BDD Decomposition for Delay Oriented Pass Transistor Logic Synthesis

Rupesh S. Shelar, *Member, IEEE,* and Sachin S. Sapatnekar, *Fellow, IEEE,*

Abstract— **We address the problem of synthesizing pass transistor logic (PTL), with the specific objective of delay reduction, through binary decision diagram (BDD) decomposition. The decomposition is performed by mapping the BDD to a network flow graph, and then applying the max-flow min-cut technique to bipartition the BDD optimally under a cost function that measures the delay and area of the decomposed implementations. Experimental results obtained by running our algorithm on the set of ISCAS'85 benchmarks show a 31% improvement in delay and a 30% improvement in area, on an average, as compared to static CMOS implementations for *xor* intensive circuits, while in case of arithmetic logic unit and control circuits that are *nand* intensive, improvements over static CMOS are small and inconsistent.**

Index Terms—**Binary Decision Diagrams (BDD's), Pass Transistor Logic, Logic Synthesis, Functional Decomposition**

I. Introduction

A. Motivation

Static CMOS has been a favorite logic style of VLSI designers for the last two decades due to its advantageous noise immunity properties and good performance. However, due to technology scaling and the increasing number of transistors on chip, the performance of static CMOS circuits comes at substantial area/power dissipation costs that may not be desirable, especially for portable appliances. Therefore, new logic families that address the power and performance challenges must be explored [1]. Among low power logic families, pass transistor logic (PTL) has great potential due to its ability to implement logic functions with a lower transistor count, smaller capacitance, and hence better performance, perhaps at the same or lower area/power cost as that of static CMOS [2–5]. Although rising transitions in NMOS-only PTL are slower than those for fully complementary logic, our simulations show that PTL will continue to result in better implementations than static CMOS in case of XOR-dominated circuits. The results of these simulations for a 3-input XOR gate, using predictive technology models [6], are displayed in Figure 1(a) and demonstrate that PTL will continue to provide benefits over CMOS for several technology generations. However, due to a lack of synthesis tools and methodologies for PTL, it is unclear whether PTL will live up to the promise of providing equal (or better) performance at lower cost as compared to static CMOS.

PTL (or one of its variants, such as CPL [2]) is known to yield better implementations as compared to static CMOS for arithmetic circuits, such as adders and multipliers. However, when these elements are synthesized along with random logic using standard cell libraries, the structural properties of the network that are suitable for PTL may remain unexploited, resulting in possibly suboptimal solutions. PTL elements are used in design even today: ASIC libraries typically contain PTL-like one-hot multiplexers and pass transistor gates because of the area/power/performance gains that they offer over static CMOS circuits, even though the latter have higher noise immunities [7]. In practice, most of these cells are employed in an ad-hoc manner after verifying that the nets driving PTL cells are appropriately buffered. Thus, although the use of PTL may be desired, it remains underutilized and more so, because of the lack of good performance-driven synthesis algorithms and methodologies exploiting the properties of PTL circuits.

It is well known that PTL is not universally better than CMOS for all types of logic structures: for NAND-intensive circuits, for example, static CMOS can result in better implementations than PTL. This is demonstrated in Figure 1(b) by our simulations for a three-input NAND gate in both logic styles, at various technology nodes. Therefore, mixed static CMOS/PTL synthesis is likely to be an attractive alternative in the future. Even for such an approach, synthesis solutions targeting performance that exploit the properties of PTL circuits must be developed, and this work may be considered as a step in that direction.

For performance-driven pass transistor logic synthesis, this article proposes an algorithm that is based on decomposing binary decision diagrams (BDD's) to minimize the delay in BDD-mapped PTL circuits with the least area penalty. Using this synthesis algorithm for PTL and standard synthesis flow for static CMOS, we answer the following questions that are of interest to VLSI designers:

- Can PTL match the delay in static CMOS circuits at a lower area cost than static CMOS standard cells? If so, then is this true only for specific kinds of circuits?
- In cases where the results favor PTL, how much is the gain over static CMOS for benchmark circuits?

These questions are even more important with technology scaling as leakage power, which is indirectly dependent on the area of a circuit, becomes a major bottleneck.

B. Previous Work

Synthesis techniques for PTL circuits have been closely related to the binary decision diagram (BDD) representation of logic functions, for several reasons: firstly, BDD-based PTL circuits are guaranteed not to have any sneak-paths[1], and secondly, the use of BDD-based methods can benefit from the plethora of efficient algorithms available for the construction of BDD's. The BDD representation of a logic function affects the PTL implementation, and therefore, BDD decomposition methods must be adapted to optimize cost functions that represent their PTL implementation.

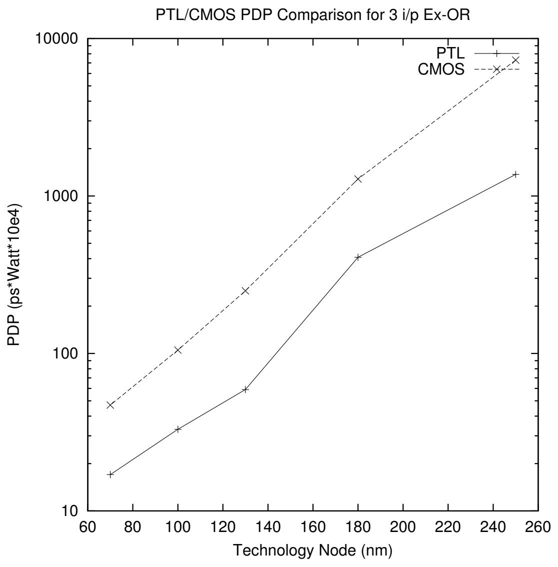
The idea of decomposing logic functions, in general, and BDD decomposition, in particular, for optimizing specific objectives is not new, although there is little work on considering PTL-based cost functions during BDD decomposition. We review some of the representative work in the area of Boolean decomposition and

Manuscript received November 4, 2003; revised November 26, 2004. This work was supported in part by Semiconductor Research Consortium (SRC) under contract 2002-TJ-1092 and under award NSF CCR-0098117. This paper was recommended by an Associate Editor Srimat Chakradhar.

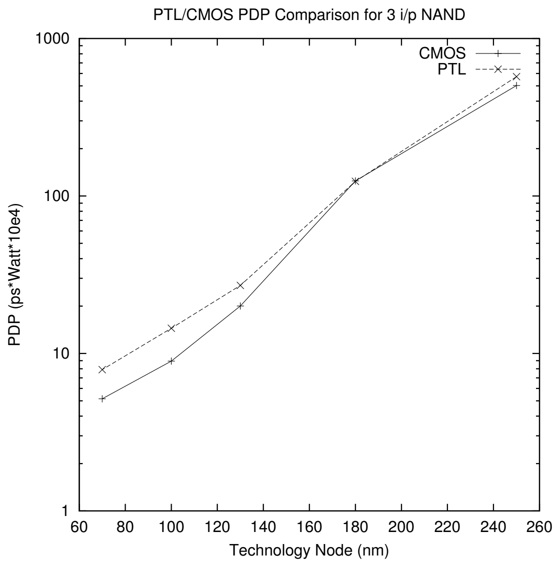
R. S. Shelar was with the Department of ECE, University of Minnesota, Minneapolis, USA. He is now with Intel Corporation, Hillsboro, OR, USA.

S. S. Sapatnekar is with the Department of ECE, University of Minnesota, Minneapolis, USA.

[1]A sneak-path is a path from V_{DD} to ground in a steady state. The sneak-paths do not exist in static CMOS circuits, since a path either in pull-up or pull-down network is active for any assignment of inputs. For PTL circuits, however, if not designed carefully, there may exist sneak paths. Employing BDD's for PTL ensures a sneak-path-free implementation, since only one path is active for any assignment of inputs in a BDD.



(a)



(b)

Fig. 1. Power-delay product (PDP) values for (a) a three-input XOR gate and (b) a three-input NAND gate. Both circuits are implemented in NMOS-only PTL and in static CMOS at various technology nodes, and the results of SPICE simulations using predictive technology models [6] are shown.

BDD decomposition. In the area of decomposition of switching functions, Ashenhurst performed pioneering work with a theorem relating column multiplicities in a partition matrix, corresponding to a partition of variables into *bound set* and *free set* of variables, with the simple disjunctive decomposability of a switching function, and also proved relevant theorems on non-simple decompositions of a switching function [8]. An excellent review on the development of theory of decomposition of switching functions in 1960's and 1970's is presented in [9]. Recently, Lai *et al.* have proposed an ordered BDD (OBDD) based function decomposition method that involves forming a cutset in the BDD, and then encoding the nodes in the cutset to yield disjunctive or non-disjunctive decompositions [10]. This OBDD-based decomposition has been applied to the synthesis of field programmable gate arrays targeting area, measured in terms of the number of configurable logic blocks, with no depth constraints. In [11], a BDD-based logic synthesis system is developed, in which transformations such as AND/OR decomposition based on 0/1 dominators, and XOR and functional MUX-based decompositions are proposed; synthesis for performance-driven PTL is not specifically targeted.

Several techniques for PTL synthesis have been suggested in the recent past. A loose upper bound of theoretical utility on the number of multiplexers required to implement a given logic function is developed in [12]. Buch *et al.* propose a greedy heuristic in [13] to decompose larger BDD's into smaller BDD's whose sizes are kept under a specified threshold. For area-driven PTL synthesis, Chaudhry *et al.* [14] present a method similar to traditional multilevel logic optimizations, first invoking the iterative application of logic transformations, and then mapping the BDD representation on to a PTL cell library. A similar philosophy has been used for performance-driven synthesis in [15]. Both [13] and [15] imply that multilevel BDD's are to be used, but the limitation of these approaches is that they are unable to predict the performance gain beforehand. Ferrandi *et al.* propose the use of PTL cell generation and subsequent binate covering of the nodes in the Boolean network using a set of heuristically generated BDD's to minimize the cost [16]. Scholl and Becker [17] report the use of multiplexer circuits for area and depth optimizations of PTL circuits. Unlike [13], they allow the threshold size of the decomposed BDD's to be varied, and their cost function allows area and depth to be traded off. Poli *et al.* propose techniques for transistor level construction of static CMOS and PTL cells from BDD's and provide a comparison of these logic styles by restricting the number of inputs to a cell to four [18]. Cho and Chen propose a genetic algorithm for technology mapping of mixed static CMOS and PTL circuits using a predefined set of PTL cells [19]; their approach does not specifically target performance driven PTL synthesis.

C. Our Contributions

In this paper, we present a novel approach to performing delay oriented PTL synthesis through the decomposition of a monolithic BDD representing a circuit. Our contributions can be summarized as follows:

- We explicitly incorporate delay and area considerations simultaneously into a global technique for finding the decomposition.
- Our bipartitioning scheme employs the max-flow min-cut technique to roughly halve the delay of a PTL implementation of a BDD with the least area overhead. The delay in a PTL circuit is well known to be linear in the number of input variables after buffer insertion, and our recursive bipartitioning approach can result in logarithmic depth reductions over the PTL implementation of the monolithic BDD. Although logarithmic depth reductions, in terms of transistors, may not translate to logarithmic delay reductions, the resulting delay reductions are still substantial. The area penalty is minimal, up to the accuracy in estimation, as the algorithm explicitly attempts to minimize this overhead by finding an optimal cut.
- Experimental results, obtained using the above techniques, on a set of ISCAS'85 benchmarks containing *xor* dominated arithmetic circuits such as multiplier and the circuits for error correcting codes, show that PTL outperforms static CMOS implementations with 31% improvement in delay and 30% improvement in area, on an average, for a 0.13μm technology. We found that in case of arithmetic logic unit (ALU) and control circuits, the improvements over static CMOS are small and inconsistent, although PTL (or its variant CPL) is known to yield cost effective implementations of adders, which are important

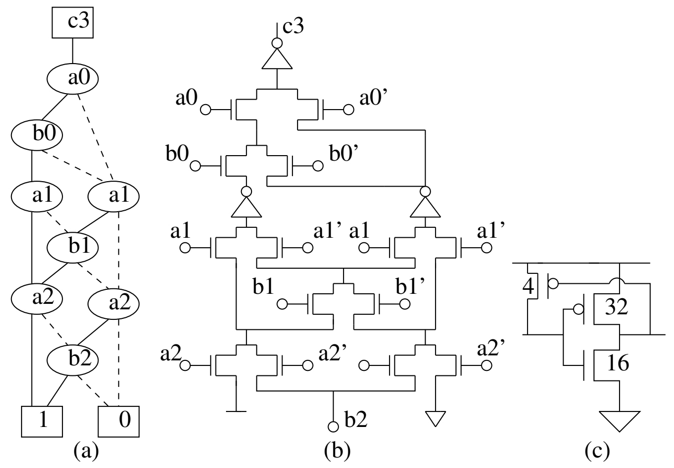


Fig. 2. (a) The BDD for Carry function for 3-bit adder (b) Its corresponding PTL implementation, using inverters with weak pull-ups, shown at the transistor level in (c).

components of ALU and control circuits. This anomaly may be attributed to the scripts in SIS [20] that are used for preprocessing and also to the structure of control logic, which is usually *nand* intensive, in these circuits. Employing our PTL synthesis algorithm in case of the designs that are inherently well suited for PTL, one may obtain performance that is close to custom designs while the use of static CMOS standard cell libraries to obtain the same performance may come at a very high area/power cost.

D. Organization of the Article

The remainder of this article is organized as follows. Section II illustrates the BDD decomposition technique that is applied for delay optimization. We transform the BDD decomposition problem to a bipartitioning problem in Section III, and describe a solution that employs the max-flow min-cut technique, without dwelling on the precise delay analysis method. In Section IV, we complete our description by outlining the delay models and the delay analysis method used in decomposition and for post-synthesis delay estimation. Section V presents experimental results on the ISCAS'85 benchmark circuits, followed by concluding remarks in Section VI. A preliminary version of this work was presented in [21].

II. PTL Implementation using Decomposed BDD's

A. The Relationship between BDD's and PTL

A BDD can be mapped on to a PTL implementation as follows. Each node of the BDD implements a Shannon expansion about the variable x associated with the node, and can be expressed as $F = x \cdot F_x + x' \cdot F_{x'}$, where F_x and $F_{x'}$ are, respectively, the Shannon cofactors of the function F. This may be translated to a multiplexer that passes F_x when x is high, and $F_{x'}$ when x is low; the procedure can then be applied recursively to the functions F_x and $F_{x'}$. Therefore, for any logic function, the BDD representation can be used to directly arrive at its PTL implementation, as shown in Figure 2. Moreover, mapping from a BDD on to a PTL circuit ensures a sneak-path-free implementation: this follows from the property of BDD's that for any assignment of inputs, only one path from the root node to a terminal node is active. For the purposes of this paper, all BDD's are reduced ordered BDD's (ROBDD's), which implies that the order of variables on any path from an output node to a leaf node is identical. We also restrict ourselves to NMOS-only PTL, although the algorithms proposed in this article are applicable to other variants of PTL, such as transmission gate PTL, albeit with different area/delay trade-offs. NMOS-only PTL, as the name suggests, employs only NMOS transistors as pass transistors. Therefore, it requires buffers with weak pull-ups after every k transistors in series to avoid long chains of pass transistors and also to recover the voltage drop across transistors while passing logic one.

B. BDD Decomposition for Delay Optimization

Mapping a BDD directly to PTL can result in delays that are linear in the number of input variables, and BDD decomposition can be applied to reduce these delays. We outline a general BDD decomposition technique with the help of the following example. Consider a carry function for a three-bit adder whose optimized BDD is shown in Figure 3(a). This BDD is built on six variables, a0, b0, a1, b1, a2 and b2, and one output, c3. We choose a cutset across the BDD that is indicated by the shaded nodes in Figure 3(a). When this cut is used to separate the upper and lower parts of the BDD, dangling edges are created in the upper part, for instance, edges from nodes labeled a1 to nodes labeled a2. We introduce dummy nodes V0, V1, V2 that replace these shaded nodes, as shown in Figure 3(b). These dummy nodes can be assigned unique codes employing one-hot or minimum-bit encoding, as shown in Table 1. The two types of encoding lead to two alternative PTL implementations with different area/delay trade-offs.

Once an encoding is chosen, the original function can be realized using a decomposition based on this cut. The encoding bits (i.e., $O_0O_1O_2$ or S_0S_1) can be utilized as *select* inputs to a multiplexer whose *data* lines correspond to the evaluated values of the BDD's rooted at the three shaded nodes as shown in Figure 3(e), depending on the value of the encoding. Therefore, each such *select* input corresponds to a BDD representation that sets the leaf nodes according to the chosen encoding. As an example, the select bit O_1 for the one-hot encoding corresponds to the combination V0 = 0, V1 = 1, V2 = 0, and is used to select the BDD rooted at the shaded node b1. By substituting these values into the dummy terminals in Figure 3(b), we can obtain the BDD for the *select* input O_1. The BDD's for other *select* inputs such as O_0 and O_2 can be obtained similarly. The multioutput BDD for $O_0O_1O_2$ is illustrated in Figure 3(c).

One-hot Encoding		Minimum-bit Encoding	
Terminal Node	$O_0O_1O_2$	Terminal Node	S_0S_1
V0	100	V0	00
V1	010	V1	01
V2	001	V2	11

TABLE I
ONE-HOT AND MINIMUM-BIT ENCODING SCHEMES FOR THE DUMMY TERMINAL NODES INTRODUCED DURING DECOMPOSITION.

If, instead, a minimum bit encoding is used, a similar procedure may be employed to derive the BDD for the *select* inputs S_0 and S_1; the corresponding multioutput BDD is depicted in Figure 3(d). We observe that depth of the BDD's for the *select* inputs is the same for one-hot encoding and for minimum-bit encoding. Note that in case of *select* functions obtained by one-hot encoding, for any assignment of a0, b0, and a1, only one of the *select* functions is true and we can use a one-hot multiplexer circuit to implement c3. On the other hand, *select* functions that correspond to minimum-bit encoding are employed as inputs to regular multiplexers. Two alternative implementations of c3 using a one-hot multiplexer and a regular multiplexer are shown in Figure 4(a) and in Figure 4(b),

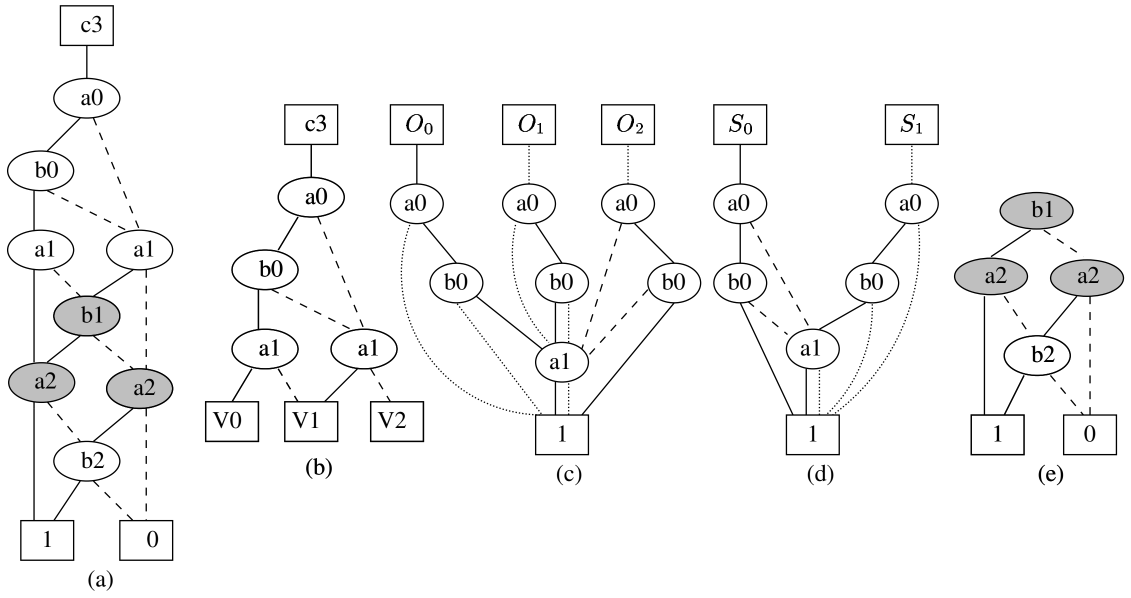


Fig. 3. (a) The BDD for the carry function for a three-bit adder, where the shaded nodes form the cut that is used to decompose the BDD. (b) The upper part of the cut, with the dangling edges replaced by dummy nodes V0, V1, V2. (c) The select function under the one-hot encoding for V0 V1 V2. (d) The select function under a minimum-bit encoding for V0 V1 V2. (e) The data function. In all of these pictures, the solid edges in the BDD denote the 1-cofactor, F_x, the dashed edges denote the 0-cofactor, $F_{x'}$, and the dotted edges denote the complemented 0-cofactor, $\overline{F_{x'}}$.

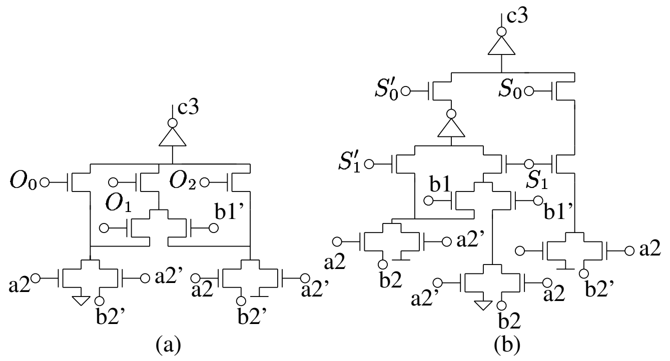


Fig. 4. Alternative implementations of c3 (a) using one-hot multiplexer, (b) using regular multiplexer. Note that, one multiplexer driven by a2' and a2 is duplicated in case of (b) because of the difference in the number of inverters from this multiplexer to the output.

respectively. The *select* inputs are simply the PTL implementations of the BDD's shown in figures 3(c) and 3(d). The *data* inputs are due to PTL implementation of the BDD's in Figure 3(e), whose terminal nodes are assigned appropriate polarity to account for the inverters at the output of one-hot or regular multiplexer.

Implementation	Active Area(μm^2)	Delay(ps)
Monolithic BDD	2.974	201
One-hot Multiplexer	3.532	103
Regular Multiplexer	3.768	174

TABLE II
A COMPARISON OF ALTERNATIVE IMPLEMENTATIONS OF C3.

Table II shows the active area and delay, obtained by circuit simulations under an excitation with a 50ps[2] transition time, for alternative implementations obtained by (1) directly mapping the BDD, and using decomposition based on (2) one-hot multiplexers and (3) regular multiplexers in 0.13 μm technology [6]. All of the pass transistors have widths of 14 λ and the inverters are sized as follows: all PMOS transistors have widths of 32λ, all NMOS transistors have widths of 16λ, and all weak pull-ups have widths of 4λ, as shown in Figure 2, where 2λ is the minimum feature size. Clearly, the one-hot multiplexer based implementation has the least delay, albeit with a slightly larger area than that obtained by directly mapping the BDD. We also observe that in the decomposed implementation using one-hot multiplexers, the depth of the circuit is halved as compared to the implementation obtained by a direct mapping of the BDD. Moreover, this procedure can be applied recursively, halving the depth at each step to result in a logarithmic depth PTL implementation.

C. Trade-offs between the Choice of a One-hot or a Regular Multiplexer

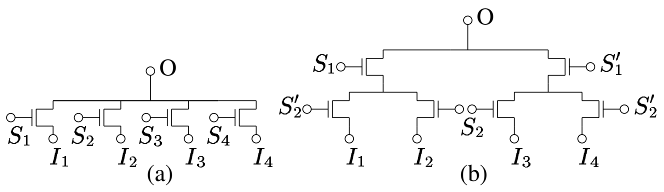


Fig. 5. Transistor-level implementation of (a) a one-hot 4:1 multiplexer, (b) a regular 4:1 multiplexer

Figures 5(a) and 5(b) show transistor-level implementations for 4:1 one-hot and regular multiplexers, respectively. In case of the one-hot multiplexer, four *select* inputs are required, of which only one can be high at a time. In contrast, the regular multiplexer has two *select* inputs which are used to select among four *data* inputs. We observe that the depth of a one-hot multiplexer circuit, as measured by the maximum number of series transistors, is always one, irrespective of the number of *data* inputs. On the other hand, the depth of a

[2]The transition time of 50ps is chosen, as it corresponds to a typical microprocessor clock period of 500ps corresponding to a 2GHz frequency.

regular multiplexer increases logarithmically with the number of *data* inputs. Apart from the delay advantage that can be obtained from this reduced depth, a one-hot multiplexer with n data inputs also uses fewer transistors than a regular multiplexer. Specifically, the number of transistors required to implement a one-hot multiplexer is n, while the corresponding number for a regular multiplexer is $2n - 2$.

The complete picture, however, is more complex. The number of *select* inputs required for a one-hot multiplexer is the same as the number of *data* inputs, and therefore, such a multiplexer requires the generation of more *select* functions than a regular multiplexer. Moreover, although the number of levels for a one-hot multiplexer is always one, its delay is not constant but increases with the number of *data* inputs. This arises since an increase in the number of transistors connected to the output results in an increase in the load driven by the one-hot multiplexer, as additional drain capacitances, which contribute to the total output capacitance, are brought in by each data input. This is one of the reasons why the logarithmic depth reductions provided by our approach, which uses one-hot multiplexers, do not translate into logarithmic delay reductions. However, this is not a significant limitation, since the obtained delay reductions, as shown in Table II, are nevertheless substantial for real circuit examples.

III. The BDD Decomposition Algorithm

The decomposition technique presented in the Section II can be thought of as a bipartitioning that halves the circuit depth and therefore, shortens the critical path and its delay. If we take a single cut across the BDD that halves the critical path, then we find that the delay in the PTL implementation using a one-hot multiplexer, which adds one extra series transistor, is approximately halved. We can apply this bipartitioning procedure recursively, such that on each application of the procedure, the critical path is roughly halved. The price being paid for this delay reduction is in terms of area, since the number of transistors required for implementation may increase as we recursively bipartition the BDD. BDD decomposition for delay reduction does not always result in an area penalty since one-hot encoding of the select functions may result in simpler Boolean functions and hence, smaller BDD's. In such a case, bipartitioning should be performed so that it approximately halves the delay and also results in area-wise good implementation. In our algorithm, we perform this bipartitioning to aim for the minimum area penalty. This approach differs from that of [22] in the objective as well as the application of bipartitioning: aim of that work is power minimization in the combinational logic blocks implemented in PTL, where bipartitioning is applied only once to a given BDD, as opposed to this work which targets delays, and applies bipartitioning recursively.

A. Recursive Bipartitioning for Performance

A key step during bipartitioning is that of identifying candidate nodes for the cut that will succeed in approximately halving the circuit delay. Our delay estimator for the PTL implementation of a given BDD assumes the insertion of a buffer after at most three pass transistors in series and inverters for complemented edges. Based on this assumption, each node in the BDD is assigned two delays:

- Delay from bottom (D_{bottom}): This is the delay of the PTL network rooted at a given BDD node.
- Delay from top (D_{top}): This is the maximum delay from a given BDD node to any of the outputs.

These delays can be evaluated employing the delay analysis procedure outlined in Section IV, which can be used to identify the critical path through the PTL network.

We define three types of nodes for delay-balanced bipartitioning:

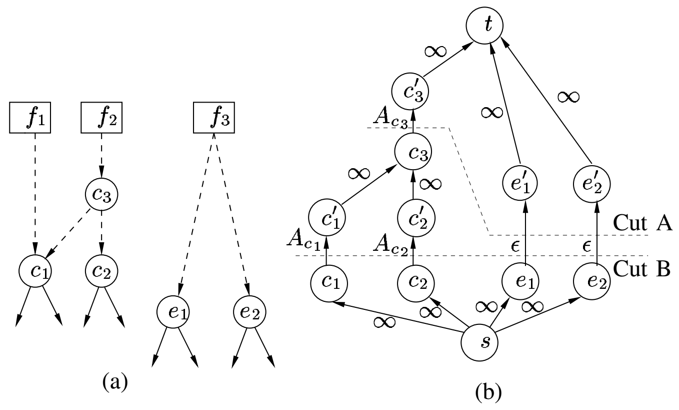


Fig. 6. Creating a flow network: (a) digraph corresponding to a BDD with essential and candidate nodes, (b) corresponding flow network.

- Essential nodes, for which D_{bottom} lies within a small range ($\pm\delta$) of half of the critical path delay ($D_{critical}$). In other words, essential nodes lie in the middle of critical path, with small tolerance δ.
- Candidate nodes, for which D_{top} and D_{Bottom} are both less than $(\frac{D_{critical}}{2} - \delta)$.
- Non-candidate nodes, which comprise all of the remaining nodes. These nodes will not be considered for inclusion in the cut.

The optimum cut will halve the critical path, ensuring that no other path in the decomposed implementation has a delay of more than half the critical path delay. Therefore, all essential nodes must be in the cut, while we have the freedom to choose among the candidate nodes. We assign an area cost, explained in the next subsection, to the candidate nodes and then use the max-flow min-cut technique [23] to find an optimum cut that halves the circuit delay with the smallest area cost.

Figure 6 shows an example of how the flow network is created. The procedure begins with a digraph corresponding to the given BDD, illustrated in Figure 6(a). In this example, let us assume that there are three nodes f_1, f_2, and f_3 corresponding to the three primary outputs, three candidate nodes c_1, c_2 and c_3, and two essential nodes e_1 and e_2. The dashed edges in Figure 6(a) (for instance, an edge from f_1 to c_1) indicate that there are directed paths between the corresponding nodes, but the nodes on these paths are not shown since none of them are essential nodes or candidate nodes. Figure 6(b) shows the corresponding flow network with one source node s and one destination node t. Each essential node in the digraph is split into two nodes: for instance, node e_1 is represented by two nodes e_1 and e'_1 with an edge from e_1 to e'_1 that is assigned a small capacity[3], ϵ. Similarly, candidate nodes are also split into two nodes: for instance, node c_1 in the digraph is represented by two nodes c_1 and c'_1, respectively. However, the edge capacity for these nodes is not ϵ, but is set to the area cost of the candidate nodes in the BDD. In this example, the edge from c_1 to c'_1 has an edge capacity of A_{c1}. The remaining edges in the flow network are assigned a capacity of ∞, and therefore will not appear in the cut. Thus, in this example two cuts are possible, Cut A and Cut B, corresponding to cutsets $A_{cutset} = \{e_1, e_2, c_3\}$ and $B_{cutset} = \{e_1, e_2, c_1, c_2\}$ in the digraph corresponding to the given BDD. The application of the

[3]The small capacity ϵ to the edges between split essential nodes ensures that all essential nodes are in the cut. A capacity of 0 cannot be associated with these edges because of the conventions in the max-flow min-cut algorithm; a capacity of 0 means that edge does not exist.

Ford-Fulkerson technique to find the minimum cut will result in one of these, depending on values of A_{c_1}, A_{c_2} and A_{c_3}. The pseudocode for the entire procedure is is shown in Algorithm 3.1. Once the cut has been determined, the vertices in the cut are replaced by dummy terminal nodes, which can be assigned unique codes and implemented as PTL circuits, as illustrated in Section II.

Algorithm 3.1 Algorithm for BDD decomposition

```
Input: A digraph $G(V, E)$ corresponding to the given BDD, the set
    of nodes $V$ in the digraph, the set of edges $E$ in the digraph
Output: An optimal cutset $S_{cut}$
1: DelayAnalysis($G$);
   // Assign $D_{top}$, $D_{Bottom} \forall v \in V$
2: $D_{critical} \leftarrow \max\{v.D_{Bottom} \forall v \in V\}$;
   // Compute the critical path delay
3: $V_{Essential} \leftarrow \{v{:}v \in V$ and $\frac{D_{critical}}{2} - \delta \leq D_{bottom} \leq \frac{D_{critical}}{2} + \delta\}$;
   // Identify the essential nodes
4: $V_{Candidate} \leftarrow \{v{:}v \in V$ and $D_{top}$, $D_{bottom} < \frac{D_{critical}}{2} - \delta\ \}$;
   // Find the candidate nodes
5: AreaCostEstimate($V_{Candidate}$);
   // Compute the area cost for all candidate nodes
6: $G_{Flow} \leftarrow$ CreateFlowNetwork($G$,$V_{Essential}$,$V_{Candidate}$);
   // Create the flow network
7: Ford-Fulkerson($G_{Flow}$, $G$, $S_{cut}$);
   // Find an optimal cut
```

This bipartitioning procedure can be applied recursively till no further delay reduction can be achieved. If we define the depth of the implementation as the maximum total number of series transistors (discounting buffers) from any input to any output, then the resulting implementation has a depth that is logarithmic in number of inputs. In contrast, the original undecomposed BDD yields an implementation whose depth is linear in the number of variables. This is stated in the following theorem:

Theorem 3.1: The recursive application of the procedure in Algorithm 3.1 to any BDD with the use of one-hot multiplexers results in an implementation that has a depth, in terms of the number of series transistors, of $O(\log(Depth_M))$, where $Depth_M$ is the depth of the PTL implementation obtained by directly mapping the BDD.

Proof Since the PTL implementation obtained by directly mapping the BDD has depth $Depth_M$, the *select* and *data* functions obtained by the first level of bipartitioning each has a depth of at most $\lceil Depth_M/2 \rceil$. The use of the one-hot multiplexer adds a constant depth of one transistor to this. The bipartitioning procedure can be applied further to these decomposed *select* and *data* functions, a process that can continue recursively. There can be at most $\lceil \log(Depth_M) \rceil$ such recursions, and after each recursion only a constant depth is added due to the one-hot multiplexer. Therefore, at the end of the recursion, the resulting implementation has $O(\log(Depth_M))$.

Unlike the regular multiplexer-based implementation for PTL circuits in [17] that obtains a logarithmic depth for the functions, where the cutset has only two nodes (for instance, *xor* function), our use of one-hot multiplexers and recursive bipartitioning results in a logarithmic depth implementation for any circuit, irrespective of the cutset size. A logarithmic depth PTL implementation for any BDD is neither claimed nor proved in [17] whose approach relies on the use of regular multiplexers, although they demonstrate such an implementation for *xor* function. It is clear that if, instead of one-hot multiplexers, regular multiplexers are employed during decomposition, the reduction in depth is, somewhat, lower: a regular multiplexer based implementation has depth with a lower bound $\Omega(\log(Depth_M)\log(Min_{Cutsize}))$ and an upper bound $O(\log(Depth_M)\log(Max_{Cutsize}))$, where $Min_{Cutsize}$ and $Max_{Cutsize}$ denotes the minimum and the maximum of cardinalities of cutsets at any bipartitioning stage, respectively.

B. Area Estimation

The flow network described above requires an estimate of the area cost for each candidate node in the BDD. To generate this estimate, we assume a BDD-mapped PTL implementation with pass transistors and buffers after every k transistors. The contribution of a node to the area cost is estimated as the sum of

- the area of the PTL implementation of the BDD rooted at a given node, and
- the area of the PTL network that terminates on the given node.

This area cost is computed in linear time by a postorder traversal of the network. At multi-fanout BDD nodes, the area cost is divided by the number of fanout edges. This heuristic is similar to the one employed in technology mappers for standard cell libraries such as [20], [24].

C. Complexity Analysis

The computation time required to find essential and candidate nodes is linear in the size of the BDD network, as it involves a traversal, similar to the critical path method [25], of the BDD. The time required for area cost estimation is also linear in the size of the network. The only computationally expensive procedure is the max-flow min-cut algorithm, which is applied to find an optimum cut with minimum area penalty. The time complexity of the Edmonds-Karp implementation of the Ford-Fulkerson algorithm for finding the max-flow and min-cut is $O(\|V\|\|E\|^2)$, where V (E) is the set of nodes (edges) in the flow network [23]. While this seems expensive, in practice the time complexity of this algorithm is hardly reflected in the CPU times for the following reasons:

1) In our case, the size of flow network is very small as compared to the size of the BDD to be bipartitioned, since only a small fraction of all the BDD nodes qualify as either essential or candidate nodes.
2) Since the capacity assignment to the nodes is such that essential nodes are assigned very small capacity and are always in an optimum cut, a majority of flow augmentations are associated only with the part of the network that involves paths with candidate nodes. In other words, the flow network effectively contains only the candidate nodes and related edges.

Since bipartitioning is applied recursively, the following recurrence equation describes the time complexity of the entire algorithm for a BDD containing n nodes

$$T(n) = T(n_{select}) + T(n_{data}) + f(n) \qquad (1)$$

where, n_{select} and n_{data} are the number of nodes in BDD's for *select* and *data* functions, respectively. The number of recursions is bounded as noted in the proof of Theorem 3.1, and this ensures the termination of the algorithm in a finite time. Although $(n_{select} + n_{data}) \geq n$ due to possible increase in the number of nodes for *select* functions because of possibly high number of *select* functions, the one-hot encoding and the node-sharing among *select functions* result in n_{select}, which is usually smaller than (or equal to) n as observed in practice on ISCAS'85 benchmarks. Being the number of BDD nodes for the data functions, which are part of original BDD, n_{data} is also smaller than (or equal to) n. To account for the possible increase in the number of nodes and to simplify the equation, $T(n_{select}) +$

$T(n_{data})$ can be approximated as $2T(n/a)$, where $1 < a < 2$. Using such an approximation, the equation can be re-written as shown below

$$T(n) = 2T(n/a) + f(n) \quad (2)$$

In the above equation, $f(n) = \Omega(n)$ due to linear time complexity of delay analysis and $f(n) = O(n^3)$ assuming the size of flow network to be same as that of the size of the BDD. Note that assumption that the size of the flow network is $O(n^3)$ is highly pessimistic for the two reasons mentioned before, but is useful enough to derive a loose upper bound on the time complexity. Applying the master theorem [23], the above equation yields

$$T(n) = \Omega(n^{\log_a 2}), T(n) = O(n^3) \quad (3)$$

In practice, the algorithm requires CPU times that vary between super-linear to quadratic in number of BDD nodes, and in absolute terms, the run-times for the ISCAS'85 benchmarks are of the order of seconds.

IV. Delay Modeling and Analysis

To identify essential and candidate nodes, it is important to perform delay analysis using a delay model that has good fidelity. The Elmore delay model [26] satisfies such a requirement while being computationally inexpensive, and has even been used in the past for timing verification of complex microprocessor chips [27]. We adapt this model for computing delays in PTL networks that are mapped directly from BDD's. It is important to note that the Elmore delay model is utilized only for identifying essential and candidate nodes during the synthesis stage, while for the post-synthesis delay analysis of netlists for the PTL and static CMOS circuits, we employ the widely used non-linear delay model (NLDM) [4], [28] that involves other factors, such as consideration of the slope of the input signal transition and load.

In the following subsections, we describe the adaptation of Elmore delay model to PTL circuits, the corresponding delay analysis procedure and the post-synthesis delay model.

A. Delay Model for BDD-mapped PTL Circuits

The insertion of buffers that break up long transistor chains can result in short pass transistor segments such as that shown in Figure 7(a). Each pass transistor in such a segment can be modeled using an RC π model, where R denotes the resistance of the transistor and C denotes the drain/source capacitance. Pass transistors offer different resistance for rising and falling transitions: typically, for NMOS pass transistors, falling transitions are faster than the rising transitions. To account for this, two different values of resistance, one for the rising and one for the falling transition, are associated with each pass transistor. The value of the resistance is obtained by characterization of a pass transistor using circuit simulator such as SPICE.

Figure 7(b) shows the corresponding RC network for the pass transistor segment in Figure 7(a). This is a special case where the pass transistor segment maps to an RC line. For more complex BDD's, it is likely that the pass transistor segment may be more complex, as illustrated in Figure 7(c). From this picture, it can be seen that BDD-mapped PTL networks, when modeled using an RC π model, look like an RC mesh rather than an RC line. In such a mesh, directions can be assigned to the resistive edges since transistors act as unidirectional switches; such methods have long been used in delay analysis tools and have even been used in very old timing verifiers such as Crystal [29]. Therefore, we can model a BDD-mapped PTL network, using RC π models, as a set of RC directed acyclic graphs (DAG's) between buffers, as shown in the figure.

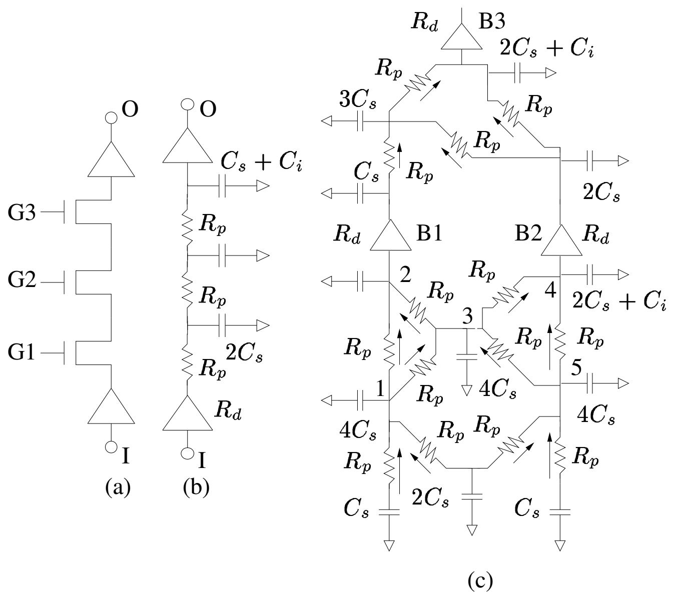


Fig. 7. (a) A PTL circuit segment with three pass transistors in series. (b) The equivalent RC model for the PTL segment in (a). (c) The equivalent RC network with assigned directions for resistances corresponding to PTL implementation in Figure 2(b). Here, R_p (R_d) corresponds to pass transistor (driver) resistance, while C_s and C_i represent the source (as well as drain) capacitance and the inverter input capacitance, respectively.

Delay analysis for an RC tree can be performed using tree traversal in linear time in the size of a tree [25], while delay analysis for RC meshes using tree/link partitioning requires $O(nm^2)$ time, where n is the number of edges in a tree and m, the number of links, which when removed from an RC mesh results in an RC tree [30]. In our case, however, the resulting RC structure is neither a tree nor a mesh, but an RC DAG – a more complex structure than trees and perhaps, simpler than meshes, from a graph theoretic perspective. The delay for an RC DAG can be defined as the maximum of the delay along any path and the Elmore delay along any path is defined similar to RC trees, as in [25]

$$D_{Elmore} = \Sigma_{i \in path} R_i \cdot C^i_{downstream} \quad (4)$$

This model of the network as an RC DAG does not take into account the logical dependencies between signals. If we consider these, we can see that depending on the input assignments, some of the resistances can be treated as open circuits when the corresponding gate signals to the transistors are not high and may be removed from the RC network. The RC DAG's that model BDD-mapped PTL networks have a peculiar property that arises from a well known property of BDD's, namely, that for any assignment of inputs, only one path from a terminal node to a given node is active. The implication of this for RC DAG's is stated by the following observation.

Observation For any assignment of inputs, a given RC DAG must reduce to an RC forest.

Proof The proof proceeds by contradiction. Assume that for some assignment of inputs, a given RC DAG does not reduce to an RC forest. It implies that there is a cycle, which in turn implies that there exists a node which is driven by two different signals – a contradiction, since only one path to any node in a BDD is active.

More details on the reduction of RC DAG's onto RC forests can be found in [31].

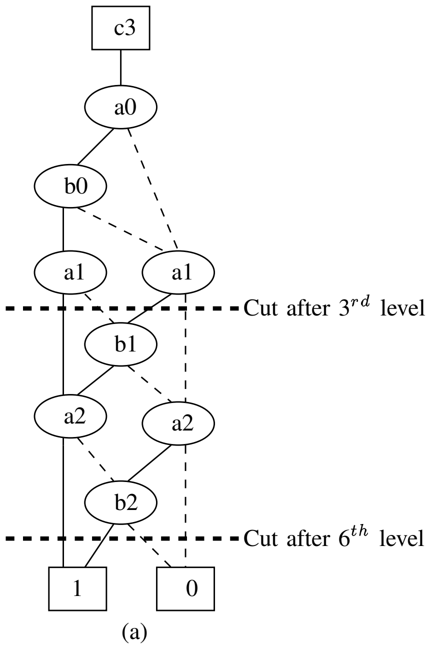


Fig. 8. A pictorial illustration of the inverter insertion heuristic in [32]: inverters are used for the edges that are indicated as being cut. Under such an assumption, at most 2^k assignments must be considered for the part of the BDD between inverters.

B. Delay Analysis for BDD-mapped PTL Circuits

To analyze RC DAG's, we may have to consider all possible input assignments; this number of such assignments is exponential in the number of inputs, assuming that the primary inputs are independent of each other. Fortunately, we can assume a reasonable PTL implementation from a given BDD that will allow us to perform delay analysis in linear time[4].

One such implementation[5] is shown in Figure 8. This assumes that each BDD node is mapped on to a PTL multiplexer and the edges that cross every multiple of the k^{th} level (edges crossing the cuts in the figure, where $k = 3$) have inverters/buffers on them [32]. This buffer insertion heuristic ensures that there is an inverter after at most k transistors in series. In case of such a BDD-mapped PTL network that has buffers or inverters after at most k transistors in series, where k is bounded by a small constant, it is adequate to consider only 2^k different assignments for parts of the RC DAG that lies between successive buffer levels to find the maximum delay. Each edge will have to be traversed no more than 2^{k-1} times, as stated by the following observation.

Observation For a PTL network that has at most k transistors in series between buffers, the total number of edges in all of the forests corresponding to various input assignments is bounded by $2^{k-1} \cdot \|E\|$, where $\|E\|$ is the number of edges in the RC DAG.

Proof Each edge in the BDD is associated with the true or complementary form of a variable in the BDD. Since inverters are inserted in such a way that at most k variables lie between two successive inverter insertion levels, only 2^k different assignments must be considered for a given part of the DAG. For exactly half of these assignments, a variable associated with the edge is true, and therefore, any edge can appear only in half of the forests. Since there are $\|E\|$ edges in the DAG corresponding to a BDD, the total number of edges in all the trees is bounded by $O(2^{k-1}\|E\|)$.

Algorithm 4.1 Algorithm for delay analysis of BDD-mapped PTL circuits

Input: Digraph $G(V, E)$ corresponding to the given BDD, number of variables n in the BDD, inverter insertion interval k, pass transistor resistance R.
Output: $D_{Bottom} \forall v \in V$

```
1:  for $level$ = 1 to $\lceil n/k \rceil$ do
2:      for $\forall v \in \{k \cdot level, k \cdot level - 1, ..., k \cdot level - (k-1)\}$, $\forall \bar{b} \in B^k$, $B = \{0,1\}$ do
3:          GetDownStreamCapacitance($v$,$\bar{b}$);
4:      end for
5:  end for
6:  In topological order, $\forall v \in V$
7:  $v.D_{Bottom} \leftarrow \max\{v'.D_{Bottom}$ + R × $v$.DownstreamCap : $(v', v) \in E\}$
8:  Procedure GetDownStreamCapacitance($v$,$\bar{b}$) {
9:  if (AllFanoutEdgesBuffered($v$)) then
10:     $v$.DownstreamCap ← $v$.Capacitance;
        // If all fanout edges are buffered
11: else
12:     $c \leftarrow 0$;
13:     for $\forall e(v, v') \in$ fanout($v$) do
14:         if $e$ is not buffered && $\bar{b} \in$ BooleanExpression($e$) then
15:             $c \leftarrow c$ + GetDownStreamCapacitance($v'$,$\bar{b}$);
                // For un-buffered edges that satisfy Boolean expression
16:         end if
17:     end for
18: end if
19: $c \leftarrow c$ + $v$.Capacitance;
    // Add the capacitance at the node
20: if $c > v$.DownStreamCap then
21:     $v$.DownStreamCap ← $c$;
        // Store maximum of $c$ and current downstream capacitance
22: end if
23: }
```

Different portions of the RC DAG can be successively considered to yield a linear time delay analysis procedure. We exploit this idea in the delay analysis algorithm. Algorithm 4.1 shows the pseudocode for the same. The maximum downstream capacitance for a given node is computed by calling `GetDownStreamCapacitance()` procedure for all possible $k-$bit Boolean assignments. This procedure traverses all the fanout edges that satisfy a specific $k-$bit Boolean assignment till the buffers are reached, adds the capacitances at the visited nodes, and stores the maximum downstream capacitance. Once the maximum downstream capacitance is computed, the delays at each node can be computed by sorting the nodes in topological order and computing the maximum arrival time at each node. The following theorem states the time complexity of the delay analysis algorithm.

Theorem 4.1: The delay analysis using the Algorithm 4.1 takes no more than $O(\|E\|)$ time, where E is the set of edges in the BDD.

Proof Using the arguments from the previous observation, it can be stated that the computation of downstream capacitance requires each edge to be visited at most 2^{k-1} times. The topological sort of the nodes requires linear time in the size of graph [23]. Therefore, time required by delay analysis routine is $O(\|E\|)$ since k is a constant.

[4]Performing the delay analysis in linear time is critical to keep the time complexity of our bipartitioning algorithm reasonable, since the delay analysis procedure is invoked during each bipartitioning call of our algorithm.

[5]Although our delay analysis procedure is targeted for this particular implementation, it can be extended to consider other PTL implementations that may use different heuristics for inverter insertion.

C. Post-synthesis Delay Models

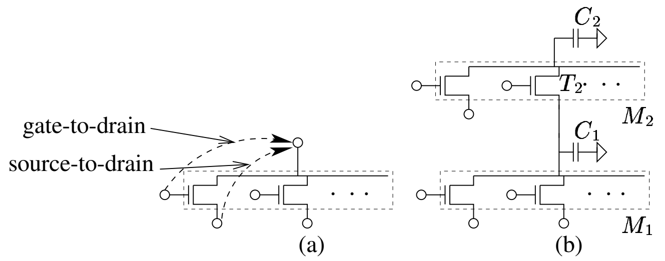


Fig. 9. (a) Timing arcs for delay analysis of PTL circuits showing two arcs, gate-to-drain and source-to-drain, for a pass transistor. (b) If pass transistor T_2 in a multiplexer M_2 is ON, a capacitive load (C_{load}) seen by a driver at any source terminal for a multiplexer M_1 is related to C_1 and C_2 as follows: $C_{load} = C_1 + C_2$, assuming zero capacitances at the sources of M_1 and no shielding effect in pass transistors.

1) Post-synthesis Delay Model for PTL: Figure 9 (a) shows timing arcs in PTL circuits, which correspond to two possible paths going through each transistor. Nonlinear delay models (NLDM's) are a popular way of representing the delays on the arcs of a timing graph. The widely used nonlinear Synopsys delay model for timing analysis involves the following equation [28]:

$$\tau = \alpha \cdot C + \beta \cdot \tau_r + \gamma \cdot C \cdot \tau_r + \delta \qquad (5)$$

In the above equation, C stands for the load capacitance, τ_r the transition slope of input signal, τ the delay from input pin to output, while α, β, γ, and δ are the parameters obtained by characterization employing a circuit simulator such as SPICE. Each timing arc in the timing graph has four NLDM parameters associated with it, which are, typically, stored in lookup tables. The sizes of these lookup tables blow up when different supply voltages and temperatures are considered. To overcome this limitation, scalable polynomial delay models (SPDM) [33] are currently employed in commercial tools. However, NLDM is still accurate with a single supply voltage and a given temperature. Therefore, for comparison purposes at the logic synthesis level, where only a single supply voltage and a single temperature is considered, non-linear delay model of the form of Equation 5 is still valid, and we use the same delay model for static CMOS and PTL synthesis results.

Adapting NLDM to timing analysis for PTL circuits requires the computation of downstream capacitance that may be beyond the given PTL multiplexer, as shown in Figure 9 (b). We employ a DAG traversal, which is similar to that of the procedure shown in Algorithm 4.1, to compute the downstream capacitance by traversing the downstream DAG. Unfortunately, the resulting pass transistor network does not possess a structure that will allow the incorporation of logical dependencies between signals during computation of downstream capacitance. This is because the accounting for these logical dependencies is at least as difficult as the NP-hard *dynamic path sensitization* problem, as pointed out in [34]. This differs from the problem of delay analysis employed during recursive bipartitioning, where the gates of all of the multiplexers are controlled by primary inputs, which, however, is not true in case of the pass transistor network obtained after recursive bipartitioning. For this reason, the analysis ignores logical dependencies and computes the downstream capacitance by simply traversing the DAG until the buffers are reached. This may result in an overestimate of the downstream capacitance and hence, an overestimate of the delay. Apart from logical dependencies, another source of pessimism in the capacitance estimate is the shielding effect due to the (nonlinear) resistance of the pass transistors that are ON.

Example	Transistors #	SPICE delay ps	Timing analysis delay ps
rd53	72	196.10	205.54
rd73	144	401.40	473.65
rd84	194	501.50	628.34
parity	100	999.90	1172.64
9sym	102	462.40	618.50

TABLE III
COMPARISON OF SPICE DELAYS WITH THE DELAYS OBTAINED FROM STATIC TIMING ANALYSIS USING NLDM ON SEVERAL COMBINATIONAL BENCHMARK CIRCUITS.

Once the downstream capacitance for all multiplexers is computed, the precharacterized NLDM parameters are used to compute the delays in an entire PTL circuit. After precharacterization, it was verified that the delay estimated by the model was indeed an overestimate, as compared to SPICE, as illustrated on several benchmark circuits in Table III. All of these circuits were mapped directly on to a PTL network with inverters with weak pull-ups inserted after every three series-connected transistors. Because of the direct mapping, the critical paths in these circuits are long and contain at least as many number of transistors as there are primary inputs (besides the inverters with weak pull-ups). Technology parameters for 0.13μm technology [6] are used for SPICE simulations and all of the transistors have a length of 0.13μm, while the widths of the transistors are as follows. The widths of NMOS pass transistor are 0.91μm, the widths for transistors in inverters with weak pull-ups are $w_n = 1.04\mu$m, $w_p = 2.08\mu$m, and $w_{pull-up} = 0.26\mu$m, where w_p, w_n, and $w_{pull-up}$ are widths of PMOS, NMOS, and weak pull-up, respectively. The critical paths are determined employing static timing analysis (STA) and simulated by applying appropriate stimulus. The transition time for rising as well as falling transition of input signal is 50ps. We can verify from the table that the delays estimated by static timing analysis are always overestimates. For long critical paths, overestimates tend to be high because of the cumulative effect.

2) Post-synthesis Delay Model for Static CMOS: We employ the same Equation 5 for the static timing analysis of static CMOS standard cell circuits. For each cell, all input-to-output timing arcs are precharacterized for NLDM parameters employing SPICE. We note that in case of static CMOS circuits, there is no pessimism in capacitive load estimation, unlike PTL circuits, since all the inputs are always connected to the gates of the transistors.

V. EXPERIMENTAL RESULTS

A. Experimental setup

The algorithms described in sections III and IV are implemented in a C++ program called Pass Transistor Logic Synthesizer (PTLS). For all of our experiments, the BDD package CUDD [35] is employed for generating BDD's, along with sifting [36] for variable ordering. We use NMOS transistors as pass transistors and use inverters with weak pull-ups after every three pass transistors in series. The inverters with weak pull-ups are also inserted to drive the gates of transistors in one-hot multiplexer for the implementations obtained by our recursive bipartitioning technique. We synthesize both the PTL and static CMOS circuits (which are used to compare the PTL circuits against) in a 0.13μm technology [6]. All transistor lengths are set to 0.13μm, and the following two sets of transistor sizes are employed for the PTL implementations:

- **Set 1**: All NMOS pass transistors have $w_n = 1.82\mu$m and inverters have sizes $w_p = 4.16\mu$m and $w_n = 2.08\mu$m, while the weak pull-up transistor in each inverters are sized to a width of 0.52μm.

- **Set 2**: All NMOS pass transistors have $w_n = 0.91\mu$m and inverters have sizes $w_p = 2.08\mu$m and $w_n = 1.04\mu$m, while the weak pull-ups in the inverters are sized to a width of 0.26μm. In other words, all the transistors in **Set 2** have half the widths of the corresponding transistors in **Set 1**.

For static CMOS circuits, we choose the lib2.genlib library in [20] and add at least two strengths ($w_{effective} = 0.78\mu$m and $w_{effective} = 1.56\mu$m) for each of the cells in the library. Simpler gates such as inverters, NAND's (up to four inputs), and NOR's (up to four inputs), have up to four strengths ($w_{effective} = 0.78\mu$m, 1.56μm, 2.34μm, 3.12μm).

Under this sizing scheme, we see that PTL circuits have uniform sizes for pass transistors and inverters, while the static CMOS implementations use better sizing, with each gate having several choices for the transistor sizes. In spite of this, we show that PTL results in implementations that have the same (or better) delay as that of static CMOS implementation with a significant area advantage in case of arithmetic, error correcting, and some control circuits in ISCAS'85 benchmark suite. If we allow a larger variety of transistor sizes for PTL circuits, it is likely that these results will improve even further in favor of PTL.

B. Synthesis Procedure

Static CMOS circuits are preprocessed by running *script.rugged* in SIS [20] before performing technology mapping for optimizing delays. For PTL synthesis, we use the same Boolean network obtained from *script.rugged*, and create a multilevel BDD representation. Our recursive bipartitioning procedure is then applied level-by-level on this multilevel BDD representation.

While creating this multilevel BDD representation, it is important to control the number of BDD nodes, since the number of transistors in the resulting implementation depends on this number. It has been shown in [11] that the use of traditional multilevel boolean network optimization, followed by the construction of BDD's for nodes in the network, results in reasonable BDD sizes. These sizes are comparable to those obtained by applying area-oriented pass transistor logic synthesis techniques such as [14]. We use these multilevel BDD representations, which have reasonable sizes, for delay oriented decomposition. Further improvements may be possible if the multilevel BDD's are preprocessed employing algorithms such as *eliminate*, as in [14], and by using better variable ordering heuristic such as symmetric sifting.

C. Analysis of Results on ISCAS'85 Benchmarks

Table IV shows the area/delay comparison between PTL circuits and their corresponding static CMOS implementations for all of the ISCAS'85 benchmarks. In this comparison, the PTL circuits in Table IV have transistor sizes from **Set 1** and **Set 2** described in Section V-A. For the same table, Column 1 shows the name of the benchmark and its functionality while Columns 2 and 3 show the area and delay, respectively, for the static CMOS implementation. Columns 4 and 5 show the area and delay, respectively, for the PTL implementation with the transistor sizes from **Set 1**, while Column 6 shows the CPU time required for our PTL synthesis algorithm on a 400 MHz Sun Ultra-Sparc 60 machine. Columns 7 and 8 show the area and delay, respectively, for the PTL implementation with the transistor sizes from **Set 2**.

For the comparison between static CMOS implementations and PTL implementations with transistor sizes from **Set 1**, the following observations can be made from Table IV.

- For circuits that implement error correcting codes, namely, C1355, C1908, and C499, for the multiplier circuit, C6288, and for the arithmetic logic unit (ALU) and control circuit, C880, PTL implementations are superior in terms of area as well as delay. On average, the area advantage is 30%, while the delay advantage is 31%.
- For ALU and Control circuits such as C2670, C7552, and C3540, the PTL implementations are superior in terms of area but could not match the static CMOS delay. On average, the area advantage is 84%, while the delay disadvantage is 22%. With the area numbers strongly favoring PTL, it is likely that static CMOS delays may be matched with sizing for PTL circuits, perhaps even while retaining some area advantage.
- For the priority decoder circuit, C432, PTL has a marginal delay advantage of 4% at the cost of a 15% area increase.
- For the ALU and selector circuit, C5315, PTL produces inferior results both in terms of area as well as delay. The area disadvantage is minor, at 2%, while the delay disadvantage is 39%.

Note that the observations are made with a pessimistic delay model for PTL, as described in Section IV-C. From the above observations, the following conclusions can be arrived at:

- For the ALU and control circuit, C5315, static CMOS results in a superior implementation. In case of other ALU and control circuits such as C2670, C3540, and C7552, static CMOS yields a superior delay, but with a significant area cost. For the ALU and control circuit, C880, using even naïve transistor sizing PTL results in a superior implementation as compared to static CMOS. All these circuits contain arithmetic components such as adders apart from control logic. Since these circuits contain *nand* intensive control logic, scripts in SIS [20], which are skewed towards control logic synthesis, perhaps destroy the structure of arithmetic components in these circuits, and PTL implementation due to our synthesis algorithm with naïve transistor sizing is not able to match (or outperform) the static CMOS delay consistently. PTL implementations still have a good area advantage that can be used by a sizer to match delays due to their static CMOS counterparts.
- The PTL implementation provides large area savings and improved delays in case of the purely arithmetic and error correcting circuits (C1355, C1908, C499, C6288) as compared to the static CMOS implementations. In these cases, SIS [20] perhaps is not able to destroy the structure as much, since most of these circuits are heavily *xor* dominated circuits.

A comparison between PTL implementations with smaller transistor sizes and static CMOS implementations is also shown in Columns 7 and 8 in Table IV. For these columns, the PTL circuit implementations use transistor sizes that are chosen according to **Set 2** in Section V-A. With the smaller transistor sizes, the delays in PTL circuits are degraded, as expected. Although the transistor sizes are halved, the delay in static CMOS is still matched in case of C499, and is within 20% for the circuits C1355, C1908, C6288, but with a large average area advantage of 166%. Given such an area margin, it is likely that an intelligent sizer may be able to match the delays in static CMOS circuits, while maintaining an area that is less than that of the PTL circuits shown in Column 4 of Table IV, which use double the transistor sizes of this case. For the set of circuits that includes ALU and control circuits such as C2670, C7552, C432, C880, C3540, and C5315, the average area advantage is 184%, while delays degrade by 35%, on an average. It is likely that improved transistor sizing in these cases may improve the results in favor of PTL, since as shown in Table IV, simplistically doubling the sizes of

Example (Functionality)	Static CMOS		PTL **Set 1**			PTL **Set 2**	
	Area	Delay	Area	Delay	CPU	Area	Delay
	μm^2	ps	μm^2	ps	s	μm^2	ps
C1355 (Error correcting codes)	4886.4	962.2	3521.6(38%)	599.2(60%)	00.18	1760.8 (177%)	1009.4 (-4%)
C1908 (Error correcting codes)	5157.2	1205.9	3726.4(38%)	980.8(20%)	00.25	1863.2 (176%)	1432.9 (-15%)
C2670 (ALU and Control)	12307.9	1208.2	8781.9(40%)	1660.2(-27%)	02.78	4390.9 (180%)	2132.6 (-43%)
C499 (Error correcting codes)	4784.0	938.7	3259.4(46%)	630.4(48%)	00.16	1692.7 (182%)	929.7 (1%)
C432 (Priority Decoder)	3370.5	1334.3	3981.6(-15%)	1278.3(4%)	00.23	1990.8 (69%)	1758.6 (-24%)
C6288 (16-bit Multiplier)	29592.5	4799.2	25771.8(14%)	4152.3(15%)	17.73	12885.9 (129%)	5563.3 (-13%)
C7552 (ALU and Control)	32051.5	1323.0	16649.5(92%)	2112.1(-37%)	10.71	8324.7 (285%)	2528.2 (-47%)
C5315 (ALU and Selector)	24042.9	1355.5	24612.1 (-2%)	2233.6 (-39%)	17.48	12306.0 (95%)	2668.5 (-49%)
C880 (ALU and Control)	5579.0	1080.0	4748.5 (17%)	952.8 (13%)	01.13	2374.2 (134%)	1430.7 (-24%)
C3540 (ALU and Control)	34280.2	1796.1	15409.0 (122%)	1850.3 (-2%)	22.42	7704.5 (344%)	2454.9 (-26%)
Avg. Improvement			**39%**	**5%**		**177%**	**-24%**

TABLE IV
AREA/DELAY COMPARISONS FOR STATIC CMOS AND PTL IMPLEMENTATIONS OF ALL ISCAS'85 BENCHMARKS.

all transistors results in a large improvement in delay while retaining significant area savings in most cases.

D. Comparison with Previous PTL Approaches

We now present a comparison of our PTL implementations with those due to previous PTL synthesis approaches such as [13], [17]. We emphasize that these approaches and that of [16] report depth in terms of number of series transistors as a delay and do not mention actual delays, which depend on capacitive load for a particular transistor and the slope of an input signal at that transistor. This is particularly important in case of PTL, since the load seen by a transistor varies significantly depending on whether transistors in downstream multiplexers are ON. Previous approaches have completely ignored this issue by considering the depth as a delay metric, which is a reasonable measure at technology independent level abstraction. It, however, suffers from inaccuracies, as it ignores load and transition time of signals, and therefore, may not even correlate well with delay due to NLDM for the mapped netlists. Since previous approaches have only reported depths, Table V shows the comparison using that metric and number of transistors in PTL implementation. The table shows the number of transistors required for the PTL implementation of the ISCAS'85 circuits due to [17], [13], and our approach in columns 2, 3, and 4, respectively, while columns 5, 6, and 7 show depths, respectively, due to [17], [13], and our algorithm. It is worth mentioning that comparison with [17] is not fair (to our method), since the underlying BDD's for their and our approach are not the same: in their work, to improve the optimization potential they heuristically grow the BDD's till the number of nodes is smaller than 100, while our method and Buch's algorithm [13] do not perform such pre-processing and use the networks obtained by *script.rugged* in SIS [20]. Since we have proved in Section III-A that our algorithm and application of one-hot multiplexer can result in logarithmic depth implementation, which is not guaranteed by [17] in their work because of the heuristic nature of the algorithm, it is likely that our approach will result in smaller depths as compared to their method, if the underlying BDD's are the same. The comparison with [13] shows that our approach results in 10% smaller depths, on an average, with 42% average reduction in the number of transistors. The inconsistencies in the improvements over all the benchmarks can be attributed to inter-node optimizations such as composition employed in [13], which changes BDD structure between different nodes in a Boolean network, while our approach focuses on BDD optimizations at a given node in the network.

Example	Number of Transistors			Depth		
	[17]	[13]	PTLS	[17]	[13]	PTLS
C1355	1474	1969	1037	24	34	21
C1908	1450	2116	1145	25	39	32
C2670	2144	3198	2876	18	28	33
C499	1490	1947	998	20	26	21
C432	908	979	1110	31	47	25
C6288	10878	10787	7794	107	159	153
C7552	7220	13268	5347	26	38	54
C5315	4842	8277	8221	30	47	43
C880	1332	1622	1467	19	32	31
C3540	4214	4997	4757	32	52	41
Total	35976	49160	34752	332	502	454

TABLE V
COMPARISON OF THE NUMBER OF TRANSISTORS AND DEPTHS RESULTING FROM OUR APPROACH WITH PREVIOUS PTL APPROACHES [13], [17]

VI. CONCLUSION

We have proposed a delay oriented synthesis algorithm for PTL circuits that uses recursive bipartitioning of BDD's. Based on the experimental results, we answer the two questions raised in Section I-A.

1) Using the delay-driven synthesis algorithm, naïve uniform transistor sizing for PTL circuits and a pessimistic delay model for PTL, we have shown that PTL can certainly improve upon the delays in static CMOS circuits by 30%, with a significant area advantage of an average of about 31% for an arithmetic circuit, error correcting circuits, and a control circuit.
2) For control circuits, static CMOS may sometimes, but not always, result in superior implementations than PTL in terms of area as well as delay.
3) Some control circuits may be implemented well in PTL with slightly degraded delays as compared to static CMOS, but with large area savings. Allowing multiple transistor sizes for PTL may improve the results in favor of PTL, but this has not been explored in our work.

ACKNOWLEDGMENT

The first author would like to thank Michel Berkelaar of Magma Design Automation, Prof. Andre Reis of Universidade Federal do Rio Grande do Sul (UFRGS), Brazil, and Clive Bittlestone of

Texas Instruments for valuable suggestions and discussions on PTL synthesis.

References

[1] International technology roadmap for semiconductors, 2001 edition: Design. `http://public.itrs.net/Files/2001ITRS/Design.pdf`, 2001.

[2] K. Yano, T. Yamanaka, T. Nishida, M. Saito, K. Shimohigsahi, and A. Shimizu. A 3.8ns CMOS 16 x 16 multiplier using complementary pass transistor logic. *IEEE Journal of Solid State Circuits*, 25(2):388–395, April 1990.

[3] K. Yano, Y. Sasaki, and K. Rikino. Top-down pass transistor logic design. *IEEE Journal of Solid-State Circuits*, 31(6):792–803, June 1996.

[4] N. H. E. Weste and K. Eshraghian. *Principles of CMOS VLSI design: A systems perspective*. Addison-Wesley, New York, New York, Second edition, October 1994.

[5] J. M. Rabey. *Digital integrated circuits: A design perspective*. Prentice-Hall India, New Delhi, India, Second edition, September 2000.

[6] Berkeley predictive technology model. `http://www-device.eecs.berkeley.edu/~ptm/download.html`.

[7] C. Bittlestone, A. Hill, V. Singhal, and N. V. Arvind. Architecting ASIC libraries and flows in nanometer era. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 776–781, June 2003.

[8] R. L. Ashenhurst. The decomposition of switching functions. In *Proceedings of the International Symposium of Theory of Switching*, volume 1, pages 74–116, April 1957.

[9] M. Davio, J-P. Deschamps, and A. Thayse. *Discrete and switching functions*. McGraw-Hill International, St. Saphorin, Switzerland, 1978.

[10] Y-T. Lai, K-R. R. Pan, and M. Pedram. OBDD-based function decomposition: algorithms and implementation. *IEEE Transactions on Computer Aided Design for IC's and Systems*, 15(8):977–990, August 1996.

[11] C. Yang and M. Ciesielski. BDD decomposition for efficient logic synthesis. In *International Conference on Computer Design*, pages 626–631, October 1999.

[12] T. Sasao. *Switching theory for logic synthesis*. Kluwer Academic Publishers, Boston, Massachusets, 2000.

[13] P. Buch, A. Narayan, A. R. Newton, and A. Sangiovanni-Vincentelli. Logic synthesis for large pass transistor circuits. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 663–670, November 1997.

[14] R. Chaudhry, T.-H. Liu, A. Aziz, and J. Burns. Area-oriented synthesis for pass transistor logic. In *Proceedings of the IEEE International Conference on Computer Design*, pages 160–167, October 1998.

[15] T.-H. Liu, M. Ganai, A. Aziz, and J. Burns. Performance driven synthesis for pass transistor logic. In *Proceedings of VLSI Design Conference*, pages 372–377, January 1999.

[16] F. Ferrandi, A. Macii, E. Macii, M. Poncino, R. Scarsi, and F. Somenzi. Symbolic algorithms for layout-oriented synthesis of pass transistor logic circuits. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 235–241, November 1998.

[17] C. Scholl and B. Becker. On the generation of multiplexer circuits for pass transistor logic. In *Proceedings of Design Automation and Test in Europe*, pages 372–378, March 2000.

[18] R. E. B. Poli, F. R. Schneider, R. P. Ribas, and A. I. Reis. Unified theory to build cell-level transistor networks from BDD's. In *Symposium on Integrated Circuits and Systems Design*, pages 199–204, September 2003.

[19] G. R. Cho and T. Chen. Synthesis of single/dual rail mixed ptl/static logic for low power applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* , 23(2):229–242, February 2004.

[20] E. M. Sentovich. SIS: A system for sequential circuit synthesis. Memorandum No. UCB/ERL M92/41, May 1992.

[21] R. S. Shelar and S. S. Sapatnekar. Recursive bipartitioning of BDD's for performance driven pass transistor logic synthesis. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, November 2001. 449–452.

[22] R. S. Shelar and S. S. Sapatnekar. Efficient algorithm for low power pass transistor logic synthesis. In *Proceedings of Asia South Pacific Design Automation Conference*, January 2002. 87–92.

[23] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. Prentice-Hall India, New Delhi, India, 1998.

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

[25] S. S. Sapatnekar and S.-M. Kang. *Design Automation of Timing-Driven Layout Synthesis*. Kluwer Academic Publishers, Boston, Massachusets, 1992.

[26] W. C. Elmore. The transient response of damped linear networks with particular regard to wideband amplifiers. *Journal of Applied Physics*, 19(2):55–63, January 1948.

[27] N. Nassif, M. P. Desai, and D. H. Hall. Robust Elmore delay models suitable for full chip timing verification of a 600 MHz CMOS microprocessor. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 230–235, June 1998.

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

[29] J. K. Ousterhout. A switch-level timing verifier for digital MOS VLSI. *IEEE Transactions on CAD of IC's and Systems*, 4(3):336–349, July 1985.

[30] P. K. Chan and K. Karplus. Computing signal delay in general RC networks by tree/link partitioning. *IEEE Transactions on CAD of IC's and Systems*, 9(8):898–902, June 1990.

[31] R. S. Shelar. *Synthesis for Nanometer Technologies*. PhD thesis, University of Minnesota, Minneapolis, May 2004.

[32] M. Munteanu, I. Bogdan, P. Ivey, and L. Seed. Single-ended pass transistor loic (SPL) - A design handbook. `http://www.shef.ac.uk/eee/esg/lowpower/pdf-papers/d4.5.pdf`, 2001.

[33] Scalable polynomial delay model. `http://www.synopsys.com/products/library/lib_comp_spdm.html`.

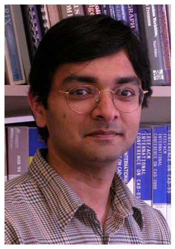
[34] M. P. Desai and Y. T. Yen. A systematic technique for verifying critical path delays in a 300 MHz Alpha CPU design using circuit simulation. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 125–130, June 1996.

[35] F. Somenzi. CUDD: CU Decision Diagram package, Release 2.3.0. `http://vlsi.colorado.edu/~fabio/CUDD/`.

[36] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 42–47, November 1993.

Rupesh S. Shelar (S'00-M'05) received the B.E. degree in instrumentation engineering from the Marathwada University, Aurangabad, India in 1997, the M.Tech. degree in electrical engineering with specialization in microelectronics from the Indian Institute of Technology, Mumbai in 1999, and the Ph.D. degree in electrical engineering from the University of Minnesota, Minneapolis in 2004.

He was a software engineer with Silicon Automation Systems, India from 1999 to 2000. He spent the summers of 2002 and 2003 at Strategic CAD Labs, Intel researching congestion-aware logic synthesis. He is currently a senior component design engineer in the Enterprise Microprocessor Group at Intel, where he works on the backend design methodology for a 65 nm Pentium 4 design. He has authored 11 papers in refereed conferences and journals. His research interests include logic synthesis, physical synthesis, and physical design.



Sachin S. Sapatnekar (S'86-M'93-SM'99-F'03) received the B.Tech. degree from the Indian Institute of Technology, Bombay in 1987, the M.S. degree from Syracuse University in 1989, and the Ph.D. degree from the University of Illinois at Urbana-Champaign in 1992. From 1992 to 1997, he was an assistant professor in the Department of Electrical and Computer Engineering at Iowa State University. He is currently a Professor in the Department of Electrical and Computer Engineering at the University of Minnesota. He has coauthored two books, "Timing Analysis and Optimization of Sequential Circuits," and "Design Automation for Timing-Driven Layout Synthesis," and is a co-editor of a volume, "Layout Optimizations in VLSI Designs," all published by Kluwer. He has been an Associate Editor for the IEEE Transactions on VLSI Systems, the IEEE Transactions on CAD, and the IEEE Transactions on Circuits and Systems II, has served on the Technical Program Committee for various conferences, as Technical Program and General Chair for the Tau workshop and the International Symposium on Physical Design. He is currently a Distinguished Visitor for the IEEE Computer Society and a Distinguished Lecturer for the IEEE Circuits and Systems Society. He is a recipient of the NSF Career Award and SRC Technical Excellence Award, and received the best paper awards at DAC 1997,2001, and 2003, and at ICCD 1998.