
ut32	711-2480	(signed) $a\times b$	Booth; Wallace tree	(10 * 10) – (17 * 17)
ut36	2855-25489	MUX	*	*
ut41	1103-5463	$a\times b$	Booth; Wallace tree	(13 * 13) – (30 * 30)

Contest Results



CNF encoding time limit: 25s
SAT solving time limit: 100s
cost = 4 * CNF encoding time + SAT solving time

Comparison with Commercial Tools

Case		Our results		Commercial tool X results		Commercial tool Y results	
#circuits		#solved	Avg runtime (s)	#solved	Avg runtime (s)	#solved	Avg runtime (s)
ut1	13	13	0.3	13	7.1	13	64.6
ut2	13	13	0.9	13	1.4	4	3827.4
ut3	13	13	3.5	8	854.1	0	5000
ut5	13	13	0.4	13	41.9	13	5.1
ut7	13	13	1.0	13	36.8	13	60
ut8	13	13	1.2	9	301.2	13	854.2
ut13	13	13	0.4	12	704.4	10	1237.6
ut14	13	13	0.7	5	2361.4	2	4308.3
ut15	13	13	0.9	6	827.8	6	2967.2
ut20	13	11	4.1	3	1256.3	3	3867.1
ut26	13	13	1.2	5	1883.8	2	4175.5
ut32	13	13	0.7	6	1538.7	4	3506.3
ut36	13	3	11.7	3	104.9	3	3871.4
ut41	13	13	1.0	2	820.7	1	4530.3
total	182	170		111		87	
avg			2.0		767.2		2733.9
ratio	1	93%	1	61%	381.3x	48%	1358.8x

Some results I

benchmarks	#primitive gates	our results	commercial tool results	description
ut1\test\test1	280	0.24	0.13	Compare (A+B)×C and (A×C)+(B×C)
ut1\test\test2	415	0.39	0.18	
ut1\test\test3	731	0.26	4.95	
ut1\test\test4	878	0.33	6.24	
ut1\test\test5	1005	0.36	11.06	
ut1\test\test6	1108	0.39	10.22	
ut1\test\test7	1187	0.42	9.68	
ut1\test\test8	1256	0.45	14.97	
ut1\test\test9	1261	0.45	9.65	
ut1\test\test10	972	0.45	6.24	
total	9093	3.74	73	

Some results II

benchmarks	#primitive gates	our results	commercial tool results	description
ut41\test\test1	1444	0.52	2564.69	Compare Wallace tree A×B and Booth A×B
ut41\test\test2	1434	1.81	>2056.73	
ut41\test\test3	2258	0.55	>1239.85	
ut41\test\test4	2734	0.76	>705.83	
ut41\test\test5	2997	0.94	>147.21	
ut41\test\test6	3254	1.02	>370.6	
ut41\test\test7	3830	1.13	>204.49	
ut41\test\test8	4139	1.3	>432.46	
ut41\test\test9	5108	1.4	>312.36	
ut41\test\test10	5463	1.71	>351.88	
total	32661	11.14	>8386	

*>X means the tool aborted at this timing point and cannot give the result

Outline

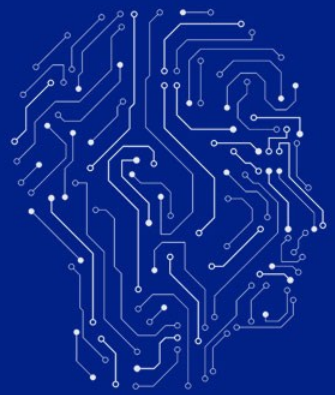
- A Coupling Area Reduction Technique Applying ODC Shifting
- Boosting Formal Verification by Macro Level Functional Checking
- Conclusion

Outline

- Difficulty of SAT in Comparing Arithmetic Logic
- Formal Verification by Macro Level Function Checking
- Experimental Results
- Conclusion

Conclusion

- We experiment a new reverse engineering and logic synthesis assisted verification methodology.
- Complicated arithmetic logics and their formulae are extracted to create internal equivalence for SAT solvers to avoid being trapped in certain exponential runs.
- This approach is orders of magnitude faster than any other known approach.
- It would be interesting to study if this Complementary Greedy Coupling scheme can also be useful for other NP-complete problems.



Thank You