6-2004

Logic Synthesis for Layout Regularity using Decision Diagrams

Malgorzata Chrzanowska-Jeske
Portland State University

Alan Mishchenko
Portland State University

Jinsong Zhang
Portland State University

Marek Perkowski
Portland State University

Citation Details

Chrzanowska-Jeske, Malgorzata, et al. "Logic synthesis for layout regularity using decision diagrams." International Workshop on Logic Synthesis. 2004.

Logic Synthesis for Layout Regularity using Decision Diagrams*

Malgorzata Chrzanowska-Jeske[1], Alan Mishchenko[2], Jinsong Zhang[1], and Marek Perkowski[1]

[1] Department of ECE, Portland State University, Portland, OR
[2] Department of EECS, UC Berkeley, Berkeley, CA

Abstract

This paper presents a methodology for logic synthesis of Boolean functions in the form of regular structures that can be mapped into standard cells or programmable devices. Regularity offers an elegant solution to hard problems arising in layout and test generation, at no extra cost or at the cost of increasing the number of gates, which does not always translate into the increase of circuit area. Previous attempts to synthesize logic into regular structures using decision diagrams suffered from an increase in the number of logic levels due to multiple repetitions of control variables. This paper proposes new techniques, which lead to fewer variable repetitions and significantly improve the performance of synthesis algorithms. Experimental results show that the generated regular circuits are larger in the number of gates and comparable in delay to the circuits without regularity produced by SIS, yet they exhibit a number of important advantages, such as localization and predictability of interconnect, reduction in the gate output load, and improved testability.

1. Introduction

The traditional design flow of VLSI IC below the RTL level consists of logic synthesis, technology mapping, and layout synthesis. These steps are frequently performed iteratively to achieve timing closure. In deep-sub-micron (DSM) technology, due to strong influence of interconnect delays on circuit performance, it is very difficult for physical design to converge when logic optimization is performed without taking layout into account.

Besides the problematic convergence issue, the algorithms to perform layout synthesis are highly complex and often require many hours of computation time to complete on large industrial designs. In the last decade, numerous incremental improvements to placement and routing have been explored, such as incremental re-mapping, local re-routing, white space insertion, buffer insertion, wire and transistor sizing. These techniques are symptomatic of the difficulties experienced by the main-stream algorithms and tools.

Correspondingly, there is a growing interest in methodologies that bring together logic synthesis and layout synthesis in an attempt to solve the interconnect problem earlier in the design flow. In recent years, many efforts [9, 10, 12, 13] were devoted to exploring various regular circuit and layout structures, as they are more predictable and have advantages from a manufacturing point of view.

Regularity of layout is especially important when circuits are mapped into programmable or field-programmable logic devices and gate arrays (PLDs, or FPLDs and FPGAs). The majority of these devices have a large portion of their routing resources available as local and neighbor-to neighbor connections. Mapping regular netlists into such devices is denser, which tends to reduce both area and delay. In some cases, a regular circuit with a larger number of gates may be implemented using a smaller area, compared to a non-regular version of the same circuit.

Starting with the early research of Akers [1], regular structures have become an attractive alternative to the traditional design styles. Several variations of the regular structures have been proposed [2, 3, 4, 5, 6, 7, 10, 11]. They provide different trade-offs between the complexity and applicability of the synthesis methods and the efficiency of the resulting implementation.

This paper explores a specialized type of decision diagrams [4, 5, 15], called *Pseudo-Symmetric Kronecker Functional Decision Diagrams* (PSKFDDs), as a vehicle for achieving efficient regular implementations. The goal is to transform Boolean functions into PSKFDDs, which are next mapped into regular circuits composed of Shannon and Davio gates. The connection between gates is limited to several wires, which dramatically simplifies placement and routing. The problems of congestion and long interconnect are eliminated because connections between gates are local, mostly neighbor-to-neighbor, and distributed evenly among the gates. Since gates are placed using a regular pattern, the length and thus delay of local interconnects between gates can be easily predicted before the final layout is generated. Because the majority of connections are short, the need for additional buffers is drastically reduced and, therefore, the total area of the final circuit is also reduced. Crosstalk between wires is reduced as well.

It has been shown that CMOS technology is well suited for regular implementations. Application of similar decision diagrams to generate regular layout for wave pipelining has been studied [7]. Our preliminary research indicates that the proposed regular implementations are highly testable for stuck-at and delay faults by propagating patterns of 0s and 1s horizontally, vertically, and diagonally though the array of regular cells. Other possible benefits of regular structures might include improved reliability and yield due to a smaller number of vias and buffers. It remains to be seen to what extent the regularity in layout can help with power/ground network noise reduction.

The main contributions of this paper are two new efficient algorithms for generating regular layout using PSKFDDs constructed for Boolean functions. The first algorithm is based on the extended set of generalized variable-pair symmetries [8]. Using the complete set of 15 generalized symmetries introduced below allows us to extend the work of [5], resulting in the improved regular synthesis for many benchmarks. The second algorithm performs heuristic PSKFDD synthesis while combining efficient variable expansion/selection with a number of look-ahead strategies yielding quality layouts in reasonable time.

The rest of the paper is organized as follows. Section 2 gives the definitions used in the paper. Section 3 discusses generalized

* This work was supported in part by the NSF grants MIP-9629419 and CCR-9988402

symmetries. Section 4 presents our adaptation of the longest paths computation. Section 5 described two synthesis algorithms: systematic and heuristic. Section 6 lists experimental results. Section 7 concludes the paper.

2. Definitions

Given a Boolean function F: $B^n \rightarrow B$, where $B = \{0,1\}$, the *negative (positive) cofactor* of F with respect to (w.r.t.) variable x is the Boolean function F_0 (F_1) derived by substituting into F instead of x the value 0 (1). We denote F_2 the exclusive sum (EXOR) of the negative and positive cofactors: $F_2 = F_0 \oplus F_1$.

Three canonical expansions of F are defined as follows:

$F = \bar{x}\, F_0 \oplus x\, F_1$ Shannon expansion (S) (1)

$F = F_0 \oplus x\, F_2$ Positive Davio expansion (pD) (2)

$F = F_1 \oplus \bar{x}\, F_2$ Negative Davio expansion (nD) (3)

Cofactors w.r.t. two and more variables are defined as repeated co-factoring w.r.t to each variable in the set. The final result does not depend on the variable order. Of particular interest to this paper are sets of cofactors w.r.t. to variable pairs. Since a pair of variables can have four polarities (00, 01, 10, 11), there are four cofactors denoted F_{00}, F_{01}, F_{10}, and F_{11}.

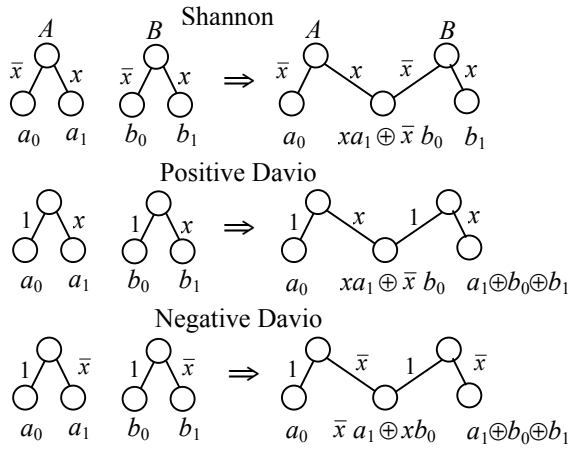

Fig. 1. Join-Vertex operation rules the for left-to-right remainder propagation.

For example, function $G = abd + a\bar{b}c + \bar{a}bc$ has the following cofactors w.r.t. the variable pair (a, b):

$$G_{00} = G(0, 0, c) = 0, G_{01} = G(0, 1, c) = c,$$
$$G_{10} = G(1, 0, c) = c, G_{11} = G(1, 1, c) = d.$$

Regular structures discussed in this paper are called *lattices*[1]. Essentially, a lattice is a set of regularly placed gates locally interconnected to form a grid. Each gate has a control signal propagating from left to right and two data signals propagating from bottom to top. However, lattice synthesis is performed from top to bottom.

The Join-Vertex operation has been introduced in [3] as a way of dealing with the incompatibility of cofactors of the adjacent nodes (cofactors a_1 and b_0 in Fig. 1). The idea of this operation is to multiplex the cofactors using the control variable x of the given level in such a way that nodes A and B shared one of the cofactors but preserved the original functions. Fig. 1 lists the Join-Vertex operation rules for Shannon, Positive Davio, and Negative Davio gates. In this paper, we consider only Kronecker lattices, which have the same type of gates throughout a level.

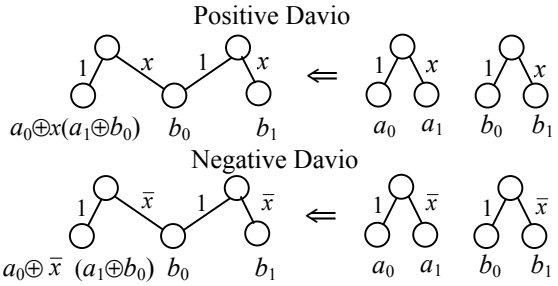

Fig. 2. Join-Vertex operation rules the for right-to-left remainder propagation.

In the case of the Davio expansions (Fig. 1), to preserve the function of node B, it is necessary to balance the negative cofactor of this node by adding a remainder to the positive cofactor. In this case, Join-Vertex is not a local operation and leads to the propagation of a remainder from left to right. Fig. 2 shows similar Join-Vertex rules for the Davio expansions, in which the wave of remainders is propagated from right to left. The heuristic synthesis algorithm described in this paper achieves additional flexibility by using both sets of propagation rules.

3. Generalized Variable Pair Symmetries

A Boolean function F has a classical non-equivalence (equivalence) two-variable symmetry [8] iff replacing the first variable by (the complement of) the second and the second variable by (the complement of) the first yields the same function.

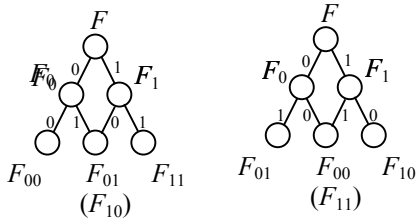

Fig. 3. The upper part of the decision diagram of a function with a classical non-equivalence symmetry (left) and equivalence symmetry (right).

Given the four cofactors of F w.r.t. a pair of variables, F_{00}, F_{01}, F_{10}, F_{11}, it is possible to give another definition of classical symmetries. F has a classical two-variable non-equivalence symmetry iff $F_{01} \oplus F_{10} = 0$ and a classical two-variable equivalence symmetry iff $F_{00} \oplus F_{11} = 0$. If the variable pair exhibiting the symmetry is ordered above other variables in the reduced decision diagram, then the upper part of the diagram looks as shown in Fig. 3.

For example, function $G = a\,b\,d + a\bar{b}c + \bar{a}bc$ has a non-equivalence symmetry in pair (a,b), while function $H = a\bar{b}d + \bar{a}\bar{b}c + abc$, has equivalence symmetry in pair (a,b).

If the function has a two-variable symmetry, then at most three out of four cofactors w.r.t. a pair of variables are different functions. It is possible to generalize the concept of classical symmetries by defining other conditions when *at most three out of the four cofactors are non-constants*. For example, the above condition is true when one out of four cofactors of the function

[1] The use of this term in the paper is not related to the set-theoretic concept of a lattice.

w.r.t. a variable pair is constant 0 or 1. This gives rise to four new symmetries called *constant-cofactor symmetries*. Another way of extending the concept of classical symmetries is by considering the Davio cofactors, that is, the EXORs of cofactors from the set { F_{00}, F_{01}, F_{10}, F_{11} }. For example, if the three two-variable cofactors satisfy $F_{00} \oplus F_{10} \oplus F_{11} = 0$, this can be considered as a new kind of symmetry called *a Kronecker symmetry*. Table 1 lists 15 types of symmetry derived using the set of 4 two-variable cofactors. Notice that all formulas in the column "Property" can be equal to constant 0 *or* constant 1. This leads to two subtypes of each of the 15 symmetries. When the expression is equal to constant 0, the symmetry is *a non-skew symmetry*; when it is equal to constant 1, the symmetry is *a skew symmetry*. Experimental results [9] show that, in MCNC benchmarks, non-skew symmetries are more common then skew.

Table 1. Classification of generalized symmetries.

#	Property	Name	Symbol
1	$F_{00} = 0/1$	Constant-cofactor symmetries	C_0
2	$F_{01} = 0/1$		C_1
3	$F_{10} = 0/1$		C_2
4	$F_{11} = 0/1$		C_3
5	$F_{10} \oplus F_{01} = 0/1$	Non-equivalence	$T_1(NE)$
6	$F_{00} \oplus F_{11} = 0/1$	Equivalence	$T_2(Eq)$
7	$F_{00} \oplus F_{01} = 0/1$	Two-cofactor (single-variable) symmetries	T_3
8	$F_{10} \oplus F_{11} = 0/1$		T_4
9	$F_{00} \oplus F_{10} = 0/1$		T_5
10	$F_{01} \oplus F_{11} = 0/1$		T_6
11	$F_{01} \oplus F_{10} \oplus F_{11} = 0/1$	Three/four-cofactor (Kronecker) symmetries	K_0
12	$F_{00} \oplus F_{10} \oplus F_{11} = 0/1$		K_1
13	$F_{00} \oplus F_{01} \oplus F_{11} = 0/1$		K_2
14	$F_{00} \oplus F_{01} \oplus F_{10} = 0/1$		K_3
15	$F_{00} \oplus F_{01} \oplus F_{10} \oplus F_{11} = 0/1$		K_4

Generalized symmetries of the given function can be computed using several methods. We have developed two different ways of computing the generalized symmetry information for all pairs of variables, *without* creating intermediate BDD nodes. However, discussion of these algorithms is beyond the intended scope of the paper.

The Reduction Types

The study of generalized symmetries is motivated by the fact that functions with these symmetries can be represented by a reduced decision diagram, with at most three nodes on the third level. Fig. 4 shows the upper part of the decision diagram for constant-cofactor symmetries (C_0-C_3) and single-variable symmetries (T_3-T_6). Figs. 3 and 4 illustrate the usefulness the first 10 symmetries in Table 1 for the generation of regular layout.

The Davio symmetries (K_0-K_4) also lead to the reduction in the number of non-constant nodes on the third level, if the Positive and Negative Davio expansions are used instead of the Shannon expansion.

For example, suppose the expansions of the first two levels are Positive Davio and Shannon respectively and the variables have the four-cofactors symmetry K_4. Notice that after the transformation of the diagram as shown in Fig. 5, the effect of these two expansion (*pD* followed by *S*) with symmetry K_4 is similar to the effect of symmetry T_4.

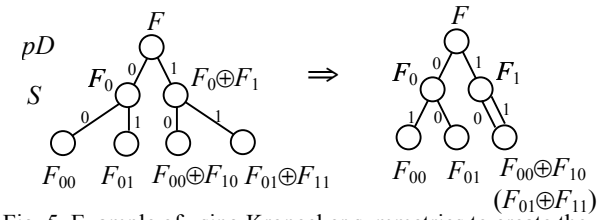

Fig. 5. Example of using Kronecker symmetries to create the planar layout.

The above example shows that symmetry T_4 (in fact, any of the first 8 symmetries) can be seen as a *reduction type,* which describes how the cofactors are combined on the third level of the decision diagram. The mapping of symmetries into reduction types for the nine possible expansion pairs is given in Table 2.

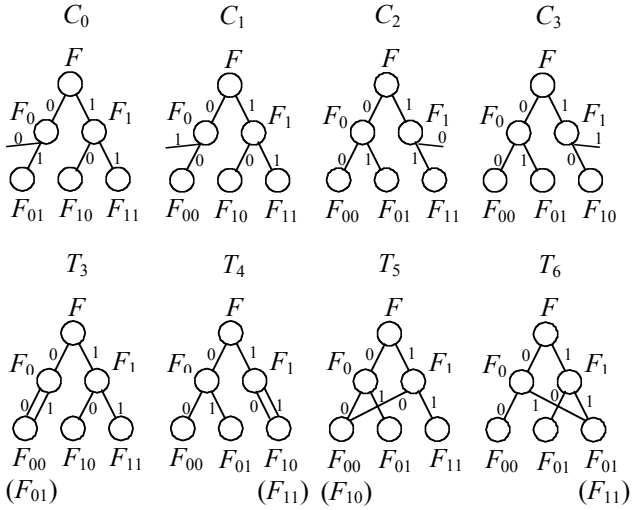

Fig. 4. The upper part of the decision diagram for a function with generalized symmetries.

Table 2. Mapping of generalized symmetries into reductions types for different expansion pairs.

Expan. Pair	Generalized Symmetries														
	C_0	C_1	C_2	C_3	*NE*	*Eq*	T_3	T_4	T_5	T_6	K_0	K_1	K_2	K_3	K_4
S – S	C_0	C_1	C_2	C_3	*NE*	*Eq*	T_3	T_4	-	-	-	-	-	-	-
S–pD	C_0	T_3	C_2	T_4	-	-	C_1	C_3	-	-	-	*Eq*	-	*NE*	-
S–nD	T_3	C_0	T_4	C_2	-	-	C_1	C_3	-	-	*Eq*	-	*NE*	-	-
pD–S	C_0	C_1	-	-	-	-	T_3	-	C_2	C_3	-	-	*Eq*	*NE*	T_4
pD–pD	C_0	T_3	-	-	*NE*	-	C_1	-	C_2	T_4	*Eq*	-	-	-	C_3
pD–nD	T_3	C_0	-	-	-	*NE*	C_1	-	T_4	C_2	-	*Eq*	-	-	C_3
nD–S	-	-	C_0	C_1	-	-	-	T_3	C_2	C_3	*Eq*	*NE*	-	-	T_4
nD-pD	-	-	C_0	T_3	-	*NE*	-	C_1	C_2	T_4	-	-	*Eq*	-	C_3
nD-nD	-	-	T_3	C_0	*NE*	-	-	C_1	T_4	C_2	-	-	-	*Eq*	C_3

Note that for the two Shannon expansions (row "*S–S*") the reduction types for the first 8 symmetries are the same as these symmetries. Other expansions lead to different combinations of cofactors, which is reflected in a change of the reduction type. The dash in Table 2 means that, for the given expansion pair and generalized symmetry, this method cannot be used to create a regular layout.

Consider another example in Fig. 6. Suppose the expansions are *nD* and *pD* and the generalized symmetry is C_2. Table 2 states that, in this case, the reduction type is C_0. The cofactor F_{10} is indeed a constant due to symmetry C_2.

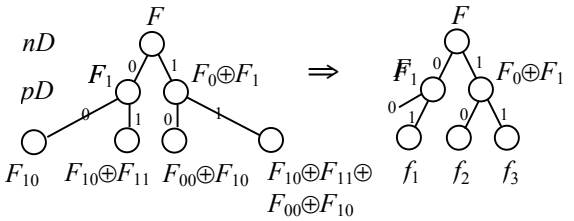

Fig. 6. Example illustrating the use of Table 2.

4. Longest Path Computation

The Symmetry Compatibility Graph

In [9], the concept of *symmetry compatibility graph* was introduced as a directed graph, whose nodes represent variables of the function and edges represent variable-pair symmetries. If two variables have no symmetries, there is no edge between the corresponding pair of nodes; otherwise, the edge is labeled by the available symmetries. Edges of the graph are directed because some symmetry types are sensitive to the ordering of variables in the pairs.

The Longest Path Computation

Given a symmetry graph, it is possible to find a sequence of variables such that each pair of adjacent variables in the sequence have a generalized symmetry. This variable sequence in called a *variable path* (a *variable chain* in [9]). When the path of sufficient length is found, the expansions are assigned to each variable on the path in such a way that the corresponding reduction type (see Table 2) allows for the planar layout to be created on each level similar to how it is created in Figs. 5 and 6. The longest path is found by depth-first search (DFS) in the symmetry graph starting from every variable that has symmetries with other variables. Because for large graphs the enumeration of all candidate paths takes time exponential in the number of variables, the DFS is restricted by allowing a limited number of backtracks. For many benchmarks, this methods yields the exact solution in a short computation time.

Table 3. Restrictions on reduction type sequences.

	C_0	C_1	C_2	C_3	*NE*	*Eq*	T_3	T_4
C_0	X	X			X	X	X	
C_1			X	X	X	X		X
C_2	X	X			X	X	X	
C_3			X	X	X	X		X
NE								
Eq								
T_3					X	X		
T_4					X	X		

Due to the presence of dashes in Table 2, not every symmetry sequence results in an applicable sequence of reduction types for every expansion pair, and therefore inclusion of variables into the path should be restricted by additional requirements.

Besides, it can be observed that certain reduction type sequences cannot be properly exploited to create the regular layout. These restrictions are summarized in Table 3. The cross in this table means that the reduction type specified in the column cannot follow immediately after the reduction type specified in the row. These restrictions can be easily incorporated into the DFS algorithm.

5. Lattice Synthesis Algorithms

The implementation of lattice synthesis discussed in this paper is based on two approaches: *systematic* and *heuristic*.

The systematic approach requires computation of generalized symmetries of the function w.r.t. all variable pairs and finding the longest path in the variable-pair symmetry graph, with the restrictions on symmetry sequences described in Section 5.

The symmetry sequence is computed for each level (except the first one) as a set of (1) an input variable to be used a control variable for this level, (2) an expansion type, (3) the symmetry between the control variable of this level and the previous one in the order, (4) the reduction type of this expansion and the previous one in the order.

If the function has no symmetries, or if the symmetry path does not include all variables, only the upper part of the diagram is synthesized using the systematic approach based on the longest path. The rest of the diagram is constructed using a heuristic algorithm, which does not guarantee that the control variables are not repeated.

We experimented with a number of heuristic algorithms and found a simple one that gives the best speed/quality trade-off. Common to both systematic and heuristic synthesis algorithms is the iterative lattice construction, level by level from top to bottom. Synthesis of each level assumes that functions to be implemented are assigned to the nodes of the lattice.

The iterative synthesis procedures treat functions assigned to the lattice nodes as the input and proceed cofactoring them according to the variable and expansion selected followed by the computation of the layout information. As the last step of synthesis for the given level, the functions are assigned to the nodes of the next level. This completes one iteration of lattice synthesis and prepares the next level for calling the synthesis procedure.

The main difference in the heuristic synthesis compared to the systematic synthesis is the need to determine the variable/expansion pair at each level. There are several criteria for selecting the variable and expansion to be used on the given level: (1) the number of join-vertex operations that should be performed on the given level; (2) the number of constant cofactor that are produced as a result of expanding all the functions of the level using this variable; (3) the sum total of variables in the support of all cofactors produced as a result of expanding all the functions of the level using this variable. First, variables/expansion pairs are evaluated according to criterium (1). If there is no tie, the best variable/expansion pair is returned, without the need for further computation. If there is a tie, the next level of the lattice is constructed for each variable and criteria (2) and (3) are used to find the best variable. This strategy corresponds to the look-ahead of depth 1.

6. Experimental Results

The algorithms have been implemented and tested using MCNC benchmarks. The resulting regular circuits for each output were written into BLIF files and verified for correctness against the original functions using SIS. The reported runtimes are in seconds on a Pentium III 933Mz computer. This runtime includes only logic synthesis. It does not include the runtime for reading the input file and constructing the BDDs of the output functions.

Tables 4 and 5 give the comparison of the presented heuristic synthesis algorithm with the previous work, [5] and [6]. In both

tables, column "Name" lists the name of the benchmark, column "Outs" lists the total number of outputs followed, in parentheses, by the 1-based number of the output of a single-output function used. Column "Ins" gives the number of inputs in the multi-output benchmark function, followed, in parentheses, by the number of inputs in the support of the given single-output function. Results in [5] do not report the number of gates, so the comparison in Table 4 is in terms of logic levels and runtime. The shaded column shows the best result for each function.

In Table 5, the comparison is in terms of logic levels and the number of nodes (gates) in the resulting circuits. Again, in all but a few cases, the proposed algorithm generated a more compact layout. The runtime for all the examples reported in Table 5 was close to one second.

Table 4. Comparison with [5].

Name	Outs	Ins	Results from [5]		Our results	
			Levels	Time	Levels	Time [s]
mux	1(1)	21(21)	31	49.27	21	0.10
term1	10(5)	34(17)	20	2.61	17	0.04
	10(6)	34(18)	21	2.98	18	0.04
	10(7)	34(19)	22	3.36	19	0.08
x2	7(7)	10(10)	11	0.82	10	0.01

Table 5. Comparison with [6].

Name	Outs	Ins	Results from [6]		Our results	
			Levels	Nodes	Levels	Nodes
apex7	37(30)	49(17)	25	148	17	47
clip	5(2)	9(9)	18	103	16	103
	5(2)	9(9)	27	220	13	71
cm162a	5(3)	14(10)	11	24	10	19
cps	109(1)	24(22)	26	134	26	244
	109(2)	24(18)	26	164	23	112
	109(3)	24(22)	39	342	27	211
duke2	29(13)	22(17)	20	90	19	95
	29(17)	22(18)	22	92	18	57
	29(21)	22(16)	19	76	16	76
	29(7)	22(18)	21	109	21	120
example2	66(23)	85(16)	21	52	16	34
	66(59)	85(14)	17	31	14	36
	66(63)	85(13)	15	37	13	42
frg2	139(99)	143(20)	22	189	20	46
	139(100)	143(19)	28	164	19	103
sao2	4(2)	10(10)	18	71	13	61
	4(3)	10(10)	16	73	12	47
	4(4)	10(10)	16	68	13	61

Table 6 compares the results of [9] with both the systematic (column "System") and the heuristic (column "Heuris") algorithms presented in the paper. The comparison is in terms of logic levels and nodes (gates). The two last rows show the sum total of entries in the cells of each column and the ratio of the results compared to the column "DVS", which is taken to be 100%. For the examples in this table, the heuristic algorithm is on average better than the systematic one.

Table 7 shows results for multi-output MCNC benchmarks. The two sections of Table 7 deal with the systematic and the heuristic algorithm. Inside each section, the column "Nodes" gives the total number of nodes for all outputs. Column "Join" shows the number of outputs that require the application for Join-Vertex operation. Number 0 in this column means that all the outputs of the given benchmark can be expanded without variable repetition. The number in parentheses given in the column "Join" indicates the number of outputs, for which the layout could be computed within less than the predefined limit (100) on the number of levels. Finally, the column "Time" gives the synthesis runtime for all outputs of each benchmark, for which synthesis completed.

Table 6. Comparison with [9].

Name	Out#	Ins	Results from [9]						Our results			
			DVS		+SVS		Heuris		System		Heuris	
			lev	nod	lev	nod	lev	nod	lev	nod	lev	nod
apex6	39	20	41	285	42	262	56	746	25	258	22	146
apex7	29	17	26	142	26	112	25	148	17	63	17	47
frg2	138	13	39	269	24	84	31	155	13	28	13	17
k2	29	20	40	158	38	136	35	269	29	364	28	291
sct	10	14	26	84	19	36	14	47	14	47	14	44
term1	8	18	43	158	44	156	44	426	18	54	18	58
ttt2	15	14	16	32	16	29	14	30	14	34	14	33
vda	25	15	46	219	32	144	28	188	18	118	24	198
x1	17	16	21	52	19	25	19	94	16	46	16	41
x3	39	20	66	681	64	611	56	746	25	258	22	146
Total			364	2080	324	1595	322	2849	189	1270	188	1021
Rat.%			100	100	89.0	76.6	88.4	137	51.9	61.1	51.6	49.0

Table 7. Evaluation of the proposed algorithms.

Name	Ins	Outs	Systematic			Heuristic		
			Nodes	Join	Time	Nodes	Join	Time [s]
clip	9	5	187	3	0.06	306	4	0.08
cordic	23	2	1496	2	1.36	298	1	0.18
9sym	9	1	33	0	0.01	33	0	0.01
alu4	14	8	620	5(3)	8.51	864	5(3)	7.90
rd84	8	4	100	0	0.03	54	0	0.03
duke2	22	29	2906	6	2.05	1347	6	0.41
misex2	25	18	208	0	0.18	187	0	0.08
cps	24	109	6120	13(1)	8.89	5573	13	2.04
vg2	25	8	1482	2	1.34	1924	2(1)	4.08
t481	16	1	125	1	0.07	52	0	0.02
seq	41	35	1699	25(19)	73.22	2365	20(16)	16.91
apex6	135	99	2753	1	3.76	1266	1	0.51
apex7	49	37	1693	3	2.00	911	0	0.34
frg2	143	139	8065	6(2)	19.23	5134	9	2.04
count	35	16	1319	0	0.84	216	0	0.09
term1	34	10	806	4	0.62	398	0	0.13
x1	51	35	1820	4(2)	11.01	1248	3	1.06
x2	10	7	102	0	0.06	67	0	0.02
x3	135	99	2468	1(1)	5.45	1230	0	0.50
k2	45	45	690	21(21)	931.05	13854	22(10)	140.90
i8	133	81	7873	78(19)	16.35	25476	44(11)	8.80
i9	88	63	595	63(56)	68.21	20019	63(1)	5.80
pair	173	137	17070	65(26)	640.34	12696	45(20)	145.96
des	256	245	18739	120(81)	60.83	14634	120(55)	33.51
dalu	75	16	0	16(16)	210.05	4043	15(11)	19.32
Total			78969	144	2065.52	114195	214	390.72
Ratio,%			100.0	100.0	100.0	144.6	148.6	18.9

Finally, table 8 gives the comparison of area and delay of the regular implementations with the results by SIS. The comparison is done using circuits, for which the proposed algorithm could synthesize all outputs. The columns "Ins" and "Outs" list the

number of inputs and outputs in each benchmark. Columns "Node" and "Level" show the total number of nodes (gates) and maximum number of levels in the regular circuits generated by the proposed algorithm. Columns "Area" and "Delay" contain the measurements for area and delay of the resulting circuits.

The SIS results are obtained by running *script.rugged* followed by mapping for delay (*map –m* 1.0) into the standard gate library *mcnc.genlib*, which comes as part of the distribution of SIS. Shannon, Positive Davio, and Negative Davio gates are assumed to have area 2 times larger and delay 1.5 times larger than a 2-input NAND in *mcnc.genlib*. The delay of the wires and the increase in delay due to multiple fanouts are not taken into account.

Table 8. Area/delay comparison with SIS.

Name	Ins	Outs	SIS		Our results				
			Area	Delay	Node	Level	Area	Del	Time [s]
rd84	8	4	283	29.84	54	8	216	14.4	0.03
clip	9	5	238	24.67	187	11	748	19.8	0.06
cordic	23	2	138	16.20	102	26	408	46.8	0.07
duke2	22	29	834	44.23	1395	21	5580	37.8	0.95
misex2	25	18	206	16.80	208	14	832	25.2	0.18
apex6	135	99	1274	35.03	1266	24	5064	43.2	0.51
apex7	49	37	409	24.90	828	24	3312	43.2	0.25
count	35	16	308	49.50	216	20	864	36.0	0.09
term1	34	10	326	18.45	268	20	1072	36.0	0.08
x1	51	35	580	18.15	974	26	3896	46.8	0.70
x3	135	99	1362	28.88	1230	24	4920	43.2	0.50
Total			5958	306.65			26912	392.4	
Rat,%			100.0	100.0			451.6	127.9	

The noticeable increase in area computed using our algorithm is largely due to the fact that each output is treated independently and logic is not shared across outputs of multi-output functions. Comparison between SIS and our representations, even on mapped circuits, still does not account for delay associated with long interconnects in SIS-generated solutions.

Additional interconnect delay in our circuits will be substantially smaller because most wires are local. When comparing circuit area, we should consider the buffers that will need to be inserted into SIS-generated circuits to reduce delay below specified limits. In our solutions buffers are not needed because connections are short. Moreover, for deep-submicron technologies, design area is limited by area occupied by interconnects, not by gate areas, so the areas of SIS and our solutions might become similar when final and complete layouts are generated.

7. Conclusions

In this paper, we propose a methodology to synthesize logic functions into regular structures using decision diagrams. The resulting regular structures are easy to place and route, because the wires connecting the cells are short and local. These structures can be mapped into both standard cells and programmable devices.

Among the two synthesis algorithms, the systematic one takes a global view of the problem by pre-computing the set of generalized symmetries for all variable pairs. The longest path in the graph of these symmetries is then used to order the variables for synthesis. The heuristic algorithm, on the contrary, uses a look-ahead strategy and thereby takes a greedy local view of the problem of ordering variables. The experimental results show that the heuristic algorithm is several times faster but sometimes loses quality compared to the systematic one.

Experiments show that regular circuits generated by the proposed algorithms come close to those generated by SIS in delay, but they are, on average, 4.5 times larger in area. The area increases for two reasons: it is the price to pay for regularity of the resulting circuits, and it is due to the fact that each output of the multi-output benchmarks is currently synthesized in isolation from other outputs. However, as discussed in the previous section the area disadvantage can disappear in deep-submicron technology when final layouts are compared. This is due mostly for two reasons: additional buffer insertion, and definition of area in terms of wires and not gates.

The future research will focus on efficient area recovery in the regular implementations of logic functions and on enhancing the ruggedness of the current implementation to be able to handle larger functions. One of the promising approaches of solving both problems is to combine regular synthesis with functional decomposition.

References

[1] S. B. Akers. "A Rectangular Logic Array," *IEEE Trans. on Computers*. Vol. C-21, pp. 848-857, 1972.

[2] T. Sasao, J. T. Butler, "Planar Multiple-Valued Decision Diagrams," *Proc. ISMVL '95*, pp. 28-35.

[3] M. Chrzanowska-Jeske, Z. Wang, "Mapping of Symmetric and Partially Symmetric Functions to the CA-Type FPGAs," *Proc. of the IEEE Midwest Symposium on Circuits and Systems, MWCAS'95, pp. 290-293, 1995.*

[4] M. Chrzanowska-Jeske, J. Zhou, "AND/EXOR Regular Function Representation, "*Proc. of the IEEEMidwest Symposium on Circuits and Systems*, MWCAS,'97, pp. 1034-1037, 1997.

[5] P. Lindgren, R. Drechsler, B. Becker, "Synthesis of Pseudo-Kronecker Lattice Diagrams," *Proc. Intl. Worshop Appl. Reed-Muller Expantions,* pp. 197 – 204, 1999.

[6] M. Chrzanowska-Jeske, Y. Xu, M. Perkowski, "Logic Synthesis for a Regular Layout," VLSI Design: An International Journal of Custom-Chip Design, Simulation and Testing, 2001.

[7] A. Mukherjee, R. Sudhakar, M. Marek-Sadowska, S. I. Long, "Wave Steering in YADDs: A Novel Non-Iterative Synthesis and Layout Technique,". *Proc. DAC'99*, pp. 466-471.

[8] C.C. Tsai, M. Marek-Sadowska, "Generalized Reed-Muller Forms as a Tool to Detect Symmetries," *IEEE Trans. Comp, vol 45, no. 1 pp. 33-40, 1996.*

[9] W. Wang, M. Chrzanowska-Jeske, "Generating Linear Arrays Using Symmetry Chain," *Proc. IWLS'99*, pp. 115-119.

[10] F. Mo, R.K. Brayton, "River PLA; A Regular Circuit structure," *Proc. DAC, pp 201-206, 2002.*

[11] F. Mo, R.K. Brayton, Regular Fabrics In Deep Sub-Micron Integrated-Circuit design," *Proc. IWLS'02, pp 7-12.*

[12] M. Palasinski, A. J. Strojwas, W. Maly, "Regularity in Physical Design," *GSRC Workshop '01*, pp. 17-18, 2001.

[13] L. Pileggi et.al. "Exploring Regular Fabrics to Optimize the Perfromance Cost Trade-OFF," *Proc. DAC '03, pp.782-787.*

[14] B. Drucker et al., "Polairized Pseudo-Kronecker Symmetry with application to the Synthesis of Lattice Decision Diagrams," *Proc. ICCIMA '98, pp.745-754.*

[15] M. Perkowski, M. Chrzanowska-Jeske, and Y. Xu. "Lattice Diagrams Using Reed-Muller Logic," *Proc. Intl. Worshop Appl. Reed-Muller Expantions,* pp. 85 - 102, 1997.