Linear Transformations and Exact Minimization of BDDs

Wolfgang Günther Rolf Drechsler

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

Abstract

We present an exact algorithm to find an optimal linear transformation for the variables of a Boolean function to minimize its corresponding ordered Binary Decision Diagram (BDD). To prune the huge search space, techniques known from algorithms for finding the optimal variable ordering are used. This BDD minimization finds direct application in FPGA design. We give experimental results for a large variety of circuits to show the efficiency of our approach.

1 Introduction

Spectral methods have been suggested as an efficient tool for circuit design in the early 1960s. Recently, they are of growing interest, since methods have been proposed that allow to efficiently identify good solutions [13, 9]. Spectral transformations are based on a transformation of the whole function table, which has 2^n elements.

Linear transformations can be seen as a restriction of this, as only n variables are transformed. Instead of operating with $2^n \times 2^n$-matrices, we only have to deal with $n \times n$-matrices. An application of linear transformations to BDD minimization is presented in [10], where linear combinations of single variables are dynamically applied. This is integrated in the widely used sifting algorithm [15] for BDD minimization. The resulting BDDs are often much smaller than sifting alone. This gain can be up to 98%. The influence of this approach on the resulting area of the FPGA design has been described in [11]. Nevertheless, the approaches presented so far are purely heuristical and cannot give any guarantees on the quality of the result.

Several design methods based on BDDs have been proposed, and recently, first promising results on transforming BDDs directly to pass transistor logic have been reported [5, 1]. There, the size of the resulting circuit directly depends on the size of the BDD. It is therefore desirable to minimize the size of the BDD. Notice that in these applications a small gain counted in the number of nodes can tremendously simplify the mapping to FPGAs or a target architecture [2, 14].

In this paper, we present the first algorithm to exactly find the optimal linear transformation, i.e. a linear transformation for which the size of the BDD is minimal with respect to the number of nodes. As reordering can be seen as a subset of linear transformations, our approach is more powerful than reordering alone. However, we make use of reordering techniques, since they allow us to prune large parts of the search space without loss of exactness. The linear transformations can easily be realized by EXOR gates. For FPGA synthesis the number of necessary EXOR gates for the transformation is also of importance, since they have to be mapped, too. Our algorithm is also applicable when we give a "penalty" for each EXOR gate used in the transformation.

We carried out experiments that show that we are able to succeed with all benchmarks from LGSynth93 with up to 6 input variables. If we make use of the penalty function we are even able to compute the exact result for functions with up to 7 input variables, since the branch&bound argument becomes stronger. So for most FPGA applications our algorithm can successfully calculate the exact solution, since often cells in FPGAs have less inputs.

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.

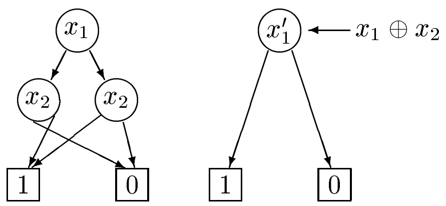

Figure 1: BDDs of the function $f = \overline{x}_1 x_2 + x_1 \overline{x}_2$

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 [3]. 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 *For $i \neq j$, an elementary transformation $\sigma_{ij} : \mathbf{B}^n \to \mathbf{B}^n$ is an automorphism of type*

$$\sigma_{ij}(x_1, \ldots, x_n) = (x_1, \ldots, x_{i-1}, x_i \oplus x_j, x_{i+1}, \ldots, x_n).$$

An elementary transformation can be written as an $n \times n$-matrix over the Galois field $(\mathbf{B}, \oplus, \cdot)$:

$$\sigma_{ij}(x_1, \ldots, x_n) = \begin{array}{cc} & \begin{array}{ccc} & & j \end{array} \\ i & \begin{pmatrix} 1 & & 0 \\ & \ddots \; 1 & \\ 0 & \ddots & 1 \end{pmatrix} \end{array} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

(All non-given matrix elements are 0.)

Linear transformations are those automorphisms obtained by a sequence of elementary transformations [10]. This is analogously done to the generation of regular matrices out of elementary matrices. In the following we do not distinguish between linear transformations and regular matrices.

As shown in [12], 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 $O(2^n)$.

In this paper, we denote a permutation of the variables with π. If $x_i = \pi(k)$ for a variable x_i, then x_i is the kth element of the variable ordering π, i.e. x_i is in the kth *level* of the BDD. For a set of variables $I \subseteq X_n$, let $\Pi(I)$ be the set of permutations π that map the variables from I to I. (We also write $\pi(I) = I$.)

For linear transformations τ_1 and τ_2, let $\tau_2 \circ \tau_1$ denote the concatenation of τ_1 and τ_2, i.e. the linear transformation $(\tau_2 \circ \tau_1)(x_1, .., x_n) = \tau_2(\tau_1(x_1, .., x_n))$. Identifying linear transformations and regular matrices, this can be seen as a multiplication of two regular matrices.

The number of nodes in a BDD for function f and linear transformation τ is given by $\#nodes(f, \tau)$. The nodes in level k marked with variable x_i under linear transformation τ are given by $\#nodes_{x_i}(f, \pi)$.

We now consider the following problem:
How can we determine an optimal linear transformation such that the number of BDD nodes is minimal?

As mentioned in the introduction it is straightforward to realize the linear transformations by EXOR gates. For this, these gates have also to be considered during the minimization process. Depending on the design style or FPGA type a different penalty should be given. (For simplicity of the presentation we do not consider this penealty during the description of our algorithm. But notice that it can be incorporated without any problems.) In our experiments we show how the choice of the penalty influences the result (see Section 6).

3 Optimal Variable Ordering

Our algorithm is based on techniques of exact minimization by finding the optimal variable ordering. In [7] an exact algorithm has been presented where the number of variable orderings which have to be considered could be reduced significantly in comparison with the trivial idea of constructing BDDs for all $n!$ variable orderings. The main idea behind that algorithm is the following lemma:

Lemma 1 *Let $f : \mathbf{B}^n \to \mathbf{B}^m$, $I \subseteq X_n$, $k = |I|$, and $x_i \in I$. Then there exists a constant c such that $\#nodes_{x_i}(f, \pi) = c$ for each $\pi \in \Pi(I)$ with $\pi(k) = x_i$.*

More informally the lemma states that the number of nodes in a level is constant, if the ordering of the variables above that level or below that level is changed.

The lemma can be used for variable reordering as follows: for $I \subseteq X_n$, let

$$\mathit{min\_cost}_I = \min_{\pi \in \Pi(I)} \sum_{i=1}^{|I|} \#nodes_{x_i}(f, \pi),$$

and let π_I denote some permutation leading to that minimum. Assume for a fixed $I \subseteq X_n$ with $|I| = k$ that we know $\mathit{min\_cost}_{I'}$ for all $I' \subseteq I$ with $|I'| = k-1$. Then we get

$$\mathit{min\_cost}_I = \min_{x_i \in I}(\mathit{min\_cost}_{I \setminus \{x_i\}} + \#nodes_{x_i}),$$

with $\#nodes_{x_i}$ being the number of nodes in level k under some permutation π with $\pi(I \setminus \{x_i\}) = I \setminus \{x_i\}$ and $\pi(k) = x_i$. So the optimal ordering can be computed iteratively by computing $\mathit{min\_cost}_I$ for each k-element subset I for increasing k's, until $k = n$.

Remark 1 *For increasing k the considered level k moves down. Likewise, one can start at the top level and consider the levels upwards. As $\Pi(I) = \Pi(X_n \setminus I)$, Lemma 1 can be used the same way in that case. Actually, this is done by all previously presented algorithms except [6] (for a discussion see [6] and Subsection 5.2).*

4 Linear Transformations

As in the case of permutations, the first approach to exact minimization would be to construct BDDs for all linear transformations. Obviously, this approach is only applicable to "tiny" functions. In the following we will prove that some parts of the search space do not have to be considered.

To keep this paper self-contained, we give some theory about linear transformations.

Lemma 2 *For any linear transformation τ, there exists a permutation π and a linear transformation ϱ so that $\tau = \pi \circ \varrho$ and for matrix $R = (r_{ij})_{i,j}$ of ϱ holds $r_{i,i} = 1$ for $1 \leq i, j \leq n$. We call ϱ a normalized linear transformation.*

Proof: Let $T = (t_{ij})_{i,j}$ be the $n \times n$-matrix of τ over $\mathbf{B}$. As T is regular, there exists p_1 such that $t_{p_1,1} = 1$ (otherwise, the first column of T would be the vector 0, in contradiction to T being a regular matrix). In other words, there exists a permutation $\pi^{(1)}$ such that $T = \pi^{(1)} \cdot R^{(1)}$ for some $R^{(1)} = (r_{ij}^{(1)})_{i,j}$ with $r_{11}^{(1)} = 1$.

As $R^{(1)}$ is regular, by induction we get $\pi^{(2)}, \ldots, \pi^{(n)}$ and $R^{(n)} = (r_{ij}^{(n)})_{i,j}$ such that $r_{ii}^{(n)} = 1$ for all $1 \leq i \leq n$. For $\pi := \pi^{(1)} \cdot \ldots \cdot \pi^{(n)}$ and $R := R^{(n)}$, the postulated conditions hold. □

As a consequence, instead of constructing BDDs for all linear transformations, we can also construct the BDDs for all permutations and for all normalized linear transformations. A combination of Lemma 1 and the following theorem is the key to our approach.

Theorem 1 *Let $f : \mathbf{B}^n \to \mathbf{B}^m$. Then there exists a constant c such that $\#nodes_{x_i}(f, \varrho) = c$ for each normalized linear transformation ϱ with corresponding matrix $(r_{ij})_{i,j}$ and*

1. *$r_{ii} = 1$ for all $1 \leq i \leq n$,*
2. *$r_{ij} = 0$ for all $i < k$ and $j \geq k$,*
3. *$r_{ij} = 0$ for all $i \geq k$ and $j < k$,*
4. *$r_{ij} = 0$ for all $i > k$ and $j \leq k$,*

i.e. a matrix of type

$$\left(\begin{array}{ccc|cccc}
1 & & * & 0 & \cdots\cdots & & 0\\
& \ddots & & \vdots & & & \vdots\\
* & & 1 & 0 & \cdots\cdots & & 0\\ \hline
0\cdots & & 0 & 1 & * & \cdots & *\\ \hline
\vdots & & 0 & 0 & 1 & & *\\
\vdots & & & \vdots & & \ddots & \\
0\cdots & \cdots & & 0 & * & & 1
\end{array}\right)$$

Proof: A similar technique can be used as in the proof of Lemma 1. Elementary transformations within the levels $k+1, \ldots, n$ do not affect the number of nodes in level k, as these nodes represent the cofactors of the BDD's functions with respect to $x_{k+1}, \ldots, x_n$. Changing the encoding of $x_{k+1}, \ldots, x_n$ does not change the number of different cofactors.

Additionally, applying elementary transformations σ_{ki} with $i > k$ does not change the number of nodes in level k, as their number is independent of their encoding.

Elementary transformations within levels $1, \ldots, k-1$ do not affect the number of nodes in level k, which can be proved similar to Lemma 1. □

This means that we can apply elementary transformations σ_{ij} with $i < k$ and $j < k$ or with $i > k$ and $j \geq k$ without changing the number of nodes in level k.

Furthermore, we have to proof that by this method we traverse the whole search space.

Theorem 2 *Any normalized regular $n \times n$-matrix R can be written in the form*

$$R = U_n \cdot D_n \cdot U_{n-1} \cdot D_{n-1} \cdot \ldots \cdot U_1 \cdot D_1$$

with

$$U_k = \begin{array}{c|cccccc} & & & k & k+1 & & \\ \hline & 1 & & & & & \\ & & \ddots & & & & \\ k & & & 1 & u_{k+1}^{(k)} & \cdots & u_n^{(k)} \\ & & & & \ddots & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{array}$$

and

$$D_k = \begin{array}{c|cccccc} & & & k & & & \\ \hline & 1 & & & & & \\ & & \ddots & & & & \\ k & & & 1 & & & \\ k+1 & & & d_{k+1}^{(k)} & \ddots & & \\ & & & \vdots & & \ddots & \\ & & & d_n^{(k)} & & & 1 \end{array}.$$

(All non-given matrix elements are 0*.)*

The proof makes only use of well known properties of matrix theory. (We leave it out here; it can be found in [8].)

It can easily be seen that all intermediate matrices $U_i \cdot D_i \cdot \ldots \cdot U_n \cdot D_n$ and $D_i \cdot U_{i+1} \cdot D_{i+1} \cdot \ldots \cdot U_n \cdot D_n$ are normalized. So we can prune the search space whenever we get an "un-normalized" matrix. We do not mention this further, as it would complicate the algorithm unnecessarily.

5 Finding the Optimal Linear Transformation

Algorithms for finding the optimal variable ordering are based on iteratively considering all levels of the BDD [7, 6]. Considering one level means calculating the minimum number of nodes for this level using the calculation of the step before. In case of reordering, this has to be done for all k-element subsets $I \subseteq X_n$, with increasing k. Furthermore, in our more complex case this additionally has to consider linear transformations, as shown in the previous section. In this regard, our algorithm can be seen as an extension to reordering algorithms.

The implementation aspects of linear transformations are similar to [10].

5.1 Algorithm

A sketch of the exact algorithm is given in Figure 2. For increasing k, we consider all subsets I of size k and all linear transformations ϱ based on elementary transformations σ_{ij} with $i < k < j$ or $j < k < i$. For each such combination, we store the number of nodes in levels 1 to k (*min_cost*), the best permutation, and the best linear transformation for these levels in a table. We use (I, ϱ) as a hash key for our tables.

The table is initialized with $I = \emptyset$, *min_cost* $= 0$, some initial ordering π and linear transformation ϱ. For any BDD level k (for increasing k) we consider all entries in *table*. Without pruning, this would be all subsets $I \subseteq X_n$ of size k and all linear transformations

```
compute_optimal_linear_transformation (f) {
  init table with linear transformation;
  for all BDD levels k (top→down) {
    next_table := ∅;
    for each entry t in table {
      set t's permutation and linear transformation;
      prune if lower bound is large enough;
      for each variable x_i in lower levels {
        for each linear transformation of x_i
              and lower levels {
          c := actual cost;
          key := key of new permutation /
              linear transformation;
          if (key ∉ next_table or c < next_table[key])
            add to next_table;
        }
      }
    }
  }
  set best permutation and linear transformation;
}
```

Figure 2: Sketch of the algorithm

ϱ based on elementary transformations σ_{ij} with $i < k$ and $j \geq k$ or with $i \geq k$ and $j < k$.

Then the remaining variables x_i which are not in $X_n \setminus I$ are shifted to level k. Notice that this does not change the number of nodes in levels $1, \ldots, k-1$. For each variable x_i in level k, all linear transformations of lower variables x_j with x_i and all linear transformations of x_i with lower variables x_j are set (see Theorem 2). We store the result in *next_table* either if no such element is in it or if the number of nodes in levels $1, \ldots, k$ is less than before.

In the last iteration, i.e. $k = n$, the minimum BDD size over all variable orderings and normalized linear transformations is computed.

5.2 Lower Bound Technique

The lower bound is the smallest number of nodes possible with that permutation and linear transformation in levels $1, \ldots, k$. Obviously, we can not determine the optimal number within reasonable time, so we have to give a lower estimation for that number, which can be computed efficiently.

Lower bounds for the size of BDDs have been proven in [4] using lower bound techniques from VLSI design. These techniques were applicated in an automated way for the first time in [6], giving good estimations for the lower bound in case of "pure" reordering. We do not repeat the exact method here, but it can be easily seen that the same technique can be used in case of linear transformations, too. (For more details see [6].)

name	i/o	ord	lsift	ex	time	MB
cm42a	4/10	20	20	20	1	1
cm82a	5/3	12	8	7	18	1
c17	5/2	7	9	7	15	1
decod	5/16	32	32	32	56	1
majority	5/1	8	6	6	32	1
rd53-hdl	5/3	17	15	10	32	1
cm138a	6/8	18	18	18	2416	26
bcd.div3	4/4	14	15	12	1	1
dc1	4/7	22	23	21	1	1
dekoder	4/7	19	19	17	1	1
newcwp	4/5	9	9	8	1	1
wim	4/7	20	21	18	1	1
bw	5/28	101	108	95	47	1
rd53	5/3	17	15	10	32	1
xor5	5/1	6	4	2	36	1
newbyte	5/8	17	17	17	43	1
p82	5/14	56	58	50	31	1
bench	6/8	89	89	78	7846	46
fout	6/10	119	118	108	9606	46
m1	6/12	44	44	42	1207	11
newapla2	6/7	17	17	17	2344	26
pope.rom	6/48	203	222	197	5276	16
sqr6	6/12	63	63	61	6181	26

Table 1: Experimental results

6 Experimental Results

In this section we describe experimental results that have been carried out on a *SUN Ultra 1-167*. For all our experiments we used an upper memory limit of 100 MBytes. Our algorithm has been integrated in the CUDD package [16].

In a first series of experiments, all EXOR gates that are needed for the realization of the linear transformations are counted as zero. Results for all LGSynth93 benchmarks with less than 7 inputs are given in Table 1. In column "i/o" the number of inputs and outputs is given, respectively. In column "ord", the minimum number of BDD nodes under all variable orderings is given. Results for linear sifting [10] are given in column "lsift", measured for a fixed set of parameters. The minimum number of nodes for all linear transformations is given in column "ex". Columns "time" and "MB" refer to runtime in seconds and memory usage in MB of our exact algorithm, respectively. All times are given in CPU seconds.

It can be seen that in about half of the cases, the resulting BDD sizes for the best linear transformation are sometimes significantly smaller than with exact variable reordering. Linear sifting takes an intermediate place.

In a second series of experiments we compared the minimum number of nodes when giving a "penalty" p for each elementary linear transformation, i.e. each elementary linear transformation costs as much as p additional BDD nodes. The results are given in Table 2. In columns $p = 0$, $p = 1$ and $p = 2$, the sum of BDD size and p times the number of elementary linear transformations is given, respectively. Sharing EXOR gates between the transformations has not been considered.

It can be seen that even with penalties (which is more realistic for synthesis) linear transformations can improve the costs. The higher the penalties the less linear transformations have to be considered during the minimization process. Thus, our algorithm becomes even more efficient with respect to runtime in this case, and larger functions can be handled within our space limitations. In most practical examples where Boolean functions are mapped to FPGA cells, our algorithm finds the optimal result.

7 Conclusions

Linear transformations can be used for minimizing BDDs. In this paper we presented the first exact algorithm for determining the optimal linear transformation for finding the BDD of minimal size. As the search space is much larger than in case of variable reordering alone, it succeeds only for small circuits. We studied the problem from a theoretical and practical point of view. Using a clever pruning technique we are able to compute exact results for all problem instances with less than 6 variables and for several with 7 variables. For most cell oriented synthesis applications, e.g. FPGAs, this is already sufficient. The resulting BDD sizes using linear transformations are often much smaller, as a comparison to a heuristic greedy algorithm has shown.

References

[1] V. Bertacco, S. Minato, P. Verplaetse, L. Benini, and G. De Micheli. Decision diagrams and pass transistor logic synthesis. In *Int'l Workshop on Logic Synth.*, 1997.

[2] S.D. Brown, R.J. Francis, J. Rose, and Z.G. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publisher, 1992.

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

[4] 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.

[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.

name	i/o	ord	$p = 0$	time	$p = 1$	time	$p = 2$	time
cm42a	4/10	20	20	1	20	1	20	1
cm82a	5/3	12	7	18	10	11	12	2
c17	5/2	7	7	15	7	3	7	1
decod	5/16	32	32	56	32	20	32	5
majority	5/1	8	6	32	8	9	8	2
rd53-hdl	5/3	17	10	33	15	21	16	9
cm138a	6/8	18	18	2416	18	53	18	4
5xp1-hdl	7/10	42	—	—	41	17089	42	7914
s27	7/4	10	10	10534	10	346	10	28
z4ml	7/4	17	10	12828	14	6875	17	3891
z4ml-hdl	7/4	17	10	10361	14	5722	17	1710
bcd.div3	4/4	14	12	1	14	1	14	1
dc1	4/7	22	21	1	22	1	22	1
dekoder	4/7	19	17	1	19	1	19	1
newcwp	4/5	9	8	1	9	1	9	1
wim	4/7	20	18	1	19	1	20	1
bw	5/28	101	95	47	99	46	100	44
rd53	5/3	17	10	32	15	21	16	9
xor5	5/1	6	2	36	6	16	6	2
newbyte	5/8	17	17	43	17	5	17	2
p82	5/14	56	50	31	53	28	55	28
bench	6/8	89	78	7846	85	7316	88	7350
fout	6/10	119	108	9606	113	9514	114	9142
m1	6/12	44	42	1207	44	714	44	558
newapla2	6/7	17	17	2344	17	54	17	5
pope.rom	6/48	203	197	5276	200	4222	201	3557
sqr6	6/12	63	61	6181	63	3794	63	1616
5xp1	7/10	42	—	—	40	19922	42	10440
con1	7/2	15	—	—	15	13145	15	1360
Z5xp1	7/10	42	—	—	—	—	42	8102

Table 2: Costs for different penalties

[6] R. Drechsler, N. Göckel, and W. Günther. Fast exact minimization of BDDs. Technical Report 94, Albert-Ludwigs-University, Freiburg, Oct. 1997.

[7] S.J. Friedman and K.J. Supowit. Finding the optimal variable ordering for binary decision diagrams. In *Design Automation Conf.*, pages 348–356, 1987.

[8] W. Günther and R. Drechsler. Linear transformations and exact minimization of BDDs. Technical Report 98, Albert-Ludwigs-University, Freiburg, Dec. 1997.

[9] J. P. Hansen and M. Sekine. Synthesis by spectral translation using boolean decision diagrams. In *Design Automation Conf.*, pages 248–253, June 1996.

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

[11] C. Meinel, F. Somenzi, and T. Theobald. Function decomposition and synthesis using linear sifting. In *ASP Design Automation Conf.*, Feb. 1998.

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

[13] M. Miller. A spectral method for Boolean function matching. In *European Design & Test Conf.*, page 602, 1996.

[14] R. Murgai, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. *Logic Synthesis for Field-Programmable Gate Arrays.* Kluwer Academic Publisher, 1995.

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

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