A Fast Global Gate Collapsing Technique for High Performance Designs using Static CMOS and Pass Transistor Logic *

Yanbin Jiang†, Sachin S. Sapatnekar ‡ and Cyrus Bamji*
† Department of ECE, Iowa State University, Ames, IA 50011 (ybjiang@iastate.edu)
‡ Department of ECE, University of Minnesota, Minneapolis, MN 55455 (sachin@ece.umn.edu)
* Cadence Design Systems, San Jose, CA 95052 (cyrus@cadence.com)

Abstract

A new design methodology for mapping circuits is discussed in this paper. It proposes two new techniques for mapping circuits. The first method, known as the odd-level transistor replacement (OTR) method, has a goal that is similar to that of technology mapping, but without the restriction of a fixed library size. The second technique, the Static/PTL method, uses a mix of static CMOS and pass transistor logic (PTL) to realize the circuit, using the relation between PTL and binary decision diagrams. The methods are very efficient and can handle all of the ISCAS85 benchmark circuits in minutes. A comparison of the results with traditional technology mapping using SIS on different libraries shows an average delay reduction about 40% for OTR, and an average delay reduction above 50% for the Static/PTL method.

1 Introduction

Technology mapping has been a cornerstone in the logic synthesis process and this area has been well studied in the past. Existing technology mapping techniques can be classified into four categories: rule-based mapping [5], graph matching [6], direct mapping [7] and functional mapping [8]. Traditional methods for technology mapping are directed towards a *specific* library and are targeted towards objectives such as minimizing the circuit delay, minimizing the area and reducing the power dissipation. Using a pre-characterized library methodology has the inherent major disadvantage that the quality of the results is dependent on the richness of the library.

*This work was supported in part by a Lucent Technologies DAC Graduate Scholarship and the National Science Foundation under contracts MIP-9502556 and MIP-9796305.

In our work, individual complex gates are generated on the fly, instead of using a pre-characterized library. The translation of this new gate to a layout can be performed using a module generator. The design methodology, shown in Figure 1, can be extended to work with the traditional methodology, so that in case a library is available, the complex gates may be implemented using either the cells in the given library or the virtual library.

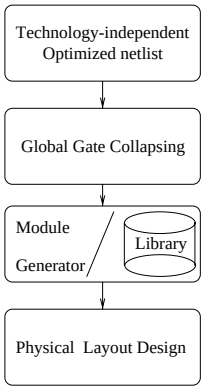

Figure 1. New design methodology.

The focus of this paper is on the *gate collapsing* phase which is the basis for this new design methodology; this is the phase where the complex gates are generated. The discussion about the module generator for layout of the complex gates is beyond the scope of this paper.

The essential idea of gate collapsing is to begin with a decomposition of the circuit and then to combine or collapse these simple gates into more complex gates. Our procedure works on a virtual library that is assumed to have all types of cells so that the global gate collapsing technique can have the full flexibility of finding the optimum possible combination of standard gates in a network.

The input to global gate collapsing comes from the output of technology-independent optimization, and the result of the procedure is a network where the input netlist is collapsed into an optimal set of complex gates corresponding to that decomposition. In this work, we consider gate collapsing on the circuit using two forms of logic styles: complex static CMOS gates, and pass transistor logic (PTL).

The matching method of traditional technology mappers such as MIS [2] cannot be applied on the virtual library since the number of possible templates is far too large. Therefore, this paper develops techniques for generating the complex gates for the virtual library. We propose two methods for this purpose:

- Topological mapping: Odd-level transistor replacement (OTR) for mapping to complex static CMOS gates
- Boolean functional mapping: Using binary decision diagrams (BDD's) to map logic to pass transistor logic.

The organization of the paper is as follows. The first technique for gate collapsing, called the OTR method, is described in Section 2. Next, we consider the problem of mixed static/PTL mapping in Section 3. Experimental results are presented in Section 4, followed by concluding remarks in Section 5.

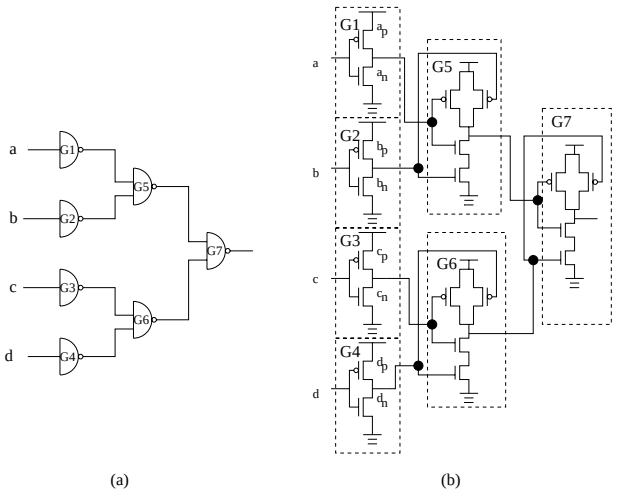

Figure 2. A circuit for gate collapsing.

2 Odd-level Transistor Replacement (OTR) Method

2.1 An Example

We will now present a method for building complex gates, based on a simple topological technique that permits subcircuits with an odd number of gate levels to be collapsed into a single complex gate.

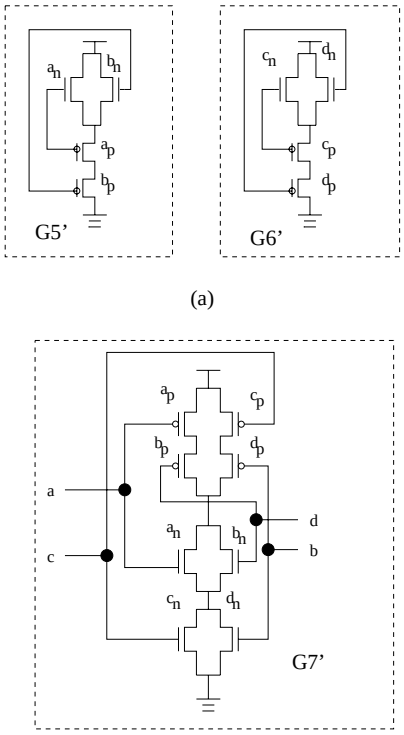

Figure 3. The OTR gate collapsing procedure.

The basic idea of the OTR method is to use the pull-down (pull-up) transistor structure from the gates at the previous level gates to replace the pull-up (pull-down) transistors of the gates at the next level. To illustrate this, consider the circuit in Figure 2(a) consisting of gates G1 through G7. This structure has 20 transistors in all, and a transistor-level version is shown in Figure 2(b).

We use the pull-down (pull-up) transistors in G1 and G2 to replace the pull-up (pull-down) transistors in G5 to obtain the gate G5', a nontraditional static CMOS gate, shown in Figure 3(a). Similarly, the transistors in G3 and G4 are inserted into G6 to get another nontraditional static CMOS gate, G6'. We treat these nontraditional gates as intermediate synthesis stages and we will eliminate them in the next step by performing

the same operation, replacing the pull-down (pull-up) block of G7 by the pull-up (pull-down) blocks of the intermediate gates G5' and G6', respectively. The detailed illustration of the final collapsed gate is shown in Figure 3(b). Note that the final implementation has only 8 transistors, a transistor count reduction of 60%.

From the principle illustrated in this example, it is easy to see that if we collapse an even number of levels of gates, we will be left with a nontraditional static CMOS gate, whereas if we collapse an odd number of levels, we will return to the traditional CMOS complex gate structure, and therefore we call this technique the *odd-level transistor replacement (OTR) method.*

2.2 Delay Estimation

The delay is characterized in the look-up table as a function of the switching position, transistor size, input slope S, and loading capacitance C. We assume in our implementation that each transistor in a gate has the same size. This makes the task of layout easier, and compacts the size of the look-up table. Some further improvements are possible by allowing transistors to be sized individually. Our experimental results show that even under our implementational assumption, substantial area/performance improvements are possible. Moreover, the theoretical framework presented here can be extended to the case of nonuniform sizes.

Given a switching position and a transistor size, a traditional look-up table is a two-dimensional array of values parameterized by S and C, as shown in Figure 4(a). This table requires a large amount of memory

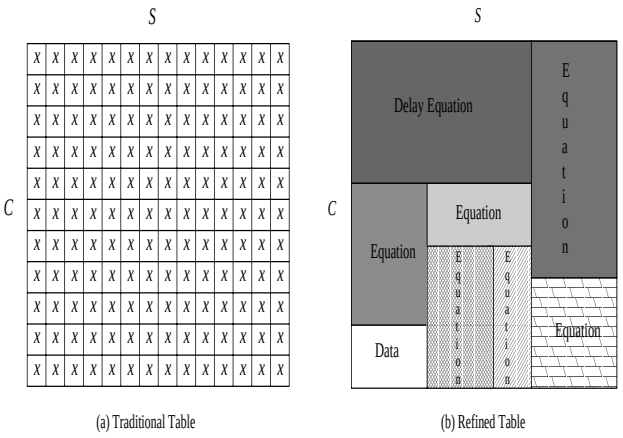

Figure 4. Look-up tables.

which can make the look-up speed slow since it is impossible to store all such tables for all possible switching positions and transistor sizes in the cache or in the RAM. In order to refine this look-up table method, we compact the information in this table into a delay characteristic delay equation for each such two-dimensional array. For purposes of characterization, we find a least-squares fit to the characteristic delay equation from [10] which is of the type used by Synopsys:

$$D = \alpha * S + \beta * C + \gamma * S * C + \omega$$

where α, β, γ, ω are constants. However, if we attempt to find a single delay equation for the entire table, the accuracy of the characterization may be poor. Therefore, we use a set of equations that capture the information embedded in a subset of the data, ensuring that the accuracy of each such fit is within a prescribed range, ϵ. The entire data can be fitted accurately to a small set of delay equations, and any data points that have an error larger than ϵ from the set of equations are stored as pure data. The overall structure of the storage is as shown in Figure 4(b).

The procedure for finding the values of α, β, γ and ω requires a least-squares minimization of the following form:

$$\min F = \sum_i [(\alpha * S_i + \beta * C_i + \gamma * S_i * C_i + \omega) - D_i]^2$$

where the summation is performed over all SPICE-measured data points i, and C_i, S_i and D_i, respectively, denote the load capacitance, slope and delay corresponding to the i^{th} data point. This unconstrained minimization can be performed by setting the partial derivatives of F with respect to each of the parameters to zero, i.e., $\frac{\partial F}{\partial \alpha} = 0$; $\frac{\partial F}{\partial \beta} = 0$; $\frac{\partial F}{\partial \gamma} = 0$; $\frac{\partial F}{\partial \omega} = 0$.

We solve the above system of linear equations to find the values of α, β, γ and ω.

2.3 Outline of the Algorithm

We now present a dynamic programming based approach to solve the problem of area/power minimization under delay constraints. In Section 5, we show the results of applying this technique to find the minimum delay, but the method can equally well be used to solve the constrained optimization problem.

To understand the difficulty of this problem, we observe that technology mapping, a special case of global gate-collapsing, is known to be NP-complete for directed acyclic graph structures [6]. A technique that has been routinely and successfully used in technology mapping is to decompose a DAG into a set of trees and to perform mapping on those trees (for example, in [3, 6]), with the trees being selected in such a way that they are all rooted at gates with multiple fanouts

or at gates at the primary output. We persist with this approach in our work.

The algorithm is based on dynamic programming and uses OTR combinations to generate possible complex gates. As in [3], we begin with a 2-input NAND gate decomposition of the circuit, though we emphasize that any other initial circuit can also be used. The pseudocode for the algorithm is shown below:

Algorithm Outline

```
Input:  Initial circuit decomposed into
 inverters and 2-input NAND gates.
Output:  Optimum network of complex gates.
{
 levelize the circuit
 find_roots
 sort_roots
 from primary inputs to primary outputs
  for each root generate tree
  apply dynamic programming
  for each node in the tree from leaf nodes
  to the root
   find_all_possible_collapsing_solutions
   store non_inferior_solutions
   [Area, Delay]
 find optimum solution based on all
 generated noninferior states
}
```

The dynamic programming procedure [4] proceeds by associating a set of states with each node, where a node corresponds to a gate output. A state corresponds to a partial solution that corresponds to a possible configuration of collapsed gates for the subtree rooted at that node. The state information for each node is a pair [Area,Delay], calculated from the primary inputs up to that node. The complex gates are chosen so that the number of series-connected mosfets on a path to V_{dd} or ground does not exceed a user-specified number k.

Dynamic programming proceeds by enumerating the possible states at a node, and eliminating all partial solutions at each step that are provably suboptimal. Finally, when all noninferior states have been enumerated, the optimal state is chosen and the corresponding circuit configuration is determined.

3 Combined Static CMOS/ Pass Transistor Logic Design

3.1 Fundamentals

In this section, we develop techniques for the synthesis of circuits with a combination of static CMOS and PTL and present a procedure that partitions a circuit into static CMOS and PTL to achieve the minimum delay.

In dealing with PTL, a designer must be aware of the following limitations:
(1) For an nMOS (pMOS) transistor, the low-to-high (high-to-low) transition is imperfect and therefore PTL cannot achieve full voltage swings, resulting in reduced noise margins.

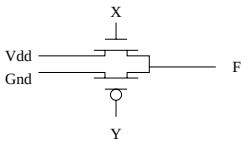

Figure 5. An example of a sneak path.

(2) It is possible for sneak paths between V_{dd} and ground to exist unless the circuit is designed carefully. An example of sneak paths is shown in Figure 5: if $X = 1$ and $Y = 0$ at the same time, then F is connected to connect both power supply and ground simultaneously.

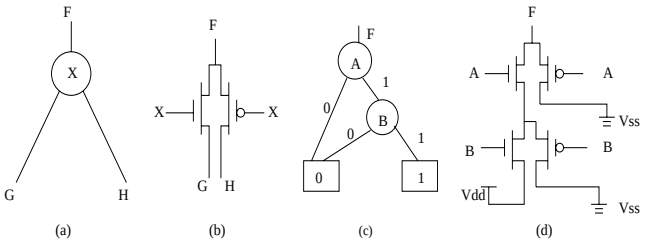

Figure 6. Circuit example.

Pass transistors can be used to build a 2-input multiplexer, leading to a one-to-one correspondence between BDD's and their PTL implementations. Since a BDD can represent any logic function, we can use the BDD representation to directly arrive at a PTL implementation of a complex gate. In Figure 6, we show the correspondence between a BDD node and a pass transistor, build the BDD representation for the 2-input AND gate, and arrive at the pass transistor implementation of the BDD. Figure 6(a) shows a BDD node whose PTL implementation is shown in Figure 6(b). Using this as a basis for design, we take the BDD in

Figure 6(c), representing a two-input AND gate, and build the the corresponding PTL implementation as shown in (d).

3.2 Outline of the Algorithm

The dynamic programming approach for gate collapsing proposed in Section 2.4 is extended to give us a technique for building mixed static/PTL circuits. The basic idea is to use BDD's to represent a candidate logic function that can be implemented in PTL during dynamic programming. The implementation uses the BDD package described in [1].

In using PTL, as in the case of complex static gates, we must ensure that the number of pass transistors in series should be no more than a predetermined number p. In other words, while generating BDD's, we do not permit the depth of the BDD to become larger than p, and at that point, we force the use of a static gate at the fanout. The final circuit is likely to contain pieces of pass transistor logic that are isolated from each other by static CMOS gates. The purpose of the static CMOS gates is to isolate the pass-transistor parts and to restore the level of degraded signals.

The dynamic programming approach here is used to determine how the circuit should be partitioned between static CMOS and PTL implementations, with OTR-based gate collapsing being used for the static CMOS segment. In our current implementation, we use a PTL delay model that is similar to that used for the static CMOS logic, with the constants being altered. However, we stress that more complex and accurate delay models can also be incorporated into the optimizer.

For both the OTR and the static/PTL methods, we need to build complex gates, either in static form or as PTL. Suppose that for each node, it is possible to build C possible complex gates, that a complex gate can have a maximum of I inputs, and that each node can have up to M [Area, Delay] pairs stored during dynamic programming. Therefore, for each node, the amount of computation for calculating the [Area, Delay] pairs is $O(C \cdot I \cdot M)$. In general, C and I are bounded, and so the computation complexity can be written as $O(M)$. Since the dynamic programming technique handles each of the N gates in the circuit, the computation complexity of our algorithm is $O(N \cdot M)$.

4 Experimental Results

The methods described in this paper were both implemented in C on a SUN Sparc 1/170 workstation. For purposes of comparison, results were generated using SIS [9], OTR (Section 2) and mixed-static CMOS/PTL methods (Section 3) on the ISCAS'85 benchmark circuits. The circuits were first decomposed into inverters and 2-input nand gates network using SIS. Next, we performed a minimum circuit delay technology mapping in SIS for the circuits using the libraries nand-nor.genlib, mcnc.genlib and lib2.genlib. The libraries are modified and recharacterized under our SPICE parameters to have three cells of each type. We set the the values of the parameters k and p (described in Sections 2.4 and 3.3, respectively) to 4 in our work.

A comparison of the results of SIS and OTR, shows that that OTR provides better results than SIS results, with average delay reductions of over 40%, and area reductions of around 10%. A comparison of Static CMOS/PTL results with the results of SIS and OTR show much greater performance enhancements from the use of PTL in the circuits. The average delay reduction is about 50% and the area reduction above 70% over the results of SIS.

These results are indicative of the the power of our technique, and it is important to note that SIS simply cannot work in our new design methodology because as it cannot work on a virtual library, and requires all allowable gates to be listed and characterized in the library, which could be a prohibitive overhead. Our methods are fast and the largest ISCAS85 circuit can be handled in minutes.

5 Conclusion

We have presented the idea of global gate collapsing for pure static CMOS designs, and of using BDD's to realize mixed technology design using a combination of static CMOS and PTL. Our goal has been to present a general technique for performing overall circuit optimization using purely topological and Boolean functional techniques for static CMOS and small BDD's for PTL. The results obtained show that both techniques are very fast and show significant improvements over existing approaches.

Acknowledgements

The authors would like to thank Ruchir Puri, Leon Stok and Daniel Brand for the discussion about the pass-transistor logic work.

References

[1] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a bdd package. *Proceed-*

Table 1. Experimental Results of SIS and OTR Methods

Circuit	nand-nor.genlib			mcnc.genlib			lib2.genlib			OTR method		
	Minimum Delay(ns)	Area (unit)	CPU Time(ns)	Minimum Delay(ns)	Area (unit)	CPU Time(s)	Minimum Delay(ns)	Area (unit)	CPU Time(s)	Minimum Delay(ns)	Area (unit)	CPU Time(ns)
C432	50.68	8072	10.6	45.97	7640	16.3	43.29	7824	14.1	24.53	3354	4.86
C499	39.11	13880	16.8	36.66	12640	27.4	32.59	13088	23.8	24.15	6912	7.93
C880	35.91	10808	13.1	33.36	10652	22.8	31.82	10255	21.2	24.08	5573	6.99
C1355	43.12	16192	19.0	40.93	16752	30.9	38.35	17594	28.2	24.99	7392	7.87
C1908	52.64	19696	24.3	47.75	18464	37.6	41.07	20405	33.2	30.07	11004	13.49
C2670	50.30	26416	31.5	44.19	23008	57.1	38.77	21688	59.2	30.41	16098	22.29
C3540	71.86	27328	46.8	65.38	29328	74.4	60.24	31264	65.7	52.36	21600	29.30
C5315	66.52	59024	73.0	64.10	56088	129.1	60.83	51042	107.5	35.50	35670	60.31
C6288	197.47	54886	78.1	195.65	54762	143.7	192.66	53792	129.3	100.66	28992	23.83
C7552	61.32	62680	155.3	55.49	59936	371.2	51.15	57224	292.3	28.96	43896	70.02
Avg. Imp.	42.3%	43.2%		37.7%	41.2%		32.3%	41.0%				

Table 2. Experimental Results of SIS and PTL Methods

Circuit	nand-nor.genlib			mcnc.genlib			lib2.genlib			Static CMOS/PTL method		
	Minimum Delay(ns)	Area (unit)	CPU Time(ns)	Minimum Delay(ns)	Area (unit)	CPU Time(s)	Minimum Delay(ns)	Area (unit)	CPU Time(s)	Minimum Delay(ns)	Area (unit)	CPU Time(ns)
C432	50.68	8072	10.6	45.97	7640	16.3	43.29	7824	14.1	19.24	1372	190.90
C499	39.11	13880	16.8	36.66	12640	27.4	32.59	13088	23.8	19.50	3098	196.66
C880	35.91	10808	13.1	33.36	10652	22.8	31.82	10255	21.2	19.18	2432	168.50
C1355	43.12	16192	19.0	40.93	16752	30.9	38.35	17594	28.2	20.77	3450	376.49
C1908	52.64	19696	24.3	47.75	18464	37.6	41.07	20405	33.2	25.28	4753	296.05
C2670	50.30	26416	31.5	44.19	23008	57.1	38.77	21688	59.2	26.75	5980	621.61
C3540	71.86	27328	46.8	65.38	29328	74.4	60.24	31264	65.7	36.20	9059	887.46
C5315	66.52	59024	73.0	64.10	56088	129.1	60.83	51042	107.5	25.92	11601	990.18
C6288	197.47	54886	78.1	195.65	54762	143.7	192.66	53792	129.3	88.33	14403	1306.02
C7552	61.32	62680	155.3	55.49	59936	371.2	51.15	57224	292.3	21.74	18350	1093.61
Avg. Imp.	54.0%	76.2%		50.3%	75.4%		46.0%	75.4%				

ings of the ACM/IEEE Design Automation Conference, pages 40–45, 1990.

[2] R. Brayton, E. Detjens, S. Krishna, T. Ma, P. McGeer, L. Pei, N. Phillips, R. Rudell, R. Segal, A. Wang, R. Yung, and A. Sangiovanni-Vincentelli. Multiple-level logic optimization system. *Proceedings of the International Conference on Computer-Aided Design*, pages 356–359, 1986.

[3] K. Chaudhary and M. Pedram. A near optimal algorithm for technology mapping minimizing area under delay constraints. *Proceedings of ACM/IEEE Design Automation Conference*, pages 492–498, 1992.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Introduction to algorithms. *McGraw-Hill Book Company, New York, New York*, 1990.

[5] D. Gregory, K. Bartlett, A. de Geus, , and G. Hachtel. Socrates: A system for automatically synthesizing and optimizing combinational logic. *Proceedings of the 23rd Design Automation Conference*, pages 79–85, 1986.

[6] K. Keutzer. Dagon: technology binding and local optimization by dag matching. *Proceedings of ACM/IEEE Design Automation Conference*, pages 341–347, 1987.

[7] M. Lega. Mapping properties of multi-level logic synthesis operations. *Proceedings of the IEEE International Conference on Computer Design*, pages 257–260, 1988.

[8] F. Mailhot and G. DeMicheli. Technology mapping using boolean matching and don't care sets. *Proceddings of the European Design Automation Conference*, pages 212–216, 1990.

[9] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldhana, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. *Technical Report UCB/ERL M92/41, Electronics Research Laboratory University of California at Berkeley, Berkeley, CA*, May 1992.

[10] D. F. Wong. A fast and accurate technique to optimize characterization tables for logic synthesis. *Proceedings of the ACM/IEEE Design Automation Conference*, pages 337–340, 1997.