TED+: A Data Structure for Microprocessor Verification

Pejman Lotfi-Kamran, Mohammad Hosseinabady, Hamid Shojaei,
Mehran Massoumi*, and Zainalabedin Navabi
Electrical and Computer Engineering Department, Faculty of Engineering,
University of Tehran, Tehran, Iran
*California State University, CA, USA
{plotfi, mohammad, shojai}@cad.ece.ut.ac.ir, massoumi@sbcglobal.net, and navabi@ece.neu.edu

Abstract—Formal verification of microprocessors requires a mechanism for efficient representation and manipulation of both arithmetic and random Boolean functions. Recently, a new canonical and graph-based representation called TED has been introduced for verification of digital systems. Although TED can be used effectively to represent arithmetic expressions at the word-level, it is not memory efficient in representing bit-level logic expressions. In this paper, we present modifications to TED to improve its ability for bit-level logic representation while maintaining its robustness in arithmetic word-level representation. It will be shown that for random Boolean expressions, the modified TED performs the same as BDD representation.

I. INTRODUCTION

Boolean functions are often represented and manipulated by Decision Diagrams (*DD*s). Ordered Binary Decision Diagrams (*OBDD*s) [1] are the most commonly used form of decision diagrams in EDA applications [2]. Despite its widespread use, some classes of Boolean functions cannot be represented efficiently by *OBDD*s [3] [4]. For representing these classes of Boolean functions other decision diagrams are proposed and used. As an example, Ordered Functional Decision Diagrams (*OFDD*s) [5] [6] are proposed for better representing XOR based logic [7]. *OBDD*s and their variations have been successfully used in manipulating gate-level designs, but have limitations in representing arithmetic circuits.

For representing arithmetic circuits, Word Level Decision Diagrams (*WLDD*s) are proposed. They use decomposition methods similar to the decomposition of Boolean functions, but at the arithmetic-level. *MTBDD*s [8] [9], *EVBDD*s [10], *BMD*s *[11], HDD*s [12], **BMD*s [11], *K*BMD*s [13] are examples of *WLDD*s. They are graph-based representations of functions with Boolean domain and integer range; therefore an arithmetic function should be broken down into bit-level in order for it to be represented by *WLDD*s.

With increasing complexity of digital systems, the need for higher level abstractions becomes more evident. *TED* [14] [15] [16] is proposed to satisfy this need. In *TED*, the decomposition of a function (Boolean or arithmetic) is performed along each word-level variable of the function using Taylor series expansions. It has been proven that with a special restriction on the ordering of function variables, *TED* becomes a canonical representation. *TED* can also be used for representing functions with Boolean domain and Boolean/integer range.

Although *TED* has many advantages over *WLDD*s, its weak Boolean function manipulation is its main problem. When a design consists of word-level and bit-level parts (including Boolean parts), its *TED* occupies a large amount of memory. Today, many microprocessors have a large data-path and one or several controllers. Usually the data-path contains arithmetic circuits with vector size of 32-bit or longer and the controller has many bit-level signals for controlling the data-path operations. In such cases, *WLDD*s are not the best solution because they break arithmetic operations into their bit-level parts. This detailed representation is an overkill for RT-level verification of microprocessors, and significantly degrades the verification performance. On the other hand, *TED*, that has a good arithmetic representation for data path parts of a design, does not offer a good solution for Boolean manipulation for the controller parts.

A solution is to use different Decision Diagrams for representing different part of a design. This solution leads to more difficulties in the verification process. Also using two or more different decision diagrams makes it hard and almost impossible to check equivalency of two designs, because equivalency of two designs does not mean that each of their parts are necessarily equivalent.

The aim of this paper is to provide a unique representation for handling high-level verification of data-path and controllers. The key idea is that arithmetic operations encountered in the RTL specification are simple, e.g., *x+y*, *x*y*, etc. Therefore, it is proper to degrade the performance of algebraic representation of expressions that are not common in practical designs, and instead improve the performance of Boolean function manipulations. This paper provides a unique representation for efficiently representing typical algebraic equations encountered in data-path of today's high performance processors. At the same time, our representation has a good Boolean function manipulation.

This paper is organized as follows: Section 2 presents a brief overview of *TED*. In Section 3, our *TED+* is introduced. In Section 4, Additive weights (as in *k*BMD*) will be added to the *TED+* for better representation of arithmetic and Boolean functions. Experimental results are discussed in Section 5 and conclusions are presented in the last section.

II. TAYLOR EXPANSION DIAGRAM

TED is a graph-based representation that uses the Taylor series as its decomposition method [14] [15] [16]. The Taylor series expansion of a real differentiable function *f(x)* around *x=0* is:

$$f(x) = f(0) + xf'(0) + \frac{1}{2!}x^2 f''(0) + \frac{1}{3!}x^3 f^{(3)}(0) + \cdots \quad (1)$$

where $f'(0)$, $f''(0)$ and $f^{(3)}(0)$ are first, second and third derivations of function f for $x=0$, respectively. The decomposition will be performed recursively using Equation (1).

Every node of a *TED* representation has a label that indicates its associated variable. As in most canonical decision diagrams, e.g., *OBDD*, the variables of *TED* are ordered. The function of a node is determined by the Taylor series expansions, according to Equation (1). The out-degree of a node depends on the order of the associated variable of that node. The out-degree of a terminal node is 0.

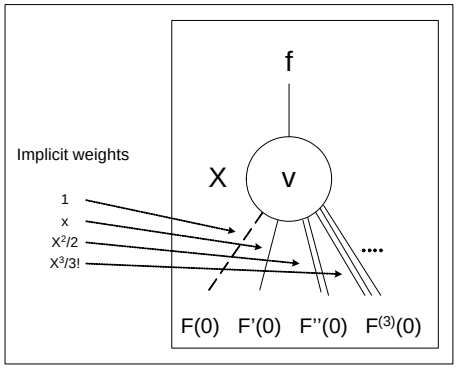

Fig. 1. Decomposition in *TED*

Figure 1 shows *TED* decomposition of function f for variable x. In this paper, we refer to the *k-th* derivative of a function rooted at a node as k-child of that node: $f(x=0)$ is the 0-child, $f'(x=0)$ is the 1-child, $f''(x=0)$ is the 2-child, etc. We also refer to the corresponding edges as 0-edge (dotted), 1-edge (solid), 2-edge (double), etc. From the Taylor expansion, it is evident that each edge has an implicit multiplicative factor: x^0 for the 0-edge, x^1 for the 1-edge, $x^2/2$ for the 2-edge, etc. In addition, each edge in a *TED* has a multiplicative weight, which is computed from the Taylor expansions. Figure 2 shows *TED* representation of x^2+y.

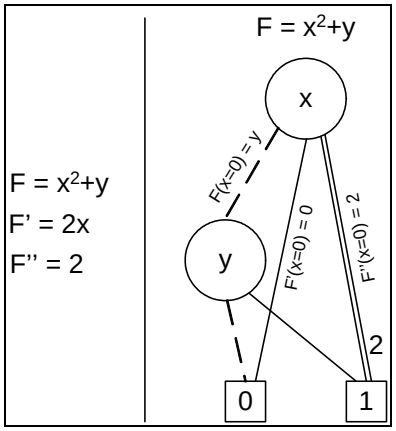

Fig. 2. *TED* representation of x^2+y

It has been proven that with a special restriction on the order of variables, *TED* becomes a canonical representation. For functions typically encountered in RTL specifications (e.g., $x - y$, $x + y$, $x * y$ and x^k for arbitrary k, etc.), *TED* is linear in the number of variables. *TED* can also represent functions containing both algebraic and Boolean expressions. To represent Boolean expressions, the following formulas should be used [14] [15] [16]:

$$Not(x) = x' = 1 - x \quad (2)$$
$$And(x, y) = x \wedge y = x * y \quad (3)$$
$$Or(x, y) = x \vee y = x + y - x * y \quad (4)$$

III. TED+

Representing Boolean functions is the main problem of *TED*. This means that the *TED* representation of a Boolean function is larger than the function's *BDD* representation.

TED+ capitalizes on the fact that representation of *TED*, when its decomposing variable has degree 1, can be simplified. This greatly simplifies the presentation of Boolean functions. As discussed in the previous section, in each level, *TED* decomposes a function into its Taylor series expansions, i.e., Equation (1). When the degree of the decomposing variable is 1, the Taylor series becomes:

$$f(x) = f(0) + xf'(0) \quad (5)$$

in which the following equation holds.

$$f'(0) = f(1) - f(0) \quad (6)$$

Using Equation (6) in (5) and rearranging the variables, Equation (7) results.

$$f(x) = (1 - x)f(0) + xf(1) \quad (7)$$

As shown, Equation (7) is similar to the Shannon decomposition, which is used as the decomposing method in Binary Decision Diagrams (*BDD*s). To improve logic representation of *TED*, in nodes with two outgoing edges (i.e., nodes with associated variable of degree 1), we use Equation (7) instead of Equation (5) for our decomposing method. By doing this and using Equations (2), (3) and (4) as *Not*, *And* and *Or* respectively, *TED+* will behave like *BDDs* for logic representation. Also because Equation (7) and Equation (5) are the same (i.e., Equation (7) is computed from Equation (5) and vice versa), the *TED+* still remains canonical.

A. TED+ for Boolean Representation

In Boolean algebra "*a * a = a*", so the degree of all variables in a Boolean expression is 1. Therefore in representing Boolean expressions with *TED+*, Shannon decomposition will be used as the decomposing method in all nodes. So *TED+* representation of Boolean functions is the same as *BDD* with the overhead of multiplicative weights for

the edges. Note that *BDD* edges do not have weights. Figure 3 shows *TED* and *TED+* representations of Boolean *Or* function.

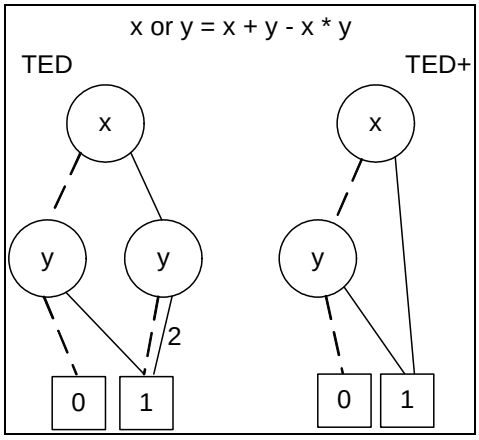

Fig. 3. *TED* and *TED+* representation of *x or y*

B. TED+ for Arithmetic Representation

Now that we have shown advantages of *TED+* over *TED* in representing Boolean expressions, let us turn our attention to arithmetic functions. It is easy to show that *TED+* is the same as *TED* for representing a collection of multiply operations. This is unlike *TED+* representation for the addition operation. The *TED* and *TED+* representations of $x + y$, illustrating the disadvantage of *TED+*, are shown in Fig. 4.

It is evident that *TED+* is not as efficient as *TED* in representing addition. In the next section, we improve *TED+* for efficient representation of algebraic (including addition) functions. We will show that our improvements also help representation of Boolean function.

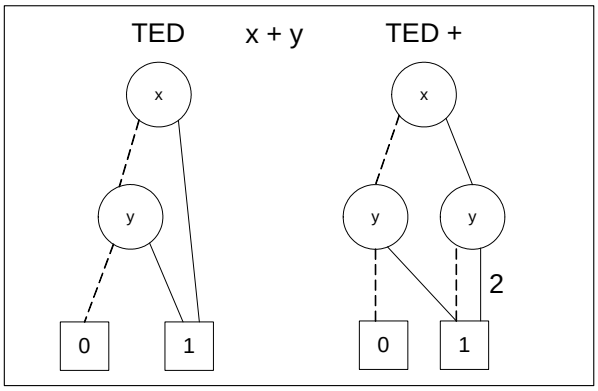

Fig. 4. *TED* and initial *TED+* representation of $x + y$

IV. Additive Weights

In the previous section, it was shown that *TED+* is not as efficient as *TED* in representing algebraic equations. On the other hand, *TED+* is as good as *BDD* in representing Boolean expressions. As discussed in Reference [17] and [18], a variation of *BDD* that adds attributes to its edges is more memory efficient than the standard *BDD*. Similar memory efficiency can be obtained for *TED+* by adding additive weights. In this section, with the addition of additive weights, we will improve the ability of *TED+* for both algebraic and Boolean representations.

It is clear that 0-childs of both Shannon and Taylor decomposition are computed the same way (i.e., 0-child of a node is computed by putting a 0 in place of the associated variable of that node). On the other hand, in nodes with more than two outgoing edges, *TED+* can only use Taylor expansion, which makes it the same as *TED*. We conclude that the weakness, mentioned in the previous section, of *TED+* for representing algebraic equations originates from the 1-child of nodes with two outgoing edges.

A special situation of this case can be resolved by additive weights. To show this special case, consider node x of function $f=g+cx$ where g is a function and c is a constant. Using the Shannon expansion in this node with two outgoing edges, yields a 1-child of the form $f(x=1)=c+g$. On the other hand, if the Taylor expansion is used for decomposition of this node, its 1-child function becomes $f'(x=0)=c$. By comparing these two, it is evident that the 1-child of *TED* (i.e., c) is simpler than that of *TED+* (i.e., $c+g$). Furthermore, the 1-child of *TED* can be represented by an edge to terminal **1** (with multiplicative weight c). To remedy this deficiency of *TED+*, additive weights have been added to its edges, in addition to multiplicative weights. The key idea is that function g is already constructed for representing the 0-child in *TED+* (i.e., $f(x=0)=g$). Therefore by adding an additive weight to each edge; it is possible to represent the 1-child of *TED+* (i.e., $f(x=1)=c+g$) by an edge with additive weight c to the 0-child (i.e., $f(x=0)=g$). This is shown in Fig 5.

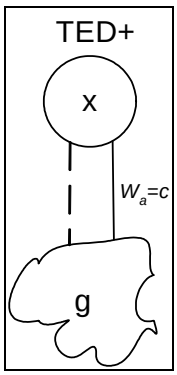

Fig. 5. *TED+* representation of $g+cx$

With the addition of additive weights, the function of the edge is determined by $w_a + w_m * g(x)$, where w_a is the additive weight and w_m is the multiplicative weight of that edge and $g(x)$ denotes the function represented by the node which this edge is pointing to.

In the above discussion, coefficient of x was a constant. It is evident that if the coefficient of an associated variable is not a constant, it is possible that the *TED* representation of that function is more compact than its *TED+*.

Table 1. *BDD*, *TED* and *TED+* construction results for various circuits

Circuits	BDD		TED		TED+	
	Nodes	Time (Sec.)	Nodes	Time (Sec.)	Nodes	Time (Sec.)
Hamming 8bit	1445	0	1970	0.3	1445	0
Address Decoder 20bit	108	0	165	0	108	0
Parity Generator 9bit	329	0	384	0.1	329	0
Comparator 4bit	431	0	587	0.1	431	0
Comparator 7bit	940	0	963	0	940	0
Comparator 16bit	198484	0.3	198531	8.3	198484	4.1
Full Adder	30	0	37	0	30	0
Adding 9 CS 2bit each	4388	0	9663	84.3	4388	0.1
Adding 9 CS 4bit each	15194	0	Unfeasible	Unfeasible	15194	0.3
Adding 9 CS 8bit each	36750	0	Unfeasible	Unfeasible	36750	0.7
CLA Adder 8bit	9224	0	12398	13.8	9224	0.2
CLA Adder 16bit	2373630	2.7	Unfeasible	Unfeasible	2373630	52.3
Array Multiplier 8bit	184685	0.1	Unfeasible	Unfeasible	184685	2.5
Multiplier 4bit	1757	0	2008	0.3	1757	0
Multiplier 8bit	173091	0.1	Unfeasible	Unfeasible	173091	2.8
c17	34	0	38	0	34	0
c432	27005	0	Unfeasible	Unfeasible	27005	0.3
c499	510732	0.1	Unfeasible	Unfeasible	510732	2
c880	2501340	1.9	Unfeasible	Unfeasible	2501340	35.9
c1355	1733831	0.2	Unfeasible	Unfeasible	1733831	5
c1908	512736	0.3	Unfeasible	Unfeasible	512736	6.9

A. Canonical Representation

To make *TED+* a canonical representation, we need some further restrictions on the graph with respect to the edge weights. These restrictions as mentioned in [13] are as follows:

1. There exists only one terminal and this terminal is labeled 0.
2. The 0-edge of a node has additive weight *0*.
3. The greatest common divisor of other weights is *1*.

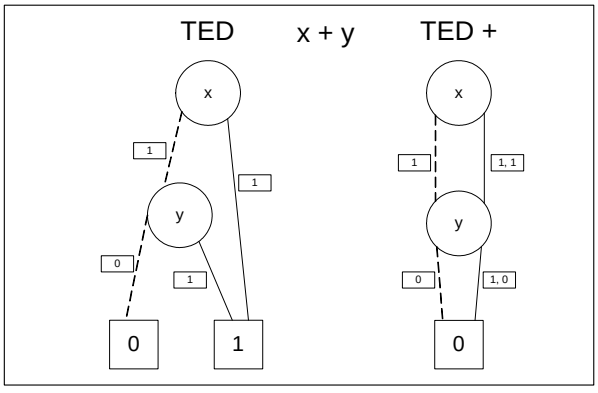

Fig. 6. *TED* and *TED+* representation of *x + y*

B. Modified Arithmetic Representation

By adding the additive weights and provisions for being canonical, a new representation for *TED+* emerges. As discussed above, by using additive weights, the *TED+* representation of addition becomes as good as *TED*. *TED+* representation of *x + y* is shown in Fig. 6. Both Additive and Multiplicative weights are shown in rectangular boxes on the graph edges. *TED* only has multiplicative weights while *TED+* has both additive and multiplicative weights. As discussed, in *TED+* 0-edges only have multiplicative weights and other edges have both additive and multiplicative weights. The first number is additive and the second one is its multiplicative weight.

0-child of both *TED* and *TED+* are the same. The 1-child of *TED* is *1* while the 1-child of *TED+* is *1+y*. By using additive weights, both representations will have the same number of nodes.

C. Modified Boolean Representation

As mentioned, our goal is to improve Boolean representation of *TED+* and make it the same as *BDD* with attributed edges.

In *BDD* with attributed edges, only non-complemented functions and sub-functions are constructed directly and other functions are represented through non-complemented functions. For this, Reference [7] adds an attribute to each edge. In this representation, complement of a function is constructed by the function pointed to by an attributed edge.

Our representation of *TED+* includes a similar feature. For complementing a function *f(x), 1-f(x)* should be constructed. Since we have additive weights in our diagram, we can extract

the additive *1*, and *–f(x)* remains as a result. Furthermore, since we have multiplicative weights, after extracting *-1* as the multiplicative weight, the only remaining part becomes *f(x)*.

Table 2. *TED* and *TED+* construction results for various algebraic equations

Equations	TED		TED+	
	Nodes	Time	Nodes	Time
$\sum_{i=1}^{100} x_i$	102	0.1	101	2.6
$\prod_{i=1}^{100} x_i$	102	0.1	101	0.1
$\prod_{i=1}^{10}\sum_{j=1}^{10} x_{i,j}^{\ 2}$	102	0	101	0
$\prod_{i=1}^{20}\sum_{j=1}^{20} x_{i,j}^{\ 2}$	402	0.2	401	0.2
$\prod_{i=1}^{10}\sum_{j=1}^{10} x_{i,j}^{\ 3}$	102	0	101	0
$\prod_{i=1}^{20}\sum_{j=1}^{20} x_{i,j}^{\ 3}$	402	0.3	401	0.4
$\prod_{i=1}^{10}\sum_{j=1}^{10} x_{i,j}(i+j)$	102	0.1	101	0.1
$\prod_{i=1}^{20}\sum_{j=1}^{20} x_{i,j}(i+j)$	402	1.3	401	2.0
$\prod_{i=1}^{6}(\sum_{j=1}^{6} x_{i,j}^{\ j})^i$	38	0.6	37	0.9
$\prod_{i=1}^{8}(\sum_{j=1}^{8} x_{i,j}^{\ j})^i$	2382	19.5	2381	27.3
$(\sum_{i=1}^{10} x_i)x + \sum_{j=1}^{10} x_j$	23	0	76	0
$(\sum_{i=1}^{20} x_i)x + \sum_{j=1}^{20} x_j$	43	0	251	0.1

V. EXPERIMENTAL RESULTS

In this section, we describe experimental results that have been carried out on a PC Pentium 4 with 1 GBytes memory. All runtimes are given in CPU seconds. *TED* and *TED+* packages are implemented by the authors with Visual C++ v6, and for *BDD*, we have used the *Cudd BDD* [19] package which is known to be very efficient.

In the first series of experiments, we compare sum of the number of nodes needed for *BDD*, *TED* and *TED+* representation of all gate outputs of several gate-level benchmarks. Table 1 provides a summary of the results obtained for these benchmark circuits. These circuits have *BDD*s with at least 30 nodes. The column labeled *BDD* shows the result of converting these circuits to *BDD*, while sub-columns show the number of *BDD* nodes and time it was spent for the conversion. *TED* and *TED+* columns show the same parameters for *TED* and *TED+* diagrams. Diagrams are built based on the same variable orderings. Unfeasible means that representation of the benchmark could not be constructed within 120 seconds.

Number of nodes in *TED+* is always the same as *BDD*. *BDD* is better than *TED+* in terms of time of conversion. This is due to the processing needed for calculating additive and multiplicative weights. The conversion time of *TED+* is less than that of *TED* because in representing Boolean expressions, the size of *TED* is larger. Moreover, the complexity of Boolean manipulation in *TED+* is less than that of *TED* (i.e., *TED+* is like *BDD* with additive and multiplicative weights in Boolean manipulation).

In all our test cases the arithmetic units are synthesized to the gate-level net-lists, before being converted to *BDD*, *TED* or *TED+*.

Circuits used for these experiments are the real world arithmetic units as well as some *ISCAS85* [20] benchmarks. All of these arithmetic units are basic components for other arithmetic units, so we can extend these results to all arithmetic units.

In the next series of experiments, we compare *TED+* with *TED* for representing algebraic equations. The equations, selected for these experiments, are those often used in RT-level design. The results of these experiments are shown in Table 2. In most cases, the number of *TED+* nodes is one less than that of *TED*. This is due to the different representation of terminal **1** in these two diagrams. Actually, the number of *TED* and *TED+* variable nodes are the same in these cases, but *TED* has two terminal nodes while *TED+* has one (i.e., terminal **1** is represented by terminal **0** with additive weight *1*).

Except a few cases, in which *TED+* uses more nodes, the number of nodes in *TED+* is the same as that of *TED*. Creation of *TED+* is more time consuming than *TED*, because the complexity of the algorithm used in *TED+* is more than *TED*.

Table 3. CPU Instruction Set

Opcode	Instruction
000	LDA
001	STA
010	ADD
011	SUB
100	JMP
101	JEZ
110	HLT

In the third series of experiments, a simple adding machine, representative of a processor, is converted to *TED* and *TED+*. This simple CPU is accumulator based with a 16-bit data-bus and a 16-bit address bus. It has 7 instructions that are shown in Table 3.

Table 4. ALU Operations

ALU_op	ALUout
00	B
01	A - B
10	A + B
11	B + 1

The ALU unit of this processor has A and B data inputs and a 2-bit select input that selects one of the four operations of the ALU. Table 4 shows the ALU operations.

The controller, shown in Fig. 7, is a state machine with four states. Most instructions take two clock periods to execute. The halt instruction halts the processor until it is reset.

The controller is transformed to transition relations and output functions and then these functions are converted to *TED* and *TED+*. Also, the data-path of the processor is converted to these diagrams simultaneously. The result of converting this processor is shown in Table 5. This table shows that compared with *TED*, *TED+* is about 36% better in terms of number of nodes and 2 times better in terms of time of conversion.

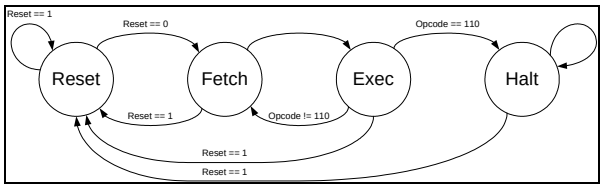

Fig. 7. CPU Controller

Although our experiment is done on a simple processor, this processor has a data path and a controller that is typical of many complex designs. Furthermore, this example has both arithmetic and Boolean parts. Therefore, it can show effectiveness of *TED+* in representing complex processors.

Table 5. *TED* and *TED+* construction results for a simple processor

	Nodes	**Time**
TED	175	0.031
TED+	112	0.015

VI. CONCLUSIONS

In this paper, *TED+* was proposed. In *TED+*, the decomposing method of nodes with two outgoing edges is changed (rearranged) to Shannon. However, this led to a worse performance of algebraic representations than *TED*. For reducing this, additive weights were added in addition to the multiplicative weights of the edges. In this way, a new data structure, for representing mixed arithmetic/Boolean designs (especially a microprocessor), was created.

TED+ is the same as *BDD* with attributed edges in representing Boolean expressions. On the other hand, *TED+* is worse than *TED* in representing some algebraic expressions. Because of rare occurrence of such cases in digital systems and also because Boolean parts are much larger than arithmetic parts in a real RT-level design, the overall performance of *TED+* is better than *TED*. Our experimental results show that many practical arithmetic expressions typically encountered in RTL specification are represented as good as *TED*. Therefore *TED+* is a good solution for representing designs containing both gate-level (Boolean expressions) and RT-level (arithmetic equations, bit-size 1 and longer) parts.

REFERENCES

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation", IEEE Trans. Comput., vol. C-35, pp. 677-691, Aug. 1986.

[2] R. E. Bryant, "Symbolic Boolean manipulation with ordered binary decision diagrams", ACM Comp. Surveys, vol. 24, pp. 293-318, Sep. 1992.

[3] B. Becker, R. Drechsler, and R. Werchner, "On the relation between BDD's and FDD's", Inform. Comput., vol. 123, no. 2, pp. 185-197, Dec. 1995.

[4] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams", Int. Conf. CAD, pp. 42-47, Nov. 1993.

[5] R. Drechsler, M. Theobald, and B. Becker, "Fast OFDD based minimization of fixed polarity Reed-Muller expressions", Eur. Design Automation Conf., pp. 2-7, Sep. 1994.

[6] U. Kebschull, E. Schubert, and W. Rosenstiel, "Multilevel logic synthesis based on functional decision diagrams", Eur. Conf. Design Automation, pp. 43-47, Mar. 1992.

[7] R. Drechsler and B. Becker, "Sympathy: Fast exact minimization of fixed polarity Reed-Muller expressions for symmetric functions", IEEE Trans. CAD, vol. 16, #1, pp. 1-5, Jan. 1997.

[8] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral Transforms for Large Boolean Functions with Applications to Technology Mapping", DAC, pp. 54-60, 1993.

[9] I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic Decision Diagrams and their Applications", ICCAD, pp. 188-191, Nov. 1993.

[10] Y-T. Lai, M. Pedram, and S. B. Vrudhula, "FGILP: An ILP Solver based on Function Graphs", ICCAD, pp. 685-689, 1993.

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

[12] E. M. Clarke, M. Fujita, and X. Zhao, "Hybrid Decision Diagrams - Overcoming the Limitation of MTBDDs and BMDs", ICCAD, 1995.

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

[14] P. Kalla, M. Ciesielski, E. Boutillon, E. Martin, "High-level design verification using Taylor Expansion Diagrams: first results", High-Level Design Validation and Test Workshop, pp. 13-17, Oct. 2002.

[15] M. Ciesielski, P. Kalla, Zhihong Zeng, B. Rouzeyre, "Taylor expansion diagrams: a new representation for RTL verification", High-Level Design Validation and Test Workshop, pp. 70-75, Nov. 2001.

[16] M. J. Ciesielski, P. Kalla, Zhihong Zheng, B. Rouzeyre, "Taylor expansion diagrams: a compact, canonical representation with applications to symbolic verification", Design, Automation and Test in Europe, pp. 285-289, Mar. 2002.

[17] S. Minato, N. Ishiura, S. Yajima, "Shared binary decision diagram with attributed edges for efficient Boolean function manipulation", DAC, pp. 52-57, Jan. 1991.

[18] K. S. Brace, R. L. Rudell, R. E. Bryant, "Efficient implementation of a BDD package", DAC, pp. 40-45, Jun. 1990.

[19] F. Somenzi, *CUDD: CU Decision Diagram Package* Release 2.3.0. University of Colorado at Boulder, 1998.

[20] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational circuits and a target translator in fortran", Int'l Symp. Circ. and Systems, Special Sess. on ATPG and Fault Simulation, pp. 663-698, 1985.