Constructive Library-Aware Synthesis Using Symmetries*

Victor N. Kravets Karem A. Sakallah

Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109
{vkravets, karem}@eecs.umich.edu

Abstract

In this paper a constructive library-aware multilevel logic synthesis approach using symmetries is described. It integrates the technology-independent and technology-dependent stages of synthesis, and is premised on the goal of relating the functional structure of a logic specification closer to the ultimate topological and physical structures. We show that symmetries interpreted as structural attributes of functions can be effectively used to induce a favorable structural implementation. These symmetries are used in bridging 1) the structural properties of the functions being synthesized, 2) the structural attributes of the implementation network, and 3) the functional content of the target library. Experimental results show that the quality of circuits synthesized using this approach is generally superior to those synthesized by traditional approaches, and that the improvement correlates with the symmetry measure in a function.

1 Introduction

The separation of technology-independent transformations from technology mapping has been widely recognized as potentially detrimental to the overall quality of synthesized circuits. The drawbacks of such separation are becoming more evident in fine-featured ICs where the cost functions employed at the technology-independent stage are becoming increasingly inaccurate. In response to these challenges of deep-submicron designs various post-processing re-mapping techniques have been proposed [1,7]. In a recent development [9] decomposition and technology mapping are combined by strictly defining the possible decomposition types, and applying them to dynamically explore the space of possible implementations using a gate level cost function. The local nature of these transformations, though, precludes addressing global functional properties which can be profitably used to improve circuit quality.

The central idea of this paper is based on the premise that functional specifications have global *structural attributes*, that can be profitably used to induce a favorable structural implementation, while reducing the run time complexity of the synthesis process. These attributes can have a profound effect on the suitability of one decomposition type over another. They can be further utilized to study requirements on the functionality of library primitives to make a particular decomposition type effective (see Fig. 1). Thus, by judiciously coupling the decomposition type with a library using structural attributes of a function we are able to merge the traditionally separate technology-independent and technology-dependent synthesis stages. The effect of the integration leads to improved synthesis quality, reflecting the global functional properties in the final circuit structure. Using symmetries as an example of structural attributes we show that such an approach can indeed yield a significant improvement in circuit quality.

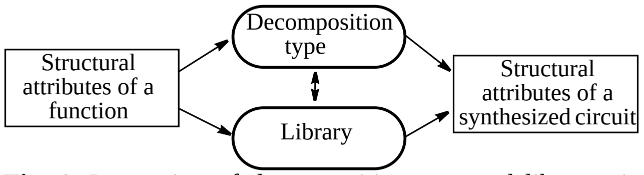

Structural attributes of a function
Decomposition type
Library
Structural attributes of a synthesized circuit

Fig. 1. Integration of decomposition type and library via structural attributes of a functional specification, and their impact on circuit quality

Symmetries as functional properties have been studied to improve synthesis quality for a long time [5, 6, 14]. In this work symmetries establish the key relation between desirable decompositions and the library of primitives which makes such decompositions possible. Specifically, symmetries allow us to define a decomposition type that restricts its decomposition functions to a small library of pre-characterized symmetric primitives, and yet achieves support-reduction in the composition function. In this work such decomposition is termed support- and depth-reducing. Through a construction process of the library we show that its symmetric properties are induced by the symmetric structure of a function being synthesized.

An algorithm incorporating this decomposition successively decomposes the node functions of a Boolean network in topological order. A node in the network is selected for decomposition only if each of its fanin nodes corresponds to a library primitive. The effect of such decomposition is an expansion of the Boolean network from primary inputs towards primary outputs as more library-specific nodes are introduced into the network, and the complexity of the yet

*This work was partially supported by DARPA under contract DAAH04-94-G-0327, and is supported by SRC-99-TJ-690.

unimplemented nodes is reduced. Such a constructive synthesis approach enables us to control decomposition possibilities based on the evolving structure of a network being synthesized.

The remainder of this paper is organized as follows. Section 2 describes support-reducing decomposition. Symmetries as structural attributes of a function are described in Section 3. The symmetry-based primitive library is described in Section 4. A constructive synthesis algorithm that integrates these concepts is described in Section 5. The algorithm is based on the iterative decomposition of a function. Experimental results are discussed in Section 6.

2 Support- and Depth-Reducing Decomposition

Multilevel synthesis can be viewed as a process of successive decompositions of a set of functional specifications to yield an appropriate factored form for these specifications in terms of a set of primitive operators. Any Boolean function $f(x_1, \ldots, x_n)$ can be expressed as

$$f = h(g_1(x_1, \ldots, x_s), \ldots, g_k(x_1, \ldots, x_s), x_1, \ldots, x_n)$$

where $x_1, \ldots, x_s$, and $g_1, \ldots, g_k$ are referred to, respectively, as the *decomposition* variables and functions, and h as the *composition* function (see Fig. 2-a). Different synthesis algorithms can be characterized by the restrictions they impose on this generic decomposition template to achieve various trade-offs between synthesis efficiency and resultant circuit quality. In particular, we define *support-reducing decomposition* by requiring the composition function h to depend on fewer variables than the original function f. The decomposition template in this case becomes

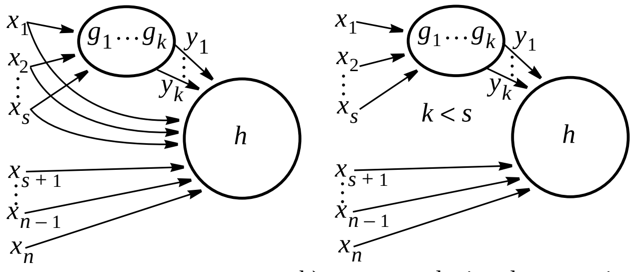
$$f = h(g_1(x_1, \ldots, x_s), \ldots, g_k(x_1, \ldots, x_s), x_{s+1}, \ldots, x_n)$$

along with the constraint $k < s$ (see Fig. 2-b).

Support-reducing decomposition is a form of disjoint decomposition in which the choice of composition and decomposition functions is carefully coordinated across the successive decomposition steps to optimize circuit structure. In addition, the decomposition is depth-reducing since it reduces the number of logic levels in the synthesized circuits by selecting decomposition according to the functional content of library primitives.[1]

We can now introduce a formal statement of the support- and depth-reducing decomposition. Referring to Fig. 2-a, in the generic decomposition case, a function f can be decomposed with respect to $g_1, g_2, \ldots, g_k$ using orthonormal expansion [2] $f = \sum t_i f_i$, where $t_i = \dot{g}_1 \dot{g}_2 \ldots \dot{g}_k$ $(0 \le i \le 2^k - 1)$; the set of s-variable functions $\{t_i\}$ forms an *orthonormal basis* in the expansion. The dots above g's in the expansion indicate either complemented or non-complemented form of the decomposition functions; their true phases are determined from the binary representation of i at each t_i. Each of the f_i's in the expansion can be selected from the range $f t_i \le f_i \le f + \bar{t}_i$. The support reduction condition is achieved

1. Note that this is slightly different from FPGA synthesis, where fanin count is the only notion of node complexity; use of look-up tables removes functional restrictions.



a) generic decomposition b) support-reducing decomposition

Fig. 2. Decomposition choices

by selecting f_i which is vacuous in the decomposition variables, and also ensuring that $k < s$ holds. The vacuous requirement on f_i implies that whenever $t_i \ne 0$ in the decomposition, f_i is unique, and is selected as a cofactor f_m, where m is a minterm from the basis function t_i. On the other hand, the depth reduction condition designates that g's must be selected as functions corresponding to library primitives.

For a given set of decomposition variables of size s there may or may not be a support-reducing decomposition. A necessary and sufficient condition for its existence is that the number of distinct cofactors f_m induced by the minterms from the space of decomposition variables is $\le 2^{s-1}$. (Here and later we assume that f is a completely specified function.) The distinct cofactors partition the minterms in the space of decomposition variables into *equivalence classes* $\{C_i\}$ such that minterms m and m' are in the same class iff $f_m = f_{m'}$.

According to the Roth and Karp classical decomposition condition [12], a decomposition exists iff there is no pair of minterms from the distinct equivalence classes which belong to the ON-set of a same function t_i. This *distinguishability requirement* implies that each of the basis elements $t_i \ne 0$ must be a subset of exactly one equivalence class. If the distinguishability requirement is not satisfied, then we would not be able to select cofactors f_i that are all vacuous in the decomposition variables, and therefore may not achieve support-reducing decomposition.

Example 1 Suppose we would like to decompose the function

$$\begin{aligned} f = {} & \bar{a}\bar{b}\bar{c}de + \bar{a}\bar{b}cd + \bar{a}b\bar{c}d + \bar{a}bc\bar{d}e + \\ & a\bar{b}\bar{c}d + a\bar{b}c\bar{d}e + ab\bar{c}\bar{d}e + abc\bar{d} \end{aligned}$$

with respect to decomposition variables $\{a, b, c\}$. The distinct cofactors and their equivalence classes induced by the space of these variables are listed below:

i	Cofactor, f_i	Equivalence class, C_i
0	de	$\bar{a}\bar{b}\bar{c}$
1	d	$\bar{a}\bar{b}c, \bar{a}b\bar{c}, a\bar{b}\bar{c}$
2	$\bar{d}e$	$\bar{a}bc, a\bar{b}c, ab\bar{c}$
3	$\bar{d}$	abc

Letting $\{t_i\}$ in the orthonormal expansion be $\{C_i\}$ we can write f in the factored form as

$$f = (\bar{a}\bar{b}\bar{c})de + (\bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c})d + (\bar{a}bc + a\bar{b}c + ab\bar{c})\bar{d}e + (abc)\bar{d}$$

There is a total of four basis elements which can be encoded with two decomposition functions. One possible encoding is

$$\begin{aligned} t_0 &= \bar{g}_1\bar{g}_2 \\ t_1 &= \bar{g}_1 g_2 \\ t_2 &= g_1\bar{g}_2 \\ t_3 &= g_1 g_2 \end{aligned}$$

These equations have a unique solution

$$\begin{aligned} g_1 &= ab + bc + ac \\ g_2 &= a \oplus b \oplus c \end{aligned}$$

yielding the following decomposed form for function f:

$$f = \bar{g}_1\bar{g}_2 de + \bar{g}_1 g_2 d + g_1\bar{g}_2\bar{d}e + g_1 g_2\bar{d}$$

■

A support-reducing decomposition degenerates to a non-disjoint decomposition if one or more g_j functions are selected as an inverter or a pass-through wire.

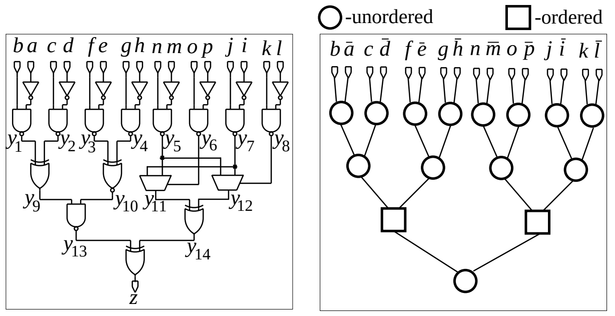
In recent years researchers have successfully applied the Roth and Karp decomposition condition in FPGA synthesis [8,11,13,18]. However, all these approaches are based on a recursive decomposition paradigm, and hence do not lend themselves easily to decompositions in the library-aware context.

3 Symmetries in Functional Specifications

A function exhibits symmetry in two variables if exchanging these variables leaves the function invariant [4]. This notion can be generalized to symmetries under phase assignment, where one of the two exchanged variables can be negated [5]. We term symmetries between variables as *first order* symmetries. For completely specified functions they form a transitivity relation which can be utilized for the efficient construction of groups of more than two symmetric variables in quadratic run time. Methods for the efficient identification of variable symmetries using binary decision diagrams [3] have recently been described in [10] and [15].

Table 1. Symmetries in benchmark circuits

Circuit	**Symmetry group counts of size *n***									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**≥10**
rd53					3					
rd73							3			
rd84								4		
9symml									1	
parity										1
pm1	4	6	7	1	3		1			
misex2	12	9	5	3	2	1	4	4		1
pcler8	24	19	2	2	2	2	2	1		1
z4ml		5	4							
lal	22	31	5	2	1	1	1	1		
pm4	3	10	10	5						
x4	175	50	39	27	4	9				
sct	24	34	6	13						
c8	67	2	1	1	1	1	1	2		
my_adder		135	17							
count	63	2	1	1	1	1	1	1	1	7
t481		8								
cordic	10	1	6	4						
comp		48								
mux	21									
i2	13			3						5



-unordered
-ordered
b a c d f e g h n m o p j i k l
y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8
y_9 y_{10} y_{11} y_{12}
y_{13} y_{14}
z
$b\,\bar{a}\;c\,\bar{d}\;f\,\bar{e}\;g\,\bar{h}\;n\,\bar{m}\;o\,\bar{p}\;j\,\bar{i}\;k\,\bar{l}$

a) circuit

b) structure of symmetries in the circuit function

Fig. 3. Reflection of the symmetric structural attributes of a function on a circuit structure

Symmetries between variables arise in many combinational circuits. To illustrate this, Table 1 captures symmetry group counts according to their sizes for some of the MCNC benchmark circuit [19]. They were collected for all the outputs of each listed benchmark. Blank entries in the table indicate 0's.

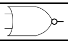
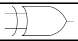
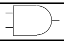
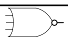
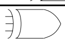
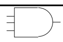
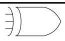
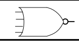
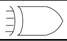
The notion of the first order variable symmetries can be extended to symmetries of a *higher order*: groups of inputs that can be simultaneously swapped without altering the functions. Higher-order group symmetries may exist between ordered, or partially ordered groups of variables. Analogous to the first order symmetries, these groups can be profitably used to improve circuit structure. In the constructive synthesis approach they tend to contract into a single variable, which can form symmetries with other variables in the functional specification of the remaining unimplemented logic.

Example 2 We consider the `t481` circuit from the MCNC benchmark suite. In its multilevel form it is given having 2072 simple gates, and is highly irregular. However, as follows from Fig. 3, the function of this circuit has a very compact realization (Fig. 3-a) reflecting the structural representation of its symmetric attributes (Fig. 3-b). The intrinsic symmetry relation depicted in Fig. 3-b can be also written as

$$\{\langle\{\{b,\bar{a}\},\{c,\bar{d}\}\},\{\{f,\bar{e}\},\{g,\bar{h}\}\}\rangle, \langle\{\{n,\bar{m}\},\{o,\bar{p}\}\},\{\{j,\bar{i}\},\{k,\bar{l}\}\}\rangle\}$$

where $\langle\ \rangle$ implies order on the symmetric groups of variables. The constructive nature of our algorithm allows us to obey this higher-order relation by restricting our attention to the first order symmetries on each iteration of the algorithm. This can be observed from the evolving symmetry relation between intermediate signals during circuit construction:

Table 2. Instances of symmetric libraries for the symmetric support- and depth-reducing decomposition

Support reduction	An instance of required cell library	# of Lib. Instances
2-to-1	  	1
3-to-1	 S_1 S_2  $S_{0,3}$ $MAJ3$ 	1
3-to-2	 $MAJ3$	3
4-to-1	$S_0 \ldots S_4$ $S_{0,1} \ldots S_{0,4}$ $S_{1,2} \ldots S_{1,4}$ $S_{2,3}$ $S_{2,4}$ $S_{3,4}$	1
4-to-2	$S_{0,1}$ $S_{0,3}$ $S_{0,4}$ $S_{1,2} \ldots S_{1,4}$ $S_{2,3}$ $S_{2,4}$	15
4-to-3	  $MAJ4$	140

$level(2) = \{ \langle \{y_1, y_2\}, \{y_3, y_4\} \rangle, \langle \{y_5, y_6\} \{y_7, y_8\} \rangle \}$,
$level(3) = \{ \{\bar{y}_9, y_{10}\}, \{\bar{y}_{11}, y_{12}\} \}$,
$level(4) = \{y_{13}, y_{14}\}$. ■

4 Cell Library Construction

All decomposition functions g_i in the algorithm are selected as functions from a cell library. Therefore, to guarantee feasibility of a support- and depth-reducing decomposition the cell library must be sufficiently complete. By using such a library we are able to implement a function's existing symmetry structure, and utilize its original symmetric attributes as synthesis progresses. In this section we derive the required primitive libraries.

Construction of complete symmetric libraries is based on the Roth and Karp condition for the existence of a decomposition. For a given set of decomposition functions the condition imposes a distinguishability requirement on the minterms from distinct equivalence classes. This requirement is to have at least one decomposition function that places a pair of minterms from the distinct equivalence classes separately into its ON-set and OFF-set. Using this requirement we can derive all gate type combinations required to achieve support-reducing decomposition for a particular partition of minterms into equivalence classes. Thereby all feasible minterm partitions for a particular decomposition type can be examined, and the smallest gate library covering all of the expected minterm partitions can then be constructed.

Table 2 summarizes the results of our derivation of symmetric libraries of four or fewer variables. The first column of the table lists the desired s-to-k subcircuit, implementing s decomposition variables with k s-input primitives. The second column gives one possible smallest gate library for this type of decomposition, whenever it is possible. Some of the cells in the table are listed using the S_α notation, where subscript α is a list of numbers identifying minterm weights in the ON-set of a function. For example, the $S_{0,3}$ symbol at the 3-to-1 entry denotes the function $abc + \bar{a}\bar{b}\bar{c}$. For each of the s-to-k decompositions in the table we have counted the number of possible complete smallest libraries. These numbers are listed in the third column, and their variety counts are given up to the complement of the cell functions.

```
function re-express($f$, $g_i$, $P$) {
    if $\forall(m, m' \in P) f_m = f_{m'}$ then return $f_m$;
    assert($i \le k$);
    if $P \le g_i$ or $P \le \bar{g}_i$ then return re-express($f$, $g_{i+1}$, $P$);
    $f^0 \leftarrow$ re-express($f$, $g_{i+1}$, $\bar{g}_i \cdot P$);
    $f^1 \leftarrow$ re-express($f$, $g_{i+1}$, $g_i \cdot P$);
    return $\bar{y}_i \cdot f^0 + y_i \cdot f^1$;
}
```

Fig. 4. The re-express algorithm in the support- and depth-reducing decomposition

It follows from the table that a complete library for 2-to-1 support-reducing decomposition of two symmetric decomposition variables is $\{\bar{a}\bar{b}, a \oplus b, ab\}$. The only variations on this library would replace some of its functions with their complements. To accommodate symmetric variables with different phases the library should also include an inverter. It is important to note that two symmetric decomposition variables do not always yield a support-reducing decomposition. Similarly, 3-to-1, 4-to-1 and 4-to-2 decompositions are also not always possible for symmetric decomposition variables. However, there are always 3-to-2 and 4-to-3 support-reducing decompositions when decomposition variables are symmetric.

A smallest complete library for three symmetric decomposition variables corresponds to one of the following sets of primitives (up to their complements): $\{a \oplus b \oplus c, MAJ3\}$, $\{a \oplus b \oplus c, abc + \bar{a}\bar{b}\bar{c}\}$, or $\{MAJ3, abc + \bar{a}\bar{b}\bar{c}\}$. A similar result for four variables states that a library composed of three gates is the smallest. It follows that a smallest complete library for support- and depth-reducing decomposition in which at least three decomposition variables are symmetric can be selected as a smallest 3-to-2 library, a smallest 2-to-1 library, an inverter and a wire. A wire makes it possible to implement only a subset of variables.

5 Overall Synthesis Flow

Our constructive synthesis algorithm is based on the iterative application of the following three steps: (1) extraction of logic to be implemented, (2) a subcircuit implementation of the logic, and (3) re-expression of the remaining unimplemented logic in terms of the newly created subcircuit. We describe each of these steps below.

5.1 Support Selection

Since groups of three or more symmetric variables always yield a support-reducing decomposition they are considered as prime candidates for the decomposition variables. We also consider the possibility of combining a group of two symmetric variables with other variables in the support selection. (A pair of two-variable mutually symmetric groups always leads to a support-reducing decomposition, except for one case). Among these candidates, a subset of three or four variables is selected which minimizes network depth.

When a function has no symmetries we rely on a heuristic of existentially quantifying one or two variables. If the new function has symmetric variables, then these variables are selected along with the quantified variables as decomposition variables. Such a selection of decomposition variables using quantification suggests the inclusion of a multiplexer into the library. If this heuristic fails to yield a feasible decomposition we greedily select a set of decomposition variables which yields the least number of equivalence classes until the decomposition is feasible.

These variables are then implemented with a single level of logic subcircuit using primitives from the technology library.

5.2 Subcircuit Construction

In the subcircuit construction step, decomposition functions are selected as library primitives. Their selection is performed by a branch-and-bound procedure which involves selecting a gate from the library, instantiating it with the decomposition support, and then trying to extend the partially-built subcircuit. The procedure recursively identifies a subset of gates such that for any two minterms belonging to the distinct equivalence classes, there is at least one function g_i that distinguishes these minterms by placing them separately into the ON-set and the OFF-set of g_i.

The result of the computation is a single-level subcircuit (with possible inverters at the gate inputs) that constitutes the decomposition functions g_i.

5.3 Re-Expression of a Function

When the g_i decomposition functions are determined, the function f is re-expressed according to the support-reducing decomposition. The support-reducing decomposition is not unique, and hence often provides flexibility in selecting a composition function. We have therefore developed an efficient algorithm (see Fig. 4) which selects the composition function h to maximize its symmetry. It presumes an ordering on the $g_1, \ldots, g_k$ decomposition functions and is invoked as *re-express*$(f, g_1, \mathbf{1})$, where $\mathbf{1}$ denotes the entire minterm space of decomposition variables. In the algorithm, new auxiliary variables y_i, associated with each of the g_i functions, become arguments to the composition function h.

6 Experimental results

We have implemented the above concepts in the M31 synthesis tool. All functions are internally represented by BDDs and manipulated using the CUDD package [17].

To evaluate the performance of M31, we synthesized the subset of MCNC benchmarks listed in Table 1 and compared them against those generated by the SIS-1.2 synthesis system [16]. These benchmarks are originally given as two-level specifications or multilevel netlists. Before synthesizing benchmarks in the multilevel format, M31 first collapsed them to two-level form. Synthesis using SIS was performed using `script.rugged.` Using this script, multilevel netlists were synthesized from both the original multilevel specification and the collapsed two-level form and the best variant is reported. For the `my_adder.blif` and `comp.blif` netlists only the multilevel form was used, since for these circuits SIS-1.2 was not able to obtain a collapsed two-level form due to their large cover sizes.

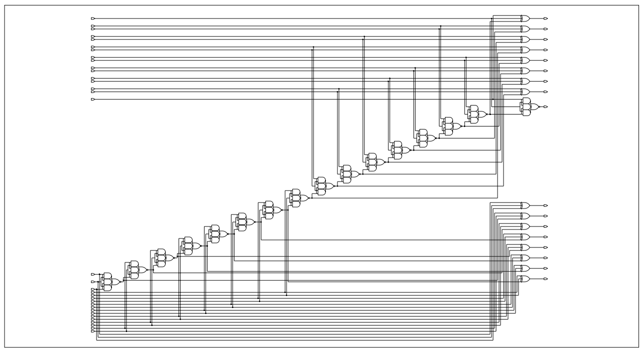
Table 3. SIS and M31 synthesis results

Circuit	nodes		logic levels		avg. top. wire length		non-mcnc gates		wires		symm.
	SIS	M31	SIS	M31	SIS	M31	SIS	M31	SIS	M31	
rd53	21	10	8	3	1.98	1.15	1	4	48	33	1.00
rd73	44	8	16	3	2.80	1.25	2	8	92	24	1.00
rd84	90	23	15	5	2.05	1.34	3	7	214	58	1.00
9symml	120	13	13	6	2.19	1.16	0	8	310	36	1.00
parity	15	8	4	3	1.00	1.04	0	7	30	23	1.00
pm1	33	30	8	4	1.62	1.49	0	0	104	100	0.74
misex2	62	76	9	6	2.26	1.13	0	5	139	220	0.71
pcler8	61	78	12	7	3.61	1.79	4	12	128	194	0.61
z4ml	22	9	10	3	2.51	1.20	2	7	47	25	0.54
lal	68	63	12	6	2.09	1.60	4	0	140	127	0.50
pm4	150	41	19	8	3.63	1.51	5	12	341	98	0.45
x4	229	235	16	8	2.95	2.25	2	51	510	518	0.43
sct	52	52	31	6	6.44	1.72	0	3	108	107	0.40
c8	65	65	10	5	2.54	1.43	0	25	168	148	0.29
my_adder	156	96	35	5	4.91	1.32	0	84	285	276	0.20
count	90	81	20	7	5.39	2.05	0	11	203	198	0.20
t481	23	23	5	5	1.21	1.20	0	2	38	40	0.12
cordic	45	38	9	7	1.83	1.48	4	4	103	91	0.10
comp	85	55	13	8	2.04	1.46	0	24	168	134	0.06
mux	16	16	5	5	1.32	1.32	15	15	47	47	0.05
i2	81	89	8	12	1.31	1.24	0	3	293	302	0.05

Experimental results are given in Table 3. The data were obtained for the `mcnc.genlib` library which was extended with three additional gates: 3-input XOR, 3-input majority, and a 2-to-1 multiplexer. It is important that aside from the symmetric gates, the MCNC library provides partially symmetric gates, which enable us to handle partially symmetric and asymmetric decomposition variables.

As a rough measure of symmetry in the logic specifications, we use the reciprocal of the number of symmetry groups: a symmetry measure of 1 corresponds to a fully symmetric function (1 symmetry group containing all variables), whereas an *n*-input function that lacks any variable symmetries has symmetry value of 1/*n*. These ratios, averaged across all circuit outputs, are given in the last column of the table. Comparison with SIS-1.2, shows that significantly better implementations are possible: M31-generated circuits have many fewer logic levels, much lower average topological wire length, and fewer connections. These improvements seem to correlate, to a first order of approximation, with the symmetry measure.

To demonstrate the extent of spatial and temporal trade-offs that can be explored by the M31 synthesis tool, Fig. 5


a) area minimized adder

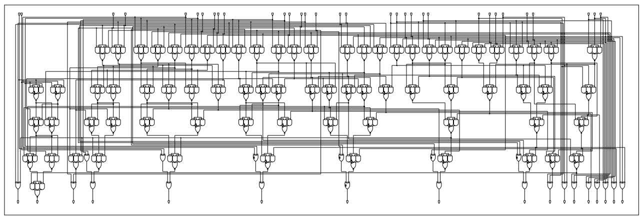

b) logic depth minimized adder

Fig. 5. Two adders with different properties synthesized constructively for the `my_adder` MCNC benchmark

gives different implementations of a 16-bit adder. The circuits were synthesized by varying the order in which decomposition variables were selected on each iteration of the constructive synthesis algorithm. The circuit in Fig. 5-a is a ripple-carry adder which has small active area at the expense of 16 levels of logic. On the other hand, the circuit in Fig. 5-b is a 5-level variant on a carry lookahead adder, which needs a larger number of gates.

M31 run times for highly symmetric functions are comparable to those of SIS-1.2. Its performance degrades as functions exhibit more asymmetry. This is hardly surprising since asymmetric functions possess different structural properties than those studied in this work. We found that the computation time of M31 is dominated by the subcircuit identification step of the algorithm. It is also important that computation of symmetries becomes expensive as the representation of a function increases. For completely-specified functions it is provable, though, that decomposition preserves symmetry in the non-decomposition variables of f in the re-expressed f. Thus, they need to be computed only initially, and then simply updated as synthesis evolves.

7 Conclusions and Future Work

The improved quality of the above implementations is the result of a coordinated strategy that ties functional structure (symmetries in this case) to an appropriately outfitted cell library through a decomposition procedure that is aware of both. This is evident by examining the results of SIS-1.2 synthesis when these extra library gates are made available to it: with one or two exceptions, SIS-1.2 had no use for these extra gates.

In our future work we plan to research other structural attributes, beyond symmetry, that might prove useful during decomposition. Differences between datapath and control logic functions may provide valuable insights into functional structure.

References

[1] L. Benini, P. Vuillod, and G. De Micheli. Iterative re-mapping for logic circuits. *IEEE TCAD IC*, CAD-17(10):948–964, October 1998.

[2] F. M. Brown. *Boolean Reasoning*. Kluwer Academic Publishers, Boston, 1990.

[3] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE TC*, C-35(6):677–691, August 1986.

[4] D. L. Dietmeyer and P. Schneider. Identification of symmetry, redundancy and equivalence of boolean functions. *IEEE TEC*, EC-16(6):804–807, December 1967.

[5] C. R. Edwards and S. L. Hurst. A digital synthesis procedure under function symmetries and mapping methods. *IEEE TC*, C-27:985–997, 1978.

[6] B.-G. Kim and D. L. Dietmeyer. Multilevel logic synthesis of symmetric switching functions. *IEEE TCAD IC*, 10(4):436–446, April 1991.

[7] W. Kunz and D. Stoffel. *Reasoning in Boolean Networks*. Kluwer Academic Publishers, 1997.

[8] Y. T. Lai, M. Pedram, and Sarma B. K. Vrudhula. BDD based decomposition of logic functions with application to FPGA synthesis. In *Proc. 30th DAC*, pages 642–647, June 1993.

[9] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness. Logic decomposition during technology mapping. In *Proc. Int. Conf. Computer Design*, pages 263–271, 1995.

[10] D. Moller, J. Mohnke, and M. Weber. Detection of symmetry of boolean functions represented by ROBDDs. In *Proc. ICCAD*, pages 680–684, October 1993.

[11] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli. Optimum functional decompositions using encoding. In *Proc. 31st DAC*, pages 408–414, June 1994.

[12] J. P. Roth and R. Karp. Minimization over boolean graphs. *IBM J. Res. and Develop.*, 6(2):227–238, April 1962.

[13] H. Sawada, T. Suyama, and A. Nagoya. Logic synthesis for look-up table based FPGAs using functional decomposition and support minimization. In *IEEE ICCAD*, pages 353–358, November 1995.

[14] H. Sawada, S. Yamashita, and A. Nagoya. Restructuring logic representations with easily detectable simple disjunctive decompositions. In *DATE*, pages 755–759, February 1998.

[15] C. Scholl, D. Moller, P. Molitor, and R. Drechsler. BDD minimization using symmetries. *IEEE TCAD IC*, 18(2):81–100, February 1999.

[16] E. M. Sentovich. SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, UC Berkeley, May 1992.

[17] F. Somenzi. *CUDD: CU Decision Diagram Package*. University of Colorado, Boulder, 2.1.2 edition, April 1997.

[18] B. Wurth, K. Eckl, and K. Antreich. Functional multiple-output decomposition: theory and an implicit algorithm. In *Proc. 32nd DAC*, pages 54–59, June 1995.

[19] S. Yang. *Logic synthesis and optimization benchmarks user guide – ver. 3.0*. MCNC, Res. Triangle Park, NC, Jan. 1991.