UMassAmherst

Verification of Gate-level Arithmetic Circuits by Function Extraction

Maciej Ciesielski, Cunxi Yu, Walter Brown, Duo Liu

University of Massachusetts, Amherst, USA

Andre Rossi

Université de Bretagne-Sud - Lab STICC, France

Introduction

- Hardware verification
  - Checking if the design meets specification
    - Equivalence checking
    - Property, model checking
    - Functional verification ←

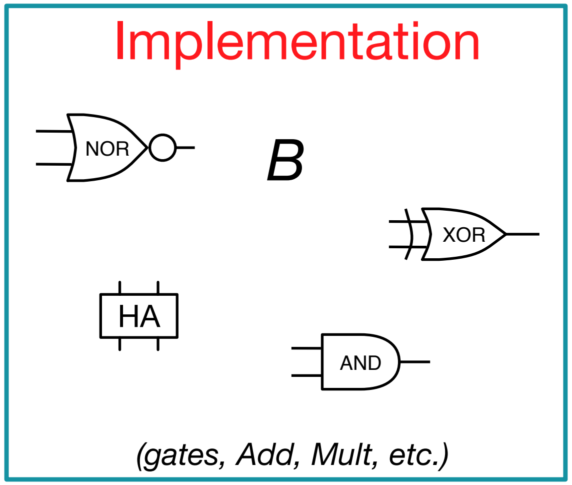
- Formal Verification methods
  - Canonical diagrams (BDD), SAT, SMT
    - Require "bit-blasting", memory explosion
  - Theorem proving
    - Requires good knowledge of the design, incomplete
  - Computer Algebra
    - Complex math, CPU runtime limitation

Related Work

- Computer Algebra method [Wienand'08, Pavlenko'11, Kalla'14]
  - 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*

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

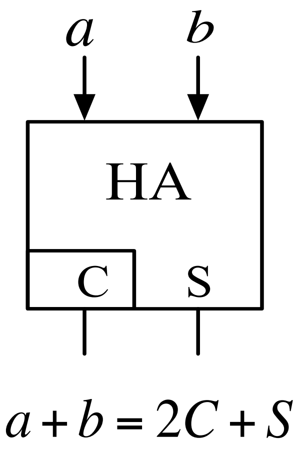
Specification F_{spec}



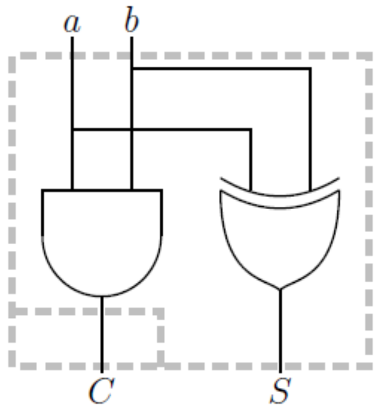
  - If $r = 0$, the circuit is correct
  - Otherwise, *canonical Groebner basis* is needed to determine if $r = 0$
    - Must model all signals as Boolean
      - include polynomials $<x^2-x>$ for all signals x
    - Complex math, unnecessary
  - Provided main motivation for our work

Previous Work

- **Linear Algebra Model** [Basith FMCAD'11, Ciesielski HVC'13]
  - Use linear algebraic model of the circuit
  - Circuit represented as network of adders (HA, FA)
    - described using only *linear equations*



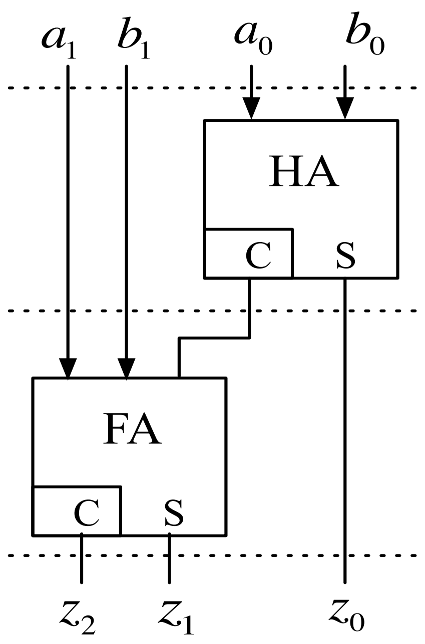
$$a + b = 2C + S$$



  - Logic gates derived from HAs (*linear*)

Linear Algebra Approach

- Input and Output Signatures



$$Sig_{in} = a_0 + 2a_1 + b_0 + 2b_1$$

Transform Sig_{in} into Sig_{out}

to prove that $Sig_{In} = Sig_{out}$

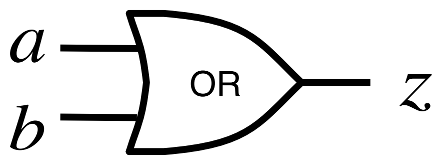
$$Sig_{out} = z_0 + 2z_1 + 4z_2$$

- Major limitations
  - Limited to linear network of HAs with few connecting gates
  - Translating gate-level to HA network (ABL) is difficult
  - Difficult to extend to non-linear circuits

This Work - Nonlinear Model

- **Algebraic Model**

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

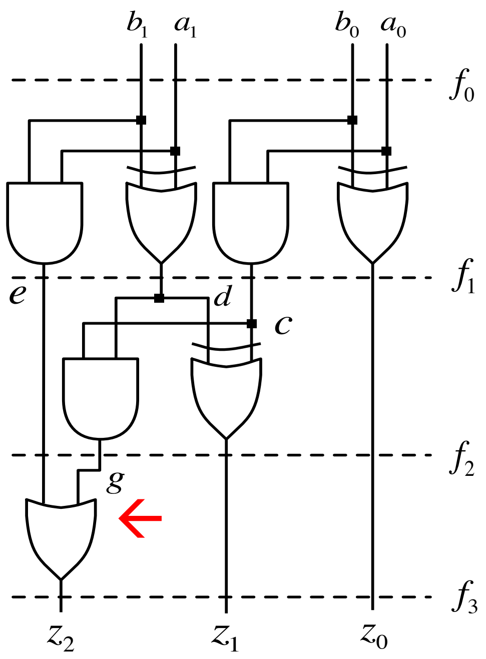


- Main idea : $F_{spec} = (Sig_{out} - Sig_{in}) \rightarrow 0 \bmod B$
  - replace by $Sig_{in} \rightarrow Sig_{out}$ (*forward rewriting*)
  - or by $Sig_{out} \rightarrow Sig_{in}$ *(backward rewriting)*

- Gate-level rewriting
  - Forward rewriting (*PI* → *PO*)
    - Polynomial division
    - Replacing input expression by output expression
  - <u>Backward rewriting </u>(PO → PI) ←
    - Replacing gate output by the expression in its inputs
      - e.g., OR gate : $z = a + b - ab$

Backward Rewriting - Method

- Example: 2-bit adder



$$Sig_{in} = (a_0 + 2a_1) + (b_0 + 2b_1)$$

Transform Sig_{out} into Sig_{in}

to prove that $Sig_{In} = Sig_{out}$

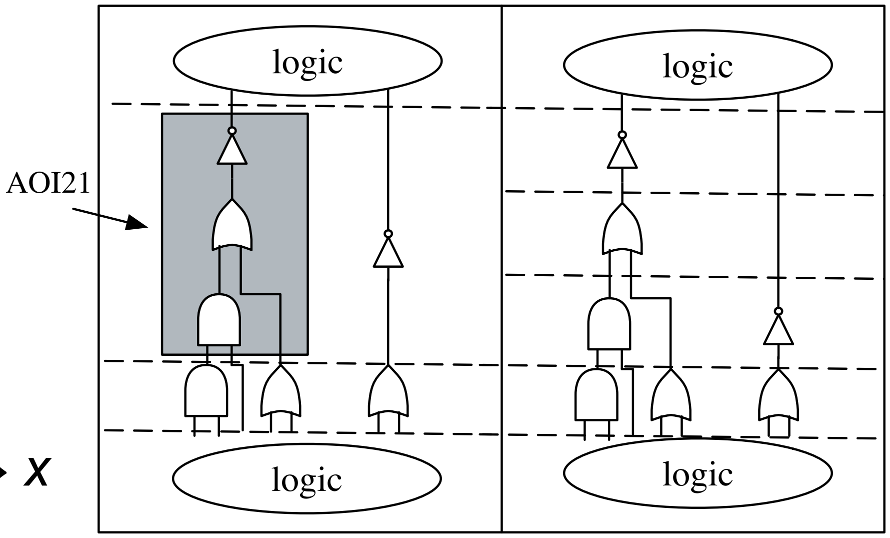
$$Sig_{out} = z_0 + 2z_1 + 4z_2$$

- Rewriting example $z_0 + 2z_1 + 4\underline{z_2}$
  - OR gate: $z_0 + 2z_1 + 4\underline{(e + g - eg)}$

Backward Rewriting - Issues

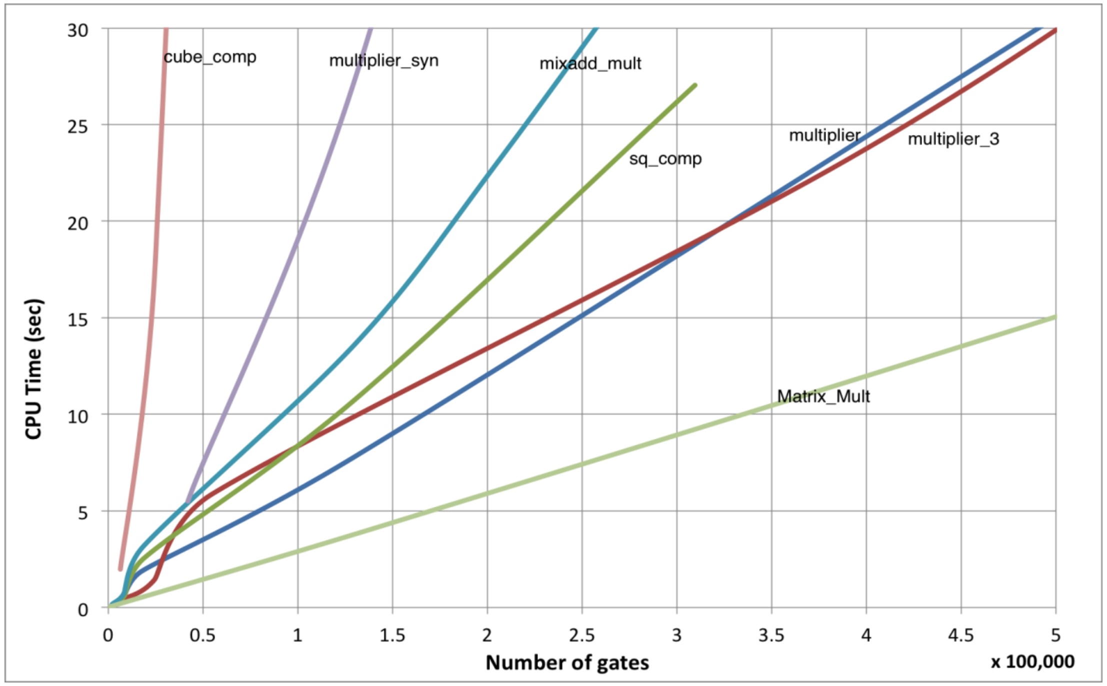
- Rewriting issues
  - Ordering of rewriting affects performance
  - Internal expressions may explode: "fat-belly" issue
  - Technology mapped circuits contain complex gates
    - *AOI21, OAI221*, etc.
  - Heavily optimized circuits are difficult

- Heuristics
  - Properly ordered rewriting
    - Topological, dependence
  - Handles complex gates
    - Provide cuts in complex gates
  - Binary signals modeled by $x^2 \rightarrow x$



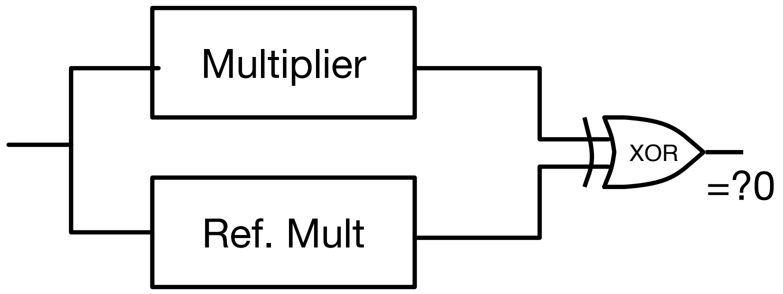
Results – CPU Performance

- Performance for original & lightly synthesized designs
  - Synthesis performed by ABC *"resyn"*
  - Verified designs
    - Multipliers, matrix multiplier, *A*B+C*, squarer, etc.
    - Up-to 5 million gates
    - 256+ bit-widths
  - ~Linear CPU time
  - Memory : quadratic in # gates



Comparison with Other Methods

- SAT
  - used ABC “miter”
- SMT
  - Algebraic model, similar to ours
  - Boolean model, similar to SAT
- Formality (Synopsys)



CSA-Multiplier		Our [sec]	SAT	SMT	Commercial
Op size	# gates		*Lingling*	*Boolector*	*Formality*
4	86	0.01	0.01	0.01	0.81
12	481	0.04	TIME OUT	2030.91	108.1
64	41.4 K	5.50	-	TIME OUT	675.4
128	164 K	39.64	-	-	TIME OUT

TIME OUT = AFTER 3600 SEC

Conclusions

- Functional verification by *extracting* arithmetic function
- Method: backward rewriting
  - Extracts the function of arithmetic circuits
  - Correctly model Boolean signals
  - More effective than forward, but complex
- Limitations
  - Less efficient for highly bit-optimized arithmetic circuits
  - Potential memory explosion
  - Bit composition of the output needs to be known
- Future work
  - Diagnostics and logic debugging (using back/forward rewriting)
  - Application to floating-point and cryptographic arithmetic circuits

Thank you !