

Technology Mapping for High Performance Static CMOS and Pass Transistor Logic Designs *

Yanbin Jiang †

Sachin S. Sapatnekar ‡

Cyrus Bamji *

† Department of ECE, Iowa State University, Ames, IA 50011

‡ Department of ECE, University of Minnesota, Minneapolis, MN 55455

* Cadence Design Systems, San Jose, CA 95134.

Abstract

Two new techniques for mapping circuits are proposed in this paper. The first method, called 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, and maps a circuit to a virtual library of complex static CMOS gates. The second technique, the Static CMOS/PTL method, uses a mix of static CMOS and pass transistor logic (PTL) to realize the circuit, and utilizes the relation between PTL and binary decision diagrams. The methods are very efficient and can handle all of the ISCAS'85 benchmark circuits in minutes. A comparison of the results with traditional technology mapping using SIS on different libraries shows an average delay reduction above 18% for OTR, and an average delay reduction above 35% for the Static CMOS/PTL method, with significant savings in the area.

1 Introduction

Technology mapping is a cornerstone of the logic synthesis process and this area has been well studied in the past. Most existing techniques for technology mapping are based on static pre-characterized libraries, and can be classified into four categories: rule-based mapping [13], graph matching [15], direct mapping [18] and functional mapping [20]. However, with changes in device technologies and the use of more complex static CMOS gates (for example, in silicon-on-insulator technology, where longer transistor stacks are permitted), the limitations of the library-based design are becoming apparent. A library of fixed size restricts the design choices that are available to a circuit designer, while a dynamic library that is generated on the fly enables a better exploration of the

*This work is supported in part by a Lucent Technologies DAC graduate scholarship, by a gift from Intel Corporation, and by the NSF under contracts MIP-9502556 and MIP-9796305.

design space. This paper develops techniques for generating complex gates on the fly and performing technology mapping for two types of technologies:

- Static CMOS: A topological mapping method called the odd-level transistor replacement (OTR) is developed for library-less mapping to complex static CMOS gates.
- Mixed static CMOS/pass transistor logic (PTL): A Boolean functional mapping method using binary decision diagrams (BDD's) to map logic to PTL is developed.

This paper employs a similar dynamic programming based framework for both of these problems, and although they are solved as separate problems, we present them together in one publication for this reason.

1.1 Library-less mapping

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 disadvantage that the quality of the results is dependent on the richness of the library: a library with a larger number of cells is likely to lead to better results than a sparsely populated library. The impact of the library size on logic synthesis was shown in [14], where it was demonstrated that using larger libraries can reduce the area usage as compared to using smaller libraries.

However, a complicating factor is the effect of deep-submicron technologies, where it has been shown [24] that the ratio of the delay of NAND/NOR gates to the inverter delay is becoming smaller than in older technologies. This encourages the use of longer chains of pull-up/pull-down logic in circuits, making the increased usage of complex gates in deep-submicron circuits more viable. While this leads to better circuit performance, it also complicates the problem of traditional library-based technology mapping. With the increasing use of complex gates in the design, the number of possible gate types increases exponentially [10]. Therefore, the number of gates in any library of a reasonable size can only capture a small fraction of the total number of possibilities, and traditional technology mapping is too restrictive.

To take full advantage of the availability of complex gates, the idea of using a dynamic “virtual library” is becoming attractive. In this approach, individual complex gates (cells) 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 complete design methodology for this paradigm is as shown in Figure 1. This methodology can be extended

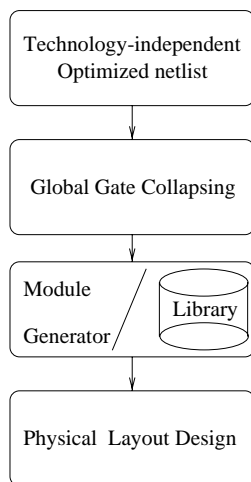


Figure 1: New design methodology

to work with the traditional design 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.

One barrier to this dynamic library approach relates to the quality of the layout generator and the accuracy of timing characterization for the cells generated on the fly. An argument that is often made in favor of a static precharacterized library is that the layout and timing estimates for these cells are much more accurate. However, it is our contention that the limitations of these libraries motivate a more serious look at dynamic cell generation. We note that as layout synthesis systems have become more mature, module generators have become capable of synthesizing layouts for arbitrary complex gates more accurately and efficiently. Several industrial layout synthesis tools such as [7, 17] have been proposed to overcome this barrier to help make the application of dynamic libraries possible. For example, C5M [7] has successfully been used to design a 400 MHz processor, and LAS [17] is a commercial tool that can be used to customize standard/complex and PTL cells and digital random logic blocks with up to 60,000 gates. It was reported in [7] that the quality of the layout generation for a circuit with complex gates is comparable to the quality of hand-crafted layout and the timing information of the complex gates maintain a high fidelity between pre-layout and post-layout generation. Therefore, for our application, we can expect that module generators like those described above may be used to generate layouts that are consistent with the technology mapping.

The matching and covering methods used by traditional technology mappers such as MIS [4] cannot be applied on the virtual library since the number of possible templates is far too large. A pattern/Boolean matching method,

which compares the pattern/logic function of the cells in the library with the pattern/Boolean representation of the target circuit, operates by choosing a covering that is the best solution over all possible matchings. Clearly, both the quality of the results and the CPU time are related to the size of the library.

There has been little prior work that solves the problem of library-less mapping. The work in [12] traverses the circuit and chooses a succession of windows within which to perform local resynthesis operations with sizing, using a sequence of locally optimal decisions to find a solution to the problem. Our technique, in contrast, uses dynamic programming to consider the implications of a decision on the entire circuit at the same time. Other related work includes [1, 2], which builds a graph representation for a logic expression and decomposes it into subexpressions; each subexpression maps on to a complex gate. The mapping algorithm works on the structure of the logic expression graph, and delays are accounted for using an RC model. The work in TABA [22] uses a BDD representation for the network, decomposes the BDD into parts and then finds a mapping for each part based on the BDD representations. TABA maps a set of Boolean equations onto a set of static CMOS complex gates under a given constraints in the number of serial transistors, and represents the latest results, prior to ours, for library-less mapping method in the literature. Another approach [23] starts with a tree of NAND and NOR nodes for the logic expression and then performs dynamic tree covering on this tree.

Our approach to solving the library-less mapping problem avoids manipulating logic expressions, and instead, uses a simple and straightforward method that is entirely topological in nature. Moreover, we use more accurate delay models than those used in past approaches.

1.2 Pass transistor logic mapping

Pass transistor logic (PTL) has recently emerged as a significant alternative to full static CMOS since it has the capability to implement a logic function with a smaller number of transistors, smaller delay and power dissipation. Several pieces of work on PTL research of realizing circuits have been published during last few years (for example, [16, 21, 25, 27, 28, 34, 35]), demonstrating the viability of this technology.

However, there has been relatively little work on design automation for PTL at the technology mapping stage, and this work is among the first published techniques to incorporate pass transistor logic with technology mapping issues. A related body of work is a recent heuristic approach to logic synthesis for PTL [5, 6], which is more focused on the logic synthesis aspects and uses a simple delay model that estimates the delay using the number of transistors on a PTL chain. The work in [19] uses a sequence of heuristic steps to perform performance-driven

PTL synthesis with buffers inserted to control the length of a PTL chain, using Elmore time constants to estimate the delay. The approach presented in [31–33] uses BDD’s to represent the logic expression and decomposes a BDD into AND/OR and XOR (XNOR, MUX) parts; it then uses static CMOS gates to realize the AND/OR parts and uses PTL for the XOR parts. Our work is differentiated from this research in that we use a more sophisticated SPICE-calibrated delay model, and use a dynamic programming approach instead of locally greedy optimizations. Moreover, we apply PTL to realize any type of complex gates, and not merely XOR gates.

Our work mixes PTL and CMOS design style to take advantage of the advantages of both. The idea of mixing PTL and CMOS is not new, and the benefit of this strategy has been proved by a fabricated chip in [30]. The basic difference between our work and theirs is the strategy used to determine the PTL segments of the circuit. The work in [30] treats those parts (in the BDD representation) with a input fixed to V_{dd} or ground as candidates to be implemented by CMOS. A similar goal was pursued in a recently published paper [11], where a greedy algorithm was used to create as large a PTL block as possible, relying on buffer insertion to enforce limits on the maximum number of series transistors permitted in the PTL segment. One difference, however, is that their work generates layouts for the circuits as well, a task that is not addressed in this work.

1.3 Contributions of this work

In this work, we propose a dynamic programming framework for technology mapping for a library-less environment and for PTL. Our work on library-less mapping focuses on the formation of gates, a procedure we call *gate collapsing*, which collapses smaller gates in a decomposed circuit into more complex gates. Our work does not address the issue of module generation for the layout of these cells.

The basic idea of gate collapsing has been used by traditional techniques for technology mapping that performed local gate collapsing through pattern matching to improve the circuit performance. The word “local” refers to the fact that the collapsed gates are constrained to belong to the available cell library in these approaches. In contrast, our approach of global gate collapsing does not tie the list of permissible gates to any specific library. Our procedure works on a virtual library that is assumed to have all possible cell types, 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. This technique can result in a solution that can be optimized for various objectives such as minimizing the circuit delay or the circuit area, or the power dissipation, etc.

For PTL mapping, our approach uses dynamic programming techniques to partition the circuit into PTL segments separated by static CMOS gates. The basic unit in a PTL circuit is a multiplexor, and there is a close relationship between the BDD representation of a circuit and its PTL implementation. Our method dynamically builds BDD's of logic functions and finds an optimal mapping, under the constraint that the number of PTL transistors in series must be constrained never to exceed a user-specified number.

Our work uses accurate delay models calibrated using SPICE. An exhaustive set of SPICE simulations is performed to characterize complex gates and PTL and an accurate look-up table is constructed, listing the gate delay as a function of parameters such as the input signal transition time, the load, the transistor sizes and position of the switching transistor within the gate. An additional contribution of this work is a new technique that is employed to reduce the size of the look-up table and the corresponding memory overhead.

The organization of the paper is as follows. The first technique for gate collapsing, called the OTR method, is described in Section 2. The method is based on an observation that uses the topological properties of the circuit in collapsing complex gates in a computationally efficient manner. Next, we consider the problem of mixed static CMOS/PTL mapping in Section 3, using the relationship between PTL structures and BDD's. Experimental results are presented after each method, and the paper ends with concluding remarks in Section 4.

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.

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). During the procedure of transforming the circuit into a complex gate, we will need to generate intermediate gates shown in Figure 3(a) for temporary use. Those intermediate gates will be transformed into a normal static CMOS gate at the end of transformation.

As shown in the figure, we will refer to the pull-down and pull-up transistor in G1 (G2) as a_n and a_p (b_n and

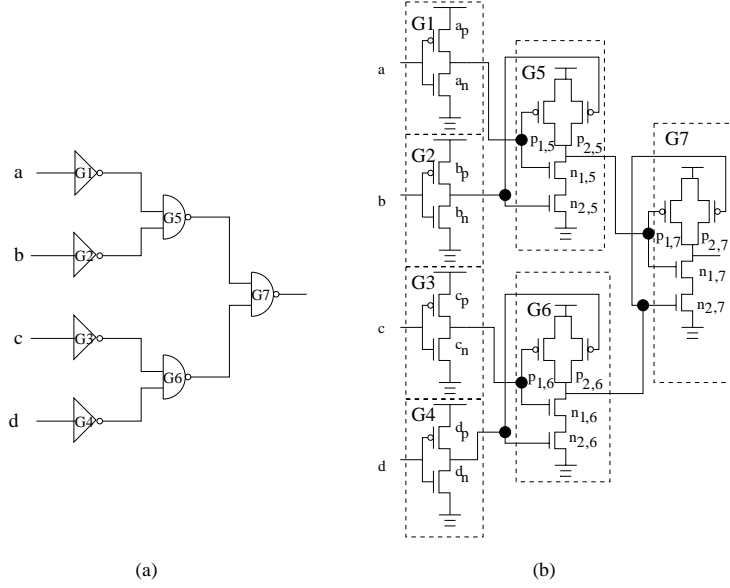


Figure 2: A circuit for gate collapsing

b_p), respectively. We use the pull-down (pull-up) transistors in G1 and G2 to replace the fanout pull-up (pull-down) transistors of these gates in G5 to obtain the gate G5', resulting an intermediate static CMOS gate shown in Figure 3(a). For example, the pull-down blocks of G1 and G2 fan out to the pull-up transistors $p_{1,5}$ and $p_{2,5}$ in G5, respectively, and hence a_n and b_n are inserted in their place to create G5'. Similarly, the transistors in G3 and G4 are inserted into G6 to obtain another intermediate static CMOS gate, G6'. We treat these intermediate 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 consisting of transistors $p_{1,7}$ and $p_{2,7}$ ($n_{1,7}$ and $n_{2,7}$) by the pull-up (pull-down) blocks of the intermediate gates G5' (G6'). Therefore, noting that for G5', the pull-up block consists of a_n and b_n and the pull-down block of a_p and b_p , that G6' has a pull-up block comprising transistors c_n and d_n and a pull-down block comprising c_p and d_p , we perform the following operations to obtain the final collapsed gate: (1) use a_n and b_n from G5' to replace $n_{1,7}$ of G7 (2) use c_n and d_n from G6' to replace $n_{2,7}$ of G7 (3) use a_p and b_p from G5' to replace $p_{1,7}$ of G7 (4) use c_p and d_p from G6' to replace $p_{2,7}$ of G7. 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 an intermediate static CMOS gate, whereas if we collapse an odd number of levels, we will return to the formal CMOS complex gate structure, and therefore we call this technique the *odd-level*

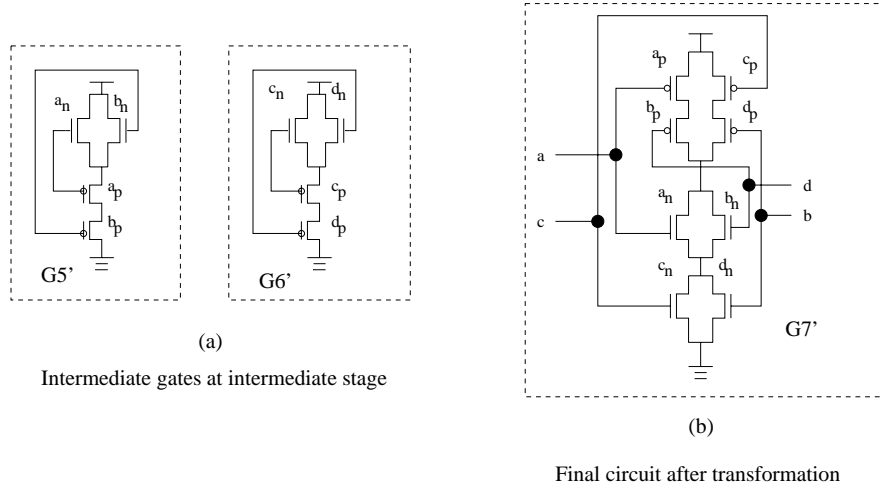


Figure 3: The procedure of OTR gate collapsing

transistor replacement (OTR) method.

2.2 Proof of Logic Correctness

Before beginning this proof, it is important to state that the OTR technique works when the network is entirely specified in terms of inverting gates, as is the case in any CMOS implementation. When the circuit is specified in terms of noninverting gates, the first step would be to convert all noninverting gates into an inverting gate followed by an inverter, and then apply the OTR method.

Theorem:

- (1) *On completion of the OTR procedure, all gates are transformed into (complex) traditional static CMOS gates.*
- (2) *The OTR method preserves the logic function $f(x_1, x_2, \dots, x_n)$ of the original circuit.*

Proof: The first part of this proof is easy to see, since by construction, the pull-up of the final gate consists purely of pMOS transistors and the n part consists purely of nMOS transistors, and each pMOS structure in the pull-up will have a dual nMOS structure in the pull-down. Therefore, the final result will be a traditional static CMOS gate.

We prove the second result on the reduction of a three-level subcircuit to one level. The proof for other odd numbers of levels l can be deduced from this proof in a constructive manner by applying this procedure to reduce the number of levels in steps to $l - 2, l - 4, \dots, 1$.

For the circuit configuration shown in Figure 4, we label the levels from 1 to 3 as shown. We will consider the situation when the output of level 3 is at logic 1; the proof for the logic 0 case is analogous.

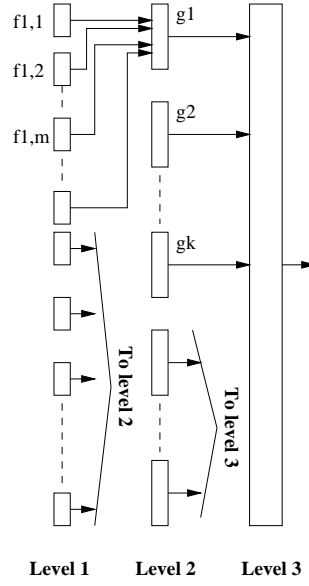


Figure 4: Circuit configuration considered in the proof

Since the level 3 output is at logic 1, it implies that there is a set of pMOS transistors that provides at least one pull-up path between V_{dd} and the output node. We will show that under the OTR scheme, the new gate will also have a pull-up path corresponding to each of these pull-up paths.

Without loss of generality, we may consider any one of these paths, P . Before proceeding, we note that in the original circuit, each of the inputs g_1, g_2, \dots, g_k of P is connected to a pull-down path in the previous level that is connected to ground, i.e.,

$$g_1 \wedge g_2 \wedge \dots \wedge g_k = 0$$

Each transistor on P is replaced by transistor segments from the intermediate nontraditional CMOS gate by applying the OTR procedure, which replaces that transistor with the pull-down of the preceding intermediate gate.

To show that after modifications, the path P continues to provide a pull-up path between V_{dd} and the output node, it suffices to show that each such preceding intermediate gate has a path from ground to its output node. We will call this Requirement (Φ) .

Consider any such level 2 gate g_i in the original circuit that excites a transistor on path P . If the level 3 output is high, then it must be true that there is one or more path in gate g_i that connects the output node to ground in the original circuit. Without loss of generality, we consider any one such path Q_i . Then it must be true that

if the level 1 gates driving this path are $f_{i,1}, f_{i,2}, \dots, f_{i,m}$, then

$$f_{i,1} \wedge f_{i,2} \wedge \dots \wedge f_{i,m} = 1$$

The OTR procedure constructed the intermediate gate by inserting pull-up stages from level 1 to replace the pull-down transistors on Q_i . Therefore, any conducting pull-up path in a level 1 gate will be conducting pull-down subpath in the intermediate gate. Therefore, all the subpaths corresponding to $f_{i,1}, \dots, f_{i,m}$ will be conducting pull-down paths in the intermediate gate at level 2, and when these are placed in series, we have a conducting pull-down path between output and ground for the intermediate version of gate g_i at level 2. Thus we have shown Requirement (Φ) and hence the proof.

2.3 Delay Estimation

In this section, we describe the technique used in this work for delay calculation for complex gates, including a new method used to reduce the amount of storage for a look-up table based approach, while maintaining accuracy.

In order to calculate delay information accurately for complex gates, the rise delay and fall delay of the complex gates are characterized in the look-up table as a function of four parameters (1) the position of the switching transistor (2) the transistor size (3) the input slope S , and (4) the loading capacitance C . The switching position here refers to which transistor in a series chain causes a gate to switch. For example, for the fall transition of a NAND gate, it is possible that the switching may be caused by a transition at any one of the three transistors in the pull-down chain, each of which will lead to different delays. Therefore, we incorporate this information into the look-up table by parameterizing it by the location of the switching transistor. In our implementation, we assume 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, although it is possible that some further timing improvements may be facilitated 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 5(a). We maintain a set of these two-dimensional tables to record this delay information. The number of such tables is given by twice the product of the total number of switching transistor positions and the number of possible transistor sizes, with the factor of two corresponding to separate tables for the rise and fall delays. This look-up table scheme requires a large amount of memory which can make

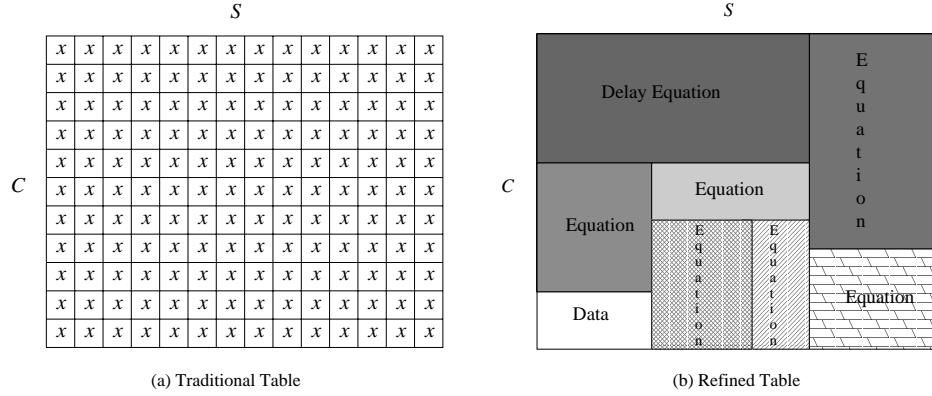


Figure 5: Look-up tables

the look-up speed slow, since it is impossible to store all such tables for all possible switching transistor 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 characteristic delay equation for each such two-dimensional array. For the purposes of characterization, we find a least-squares fit to the characteristic delay equation from [29] which is of the type used by Synopsys:

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

where $\alpha, \beta, \gamma, \omega$ 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 5(b).

Our experimental results show that we can use the delay equation to represent about 75% of the delay data points by using 4-10 different sets of coefficient for each two-dimensional table within an ϵ error of 5%.

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

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

where the summation is performed over all SPICE-measured data points i , and C_i, S_i and D_i , denote the load

capacitance, slope and delay corresponding to the i^{th} data point, respectively. This unconstrained minimization can be performed by setting the partial derivatives of F with respect to each of the parameters to zero, i.e., $\frac{\partial F}{\partial \alpha} = 0$; $\frac{\partial F}{\partial \beta} = 0$; $\frac{\partial F}{\partial \gamma} = 0$; $\frac{\partial F}{\partial \omega} = 0$. This yields a system of linear equations that is solved to find the values of α, β, γ and ω .

2.4 Outline of the Algorithm

We now present a dynamic programming based approach to solve the problem of area minimization under delay constraints. To appreciate the difficulty of this problem, we point out that technology mapping, a special case of global gate-collapsing, is known to be NP-complete for directed acyclic graph structures [15]. 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 [8, 10, 15]), 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.

It is worth pointing out that an alternative class of approaches to technology mapping begins by merging each fanout-free region into a complex gate, so that the circuit consists only of multi-fanout complex gates; the technology mapping problem can then be treated as the problem of decomposition of these gates into complex gates. We do not use this method for two reasons. First, the computational complexity of decomposing the complex gate is exponential in the number of its inputs, and there are no optimal techniques for finding a decomposition of such a network. Second, since the decomposition method only handles fanout-free regions at a time, it is essentially similar to tree mapping in that respect. It is an open question as to which kind of decomposition will eventually yield the best results, and therefore, in our work, we assume that the initial circuit has already been decomposed into a 2-input NAND gate and inverter network.

The algorithm is based on dynamic programming and uses OTR combinations to generate possible complex gates within each tree. As in [8], we begin with a 2-input NAND gate and inverter decomposition of the circuit. While other decompositions may also be used, the purpose of using a 2-input NAND gate decomposition is to increase the granularity of the initial circuit, to provide more freedom to the OTR procedure in generating arbitrary complex gates. In contrast, a coarse-grained approach will allow a smaller degree of flexibility in gate collapsing. The pseudocode for the algorithm is follows:

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
}
```

An explanation of the pseudocode is as follows. The circuit is first levelized to find the level number for each gate, which is the maximum number of gates between the primary inputs and the gate output. Next, the procedure `find_roots` is invoked to split the DAG circuit structure into a forest of trees. The function `sort_roots` then arranges the roots of these trees according to their level number. The trees are processed in order of the level number of their roots, thereby ensuring that before each tree is considered, all of its fanin nodes have been processed.

The dynamic programming procedure [9] proceeds by associating a set of states with each node, where a node corresponds to a gate output. A state is a partial solution that relates 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 method can easily be extended to consider measures such as power in this framework. The Area at a node g is given by the sum of the Area of a candidate complex gate with output g , and the node Area for all possible states at the fanin nodes of the current complex gate. 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 .

The algorithm consists of two phases. Phase one is a postorder traversal from the leaves to the root of the tree, during which dynamic programming proceeds by enumerating the possible states at a node, and eliminating all states in a partial solution that are provably suboptimal. For example, a state [Area, Delay] is provably inferior if there exists another state [Area', Delay'] such that $\text{Area} \geq \text{Area}'$ and $\text{Delay} \geq \text{Delay}'$. The pruned list of possible states at each node are used as candidate states at the next node, and so on. Under this basic framework, the

dynamic programming procedure stores only the noninferior [Area,Delay] combinations and proceeds in a manner that is fundamentally similar to that in [8]. Due to limitations of space, we do not describe any further details here.

Phase two is a preorder traversal from the root node to the leaves, where the best solution is chosen from the set of solutions generated from phase one. When all noninferior states have been enumerated, the optimal state is chosen and the corresponding circuit configuration is determined. An outline of the computational complexity is provided after the pseudocode for the Static CMOS/PTL method described in Section 3.

The delay calculation in our work is very similar to that in [8]. When we process node g during the postorder traversal, the load of node g is unknown. Hence we assume a typical value for the load of node g and find a set of solutions. After the fanout of node g has been handled, we know the exact load for node g , and we may then perform a timing recalculation for node g according to the actual load value.

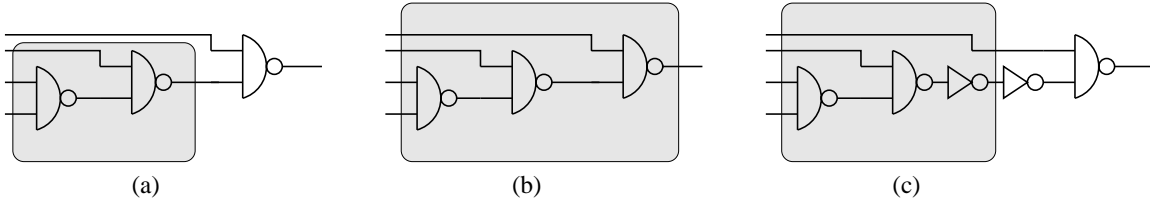


Figure 6: Increasing the versatility of the OTR procedure

The OTR procedure fundamentally requires the presence of three levels of logic on which the transform may be applied. However, our dynamic programming procedure adapts this to consider the collapsing of two levels of gates as well. Consider the situation shown in Figure 6(a), with two levels of logic. The output wire is logically equivalent to two inverters in series, and therefore, it is possible to consider two types of gate collapsing schemes, as shown in Figure 6(b) and (c). This scheme increases the versatility of choices available for the dynamic programming procedure, and this additional flexibility can give significant improvements in the results.

2.5 Experimental Results: OTR

The OTR method described in this paper was implemented in C on a SUN Sparc 1/170 workstation. For purposes of comparison, results were generated using SIS [26], and OTR on the ISCAS'85 benchmark circuits. The circuits were first decomposed into inverters and two-input NAND gates using SIS. Next, we performed a minimum circuit delay technology mapping in SIS for the circuits using the libraries nand-nor.genlib, mcnc.genlib, lib2.genlib and 44-6.genlib. To maintain compatibility between the delay models, we characterized the SIS library

with the same set of technology parameters that we used for the circuit simulations used to generate our delay models. The value of the parameter k (described in Section 2.4) was set to 4 in our work.

Our OTR results and SIS results on `nand-nor.genlib`, `mcnc.genlib` and `lib2.genlib` libraries are shown together in Table 1 for various circuits. These libraries were characterized so that the area and delay measures were consistent with OTR, and the number of power levels for each cell were consistent with those for OTR (although OTR can naturally allow a wider variety of cells). The sizes of cells (power levels) in SIS libraries were chosen based on the sizes of the transistors of the virtual cells for OTR. We chose multiple typical power levels for virtual cells and performed SPICE simulation for delays. The process of characterization for the virtual library took us about one month. 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. In this table, column 1 shows the circuit name; columns 2-4 show, respectively, the minimum delay, the corresponding area, and the CPU time of the SIS mapping results for each circuit on the `nand-nor.genlib` library. The same information is then shown for the SIS mapping results for each circuit on the `mcnc.genlib` library are shown in columns 5-7, and for the `lib2.genlib` library in columns 8-10, and finally, for our OTR method results in columns 11-13. The last line shows the average improvements of delay and area.

Here, the area defines the relative area cost of the cell, in any unit system that is convenient for a circuit designer. The relative area cost was determined from LAS [17]. The area measure used in our method was consistent with the measure used in the characterized SIS libraries.

A comparison of the results of SIS and OTR, shows that that OTR provides better results than SIS results, with average delay reductions of about 40% with about 40% smaller areas.

The reason for these improvements is that in the above modestly sized SIS libraries, none of the complex gates that have four transistors in series in the pull-up/pull-down path, while in our work, we have the flexibility of choosing *any* complex gate with up to four series transistors. In contrast, a library that contained all gates with up to four parallel chains, each with up to four series transistors, would require the characterization of 3503 gates [10]; typical libraries contain a few hundred gates. Another reason for the improvement is that the cells used by OTR have more variations of size (power levels) than the cells in the SIS libraries.

These results are indicative of the power of our technique. It is important to note that SIS simply cannot work in our new design methodology because 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

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(s)
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 OTR Methods on a Large Library

Circuit	44-6.genlib			OTR method		
	Minimum Delay(ns)	Area (unit)	CPU Time(s)	Minimum Delay(ns)	Area (unit)	CPU Time(s)
C432	32.45	6504	101.98	24.53	3354	4.86
C499	26.93	7868	174.41	24.15	6912	7.93
C880	26.81	7492	164.16	24.08	5573	6.99
C1355	28.76	11840	208.16	24.99	7392	7.87
C1908	37.02	12760	284.08	30.07	11004	13.49
C2670	38.93	18498	440.18	30.41	16098	22.29
C3540	58.67	23928	621.73	52.36	21600	29.30
C5315	42.10	39544	1014.61	35.50	35670	60.31
C6288	156.91	45726	1032.27	100.66	28992	23.83
C7552	39.06	51024	2371.93	28.96	43896	70.02
Avg. Imp.	18.7%	22.1%				

largest ISCAS'85 circuit can be handled in minutes.

Table 2 shows the results from SIS running on a large library, 44-6.genlib, a complete library with 3503 cells, in comparison with the results of OTR. In this paper, only one power level was used for this 44-6.genlib library for ease of characterization and in order to control the CPU times for SIS. It is shown that because of the larger size of the library, the results of SIS are better than those from the smaller libraries. Our OTR method continues to be better than the results of SIS even on this larger library, although the average improvement is diminished from Table 1.

After comparing our library-less mapping method with SIS, a library-based mapping method, we compare our method with another library-less mapping method TABA [22]. As mentioned earlier, TABA provides the latest results for library-less mapping in the literature. Table 3 shows the results in terms of the number of gates (including inverters)/transistors using TABA and OTR on some IWLS'93 benchmark circuits. We see that the results of OTR are better than those of TABA in 6 out of 8 circuits. Table 4 shows the number of transistors (the number of gates used by TABA is not reported) using TABA and OTR on the ISCAS'85 benchmark circuits; again, we see that OTR yields better results than TABA. In the cases where our approach does worse, it can be

Table 3: Results of TABA and OTR Methods

Circuit	TABA(4,4)	OTR
	Gates/Trans.	Gates/Trans.
5xp1	51/302	64/378
9sym	65/404	43/272
duke2	223/1110	201/996
misex1	25/148	27/158
rd53	17/82	17/82
rd73	35/174	31/152
rd84	55/290	48/252
sao2	65/362	58/320

Table 4: Transistor Numbers of TABA and OTR Methods

Circuit	TABA(4,4)	OTR
	Transistors	Transistors
C432	710	644
C499	1464	1352
C1355	1592	1304
C1908	2346	1858
C2670	2880	2842
C3540	4106	4012
C5315	5922	5542
C6288	8096	7992
C7552	8556	8298

attributed to the limitation of OTR which relies on the initial circuit decomposition.

The results can be improved still further by enhancing the set of choices available to the dynamic programming procedure by applying the procedure illustrated in Figure 6. Table 5 shows these results: column 1 shows the circuit names, and the remaining columns show the minimum delay and area obtained with and without using this approach, and the corresponding CPU time. On average, it is seen that this improves the results by an average of 26%, and a maximum of 43% in delay, while simultaneously providing area reductions of an average of 0.5%, and a maximum of 57%.

Table 5: Results of OTR Method by Adding Inverters

Circuit	Original		Modified		
	Min. Delay (ns)	Area (unit)	Min. Delay (ns)	Area (unit)	CPU Time (s)
C432	24.53	3354	18.17	1423	8.34
C499	24.15	6912	16.12	4491	17.46
C880	24.08	5573	17.76	4163	14.82
C1355	24.99	7392	18.14	8154	18.79
C1908	30.07	11004	23.90	9222	29.71
C2670	30.41	16098	25.31	25451	40.08
C3540	52.36	21600	30.04	28104	69.16
C5315	35.50	35670	26.47	34083	111.34
C6288	100.66	28992	75.25	41796	106.15
C7552	28.96	43896	23.18	39659	151.17

3 Combined Static CMOS/Pass Transistor Logic Design

3.1 PTL Fundamentals

Although static CMOS has been a mainstay of circuit design for decades, with increasing performance requirements on circuit in terms of speed and power, there is a conscious attempt to seek design styles with better performance. Several techniques such as dynamic logic and PTL have been proposed recently. 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.

PTL is widely considered to be a promising design style since it can implement most functions using fewer transistors than a static CMOS implementation. This reduces the overall capacitance, resulting in circuits with higher speed and lower power dissipation. The logic style is illustrated in Figure 7, which shows a PTL logic segment that realizes the two-input AND function. Only recently has PTL become noticed as a viable design style in its own right, and consequently there are no mature synthesis tools to realize the advantages of this logic style.

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.
- (2) It is possible for sneak paths between V_{dd} and ground to exist unless the circuit is designed carefully. An

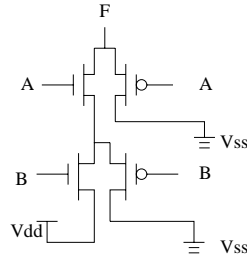


Figure 7: Realization of an AND function using PTL logic

example of sneak paths is shown in Figure 8: if $X = 1$ and $Y = 0$ at the same time, then F is connected to both power supply and ground simultaneously.

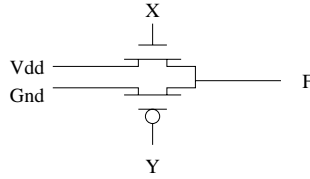


Figure 8: An example of a sneak path

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 9, 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 9(a) shows a BDD node whose PTL implementation is shown in Figure 9(b). Using this as a basis for design, we take the BDD in Figure 9(c), representing a two-input AND gate, and build the corresponding PTL implementation as shown in (d). A second example of more complex logic is shown in Figure 10.

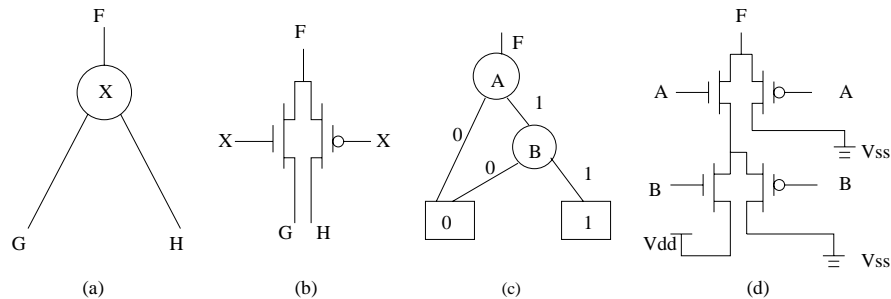


Figure 9: Circuit example 1

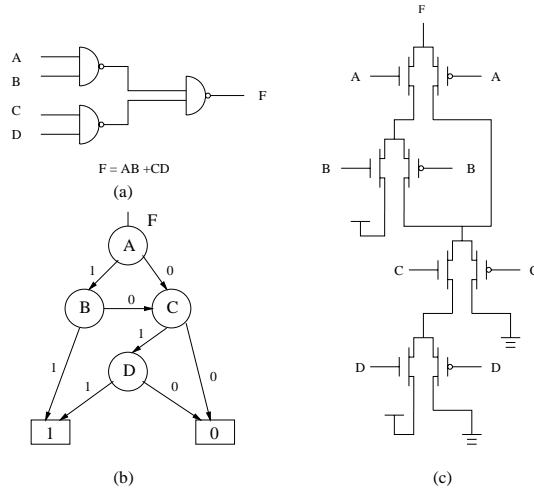


Figure 10: Circuit example 2

3.2 Fundamental Pass Transistor Cell Selection

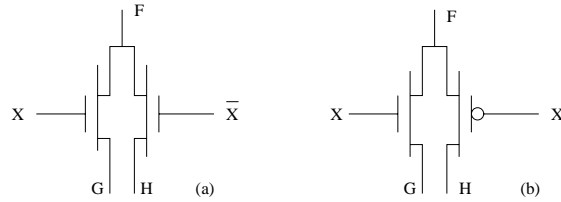


Figure 11: Two types of PTL units

In order to implement a BDD node using a 2-input multiplexer-like pass transistor, a suitable choice of the fundamental pass transistor cell must be made. As pointed out in [6], there are two possible types of fundamental pass-transistor units, as shown in Figure 11(a) and (b). The first uses a pair of nMOS pass transistors, while the other utilizes an nMOS transistor and a pMOS transistor. While the worst case noise immunity of the first configuration is better than that of the second, it requires the generation of complementary signals at the gate inputs, which results in an extra area overhead. Moreover, the extra delay in generating the complement could lead to a sneak path, which could result in a larger power dissipation. In this work, we choose a fundamental cell with one nMOS and one pMOS transistor; however, the work can be extended to handle the other configuration too.

3.3 Outline of the Algorithm

The dynamic programming approach for gate collapsing proposed in Section 2.4 is adapted to develop a technique for building mixed static CMOS/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 [3].

In using PTL, as in the case of complex static CMOS 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 CMOS 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. Since PTL generates high and low voltages that are a threshold voltage away from V_{dd} and ground, respectively, this can cause the short circuit current of the CMOS transistor to be significant. These effects are averted in practice by connecting a weak pull-up/pull-down in a feedback loop from the output of the CMOS gate to its inputs, so that the high voltage is raised to V_{dd} and the low voltage lowered to ground potential.

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 look-up table method for PTL delay model that is similar to that used for the static CMOS logic, with the coefficients being altered, i.e., using different values of coefficients in the delay equation mentioned in Section 2.3.

As in Section 2.4, and for the same reasons, the algorithm begins by decomposing the circuit into a forest of trees. For each such tree, we perform mapping by dynamic programming in a manner similar to that described in Section 2.4 to implement the design using mixed static CMOS/PTL logic. Since the threshold value p is a small number, it is computationally inexpensive to generate BDD's in the fanin cone up to a BDD depth of p . Moreover, the number of possibilities for mapping a node either into a complex gate with a bounded number, k , of series-connected MOSFETs, or as PTL with a bounded p value, is finite and small. Therefore, the computation is fast. The dynamic programming approach is guaranteed to find the optimal solution of mixed static CMOS/PTL circuits for tree structures. For DAG structures, since the approach uses techniques that have worked well for technology mapping, we expect the results to be near-optimal for this problem too, and as our experimental results show, the procedure leads to sensible designs.

The dynamic programming procedure is similar to the OTR algorithm described earlier and is represented by the following pseudocode:

Algorithm Outline

Input: Initial circuit decomposed into inverters and 2-input NAND gates.

Output: Optimum mixed PTL/static CMOS gate network.

```

{
  levelize the circuit
  find_roots
  sort_roots
  from primary inputs to primary outputs
  for each root generate tree
    for each node in the tree from leaves to the root
      apply dynamic programming procedure
      find maximum fanin cone
      generate all possible BDDs inside the maximum fanin cone
        to generate PTL solutions
      find_all_possible_collapsing_solutions
      store_non_inferior_solutions [Area, delay]
  find optimum solution of the primary outputs
}

```

The chief difference between the OTR approach and this approach is that we maintain BDD representations for all possible candidate PTL implementations; as mentioned earlier, due to the limitation on the number of series-connected PTL transistors, these BDD's operate within a maximum fanin cone and are typically small. We compute the possible states of a node g , or a PTL implementation using the BDD representation, and calculate the Area and Delay for every candidate state. As before, each state corresponds to an [Area, Delay] combination, and only the noninferior states are stored. Finally, when all noninferior states have been enumerated, the optimal state is chosen and the corresponding circuit configuration is determined.

While the nature of dynamic programming makes it inherently difficult to arrive at an accurate measure of the computational complexity, it is worthwhile to attempt an estimate of the complexity. For both the OTR and the static CMOS/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 $O(N \cdot M)$.

3.4 Experimental Results: PTL

The Static CMOS/PTL method described above was also implemented in C on a SUN Sparc 1/170 workstation. Results were generated using our mixed static CMOS/PTL method on the ISCAS'85 benchmark circuits. As before, the circuits were first decomposed into inverters and two-input NAND gates. Each PTL cell has multiple power levels as for OTR. The results generated on SIS are identical to those described earlier, but are displayed again in this table for better readability. The value of the parameter p (described in Section 3.3) was set to 4 in our experiments unless specified.

Table 6: 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(s)
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%				

Table 7: Experimental Results of SIS and OTR Methods on a Large Library

Circuit	44-6.genlib			OTR Method			Static CMOS/PTL Method		
	Minimum Delay(ns)	Area (unit)	CPU Time(s)	Minimum Delay(ns)	Area (unit)	CPU Time(s)	Minimum Delay(ns)	Area (unit)	CPU Time(s)
C432	32.45	6504	101.98	24.53	3354	4.86	19.24	1372	190.90
C499	26.93	7868	174.41	24.15	6912	7.93	19.50	3098	196.66
C880	26.81	7492	164.16	24.08	5573	6.99	19.18	2432	168.50
C1355	28.76	11840	208.16	24.99	7392	7.87	20.77	3450	376.49
C1908	37.02	12760	284.08	30.07	11004	13.49	25.28	4753	296.05
C2670	38.93	18498	440.18	30.41	16098	22.29	26.75	5980	621.61
C3540	58.67	23928	621.73	52.36	21600	29.30	36.20	9059	887.46
C5315	42.10	39544	1014.61	35.50	35670	60.31	25.92	11601	990.18
C6288	156.91	45726	1032.27	100.66	28992	23.83	88.33	14403	1306.02
C7552	39.06	51024	2371.93	28.96	43896	70.02	21.74	18350	1093.61
Avg. Imp. (OTR)	18.7%	22.1%							
Avg. Imp. (CMOS/PTL)	35.2%	67.5%							

Table 6 illustrates the results of our Static CMOS/PTL algorithm for various circuits, as compared with SIS on the three libraries (nand-nor.genlib, mcnc.genlib, lib2.genlib). In this table, column 1 shows the circuit name; columns 2-4 show, respectively, the minimum delay, the corresponding area, and the CPU time of the SIS mapping results for each circuit on the nand-nor.genlib library. The same information is then shown for the SIS mapping results for each circuit on the mcnc.genlib library are shown in columns 5-7, and for the lib2.genlib library in columns 8-10, and finally, for Static CMOS/PTL method results in columns 11-13. The last line shows the average improvements of delay and area using our approach.

Table 8: Results of BDDlopt.ptl and Our Static CMOS/PTL Methods

Circuit	BDDlopt.ptl	Our Static CMOS/PTL Method
	Gates	Gates
5xp1	74	63
9sym	36	42
alu2	214	189
alu4	628	431
cordic	38	29
f51m	49	37
my_adder	110	99
parity	15	15
rd53	24	17
rd73	45	31
rd84	58	46
t481	23	24
z4ml	19	16

A comparison of Static CMOS/PTL results with the results of SIS shows much greater performance enhancements from the use of PTL in the circuits. The average delay reduction is about 50% with simultaneous area reductions of above 70% over the results of SIS. One reason for this improvement are because PTL can easily realize non-inverting gates, while SIS must use inverting gates cascaded by an inverter to realize the same function if the library does not have the corresponding non-inverting gate. Moreover, the PTL-based complex gates have more variations of size (power level) than the cells in the SIS libraries. The authors would like to point out the results presented here are pre-layout, and that the post-layout results of our method and SIS may vary from these values. However, we expect that with the use of a high quality module generator, the general trends should be maintained.

Table 7 shows the results of SIS on 44-6.genlib, OTR and Static CMOS/PTL methods. It can be seen from this table that the static CMOS/PTL design style has the best performance among the those three techniques.

Table 6 and Table 7 show the comparison between our library-less method and a library-based method, SIS. We will compare our library-less method with other library-less methods in Table 8 and Table 9. Table 8 shows the comparison between BDDlopt.ptl [32] and our Static CMOS/PTL method in terms of the number of gates. For a fair comparison, we set $p = 3$ in our approach here since the PTL cells used by BDDlopt.ptl have up to 3 pass transistors in series. For 11 out of 13 circuits, our Static CMOS/PTL are better than BDDlopt.ptl because our method does not limit the PTL parts only to be XOR (XNOR) gates, and we use a dynamic programming

Table 9: CMOS/PTL Cells Numbers of Ferrandi and Our Methods

Circuit	Ferrandi's Method	Our Method
	CMOS/PTL cells	CMOS/PTL cells
C432	11/36	10/33
C499	5/69	12/28
C880	10/75	19/51
C1355	5/69	16/49
C1908	31/104	42/83
C2670	52/119	53/103
C3540	50/298	53/141
C5315	71/226	63/194
C6288	3/498	48/147
C7552	112/411	65/206

algorithm for selecting static CMOS and PTL parts. The two worst cases are due to the limitation of our method, due to which it relies on the initial circuit structure. In realistic designs, designers may use $p = 3$ or even 4 for generating a larger variety of acceptable gate types, in terms of delay, for better design performance. With $p = 2$ for XOR (XNOR) and $p = 1$ for MUX, BDDlopt.ptl tends to use smaller gates and results in non-optimal results.

Table 9 shows the results of the numbers of CMOS cells and PTL cells from Ferrandi's method [11] and our Static CMOS/PTL method. While most of the results in [11] are for pure PTL implementations, there is one set of results for mixed CMOS/PTL implementations, and we compare our results with those. In general, if we add the number of CMOS and PTL cells used, we see that our approach uses a smaller number of cells than Ferrandi's method. Moreover, the ratio of the number of CMOS cells to the number of PTL cells in our method is better than Ferrandi's method. For example, for the circuit C6288, Ferrandi's method uses 3 CMOS gates and 498 PTL gates; this is not likely to lead to a viable circuit without buffering to limit the number of PTL transistors in series. In contrast, our approach takes these requirements into account while finding the solution and consequently has a better balance of the number of CMOS cells to PTL cells.

4 Conclusion

We have presented a new 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. Because OTR generates complex gates according to the

initial circuit structure, it does not need to perform Boolean matching or pattern matching with cells in a library. The OTR method is fast and simple and avoids the intractable subproblems in technology mapping, such as matching and covering, by constructing complex gates topologically. The use of PTL is a powerful technique to reduce the area and power dissipation of the circuit, and our work, along with the approaches in [6,11] are early approaches in developing design automation support for PTL synthesis. We believe that the application of virtual library and static CMOS/PTL design style will play a more and more important role in the high performance design. Layout issues including cell generation, cell characterization on the fly remain an open and challenging problem. In our future work, we intend to study these problems and the problem of automated layout generation for the mapped circuits, including interconnect wires between cells.

Acknowledgement

The first author would like to thank Ruchir Puri, Leon Stok and Daniel Brand for discussions about the pass-transistor logic work and Professor Ricardo Reis for valuable references. The authors would like to thank the reviewers for their constructive suggestions.

References

- [1] M. R. C. M. Berkelaar, and J. A. G. Jess, "Technology Mapping for Standard-Cell Generators," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 470-473, 1988.
- [2] M. R. C. M. Berkelaar, "Area-Power-Delay Trade-Off in Logic Synthesis," Ph.D. Thesis, 1992.
- [3] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient Implementation of a BDD Package," in *Proceedings of the ACM/IEEE 27th Design Automation Conference*, pp. 40-45, 1990.
- [4] 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," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 356-359, 1986.
- [5] P. Buch, A. Narayan, A. R. Newton, and A. Sangiovanni-Vincentelli, "On Synthesizing Pass Transistor Networks," in *Proceedings of the International Workshop on Logic Synthesis*, pp. 1-8, 1997.
- [6] P. Buch, A. Narayan, A. R. Newton, and A. Sangiovanni-Vincentelli, "Logic Synthesis for Large Pass Transistor Circuits," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 663-670, 1997.

- [7] J. L. Burns, and J. A. Feldman, "C5M - A Control-Logic Layout Synthesis System for High-Performance Microprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 17, No. 1, January 1998.
- [8] K. Chaudhary, and M. Pedram, "A Near Optimal Algorithm for Technology Mapping Minimizing Area under Delay Constraints," in *Proceedings of the ACM/IEEE 29th Design Automation Conference*, pp. 492-498, 1992.
- [9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill Book Company, New York, New York, 1990.
- [10] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology Mapping in MIS," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 116-119, 1987.
- [11] F. Ferrandi, A. Macii, E. Macii, M. Poncino, R. Scarsi, and F. Somenzi, "Symbolic Algorithm for Layout-Oriented Synthesis of Pass Transistor Logic Circuits," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 235-241, 1998.
- [12] S. Gavrilo, A. Glebov, S. Pullela, S. C. Moore, A. Dharchoudhury, R. Panda, G. Vijayan, and D. T. Blaauw, "Library-Less Synthesis for Static CMOS Combinational Logic Circuits," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 658-662, 1997.
- [13] D. Gragory, K. Bartlett, A. de Geus, and G. Hachtel, "SOCRATES: A System for Automatically Synthesizing and Optimizing Combinational Logic," in *Proceedings of the ACM/IEEE 23rd Design Automation Conference*, pp. 79-85, 1986.
- [14] K. Keutzer, K. Kolwicz, and M. Lega, "Impact of Library Size on the Quality of Automated Synthesis," in *Proceedings of the International Symposium on Circuits and Systems*, pp. 120-123, 1987.
- [15] K. Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching," in *Proceedings of the ACM/IEEE 24th Design Automation Conference*, pp. 341-347, 1987.
- [16] T. Kuroda, and T. Sakurai, "Overview of Low-power ULSI Circuit Techniques," *IEICE Transactions on Electronics*, Vol. E78-C, No. 4, pp. 334-344, April 1995.

- [17] *LAS Users Guide*. Cadence Design Systems, Inc..
- [18] M. Lega, "Mapping Properties of Multi-level Logic Synthesis Operations," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 257-260, 1988.
- [19] T.-H. Liu, A. Aziz, and J. Burns, "Performance Driven Synthesis for Pass-Transistor Logic Circuits," in *Workshop Notes, International Workshop on Logic Synthesis*, pp. 259-259, 1998.
- [20] F. Mailhot, and G. DeMicheli, "Technology Mapping Using Boolean Matching and Don't Care Sets," in *Proceedings of the IEEE European Design Automation Conference*, pp. 212-216, 1990.
- [21] N. Ohkubo, "A 4.4ns CMOS 54x54-b Multiplier Using Pass-transistor Multiplexer," *IEEE Journal of Solid-State Circuits*, Vol. 30, No. 3, pp. 251-257, March 1995.
- [22] A. Reis, R. Reis, D. Auvergne, and M. Robert, "The library Free Technology Mapping Problem," in *Proceedings of the IEEE/ACM International Workshop on Logic Synthesis*, 1997.
- [23] A. Reis, and R. Reis, "Covering Strategies for Library Free Technology Mapping," in *Proceedings of the IEEE/ACM International Workshop on Logic Synthesis*, 1999.
- [24] T. Sakurai, and A. R. Newton, "Delay Analysis of Series-connected MOSFET Circuits," *IEEE Journal of Solid-State Circuits*, Vol. 26, No. 2, pp. 122-131, February 1991.
- [25] Y. Sasaki, "Multi-level Pass-transistor Logic for Low-power ULSIs," in *Proceedings of the IEEE Symposium on Low Power Electronics*, pp. 14-15, 1995.
- [26] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, 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, May 1992.
- [27] D. Somasekhar, and K. Roy, "Differential Current Switch Logic: A Low Power DCVS Logic Family," *IEEE Journal of Solid-State Circuits*, Vol. 31, No. 7, pp. 981-991, July 1996.
- [28] M. Tachibana, "Synthesize Pass Transistor Logic Gate by Using Free Binary Decision Diagram," in *Proceedings of the International ASIC Conference*, September 1997.

- [29] D. F. Wong, "A Fast and Accurate Technique to Optimize Characterization Tables for Logic Synthesis," in *Proceedings of the ACM/IEEE 34th Design Automation Conference*, pp. 337-340, 1997.
- [30] S. Yamashita, K. Yano, Y. Sasaki, Y. Akita, H. Chikata, K. Rikino, and K. Seki, "Pass-Transistor/CMOS Collaborated Logic: The Best of Both Worlds," in *Proceedings of the Symposium on VLSI Circuits*, pp. 31-32, 1997.
- [31] C. Yang, V. Singhal, and M. Ciesielski, "BDD Decomposition for Efficient Logic Synthesis," in *Proceedings of the International Conference on Computer Design*, 1999.
- [32] C. Yang, and M. Ciesielski, "Synthesis for Mixed CMOS/PTL Logic: Preliminary Results," in *Proceedings of the International Workshop on Logic Synthesis*, 1999.
- [33] C. Yang, V. Singhal, and M. Ciesielski, "BDD Decomposition for Efficient Logic Synthesis," in *Proceedings of the International Workshop on Logic Synthesis*, 1999.
- [34] K. Yano, Y. Sasaki, K. Rikino, and K. Seki, "Top-down Pass-Transistor Logic Design," *IEEE Journal of Solid-State Circuits*, Vol. 31, No. 6, pp. 792-803, June 1996.
- [35] K. Yano, T. Yamanaka, T. Nishida, M. Saito, K. Shimohigashi, and A. Shimizu, "A 3.8ns CMOS 16x16-b Multiplier Using Complementary Pass-transistor logic," *IEEE Journal of Solid-State Circuits*, Vol. 25, No. 2, pp. 388-395, April 1990.