Logic Debugging of Arithmetic Circuits

Samaneh Ghandali, Cunxi Yu, Walter Brown, Duo Liu, and Maciej Ciesielski
VLSI CAD Laboratory, University of Massachusetts, Amherst

Abstract

This paper presents a novel diagnosis and logic debugging method for gate-level arithmetic circuits. It detects logic bugs in a synthesized circuit caused by using a wrong gate ("gate replacement" error) that change the functionality of the circuit. The method is based on modeling the circuit in an algebraic domain and computing its *algebraic signature*. The location and type of the bug is determined by comparing signatures computed in both directions, using forward (PI to PO) and backward (PO to PI) rewriting. It will also perform automatic correction for the detected bugs. The approach is demonstrated and tested on a set of integer combinational arithmetic circuits.

Preliminaries

We follow the arithmetic verification approach proposed in [1], with the circuit modeled as a network of basic logic gates (INV, AND, OR, XOR, etc.). Each gate is represented as a pseudo-Boolean polynomial, with Boolean variables $X = \{x_1, ..., x_n\}$ and integer coefficients from Z_2^n as follows:

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

Input Signature: Sig_{in} is a polynomial in primary input variables that uniquely represents an integer function computed by the circuit. Example (Figure 1): $Sig_{in} = (a_0+2a_1)(b_0+2b_1)$ for a 2-bit unsigned multiplier.

Output Signature: Sig_{out} of the circuit is defined as a polynomial in the primary output signals, which is uniquely determined by an n-bit encoding of the output. Example (Figure 1): $Sig_{out} = z_0+2z_1+4z_2$ for a 2-bit unsigned multiplier.

Cut Signature: The cut is a set of signals that separates PIs from POs. The *signature* of a cut is a polynomial expression in signal variables of the cut that represents an integer number computed by the circuit.

Bug Identification

Our debugging method consists of: computing cut signatures by forward rewriting, backward rewriting, and comparing the two signatures for each cut to identify and fix the bugs.

A. Forward rewriting (PI-PO)

Forward (PI-PO) rewriting starts by dividing the initial polynomial, Sig_{in}, by the polynomials describing logic gates connected to the PI signals. The goal is to replace the input variables associated with the PI gates with an expression involving the corresponding gate outputs. This process is applied iteratively to the signals of the subsequent cuts until it reaches the POs.

B. Backward rewriting (PO-PI)

Backward (PO-PI) rewriting is basically a reversed symbolic simulation [1]. Starting at the POs, the output signal of each gate is replaced by the algebraic expression in its inputs, until it reaches the PIs.

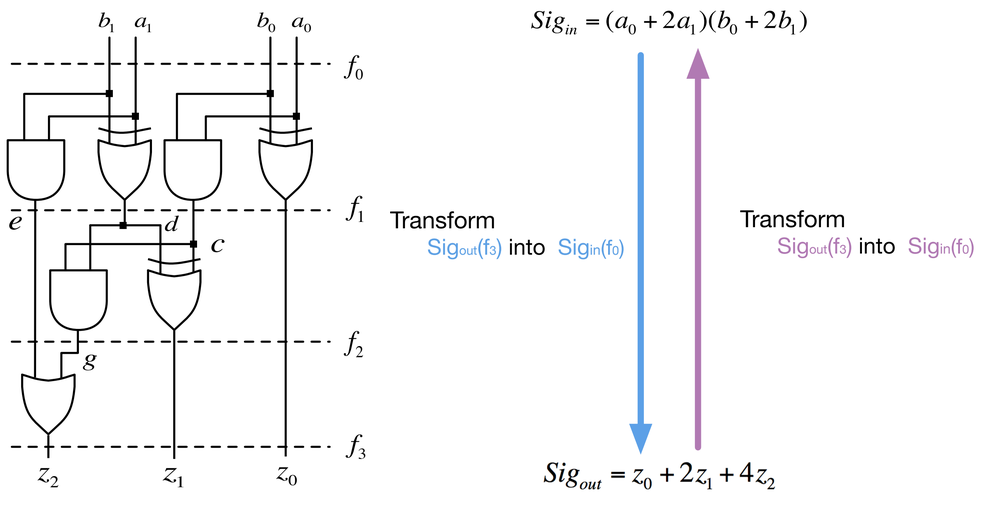

Figure 1. Forward and backward rewriting

C. Computing the signature difference

A pair of algebraic expressions is generated for each cut of the circuit: one computed by the forward and the other by the backward rewriting. The expression of Δ_i (Figure 3) will be used to identify and to correct the bug.

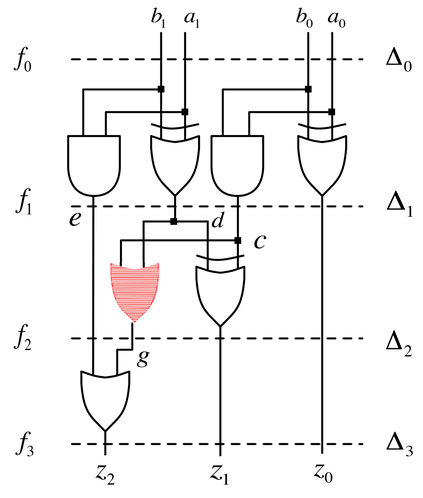

Figure 2. Buggy 2-bit multiplier

$$\Delta_1 = 4(d + c - 2dc) - 4ec - 4ed + 4ecd$$

TABLE I. Bug look-up table

Buggy \ Correct	AND	OR	XOR
AND		*a+b-2ab*	*a+b-3ab*
OR	*-a-b+2ab*		*-ab*
XOR	*-a-b+3ab*	*ab*	

Table I shows the difference in the signatures between the cut with the correct gate and the cut with the wrong gate.

Experimental Results

The debugging algorithm was implemented in C#. The experiments were conducted on a PC with Intel 1.80-GHz Core i7 processor and 6 GB of memory under Windows 8. The results include two arithmetic designs*: Wallace-tree Adder* and *CSA-Array Multiplier*. The largest, a 128-bit multiplier, contains more than 1.3 million gates. Observations:

1. The number of bugs doesn't increase the CPU time dramatically.
2. CPU time over number of gates is linear, even for non-linear circuits (multiplier).

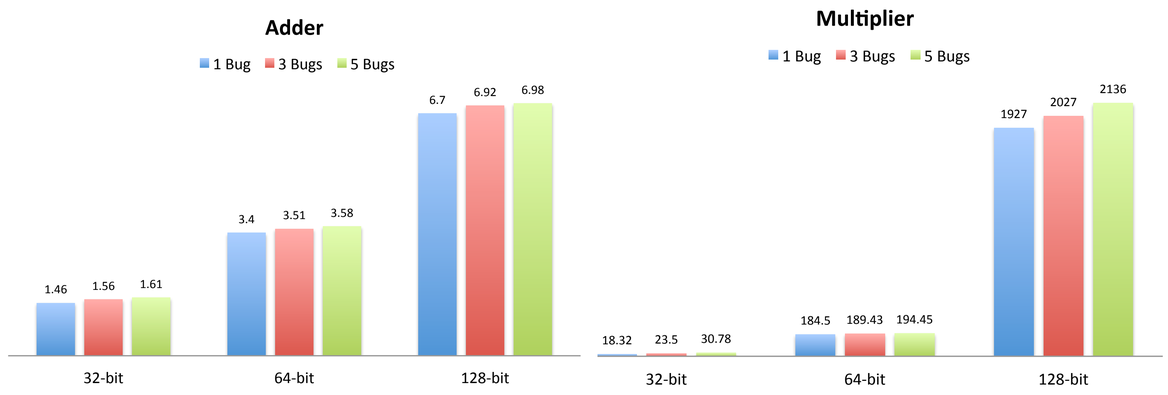

Figure 3. CPU time (sec)

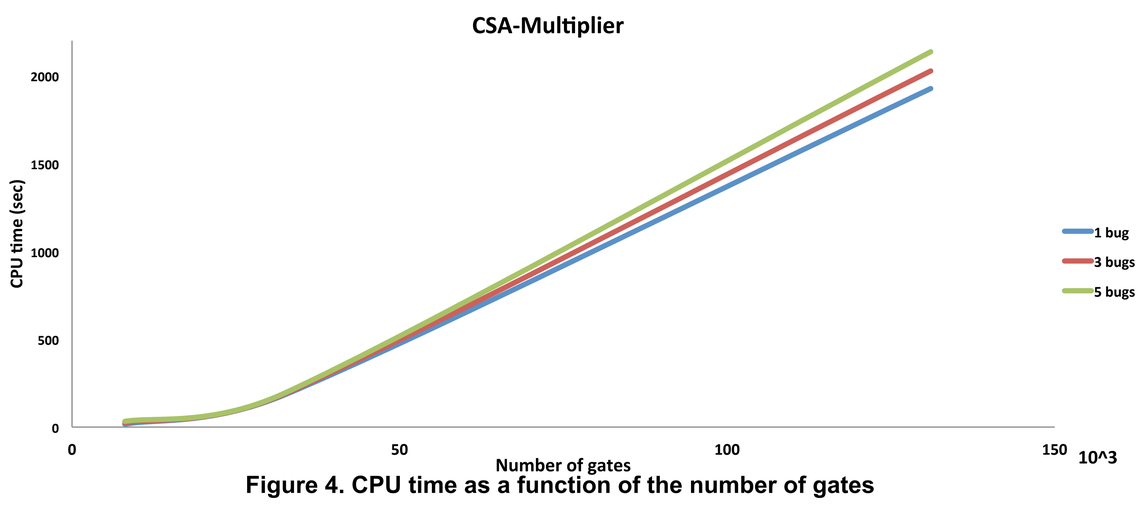

Figure 4. CPU time as a function of the number of gates

Conclusions

This work describes an original idea of identifying and correcting bugs caused by gate replacement or signal inversion in gate-level arithmetic circuits. The goal was to provide a proof of concept for the method that can be extended to solving practical problems in arithmetic circuit verification and bug correction. The method can handle multiple bugs as long as the bugs are independent. Determining the best set of cuts to improve the efficiency of the method is the major goal of future work.

[1] M. Ciesielski, C. Yu, D. Liu, W. Brown, A. Rossi. "Verification of Gate-level Arithmetic Circuits by Function Extraction" Design Automation Conference (DAC), 2015 52st ACM/EDAC/IEEE.

Department of Electrical and Computer Engineering

This work is supported in part by the National Science Foundation under NSF award No. CCF-1319496 and CCF-1419657 (REU)

Department of Electrical and Computer Engineering – University of Massachusetts, Amherst

ISVLSI 2015