High-level Design Verification using Taylor Expansion Diagrams: First Results

Priyank Kalla, University of Utah, Salt Lake City, UT, *kalla@ece.utah.edu*
Maciej Ciesielski, University of Massachusetts, Amherst, MA, *ciesiel@ecs.umass.edu*
Emmanuel Boutillon, Université de Bretagne Sud, Lorient, France, *emmanuel.boutillon@univ-ubs.fr*
Eric Martin, Université de Bretagne Sud, Lorient, France, *eric.martin@univ-ubs.fr*

Abstract—

Recently a theory of a compact, canonical representation for arithmetic expressions, called *Taylor Expansion Diagram* (TED) [1] [2], has been proposed. This representation, based on a novel, non-binary decomposition principle, raises a level of design abstraction from bits to bit vectors and words, thus facilitating the verification of behavioral and RTL specifications of arithmetic designs.

This paper presents the first practical results of using TED in the context of high-level design representation and verification. It discusses the use of TED for equivalence checking of behavioral and RTL designs and comments on its limitations. It also demonstrates the application of TEDs to verification of designs on algorithmic level and comments on their potential use in high level synthesis.

I. Introduction

The increase in size and functional complexity of digital designs necessitates the development of robust, automated verification tools at higher (behavioral or register-transfer) levels of abstraction. Binary decision diagrams (BDDs) [3] [4], binary moment diagrams (*BMDs), [5], and their derivatives (K*BMDs) [6], play an important role as canonical representations of Boolean and arithmetic functions and have significantly contributed to the success of verification tools. However, the application of these representations to designs with significant *word-level* computations, commonly encountered in high-level design specifications, is limited as these representations require that word-level operators be represented in terms of bits. This results in an unnecessarily large number of binary variables that need to be processed. Furthermore, high-level design descriptions typically contain word-level arithmetic computations interspersed with Boolean logic. Contemporary canonical representations cannot efficiently represent algebraic computations with symbolic operands while simultaneously modeling their interaction with Boolean logic.

This deficiency of contemporary canonical representations in representing algebraic, word-level computations motivated us to derive a new representation, called *Taylor Expansion Diagram* (TED) [1] [2] [7]. TED can efficiently extract word-level variables as abstract symbols, while simultaneously modeling algebraic-Boolean interface. The TED representation is based on a different, non-binary decomposition principle than those employed by the decision or binary moment diagrams; it uses Taylor series expansion instead.

The paper is organized as follows. In Section II, a brief overview of TED formalism is given. Section III presents the first results of using TED for RTL verification. Section IV describes the application of TED for verification at a higher level of design abstraction, i.e., algorithm verification. It also comments on a potential application of TED to high level synthesis. Section V contains final conclusions and comments on future work.

II. TED Representation

First, we briefly review the theory of TED and illustrate its construction. Let $f(x, y, ...)$ be a real, differentiable function corresponding to an algebraic expression F. Using Taylor series expansion w.r.to variable x at an initial point, $x_0 = 0$, the function can be represented as:

$$f(x, y, \ldots) = f(x = 0) + xf'(x = 0) + \frac{1}{2}x^2 f"(x = 0) + ...$$

where $f'(x)$, $f''(x)$, etc, are the successive derivatives of f w.r.to x. For a large class of expressions typically encountered in RTL and high level designs (polynomials, etc), the terms of the expansion, $f(x = 0)$ and the successive derivatives of f evaluated at $x_0 = 0$, do not depend on x, but depend on the remaining variables in the support of f. The expressions corresponding to those terms are then decomposed with respect to the remaining variables, one at a time.

The resulting decomposition is stored as a directed acyclic graph whose nodes represent the terms of the expansion. The number of children at each node depends on the order of the polynomial expression (w.r.t its decomposing variable) rooted at that node. Analogous to BDDs and *BMDs, the resulting graph is reduced and normalized, although the reduction and normalization rules are governed by different principles. The resulting *reduced, ordered, normalized* Taylor Expansion Diagram is *canonical* for a fixed variable ordering.

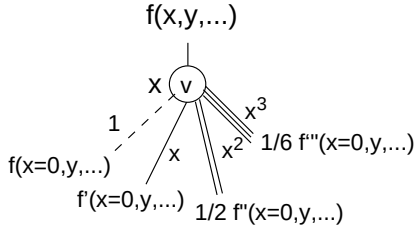

Fig. 1. Decomposition node in TED

Figure 1 shows one-level decomposition of function $f(x, y, \ldots)$ at variable x. The nodes $f(x = 0, y, \ldots)$, $f'(x =$

$0, y, \ldots$), etc, represent functions that depend only on the remaining variables: y, etc.

We shall refer to the k-th derivative of a function rooted at node v with variable x as a *k-child* of v: $f(x{=}0)$ is a 0-child, $f'(x{=}0)$ is a 1-child, $f''(x{=}0)$ is a 2-child, etc. We shall also refer to the corresponding arcs as *0-edge* (dotted), *1-edge* (solid), *2-edge* (double), etc. Furthermore, the graph edges are assigned weights, computed directly from the coefficient of Taylor expansion.

Figure 2 shows the construction of the TED for a simple algebraic expression, $F = (A+B)(A+2C) = A^2 + AB + 2AC + 2BC$. Let the ordering of variables be A, B, C. The decomposition is performed first with respect to variable A. The terms of the Taylor expansion are: $F(A=0,B,C) = 2BC$, $F'(A=0,B,C) = B + 2C$, and $\frac{1}{2} \cdot F''(A=0,B,C) = 1$. This decomposition is depicted in Figure 2(a). Next the Taylor expansion is applied to the resulting non-trivial terms $F(0), F'(0)$ with respect to variable B, shown in Figure 2(b), and subsequently with respect to variable C. The resulting diagram is depicted in Figure 2(c). Note the multiplicative weights assigned to the edges (default weight is 1).

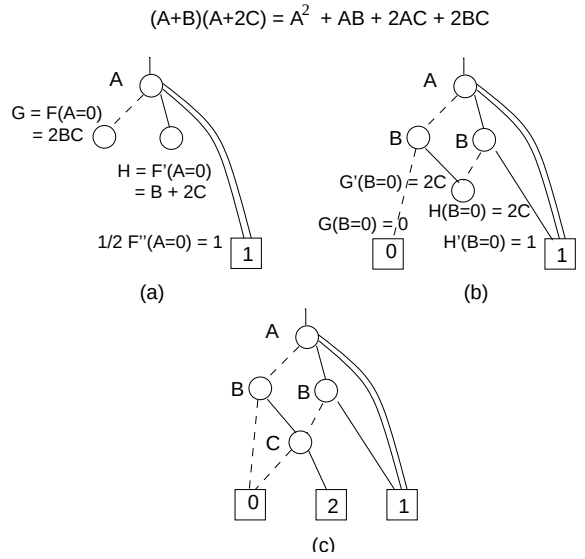

Fig. 2. TED construction for $(A+B)(A+2C)$

Conversely, the function corresponding to the TED in Figure 2 can be derived as follows. Every path from the root node to a non-zero vertex corresponds to a non-zero term in the expression. Traversing the 2-edge from root node A to the terminal node **1** leads to the term: $A^2 \cdot 1 = A^2$. Similarly, following the path of 1-edges from A to B to **1**, corresponds to $A \cdot B \cdot 1 = AB$. Finally, the paths leading to terminal **2** give $A \cdot B^0 \cdot C \cdot 2 = 2AC$ and $A^0 \cdot B \cdot C \cdot 2 = 2BC$. All the terms are then added, giving the expression $F = A^2 + AB + 2AC + 2BC$.

Figure 3 shows an example of TED representation for X^2, where X is encoded as an n-bit (here 3-bit) vector, i.e., $X = \sum_{i=0}^{n-1} 2^i x_i$. Sometimes such a decomposition of algebraic variables into bits is needed, if the individual bits appear explicitly in the design. In general, the complexity of TED for X^k is linear in the number of bits for a fixed value of k, regardless of the degree k.

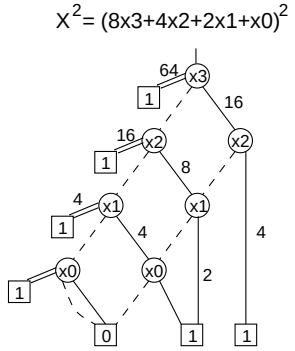

Fig. 3. TED representation for X^2 encoded in 3 bits

Complex expressions can be composed from the simpler ones using composition rules described in [2]. The composition of two TEDs is similar to that of BDD's APPLY operator in the sense that it is a recursive process. Starting from the roots of the two TEDs, the TED of the result for a given operation is constructed by recursively constructing all the non-zero terms of the two functions, and combining them to form the diagram for the new function.

In [2] and [7] we described in detail how high-level RTL computations can be represented as polynomials of finite degree and how TEDs can be constructed for various RTL designs using ADD and MULT composition operations. We shall now present the first experimental results using TED representation.

III. Implementation and Experimental Results

Taylor Expansion Diagrams have been implemented as a software library package integrated within an RTL validation and verification framework of VIS [8] [7]. The data structure stores the graph as an interconnection of nodes and edges and provides capabilities to perform all possible operations on TEDs, such as: ADD, SUB, MULT, MULTIPLEX, REDUCE, NORMALIZE, EQUIVALENCE_VERIFY, etc. It takes as input a behavioral or RTL description of the design in Verilog format. A parser analyzes the design to identify both word-level (algebraic) and Boolean variables. Starting from the primary inputs, TEDs are constructed for all intermediate signals using the TED operators until the TEDs for the primary outputs are generated. Analogous to BDDs and *BMDs, the TED package generates graph structure in a reduced and canonical form and stores it in a look-up table implemented as a *unique table*.

Using the TED package we performed a number of experiments to analyze the power of TED representations. We now present the first set of results of this prototype tool for a number of practical applications.

A. RTL Verification

During the process of high-level/architectural synthesis, the HDL description often proceeds through a series of high-level transformations. For example, RTL computation $AC + BC$ can be transformed into an equivalent one,

$(A + B)C$, which better utilizes the hardware resources. TEDs can be used to verify the correctness of such transformations by proving equivalence of the above expressions, regardless of the word size of the algebraic variables. We performed numerous experiments to verify the equivalence of purely algebraic expressions. Results indicate that both time and memory usage required by TEDs to perform such a verification are orders of magnitude smaller as compared to both BDDs and *BMDs.

An important feature of TEDs is their capability to represent designs containing both algebraic operators and Boolean logic. This is illustrated in Figure 4, where the two functionally equivalent designs differ in the organization of the computation performed. The TED representation can be used to prove their equivalence. First, the arithmetic variables A and B are *partially expanded* into arithmetic $(A_{hi}, A_{lo},\ B_{hi}, B_{lo})$ and bit-level (a_k, b_k) variables. The TEDs are then built for arithmetic operators from their components. The TEDs for the Boolean logic s_1 and s_2 as a function of a_k, b_k are then constructed, and composed with the corresponding arithmetic functions to create the final representations for the outputs.

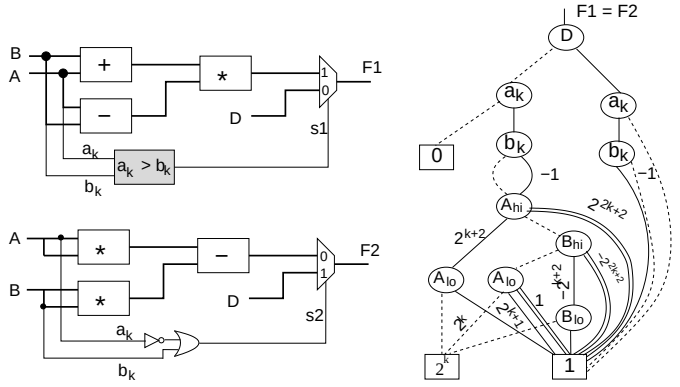

Fig. 4. Equivalence check for two RTL designs using TED. The TEDs for the two designs are isomorphic

We used a generic design $(F2)$ of Figure 4 and performed experiments to verify the equivalence of functions $F1$ and $F2$. In order to observe the power of TEDs in the presence of an algebraic-Boolean interface, the size of the algebraic signals A, B was kept constant at 32 bits, and the word size of the comparator, or the equivalent Boolean logic, was varied. As the size of Boolean logic in the design increases, the number of bits extracted from A, B also increase (the figure shows it for single bits). Table I lists our preliminary results obtained with TED for this design and compares them to those of *BMDs. We observed that as the size of Boolean logic increases, TED behavior converges to that of *BMDs.

Limitations: For designs containing Boolean signals or bit-vectors derived from internal word-level signals, algebraic variables corresponding to those word-level signals have to be decomposed into smaller bit vectors or single bits; at this point the power of abstraction of TEDs is lost. One should also keep in mind that TEDs are applicable to functions that are continuous and have finite Taylor series expansion. Therefore functions such as k^x, and algebraic

TABLE I
SIZE OF TED VS. BOOLEAN LOGIC

bits (k)	*BMD		TED		Norm. TED	
	Size	CPU	Size	CPU	Size	CPU
4	4620	107s	783	24s	194	44s
8	15K	87s	5174	13s	998	74s
12	19K	93s	5112	12.9s	999	92s
16	23.9K	249s	22.3K	94s	4454	104s
18	-	>12 hrs	67.9K	22min	12.8K	29mins
20	-	>12 hrs	-	>12 hrs	-	>12 hrs

expressions involving min, max and absolute values cannot be represented by TEDs.

B. System-level Verification

We performed experiments to evaluate the power of TED representation to represent large, system-level designs, such as an $n \times n$ array of processing elements (PEs), shown in Figure 5 (a). The design is part of a low power design of a motion estimation architecture. The original design has been simplified to accommodate the limitation of the TED representation: computations $|A - B|$ were replaced by $A - B$ since TED cannot efficiently deal with computation of absolute values. Similarly, the word-wide comparators were removed as they would need to be modeled by Boolean logic. Each processing element can be programmed to perform different types of computations, such as those shown in Figure 5 (b) and (c).

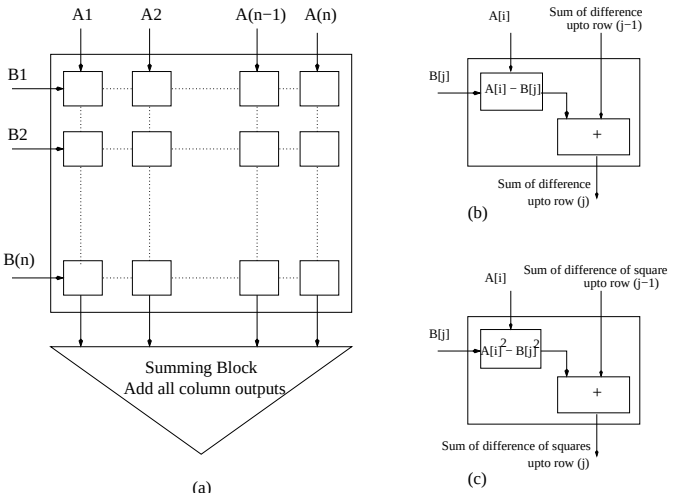

Fig. 5. An $n \times n$ array of processing elements.

Taylor Expansion Diagrams were used to represent the final expression corresponding to both types of computations performed by the array. For each computation, the PE takes as input two 8-bit numbers A and B and computes either $A - B$ or $A^2 - B^2$. Tables II and III show the results of experiments when configuring each PE to compute $A_i - B_j$ and $A_i^2 - B_j^2$, respectively. The size of the $n \times n$ array varied for $n = 4$ to $n = 16$. The results for non-normalized and normalized TEDs are compared with those for *BMDs. We were unable to construct the BDDs for any size n of the array.

TABLE II
PE COMPUTATION: $(A_i - B_j)$.

Array size $(n \times n)$	*BMD		TED		Norm. TED	
	Size	CPU	Size	CPU	Size	CPU
4× 4	66	3.1s	11	1s	10	1s
6× 6	98	3.4s	15	1.5s	14	1.5s
8× 8	130	3.5s	19	1.5s	18	2s
16×16	258	9 s	35	2 s	34	3.8s

From Table II it is clear that while we were able to construct both *BMDs and TEDs for all the designs, the number of nodes required for TEDs is much smaller. Since the computations performed by the array are only add and subtract, *BMDs are able to represent them linearly. However, when the PE is configured to compute $A_i^2 - B_j^2$, and the size of the array is increased from 4×4 to 16×16, *BMDs are unable to represent the final computation in computer memory. The results for this experiments are given in Table III.

TABLE III
PE COMPUTATION: $(A_i^2 - B_j^2)$.

Array size $(n \times n)$	*BMD		TED		Norm. TED	
	Size	CPU	Size	CPU	Size	CPU
4×	123	3s	11	1.2s	10	1.2s
8× 8	6842	112s	19	1.5s	18	1.6s
16× 16	out of mem	-	35	7s	34	8.8s

IV. High Level Verification and Synthesis

As shown earlier, TED provides a simple canonical way to express abstract algebraic expressions. Thus two questions arise: First, can TED be used to verify the equivalence of two mathematical expressions representing designs on algorithmic level? Second, can TED can be used for high level synthesis, similarly to the way BDDs are used for RTL synthesis? The remainder of this section addresses these issues.

A. Verifying Equivalence of Algorithmic Specifications

Recently, we have observed the power of Taylor Expansion Diagrams to represent and verify designs on higher levels of abstraction, namely at the *algorithm level*. Many arithmetic computations present in DSP algorithms involve manipulation of polynomials, which can efficiently be represented with TEDs. This suggests applicability of TEDs to algorithm verification, verification of error correction codes, cryptography, etc.

In order to check the capability of TED to verify algorithmic specifications, we selected a non trivial example from the DSP world, namely the computation based on FFT. Using TED as canonical representation, we proved the equivalence between the following computations: 1) FFT, followed by product computation and inverse FFT (IFFT) operations of the result, and 2) the temporal convolution on the same input vector. These computations are illustrated in Figures 6 and 7. The computation flow in Figure 6(a) first performs FFT operation on two vectors of real numbers, $A(i)$ and $B(i)$; it then computes the product of the results (producing complex numbers $FAB(i)$); and finally computes the inverse FFT operation, $IFFT(i)$. Figure 6(b) shows the TED graph, automatically generated by the package, for one of the intermediate results, the real part of $FAB2$. Figure 6(c) shows the TED graph for one of the outputs, the real part of $IFFT0$.

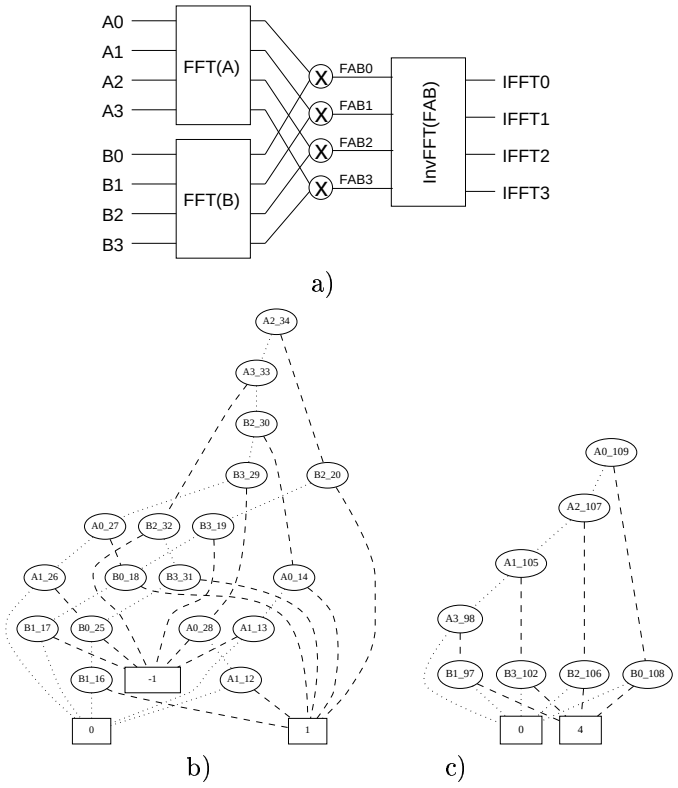

Fig. 6. FFT computation: a) Block diagram; b) TED for real part of FAB2; c) TED for real part of IFFT0

The computation shown in Figure 7 performs convolution operation directly in the time domain, producing a vector of real numbers, $C(i) = \sum_{k=0}^{n-1} A(k) \cdot B(i-k)$.

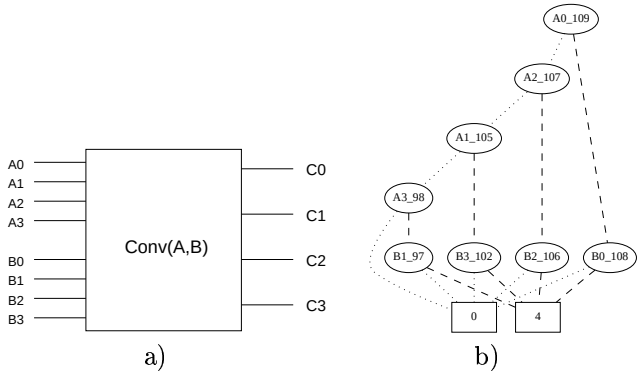

Fig. 7. Convolution computation: a) Block diagram; b) TED for real part of $C0$, isomorphic with $IFFT0$

Using current version of our software we were able to prove the equivalence of the two algorithms shown in the figures. That is, the TED graphs for $IFFT(i)$ and $C(i)$

are isomorphic for all values of i in the input vectors. The TED graph shown in Figure 7 represents function $C(0) = 4\{A(0)B(0) + A(1)B(3) + A(2)B(2) + A(3)B(1)\}$, which is identical to that of $IFFT(0)$.

Notice that FFT transforms a vector of real numbers (A_i, B_i) into a vector of *complex numbers* (FAB_i). This does not pose any problems for the TED representation that works equally well over the complex domain. The real and imaginary parts of the result of a computation can be stored as separate, properly labeled outputs. In fact, it can be easily shown that TED diagrams can represent polynomial function over an arbitrary Field. The only modification required is that the weights on the edges are no longer real or integer numbers, but elements of the field, and that the composition (multiplication and addition) is performed with the respective operator of the field.

As an example, consider the Galois Field GF(8) defined by the polynomial root $P[X] = X^3 + X + 1$. Let α^0 denote the primitive element of GF(8). Consider a polynomial $Q[XY] = (\alpha^4 X + \alpha^2 Y)(\alpha^3 X + \alpha^1 Y)$. It can be shown that the TED of $Q[XY]$ is isomorphic to the TED of $R[XY] = \alpha^0 X^2 + \alpha^3 Y^2$ which proves the equivalence of the two expression over GF(8).

B. Application to High Level Synthesis

We have recently observed that TED can also be employed in architectural synthesis. In this application, the canonical structure of the TED can serve as a common representation of a generic, infinite precision computation, where all symbolic variables have infinite word width. Several architectural solutions (data flows) can be derived from such a representation. In practice, they will differ according to some metric, such as hardware cost of operators and "noise" due to a finite precision (finite number of operand bits), etc. This point is illustrated in Figure 8. The two solutions differ in cost, complexity and the computation noise; each multiplication introduces a truncation error, while the addition result can be kept exact by adding a single carry-out bit.

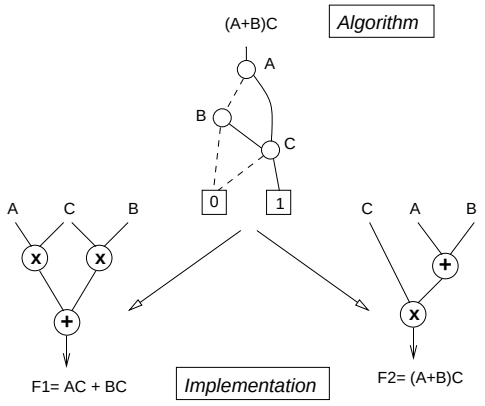

Fig. 8. Application of TED to high level synthesis

Another potential application of TED would use the canonical structure of the TED graph to find common factors in large expressions. Consider for example the following expressions: $F = ab + ac + bc + ad + bd$ and $G = ab + ae + be + ad + bd$, where proper factorization is needed to efficiently represent the computation. Here, the optimal factorization $F = (a + b)(c + d) + ab$, and $G = (a + b)(e + d) + ab$, would result in only four additions and three multiplications for each function. This problem is similar to that of BDD-based logic synthesis that rely on structural BDD decomposition, employed for example by the BDS system [9].

V. Conclusions and Future Work

Based on our initial experience, TED seems to be a promising representation for RTL and algorithmic level verification. It also hold some promise for high level synthesis. Compared to classical high level synthesis, TED offers an opportunity to simplify, factorize and decompose the mathematical expression according to a given pattern, to be efficiently implemented in hardware.

A number of open issues remain to be solved in order to make TED a successful high level verification and/or synthesis tool. One of them is finding an ordering of variables that will minimize the size of the diagram. While good ordering algorithms exist for BDDs in terms of Boolean variables, these techniques will have to be reevaluated in the presence of algebraic and binary variables of TED.

Another aspect of TEDs that deserves future research is their use in symbolic simulation. Current methods use BDDs or textual equations to accumulate symbolic expressions in the course of symbolic simulation. However, the use of BDDs to store practical traces is limited by their exponential size, while textual equations are neither canonical nor reducible. TEDs provide a compact and canonical representation for symbolic computations representing polynomial functions. Thus, the use of TEDs for symbolic simulation appears to be promising.

References

[1] M. Ciesielski, P. Kalla, Z. Zeng, and B. Rouzeyre, "Taylor Expansion Diagrams: A New Representation for RTL Verification," in *IEEE Intl. High Level Design Validation and Test Workshop (HLDVT'01)*, 2001, pp. 70–75.

[2] M. Ciesielski, P. Kalla, Z. Zeng, and B. Rouzeyre, "Taylor Expansion Diagrams: A Compact Canonical Representation with Applications to Symbolic Verification," in *Design Automation and Test in Europe, DATE-02*, 2002, pp. 285–289.

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

[4] K. S. Brace, R. Rudell, and R. E. Bryant, "Efficient Implementation of the BDD Package," in *DAC*, 1990, pp. 40–45.

[5] R. E. Bryant and Y-A. Chen, "Verification of Arithmetic Functions with Binary Moment Diagrams," in *DAC*, 1995.

[6] R. Drechsler, B. Becker, and S. Ruppertz, "The K*BMD: A Verification Data Structure," *IEEE Design & Test*, pp. 51–59, 1997.

[7] Priyank Kalla, *An Infrastructure for RTL Validation and Verification*, Ph.D. thesis, University of Massachusetts Amherst, Dept. of ECE, Amherst, 2002.

[8] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vencentelli, F. Somenzi, A. Aziz, S-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. Ranjan, S. Sarwary, G. Shiple, S. Swamy, and T. Villa, "VIS: A System for Verification and Synthesis," in *Computer Aided Verification*, 1996.

[9] C. Yang and M. Ciesielski, "BDS: A BDD-based Logic Optimization System," *IEEE Trans. on Computer-Aided Design*, vol. 21, no. 7, pp. 866–876, July 2002.