Recursive Bipartitioning of BDDs for Performance Driven Synthesis of Pass Transistor Logic Circuits*

Rupesh S. Shelar
Department of Elec. & Comp. Engg.
University of Minnesota
Minneapolis, MN 55455.
rupesh@ece.umn.edu

Sachin S. Sapatnekar
Department of Elec. & Comp. Engg.
University of Minnesota
Minneapolis, MN 55455.
sachin@ece.umn.edu

ABSTRACT

In this paper, we address the problem of performance oriented synthesis of pass transistor logic (PTL) circuits using a binary decision diagram (BDD) decomposition technique. We transform the BDD decomposition problem into a recursive bipartitioning problem and solve the latter using a max-flow min-cut technique. We use the area and delay cost of the PTL implementation of the logic function to guide the bipartitioning scheme. Using recursive bipartitioning and a one-hot multiplexer circuit, we show that our PTL implementation has logarithmic delay in the number of inputs, under certain assumptions. The experimental results on benchmark circuits are promising, since they show the significant delay reductions with small or no area overheads as compared to previous approaches.

1. INTRODUCTION

With the advent of the system-on-chip era and therefore, with the motivation of packing more logic functionality on the chip, other logic styles such as domino and pass transistor logic (PTL) are being explored as alternatives to static CMOS. The primary benefits of PTL include the potential for a lower transistor count, lower capacitance, smaller delays and reduced power consumption [1, 2]. Synthesis techniques for PTL circuits have been closely related to the binary decision diagram (BDD) representation of logic functions, for reasons such as elimination of sneak paths and the availability of efficient algorithms for the construction of BDD's. Buch *et al.* propose a greedy heuristic to decompose the BDD's into smaller BDD's whose sizes are kept under a specified threshold [3]. For area-driven PTL synthesis, Chaudhary *et al.* propose an approach [4] similar to traditional multi-level logic optimizations involving iterative application of transformations after which the BDD representation is mapped on to a PTL cell library. A similar philosophy has been used for performance driven synthesis. Both [3] and [5] imply that multi-level BDD's are to be used, but the limitation of these approaches is that they are unable to predict the performance gain beforehand. Other relevant work on BDD optimization includes [6], 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 is not specifically targeted. Becker *et al.* reported the use of multiplexer circuits for area and delay optimizations of PTL circuits [7]. Unlike [3], they allowed varied threshold size of BDD and their cost function allows area and depth to be traded off.

In this paper, we present a novel approach to performing PTL synthesis through a decomposition of a monolithic BDD representing a circuit. We employ a bipartitioning scheme that uses max-flow min-cut technique to halve the depth of a PTL implementation of a BDD with the least area overhead. We first illustrate how the BDD can be partitioned into smaller pieces and implemented as a PTL circuit using multiplexers. We apply bipartitioning recursively and with the use of a one-hot multiplexer circuit, we show that it results in the implementation with logarithmic depth in number of inputs. Unlike many previous techniques for PTL circuit synthesis, we predict the delays in the circuit beforehand through a theoretical analysis of the circuit delay that motivates our partitioning algorithm.

2. PTL IMPLEMENTATION

2.1 Delays in PTL Implementation

Pass transistor logic can be used to build a 2-input multiplexer; there is a one-to-one correspondence between BDD's and their PTL implementations. Each node of a BDD implements a Shannon expansion, $F = x \cdot F_x + x' \cdot F_{x'}$ about the variable x associated with the node, 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 be applied recursively to the functions F_x and $F_{x'}$ to arrive at PTL implementation of a given BDD representation. For the purposes of this paper, all BDD's are reduced ordered BDD's (ROBDD's), implying that the order of variables on any path from an output node to a leaf node is identical.

Consider a PTL implementation of a function that depends on N inputs obtained by direct mapping of a BDD on those N inputs. We estimate the delay along the crit-

*This work was supported in part by the SRC under award 99-TJ-692.

ical path, $Delay_M$, as the delay along a path containing $N-1$ pass transistors in series assuming that buffers are also added when k pass transistors are in series to regenerate the signal; we will assume $k = 3$ here, but a similar analysis can be carried out for any other value of k.

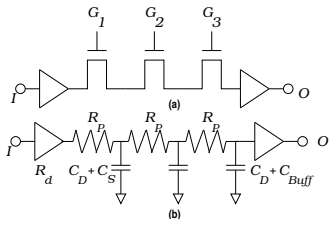

Figure 1: (a) Three pass transistors in series (b) The equivalent RC model

To estimate $Delay_M$, we build an equivalent RC model for the pass transistor circuit, shown in Figure 1. Assuming that the resistance of a pass transistor is R_p, and the resistance of the driver node (buffer) is R_d, we can calculate the Elmore delay of this structure as $R_d \cdot (3C_D + 3C_S + C_{Buff}) + R_p \cdot (6C_D + 3C_S + 3C_{Buff})$, where $C_{D(S)}$ is the drain (source) capacitance and C_{Buff} is the input capacitance of buffer. In case of a chain of $N-1$ pass transistors with one buffer placed at every third pass transistor, there are $\lfloor (N-1)/3 \rfloor$ three-transistor segments of the type shown in Figure 1. Therefore, the worst-case delay, is given by

$$Delay_M = \lfloor (N-1)/3 \rfloor (R_p(6C_D + 3C_S + 3C_{Buff}) + R_d(3C_D + 3C_S + C_{Buff})) = O(N-1) \quad (1)$$

2.2 Decomposition of the BDD

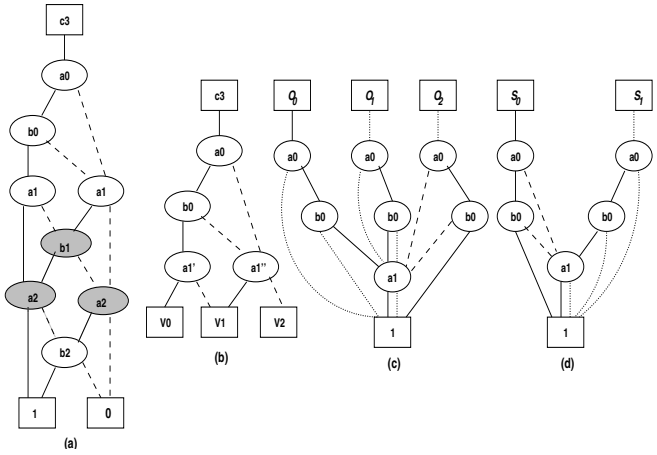

Figure 2: (a) Carry Function for 3-bit Adder, (b) Introducing dummy nodes V0, V1, V2 (c) Select function for one-hot encoding, (d) Select function for minimum-bit encoding.

In this subsection, we outline a general BDD decomposition technique for delay reduction using the following example. Consider a carry output function for a 3-bit adder whose optimized BDD, on 6 input variables a0, b0, a1, b1, a2 and b2, is shown in Figure 2. We take a cut across the BDD as shown by the shaded nodes in Figure 2(a) and introduce dummy nodes V0, V1, V2 to replace these shaded nodes, as shown in Figure 2(b) These dummy nodes can be assigned unique codes using one-hot or minimum-bit encoding as shown in Table 1.

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 1: Encodings for dummy terminal nodes

After encoding, the next step in decomposition is to construct the BDD's corresponding to the *select* and *data* inputs of a multiplexer. 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_0 corresponds to the combination V0 = 1, V1 = V2 = 0. By substituting these values into the dummy terminals in Figure 2(b), we can obtain the BDD for the *select* input O_0 as shown in Figure 2(c). The BDD's for other *select* inputs such as O_1 and O_2 can be obtained similarly. Figure 2(d) shows the BDD's for *select* inputs S_0 and S_1, respectively. We observe that depth of the BDD's for the *select* inputs is the same for one-hot encoding as well as minimum-bit encoding. Note that in case of *select* functions obtained by one-hot encoding, for any assignment of a0, b0, a1, only one of the *select* functions is true and we can use a one-hot multiplexer circuit to implement c3. The implementation of c3 using a one-hot multiplexer and a regular multiplexer is shown in Figure 3. The *select* inputs are simply the PTL implementations of the BDD's shown in Figures 2(c) and 2(d). Table 2 shows the active area and delay, obtained by HSPICE simulations for 100ps transition time exciting input, for implementations obtained by direct mapping of BDD, one-hot multiplexer based and regular multiplexer based implementation. Clearly, one-hot multiplexer based implementation has the least delay with slightly larger area than the one obtained by direct mapping of BDD. We also observe that in the decomposed implementation using the one-hot multiplexer, the depth of the circuit is halved as compared to the implementation obtained by a direct mapping of the BDD.

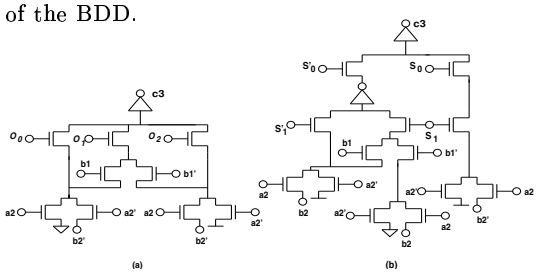

Figure 3: Various implementation of c3

Implementation	Active Area(μ^2)	Delay(ps)
Monolithic BDD	3.25	708.4
One-hot Multiplexer	4.125	409.4
Regular Multiplexer	4.875	458.5

Table 2: Comparison of various implementations

3. BIPARTITIONING

The decomposition technique presented in Section 2 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 halving the critical path, then using equation (1), we find that the delay 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 halved. The price being paid for delay

reduction is in terms of area, since the number of transistors increases as we recursively bipartition the BDD. We can perform this bipartitioning for the minimum area penalty.

3.1 The Algorithm for Bipartitioning

Our aim is to find an optimum cut that halves the critical path, measured in terms of Elmore delay and has the least area penalty. We estimate the Elmore delay assuming PTL implementation of a given BDD with buffers after every three pass transistors in series and inverters for complemented edges. We assign two delays to each node.

Delay from Bottom(D_{bottom}) : This is the Elmore delay of the PTL network rooted at a given node.

Delay from Top(D_{top}) : This is the maximum Elmore delay from node to any of the outputs.

These delays can be found out using a PERT-like traversal on the digraph. Clearly, the critical path is a path on which the node with maximum Elmore delay lies. We define two types of nodes for delay balanced bipartitioning:

Essential Nodes, for which D_{bottom} equals half of the critical path delay. In other words, essential nodes lie in the middle of critical path.

Candidate Nodes, for which D_{top} and D_{Bottom} are both less than half of the critical path delay.

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. All essential nodes must be in the cut while we have a freedom to choose among the candidate nodes. We assign an area cost candidate nodes assuming PTL implementation of the circuit and then use the max-flow min-cut technique [8] to find the optimum cut that halves the circuit delay at the least area cost.

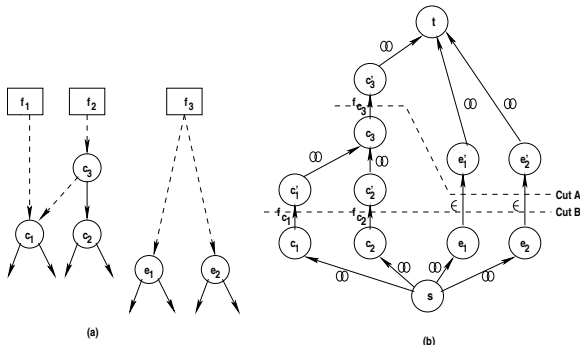

Figure 4: Creating a flow network

Figure 4 shows the creation of the flow network from a digraph corresponding to the given BDD. Figure 4(a) shows the digraph corresponding to a BDD, in which 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. Dashed edges in Figure 4(a) (for instance, an edge from f_1 to c_1) indicate that there are directed paths between the corresponding nodes. Figure 4(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 in the digraph is represented by two nodes e_1 and e_1' with an edge from e_1 to e_1' of a small capacity ϵ. Similarly, candidate nodes in the digraph are represented by splitting them into two nodes, for instance, node c_1 in the digraph is represented by two nodes c_1 and c_1', respectively, with an edge of capacity f_{c1} from c_1 to c_1'. Since we want essential nodes to be included in the optimum cut, we assign a small capacity to the edge between the split essential nodes, and since we want to choose the candidate nodes with the least area penalty, we assign a capacity proportional to the area cost of candidate nodes to the edges between split candidate nodes. The remaining edges in the flow network are assigned a capacity of ∞, and therefore, will not appear in the cut. Thus, there are two possible cuts, Cut A and Cut B, corresponding to cut-sets $A_{cutset} = \{e_1, e_2, c_3\}$ and $B_{cutset} = \{e_1, e_2, c_1, c_2\}$ in the digraph corresponding to the given BDD. Application of the Ford-Fulkerson technique to find the minimum cut will result in one of these, depending on values of f_{c_1}, f_{c_2} and f_{c_3}. The pseudocode for the algorithm is shown in Figure 5.

```
Input:  G(V,E) = Digraph corresponding to given
BDD, V = Nodes, E = Edges.
Output:  S_cut = Optimum cut-set.
Steps:
PERT_Traversal(G); /* Assign D_top, D_Bottom ∀v∈V */
D_crit = max{D_Bottom ∀v∈V}; /* Critical Path Delay */
V_Essential = {v:v∈V and D_bottom = D_crit/2 };
V_Candidate = {v:v∈V and D_top, D_bottom < D_crit/2 };
AreaCostEstimate(V_Candidate);
G_Flow= CreateFlowNetwork(G,V_Essential,V_Candidate);
Ford-Fulkerson(G_Flow, G, S_cut);/*Find optimum cut */
```

Figure 5: Pseudocode for Algorithm

Once the cut is determined, the vertices in the cut are replaced by dummy terminal nodes, which can be assigned unique codes, and implemented, as illustrated in subsection 2.2. The bipartitioning procedure can be applied recursively till no further delay reduction can be achieved and the resulting implementation has a delay which is logarithmic in terms of number of inputs, as stated by the following theorem, without proof due to space limitation.

THEOREM 3.1. *The recursive application of an algorithm in Figure 5 to any BDD on N input variables with the use of one-hot multiplexers results in an implementation which has the Elmore delay of $O(\log N)$, under the assumption of constant delay in the one-hot multiplexer circuit.*

Unlike the multiplexer based implementation for PTL circuits proposed by [7] that obtains a logarithmic depth for only *xor* functions, our use of one-hot multiplexers and recursive bipartitioning results in a logarithmic delay implementation for any circuit, irrespective of the cut-set size.

4. EXPERIMENTAL RESULTS

We have implemented the recursive bipartitioning algorithm and the decomposition procedure as a C++ program called PTLS (Pass Transistor Logic Synthesizer). PTLS uses the BDD package CUDD [12] for generating BDD's and we used sifting [13] for variable ordering. We used NMOS transistors of size $0.5\mu/0.25\mu$ as pass transistors. Inverters were inserted after at most three transistors in series, and in case of implementations using PTLS, to drive the gate nodes of the one-hot multiplexer. For the inverter, we chose $W_p/L_p = 1\mu/0.25\mu$ and $W_n/L_n = 0.5\mu/0.25\mu$. The delays were measured using static timing analysis of the resulting transistor netlist, We have used PTLS to synthesise MCNC and ISCAS'85 benchmark circuits and compared its results with other libraryless synthesis techniques such as TABA [10],

Example	PTLS			Ferrandi *et al.* [9]	Buch *et al.* [3]	TABA [10]	OTR [11]
Example	#. of Trans.	Delay(ps)	CPU Time(s)	# of Trans.	# of Trans.	# of Trans.	# of Trans.
C1355	1006	2010.92	0.88	1013	1969	1592	1304
C1908	1228	1775.53	1.86	1526	2116	2346	1858
C432	1088	612.76	1.25	727	979	710	644
C499	1072	2061.88	0.91	1013	1947	1464	1352
C2670	3045	640.17	21.4	2674	3194	2880	2842
C6288	9121	744.52	124.01	7073	10787	8096	7992
Total	16562			14026	20992	17088	15992

Table 4: Comparison of PTLS with Ferrandi's method, Buch's Method, TABA and OTR

Example	Monolithic		PTLS			TABA	OTR
	# of Trans.	Delay (ps)	# of Trans.	Delay (ps)	Recur. level	# of Trans.	# of Trans.
5xp1	284	691.49	256	234.25	2	302	378
9sym	110	1183.17	99	456.43	2	404	272
misex1	142	562.38	130	431.94	1	148	158
rd53	82	513.80	56	356.49	3	82	82
rd73	146	825.49	104	488.17	1	174	152
rd84	202	1000.59	153	473.83	2	290	252
sao2	326	1166.43	439	362.27	2	362	320
C17	32	338.70	29	226.19	1	-	-
alu2	1002	1816.96	1078	298.58	3	-	-
cm138	70	562.38	98	480.28	1	-	-
cm163	126	773.67	145	250.17	2	-	-
cmb	158	1267.97	124	654.09	2	-	-
comp	1070	4211.63	1640	476.36	8	-	-
parity	112	2285.84	92	674.55	3	-	-
t481	160	2240.37	277	162.15	4	-	-
z4ml	130	777.79	90	429.92	1	-	-
f51m	252	945.52	569	178.28	3	-	-
my_adder	1204	5417.5	1281	1243.93	5	-	-

Table 3: Monolithic BDD implementation vs. PTLS

OTR [11], and also with pass transistor logic synthesis techniques such as [3] and [9]. Table 3 shows the number of transistors required and delays for implementations using monolithic BDD's and PTLS and number of transistors required for implementations using TABA and OTR; delay figures for TABA and OTR are unavailable and '-' entries in Columns 6 and 7 shows that the corresponding information is unavilable. We observe that the implementations obtained by using PTLS has significantly less number of transistors than TABA and OTR in all cases except sao2. This is because BDD representations for all those examples are compact and recursive bipartitioning using one-hot encoding results in simplified expressions. We observe significant delay reductions in all cases with marginal increase in area for the comparison of PTLS implementations with corresponding monolithic BDD's implementations.

In Table 3, column 5 shows the maximum reucrsion level for the recursive bipartitioning scheme. We observe that for benchmarks like comp, t481, my_adder and parity when bipartitioning has been applied 8, 4, 5 and 3 times, respectively, delays are reduced drastically. On the other hand, for small benchmarks like C17, rd53 and misex1 delays did not reduce significantly. In these cases bipartitioning could be applied only once.

In case of ISCAS'85 benchmarks, we ran *script.rugged* for minimizing boolean network, and then, created multilvel BDD representation to apply recursive bipartitioning procedure on this multilevel BDD representation level-wise. Table 4 shows a comparison of the number of transistors for ISCAS'85 benchmarks with competing methods. Column 3 and Column 4 in Table 4 shows the delays obtained by static timing analysis and run time for PTLS on SUN Ultra-60 workstation, respectively. We could not compare the delays of our method with those reported by other methods because of the unavailibility of parameters for delay models. Since none of the above techniques except OTR [11] targets performance oriented synthesis, our method is quite likely to outperform other methods. The implementation cost of PTLS is found to be comparable to all competing approaches and is better in many cases. More importantly, the delay is likely to be much better since our approach uses decomposition for delay reduction, unlike other approaches. In case of PTLS, preprocessing the multilevel BDDs using algorithms such as *eliminate* reported in [4] could result in smaller BDD node count and therefore, less number of transistors in the implementation.

5. REFERENCES

[1] K. Yano *et al.* A 3.8ns CMOS 16 x 16 multiplier using complementary pass transistor logic. *IEEE Journal of Solid State Circuits*, 25(2):388–395, Apr. 1990.

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

[3] P. Buch *et al.* Logic synthesis for large pass transistor circuits. In *Proc. ICCAD*, pages 663–670, Nov. 1997.

[4] R. Chaudhary *et al.* Area oriented synthesis for pass transistor logic. In *Proc. ICCD*, pages 160–167, Oct. 1998.

[5] T. Liu *et al.* Performance driven synthesis for pass transistor logic. In *Proc. of the VLSI Design Conference*, pages 372–377, Jan. 1999.

[6] C. Yang and M. Ciesielski. BDD decomposition for efficient logic synthesis. In *Proc. ICCD*, pages 626–631, Oct. 1999.

[7] C. Scholl and B. Becker. On the generation of multplexer circuits for pass transistor logic. In *Proc. DATE*, pages 372–378, Mar. 2000.

[8] T. H. Cormen*et al. Introduction to Algorithms.* Prentice-Hall India, New Delhi, 1998.

[9] F. Ferrandi *et al.* Symbolic algorithms for layout oriented synthesis of pass transistor logic circuits. In *Proc. ICCAD*, pages 235–241, Nov. 1998.

[10] A. Reis*et al.* The library free technology mapping problem. In *Proc. IWLS*, May 1997.

[11] Y. Jiang, S. S. Sapatnekar, and C. Bamji. A fast global gate collapsing technique for high performance designs using static cmos and pass transistor logic. In *Proc. ICCD*, pages 276–281, Oct. 1998.

[12] F. Somenzi. CUDD: CU Decision Diagram package, release 2.3.0. http://vlsi.colorado.edu/ fabio/CUDD/.

[13] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. DAC*, pages 42–47, Jun. 1993.