Arithmetic Ternary Decision Diagrams Applications and Complexity

Tsutomu Sasao
Department of Computer Science and Electronics
Kyushu Institute of Technology
Iizuka 820-8502, Japan
sasao@cse.kyutech.ac.jp

July 19, 1999

Abstract

In a binary decision diagram (BDD), a non-terminal node representing a function $f = \bar{x}f_0 \vee xf_1$ has two edges for f_0 and f_1. In the arithmetic ternary decision diagram (Arith_TDD), each non-terminal node has three edges, where the third edge denotes $f_2 = f_0 + f_1$, and $+$ is an integer addition. The Arith_TDD represents the extended weight function, an integer function showing the numbers of true minterms in the cubes. The Arith_TDD is useful to detect functional decompositions, prime implicants and prime implicates. Experimental results compare the sizes of BDDs and various TDDs for benchmark functions.

I. Introduction

A binary decision diagram (BDD) represents a two-valued logic function f. Let $f = \bar{x}f_0 \vee xf_1$ be the Shannon expansion of f with respect to variable x. Then, the sub-graphs of the BDD represent f_0 and f_1, as shown in Fig. 1.1. Note that a path in the BDD from the root node to a terminal node represents an assignment of values to the variables. The value of the terminal node is the function value for that assignment.

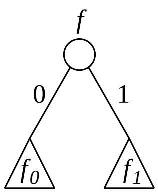


Fig. 1.1. BDD.

In this paper, we assume that the orderings of the input variables are the same for all paths from the root node to a leaf node, i.e., we consider only ordered decision diagrams (DDs).

In [20], we introduced various TDDs. Each TDD has interesting application in logic synthesis.

Let $B = \{0, 1\}$ and $T = \{0, 1, 2\}$. Let $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ be a ternary vector such that $\alpha_i \in T$. Then,

$$x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n} \quad (1)$$

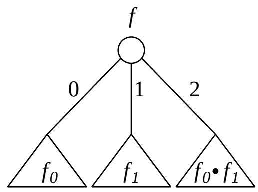


Fig. 1.2. AND_TDD.

represents a product of n variables, where

$$x^{\alpha} = \begin{cases} \bar{x} \text{ when } \alpha = 0, \\ x \text{ when } \alpha = 1, \\ 1 \text{ when } \alpha = 2. \end{cases}$$

In $\boldsymbol{\alpha}$, $\alpha_i = 2$ denotes that x_i can be either 0 or 1. Thus, $\boldsymbol{\alpha}$ represent a cube in an n-dimensional space.

An AND_TDD represents the set of all the implicants of a two-valued logic function. An AND_TDD represents a mapping F: $T^n \rightarrow B$, where $F(\boldsymbol{\alpha}) = 1$ iff the product $x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ is an implicant of f. An AND_TDD represents the set of all the implicants, and is constructed as shown in Fig. 1.2. Here the rightmost sub-graph represents $f_0 \cdot f_1$. In an AND_TDD, 0 and 1 denote the values of the corresponding variable, while 2 denotes that the value is don't care. And, each 1-path corresponds to an implicant of f. In general, reduced ordered AND_TDDs do not represent all the implicants, while the quasi-reduced AND_TDDs represent all the implicants.

An EXOR_TDD represents the extended truth vector of a two-valued logic function f. The extended truth vector $EXT(f : \boldsymbol{\alpha})$ for an n-variable function consists of 3^n elements, and is useful for optimization of AND-EXOR expressions [14, 17]. The EXOR_TDD represents the mapping $F : T^n \rightarrow B$, $F(\boldsymbol{\alpha}) = 1$ iff $EXT(f : \boldsymbol{\alpha}) = 1$. The EXOR_TDD is constructed as shown in Fig. 1.3, where the rightmost sub-graph represents the EXOR of f_0 and f_1.

In this paper, we introduce another type of TDDs, an arithmetic TDD (Arith_TDD), which is useful for functional decomposition, and generation of prime implicants and prime implicates. In the Arith_TDD, the rightmost

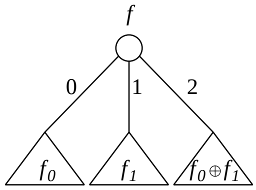


Fig. 1.3. EXOR_TDD.

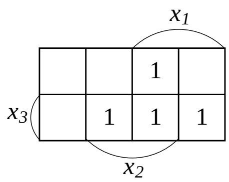


Fig. 2.1. Map for a switching function.

sub-graph represents the integer sum of f_0 and f_1.

This paper is organized as follows: Section II defines the extended weight functions and the Arith_TDD. Section III shows a method to find functional decompositions using Arith_TDDs. Section IV shows a method to generate prime implicants, and prime implicates by using Arith_TDDs.

II. Extended Weight Function and Arithmetic Ternary Decision Diagram

In this section, we will define extended weight functions, and arithmetic TDDs.

A. Extended Weight Function

Definition 2.1 *Let* $B = \{0,1\}$ *and* $T = \{0,1,2\}$. *For a switching function* $f : B^n \to B$, *define the* **extended weight function** $\mathcal{F} : T^n \to \{0,1,2,3,\ldots,2^n\}$. *If* $\boldsymbol{a} \in B^n$, *then* $\mathcal{F}(\boldsymbol{a}) = f(\boldsymbol{a})$. *Otherwise,* $\mathcal{F}$ *is computed as follows:*

$$\begin{aligned}\mathcal{F}(\alpha_1,\alpha_2,\ldots,\overset{i}{2},\ldots,\alpha_n) &= \\ \mathcal{F}(\alpha_1,\alpha_2,\ldots,0,\ldots,\alpha_n) &+ \\ \mathcal{F}(\alpha_1,\alpha_2,\ldots,1,\ldots,\alpha_n)&\end{aligned}$$

$\mathcal{F}(\boldsymbol{\alpha})$ *denotes numbers of true minterms in the cube* $\boldsymbol{\alpha} \in T^n$. *For short, a switching function is simply called a function.*

Example 2.1 *Consider the function in Fig. 2.1:* $f(x_1, x_2, x_3) = x_1x_2 \vee x_2x_3 \vee x_3x_1$. *It is the majority function of three variables. The extended weight function* $\mathcal{F}$ *is shown in Fig. 2.2. Note that the map consists of* $3^3 = 27$ *cells. Each cell corresponds to a cube* $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_3)$, *and the integer in the cell denotes the number of true minterms in the cube* $\boldsymbol{\alpha}$. *Especially, for* $\boldsymbol{\alpha} = (2,2,2)$, $\mathcal{F}(\boldsymbol{\alpha})$ *denotes the total number of true minterms in* f. ∎

Extended weight functions are useful for functional decompositions, and generation of prime implicants (prime implicates).

	$x_1 = 0$			$x_1 = 1$			$x_1 = 2$		
x_3 \ x_2	0	1	2	0	1	2	0	1	2
0	0	0	0	0	1	1	0	1	1
1	0	1	1	1	1	2	1	2	3
2	0	1	1	1	2	3	1	3	4

Fig. 2.2. Map for an extended weight function $\mathcal{F}$.

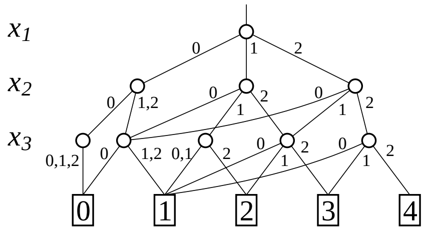


Fig. 2.3. Arith_TDD for $x_1x_2 \vee x_2x_3 \vee x_3x_1$.

B. Arithmetic Ternary Decision Diagram

An arithmetic TDD (Arith_TDD) is a graph-based representation of an extended weight function.

In the Arith_TDD,

1) Each path from the root node to a terminal node corresponds to a cube, and
2) The value of the terminal node represents the number of true minterms in the cube.

Example 2.2 *Fig. 2.3 shows the Arith_TDD for the three-variable majority function.* ∎

III. Functional Decomposition

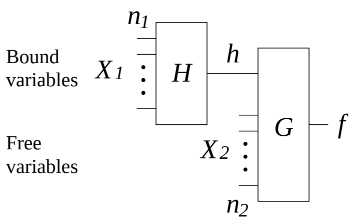
In this section, we will show the application of Arith_TDD to functional decomposition.

A. Definitions and Basic Properties

Functional decomposition is a basic technique to design multi-level logic networks. Although only a small fraction of the functions have decompositions, many useful functions do have decompositions [21]. Also, decomposed realizations are usually more economical than non-decomposed one.

Definition 3.1 *Let the set of the input variables be* $\{X\} = \{x_1, x_2, \ldots, x_n\}$. $(X_1, X_2, \ldots, X_r)$ *is a* **partition** *of* X *if* $\{X_i\} \cap \{X_j\} = \phi$ *and* $\{X_1\} \cup \{X_2\} \cup \cdots \cup \{X_r\} = \{X\}$. *Especially when* $r = 2$, *the partition is a* **bipartition**.

Definition 3.2 *A function* f *has a* **simple disjoint decomposition** *iff it is represented as* $f(X) = g(h(X_1), X_2)$, *where* (X_1, X_2) *is a bipartition of* X, *and* g *and* h *are switching functions. If* $|X_1| \geq 2$ *and* $|X_2| \geq 1$, *then the decomposition is* **non-trivial**, *and* f *is* **decomposable**.



$$f(X_1, X_2) = g(h(X_1), X_2)$$

Fig. 3.1. Functional decomposition.

$X_2 = (x_3, x_4)$ \ $X_1 = (x_1, x_2)$	00	01	10	11
00	1	1	1	1
01	1	0	0	1
10	0	1	1	0
11	0	0	0	0

Fig. 3.2. Decomposition table ($\mu = 2$).

We also assume that functions with up to two variables are decomposable. When a function f has a decomposition, f can be realized by the decomposed network shown in Fig. 3.1, and each block can be designed independently.

Definition 3.3 *Let $f(X)$ be a function, and (X_1, X_2) be a bipartition of X. Let $n_1 = |X_1|$ and $n_2 = |X_2|$. The* **decomposition table** *$T(f : X_1, X_2)$ of f has 2^{n_1} columns and 2^{n_2} rows, each column has distinct binary label of n_1 bits, each row has distinct binary label of n_2 bits, and the corresponding entry of the table shows the value of f.*

Example 3.1 *Let $f(X)$ the a four-variable function, (X_1, X_2) be a bipartition of X, where $X_1 = (x_1, x_2)$ and $X_2 = (x_3, x_4)$. A decomposition table $T(f : X_1, X_2)$ is shown in Fig. 3.2.* ∎

Definition 3.4 *The number of different column patterns in the decomposition table $T(f : X_1, X_2)$ is the* **column multiplicity** *and is denoted by $\mu(f : X_1, X_2)$. The number of different row patterns in the decomposition table is a* **row multiplicity** *and denoted by $\nu(f : X_1, X_2)$.*

Example 3.2 *In the decomposition table in Fig. 3.2, the column multiplicity is two: $\mu(f : X_1, X_2) = 2$. The row multiplicity is four: $\nu(f : X_1, X_2) = 4$.* ∎

Theorem 3.1 *A function $f(X)$ has a simple disjoint decomposition $f(X) = g(h(X_1), X_2)$ iff $\mu(f : X_1, X_2) \leq 2$.*

Theorem 3.2 *A function $f(X)$ has a simple disjoint decomposition $f(X) = g(h(X_1), X_2)$ iff $\nu(f : X_1, X_2) \leq 4$, and the functions represented by the distinct rows are either h, $\bar{h}$, 0 (constant zero function), or 1 (constant one function).*

Example 3.3 *In the decomposition table in Fig. 3.2, the column multiplicity is two, so the function is decomposable. Note that the first row denotes a constant 1 function, the second row denotes h, the third row denotes $\bar{h}$, and the last row denotes the constant 0. This table also satisfies the conditions of Theorem 3.2. On the other hand, in the decomposition table in Fig. 3.3, the column multiplicity is four, so the function is undecomposable for this bipartition.* ∎

$X_2 = (x_3, x_4)$ \ $X_1 = (x_1, x_2)$	00	01	10	11	Row weight
00	1	0	1	1	3
01	1	0	0	1	2
10	1	0	1	0	2
11	1	0	0	0	1
Column weight	4	0	2	2	

Fig. 3.3. Undecomposable function ($\mu = 4$).

$X_2 = (x_3, x_4)$ \ $X_1 = (x_1, x_2)$	00	01	10	11	Row weight	
00	0	0	1	1	2	w_1
01	1	1	0	1	3	w_2
10	1	1	1	0	3	
11	0	0	0	0	0	
Column weight	2	2	2	2		

Fig. 3.4. Undecomposable function.

Unfortunately, the direct application of Theorems 3.1 and 3.2 requires to check $O(2^n)$ different bipartitions, where n is the number of input variables. Especially when the function does not have any decomposition, we have to check $2^n - n - 2$ different bipartitions. Also, the size of the decomposition tables is 2^n, which is very large to build.

B. Functional Decomposition Using Extended Weight Functions

Definition 3.5 *In the decomposition table $T(f : X_1, X_2)$, let $CWM(f : X_1, X_2)$ be the number of different column weights, and $RWM(f : X_1, X_2)$ be the number of different row weights.*

Theorem 3.3 *In the decomposition table $T(f : X_1, X_2)$,*

1) *If $CWM(f : X_1, X_2) > 2$, then f is undecomposable for this bipartition.*
2) *If $RWM(f : X_1, X_2) > 4$, then f is undecomposable for this bipartition.*
3) *If there exist two row weights w_1 and w_2 such that $0 < w_1 < w_2 < 2^{n_1}$ and $w_1 + w_2 \neq 2^{n_1}$, then f is undecomposable for this bipartition.*

Example 3.4 *In Fig. 3.3, $CWM(f : X_1, X_2) = 3$, and $RWM(f : X_1, X_2) = 3$. So, by Theorem 3.3, f is undecomposable for this bipartition.* ∎

Example 3.5 *In Fig. 3.4, the column weights are all 2. Thus, $CWM(f : X_1, X_2) = 1$, and we cannot apply Condition 1) of Theorem 3.3. However, consider the row weights. Two row weights exist: $w_1 = 2$ and $w_2 = 3$,* *that are neither 0 nor $2^2 = 4$. Since $w_1 + w_2 = 2 + 3 \neq 4$, we can apply Condition 3) of Theorem 3.3, and show that f is undecomposable for this bipartition.* ∎

By Theorem 3.3, we have the following:

Algorithm 3.1 *(Functional Decompositions)*

1. *Obtain $\mathcal{F}$ from f.*
2. *For $n_1 = 2$ to $n-1$ do the followings:*
3. *Select a new partition $X = (X_1, X_2)$, where $|X_1| = n_1$. If all the partitions are selected, then stop.*
4. *If $CWM(f : X_1, X_2) > 2$, then f is undecomposable with (X_1, X_2). Go to 3.*
5. *If $RWM(f : X_1, X_2) > 4$, then f is undecomposable with (X_1, X_2). Go to 3.*
6. *Let w_1 and w_2 be weights, where $0 < w_1 < w_2 < 2^{n_1}$. If $(w_1 + w_2 \neq 2^{n_1})$, then f is undecomposable with (X_1, X_2). Go to 3.*
7. *Check if f is decomposable with (X_1, X_2) by using the BDD [15].*

C. Complexity Analysis

Here, we will consider the complexity of Algorithm 3.1. Assume that we have the Arith_TDD for f. Also assume that the value of $\mathcal{F}$ can be evaluated in n steps, where n represents the number of input variables. Step 3 selects $C(n, n_1)$ different combinations. For steps 4 $\sim$ 6, we assume that the number of look-up is bounded by the constant c_1. Also, assume that function is undecomposable, and one of the tests for steps 4 $\sim$ 6 is always true. Then, the computational complexity is bounded by

$$c_1 \sum_{n_1=2}^{n-1} C(n, n_1) = c_1[2^n - C(n,0) - C(n,1) - C(n-n)]$$
$$= c_1[2^n - 2 - n].$$

This means that the most computation time is spent for the construction of Arith_TDD, which is $O(3^n)$.

IV. Generation of Prime Implicants and Prime Implicates

Prime implicants and prime implicates are important in the design of AND-OR and OR-AND two-level logic networks, respectively [12]. In this section, we will show a method to generate prime implicants and prime implicates by using extended weight functions.

A. Definitions

Definition 4.1 *In two logic functions f and g, if $g(\boldsymbol{x}) = 1$ for all $\boldsymbol{x}$ such that $f(\boldsymbol{x}) = 1$, then g* **contains** *f, denoted by $f \leq g$. If a logic function f contains a product c, then c is an* **implicant** *of f. Furthermore, if c is a minterm, then c is a* **true minterm** *of f. A product P is a* **sub-product** *of Q, if all the literals in P also appear in Q. Let P be an implicant of a logic function f. If no other implicant Q of f is a sub-product of P, then P is a* **prime implicant** *(PI) of f.*

$X_2 = (x_3, x_4)$ \ $X_1 = (x_1, x_2)$	00	01	10	11
00	1	0	1	1
01	1	0	0	1
10	1	0	1	0
11	1	0	0	0
Column weight	4	0	2	2

Fig. 4.1. Detection of implicants and implicate.

B. Detection of Implicants and Implicates Using Extended Weight Function

We will show the idea by using a simple example.

Example 4.1 *Consider the function f in Fig. 4.1.*

1) *The weight of the first column is four. When the weight is $2^{n_2} = 2^2 = 4$, all the vertices in the cubes correspond to true minterms. This means that $\bar{x}_1\bar{x}_2$ is an implicant of f.*
2) *The weight of the second column is zero. This shows that all the vertices in the cubes correspond to false minterms. This means that $\bar{x}_1 x_2$ is an implicant of $\bar{f}$, i.e., $(x_1 \vee \bar{x}_2)$ is an implicate of f [12].*

In general, we have the following:

Theorem 4.1 *Let (X_1, X_2) be a partition of X, $n_1 = |X_1|$, $n_2 = |X_2|$, and $\boldsymbol{a} = (a_1, a_2, \ldots, a_{n_1}) \in B^{n_1}$ be an assignment of X_1. If $\mathcal{F}$ $(\boldsymbol{a}_1, X_2) = 2^{n_2}$, then $x_1^{a_1} x_2^{a_2} \cdots x_{n_1}^{a_{n_1}}$ is an implicant of f. If $\mathcal{F}(\boldsymbol{a}_1, X_2) = 0$, then $x_1^{\bar{a}_1} \vee x_2^{\bar{a}_2} \vee \cdots \vee x_{n_1}^{\bar{a}_{n_1}}$ is an implicate of f.*

Algorithm 4.1 *(Prime Implicants and Prime Implicates)*

1. *Obtain $\mathcal{F}$ from f.*
2. *For $n_1 = 2$ to $n-1$ do the followings:*
3. *Generate a partition (X_1, X_2), where $n_1 = |X_1|$ and $n_2 = |X_2|$.*
4. *Generate the next vector $\boldsymbol{a}_1$ in $\{0,1\}^{n_1}$, and do steps 5 and 6. If all the vectors are generated, then go to 3.*
5. *If $\mathcal{F}$ $(\boldsymbol{a}_1, X_2) = 2^{n_2}$, then $X_1^{\boldsymbol{a}_1}$ is a prime implicant of f. Modify $\mathcal{F}$ as follows: $\mathcal{F}(\boldsymbol{a}_1, \boldsymbol{b}_2, X_3) = -1$, where $\{X_3\} \subset \{X_2\}$, $\boldsymbol{b}_2 \in B^{n_2 - n_3}$, and $n_3 = |X_3|$.*
6. *If $\mathcal{F}(\boldsymbol{a}_1, X_2) = 0$, then $\bar{X}_1^{\boldsymbol{a}}$ is a prime implicate of f. Modify $\mathcal{F}$ as follows: $\mathcal{F}(\boldsymbol{a}_1, \boldsymbol{b}_2, X_3) = -2$, where $\{X_3\} \subset \{X_2\}$, and $\boldsymbol{b}_2 \in B^{n_2 - n_3}$.*
7. *In $\mathcal{F}$, positive numbers denote prime implicants, and zeros denote prime implicates.*

C. Complexity Analysis

Here, we will consider the complexity of Algorithm 4.1. Assume that we have the Arith_TDD for f. Also assume that the value of $\mathcal{F}$ can be evaluated in n steps, where n represents the number of input variables. Step 3 select

TABLE 5.1
Sizes of BDDs and Arith_TDDs for Different Numbers Input Variables.

IN	BDD	Arith_TDD	Terminal
5	13	50	13
6	25	131	18
7	39	342	32
8	74	894	51
9	126	2293	78
10	241	6390	116
11	419	17357	193
12	730	47090	280
13	1268	130028	429
14	2292	360285	629
15	4310	1011683	909
16	8310	2864597	1266

one of $C(n, n_1)$ combinations. Step 4 selects 2_1^n different combinations. Then, the computational complexity is bounded by

$$\sum_{n_1=2}^{n-1} C(n, n_1) 2^{n1} = 3^n - 2^n - 1 - 2n.$$

This means that the computation time is $O(3^n)$.

V. Experimental Results

A. Sizes of Arith_TDDs for Different Number of Inputs

We generated pseudo random logic functions of n variables for $n = 5 \sim 16$. The numbers of true minterms were near to 2^{n-1}. Table 5.1 shows the numbers of non-terminal nodes in BDDs and Arith_TDDs for different values of n. It also shows the numbers of terminal nodes in Arith_TDDs. We can observe that the sizes of BDDs increase with $O(2^n)$, while the sizes of Arith_TDDs increase with $O(3^n)$.

B. Sizes of Arith_TDDs for Different Densities

We generated pseudo random logic functions of 10 variables having different numbers of true minterms. The density is defined by

$$den(f) = \frac{\text{\# of true minterms}}{2^n} \times 100.$$

Table 5.2 shows the numbers of non-terminal nodes in BDDs and Arith_TDDs for different values of den. Each value is the average of 10 functions. The size of Arith_TDDs takes their maximum value when the density is near 60%. This implies that when the density of the function f is greater than 50%, we should use $\bar{f}$ instead of f, since we can also use $\bar{f}$ to find decompositions, prime implicants, and prime implicates.

C. Sizes of Arith_TDDs for Various Benchmark Functions

Table 5.3 compares the sizes of Arith_TDDs and BDDs for various benchmark functions [25]. *9sym* is a totally symmetric function and the sizes of the BDD and the Arith_TDD are small. We constructed a BDD and an Arith_TDD for each output separately. IN denotes the number of variables. For example, *adr4* has 5 outputs, where the function representing the least significant bit (lsb) depends on only two variables: It is the EXOR function. Since *adr4* is a partially symmetric function, the sizes of DDs are relatively small. On the other hand, in the case of *chkn*, the size of the Arith_TDD for the 4-th output is very large: About 10^5 times larger than the corresponding BDD. It is known that *t481* is a completely decomposable function [21]. So, the sizes of both DDs are relatively small.

TABLE 5.2
Sizes of BDDs and Arith_TDDs for Different Densities.

Density (%)	BDD	Arith_TDD	Terminal
10	137.6	3046.3	50.3
20	182.9	4601.9	79.1
30	211.0	5573.4	97.6
40	229.9	6149.0	112.6
50	234.2	6494.5	122.7
60	232.0	6463.4	127.1
70	212.3	6124.5	131.5
80	183.0	5384.4	117.3
90	138.1	3831.8	101.6

TABLE 5.3
Sizes of BDDs and Arith_TDDs for Various Benchmark Functions.

Function name	IN	BDD	Arith TDD	Terminal
9sym	9	33	149	19
adr4	8	11	167	44
	8	19	120	26
	6	13	45	12
	4	7	15	6
	2	3	4	3
chkn	17	19	392	81
	16	21	302	44
	19	39	2295	228
	26	98	11965587	318955
	20	32	3503	732
	20	44	9786	1870
	18	48	19485	2321
mlp4	8	16	128	21
	8	34	491	37
	8	52	688	40
	8	48	680	39
	8	41	506	31
	6	17	107	14
	4	7	20	6
	2	2	3	2
t481	16	32	5334	386
tial	8	61	648	23
	10	100	2668	82
	12	172	10072	285
	14	219	56503	430
	13	117	25943	828
	14	214	45026	959
	12	73	9159	520
	10	23	1063	96

VI. Conclusions and Comments

In this paper, we introduced arithmetic ternary decision diagrams (Arith_TDDs), that denote the extended weight functions, integer functions showing the numbers of true minterms in the cubes. In the Arith_TDD, the third edge denotes $f_2 = f_0 + f_1$, where + is an integer addition. The Arith_TDDs, and extended weight functions are useful to detect functional decompositions, prime implicants and prime implicates. Similar idea are also used in [7, 9], but they used linear arrays or matrices in their computations.

Experimental results showed that Arith_TDDs are much larger than BDDs. Therefore, the applications of Arith_TDDs are limited to the functions with small number of inputs. However, Arith_TDD or extended weight functions contain useful information that would be useful for other applications.

Since Arith_TDD is often very large to build, we can use mod_p_Arith_TDD instead. In the mod_p_Arith_TDD, the third edge denotes $f_2 = f_0 + f_1 \pmod p$. By using mod_p_Arith_TDDs for $p = 2, 3, 5$, and 7, we can find the bipartitions for which f has no decompositions. Mod_p_Arith_TDDs are smaller than the Arith_TDD, but they are still larger than corresponding BDDs. A decomposition method using only the mod_2_Arith_TDD (i.e., EXOR_TDD) has been developed independently [24].

Acknowledgments

This work was supported in part by a Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Culture, and Sports of Japan, as well as the Takayanagi Foundation for Electronics Science and Technology. Mr. Matsuura worked for programming, experiments, drawing pictures, and edition of the LaTeX files.

References

[1] R. L. Ashenhurst, "The decomposition of switching functions," *In Proceedings of an international symposium on the theory of switching*, pp. 74-116, April 1957.

[2] V. Bertacco and M. Damiani, "The disjunctive decomposition of logic functions," *Proc. ICCAD*, pp. 78-82, Nov. 1997.

[3] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677-691, Aug. 1986.

[4] S-C. Chang, M. Marek-Sadowska, and T. Hwang, "Technology mapping for LUT FPGA's based on decomposition of binary decision diagrams," *IEEE Trans. CAD*, Vol. CAD-15, No. pp. 1226-1236, Oct. 1996.

[5] H. A. Curtis, *A New Approach to the Design of Switching Circuits*, D. Van Nostrand Co., Princeton, NJ, 1962.

[6] M. A. Harrison, *Introduction to Switching and Automata Theory*, McGraw-Hill, 1965.

[7] S. L. Hurst, D. M. Miller, J. C. Muzio, *Spectral Techniques in Digital Logic*, Academic Press, Bristol, 1985.

[8] Y-T. Lai, M. Pedram, and S. B. K. Vrudhula, "EVBDD-based algorithm for integer linear programming, spectral transformation, and functional decomposition," *IEEE Trans. CAD*, Vol. 13, No. 8, pp. 959-975, Aug. 1994.

[9] R. J. Lechner, "Harmonic analysis of switching functions," in *A. Mukhopadhay, Ed., Recent Developments of in Switching Theory*, Academic Press, 1971.

[10] Y. Matsunaga, "An exact and efficient algorithm for disjunctive decomposition," *SASIMI'98*, pp. 44-50, Oct. 1998.

[11] S. Minato, "Fast generation of prime-irredundant covers from binary decision diagrams," *IEICE Trans. Fundamentals*, Vol. E76-A, No. 6, pp. 976-973, June 1993.

[12] S. Muroga, *Logic Design and Switching Theory*, John Wiley & Sons, 1979.

[13] G. Papakonstantinou, "Minimization of modulo-2 sum of products," *IEEE Trans. Comput.*, Vol. C-28, pp. 163-167, 1979.

[14] T. Sasao (ed.), *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[15] T. Sasao, "FPGA design by generalized functional decomposition," Chapter 11 in [14].

[16] T. Sasao, "Optimization of pseudo-Kronecker expressions using multiple-place decision diagrams," *IEICE Transactions on Information and Systems*, Vol. E76-D, No. 5, pp. 562-570, May 1993.

[17] T. Sasao and M. Fujita (ed.), *Representations of Discrete Functions*, Kluwer Academic Publishers, 1996.

[18] T. Sasao and F. Izuhara, "Exact minimization of FPRMs using multi-terminal EXOR TDDs," Chapter 8 in [17].

[19] T. Sasao, "Ternary decision diagrams and their applications," Chapter 12 in [17].

[20] T. Sasao, "Ternary decision diagrams: Survey," *Proc. International Symposium on Multiple-valued Logic*, pp. 241-250, Nova Scotia, May 1997.

[21] T. Sasao and M. Matsuura, "DECOMPOS: An integrated system for functional decomposition," *1998 International Workshop on Logic Synthesis*, Lake Tahoe, June 1998.

[22] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

[23] T. Sasao, "Totally undecomposable functions: Applications to efficient multiple-valued decompositions," *Proc. International Symposium on Multiple-valued Logic*, pp. 59-65, May 1999.

[24] R. S. Stanković, "Matrix-valued EXOR-TDDs in decomposition of switching functions," *Proc. International Symposium on Multiple-valued Logic*, pp. 154-159 May 1999.

[25] S. Yang, "Logic synthesis and optimization benchmark user guide, Version 3.0," *MCNC*, Jan. 1991.