Efficient Layout Synthesis Algorithm for Pass Transistor Logic Circuits[*]

Rupesh S. Shelar and Sachin S. Sapatnekar
Department of Electrical and Computer Engineering,
University of Minnesota, Minneapolis, MN 55455.
Email: {rupesh, sachin}@ece.umn.edu

Abstract

In this paper, we address the problem of layout generation of pass transistor logic (PTL) circuits from binary decision diagrams (BDD's) by laying out groups of transistors in rows. We use a max-flow min-cut based recursive bipartitioning procedure followed by a greedy heuristic to assign the BDD nodes to rows, such that the number of wires to be routed across the rows is minimized. Next, we use an Eulerian trail approach to form clusters of transistors among which diffusion-sharing is possible. These clusters are then placed using a linear tree placement algorithm that ensures optimality in the wiring overhead for connecting clusters in the same row. The experimental results over benchmark circuits are promising.

1 Introduction

Pass transistor logic circuits have a great potential to serve as an alternative to static CMOS because of smaller area due to the use of NMOS-only pass transistors and the consequent reduction in capacitances, and for their potential for high speed, low power implementations [1, 2]. Although they are often employed in arithmetic logic units, their wider use is limited due to unavailability of good layout tools. In this paper, we address this problem and propose an automated layout generator for PTL circuits.

The layout generation problem for PTL circuits has received relatively less attention as compared to logic synthesis for PTL [3–5] for various optimizations. In [6], Macii *et al.* propose a layout generator for PTL circuits that is suitable for standard cell layouts. The limitation of their approach is that they do not fully exploit diffusion-sharing between the PTL multiplexers, which could potentially save a large amount of area. The PTL layout work by Yano *et al.* does utilize diffusion-sharing using the idea of Eulerian trails in PTL multiplexers, but its applicability is limited to small cells, typically up to 4-5 inputs [7]. The limitation to small cells is justified by the fact that buffers must be inserted after every few pass transistors, and that blocks of pure pass transistor logic that lies between buffers, which correspond to cells in their work, tend to have a small number of inputs. This approach corresponds to traditional cell-based approach that generates layouts for individual PTL cells and assembles these by placing blocks of cells in rows in accordance with a standard cell layout methodology.

Our algorithm has advantages over both of the above approaches. Unlike [7], our approach takes a more flexible view of the boundaries between cells. While the layouts generated by our algorithm fit into the outline of a standard cell layout methodology, we do not limit individual cells to have rectangular boundaries, and permit a more fluid boundary between individual cells of pass transistor logic. Related work that also uses a flexibile boundaries between the cells was proposed recently by Gopalakrishnan *et al.* but in the context of static CMOS cells [8]; our algorithm differs from the latter in the use of linear tree placement, biparitioning and row assignment as well as in the cluster formations. We assign groups of transistors to rows and minimize the layout area by maximizing diffusion-sharing, minimizing the wiring area required to route the intra-row signals and minimizing the number of inter-row signals.

The organization of the rest of the paper is as follows. In Section 2, we illustrate a layout model for PTL circuits, followed by a description in Section 3 of diffusion-sharing in PTL circuits based on the adjacency relation between the BDD nodes. Section 4 provides a detailed description of various steps in the algorithm, and Section 5 presents the experimental results followed by conclusion.

2 Layout Model

Due to the well known relationship between PTL and binary decision diagrams (BDD's), we use the BDD representation of a logic function as the input to our approach. Figure 1(a) shows a PTL multiplexer represented by a BDD[1] node; a Boolean function f is represented by $f = Af_A + \overline{A}f_{\overline{A}}$, where f_A and $f_{\overline{A}}$ are the Shannon cofactors of the function f with respect to the variable A. This BDD node is well known to map on to a multiplexer, and Figure 1(b) shows a layout for this multiplexer that is

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

[1]For the purposes of this paper, all BDD's are reduced ordered BDD's (ROBDD's).

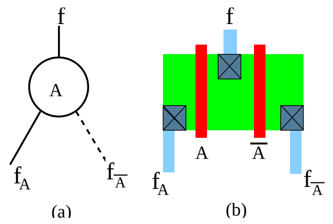

Figure 1. Layout of a multiplexer: (a) A BDD node (b) Its corresponding layout

made compact by allowing the drains of two multiplexer transistors share the diffusion. Empirically, it has been observed in [7] that exploiting diffusion-sharing at the output of a multiplexer produces a good quality layouts for small cells, and also reduces the search space of a layout generator. Therefore, our work also employs the approach of maximizing diffusion-sharing. We allow transistors to be laid out either horizontally or vertically to maximize diffusion-sharing among transistors unlike [6].

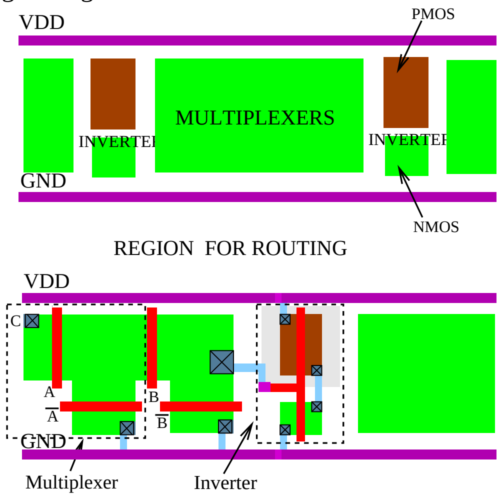

Figure 2. A row-based layout scheme for PTL

A typical pass transistor logic circuit contains a set of multiplexers, normally implemented using NMOS pass transistors, and inverters after every three pass transistors in series to boost the signals and also ensure correct parity of signals in the PTL network. Our layout model places the inverters and PTL multiplexers in rows of fixed height, as shown in Figure 2. The figure shows two rows of logic with an intervening channel for routing, with each row containing PTL multiplexers and inverters. The lower row shows a detailed view of the inside of some typical blocks, illustrating the layout of a 3-input NAND gate using PTL multiplexers and an inverter. Each row has the same height and rows are separated by a space for routing both intra-row and inter-row signals. We minimize this area by performing optimal linear tree placement

The proposed layout model can be extended easily to accommodate static CMOS cells due to the placement of power supply and ground lines for each row; however, the mixed synthesis of PTL and static CMOS is not addressed here and is a topic for future work.

3 Diffusion Sharing in PTL circuits

The manner in which diffusion can be shared is dependent on the shapes of layouts and the sizes of the transistors. Shape level optimizations include making a choice between laying out the transistors in a multiplexers either horizontally as shown in Figure 1(b), or vertically, as shown in Figure 3(a), or using a hybrid of the two, as shown in Figure 3(b). In terms of sizing, we consider only uniformly sized transistors in this paper. However, the layout of arbitrarily sized transistors can be handled using our framework by considering transistor folding in case of large sized transistors and weighting the BDD nodes corresponding to such transistors with an appropriately large area cost during bipartitioning and row assignment. Subsequently, shape level optimizations and diffusion-sharing may be used to handle arbitrary transistor sizes. We do not, however, consider this issue in our work and expect to address it in future work.

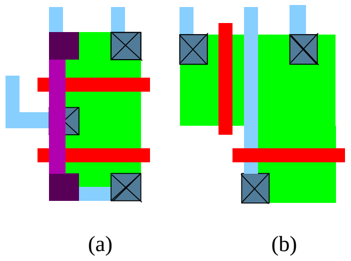

Figure 3. Different multiplexers layout schemes: (a) with vertical transistors (b) with horizontal as well as vertical transistors

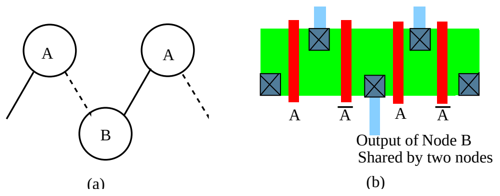

Figure 4. An example of input diffusion-sharing: (a) A BDD (b) Its corresponding PTL implementation

In case of PTL circuits synthesized from BDD's, we observe that diffusion sharing between any two nodes may be possible in only the following two cases:

Input diffusion-sharing Two nodes share the same co-factor, as shown in Figure 4(a).

Output diffusion-sharing One node is a co-factor of the other node, as shown in Figure 5(a).

The layouts for each case are shown in Figures 4(b) and 5(b), respectively. Note that in Figure 5(b), the transistor associated with input $\overline{B}$ is laid out vertically and so that it shares the diffusion at the output of the multiplexer with the transistor associated with input B, and also with the transistor associated with input A of the adjacent multiplexer. If two nodes share both cofactors, as shown in Figure 6(a), then diffusion-sharing can be obtained as shown in Figure 6(b).

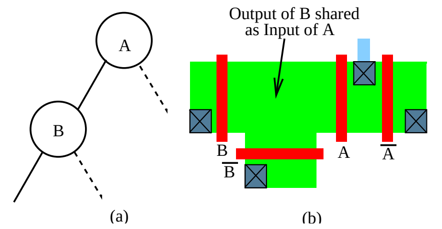

Figure 5. An example of output diffusion-sharing: (a) A BDD (b) Its corresponding PTL implementation

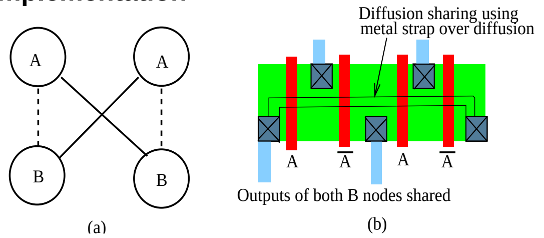

Figure 6. A diffusion-sharing scheme for the case when two cofactors are shared: (a) A BDD (b) Its PTL implementation

4 Algorithm

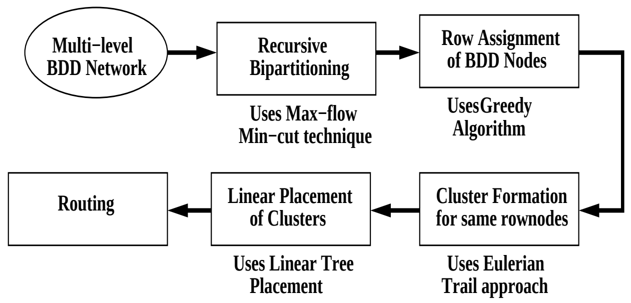

Figure 7. An overview of the algorithm

Figure 7 outlines the overall algorithm for layout of PTL circuits. The algorithm begins with a multilevel BDD network[2] of the circuit and uses a recursive bipartitioning scheme to assign the BDD's corresponding to various regions to different parts of the layout. This is followed by greedy assignment of the rows to the BDD nodes such that the number of signals across different rows is minimized and each row has approximately same number of nodes and inverters. This bipartitioning approach can be interpreted as performing a coarse layout region assignment, similar to the use of partitioning in placement problems, and can

[2]A multilevel BDD network [9] is similar to a multilevel Boolean network, and uses a BDD to describe the functionality at each level of the network.

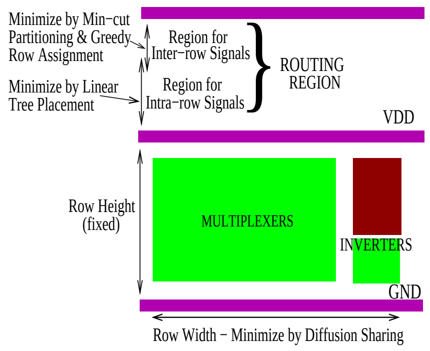

Figure 8. A pictorial view of area minimization strategies for the layout

be formulated as a max-flow min-cut problem. The greedy procedure assigns nodes in each BDD in the multilevel representation to rows, and takes into account possibilities of diffusion sharing. The assignment is performed in such a way that the number of wires from any row to another row is minimized. After the row assignment is completed, the multiplexers have been assigned to rows, but their positions within a row have not been finalized. To do so, we first cluster nodes in each row using an Eulerian trail approach to maximize diffusion-sharing. Once the clusters have been formed, they are assigned specific positions in each row; this is performed using a linear tree placement algorithm proposed by Yannakakis [10]. Figure 8 summarizes the features of our approach for minimizing the area of the layout.

4.1 Recursive Bipartitioning

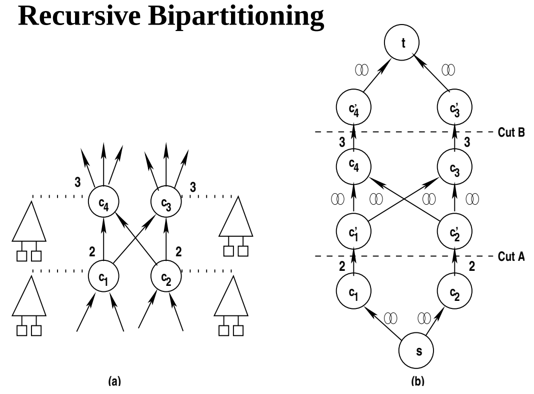

Figure 9. Recursive bipartitioning: (a) A multi-level BDD network, (b) Its corresponding flow network

The multilevel BDD network can be treated as a directed graph, where each vertex corresponds to a BDD and a directed edge means that output of a vertex represented by a BDD is an input to the BDD of another vertex; the edges are assigned directions leaving from primary inputs and pointing towards primary outputs. Our aim is to find a cut that partitions a given multilevel BDD network into approximately equal parts in terms of the area occupied by the transistors and has minimum connectivity among the partitions.

To obtain such an area-balanced partition, we associate two costs with each node in the multilevel BDD network that are defined as follows.

Lower Cost (A_{lower}**)** : This is the cost of a PTL implementation of the multilevel BDD network rooted at a given node.

Upper Cost (A_{upper}**)** : This is the cost of a PTL implementation of of the multilevel BDD network rooted at a given node, assuming the directions of edges reversed.

Since we are interested in area balanced partition, we define candidate nodes for the cut as follows.

Candidate Nodes are the nodes for which $|A_{lower} - A_{upper}| \leq \Delta$, where Δ indicates flexibility while choosing candidate nodes.

To ensure the minimum connectivity, we transform bipartitioning problem to the max-flow min-cut problem in which each candidate node is assigned a flow capacity equal to its number of fan-outs. Figure 9 shows a part of multilevel BDD network containing candidate nodes c_1, c_2, c_3, c_4 and its corresponding flow network with a source node s and a destination node t. Figure 9(b) shows two possible cuts: Cut A with a capacity of 4, and Cut B with a capacity of 6. The application of the Ford-Fulkerson algorithm [11] finds the min-cut, which in this case is Cut A and the corresponding nodes in the cut are c_1 and c_2. In the resulting partition, nodes c_1, c_2 and their predecessors will be in one subset, while nodes c_3, c_4 and their descendents will be in another subset. Although similar to [5], our recursive bipartitioning differs from the latter in the objective as well as the application.

4.2 Greedy Algorithm for Row Assignment

After recursive bipartitioning of the multilevel BDD's, the BDD's are assigned an ordering for layout. To generate a compact layout for each BDD, we must assign the nodes in each BDD to rows such that the diffusion-sharing among the nodes in the same row is maximized. We use a greedy algorithm for this purpose, as described in Figure 10. The set of multiplexer nodes assigned to each row is first initialized to the empty set. Subsequently, the algorithm successively adds nodes to the set as long as a user-defined area capacity of the row is not exceeded; this user-defined area capacity can be used to control the width of the layout. In each step, the node (found by the MostAdjacentNode() routine) that is greedily added to the set is the one that is adjacent to the maximum number of nodes that are already in the set; any ties are broken in favor of a node with a lower variable index in the variable ordering for the BDD. This strategy serves two purposes: maximizing the possibility of diffusion sharing and minimizing the number of edges going across different rows. This greedy algorithm implicitly minimizes the row area since it considers the effects of diffusion-sharing. The following proposition states the complexity of the algorithm without proof due to space limitation.

```
Input:  $G(V,E)$, graph underlying a given BDD
Output:  Mapping $f : V \to \{0, 1, \cdots, row_{max}\}$
Steps:
$V_{left} \leftarrow V$; $row \leftarrow 0$;
while($V_{left} \neq \Phi$)
    {
    $Area \leftarrow 0$; $S_{row} \leftarrow \Phi$;
    while($Area$ < $MaximumArea$)
       {
       $v \leftarrow$ MostAdjacentNode($S_{Row}$,$G$,$V_{left}$);
       $S_{Row} \leftarrow S_{Row} \bigcup v$;
       $V_{left} \leftarrow V_{left}$ - $v$;
       $Area \leftarrow$ AreaEstimation($S_{Row}$);
       }
    AssignRow($S_{Row}$, $row$);
    ++$row$;
    }
```

Figure 10. Pseudo-code for the greedy algorithm that assigns multiplexer nodes to rows

Proposition 4.1 *The algorithm in Figure 10 terminates in* $O(N \cdot \|S_{Row}\| \cdot d_{max})$ *time, where* N *is the total number of nodes,* $\|S_{Row}\|$ *is the cardinality of the row and* d_{max} *is the maximum degree among all nodes.*

4.3 Clustering

The concept of Eulerian trails has often been used for diffusion-sharing optimizations in static CMOS standard cells. In our problem, we use this notion of Eulerian trails, tailored specially for PTL layout.

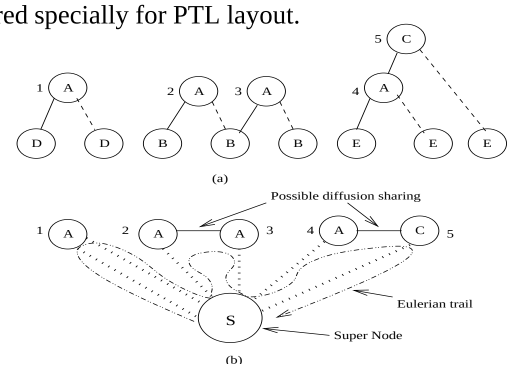

Figure 11. Cluster formation: (a) A group of BDD nodes to be placed (b) The corresponding Eulerian graph

The width of each row can be minimized by diffusion-sharing among various multiplexers placed in the same row. To form cluster of multiplexers that can share diffusion, we use the Eulerian trail algorithm [12] after Eulerizing a given graph by adding a super node and connecting nodes with an odd[3] degree to it. The Eulerian trail algorithm [12] was used in [13] for two dimensional micro-cell placement in the context of CMOS cells. The nodes in our Eulerized graph for

[3]The nodes with 0 degree are also connected to the super node and this does not affect the number of diffusion breaks.

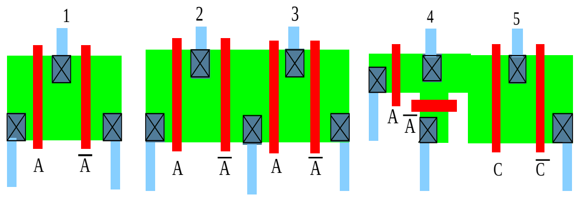

Figure 12. Diffusion sharing clusters corresponding to Figure 11

a row correspond to BDD nodes assigned to the row, and the edge set consists of two types of edges: ones that denote possible diffusion-sharing and the others that denote diffusion breaks. As explained in the Section 3, in case of PTL, diffusion-sharing can occur in case of input sharing, illustrated in Figure 4, or output sharing, as shown in Figure 5. To capture this relationship, we introduce edges between the corresponding nodes in the Eulerized graph. Dummy edges indicating diffusion breaks connect the odd degree nodes to a super node. Figure 11(a) shows an example with five BDD nodes, corresponding to variables A and C, to be placed in a row assuming that nodes with labels B, D and E are already placed in previous rows. The corresponding Eulerized graph is shown in Figure 11(b) and Figure 12 shows the cluster formations corresponding to the Eulerian trails in Figure 11(b). Three clusters are identified: one with one multiplexer and two with two multiplexers each.

4.4 Linear Placement

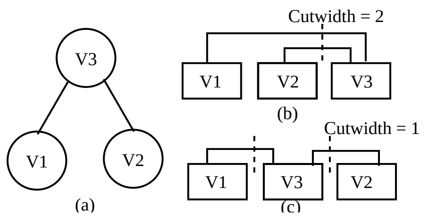

Figure 13. Linear placement for laying out the clusters: (a) A cluster tree, (b) A sub-optimal placement (c) An optimal placement

Although every Eulerian trail yields a solution with the same minimum width, the routing cost varies for different trails, corresponding to different arrangements of the diffusion-sharing clusters. To optimize this cost, we perform linear tree placement of the clusters while preserving the order of the multiplexers in each cluster. Since the multiplexer order in each cluster is preserved, altering the order of the clusters will not affect the row width. The objective here is to minimize the cost involved in routing the signals that connect different clusters in the same row.

We define a graph, called a cluster graph, $G(V, E)$ on clusters in which every vertex $v \in V$ corresponds to a cluster and there is an edge between the vertices if there are signals connecting the corresponding clusters. If $G(V, E)$ is a tree, then use of Yannakakis' algorithm [10] results in an optimal placement that minimizes the cut-width; here the cut-width corresponds to the number of rows required to route the signals between different clusters. Yannakakis' algorithm uses dynamic programming to build the solution in a bottom up manner and runs in $O(N \times \log N)$ time, where N is the number of vertices. If the cluster graph is not a tree, then we arbitrarily remove some of the edges to make it a tree; in this case, however, the optimality of solution is not ensured. It is seen that for BDD's, the cluster graph is typically close to a tree. We observe that a cycle can appear in a cluster graph if there are forward edges in the BDD that connect a node to its ancestor, and the node at either end of the the forward edge and the parent of the node are placed in the different clusters of the same row. However, such a structure is likely to result in a great deal of diffusion-sharing, and this increases the probability that all such structures would be placed within the same cluster. Therefore, cycles between clusters, while not impossible, are improbable; it is also verfied from experiments on the benchmark examples. Figure 13(a) shows a cluster tree containing three clusters V1, V2 and V3, with connections between V1 and V3 and between V2 and V3. Figure 13(b) shows a suboptimal placement of these clusters that results in a cut-width of two, implying that number of rows required to route the signals will be two if we use two metal layers. Figure 13(c) shows an optimal placement with cut-width of 1 that will be found by Yannakakis' algorithm, implying that one row suffices to route the signals between the clusters. This example shows the impact of linear tree placement and its contribution to minimizing the area.

5 Experimental Results and Conclusion

We have implemented the algorithms discussed in this paper in a C++ program that generates layouts for PTL circuits from BDD's; the CUDD package [14] is used to construct multilevel BDD's. Inverters are inserted after every three pass transistors in series; we use a heuristic for inverter insertion. The technology that we use is scalable CMOS technology with one poly and three metal layers. Poly is used for routing inputs while metal1 and metal2 are used for routing intra-row signals while metal3 is used for routing inter-row signals. Metal1 is allowed to route in vertical as well as horizontal direction while Metal2 is allowed to route in only horizontal direction and Metal3 is allowed to route in mostly vertical direction. We use the left-edge algorithm for routing intra-row and inter-row signals. More sophisticated routing algorihtms such as those using dog-legs, or the use of a commercial router will, of course, result in smaller routing channels and therefore, greater area reduction. Our results with the left-edge algorithm for routing are, therefore, upper bounds on the area.

Table 1 shows the experimental results on several benchmarks. Column 2 and Column 3 shows the width and height of the layout in terms of λ for the example in Column 1 while Column 4 shows the number of rows. Column 5

Example	Our Area (μ^2)	Area [6] (μ^2)	Std. Cells [6] (μ^2)
rd53	564	-	-
rd73	1134	-	-
rd84	2016	-	-
9symml	1310	-	-
cmb	1250	-	-
parity	1224	-	-
C17	165	-	-
z4ml	914	-	-
sao2	3220	-	-
f51m	3300	-	-
misex1	1404	-	-
C1355	8584 (45%)	15725	19331
C1908	18573 (28%)	23845	29442
C499	10115 (35%)	15725	19331
C6288	147712 (1%)	149235	276288

Table 1. Layout area for benchmark circuits

shows the area of the layout for a 0.25μm technology, while Column 6 shows the area required for layout generation algorithm reported in [6]; the comparison is fair since their work also works with a 0.25μm technology. Column 7 shows the area required for a static CMOS based standard cell implementation as reported in [6]. We observe that our algorithm produces more compact layout than those produced by [6] although there is further scope for improvement in our results if we use more sophisticated routing algorithms. The area reduction can be attributed to diffusion sharing, linear tree placement, greedy row assignment as well as min-cut bipartitioning.

Figure 14 shows the placement of multiplexer clusters and inverters in three rows for an MCNC benchmark rd73; for clarity, the routing connections are not shown explicitly. The benchmark rd73 has 7 inputs, 3 outputs with a BDD representation that has a size of 42 nodes while its transistor level implmentation include 80 transistors apart from 8 inverters. Figure 14 clearly shows the effect of greedy row assignment that results in area-balanced rows with high amount of diffusion sharing among each row. Moreover, it is quite clear that the flexibility of placing transistors either horizontally or vertically serves to reduce the area and create compact layout. All rows show the effect of diffusion sharing and its contribution to width minimization.

In this paper, we have described an algorithm for layout of PTL circuits. Our algorithm has advantages over previously published approaches and this is also justified by experimental results on MCNC benchmarks.

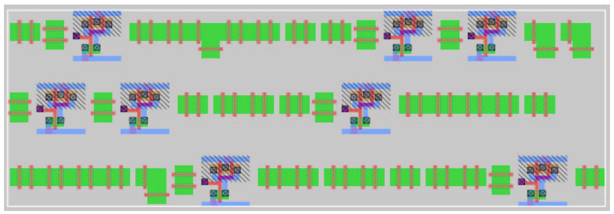

Figure 14. Effect of row assignment, clustering, Linear tree Placement on rd73

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

[5] R. S. Shelar and S. S. Sapatnekar. Recursive Bipartitioning of BDD's for Performance Driven Pass Transistor Logic Synthesis. In *Proc. ICCAD*, pages 449–452, Nov. 2001.

[6] L. Macchiarulo, L. Benini, and E. Macii. On-the-fly layout generation for PTL macrocells. In *Proc. DATE*, pages 546–551, Mar. 2001.

[7] Y. Sasaki, K. Rikino, and K. Yano. ALPS: An automatic layouter for pass-transistor cell synthesis. In *Proc. ASP-DAC*, pages 227–232, Feb 1998.

[8] P. Gopalakrishnan and R. Rutenbar. Direct transistor-level layout for digital blocks. In *Proc. ICCAD*, pages 577–584, Nov. 2001.

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

[10] M. Yannakakis. A polynomial algorithm for the min-cut linear arrangement of trees. *Journal of the Association for Computing Machinery*, 32(4):950–988, Oct. 1985.

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

[12] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, New York, 1998.

[13] M. A. Riepe and K. A. Sakallah. Transistor level micro-placement and routing for two-dimensional digital vlsi cell synthesis. In *Proc. ISPD*, pages 74–81, Apr. 1999.

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