Universität Bremen

AGRA

A Conflict-Aware Learning Approach to SCA Verification for MAC Architectures

Jan Kleinekathöfer, Lennart Weingarten, Kamalika Datta, Rolf Drechsler

ja_kl@uni-bremen.de

ARITH 2026

Bundesministerium für Forschung, Technologie und Raumfahrt

Gefördert durch DFG Deutsche Forschungsgemeinschaft

Motivation

Formal Verification & Correctness

Ensures 100% functional correctness, crucial for avoiding expensive real-world defects like the **Pentium BUG**.

The Challenge

Complex arithmetic (Multiplication, MAC) is challenging for traditional formal methods:

- **SAT Solvers:** Struggle with propositional logic of multipliers.
- **BDDs:** Suffer from exponential size explosion.

SCA Limitations

Symbolic Computer Algebra (SCA) is suited for MAC verification but faces a critical hurdle:

Intermediate Polynomial Explosion

Intermediate terms grow excessively, exceeding memory limits for complex architectures.

Goal: Mitigate Term Explosion through Conflict-Aware Learning

Overview

Goal

Formal Verification of Integer MAC circuits (AxB+C)

Circuits & Experiments

- Known architectures & synthesized designs
- Improvement over state-of-the-art

Verification Technique: SCA

Optimizations:

- Dynamic Ordering
- Polarity Optimization
- **Conflict Optimization**

SCA Terminology:

3 ab + 4 cf - 3 ef

Polynomial

Monomial (e.g., 3ab)

Coefficient: (e.g., 3)

Variable: (e.g., a, b)

Agenda

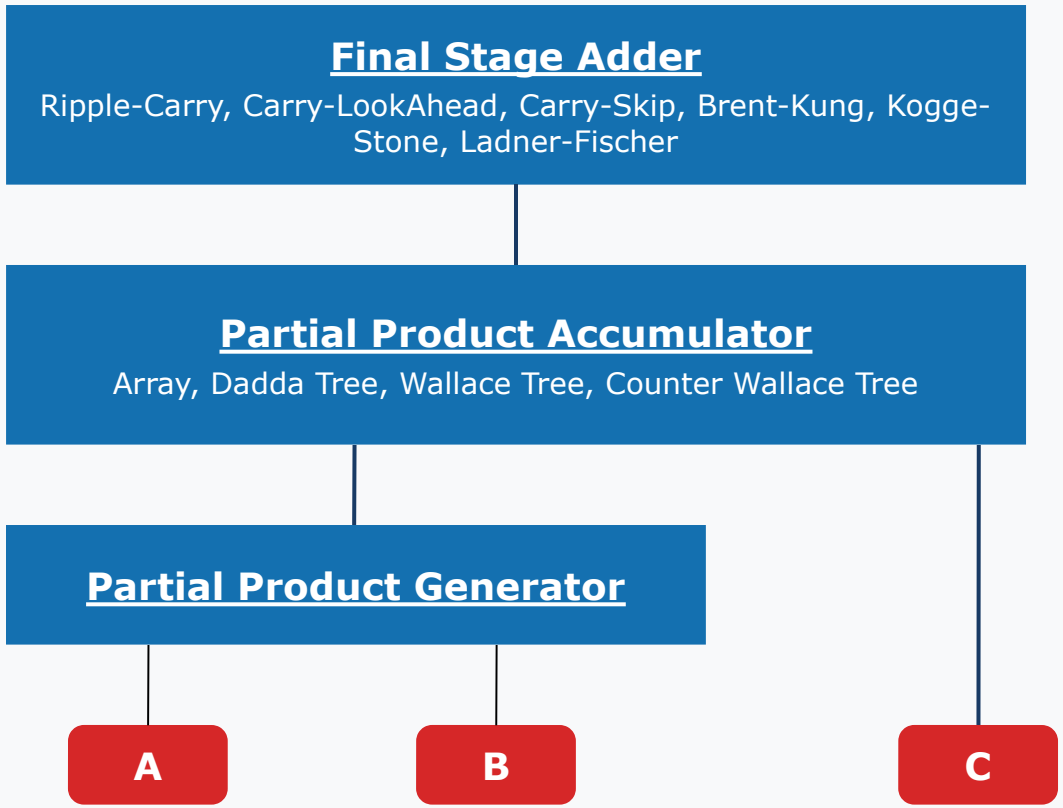
01 MAC Architectures

02 SCA Verification

03 Conflict Optimization

04 Experiments on various MAC architectures

MAC Architectures



Merged Arithmetic

- MAC is more than just MUL + ADD
- MAC: Inputs of C are embedded directly into partial products

Block Complexity

- **Partial Product Generator:** Requires n^2 AND gates
- **Partial Product Accumulation:** Compression achieved using Full and Half Adders
- **Final Stage Adder:** Performance is strongly architecture dependent

Symbolic Computer Algebra(SCA)

Example:

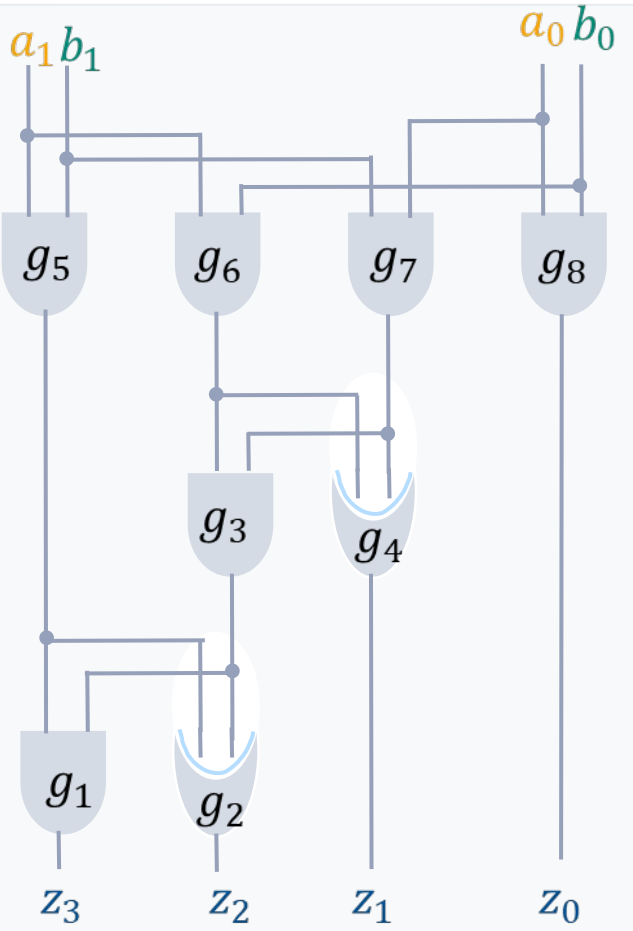
- Verification of a simple 2-bit multiplier
- Factors are represented as
  - $A = (2a_1 + a_0)$
  - $B = (2b_1 + b_0)$
- Result is represented as $Z = 8z_3 + 4z_2 + 2z_1 + z_0$

Observation:

$Z = A \times B \rightarrow Z - A \times B = 0$

$8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0) = 0$

Specification Polynomial



Symbolic Computer Algebra(SCA)

Example:

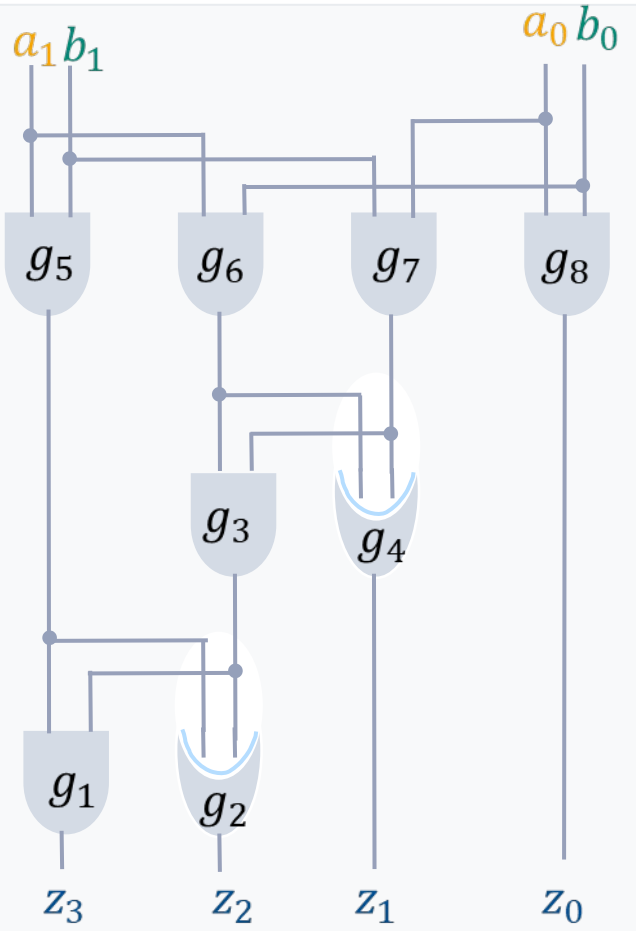
- Verification of a simple 2-bit multiplier
- Factors are represented as
  - $A = (2a_1 + a_0)$
  - $B = (2b_1 + b_0)$
- Result is represented as $Z = 8z_3 + 4z_2 + 2z_1 + z_0$

Observation:

$Z = A \times B \rightarrow Z - A \times B = 0$

$8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0) = 0$

Specification Polynomial



Symbolic Computer Algebra(SCA)

Example:

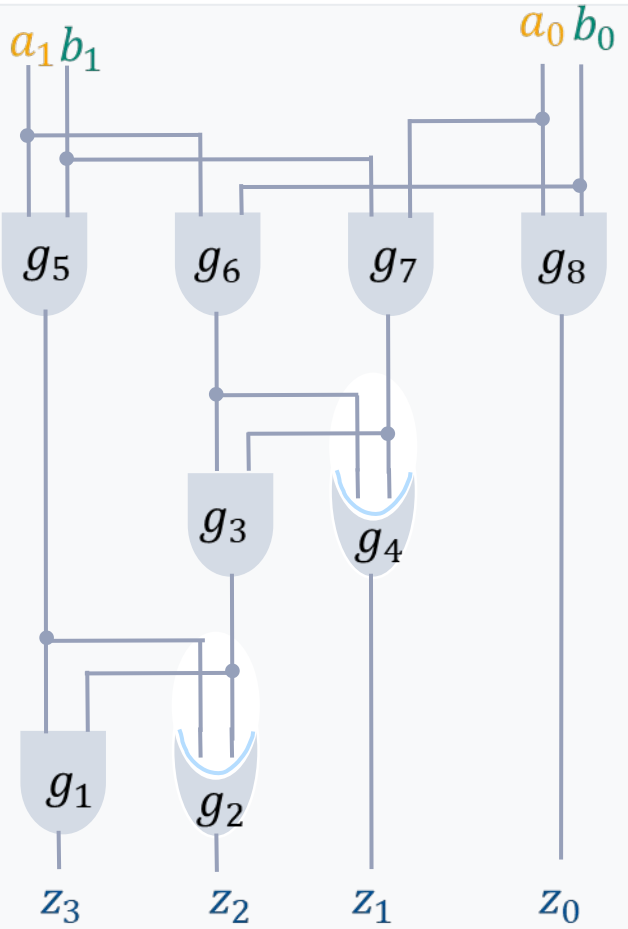
- Verification of a simple 2-bit multiplier
- Factors are represented as
  - $A = (2a_1 + a_0)$
  - $B = (2b_1 + b_0)$
- Result is represented as $Z = 8z_3 + 4z_2 + 2z_1 + z_0$

Observation:

$Z = A \times B \rightarrow Z - A \times B = 0$

$8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0) = 0$

Specification Polynomial



Symbolic Computer Algebra(SCA)

Example:

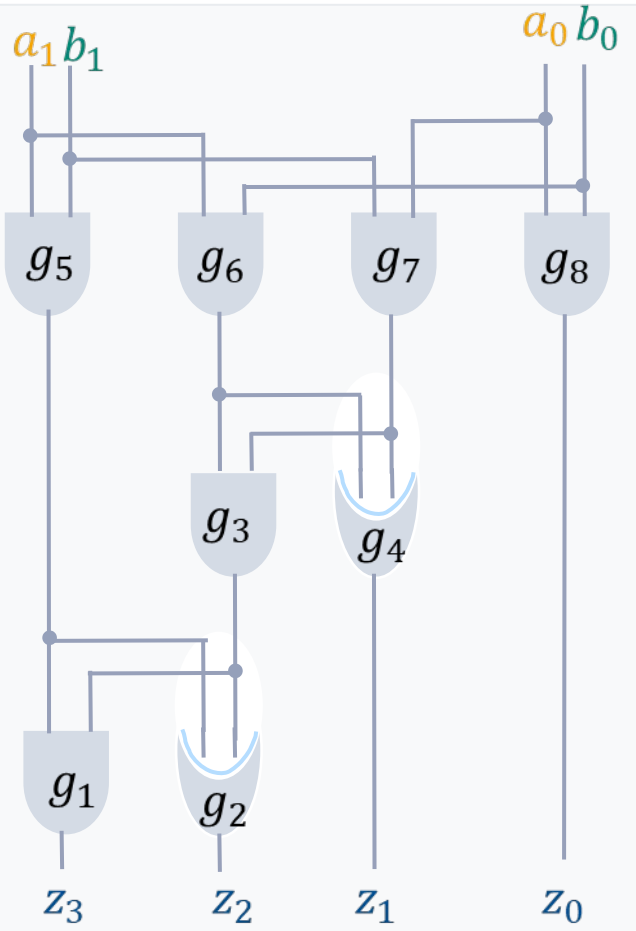
- Verification of a simple 2-bit multiplier
- Factors are represented as
  - $A = (2a_1 + a_0)$
  - $B = (2b_1 + b_0)$
- Result is represented as $Z = 8z_3 + 4z_2 + 2z_1 + z_0$

Observation:

$Z = A \times B \rightarrow Z - A \times B = 0$

$8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0) = 0$

Specification Polynomial



SCA Gate Polynomials

For polynomial representations of gates use word-level interpretation.

Negation: f = 1 - a

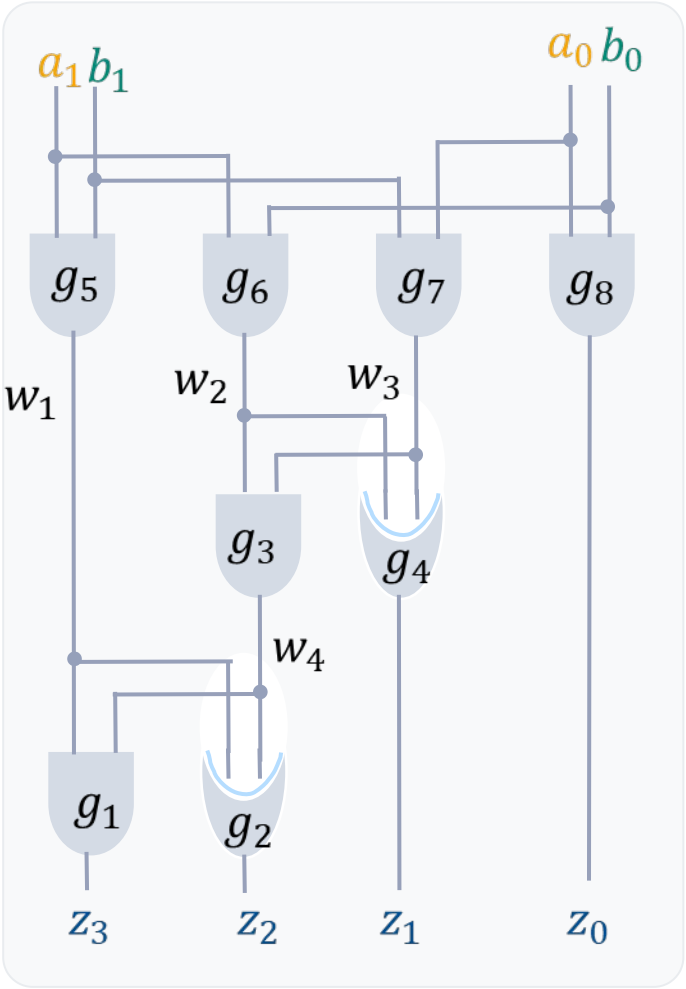
Conjunction: f = a · b

Disjunction: f = a + b - ab

EXOR: f = a + b - 2ab

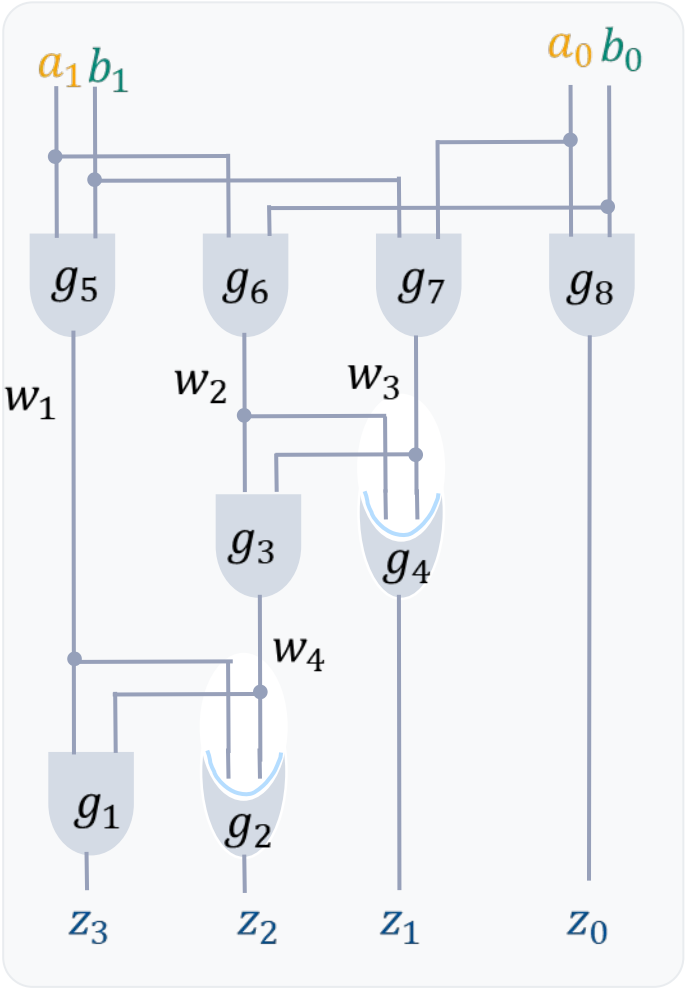
SCA Term Rewriting

$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$



SCA Term Rewriting

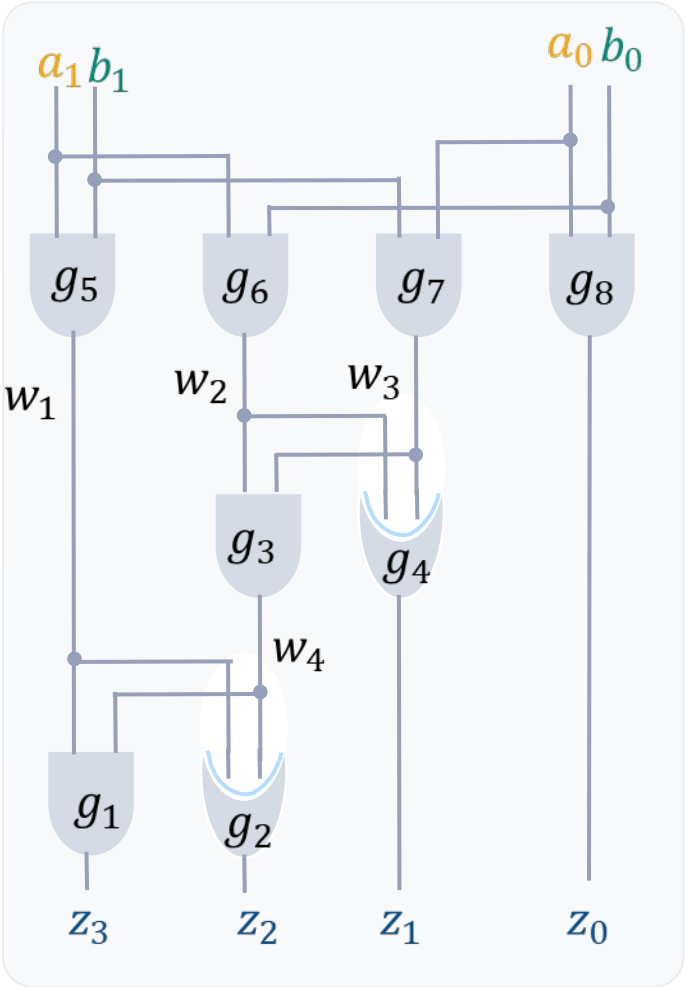
$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$



SCA Term Rewriting

$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

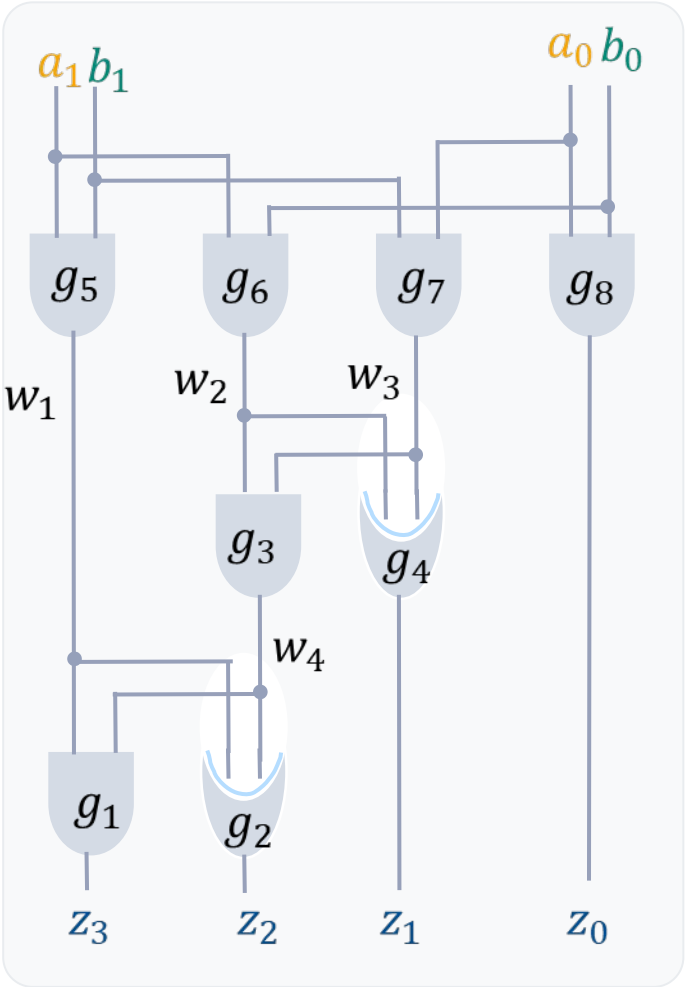
$$SP_1 = 8w_1w_4 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$



SCA Term Rewriting

$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

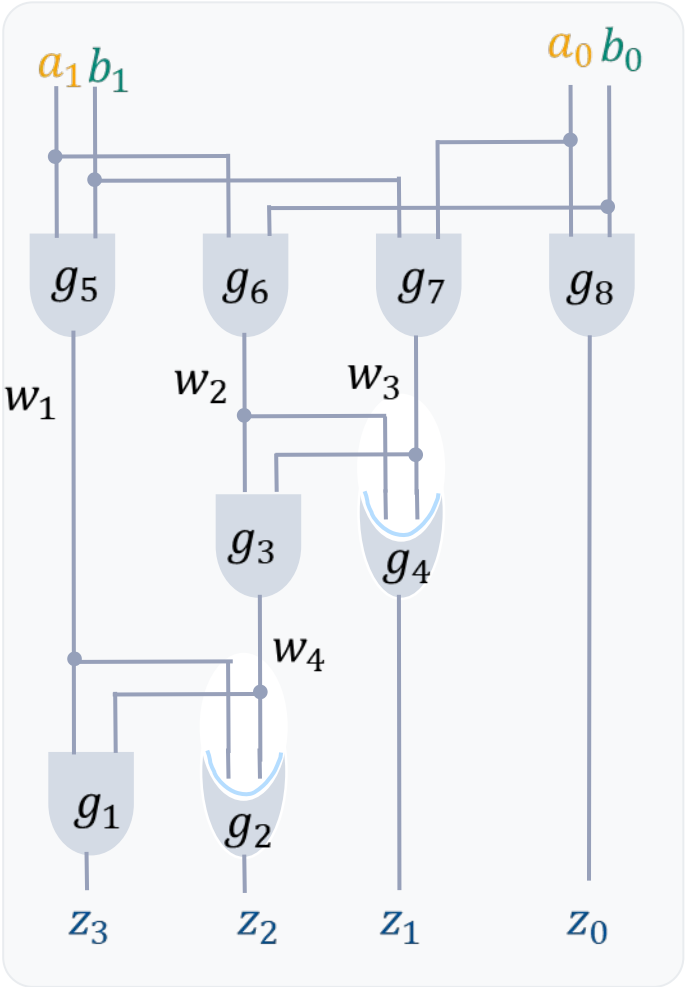
$$SP_1 = 8w_1w_4 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$



SCA Term Rewriting

$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_1 = 8w_1w_4 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$SP_2 = 8w_1w_4 + 4(w_1 + w_4 - 2w_1w_4) + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

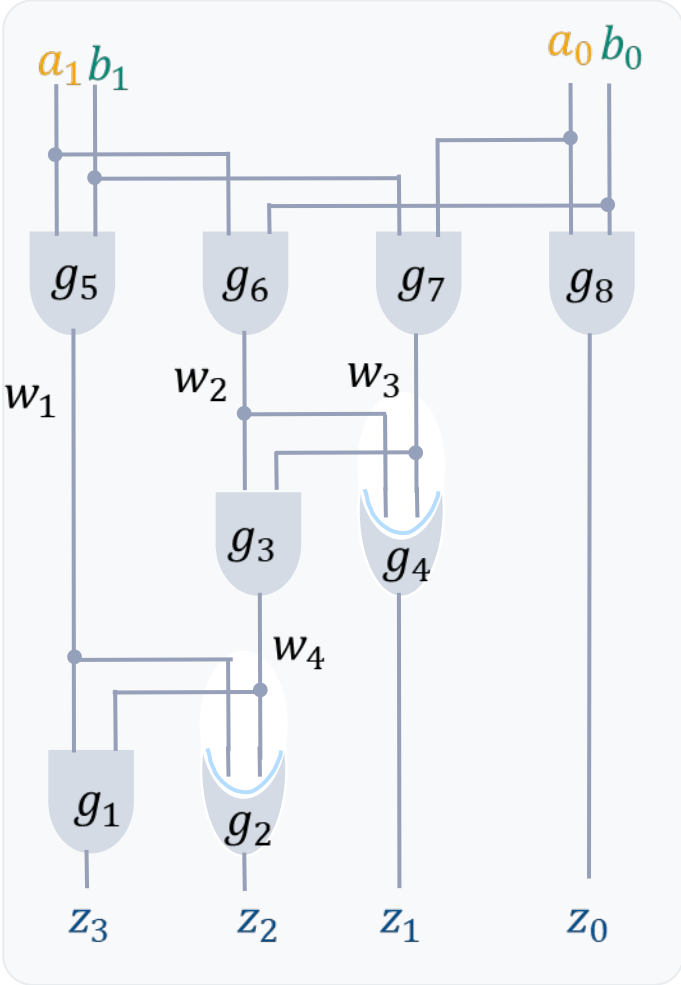


SCA Term Rewriting

$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_1 = 8w_1w_4 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

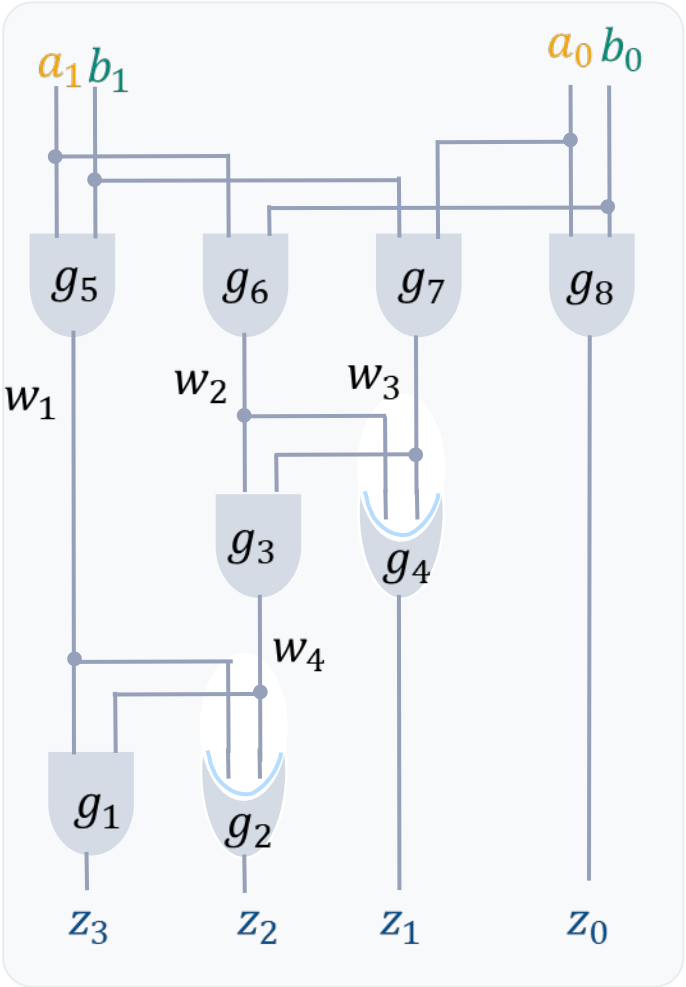
$$SP_2 = 8w_1w_4 + 4(w_1 + w_4 - 2w_1w_4) + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$



SCA Term Rewriting

$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_1 = 8w_1w_4 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$SP_2 = 8w_1w_4 + 4(w_1 + w_4 - 2w_1w_4) + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$= 4w_1 + 4w_4 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

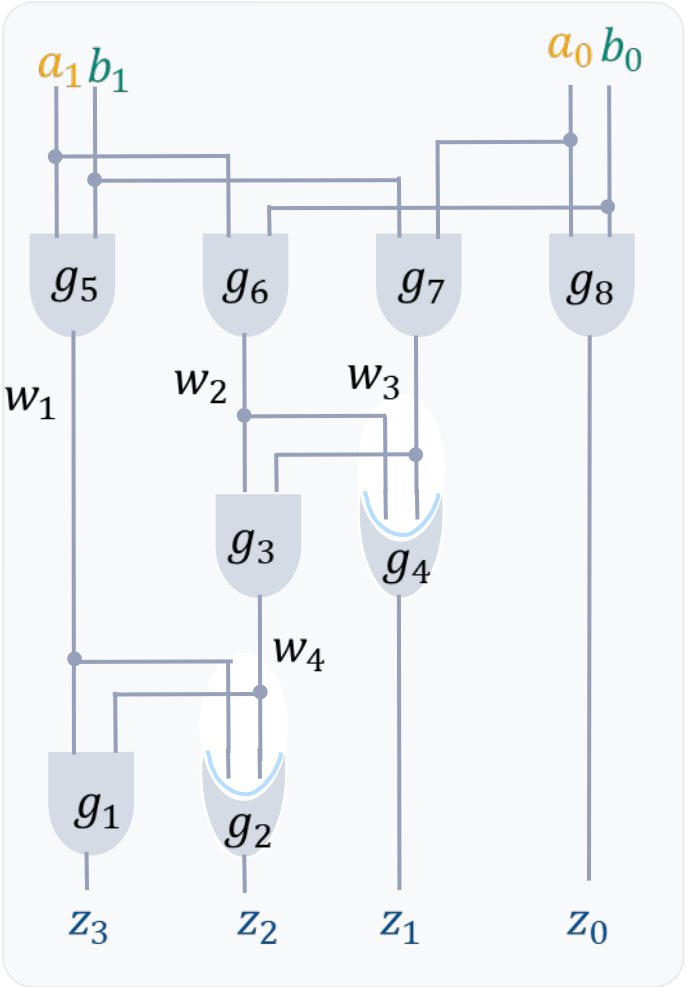


SCA Term Rewriting

$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_1 = 8w_1w_4 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_2 = 8w_1w_4 + 4(w_1 + w_4 - 2w_1w_4) + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$= 4w_1 + 4w_4 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$



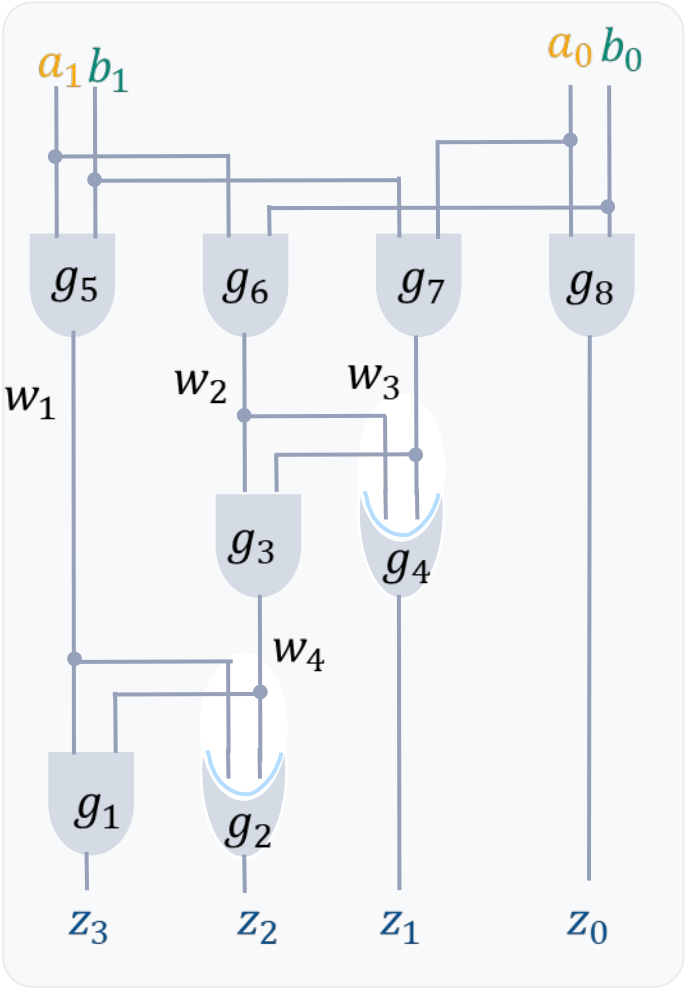
SCA Term Rewriting

$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_1 = 8w_1w_4 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_2 = 8w_1w_4 + 4(w_1 + w_4 - 2w_1w_4) + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$= 4w_1 + 4w_4 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_{3,4} = 4w_1 + 4w_2w_3 + 2(w_2 + w_3 - 2w_2w_3) + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$



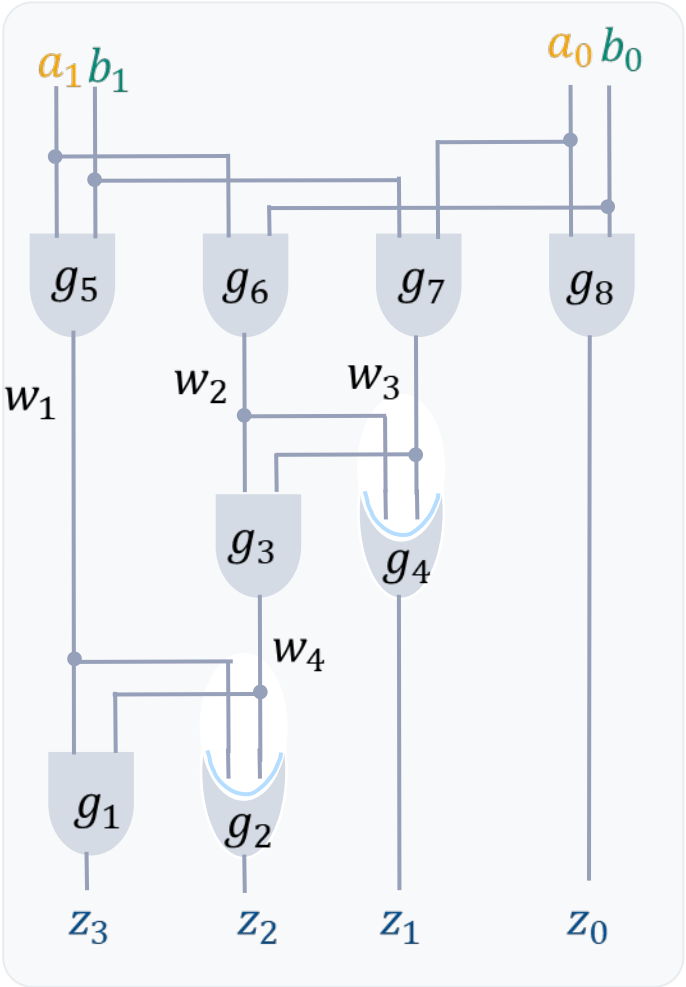
SCA Term Rewriting

$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_1 = 8w_1w_4 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_2 = 8w_1w_4 + 4(w_1 + w_4 - 2w_1w_4) + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$= 4w_1 + 4w_4 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_{3,4} = 4w_1 + 4w_2w_3 + 2(w_2 + w_3 - 2w_2w_3) + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$



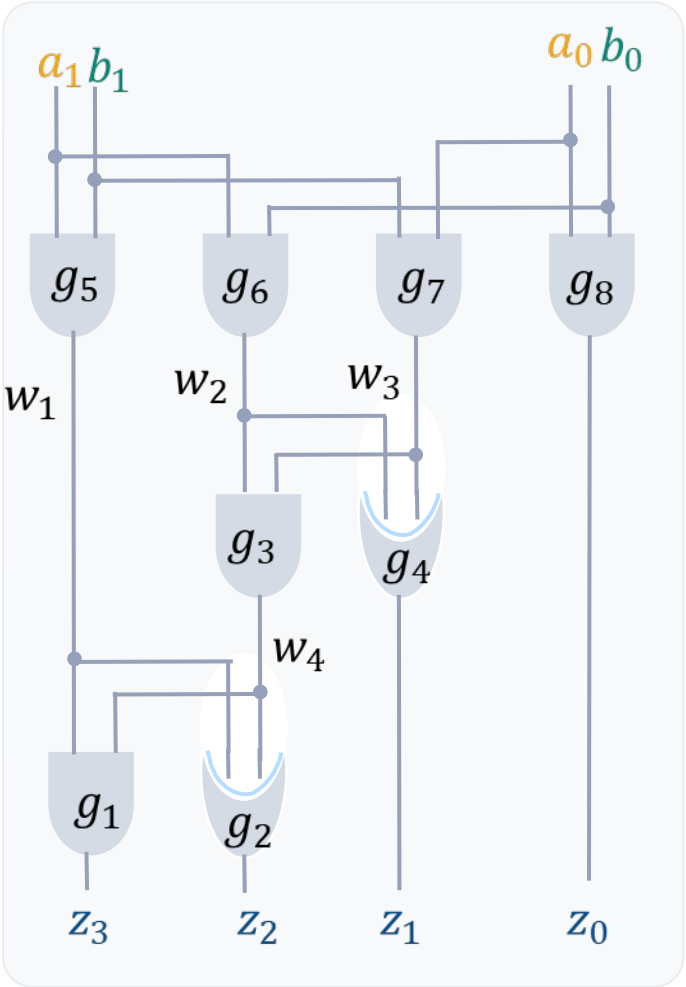
SCA Term Rewriting

$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_1 = 8w_1w_4 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_2 = 8w_1w_4 + 4(w_1 + w_4 - 2w_1w_4) + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$= 4w_1 + 4w_4 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_{3,4} = 4w_1 + 4w_2w_3 + 2(w_2 + w_3 - 2w_2w_3) + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$= 4w_1 + 2w_2 + 2w_3 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$



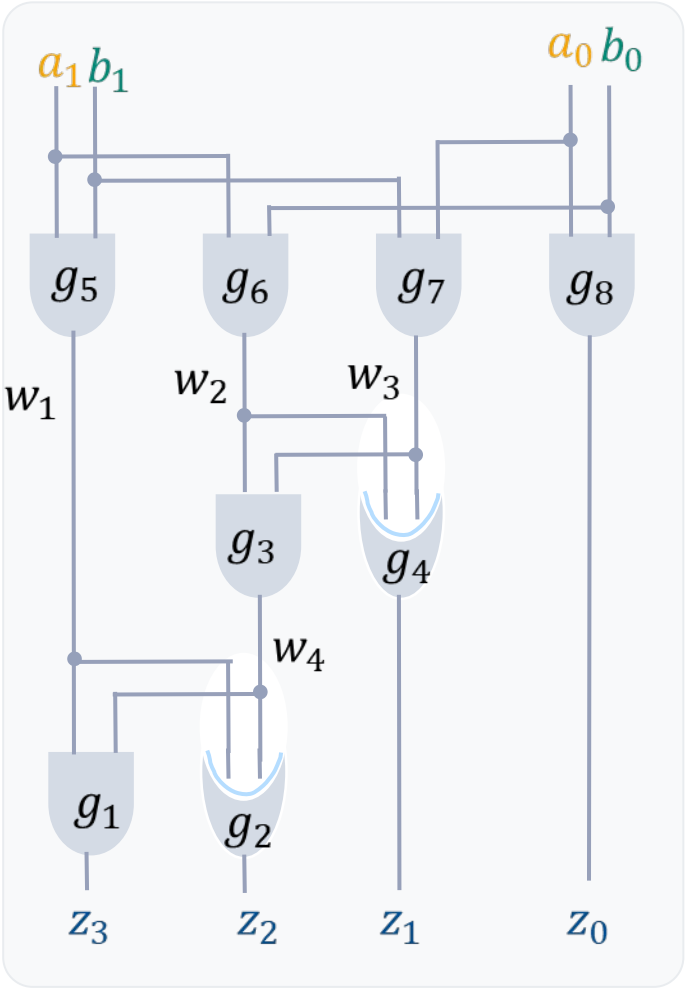
SCA Term Rewriting

$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_1 = 8w_1w_4 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_2 = 8w_1w_4 + 4(w_1 + w_4 - 2w_1w_4) + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$= 4w_1 + 4w_4 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_{3,4} = 4w_1 + 4w_2w_3 + 2(w_2 + w_3 - 2w_2w_3) + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$= 4w_1 + 2w_2 + 2w_3 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$



SCA Term Rewriting

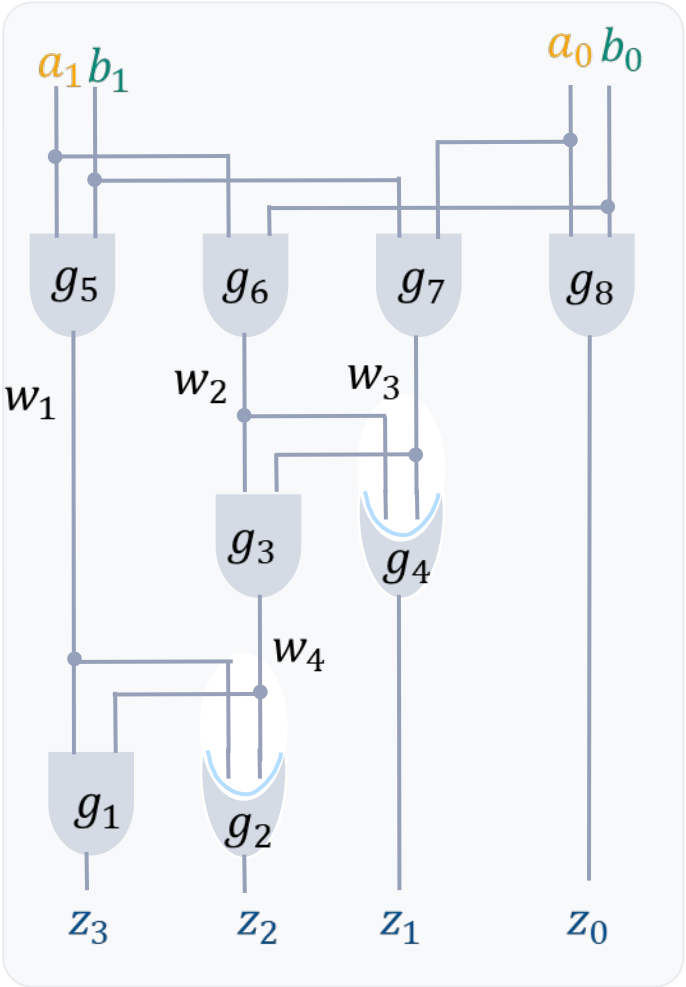
$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_1 = 8w_1w_4 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_2 = 8w_1w_4 + 4(w_1 + w_4 - 2w_1w_4) + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$= 4w_1 + 4w_4 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_{3,4} = 4w_1 + 4w_2w_3 + 2(w_2 + w_3 - 2w_2w_3) + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$= 4w_1 + 2w_2 + 2w_3 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_{5,6,7,8} = 4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$



SCA Term Rewriting

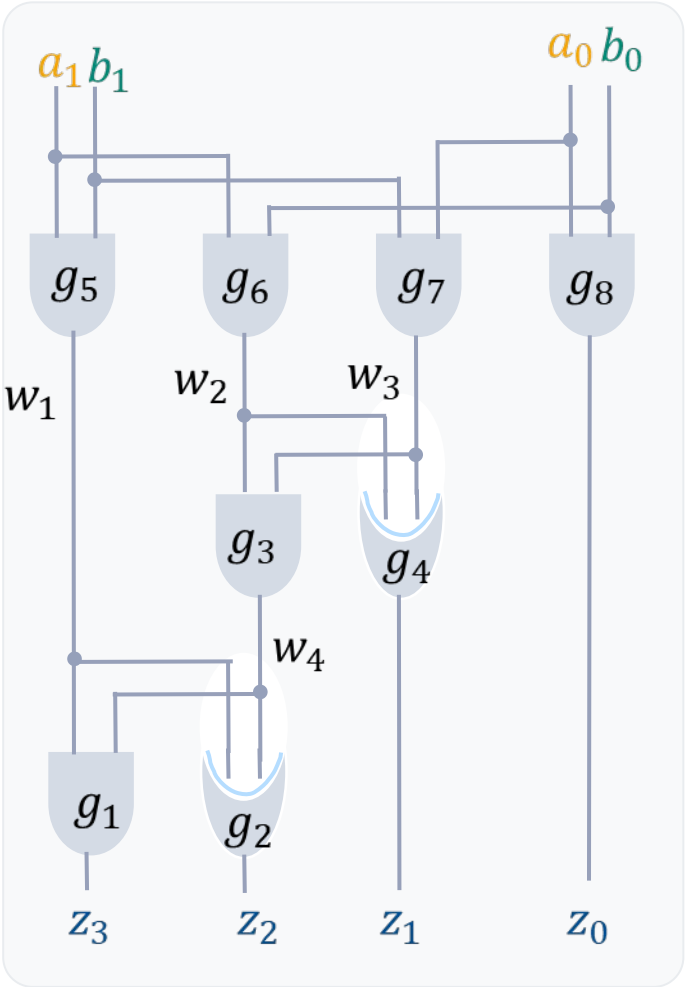
$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_1 = 8w_1w_4 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_2 = 8w_1w_4 + 4(w_1 + w_4 - 2w_1w_4) + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$= 4w_1 + 4w_4 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_{3,4} = 4w_1 + 4w_2w_3 + 2(w_2 + w_3 - 2w_2w_3) + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$= 4w_1 + 2w_2 + 2w_3 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_{5,6,7,8} = 4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$



SCA Term Rewriting

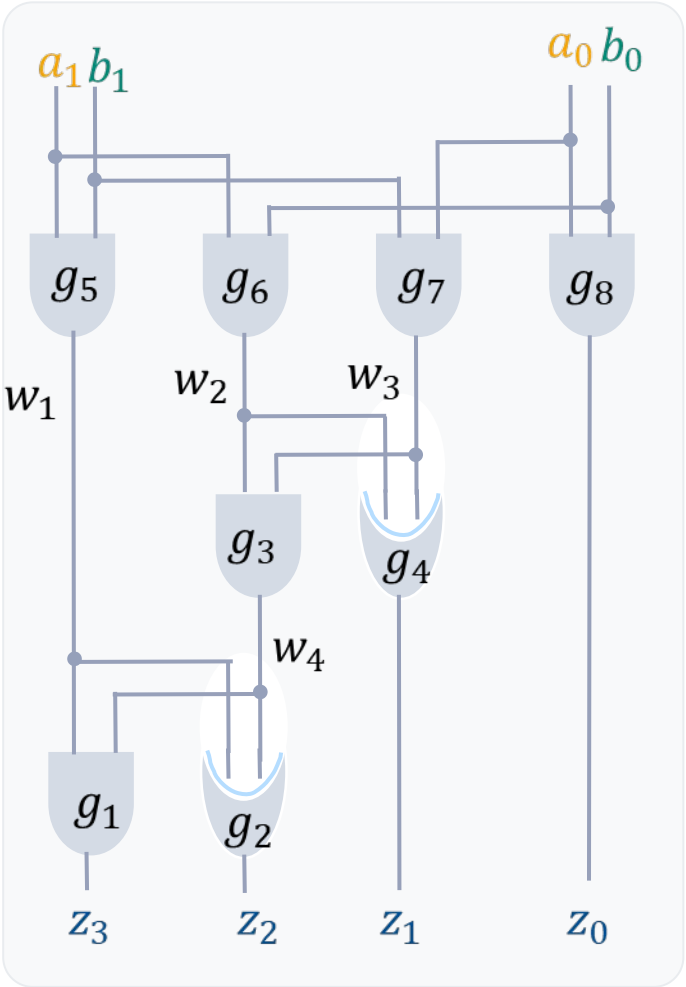
$$SP = 8z_3 + 4z_2 + 2z_1 + z_0 - (2a_1 + a_0) \cdot (2b_1 + b_0)$$
$$= 8z_3 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_1 = 8w_1w_4 + 4z_2 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_2 = 8w_1w_4 + 4(w_1 + w_4 - 2w_1w_4) + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$= 4w_1 + 4w_4 + 2z_1 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_{3,4} = 4w_1 + 4w_2w_3 + 2(w_2 + w_3 - 2w_2w_3) + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$
$$= 4w_1 + 2w_2 + 2w_3 + z_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0)$$

$$SP_{5,6,7,8} = 4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0 - (4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0) = 0$$



SCA Optimizations

Problem: Intermediate Term Explosion

Not all functions can be efficiently represented by SCA

Spec Polynomial ✔️ Final Polynomial ✔️ Intermediate Polynomials ✗

Proposed Optimizations

Dynamic Ordering

Order substitutions dynamically based on their impact on polynomial size to prevent blow-up.

Polarity Optimization

Negate selected variables to choose a more compact arithmetic representation.

Dynamic Ordering Process

The "Look-Ahead" Strategy

- Selects substitutions based on their impact on **polynomial size** to prevent term explosion.
- **Trial Substitutions:** For each available gate in the current cut, a tentative substitution is performed.
- **Decision Logic:**
  - **IF** growth < threshold (1.3): Stop search and apply immediately.
  - **ELSE**: Select the candidate with the smallest relative increase.
- Effectively manages intermediate representations to ensure shorter verification runtimes.

Phase Optimization

Concept: Reducing Monomial Count

- Mitigates excessive polynomial growth by negating selected signals.
- Analogous to Fixed Polarity Reed-Muller expressions at the Boolean level.
- Heuristic evaluation: Flipping signal polarity is checked during substitution.

Example: OR Function

Positive Polarity:

$a + b + c - ab - ac - bc + 2abc$

Complemented Form (Compact):

$1 - \overline{abc}$

Conflict Optimization

- Assume a circuit representing a function **f:IN → OUT**
- After substitution until a cut **C**, the circuit is divided into:

- Substituted part g:C → OUT
- Remaining part h:IN → C

- Intermediate polynomial represents **g**
- **h** is not necessarily injective ⇒ polynomial represents **g** for irrelevant input combinations

Conflict Optimization

Overview

- In a polynomial each monomial represents a partial variable assignment
- Two signals are in a **conflict** if they cannot evaluate to 1 at the same time
- Monomials containing conflicting variables can be removed ⇒ **polynomial size reduction**
- *Example:* Half adder outputs (vanishing monomials introduced by Mahzoon et al. [1])

Challenges

- Calculate conflicting signals
- Remove monomials containing conflicts during substitution

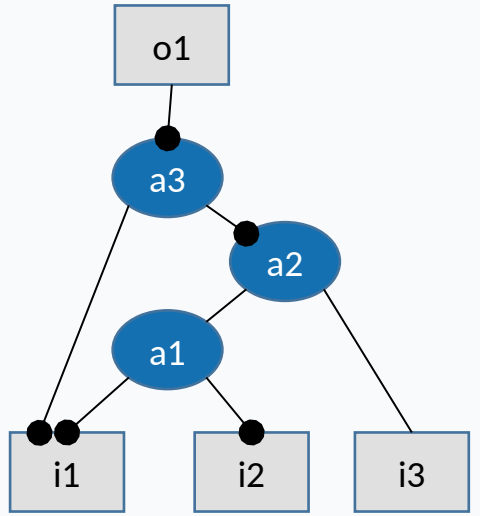
[1] Mahzoon et al., 2018. PolyCleaner: Clean your polynomials before backward rewriting to verify million-gate multipliers. ICCAD '18

Conflict Calculation Process

Algorithmic Overview

- Pre-processing step via **Implication Propagation**
- Iterates over every signal `s` and constant value `v` $\in$ `{0,1}`
- Uses **Forward** and **Backward** implication rules
- Identifies **Mutually Exclusive** signal pairs

Example



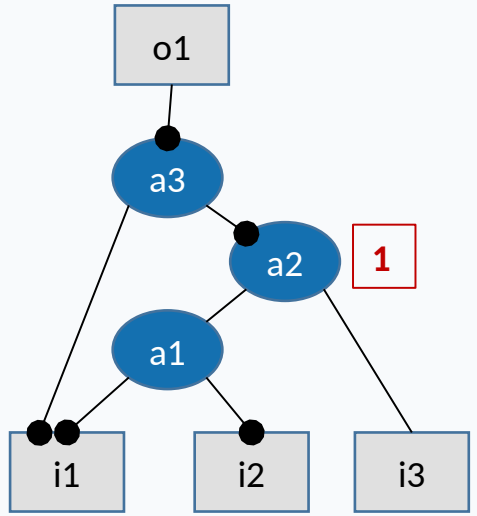
Conflict Calculation Process

Algorithmic Overview

- Pre-processing step via **Implication Propagation**
- Iterates over every signal **`s`** and constant value **`v`** $\in$ **`{0,1}`**
- Uses **Forward** and **Backward** implication rules
- Identifies **Mutually Exclusive** signal pairs

Example

1

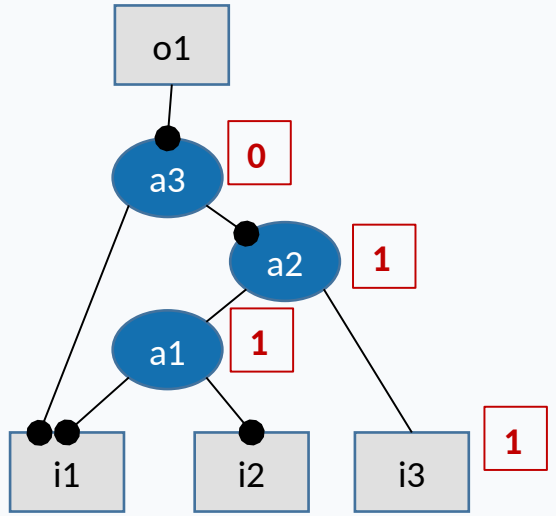


Conflict Calculation Process

Algorithmic Overview

- Pre-processing step via **Implication Propagation**
- Iterates over every signal **`s`** and constant value **`v`** $\in$ **`{0,1}`**
- Uses **Forward** and **Backward** implication rules
- Identifies **Mutually Exclusive** signal pairs

Example

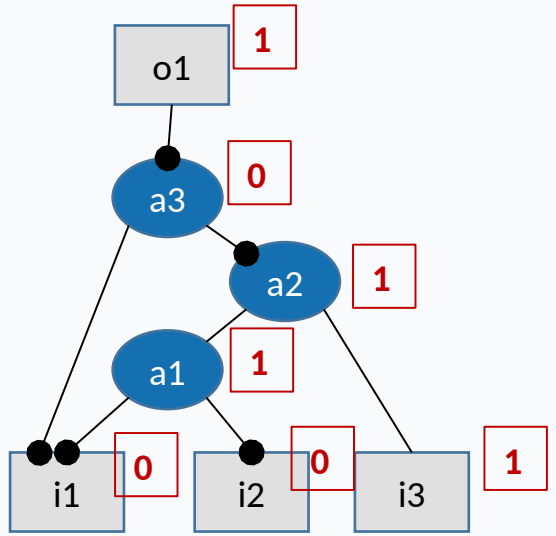


Conflict Calculation Process

Algorithmic Overview

- Pre-processing step via **Implication Propagation**
- Iterates over every signal **`s`** and constant value **`v`** $\in$ **`{0,1}`**
- Uses **Forward** and **Backward** implication rules
- Identifies **Mutually Exclusive** signal pairs

Example

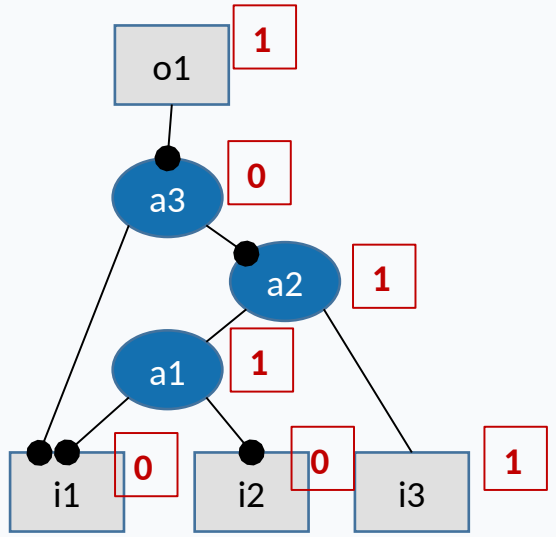


Conflict Calculation Process

Algorithmic Overview

- Pre-processing step via **Implication Propagation**
- Iterates over every signal **`s`** and constant value **`v`** $\in$ **`{0,1}`**
- Uses **Forward** and **Backward** implication rules
- Identifies **Mutually Exclusive** signal pairs

Example



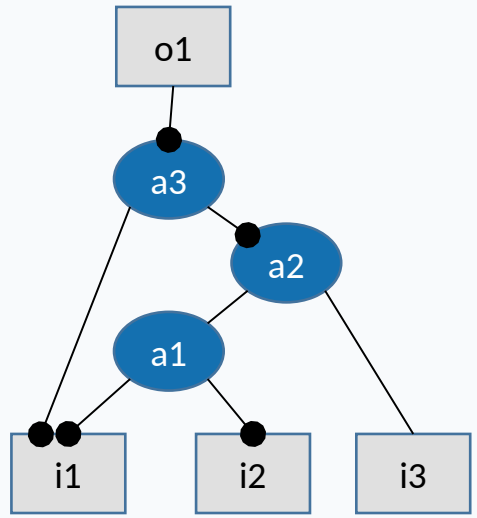
Signal **a2** is in conflict with **a3**, **i1** and **i2**

Conflict Removal

- After each substitution step scan all edited monomials for conflicts
- Conflicts can be stored in map: signal ⇒ conflicting signals

Substitution Process

$$
\begin{aligned}
& o1 \\
=^{o1}\ & 1 - a3 \\
=^{a3}\ & a2 + i1 - a2i1 \\
=\ & a2 + i1 \\
=^{a2}\ & a1i3 + i1 \\
=^{a1}\ & i3 - i1i3 - i2i3 + i1i2i3 + i1
\end{aligned}
$$



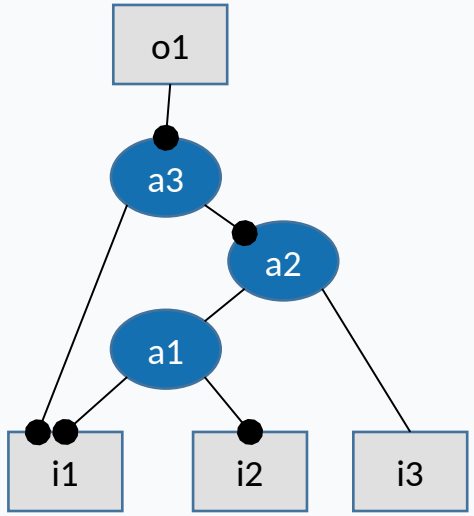
Conflict Removal

- After each substitution step scan all edited monomials for conflicts
- Conflicts can be stored in map: signal ⇒ conflicting signals

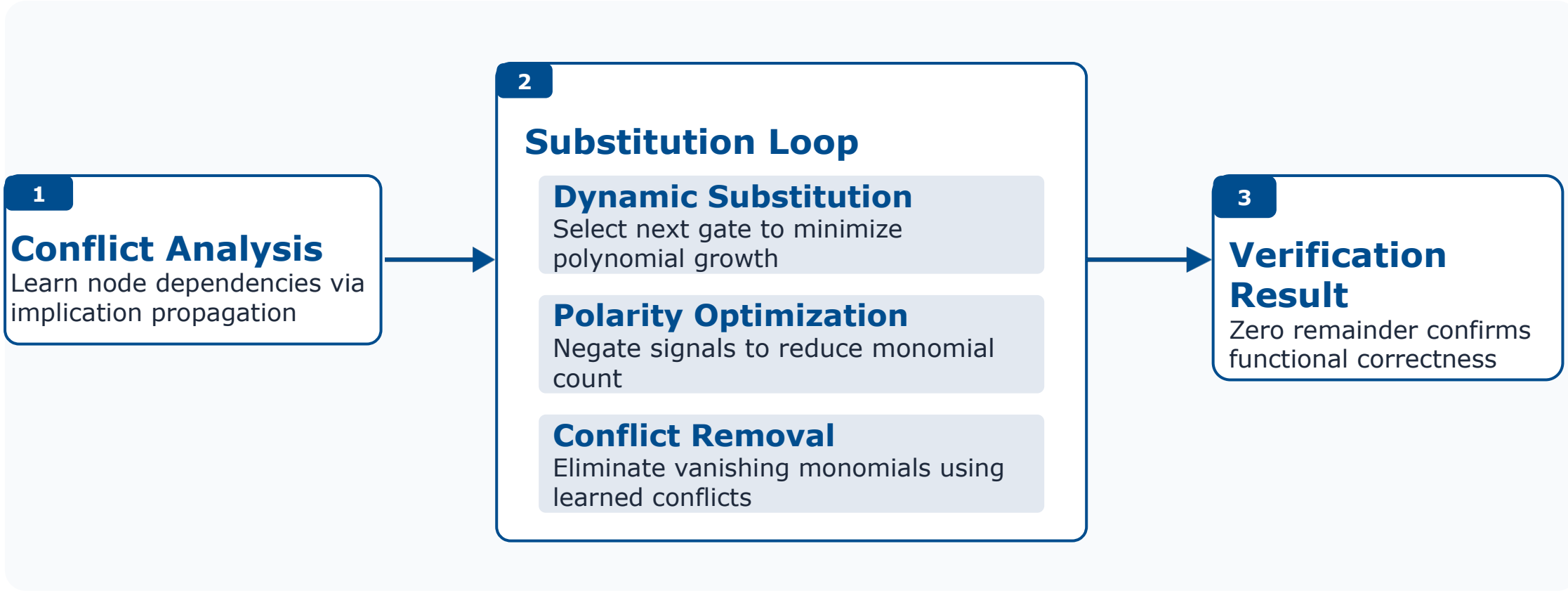
Substitution Process

$$
\begin{aligned}
& o1 \\
=^{o1}\ & 1 - a3 \\
=^{a3}\ & a2 + i1 - a2i1 \\
=\ & a2 + i1 \\
=^{a2}\ & a1i3 + i1 \\
=^{a1}\ & i3 - i1i3 - i2i3 + i1i2i3 + i1
\end{aligned}
$$

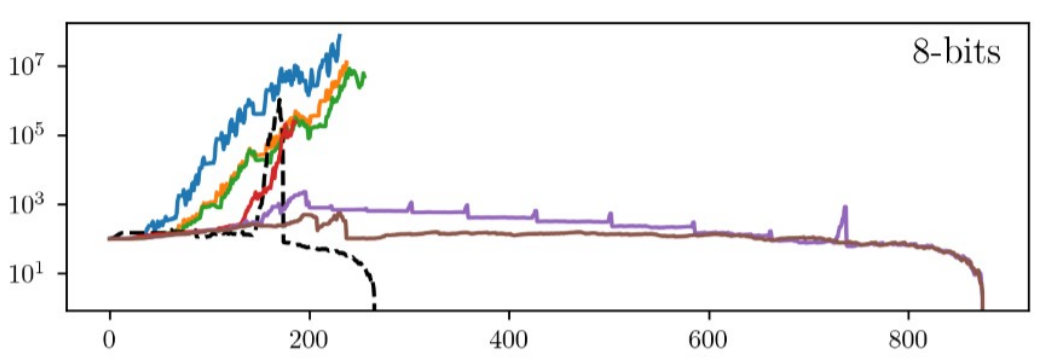
Signal **a2** is in conflict with **a3**, **i1** and **i2**

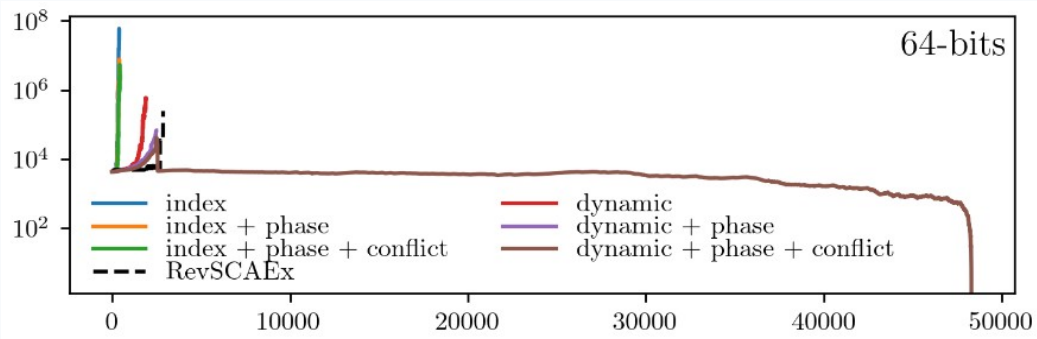


SCA-Based Verification Engine Design



Configuration Evaluation: WT-KS





Performance Analysis

Key Observations:

- **Dynamic + Phase + Conflict:** Achieves the most stable and compressed intermediate polynomial expressions.
- **Scalability:** Successfully verifies 64-bit designs where baseline and partial optimizations time out.
- **Term Explosion:** Effectively mitigates intermediate term explosion through learned conflicts.

Results on Structural MACs

Design	8-Bit		16-Bit		32-Bit	
	MP	VT	MP	VT	MP	VT
AR_CL	637	0.2928	5874	21.1273	T.O.	T.O.
AR_CS	420	0.0681	522	0.4039	100194	198.9790
AR_KS	172	0.0438	1380	1.5311	5672	97.5673
AR_RC	170	0.0436	543	0.3768	**1837**	**5.9557**
DT_CL	815	0.5289	T.O.	T.O.	T.O.	T.O.
DT_CS	261	0.0607	T.O.	T.O.	T.O.	T.O.
DT_KS	476	0.0726	1943	1.1814	26984	148.7580
DT_RC	182	0.0440	678	0.4853	2179	7.3185
WT_CL	4862	4.7639	595692	1334.5800	T.O.	T.O.
WT_CS	246	0.0553	2166	1.1536	T.O.	T.O.
WT_KS	544	0.2005	2206	0.7927	**26942**	**20.8172**
WT_RC	170	0.0482	618	0.4587	2136	7.2476
Σ	**28/28**	**28/28 RevSCA**	**25/28**	**12/28 RevSCAExt**	**21/28 RevSCAExt**	**12/28 RevSCAExt**

Optimized MACs

B	NN	Proposed					RevSCAExt		Transform	
		MP	VT	PN	DS	NC	MP	VT	MP	VT
2	38	21	0.002	11	1	14	19	0.001	13	0.001
4	147	48	0.009	42	1	55	49	0.003	34	0.002
6	334	82	0.013	81	1	126	418960	176.248	90	0.009
8	596	138	0.039	133	1	224	5819510	1445.410	336	0.033
10	919	181	0.072	220	1	346	T.O.	T.O.	1261	0.128
12	1312	278	0.179	279	1	664	T.O.	T.O.	T.O.	T.O.
16	2295	458	0.196	436	1	1015	T.O.	T.O.	T.O.	T.O.
32	8870	2698	3.431	1830	1.01	4967	T.O.	T.O.	T.O.	T.O.
64	34521	34274	833.401	7060	1.06	367991	T.O.	T.O.	T.O.	T.O.

Conclusion

Approach

- **Conflicts** are used to reduce intermediate polynomial size
- **Implication** based calculation of conflicts
- **Conflict removal** during substitution
- Embedded in engine using **dynamic substitution** and **phase optimization**

Results

- Conflicts make verification of many MAC architectures feasible
- Performance on structural and optimized designs outperforms state-of-the-art

Universität Bremen

AGRA

A Conflict-Aware Learning Approach to SCA Verification for MAC Architectures

Jan Kleinekathöfer, Lennart Weingarten, Kamalika Datta, Rolf Drechsler

ja_kl@uni-bremen.de

ARITH 2026

Bundesministerium für Forschung, Technologie und Raumfahrt

Gefördert durch DFG Deutsche Forschungsgemeinschaft