Taylor Expansion Diagrams: A Canonical Representation for Verification of Data Flow Designs

Maciej Ciesielski[1], Priyank Kalla[2], Serkan Askar[1]

[1]Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA-01003

[2]Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT 84112

Accepted for publication in IEEE Transactions on Computers
TC-0118-0404
April 28, 2006

This work has been supported by a grant from the National Science Foundation, CCR-0204146, and in part by the international NSF/CNRS/DAAD supplement grant, INT-0233206.

Abstract—

Taylor Expansion Diagram (TED) is a compact, word-level, canonical representation for data flow computations that can be expressed as multi-variate polynomials. TEDs are based on a decomposition scheme using Taylor series expansion that allows to model word-level signals as algebraic symbols. This power of abstraction, combined with canonicity and compactness of TED, makes it applicable to equivalence verification of dataflow designs. The paper describes the theory of TEDs and proves their canonicity. It shows how to construct a TED from a HDL design specification and discusses application of TEDs in proving equivalence of such designs. Experiments were performed with a variety of designs to observe the potential and limitations of TEDs for dataflow design verification. Application of TEDs to algorithmic and behavioral verification is demonstrated.

I. Introduction

Design verification is the process of ensuring the correctness of designs described at different levels of abstraction during various stages of the design process. Continuous increase in the size and complexity of digital systems has made it essential to address verification issues at early stages of the design cycle. Identification of errors early in the design process can significantly expedite time-to-market and make the design and verification process more efficient. To address this problem, robust, automated verification tools that can handle designs at higher levels of abstraction, such as at behavioral and algorithmic levels, need to be developed.

Having matured over the years, formal design verification methods, such as theorem proving, property and model checking, equivalence checking, etc., have found increasing application in industry. Canonic graph-based representations, such as Binary Decision Diagrams (BDDs) [1], Binary Moment Diagrams (BMDs) [2] and their variants (PHDDs [3], K*BMDs [4], etc.) play an important role in the development of computer-aided verification tools. In particular, these representations have been be used to model RTL designs and prove their equivalence at the bit level. However, these representations are limited in their ability to represent algebraic computations in abstract, higher-level, symbolic forms.

This motivated us to derive a new representation for high-level design descriptions, termed **Taylor Expansion Diagrams** (TEDs). This representation is particularly suited for modeling and supporting equivalence verification of designs specified at the behavioral level [5] [6]. Similar to BDDs and BMDs, TED is a canonical, graph-based representation. In contrast to BDDs and BMDs, TED is based on a non-binary decomposition principle, modeled along the Taylor's series expansion [7], [8]. With this new data structure, word-level signals are represented as algebraic symbols, raising the level of abstraction of the design to higher levels. The power of abstraction of TEDs allows one to represent behavioral dataflow designs efficiently, with memory requirements several orders of magnitude smaller than those of other known representations.

TEDs are applicable to modeling, symbolic simulation and equivalence verification of dataflow and algorithm-dominant designs, such as digital signal processing for audio, video and multimedia applications and embedded systems. Computations performed by those designs can often be modeled as polynomials and readily be represented with TEDs. The test for functional equivalence is then performed by checking isomorphism of the resulting graphs. While application of TEDs is limited to those designs whose computations can be expressed as multi-variate polynomials, their use for algorithmic verification is particularly appealing. Experimental results confirm the potential of TEDs for equivalence verification of dataflow oriented designs at behavioral and algorithmic levels.

The paper is organized as follows. Section II reviews contemporary canonic representations and discusses their limitations. A complete theory of TED is described in Section III. The composition operations and analysis of their complexity are presented in Section IV, while Section V describes the construction of TEDs for RTL designs and discusses their limitations. Section VI describes the implementation of the TED package along with some experimental results. Finally, Section VII concludes the paper with comments and directions for future work.

II. Review of Previous Work

In the realm of high-level design verification, the issue of abstraction of symbolic, word-level computations have received a lot of attention. This is visible in theorem-proving techniques, automated decision procedures for Presburger arithmetic [9] [10], techniques using algebraic manipulation [11], symbolic simulation [12], or in the decision procedures that use "a combination of theories" [13] [14], etc. Term re-writing systems, particularly those used for hardware verification [15] [16] [17], etc., also represent computations in high-level symbolic forms. The above representations and verification techniques, however, do not rely on canonical forms. For example, verification techniques using term rewriting are based on rewrite rules that lead to normal forms. Such forms may produce false negatives, which may be difficult to analyze and resolve. In contrast, the TED representation proposed in this paper not only abstracts bit-vector arithmetic computations as polynomials, but also represents them canonically.

Various forms of *high-level logics* have been used to represent and verify high-level design specifications. Such representations are mostly based on quantifier free fragments of first order logic. The works that deserve particular mention include: the logic of equality with uninterpreted functions (EUF) [18] and with memories (PEUFM) [19] [20], and the logic of counter arithmetic with lambda expressions and uninterpreted functions (CLU) [21]. To avoid exponential explosion of BDDs, equivalence verification is generally performed by transforming high-level logic description of the design into propositional logic formulas [21] [14] [19] and employing satisfiabil-

ity tools [22] [23] for testing the validity of the formulas. While these techniques have been successful in the verification of control logic and pipelined microprocessors, they have found limited application in the verification of large data-path designs.

Word-Level ATPG techniques [24] [25] [26] [27] [28] have also been used for RTL and behavioral verification. However, their applications are generally geared toward simulation, functional vector generation or assertion property checking, but not so much toward high-level equivalence verification of arithmetic datapaths.

A. Decision Diagram Based Representations

Reduced Ordered Binary Decision Diagrams (ROBDDs, or BDDs for short) [1], along with their efficient implementation as software packages [29], are credited with significantly increasing the efficiency of equivalence checking for logic designs. BDD represents a set of binary valued decisions in a rooted directed acyclic graph (DAG), based on a recursive Shannon decomposition. This decomposition, combined with a set of reduction rules, makes the resulting diagram minimal and canonical for a given ordering of variables [1].

BDDs have found wide application in a number of verification problems, including combinational equivalence checking [30], implicit state enumeration [31], symbolic model checking [32] [33], etc. However, as the designs have grown in size and complexity, the size-explosion problems of BDDs have limited their scope. Most BDD-based verification systems, such as SMV [33] and VIS [34], have been successful in verifying control-dominated applications. However, for designs containing large arithmetic data-path units, BDDs have not been very successful due to prohibitive memory requirements, especially for large multipliers.

Numerous attempts have been made to extend the capabilities of verification engines to target arithmetic units. Majority of these methods are based on generic *Word Level Decision Diagrams* (WLDDs), graph-based representations for functions with a Boolean domain and an integer range. Most of the WLDD representations are based on a point-wise, or *binary* decomposition principle. Different flavors of Boolean decomposition (Shannon, Davio, Reed-Muller, etc.) are used to decompose functions w.r.t. their bit-level variables, leading to different *Decision Diagrams*. In addition to BDDs [1] and *Partitioned BDDs* [35], they include edge-valued BDDs (EVBDDs) [36], and functional decision diagrams (FDDs, KFDDs) [37] [38]. By extending BDDs to allow numeric leaf values point-wise decomposition leads to different *Multi-terminal* BDDs, or MTBDDs [39], and Algebraic Decision Diagrams (ADDs) [40]. However, the decomposition at each variable is still binary. As a result, a linear increase in the size of input variables results, in the worst case, in an exponential increase in the size of decision diagrams. A thorough review of WLDDs can be found in [41].

B. Moment Diagram Based Representations

Binary Moment Diagrams, BMDs, *BMDs [2], and their derivatives (PHDDs [42], K*BMD [4], etc.), depart from a point-wise decomposition and perform a decomposition of a linear function based on its first two moments. BMD uses a modified Shannon's expansion, in which a binary variable is treated as a (0,1) integer: $f(x) = x \cdot f_x + x' \cdot f_{x'} = x \cdot f_x \ + \ (1-x) \cdot f_{\overline{x}} = f_{\overline{x}} \ + \ x \cdot (f_x - f_{\overline{x}})$ where "$\cdot$", "+" and "-" denote algebraic multiplication, addition and subtraction, respectively. The above decomposition is termed as *moment* decomposition, where $(f_x - f_{\overline{x}})$ is the *linear moment* and $f_{\overline{x}}$ the *constant moment*. In this form, f can be viewed as a *linear function* in its variables x, and $(f_x - f_{\overline{x}})$ as the partial derivative of f with respect to x.

Binary moment diagrams provide a concise representation of integer-valued functions defined over bit vectors (words), $X = \sum_i 2^i x_i$, where each x_i is a binary variable. The binary moment decomposition is recursively applied to each variable x_i. In such defined BMD, multiplicative constants reside in the terminal nodes. The constants can also be represented as multiplicative terms and assigned to the edges of the graph, giving a rise to *Multiplicative Binary Moment Diagram*, or *BMD [2]. An example of such a diagram is depicted in the left part of Fig. 1. Several rules for manipulating edge weights are imposed on the graph to allow the graph to be canonical. For linear and multi-linear expressions *BMD representation is linear in the number of variables. However, the size of *BMD for X^k, where X is an n-bit vector, is $\mathcal{O}(n^k)$. Thus, for high-degree polynomials defined over words with large bit-width, as commonly encountered in many DSP applications, *BMD remains an expensive representation.

K*BMD [4] attempts to make the BMD decomposition more efficient in terms of the graph size. This is done by admitting multiple decomposition types, such as Davio, Shannon, etc., to be used in a single graph, and allowing both the multiplicative and additive weights assigned to the graph edges. However, a set of restrictions imposed on the edge weights to make it canonical makes such a graph difficult to construct.

C. Symbolic Algebra Methods

Many computations encountered in behavioral design specifications, can be represented in terms of polynomials. This includes digital signal and image processing designs, digital filter designs, and many designs that employ complex transformations, such as DCT, DFT, FFT, etc. Polynomial representations of discrete functions have been explored in literature long before the advent of contemporary canonical graph-based representations. Particularly, Taylor's expansion of Boolean functions has been studied in [43] [44]. However, these works mostly targeted classical switching theory problems: logic minimization, functional decomposition, fault detection, etc. The issue of abstraction of bit-vectors and symbolic representation of computations for high-level synthesis and formal verification was not their focus.

Polynomial models of high-level design specifications have been used recently in the context of behavioral synthesis for the purpose of component mapping [45], [46], [47]. A polynomial representation is created for each component (operator) from the library and the polynomials are matched by comparing their coefficients. However, storing and comparing large matrices of such coefficients is inefficient for large multi-variate polynomials. The TED representation described in this paper can provide a more robust data structure for performing these tasks efficiently, due to its compact and canonical structure.

Several commercial symbolic algebra tools, such as Maple [48], Mathematica [49], and MatLab [50], use advanced symbolic algebra methods to perform efficient manipulation of mathematical expressions. These tools have also been used for the purpose of polynomial mapping, namely to perform simplification modulo polynomial [47]. However, despite the unquestionable effectiveness and robustness of these methods for classical mathematical applications, they are less effective in modeling of large scale digital circuits and systems. We believe that these methods can benefit from canonical representations such as TED, in particular for component matching and equivalence checking.

It is interesting to note that symbolic algebra tools offered by Mathematica and alike cannot unequivocally determine the *equivalence* of two polynomials. The equivalence is checked by subjecting each polynomial to a series of *expand* operations and comparing the coefficients of the two polynomials ordered lexicographically. As stated in the manual of Mathematica 5, Section 2.3.1, *"there is no general way to find out whether an arbitrary pair of mathematical expressions are equal"* [49]. Furthermore, Mathematica *"cannot guarantee that any finite sequence of transformations will take any two arbitrarily chosen expressions to a standard form."* (Mathematica 5, Section 2.62). In contrast, the TED data structure described in the sequel provides an important support for equivalence verification by offering a canonical representation for multi-variate polynomials.

III. Taylor Expansion Diagrams

A known limitation of all decision and moment diagram representations is that *word-level* computations, such as $A + B$, require the decomposition of the function with respect to *bit-level* variables $A[k], B[k]$. Such an expansion creates a large number of variables in the respective diagram framework and requires excessive memory and time to operate upon them. In order to efficiently represent and process the HDL description of a large design, it is desirable to treat the word-level variables as *algebraic symbols*, expanding them into their bit-level components only when necessary.

Consider the *BMD for $A \cdot B$, shown in Fig. 1 (a), which depicts the decomposition with respect to the bits of A and B. It would be desirable to group the nodes corresponding to the individual bits of these variables to *abstract* the integer variables they represent, and use the abstracted variables directly in the design. Fig. 1 depicts the

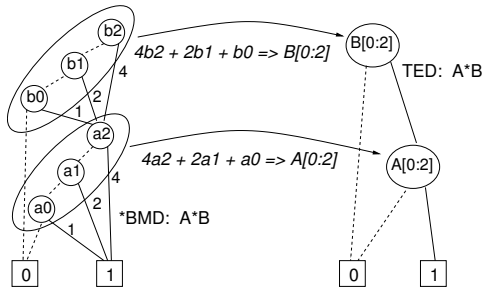

Fig. 1. Abstraction of bit-level variables into algebraic symbols for $F = A \cdot B$.

idea of such a *symbolic abstraction* of variables from their bit-level components.

In order to achieve the type of abstracted representation depicted above, one can rewrite the moment decomposition $f = f_{\overline{x}} + x \cdot (f_x - f_{\overline{x}})$ as $f = f(x=0) + x \cdot \frac{\partial(f)}{\partial x}$. This equation resembles a truncated *Taylor series expansion* of the linear function f with respect to x. By allowing x to take integer values, the binary moment decomposition can be generalized to a Taylor's series expansion. This way one can represent integer variables without expanding them into bits.

A. The Taylor Series Expansion

Let $f(x)$ be a continuous, differentiable function defined over the domain R of real variables. The Taylor series expansion of f w.r.t. variable x at an initial point, $x_0 = 0$, is represented as follows [8]:

$$f(x) \;=\; \sum_{k=0}^{\infty} \frac{1}{k!}(x-x_0)^k f^k(x_0) \;=\; f(0)+xf'(0)+\frac{1}{2}x^2 f''(0)+.... \quad (1)$$

where $f'(x_0)$, $f''(x_0)$, etc., are first, second, and higher order derivatives of f with respect to x, evaluated at $x_0 = 0$. The Taylor series expansion can be suitably adapted to represent computations over integer and Boolean variables, as commonly encountered in HDL descriptions. Arithmetic functions and dataflow portions of those designs, can be expressed as multi-variate polynomials of finite degree for which Taylor series is finite.

Let $f(x, y, \ldots)$ be a real differentiable function in variables $\{x, y, \ldots\}$. Assume an algebra $(R, \cdot, +)$ over real numbers R. Using the Taylor series expansion with respect to a variable x, function f can be represented as: $f(x, y, \ldots) = f(x = 0, y, \cdots) + xf'(x = 0, y, \cdots) + \frac{1}{2}x^2 f"(x = 0, y, \cdots) + ...$ The derivatives of f evaluated at $x = 0$ are independent of variable x, and can be further decomposed w.r.t. the remaining variables, one variable at a time. The resulting recursive decomposition can be represented by a decomposition diagram, called the *Taylor Expansion Diagram*, or TED.

Definition III.1: The **Taylor Expansion Diagram**, *or* **TED**, *is a directed acyclic graph* (Φ, V, E, T)*, representing a multi-variate polynomial expression* Φ*. V is the set of nodes, E is the set of directed edges, and T is the set of terminal nodes in the graph. Every node* $v \in V$ *has an index* $var(v)$ *which identifies the decomposing variable. The function at node v is determined by the Taylor series expansion at* $x = var(v) = 0$*, according to equation 1. The number of edges emanating from node v is equal to the number of nonempty derivatives of f (including* $f(0)$*) w.r.t. variable* $var(v)$*. Each edge points to a subgraph whose function evaluates to the respective derivative of the function with respect to* $var(v)$*. Each subgraph is recursively defined as TED w.r.t. the remaining variables. Terminal nodes evaluate to constants.*

Starting from the *root*, the decomposition is applied recursively to the subsequent children nodes. The internal nodes are in one-to-one correspondence with the successive derivatives of function f w.r.t. variable x evaluated at $x = 0$. Figure 2 depicts one-level decomposition of function f at variable x. The k-th derivative of a function rooted at node v with $var(v) = x$ is referred to as a *k-child* of v; $f(x{=}0)$ is a 0-child, $f'(x{=}0)$ is a 1-child, $\frac{1}{2!}f''(x{=}0)$ is a 2-child, etc. We shall also refer to the corresponding arcs as *0-edge* (dotted), *1-edge* (solid), *2-edge* (double), etc.

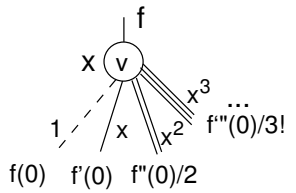

Fig. 2. A decomposition node in a TED.

Example: Figure 3 shows the construction of a TED for the algebraic expression $F = (A+B)(A+2C) = A^2+A\cdot(B+2\cdot C+2\cdot B\cdot C)$. Let the ordering of variables be A, B, C. The decomposition is performed first with respect to variable A. The constant term of the Taylor expansion $F(A = 0) = 2 \cdot B \cdot C$. The linear term of the expansion gives $F'(A = 0) = B+2\cdot C$; the quadratic term is $\frac{1}{2}\cdot F''(A = 0) = \frac{1}{2}\cdot 2 = 1$. This decomposition is depicted in Fig. 3 (a). Now the Taylor series expansion is applied recursively to the resulting terms with respect to variable B, as shown in Fig. 3(b), and subsequently with respect to variable C. The resulting diagram is depicted in Fig. 3(c), and its final reduced and normalized version (to be explained in Section III-B) is shown in Fig. 3(d). The function encoded by the TED can be evaluated by adding all the paths from non-zero terminal nodes to the root, each path being a product of the variables in their respective powers and the edge weights, resulting in $F = A^2+AB+2AC+2BC$.

Using the terminology of computer algebra [51], TED employs a *sparse recursive representation*, where a multivariate polynomial $p(x_1, \cdots, x_n)$ is represented as:

$$p(x_1, \cdots, x_n) = \sum_{i=0}^{m} p_i(x_1, \cdots, x_{n-1}) x_n^i \quad (2)$$

The individual polynomials $p_i(x_1, \cdots, x_{n-1})$ can be viewed as "coefficients" of the leading variable x_n at the decomposition level corresponding to x_n. By construction, the sparse form stores only non-zero polynomials as the nodes of the TED.

B. Reduction and Normalization

It is possible to further reduce the size of an ordered TED by a process of TED *reduction* and *normalization*. Analogous to BDDs and *BMDs, Taylor Expansion Diagrams can be reduced by removing redundant nodes and merging isomorphic subgraphs. In general, a node is redundant if it can be removed from the graph, and its incoming edges can be redirected to the nodes pointed to by the outgoing edges of the node, without changing the function represented by the diagram.

Definition III.2: A TED node is **redundant** *if all of its non-0 edges are connected to terminal 0.*

If node v contains only a constant term (0-edge), the function computed at that node does not depend on the variable $var(v)$, associated with the node. Moreover, if all the edges at node v point to the terminal node 0, the function computed at the node evaluates to zero. In both cases, the parent of node v is reconnected to the 0-child of v, as depicted in Fig. 4.

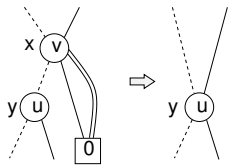

Fig. 4. Removal of redundant node with only a constant term edge.

Identification and merging of isomorphic subgraphs in a TED are analogous to that of BDDs and *BMDs. Two TEDs are considered *isomorphic* if they match in both their structure and their attributes; *i.e.* if there is a one-to-one mapping between the vertex sets and the edge sets of the two graphs that preserve vertex adjacency, edge labels and terminal leaf values. By construction, two isomorphic TEDs represent the same function.

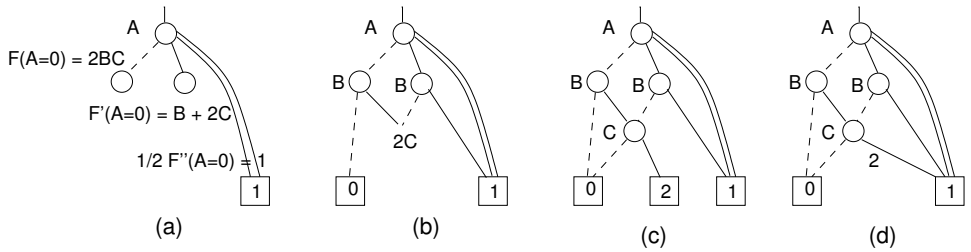

Fig. 3. Construction of a TED for $F = (A + B)(A + 2C)$: (a)-(c) decomposition w.r.t. individual variables; (d) normalized TED

In order to make the TED canonical, any redundancy in the graph must be eliminated and the graph must be reduced. The reduction process entails merging the isomorphic sub-graphs and removing redundant nodes.

Definition III.3: A Taylor expansion diagram is **reduced** *if it contains no redundant nodes and has no distinct vertices v and v', such that the subgraphs rooted at v and v' are isomorphic. In other words, each node of the reduced TED must be unique.*

It is possible to further reduce the graph by performing a procedure called *normalization*, similar to the one described for *BMDs [2]. The normalization procedure starts by moving the numeric values from the non-zero terminal nodes to the terminal edges, where they are assigned as edge *weights*. This is shown in Fig. 3(d) and Fig. 5(b). By doing this, the terminal node holds constant 1. This operation applies to all terminal edges with terminal nodes holding values different than 1 or 0. As a result, only terminal nodes 1 and 0 are needed in the graph. The weights at the terminal edges may by further propagated to the upper edges of the graph, depending on their relative values. The TED normalization process that accomplishes this is defined as follows.

Definition III.4: A reduced, ordered TED representation is **normalized** *when:*

- *The weights assigned to the edges spanning out of a given node are relatively prime.*
- *Numeric value 0 appears only in the terminal nodes.*
- *The graph contains no more than two terminal nodes, one each for 0 and 1.*

By ensuring that the weights assigned to the edges spanning out of a node are relatively prime, the extraction of common subgraphs is enabled. Enforcing the rule that none of the edges be allowed zero weight is required for the canonization of the diagram. When all the edge weights have been propagated up to the edges, only the value 0 and 1 can reside in the terminal nodes.

The normalization of the TED representation is illustrated by an example in Fig. 5. First, as shown in Fig. 5(b), the constants (6, 5) are moved from terminal nodes to terminal edges. These weights are then propagated up along the linear edges to the edges rooted at nodes associated with variable B, see Fig. 5(c). At this point the isomorphic subgraphs $(B+C)$ are identified at the nodes of B and the graph is subsequently reduced by merging the isomorphic subgraphs, as shown in Fig. 5(d).

It can be shown that normalization operation can reduce the size of a TED exponentially. Conversely, transforming a normalized TED to a non-normalized TED can, in the worst-case, result in an exponential increase in the graph size. This result follows directly from the concepts of normalization of BMDs to *BMDs [2].

C. Canonicity of Taylor Expansion Diagrams

It now remains to be shown that an ordered, reduced and normalized Taylor Expansion Diagram is canonical; *i.e.* for a fixed ordering of variables, any algebraic expression is represented by a unique reduced, ordered and normalized TED. First, we recall the following Taylor's Theorem, proved in [8].

Theorem 1 (Taylor's Theorem [8]) *Let $f(x)$ be a polynomial function in the domain R, and let $x = x_0$ be any point in R. There exists one and only one unique Taylor's series with center x_0 that represents $f(x)$ according to the equation 1.*

The above theorem states the uniqueness of the Taylor's series representation of a function, evaluated at a particular point (in our case at $x = 0$). This is a direct consequence of the fact that the successive derivatives of a function evaluated at a point are unique. Using the Taylor's theorem and the properties of reduced and normalized TEDs, it can be shown that an ordered, reduced and normalized TED is canonical.

Theorem 2: For any multivariate polynomial f with integer coefficients, there is a unique (up to isomorphism) ordered, reduced and normalized Taylor Expansion Diagram denoting f, and any other Taylor Expansion Diagram for f contains more vertices. In other words, an ordered, reduced and normalized TED is minimal and canonical.

Proof: The proof of this theorem follows directly the arguments used to prove the canonicity and minimality of BDDs [1] and *BMDs [2].

Uniqueness. First, a reduced TED has no trivial redundancies; the redundant nodes are eliminated by the reduce operation. Similarly, a reduced TED does not contain any isomorphic subgraphs. Moreover, after the normalization step, all common subexpressions are shared by further application of the reduce operation. By virtue of the Taylor's Theorem all the nodes in an ordered, reduced and normalized TED are unique and distinguished.

Canonicity. We now show that the individual Taylor expansion terms, evaluated recursively, are uniquely represented by the internal nodes of the TED. First, for polynomial functions the Taylor series expansion at a given point is finite and, according to the Taylor's Theorem, the series is unique. Moreover, each term in the Taylor's series corresponds to the successive derivatives of the function evaluated at that point. By definition, the derivative of a differentiable function evaluated at a particular point is also unique. Since the nodes in the TED correspond to the recursively computed derivatives, every node in the diagram uniquely represents the function computed at that node. Since every node in an ordered, reduced and normalized TED is distinguished and it uniquely represents a function, the Taylor Expansion Diagram is canonical.

Minimality. We now show that a reduced, ordered and normalized TED is also minimal. This can be proved by contradiction. Let G be a graph corresponding to a reduced, normalized and hence canonical TED representation of a function f. Assume there exists another graph G', with the same variable order as in G, representing f that is smaller in size than G. This would imply that graph G could be reduced to G' by the application of reduce and normalize operations. However, this is not possible as G is a reduced and normalized representation and contains no redundancies. The sharing of identical terms across different decomposition levels in the graph G has been captured by the reduction operation. Thus G' cannot have a representation for f with fewer nodes than G. Hence G is a minimal and canonical representation for f. □

D. Complexity of Taylor Expansion Diagrams

Let us now analyze the worst-case size complexity of an ordered and reduced Taylor Expansion Diagram. For a polynomial function of degree k, decomposition with respect to a variable can produce $k+1$ *distinct* Taylor expansion terms in the worst-case.

Theorem 3: Let f be a polynomial in n variables and maximum degree k. In the worst case, the ordered, reduced, normalized Taylor Expansion Diagram for f requires $\mathcal{O}(k^{n-1})$ nodes and $\mathcal{O}(k^n)$ edges.

Proof: The top-level contains only one node, corresponding to the first variable. Since its maximum degree is k, the number of distinct children nodes at the second level is bounded by $k+1$. Similarly, each of the nodes at this level produces up to $k+1$ children nodes at the next level, giving a rise to $(k+1)^2$ nodes, and so on. In the worst case the number of children increases in geometric progression, with the level i containing up to $(k+1)^{i-1}$ nodes. For an n-variable function,

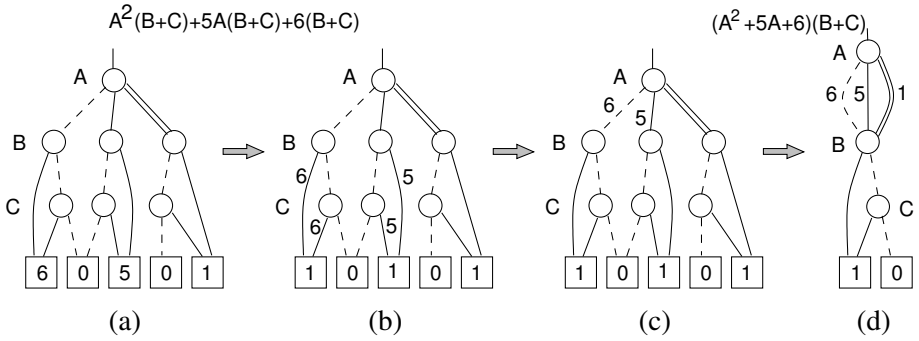

Fig. 5. Normalization of the TED for $F = (A^2 + 5A + 6)(B + C)$

there will be n-1 such levels, with the n-th level containing just two terminal nodes, 1 and 0. Hence the total number of internal nodes in the graph is $N = \sum_{i=0}^{n-1}(k+1)^i = \frac{(k+1)^n-1}{k}$. The number of edges E can be similarly computed as $E = \sum_{i=1}^{n}(k+1)^i = \frac{(k+1)^{n+1}-1}{k} - 1$, since there may be up to $(k+1)^n$ terminal edges leading to the 0 and 1 nodes. Thus, in the worst-case, the total number of internal nodes required to represent an n-variable polynomial with degree k is $\mathcal{O}(k^{n-1})$ and the number of edges is $\mathcal{O}(k^n)$. □

One should keep in mind, however, that the TED variables represent symbolic, word-level signals, and the number of such signals in the design is significantly smaller than the number of bits in the bit-level representation. Subsequently, even an exponential size of the polynomial with a relatively small number of such variables may be acceptable. Moreover, for many practical designs the complexity is not exponential.

Finally, let us consider the TED representation for functions with variables encoded as n-bit vectors, $X = \sum_{i=0}^{n-1} 2^i x_i$. For linear expressions, the space complexity of TED is linear in the number of bits n, the same as *BMD. For polynomials of degree $k \geq 2$, such as X^2, etc., the size of *BMD representation grows polynomially with the number of bits, as $\mathcal{O}(n^k)$. For K*BMD the representation also becomes nonlinear, with complexity $\mathcal{O}(n^{k-1})$, for polynomials of degree $k \geq 3$. However, for ordered, reduced and normalized TEDs, the graph remains linear in the number of bits, namely $\mathcal{O}(n \cdot k)$, for any degree k, as stated in the following theorem.

Theorem 4: Consider variable X encoded as an n-bit vector, $X = \sum_{i=0}^{n-1} 2^i x_i$. The number of internal TED nodes required to represent X^k in terms of bits x_i, is $k(n-1)+1$.

Proof: We shall first illustrate it for the quadratic case $k = 2$. Let W_n be an n-bit representation of X: $X = W_n = \sum_{i=0}^{n-1} 2^i x_i = 2^{(n-1)} x_{n-1} + W_{n-1}$ where $W_{n-1} = \sum_{i=0}^{n-2} 2^i x_i$ is the part of X containing the lower (n-1) bits. With that, $W_n^2 = (2^{n-1}x_{n-1} + W_{n-1})^2 = 2^{2(n-1)}x_{n-1}^2 + 2^n x_{n-1} W_{n-1} + W_{n-1}^2$. Furthermore, let $W_{n-1} = (2^{n-2}x_{n-2} + W_{n-2})$, and $W_{n-1}^2 = (2^{2(n-2)}x_{n-2}^2 + 2^{n-1}x_{n-2}W_{n-2} + W_{n-2}^2)$.

Notice that the constant term (0-edge) of W_{n-1} w.r.to variable x_{n-2} contains the term W_{n-2}, while the linear term (1-edge) of W_{n-1}^2 contains $2^{n-1}W_{n-2}$. This means that the term W_{n-2} can be *shared* at this decomposition level by two different parents.As a result, there are exactly two non-constant terms, W_{n-2} and W_{n-2}^2 at this level.

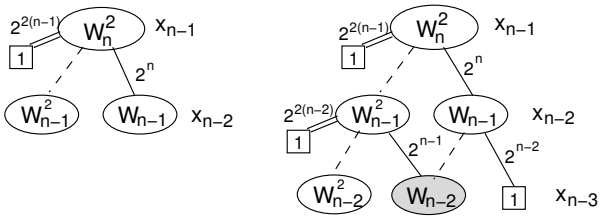

Fig. 6. Construction of TED for X^2 with n bits

In general, at any level l, associated with variable x_{n-l}, the expansion of terms W_{n-l}^2 and W_{n-l} will create *exactly two* different non-constant terms, one representing W_{n-l-1}^2 and the other W_{n-l-1}; plus a constant term 2^{n-l}. The term W_{n-l-1} will be shared, with different multiplicative constants, by W_{n-l}^2 and W_{n-l}.

This reasoning can be readily generalized to arbitrary integer degree k; at each level there will always be exactly k different non-constant terms. Since on the top variable (x_{n-1}) level there is only one node (the root), and there are exactly k non-constant nodes at each of the remaining $(n-1)$ levels, the total number of nodes is equal to $k(n-1)+1$. □

E. Limitations of Taylor Expansion Diagram Representation

It should be obvious from the definition of TED that it can only represent those functions that have *finite* Taylor expansion, and in particular multi-variate polynomials with finite integer degrees. For polynomials of finite integer degree $k \geq 1$, successive differentiation of the function ultimately leads to zero, resulting in a finite number of terms. However, those functions that have infinite Taylor series (such as a^x, where a is a constant) cannot be represented with a finite TED graph.

Another natural limitation of TEDs is that they cannot represent relational operators (such as comparators, $A \geq B, A == B$, etc.) in symbolic form. This is because Taylor series expansion is defined for functions and not for relations. Relations are characterized by discontinuities over their domain and are not differentiable. In order to use TEDs to represent relational operators, often encountered in RTL descriptions, the expansion of word-level variables and bit vectors into their bit-level components is required. Finally, TEDs cannot represent modular arithmetic. This issue will be discussed in Section V-C in the context of RTL verification.

IV. Composition Operations for Taylor Expansion Diagrams

Taylor Expansion Diagrams can be composed to compute complex expressions from simpler ones. This section describes general composition rules to compute a new TED as an algebraic sum (+) or product ($\cdot$) of two TEDs. The general composition process for TEDs is similar to that of the APPLY operator for BDD's [1], in the sense that the operations are recursively applied on respective graphs. However, the composition rules for TEDs are specific to the rules of the algebra $(R, \cdot, +)$.

Starting from the roots of the two TEDs, the TED of the result is constructed by recursively constructing all the non-zero terms from the two functions, and combining them, according to a given operation, to form the diagram for the new function. To ensure that the newly generated nodes are unique and minimal, the REDUCE operator is applied to remove any redundancies in the graph.

Let u and v be two nodes to be composed, resulting in a new node q. Let $var(u) = x$ and $var(v) = y$ denote the decomposing variables associated with the two nodes. The top node q of the resulting TED is associated with the variable with the higher order, i.e., $var(q) = x$, if $x \geq y$, and $var(q) = y$ otherwise. Let f, g be two functions rooted at nodes u, v, respectively, and h be a function rooted at the new node q.

For the purpose of illustration, we describe the operations on linear expressions, but the analysis is equally applicable to polynomials

of arbitrary degree. In constructing these basic operators, we must consider several cases:
1. Both nodes u, v are terminal nodes. In this case a new *terminal* node q is created as $val(q) = val(u) + val(v)$ for the ADD operation, and as $val(q) = val(u) \cdot val(v)$ for the MULT operation.
2. At least one of the nodes is non-terminal. In this case the TED construction proceeds according to the variable order. Two cases need to be considered here: (a) when the top nodes u, v have the same index, and (b) when they have different indices. The detailed analysis of both cases is given in [6]. Here we show the multiplication of two diagrams rooted at variables u and v with the same index.

$$h(x) = f(x) \cdot g(x) = (f(0) + xf'(0)) \cdot (g(0) + xg'(0)) \quad (3)$$
$$= [f(0)g(0)] + x[f(0)g'(0) + f'(0)g(0)] + x^2[f'(0)g'(0)].$$

In this case, the 0-child of q is obtained by pairing the 0-children of u, v. Its 1-child is created as a sum of two cross products of 0- and 1-children, thus requiring an additional ADD operation. Also, an additional 2-child (representing the quadratic term) is created by pairing the 1-children of u, v.

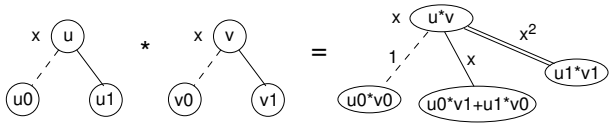

Fig. 7. Multiplicative composition for nodes with same variables.

Figure 8 illustrates the application of the ADD and MULT procedures to two TEDs. As shown in the figure, the root nodes of the two TEDs have the same variable index. The MULT operation requires the following steps: (i) performing the multiplication of their respective constant (0-) and linear (1-) children nodes; and (ii) generating the sum of the cross-products of their 0- and 1-children. On the other hand, the two TEDs corresponding to the resulting cross-product, as highlighted in the figure, have different variable indices for their root nodes. In this case, the node with the lower index corresponding to variable C is added to the 0-child of the node corresponding to variable B.

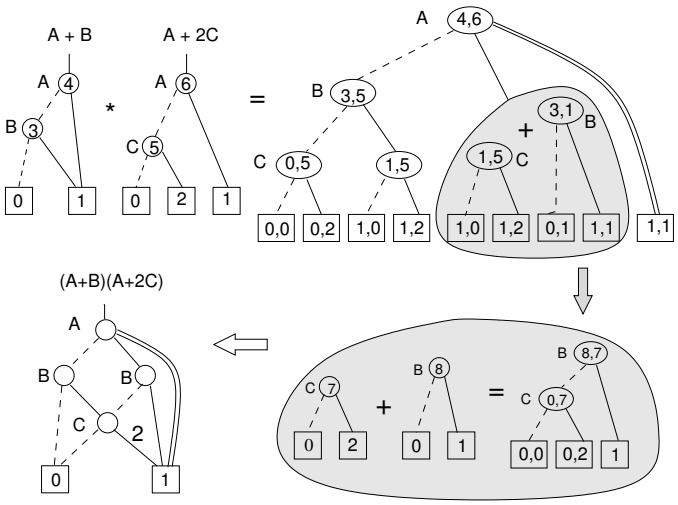

Fig. 8. Example of MULT composition: (A+B)(A+2C).

It should be noted that the ADD and MULT procedures described above will initially produce non-normalized TEDs, with numeric values residing only in the terminal nodes, requiring further normalization. When these operations are performed on normalized TEDs, with weights assigned to the edges, then the following modification is required: when the variable indices of the root nodes of f and g are different, the edge weights have to be propagated down to the children nodes recursively. Downward propagation of edge weights results in the dynamic update of the edge weights of the children nodes. In each recursion step, this propagation of edge weights down to the children proceeds until the weights reach the terminal nodes. The numeric values are updated only in the terminal nodes. Every time a new node is created, the REDUCE and NORMALIZE operations are required to be performed in order to remove any redundancies from the graph and generate a minimal and canonical representation.

We now analyze the computational complexity of the basic TED operations described above. Let $|f|$ and $|g|$ be the size, expressed in the number of nodes, of the two TEDs, f and g, respectively. The number of recursive calls made to ADD is bounded by $\leq (|f| \cdot |g|)$. The MULT operation has higher complexity than ADD. The worst case for the multiply operation would occur when each node in f is multiplied by each node in g, resulting in $(|f| \cdot |g|)$ recursive MULT calls. However, each multiply operation further relies on $(|f| \cdot |g|)$ recursive calls to the ADD operation, in the worst case (see Fig. 8).

In order to derive an absolute worst-case upper bound for the composition operations, we have to consider the case where edge-weights have to be propagated all the way down to terminal nodes. In such cases, non-normalized TEDs are dynamically created from their normalized counterparts. As discussed in Section III-B, non-normalized TEDs can be exponentially more complex than normalized TEDs. This may result in an exponential worst-case complexity of the composition operations. However, it should be noted that the number of calls to the MULT operation can be efficiently reduced by using a computed table to store the results, as is done in the recent implementation. A detailed analysis of the complexity of basic TED operations is presented in [52]. It should be noted that the multiplication of two *multi-variate* polynomials has been shown to be exponential. For instance, time complexity of Karatsuba algorithm for multiplying two n-variate polynomials with maximum degree d is $\mathcal{O}((d+1)^{n\ log_2 3})$ [51].

At the first glance, time complexity for TED construction appears to be prohibitive. However, we observed that for dataflow computations specified at sufficiently high level (see Section VI) the composition operations do not exhibit exponential complexity. The number of symbolic variables is orders of magnitude smaller than that of Boolean variables present in the *BMD and BDD representations. Exponential complexity can be observed in cases when the Boolean variables start dominating in the design, in which case the behavior of TED starts approaching that of a *BMD.

V. Design Modeling and Verification with TEDs

Using the operations described in the previous section, Taylor Expansion Diagrams can be constructed to represent various computations over symbolic variables in a compact, canonical form. The compositional operators ADD and MULT can be used to compute any combination of arithmetic functions by operating directly on their TEDs. However, the representation of Boolean logic, often present in the RTL designs, requires special attention since the output of a logic block must evaluate to Boolean rather than to an integer value.

A. Representing Boolean Logic

We now define TED operators for Boolean logic, OR, AND, and XOR, where both the range and domain are Boolean. Figure 9 shows TED representations for these basic Boolean operators. In the diagrams, x and y are Boolean variables represented by binary variables, and $+$ and $\cdot$ represent algebraic operators of ADD and MULT, respectively. The resulting functions are 0,1 integer functions. These diagrams are structurally identical to their *BMD counterparts [53] [2].

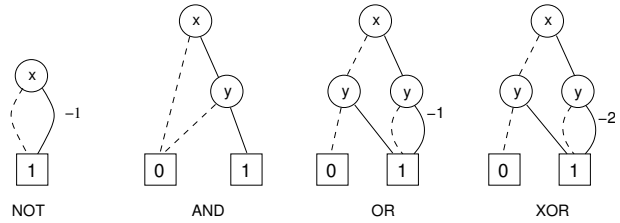

Fig. 9. TED representation for Boolean operators: a) NOT: $x' = (1 - x)$; b) AND: $x \wedge y = x \cdot y$; c) OR: $x \vee y = x + y - xy$; d) XOR: $x \oplus y = x + y - 2xy$.

Similarly one can derive other operators which rely on Boolean variables as one of their inputs, with other inputs being word-level.

One such example is the multiplexer, $\text{MUX}(c, X, Y) = c \cdot X + (1-c) \cdot Y$, where c is a binary control signal, and X and Y are word-level inputs.

In general, TED, which represents an integer-valued function, will also correctly model designs with arithmetic and Boolean functions. Note that the ADD (+) function will always create correct integer result over Boolean and integer domains, because Boolean variables are treated as binary (0,1), a special case of integer. However the MULT (·) function may create powers of Boolean variables, x^k, which should be reduced to x. A minor modification of TED is done to account for this effect so that the Boolean nature of variable x can be maintained in the representation. Such modified Taylor Expansion Diagrams are also canonical.

B. Verification of RTL and Behavioral Designs

TED construction for an RTL design starts with building trivial TEDs for primary inputs. Partial expansion of the word-level input signals is often necessary when one or more bits from any of the input signals fan out to other parts of the design. This is the case in the designs shown in Fig. 10 (a) and (b), where bits $a_k = A[k]$ and $b_k = B[k]$ are derived from word-level variables A and B. In this case, the word-level variables must be decomposed into several word-level variables with shorter bit-widths. In our case, $A = 2^{(k+1)}A_{hi} + 2^k a_k + A_{lo}$ and $B = 2^{(k+1)}B_{hi} + 2^k b_k + B_{lo}$, where $A_{hi} = A[n-1{:}k{+}1]$, $a_k = A[k]$, and $A_{lo} = A[k\text{-}1{:}0]$; and similarly for variable B. Variables $A_{hi}, a_k, A_{lo}, B_{hi}, b_k, B_{lo}$ form the *abstracted* primary inputs of the system. The basic TEDs are readily generated for these abstracted inputs from their respective bases (A_{hi}, a_k, A_{lo}), and (B_{hi}, b_k, B_{lo}).

Once all the abstracted primary inputs are represented by their TEDs, Taylor Expansion Diagrams can be constructed for all the components of the design. TEDs for the primary outputs are then generated by systematically composing the constituent TEDs in the topological order, from the primary inputs to the primary outputs. For example, to compute $A + B$ in Fig. 10 (a) and (b), the ADD operator is applied to functions A and B (each represented in terms of their abstracted components). The subtract operation, $A - B$, is computed by first multiplying B with a constant -1 and adding the result to the TED of A. The multipliers are constructed from their respective inputs using the MULT operator, and so on. To generate a TED for the output of the multiplexers, the Boolean functions s_1 and s_2 first need to be constructed as TEDs. Function s_1 is computed by transforming the single-bit comparator $a_k > b_k$ into a Boolean function and expressed as an algebraic equation, $s_1 = a_k \wedge \overline{b_k} = a_k \cdot (1 - b_k)$, as described in Section V-A. Similarly, $s_2 = \overline{a_k} \vee b_k$ is computed as $s_2 = 1 - a_k \cdot (1 - b_k)$ and represented as a TED. Finally, the TEDs for the primary outputs are generated using the MUX operator with the respective inputs. As a result of such a series of composition operations, the outputs of the TED represent multivariate polynomials of the primary inputs of the design.

After having constructed the respective ordered, reduced, and normalized Taylor Expansion Diagram for each design, the test for functional equivalence is performed by checking for isomorphism of the resulting graphs. If the corresponding diagrams are isomorphic, they represent equivalent functions. Fig. 10(c) shows the *isomorphic* TED for the two designs, demonstrating that they are indeed equivalent. In fact, the generation of the TEDs for the two designs under verification takes place in the same TED manager; when the two functions are equivalent, both top functions point to the same root of the common TED.

C. Limitations of TEDs in RTL Verification

The proposed TED representation naturally applies to functions that can be modeled as *finite polynomials*. However, the efficiency of TED relies on its ability to encode the design in terms of its *word-level* symbolic inputs, rather than bit-level signals. This is the case with the simple RTL designs shown in Figure 10, where all input variables and internal signals have simple, low-degree polynomial representation. The abstracted word-level inputs of these designs are created by partial bit selection (a_k, b_k) at the primary inputs, and a polynomial function can be constructed for its outputs. However, if any of the internal or output signals is partitioned into sub-vectors, such sub-vectors cannot be represented as polynomials in terms of the symbolic, word-level input variables, but depend on the individual bits of the inputs. The presence of such signal splits creates a fundamental problem for the polynomial representations, and TEDs cannot be used efficiently in those cases. For similar reasons TED cannot represent modular arithmetic. An attempt to fix this problem was proposed in [54], by modeling the discrete functions as finite, word-level polynomials in Galois Field (GF). The resulting polynomials, however, tend to be of much higher degree than the original function, with the degree depending on the signal bit-width, making the representation less efficient for practical applications. This is the case where TEDs can exhibit space explosion similar to that encountered in BDDs and BMDs.

Despite these limitation, TEDs can be successfully used for verifying equivalence of high-level, behavioral and algorithmic descriptions. Such algorithmic descriptions typically do not exhibit signal splits, hence resulting in polynomial functions over word-level input signals.

VI. Implementation and Experimental Results

We have implemented a prototype version of TED software for behavioral HDL designs using as a front end a popular high-level synthesis system GAUT [55]. This system was selected due to its commercial quality, robustness, and its open architecture. The input to the system is behavioral VHDL or C description of the design. The design is parsed and the extracted data flow is automatically transformed into canonical TED representation.

The core computational platform of the TED package consists of a *manager* that performs the construction and manipulation of the graph. It provides routines to uniquely store and manipulate the nodes, edges and terminal values, in order to keep the diagrams canonical. To support canonicity, the nodes are stored in a hash table, implemented as *unique table*, similar to that of the CUDD package [29],[56]. The table contains a *key* for each vertex of the TED, computed from the node index and the attributes of its children and the edge weights. As a result, equivalence test between two TEDs reduces to a simple scalar test between the identifiers of the corresponding vertices.

Variable ordering. As shown in this paper, TEDs are a canonical representation subject to the imposition of a total ordering on the variables. Therefore it is desirable to search for a variable order that would minimize the size of TEDs. We have recently developed a dynamic variable ordering for TEDs based on local swapping of adjacent variables in the diagram, similar to those employed in BDD ordering [57], [58]. It has been shown that, similarly to BDDs, local swapping of adjacent variables does not affect the structure of the diagram outside of the swapping area. We are currently experimenting with different static ordering heuristics, including the ordering of variables that correspond to constant coefficients. Due to initial nature of these heuristics, in our initial experiments we have used the default (topological) order in which the signals appear in the design specification.

A. Experimental Setup

Several experiments were performed using our prototype software on a number of dataflow designs described in behavioral VHDL. The designs range from simple algebraic (polynomial) computations to those encountered in signal and image processing algorithms. Simple RTL designs with Boolean-algebraic interface were also tested. We wish to emphasize that the goal of these experiments was to demonstrate, as a proof of concept, the application of TED to high-level dataflow design representation and verification, and in particular to functional equivalence checking of behavioral HDL specifications, rather than to develop a complete equivalence verification system.

Our experimental setup is as follows. The design described in behavioral VHDL or C is parsed by a high-level synthesis system GAUT [55]. The extracted data flow is then automatically translated into a canonical TED representation using our software. Statistics related to graph size and composition time are reported. We have compared TEDs against *BMDs to demonstrate the power of abstraction of TED representation. For this purpose, each design was synthesized into a structural netlist from which *BMDs were constructed. In most cases BDDs could not be constructed due to their prohibitive size, and they are not reported. Experiments confirm that word-size abstraction by TEDs results in much smaller graph size and computation times as compared to *BMDs.

B. Verification of High-level Transformations

During the process of architectural synthesis, the initial HDL description often proceeds through a series of high-level transformations.

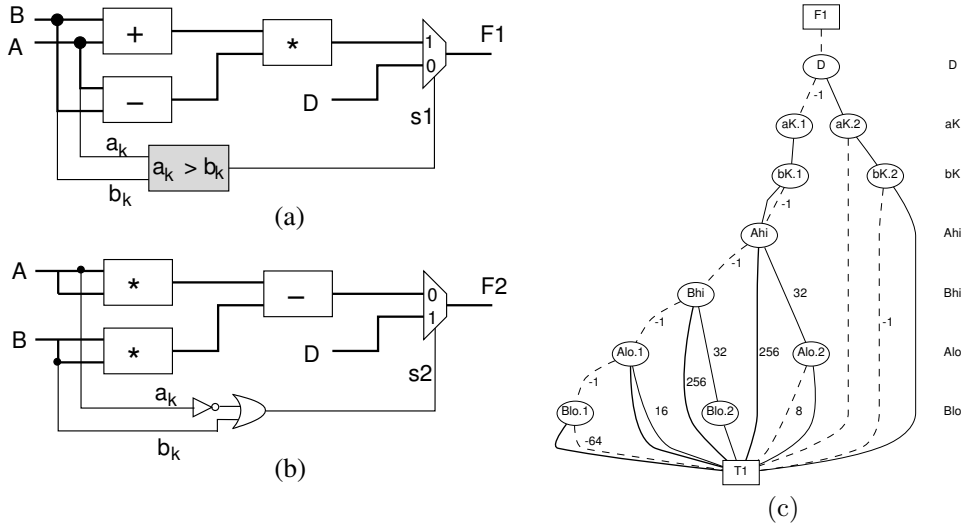

Fig. 10. RTL verification using canonical TED representation: (a), (b) Functionally equivalent RTL modules; (c) The isomorphic TED for the two designs.

For example, computation $AC+BC$ can be transformed into an equivalent one, $(A+B)C$, which better utilizes the hardware resources. TEDs are ideally suited to verify the correctness of such transformations by proving equivalence of the two expressions, regardless of the word size of the input/output signals. We performed numerous experiments to verify the equivalence of such algebraic expressions. Results indicate that both time and memory usage required by TEDs is orders of magnitude smaller as compared to *BMDs. For example, the expression $(A+B)(C+D)$, where A, B, C, D are n-bit vectors, has a TED representation containing just 4 internal nodes, regardless of the word size. The size of *BMD for this expression varies from 418 nodes for the 8-bit vectors, to 2,808 nodes for 32-bit variables. BDD graphs could not be constructed for more than 15 bits.

C. RTL Verification

As mentioned earlier, TEDs offer the flexibility of representing designs containing both arithmetic operators and Boolean logic. We used the generic designs of Figure 10 and performed a set of experiments to observe the efficiency of TED representation under varying size of Boolean logic. The size of the algebraic signals A, B was kept constant at 32 bits, while the word size of the comparator (or the equivalent Boolean logic) was varied from 1 to 20. As the size of Boolean logic present in the design increases, the number of bits extracted from A, B also increases (the figure shows it for single bits). Table I gives the results obtained with TED and compares them to those of *BMDs. Note that, as the size of Boolean logic increases, TED size converges to that of *BMD. This is to be expected as *BMDs can be considered as a special (Boolean) case of TEDs.

TABLE I
SIZE OF TED VS. BOOLEAN LOGIC

bits (k)	*BMD		TED	
	Size	CPU time	Size	CPU
4	4620	107 s	194	44 s
8	15K	87 s	998	74 s
12	19K	93 s	999	92 s
16	23.9K	249 s	4454	104 s
18	timeout	>12 hrs	12.8K	29 min
20	timeout	>12 hrs	timeout	>12 hrs

D. Array Processing

An experiment was also performed to analyze the capability of TEDs to represent computations performed by an array of processors. The design that was analyzed is an $n \times n$ array of configurable Processing Elements (PE), which is a part of a low power motion estimation architecture [59]. Each processing element can perform two types of computations on a pair of 8-bit vectors, A_i, B_i, namely $(A_i - B_j)$ or $(A_i^2 - B_j^2)$, and the final result of all PEs is then added together. The size of the array was varied from 4×4 to 16×16, and the TED for the final result was constructed for each configuration.

When the PEs are configured to perform subtraction $(A_i - B_j)$, both TEDs and *BMDs can be constructed for the design. However, when the PEs are configured to compute $A_i^2 - B_j^2$, the size of *BMDs grows quadratically. As a result, we were unable to construct *BMDs for the 16×16 array of 8-bit processors. In contrast, the TEDs were constructed easily for all the cases. The results are shown in Table II. Note that we were unable to construct the BDDs for any size n of the array for the quadratic computation.

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

Array size ($n \times n$)	*BMD		TED	
	Size	CPU time	Size	CPU time
4× 4	123	3 s	10	1.2 s
6× 6	986	3.4 s	14	1.5 s
8× 8	6842	112 s	18	1.6 s
16× 16	out of mem	-	34	8.8 s

E. DSP Computations

One of the most suitable applications for TED representation are algorithmic descriptions of dataflow computations, such as digital signal and image processing algorithms. For this reason, we have experimented with the designs that implement various DSP algorithms.

Table III presents some data related to the complexity of the TEDs constructed for these designs. The first column in the Table describes the computation implemented by the design. These include: FIR and IIR filters, fast Fourier transform (FFT), elliptical wave filter ($Elliptic$), least mean square computation ($LMS128$), discrete cosine transform (DCT), matrix product computation ($ProdMat$), Kalman filter ($Kalman$), etc. Most of these designs perform algebraic computations by operating on vectors of data, which can be of arbitrary size. The next column gives the number of inputs for each design. While each input is a 16-bit vector, TED represents them as word-level symbolic variable. Similarly, the next column depicts the number of 16-bit outputs. The remaining columns of the table show: BMD size (number of nodes), CPU time required to construct the BMD for the 16-bit output words, TED size (number of nodes) required to represent the entire design; CPU times required to generate TED diagrams does not account for the parsing time of the GAUT front end.

TABLE III
SIGNAL PROCESSING APPLICATIONS

Design	No. of inputs	No. of outputs	*BMD size (nodes)	BMD CPU time (s)	TED size (nodes)	TED CPU time (s)
Dup-real	3x16	1x16	92	10	5	1
IIR	5x16	1x16	162	13	7	1
FIR16	16x16	1x16	450	25	18	1
FFT	10x16	8x16	995	31	29	1
Elliptic	8x16	8x16	922	19	47	1
LMS128	50x16	1x16	8194	128	52	1
DCT	32x16	16x16	2562	77	82	1
ProdMat	32x16	16x16	2786	51	89	1
Kalman	77x16	4x16	4866	109	98	1

Fig. 11 depicts a multiple-output TED for the elliptical wave filter (design *elliptic*), where each root node corresponds to an output of the design.

F. Algorithmic Verification

In this final set of experiments we demonstrate the natural capability of Taylor Expansion Diagrams to verify equivalence of designs described at the *algorithmic* level. Consider two dataflow designs computing convolution of two real vectors, $A(i), B(i), i = 0, \ldots N-1$, shown in Fig. 12. The design in Fig. 12(a) computes FFT of each vector, computes product of the FFT results, and performs the inverse FFT operation, producing output $IFFT$. The operation shown in Fig. 12(b) computes convolution directly from the two inputs, $C(i) = \sum_{k=0}^{N-1} A(k) \cdot B(i-k)$. TED was used to represent these two computations for $N = 4$ and to prove that they are indeed equivalent. Fig. 13 depicts the TED for vector C of the convolution operation, isomorphic with the vector $IFFT$. All graphs are automatically generated by our TED-based verification software.

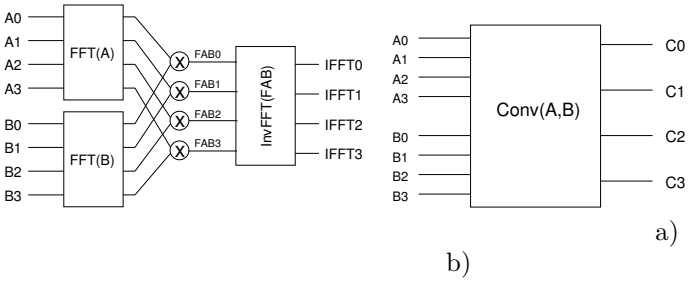

Fig. 12. Equivalent Computations: (a) FFT-Product-Inv(FFT); (b) Convolution.

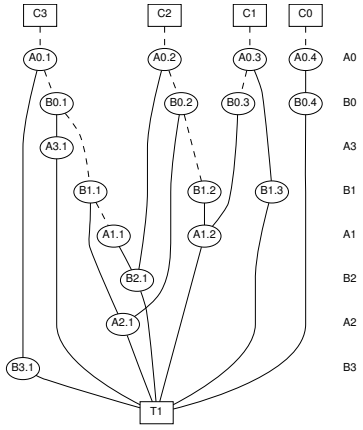

Fig. 13. TED for convolution vector C, isomorphic with $IFFT$

As illustrated by the above example, TEDs can be suitably augmented to represent computations in the complex domain. In fact, it can be shown that TEDs can represent polynomial functions over an *arbitrary field*. The only modification required is that the weights on the graph edges be elements of the field, and that the composition (MULT and ADD) be performed with the respective operators of the field. Subsequently, TEDs can also be used to represent computations in *Galois field* [54].

VII. Conclusions and Future Work

This paper has presented a compact, canonical, graph-based representation, called Taylor Expansion Diagram (TED). It has been shown that, for a fixed ordering of variables, TED is a canonical representation that can be used to verify equivalence of arithmetic computations in dataflow designs. The power of abstraction of TEDs makes them particularly applicable to dataflow designs specified at the behavioral and algorithmic level. The theory of TEDs has been presented, and the various operations on TEDs described that make the graph minimal and canonical. Size complexity of the representation and time complexity of its composition operations has also been analyzed and compared to other contemporary representations. It has been shown how TEDs can be constructed for behavioral and some RTL design descriptions.

An initial implementation of a TED package and experimental results have been described. Experiments were conducted over a number of designs to observe the power and limitations of the TED representation. The experiments demonstrate the applicability of TED representation to verification of behavioral and algorithmic designs. Of particular promise is the use of TEDs in equivalence verification of behavioral and algorithmic descriptions, where the use of symbolic, word-level operands, without the need to specify their bit-width, is justified. For large systems, involving complex bit-select operations, relational operators and memories, TEDs can be used to represent datapath portions of the design that can be modeled as polynomials. Equivalence checking of such complex designs typically involves finding structurally similar points of the designs under verification. TED data structure can be used here to raise the level of abstraction of large portions of designs, aiding in the identification of such similar points and in the overall verification process. In this sense TEDs complements existing representations, such as BDDs and *BMDs, in places where the level of abstraction can be raised.

A number of open problems remain to be researched to make TEDs a reliable data structure for high-level design representation and verification. While a simple VHDL and C interface has been already provided based on the GAUT high-level synthesis system, a front-end interface to the TED data structure should be developed for designs described in Verilog and System C. The recently developed dynamic variable ordering needs to be tested and integrated with the system. Also, a robust static variable ordering needs to be investigated. Finally, we have recently demonstrated the potential of TEDs in symbolic factorization and architectural synthesis, especially for DSP designs. TEDs can be used to perform top-level transformations of dataflow graphs and for architectural space exploration [60]. It can also be used for DSP transform optimization by means of common subexpression elimination and factorization [61]. A prototype software for TED-based verification and behavioral transformations, *TEDify*, is available on the web [62].

In summary, TEDs can play a fundamental role in providing an efficient data structure for those applications that can be modeled in

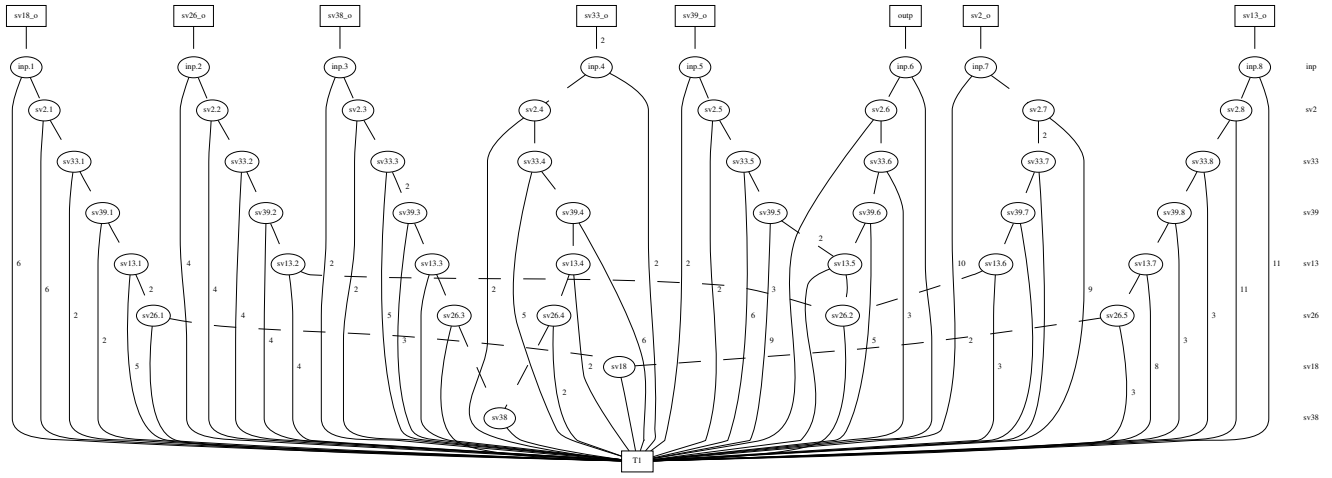

Fig. 11. Elliptic Wave Filter: TED structure obtained automatically from VHDL description.

terms of polynomials, and in particular in high level design representations and verification. We also believe that TEDs can enhance the effectiveness of symbolic methods offered by commercial tools, such as Mathematica and Matlab, for the purpose of formal verification and synthesis of digital systems design.

ACKNOWLEDGMENTS

The authors are indebted to Emmanuel Boutillon of LESTER, Université de Bretagne Sud, Lorient, France, for his invaluable input regarding the application of TEDs to algorithmic verification. The authors also would like to thank Pierre Bomel of LESTER, for help with the GAUT system, and Namrata Shekhar of University of Utah, for performing BMD experiments.

References

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

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

[3] Y. A. Chen and R. E. Bryant, "*PHDD: An Efficient Graph Representation for Floating Point Verification," in *Proc. ICCAD*, 1997.

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

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

[6] 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.

[7] B. Taylor, *Methodus Incrementorum Directa et Inversa*, 1715.

[8] E. Kryrszig, *Advanced Engineering Mathematics*, John Wiley and Sons, Inc., 1999.

[9] H. Enderton, *A mathematical Introduction to Logic*, Academic Press New York, 1972.

[10] T. Bultan and et. al, "Verifying Systems with Integer Constraints and Boolean Predicates: a Composite Approach," in *Proc. Int'l. Symp. on Software Testing and Analysis*, 1998.

[11] S. Devadas, K. Keutzer, and A. Krishnakumar, "Design Verification and Reachability Analysis using Algebraic Manipulation," in *Proc. Intl. Conference on Computer Design*, 1991.

[12] G. Ritter, "Formal Verification of Designs with Complex Control by Symbolic Simulation," in *Advanced Research Working Conf. on Correct Hardware Design and Verification Methods (CHARME)*, Springer Verlag LCNS, Ed., 1999.

[13] R. E. Shostak, "Deciding Combinations of Theories," *Journal of ACM*, vol. 31, no. 1, pp. 1–12, 1984.

[14] Aaron Stump, Clark W. Barrett, and David L. Dill, "CVC: A Cooperating Validity Checker," in *14th International Conference on Computer Aided Verification (CAV)*, Ed Brinksma and Kim Guldstrand Larsen, Eds. 2002, vol. 2404 of *Lecture Notes in Computer Science*, pp. 500–504, Springer-Verlag, Copenhagen, Denmark.

[15] M. Chandrashekhar, J. P. Privitera, and J. W. Condradt, "Application of term rewriting techniques to hardware design verification," in *Proc. Design Automation Conf.*, 1987, pp. 277–282.

[16] Z. Zhou and W. Burleson, "Equivalence Checking of Datapaths Based on Canonical Arithmetic Expressions," in *Proc. Design Automation Conference*, 1995.

[17] S. Vasudevan, "Automatic Verification of Arithmetic Circuits in RTL using Term Rewriting Systems," M.S. thesis, University of Texas, Austin, 2003.

[18] J. Burch and D. Dill, *Automatic Verification of Pipelined Microprocessor Control*, Springer-Verlag, 1994.

[19] R. Bryant, S. German, and M. Velev, "Processor Verification using Efficient Reductions of the Logic of Uninterpreted Functions to Propositional Logic," *ACM Trans. Computational Logic*, vol. 2, no. 1, pp. 1–41, 2001.

[20] M. Velev and R. Bryant, "Effective use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors," *Journal of Symbolic Computation*, vol. 35, no. 2, pp. 73–106, 2003.

[21] R. Bryant, S. Lahiri, and S. Seshia, "Modeling and Verifying Systems using a Logic of Counter Arithmetic with Lambda Expressions and Uninterpreted Functions," *CAV*, 2002.

[22] M. Moskewicz, C. Madigan, L. Zhang Y. Zhao, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. of 38th Design Automation Conf.*, June 2001, pp. 530–535.

[23] E. Goldberg and Y. Novikov, "BerkMin: A Fast and Robust Sat-Solver," in *Proc. Design Automation and Test in Europe, DATE-02*, 2002, pp. 142–149.

[24] C.-Y. Huang and K.-T. Cheng, "Using Word-Level ATPG and Modular Arithmetic Constraint Solving Techniques for Assertion Property Checking," *IEEE Trans. CAD*, vol. 20, pp. 381–391, 2001.

[25] M. Iyer, "RACE: A word-level ATPG-based Constraints Solver System for Smart Random Simulation," in *Internationall Test Conf., ITC-03*, 2003, pp. 299–308.

[26] R. Brinkmann and R. Drechsler, "RTL-Datapath Verification using Integer Linear Programming," in *Proc. ASP-DAC*, 2002.

[27] Z. Zeng, P. Kalla, and M. Ciesielski, "LPSAT: A unified approach to RTL satisfiability," in *Proc. DATE*, March 2001, pp. 398–402.

[28] F. Fallah, S. Devadas, and K. Keutzer, "Functional Vector Generation for HDL Models using Linear Programming and 3-Satisfiability," in *Proc. Design Automation Conference*, 1998, pp. 528–533.

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

[30] O. Coudert and J.C. Madre, "A Unified Framework for the Formal Verification of Sequential Circuits," in *Proc. ICCAD*, 1990, pp. 126–129.

[31] H.J. Touati, H. Savoj, B. Lin, R.K. Brayton, and A. Sangiovanni-

Vincentelli, "Implicit State Enumeration of Finite State Machines using BDDs," in *Proc. ICCAD*, 1990, pp. 130–133.

[32] E. A. Emerson, "Temporal and Modal Logic," in *Formal Models and Semantics*, J. van Leeuwen, Ed., vol. B of *Handbook of Theoretical Computer Science*, pp. 996–1072. Elsevier Science, 1990.

[33] K. L. McMillan, *Symbolic Model Checking*, Kluwer Academic Publishers, 1993.

[34] 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.

[35] A. Narayan and *et al.*, "Partitioned ROBDDs: A Compact Canonical and Efficient Representation for Boolean Functions," in *Proc. ICCAD*, '96.

[36] Y-T. Lai, M. Pedram, and S. B. Vrudhula, "FGILP: An ILP Solver based on Function Graphs," in *ICCAD*, 93, pp. 685–689.

[37] U. Kebschull, E. Schubert, and W. Rosentiel, "Multilevel Logic Synthesis based on Functional Decision Diagrams," in *EDAC*, 1992, pp. 43–47.

[38] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski, "Efficient Representation and Manipulation of Switching Functions based on Order Kronecker Function Decision Diagrams," in *DAC*, 1994, pp. 415–419.

[39] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral Transforms for Large Boolean Functions with Applications to Technology Mapping," in *DAC*, 93, pp. 54–60.

[40] I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic Decision Diagrams and their Applications," in *ICCAD*, Nov. 93, pp. 188–191.

[41] S. Horeth and Drechsler, "Formal Verification of Word-Level Specifications," in *DATE*, 1999, pp. 52–58.

[42] Y-A. Chen and R. Bryant, "PHDD: An Efficient Graph Representation for Floating Point Circuit Verification," in *IEEE Int. Conference on Computer-Aided Design*, 1997, pp. 2–7.

[43] G. Bioul and M. Davio, "Taylor Expansion of Boolean Functions and of their Derivatives," *Philips Research Reports*, vol. 27, no. 1, pp. 1–6, 1972.

[44] A. Thayse and M. Davio, "Boolean Differential Calculus and its Application to Switching Theory," *IEEE Trans. on Comp.*, vol. C-22, no. 4, pp. 409–420, 1973.

[45] J. Smith and G. DeMicheli, "Polynomial Methods for Compontent Matching and Verification," in *International Conference on Computer-Aided Design, ICCAD'98*, 1998.

[46] J. Smith and G. DeMicheli, "Polynomial Methods for Allocating Complex Compontents," in *Design Automation and Test In Europe Conference, DATE'99*, 1999.

[47] A. Peymandoust and G. DeMicheli, "Application of Symbolic Computer Algebra in High-Level Data-Flow Synthesis," *IEEE Trans. on Computer-Aided Design*, vol. 22, no. 9, pp. 1154–1165, Sept. 2003.

[48] Maple, ," http://www.maplesoft.com.

[49] Mathematica, ," http://www.wri.com.

[50] The MathWorks, "Matlab," http://www.mathworks.com.

[51] F. Winkler, *Polynomial Algorithms in Computer Algebra*, Springer, 1996.

[52] G. Fey, R. Drechsler, and M. Ciesielski, "Algorithms for Taylor Expansion Diagrams," in *IEEE Intl. Symposium on Multi-Valued Logic, ISMVL'04*, 2004.

[53] F. Brown, *Boolean Reasoning*, Kluwer Academic Publishers, 1990.

[54] D. Pradhan, S. Askar, and M. Ciesielski, "Mathematical Framework for Representing Discrete Functions as Word-level Polynomials," in *IEEE Intl. High Level Design Validation and Test Workshop, HLDVT-03*, 2003, pp. 135–139.

[55] LESTER, Université de Bretagne Sud, "GAUT, Architectural Synthesis Tool," http://lester.univ-ubs.fr:8080, 2004.

[56] F. Somenzi, "Colorado Decision Diagram Package," Computer Programme, 1997.

[57] R. Rudell, "Dynamic Variable Ordering for Binary Decision Diagrams," in *Proc. Intl. Conf. on Computer-Aided Design*, Nov. 1993, pp. 42–47.

[58] D. Gomez-Prado, Q. Ren, S. Askar, M. Ciesielski, and E. Boutillon, "Variable Ordering for Taylor Expansion Diagrams," in *IEEE Intl. High Level Design Validation and Test Workshop, HLDVT-04*, 2004, pp. 55–59.

[59] P. Jain, "Parameterized Motion Estimation Architecture For Dynamically Varying Power and Compression Requirements," M.S. thesis, Dept. of Electrical and Computer Engineering, University of Massachusetts, 2002.

[60] M. Ciesielski, S. Askar, E. Boutillon, and J. Guillot, "Behavioral Transformations for Hardware Synthesis and Code Optimization based on Taylor Expansion Diagrams," Dec. 2005, Patents pending, USSN 11/292,493 and PCT/US05/43860.

[61] J. Guillot, E. Boutillon, D. Gomez-Prado, S. Askar, Q. Ren, and M. Ciesielski, "Efficient Factorization of DSP Transforms using Taylor Expansion Diagrams," in *Design Automation and Test in Europe, DATE-06*, 2006.

[62] M. Ciesielski, S. Askar, D. Gomez-Prado, and Q. Ren, TEDify - Software for construction and optimization of TEDs, with application to verification and synthesis of behavioral designs, http://tango.ecs.umass.edu/TED/Doc/html.