ALGEBRAIC APPROACH TO ARITHMETIC DESIGN VERIFICATION

Mohamed Abdul Basith*, Tariq Ahmad*, André Rossi †, and Maciej Ciesielski*
*ECE Department, University of Massachusetts Amherst
email: {basith,tbashir,ciesiel}@ecs.umass.edu
†Lab-STICC, Université de Bretagne Sud, Lorient, France
email: andre.rossi@univ-ubs.fr

Keywords: Formal verification, Equivalence checking, Arithmetic bit level, SMT

***Abstract*—The paper describes an algebraic approach to functional verification of arithmetic circuits specified at bit level. The circuit is represented as a network of half adders, full adders, and inverters, and modeled as a system of linear equations. The proof of functional correctness of the design is obtained by computing its algebraic signature using standard LP solver and comparing it with the reference signature provided by the designer. Initial experimental results and comparison with SMT solvers show that the method is efficient, scalable and applicable to large arithmetic designs, such as multipliers.**

I. INTRODUCTION

With the increased size and complexity of integrated circuits (IC) and systems on chip (SoC), design verification becomes a dominating factor of the overall design flow. Of particular importance (and difficulty) is verification of arithmetic datapaths and their components, such as multipliers. Unlike gate-level logic designs, which can be handled using Boolean methods, arithmetic designs require treatment on higher abstraction levels. Techniques based on decision diagrams or SAT solvers that work at the bit level are not scalable for complex arithmetic systems as they require "bit-blasting", flattening of the entire design into bit-level netlists. Modern verification methods use SMT solvers and symbolic algebra techniques, but they suffer from lack of adequate models that can harness the inherent bit-level nature of arithmetic circuits.

The work described in this paper aims at overcoming some of these limitations. It presents a novel approach to functional verification of bit-level arithmetic circuits using linear algebra techniques. The proof of correctness is obtained by modeling the arithmetic circuit as a network of half/full adders and computing its algebraic signature using a standard LP solver. The computed signature is then compared to the reference signature provided by the designer.

II. PREVIOUS WORK

Several approaches have been proposed to check an arithmetic circuit against its specification at a higher level of abstraction. Different variants of decision diagrams and canonical graph-based representations have been proposed for this purpose, including BDDs [1], BMDs [2], TEDs [3] and others. BDDs have been used extensively in logic synthesis, symbolic simulation and SAT but their application to verification of arithmetic circuits is limited due to high memory requirements. BMDs and TEDs provide more efficient representation of arithmetic circuits but require word-level information about the design, which is often not available or is hard to extract from bit-level netlists.

Computer symbolic algebra methods have been applied to model arithmetic designs as polynomials over finite rings [4]. Their applicability to verification of arithmetic circuits is also limited as it relies on a word-level representation of the datapaths. An approach to verification of bit-level implementations using theory of Grobner basis over fields has been proposed by [5] and adopted by others. A technique based on term rewriting was proposed [6] for RTL equivalence checking, using a database of rewrite rules for typical multiplier implementation schemes. However, the method cannot be automated for non-standard implementations.

In [7] a gate level network of an addition circuit (a basic component of the multiplier) is modeled as a network of half adders, called *arithmetic bit-level* (ABL) network. ABL components are modeled by polynomials over unique ring, and the normal forms are computed w.r.t. the Grobner basis over rings $Z/2^n$ using modern computer algebra algorithms. In our view this model is unnecessarily complicated and does not scalable to practical designs. A simplified version of this technique has been recently proposed whereby the expensive Grobner base computation is replaced by direct generation of polynomials representing individual outputs in terms of the primary inputs [8]. However, no general method for deriving such (potentially very large) polynomials and comparing them in a systematic way against the specification has been proposed. Our paper addresses this issue using efficient linear algebra techniques.

Another approach to solving arithmetic verification problems is based on SMT (Satisfiability Modulo Theories). SMT techniques combine SAT with specialized solvers for some well-defined theories, such as Boolean logic, linear integer arithmetic, theory of equality of uninterrupted functions, and others [9] [10]. While the application of SMT solvers to property and model checking is unquestionable, their use in functional verification of custom arithmetic circuits has not been yet addressed. This paper proposes a new theory that can enhance capabilities of SMT solvers.

III. ALGEBRAIC MODEL

It can be shown that any (logic or arithmetic) circuit can be expressed as a network of half-adders (HA), full-adders (FA) and inverters. Each arithmetic or logic operator is then modeled with a set of linear equations that relate the input and output signals. This section describes modeling of

the arithmetic network and its components using algebraic equations.

A half-adder (HA) with binary inputs $a,\ b$ and outputs S (sum) and C (carry out) is represented as

$$a + b = 2C + S \tag{1}$$

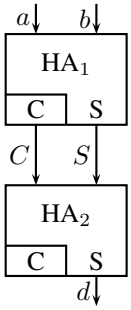
Similarly, a full adder (FA) with inputs $a,\ b,\ c_{in}$ and outputs S and C is represented as

$$a + b + c_{in} = 2C + S \tag{2}$$

Logic gates can be similarly represented by algebraic equations by deriving their functions from a half adder. Specifically, $\text{XOR}(a, b)$ is simply a sum output, S, of the half adder $\text{HA}(a, b)$, and the $\text{AND}(a, b)$ is the carry-out output, C, of $\text{HA}(a, b)$. Equations for an OR gate, $d = \text{OR}(a, b)$, can be similarly derived from the carry out (AND) output of the HA by inverting its inputs and outputs, $(1-a)+(1-b) = 2(1-d)+S$, resulting in $a+b = 2d-S$. Combining this equation with the equation (1) for HA gives $C+S = d$. As a result, an $\text{OR}(a, b)$ gate can be modeled with the following equations involving two half adders:

$$\begin{cases} a + b = 2C + S \\ C + S = d \end{cases} \tag{3}$$

Figure 1 shows the HA model for basic logic gates (AND, OR, XOR). The correctness of the equations can be verified with the attached truth table. Finally, the inverter gate $y = \text{INV}(x)$ can be trivially modeled by the following equation: $x + y = 1$.



a	b	C	S	d
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	0	1

Fig. 1. Modeling of logic gates using HA operators: $a+b = 2C+S; C+S = d$, where $S = \text{XOR}(a, b)$, $C = \text{AND}(a, b)$ and $d = \text{OR}(a, b)$.

Using these models, an arithmetic circuit can be represented by a system of linear equations, with variables x representing inputs (x_I), outputs (x_O) and internal signals (x_S). There is one equation for each HA, FA, XOR gate or AND gate, and a pair of equations (3) for an OR gate (c.f. Figure 1).

Algebraic equations representing the network are then combined in order to eliminate the internal variables from the equations and to represent the outputs of the circuit solely in terms of the primary inputs. The resulting expression is called *Algebraic Signature* of the design, denoted $Sig(N)$. Formally, algebraic signature is obtained by finding a linear combination of the network equations that results in an expression that relates the input and output variables.

The algebraic signature is then compared to the *Reference Signature* of the network, $Ref(N)$, which provides the expected relationship between primary inputs and outputs of the network (the golden model). The reference signature is basically the difference between the n-bit encoding of the output word (output signature) and a linear combination of input signals (input signature).

Reference signature is provided by the designer and can be obtained directly from the specification of the design. For example:

7-3 counter: The input signature of the 7-3 counter is simply the sum of the input bits, $x_1, \ldots, x_7$. With the output encoded in three bits, x_8, x_9, x_{10} the reference signature is

$$Ref(N) = (4x_8+2x_9+x_{10})-(x_1+x_2+x_3+x_4+x_5+x_6+x_7) \tag{4}$$

n-bit adder: For an n-bit binary adder, N_A, with inputs $\{a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}\}$ and outputs $\{S_0, \ldots, S_{n-1}, C_n\}$, the reference signature is given by:

$$Ref(N_A) = 2^n C_n + \sum_{i=0}^{n-1} 2^i S_i - (\sum_{i=0}^{n-1} 2^i a_i + \sum_{i=0}^{n-1} 2^i b_i) \tag{5}$$

2×2-bit unsigned multiplier: Since the multiplier is a non-linear circuit, we first need to convert its primary inputs $\{a_0, a_1, b_0, b_1\}$ into new variables (partial product terms), pp_I, as follows:

$$\begin{aligned} A \cdot B &= (2a_1 + a_0).(2b_1 + b_0) \\ &= 4a_1b_1 + 2a_1b_0 + 2a_0b_1 + a_0b_0 \\ &= 4pp_3 + 2pp_2 + 2pp_1 + pp_0 \end{aligned} \tag{6}$$

The variables pp_i are primary inputs to the multiplier. Assuming that the multiplier's result is encoded in 4 bits, $\{z_0, z_1, z_2, z_3\}$, the reference signature is given by:

$$Ref(N_{M2}) = (8z_3+4z_2+2z_1+z_0)-(4pp_3+2pp_2+2pp_1+pp_0) \tag{7}$$

The reference equation for signed multiplier can be derived similarly.

We shall now illustrate the idea of computing the algebraic signature using the following example.

Example 1. Figure 2 represents a 7-3 counter, a circuit that counts the number of 1s at the inputs $\{x_1, \ldots, x_7\}$ and encodes the result in a 3-bit word $S_2, S_1, S_0 = \{x_8, x_9, x_{10}\}$.

The following equations can be derived for this network using the FA model described above.

$$\begin{cases} x_1 + x_2 + x_3 - 2x_{11} - x_{12} = 0 \\ x_4 + x_5 + x_6 - 2x_{13} - x_{14} = 0 \\ x_{12} + x_{14} + x_7 - 2x_{15} - x_{10} = 0 \\ x_{11} + x_{13} + x_{15} - 2x_8 - x_9 = 0 \end{cases} \tag{8}$$

The algebraic signature of the 7-3 counter is obtained by multiplying the individual rows of equation 8 by coefficients $\alpha = \{-1, -1, -1, -2\}$, respectively, and adding them to produce the following expression:

$$Sig(N) = (4x_8+2x_9+x_{10})-(x_1+x_2+x_3+x_4+x_5+x_6+x_7) \tag{9}$$

As we can see, the computed algebraic signature is identical to its reference signature (4) proving that the design is correct, i.e., it performs the expected function.

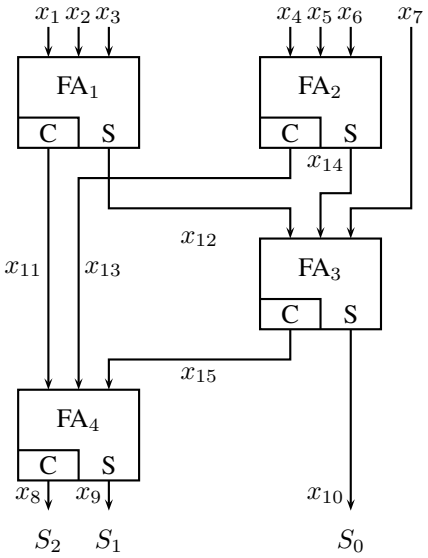


Fig. 2. Arithmetic network of a 7-3 counter.

IV. Mathematical Formulation

Let n be the total number of signals in the network, each represented by a variable, and m be the number of linear equations in the system. The network can be represented in matrix form as

$$Ax = b \tag{10}$$

where A is an $m \times n$ matrix, x is an n-vector representing the signals, and b is a constant vector of size m. Vector x of signal variables is further partitioned into the set of input signals x_I, output signals x_O, and internal signals x_S so the above system of equations can be written as: $A_I x_I + A_O x_O + A_S x_S = b$. A_I, A_O, A_S are sub-matrices of A restricted to the columns associated with input, output and internal signals, respectively. For the 7-bit counter of Fig. 2 we have $x_I = [x_1, \cdots, x_7]^T$, $x_O = [x_8, x_9, x_{10}]^T$, $x_S = [x_{11}, x_{12}, x_{13}, x_{14}, x_{15}]^T$, and b=0. Matrix A is given as follows:

$$A = \left[\begin{array}{ccccccc|ccc|ccccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 1 & 0 & 1 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}\right] \tag{11}$$

Similarly, the reference signature can be represented in this system as

$$Ref(N) = [r_O, -r_I]^T \cdot [x_O, x_I] \tag{12}$$

where x_O and x_I are the sets of variables representing output and input signals, and r_O, r_I are integer signature vectors associated with these variables. For the 7-3 counter example, with $x_O = [x_8, x_9, x_{10}]^T$ and $x_I = [x_1, \cdots, x_7]^T$, we have

$$Ref(N) = [4\ 2\ 1\ , -1\ -1\ -1\ -1\ -1\ -1\ -1] \cdot [x_O,\ x_I] \tag{13}$$

Given the reference signature $Ref(N)$, provided by the user, and its corresponding reference vector $[r_O, -r_I]$, the system computes the algebraic signature vector $r = [r_O, -r_I, r_S]$ of the network. The goal is to determine if the computed algebraic signature $Sig(N) = r^T x$ matches the given reference signature $Ref(N) = [r_O, -r_I]^T \cdot [x_O, x_I]$

As explained in Section III, signature $Sig(N) = r^T x$ is obtained as a linear combination α of the rows of Ax. Our goal is to compute vector α such that

$$[A_O, A_I, A_S]^T \alpha = [r_O, -r_I, r_S] \tag{14}$$

This is done by first solving the following linear system for α using standard LP solver:

$$\begin{cases} A_O^T \alpha = r_O \\ A_I^T \alpha = -r_I \end{cases} \tag{15}$$

Here r_S is relaxed, i.e., the internal variables are not taken into account. If this system has no solution, i.e., there is no linear combination of rows of Ax that will produce an algebraic signature whose inputs and outputs match those of the reference signature $Ref(N)$, the circuit is *incorrect* (w.r.t. that signature). If the system has a solution, the signature vector r_S associated with internal variables is computed as follows:

$$r_S = A_S^T \alpha \tag{16}$$

Ideally we are interested in having the internal variables eliminated ($r_S = 0$) as a condition for satisfying the reference signature. Applying this approach to the 7-3 counter circuit, we obtain α^T = [-1 -1 -1 -2], from which the signature vector can be calculated as $r = A^T \alpha$. The computed $r = r_O$ and r_I match those of the reference equation and $r_s = A_S^T \alpha = 0$; that is, all the internal signals have been eliminated from the signature.

But what if the computed signature $Sig(N)$ contains internal signals, i.e., if $r_S \neq 0$? We refer to such an expression as a *residual expression*, $RE(N) = Sig(N) - Ref(N)$. Does the existence of $RE(N)$ mean that the system does not satisfy the reference signature and the design is incorrect? It can be shown that this is not necessarily the case and that $r_S = 0$ is a sufficient but not a necessary condition for the design to be correct. In fact, a sufficient and necessary condition for circuit correctness is that RE reduces to zero for all the variable valuations that are produced by the network. In this case the network signature matches exactly the reference signature and the design is correct. This is illustrated with the following example.

Example 2. Consider a 2×2 signed multiplier network, shown in Figure 3. The combination of HA$_3$ and HA$_4$ models an OR gate. Inputs to the network are partial product terms pp_i, generated from the actual inputs of the multiplier, a_1, a_0, b_1, b_0, by a standard partial product generator. Hence, the expected input signature for the network is:

$$\begin{aligned} Sig_I(N) &= (-2a_1 + a_0)(-2b_1 + b_0) \\ &= 4a_1b_1 - 2a_1b_0 - 2a_0b_1 + a_0b_0 \\ &= 4pp_3 - 2pp_2 - 2pp_1 + pp_0 \end{aligned} \tag{17}$$

Hence the reference signature for this design is: $Ref(N) = -8z_3 + 4z_2 + 2z_1 + z_0 - 4pp_3 + 2pp_2 + 2pp_1 - pp_0$, where the first four terms are the output signature, obtained directly from the encoding of the output bits.

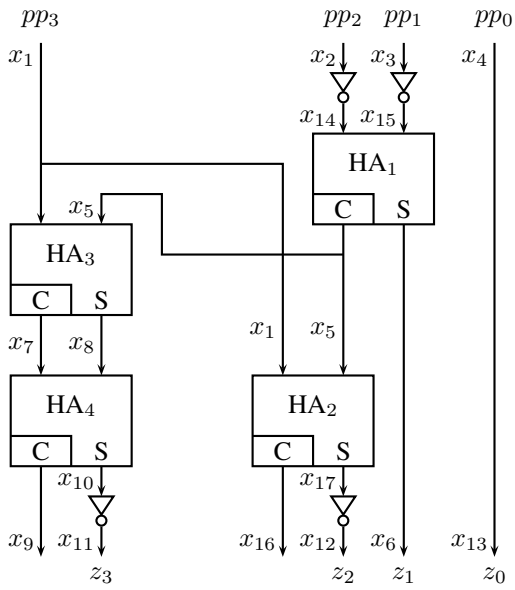


Fig. 3. Signed 2×2 multiplier network.

The algebraic signature $Sig(N)$ computed by the system is:

$$Sig(N) = \\ -8z_3 + 4z_2 + 2z_1 + z_0 - 4pp_3 + 2pp_2 + 2pp_1 - pp_0 \\ +16x_9 - 4x_8 + 4x_{17} \tag{18}$$

We note that the signature contains a residual expression $RE = -16x_9 + 4x_8 - 4x_{17}$. However, it can be shown that this expression always evaluates to zero. Namely, x_9=0 since it is the carry-out C output of HA$_4$ modeling the OR gates, which is always zero (refer to the truth table in Figure 1). The remaining variables, $x_8,\ x_{17}$, are two equivalent outputs S of HA$_2$ and HA$_3$ that share the same inputs. Hence x_8=x_{17}, which reduces RE to zero. Such an analysis of internal equivalences allows one to determine whether the residual expression evaluates to zero. If it does, the network performs the desired function expressed by the reference signature and the circuit is considered correct. Otherwise the circuit is incorrect, i.e., it does not perform the function described by the reference signature.

V. Experimental Results

The arithmetic verification technique described in the paper has been implemented as a prototype program written in C. The program uses GLPK package [11] to solve the linear system needed to compute an algebraic signature of the network.

A detailed flow of the verification procedure based on algebraic signature computation is shown in Fig. 4. The input to the system is the description of the arithmetic network N, composed of arbitrary logic gates, HA and FA operators, along with the reference signature provided by the designer. The system computes a complete signature of the network and reports if there is a non-empty residual expression $RE(N)$. If $RE \neq 0$, additional constraints need to be extracted from the network and imposed on RE in an attempt to prove that it is zero. These constraints come in two flavors: 1) signal *equalities*, caused by fanout of internal signals, e.g., $x_8 = x_{17}$ in Example 2; and 2) Boolean *constants*, such as x_9=0 in Example 2.

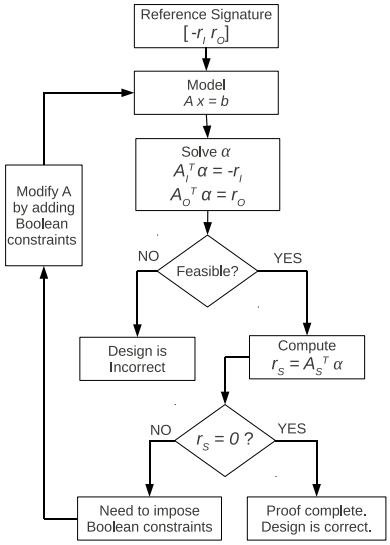


Fig. 4. Flowchart of the functional verification system.

Note that by construction (equation 15) the signature vector of a correctly designed circuit will always match its reference signature, otherwise the system has no solution and the circuit is declared incorrect.

We conducted a set of experiments on a number of arithmetic circuits, including large integer multipliers. First, a bit-level structural verilog code was generated for each multiplier using a generic multiplier generator software. (courtesy of the University of Kaiserslauten). The verilog code was parsed to transform the multiplier circuit to a network of HA, FA and basic logic gates from which a system of linear equations was generated, as described in Section IV. Finally, our program with link to GLPK was used to compute the algebraic signature for the network, given the expected reference signature.

Since multipliers are non-linear networks, we concentrated on the part of the designs which uses partial products as its inputs. Equation 17 illustrates the generation of partial product, $a_i b_j$, for a 2-bit area multiplier. Similar expressions can be readily obtained for Booth-recoded products. Such recoded product generator can be easily proved using Boolean methods.

Table I shows our results for a set of signed integer multipliers up to 256 $\times$ 256 bits. The experiment was conducted on a 2 GHz machine running Linux, with Intel(R) Dual Core(TM) T3200 processor and 3GB RAM. Since most of the research in this field has been done in the context of property checking rather than strictly functional verification, we could only compare our results to those in [12], for arithmetic proof (AP) of integer multipliers. The table gives the size of the multiplier (in the number of bits n of each operand); the number of linear equations ($constr$); the CPU time to compute

the signature and the CPU time for arithmetic proof (AP) of integer multipliers, reported in [12]. The AP results were computed on a comparable 64-bit 2 GHz Power5 machine, and reported only for 24, 53 and 64 bit integer multipliers.

The computed signatures were free of residual expressions after imposing simple Boolean constraints (constants 0) related to the OR gate configuration discussed earlier.

Size (n)	This work mult $n \times n$		AP [12]	Z3	Yices
	Constr.	CPU (sec)	sec	sec	sec
3	21	0.00	-	0.23	0.02
4	44	0.00	-	466.36	0.05
8	216	0.00	-	MO	TO
16	944	0.02	-	MO	TO
24	2184	0.04	7	MO	TO
30	3450	0.07		MO	TO
32	3936	0.09	-	MO	TO
53	8268	0.77	480	MO	TO
64	12096	1.14	840	MO	TO
128	48768	17.09	-	MO	TO
192	110016	45.23	-	MO	TO
256	195840	151.95	-	MO	TO

TABLE I
CPU TIME FOR COMPUTING ALGEBRAIC SIGNATURE OF n-BIT INTEGER SIGNED MULTIPLIERS. (MO = OUT OF MEMORY 3 GB; TO = TIMEOUT AFTER 1800 SEC)

The runtime complexity of the procedure to compute algebraic signature of the network is less then $O(n^2)$ in terms of the number of gates in a gate-level implementation of the design, c.f. Figure 5. In principle, given a network N described

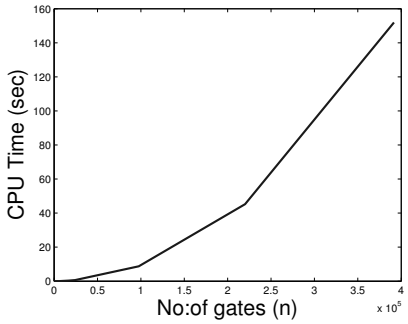


Fig. 5. Runtime complexity of the computation of algebraic signature.

by a linear system $Ax = b$, checking if the network satisfies the reference signature $Ref(N)$ can be cast as a SAT problem. Specifically, we need to show that $(Ax = b) \wedge (Ref(N) \neq 0)$ is unsatisfiable (unSAT). We performed this test for the multiplier circuits using two SMT solvers, Yices and Z3, that support Linear Integer Arithmetic as one of their theories. The results are shown in the last two columns of Table I. The SMT solvers were not able to solve this problem for multipliers with more than 8 bits. Z3 runs out of memory (3 GB) while Yices is unable to complete the computation in 30 minutes.

VI. CONCLUSIONS AND FUTURE WORK

The purpose of this work was to show a potential of the proposed algebraic technique to verify functionality of arithmetic circuits. The method is based on computing algebraic signature and comparing it with the reference signature that uniquely defines behavior of the design. If the computed signature contains non-zero residual expression RE, the signature computation must be followed by a proof that RE reduces to zero. This requires extracting constraints that are not properly captured by the linear model. Alternatively, such constraints can be imposed on the linear system directly. In this case the correct design should have no residual expression. In fact this was the case with the multipliers presented in Section V. We believe that such constraints are not hard to extract and are related to only a few types of configurations, such as constant 0 and equivalence of signals derived from a fanout, as discussed earlier. This issue is currently under investigation.

The described technique is also applicable to property checking, by representing the property by its algebraic signature and checking if it is consistent with the signature of the network. The feasibility of the resulting linear system will indicate whether such a consistency is maintained or not.

Finally, the method is limited to designs with known reference signature and such a signature must be a linear expression. This is certainly the case for portions of the designs composed of half adder networks (such as Wallace trees) often encountered in complex arithmetic designs. Application to other types of circuits needs to be examined.

ACKNOWLEDGMENTS

This work has been supported by a grant from the National Science Foundation under award No. CCF-0702506.

REFERENCES

[1] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," in *IEEE Trans. on Computers*, August 1986, vol. 35, pp. 677–691.

[2] R. Bryant and Y. Chen, "Verification of Arithmetic Functions with Binary Moment Diagrams," in *Proc. Design Automation Conference*, 1995, pp. 535–541.

[3] M. Ciesielski, P. Kalla, and S. Askar, "Taylor Expansion Diagrams: A Canonical Representation for Verification of Data Flow Designs," *IEEE Trans. on Computers*, vol. 55, no. 9, pp. 1188–1201, Sept. 2006.

[4] N. Shekhar, P. Kalla, and F. Enescu, "Equivalence Verification of Polynomial Data-Paths Using Ideal Membership Testing," in *IEEE Trans. on Computer-Aided Design*, July 2007, vol. 26, pp. 1320–1330.

[5] Y. Watanabe, N. Homma, T. Aoki, and T. Higuchi, "Application of Symbolic Computer Algebra to Arithmetic Circuit Verification," in *Proc. Intl. Conf. on Computer Design*, 2007, pp. 25–32.

[6] S. Vasudevan, V. Viswanath, R. W. Sumners, and J. A. Abraham, "Automatic Verification of Arithmetic Circuits in RTL using Stepwise Refinement of Term Rewriting Systems," in *IEEE Trans. on Computers*, 2007, vol. 56, pp. 1401–1414.

[7] O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G. Greuel, "An Algebraic Approach for Proving Data Correctness in Arithmetic Data Paths," in *Proc. Intl. Conf. on Computer-Aided Verification*. July 2008, pp. 473–486, Springer-Verlag Berlin Heidelberg 2008.

[8] E. Pavlenko, M. Wedler, D. Stoffel, and W. Kunz, "STABLE: A new QF-BV SMT Solver for hard Verification Problems combining Boolean Reasoning with Computer Algebra," in *Proc. Design Automation and Test in Europe*, 2011.

[9] A. Biere, M. Heule, H. V. Maaren, and T. Walsch, *Satisfiability Modulo Theories in Handbook of Satisfiability*, IOS Press, 2008, Chapter 12.

[10] D. Kroening and O. Strichman, *Decision Procedures, An Algorithmic Point of View*, Springer, 2008.

[11] "GNU, GLPK Linear Programming Kit," http://www.gnu.org/software/glpk/, 2009.

[12] U. Krautz, M. Wedler, W. Kunz, K. Weber, C. Jacobi, and M. Pflanz, "Verifying Full-Custom Multipliers by Boolean Equivalence Checking and an Arithmetic BitLevel Proof," in *Proc. Asia and South Pacific Design Automation Conference*, 2008, pp. 398–403.