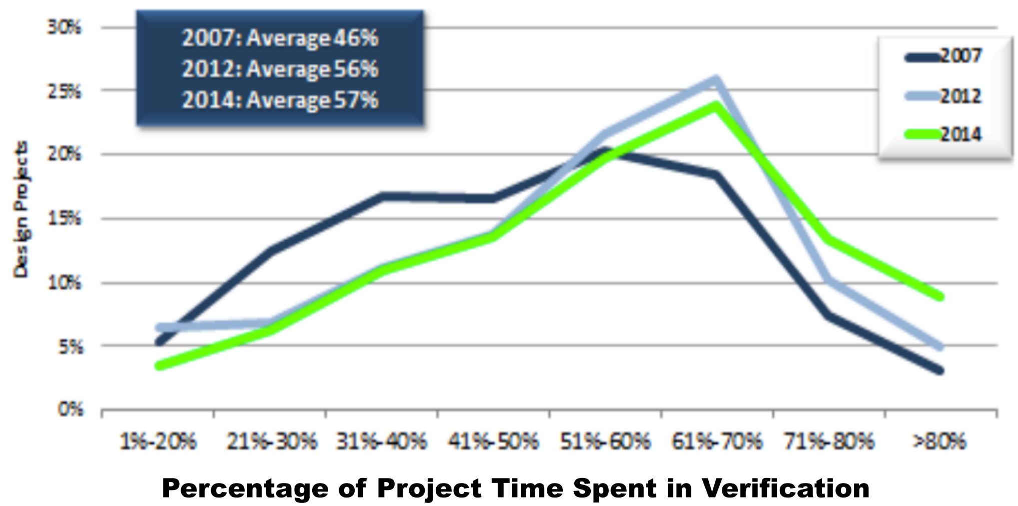
UMassAmherst

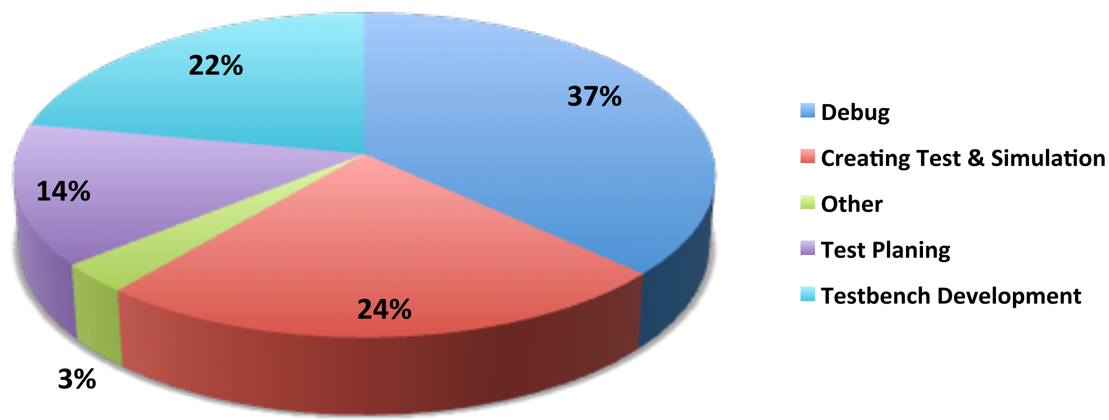
Efficient Parallel Verification of Galois Field Multipliers

Cunxi Yu, Maciej Ciesielski
ECE Department
University of Massachusetts, Amherst

Why *Research* on *Verification* ?

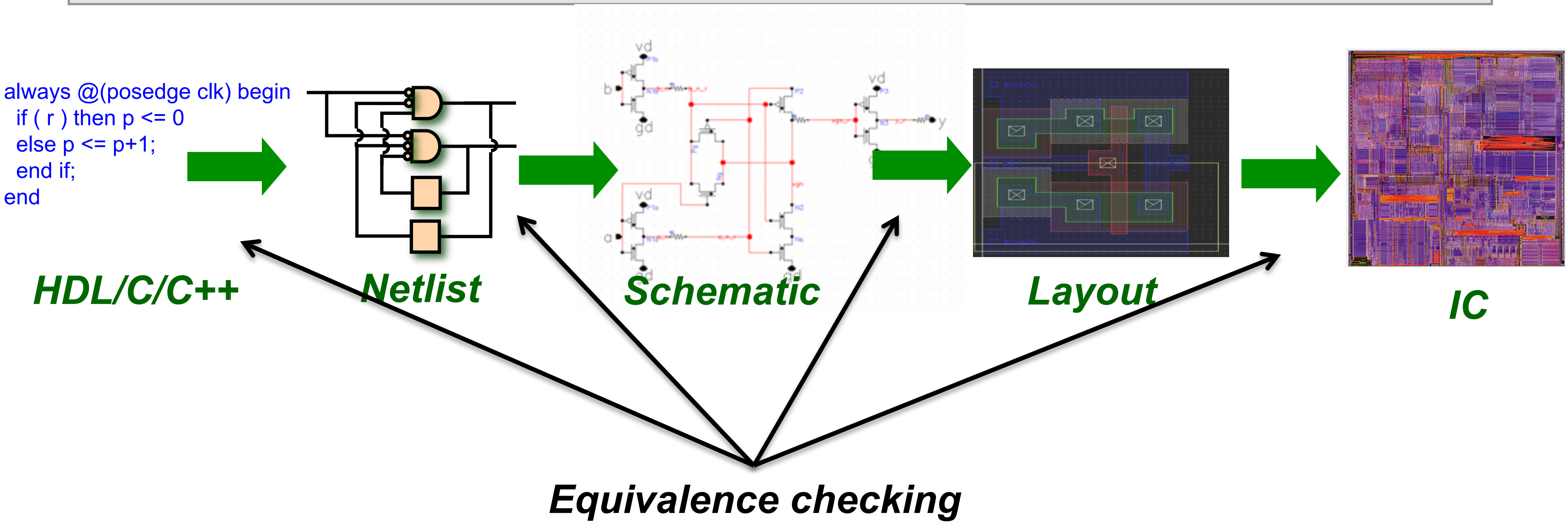
- ❑ Verification cost
  - 57% in 2014
    - ¼ designs → 61-70%
  - Increasing
- ❑ Verification works
  - Debugging
  - Test bench
  - Test planning





Harry D. Foster. "Trends in functional verification: A 2014 industry study". DAC'15.

Hardware Verification



- We focus on logical implementation
  - <u>Gate-level Galois Field Arithmetic Circuits</u>
    - Pre-synthesized and post-synthesized multipliers
    - Including *Montgomery* and *Mastrovito* Multipliers

Galois Field

- Finite Fields
  - Number system with a finite number of elements
    - Crytopgraphy systems, e.g. *Advanced Encryption Standard (AES)*
  - Prime field
    - GF(p) finite number of integers {1, 2, ..., p−1} , p is prime number
  - Extension field
    - A={a0,a1} in GF (2^2), is $A(x)=a_0+a_1x$, $a_i \in \{0,1\}$
- Example
  - 2-bit integer multiplication: $r_0+2r_1+4r_2+8r_3$
  - GF(2^2), irreducible poly $P(x)=x^2+x+1$
    - Many P(x) exist in GF(2^n) (n>=4)

		a_1	a_0
		b_1	b_0
		a_1b_0	a_0b_0
	a_1b_1	a_0b_1	
r_3	r_2	r_1	r_0

	a_1	a_0
	b_1	b_0
	a_1b_0	a_0b_0
a_1b_1	a_0b_1	
s_2	s_1	s_0
	s_2	s_2
	z_1	z_0

Introduction

❑ Hardware verification

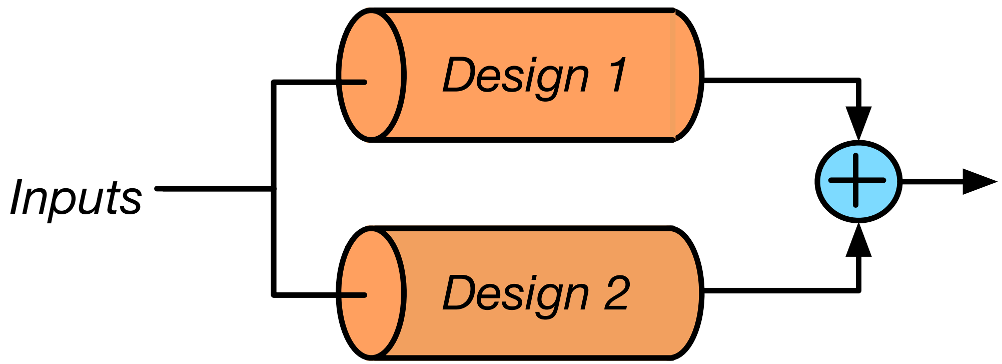
- Checking if the design meets specification
  - Equivalence checking (*EC*)
  - Property, model checking
  - Functional verification

❑ Verification Techniques

- Canonical diagrams (*BDDs, BMDs*), *SAT/SMT*
  - Require "bit-blasting", memory explosion
- Theorem proving (*ACL2, HOL*)
  - Requires domain knowledge, complex for gate-level
- Computer algebraic
  - Finite field arithmetic [Lvov'FMCAD11][Kalla'DAC14, TCAD'13]
  - Integer arithmetic [DAC'15] [TCAD'16]
  - Floating point arithmetic [Drechsler'FMCAD16]

Equivalence Checking (*EC*)

- A method to check two behavior equivalence



- <u>Combinational</u> Equivalence checking (*CEC*)
  - Exhaustive simulation
  - Canonical methods, e.g. *BDDs, BMDs, TEDs*
    - *Poor scalability*
  - Solve Boolean *Satisfiability* using *SAT/SMT/ILP* solvers
    - Build a "miter"; check if the "*miter*" is *unSAT*
    - Build *a pseudo-Boolean "miter" in SMT/ILP*

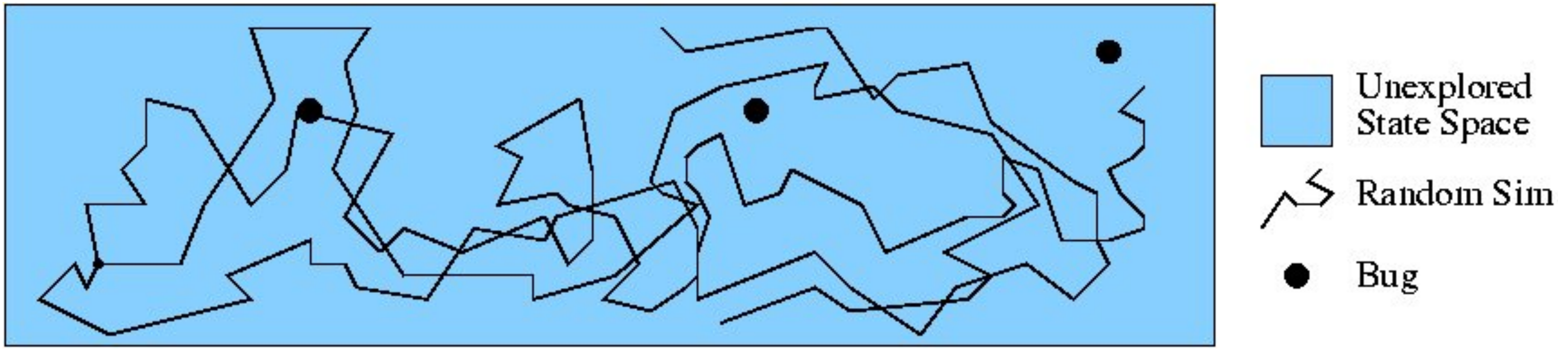
Simulation

- A "random walk" through the state space of the design
  - Test bench

+ Scalable: applicable to designs of any size

+ **Very** robust set of tools & methodologies available for this technique
  + Constraint-based stimulus generation; random biasing
  + Clever testcase generation techniques

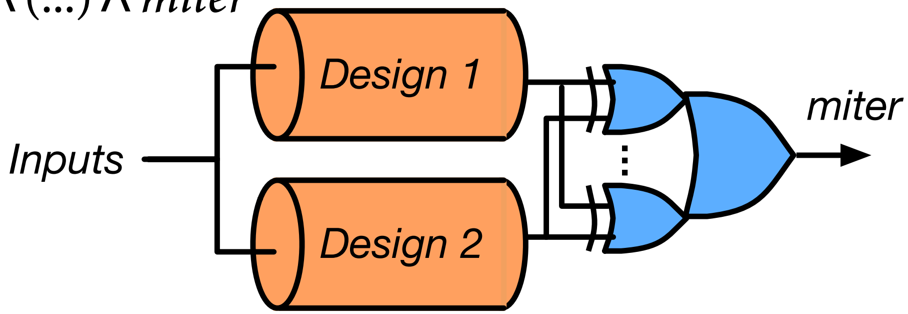
– Explicit one-state-at-a-time nature *severely limits attainable coverage*
  – Suffers from incomplete **coverage problem**: often fails to expose



Slide from Jason Baumgartner, IBM Austin, 2011

Boolean *Satisfiability* using SAT/SMT

- Check whether the *miter* is *satisfiable*
  - Specifically: $(clause_1) \wedge (clause_2) \wedge (...) \wedge miter$
    - SAT solvers: *miniSAT*, etc.

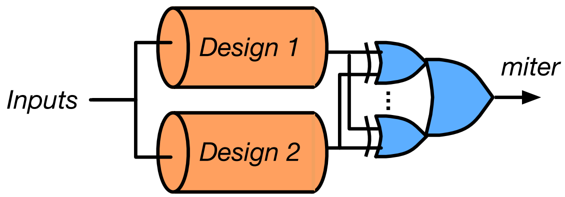


- Convert a netlist to *Conjunction Normal Format* (CNF)
  - AND: $(a \vee \neg x) \wedge (b \vee \neg x) \wedge (\neg a \vee \neg b \vee x)$
  - OR : $(\neg x \vee out) \wedge (\neg c \vee out) \wedge (x \vee c \vee \neg out)$

- Performance
  - More scalable than BDD/*BMD
  - Exponential runtime for hard problem

Evaluation of BDD/SAT/SMT/ABC

- Evaluation of existing formal methods [Kalla'TCAD13]
  - *SAT*: MiniSAT, CrytoSAT, PicoSAT
  - *SMT*: Yices, Beaver, CVC4, Z3, Boolector
  - *BDD*: CUDD Package
  - *ABC*

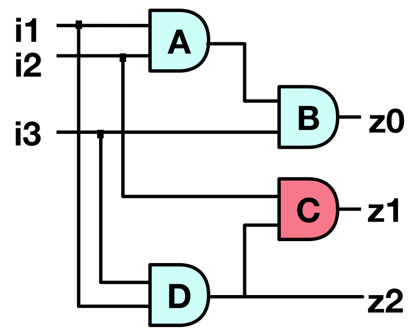


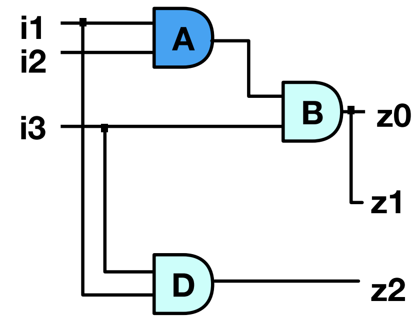
	Word size of the operands k-bits		
Solver	8	12	16
MiniSAT	22.55	TO	TO
CryptoMiniSAT	7.17	16082.40	TO
PicoSAT	14.85	TO	TO
Yices	10.48	TO	TO
Beaver	6.31	TO	TO
CVC	TO	TO	TO
Z3	85.46	TO	TO
Boolector	5.03	TO	TO
ABC	242.78	TO	TO
BDD	0.10	14.14	1899.69

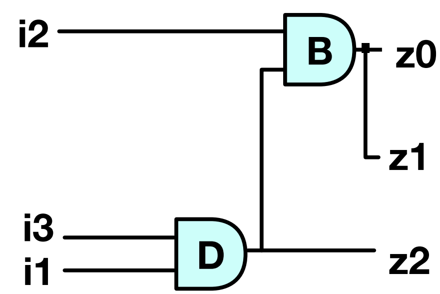
TO = timeout of 10 h.

Transformation-based Verification

- Complexity reduction
  - Redundancy removal
  - Combinational rewriting
    - And-Inv-Graph (AIG) [11]







- Example: Mastrovito Mult [Kalla'TCAD13]
  - *FRAIG* – Functional reduced AIG
    - *Miter* of two multipliers
      - Ideally should be reduced to an empty AIG
    - Percentage of AIG nodes eliminated before/after *FRAIG*

Size k	8	16	32	64	96	128	163
N_1	218	832	2226	7412	15 576	26 422	42 273
N_2	198	756	2160	7232	15 384	26 098	41 947
Similarity	9.17%	9.13%	2.96%	2.42%	1.23%	1.23%	0.77%

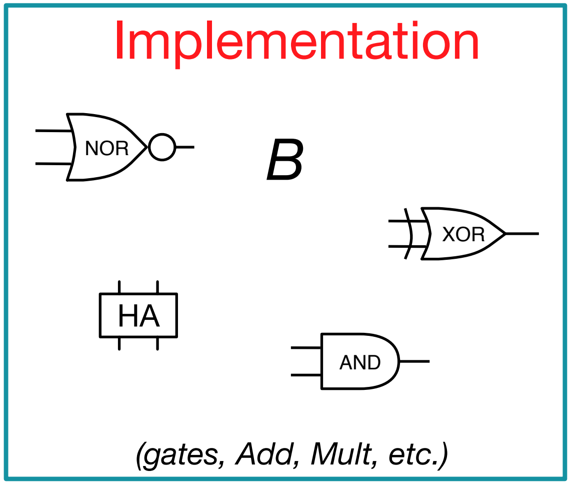
N_1, N_2 are the number of nodes counted before and after structural hashing, respectively.

Computer Algebraic method

- **Computer Algebra method** [Wienand'08, Pavlenko'11, Kalla'13, Drechsler'16]
  - Circuit represented in *arithmetic bit level* *(ABL)*
    - Specification F_{spec} and implementation B defined as polynomials in Z_{2^n}
    - Reduce F_{spec} modulo B by *polynomial divisions*

$$F_{spec} \xrightarrow{B}_{+} r$$

Specification F_{spec}



  - If $r = 0$, the circuit is correct
- **Algebraic Techniques**
  - Polynomial divisions: to check if r = 0
    - Otherwise, determine if *r is 0-polynomial using* canonical *Groebner basis*
  - Algebraic rewriting
    - Rewriting the signature based on a topological order of the network [DAC'15]

Previous Work

- Replace gate output by its equation
  - Substitution
    - Replace variables using algebraic model
  - Simplification
    - Eliminate monomials with coefficients “zero”
  - Must rewrite entire Signature

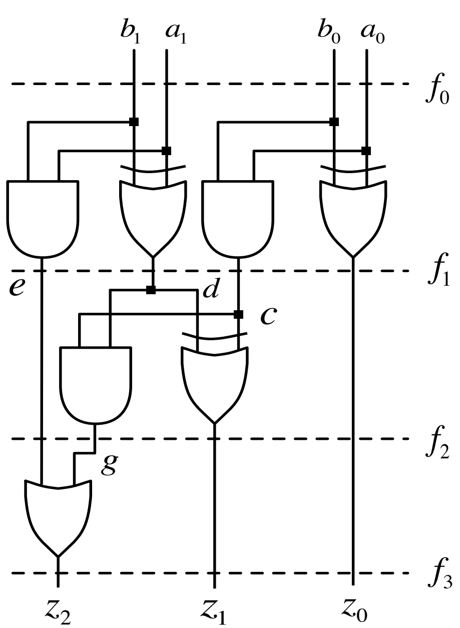
$$f_3 = 4z_2+2z_1+z_0$$

$$f_2=4(g + e - eg) + 2z_1+ z_0$$
$$=4g + 4e - 4eg+2z_1 + z_0$$

$$f_1 = 4e + 4(cd) - 4e(cd) + 2(c + d - 2cd)+z_0$$
$$= 4e + 2c +2d + z_0 - 4ecd$$

$$f_0 = 4(a_1b_1) + 2(a_0b_0) + 2(a_1+ b_1 - 2a_1b_1) + (a_0 + b_0 - 2a_0b_0) - 4(a_1b_1) (a_0b_0) (a_1 + b_1 -2a_1b_1)$$

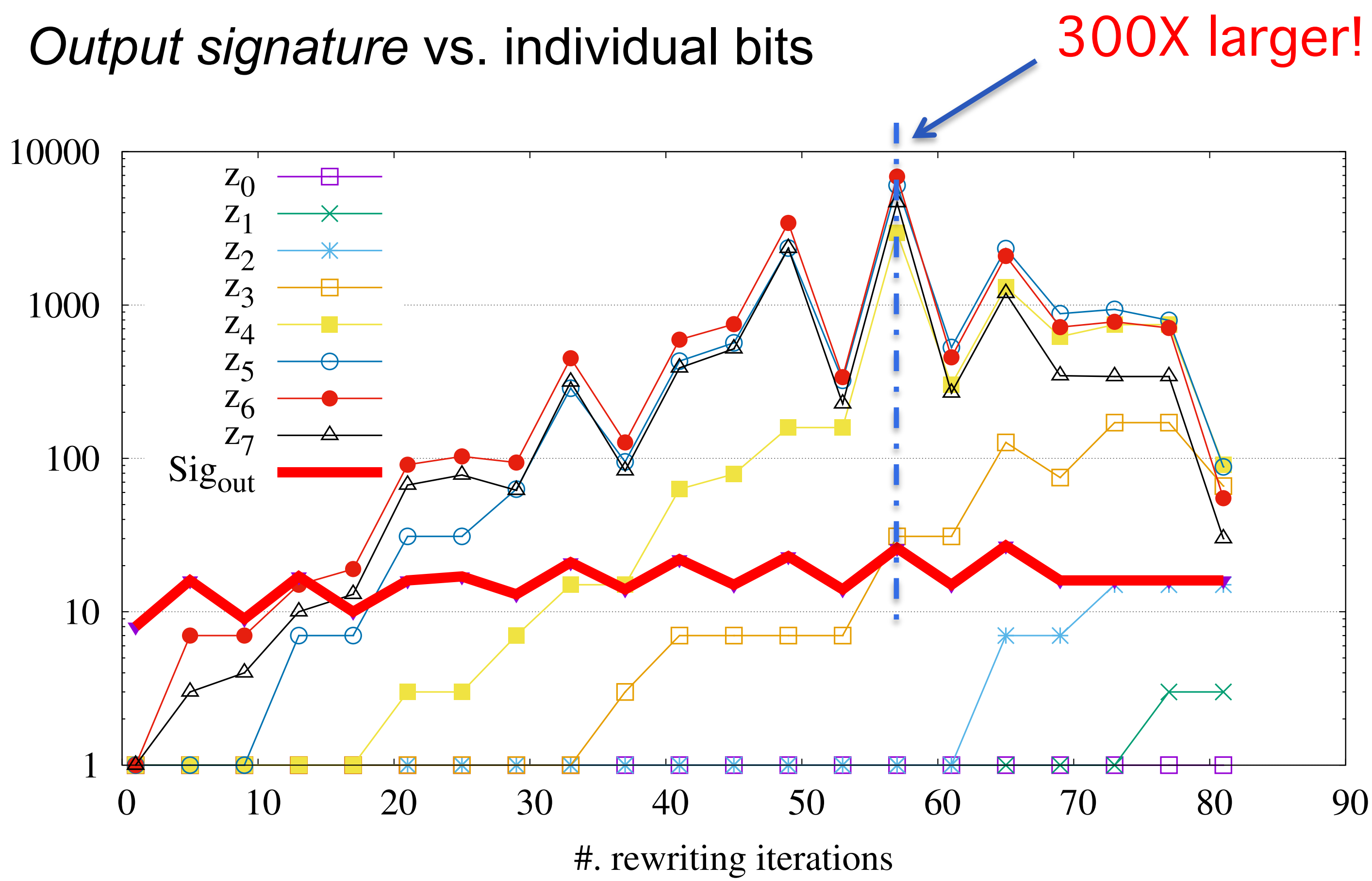
$$= 2a_1+ 2b_1 + a_0 + b_0$$



Matches the *input signature*. Circuit is correct.

Previous Work

- Expression reduction: 4-bit multiplier
  - Large number of reductions between each output bit
  - *Output signature* vs. individual bits



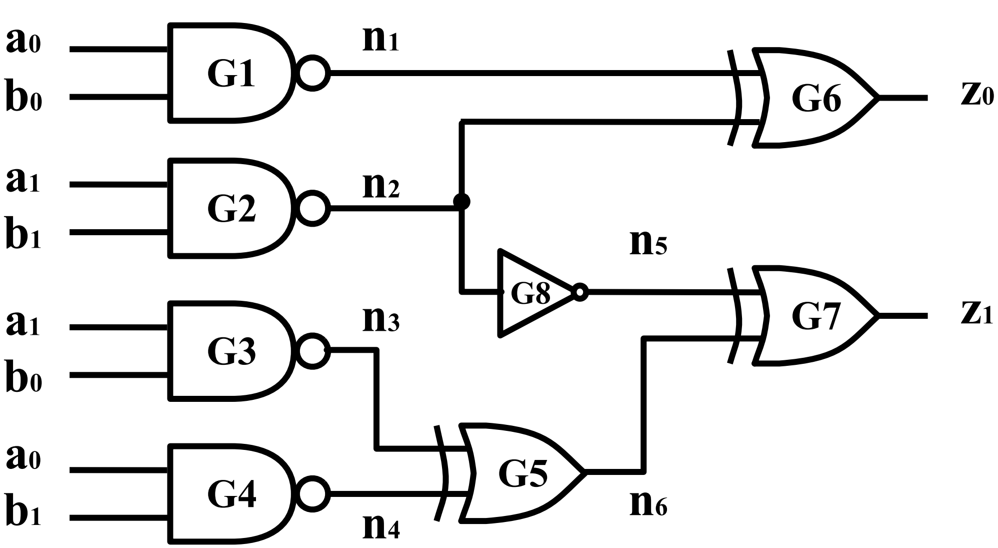
Verification of GF Multipliers

- Finite field multiplier
  - Function: *A(x)*B(x) mod P(x)*
  - Irredundant polynomial: $P(x) = x^2+x+1$
    - *equals to A*B mod 7*
- Example: 2-bit GF Multiplier
  - $P(x) = x^2+x+1$
    - $s_0 = a_0b_0$
    - $s_1 = a_1b_0 \oplus a_0b_1$
    - $s_2 = a_1b_1$
    - $z_0 = s_0 \oplus s_1$
    - $z_1 = s_1 \oplus s_2$
  - $z_0 = a_0b_0 \oplus a_1b_0 \oplus a_0b_1$
  - $z_1 = a_1b_0 \oplus a_0b_1 \oplus a_1b_1$

	a_1	a_0
	b_1	b_0
	a_1b_0	a_0b_0
a_1b_1	a_0b_1	
s_2	s_1	s_0
	s_2	s_2
	z_1	z_0

Verification of GF(2^m) Multipliers

- Finite field multiplier
  - Function: *A(x)*B(x) mod P(x)*
  - Irredundant polynomial: $P(x) = x^2+x+1$
    - *equals to A*B mod 7*
- Modeling in finite field
  - Post-synthesized 2-bit GF multiplier



input signature:

$$A = x^1 a_0 + x^2 a_1$$

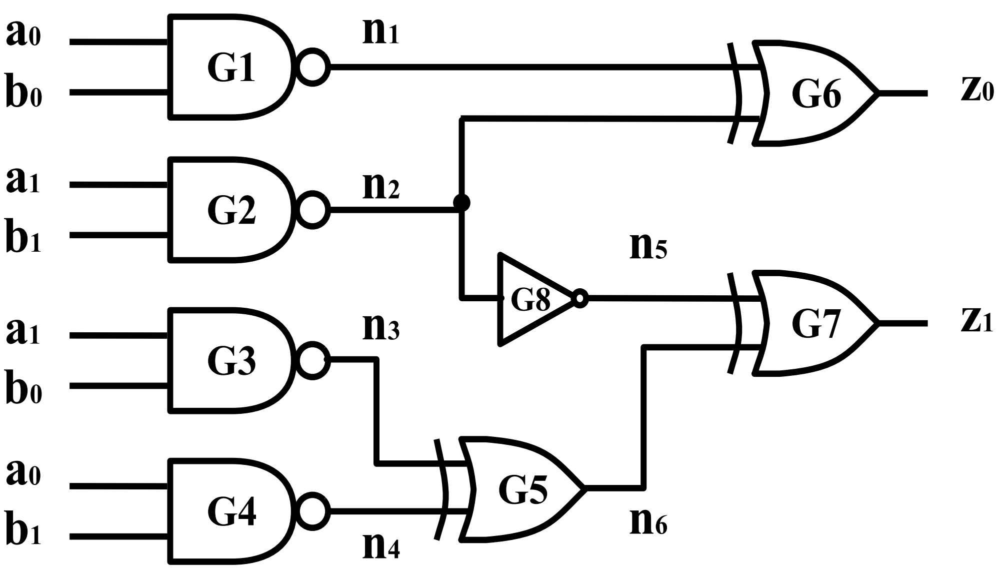
$$B = x^1 b_0 + x^2 b_1$$

output signature

$$Z = x^1 z_0 + x^2 z_1 \mod P(x)$$

Verification of GF(2^m) Multipliers

- 2-bit GF(2^2) multiplier
  - Irredundant polynomial: $P(x) = x^2+x+1$
  - Function: $Z = z_0 + z_1 * x$
    - $z_0 = a_0b_0 \oplus a_1b_0 \oplus a_0b_1$
    - $z_1 = a_1b_0 \oplus a_0b_1 \oplus a_1b_1$
- Modeling in finite field



$$B \left\{ \begin{aligned} G1 &: n_1 = 1 + a_0b_0 \\ G2 &: n_2 = 1 + a_1b_1 \\ G3 &: n_3 = 1 + a_1b_0 \\ G4 &: n_4 = 1 + a_0b_1 \\ G5 &: n_6 = n_3 + n_4 \\ G6 &: z_0 = n_1 + n_2 \\ G7 &: z_1 = n_5 + n_6 \\ G8 &: n_5 = 1 + n_2 \end{aligned} \right.$$

Verification of GF(2^m) Multipliers

- 2-bit GF(2^2) multiplier
  - Irredundant polynomial: $P(x) = x^2+x+1$
  - Function: $Z = z_0 + z_1 * x$
    - $z_0 = a_0b_0 \oplus a_1b_0 \oplus a_0b_1$
    - $z_1 = a_1b_0 \oplus a_0b_1 \oplus a_1b_1$
- Modeling in finite field
  - Each rewriting result ($F_0, F_1, \ldots F_i \in GF(2^m)$)
  - Theorem 1: Algebraic model $\in GF(2)$

$$\neg a = 1 - a$$
$$a \wedge b = a \cdot b$$
$$a \vee b = a + b - a \cdot b$$
$$a \oplus b = a + b - 2a \cdot b$$

mod 2 →

$$\neg a = (1 + a) \bmod 2$$
$$a \wedge b = a \cdot b$$
$$a \vee b = (a + b + a \cdot b) \bmod 2$$
$$a \oplus b = (a + b) \bmod 2$$

Verification of GF(2^m) Multipliers

- 2-bit GF(2^2) multiplier
  - Irredundant polynomial: $P(x) = x^2+x+1$
  - Function: $Z = z_0 + z_1 * x$ → $F_{spec} = a_0b_0+a_1b_1+(a_1b_1+a_1b_0+a_0b_1)*x$
    - $z_0 = a_0b_0 \oplus a_1b_0 \oplus a_0b_1$
    - $z_1 = a_1b_0 \oplus a_0b_1 \oplus a_1b_1$
- Modeling in finite field
  - Each rewriting result ($F_0, F_1, \ldots F_i \in GF(2^m)$)
  - Theorem 1: Algebraic model $\in GF(2)$

$$\neg a = 1 - a$$
$$a \wedge b = a \cdot b$$
$$a \vee b = a + b - a \cdot b$$
$$a \oplus b = a + b - 2a \cdot b$$

mod 2 →

$$\neg a = (1 + a) \bmod 2$$
$$a \wedge b = a \cdot b$$
$$a \vee b = (a + b + a \cdot b) \bmod 2$$
$$a \oplus b = (a + b) \bmod 2$$

Verification of GF(2^m) Multipliers

- Finite field multiplier
  - Function: *A(x)*B(x) mod P(x)*
  - Irredundant polynomial: $P(x) = x^2+x+1$
    - *equals to A*B mod 7*
- Modeling in finite field
  - Each rewriting result ($F_0, F_1, \ldots F_i \in GF(2^m)$)
  - Theorem 1: Algebraic model $\in$ GF(2)
  - Theorem 2: Coefficients of each monomial $\in$ GF(2)
    - Provides eliminations/polynomial reductions

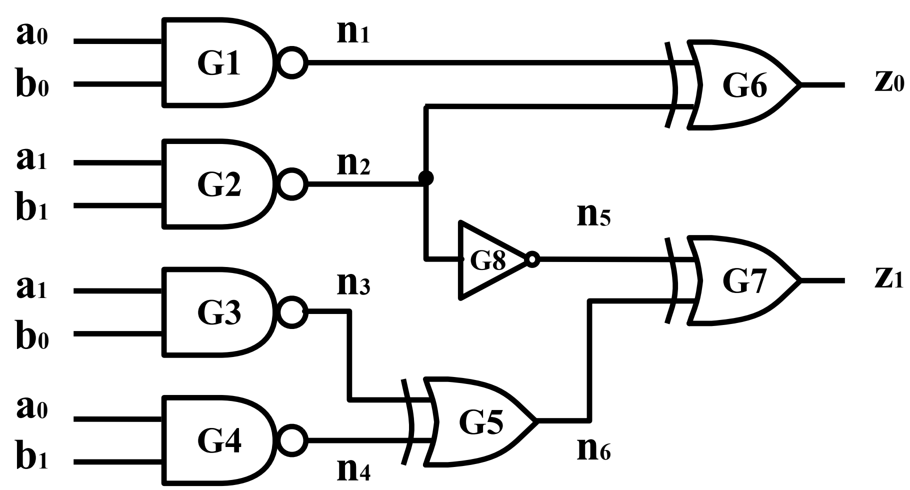
$$\neg a = 1 - a$$
$$a \wedge b = a \cdot b$$
$$a \vee b = a + b - a \cdot b$$
$$a \oplus b = a + b - 2a \cdot b$$

mod 2

$$\neg a = (1 + a) \bmod 2$$
$$a \wedge b = a \cdot b$$
$$a \vee b = (a + b + a \cdot b) \bmod 2$$
$$a \oplus b = (a + b) \bmod 2$$

Verification of GF(2^m) Multipliers

- Single-thread verification
  - Order = <7,6,5,8,4,3,2,1>



Sig_{out}: $F_0 = z_0+z_1*x$

G7: $F_1 = z_0+(n_5+n_6)*x$

G6: $F_2 = n_1+n_2+(n_5+n_6)*x$

G5: $F_3 = n_1+n_2+(n_3+n_4+n_5)*x$

G8: $F_4 = n_1+n_2+(n_3+n_4+n_2+1)*x$

G4: $F_5 = n_1+n_2+(n_2+n_3+a_0b_1)*x + 2x$

G3: $F_6 = n_1+n_2+(n_2+a_1b_0+a_0b_1)*x + x$

G2: $F_7 = n_1+a_1b_1+1+(a_1b_1+a_1b_0+a_0b_1)*x +2x$

G1: $F_8 = a_0b_0+a_1b_1+(a_1b_1+a_1b_0+a_0b_1)*x +2$

Sig_{in} = $F_9 = a_0b_0+a_1b_1+(a_1b_1+a_1b_0+a_0b_1)*x$

"+" is addition "add, mod 2"

Verification of GF(2^m) Multipliers

- Theorem 3: Reductions exist only within each output element

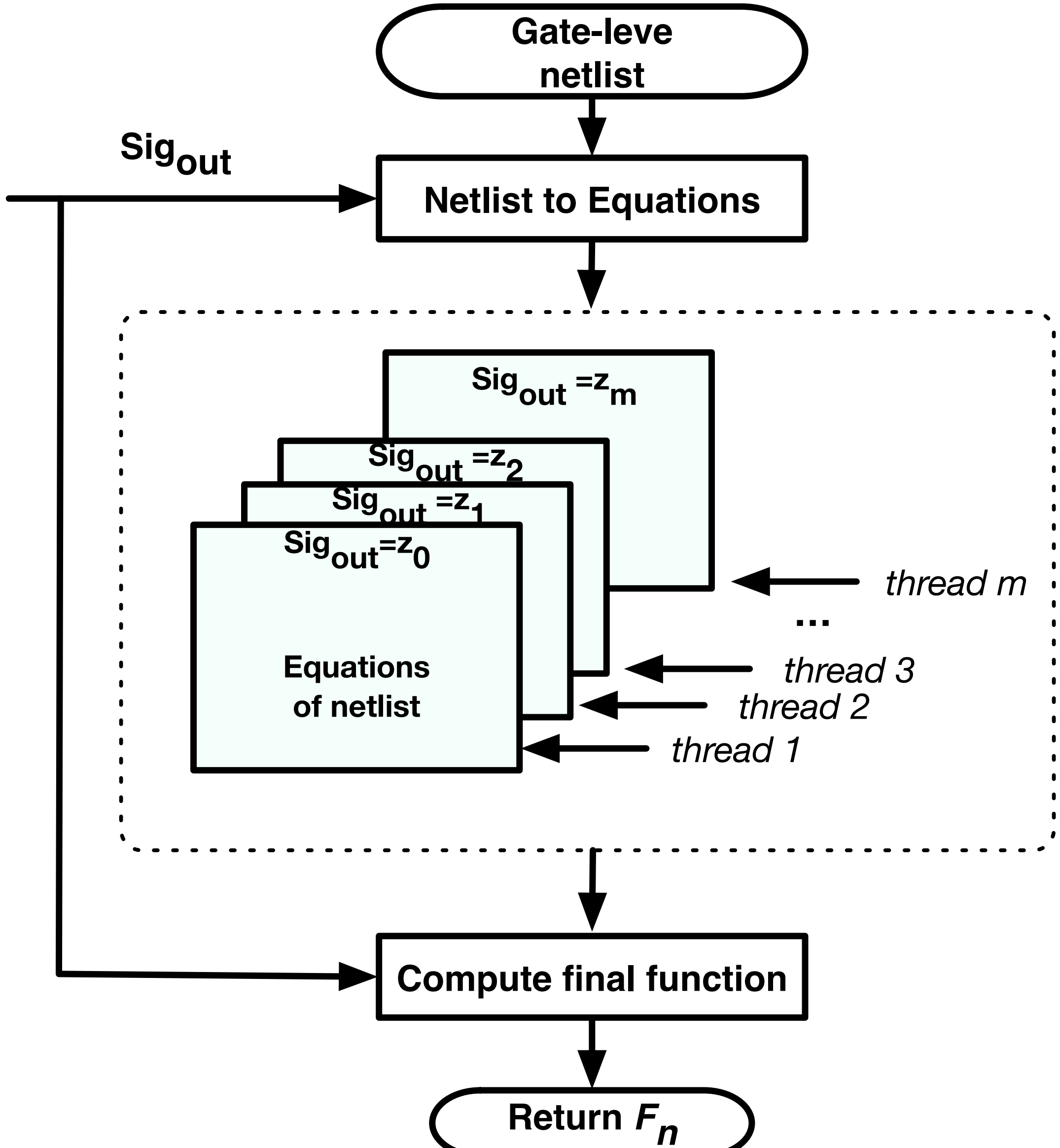
Theorem 3: *Given a* $GF(2^m)$ *multiplier with* $Sig_{out} = F_0 = z_0x^0 + z_1x^1 + ... + z_mx^m$; *and* $F_i{=}E_0x^0 + E_1x^1 + ... + E_mx^m$, *where* E_i *is an algebraic expression in* $GF(2)$. *Then, the polynomial reduction is possible only within the expression* E_i.

Sig_{out0}=z_0	elim	Sig_{out1}=x·z_1	elim
G7: z_0	-	G7: x(n_5+n_6)	-
G6: n_1+n_2	-	G6: x(n_5+n_6)	-
G5: n_1+n_2	-	G5: x(n_3+n_4+n_5)	-
G8: n_1+n_2	-	G8: x(n_3+n_4+n_2)+x	-
G4: n_1+n_2	-	G4: x(n_2+n_3+a_0a_1)+2x	2x
G3: n_1+n_2	-	G3: x(n_2+a_1b_0+a_0b_1)+x	-
G2: n_1+a_1b_1+1	-	G2: x(a_1b_1+a_1b_0+a_0b_1)+2x	2x
G1: a_0b_0+a_1b_1+2	2	G1: x(a_1b_1+a_1b_0+a_0b_1)	-
Sig_{in}=a_0b_0+a_1b_1+x(a_1b_1+a_1b_0+a_0b_1)			

Parallel Verification Flow

m-threads

for GF(2^m)



Results

- Results compared to [Tim'DAC14]
  - Mastrovito
    - 32- to 571-bit, avg **43x** speedup T=20
  - Montgomery multipliers
    - 32- to 283-bit, avg **16x** speedup T=20
  - Other solvers (SAT,SMT) time out at 32-bit

Montgomery		[5]		This work				
Op size	# equations	Runtime (sec)	Mem (MB)	Runtime (sec)				Mem*
				T=5	*T=10*	*T=20*	*T=30*	*T=1**
32	4,352	1.98	3	3.49	2.16	1.31	2.08	8 MB
48	9,602	14.19	13	17.71	10.67	9.16	6.01	16 MB
64	16.898	63.48	21	44.86	30.57	28.3	27.22	27 MB
96	37,634	554.6	45	234.3	157.8	133.1	142.3	59 MB
128	66,562	1924	68	208.9	121.3	115.8	110.4	95 MB
163	107,582	12063	101	1615.7	1172.3	1094.9	1008.1	161 MB
233	219,022	*TO*	168	722.3	564.8	457.7	479.8	301 MB
283	322,622	*TO*	380	19745	17640	15300	14820	488 MB

Results

- Complexity depends on irreducible poly P(x)
  - P(x) = x^4+x^3+1, XOR operations = 3+1+2+3=9
  - P(x) = x^4+x+1, XOR operations = 1+2+2+1=6

			a_3	a_2	a_1	a_0
			b_3	b_2	b_1	b_0
			a_3b_0	a_2b_0	a_1b_0	a_0b_0
		a_3b_1	a_2b_1	a_1b_1	a_0b_1	
	a_3b_2	a_2b_2	a_1b_2	a_0b_2		
a_3b_3	a_2b_3	a_1b_3	a_0b_3			
s_6	s_5	s_4	s_3	s_2	s_1	s_0

$P(x)=x^4+x^3+1$

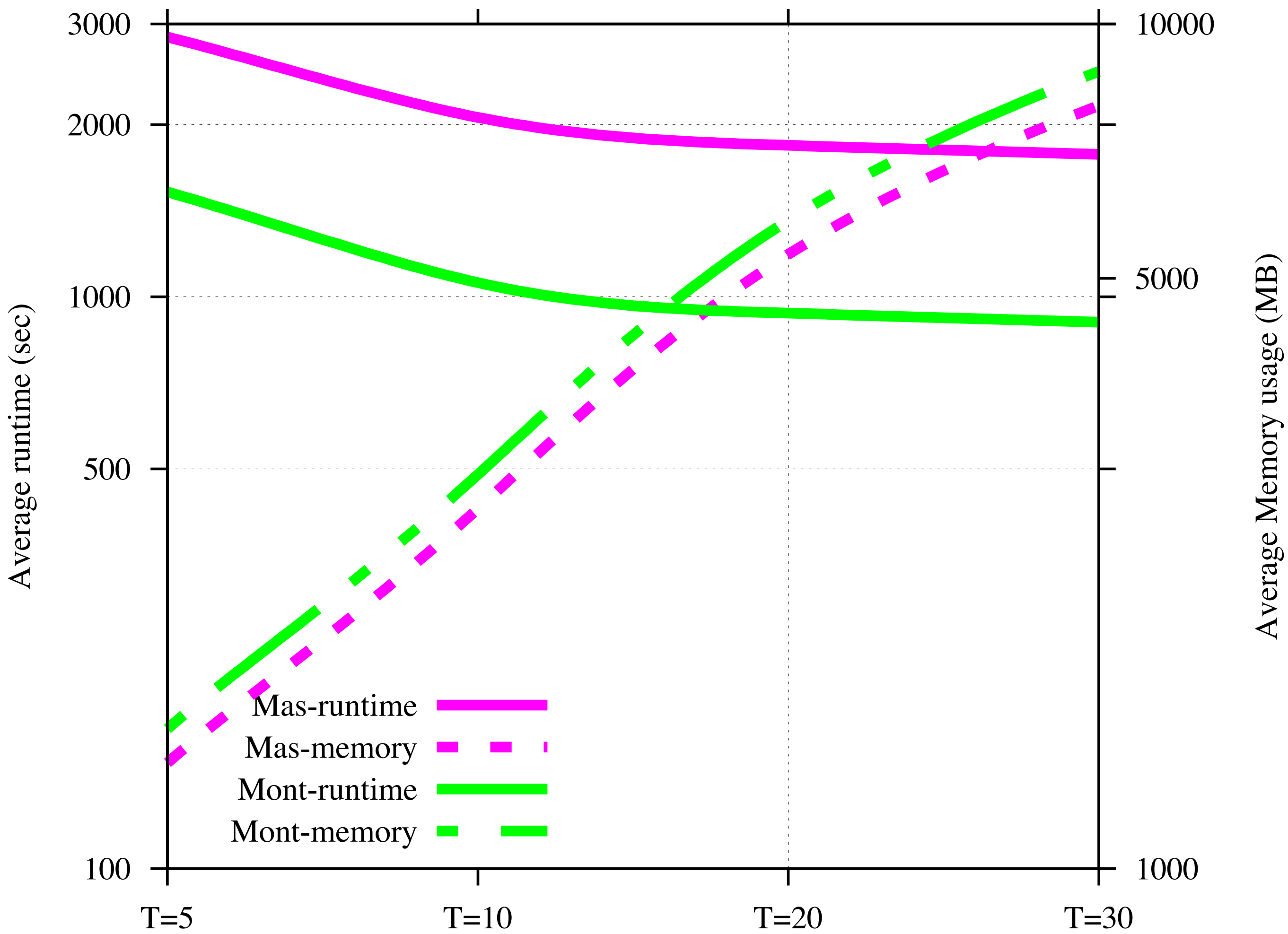
s_3	s_2	s_1	s_0
s_4	0	0	s_4
s_5	0	s_5	s_5
s_6	s_6	s_6	s_6
z_3	z_2	z_1	z_0

$P(x)=x^4+x+1$

s_3	s_2	s_1	s_0
0	0	s_4	s_4
0	s_5	s_5	0
s_6	s_6	0	0
z_3	z_2	z_1	z_0

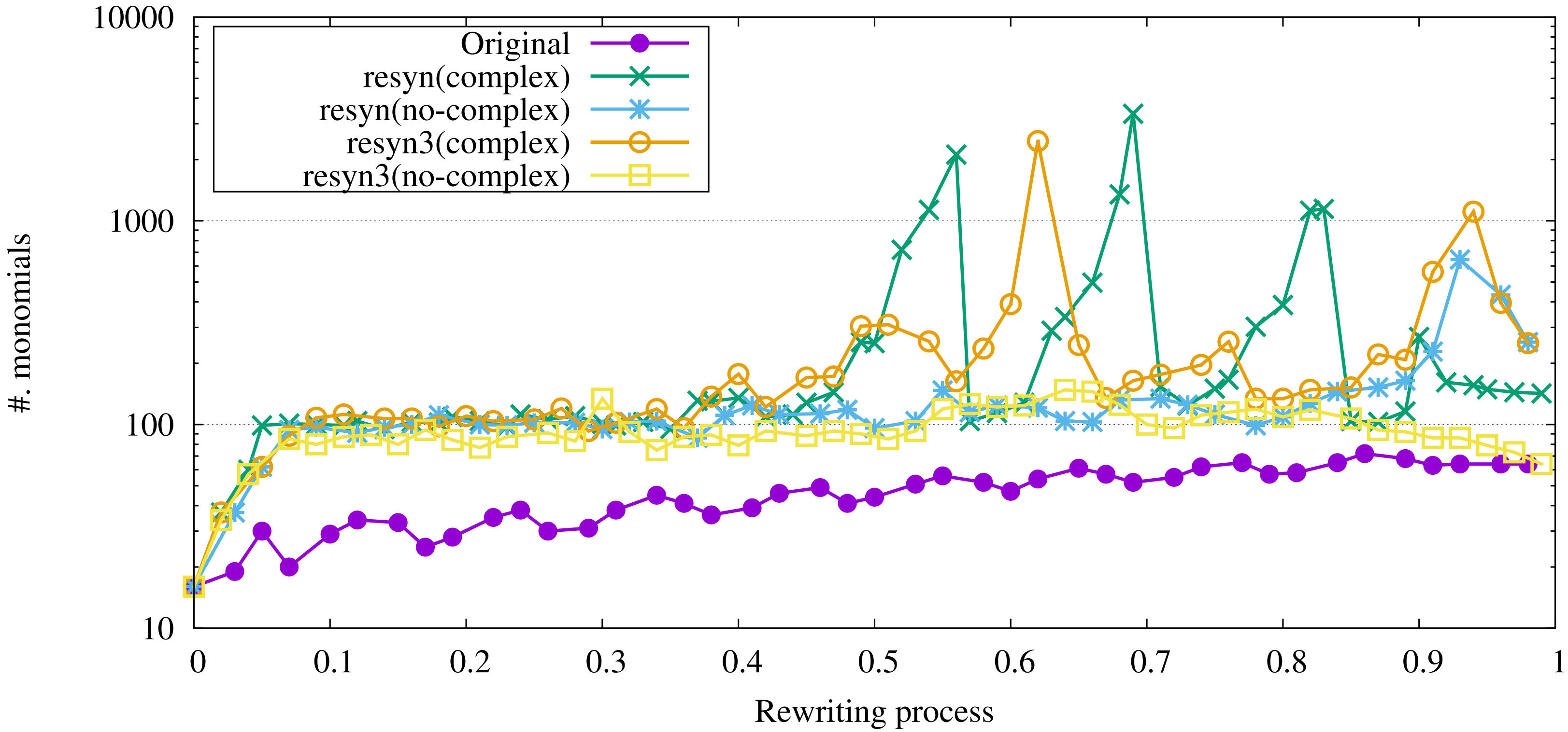
Performance of Parallel

- Memory vs. Runtime



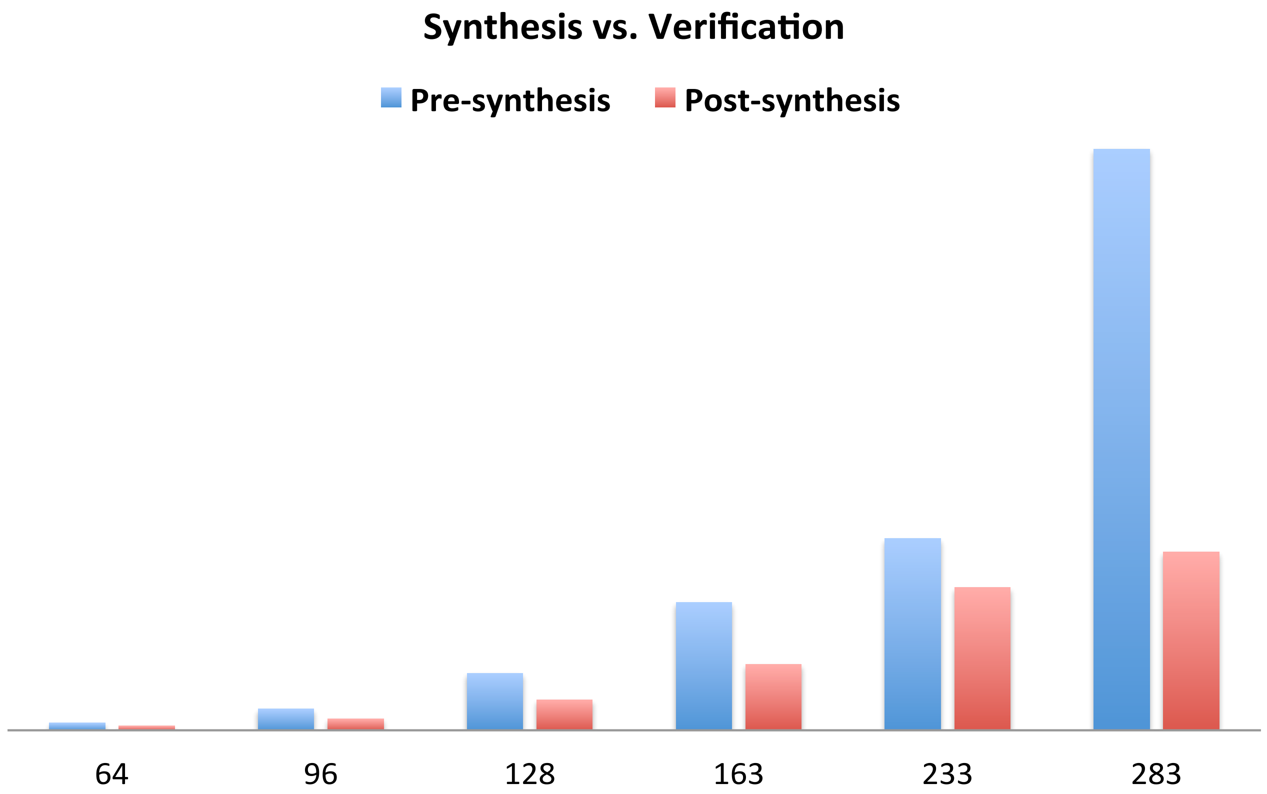
Synthesis vs. Verification

- Synthesis effect: 8-bit integer multiplier [TCAD'16]
  - Bit-optimization, technology mapping
  - Increases the verification complexity



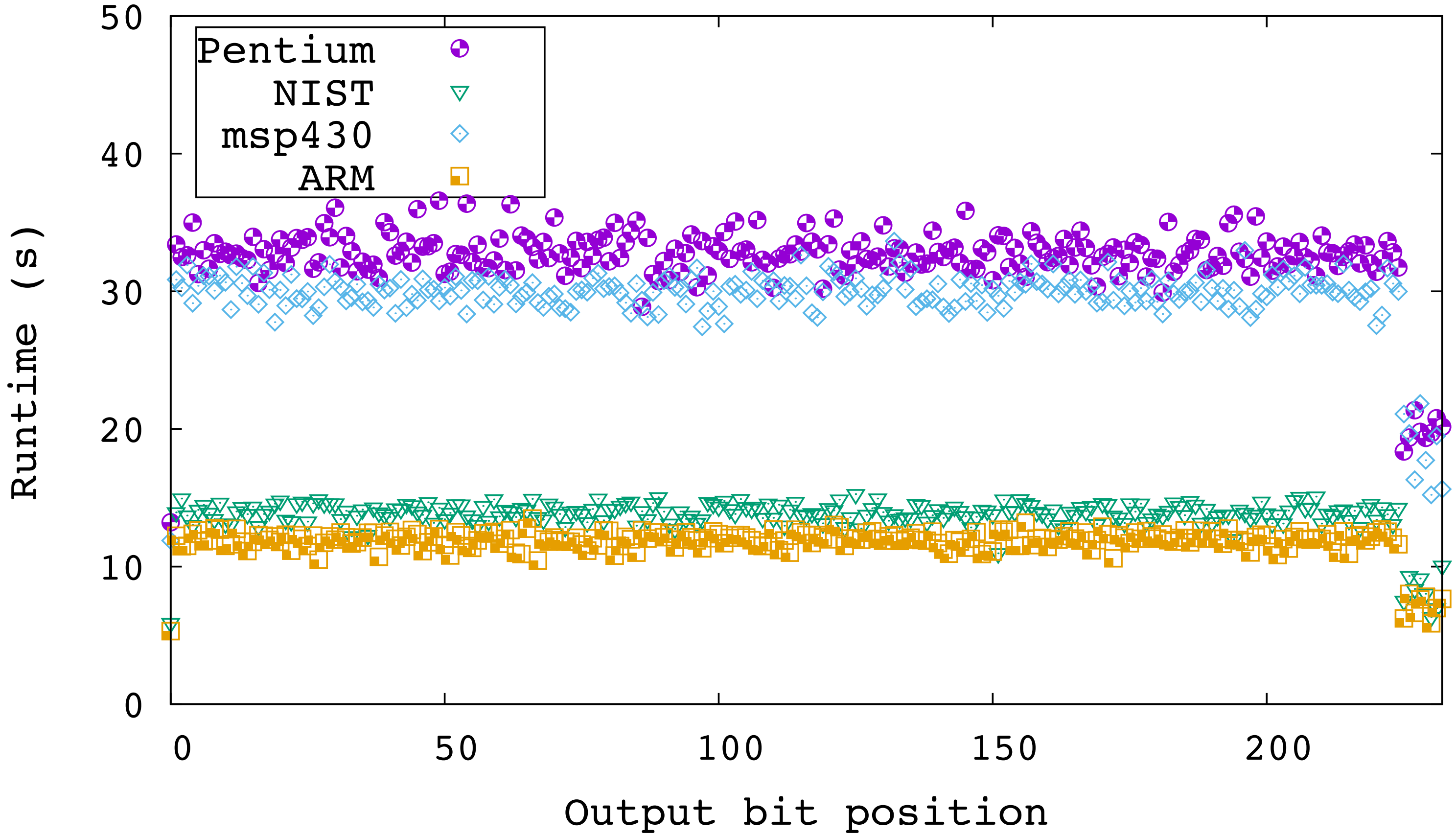
Synthesis vs. Verification

- Synthesis effect on $GF(2^m)$ multipliers
  - Bit-optimization, technology mapping
  - Decreases the verification complexity
    - Runtime comparison



Results

- Runtime of each output elements
  - GF(2^{233}) multipliers implemented using different P(x)



Thank you !