Library Based Technology Mapping Using Multiple Domain Representations[1]

J. Bullmann* E. Schubert* U. Kebschull+ W. Rosenstiel*

*Universität Tübingen
WSI-TI
Sand 13
72076 Tübingen, Germany

+Forschungszentrum Informatik (FZI)
Department SiM
Haid-und-Neu-Straße 10-14
76131 Karlsuhe, Germany

Abstract

The use of signatures as efficient filters in boolean matching is a crucial step in technology mapping and/or formal verification. In this work we combine well known representations of boolean functions, the binary decision diagrams, with those in the spectral domain, the functional decision diagrams and the equivalence decision diagrams. We obtain signatures of Boolean functions and their variables, which are easy to compute but can reduce the problem of aliases though.

1 Introduction

The test on equality of two Boolean functions is an important problem in logic synthesis. It is expensive for a given the variable correspondence of both functions. If the correspondence is *not* known, this problem is even harder. Suppose variable names have been changed by preprocessing optimization stages and final verification of the optimization results is required. On the other hand, in library based technology mapping a key task is to match intermediate circuit functions against the library of existing cell functions. Such a match-operation involves finding a correspondence of variables and a polarity value for each variable.

The variable correspondence problem can be solved through complete enumeration of the solution space. Considering a Boolean function of n inputs, $n!$ variable correspondences and 2^n polarities have to be tested. Obviously, more sophisticated approaches to this problem are needed. A key to the problem is the use of *filters*. A filter is a procedure that efficiently decides, whether two Boolean functions either *surely differ* or *are equal at certain probability*. Such filters test necessary conditions of equality. A very simple filter is the check on equality of both functions' input numbers. More sophisticated filters are signatures of Boolean functions.

A signature is a reasonably efficient computable characteristic value of a Boolean function. To compare two functions f and g the signatures $S(f)$ and $S(g)$ are compared first. If they differ, the functions are different. If the signatures equal, in some cases the functions themselves equal and in others they differ. Here, a non-identity of two functions has not been revealed by the signature.

One of the basic works in the area of technology mapping via Boolean matching is [8]. Here, Mailhot and De Micheli describe the task of matching an incompletely specified function f (with a don't care set f_{DC}) against a library function g. A variable permutation π and a polarity mapping ϕ are needed, such that one of $f = \phi \circ \pi \circ g$ or $\overline{f} = \phi \circ \pi \circ g$ holds. The problem of tautology test is solved through repeated Shannon decomposition. To shrink the search space, monotony and symmetry of variables are used. The final test on equivalence is performed via compatibility graphs. In [10], Schlichtmann et al. describe a technology mapping on Actel FPGA. Signatures of Boolean functions are applied as filters to reduce computing time. Here, only variable permutations are considered. The investigated signatures are the *function weight* and others derived from it, like *variable cofactor weight*. An important work by Mohnke and Malik is [9]. A BDD based technique to compute variable permutations is complemented with a polarity matching method. The above signatures are revisited and so called *breakup signatures* are introduced. They are defined as the number of minterms having a fixed Hamming distance to a preselected *origo* minterm. The polarity mapping is done using the cofactor signature. Tsai and Marek-Sadowska have introduced signatures based on spectral representations of Boolean functions in [12]. Here, a function weight is defined on π-terms and also π-term based variable weights are derived from it. These signatures are computed using FDD [6, 5].

In the above mentioned papers it turned out that the number of terms in function representations, i. e. the *function weight* [7] and refinements, the *variable weights* are signatures computable with reasonable time effort. Problematic is the weakness of these signatures. The number of aliases on average is too high and so more sophisticated breakup signatures, e. g. the Hamming distance, have to be applied.

In this paper we approach the signature problem using different representations of Boolean functions: these are the sum-of-products, the Reed-Muller expansion, and the equivalence polynomial. We define the function and variable weights on these different representations of the Boolean functions. Additionally, we convolute different representations of one function and perform function and variable weight count in these convolutions. The so gained new signatures yield higher accuracy resulting in less aliases.

2 Background

Consider the Boolean functions $f = (a \wedge b) \vee c$, $g = (c \wedge a) \vee b$, and $h = (a \oplus b) \vee c$. All three of them show a different behaviour under their input variables a, b, and c. Anyway, the structure of functions f and g is the same. *Boolean matching* of two functions is revealing whether they have different structure or not.

To state this more formally, we use the *permutation* of the function's input variables $X = \{x_0, \ldots, x_{n-1}\}$. A bijective function $\pi : X \rightarrow X$ is called a *variable permutation*. Note, that π is no Boolean function but a symbol manipulation function.

Having two functions f and g defined on the variable set X, we say f and g match, if and only if

$$\exists \pi : f(x_0, \ldots, x_{n-1}) = g(\pi(x_0), \ldots, \pi(x_{n-1})).$$

[1] Part of this work was supported by DFG project RO 1030/3-1 "Konzepte eines Entwurfssystems für komplexe Programmierbare Gate Arrays".

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00

The concept of Boolean matching is extendible to the free choice of polarities. To be more precise, we need the notion of the *polarity mapping*. A function $\phi : X \rightarrow X' = \{x, \overline{x} | x \in X\}$ is called a *polarity mapping* if and only if $\forall x \in X : \phi(x) \in \{x, \overline{x}\}$. Function ϕ is like π a symbol manipulating function.

Taking into account both variable permutation and polarity mapping, Boolean matching means finding a pair $\phi \circ \pi$ so that $f = \phi \circ \pi \circ g$. In other words, two Boolean functions f and g defined on X match if and only if

$$\exists \pi, \phi : f(x_0, \ldots, x_{n-1}) = g(\pi(\phi(x_0)), \ldots, \pi(\phi(x_{n-1}))).$$

If such a pair $\phi \circ \pi$ exists, f and g are called π-ϕ-identical and have similar structure. Functions f and g in the example above are π-ϕ-identical with $\phi(x) = x$ and $\pi(a) = c$, $\pi(b) = a$, and $\pi(c) = b$.

Finding out the permutation π and the polarity mapping ϕ is generally a high effort task. The most straightforward approach to this problem is trying out all $n!$ permutations each combined with 2^n different polarity mappings. Obviously, the time consumption of this strategy is not affordable even with small n.

Key to the problem is to define a procedure S^F, that computes a characteristic value $S^F(f)$ of the function f *at reasonable time effort*. This characteristic value needs to be π- and ϕ-independent, i. e. $S^F(f) = S^F(\phi \circ \pi \circ f)$ for arbitrary π and ϕ. Using procedure S^F as a *filter*, the matching of two Boolean functions f and g speeds up considerably: if $S^F(f) \neq S^F(g)$, not a single $\phi \circ \pi$-pair needs to be checked. A value $S^F(f)$ is called a *function signature* of f.

One step further is to define characteristic values $S^V(f, i)$ for each variable x_i of function f. Using these, all permutations π having $\pi(x_i) = x_j$ can be omitted in search, if $S^V(f, i) \neq S^V(g, j)$. As this characteristic value exists for each *variable* in a function, it is called a *variable signature*.

Note that the signatures only give sufficient information to tell that two Boolean functions *do not* have the same structure (or pairs of variables *do not* correspond). The case when $S^F(f) = S^F(g)$ but f and g are not matchable we call an *alias*. Similarly, having $S^V(f, i) = S^V(g, j)$ for a pair of not corresponding variables x_i, x_j is a *variable alias*. The number of aliases a signature shows is a quality measure of a signature; the smaller, the better.

2.1 Boolean function representation

The computation and manipulation of binary decision diagrams (BDD) is described in [1]. A BDD is the binary tree of the coefficients of the two-level sum-of-products (SOP) of a Boolean function, reduced through application of two rules:

- combine isomorphic subtrees,
- eliminate nodes with isomorphic children.

As BDDs are codings of the SOP of a function, they represent the behavioural domain.

An alternative to the SOP is the fixed-polarity Reed-Muller expansion (RME), also known as an exclusive-or polynomial. Functional decision diagrams (FDD) are introduced in [6], their computation and manipulation algorithms are given in [5]. FDD are defined as binary-tree based representations of the coefficients of the fixed-polarity Reed-Muller expansion. Thus they represent Boolean functions in the functional domain. For their reduction, the same rules are applied as with BDD.

The equivalence polynomial (EP) is known as the dual form of the fixed-polarity Reed-Muller expansion [2]. A graph based representation of the equivalence polynomial is the equivalence decision diagram (EDD). Here too, the reduction rules introduced for BDD are applied. In [4, 3] it is shown that the EDD is a third non isomorphic decision diagram.

Regard the example Boolean function $f = \bar{x}_2 x_1 \bar{x}_0 + x_2 \bar{x}_1 x_0 + x_2 \bar{x}_1 \bar{x}_0 + x_2 x_1 \bar{x}_0$ in SOP representation. Its RME is $f = x_2 x_1 \oplus x_2 \oplus x_1 x_0 \oplus x_1$, while the equivalent EP is $f = (x_2 + x_1) \equiv (x_1 + x_0) \equiv x_1 \equiv x_0 \equiv 0$. We now show how to construct a decision diagram for the RME of the given function. First step is to create a *characteristic set* that describes the π-terms of the RME: $M_{RME} = \{\{x_2, x_1\}, \{x_1, x_0\}, \{x_2\}, \{x_1\}\}$. This set can be written in another form as a set of bitstrings: $\{110, 011, 100, 010\}$. In these bitstrings a 1 bit in position i means that variable x_i occurs in the relating π-term. The set of bitstrings can be stored as a decision diagram. Since we consider the RME of the function, the resulting decision diagram is a functional decision diagram, shown in Figure 1.

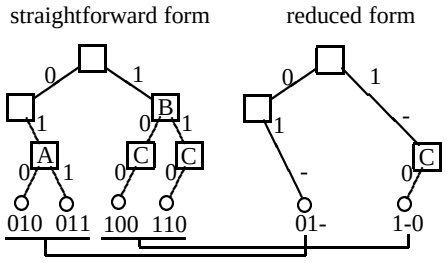

Figure 1: Functional Decision Diagram

In this decision diagram nodes A and B are eliminated as they have identical 0- and 1-subtrees. It can be observed, that subtree C is needed only once after the reduction process.

A decision diagram can be constructed from the SOP-representation in a similar manner. Here the characteristic set is $M_{SOP} = \{\{x_1\}, \{x_2, x_0\}, \{x_2\}, \{x_2, x_1\}\}$. This set is again transformed to a set of bitstrings: $\{010, 101, 100, 110\}$. Here, a 1 bit at position i means that variable x_i occurs positive in the corresponding minterm, a 0 means that it occurs negative. The resulting binary decision diagram is shown in Figure 2.

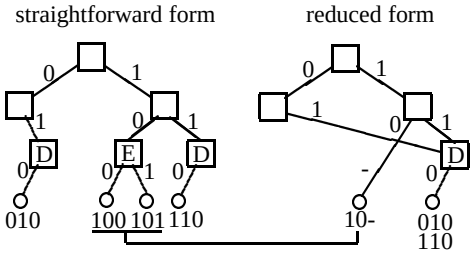

Figure 2: Binary Decision Diagram

In this example binary decision diagram, both reduction rules are used. Subtree D is eliminated once and the remaining one is used two times instead. As the 0- and 1-subtrees of node E are identical, E is eliminated like A and B above.

The construction of the equivalence decision diagram is done analogous to the functional decision diagram. Thus it is left to the reader.

For Boolean function representation we use all three concepts: binary decision diagrams plus functional decision diagrams plus equivalence decision diagrams. They form *multiple domain representations*, since all Boolean functions are represented in behavioural and in the spectral domains at the same time.

3 Behavioural and spectral signatures

Our approach of the signature problem is founded on the function and variable weights as described in related papers. In a first step, we extend these types of signature to the three representation domains SOP, RME, and EP. Considering a Boolean function f, defined on the variable set $X = \{x_0, \ldots, x_{n-1}\}$ we have

$$\begin{aligned} f &= \bigvee_{m \in M_{SOP}} \Big(\bigwedge_{v \in m} v \wedge \bigwedge_{v \notin m} \overline{v} \Big) \qquad \text{(SOP)} \\ &= \bigoplus_{\pi \in M_{RME}} \bigwedge_{v \in \pi} v \qquad \text{(RME)} \\ &= \bigequiv_{\Pi \in M_{EP}} \bigvee_{v \in \Pi} v \qquad \text{(EP).} \end{aligned}$$

For each domain $d \in \{SOP, RME, EP\}$ of function f, there is a unique characteristic set M_d, that describes the function. With this set in mind, we now are able to define *domain specific weights*.

The *domain specific function weight* $W_d^F(f)$ of function f in domain d is the size of f's characteristic set M_d

$$W_d^F(f) = |M_d|.$$

Similarly, the variable weights are defined: the *domain specific variable weight* $W_d^V(f, i)$ of function f's variable x_i in the domain d is

$$W_d^V(f, i) = |\{x | x \in M_d \wedge x_i \in x\}|.$$

Thus, computing the described weights of a function is counting the number of terms in its characteristic sets. As the characteristic sets can grow quite large, we use the efficient decision diagrams described above for their representation.

3.1 Mixing behavioural and spectral representations

The generalized formulas for function and variable weights can be applied not only to the characteristic sets M_d of the representations $d \in \{SOP, RME, EP\}$, but also to combinations of them. We yield six new characteristic sets through the formulas

$$\begin{aligned} M_{SaR} &= M_{SOP} \cap M_{RME} \\ M_{SxR} &= M_{SOP} \cup M_{RME} \setminus M_{SaR} \\ M_{SaE} &= M_{SOP} \cap M_{EP} \\ M_{SxE} &= M_{SOP} \cup M_{EP} \setminus M_{SaE} \\ M_{EaR} &= M_{EP} \cap M_{RME} \\ M_{ExR} &= M_{EP} \cup M_{RME} \setminus M_{EaR}. \end{aligned}$$

These new sets do not have a straightforward meaning like M_{SOP}, M_{RME}, and M_{EP}. We call them *synthetic* sets [11]. They characterize the function f in new ways. Applying the weight formulas defined above to them, we obtain new characteristic values, resulting in new signatures. The computation of the synthetic sets can be easily done using $\oplus$- and $\wedge$-operations between decision diagrams of different types.

A remaining question is: how much redundancy do these synthetic sets contain? Do we need all six of them, or are fewer of them containing the major amount of information? We answer this in section 4.

3.2 Permutation independence

As shown in [10, 5, 12, 3] the weights on the domains SOP, RME, and EP are π-independent. They do not depend on the variable ordering of the function. Essential for signatures, this property has to be proven for the new synthetic weights based on the characteristic sets M_{SaR}, M_{SaE}, M_{EaR}, M_{SxR}, M_{SxE}, and M_{ExR}.

Every permutation π can be written as a sequence $\tau_1 \circ \cdots \circ \tau_m$ of simple permutations, each only swapping two variables. Therefore, it is sufficient to show such a τ-independence of the new signatures. We first prove the τ-independence of W_d^F when applied to one of the characteristic sets introduced above, call it M_d. To do this, we show that the size of M_d does not differ from the size of set M'_d that results from applying τ to each variable in each member of M_d. Suppose τ exchanges the variables x_i and x_j and let us transform M_d memberwise "by hand". For each $x \in M_d$:

1. $x_i \in x$, $x_j \in x$ or $x_i \notin x$, $x_j \notin x$. In this case x is not affected by τ and $x \in M_d \Rightarrow x \in M'_d$.
2. $x_i \in x$, $x_j \notin x$. Replacing x_i by x_j, τ changes x to x'.
   (a) If also $x' \in M_d$, it will be transformed back to x by τ. So in this case, both x and x' belong to M'_d.
   (b) If $x' \notin M_d$, it will be in M'_d replacing $x \in M_d$.

Note that through this construction of M'_d out of M_d the sizes of both sets are equal.

When looking at the variable weight definition $W_d^V(f, i) = |\{x | x \in M_d \wedge x_i \in x\}|$, things seem to be more complicated at first glance. If τ exchanges the variables x_i and x_j, we have to show, that $W_d^V(f, i) = W_d^V(\tau \circ f, j)$ holds. Here, we are reusing the above constructive approach, performing the variable weight count simultaneously on the sets M_d and M'_d. We walk through them and inspect pairs of corresponding members $x \in M_d$ and $x' \in M'_d$.

1. $x_i \in x$. From M'_d's above construction it follows, that $x_j \in x'$, if x' is created from x trough τ. Thus, both x and x' contribute to the variable weight count of the functions f and $\tau \circ f$ respectively.
2. $x_i \notin x$. Then $x_j \notin x'$ and x and x' do not contribute to the variable weight count of the functions.

We are now free to use function and variable weights in combination with all kinds of characteristic sets defined above as signatures.

3.3 Polarity normalization

When variable polarities of a Boolean function are changed, the numbers of terms in spectral representations change. So, our new signatures are not ϕ-independent. To compensate this, when matching two Boolean functions f and g, we first transform their polarities to *pseudo-canonical* forms. Having found such pseudo-canonical or *normalized* polarities for both functions to compare, the second step will be to establish a variable permutation π. Having found this, the Boolean matching operation is complete.

We base the step of variable polarity normalization on the SOP variable and function weights. We define the *normalizing* polarity mapping of a Boolean function f this way:

$$\phi(x_i) = \begin{cases} x_i & \text{if } W_{SOP}^F(f) < 2 \cdot W_{SOP}^V(f, i) \\ \overline{x_i} & \text{if } W_{SOP}^F(f) > 2 \cdot W_{SOP}^V(f, i) \\ * & \text{otherwise.} \end{cases}$$

This easily computable *pseudo-canonical* polarity mapping shrinks the polarity search space considerably in most cases. However,

sometimes there are still undefined polarities left, also called *polarity aliases*. In a function f, the number of polarity aliasing variables is $A^P(f) = |\{i|\phi(x_i) = *\}|$.

After normalization, in function f remain $A^P(f)$ undetermined variable polarities. Trying to match functions f and g, the computation of the variable permutation has to be performed for each of the $2^{A^P(f)+A^P(g)}$ different polarity mappings.

3.4 Variable order normalization

Having solved the polarity matching problem as the first step in Boolean matching, we now focus on finding an admissible permutation, having $f = \pi \circ g$. Instead, we determine two order normalizing permutations, such that $\pi_f \circ f = \pi_g \circ g$ holds. We call π an *order normalizing* permutation of function f if

$$i < j \Rightarrow W_d^V(\pi \circ f, i) < W_d^V(\pi \circ f, j)$$

holds for all $i \neq j$.

We now have the set Π_f of all f's order normalizing permutations. If function f is symmetric in variables x_i and x_j ($f = \tau_{ij} \circ f$ holds), we only need to check one permutation of the pair π and $\pi' = \tau_{ij} \circ \pi$ of Π_f. Thus, one of them may be removed. Using the symmetric variables of the considered function f, Π_f is shrinked.

The number of permutations to be tried out when matching f and g is $|\Pi_f| \cdot |\Pi_g|$.

4 Results

We tested our new signatures in two different ways. First, we computed signatures for each of the 766 functions in the Actel-2 cell library. Here, we were able to obtain *no* aliases! In other words, application of synthetic signatures yield maximum accuracy at moderate computation effort. In Table 1 it is shown, how the results get more accurate, when synthetic signatures are taken into account.

Here the quality q of the signature is measured as the quotient

$$q = \frac{\text{\# of different signatures in the library}}{\text{\# of different functions in the library}}$$

as proposed in [10]. In the rightmost column, the storage size of the computed signatures is given. In the central column, the considered signature types are listed. It can be observed that the overall accuracy value q gets better when more signatures are taken into account. In the lower three lines, the maximum achievable accuracy of 1 is reached.

Table 2 shows that not all signatures based on synthetic sets are needed. Maximum accuracy can be achieved when selecting the subset $\{M_{SOP}, M_{RME}, M_{SxR}, M_{SaE}\}$. We compare this to the results given in [10].

q	considered signature types	size in bit
0.3981	M_{SOP}	(n^2+n)
0.5208	above $+ M_{RME}$	$2\cdot(n^2+n)$
0.5248	above $+ M_{EP}$	$3\cdot(n^2+n)$
0.9686	above $+ M_{SxR}$	$4\cdot(n^2+n)$
0.9986	above $+ M_{SxE}$	$5\cdot(n^2+n)$
0.9986	above $+ M_{ExR}$	$6\cdot(n^2+n)$
1	above $+ M_{SaR}$	$7\cdot(n^2+n)$
1	above $+ M_{SaE}$	$8\cdot(n^2+n)$
1	above $+ M_{EaR}$	$9\cdot(n^2+n)$

Table 1: Increasing signature accuracy in technology mapping

q	considered signature types	size in bit
0.8885	S_{SFP} [10]	$2n^2$
0.9346	S_{MW}, S_{SFP} [10]	$4n^2+n$
1	$M_{SOP}, M_{RME}, M_{SxR}, M_{SaE}$	$4\cdot(n^2+n)$

Table 2: Signature accuracy in technology mapping, compared to Schlichtmann et al.

Moreover, we applied our signatures to benchmarks of the MCNC-93 set. Here, signatures are used to find variable correspondences in pairs of Boolean functions prior to formal verification. The results are shown in Table 3. Even for large functions with many inputs few variable order aliases are observed. So, our signatures perform well in this context too.

The leftmost three columns in the table show the specification of the benchmark. In column four, the average number of variable permutations of each output function is given. This average is computed over all primary output functions in each single benchmark. If this number is greater than 1, aliasing occurred. Otherwise, the signatures yielded maximum possible accuracy. In the next column, the previously defined accuracy value q is given. The last column shows the CPU time in seconds, used to calculate the signature values.

The latter results underline the importance of work in the area of Boolean function signatures. Applying signatures, the time effort of comparing Boolean functions without knowing their variable correspondence can be reduced to a practical measure. Considering even one of the bad results, like e. g. *cordic*, the the gain is dramatic: 12 permutations to try, instead of $23!$ when working without signature filters.

Benchmark	# In	# Out	avg. # Tries	q	CPU time
acc	50	69	1.19	0.840	9.96
al2	16	47	1.00	1	0.15
apex1	45	45	1.09	0.917	8.64
apex3	54	50	1.38	0.724	10.48
b3	32	20	1.10	0.909	1.69
b4	33	23	1.00	1	0.35
bca	26	46	1.00	1	109.24
bcc	26	45	1.00	1	84.81
bcd	26	38	1.03	0.971	32.08
chkn	29	7	1.14	0.877	0.78
cordic	23	2	12.00	0.083	22.19
duke2	22	29	1.48	0.676	0.79
e64	65	65	1.00	1	1.99
ex4	128	28	29.93	0.033	0.73
exep	30	63	1.00	1	2.38
ibm	48	17	1.00	1	0.74
in2	19	10	1.10	0.909	0.45
in3	35	29	1.14	0.877	0.83
in4	32	20	1.10	0.909	1.93
in6	33	23	1.00	1	0.35
in7	26	10	1.00	1	0.24
mark1	20	31	1.00	1	0.41
misg	56	23	15.52	0.064	0.13
mish	94	43	1.00	1	0.09
rckl	32	7	1.00	1	0.88
seq	41	35	1.14	0.877	10.20
shift	19	16	125.94	0.008	0.31
signet	39	8	1.00	1	33.41
spla	16	46	1.74	0.575	0.94
x1dn	27	6	1.00	1	28.45
x6dn	39	5	1.20	0.833	1.52
x7dn	66	15	1.20	0.833	2.54
xparc	41	73	1.05	0.952	14.13

Table 3: Accuracy of signatures in formal verification

5 Conclusion

In this paper, we have introduced new kinds of signatures. They are based on different representations of Boolean functions, the sum-of-products (SOP), Reed-Muller expansion (RME), and equivalence polynomial (EP). Yielding a high accuracy, our signatures are applicable in Boolean matching for both technology mapping, and verification. Testing our approach against a large set of examples, we could show its time and space efficiency.

References

[1] R.E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation . *IEEE Trans. on Comp.*, 35(8), Aug 1986.

[2] D.H. Green. Dual Forms of Reed-Muller expansions. *IEE Proceedings*, 141(3), May 1994.

[3] U. Kebschull. *Verhaltensbasierte und spektrale Logiksynthese mehrstufiger Schaltnetze unter Verwendung von Binärbäumen*, volume Reihe 10 Nr. 335 of *Informatik/Kommunikationstechnik*. Fortschr.-ber. VDE, 1995.

[4] U. Kebschull, J. Bullmann, E. Schubert, and W. Rosenstiel. Darstellungsabhängige Minimierung mehrstufiger Schaltnetze. In *Proc. 1. GI/ITG Workshop Anwenderprogrammierbare Schaltungen*, 1994.

[5] U. Kebschull and W. Rosenstiel. Efficient Graph-Based Computation and Manipulation of Functional Decision Diagrams . In *Proc. EDAC 93*, 1993.

[6] U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel Logic Synthesis Based on Functional Decision Diagrams . In *Proc. EDAC*, 1992.

[7] Y.T. Lai, S. Sastry, and M. Pedram. Boolean Matching using Binary Decision Diagrams with Application to Logic Synthesis and Verification . In *Proc. ICCD*, 1992.

[8] F. Mailhot and G. De Micheli. Technology Mapping Using Boolean Matching and Don't Care Sets. In *Proc. EADC*, 1990.

[9] J. Mohnke and S. Malik. Permutation and Phase Independent Boolean Matching. In *Proc. EDAC, EURO ASIC 93*, Feb. 1993.

[10] U. Schlichtmann, F. Brglez, and M. Hermann. Characterization of Boolean Functions for Rapid Matching in EPGA Technology Mapping. In *Proc. 29th DAC*, 1992.

[11] E. Schubert and W. Rosenstiel. Combined Spectral Techniques for Boolean Matching. In *Proc. ACM Symposium on FPGA, Monterey*, 1996.

[12] C.-C. Tsai and M. Marek-Sadowska. Boolean Matching Using Generalized Reed-Muller Forms. In *Proc. 31st. DAC*, 1994.