BDD Minimization by Linear Transformations

Wolfgang Günther
Rolf Drechsler

Institute of Computer Science
Albert-Ludwigs-University
Am Flughafen 17
79110 Freiburg im Breisgau, Germany

Report 104, May 1998

BDD Minimization by Linear Transformations

Technical Report No. 104

Wolfgang Günther Rolf Drechsler

Institute of Computer Science, Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
{guenther,drechsle}@informatik.uni-freiburg.de

Abstract

Binary Decision Diagrams (BDDs) are a powerful tool and are frequently used in many applications in VLSI CAD, like synthesis and verification. Unfortunately, BDDs are very sensitive to the variable ordering and their size often becomes infeasible. Recently, a new approach for BDD minimization based on linear transformations, i.e. a special type of spectral techniques, has been proposed.

In this paper we study this minimization method in more detail. While so far only experimental results are known, we prove for a family of Boolean functions that by linear transformations an exponential blow up of the BDD size can be prevented. Furthermore, we present a heuristic method that allows to significantly reduce the BDD sizes. By experiments we give a comparison to the state-of-the-art method to demonstrate the advantages of our approach.

1 Introduction

Binary Decision Diagrams (BDDs) are the state-of-the-art data structure in VLSI CAD and in the meantime are integrated in many commercial tools. They are widely used in applications, like synthesis [10, 5] and verification [11, 1, 4]. The major drawback of all BDD based systems is that the underlying data structure is very sensitive to a chosen variable ordering, and for some functions even no efficient representation exists, like for the Boolean multiplication [3].

For this, several extensions of the basic BDD structure have been proposed in the last few years. A very promising extension is based on spectral techniques, since the BDD data structure itself remains unchanged, and only an encoding of the inputs is carried out: Instead of transforming the function table, the input variables are transformed by *linear transformations*. In contrast to spectral methods that operate with $2^n \times 2^n$-matrices, for linear transformations $n \times n$-matrices are sufficient. An efficient heuristic for BDD minimization under linear transformations has been presented in [12], which is called "linear sifting". The widely used sifting algorithm [14] for dynamic BDD minimization has been extended by an effective linear operator which combines neighboring variables of the BDD. The algorithm runs very fast, since the basic operation is (very similar to variable reordering [7]) a local operation. By this, it is possible to obtain much smaller BDD sizes than with sifting alone.

Recently, an exact algorithm for BDD minimization under linear transformations has been proposed [8], by which the size of the BDDs could be reduced significantly in comparison to linear sifting. But although an efficient branch&bound technique known from BDD minimization [6] is used, this approach is only applicable to small functions with up to 7 variables.

In this paper, we study linear transformations from a theoretical and practical point of view: Up to now only experimental studies have been performed and it was not clear by how many nodes the BDD can be reduced in the best case. For a family of Boolean function we prove that the BDD size can in some cases be reduced by an exponential factor, i.e. each BDDs requires an exponential number of nodes, while the transformed BDD has linear size. Then, a new heuristic for BDD minimization based on linear transformations is presented. It can be seen as an extension of the window permutation algorithm [9, 14] to linear transformations. Small windows of neighboring variables are optimized exactly, and these windows are shifted over the BDD. Our approach is based on a combination of the linear sifting algorithm and the window optimization algorithm which optimizes the windows under linear transformations. Experimental results are given to compare our new heuristic to linear sifting. It turns out that in many cases significant reductions can be obtained, i.e. the resulting BDDs are up to a factor of three smaller.

The paper is structured as follows: In Section 2, some basic notations and definitions are introduced. The theoretical result for the effectiveness of linear transformations is presented in Section 3. In Section 4 the minimization techniques are discussed. They are compared by experiments in Section 5. Finally, the paper is summarized.

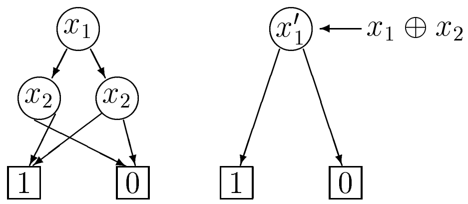

Figure 1: BDDs of the function $f = \overline{x}_1 x_2 + x_1 \overline{x}_2$

2 Preliminaries

2.1 Binary Decision Diagrams

Boolean variables can assume values from $\mathbf{B} := \{0, 1\}$. In the following, we consider Boolean functions $f : \mathbf{B}^n \to \mathbf{B}^m$ over the variable set $X_n = \{x_1, \ldots, x_n\}$.

As well-known, each Boolean function $f : \mathbf{B}^n \to \mathbf{B}$ can be represented by a *Binary Decision Diagram* (BDD) i.e. a directed acyclic graph where a Shannon decomposition

$$f = \overline{x}_i f_{x_i=0} + x_i f_{x_i=1} \quad (1 \leq i \leq n)$$

is carried out in each node.

A BDD is called *ordered* if each variable is encountered at most once on each path from the root to a terminal node and if the variables are encountered in the same order on all such paths. A BDD is called *reduced* if it does not contain vertices either with isomorphic sub-graphs or with both edges pointing to the same node. Reduced and ordered BDDs are unique, i.e. for each Boolean function the BDD can be uniquely determined.

BDDs are defined analogously for multi-output functions $f : \mathbf{B}^n \to \mathbf{B}^m$ as for the case of single-output functions: A BDD G_j for each component function f_j $(1 \leq j \leq m)$ is used for the *shared* BDD representation G for f. The order of the variables is fixed over all G_js.

For functions represented by reduced, ordered BDDs efficient manipulations are possible [2]. In the following, only reduced, ordered BDDs are considered and for briefness these graphs are called BDDs.

2.2 Transformations

Definition 1 A *re-encoding* of a set of input variables $X_n = \{x_1, \ldots, x_n\}$ is an automorphism $\tau : \mathbf{B}^n \to \mathbf{B}^n$, i.e. a mapping for which the inverse mapping exists. The re-encoded input variables are denoted by $\tau(x_1, \ldots, x_n)$.

Example 1 For function $f = \overline{x}_1 x_2 + x_1 \overline{x}_2$, the transformation $\tau(x_1, x_2) = (x_1 \oplus x_2, x_2)$ is an automorphism. The BDDs for f with and without transformation τ are given in Figure 1.

As described above, we restrict ourselves to linear transformations, which will be defined in the following.

Definition 2 *A linear transformation is a linear re-encoding of the set of input variables, i.e. a re-encoding which can be represented by a $n \times n$-matrix over the Galois field* $(\mathbf{B}, \oplus, \cdot)$.

As shown in [13], the number of possible linear transformations is

$$\prod_{i=0}^{n-1} (2^n - 2^i),$$

which is much larger than $n!$, the number of possible variable orderings, but much smaller than $2^n!$, the number of all possible automorphisms. Similar estimations hold for the space requirements: permutations require $O(n)$ space, linear transformations $O(n^2)$ and general automorphisms $O(2^n)$.

3 Exponential Gap

Up to now several promising experimental results have been reported, but it was not clear whether linear transformations only give an improvement of a constant, a polynomial or even an exponential factor.

In the following we prove a theorem which demonstrates the effectiveness of linear transformations to BDD minimization: There exists a family of Boolean functions for which BDDs are exponential in size under each variable ordering, but there are linearly transformed BDDs with linear size. For the proof an argument of lower bound computation from [3] is needed that is briefly reviewed.

Let $f : \mathbf{B}^n \to \mathbf{B}$ be a Boolean function and let (L, R) be a partition of X_n. Then, any input assignment $x : X_n \to \mathbf{B}$ can be split into a *left input assignment* $l : L \to \mathbf{B}$ and a *right input assignment* $r : R \to \mathbf{B}$. We use the notation $x = lr$. A set $\mathbf{F}_{(L,R)}$ of left input assignments is called a *fooling set* for the partition (L, R) iff for each two distinct $l, l' \in \mathbf{F}_{(L,R)}$ there is a right input assignment r with $f(lr) \neq f(l'r)$. Fixing a real parameter $\omega \in (0, 1)$, a partition (L, R) is called a *balanced partition* iff

$$\lfloor \omega \cdot n \rfloor \leq |L| \leq \lceil \omega \cdot n \rceil.$$

Lemma 1 If, for each balanced partition (L, R) with respect to some ω, f has a fooling set $\mathbf{F}_{(L,R)}$ of size c, then the BDD for f has at least c nodes[1].

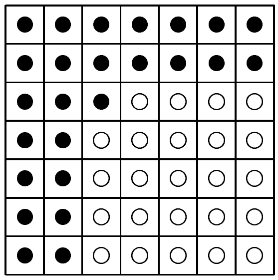

Figure 2: A 7×7 square of points (see Lemma 2)

To simplify the main proof, we first prove a lemma concerning a counting argument on matrices.

Definition 3 Consider an $n \times n$-square of points (see Figure 2). A *line* is either a horizontal row or a vertical column of the square. A line is *covered* iff all its points are covered, and a line is *active* iff at least one point of the line is covered and at least one point of the line is uncovered.

Lemma 2 Let $\lfloor 0.5n^2 \rfloor \leq k \leq \lceil 0.5n^2 \rceil$. If k points of an $n \times n$-square of points are covered, then there are at least n active lines in the square.

Proof: Let k points of an $n \times n$-square be covered, and let k be within the given range. We distinguish the following four cases.

Case 1: There are only horizontal covered lines. Then all n vertical lines are active.

Case 2: There are only vertical covered lines. This case is analogously to case 1.

Case 3: There are both horizontal and vertical covered lines. Then all of the $2 \cdot n$ lines of the square have a covered point. Furthermore, we can assume the covered lines to be at the left and at the top of the square, since exchanging lines does not change the number of active lines. An example for this situation in case of $n = 7$ is given in Figure 2. Since about half of the points are covered, there is an uncovered rectangle of about half of the points on the lower right of the square (the uncovered area can be even larger). Furthermore, there are at least as many active horizontal lines as the rectangle's vertical length and at least as many vertical lines as the rectangle's horizontal length. As the sum of horizontal and vertical lines is minimal for a given constant area in case of a square, we can assume that the uncovered rectangle is a square. Since by assumption there are at least $0.5 \cdot n^2 - 1$ uncovered points, the uncovered square has a size of at least

$$\sqrt{0.5n^2 - 1} - 1.$$

[1]This number includes constant nodes.

Since for $n \geq 5$, it holds

$$\frac{1}{2}n^2 - 1 \geq \frac{4}{9}n^2,$$

the size is at least

$$\sqrt{\frac{4}{9}n^2} - 1 = \frac{2}{3}n - 1 \geq \frac{1}{2}n,$$

whereas the last inequation holds for $n \geq 6$. Thus, for $n \geq 6$, the square has size at least $\frac{1}{2}n$. As the number of active lines is double the size, there are at least n active lines. (Asymptotically, there are even $\sqrt{2} \cdot n$ such lines. Furthermore, a more detailed analysis shows that the same property holds for $2 \leq n < 6$.)

Case 4: There are neither horizontal nor vertical covered lines. A computation similar to the previous case shows again the minimum number of n active lines. $\square$

Theorem 1 There exists a family of Boolean functions $f : \mathbf{B}^{n^2} \to \mathbf{B}$ such that any BDD representing the function has size $\Omega(2^n)$, but there is a linear transformation such that the transformed function can be represented by a BDD of size $2 \cdot n + 1$.

Proof: Consider the function $f : \mathbf{B}^{n^2} \to \mathbf{B}$ given by

$$f(x_1, \ldots, x_{n^2}) = \sum_{i=0}^{n-1} \bigoplus_{j=1}^{n} x_{n \cdot i + j} + \sum_{i=1}^{n-1} \bigoplus_{j=0}^{n-1} x_{n \cdot j + i}.$$

The function can be seen as disjunction of $2n - 1$ EXOR chains. Assuming the n^2 variables in an $n \times n$-matrix, these EXOR chains are the EXOR of horizontal or vertical lines:

$$\begin{pmatrix} x_1 & x_2 & \cdots & x_n \\ x_{n+1} & x_{n+2} & & x_{2n} \\ \vdots & & \ddots & \vdots \\ x_{n(n-1)+1} & x_{n(n-1)+2} & \cdots & x_{n^2} \end{pmatrix}.$$

The proof for the lower bound on the resulting BDD size is based on Lemma 1. Let $\omega = 0.5$ and (L, R) be an arbitrary balanced input partition. Then, by Lemma 2 there are at least $n - 1$ active EXOR chains, since the vertical line on the right can not be covered, as the corresponding EXOR chain is missing in f. Thus one of the n chains (which the lemma claims) can be missing. Since satisfiability of one of these EXOR chains is independent of another, the fooling set $\mathbf{F}_{(L,R)}$ has at least size 2^{n-1}. Thus, by Lemma 1, the BDD representing f has at least $\Omega(2^n)$ nodes.

Now we have to show that there is a linear transformation such that the BDD under that transformation has $2 \cdot n + 1$ nodes. A straightforward computation shows that for transformation

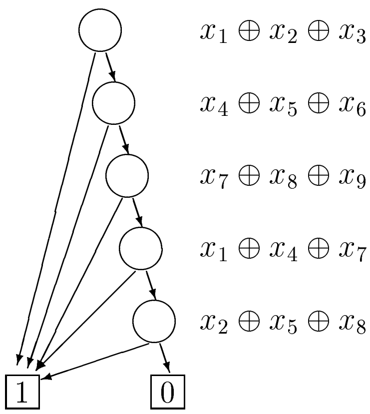

Figure 3: The transformed BDD for $n = 3$

$\tau(x) = (\tau_1(x), \ldots, \tau_{n^2}(x))$ given by

$$\tau_i(x_1, \ldots, x_{n^2}) = \begin{cases} \bigoplus_{j=1}^{n} x_{n\cdot(i-1)+j} & i = 1, \ldots, n \\ \bigoplus_{j=0}^{n-1} x_{n\cdot j+i-n} & i = n+1, \ldots, 2n-1 \\ x_{i-2n+1} & i = 2n, \ldots, n^2, \end{cases}$$

the resulting function is the disjunction of the first $2n - 1$ transformed variables and thus the BDD needs $2n + 1$ nodes (including the constant nodes; see Figure 3 for $n = 3$). $\square$

Thus, there exist functions that can not be handled efficiently using BDDs, while linear transformed BDDs allow a very compact representation.

An experimental evaluation of the function is given in Section 5.

4 Minimization Techniques

In this section methods to minimize BDDs under linear transformations are presented. First, we briefly review a method for exact minimization and then describe linear sifting. Based on this we describe our new heuristic.

4.1 Exact Minimization

The problem of exact minimization of BDDs under linear transformations can be seen as an extension to the problem of finding the exact variable ordering for a given Boolean function, since the set of permutations is a subset to the set of linear transformations. An algorithm for exact minimization has been presented in [8]. It has been shown that similar techniques to

```
window_optimization(f, size) {
    linear_sifting(f);
    for (k = 2; k ≤ size; k++) {
        do {
            mark positions 1 to n − size + 1;
            while there is a marked position p {
                optimize window p, .., p + size − 1;
                if (improvement) {
                    mark positions p − 1 and p + 1;
                }
            }
            sifting(f);
        } while (improvement by sifting);
    }
}
```

Figure 4: The window optimization algorithm

the case of permutations can be used if general linear transformations are decomposed into a permutation and a "normalized" linear transformation. However, due to the huge search space, the algorithm is only applicable to functions with up to 7 input variables.

4.2 Linear Sifting

An effective heuristic approach has been presented in [12]. It can be seen as an extension of sifting [14] by a linear operator: instead of only considering level exchanges, linear sifting also considers the effect of the linear combination of adjacent levels. By this, significant reductions of the BDD sizes are possible, and the algorithm is also applicable to large functions.

In the following we will use a heuristic that is a generalization of linear sifting.

4.3 Window Optimization

The major drawback of linear sifting is that only one variable is considered in one step. For this, in the following we optimize the BDD with respect to linear transformations within a window of m variables exactly using the algorithm from [8][2]. The larger m is chosen the more time

[2] The underlying principle is similar to the one suggested for BDD reordering in [9, 14].

consuming does the algorithm become, but also the quality of the final result improves.

More precisely, for all possible window positions, it is stored whether the position has to be considered. (In the beginning, all positions are to be considered.) If a position has been chosen and the size could be reduced, the neighboring positions are marked as "promising" candidates. This is done while there are positions left. Then, sifting is called[3]. If sifting improves the solution, all window positions are marked as to be considered again. Otherwise, the window size is increased, until a maximum limit is reached. This limit can be given by the user: For large window sizes better results are obtained, but usually windows of only up to 5 variables can be handled within reasonable runtimes.

A sketch of the algorithm is given in Figure 4.

As outlined above, due to the large runtime of the exact method, windows have to be chosen small. However, the size of the window has a large effect on the quality of the results (see Section 5).

If the algorithm is modified to start with the maximum window size, i.e. the loop in Line 3 is omitted, the resulting BDD sizes may become worse, since the heuristic may get stuck in local minima. In general experiments have shown that processing of increasing window sizes has a positive global effect. Furthermore, runtimes are not significantly smaller, since the runtime for the processing of the small window sizes is small: the time for the exact method to process one window increases significantly if the window size becomes larger due to the exponential worst case behavior of the exact algorithm.

5 Experimental Results

In this section we describe experimental results that have been carried out on a *SUN Ultra 1-140*. For all our experiments we used an upper memory limit of 100 MBytes. Our algorithm has been integrated in the CUDD package [15].

In a first series of experiments, we studied the size of the BDD and the linearly transformed BDD for function f from Theorem 1. The results in Table 1 clearly demonstrate the exponential growth for the BDD sizes (in terms of the square root of the number of variables). In contrast the linearly transformed BDD remains very small, i.e. for 100 variables only 20 BDD nodes are needed.

In a next series of experiments we studied benchmark functions from LGSynth91 and ISCAS85. All circuits are minimized using our window optimization algorithm. The results are given in Table 2. In column *circuit*, the name of the circuit is given, for which the number of inputs and

[3]It is also possible to call linear sifting at the beginning to get a good initial solution.

n	in	BDD	linear
3	9	37	6
4	16	149	8
5	25	461	10
6	36	1357	12
7	49	3821	14
8	64	11261	16
9	81	29429	18
10	100	69101	20

Table 1: BDD sizes for function f from Theorem 1

			size=2		size=3		size=4	
circuit	in	out	cost	time	cost	time	cost	time
alu4	14	8	355	1.0	353	2.9	328	71.4
s1196	32	32	718	1.5	613	13.4	613	129.6
c1908	33	25	1182	11.1	1081	41.1	1034	303.6
c432	36	7	1210	1.2	1210	8.6	1210	336.9
c499	41	32	876	59.2	871	58.1	733	563.9
c1355	41	32	887	237.9	702	296.9	602	1429.3
c3540	50	22	23832	119.1	23830	350.8	23376	20932.6
c880	60	26	8449	28.2	8432	76.9	8410	4556.3
dalu	75	16	782	5.2	782	13.9	781	272.5
s1423	91	79	1466	10.9	1456	24.6	1456	423.5
c5315	178	123	1553	75.7	1549	102.1	1512	1269.8
c7552	207	108	4778	216.3	4711	379.2	4676	2403.8
c2670	233	140	2600	134.6	2500	401.2	2472	2162.6
i10	257	224	53676	3720.5	53319	4577.8	53319	31403.2

Table 2: Different window sizes

circuit	in	out	Sifting nodes	time	Linear Sifting nodes	EXOR	time	Window Optimization nodes	EXOR	time
alu4	14	8	429	0.3	411	10	0.5	346	7	2.9
s1196	32	32	598	0.4	706	13	1.0	605	8	13.4
c1908	33	25	6253	4.3	1416	88	9.1	983	98	41.1
c432	36	7	1210	0.3	1210	0	0.6	1210	0	8.6
c499	41	32	26408	23.3	3803	292	43.2	590	281	58.1
c1355	41	32	29562	24.0	12224	161	218.4	458	244	296.9
c3540	50	22	23828	52.2	23813	22	101.6	23811	19	350.8
c880	60	26	8409	7.6	8436	29	21.4	8417	15	76.9
dalu	75	16	829	1.2	780	4	3.2	778	4	13.9
s1423	91	79	1767	3.0	1464	29	5.4	1444	12	24.6
c5315	178	123	1778	4.5	1476	136	39.7	1421	128	102.1
c7552	207	108	7906	23.7	4657	289	134.5	4473	238	379.2
c2670	233	140	3985	8.6	2747	218	22.1	2322	178	401.2
i10	257	224	67791	234.9	53887	120	1905.3	53236	83	4577.8

Table 3: Sifting vs. linear sifting vs. window optimization

outputs are given in columns *in* and *out*, respectively. For each circuit three window sizes are considered. For these experiments we minimized the number of BDD nodes and the number of EXOR nodes needed to carry out the transformation in parallel. (Notice that when using linear transformed BDDs in synthesis also the cost of the transformation function has to be considered.) The *cost* given is the sum of BDD nodes and EXOR nodes. *time* denotes the runtime in CPU seconds. As can be seen, a larger window size often guarantees better results (see e.g. c1355), but can also require much larger runtimes (see e.g. c3540).

In a third series of experiments, we compared the sifting algorithm, the linear sifting algorithm and our window optimization algorithm (see Table 3). For sifting and linear sifting, we used the implementation of the CUDD package [15]. Since linear sifting does not optimize the cost needed for the EXOR realization, from now on we explicitly give both the BDD node count (column *nodes*) and the cost for the realization of the linear transformation (column *EXOR*). For the window optimization algorithm we chose a window size of 3, by which the runtimes remain moderately. It turns out that the results obtained by linear sifting can often be further improved, sometimes even significantly (see e.g. *c1355*), while the runtime overhead is small. Especially, in applications, like synthesis, a reduction of more than a factor of two tremendously improves the

circuit	in	out	Reordering	Linear Sifting [12]	Window Optimization
alu4	14	8	350	350	305
s1196	32	32	598	599	597
c1908	33	25	5526	1822	696
c432	36	7	1064	1209	1064
c499	41	32	25866	520	356
c1355	41	32	25866	484	371
c3540	50	22	23828	23823	23358
c880	60	26	4053	4080	4044
dalu	75	16	689	594	588
s1423	91	79	1796	1477	1365
c5315	178	123	1719	1808	1353
c7552	207	108	2212	1921	1001
c2670	233	140	1774	998	839
i10	257	224	20660	19295	19298

Table 4: Comparison of best-ever values

result.

Finally, we compared the best-ever values from [15] for pure reordering to the best-ever values of linear sifting [12] and to window optimization presented above (see Table 4). Window optimization clearly outperforms the results of both other approaches. In some cases an improvement of nearly a factor of three is possible (see c1908).

All in all, our experiments clearly demonstrate how powerful the new minimization heuristic based on window optimization is. We clearly outperform the best known heuristic and showed that for most functions further minimization is possible.

6 Conclusions

Linear transformations have been shown to be an effective tool for BDD based synthesis. We proved that for a family of functions there is an exponential reduction of BDD size possible. This theoretical result showed for the first time that an exponential improvement can be obtained using linear transformations.

We presented a new heuristic algorithm based on window optimization. Since the window's size can be controlled by the user also the runtime of the algorithm can be adapted dynamically.

It turned out by experiments that the window optimization algorithm could often (with small runtime overhead) significantly reduce the BDD size. Using our new technique improvements of up to a factor of 3 over the best known results could be obtained. Furthermore, in contrast to linear sifting our algorithm also considers the cost of the linear transformation, which is an essential aspect for synthesis.

It is focus of current work to integrate the algorithm in a BDD based synthesis and verification system.

References

[1] D.P. Appenzeller and A. Kuehlmann. Formal verification of a PowerPC microprocessor. In *Int'l Conf. on Comp. Design*, pages 79–84, 1995.

[2] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[3] R.E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Trans. on Comp.*, 40:205–213, 1991.

[4] R.E. Bryant. Binary decision diagrams and beyond: Enabeling techniques for formal verification. In *Int'l Conf. on CAD*, pages 236–243, 1995.

[5] P. Buch, A. Narayan, A.R. Newton, and A.L. Sangiovanni-Vincentelli. Logic synthesis for large pass transistor circuits. In *Int'l Conf. on CAD*, pages 663–670, 1997.

[6] R. Drechsler, N. Drechsler, and W. Günther. Fast exact minimization of BDDs. In *Design Automation Conf.*, 1998.

[7] M. Fujita, Y. Matsunaga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level synthesis. In *European Conf. on Design Automation*, pages 50–54, 1991.

[8] W. Günther and R. Drechsler. Linear transformations and exact minimization of BDDs. In *Great Lakes Symp. VLSI*, pages 325–330, 1998.

[9] N. Ishiura, H. Sawada, and S. Yajima. Minimization of binary decision diagrams based on exchange of variables. In *Int'l Conf. on CAD*, pages 472–475, 1991.

[10] L. Lavagno, P. McGeer, A. Saldanha, and A.L. Sangiovanni-Vincentelli. Timed shannon circuits: A power-efficient design style and synthesis tool. In *Design Automation Conf.*, pages 254–260, 1995.

[11] S. Malik, A.R. Wang, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Int'l Conf. on CAD*, pages 6–9, 1988.

[12] C. Meinel, F. Somenzi, and T. Theobald. Linear sifting of decision diagrams. In *Design Automation Conf.*, pages 202–207, 1997.

[13] C. Meinel and T. Theobald. Local encoding transformations for optimizing OBDD-representations of finite state machines. In *FMCAD*, volume 1166, pages 404–418, 1996.

[14] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.

[15] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.1.2.* University of Colorado at Boulder, 1997.