Applications of Multi-Terminal Binary Decision Diagrams

E. Clarke M. Fujita* X. Zhao

April, 1995

CMU-CS-95-160

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

* Fujitsu Laboratories of America Inc.
77 Rio Robles
San Jose, CA 95134

This research was sponsored in part by the National Science Foundation under Grant No. CCR-9217549, by the Semiconductor Research Corporation under Contract No. 94-DJ-294, and by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under Grant No. F33615-93-1-1330. The US Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation thereon.

Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Wright Laboratory or the United States Government.

Keywords: binary decision diagrams, multi-terminal binary decision diagrams, binary moment diagrams, hybrid decision diagrams, word level properties, arithmetic circuit, Pentium, division circuit

Abstract

Functions that map boolean vectors into the integers are important for the design and verification of arithmetic circuits. MTBDDs and BMDs have been proposed for representing this class of functions. We discuss the relationship between these methods and describe a generalization called hybrid decision diagrams which is often much more concise.

The Walsh transform and Reed-Muller transform have numerous applications in computer-aided design, but the usefulness of these techniques in practice has been limited by the size of the boolean functions that can be transformed. Currently available techniques limit the functions to less than 20 variables. In this paper, we show how to compute concise representations of the Walsh transform and Reed-Muller transform for functions with several hundred variables.

We show how to implement arithemetic operations efficiently for hybrid decision diagrams. In practice, this is one of the main limitations of BMDs since performing arithmetic operations on functions expressed in this notation can be very expensive.

In order to extend symbolic model checking algorithms to handle arithmetic properties, it is essential to be able to compute the BDD for the set of variable assignments that satisfy an arithmetic relation. Bryant and Chen do not provide an algorithm for this.

In our paper, we give an efficient algorithm for this purpose. Moreover, we prove that for the class of linear expressions, the time complexity of our algorithm is linear in the number of variables. Our techniques for handling arithmetic operations and relations are used intensively in the verification of an SRT division algorithm similar to the one that is used in the Pentium.

1. Introduction

Large integer matrices arise naturally in the design and verification of arithmetic circuits. In this paper, we describe how to represent and manipulate such matrices efficiently using *Multi-Terminal Binary Decision Diagrams* (MTBDDs) [6]. An MTBDD is like an ordinary Binary Decision Diagram except that the terminal nodes can be arbitrary integer values instead of just 0 and 1. Previously, we have demonstrated how MTBDDs can be used to represent functions that map boolean vectors into the integers. Our representation for integer matrices is based on this technique. An integer matrix with dimensions $2^m \times 2^n$ can be treated as a function that maps boolean vectors of length $m+n$ into the integers. Various matrix operations can be performed by operations on the corresponding integer functions.

The Walsh transform and the Reed-Muller transform [9] have numerous applications in computer aided design, particularly in synthesis and testing of circuits. Unfortunately, the usefulness of these techniques in practice has been limited by the size of the boolean functions that can be handled by the transform. Since these transforms are given as vectors with length of 2^n where n is the number of variables in the function, currently available techniques limit the functions to less than 20 variables. Since the Walsh matrix and the Reed-Muller matrix have simple recursive definitions, they can be encoded efficiently by MTBDDs. In this manner, we can compute concise representations for the transforms of functions with several hundred variables.

Recently, Bryant and Chen [4] have proposed Binary Moment Diagrams (BMDs) for representing functions that map boolean vectors into the integers. We show that the BMD of a function is the MTBDD that results from applying the inverse Reed-Muller transformation [10] to the function. The transformation can be computed using the techniques that we have developed for manipulating large matrices. The transformation matrix in this case is the Kronecker product [2] of a number of identical 2×2 matrices. We show that the Kronecker products of other 2×2 matrices behave in a similar way. In fact, the transformations obtained from Kronecker products of other matrices will in many cases be more concise than the BMD. We have further generalized this idea so that the transformation matrix can be the Kronecker product of different matrices. In this way, we obtain a representation, called Hybrid Decision Diagram (HDD), that is more concise than either the MTBDD or the BMD.

One of the main limitations of Bryant and Chen's work is that performing arithmetic operations on functions represented by BMDs is very expensive. We show how these operations can be implemented not only for BMDs, but for hybrid decision diagrams as well. Although the worst case complexity of some of these operations is exponential, our algorithms work quite well in practice.

Most of the properties that we want to verify about arithmetic circuits can be expressed as arithmetic relations. In order to extend symbolic model checking algorithms [5] to handle arithmetic properties, it is essential to be able to compute the BDD for the set of variable assignments that satisfy a relation. Bryant and Chen do not provide an algorithm for this. In this paper, we give an efficient algorithm for this purpose. Moreover, we show that for the class of linear expressions, the time complexity of our algorithm is linear in the number of variables. Our techniques for handling arithmetic operations and relations are used intensively in the verification of an SRT division algorithm similar to the one that is

used in the Pentium.

Our paper is organized as follows: Section 2 gives the basic properties of MTBDDs that are used in the remainder of the paper. Section 3 shows how the results of the previous section can be used to implement standard operations like addition and multiplication of very large integer matrices. Section 4 describes how BDDs can be obtained for recursively defined integer matrices and shows how to compute the spectral transforms for boolean functions. In Section 4 we also illustrate the power of this representation by computing the transforms of several very large boolean functions. Section 5 describes the relationship between BMDs and the inverse Reed-Muller transformation. This section also introduces Kronecker product and shows how it can be used to generalize BMDs. The next section introduces hybrid decision diagrams and provides experimental evidence to show the usefulness of this representation. In Section 7, we show how arithmetic operations can be implemented. In Section 8, we give an efficient algorithm for computing the set of assignments that satisfy an arithmetic relation expressed in terms of hybrid decision diagrams. The paper concludes in Section 9 with a brief summary and a discussion of directions for future research.

2. Multi-terminal binary decision diagrams

Ordered binary decision diagrams (BDDs) are a canonical representation for boolean formulas proposed by Bryant [3]. They are often substantially more compact than traditional normal forms such as conjunctive normal form and disjunctive normal form. They can also be manipulated very efficiently. Hence, BDDs have become widely used for a variety of CAD applications, including symbolic simulation, verification of combinational logic and, more recently, verification of sequential circuits.

A BDD is similar to a binary decision tree, except that its structure is a directed acyclic graph rather than a tree, and there is a strict total order placed on the occurrence of variables as one traverses the graph from root to leaf. Algorithms of linear complexity exist for computing BDD representations of $\neg f$ and $f \vee g$ from the BDDs for the formulas f and g.

Let $f : B^m \rightarrow Z$ be a function that maps boolean vectors of length m into integers. Suppose $n_1, \ldots, n_N$ are the possible values of f. The function f partitions the space B^m of boolean vectors into N sets $\{S_1, \cdots, S_N\}$, such that $S_i = \{\, \bar{x} \mid f(\bar{x}) = n_i \,\}$. Let f_i be the characteristic function of S_i, we say that f is in *normal form* if $f(\bar{x})$ is represented as $\sum_{i=1}^{N} f_i(\bar{x}) \cdot n_i$. This sum can be represented as a BDD with integers as its terminal nodes. We call such DAGs *Multi-Terminal BDDs* (MTBDDs) [1, 6].

Any arithmetic operation $\odot$ on MTBDDs can be performed in the following way.

$$\begin{aligned} h(\bar{x}) &= f(\bar{x}) \odot g(\bar{x}) \\ &= \sum_{i=1}^{N} f_i(\bar{x}) \cdot n_i \odot \sum_{j=1}^{N'} g_j(\bar{x}) \cdot n'_j \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N'} f_i(\bar{x}) g_j(\bar{x}) (n_i \odot n'_j) \end{aligned}$$

$$= \sum_{k=1}^{N''} \bigvee_{n_i \odot n'_j = n''_k} f_i(\bar{x}) g_j(\bar{x}) n''_k$$

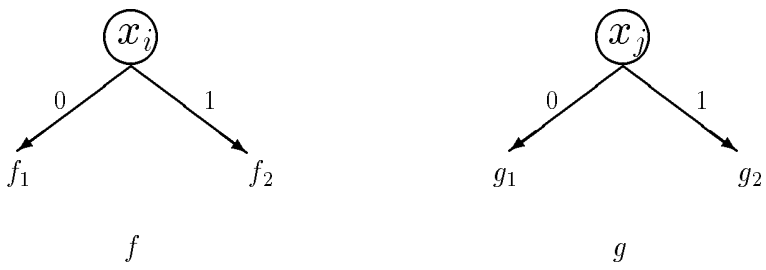

Figure 1: BDDs for f and g

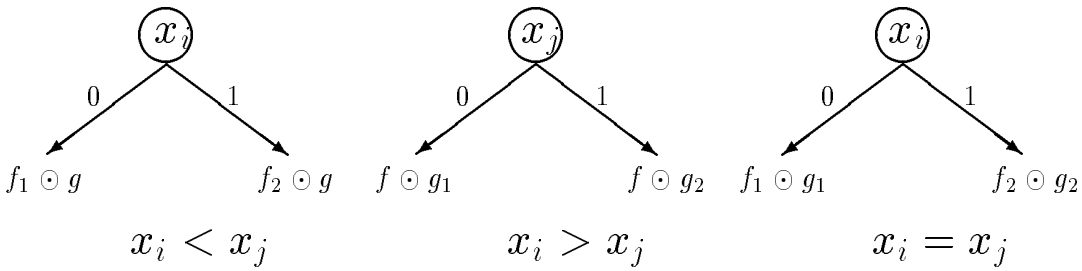

Figure 2: BDD of $f \odot g$

We now give an efficient algorithm for computing $f(\bar{x}) \odot g(\bar{x})$.

- If f is a leaf, then for each leaf of g, apply $\odot$ with f as the first argument.
- If g is a leaf, then for each leaf of f, apply $\odot$ with g as the second argument.
- Otherwise, f and g have the form in Figure 1, and the BDD for $f \odot g$, depending on the relative order of x_i and x_j, is given in Figure 2.

The resulting diagram may not be in normal form. In order to convert it into normal form, a *reduction* phase is needed. The algorithm for this phase is essentially identical to the reduction phase in Bryant's algorithm for constructing BDDs [3] .

Functions that map boolean vectors into the integers can also be represented as arrays of BDDs. These BDDs have boolean values and each corresponds to one bit of the binary representation of the function value. In general, it is quite expensive to perform operations using this representation.

3. Matrix Operations

Let M be a $2^k \times 2^l$ matrix over N. It is easy to see that M can be represented as a function $M : B^{k+l} \rightarrow N$, such that $M_{ij} = M(\bar{x}, \bar{y})$, where $\bar{x}$ is the bit vector for i and $\bar{y}$ is the bit vector for j. Therefore, matrices with integer values can be represented as integer valued functions using the MTBDD representation in Section 2. We need the following operations for integer matrices for computing the spectral transforms: *absolute value*, *scalar multiplication*, *addition*, *sorting a vector of integers*, *summation over one dimension*, and *matrix multiplication*. The first three operations are trivial and will not be discussed in this paper.

- Summing matrices over one dimension

  It is sometimes desirable to obtain a 2^n vector from a $2^n \times 2^m$ matrix that each element in the vector is the summation of the corresponding column, i.e. $M'_i = \sum_{j=0}^{2^m-1} M_{ij}$. When the matrices are expressed in terms of integer valued functions, the equation becomes $M'(\bar{x}) = \sum_{\bar{y}} M(\bar{x}, \bar{y})$, where $\sum_{\bar{y}}$ means "sum over all possible assignments to $\bar{y}$". In practice, $\sum_{\bar{y}} M(\bar{x}, \bar{y})$ can be computed as:

$$\begin{aligned} & \sum_{y_1 y_2 \cdots y_m} M(\bar{x}, y_1, y_2, \ldots, y_m) \\ = & \sum_{y_1 y_2 \cdots y_{m-1}} \sum_{y_m} M(\bar{x}, y_1, y_2, \ldots, y_m) \\ = & \sum_{y_1 y_2 \cdots y_{m-1}} (M(\bar{x}, y_1, y_2, \ldots, y_{m-1}, 0) + M(\bar{x}, y_1, y_2, \ldots, y_{m-1}, 1)) \end{aligned}$$

  In this way, each variable in $\bar{y}$ is eliminated by performing an addition.

  This operation can also be used to sum the elements of a vector and to obtain a two dimensional matrix from a three dimensional matrix by summing over one dimension. Although this operation works well in many cases, the worst case complexity can be exponential in the number of variables.

- Sorting vectors

  Frequently, it is useful to rearrange the elements in a vector so that they are in non-decreasing order. When the number of different values in the vector is not very large, the sorted vector can be represented concisely without using MTBDDs. In order to uniquely determine a sorted vector, we only need to know the set of different values and the number of occurrences of each value. Thus, the sorted vector can be represented as a list with length m, where m is the number of different values. Each element in the list contains the value and number of its occurrences.

  It is easy to find the set of different values, since it is only necessary to collect all of the terminal nodes in the MTBDD. The number of occurrences N_k of a possible value C_k can be calculated as $N_k = \sum_{i=0}^{2^n-1}$(if $M_i = C_k$ then 1 else 0). The operation of summation over a vector discussed previously can be applied to compute this sum. Although, in general, the complexity of the summation operation does not have a satisfactory upper bound, summation over a vector takes time linear in the size of

the MTBDD representing the vector. Thus the complexity of the sorting operation is linear in both the number of distinct values in the vector and the size of the MTBDD representation of the vector.

- Matrix multiplication

  Suppose that two matrices A and B have dimensions $2^k \times 2^l$ and $2^l \times 2^m$, respectively. Let $C = A \times B$ be the product of A and B, C will have dimension $2^k \times 2^m$. If we treat A and B as integer valued functions, we can compute the product matrix C as

$$C(\bar{x}, \bar{z}) = \sum_{\bar{y}} A(\bar{x}, \bar{y}) B(\bar{y}, \bar{z})$$

  using the summation operation discussed above. In general, the complexity of this operation can also be exponential in the number of variables.

4. Spectral transformations of boolean functions

Two of the most commonly used transformations in digital circuit design are the Walsh transform and the Reed-Muller transform [9]. In this section, we will show how the MTBDD based techniques described previously can be used to compute concise representations of the spectra for these transformations.

The Walsh matrix T_n has the recursive definition:

$$T_0 = 1 \qquad T_n = \begin{bmatrix} T_{n-1} & T_{n-1} \\ T_{n-1} & -T_{n-1} \end{bmatrix}$$

Each element of the matrix is determined by its row and column coordinates. We will encode the 2^n columns by variables $y_n, \ldots, y_1$ and the 2^n rows by the variables $x_n, \ldots, x_1$. T_n can be represented as an integer valued function:

$$\begin{aligned} T_n(y_n, \ldots, y_1, x_n, \ldots, x_1) &= \begin{cases} T_{n-1}(y_{n-1}, \ldots, y_1, x_{n-1}, \ldots, x_1) & \text{if } (x_n y_n \neq 1) \\ -T_{n-1}(y_{n-1}, \ldots, y_1, x_{n-1}, \ldots, x_1) & \text{if } (x_n y_n = 1) \end{cases} \\ &= T_{n-1}(y_{n-1}, \ldots, y_1, x_{n-1}, \ldots, x_1) \cdot (\text{if } x_n y_n = 1 \text{ then } -1 \text{ else } 1) \end{aligned}$$

The above recursive definition can be expressed by an MTBDD as shown in Figure 3.

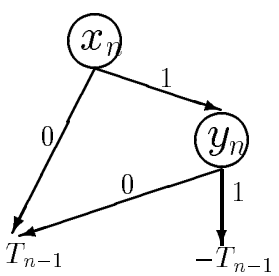

Figure 3: MTBDD for T_n

The Walsh transform maps a boolean vector f with length 2^n to an integer vector of length 2^n, denoted by W_f, in which each component is between -2^n to 2^n. The transform

can be easily expressed using the Walsh matrix, $\mathbf{W_f} = \mathbf{T_n} \times (\mathbf{1} - \mathbf{2f})$ [9]. For example, the vector $[0,1,1,1,1,0,0,0]^T$ is mapped into $[0,0,0,0,-4,4,4,4]^T$.

Likewise, the Reed-Muller matrix has the recursive definition:

$$S_0 = 1 \qquad T_n = \begin{bmatrix} S_{n-1} & 0 \\ S_{n-1} & S_{n-1} \end{bmatrix}$$

which can be expressed by

$$\begin{aligned} S_n(y_n, \ldots, y_1, x_n, \ldots, x_1) &= \begin{cases} S_{n-1}(y_{n-1}, \ldots, y_1, x_{n-1}, \ldots, x_1) & \text{if } (\neg x_n) \cdot y_n = 0 \\ 0 & \text{if } (\neg x_n) \cdot y_n = 1 \end{cases} \\ &= \text{if } ((\neg x_n) \cdot y_n) \text{ then } 0 \text{ else } S_{n-1}(y_{n-1}, \ldots, y_1, x_{n-1}, \ldots, x_1) \end{aligned}$$

and has the MTBDD representation in Figure 4.

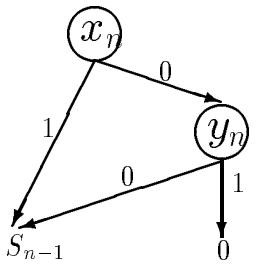

Figure 4: MTBDD for S_n

The Reed-Muller transformation maps a boolean vector of length 2^n into another boolean vector of the same length. This transformation can be expressed by the matrix multiplication $\mathbf{R_f} = \mathbf{S_n} \times \mathbf{f}$. However, during the matrix mutiplication, integer addition is replaced by XOR in order to perform the modulo 2 arithmatic. For example, the vector $[0,1,1,1,1,0,0,0]^T$ is mapped into $[0,1,1,1,1,0,0,0]^T$.

When the number of variables is large, the transformations can be computed by representing the matrices and the vectors as MTBDDs and matrix operations can be performed as described in Section 3 and Section 4.

To illustrate the power of these techniques, we have computed the Walsh transformation and Reed-Muller transformation for some large combinatorial circuits, including two adders and some of the ISCAS benchmarks (Table 1). The examples were run on a DEC-5000 and run time is shown in seconds.

5. Kronecker transformations

Recently, Bryant and Chen[4] have developed a new representation for functions that map boolean vectors to integer values. This representation is called the Binary Moment Diagram (BMD) of the function. Instead of the Shannon expansion $f = xf_1 + (1-x)f_0$, they use the expansion $f = f_0 + xf'$, where f' is equal to $f_1 - f_0$. After merging the common subexpressions, a DAG representation for the function is obtained. They prove in their

example circuit					Walsh coef.		R-M coef.	
circuit	|input|	output	# of gates	|BDD|	|MTBDD|	time	|MTBDD|	time
c1908	33	9	880	3607	1850	44	27748	184
c3540	50	361	1669	520	15985	171	4679	8.2
c5315	178	813	2307	1397	7069	328	2647	25
50-bit adder	100	C_{50}	250	151	7456	23	249	2.3
100-bit adder	200	C_{100}	500	301	29906	128	499	11

Table 1: Experimental results for spectral transformations

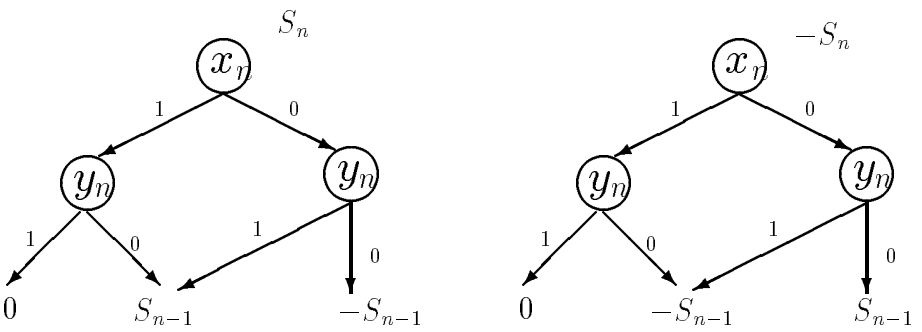

Figure 5: MTBDD for S_n

paper that this gives a compact representation for certain functions which have exponential size if represented by MTBDDs directly.

There is a close relationship between this representation and the inverse Reed-Muller transformation [10]. The matrix for the inverse Reed-Muller transformation is defined recursively by

$$S_0 = 1 \qquad S_n = \begin{pmatrix} S_{n-1} & 0 \\ -S_{n-1} & S_{n-1} \end{pmatrix}$$

which has a linear MTBDD representation as shown in Figure 5. Let $\overline{i} \in B^n$ be the binary representation of the integer $0 \leq i < 2^n$. A function $f : B^n \rightarrow N$ can be represented as a column vector where the value of the i-th entry is $f(\overline{i})$. We will not distinguish between a function and its corresponding column vector. The inverse Reed-Muller transformation can be obtained by multiplying the transformation matrix and the column vector $\widehat{f} = S \times f$ using the technique described in previous section.

Theorem 1 *The MTBDD of $\widehat{f}$ is isomorphic to the BMD of f.*

The Kronecker product of two matrices is defined as follows:

$$A \otimes B = \begin{pmatrix} a_{11} & \ldots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \ldots & a_{nm} \end{pmatrix} \otimes B = \begin{pmatrix} a_{11}B & \ldots & a_{1m}B \\ \vdots & & \vdots \\ a_{n1}B & \ldots & a_{nm}B \end{pmatrix}$$

The inverse Reed-Muller matrix can be represented as the Kronecker product of n identical 2×2 matrices:

$$S_n = \begin{pmatrix} S_{n-1} & 0 \\ -S_{n-1} & S_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \otimes S_{n-1} = \underbrace{\begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \otimes \ldots \otimes \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}}_{n}$$

The inverse Reed-Muller transformation is not unique in this respect. Other transformations that are defined as Kronecker products of 2×2 matrices may also provide concise representations for functions mapping boolean vectors into integers. In particular, Reed-Muller matrix R_n and Walsh matrix W_n can be represented as Kronecker products shown below:

$$R_n = \begin{pmatrix} R_{n-1} & 0 \\ R_{n-1} & R_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \otimes R_{n-1} = \underbrace{\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \otimes \ldots \otimes \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}}_{n} \quad \text{and}$$

$$W_n = \begin{pmatrix} W_{n-1} & W_{n-1} \\ W_{n-1} & -W_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes W_{n-1} = \underbrace{\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \ldots \otimes \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}}_{n}.$$

Although a Kronecker transformation can be performed by matrix multiplication, there is a more efficient way of computing it. It is a well known property of the Kronecker product that

$$\bigotimes_{i=0}^{k} A_i = \prod_{i=0}^{k} (I_{2^i} \otimes A_i \otimes I_{2^k})$$

where each A_i is a 2×2 matrix and I_k is the identity matrix of size $k \times k$. A transformation of the form $(I_{2^i} \otimes A_i \otimes I_{2^k})$ is called a *basic transformation*. Let $A_i = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}$, and let g be a function represented as a MTBDD, then the basic transformation $g' = (I_{2^i} \otimes A_i \otimes I_{2^k}) \times g$ can be computed as

$$g' = \text{ if } x_i \text{ then } a_{00}\ g|_{x_i=0} + a_{01}\ g|_{x_i=1} \text{ else } a_{10}\ g|_{x_i=0} + a_{11}\ g|_{x_i=1}.$$

As a result of this observation, the Kronecker transformation can be performed by a series of basic transformations. Moreover, it can be proved that the order of the basic transformations does not effect the final result.

In fact, the Kronecker product of any non-singular 2×2 matrices can be used as a transformation matrix and will produce a canonical representation for the function. If the entries of the 2×2 matrix are restricted among $\{0, 1, -1\}$, there are six interesting matrices

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ -1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}, \text{ and } \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}.$$

All other matrices are either singular or would produce BDDs that are isomorphic to one of the six matrices.

We have applied these transformations to the functions discussed in paper[4]. The transformation can be partitioned into two groups of three each. The MTBDDs of the results after applying the transformations in the same group have the same complexity.
Let $X = \sum_{i=0}^{n} x_i 2^i, Y = \sum_{j=0}^{m} y_j 2^j, X_j = \sum_{i=0}^{n_j} x_{ij} 2^i$, then

base matrix	X	X^2	XY	X^k	$\prod_{j=0}^{k} X_j$
$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$	$O(2^n)$	$O(2^{2n})$	$O(2^{n+m})$	$O(2^{kn})$	$O(\prod_{j=0}^{k} 2^{n_j})$
$\begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$ $\begin{pmatrix} 0 & 1 \\ -1 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$	$O(n)$	$O(n^2)$	$O(nm)$	$O(n^k)$	$O(\prod_{j=0}^{k} n_j)$

The possibility of using BMDs to represent boolean functions is discussed in [4]. In general, the BMD does not appear to be better than the ordinary BDD for representing boolean functions. In order to see why this is true, consider the boolean Reed-Muller transformation, which is sometimes called the Functional Decision Diagram or FDD[8]. This transformation can be obtained by applying the modulo 2 operations to all of the terminal nodes of the BMD. Consequently, the size of FDD is always smaller than the size of the BMD. Since the inverse boolean Reed-Muller transformation is the same as the boolean Reed-Muller transformation, the FDD of the FDD is the original BDD. Therefore, for every function f such that $|\text{FDD}_f| < |\text{BDD}_f|$, there exists another function f' which is the boolean Reed-Muller transform of f such that $|\text{BDD}_{f'}| < |\text{FDD}_{f'}|$. In particular, both the BMD and the FDD representations for the middle bit of a multiplier are still exponential.

6. Hybrid decision diagrams

In the previous sections, we have discussed transformations that can be represented as the Kronecker product of a number of identical 2×2 matrices. If the transformation matrix is a Kronecker product of different 2×2 matrices, we still have a canonical representation of the function. We call transformations obtained from such matrices *hybrid transformations.*

A similar strategy has been tried by Becker [7]. However, his technique only works for the boolean domain. When using his technique, all of the transformation matrices, the original function and the resulting function must have boolean values. Our technique, on the other hand, works over the integers. By allowing integer values, we can handle a wider range of functions. Moreover, we can obtain larger reduction factors since we have more choices for transformation matrices.

We can apply this idea to reduce the size of MTBDD representation of functions. Since there is no known polynomial algorithm to find the hybrid Kronecker transformation that minimizes MTBDD size, we use a greedy algorithm to reduce the size. If we restrict the entries in the matrix to the set $\{0, 1, -1\}$, then there are six matrices we can try. For each variable, we select the matrix that gives the smallest MTBDD size. The MTBDDs obtained from such transformations are called Hybrid Decision Diagrams (HDDs). We have tried this method on the ISCAS85 benchmark circuits. In some cases we have been able to reduce the size of BDD representation by a factor of 1300. However, reductions of this magnitute

usually occur when the original function has a bad variable ordering. If dynamic variable ordering is used, then our method gives a much smaller reduction factor.

example circuit			without reordering			with reordering		
circuit	|input|	output	|BDD|	|BMD|	|HDD|	|BDD|	| BMD |	|HDD|
c1355	41	1327	9419	1217689	2857	4407	478903	1518
c1908	33	12	3703	140174	1374	1581	154488	632
c5315	178	676	679593	2820	521	108	5106	107

Table 2: Experimental results for hybrid transformations of some ISCAS85 circuits

We have tried several techniques to increase the number of possible matrices. The first technique involves increasing the number of entries in the matrices. This can be accomplished by allowing the entries to take larger values or by using the complex numbers $\{0, 1, -1, i, -i, 1+i, 1-i, i-1, -i-1\}$. Unfortunately, neither extension improved the results significantly.

The second technique involves using transformation matrices that are Kronecker products of larger matrices. For example, we have tried hybrid Kronecker transformations based on 4×4 matrices instead of 2×2 matrices. Although we have been able to reduce the BDD size even further using this technique, the time it takes to find such transformations is much bigger since the number of possibilities is considerably larger.

Note that our technique can achieve comparable and sometimes better results than dynamic variable reordering. Thus, in some cases, it can serve as an alternative to dynamic variable reordering. We conjecture that the combination of both techniques together may result in reductions that neither technique can achieve alone.

7. Arithmetic operations on hybrid decision diagrams

In order to make the techniques described in the previous sections more useful, it is desirable to be able to perform various arithmetric operations on on hybrid BDDs. In this paper, we only consider the cases of addition and multiplication of two integers.

Suppose that f is transformed into f' by the matrix T_1 and g is transformed into g' by the matrix T_2 using the techniques discussed in the previous sections. Scalar multiplication is simple to perform.

$$(cf)' = T_1 \times (cf) = cT_1 \times f = cf'$$

When $T_1 = T_2$, finding the sum of two function is also simple.

$$(f+g)' = T_1 \times (f+g) = T_1 \times f + T_1 \times g = f' + g'$$

If $T_1 \neq T_2$, the transformation applied to the sum must be determined first. Suppose we use T_2 as the transformation matrix for the result,

$$(f+g)' = T_2 \times (f+g) = T_2 \times f + T_2 \times g = T_2 \times T_1^{-1} \times f' + g'.$$

Next, we consider how to perform multiplication. We choose T_2 as the transformation matrix for $(f \cdot g)$. Suppose the top level variable is x_i. Assume the top level transform for f is $\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ with inverse $\begin{pmatrix} a'_{11} & a'_{12} \\ a'_{21} & a'_{22} \end{pmatrix}$. Assume also the top level transform for g is $\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$ with inverse $\begin{pmatrix} b'_{11} & b'_{12} \\ b'_{21} & b'_{22} \end{pmatrix}$. Then $T_2 = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \otimes S_2 = \begin{pmatrix} b_{11}S_2 & b_{12}S_2 \\ b_{21}S_2 & b_{22}S_2 \end{pmatrix}$.

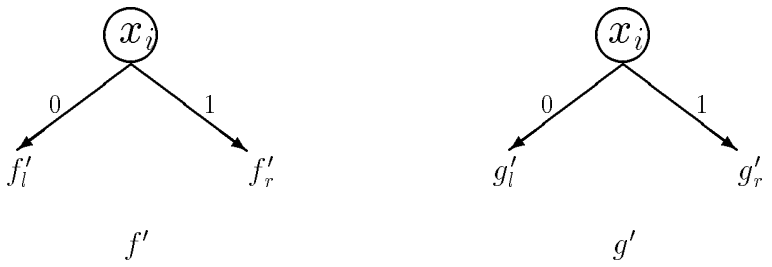

Figure 6: BDDs for f' and g'

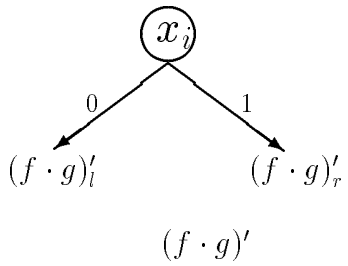

Figure 7: BDD of $(f \cdot g)'$

$$\begin{aligned}
(f \cdot g)' &= T_2 \times (f \cdot g) \\
&= \begin{pmatrix} b_{11}S_2 & b_{12}S_2 \\ b_{21}S_2 & b_{22}S_2 \end{pmatrix} \times \begin{pmatrix} f_0 \cdot g_0 \\ f_1 \cdot g_1 \end{pmatrix} \\
&= \begin{pmatrix} b_{11}(f_0 \cdot g_0)' + b_{12}(f_1 \cdot g_1)' \\ b_{21}(f_0 \cdot g_0)' + b_{12}(f_1 \cdot g_1)' \end{pmatrix}.
\end{aligned}$$

Consequently,

$$\begin{aligned}
(f \cdot g)'_l &= b_{11}(f_0 \cdot g_0)' + b_{12}(f_1 \cdot g_1)' \\
&= b_{11}((a'_{11}f_l + a'_{12}f_r) \cdot (b'_{11}g_l + b'_{12}g_r))' + b_{12}((a'_{21}f_l + a'_{22}f_r) \cdot (b'_{21}g_l + b'_{22}g_r))' \\
&= (b_{11}a'_{11}b'_{11} + b_{12}a'_{21}b'_{21})(f_l \cdot g_l)' + (b_{11}a'_{11}b'_{12} + b_{12}a'_{21}b'_{22})(f_l \cdot g_r)' \\
&\quad +(b_{11}a'_{12}b'_{11} + b_{12}a'_{22}b'_{21})(f_r \cdot g_l)' + (b_{11}a'_{12}b'_{12} + b_{12}a'_{22}b'_{22})(f_r \cdot g_r)'
\end{aligned}$$

$$(f \cdot g)'_r = b_{21}(f_0 \cdot g_0)' + b_{22}(f_1 \cdot g_1)'$$

$$
\begin{aligned}
&= b_{21}((a'_{11}f_l + a'_{12}f_r)\cdot(b'_{11}g_l + b'_{12}g_r))' + b_{22}((a'_{21}f_l + a'_{22}f_r)\cdot(b'_{21}g_l + b'_{22}g_r))' \\
&= (b_{21}a'_{11}b'_{11} + b_{22}a'_{21}b'_{21})(f_l\cdot g_l)' + (b_{21}a'_{11}b'_{12} + b_{22}a'_{21}b'_{22})(f_l\cdot g_r)' \\
&\quad +(b_{21}a'_{12}b'_{11} + b_{22}a'_{22}b'_{21})(f_r\cdot g_l)' + (b_{21}a'_{12}b'_{12} + b_{22}a'_{22}b'_{22})(f_r\cdot g_r)'
\end{aligned}
$$

Since both $(f\cdot g)'_l$ and $(f\cdot g)'_r$ can be computed in terms of $(f_l\cdot g_l)', (f_l\cdot g_r)', (f_r\cdot g_l)'$, and $(f_r\cdot g_r)'$, we can compute the transformation of the product in a recursive manner. If we store these intermidiate results, the total number of recursive calls to compute $(f\cdot g)'$ will be at most $|f'||g'|$. Because of the additions that are needed in the computation, the worst case complexity can still be exponential. However, in practice, this algorithm works quite well. As an example, in Table 3, we show the time it takes to compute the hybrid decision diagram for $(\sum_{i=0}^{n} x_i 2^i)\cdot(\sum_{j=0}^{n} y_j 2^j)$ from the hybrid decision diagrams for $(\sum_{i=0}^{n} x_i 2^i)$ and $(\sum_{j=0}^{n} y_j 2^j)$.

n	10	20	30	40	50	60	70	80	90	100
time(sec)	1.6	2.0	2.2	2.5	3.0	3.5	3.5	4.5	5.5	6.6
|HDD|	139	479	1019	1759	2699	3839	5179	6719	8459	10399

Table 3: Experimental results for computing $(\sum_{i=0}^{n} x_i 2^i)\cdot(\sum_{j=0}^{n} y_j 2^j)$

Now that we are able to add and multiply functions, we can perform all of the standard logical operations. For example $(\neg f)' = (1-f)' = 1' - f'$ and $(f\wedge g)' = (f\cdot g)'$.

8. Equations and inequalities

Frequently, it is useful to be able to compute the set of assignments that satisfy $f_1 \sim f_2$, where $\sim$ can be one of $=, \neq, <, \leq, >$, or $\geq$. For example, the following inequality is extremely important for the correctness of the radix-4 SRT floating point division algorithm.

$$-2\cdot\text{divisor} \leq 3\cdot\text{remainder} \leq 2\cdot\text{divisor}$$

Both *divisor* and *remainder* in the inequality can be regarded as arrays of boolean variables. In order to verify the correctness of the algorithm, it is necessary to determine the set of assignments to these variables that make the inequality true.

Finding the set of assignments that satisfy an inequality can be reduced to the problem of finding the set of assignments that make a function f positive. Equations can be handled in a similar manner. A straightforward way of solving the problem is to convert f to an MTBDD and then pick the terminal nodes with the correct sign. However, this does not work very well in general, because some functions have MTBDDs with exponential size but hybrid BDDs of polynomial size. For example, let $f_1 = \sum_{i=0}^{m} x_i 2^i$ and $f_2 = \sum_{j=0}^{m} y_j 2^j$. Both of these functions and their difference have linear size BMDs. The BDD for the set of assignments satisfying $f_1 - f_2 > 0$ also has linear size. But the MTBDD size for $f_1 - f_2$ is exponential.

We have developed an algorithm that can substantially reduce the cost for computing arithmetic relations between certain functions represented by hybrid decision diagrams. In the process, we only need to know the sign of the function values. Thus, if we find out that all of the values in a sub-HDD have the same sign, we can conclude that all assignments in the sub-HDD will have the same value for the relation. Consequently, we don't need to continue to expand this sub-HDD.

To obtain a good algorithm for this problem, it is necessary to determine efficiently if a sub-HDD has uniform sign. This can be achieved by computing upper and lower bounds for the sub-HDD. The algorithm given below determines this information. If the intermediate results are stored, the algorithm takes time linear in the number of BDD nodes.

```
bound_values(f, upper, lower)
begin
  if(f is terminal node)
    upper = lower = f.value;

  bound_values(left(f), upper1, lower1);
  bound_values(right(f), upper2, lower2);

  let {{a11, a12}, {a21, a22}} be the inverse matrix at node f;

  upper11 = if a11>0 then a11*upper1 else a11*lower1;
  upper12 = if a12>0 then a12*upper2 else a12*lower2;
  upper21 = if a21>0 then a21*upper1 else a21*lower1;
  upper22 = if a22>0 then a22*upper2 else a22*lower2;

  lower11 = if a11>0 then a11*lower1 else a11*upper1;
  lower12 = if a12>0 then a12*lower2 else a12*upper2;
  lower21 = if a21>0 then a21*lower1 else a21*upper1;
  lower22 = if a22>0 then a22*lower2 else a22*upper2;

  upper = max(upper11 + upper12, upper21 + upper22);
  lower = min(lower11 + lower12, lower21 + lower22);
end
```

The improved algorithm for computing the BDD for the set of assignments that make the function f positive is given below. A similar algorithm is used to find the set of assignments that make a function zero.

```
bdd greater_than_0(f)
begin
  if(f is terminal node)
    if(f.value > 0)
      return(True);
```

```
    else
      return(False);

  bound_values(f, upper, lower);
  if(upper <= 0)
    return(False);
  if(lower > 0)
    return(True);

  let {{a11, a12}, {a21, a22}} be the inverse matrix at node f;
  left = greater_than_0(a11 * left(f) + a12 * right(f));
  right = greater_than_0(a21 * left(f) + a22 * right(f));
  return(bdd_if_then_else(level(f), left, right));
end
```

This algorithm works extremely well for verification of arithmetic circuits. The following theorem guarantees the efficiency of this algorithm for the set of *linear expressions* when the Hybrid Decision Diagrams are BMDs. Most of the formulas that occur during the verification of the SRT division algorithm are in this class. These expressions have the form $f = \sum_{i=1}^{m} c_i f_i$, where $f_i = \sum_{j=0}^{n} x_{ij} 2^j$ for $1 < i < m$ and the c_i's are integer constants. We use the variable ordering $x_{1n}, x_{2n}, \ldots, x_{mn}, \ldots, x_{10}, x_{20}, \ldots, x_{m0}$. Because $f|_{x_{ij}=1} - f|_{x_{ij}=0} = c_i 2^j$ is a constant, the BMD for f is shown in Figure 8.

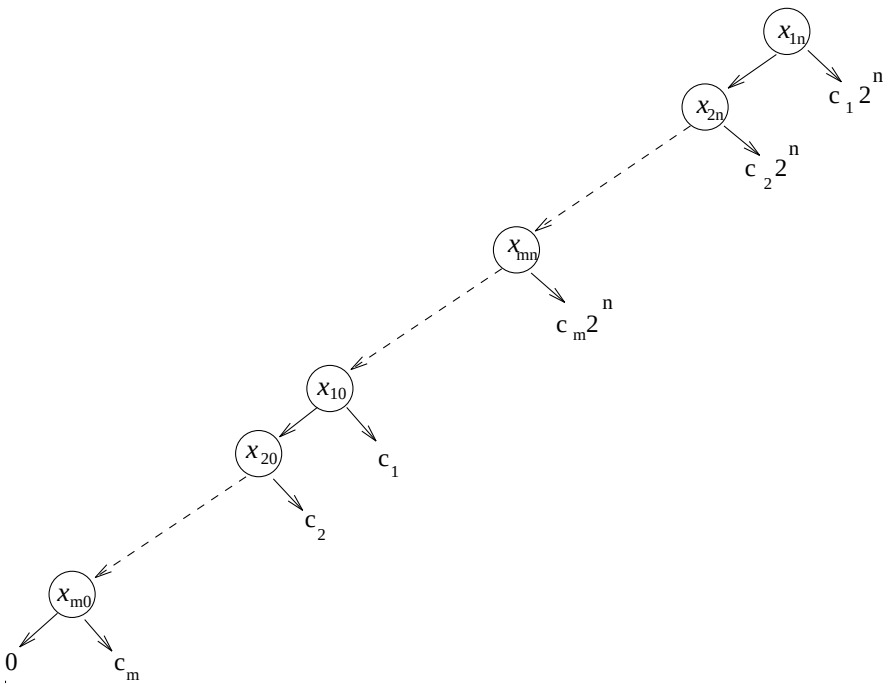

Figure 8: BMD for $\sum_{i=1}^{m} c_i f_i$

Theorem 2 *The complexity of* `greater_than_0` *for* f *is* $O(n^2 \sum_{k=1}^{m} |c_k|)$.

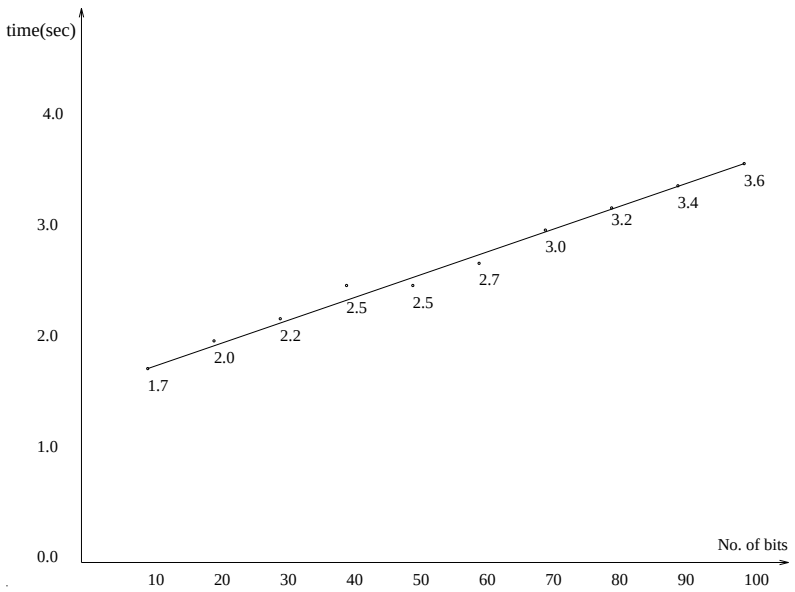

Figure 9: time to compute $-2 \cdot \text{divisor} \leq 3 \cdot \text{remainder} \leq 2 \cdot \text{divisor}$

In the case of linear inequalities, all the new BMDs that are generated have the form of $c + g$, where c is a constant and g is an existing BMD. If we remember the constant without actually adding it to the BMDs, we are able to avoid generating new BMD nodes. After introducing this technique, the complexity to compute `greater_than_0(f)` can be further reduced to $O(n \sum_{k=1}^{m} |c_k|)$. For the example we considered at the beginning of the section, the relationship between the time it takes to compute the inequality and the number of bits is shown in the Figure 9.

9. Summary and directions for future research

In this paper, we have used MTBDDs to represent functions that map boolean vectors into integers. We have also shown how to represent large integer matrices concisely and perform standard matrices operations such as scalar multiplication, matrix addition and matrix multiplication.

The Walsh and Reed-Muller transforms are given by matrices that have simple recursive definition. Because of this, the transforms can be computed efficiently using MTBDDs. In fact, we are able to find the transforms of boolean functions with several hundred variables.

We discuss the relationship between spectral transforms and binary moment diagrams and describe a generalization called the hybrid decision diagram which is often much more concise. We also give an efficient implementation of arithmetic operations on hybrid decision diagrams.

Computing the BDD for the set of variable assignments that satisfy an arithmetic relation is important for reasoning about arithmetic circuits. We give an efficient algorithm for this purpose. Finally, we prove that for the class of linear expressions, the time complexity of our algorithm is linear in the number of variables.

In [6], we show how our technique for computing the Walsh transform can be used in technology mapping. Permutation and complementation of input variables does not change the sorted absolute values of the Walsh spectrum of a boolean function. Thus, by comparing the Walsh spectra of two boolean functions, we obtain a necessary condition for determining if one can be changed to the other by these operations.

The algorithms for performing arithmetic operations and finding the set of variable assignments that satisfy an arithmetic relation make it possible to extend the symbolic model checking algorithms so that they can be used to verify the properties of data paths in addition to controlling circuitry. In a forthcoming paper, we plan to discuss how this technique can be used to verify an SRT algorithm that is similar to the division circuit that is used in the Pentium processor.

There are other possible applications of the techniques discussed in this paper. MTBDDs enable us to represent and manipulate very large matrices efficiently. Some potential applications include image compression, numerical solution of partial differential equations and computation of limit state probabilities for Markov Chains. Since hybrid decision diagrams tend to be more concise than multi-terminal BDDs, they may prove even more useful for this type of application.

References

[1] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of the 1993 Proceedings of the IEEE International Conference on Computer Aided Design.* IEEE Computer Society Press, November 1993.

[2] R. Bellman. *Introcution to matrix analysis*, chapter 5. McGraw-Hill, 1970.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 1986.

[4] R. E. Bryant and Y. A. Chen. Verification of arithmetic functions with binary moment diagrams. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference.* IEEE Computer Society Press, June 1995.

[5] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[6] E. M. Clarke, K. McMillan, X. Zhao, M. Fujita, and J. Yang. Spectral transforms for large boolean functions with applications to technology mapping. In *Proceedings of the 30th ACM/IEEE Design Automation Conference.* IEEE Computer Society Press, June 1993.

[7] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski. Efficient representation and manipulation of switching functions based on ordered kroenecker functional decision diagrams. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference.* IEEE Computer Society Press, June 1994.

[8] R. Drechsler, M. Theobald, and B. Becker. Fast ofdd based minimization of fixed polarity reed-muller expressions. In *Proceedings of the Proceedings of the European Design Automation Conference*. IEEE Computer Society Press, June 1994.

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

[10] D. E. Muller. Application of boolean algebra to switching circuit design and error detection. *IRE Trans.*, 1:6–12, 1954.