

SYNTHESIS OF PSEUDO KRONECKER LATTICE DIAGRAMS

Per Lindgren*

Division of Computer Engineering
Luleå University of Technology
97187 Luleå, Sweden
pln@sm.luth.se

Rolf Drechsler

Bernd Becker

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

ABSTRACT

The design process of digital circuits is often carried out in individual steps, like logic minimization, mapping and routing. This leads to quality loss e.g., in cases where highly optimized netlists fit badly onto the target architecture. Lattice diagrams have been proposed as one possible solution. They offer a regular two dimensional structure, thus overcoming the routing problem. However elegant, presented methods have only shown to find practical lattice representations for small functions.

We present heuristic synthesis methods for Pseudo-Symmetric Pseudo Kronecker Decision Diagrams (PSP-KDDs) applicable to incompletely specified multiple output functions. The lattice structure maps directly to both ASICs and fine grain FPGAs. Our method (combining logic minimization, mapping and routing) seeks to minimize area and delay by heuristic methods. Experimental results on a set of MCNC benchmarks show superior quality to previous methods and in many cases even optimal depth results for unfolded lattices.

1. INTRODUCTION

The design process of digital circuits is usually carried out in steps such as logic minimization, mapping and routing. Over the years, each individual discipline has matured and highly sophisticated tools have been developed. However, it is difficult for tools upstream the design process to foresee all effects to downstream design steps. This leads to quality loss for example in cases where highly optimized netlists fit badly onto target architectures. By restricting synthesis to only consider e.g., PLAs, certain properties can be ensured, such as regular, easily routable layouts with nice testability. However, the expressive power of such 2-level structures is often a limiting factor.

The introduction of *Binary Decision Diagrams* (BDDs) [2] has brought new means to multi-level synthesis. In its simplest form a BDD maps to a MUX based multilevel circuit. Also, more general DDs such as *Functional DDs* (FDDs) [5] and *Pseudo Kronecker DDs* (PKDDs) [8, 6] can be mapped accordingly. The size of the circuit is related to the number of diagram nodes, thus heuristic methods for diagram minimization can be applied. However, the number of nodes at one level is worst case exponential (to the number of levels). This makes the routing of such circuits often cumbersome, resulting in large area and delay overhead.

As one alternative lattice diagrams have been proposed. The number of nodes at each level is linear (to the level) which makes the diagram fit onto a two dimensional structure, such overcoming the routing problem. In the case of totally symmetric functions it is known that e.g. BDDs form such lattice structures. Unfortunately, totally symmetric functions are rare, and we need a way to make functions “pseudo-symmetric” in order to derive lattice structures. This can be

done by re-introducing support variables in similar fashion to the variable repetition of *Universal Akers Arrays* [1]. Akers method, always considering worst case functions, was shown to render exponential lattice depth and was therefore considered infeasible.

Nevertheless, lattice structures have gained renewed interest as they offer properties (e.g. easily routable layout and predictable path length) sought for a number of current technologies. Synthesis methods have been revisited and a number of improvements presented [7, 4, 10, 3]. Recently, practical results have been shown for a number of small circuits.

Our approach using *Pseudo-Symmetric Pseudo Kronecker DDs* (PSPKDDs) [7] combines properties from Pseudo-Symmetric DDs [4] and Pseudo Kronecker DDs [8]. The optimization goal is to minimize the number of levels in the lattice representation of the function. In general this is a complex task only manageable to tackle using heuristic methods as e.g., in [7] considering only certain combination of decomposition types. Our method is different in that we consider all types of decompositions but *only* in such a way that the choice of decomposition type has only a local effect on the lattice. For each node we consider choice of decomposition type, input variable polarity and negation of the interconnections. For the selection of decomposition variables we propose two strategies; pre-processed and dynamic orderings. The latter often is able to produce better results at the cost of run time, while the pre-processing (although very fast) often leads to better results than a naive ordering. Experimental results on a set of MCNC benchmarks show superior quality in comparison to previous methods.

2. PSEUDO-SYMMETRIC DECISION DIAGRAMS

In this section we briefly review previous work on lattice diagrams presented in [1, 7, 4, 3, 10].

Let f_0 (f_1) denote the *cofactor* of f with respect to $\bar{x}(x)$. A Boolean function $f : B^n \rightarrow B$ can then be *Shannon* decomposed as $f = \bar{x}f_0 + xf_1$, where $+$ is the OR operator. To the cofactors f_0 (f_1) we can apply Shannon decomposition until constants $\{0, 1\}$ are reached. The set of decompositions can be modeled by a graph structure called *Binary Decision Diagram* (BDD). In the following we also assume BDDs to be *ordered* and *reduced*.

It is easily shown that BDDs for totally symmetric functions form lattice structures. Unfortunately, in a BDD representation for other functions, the number of required nodes at one level is (worst case) exponential to the level [2]. However, for many practical functions a good variable order reduces the required number of nodes. Even if the node count is decreased there is no guarantee that the diagram will map to a two dimensional structure. To address this problem *Pseudo-Symmetric BDDs* (PSBDDs) have been proposed [4]. Assume the (sub)function f and the cofactors f_{00}, f_{01}, f_{10} and f_{11} w.r.t. two arbitrary variables a and b which f is *not* symmetric in. Let $f_{join} = bf_{01} + \bar{b}f_{10}$.

*The co-operative work was conducted at Albert-Ludwigs-University on a grant from the Swedish Institute.

$$\begin{aligned}
f &= \bar{a}\bar{b}f_{00} + \bar{a}bf_{01} + a\bar{b}f_{10} + abf_{11} \\
f &= \bar{a}\bar{b}f_{00} + \bar{a}bf_{join} + a\bar{b}f_{join} + abf_{11} \\
f &= \bar{a}\bar{b}f_{00} + (\bar{a}b + a\bar{b})f_{join} + abf_{11}
\end{aligned}$$

By the re-introduction of variable b , the diagram is made “pseudo-symmetric” as to fit onto a two dimensional lattice structure. The functions produced contain at least one less occurrence of the re-introduced variable. This ensures that a simple synthesis algorithm will terminate (however, rendering exponential number of lattice levels [1]). Note, that from the variable assignments (path functions) in the lattice additional *Don't Care* (DC) information can be directly derived. Let S be the union of all path functions (cubes) for a node n . The additional DC set for n is \bar{S} . E.g. the sub-function f_{11} is evaluated as abf_{11} (i.e., $S = ab$). The complement of the path function $\bar{S} = \overline{ab} = \bar{a} + \bar{b}$ can be added to the DC set for f_{11} .

Pseudo-Symmetric Functional DDs (PSFDDs) combine the properties of PSBDDs and FDDs. Let f_0 (f_1) denote the cofactor of f with respect to $\bar{x}(x)$ and f_2 is defined as $f_2 = f_0 \oplus f_1$, \oplus being the Exclusive OR operation. A Boolean function $f : B^n \rightarrow B$ can then be represented by a positive (negative) Davio decomposition, $f = f_0 \oplus xf_2$ ($f = f_1 \oplus \bar{x}f_2$). An FDD is a graph representation of the set of decompositions applied. We assume the FDD to be ordered and reduced. Without loss of generality we only consider positive Davio FDDs in the following. Assume the function f represented by an FDD (defining a positive polarity Reed Muller expression): $f = f_{00} \oplus bf_{02} \oplus af_{20} \oplus abf_{22}$. In order to make the diagram “pseudo-symmetric”, we set $n_4 = f_{00}$, $n_5 = bf_{02} \oplus \bar{b}f_{20}$ (by a *Join-EXOR*[3] operation). To make the evaluation of the diagram consistent a residue function has to be applied, resulting in $n_6 = f_{02} \oplus f_{20} \oplus f_{22}$.

$$\begin{aligned}
f &= f_{00} \oplus (b \oplus a)(bf_{02} \oplus \bar{b}f_{20}) \oplus ab(f_{22} \oplus f_{02} \oplus f_{20}) \\
f &= f_{00} \oplus bf_{02} \oplus abf_{02} \oplus \bar{a}bf_{20} \oplus ab(f_{22} \oplus f_{02} \oplus f_{20}) \\
f &= f_{00} \oplus bf_{02} \oplus af_{20} \oplus abf_{22}
\end{aligned}$$

The procedure, propagating the residue, is carried out from left to right (or right to left for negative Davio FDDs) for each level. The possibility to extract DC information is reduced compared to that of PSBDDs, as path functions contain less literals. E.g., the additional DC set for n_5 is $\overline{a + b} = \bar{a}\bar{b}$.

The basic PSBDD structure described above utilizes only one type of symmetry, i.e., $f_{01} = f_{10}$. An extended set of symmetries $\{(f_{00} = f_{01}), (f_{00} = f_{10}), (f_{00} = f_{11}), (f_{01} = f_{10}), (f_{01} = f_{11}), (f_{10} = f_{11})\}$ can be exploited if the polarity of the decomposition (control) variable is assigned for each node. If we also consider complementation of the cofactors for each node, the number of detectable symmetries is doubled, to a total of 12 [10].

3. LATTICE SYNTHESIS METHOD

This section describes the lattice structure in consideration. We outline a comprehensive synthesis methodology and give in depth description of the heuristics involved.

3.1. Pseudo-Symmetric Pseudo Kronecker DDs

According to previous definitions, let f_0 (f_1) denote the cofactor of f with respect to $\bar{x}(x)$ and f_2 is defined as $f_2 = f_0 \oplus f_1$, \oplus being the Exclusive OR operation. A Boolean function $f : B^n \rightarrow B$ can then be represented by one of the following formulae:

$$f = \bar{x}f_0 \oplus xf_1 \quad \text{Shannon } (S) \quad (1)$$

$$f = f_0 \oplus xf_2 \quad \text{positive Davio } (pD) \quad (2)$$

$$f = f_1 \oplus \bar{x}f_2 \quad \text{negative Davio } (nD) \quad (3)$$

If we apply to a function f either S , pD or nD we get two sub-functions. To each sub-function again S , pD or nD can be applied. This is done until constant functions are reached. A *Pseudo Kronecker DD* (PKDD) is a graph representation of the decompositions applied. As for BDDs and FDDs we consider PKDDs to be ordered and reduced w.r.t. decomposition type. PSPKDDs combine the properties of PKDDs and Pseudo-Symmetric lattice diagrams [7]. A general process to derive PSPKDDs is complicated as decomposition type for each node is individual and residues must be allowed to propagate in *both* directions in contrast to Section 2, where the decomposition type is fixed for each level. Furthermore, the set of possible symmetries is vastly enlarged. Even for small functions the search space is huge. In [7] it was observed that by limiting the decomposition types to a combination of $\{S, pD\}$, residues are limited to propagation in one direction allowing a similar synthesis method as for PSFDDs. Our method considers all decomposition types $\{S, pD, nD\}$ under the condition that dual propagation can be avoided (see Section 3.2).

3.2. Synthesis Methodology

We do not consider the entire search space for PSPKDDs. Instead we suggest a heuristic method to guide the search towards good solutions. As the main optimization criterion, we have chosen the number of lattice levels required to represent the function.

Before getting “knee deep” in details we give a brief outline of our synthesis method.

1. Choose decomposition variable.
2. Traverse current level from left to right, for each node determine decomposition type, polarity of the input, and complementation of the cofactors such that the node becomes symmetric or “pseudo-symmetric” with its left neighbor.
3. Increase level, repeat from 1. until all cofactors are constant.

The lattice cells are stored in a matrix LA (directly corresponding to the physical ASIC layout or fine grain FPGA.) The i -th node at level j can be found at $LA[j - i, i - 1]$ in the matrix, (it's left neighbor is the $(i - 1)$ -th node etc.). The synthesis process starts at level 1 ($LA[0, 0]$) for *Single Output* (SO) functions and level m ($LA[m - 1, 0]$) in the case of *Multiple Output* (MO) functions (see Section 3.5).

3.3. Lattice Node Synthesis

The local optimization criteria is twofold: firstly to seek constant cofactors (such to terminate a part of the diagram) and secondly to avoid re-introducing support variables to the sub functions (which helps to reduce the number of required levels).

The node synthesis algorithm starts by computing the cofactors f_0 , f_1 and f_2 w.r.t. the decomposition variable x . In the search for constant cofactors, the DC set assigned to the node is utilized, i.e., $f^{ON} + f^{DC} = 1$ yields a constant 1 function. If two of the cofactors are constant, the function can be captured by a single lattice node (and this part of the diagram is terminated). In other cases we compute a cost estimate for the cofactors f_0^{est} , f_1^{est} and f_2^{est} . As computationally efficient cost estimate we measure the number of nodes for a BDD representation (ON set) under the current variable order (see below).

One can divide the node selection algorithm into two fundamentally different cases.

- (A) the case when the node n has *no* left neighbor or the case when the left neighbor does not connect to a right successor.
- (B) the case where we have a left neighbor, connecting to a right successor (having the function f_{join}).

In the first case we have unlimited freedom to choose decomposition type as we do not have to apply any join operation at this stage. From the cost estimates of the cofactors, the two least costly are chosen. If the decomposition variable is not in the support of f , the node is essentially an *extender* to the left successor. If one of the cofactors is constant, we assign the constant to the right successor, and choose decomposition by the estimated cost for the other cofactors. This is to ensure that the right neighbor will be of type (A) thus have maximum flexibility. In other case, we choose decomposition from the estimated cost. If a Davio node is chosen, we assign f_2 to the right successor (which we later show to simplify the search for symmetries).

The second case is somewhat more delicate, since we have to ensure that neighboring nodes are symmetric or “pseudo-symmetric” to fit the lattice structure. First we check if f_0 or f_1 can be unified to f_{join} or \bar{f}_{join} and in such case choose a Shannon decomposition accordingly. In this, we exploit the set of extended symmetries. To choose input variable polarity and negation of the cofactors (successors).

If no such symmetry can be directly found, we check whether one of the cofactors $\{f_0, f_1\}$ is constant. If found, we assign the constant to the left successor (such avoiding to apply a *join* operation and the re-introduction of a support variable) and the non-constant cofactor to the right successor. If none of these requirements are met, we have to apply a *join* operation. To avoid invoking a residue function, we consistently use Shannon decomposition. This ensures the right subfunction always to be associated with a literal, unless constant. The theoretical motivation to favor Shannon nodes in *join* operations is that the DC set grows faster than for Davio nodes.

3.4. Variable Ordering Heuristics

The ordering in which the decompositions are applied heavily affects the depth of the corresponding lattice. Our synthesis method does not inflict any ordering restrictions. Our two most promising ordering methods are both based on the simple restriction that the decomposition variable chosen, will also be re-introduced until it does not occur in the support of any subfunction. The techniques discussed are (A) pre-processed ordering and (B) dynamic reordering. The estimation function used in the node synthesis algorithm, measures the number of nodes required for a BDD representation.

(A) As applied in a pre-processing step, we found that the *Group Sifting* gave the best results, in cases vastly improving on the initial (naive) ordering. The *Group Sifting* keeps symmetric (or nearly symmetric) variables together during the sifting procedure, which clearly is helpful towards finding symmetries in the diagram (see previous section and [10]).

However, as applied in a pre-processing step, symmetries might be changed during the synthesis and the ordering no more plausible.

(B) To address the shortcoming of method (A) we propose a dynamic re-ordering method. The idea is simple: Choose a remaining support variable as decomposition variable. Compute the corresponding lattice. Store best result, continue with next variable until all variables tried.

The procedure is accelerated by aborting the lattice computation as soon as current level equals best level so far obtained. By using the pre-ordering (A), a good initial solution is often found making the pruning even more efficient. Experimental results show equal or superior results (to naive/pre-ordering) in all cases, however still at the cost of extensive CPU time.

3.5. Multiple Output Functions

In [7] a framework of lattice diagrams also applicable to MO functions was outlined. The method is elegant, as the syn-

Table 1: PSBDD vs. PSPKDD.

name			PSBDD	PSPKDD	
	out	in		const	est
apex7	37(3)	49(37)	55	-	43
b9	21(0)	41(11)	17	14	11*
cm85a	3(2)	11(10)	12	10*	10*
cordic	2(1)	23(23)	51	37	55
dalu	16(2)	75(47)	103	-	-
frg2	139(59)	143(22)	37	57	26
t481	1(0)	16(16)	147	57	19
term1	10(9)	34(18)	74	19	18*
ttt2	21(15)	24(14)	18	15	14*
x1	35(10)	51(25)	38	97	36
vda	39(0)	17(15)	61	78	31

thesis algorithm does not have to be re-tailored. Simply, for an m output function F , let the functions $f^1 \dots f^m$ appear at the m -th level of the lattice and apply the synthesis process previously described. However, we observe that the synthesis results are strongly dependent on the ordering (grouping) of the output functions. We propose a simple heuristic to address this problem. Let $O_1 \dots O_m$ be the lattice functions at level m , all initially set to constant 0:

Compute the cost of each output function. Set O_1 to be the cheapest. Let $i = 2$. Set O_i to one of the remaining output functions. Compute the cost (lattice depth) of $O_1 \dots O_i$ and store best result. Repeat previous step until all remaining output functions are chosen. Recall the best result. Increase i . Choose next output until $i = m$.

The method greedily chooses the right neighbor that fits best. Previously best results can be used to prune the computation and improve CPU time. As our simple heuristic lacks backtracking facility, the quality cannot be guaranteed. However, our experimental results in the next section, show that the heuristic often significantly improves on the naive grouping.

4. EXPERIMENTAL RESULTS

In this section we present experimental results for a set of MCNC PLA benchmark functions performed on a *Sun Ultra 1* workstation with 256Mb RAM. For the implementation we used the CUDD 2.2.0 BDD-package [9]. For the experiments we set a hard limit to 300 levels for the PSPKDDs. CPU times are measured in seconds. Numbers in parenthesis denote considered output function and actual number of support variables. Symmetric functions are marked by “s”. “-” indicates that the result could not be obtained within the hard limit set. “*” indicates that the result is optimal for unfolded lattices, i.e., the number of input variables is equal to the depth.

In a first set of experiments we give a comparison to PSBDDs (see Table 1). Column *PSBDD* shows the results from [10] without the “merging” discussed above. Our results are obtained using the synthesis method from Section 3.2 with the dynamic variable ordering approach. We could not produce results for the “dalu” benchmark as intermediate results broke the hard limit. The two columns “const” and “est” show our results assigning Davio nodes only in cases directly inflecting constant cofactors and the assignment from the proposed estimation function respectively. Except for “cordic” the estimation leads to better, optimal or near optimal solutions.

In a second set of experiments we compare to PSFDDs (see Table 2). Columns PSFDD “re” and “org” show the results from [3] with and without their reordering method applied. Columns PSPKDD show our results using the synthesis method from Section 3.2 and Davio nodes are chosen by cost estimation. The effect of pre- and dynamic variable ordering are shown in columns “pre” and “dyn” respectively. Our proposed dynamic reordering method shows superiority to best known results and often optimal solutions. The pre-ordering, although extremely fast, achieves results that in

Table 2: PSFDD vs. PSPKDD.

name	out	in	PSFDD		PSPKDD			CPU		
			org	re	org	pre	dyn	org	pre	dyn
b9	21(2)	41(9)	12	16	10	10	9*	0.03	0.03	0.68
	21(12)	41(9)	15	20	13	9*	9*	0.03	0.03	0.68
	21(20)	41(9)	19	16	17	9*	9*	0.03	0.02	0.68
c8	18(17)	28(11)	26	11*	11*	11*	11*	0.02	0.02	0.97
cht	36(28)	47(5)	7	9	5*	5*	5*	0.02	0.02	0.24
cm162a	5(2)	14(10)	12	11	10*	11	10*	0.02	0.03	0.81
cm163a	5(3)	16(9)	10	11	9*	9*	9*	0.02	0.02	0.66
count	16(0)	35(5)	6	8	5*	5*	5*	0.02	0.02	0.24
example2	16(3)	35(8)	19	13	9	8*	8*	0.03	0.02	0.54
	66(2)	85(6)	-	8	6*	6*	6*	0.02	0.02	0.33
mux	66(23)	85(5)	-	7	5*	5*	5*	0.03	0.02	0.24
	1(0)	21(21)	-	107	88	32	31	6.72	0.06	49.27
pcl	9(6)	19(10)	12	12	10*	10*	10*	0.02	0.02	0.82
sct	15(4)	19(7)	9	10	8	8	7*	0.02	0.02	0.43
	15(5)	19(8)	12	11	9	8*	8*	0.02	0.02	0.54
	15(6)	19(9)	14	12	10	9*	9*	0.03	0.02	0.67
term1	15(7)	19(10)	17	13	11	10*	10*	0.03	0.02	0.82
	15(8)	19(11)	20	14	12	11*	11*	0.03	0.02	0.98
	10(4)	34(17)	79	79	107	20	20	0.51	0.05	2.61
	10(5)	34(18)	144	144	109	21	21	0.62	0.06	2.98
	10(6)	34(19)	174	174	184	31	22	2.20	0.09	3.36
	7(6)	10(10)	21	13	14	11	11	0.03	0.03	0.82
x2	71(60)	94(7)	15	20	9	7*	7*	0.03	0.02	0.43
x4	21(7)	24(5)	8	9	5*	5*	5*	0.02	0.02	0.24
ttt2	21(10)	24(7)	8	8	7*	7*	7*	0.02	0.02	0.42
	21(15)	24(14)	110	33	14*	34	14*	0.03	0.09	1.55
	21(18)	24(7)	22	16	7*	10	7*	0.02	0.03	0.43

Table 3: Grouping of outputs for PSPKDDs.

name	out	in	Org		Greedy	
			lev	CPU	lev	CPU
5xp1	10	7	83	1.2	27	19.7
add6	7	12	18	1.8	18	31.9
co14 ^s	14	1	14	1.5	14	1.5
in2	10	19	291	51.9	195	758.8
in7	10	26	63	9.1	57	210.3
inc	9	7	35	0.6	33	18.6
rd53 ^s	3	5	14	0.3	14	1.6
rd73 ^s	3	7	16	0.5	19	2.9
rd84 ^s	4	8	21	0.6	24	6.0
sao2	4	10	43	1.2	41	10.2
t481	1	16	19	2.0	19	2.0

cases parallel that of our dynamic method, e.g., the “mux” benchmark having only one more level while being almost 1000 times faster to calculate. Often the pre-reordering also improves on the run time (compared to the naive ordering) as a much smaller lattice is constructed, e.g., “term1” benchmark where CPU requirements are reduced by a factor 10 to 20. Both our ordering methods show overall consistency and the results outperform those previously presented.

In [7] concepts for a framework of lattice diagrams also applicable to MO functions were outlined. However, the literature still lacks experimental results. In the last set of experiments we show our first results applying our synthesis method to a set of benchmarks containing MO functions, see Table 3. Davio nodes are chosen by cost estimation and the dynamic approach is used for the variable ordering. Columns *Org* and *Greedy* show the results using the original (naive) grouping and the results obtained using the greedy heuristic approach. The number of levels (“lev”) shown is the total lattice depth, thus the number of active levels is “lev” - (*out* - 1). For larger functions results tend to become unpractical. However, the greedy heuristic often succeeds to reduce the overall lattice depth (with up to 67 % for “5xp1”).

5. CONCLUSIONS AND OPEN PROBLEMS

We have proposed a comprehensive synthesis method for PSPKDDs combining logic minimization, mapping and routing. Heuristic methods are developed to minimize the over-

all PSPKDD depth. Results on a set of MCNC benchmarks show superior quality to previous approaches and first results for multiple output functions are presented.

Target for future research is an improved grouping algorithm. An alternative approach is to partition the lattice. Functions having similar support are minimized together building a group, later to be globally placed and routed on the lattice. This method preserves local routability and might be applicable to functions infeasible to map by our current method.

6. REFERENCES

- [1] S.B. Akers. A rectangular logic array. *IEEE Trans. on Comp.*, C-21:848–857, 1972.
- [2] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.
- [3] M. Chrzanowska-Jeske, X.Y. Ma, and W. Wang. Pseudo-symmetric functional decision diagrams. In *Int'l Symp. Circ. and Systems*, pages VI:175–VI:178, 1998.
- [4] M. Chrzanowska-Jeske, Z. Wang, and Y. Xu. A regular representation for mapping to fine-grain, locally-connected FPGAs. In *Proc. Midwest Symp. Circ. Syst.*, pages 2749–2752, 1997.
- [5] U. Kechschull and W. Rosenstiel. Efficient graph-based computation and manipulation of functional decision diagrams. In *European Conf. on Design Automation*, pages 278–282, 1993.
- [6] M.A. Perkowski. The generalized orthonormal expansion of functions with multiple-valued inputs and some of its application. In *Int'l Symp. on Multi-Valued Logic*, pages 442–450, 1992.
- [7] M.A. Perkowski, M. Chrzanowska-Jeske, and Y. Xu. Lattice diagrams using Reed-Muller logic. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 85–102, 1997.
- [8] T. Sasao and J. Butler. A design method for look-up table type FPGA by pseudo-Kronecker expansion. In *Int'l Symp. on Multi-Valued Logic*, pages 97–106, 1994.
- [9] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.2.0*. University of Colorado at Boulder, 1998.
- [10] W. Wang and M. Chrzanowska-Jeske. Optimizing pseudo-symmetric binary decision diagrams using multiple symmetries. In *Int'l Workshop on Logic Synth.*, pages 334–340, 1998.