Logic Synthesis for Large Pass Transistor Circuits

Premal Buch Amit Narayan A. Richard Newton A. Sangiovanni-Vincentelli

Department of Electrical Engineering & Computer Sciences
University of California, Berkeley, CA 94720

Abstract

Pass transistor logic (PTL) can be a promising alternative to static CMOS for deep sub-micron design. In this work, we motivate the need for CAD algorithms for PTL circuit design and propose decomposed BDDs as a suitable logic level representation for synthesis of PTL networks. Decomposed BDDs can represent large, arbitrary functions as a multi-stage circuit and can exploit the natural, efficient mapping of a BDD to PTL.

A comprehensive synthesis flow based on decomposed BDDs is outlined for PTL design. We show that the proposed approach allows us to make logic-level optimizations similar to the traditional multi-level network based synthesis flow for static CMOS, and also makes possible optimizations with a direct impact on area, delay and power of the final circuit implementation which do not have any equivalent in the traditional approach. We also present a set of heuristical algorithms to synthesize PTL circuits optimized for area, delay and power which are key to the proposed synthesis flow.

Experimental results on ISCAS benchmark circuits show that our technique yields PTL circuits with substantial improvements over static CMOS designs. In addition, to the best of our knowledge this is the first time PTL circuits have been synthesized for the entire ISCAS benchmark set.

1 Introduction

Static CMOS has long been the design style of choice for IC designers due to the ease of designing safe, scalable circuits. However, switching capacitances in a static CMOS circuit can be fairly large. With the shrinking feature sizes and increasing transistor counts on chips, the push for higher speed and lower power makes it necessary to look for alternative design styles which can offer better performance characteristics to static CMOS. These include pass-transistor-based logic families, domino-like dynamic logic styles etc.

Among these, pass transistor logic (PTL) circuits offer great promise. Compared to domino circuits, they are less susceptible to crosstalk problems, which is a major issue in deep sub-micron technology. Several case studies ([4][21]) have shown that PTL can implement most functions with fewer transistors than static CMOS. This reduces the overall capacitance, resulting in faster switching times and lower power. It was reported in [21] that a complementary PTL multiplier was twice as fast as conventional CMOS due to lower input capacitance and higher logic functionality. At a supply voltage of 4V, PTL designs typically consume 30% less power than static CMOS designs ([5]). To illustrate this point, we take a function F = A'+BC'. Fig. 1(a) shows our implementation of this function in PTL and Fig. 1(b) shows the corresponding static CMOS implementation. Clearly, the PTL design style can yield a circuit which can be much more compact than static CMOS. It was reported in [22] that the PTL yielded a 32% improvement in area, 29% improvement in delay and a 47% improvement in power over a static CMOS OR/NAND-based implementation of this function.

This work was supported in part by SRC contract 97-DC324 abd by DEC

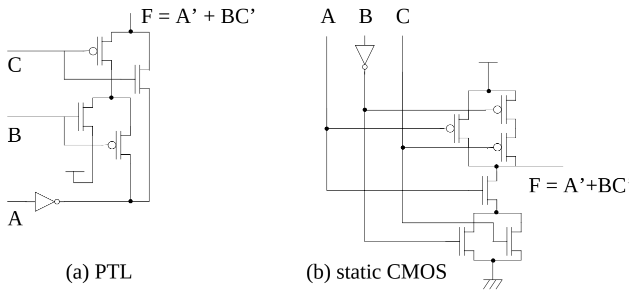


Figure 1: Comparing pass transistor and static CMOS implementations of an example function F = A'+BC'

The circuit in Fig. 1(b) can in fact, also be interpreted as a PTL circuit. The only difference between PTL and static CMOS is that in static CMOS, unlike PTL, all paths from V_{dd} to the output are connected via pMOS (the pull-up network) and paths from output to ground are connected via nMOS (pull-down network). Thus, static CMOS can be viewed as restricted case of PTL. These restrictions make the task of synthesizing safe, large static CMOS circuits easier, but reduce the potential of circuit optimization. Thus, given a methodology to synthesize safe, large circuits, PTL can be more attractive then static CMOS.

The lack of such a methodology is why the use of pass transistors in industry circuits has been very limited. While there have been several attempts in this area ([3][7][10][13][15][16][17][18][22]), limitations of some of which are discussed later in the paper, there are no algorithms which can be used to design safe, large PTL circuits. Thus, while designers can manually design very efficient small PTL circuits as in Fig. 1, a satisfactory solution to automatic synthesis of circuits realizing the expected benefits of PTL does not exist.

In this work we address this void by proposing a decomposed BDD-based approach which exploits some of the strengths of PTL logic and is scalable in that it can be used to obtain compact, multi-stage transistor-level circuits for large, arbitrary designs.

The main contribution of this work is as follows: a comprehensive synthesis flow is outlined for PTL design starting from an unoptimized logic level netlist, all the way up to generating a spice netlist. For this, a suitable logic level abstraction based on decomposed BDDs is proposed which allows us to make logic level optimizations similar to the traditional multi-level network based synthesis flow for static CMOS. This representation takes advantage of the correspondence between PTL circuits and BDDs without suffering from the drawbacks imposed by properties of monolithic BDDs. A straightforward mapping exists from this logic level abstraction to a transistor-level PTL netlist which preserves all the interconnection information. This makes possible optimizations with a direct impact on area, delay and power of the final circuit implementation. We present a set of heuristical algorithms to synthesize PTL circuits optimized for area, delay and power which are key to the proposed synthesis flow. Initial experimental results on ISCAS benchmark circuits show that our technique yields PTL circuits with substantial improvements over conventional static CMOS designs. To the best of our knowledge this is the first time PTL circuits have been synthesized for the entire ISCAS benchmark sets.

This paper is structured as follows: in Section 2, we argue why a BDD-based approach is suitable for PTL circuit synthesis and review the shortcomings of monolithic BDD-based approaches. In Section 3 we motivate decomposed BDDs as a suitable logic level abstraction for PTL synthesis. Section 4 compares the proposed decomposed BDD-based synthesis flow and the traditional approach for static CMOS. Section 5 presents decomposition techniques to obtain PTL circuits optimized for area, delay and power. Section 6 presents the experimental results. Section 7 outlines issues for future research and Section 8 concludes with a summary of this work.

2 PTL Networks and BDDs

One of the main strengths of static CMOS designs is that they are guaranteed to not have a steady-state sneak path connecting a node to both power supply and ground at the same time under some input combination. From Section 1, PTL admits more general circuit structures than static CMOS. However, it suffers from the drawback that there is no guarantee on the absence of sneak paths in the circuit. Hence, special care needs to be taken to ensure that the circuit is sneak path-free. For example, the PTL circuit in Fig. 2 requires only three transistors to implement the example function from Fig. 1. However, this circuit has a sneak path as shown, forcing the output to be connected to both ground and power supply at the same time when A=1, B=0, C=0. We therefore need a methodology to synthesize PTL circuits which ensure the absence of such sneak paths.

The basic unit in PTL is a MOS transistor which is used as a switch. When the control signal at the MOS gate is enabled, the input (drain/source) is connected to the output (source/drain). The output is in a high impedance state when the control signal is disabled. This switching characteristic of the MOS makes it very easy to implement a multiplexer in PTL as a wired OR of transistors.

A 2-input multiplexer implements the same functionality as a BDD node, with the BDD node variable corresponding to the control signal of the multiplexer and the outgoing and incoming branches of the BDD node corresponding to the inputs and output of the multiplexer respectively. Fig. 3 shows two different ways of implementing a BDD node using two MOS transistors.

Thus, the BDD representation of the target function can be very easily mapped to a multiplexer network, which in turn can be implemented compactly using pass transistors. This provides a way to construct efficient PTL circuits [15]. In fact, the PTL implementation in Fig. 1(a) corresponds to the BDD of F, as shown in Fig. 4.

The main advantage of such a BDD-based approach is that it always gives correct, sneak-path-free circuits, since at a time, only one path connecting the ground/power supply to the output is active. Using the two different implementations of a BDD node from Fig. 3 yields the two circuits shown in Fig. 5(a)and 5(b), both smaller than the static CMOS implementation in Fig. 5(c). Note that the nMOS-only implementation in Fig. 5(b) uses more transistors than the implementation in Fig. 5(a) because it needs

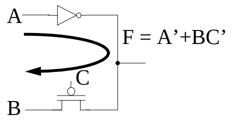


Figure 2: A PTL circuit with a sneak path

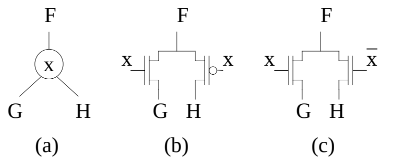


Figure 3: Implementing a BDD node in PTL

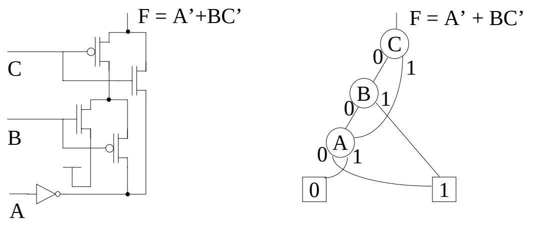
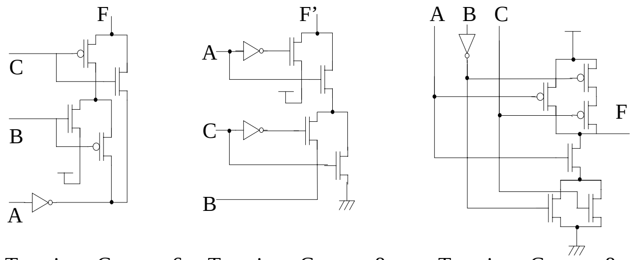


Figure 4: Comparing pass transistor implementations of the example function of Fig. 1 with its BDD



Transistor Count = 6	Transistor Count = 8	Transistor Count = 8
Gate Area = 9 λ^2	Gate Area = 10 λ^2	Gate Area = 12 λ^2
(a)	(b)	(c)

Figure 5: alternative BDD-based implementation of the example function from Fig. 1 (F = A'+BC')

signal and $\overline{\text{signal}}$ for each BDD node. However, it is quite competitive in terms of gate area. This is due to the fact that to obtain a similar current drive, pMOS has to be twice as big as nMOS in terms of the gate size (a minimum size pMOS has dimensions 3λ×λ while a minimum size nMOS is 1.5λ×λ). This results in higher active gate area per transistor in the case of static CMOS and a pMOS/nMOS PTL. Also, in a pMOS/nMOS PTL, a pMOS can be in a path propagating a "1" and an nMOS can be in a path propagating "0", resulting in output levels of V_t and V_{dd}-V_t for "0" and "1" respectively. In comparison, in the nMOS-only case, the voltage is V_{dd}-V_t for output "1", and 0V for output "0" since nMOS are good conductors of "0".

This has three advantages:

- Each nMOS is at a better operating point when propagating "0" and has a higher drive, resulting in a faster circuit.
- The output has a better noise margin, which can be particularly important if it is driving MOS gates (of buffers or subsequent stages).
- Apart from the savings in active gate area, the smaller size of nMOS also means a lower gate capacitance. This results in a lower switching capacitance for the circuit making it faster and also reducing its power dissipation.

In fact, for large circuits, we found that the overhead of generating $\overline{\text{signal}}$ was quite small (in most cases, particularly in case of large circuits, $\overline{\text{signal}}$ was required in the circuit anyway as $\overline{\text{A}}$ is in Fig. 5(a)), and the gate area savings and performance gains more than offset this. For this reason, we have used the nMOS-only implementation of Fig. 3(c) in synthesizing transistor-level circuits.

While a BDD-based PTL network can be quite compact, a naive BDD-based methodology for implementing PTL circuits suffers from the drawback that for many functions of practical interest, the size of a BDD representing the function can be exponential in the number of inputs. Also, a circuit generated from a monolithic BDD can have long chains of transistors corresponding to long paths from the root to the 0/1 terminals for the BDD. This is equivalent to implementing a single-stage static CMOS circuit and can make the circuit very slow.

A technique for generating PTL circuits in which buffers are inserted in the monolithic BDD to solve the speed problem is given in [22]. However, this approach still suffers from the BDD size problem. A multi-level pass transistor logic is introduced in [17], which tries to maximize the logic shared between different parts of the circuit by looking at the structure of a monolithic BDD. Using a monolithic BDD as the starting point and modifying its structure has two disadvantages: first, the approach will not be viable for large circuits with exponentially sized BDDs. Secondly, even when a monolithic BDD can be built, the resulting circuit is highly unoptimal in area because the optimizations are based on the topology of the BDD and not the logic implemented in it, thereby restricting the sharing to sub-graphs found in the starting monolithic BDD.

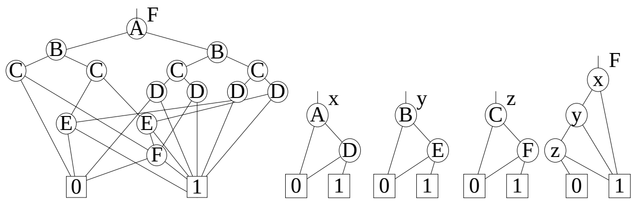
3 PTL Networks and Decomposed BDDs

We propose a synthesis approach which does not construct monolithic BDDs for the circuit at all. The common problem of the previous works outlined in Section 2 is that they try to improve a monolithic BDD-based solution. Our approach is truly multi-stage in that we always work with a multi-level representation of the PTL circuit which is similar to the traditional multi-level network for static CMOS. For such a flow, we propose decomposed BDDs as a suitable logic level abstraction of the circuit which exploits the correspondence between PTL circuits and BDDs without suffering from the drawbacks imposed by properties of monolithic BDDs which may be useful for logic level data representation but are unnecessary for circuit generation (e.g. canonicity).

The growth in the BDD size can be controlled by introducing new, intermediate variables during the construction of the BDD itself. These intermediate variables are called decomposition points and the resulting set of BDDs (BDDs of the decomposition points, and the BDD of the target function in terms of the primary inputs and decomposition points) is called a decomposed BDD [8]. An example of a decomposed BDD is shown in Fig. 6[1]. Note that the output of a decomposition point BDD can be a node variable for the BDDs of subsequently introduced decomposition points or the target function. From Section 2, this corresponds to the output of a decomposition point driving MOS gates in the circuits of subsequent decomposition points or the target function. The resulting circuit is then a multi-stage circuit with cells in any given stage being driven by the primary inputs and the outputs of preceding stages.

The intuition behind the savings in BDD size due to decomposition is as follows: in general, when constructing the graph of a function $F = G_1$ <op> G_2, the size of F, $|F|$, is $O(|G_1||G_2|)$, where $|G_1|$ and $|G_2|$ are the sizes of the input graphs. By introducing decomposition points for G_1 and G_2, the size of the decomposed BDD is reduced to $O(|G_1|+|G_2|)$. Thus, decomposition can be very useful when there is a memory explosion due to a difficult BDD manipulation during BDD construction. The trade-off here is that while monolithic ROBDDs are canonical for a given ordering, a decomposed BDD is not, since a BDD for a given function can be decomposed in many ways. This however does not pose a problem in our case, since we are interested in generating PTL circuits and not in manipulating BDD as a data structure.

1. Although a more efficient ordering for this monolithic BDD exists [4], for the given ordering this case serves to illustrate the potential BDD size reduction due to decomposition.



F = AD + BE + CF   x = AD, y = BE, z = CF

Figure 6: Comparing monolithic and decomposed ROBDDs

Note that our approach is orthogonal to the approach of [22] in that, decomposed BDDs can be used to obtain a compact, BDD representation of the circuit. Each individual BDD can then be optimized by the techniques presented in this work and then mapped to a transistor-level circuit with appropriate buffering using [22]. Similarly, optimization algorithms for area, delay and power presented here can be applied to BDDs generated using [17] as well. In Section 5.2, we provide some more arguments on why, from a delay perspective for large circuits, a decomposed BDD approach is better than a monolithic BDD-based approach combined with buffer insertion.

Finally, we would like to mention that the idea of introducing intermediate variables to control the size of BDDs has previously been used in [8][9] for unrelated problems. In these papers decomposition was used in a different context - In [8] decomposition was used to reduce the intermediate memory requirements during BDD construction and in [9] it was used for cycle-based simulation. In this work, we apply decomposition to construct a compact, decomposed BDD representation of the target logic function which can be directly mapped to a PTL network. The objective then is to develop decomposition techniques such that the PTL network corresponding to the resulting BDD is optimized for the desired objectives (e.g. area, speed, power).

4 A Synthesis Flow for PTL Design

Apart from proposing a decomposed BDD-based representation for PTL synthesis, a major contribution of this work is a comprehensive synthesis flow for PTL design.

Fig. 7 outlines the key steps of the traditional multi-level network based synthesis flow for static CMOS. We propose an analogous synthesis flow where a decomposed BDD is used to represent a circuit similar to the multi-level network in the traditional flow and each decomposition point BDD is manipulated similar to a complex node in the multi-level network.

A big advantage of the BDD-based PTL network design is that the one-to-one mapping between the BDD and the PTL network makes the technology mapping problem very straightforward. As a result, we can perform circuit level optimizations by manipulating the BDD. The fact that mapping preserves the circuit structure allows us to make high-level changes which can have significant impact on area, power and performance, but for which gains made at the high level hold at the circuit level as well. This addresses a big problem with the existing multi-level network based synthesis flow where technology independent

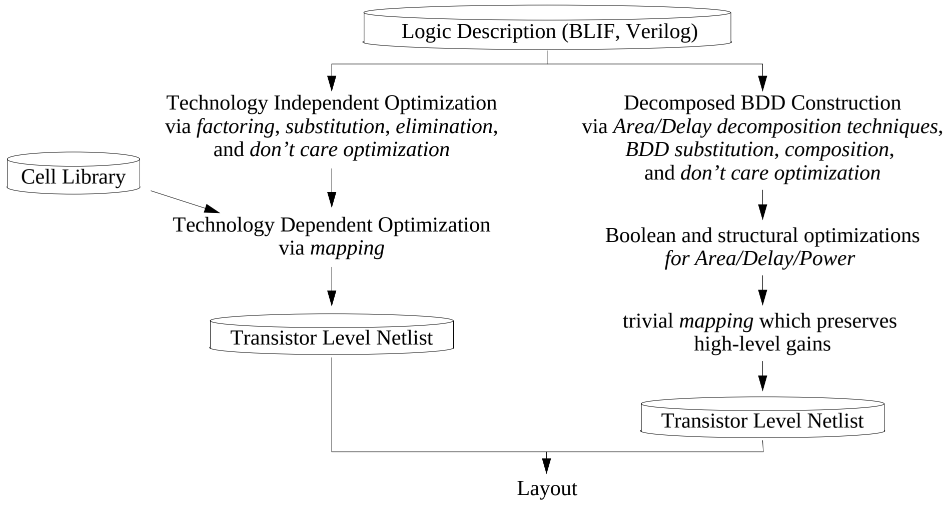

Logic Description (BLIF, Verilog)
Technology Independent Optimization via *factoring*, *substitution*, *elimination*, and *don't care optimization*
Decomposed BDD Construction via *Area/Delay decomposition techniques*, *BDD substitution*, *composition*, and *don't care optimization*
Cell Library
Technology Dependent Optimization via *mapping*
Boolean and structural optimizations *for Area/Delay/Power*
trivial *mapping* which preserves high-level gains
Transistor Level Netlist
Transistor Level Netlist
Layout

Figure 7: The traditional static CMOS synthesis flow vs. the proposed decomposed BDD synthesis flow

optimizations are becoming increasingly irrelevant with respect to the final performance of the transistor-level design because the technology mapping does not preserve the structure. This is particularly important in the context of deep sub-micron designs, where logic level optimizations need to be driven by physical issues which depend on the circuit structure and topology.

The *factoring* operation of the conventional flow aims at extracting common sub-expressions out of a function description. This is similar to selecting good decomposition points in the proposed flow. *Substitution* is similar to using a decomposition point as a BDD variable in the construction of the BDDs of subsequent decomposition points and the target function. *Elimination* is similar to composition operation on decomposition point BDDs, where a decomposition point BDD is composed into the BDDs of the rest of the circuit and the BDD node variable corresponding to the decomposition point eliminated if there is an overall saving in BDD nodes. Design optimization using don't cares can be employed in the proposed flow in a fashion very similar to the conventional flow. This is discussed in more detail in Section 7.1.

Apart from above operations which are analogous to optimization steps in the conventional synthesis flow, the decomposed BDD-based approach allows us to optimize circuits in several ways which have no equivalent in the conventional multi-level network based synthesis flow. These are outlined in Section 5.

5 Decomposition Techniques for BDD-based PTL Networks

5.1 Area Minimization

Since each node of a BDD corresponds to a PTL multiplexer cell, minimizing the area of the final circuit implementation is the same as minimizing the size of the decomposed BDD representation.

We employ a simple, greedy heuristic to control the size of the decomposed BDD by monitoring the BDD size while it is constructed. This is similar to [8]. When building the BDD depth-first from inputs to outputs, a decomposition point is introduced whenever the BDD size increases by a disproportionate amount. This attempts to avoid difficult BDD manipulations. A decomposition point is also introduced when an individual BDD grows beyond a threshold value. This ensures that none of the individual BDDs in the decomposed representation exceeds the threshold. This is particularly important in the PTL context since both resistance and capacitance increase linearly with the number of transistor in series. Thus, a very deep BDD can result in a slow circuit.

Due to the local, greedy nature of our heuristic, it is possible that the introduction of a decomposition point prevents Boolean simplification in the target function BDD. To discover some of these simplifications the decomposition points are composed back into the target function BDD as long as the overall BDD size reduces. An example of BDD size reduction by composition due to Boolean simplification is shown in Fig. 8. Since the amount of reduction is dependent on the order of composition, we experiment with several different orderings to determine a good choice.

Complementary edges can be used to reduce the size of the BDDs even further. A complementary edge introduces an inverter in the circuit, saves at least one BDD node and in the best case reduces the BDD size by half [1]. Thus, the net transistor count can only decrease. Also, these inverters provide the added benefit of restoring the signal to the rail values, thus offsetting any signal degradation due to its passage through a long pass transistor chain. Additionally, the output of decomposition points are buffered if they are connected to MOS gates of a subsequent stage.

Further, when synthesizing PTL networks from a decomposed BDD, a global variable ordering for all BDDs is not required. This provides an additional flexibility for reducing the size of each BDD by reordering them independently.

5.2 Performance

In a monolithic BDD implementation, the critical path cannot be longer than the number of input variables *n* and can be as low as log *n*. Decomposition introduces extra control variables whose critical paths can be in series with the critical path of the primary outputs' BDD. Note that the critical path length of the decomposition point BDD is bounded by the number of its variables, which can be more then *n* if the decomposition point is expressed

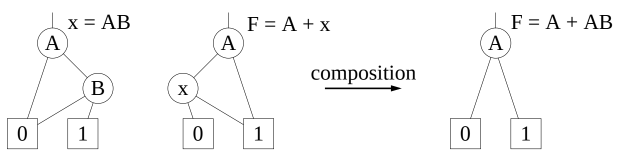


Figure 8: BDD size reduction via composition

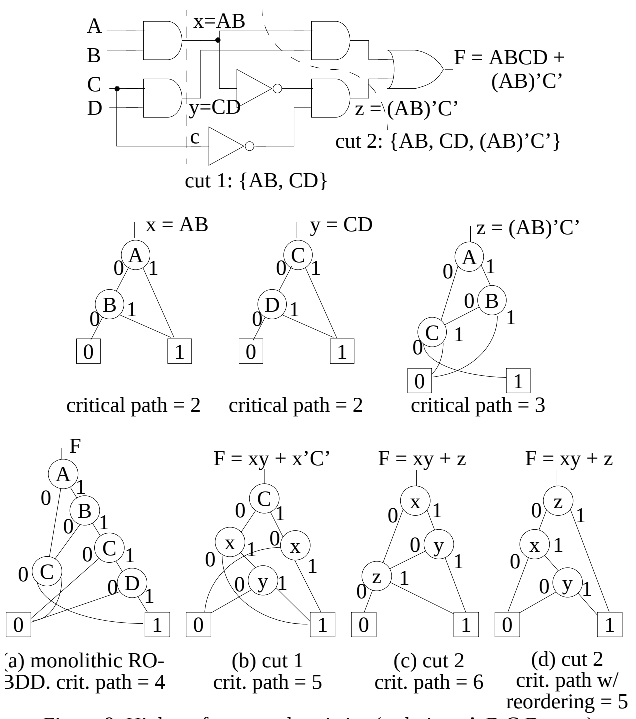


Figure 9: High performance heuristics (ordering: A,B,C,D,x,y,z)

in terms of other decomposition points. The critical path of the decomposed BDD is then bounded by *max* {critical paths of decomposition point BDDs, length of the longest path in the primary output BDDs}. Thus, the critical path length of the decomposed BDD is bounded by the critical path of the corresponding monolithic BDD. However, when circuit level issues are considered, the quadratic dependence of delay on the transistor chain length more than offsets the advantages of a shorter critical path. For today's static CMOS it is known that transistor chains longer than 3-4 transistors in series can be unacceptably slow [12]. A monolithic BDD-based circuit would require buffer insertion as in [22] for all but the smallest circuits. In comparison, a decomposed BDD-based circuit where outputs of decomposition points are buffered allows us to exploit area gains (and the associated reduction in switching capacitance) while controlling the length of unbuffered chains. Selecting decomposition points with appropriate thresholds on the depth of decomposition point BDDs is thus a more powerful strategy than selecting buffer insertion points in a monolithic BDD.

Apart from controlling the depth of decomposition point BDDs, the choice of decomposition points can be targeted at delay minimization when speed is the main concern. If a cutset[1] of the circuit is selected as the set of decomposition points, then the critical path in the BDDs of the primary outputs is bounded by the cutset cardinality (because BDDs of the primary outputs can be constructed in terms of the cutset variables only). Using the minimum cardinality cutset of the circuit as the decomposition set is then a good heuristic to reduce the critical path length. An example to illustrate this heuristic is shown in Fig. 9. Fig. 9(b) and 9(c) compare the critical path length when two different cutsets are chosen as decomposition points. The critical path is lower when the mincut is selected as the decomposition set (Fig. 9(b)) but longer than the monolithic ROBDD (Fig. 9(a)).

1. set of nodes such that all paths from primary inputs to primary outputs pass through some node in the set

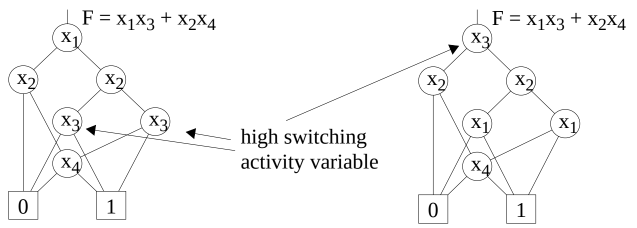


Figure 10: Reducing occurrences of high switching activity node

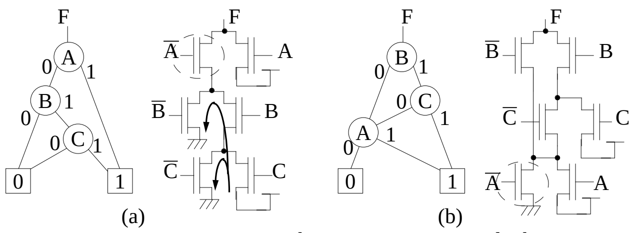


Figure 11: Low Power heuristic to minimize glitching: F = A + BC, $p(A=1) \approx 1 \Rightarrow p(F=1) \approx 1$

BDD variable ordering has a great impact on the BDD size and consequently the circuit area. This ordering can also influence the circuit power and speed. Placing late arriving signals closer to the outputs can speed-up the circuit by minimizing the number of transistors that need to be charged after the late signal arrives. Signal flow in a BDD-based PTL network corresponds to traversing the BDD from leaf nodes up. Thus it is advantageous to place late arriving control variables close to the top of the BDD. Variables in a BDD can be swapped pairwise as long as the resulting variable order does not cause the BDD size to increase significantly. In the example of Fig. 9, placing the late arriving signal z at the top (Fig. 9(d)) reduces the critical path by 1 unit over Fig. 9(c). The best known dynamic reordering algorithms for BDD size ([14][11]) move each variable or a block of variables throughout the order to find an optimal position for the variable. A similar reordering can be performed for delay, where the optimal position is the one resulting in the smallest depth BDD instead of the smallest sized BDD. As in Section 5.1, each decomposition point BDD can be reordered independently to optimize for total delay.

5.3 Low Power

Power dissipation in a circuit is a function of switching capacitance and switching activity. It is thus desirable that the capacitance connected to nodes with high switching activity is minimized. Since the gate capacitance of a transistor is substantially higher than the drain/source capacitances, this translates into ensuring that the high switching activity nodes are not connected to the gates of too many transistors. Note that neglecting drain-source capacitance switching is analogous to ignoring the internal node switching in a static CMOS gate.

In the case of PTL networks, only control variables are connected to the gate terminals of transistors. In our decomposed BDD-based approach, the control variables consist of primary inputs and decomposition points. Note that every node in the BDD is implemented as a multiplexer in the corresponding circuit and the node variable in the BDD is connected to the gates of two transistors of the multiplexer. Minimizing the occurrences of high switching activity node then translates into minimizing the occurrences of the corresponding variable in the BDD. Re-ordering BDD variables can be used to achieve this. Fig. 10 illustrates a case where re-ordering reduces the occurrences of the high switching activity variable at the expense of more occurrences of a lower switching activity variable.

A node in the PTL network is charged high when there is a path connecting it to the power supply and discharged when there is a path connecting it to ground. Even when the output does not change, glitching (charging and discharging of internal nodes) can consume a significant amount of power. Glitching can be minimized by placing variables which have a low switching probability close to the bottom of the BDD. This implies that the transistors controlled by these variables are close to the power supply and ground in the PTL network. Depending upon their state, this will cut-off the rest of the PTL network from the power supply or ground, resulting in a lower switching power dissipation. In the example in Fig. 11, since the probability of A being high is almost 1, the nMOS connected to $\overline{A}$ is almost always cut-off, and F is almost always 1. The ordering in Fig. 11(a) can however result in a significant power dissipation due to the internal nodes being charged through the nMOS connected to C and discharged through the nMOS connected to $\overline{B}$ and $\overline{C}$. Compared to this, the ordering in Fig. 11(b) has no internal power dissipation as the nMOS connected to $\overline{A}$ cuts-off the rest of the circuit from ground.

Note that these heuristics are similar to re-ordering transistors for low power at the circuit level in static CMOS ([6]). However, in our approach, technology mapping is straightforward and there is a one-to-one correspondence between BDDs and PTL circuits. This allows us to perform Boolean manipulations at a high level in which we can trade-off circuit area for power, rather then making restricted structural changes at the circuit level.

6 Results

The techniques described in this paper have been tested on ISCAS benchmarks circuits, which include circuits which are *hard* for monolithic BDD-based approaches (e.g. C6288). In the following we present results comparing our PTL synthesis algorithm with different static CMOS synthesis algorithms to demonstrate the area and delay gains achieved by our approach, and HSPICE simulations to verify the validity of the logic level gains.

The PTL synthesis algorithm was implemented in the SIS framework. It is compared against four synthesis scripts for static CMOS: area and delay optimization scripts which do not use don't cares, and *script.rugged* and *script.delay* of SIS. Technology mapping was performed using three different libraries: msu.genlib, 33-4.genlib, and 44-3.genlib. All experiments were carried out on a 400 MHz DEC Alpha with a SPECint_92 rating of 341, DEC 21164 CPU, 4Mb cache and 2Gb total memory.

PTL circuits were synthesized with four different threshold parameters. This threshold parameter from Section 5.1 controls the depth of decomposition point BDDs. For a given logic circuit, the best of the four PTL transistor-level circuits was selected and data for this circuit is presented in all tables. Compared to this, static CMOS circuit in each table is the best of several test runs with different parameters. Moreover, not the same circuit is used in all tables. That is, the same PTL circuit data is compared with the area optimized static CMOS circuit in the area columns of the tables and against the delay optimized static CMOS circuit in the delay columns. The gains achieved by PTL are thus very conservative, since the area-optimal static CMOS circuit is far from delay-optimal and vice versa.

Table 1 compares the PTL results against results from area and delay optimization scripts for static CMOS which do not use don't cares (mapped for area and delay respectively using msu.genlib). Since our PTL implementation does not perform don't care optimization yet, this table gives the best picture of the efficiency of our PTL algorithm. Column 1 contains the names of the ISCAS benchmark circuits. Column 2 and 3 compare the active gate area (measured in λ^2) of circuits synthesized by the PTL and minimum area static CMOS algorithm. Column 4 contains the relative gain in area achieved by our PTL algorithm over the static CMOS algorithm. Columns 5 and 6 compare the critical path length of the circuits generated by PTL and minimum delay static CMOS. Column 7 contains the relative gains of PTL over the static CMOS algorithm.

Table 2 presents results in the same format, this time comparing the PTL data of Table 1 with static CMOS results optimized using local optimizations and full don't cares. The static CMOS results in Column 2 are optimized using *script.rugged* and the results in Column 5 are optimized using *script.delay*. Note that the current PTL implementation does not perform local optimizations or don't care optimizations. This is not an algorithmic limitation and will be implemented in the near future (techniques for this have already been outlined in Section 7). In spite of this handicap, the current PTL implementation yields impressive gains over the *script.rugged* and *script.delay*.

Table 3 compares the runtimes of the PTL synthesis algorithm (Column 6) with the area and delay optimization scripts without don't cares, and *script.rugged* and *script.delay* (Columns 2, 3, 4 and 5 respectively). Note that the output of static CMOS algorithms is a mapped logic network while the output of the PTL algorithm is an HSPICE netlist. The results clearly indicate that PTL synthesis is substantially faster than all static CMOS techniques.

When using the critical path length as the metric to compare delays of two circuits, it is important to ensure that the amount of logic implemented in a cell is similar for each case, because a circuit with large individual cells can have a small critical path but be slow due to high cell propagation delay. Table 4 compares the cell count of the PTL circuit with the area and delay optimized static CMOS circuits of Table 1 in Columns 2, 3 and 4 respectively, and the average cell size in Columns 5, 6, and 7 respectively. The data indicates that the PTL circuit indeed uses fewer cells with more logic in each individual cell. Thus, while the delay of each cell is not directly related to the cell size (since each cell in the PTL circuit implements a BDD structure, while the static CMOS cells implement a series-parallel pull-up and pull-down tree structure), we do need further analysis to ensure that the savings in the critical path length translate to delay reduction in the final transistor-level circuit. We analyze a full adder circuit and perform HSPICE analysis to examine the delay trade-off between a larger cell implementing a BDD structure vs. a smaller series-parallel logic cell. These results are presented in Fig. 12.

As an aside, we also synthesized static CMOS using larger libraries like 33-4.genlib (87 cells, average cell size: 27.4 λ^2) and 44-4.genlib (625 cells, average cell size 43.4 λ^2) to see if a greater choice of cells, including very large cells, improved the static CMOS results. However, we found that there was no major change in the results, and in fact, the synthesis runtimes for static CMOS increased by factors of 5-100 due to the library size.

Table 5 presents the logic synthesis and HSPICE analysis results for a full adder circuit implemented in PTL and static CMOS. Columns 2, 3, 4, and 5 provide logic level data (area, number of transistors, number of cells and average cell size respectively) for the two test cases. Columns 6 and 7 contain the slowest rise and fall times from an exhaustive HSPICE simulation. Fig. 12(a) and 12(b) show the rise and fall time waveform plot. The results indicate that PTL has a smaller fall time and the same rise time as static CMOS. This is to be expected since an nMOS-only PTL circuit is good at conducting “0”. The critical path thus seems to be a good indicator of the fall times. In general, a lower area implies a smaller switching capacitance, which does indeed correlate with a faster circuits. Thus, the 20-50+% gains achieved at logic level should translate to gains at the transistor level, albeit in slightly smaller numbers. Columns 8 and 9 present the average and rms power dissipation results. The static CMOS circuit has a lower average power dissipation but a higher rms power dissipation. Fig. 12(c) indicates that the static CMOS has a higher peak power dissipation as well. While the lower average power dissipation of static CMOS is good from the battery life perspective, a higher peak and rms power dissipation is undesirable from the electromigration and IR drop point of view.

Circuit	Area			Delay		
	static CMOS	PTL	gain	static CMOS	PTL	gain
C17	54.0	58.5	-8 %	3	2	33 %
C432	1620.0	1468.5	9 %	18	23	-28 %
C499	3424.5	2920.5	15 %	12	9	25 %
C880	2673.0	2433.0	9 %	16	9	44 %
C1355	3424.5	2953.5	14 %	14	6	57 %
C1908	4851.0	3174.0	35 %	20	14	30 %
C2670	5787.0	4797.0	17 %	18	11	39 %
C3540	9279.0	7495.5	19 %	28	18	36 %
C5315	13266.0	12415.5	6 %	24	11	54 %
C6288	21321.0	16180.5	24 %	120	70	42 %
C7552	18310.5	19902.0	-9 %	21	14	33 %

Table 1 : Best area & best delay static CMOS vs. PTL

Circuit	Area			Delay		
	script. rugged	PTL (no DC)	gain	*script. delay*	PTL (no DC)	gain
C17	49.5	58.5	-16 %	3	2	33 %
C432	1233.0	1468.5	-19 %	18	23	-28 %
C499	3244.5	2920.5	10 %	11	9	18 %
C880	2596.5	2433.0	6 %	16	9	44 %
C1355	3244.5	2953.5	9 %	11	6	45 %
C1908	3226.5	3174.0	2 %	18	14	22 %
C2670	4491.0	4797.0	-9 %	-	11	-
C3540	8176.5	7495.5	8 %	26	18	31 %
C5315	9985.5	12415.5	-24 %	23	11	52 %
C6288	19885.5	16180.5	19 %	61	70	-15 %
C7552	-	19902.0	-	-	14	-

Table 2 : Best area & best delay static CMOS using DC vs. PTL (which does not use DC).

Circuit	static CMOS				PTL
	area	delay	*script.rugged*	*script.delay*	
C17	0.01	0.01	0.10	0.10	0.01
C432	0.6	0.6	113.2	91.4	0.2
C499	1.1	1.0	12.9	10.5	0.6
C880	0.9	0.9	4.3	11.0	0.4
C1355	1.4	1.4	13.3	24.4	0.6
C1908	1.7	1.7	15.9	47.7	1.1
C2670	3.0	2.6	100.4	-	1.8
C3540	4.3	3.5	30.0	371.4	2.8
C5315	7.2	5.9	22.9	664.9	5.1
C6288	5.5	6.1	65.0	287.2	10.4
C7552	10.1	8.6	-	-	14.3

Table 3 : Static CMOS vs. PTL: runtime (in seconds)

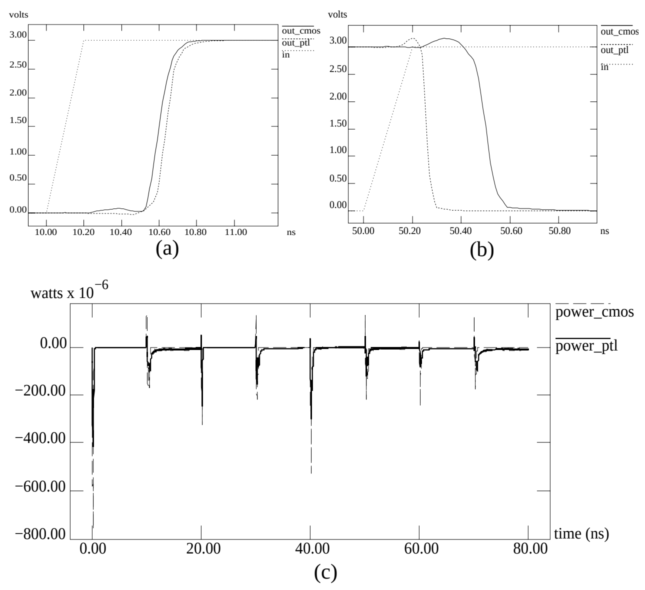


Figure 12: HSPICE results on timing and power dissipation of static CMOS and PTL circuits for a full adder

Circuit	Cell Count			Average Cell Size		
	static CMOS		PTL	static CMOS		PTL
	area opt	delay opt		area opt	delay opt	
C17	6	6	3	6.0	6.0	13.0
C432	141	144	58	7.7	7.7	16.9
C499	238	243	133	9.6	9.8	14.6
C880	223	215	94	8.0	8.8	17.2
C1355	238	235	41	9.6	10.8	48.0
C1908	356	345	189	9.1	10.1	11.2
C2670	425	514	227	9.1	8.8	14.1
C3540	778	789	382	8.0	8.5	13.1
C5315	1030	1238	316	8.6	8.7	26.2
C6288	2326	2340	929	6.1	6.1	11.6
C7552	1596	1512	989	7.6	8.3	13.4

Table 4 : Static CMOS vs. PTL: cell counts and average cell size

Logic	Area (λ^2)	# MOS	# Cell	Avg Cell Size (λ^2)	Delay			Power	
					Crit Path	HSPICE		Avg.	RMS
						t_{rise}	t_{fall}		
PTL	42	22	1	42.0	1	0.6ns	0.2ns	30µW	83µW
CMOS	81	36	8	10.1	4	0.6ns	0.5ns	17µW	122µW

Table 5 : Simulation data for a full adder circuit

7 Enhancing the Decomposed BDD-based Approach

7.1 Don’t Care Optimization

The PTL synthesis tool benchmarked in Section 6 does not use don’t cares for design optimization. Don’t cares provide a significant amount of flexibility in minimizing a circuit as witnessed from the improvement of the static CMOS area and delay results between Table 1 and Table 2 in Section 6.

Extending the proposed approach to handle don’t cares is relatively straightforward. Several heuristics to minimize BDD size using don’t cares are presented in [20]. Since the area of a decomposed BDD-based PTL circuit is proportional to the BDD size, the results of [20] can be applied to PTL synthesis directly. The synthesis algorithm would then be modified as follows: after generating the decomposed BDD representation, we minimize the target function BDD and each decomposition point BDD travelling from the primary outputs of the circuit to primary inputs. In the context of the multi-stage circuit represented by the decomposed BDDs, travelling from the outputs to the inputs amounts to first minimizing the target function BDD and then each decomposed BDD in the reverse order of decomposition point introduction. For each BDD, we compute the compatible observability don't cares for the output function in terms of the primary inputs. This is mapped to a local don't care set via image computation. The don't care set construction is the same as in the case of the multi-level network minimization and we refer the reader to [18] for more details. The heuristics of [20] are then applied to minimize the BDD. Based on the results reported in [20], we expect this extension to yield significant reduction in the area of the PTL circuits.

7.2 Synthesis of Mixed static CMOS/PTL Circuits

This work has proposed the use of PTL for large deep sub-micron designs. PTL can provide substantial gains in area and delay over static CMOS, while the static CMOS has the advantage of a well-established design flow for synthesizing robust circuits. Static CMOS may be preferable over PTL in cases where a static CMOS implementation of a gate is particularly efficient, or where an nMOS conducting "1" is not allowed.

The PTL synthesis flow proposed in this work is very general in nature and allows synthesis of mixed static CMOS/PTL circuits which can leverage the strengths of static CMOS as well as PTL as appropriate. Each decomposition point BDD can be viewed as a complex node and can be implemented by static CMOS logic or PTL as desired.

Among other issues, currently we use ROBDDs as the underlying data structure for the decomposed BDDs. General BDDs ([2]), which allow input variables to appear multiple times along any path in the BDD, may be more appropriate from the PTL network design point of view since we are more interested in compactness than in canonicity. We plan to look into incorporating general BDDs in our synthesis algorithm.

8 Conclusions

PTL can be a promising alternative to static CMOS for deep sub-micron design. In this work, we have motivated the need for CAD algorithms for PTL circuit design and have proposed a methodology for synthesizing PTL circuits. The main contributions of this work are the following:

- A decomposed BDD-based representation is proposed to take advantage of the correspondence between PTL circuits and BDDs without suffering from the drawbacks imposed by properties of monolithic BDDs.
- A comprehensive synthesis flow is outlined for PTL design. We showed that the proposed approach allows us to make logic level optimizations similar to the traditional multi-level network based synthesis flow for static CMOS, and also makes possible optimizations with a direct impact on area, delay and power of the final circuit implementation which do not have any equivalent in the traditional approach. Using these techniques we were able to synthesize PTL circuits for the entire ISCAS benchmark set.
- A set of heuristical algorithms to synthesize PTL circuits optimized for area, delay and power which are key to the proposed synthesis flow, are presented. These algorithms are very intuitive and simple and have a great impact on the optimality of the resulting circuit.

Initial experimental results on ISCAS benchmark circuits show that our technique yields PTL circuits with substantial improvements in area and delay over conventional static CMOS designs. We believe that with more research in this area PTL can become a viable alternative to static CMOS, and that this work is the first step in that direction.

9 Acknowledgments

The authors would like to thank Prof. Takayasu Sakurai and Ravi Gunturi for useful discussions on this work.

References

[1] S. Akers, "Binary decision diagrams," *IEEE Trans. on Computers*, vol. C-27, no. 6, June 1978.

[2] P. Ashar. A. Ghosh, S. Devadas, "Boolean satisfiability and equivalence checking using general binary decision diagrams," *ICCD* 1991.

[3] W. Al-Assadi, A.P. Jayasumana, and Y.K. Malaiya, "Pass-transistor logic design," *Int'l J. Electronics*, vol. 70, no. 4, 1991.

[4] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Computers*, vol. C-35, no. 8, Aug.1986.

[5] A. P. Chandrakasan, S. Sheng, and R.W. Brodersen, "Low Power CMOS Digital Design," *IEEE JSSC*, vol. SC-20, 1985.

[6] R. Hossain, M. Zheng, and A. Albicki, "Reducing power dissipation in CMOS circuits by signal probability based transistor reordering," *IEEE Trans. CAD*, vol.15, no. 3, March 1996.

[7] A. Jaekel, G.A. Jullien, and S. Bandyopadhyay, "A multilevel factorization technique for pass transistor logic," *9th Int'l Conf. on VLSI Design*, Jan. 1996.

[8] J. Jain, A. Narayan, C. Coelho, S.P. Khatri, A. Sangiovanni-Vincentelli, R.K. Brayton, and M. Fujita, "Decomposition techniques for efficient ROBDD construction,", *Int'l Conf. in FM-CAD*, 1996.

[9] P.C. McGeer, K.L. McMillan, A. Saldanha, A. Sangiovanni-Vincentelli, and P. Scaglia, "Fast discrete function evaluation using decision diagrams," *ICCAD*, Nov. 1995.

[10] J.L. Neves, and A. Albicki, "A pass transistor regular structure for implementing multi-level combinational circuits," *7th Int'l ASIC Conf. and Exhibit*, 1994.

[11] S. Panda, and F. Somenzi, "Who are the variables in your neighborhood," *ICCAD*, Nov. 1995.

[12] J. Rabaey, *Digital Integrated Circuits*, Prentice-Hall, Inc. 1996.

[13] D. Radhakrishnan, S.R. Whitaker, and G.K. Maki, "Formal design procedures for pass transistor switching circuits," *IEEE JSSC*, vol. SC-20, no. 2, April 1985.

[14] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *ICCAD*, Nov. 1993.

[15] T. Sakurai, B. Lin, and A.R. Newton, "Multiple-Output Shared Transistor Logic (MOSTL) Family Synthesized Using Binary Decision Diagram," *Tech. Report, Univ. of California, Berkeley*, M90/21, 1990.

[16] F. Salice, "Automatic synthesis of logic functions using transmission gates," *J. Microelectronic Systems Integration*, vol. 3, no.1, 1995.

[17] Y. Sasaki, K. Yano, S. Yamashita, H. Chikata, K. Rikino, K. Uchiyama, and K. Seki, "Multi-level pass-transistor logic for low-power ULSIs," *Int'l Symp. on Low Power Electronics*, Oct. 1995.

[18] H. Savoj, "Don't cares in multi-level network optimization," *Ph.D. Dissertation, Univ. of California, Berkeley*, M92/122, Oct. 1992.

[19] M. Shamanna, K. Cameron, S.R. Whitaker, "Multiple-input, multiple-output pass transistor logic," *Int'l J. Elect.*, vol.79, no.1, 1995.

[20] T. Shiple, R. Hojati, A. Sangiovanni-Vincentelli, R.K. Brayton, "Heuristic minimization of BDDs using don't cares," *DAC* 1994.

[21] K. Yano, T. Yamanaka, T. Nishida, M. Saito, K. Shimohigashi, and A. Shimizu, "A 3.8-ns CMOS 16x16-b multiplier using complementary pass-transistor logic," *IEEE JSSC*, vol. 25, no. 2, April 1990.

[22] K. Yano, Y. Sasaki, K. Rikino, and K. Seki, "Top-down pass-transistor logic design," *IEEE JSSC*, vol. 31, no. 6, June 1996.