Synthesis for Regularity using Decision Diagrams

Malgorzata Chrzanowska-Jeske
Electrical and Computer Engineering
Portland State University
Portalnd, OR 97229
jeske@ece.pdx.edu

Alan Mishchenko
Department of EECS
UC Berkeley
Berkeley CA 94708
alanmi@eecs.berkeley.edu

Abstract

Presented are new algorithms for synthesizing Boolean functions as regular logic structures. These regular structures can be mapped directly (without place&route) to a standard-cell library designed for regularity or to locally-connected programmable devices. The advantage of regular structures is that for a planar embedding the number of nodes in the expansion level grows at most linearly with the number of expansion variables. Regularity offers a predicable solution to hard problems arising in layout, at no extra cost or at the cost of increasing the number of gates, but without necessarily increasing circuit area. Increasing the number of logic levels does not translate into an increase in overall circuit delay, because regular, neighbor-to-neighbor connections reduce the wire delay, the dominant factor in deep sub-micron technology. This paper proposes new techniques which lead to less variable repetition and significantly improve the performance of synthesis algorithms. Experimental results much better than previously published data are very encouraging.

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 considering layout.

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 physical synthesis have been explored, such as incremental re-mapping, local re-routing, white space and buffer insertion, wire and transistor sizing. Correspondingly, there is a growing interest in approaches that integrate logic and layout synthesis in an attempt to solve the interconnect problem earlier in the design flow. Recently, many efforts were devoted to exploring various regular circuit and layout structures [9,10,12], as they are more predictable and have advantages from a manufacturing viewpoint.

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, 10,11,14]. 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 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 can be easily predicted before the final layout is generated. Because the majority of connections are short, the need for additional buffers is reduced and, the total area of the final circuit is also reduced. 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].

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 [8,13]. Using the set of 15 generalized symmetries allows us to extend the work of [5], resulting in the improved regular design for many benchmarks. The second algorithm performs heuristic PSKFDD synthesis while combining efficient variable expansion/selection with a number of look-ahead strategies.

The rest of the paper is organized as follows. Section 2 gives the definitions used in the paper. Section 3 discusses generalized symmetries. Section 4 presents our adaptation of the longest paths computation. Section 5 described two synthesis algorithms. 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)

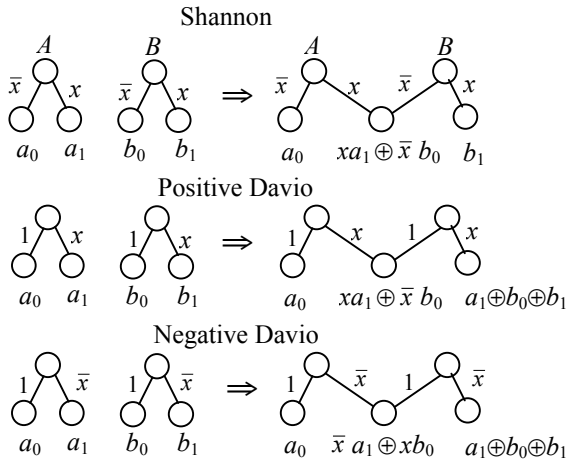

Fig.1. Join-Vertex operation rules the for left-to-right propagation.

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}.

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]. 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. 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 diagrams, which have the same type of gates throughout a level.

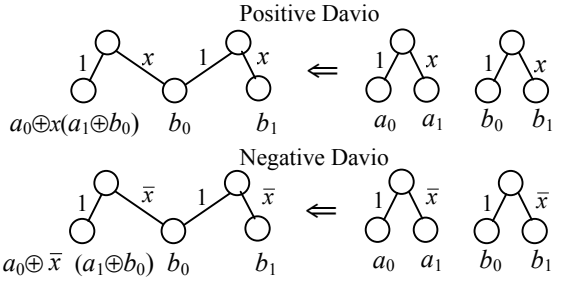

Fig. 2. Join-Vertex operation - the for right-to-left propagation.

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 rules, 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. The generated function representation is called Pseudo-Symmetric Kronecker Functional Decision Diagram (PSKFDD).

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.

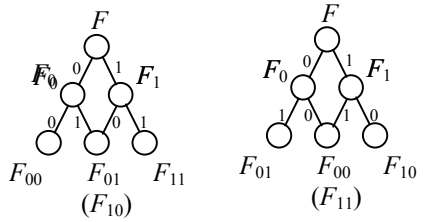

Fig. 3. The upper part of the decision diagram of a function with a classical T_1(*NE* symmetry (left) and T_2(*Eq* symmetry (right).

[1] It is not related to the set-theoretic concept of a lattice.

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.

It is possible to generalize [13] the concept of classical symmetries by defining other conditions when *at most three out of the four cofactors are non-constants*. 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. Generalized symmetries of the given function can be computed using several methods, however, discussion of these algorithms is beyond the intended scope of the paper.

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

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.

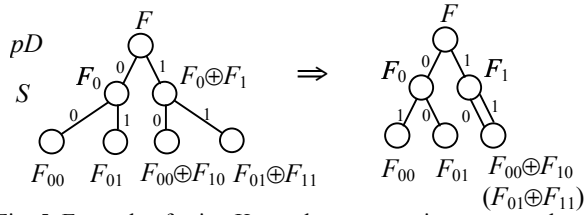

Fig. 5. Example of using Kronecker symmetries to create the planar layout.

The 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.

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.

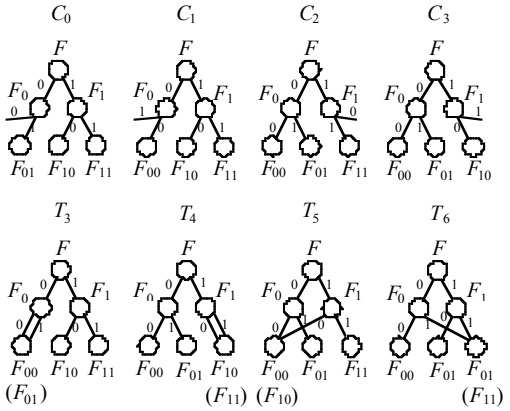

Fig. 4. The upper part of the decision diagram for a function with generalized symmetries.

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

The dash in Tab. 2 means that, for the given expansion pair and generalized symmetry, a regular layout cannot be created.

4. Longest Path Computation

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 symmetries. Edges of the graph are directed because some symmetries are sensitive to the ordering of variables in the pairs.

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. 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 number of allowed backtracks is restricted. For many benchmarks, this method yields the exact solution in a short computation time.

Not every symmetry sequence results in an applicable sequence of reduction types for every expansion pair, and thus inclusion of variables into the path should be restricted by additional requirements. Also, it can be observed that certain reduction type sequences cannot be exploited to create the regular layout. These restrictions can be easily incorporated into the DFS algorithm.

5. Synthesis Algorithms

The implementation of PSKFDD synthesis 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. 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 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. Common to both systematic and heuristic synthesis algorithms is the iterative lattice construction, level by level from top to bottom.

The main difference in the heuristic synthesis compared to the systematic 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 criterion (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. 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, we compare the numbers of logic levels with the number of nodes (gates). 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(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(3)	24(22)	39	342	27	211
Duke2	29(17)	22(18)	22	92	18	57
	29(7)	22(18)	21	109	21	120
example2	66(23)	85(16)	21	52	16	34
	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

Table 6 compares the results of [9] with both the systematic ("System") and the heuristic ("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", taken to be 100%. For the examples in this table, the heuristic algorithm is on average better than the systematic.

Table 7 shows results for multi-output MCNC benchmarks. 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 in the column "Join" gives 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

7. Conclusions

We proposed a methodology to synthesize logic functions into regular structures using decision diagrams. The resulting regular structures are easy to layout, because the signal wires connect only adjacent cells, and control wires are local. The structures can be mapped into library of pre-designed Davio and Shannon cells.

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 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.

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

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 MWCAS'95, pp. 290-293, 1995.*

[4] M. Chrzanowska-Jeske, J. Zhou, "AND/EXOR Regular Function Representation, "*Proc. of MWCAS, pp. 1034*, 1997.

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

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

[14] M. Perkowski, et al.. "Lattice Diagrams Using Reed-Muller Logic," *Proc. Intl. Worshop Appl. Reed-Muller Expantions,* pp. 85 - 102, 1997.