PAPER *Special Section on VLSI Design and CAD Algorithms*

Exact Minimization of FPRMs for Incompletely Specified Functions by Using MTBDDs

Debatosh DEBNATH[†a)], ***Nonmember and*** **Tsutomu SASAO**[††b)], ***Member***

SUMMARY Fixed polarity Reed-Muller expressions (FPRMs) exhibit several useful properties that make them suitable for many practical applications. This paper presents an exact minimization algorithm for FPRMs for incompletely specified functions. For an n-variable function with α unspecified minterms there are $2^{n+\alpha}$ distinct FPRMs, and a minimum FPRM is one with the fewest product terms. To find a minimum FPRM the algorithm requires to determine an assignment of the incompletely specified minterms. This is accomplished by using the concept of *integer-valued functions* in conjunction with an *extended truth vector* and a *weight vector*. The vectors help formulate the problem as an assignment of the variables of integer-valued functions, which are then efficiently manipulated by using multi-terminal binary decision diagrams for finding an assignment of the unspecified minterms. The effectiveness of the algorithm is demonstrated through experimental results for code converters, adders, and randomly generated functions.

key words: *AND-EXOR, Reed-Muller expression, FPRM, exact minimization, incompletely specified function*

1. Introduction

A fixed polarity Reed-Muller expression (FPRM) is one of the canonical AND-EXOR expressions [23]. FPRMs are a generalization of positive polarity Reed-Muller expressions (PPRMs). A PPRM, which is unique for a completely specified function, is an AND-EXOR expression with only uncomplemented literals. PPRMs are also known as Zhegalkin polynomials after the Russian logician Ivan I. Zhegalkin who first published this canonical form [39]. Each variable in an FPRM can appear either in complemented or uncomplemented form. An n-variable completely specified function has 2^n distinct FPRMs. For incompletely specified functions, the number of FPRMs increases exponentially with the increase in the number of unspecified minterms. There exists $2^{n+\alpha}$ distinct FPRMs for an n-variable function with α unspecified minterms, and the objective of the exact minimization is to find one of those FPRMs that requires the fewest product terms. Such an FPRM is often referred to as the exact minimum FPRM of the function.

It is believed that AND-EXOR expressions require fewer product terms than conventional sum-of-products expressions (SOPs) for arithmetic and error checking circuits [24], [26]. FPRMs for these functions often also require fewer product terms than SOPs [23]–[26]. In addition, FPRMs have useful properties that are unavailable in other classes of expressions. This makes FPRMs suitable for many practical applications, several of which are summarized in the following:

1) Two- and multi-level testable networks. Sarabi and Perkowski showed that n-input two-level networks that realize FPRMs can be tested for single stuck-at fault by using only $n + 4$ test vectors [22]. The test vectors can be easily generated once the FPRM is available. Tsai and Marek-Sadowska used FPRM-based two-level initial networks to derive 100 percent single stuck-at fault testable multi-level networks [33]. A predetermined set of test patterns serve as test vectors, which makes test pattern generation unnecessary.

2) Area efficient multi-level networks. Tsai and Marek-Sadowska took advantage of FPRMs in designing area efficient multi-level circuits [34]. The circuits are derived from FPRMs mainly by applying algebraic factorizations and redundancy removal techniques [8]. For a set of arithmetic and other benchmark functions, the method obtains a good percentage of improvement in area as compared to Berkeley SIS [29]. Chattopadhyay et al. also employed FPRMs in multi-level minimizer KGPMIN that obtains high quality solutions [5]. Many circuits exhibit a sporadic combination of AND-OR and AND-EXOR logic. KGPMIN uses SOPs and FPRMs to evaluate the suitability in realizing each of the small portions of a circuit by using one of these logic.

3) Low-power logic synthesis. Narayanan and Liu took advantage of special properties of FPRMs in developing a low-power multi-level logic synthesis technique for EXOR-based circuits [18], [19]. The method often outperforms SIS in optimizing both area and power at the same time. Tsai and Marek-Sadowska's FPRM-based multi-level minimization algorithm [34] also produces circuits with lower power consumption than that obtained by using SIS.

4) Boolean matching and symmetry detection. Boolean matching is a crucial step in cell-library binding that determines if a cell from a library can implement a portion of the technology-independent network [8]. Davio et al. [7], Tsai and Marek-Sadowska [35], and Chang and Falkowski [4] utilized FPRMs as a tool to find Boolean matching which is also known as detection of equivalence relations of switching functions [8]. Tsai and Marek-Sadowska also used

Manuscript received March 23, 2005.
Manuscript revised June 14, 2005.
Final manuscript received August 11, 2005.

[†]The author is with the Department of Computer Science and Engineering, Oakland University, Rochester, Michigan 48309, U.S.A.
[††]The author is with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka-shi, 820-8502 Japan.
a) E-mail: debnath@oakland.edu
b) E-mail: sasao@cse.kyutech.ac.jp

DOI: 10.1093/ietfec/e88–a.12.3332

Copyright © 2005 The Institute of Electronics, Information and Communication Engineers

FPRMs to detect symmetric variables of switching functions [32]. The presence of symmetric variables helps solve many problems in logic synthesis more efficiently through the use of specific algorithms that take advantage of such information [26], [30], [32], [35].

5) Image compression. Iravani and Perkowski outlined a method for image compression by using an FPRM minimizer for incompletely specified functions [15, p.94]. Image compression by using minimized FPRMs for completely specified functions are also gaining attention [11], [15].

For completely specified functions, numerous exact and heuristic minimization algorithms for FPRMs exist [9], [10], [25], [37]. However, little research has been done to minimize FPRMs for incompletely specified functions.

Tran discussed a graphical procedure, which is based on a trial-and-error method, to simplify FPRMs for incompletely specified functions [31]. The method can be applicable to functions with up to six variables. By using spectral techniques [14], Varma and Trachtenberg developed heuristic algorithms to simplify PPRMs for incompletely specified functions [36]. Chang and Falkowski reported methods to simplify FPRMs for incompletely specified functions [2], [3]. Zilic and Vranesic presented heuristic schemes to compute multiple-valued Reed-Muller transform for incompletely specified functions that are also applicable to binary-valued functions [40], [41]. Popel developed a heuristic minimization technique for FPRMs for incompletely specified functions by using information theoretical approach [20].

McKenzie et al. developed a branch and bound algorithm for the exact minimization of PPRMs for incompletely specified functions [17]. Green described an exhaustive search method [12]. Zakrevskij formulated the exact minimization of PPRMs for incompletely specified functions as a solution of a system of linear logical equations, and presented experimental results for functions with up to 20 specified minterms [38]. McKenzie et al. [17] and Zakrevskij [38] also considered heuristic simplification methods. Recently, Habib developed an exact minimization algorithm for FPRMs that can handle incompletely specified functions; however, no benchmark results are presented [13].

In this paper we present an algorithm to obtain exact minimum FPRMs for incompletely specified functions. The method is based on the computation of an *extended truth vector* and a *weight vector* from a function [7], [25]. These vectors are also used for the exact minimization of FPRMs for completely specified functions [7], [25]. The extended truth vector is computed from the truth vector, and the weight vector is computed from the extended truth vector. For completely specified functions each of the entries of the extended truth vector and the weight vector holds a binary and an integer value, respectively. The smallest integer value in the weight vector represents the number of product terms in the exact minimum FPRM.

The most challenging task in optimizing an FPRM for an incompletely specified function is to find an assignment of the unspecified minterms that minimizes the number of product terms required by the expression. By using extended truth vectors and weight vectors, we formulate this problem as an assignment of an *integer-valued function* (Sect. 2). To optimize FPRMs for a function with α unspecified minterms we represent each of these minterms by a binary variable. As a result each of the entries of the extended truth vector and the weight vector becomes an α-variable binary- and integer-valued function, respectively. An exact minimum FPRM corresponds to an assignment of the α binary variables that minimizes at least one of the entries in the weight vector. In other words in finding an exact minimum FPRM we look for an assignment of the variables for which at least one of the integer-valued functions in the weight vector evaluates to the smallest integer value. This integer represents the number of product terms in the exact minimum FPRM for the incompletely specified function. We use multi-terminal binary decision diagrams (MTBDDs) to efficiently represent and manipulate integer-valued functions [6]. Once an MTBDD is built the assignment of the variables that evaluates the integer-valued function to the smallest value is straightforward.

In light of the existing techniques the novelty and significance of our contribution are summarized in the following:

1) Problem formulation. We formulate the problem of exact minimization of FPRMs for an incompletely specified function as assignment of the variables of integer-valued functions. Although the concept of extended truth vectors and weight vectors are known for over 25 years [7], their application in solving exact minimization of FPRMs for incompletely specified functions is unique.

2) Application of MTBDDs. To the best of our knowledge this is the first attempt in using MTBDDs to optimize FPRMs for incompletely specified functions. It should be noted that MTBDDs are also employed by Sasao and Izuhara for minimizing FPRMs for *completely* specified functions [25]. The MTBDDs in this paper and that in the Sasao-Izuhara's paper represent different kinds of information for the given functions. Moreover, inputs to both of the algorithms are different: one accepts completely specified and the other accepts incompletely specified functions.

3) Techniques for building MTBDDs. To find an assignment of the unspecified minterms the algorithm requires to perform arithmetic additions of a set of integer-valued functions. The operands and the result of the additions are represented by MTBDDs. Although there are excellent public domain software such as CUDD to help implement these tasks [30], a straightforward implementation by using even such efficient software often requires excessive computation time and memory resources for solving a simple problem. We developed a systematic method to overcome this limitation. First, some operations on integer-valued functions can be done very efficiently by exploiting their properties. Section 2 presents two of such properties that we take advantage of when building MTBDDs. We are unaware of if such techniques are used previously in efficient construction of MTBDDs. Second, to add a large set of integer-valued functions a series of additions is required which generates many intermediate results. Although the final result in many cases has a manageable MTBDD representation, the intermediate results are often too large to store. To alleviate the problem we developed a method for the selection of operands that try to delay any increase in the number of variables on which each of the intermediate MTBDDs depend (Sect. 4). The strategy is based on the intuition that an MTBDD increases in size if the number of variables on which it depends also increases.

4) Experimental results. We obtained experimental results for arithmetic functions, code converters, and randomly generated functions. To the best of our knowledge no such experimental results are reported for exact minimum FPRMs for incompletely specified functions.

The remainder of the paper is organized as follows: Sect. 2 introduces terminology and presents basic properties. Section 3 outlines computation methods for extended truth vectors and weight vectors which are the basis of the proposed work. Section 4 develops a systematic method for the exact minimization. Section 5 reports experimental results. Section 6 presents conclusion and outlines future work.

2. Definitions and Basic Properties

This section defines the basic terminology and states important properties that are necessary to explain the material in the paper. In this paper the operators '+' and '$\oplus$' indicate arithmetic and mod-2 additions, respectively; the operator '$\times$' indicates arithmetic multiplication.

Definition 1: An n-variable *switching function* f is a mapping $f : \{0,1\}^n \to \{0,1\}$. And an n-variable *integer-valued function* g is a mapping $g : \{0,1\}^n \to \{0,1,\ldots,p-1\}$ where $p \geq 2$.

It should be noted that switching functions are a subset of integer-valued functions, and in this paper a switching function is often referred to as a *function* or as a *binary-valued function*.

Definition 2: An n-variable integer-valued function $f(x_1, x_2,\ldots,x_n)$ can be written as $\sum_{j=0}^{2^n-1} m_j x_1^{b_1} x_2^{b_2} \cdots x_n^{b_n}$, where $m_j \in \{0,1,\ldots,p-1\}$ $(p \geq 2)$, $b_1, b_2,\ldots,b_n \in \{0,1\}$ such that $b_1 b_2 \cdots b_n$ is the n-bit binary number representing j, $x_i^{b_i} = \bar{x}_i$ when $b_i = 0$, $x_i^{b_i} = x_i$ when $b_i = 1$, and $i = 1,2,\ldots,n$. Then $[m_0, m_1,\ldots,m_{2^n-1}]$ is the *truth vector* of f.

Example 1: The truth vector of the three-variable switching function $\bar{x}_1\bar{x}_2\bar{x}_3 \vee x_1$ is $[1,0,0,0,1,1,1,1]$, and that of the three-variable integer-valued function $3x_1 + 4x_2x_3 + 2\bar{x}_3$ is $[2,0,2,4,5,3,5,7]$.

Property 1: Let f be a switching function. Then $f+\bar{f} = 1$.

Example 2: Let the two-variable switching function f be $[1,0,0,0]$. Then $f + \bar{f} = [1,0,0,0] + [0,1,1,1] = [1,1,1,1] = 1$.

Property 2: Let f be an integer-valued function. Then $\underbrace{f+f+\cdots+f}_{k \text{ operands}} = k \times f$.

Example 3: Let the two-variable integer-valued function f be $[1,0,3,5]$. Then $f+f+f = 3\times[1,0,3,5] = [3,0,9,15]$.

Definition 3: An n-variable switching function $f(x_1, x_2, \ldots, x_n)$ can be written as a *fixed polarity Reed-Muller expression (FPRM)* $\bigoplus_{j=0}^{2^n-1} a_j x_1^{b_1} x_2^{b_2} \cdots x_n^{b_n}$, where $a_j \in \{0,1\}$, $b_1, b_2,\ldots,b_n \in \{0,1\}$ such that $b_1 b_2 \cdots b_n$ is the n-bit binary number representing j, $x_i^{b_i} = 1$ when $b_i = 0$, $x_i^{b_i} \in \{\bar{x}_i, x_i\}$ such that for each i either $\bar{x}_i$ or x_i appear throughout the expression when $b_i = 1$, and $i = 1,2,\ldots,n$.

In an FPRM each variable can appear either in complemented or uncomplemented form, i.e., polarity of each variable can be chosen in two ways. Thus, for an n-variable completely specified function there are 2^n distinct FPRMs.

In this paper we consider FPRMs for single-output functions, because the presented algorithm can handle only such functions. Multiple-output functions are treated as a set of single-output functions.

Definition 4: *Polarity vector* $(b_1, b_2,\ldots,b_n)$ for an FPRM of an n-variable switching function $f(x_1, x_2,\ldots,x_n)$ is a binary vector with n elements, where $b_i = 0$ indicates variable x_i is used in the uncomplemented form (x_i) and $b_i = 1$ indicates variable x_i is used in the complemented form $(\bar{x}_i)$.

Example 4: Let the three-variable switching function f be $x_1x_3 \vee \bar{x}_2\bar{x}_3$ and $(0,1,1)$ be a polarity vector for an FPRM of f. Since x_1x_3 and $\bar{x}_2\bar{x}_3$ are disjoint, we can write $f = x_1x_3 \oplus \bar{x}_2\bar{x}_3$. By putting $x_3 = 1 \oplus \bar{x}_3$ in the expression for f, we have $f = x_1(1 \oplus \bar{x}_3) \oplus \bar{x}_2\bar{x}_3 = x_1 \oplus x_1\bar{x}_3 \oplus \bar{x}_2\bar{x}_3$, which is the FPRM for f with polarity vector $(0,1,1)$.

3. Extended Truth Vector and Weight Vector

For an n-variable completely specified switching function there are 2^n distinct FPRMs, and the minimization problem is to find a polarity vector that produces an FPRM with minimum number of products. Once the polarity vector is determined, generation of an FPRM is relatively easy [7], [25].

Figure 1 illustrates a method for the exact minimization of FPRMs for three-variable switching function. The method is based on the computation of *extended truth vector* and *weight vector* [7], [25]. The extended truth vector $[t_0, t_1,\ldots,t_{26}]$ is computed from the truth vector $[m_0, m_1,\ldots,m_7]$ of a given switching function, and the weight vector $[w_0, w_1,\ldots,w_7]$ is computed from the extended truth vector. In general, for an n-variable completely specified switching function, extended truth vector is a binary vector $[t_0, t_1,\ldots,t_{3^n-1}]$ with 3^n elements, and weight vector is an integer vector $[w_0, w_1,\ldots,w_{2^n-1}]$ with 2^n elements. Each element of the weight vector is associated with a *polarity vector*, which is shown at the rightmost side in Fig. 1. In general, for an n-variable switching function f, polarity vector for w_j is a binary vector $(b_1, b_2,\ldots,b_n)$

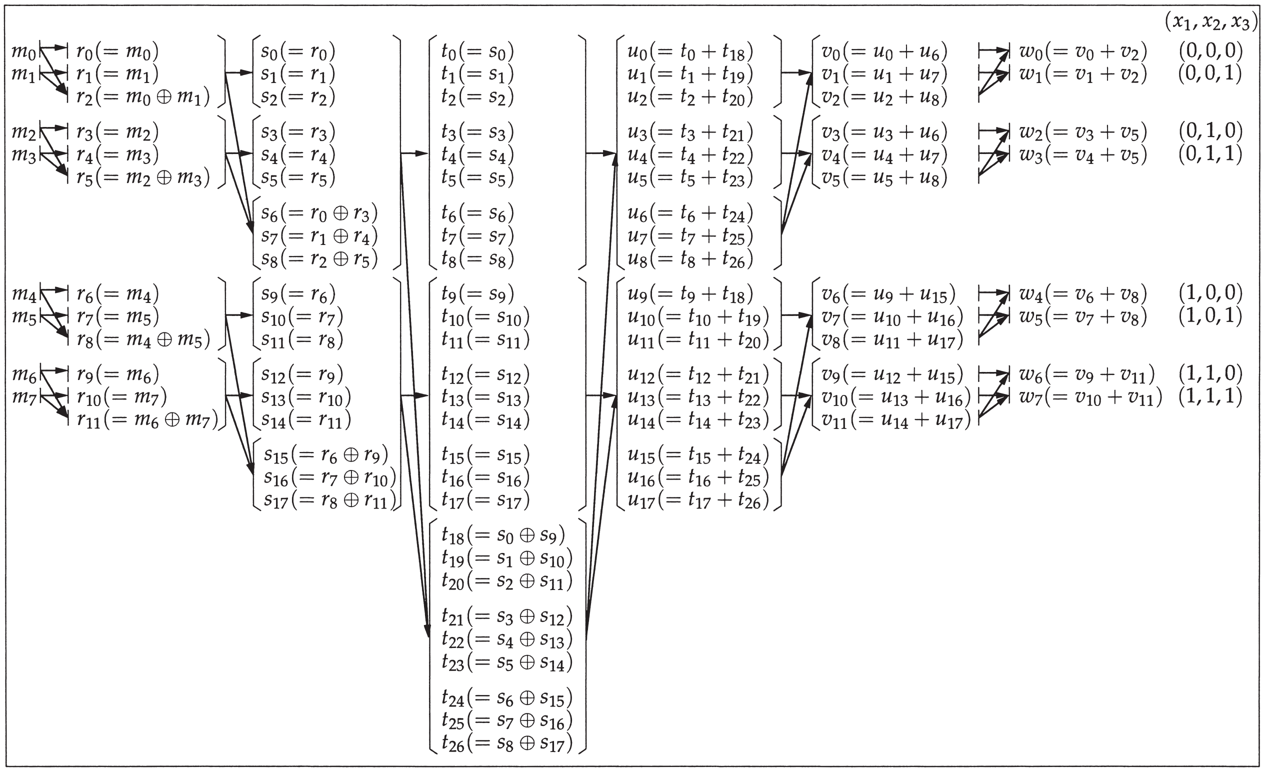

Fig. 1 Computation of extended truth vector and weight vector for three-variable switching function.

such that $b_1b_2\cdots b_n$ is the n-bit binary number representing $j\,(j=0,1,\ldots,2^n-1)$, and w_j represents the number of products in the FPRM for f with polarity vector $(b_1,b_2,\ldots,b_n)$.

In Fig. 1 the calculations are done in six main steps. The binary values r_i's, s_i's, and t_i's are calculated from m_i's, r_i's, and s_i's, respectively, and the integer values u_i's, v_i's, and w_i's are calculated from t_i's, u_i's, and v_i's, respectively, where i can take different values as shown in Fig. 1. Expressions for several t_i's and w_0 for three-variable function $[m_0,m_1,\ldots,m_7]$ are obtained from Fig. 1 as follows:

$$\begin{aligned} w_0 &= v_0 + v_2 \\ &= u_0 + u_6 + u_2 + u_8 \\ &= t_0 + t_{18} + t_6 + t_{24} + t_2 + t_{20} + t_8 + t_{26} \\ &= t_0 + t_2 + t_6 + t_8 + t_{18} + t_{20} + t_{24} + t_{26}, \end{aligned} \tag{1}$$

where

$$\begin{aligned} t_0 &= s_0 = r_0 = m_0, \\ t_2 &= s_2 = r_2 = m_0 \oplus m_1, \\ t_6 &= s_6 = r_0 \oplus r_3 = m_0 \oplus m_2, \\ t_8 &= s_8 = r_2 \oplus r_5 = m_0 \oplus m_1 \oplus m_2 \oplus m_3, \\ t_{18} &= s_0 \oplus s_9 = r_0 \oplus r_6 = m_0 \oplus m_4, \\ t_{20} &= s_2 \oplus s_{11} = r_2 \oplus r_8 = m_0 \oplus m_1 \oplus m_4 \oplus m_5, \\ t_{24} &= s_6 \oplus s_{15} = r_0 \oplus r_3 \oplus r_6 \oplus r_9 \\ &= m_0 \oplus m_2 \oplus m_4 \oplus m_6, \\ t_{26} &= s_8 \oplus s_{17} = r_2 \oplus r_5 \oplus r_8 \oplus r_{11} = m_0 \oplus m_1 \\ &\quad \oplus m_2 \oplus m_3 \oplus m_4 \oplus m_5 \oplus m_6 \oplus m_7. \end{aligned} \tag{2}$$

In a similar manner, expressions for w_1 and the associated t_i's can be obtained from Fig. 1 as follows:

$$\begin{aligned} w_1 &= v_1 + v_2 \\ &= u_1 + u_7 + u_2 + u_8 \\ &= t_1 + t_{19} + t_7 + t_{25} + t_2 + t_{20} + t_8 + t_{26} \\ &= t_1 + t_2 + t_7 + t_8 + t_{19} + t_{20} + t_{25} + t_{26}, \end{aligned}$$

where

$$\begin{aligned} t_1 &= s_1 = r_1 = m_1, \\ t_7 &= s_7 = r_1 \oplus r_4 = m_1 \oplus m_3, \\ t_{19} &= s_1 \oplus s_{10} = r_1 \oplus r_7 = m_1 \oplus m_5, \\ t_{25} &= s_7 \oplus s_{16} = r_1 \oplus r_4 \oplus r_7 \oplus r_{10} \\ &= m_1 \oplus m_3 \oplus m_5 \oplus m_7, \end{aligned}$$

and the expressions for t_2, t_8, t_{20}, and t_{26} are shown in Eq. (2). Expressions for other w_i's can be obtained in a similar manner.

4. Minimization Techniques

For an n-variable incompletely specified switching function with α unspecified minterms there are $2^{n+\alpha}$ distinct FPRMs, and the minimization problem is to find a polarity vector and an assignment of the unspecified minterms to 0's and 1's that produce an FPRM with minimum number of products. Once the polarity vector and the assignment of the unspecified minterms are determined, an FPRM can be easily generated [7], [25]. This section presents the key techniques for the exact minimization of FPRMs for incompletely specified functions. The algorithm is based on extended truth vectors and weight vectors, and the problem is formulated as an assignment of the variables of integer-valued functions. The novelty of our contribution are summarized in Sect. 1.

For an n-variable switching function with α unspecified minterms $d_1, d_2, \ldots, d_\alpha$, extended truth vector is a vector of switching functions $t_i(d_1, d_2, \ldots, d_\alpha)$ $(i = 0, 1, \ldots, 3^n - 1)$, and weight vector is a vector of integer-valued functions $w_j(d_1, d_2, \ldots, d_\alpha)$ $(j = 0, 1, \ldots, 2^n - 1)$. For three-variable case, all the t_i's and w_j's can be obtained from Fig. 1. Extension to the functions with more variables is straightforward.

Definition 5: Let the *minimum value* of the α-variable integer-valued function $w(d_1, d_2, \ldots, d_\alpha)$, denoted by w^{min}, be $\min_{0 \le i \le 2^\alpha - 1} m_i$, where $[m_0, m_1, \ldots, m_{2^\alpha - 1}]$ represents the truth vector for w.

Let $[w_0, w_1, \ldots, w_{2^n-1}]$ be the weight vector for an n-variable incompletely specified switching function $f(x_1, x_2, \ldots, x_n)$, and w_j^{min} be the minimum value for $w_j(d_1, d_2, \ldots, d_\alpha)$ where $d_1, d_2, \ldots, d_\alpha$ represent unspecified minterms of f. Let $0 \le k \le 2^n - 1$ and $a_1, a_2, \ldots, a_\alpha \in \{0, 1\}$ such that $w_k(a_1, a_2, \ldots, a_\alpha) = \min_{0 \le j \le 2^n - 1} w_j^{min}$. Let $c_1 c_2 \cdots c_n$ be the n-bit binary number representing k. Then, $(a_1, a_2, \ldots, a_\alpha)$ represents an assignment of $(d_1, d_2, \ldots, d_\alpha)$ and $(c_1, c_2, \ldots, c_n)$ represents a polarity vector that produce a minimum FPRM for f.

Example 5: Consider a three-variable switching function $f(x_1, x_2, x_3)$ whose truth vector $[m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7] = [d_0, 0, d_2, d_3, 1, 1, 1, 0]$, where d_0, d_2, and d_3 are unspecified minterms. By putting the value of m_i $(i = 0, 1, \ldots, 7)$ in Eq. (2), we have

$$\begin{aligned} t_0 &= d_0, & t_{18} &= 1 \oplus d_0, \\ t_2 &= d_0, & t_{20} &= d_0, \\ t_6 &= d_0 \oplus d_2, & t_{24} &= d_0 \oplus d_2, \\ t_8 &= d_0 \oplus d_2 \oplus d_3, & t_{26} &= 1 \oplus d_0 \oplus d_2 \oplus d_3. \end{aligned} \tag{3}$$

By using Property 1 to Eq. (3), we have $t_{18} + t_{20} = 1$ and $t_8 + t_{26} = 1$. Also, by using Property 2 to Eq. (3), we have $t_6 + t_{24} = 2(d_0 \oplus d_2)$ and $t_0 + t_2 = 2d_0$. Thus, from Eq. (1) and Eq. (3), we obtain

$$w_0 = 2 + 2d_0 + 2(d_0 \oplus d_2). \tag{4}$$

Equation (4) shows that w_0 cannot be less than 2 and it is independent of d_3. By inspection, we have $w_0 = 2$, when $d_0 = d_2 = 0$; however $w_0 = 6$, when $d_0 = 1$ and $d_2 = 0$. Thus, the minimum value for w_0 is 2. Similarly, we can obtain minimum value for w_i, when $i = 1, 2, \ldots, 7$.

To manipulate integer-valued function we use multi-terminal binary decision diagram (MTBDD) [6]. An MTBDD, which is a natural extension of binary decision diagram (BDD) [1], is a directed acyclic graph with multiple terminal nodes each of which has an integer value. Arithmetic operations, such as addition and multiplication, between integer-valued functions can be efficiently performed by using MTBDDs. It should be noted that switching functions are a subset of integer-valued functions and an MTBDD for a switching function is a BDD. We use MTBDD data structure to perform Boolean operations between switching functions.

A straightforward method to build MTBDDs for weight vector requires excessive computation time and memory resources, because they represent all possible FPRMs for the given incompletely specified function. However, we are only interested in an FPRM with the fewest products. Suppose we have an FPRM for the given function with $t_{\text{threshold}} + 1$ products, then it is sufficient to search for an FPRM with $t_{\text{threshold}}$ or fewer products. If such an FPRM does not exist then the FPRM with $t_{\text{threshold}} + 1$ products is the minimum FPRM. Thus, to restrict the search space without sacrificing the minimality of the solution, we use *threshold value*, $t_{\text{threshold}}$, during construction of MTBDDs. The threshold value can be obtained by using any simplification program for FPRMs.

Based on the above discussions, we develop the following algorithm for exact minimization of FPRM for incompletely specified n-variable switching function f.

Algorithm 1 (Exact Minimization):

1. Get the user supplied threshold value, $t_{\text{threshold}}$.
2. Prepare the extended truth vector $[t_0, t_1, \ldots, t_{3^n-1}]$ for f. (Figure 1 shows the computation method for the extended truth vector for three-variable functions. Extension to the functions with more variables is straightforward. Each element of the extended truth vector is a switching function represented as an MTBDD.)
3. Let $[w_0, w_1, \ldots, w_{2^n-1}]$ be the weight vector for f. For $i = 0$ to $2^n - 1$, do the following:
   (a) Gather elements from the extended truth vector such that
   $$w_i = \sum_{t \in T_i} t, \tag{5}$$
   where $T_i \subset \{t_0, t_1, \ldots, t_{3^n-1}\}$. (Figure 1 shows how to gather elements corresponding to w_i from the extended truth vector for three-variable function. Extension to the functions with more variables is straightforward. It is obvious from Fig. 1 that the number of elements in w_i is 2^n.)
   (b) Apply Properties 1 and 2 to Eq. (5). Thus, we have $w_i = a + \sum_{0 \le j \le L-1} b_j u_j$, where $a \ge 0$, $b_j \ge 1$, $u_j \in T_i$, and $L \le 2^n$.
   (c) Construct an MTBDD for w_i. During construction: (i) if any terminal value of an intermediate MTBDD is greater than $t_{\text{threshold}}$, set that terminal value to ∞; (ii) if an intermediate MTBDD represents constant ∞, stop the construction and assign MTBDD for w_i to ∞.
   (d) Obtain the minimum value w_i^{min} from the MTBDD for w_i. (This corresponds to finding a path from

```
1 procedure $Build\_MTBDD(a, S, D_{all})$ {
2     $D_{remain}, D_{this}$: set of support variables;
3     $S_{save}, S_{this}$: subset of $S$;
4     $c_{save}, c_{this}$: integer (counter);
5     $w_i \leftarrow a$;
6     until $S \neq \emptyset$ do {
7         $D_{remain} \leftarrow D_{all} - support(w_i)$;
8         $c_{save} \leftarrow 0$;
9         for each $d_{remain} \in D_{remain}$ do {
10            $D_{this} \leftarrow \{d_{remain}\} \cup support(w_i)$;
11            $c_{this} \leftarrow 0$; $S_{this} \leftarrow \emptyset$;
12            for each $b_j u_j \in S$ do {
13                if $support(u_j) \subseteq D_{this}$ then {
14                    $c_{this} \leftarrow c_{this} + b_j$;
15                    $S_{this} \leftarrow S_{this} \cup \{b_j u_j\}$;
16                }
17            }
18            if $c_{this} > c_{save}$ then {
19                $c_{save} \leftarrow c_{this}$;
20                $S_{save} \leftarrow S_{this}$;
21            }
22        }
23        if $S_{save} = \emptyset$ then
24            $S_{save} \leftarrow$ an element of $S$;
25        for each $b_j u_j \in S_{save}$ do
26            $w_i \leftarrow w_i + b_j u_j$; /* MTBDD operation */
27        $S \leftarrow S - S_{save}$;
28    }
29    return $w_i$; /* MTBDD */
30 }
```

Fig. 2 Pseudocode of the procedure *Build_MTBDD*.

the root to a terminal node of the MTBDD that gives a minimum value for w_i. The path represents an assignment of the unspecified minterms of the given function.)

(e) If $w_i^{min} <= t_{\text{threshold}}$: (i) save the polarity vector and an assignment of the unspecified minterms corresponding to w_i^{min}; (ii) $t_{\text{threshold}} \leftarrow w_i^{min} - 1$.

4. If any polarity vector is saved in step 3(e) then obtain an FPRM by using the most recently saved polarity vector and assignment of the unspecified minterms, otherwise report "No solution exists with $t_{\text{threshold}}$ or fewer products."

To build an MTBDD for w_i at step 3(c) we must do arithmetic addition of a set of MTBDDs, which can be arranged in numerous ways to perform addition. As outlined in Sect. 1 the arrangement influences the computation time and the sizes of the intermediate MTBDDs during addition. A naive arrangement requires excessive memory resources and long computation time. To build an MTBDD for w_i we use the procedure *Build_MTBDD*(a, S, D_{all}), the pseudocode of which is shown in Fig. 2. In the pseudocode a represents an integer that is introduced in step 3(b), S represents $\{b_0u_0, b_1u_1, \ldots, b_{L-1}u_{L-1}\}$ which is also introduced in step 3(b), D_{all} represents $\{d_1, d_2, \ldots, d_\alpha\}$ each of which stands for an unspecified minterm, and $support(w_i)$ represents the set of variables on which w_i depends.

The procedure first choses an MTBDD that depends on the fewest variables and then arranges other MTBDDs to slowly increase the number of variables in the intermediate MTBDDs. Lines 9 to 22 in Fig. 2 select $S_{save} \subseteq S$ such that $|support(w_i) \cup support(U)| - |support(w_i)| = 1$ and $\sum_{b_j u_j \in S_{save}} b_j$ is maximum, where $U = \bigcup_{b_j u_j \in S_{save}} support(u_j)$ and $|X|$ represents the number of elements in set X. In other words, lines 9 to 22 select a set of MTBDDs such that when the MTBDDs are added with an MTBDD for w_i, the resulting MTBDD depends on one more variable than the MTBDD for w_i depends. When such MTBDDs do not exist, i.e., S_{save} is empty, an MTBDD is selected on line 24 of the pseudocode. Specifically, the MTBDD $b_j u_j \in S$ is selected such that $|support(u_j) \cup support(w_i)|$ is minimized.

5. Experimental Results

We implemented the proposed method for the exact minimization of FPRMs for incompletely specified functions by using CUDD package [30] and conducted experiments on a 2.40 GHz Pentium 4 PC with two gigabytes memory running Linux. The computation time of the algorithm mainly depends on the number of unspecified minterms, the threshold value supplied to Algorithm 1, and the number of variables on which the function depends. The current implementation works favorably for many functions with eight or fewer variables. However, for functions with nine or more variables it often requires long computation time and excessive memory resources.

Table 1 shows experimental results for a set of randomly generated functions. In the table, $f(n, t, d, s)$ represents n-variable functions with t true and d unspecified minterms, where s represents non-zero seeds for the function generator. More details about these functions are provided in Appendix A, which also shows a C program for generating these functions. Columns two to five show the number of product terms required by the best FPRMs when four different methods are used to assign the unspecified minterms. Columns *dc0* and *dc1* represent the number of product terms when all the unspecified minterms are assigned to 0 and 1, respectively. Data for the *rand* column represent the best solutions that are obtained after 100,000 random assignments of the unspecified minterms. The exact minimum number of product terms that are obtained by using the proposed algorithm are provided under *exact*. The next column under *thres* shows the threshold values that are supplied to Algorithm 1. The peak number of live MTBDD nodes and the total CPU seconds are reported under columns *peak* and *time*, respectively.

We have conducted experiments by using several eight-variable functions to show the effect of the number of unspecified minterms and threshold values on the requirement of memory resources and computation time. All the functions in Table 2 have the same true minterms, and they differ in unspecified minterms. For each of the functions three different threshold values are used. Table 2 shows that lower threshold values help obtain solutions quickly and use smaller memory resources. The table also shows how the quality of the solutions improve as the number of unspecified minterms increases.

Table 1 Experimental results for randomly generated functions.

$f(n, t, d, s)$	*dc0*	*dc1*	*rand*	*exact*	*thres*	*peak*	*time(s)*
$f(6, 15, 30, 25)$	22	22	12	9	10	1732	0.11
$f(6, 12, 40, 50)$	18	18	12	6	10	11133	0.16
$f(7, 35, 50, 5)$	48	48	32	21	25	41733	2.41
$f(7, 20, 80, 5)$	34	34	28	10	15	70379	5.96
$f(7, 20, 90, 5)$	34	34	28	8	12	134166	6.72
$f(8, 8, 240, 60)$	38	38	33	3	4	371748	33.89
$f(8, 15, 230, 25)$	64	43	37	5	6	520119	107.39
$f(8, 25, 200, 50)$	72	72	64	12	12	1075438	617.27
$f(8, 100, 80, 10)$	107	105	81	51	51	5638312	972.15
$f(8, 35, 180, 10)$	84	81	71	15	15	4931098	1437.89
$f(8, 60, 160, 5)$	106	77	71	21	22	9874309	1784.32
$f(8, 100, 90, 10)$	107	103	86	47	48	20490584	3705.45
$f(8, 80, 100, 50)$	108	99	84	41	45	11554627	4167.28
$f(9, 250, 50, 5)$	217	217	188	167	170	2072822	636.81
$f(9, 200, 50, 5)$	228	215	192	172	175	6987484	1834.87
$f(9, 15, 480, 80)$	108	93	85	6	7	8192276	4917.10
$f(10, 500, 40, 25)$	474	453	425	397	420	4283146	1913.74
$f(11, 1000, 30, 1)$	951	935	921	887	900	281433	107.66
$f(12, 2000, 30, 25)$	1938	1938	1895	1874	1880	455409	648.93
$f(14, 8000, 30, 50)$	7980	7951	7882	7836	7850	855023	1279.48

Table 2 Effect of unspecified minterms and threshold values.

$f(n, t, d, s)$	*exact*	*thres*	*peak*	*time(s)*
$f(8, 50, 10, 10)$	74	74	301	0.03
		79	421	0.07
		84	587	0.11
$f(8, 50, 30, 10)$	64	64	7091	0.51
		69	23575	1.36
		74	44806	2.99
$f(8, 50, 50, 10)$	56	56	152344	16.29
		61	560865	68.24
		66	769065	123.82
$f(8, 50, 70, 10)$	53	53	5430836	1219.83
		58	8648543	2652.20
		63	14349474	4846.92
$f(8, 50, 90, 10)$	44	44	12963544	2340.06
		49	17547953	4329.68
		54	23184680	6103.72
$f(8, 50, 110, 10)$	33	33	19620571	3854.68
		38	26082221	5446.19
		43	31754252	7593.02

Table 3 Experimental results for arithmetic circuits and code converters.

out-id	*#true*	*#dc*	*dc0*	*dc1*	*rand*	*exact*	*peak*	*time(s)*
1-digit BCD adder								
0	50	156	18	18	18	2	4726928	54.85
2-digit BCD-to-binary converter								
0	50	156	9	9	9	1	315419	15.31
1	50	156	15	15	15	2	37093	11.19
2	48	156	15	15	15	3	315462	16.26
3	48	156	21	21	21	6	80316	27.72
4	48	156	20	20	20	11	11019893	9735.71
5	36	156	20	20	20	12	7787160	8259.94
6	36	156	14	8	8	3	35322	8.12
2-digit decimal incrementer								
0	50	156	9	9	9	1	37070	12.31
1	40	156	6	6	6	2	25907	7.63
2	40	156	6	6	6	2	37078	11.72
3	20	156	18	16	16	3	43057	16.59
4	50	156	12	12	12	2	37094	7.58
5	40	156	4	4	4	2	28632	6.81
6	40	156	4	4	4	2	72494	13.44
7	20	156	10	8	8	3	54777	6.82
2-digit ternary adder								
0	27	175	24	13	13	4	121987	13.73
1	27	175	24	13	13	4	123796	14.23
3-digit ternary-to-binary converter								
0	13	37	13	13	8	3	280420	0.91
1	13	37	14	14	10	6	280435	0.92
2	12	37	14	14	12	8	280439	2.47
3	11	37	12	9	9	4	280426	0.89
4	11	37	13	9	8	3	280422	0.79
4-digit ternary-to-binary converter								
0	40	175	40	39	37	4	122003	11.96
5	32	175	44	30	28	7	17670538	387.32
6	17	175	16	16	12	3	447850	19.44

Table 3 shows experimental results for arithmetic functions and code converters. An overview of the two-digit BCD-to-binary and four-digit ternary-to-binary converters are provided in Appendix B. The two-digit decimal incrementer represents combinational part of a two-digit decimal counter, where numbers are represented in binary-coded decimal format. Since the current version of the program works only for single output functions, we report experimental results for each of the outputs separately. In the table, the column *out-id* identifies each of the outputs; the least significant output bit of a function is considered to have *out-id* 0. It should be noted that the program is unable to obtain solutions for some of the outputs. The columns *#true* and *#dc* show the numbers of true and unspecified minterms, respectively. The remaining columns share their meanings with Table 1. Among six circuits considered in Table 3

Table 4 Effect of variable reordering of MTBDDs.

$f(n,t,d,s)$	#r	rtime(s)	peak	mem	time(s)
$f(7,40,80,5)$	0	0.00	4922534	376.73	43.36
	20	1765.71	1973147	101.12	1795.86
$f(8,80,100,50)$	0	0.00	11554627	664.97	4130.98
	46	14016.34	6395455	215.43	18067.74

only three-digit ternary-to-binary converter has six inputs, the other circuits have eight inputs. The peak number of live MTBDD nodes and the total CPU seconds in Table 3 are obtained by using threshold values that are one more than the number of product terms in the exact minimum FPRMs. We note that experimental results in Tables 1 and 3 are gathered without using variable reordering of MTBDDs.

We have incorporated variable reordering of MTBDDs in our algorithm. Table 4 demonstrates the effect of variable reordering on computation time and memory usages. Data are collected with and without variable reordering by using default settings of the CUDD, and sifting algorithm is used for reordering [21]. In the table, columns *#r* and *rtime* report the number of times variable reorderings are done and the CPU seconds for performing only variable reorderings, respectively. Columns *mem* and *time* show the memory requirement in megabytes and the total CPU seconds, respectively. The other two columns share their meanings with the previous tables. Table 4 shows that when variable reordering is used computation time increases dramatically with an appreciable decrease in memory requirement. Our experiments with other functions also reveal similar tendencies.

It should be noted that a comparison with the other exact minimization algorithms is difficult, because none of them explicitly reported any benchmark functions.

6. Conclusions and Comments

Minimization of FPRMs for incompletely specified functions is important at least for two reasons. First, FPRMs can play important roles in several areas of logic synthesis. Second, there are many practical functions that are incompletely specified, and very little has been done in optimizing FPRMs for such functions. Our experimental results demonstrate that extended truth vectors and weight vectors along with the concept of integer-valued functions are useful for the exact minimization of FPRMs for incompletely specified functions.

MTBDDs also play crucial roles in efficiently representing and manipulating integer-valued functions. However, MTBDDs can be practically used for handling problems with up to certain sizes. Variable reordering of MTBDDs is helpful in reducing the memory requirement when optimizing FPRMs for many functions. However, variable reordering does not significantly improve the usefulness of the algorithm, because reordering large MTBDDs takes extremely long computation time. For an n-variable function the algorithm requires to build at most 2^n MTBDDs, and no single variable order works well for two of those MTBDDs. The threshold value supplied to the algorithm also has a profound impact on the computation time and memory requirement. The impact is even greater when functions with many unspecified minterms are considered. Therefore, a powerful heuristic minimizer should be used for obtaining quality threshold values.

The exact minimization of FPRMs for incompletely specified functions is challenging if the number of unspecified minterms is large, because the search space increases exponentially with an increase in the unspecified minterms. The problem gets more complex as the number of true, false, and the unspecified minterms become nearly equal. Therefore, heuristic minimizers should be developed. Another promising problem in the area is optimization of FPRMs for incompletely specified functions considering area and power reduction of their multi-level realizations based on factored forms.

Acknowledgement

This work was supported in part by the Japan Society for the Promotion of Science and in part by the Ministry of Education, Science, Culture, and Sports of Japan.

References

[1] R.E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Trans. Comput., vol.C-35, no.8, pp.677–691, Aug. 1986.
[2] C.-H. Chang and B.J. Falkowski, "Flexible optimization of fixed polarity Reed-Muller expansions for multiple output completely and incompletely specified Boolean functions," Proc. Asia and South Pacific Design Automation Conference, pp.335–340, Sept. 1995.
[3] C.-H. Chang and B.J. Falkowski, "Adaptive exact optimisation of minimally testable FPRM expansions," IEE Proc., Comput. Digit. Tech., vol.145, no.6, pp.385–394, Nov. 1998.
[4] C.-H. Chang and B.J. Falkowski, "Boolean matching filters based on row and column weights of Reed-Muller polarity coefficient matrix," VLSI Design, vol.14, no.3, pp.259–271, 2002.
[5] S. Chattopadhyay, S. Roy, and P.P. Chaudhuri, "KGPMIN: An efficient multi-level multi-output AND-OR-XOR minimizer," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.16, no.3, pp.257–265, March 1997.
[6] E.M. Clarke, K.L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large Boolean functions with applications to technology mapping," Formal Methods Syst. Des., vol.10, no.2, pp.137–148, April 1997.
[7] M. Davio, J.-P. Deschamps, and A. Thayse, Discrete and Switching Functions, McGraw-Hill, 1978.
[8] G. De Micheli, Synthesis and Optimization of Digital Circuits, McGraw-Hill, 1994.
[9] R. Drechsler, M. Theobald, and B. Becker, "Fast OFDD-based minimization of fixed polarity Reed-Muller expressions," IEEE Trans. Comput., vol.45, no.11, pp.1294–1299, Nov. 1996.
[10] R. Drechsler, B. Becker, and N. Drechsler, "Genetic algorithm for minimisation of fixed polarity Reed-Muller expressions," IEE Proc., Comput. Digit. Tech., vol.147, no.5, pp.349–353, Sept. 2000.
[11] B.J. Falkowski, "Compact representations of logic functions for lossless compression of grey-scale images," IEE Proc., Comput. Digit. Tech., vol.151, no.3, pp.221–230, May 2004.
[12] D.H. Green, "Reed-Muller expansions of incompletely specified functions," IEE Proc., Comput. Digit. Tech., vol.134, no.5, pp.228–236, Sept. 1987.

[13] M.K. Habib, "A new approach to generate fixed-polarity Reed-Muller expansions for completely and incompletely specified functions," Int. J. Electron., vol.89, no.11, pp.845–876, Nov. 2002.

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

[15] K. Iravani and M.A. Perkowski, "Image compression based on Reed-Muller transforms," Proc. International Conference on Computational Intelligence and Multimedia Applications, pp.81–95, Feb. 1998.

[16] B.W. Kernighan and D.M. Ritchie, The C Programming Language, Second ed., Prentice-Hall, 1988.

[17] L. McKenzie, A.E.A. Almaini, J.F. Miller, and P. Thomson, "Optimisation of Reed-Muller logic functions," Int. J. Electron., vol.75, no.3, pp.451–466, Sept. 1993.

[18] U. Narayanan and C.L. Liu, "Low power logic synthesis for XOR based circuits," Proc. IEEE/ACM International Conference on Computer-Aided Design, pp.570–574, Nov. 1997.

[19] U. Narayanan, Algorithmic Techniques for Logic Synthesis of Low Power VLSI Circuits, Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1998.

[20] D. Popel, "Optimization of fixed polarity Reed-Muller representations of Boolean functions based on information theoretical approach," Proc. International Conference on Computer-Aided Design of Discrete Devices, pp.65–71, 1999.

[21] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," Proc. IEEE/ACM International Conference on Computer-Aided Design, pp.42–47, Nov. 1993.

[22] A. Sarabi and M.A. Perkowski, "Fast exact and quasi-minimal minimization of highly testable fixed polarity AND/XOR canonical networks," Proc. IEEE/ACM Design Automation Conference, pp.30–35, June 1992.

[23] T. Sasao, "AND-EXOR expressions and their optimization," in Logic Synthesis and Optimization, ed. T. Sasao, Kluwer Academic Publishers, 1993.

[24] T. Sasao, "Representations of logic functions using EXOR operators," in Representations of Discrete Functions, ed. T. Sasao and M. Fujita, Kluwer Academic Publishers, 1996.

[25] T. Sasao and F. Izuhara, "Exact minimization of FPRMs using multi-terminal EXOR TDDs," in Representations of Discrete Functions, ed. T. Sasao and M. Fujita, Kluwer Academic Publishers, 1996.

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

[27] T. Sasao, "Radix converters: Complexity and implementation by LUT cascades," Proc. IEEE 35th International Symposium on Multiple-Valued Logic, pp.256–263, May 2005.

[28] T. Sasao and M. Matsuura, "BDD representation for incompletely specified multiple-output logic functions and its applications to functional decomposition," Proc. IEEE/ACM Design Automation Conference, pp.373–378, June 2005.

[29] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," Technical Report, Memorandum No. UCB/ERL M92/41, University of California at Berkeley, May 1992.

[30] F. Somenzi, CUDD: CU Decision Diagram Package, Release 2.4.0, University of Colorado at Boulder, Feb. 2004.

[31] A. Tran, "Graphical method for the conversion of minterms to Reed-Muller coefficients and the minimization of exclusive-OR switching functions," IEE Proc., Comput. Digit. Tech., vol.134, no.2, pp.93–99, March 1987.

[32] C. Tsai and M. Marek-Sadowska, "Generalized Reed-Muller forms as a tool to detect symmetries," IEEE Trans. Comput., vol.45, no.1, pp.33–40, Jan. 1996.

[33] C. Tsai and M. Marek-Sadowska, "Logic synthesis for testability," Proc. IEEE/ACM Great Lakes Symposium on VLSI, pp.118–121, March 1996.

[34] C. Tsai and M. Marek-Sadowska, "Multilevel logic synthesis for arithmetic functions," Proc. IEEE/ACM Design Automation Conference, pp.242–247, June 1996.

[35] C. Tsai and M. Marek-Sadowska, "Boolean functions classification via fixed polarity Reed-Muller forms," IEEE Trans. Comput., vol.46, no.2, pp.173–186, Feb. 1997.

[36] D. Varma and E.A. Trachtenberg, "Computation of Reed-Muller expansions of incompletely specified Boolean functions from reduced representations," IEE Proc., Comput. Digit. Tech., vol.138, no.2, pp.85–92, March 1991.

[37] L. Wang and A.E.A. Almaini, "Exact minimisation of large multiple output FPRM functions," IEE Proc., Comput. Digit. Tech., vol.149, no.5, pp.203–212, Sept. 2002.

[38] A. Zakrevskij, "Minimizing polynomial implementation of weakly specified logic functions and systems," Proc. 3rd International Workshop on Applications of the Reed-Muller Expansion in Circuit Design, pp.157–166, Sept. 1997.

[39] I.I. Zhegalkin, "The technique of calculation of statements in symbolic logic," Mathe. Sbornik, vol.34, pp.9–28, 1927 (in Russian).

[40] Z. Zilic and Z.G. Vranesic, "A multiple-valued Reed-Muller transform for incompletely specified functions," IEEE Trans. Comput., vol.44, no.8, pp.1012–1020, Aug. 1995.

[41] Z. Zilic and Z.G. Vranesic, "Polynomial interpolation for Reed-Muller forms for incompletely specified functions," J. Multiple-Valued Logic, vol.2, pp.217–243, Aug. 1997.

Appendix A: Randomly Generated Functions

To generate the truth vectors for the test instances shown in Table 1 we used the C program shown in Fig. A· 1, the first two lines of which have been adapted from Kernighan and Ritchie [16, p.46]. The subroutine $f(n, t, d, s)$ returns a pointer v for the truth vector $[v[0], v[1], \ldots, v[2^n - 1]]$ for an n-variable switching function with t true and d unspecified minterms. The element $v[i]$ $(0 \leq i \leq 2^n - 1)$ of the truth vector represents a false, true, or unspecified minterm if it is 0, 1, or 2, respectively. The subroutine is based on a random number generator which should be provided with an initial non-zero seed through s. The truth vectors are generated on a 32-bit machine where `unsigned int` and `long` are 32 bits.

Appendix B: Code Converters

This section shows an outline of the code converters [27], [28] for which experimental results are presented in Table 3. In the following, subscripts are used to indicate the base of the number system. A base 10 should be assumed if a subscript is not shown. For example 1001_2 is a binary number

```
#define E(m) s = s * 1103515245 + 12345, \
    r = (unsigned int) (s >> 16) % w, m > 0
char *f(int n, int t, int d, unsigned long s)
{
    int r, w = 1 << n;
    char *v = (char *) calloc(w, 1);
    while (E(t)) if (!v[r]) v[r] = 1, t--;
    while (E(d)) if (!v[r]) v[r] = 2, d--;
    return v;
}
```

Fig. A· 1 Program to generate test instances used in Table 1.

that is equivalent to the decimal number 9 which is represented by 9_{10} or simply 9.

Two-digit BCD-to-binary converter. Binary-coded decimal (BCD) numbers use four bits to represent one digit. BCD numbers 0000_2, 0001_2, 0010_2, 0011_2, 0100_2, 0101_2, 0110_2, 0111_2, 1000_2, and 1001_2 represent 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, respectively. The remaining six 4-bit numbers, i.e., 1010_2, 1011_2, 1100_2, 1101_2, 1110_2, and 1111_2 are not used and can be considered as don't care. The two-digit BCD-to-binary converter has eight inputs. And two-digit BCD numbers can represent 100 different numbers, i.e., from 00_{10} to 99_{10}. Therefore, out of 256 possible input combinations that can be applied to the eight inputs of the two-digit BCD-to-binary converter, only 100 combinations are completely specified. The remaining 156 input combinations are don't care. Because the largest two-digit BCD number is 99, a two-digit BCD-to-binary converter has seven outputs.

Four-digit ternary-to-binary converter. In the binary-coded ternary representations, two bits are used to represent a ternary digit (a.k.a. a trit). In this representation 00_2 denotes 0, 01_2 denotes 1, 10_2 denotes 2, but 11_2 is not used and can be considered as don't care. Therefore, the four-digit ternary-to-binary converter has eight inputs. The smallest and the largest numbers that can be applied to the inputs of the four-digit ternary-to-binary converter are 0000_3 and 2222_3 ($= 2\times 3^3 + 2\times 3^2 + 2\times 3^1 + 2\times 3^0 = 80$), respectively. Therefore, out of 256 input combinations, 81 combinations are completely specified and the remaining 175 combinations are don't care. A four-digit ternary-to-binary converter has seven outputs because the largest number it requires to produce is 80.

Debatosh Debnath received the B.Sc.Eng. and M.Sc.Eng. degrees from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 1991 and 1993, respectively, and the Ph.D. degree from the Kyushu Institute of Technology, Iizuka, Japan, in 1998. He held research positions at the Kyushu Institute of Technology from 1998 to 1999 and at the University of Toronto, Ontario, Canada, from 1999 to 2002. In 2002, he joined the Department of Computer Science and Engineering at the Oakland University, Rochester, Michigan, as an Assistant Professor. His research interests include logic synthesis, design for testability, multiple-valued logic, and CAD for field-programmable devices. He was a recipient of the Japan Society for the Promotion of Science Postdoctoral Fellowship.

Tsutomu Sasao received the B.E., M.E., and Ph.D. degrees in electronics engineering from Osaka University, Osaka, Japan, in 1972, 1974, and 1977, respectively. He has held faculty/research positions at Osaka University, Japan, the IBM T.J. Watson Research Center, Yorktown Heights, New York, and the Naval Postgraduate School, Monterey, California. He is now a Professor of the Department of Computer Science and Electronics at the Kyushu Institute of Technology, Iizuka, Japan. His research areas include logic design and switching theory, representations of logic functions, and multiple-valued logic. He has published more than nine books on logic design, including Logic Synthesis and Optimization, Representation of Discrete Functions, Switching Theory for Logic Synthesis, and Logic Synthesis and Verification, Kluwer Academic Publishers, 1993, 1996, 1999, and 2001, respectively. He has served as Program Chairman for the IEEE International Symposium on Multiple-Valued Logic (ISMVL) many times. Also, he was the Symposium Chairman of the 28th ISMVL held in Fukuoka, Japan, in 1998. He received the NIWA Memorial Award in 1979, Distinctive Contribution Awards from the IEEE Computer Society MVL-TC for papers presented at ISMVLs in 1986, 1996, 2003 and 2004, and Takeda Techno-Entrepreneurship Award in 2001. He has served as an Associate Editor of the IEEE Transactions on Computers. He is a fellow of the IEEE.