1

EDGE VALUED BINARY DECISION DIAGRAMS

Sarma B.
K. Vrudhula and Massoud Pedram and Yung-Te Lai

Department of Electrical and Computer Engineering
University of Arizona
Tuscon, Arizona
and
Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California
and
Hitachi Labs
179 East Tasman Dr.
San Jose, California

Abstract— We describe a canonical and compact data structure, called Edge Valued Binary Decision Diagrams (EVBDD), for representing and manipulating pseudo Boolean functions (PBF). EVBDDs are particularly useful when both arithmetic and Boolean operations are required. We describe a general algorithm on EVBDDs for performing any binary operation that is closed over the integers. Next, we discuss the relation between the probability expression of a Boolean function and its representation as a pseudo Boolean function. Utilizing this, we present algorithms for computing the probability spectrum and the Reed-Muller spectrum of a Boolean function directly on the EVBDD. Finally, we describe an extension of EVBDDs which associates both an additive and a multiplicative weight with the true edges of the function graph.

1.1 INTRODUCTION

The development of efficient techniques for representing and manipulating Boolean functions lies at the heart of much of the recent advances in logic synthesis, verification and testing. Reduced, ordered binary decision diagrams (ROBDD) [3, 4, 5] and their numerous variants, provide such a representation, and thus have become ubiquitous in many problems that arise in computer-aided design of digital systems.

In many applications it is often convenient and more efficient to represent and manipulate Boolean functions at the *word* level or in the integer domain. For example, in attempting to verify data path components that perform various types of arithmetic operations, it is easier to specify the behavior in the integer domain, and transform the logic implementation into a similar representation. Other applications where integer level representations are preferred are spectral analysis of Boolean functions [16], multiple output decomposition [1, 14, 23, 26], etc. The advantages of integer level representations is not just limited to logic design. A large collection of discrete, or combinatorial optimization problems that involve binary or integer variables, but where the range of the objective function is some finite subset of integers, also require compact representations and algorithms for their manipulation [26].

In this chapter, we describe a canonical and compact representation of *pseudo* Boolean functions, called *Edge Valued Binary Decision Diagrams* (EVBDD) [21, 26]. EVBDDs can be used to efficiently represent and manipulate integer valued functions. Since Boolean functions are a subset of pseudo Boolean functions, EVBDDs can also be used to represent Boolean functions. This allows the construction of general procedures for performing both arithmetic and Boolean operations. Following a description of EVBDDs and their properties, we show how certain *spectral* transformations on Boolean functions can be carried out on EVBDDs. One particular transform, called the *probability spectrum* (PS) [19], will be examined. The PS of a Boolean function has important applications in testing of digital circuits [19]. Additionally, there is a direct relation between the PS and the *Reed-Muller* (RM) coefficients of a Boolean function [19]. We present an algorithm that transforms an EVBDD representation of a Boolean function to another EVBDD that represents its PS. Next, we show how the EVBDD representation of the PS can be transformed into another EVBDD that represents the RM spectrum. Finally, we show that by associating both an additive and a multiplicative weight with the edges of an EVBDD, we can develop a more compact representation of *pseudo* Boolean functions. The new data structure, which was proposed in [39] and is called *Factored Edge Valued Binary Decision Diagram*, reduces complexity of certain arithmetic operations and allows direct implementation of the complement edges.

1.2 PSEUDO BOOLEAN FUNCTIONS

Definition 1.2.1 *A pseudo Boolean function (PBF) $f(x_0, \ldots, x_{n-1})$, with $x_i \in \{0,1\}$, is a polynomial that is linear in each variable and whose coefficients are elements of the set of integers $\mathcal{I}$. The general representation of a PBF is of the form*

$$f(x_0, \ldots, x_{n-1}) = \sum_{\substack{(i_0,\cdots,i_{n-1}) \\ i_j \in \{0,1\}}} a_{i_0,\cdots,i_{n-1}} x_0^{i_0} x_1^{i_1} \cdots x_{n-1}^{i_{n-1}}, \; x_i \in \{0,1\}, \; a_{i_0,\cdots,i_{n-1}} \in \mathcal{I} \quad (1.2.1)$$

In (1.2.1), if $a_{0,0,\cdots,0} = 0$, then $f(x_0, \ldots, x_{n-1})$ are said to be *constant-free.*

With respect to a specific ordering $\langle x_0, x_1, \ldots, x_{n-1} \rangle$ of the variables, (1.2.1) can be expressed as

$$f(x_0, \ldots, x_{n-1}) = c + x_0(\alpha + f_\ell(x_1, \cdots, x_{n-1})) + (1 - x_0)(\beta + f_r(x_1, \cdots, x_{n-1})) \quad (1.2.2)$$

where $f_\ell(x_1, \cdots, x_{n-1})$ and $f_r(x_1, \cdots, x_{n-1})$ are constant-free, and $c, \alpha, \beta \in \mathcal{I}$.

For the sake of brevity, the right side of (1.2.2) will be indicated by a 6-tuple, $\langle c,\ x_{n-1},\ f_\ell,\ f_r,\ \alpha,\ \beta \rangle$. The representation given in (1.2.2) is analogous to the Shannon decomposition of a Boolean function. However, (1.2.2) is not in *standard* or *canonical* form. To see what this means, consider the function $f(x_0,\ x_1,\ x_2)\ = 3+2x_0-7x_0x_1-5x_0x_2+6x_0x_1x_2+3x_1-5x_1x_2$. Below are two different representations of this function in the form of (1.2.2).

$$
\begin{aligned}
f(x_0, x_1, x_2) &= \\
&1+x_0(4-4x_1-5x_2+x_1x_2)+(1-x_0)(2+3x_1-5x_1x_2) = \quad (1.2.3) \\
&8+x_0(-3-4x_1-5x_2+x_1x_2)+(1-x_0)(-5+3x_1-5x_1x_2)(1.2.4)
\end{aligned}
$$

In fact there are an infinite number of expressions of the form given in (1.2.2) that denote the same function. Therefore, given two 6-tuples, $\langle c_1,\ x_0,\ f_\ell,\ f_r,\ \alpha_1,\ \beta_1\rangle$, and $\langle\ c_2,\ x_0,\ g_\ell,\ g_r,\ \alpha_2,\ \beta_2\ \rangle$, one cannot determine whether or not these two representations denote the same function by simply checking the equality of the components. Among all the expressions of the form (1.2.2), we designate that one in which $\beta = 0$ to be the *standard* or *canonical* form. Then two such representations denote the same function if and only if the corresponding components of their canonical forms are equal.

1.3 EDGE VALUED BINARY DECISION DIAGRAMS

An Edge Valued Binary Decision Diagram (EVBDD) is a graph representation of a pseudo Boolean function, that is expressed in standard form, e.g., (1.2.2) with $\beta = 0$.

Definition 1.3.1 *An* EVBDD *is a tuple* $\langle c, \mathbf{f}\rangle$*, where* c *is a constant, and* $\mathbf{f}$ *is a directed acyclic graph consisting of two types of nodes:*

1. *A non-terminal node* $\mathbf{v}$ *described by a 4-tuple* $\langle var(v),\ child_\ell(\mathbf{v}),\ child_r(\mathbf{v}),\ value\rangle$*, where* $var(\mathbf{v}) \in \{x_0, \cdots, x_{n-1}\}$*,* $child_\ell(\mathbf{v})$ *and* $child_r(\mathbf{v})$ *are* EVBDD*s that represent the subexpressions* $f_\ell(x_1, \ldots, x_{n-1})$ *and* $f_r(x_1, \ldots, x_{n-1})$ *in (1.2.2) and* $value = \alpha$ *in (1.2.2).*

2. *The single terminal node, denoted by* $\mathbf{0}$*, which represents the value* 0*.*

Figure 1.3.1 EVBDD representation of the pseudo Boolean function shown in (1.2.2) ($\beta = 0$).

Definition 1.3.2 *An* EVBDD $\langle c, \mathbf{f} \rangle$ *denotes the arithmetic function* $c + f : \{0,1\}^n \rightarrow$ *integer where* f *is the function* f *denoted by* $\mathbf{f} = \langle x, \mathbf{f}_\ell, \mathbf{f}_r, value \rangle$*. The terminal node* $\mathbf{0}$ *represents the constant function* $f = 0$*, and* $\langle x, \mathbf{f}_\ell, \mathbf{f}_r, value \rangle$ *denotes the arithmetic function* $f = x \cdot (value + f_\ell) + (1 - x) \cdot f_r$*.*

Figure 1.3.1 shows a general EVBDD representation of (1.2.2).

1.3.1 Reduced and Ordered EVBDDs

The decomposition of a *PBF* given in (1.2.2) assumes a specific ordering of the variables. This means that there is an index function $index(x) \in \{0, \ldots, n-1\}$, such that for every non-terminal node $\mathbf{v}$, either $child_\ell(\mathbf{v})$ is a terminal node or $index(var(\mathbf{v})) < index(var(child_\ell(\mathbf{v})))$, and either $child_r(\mathbf{v})$ is a terminal node or $index(var(\mathbf{v})) < index(var(child_r(\mathbf{v})))$. If $\mathbf{v}$ is the terminal node, then $index(\mathbf{v}) = n$. Such an EVBDD is said to be *ordered.* An EVBDD is *reduced* if there is no non-terminal node such that $child_\ell(\mathbf{v}) = child_r(\mathbf{v})$ with $value = 0$, and there are no two nodes $\mathbf{u}$ and $\mathbf{v}$ such that $\mathbf{u} = \mathbf{v}$. We only consider *reduced, ordered* EVBDDs.

Figure 1.3.2 EVBDD of $f(x_0,\ x_1,\ x_2) = 3 + 2x_0 - 7x_0x_1 - 5x_0x_2 + 6x_0x_1x_2 + 3x_1 - 5x_1x_2$.

Example 1.3.1 *Consider the function $f(x_0, x_1, x_2)$ given in (1.2.4). With respect to the ordering $\langle x_0,\ x_1,\ x_2\rangle$, the function can be decomposed as follows:*

$$\begin{aligned} f(x_0,\ x_1,\ x_2) &= 3 + 2x_0 - 7x_0x_1 - 5x_0x_2 + 6x_0x_1x_2 + 3x_1 - 5x_1x_2 \\ &= 3 + x_0(2 - 4x_1 - 5x_2 + x_1x_2) + (1 - x_0)(3x_1 - 5x_1x_2). \end{aligned}$$

$$\begin{aligned} 2 - 4x_1 - 5x_2 + x_1x_2 &= 2 + x_1(-4 - 4x_2) + (1 - x_1)(-5x_2) \\ 3x_1 - 5x_1x_2 &= 0 + x_1(3 - 5x_2) + (1 - x_1)(0) \\ -4 - 4x_2 &= -4 + x_2(-4 + 0) + (1 - x_2)(0) \\ -5x_2 &= 0 + x_2(-5 + 0) + (1 - x_2)(0) \\ 3 - 5x_2 &= 3 + x_2(-5 + 0) + (1 - x_2)(0). \end{aligned}$$

Figure 1.3.2 shows the EVBDD *representation of $f(x_0,\ x_1,\ x_2)$.* □

A path in an EVBDD corresponds to an assignment of values to the variables associated with the path. The value of a *PBF* $f(x_0,\ \cdots,\ x_{n-1})$, for a given assignment $(x_0,\ \cdots,\ x_{n-1})$ is obtained by summing the values along the path as follows.

Definition 1.3.3 *Given an* EVBDD $\langle c, \mathbf{f} \rangle$ *representing* $f(x_0, \ldots, x_{n-1})$ *and a function* Φ *that for each variable* x *assigns a value* $\Phi(x)$ *equal to either 0 or 1, the function EVeval is defined as*

$$EVeval(\langle c, \mathbf{f} \rangle, \Phi) = \begin{cases} c & \mathbf{f} \textit{ is the terminal node } \mathbf{0} \\ EVeval(\langle c + value, child_\ell(\mathbf{f}) \rangle, \Phi) & \Phi(variable(\mathbf{f})) = 1 \\ EVeval(\langle c, child_r(\mathbf{f}) \rangle, \Phi) & \Phi(variable(\mathbf{f})) = 0 \end{cases}$$

An extension of OBDDs, called Multi-valued Decision Diagrams (MDDs), was given in [38]. In an MDD, a nonterminal node can have more than two children and a terminal node assumes integer values. All operations are carried out through the CASE operator, which although works for arbitrary discrete functions, cannot directly perform arithmetic operations. Recently, another extension to OBDDs, called Multi-Terminal Binary Decision Diagram (MTBDD) was presented in [6, 8]. An MTBDD corresponds to a fully *expanded* version of an EVBDD. In general, for functions where the number of distinct terminal values is large, an MTBDD will require larger number of nodes than an EVBDD. However, for functions where the number of distinct terminal values is small, an MTBDD may require less storage space depending on the number of nodes in the corresponding graphs.

An EVBDD requires $n + 1$ nodes to represent $2^{n-1}x_0 + \ldots + 2^0 x_{n-1}$ (an n-bit integer), while an MTBDD requires $2^{n+1} - 1$ nodes to represent the same function. When there are only two different terminal nodes (e.g., 0 and 1), EVBDDs, MTBDDs, and OBDDs are equivalent in terms of the number of nodes and the topology of the graph [26]. In this case, an EVBDD will require more space to represent the the edge-values. The worst case time complexity for performing operations on EVBDDs is the same as that for MTBDDs. However, many operations satisfy certain properties [26] that can be exploited so that EVBDDs are much more efficient than MTBDDs.

In [17] a useful calculus for manipulating pseudo-Boolean functions (referred by authors as **A**-Transforms) has been described. Also, to represent such pseudo-Boolean functions, a graph representation called Semi-Numeric Decision Diagrams (snDD) is introduced. In snDDs apart from standard BDD nodes, various operator nodes are also allowed; an operator node corresponds to basic arithmetic operations such as +, -, *. Also, numeric values are allowed to be stored graph terminals. Thus snDDs contain MTBDDs or ADDs.

1.3.2 Operations on EVBDDs

Let $\langle c_f,\ \mathbf{f}\rangle$, with $\mathbf{f} = \langle x,\ \mathbf{f}_\ell,\ \mathbf{f}_r,\ v_f\rangle$, and $\langle c_g,\ \mathbf{g}\rangle$, with $\mathbf{g} = \langle y,\ \mathbf{g}_\ell,\ \mathbf{g}_r,\ v_g\rangle$, be EVBDDs that represent two *PBF*s. We present a procedure, called *apply*, that computes $\langle c_h,\ \mathbf{h}\rangle = \langle c_f,\ \mathbf{f}\rangle\ op\ \langle c_g,\ \mathbf{g}\rangle$, where *op* is any binary operator that is closed over the integers. Examples of *op* include the integer arithmetic operators, such as $+$, $-$, and multiplication by a constant; relational operators, minimum and maximum, logical shift, and modulo c, for some constant c. Furthermore, if the EVBDD represents a Boolean function, then *apply* can be used directly to perform Boolean operations, as well.

Let $\mathbf{h} = \langle\ var(\mathbf{h}),\ \mathbf{h}_\ell,\ \mathbf{h}_r,\ v_h\rangle$, and consider the case where $index(x) < index(y)$. This means that y will appear below x in $\mathbf{h}$ and $var(\mathbf{h}) = x$. Now, when $x = 1$, $\langle c_f,\ \mathbf{f}\rangle_{x=1} = \langle c_f + v_f,\ \mathbf{f}_\ell\rangle$, and when $x = 0$, $\langle c_f,\ \mathbf{f}\rangle_{x=0} = \langle c_f,\ \mathbf{f}_r\rangle$. Therefore, $\langle c_h,\ \mathbf{h}\rangle_{x=1} = \langle c_{h_\ell},\ \mathbf{h}_\ell\rangle = \langle c_f + v_f,\ \mathbf{f}_\ell\rangle\ op\ \langle c_g,\ \mathbf{g}\rangle$, and $\langle c_h,\ \mathbf{h}\rangle_{x=0} = \langle c_{h_r},\ \mathbf{h}_r\rangle = \langle c_f,\ \mathbf{f}_r\rangle\ op\ \langle c_g,\ \mathbf{g}\rangle$. Once the *left* and *right* children of $\langle c_h,\ \mathbf{h}\rangle$ have been computed, they have to be combined and the result must be expressed in the canonical form. The step involved in this *standardization* is expressed as follows:

$$\begin{aligned} & x(c_{h_\ell} + h_\ell) + (1-x)(c_{h_r} + h_r) \\ & \quad = c_{h_r} + x(c_{h_\ell} - c_{h_r} + h_\ell) + (1-x)(h_r). \end{aligned}$$

Therefore, the result returned is

$$(c_h,\ \mathbf{h}) = (c_{h_r},\ \langle x,\ \mathbf{h}_\ell,\ \mathbf{h}_r,\ c_{h_\ell} - c_{h_r}\rangle).$$

Figure 1.3.3 shows this final step. A similar situation exists if $index(x) \geq index(y)$.

The steps involved in the procedure *apply* are shown in Figure 1.3.4. There are a set of *terminal* or default cases specific to each operator, the results of which

Figure 1.3.3 Combining the left and right children and converting to a canonical form.

can be returned without any further computation. As with ROBDD implementations, the efficient construction and manipulation of EVBDDs is made possible through the use of two tables that permit sharing of previously computed subgraphs without having to reconstruct them in the course of a computation. Thus, at any time only one EVBDD for each distinct function is ever maintained. The first table, called the *unique_table*, contains a unique entry for each EVBDD node. The second table, called the *computed_table* maintains the correspondence between a function that has been computed and the node in the *unique_table*. Thus in the course of a computation, *computed_table* is first checked, and if the result is there, it is returned. Otherwise, after the operation is performed, the node associated with the new EVBDD is added to *unique_table* and the function denoted by the node is entered into *computed_table*, along with a pointer to the node in *unique_table*. Additional enhancements such as the use of complement edges [26] are also possible.

Steps 4, 5, 13 and 14 of *apply* may generate new EVBDDs since equality of two EVBDDs $\langle c_f,\ \mathbf{f}\rangle$ and $\langle c_g,\ \mathbf{g}\rangle$ requires $c_f = c_g$ and $\mathbf{f}$ and $\mathbf{g}$ to be isomorphic. This implies that the time complexity of *apply* is not simply $O(|\langle c_f, \mathbf{f}\rangle| \cdot |\langle c_g, \mathbf{g}\rangle|)$, where $|\langle c_f, \mathbf{f}\rangle|$ and $|\langle c_g, \mathbf{g}\rangle|$ denote the number of nodes in EVBDDs $\langle c_f, \mathbf{f}\rangle$ and $\langle c_g, \mathbf{g}\rangle$. A complete analysis of the time complexity of *apply* is beyond the scope of this paper. Details appear in [26]. However, most of the operators in question satisfy certain properties which can be exploited to reduce the time

```
$apply(\langle c_f, \mathbf{f} \rangle, \langle c_g, \mathbf{g} \rangle, op)$
{
    if $(terminal\_case(\langle c_f, \mathbf{f} \rangle, \langle c_g, \mathbf{g} \rangle, op)$
        return$(\langle c_f, \mathbf{f} \rangle \ op \ \langle c_g, \mathbf{g} \rangle))$;
    if $(comp\_table\_lookup(\langle c_f, \mathbf{f} \rangle, \langle c_g, \mathbf{g} \rangle, op, ans))$
        return$(ans)$;
    if $(index(\mathbf{f}) \geq index(\mathbf{g}))$ {
        $\langle c_{g_l}, \mathbf{g_l} \rangle = \langle c_g + value(\mathbf{g}), child_l(\mathbf{g}) \rangle$;
        $\langle c_{g_r}, \mathbf{g_r} \rangle = \langle c_g, child_r(\mathbf{g}) \rangle$;
        $var = variable(\mathbf{g})$;
    }
    else {
        $\langle c_{g_l}, \mathbf{g_l} \rangle = \langle c_{g_r}, \mathbf{g_r} \rangle = \langle c_g, \mathbf{g} \rangle$;
        $var = variable(\mathbf{f})$;
    }
    if $(index(\mathbf{f}) \leq index(\mathbf{g}))$ {
        $\langle c_{f_l}, \mathbf{f_l} \rangle = \langle c_f + value(\mathbf{f}), child_l(\mathbf{f}) \rangle$;
        $\langle c_{f_r}, \mathbf{f_r} \rangle = \langle c_f, child_r(\mathbf{f}) \rangle$;
    }
    else { $\langle c_{f_l}, \mathbf{f_l} \rangle = \langle c_{f_r}, \mathbf{f_r} \rangle = \langle c_f, \mathbf{f} \rangle$;}
    $\langle c_{h_l}, \mathbf{h_l} \rangle = apply(\langle c_{f_l}, \mathbf{f_l} \rangle, \langle c_{g_l}, \mathbf{g_l} \rangle, op)$;
    $\langle c_{h_r}, \mathbf{h_r} \rangle = apply(\langle c_{f_r}, \mathbf{f_r} \rangle, \langle c_{g_r}, \mathbf{g_r} \rangle, op)$;
    if $(\langle c_{h_l}, \mathbf{h_l} \rangle == \langle c_{h_r}, \mathbf{h_r} \rangle)$ return $(\langle c_{h_l}, \mathbf{h_l} \rangle)$;
    $\mathbf{h} = find\_or\_add(var, \mathbf{h_l}, \mathbf{h_r}, c_{h_l} - c_{h_r})$;
    $comp\_table\_insert(\langle c_f, \mathbf{f} \rangle, \langle c_g, \mathbf{g} \rangle, op, \langle c_{h_r}, \mathbf{h} \rangle)$;
    return $(\langle c_{h_r}, \mathbf{h} \rangle)$;
}
```

Figure 1.3.4 Procedure apply.

complexity. As a result, the time complexity of binary arithmetic operations, Boolean operations, and relational operations is $O(|\langle c_f, \mathbf{f} \rangle| \cdot |\langle c_g, \mathbf{g} \rangle|)$. Thus, when Boolean functions are represented by EVBDDs, the time complexity is the same as in the case of ROBDDs.

1.3.3 Some Applications of EVBDDs

In this section we present a summary of some important applications of EVBDDs that have been in investigated [26].

Integer Linear Programming: Integer linear programming (ILP) is an NP-hard problem that appears in many applications. In [18] an OBBD based approach for solving 0-1 programming problems is presented. For operations involving integers such as conversion of linear inequality constraints into Boolean functions and optimization of non-binary objective functions, BDDs are not directly applicable. This shortcoming limits the caching of intermediate computations to only Boolean operations. Our approach to solving the ILP is to combine the benefits of EVBDDs (e.g., subgraph sharing and caching of intermediate results) with the state-of-art ILP techniques. In [26], we describe an algorithm using EVBDDs that computes optimal solution to a given linear objective function subject to linear constraints. Even without the use of sophisticated bounding techniques, experimental results show that the EVBDD based ILP solver is as efficient as a commercial ILP package.

Hierarchical Verification: The process of logic verification is to show the equivalence between the specification and the implementation. OBDDs can be used to verify logic circuits, only at the *Boolean level.* For example, to verify a 64-bit adder, one would have to first derive the OBDDs ($obdd_{imp}$) for each of the 65 outputs of the logic circuit and the OBDDs ($obdd_{spec}$) from the 65 Boolean expressions of each output function and then show equivalence. This would only show that each of the 65 outputs does realize the corresponding Boolean expression. However, since EVBDDs can be used to represent discrete integer and Boolean functions, they provide a means to verify the a logic circuit, where the specification is expressed in the integer domain. Thus the specification of a 65-bit adder would simply be $x + y + c$, where c is the carry-in. If $(b_0, b_1, \ldots, b_{n-1})$ represent the outputs of a 65-bit adder, then the verification process would first construct 65 EVBDDs for the b_i's and construct a single EVBDD for $2^0 \times b_0 + \ldots + 2^{64} \times b_{64}$. This EVBDD would be compared with the EVBDD of the specification, thus showing that the adder does indeed perform *addition*. Details of the use of EVBDDs for verification are described in [21].

Decomposition of Multiple Output Boolean Functions: The decomposition of Boolean functions had been the subject of extensive research. Much of the classical work on decomposition based on Karnaugh maps [1, 9] and cubes [34], and more recently using OBDDs [2, 7, 23, 36], are applicable to single Boolean functions. EVBDDs allow decomposition of multiple Boolean functions. In [26] a set of EVBDD based algorithms for decomposition of single and multiple output functions are described. This includes disjunctive and non-disjunctive decomposition, and decomposition of completely and incompletely specified functions.

1.4 THE PROBABILITY TRANSFORM AND ITS SPECTRUM

Many properties of Boolean functions that are difficult to deduce in the Boolean domain are often very easy to establish using an alternate representation. The alternate representation is called the *spectral* domain. A spectral transformation of a Boolean function of n variables is typically represented in the following form:

$$T_n \widetilde{Z}_n = \widetilde{R}_n, \tag{1.4.1}$$

where $\widetilde{Z}_n$ is a $2^n \times 1$ vector representing the truth table of the function, and T_n is a $2^n \times 2^n$ transformation matrix. $\widetilde{R}_n$ is a $2^n \times 1$ vector, and is called the *spectrum* of the Boolean function. Different transformation matrices generate different spectra. Some of the more extensively studied transformations are the Hadamard, Walsh and Reed-Muller [16, 33, 29]. These transforms have found extensive use in function classification, verification, logic synthesis, testing and fault diagnosis.

The key obstacle in using (1.4.1) is that the transformation matrix is of size $2^n \times 2^n$. Thus (1.4.1) can be used in practice only when n is small. This is where EVBDDs play an important role. In situations where T_n has a recursive structure, the transformation given in (1.4.1) can be carried out directly on the EVBDD representation of the Boolean function.

In this section, we examine one particular transform, called the *probability* transform , which is also known as the *algebraic* transform . The fundamental properties of this transform and its extensive applications to testing were investigated by Kumar [19]. However, many of the results presented in [19] require computations whose complexity is exponential in the number of variables. The probability spectrum of a Boolean function is directly related to the pseudo Boolean representation of the function. Using this relation, and the fact that EVBDDs provide a canonical and compact representation of *PBF*s, we present an algorithm to compute the probability spectrum directly on the EVBDD. The resulting structure is called a *spectral* EVBDD. The only difference between an EVBDD and the corresponding spectral EVBDD is in the interpretation - the sum of the values along the edges of a path in a spectral EVBDD is the spectral coefficient associated with the input assignment. After showing how the probability spectrum can be computed using EVBDDs, we show how the spectral EVBDD can be transformed, so that the resulting structure represents the Reed-Muller spectrum of a Boolean function. The resulting structure is called an RMEVBDD.

Let $f(x_0, \cdots, x_{n-1})$ be a Boolean function. Let $X_i = Pr(x_i = 1)$, with $X_i \in [0,1]$, and $F(X_0, \cdots, X_{n-1}) = Pr(f(x_0, \cdots, x_{n-1}) = 1)$. $F(X_0, \cdots, X_{n-1})$ is called the *probability expression* of $f(x_0, \cdots, x_{n-1})$. The probability expression $F(X_0, \cdots, X_{n-1})$ is obtained by the repeated application of the following rules [19, 31]:

$$\begin{aligned} Pr(\overline{x_i} = 1) &= 1 - X_i \\ Pr(x_i \wedge x_i = 1) &= X_i \\ Pr(f_1 \vee f_2 = 1) &= F_1 + F_2 - F_1F_2, \end{aligned}$$

where $F_i = Pr(f_i = 1)$.

The probability expression $F(X_0, \cdots, X_{n-1})$ represents a real valued function whose domain and range are $[0,1]^n$ and $[0,1]$, respectively. Now, if the variables X_i are restricted to be *integers* that can assume either 0 or 1, (i.e., $X_i \in \{0,1\}$), then the expression $F(X_0, \cdots, X_{n-1})$ is the pseudo Boolean representation of $f(x_0, \cdots, x_{n-1})$.

Example 1.4.1 *Consider* $f(x_0, x_1, x_2) = \overline{x_0}x_1 \vee x_1x_2$. *Applying the above rules, we obtain*

$$F(X_0, X_1, X_2) = X_0X_1X_2 - X_0X_1 + X_1. \qquad (1.4.2)$$

Now the pseudo Boolean representation of $f(x_0, x_1, x_2)$ is the same as the right side of (1.4.2), with $X_i \in \{0,1\}$. $\square$

Definition 1.4.1 *(from [19]) With respect to the ordering, $\langle X_0, \cdots, X_{n-1}\rangle$, the probability* spectrum *of a Boolean function $f(x_0, \cdots, x_{n-1})$ is a vector $\widetilde{S}_f = [s_0, s_1, \cdots, s_{2^n-1}]^T$, where s_i is the coefficient of the term $X_0^{i_0} X_1^{i_1} \cdots X_{n-1}^{i_{n-1}}$ in $F(X_0, \cdots, X_{n-1})$, and $\langle i_0 i_1 \ldots i_{n-1}\rangle$ is the binary representation of the integer i.*

Example 1.4.2 *Consider the probability expression given in (1.4.2). Expanding the expression into a canonical sum of products form, we obtain*

$$\begin{aligned} F(X_0, X_1, X_2) &= 0 \cdot X_0^0 X_1^0 X_2^0 + 0 \cdot X_0^0 X_1^0 X_2^1 + 1 \cdot X_0^0 X_1^1 X_2^0 + 0 \cdot X_0^0 X_1^1 X_2^1 \\ &+ 0 \cdot X_0^1 X_1^0 X_2^0 + 0 \cdot X_0^1 X_1^0 X_2^1 + (-1) \cdot X_0^1 X_1^1 X_2^0 + 1 \cdot X_0^1 X_1^1 X_2^1. \end{aligned}$$

The probability spectrum $\widetilde{S}_f = [0\ 0\ 1\ 0\ 0\ 0\ -1\ 1]^T$ $\square$

Theorem 1.4.1 *(from [19]) Define a $2^n \times 2^n$ matrix P_n as follows:*

$$P_0 = 1, \qquad P_n = \begin{bmatrix} P_{n-1} & 0 \\ -P_{n-1} & P_{n-1} \end{bmatrix}.$$

Let $\widetilde{Z}$ be a $2^n \times 1$ vector of values of $f(x_0, \cdots, x_{n-1})$. Then the probability spectrum $\widetilde{S}_f$ is given by

$$\widetilde{S}_f = P_n \times \widetilde{Z}. \tag{1.4.3}$$

$\square$

We now show how the transformation given in (1.4.3) can be carried out on an EVBDD representation of f. With respect to the ordering $\langle x_0, \cdots, x_{n-1}\rangle$, $\widetilde{Z}_n$ can be partitioned as $[\widetilde{Z}_{n-1}^0, \widetilde{Z}_{n-1}^1]^T$, where $\widetilde{Z}_{n-1}^0$ ($\widetilde{Z}_{n-1}^1$) corresponds to the min-terms of f with $x_0 = 0$ ($x_0 = 1$). Then (1.4.3) can be expressed as

$$\begin{bmatrix} \widetilde{S}_f^0 \\ \widetilde{S}_f^1 \end{bmatrix} = \begin{bmatrix} P_{n-1} & 0 \\ -P_{n-1} & P_{n-1} \end{bmatrix} \begin{bmatrix} \widetilde{Z}_{n-1}^0 \\ \widetilde{Z}_{n-1}^1 \end{bmatrix} = \begin{bmatrix} P_{n-1} \times \widetilde{Z}_{n-1}^0 \\ P_{n-1} \times \widetilde{Z}_{n-1}^1 - P_{n-1} \times \widetilde{Z}_{n-1}^0 \end{bmatrix}. \tag{1.4.4}$$

In (1.4.4), $\widetilde{Z}^0_{n-1}$ and $\widetilde{S}^0_f$ ($\widetilde{Z}^1_{n-1}$ and $\widetilde{S}^1_f$) correspond to the *right* (*left*) children of the EVBDD of $f(x_0, \cdots, x_{n-1})$, and the resulting spectral EVBDD, respectively. Thus, (1.4.4) states that to compute the spectral EVBDD we perform the following steps recursively:

1. replace the *right child* with the spectral EVBDD of the *right child*,
2. replace the *left child* with the spectral EVBDD of the *left child* minus the spectral EVBDD of the *right child*.

Note: Since an EVBDD is reduced, a node of the form $v = \langle x, \mathbf{f}, \mathbf{f}, 0\rangle$ would never appear. However, the spectral transform of the subgraph rooted at v would result in a node $v' = \langle x, \mathbf{0}, \mathbf{f}', 0\rangle$, where $\mathbf{f}'$ is the result of spectral transform when applied to the right child of v. For this reason, the algorithm for computing the spectral transform must keep track of the level of each node. If the index of the variable associated with the node currently being visited is greater than its level, then an a new EVBDD, rooted at a node of the form v' is returned. The basic steps of the procedure that converts an EVBDD of a Boolean function to a spectral EVBDD is shown in Figure 1.4.1.

Example 1.4.3 *Consider the Boolean function* $f(x_0, x_1, x_2) = \overline{x_0}x_1 \vee x_1x_2$ *of Example 1.4.1. Its representation as a* PBF *is given in Equation 1.4.2. Using this, its* EVBDD *representation is shown Figure 1.4.2. The execution of the procedure evbdd_to_spevbdd on the* EVBDD *of Figure 1.4.2 results in the spectral* EVBDD *shown in Figure 1.4.3. Summing the values along the edges of the spectral* EVBDD *will result in the spectral coefficients shown in Example 1.4.2.* □

```
$evbdd\_to\_spevbdd(ev, level, n)$
{
    if ($level == n$) return $ev$;
    if ($ev == 0$) return 0;
    if ($index(ev) > level$) {
        $right = evbdd\_to\_spevbdd(ev, level + 1, n)$;
        $left = 0$;
        return $new\_evbdd(level, left, right)$;
    }
    $right = evbdd\_to\_spevbdd(child_r(ev), level + 1, n)$;
    $sp_l = evbdd\_to\_spevbdd(child_l(ev), level + 1, n)$;
    $left = evbdd\_sub(sp_l, right)$;
    return $new\_evbdd(level, left, right)$;
}
```

Figure 1.4.1 Procedure to convert an EVBDD to a spectral EVBDD.

Figure 1.4.2 EVBDD of $f(x_0, x_1, x_2) = \overline{x_0}x_1 \vee x_1x_2$.

1.5 REED-MULLER COEFFICIENTS

The standard symbolic interpretation of an OBDD results in a sum of products representation of a Boolean function. It is well known [29, 33, 41], that another canonical representation is possible if only *AND* and *XOR* operators are used. The resulting expression is called the Reed-Muller (RM) representation of the

Figure 1.4.3 Spectral EVBDD of the EVBDD shown in Figure 1.4.2.

function. Specifically, if $f(x_0, \cdots, x_{n-1})$ is a Boolean function, then the RM representation of f has the form

$$f(x_0, \cdots, x_{n-1}) = a_{-1} \oplus a_0 \tilde{x}_{n-1} \oplus a_1 \tilde{x}_{n-2} \oplus \cdots \oplus a_{2^n-1} \tilde{x}_0 \tilde{x}_1 \cdots \tilde{x}_{n-1}, \quad (1.5.1)$$

where $a_i \in \{0, 1\}$, and $\tilde{x}$ is either complemented or un-complemented variable x_i.

We assume that each variable on the right side of (1.5.1) appears either as complemented or un-complemented, but not both. Digital circuits built using the RM form have a number of useful properties. They belong to a class of easily testable networks since test sets are independent of the function realized by the circuit and depend linearly on the number of inputs [32]. This has motivated interest in minimization of RM forms. Minimizing an RM form requires determining the polarity of each variable so that the resulting RM expression for the function has the least number terms.

The RM coefficients of a Boolean function, represented by an EVBDD can be obtained using a transformation nearly identical to the one given in Theorem 1.4.1. The resulting representation will be called an RMEVBDD. Let $\widetilde{A}_f$ denote the RM coefficients of a Boolean function f, ordered in the same way as $\widetilde{S}_f$

(see Theorem 1.4.1). Define a $2^n \times 2^n$ matrix $\mathcal{R}_n$ as follows:

$$\mathcal{R}_0 = 1, \qquad \mathcal{R}_n = \begin{bmatrix} \mathcal{R}_{n-1} & 0 \\ \mathcal{R}_{n-1} & \mathcal{R}_{n-1} \end{bmatrix}.$$

Then the RM spectrum $\widetilde{A}_f$ is given by

$$\widetilde{A}_f = \mathcal{R}_n \times \widetilde{Z}_n. \tag{1.5.2}$$

The RM spectrum can also be computed from the probability spectrum [19]. The relation is simply

$$\widetilde{A}_f = |\widetilde{S}_f| \; mod \; 2. \tag{1.5.3}$$

Thus, the RMEVBDD can be generated using either (1.5.2) or (1.5.3). Thus, to transform an EVBDD of a Boolean function to an RMEVBDD, line 10 of the procedure *evbdd_to_spevbdd* shown in Figure 1.4.1 has to be modified as follows:

```
10      $left = evbdd\_add(sp_l, right)$;
```

Example 1.5.1 *Figure 1.5.1 shows the* RMEVBDD *of the function* $f(x_0, x_1, x_2) = \overline{x_0}x_1 \vee x_1x_2$ *of Example 1.4.1. It was derived from the* EVBDD *shown in Figure 1.4.2. In an* RMEVBDD*, a path from the root to the terminal node that involves a* right *edge of a node* v*, such that* $var(v) = x_i$*, corresponds to a product term in which* x_i *is not present in the RM representation. For example, consider the path corresponding to the assignment* $x_0 = 0, x_1 = 1, x_2 = 1$*, in the* RMEVBDD *shown in Figure 1.5.1. This assignment corresponds to a path that includes the right edge of a node* v*, with* $var(v) = x_0$*. Therefore, this path corresponds to the term* x_1x_2*. The associated RM coefficient is the sum of the edge values along this path, i.e.,* $0 + 0 + 1 + -1 = 0$. □

There are 2^n RM forms for a function $f(x_0, \cdots, x_{n-1})$, corresponding to the 2^n possible combinations of polarities for each variable. From (1.5.3), we see

Figure 1.5.1 RMEVBDD of the EVBDD shown in Figure 1.4.2.

that finding the minimal RM expression of a function requires identifying the polarity vector that will result in the least number of odd terms [19]. The following theorem shows the effect of changing the polarity of a variable in the RMEVBDD representation of a Boolean function.

Theorem 1.5.1 *Let E and E' represent the* RMEVBDD*s of $f(x_0, \cdots, x_{i-1}, x_i, x_{i+1}, \cdots, x_{n-1})$ and $f(x_0, \cdots, x_{i-1}, \overline{x}_i, x_{i+1}, \cdots, x_{n-1})$, respectively. Then E' is obtained from E by performing the following transformation on every node u, such that $var(u) = x_i$.*

$$child_r(u) \leftarrow (child_\ell(u) + child_r(u)) \ mod\ 2.$$

Figure 1.5.2 shows the procedure to switch the polarity of a variable in an RMEVBDD.

```
/* $swith\_polarity$ switches the polarity of
a variable in an RMEVBDD.
$ev$ is an RMEVBDD. $i$ is the index of the
variable whose polarity is to be switched.
/
$switch\_polarity(ev, level, i, n)$
{
    if $(level == n)$ return $ev$;
    if $(ev == 0)$ return 0;
    if $(index(ev) > level)$ {
        $right = 0$;
        return $new\_evbdd(level, child_l(ev), right)$;
    }
    if $(index(ev) == i)$ {
        $sp_r = evbdd\_add(child_l(ev), child_r(ev))$;
        $right = mod2(sp_r)$;
        return $new\_evbdd(level, child_l(ev), right)$;
    }
    $left = switch\_polarity(child_l(ev), level + 1, i, n)$;
    $right = switch\_polarity(child_r(ev), level + 1, i, n)$;
    return $new\_evbdd(level, left, right)$;
}
```

Figure 1.5.2 Procedure to switch the polarity of a variable in an RMEVBDD.

1.6 FACTORED EDGE VALUED BINARY DECISION DIAGRAMS

Factored Edge Valued Binary Decision Diagrams (FEVBDD) are an extension to EVBDDs which were presented in [39]. By associating both an additive and a multiplicative weight with the true-edges, FEVBDDs offer a more compact representation of linear functions, since common sub-functions differing only by an *affine* transformation can now be expressed by a single subgraph. Additionally, they allow the notion of complement edges to be transferred from OBDDs to FEVBDDs.

Definition 1.6.1 *An* FEVBDD *is a tuple* $\langle c, w, \mathbf{f}\rangle$*, where* c *and* w *are constant values, and* $\mathbf{f}$ *is a rooted, directed acyclic graph consisting of two types of vertices:* [1]

1. *A non-terminal vertex* $\mathbf{v}$ *is represented by a 5-tuple* $\langle var(\mathbf{v}), child_\ell(\mathbf{v}), child_r(\mathbf{v}), value, factor\rangle$ *where value and factor are rational numbers and* $var(\mathbf{v}) \in \{x_0, \ldots, x_{n-1}\}$*.*
2. *The single terminal vertex* $\mathbf{v}$ *with value 0 is denoted by* $\mathbf{0}$*. By definition all branches leading to* $\mathbf{0}$ *have an associated weight* $w = 0$*.*

There is no non-terminal vertex $\mathbf{v}$ *such that* $child_\ell(\mathbf{v}) = child_r(\mathbf{v})$*,* $value = 0$*, and* $factor = 1$*, and there are no two nonterminal vertices* $\mathbf{v}$ *and* $\mathbf{u}$ *such that* $\mathbf{v} = \mathbf{u}$*. Furthermore, there exists an index function* $index(x) \in \{0, \ldots, n-1\}$ *such that the following holds for every nonterminal vertex. If* $child_\ell(\mathbf{v})$ *is also nonterminal, then we must have* $index(var(\mathbf{v})) < index(var(child_\ell(\mathbf{v})))$*. If* $child_r(\mathbf{v})$ *is nonterminal, then we must have* $index(var(\mathbf{v})) < index(var(child_r(\mathbf{v})))$*.*

Definition 1.6.2 *A* FEVBDD $\langle c_f, w_f, \mathbf{f}\rangle$ *denotes the arithmetic function* $c_f + w_f \cdot f$ *where* f *is the function* f *denoted by* $\mathbf{f} = \langle x, \mathbf{f}_\ell, \mathbf{f}_r, value, factor\rangle$*. The ter-*

[1] Here, we will only describe the "rational rule" for weight normalization which requires the use of fractions. See [39] for the "GCD rule" which requires a multiplicative weight to be associated with both edges.

minal node **0** *represents the constant function* $f = 0$, *and* $\langle x, \mathbf{f}_\ell, \mathbf{f}_r, value, factor\rangle$ *denotes the arithmetic function* $f = x \cdot (value + factor \cdot f_\ell) + (1 - x) \cdot f_r$.

As an example, we construct the various function graphs based on the different decompositions of function f given in its tabular form in Figure 1.6.1.

$$\begin{aligned} f(x,y,z) &= 15(1-x)(1-y)(1-z) + 6(1-x)(1-y)z + \qquad (1.6.1)\\ &\quad 5(1-x)y(1-z) + 2(1-x)yz + \\ &\quad 13x(1-y)(1-z) + 7x(1-y)z + \\ &\quad 5xy(1-z) + 2xyz \\ &= 15 + x(-2 + y(-8 + z(-3)) + (1-y)(z(-6))) + \qquad (1.6.2)\\ &\quad (1-x)(y(-10 + z(-3)) + (1-y)(z(-9))) \\ &= 15 - 9(x(\frac{2}{9} + \frac{2}{3}(y(\frac{4}{3} + \frac{1}{2}z) + (1-y)z)) + \qquad (1.6.3)\\ &\quad (1-x)(y(\frac{10}{9} + \frac{1}{3}z) + (1-y)z)). \end{aligned}$$

Equation (1.6.1) is in a form that directly corresponds to the function decomposition for MTBDDs or ADDs and the tabular form. Equations (1.6.2) and (1.6.3) reflect the structure of the decomposition rules for EVBDDs and FEVBDDs, respectively. The different function graphs are shown in Figure 1.6.1.

A path in an FEVBDD corresponds to an assignment of values to the variables associated with the path. The value of a *PBF* $f(x_0, \cdots, x_{n-1})$, for a given assignment $(x_0, \cdots, x_{n-1})$ is obtained as follows.

Definition 1.6.3 *Given a* FEVBDD $\langle c_f, w_f, \mathbf{f}\rangle$ *representing* $f(x_0, \ldots, x_{n-1})$ *and a function* Φ *that for each variable* x *assigns a value* $\Phi(x)$ *equal to either 0 or 1, the function FEVeval is defined as:*

$$FEVeval(\langle c_f, w_f, \mathbf{f}\rangle, \Phi) =$$

$$\begin{cases} c_f & \mathbf{f} \textit{ is the terminal node } \mathbf{0} \\ c_f + w_f \cdot FEVeval(\langle value, factor, child_\ell(\mathbf{f})\rangle, \Phi) & \Phi(variable(\mathbf{f})) = 1 \\ c_f + w_f \cdot FEVeval(\langle 0, 1, child_r(\mathbf{f})\rangle, \Phi) & \Phi(variable(\mathbf{f})) = 0 \end{cases}$$

Figure 1.6.1 Example of various function graphs.

Figure 1.6.2 FEVBDD representation of the four output functions of a 3-bit adder.

An example of a FEVBDD representing a Boolean function with complement edges is given in Figure 1.6.2. This FEVBDD represents the four output functions of a 3-bit adder. It has the same topology (except for the terminal edges) as the corresponding OBDD depicted in the same figure. As it is shown in this example, FEVBDDs successfully extend the use of EVBDDs to represent Boolean functions as they inherently offer a way to represent complement edges. Furthermore, the Boolean operation '*not*' can now be performed in constant time since it only requires manipulation of the weights of the root node.

A FEVBDD-based matrix package was introduced in [39]. This package was used to solve the Chapman-Kolmogorov equations. Experimental results showed that in the majority of cases FEVBDDs win over the corresponding EVBDD representation of the matrices in terms of number of nodes and overall memory consumption. In general, however, since the memory consumption per node of the FEVBDD is higher than that of the EVBDD, if the number of nodes in the FEVBDD and EVBDD are the same (for example, due to the sparse structure of matrices), then EVBDDs will require less memory.

1.7 SUMMARY

Edge Valued Binary Decision Diagrams (EVBDD) are a novel data structure for representing discrete functions. EVBDDs generalize ROBDDs and are particularly useful for performing both integer and Boolean operations. This extension to the *word* level provides a basis for a variety of applications, including verification (where the specification can be expressed in the arithmetic domain), multiple output decomposition of logic functions, discrete function optimization and others. In this chapter we showed how *evbdd*s can be used to efficiently represent and manipulate various types of spectra of Boolean functions. In particular, we showed how the probability spectrum and the Reed-Muller spectrum of a Boolean function can be computed directly on an EVBDD without having to resort to traditional methods that require matrices of exponential size. The relation between the probability spectrum and the Reed-Muller coefficients of a Boolean function was utilized to develop an algorithm for constructing the EVBDD that represents the Reed-Muller form when the polarity of one or more variables is changed. The current direction of this work is to utilize this algorithm to develop efficient heuristics for generating a near minimal Reed-Muller form of a Boolean function. In addition, we showed an extension of EVBDDs which associates both an additive and a multiplicative weight with the true edges of the function graph in order to achieve more compact representations, and in some cases, more efficient operations.

REFERENCES

[1] R. L. Ashenhurst, "The decomposition of switching functions," *Ann. Computation Lab.*, Harvard University, vol. 29, pp. 74-116, 1959.

[2] M. Beardslee, B. Lin and A. Sangiovanni-Vincentelli, "Communication based logic partitioning," *Proc. of the European Design Automation Conference*, pp. 32-37, 1992.

[3] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," *Proc. of the 27th Design Automation Conference*, pp. 40-45, 1990.

[4] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, C-35(8): 677-691, August 1986.

[5] R. E. Bryant, "Symbolic Boolean manipulation with ordered binary-decision diagrams," *Computing Surveys*, Vol. 24, No. 3, pp. 293-318, Sept. 1992.

[6] E. M. Clarke, M. Fujita, P. C. McGeer, K. L. McMillan, and J. C.-Y. Yang, "Multi-terminal binary decision diagrams: An efficient data structure for matrix representation," *International Workshop on Logic Synthesis*, pp. 6a:1-15, May 1993.

[7] S-C. Chang and M. Marek-Sadowska, "Technology mapping via transformations of function graphs," *Proc. International Conference on Computer Design*, pp. 159-162, 1992.

[8] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. C.-Y. Yang, "Spectral transforms for large Boolean functions with applications to technology mapping," *Proc. of the 30th Design Automation Conference*, pp. 54-60, 1993.

[9] H. A. Curtis, *A New Approach to the Design of Switching Circuits*, Von Nostrand, Princeton, N.J., 1962.

[10] C. R. Edwards, "The application of the Rademacher-Walsh transform to Boolean function classification and threshold-logic synthesis," *IEEE Transactions on Computers*, C-24, pp. 48-62, 1975.

[11] B. J. Falkowski, I. Schafer and M. A. Perkowski, "Effective computer methods for the calculation of Rademacher-Walsh spectrum for completely and incompletely specified Boolean functions," *IEEE Transaction on Computer-Aided Design*, Vol. 11, No. 10, pp. 1207-1226, Oct. 1992.

[12] P. L. Hammer and S. Rudeanu, *Boolean Methods in Operations Research and Related Areas*, Heidelberg, Springer Verlag, 1968.

[13] P. L. Hammer and B. Simeone, "Order relations of variables in 0-1 programming," *Annals of Discrete Mathematics*, 31, North-Holland, pp. 83-112, 1987.

[14] S. He and M. Torkelson, "Disjoint decomposition with partial vertex chart," *International Workshop on Logic Synthesis*, pp. p2a 1-5, 1993.

[15] M. Helliwell and M. Perkowski, "A fast algorithm to minimize multi-output mixed-polarity generalized Reed-Muller forms," *Proc. 25th Design Automation Conf.*, pp. 427-432, 1988.

[16] S. L. Hurst, D. M. Miller and J. C. Muzio, *Spectral Techniques in Digital Logic,* London, U.K. : Academic, 1985.

[17] J. Jain, J. A. Abraham, J. Bitner, and D. S. Fussel, "Probabilistic verification of boolean functions," *Formal Methods in Systems Design*, 1(1), pp. 63-115, July 1992.

[18] S-W. Jeong and F. Somenzi, "A new algorithm for 0-1 programming based on binary decision diagrams," Logic Synthesis and Optimization, Sasao ed., Kluwer Academic Publishers, pp. 145-165, 1993.

[19] S. K. Kumar, *Theoretical Aspects of the Behavior of Digital Circuits Under Random Inputs.* Technical Report DISC/81-3, Dept. of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA., Sept. 1981.

[20] Y-T. Lai and S. Sastry, "HINTS: A hardware interpretation system," International Workshop on Formal Methods in VLSI Design 1991.

[21] Y-T. Lai and S. Sastry, "Edge-valued binary decision diagrams for multi-level hierarchical verification," *Proc. of 29th Design Automation Conf.*, pp. 608-613, 1992.

[22] Y-T. Lai, S. Sastry and M. Pedram, "Boolean matching using binary decision diagrams with applications to logic synthesis and verification," *Proc. International Conf. on Computer Design*, pp.452-458, 1992.

[23] Y-T. Lai, M. Pedram and S. Sastry, "BDD based decomposition of logic functions with application to FPGA synthesis," *Proc. of 30th Design Automation Conf*, pp. 642-647, 1993.

[24] Y-T. Lai, K-R. Pan, M. Pedram, and S. Vrudhula, "FGMap: A technology mapping algorithm for Look-Up Table type FPGAs based on function graphs," *International Workshop on Logic Synthesis*, 1993.

[25] Y-T. Lai, M. Pedram and S. B. K. Vrudhula, "FGILP: An integer linear program solver based on function graphs," *Proc. Int'l Conf. on Computer Aided Design,* November 1993, pages 685-689.

[26] Y-T. Lai, M. Pedram, and S. B. Vrudhula, "EVBDD-based Algorithms for linear integer programming, spectral transformation and function decomposition," *IEEE Transactions on CAD*, vol. 13, no. 8, pp. 959-975, Aug. 1994.

[27] Y-T. Lai, M. Pedram, and S. B. Vrudhula, "Formal verification using edge-valued binary decision diagrams," to appear in *IEEE Trans. on Computers,* 1996.

[28] H-T. Liaw and C-S Lin, "On the OBDD-representation of general Boolean functions," *IEEE Trans. on Computers*, C-41(6): 661-664, June 1992.

[29] D. E. Muller, "Application of Boolean algebra to swithcing circuit design and error detection," *IEEE Trans. on Computers*, vol. C-21, pp. 6-12, 1974.

[30] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*, Wiley, New York, 1988.

[31] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Transactions on Computers*, vol. C-24, pp. 668-670, June 1975.

[32] S. M. Reddy, "Easily testable realizations for logic functions," *IEEE Trans. on Computers*, vol. C-21, pp. 1183-1188, 1972.

[33] I. S. Reed, "A class of multiple-error-correcting codes and their decoding scheme," *IRE Trans.*, vol. IT-4, pp. 38-49, 1954.

[34] J. P. Roth and R. M. Karp, "Minimization over Boolean graphs," *IBM Journal.*, pp. 227-238, April 1962.

[35] A. Sarabi and M. Perkowski, "Fast exact and quasi-minimal minimization of highly testable fixed polarity AND/XOR canonical networks," *Proc. 29th Design Automation Conf.*, pp. 30-35, 1992.

[36] T. Sasao, "FPGA design by generalized functional decomposition," *Logic Synthesis and Optimization*, Sasao ed., Kluwer Academic Publishers, pp. 233-258, 1993.

[37] D. Singh, J. Rabaey, M. Pedram, F. Catthoor, S. Rajgopal, N. Sehgal and T. Mozdzen. "Power-conscious CAD tools and methodologies: a perspective," *Proc. of the IEEE,* April 1995.

[38] A. Srinivasan, T. Kam, S. Malik and R. Brayton, "Algorithms for Discrete Function Manipulation," *Proc. Int. Conf. CAD*, pp. 92-95, 1990.

[39] P. Tafertshofer and M. Pedram, *Factored EVBDDs and Their Application to Matrix Representation and Manipulation,* CENG Technical Report 94-27, Department of EE-Systems, University of Southern California, October 1994 (Also to appear in Formal Methods in System Design, Kluwer Academic Publishers, 1996).

[40] J. S. Wallis, "Hadamard Matrices," *Lecture Notes No. 292,* Springer-Verlag, New York, 1972.

[41] X. Wu, X. Chen and S. L. Hurst, "Mapping Reed-Muller coefficients and the minimization of Exclusive-OR switching functions," *Proc. IEE*, vol. E129, pp. 15-20, 1982.