Performance Optimization in Layout Driven Synthesis

by

Nels Vander Zanden
Allen C. H. Wu
Daniel Gajski

Technical Report 89-21

Information and Computer Science Department
University of California, Irvine
Irvine, CA. 92717

Abstract

This paper describes a method for incorporating layout parameters to better meet performance constraints. We define an algorithm for synthesis of high-performance designs and present a synthesis tool for use with custom layout generators. Experimental results indicate such an approach produces faster layouts and permits greater area/time tradeoffs than traditional logic synthesis systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

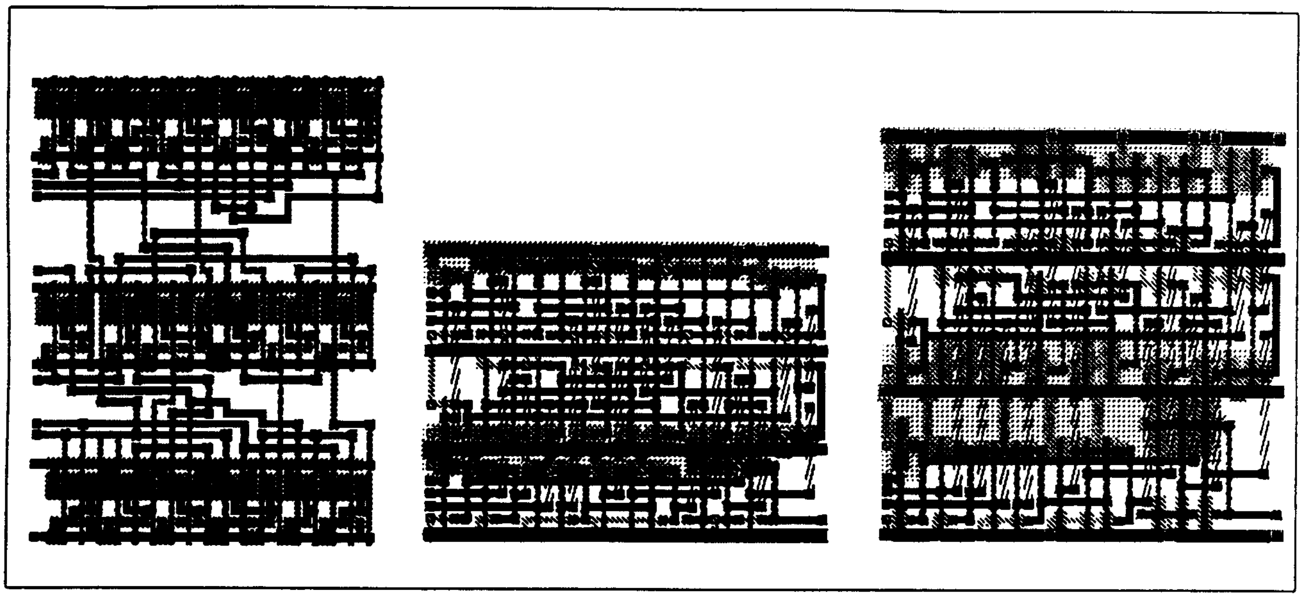
Logic synthesis tools help reduce design complexity, thereby increasing productivity and reducing chip development time. They transform a functionally correct design consisting of generic components into one that has been optimized to meet a designer's constraints for a given component library. Numerous logic synthesis tools have been developed: LSS [JoTr86], SOCRATES [GrBa86], LOGICIAN [BeOw87], MIS [BrRu87], BOLD [BoHa87], DAGON [Ke87], and MILO [VaGa88]. Commercially available tools include the Logic Consultant [Kim87], the Synopsys Design Optimizer, and Silc Technologies Silcsyn. These tools incorporate synthesis and optimization techniques in their design cycle.

Logic synthesis begins with basic functional or structural descriptions such as boolean equations, gate-level schematics, or PLA formats. Descriptions are collapsed into two-level SOP form and minimized to eliminate redundant terms. Factorization can be employed to minimize the number of gates and consequently the number of transistors. Finally, the design is converted to technology-specific components, such as those in gate-array or standard cell libraries, and optimization is performed to meet constraints for time and area.

Conventional logic synthesis systems produce designs for gate array and standard cell libraries. Such semi-custom libraries are popular for their simplicity in layout and the ability to use automatic layout tools. A major weakness of the semi-custom approach is the larger area required for routing, producing a lower layout density than custom methods. In addition, all transistors are of the same size, capable of driving some expected average load. This average load is usually greater than what is required, making cells larger than necessary. When a larger load than the preset value must be driven, speed is sacrificed.

In order to achieve better layouts, more attention has been focused on tools that offer custom layout. Custom layout permits greater control over layout parameters, providing smaller area and faster designs than its gate-array or standard cell counterparts. For example, Figure 1 shows three layouts for the same logical function. The first layout was done using standard cells; the second and third, using custom layout, were optimized for area and time, respectively, by varying transistor sizing and complex gate formation. The second layout has 22% less area than the standard cell version and the third layout is 30% faster than the standard cell version.

A number of layout generators that produce custom layouts from gate-level input have been reported (TOPOLOGIZER [Kol85], LES [LiGa88], PAMS



(a) Standard Cell Layout (b) Custom Layout for Area (c) Custom Layout for Speed

Figure 1: Standard Cell Layout vs. Custom Layout

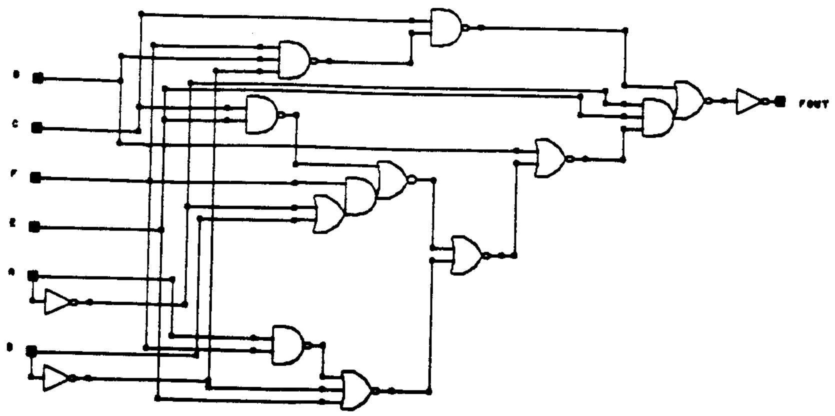
[TsCe88], SOLO [BaAl88], CLAY [KoLu88]). They offer improved routing, such as through-the-cell instead of routing channels, generate cells dynamically instead of having fixed libraries, and can size transistors as instructed by user input, insteading of having a fixed size. With these improved layout capabilities, new techniques are required to consider layout in the synthesis process and manipulate layout parameters to produce higher-quality designs. Such considerations have not been incorporated in previous synthesis systems. Hence they fail to achieve optimal results when a custom layout is used. For example, consider Figure 2 which shows two different implementations of the same logical function. The design in Figure 2(a) has been optimized specifically for area. Figure 2(b) displays the implementation that has been optimized for time. Both designs were produced using standard cell synthesis techniques. When the layout was implemented in standard cells, the optimization achieved the desired effect. However, when these implementations were run through custom layout generators, the design of Figure 2(b) had the best speed AND the best area as shown in Figure 2(c).

2. Parameters for Custom Layout

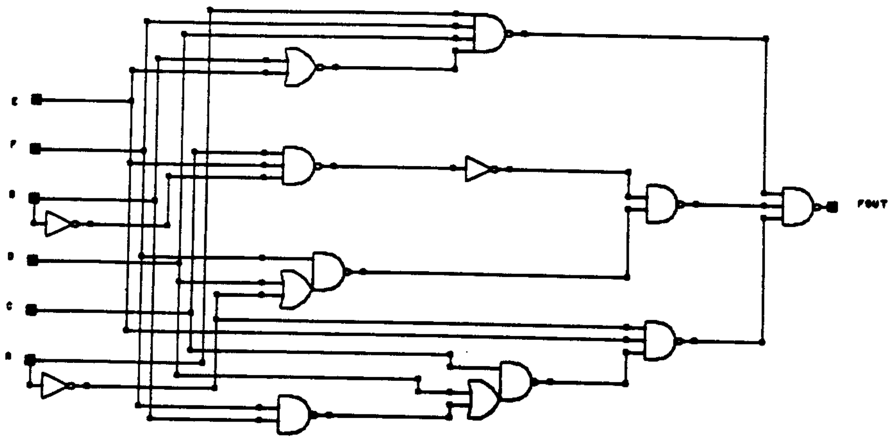
2.1. Complex Gates

Layout driven synthesis is not restricted to a fixed library of components. In many cases multi-level Boolean functions can be implemented as a single complex gate. Complex gates have fewer connections and fewer transistors than multiple gate implementations and hence may reduce area and improve performance. An example of a complex gate is shown in Figure 3(a) and contrasted with the multiple gate implementation in Figure 3(b) (with inverters assumed to be pushed back to previous levels). Both implementations are shown in the CMOS technology with cells having a P and N transistor section.

In layout driven synthesis, the optimizer has great flexibility in deciding which gates to combine into a single gate. The type of complex gates formed is



(a) Optimized for Area

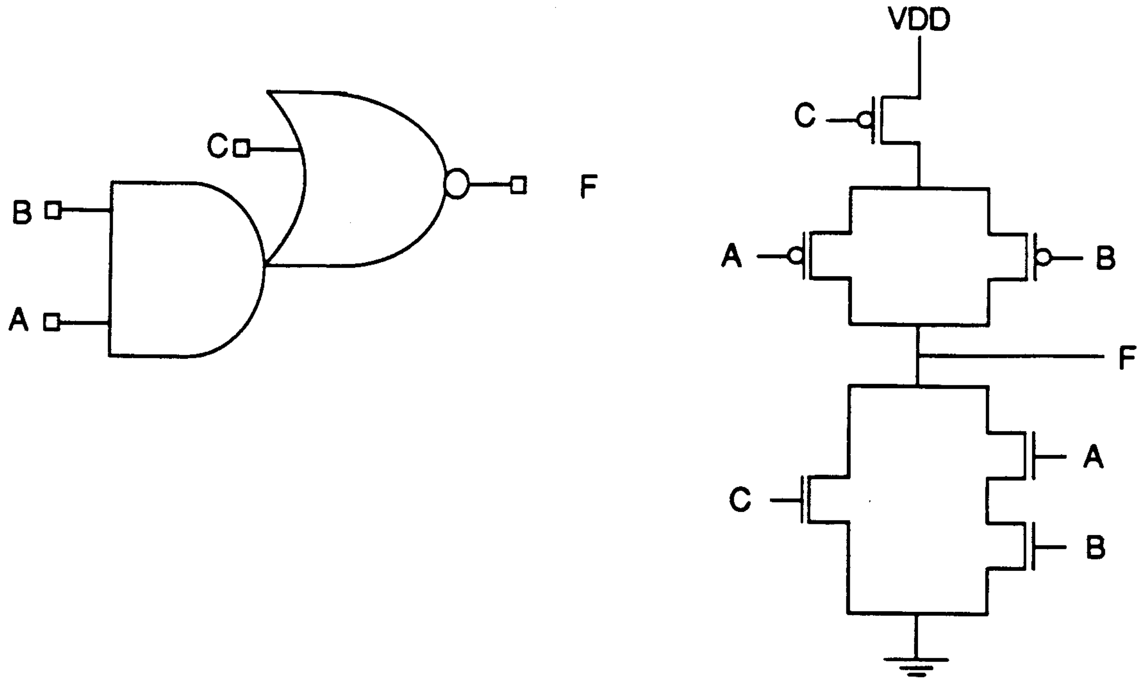


(b) Optimized for Time

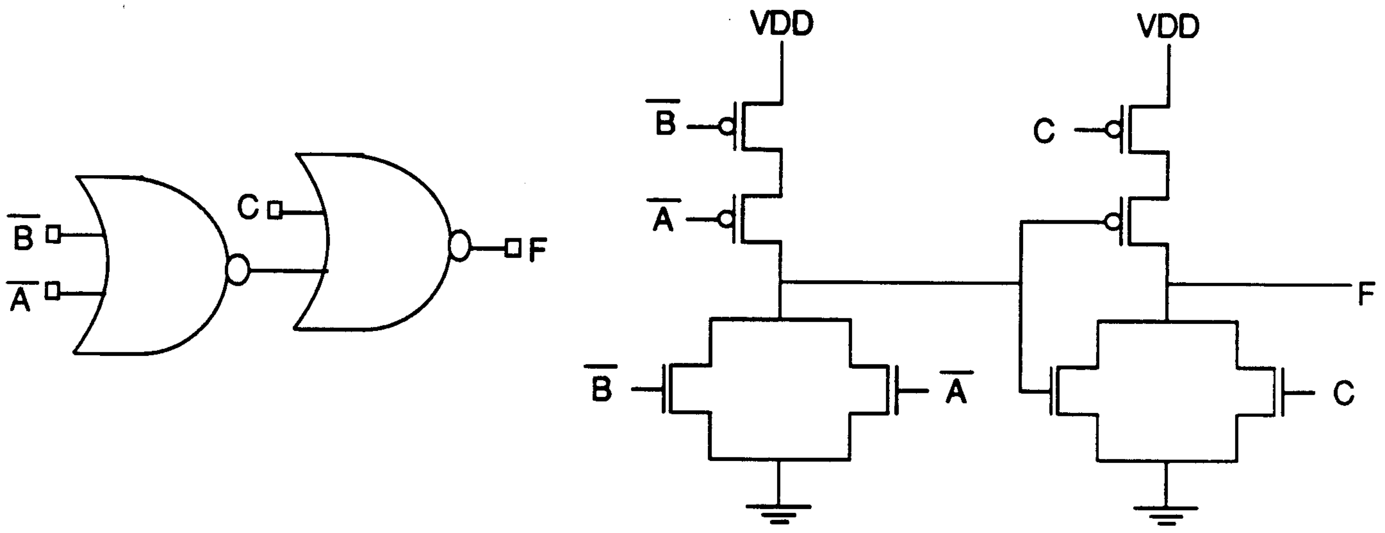
Optimized for:	Custom Layout		
	# of Tran-sistors	Layout Area (um^2)	Delay (ns)
Speed	56	56048	9.61
Area	54	59800	9.78

(c) Layout Comparison

Figure 2: Custom Layout Results Using Synthesis for Standard Cells



(a)



(b)

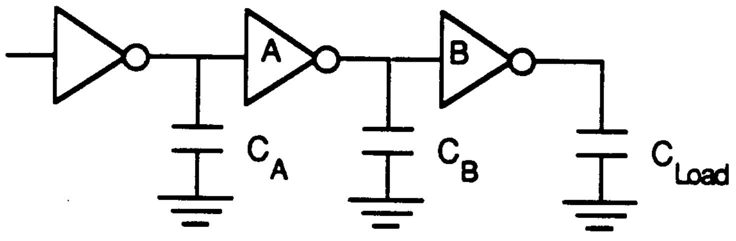
Figure 3: Complex Gate Formation

limited mainly by the buildup of capacitance, the load the gate must drive, and the transistor speed. As more transistors are placed in series, the resistance and parasitic capacitance of the gate increase, resulting in longer delay times. In CMOS, where gates have an N and P transistor section, carrier mobility in P-type transistors is twice as slow as N-type transistors. NOR gates, which have P-type transistors in series, can be 2 to 3 times slower than gates with N-type transistors in series (such as NAND gates). This makes it beneficial to limit the number of transistors that a complex gate has in series and parallel. In CMOS, fewer P-type transistors should be allowed in series than N-type transistors. Hence, a compromise must be made between the solution consisting of only single gates and the solution involving complex gates with a large number of transistors. As a general rule, complex gates with few transistors are desired along paths where timing is critical while complex gates with more transistors are desired in sections where area is most important.

2.2. Transistor Sizing

Another concern in custom layout is transistor sizing. As discussed in [FiDu85], [He87], [Ci87], and [ObKa88], choosing proper transistor sizes is key to circuit performance. Figure 4 demonstrates how changes in transistor size affect speed. Increasing the size of transistors in Gate B improves its speed. However, the larger transistors create a larger capacitive load for Gate A. This slows down Gate A and hence the entire path unless Gate A's transistors are increased based upon the new load. The size vs delay relationship is a convex curve as in Figure 4(b). Thus sizing a gate's transistors excessively large can increase the total path delay if the transistor sizes of the gates that drive it are not increased as well.

Transistor sizing gives greater control over area/speed tradeoffs. For example, in CMOS where NOR gates tend to be slower than NAND gates, synthesis systems try to avoid using NORs or use NORs with fewer inputs.



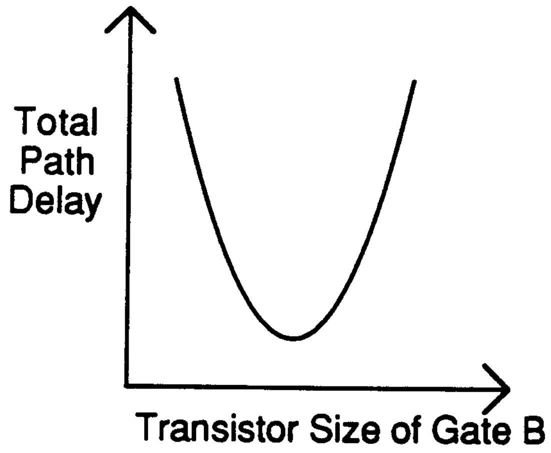
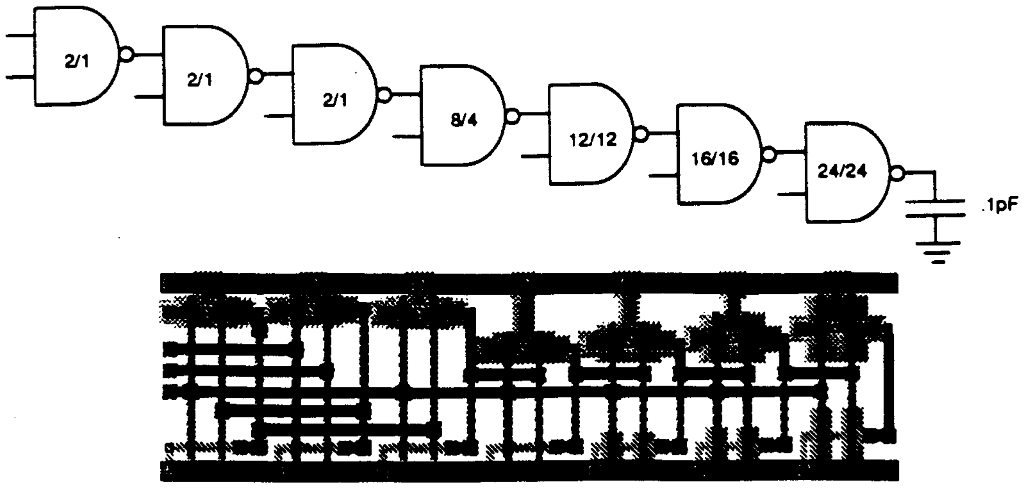


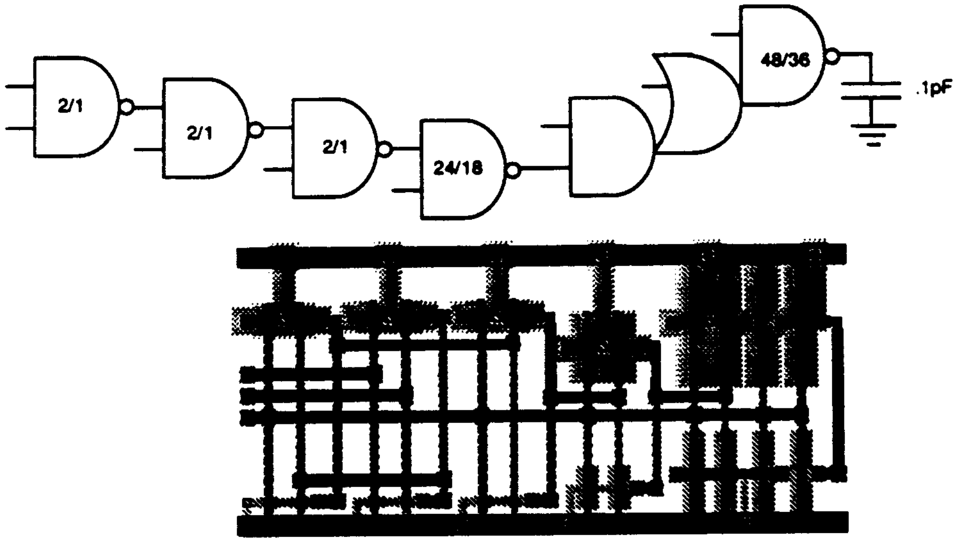
Figure 4: Effect of Transistor Sizing on Path Delay

Layout driven synthesis systems can increase the size of the P transistors, thereby improving speed (i.e., making the gate as fast as a NAND) at the expense of increased transistor area.

Since transistor sizing is not variable in standard cells there is no need to consider load when using complex gates. However, for custom layout, gates must be combined in such a way to avoid a number of problems. For example, as the load a complex gate drives increases, transistor sizes must be made larger to prevent a decrease in speed. Larger transistors require strips with greater height creating the problem shown by the two layouts in Figure 5. The first layout consists of seven NAND gates. In the second layout the final three NAND gates have been combined into a complex gate. Both designs drive an output of .1 picofarads (roughly 2-4 fanouts). Even though the transistor area for the complex gate is less than that for the individual gates, total area has been increased. One can see from Figure 5(b) that a tall cell in a strip is not compatible with shorter cells, generating wasted space along the top and bottom of the channel. In addition, larger transistor sizes increase the load for all gates in the preceding stage, requiring them to be increased in size if the present speed is to be maintained. This creates a chain reaction affecting all preceding stages that can significantly increase the area. The design of Figure 5(b) is slower than that of Figure 5(a). The delays could be made equal by using even larger transistors sizes, resulting in a much larger area for the same delay. Hence the



Area = $312 * 90 = 28080um^2$, $T_{rise} = 14.78ns$, $T_{fall} = 13.89ns$
(a)



Area = $256 * 122 = 31232um^2$, $T_{rise} = 14.44ns$, $T_{fall} = 15.11ns$
(b)

Transistor sizes shown inside gates as PFET size/NFET size
Only the last 4 NAND gates' sizes are shown

Figure 5: Effect of Large Transistor Sizes in Complex Gates

formation of complex gates should be dependent upon the load. Single gates or complex gates with few transistors should be used to drive heavy loads. In general capacitive loads are larger closer to output pins and smaller closer to input pins. This indicates that complex gates with a larger number of transistors can be created near input pins, while fewer transistors should be used in complex gates near output pins.

2.3. Placement and Routing

Control over component placement is important as cells along paths having critical delays should be placed close to one another. This prevents long wires from connecting them and introducing further delays. Further, large size gates should be placed on the edges of the floorplan since routing is sparse near boundaries. Same size gates can be placed in the same row to reduce wasted space from uneven cells. This increases the routing but decreases overall area. Such a scheme could have been used to correct the problem of Figure 5(b).

A related concern is in routing. Routing should be performed on critical components first to ensure that they get the shortest paths. In addition, long wires should be placed in the metal layer for better speed. A synthesis program can aid layout by assigning priorities for the layout program, indicating which components need to be placed close together or which cells are similar in size.

2.4. I/O Positioning

Placement of I/O pins also is crucial for quality layouts. Pins can be placed near the modules having critical timing or placed in a position that reduces overall routing and saves area. They must be kept well distributed to prevent congestion in the center of the module and wasted space around the boundary of the module. Thus information on good I/O positioning can help layout tools create faster or more dense layouts.

3. Strategies for Layout Driven Synthesis

There are several strategies for controlling complex gate formation and transistor sizing.

One simple strategy is to form complex gates first and then size the transistors. During the complex gate formation stage, transistors are assumed to be unit sized. For example, in a 3 micron CMOS technology all NFETs and PFETs would initially be given a size of 3 microns. The weakness of this strategy is that transistor sizes do not influence complex gate formation and thus create complex gates with many large transistors. This unnecessarily increases the layout area.

Using a second strategy, transistors are sized first and then the complex gates created. With this approach more realistic transistor sizes are used in deciding how to form complex gates. Those gates with small transistor sizes are used to build complex gates; those with large sizes are mostly left untouched. The new complex gate's transistor size is based upon the largest transistors of the gates that were merged. When this new complex gate is inserted, the gates that drive it may be driving larger transistors than they were before. Since these gates must drive a larger capacitive load, they are now undersized if the present speed is to be maintained.

The third and most complicated strategy involves forming complex gates and sizing transistors at the same time. Before a decision is made to create a complex gate, all gates that the new complex gate will drive must be processed. That is, no changes in transistor sizes or complex gates can be made along any paths that can be reached from the output of the new complex gate. Doing so could change the load that the complex gate is required to drive, resulting in the problems encountered in the second strategy. After a new complex gate is created, all gates that drive it are resized. This approach will presumably provide the best results.

4. Algorithm for Layout Driven Synthesis

We present an algorithm for producing high-performance CMOS custom layouts. Our algorithm consists of four phases as shown in Figure 6. The input is a set of boolean equations. These equations are first minimized and then factored by MISII to make use of common terms. Parameters for the maximum number of NFETs and the maximum number of PFETs allowed in series are provided for the next phases to place limits on the size of complex gates created. In the second phase, the algorithm reduces the number of levels along the

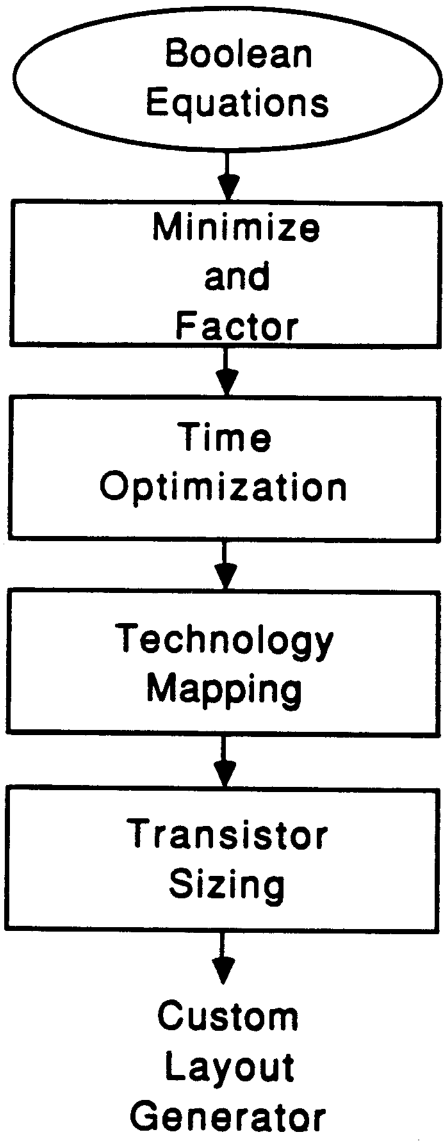


Figure 6: Algorithm for Layout Driven Synthesis

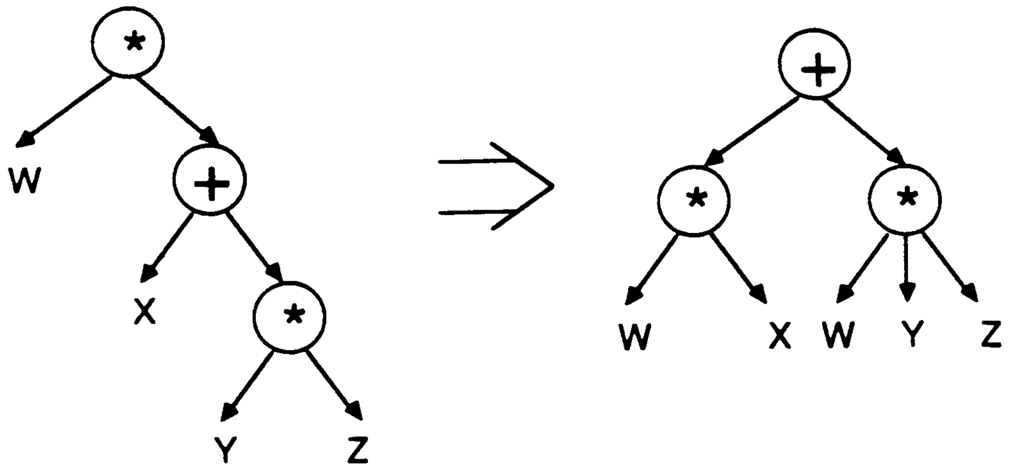
longest paths by performing balanced factoring. This technique attempts to partially collapse the critical path then refactor so as not to exceed the maximum number of transistors allowed per gate. The algorithm's third phase then performs technology mapping by combining gates into complex gates. The fourth phase sizes the transistors, producing a design that can be passed to a custom layout generator.

4.1. Time Optimization

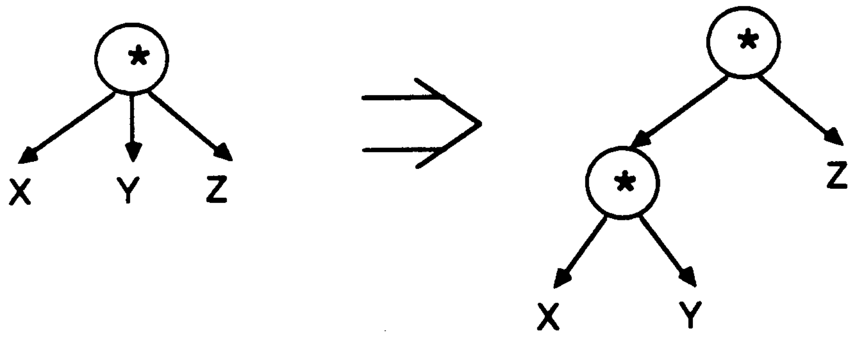
The general idea of timing optimization is to perform a limited collapse of nodes along the critical path. This requires some duplication of logic, creating a design with greater breadth and shorter depth. Some logic along critical paths can be removed and added along non-critical paths. More than just a partial collapse is required, however, to produce high-performance designs. We employ a balanced factoring method that addresses the problem of how many transistors to place in each gate. Gates are factored to get the number of inputs per gate that achieves the best speed rather than attempting to reduce the number of transistors.

Three operations are used to reduce the number of levels: distribute, extract, and merge. These operations are shown as applied to a graph of operator nodes in Figure 7. Distribution transfers logic across other nodes, allowing some logic to be shifted to non-critical paths. Extraction removes non-critical inputs from a node, reducing the delay through the node. Merging combines two nodes having the same operator. Only two *critical* nodes should be merged in order to reduce the delay. Use of the three operations is illustrated in Figure 8. Figure 8(a) is a design of five levels. Distributing node 2 over node 3 produces the design of Figure 8(b) which has one fewer level. Using the extract operation then on node 4 balances out the paths through that node (Figure 8(c)). Node 2 can be merged into node 0 (Figure 8(d)) and an extract operation applied to node 0 (Figure 8(e)) to balance the path lengths. In this manner, critical paths can be shortened by shifting logic to non-critical paths.

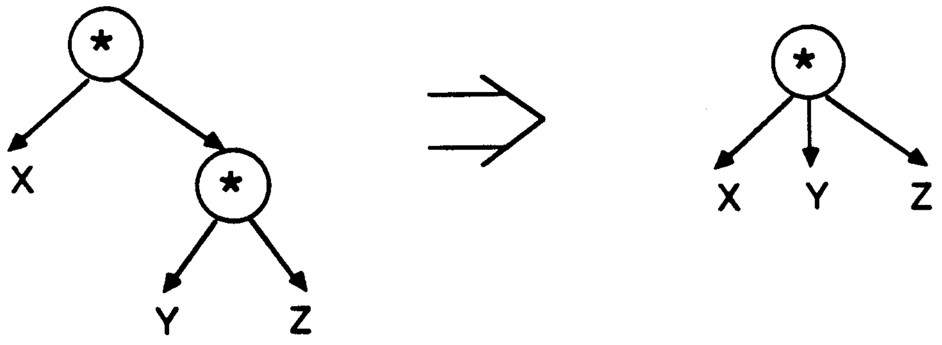
The input equations which have been minimized and factored are converted into a tree structure similar to that in Figure 8. Nodes may have only as many inputs as the number of transistors allowed in series. If performing the optimization at a node creates a new longest path or requires the addition of too many nodes, the optimization will not be performed. We perform the timing optimization going from inputs to outputs. Duplication of logic through distribute operations is preferred closer to input pins as there are fewer gates to be duplicated. Also transistor sizes are smaller closer to the inputs and duplicated logic can be placed in complex gates with more transistors. These factors help to contain the extra area that results.



(a) Distribute Operation



(b) Extract Operation



(c) Merge Operation

Figure 7: Critical Path Reduction Operations

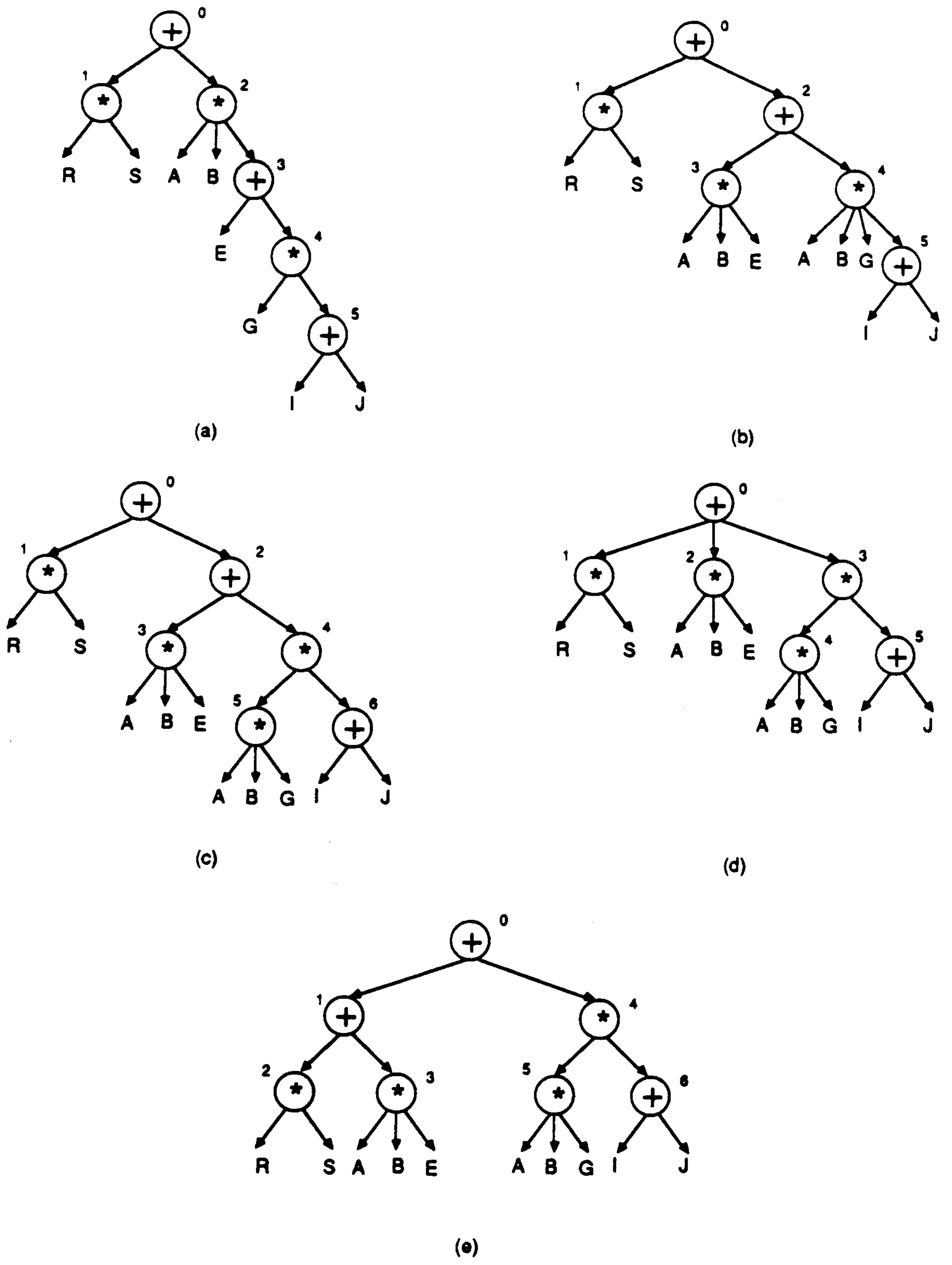


Figure 8: Example of Shortening Critical Path

Algorithm 1: Balanced Factoring

```
{restructure critical paths}
Let
  V be the set of vertices in the design;
  CP be the set of vertices along the critical path;

procedure balanced_factor(V)
begin
  $CP_o$ = find_critical_path(V);
  can_improve = TRUE;
  while (can_improve)
    balance(V, $CP_o$)
    $CP_n$ = find_critical_path(V);
    if ($CP_o = CP_n$) then
      can_improve = FALSE
    else
      $CP_o = CP_n$;
    end
  end
end
```

```
{Redistribute Logic Along the Critical Path}
Let
  $D_{[i,V]}$ be the cumulative delay at node i of design V;
  $D_{accept}$ be a delay reduction constraint;
  $AN_{[i]}$ be the increase in number of nodes due to optimization of node i;
  $A_{accept}$ be an area increase constraint;

procedure balance(V, CP)
begin
  for all i $\in$ CP
    $V_{new}$ = distribute(V, i, i+1);
    $V_{new}$ = extract($V_{new}$, i+1);
    $V_{new}$ = merge($V_{new}$, i+1, i-1);
    $V_{new}$ = extract($V_{new}$, i-1);
    if ($D_{[i,V]} - D_{[i-1,Vnew]} > D_{accept}$) and ($AN_i < A$) then
```

```
        V = $V_{new}$;
      **end**
    **end**
  **end**
```

```
  {Distribute node i over node i+1}
  Let
    $S_i$ be the set of input nodes to node i;

  **procedure** distribute(V, i, c)
  **begin**
    remove c from $S_i$
    **For all** j $\in S_c$
      **if** operator(i) = operator(j) **then**
        **For all** k $\in S_i$
          l = duplicate_tree(k);
          add l to $S_j$;
        **end**
      extract(V, j);
    **else**
      m = duplicate_tree(i);
      remove j from $S_c$;
      add m to $S_c$;
      add j to $S_m$;
      extract(V, m);
    **end**
  **end**
  add c to $S_{i-1}$;
**end**
```

```
{Remove Non-Critical Inputs From Critical Nodes}
Let
  $CD_i$ be the cumulative delay at the critical input of node i;
  $S_i$ be the set of inputs of node i;
  $T_c$ be the transistor limit constraint
  P be the delay percentage
```

```
procedure extract(V, i)
begin
  input_cnt = $T_c$;
  sort $S_i$ by delay (worst delay to least delay)
  if (length($S_i$) > 1) then
    k = node_duplicate(i);
  end
  if (length($S_i$) > $T_c$) then
    For all j $\in S_i$
      if (input_cnt = $T_c$ and length($S_i$) > 1) then
        n = node_duplicate(i);
        add n to $S_k$;
        k = n;
        input_cnt = 1
      else
        input_cnt = input_cnt + 1;
      end
      add j to $S_k$;
      remove j from $S_i$;
    end
  end
  num_inputs = 0;
  $T_i = S_i$;
  k = node_duplicate(i);
  For all j $\in S_i$
    if $D_j < (P * CD_i)$ then
      add j to $S_k$;
      remove j from $S_i$;
      num_inputs = num_inputs + 1;
    end
  end
  if (num_inputs > 1) then
    add k to $S_i$;
  else
    $S_i = T_i$;
  end
end
```

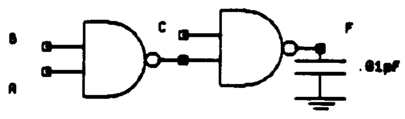
```
{Combine Critical Nodes}
procedure merge(V, $N_i$, $N_{i-1}$)
begin
  if (operator($N_i$) = operator($N_{i-1}$))
    For all j $\in N_i$
      remove j from $N_i$;
      add j to $N_{i-1}$;
    end
  end
end
```

■

4.2. Technology Mapping

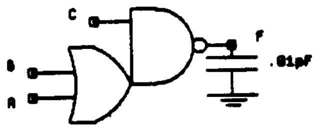
The third phase involves technology mapping. To perform the mapping, estimates of the gate delay times are made in order to identify the critical path. Each gate is converted to a NAND/NOR gate and assigned a delay based upon the number of NFETs in series, the number of PFETs in series, and the load that the gate must drive. The critical path is found using a method similar to that discussed in [YeGh88].

The next step is complex gate formation. The algorithm forms complex gates first along the critical path. The combining algorithm goes from input pins to output pins, which tends to produce large complex gates near the inputs and small complex gates at the outputs (as there are fewer gates left to combine). Complex gate formation is restricted by the user entered parameters for maximum transistors in series, the number of fanouts that the gate must drive, and the amount of slack. We have found that single gates or complex gates in which the critical path passes through only one gate level of the complex gate are best along the critical path. Figure 9 shows an example demonstrating this with actual delay times in a 3 micron CMOS technology. The complex gate formed in Figure 9(b) has only a small effect on the delay through the critical path. However, the complex gate achieves a reduction of active transistor area (routing area is not included). Figure 9 also shows that if the critical path ran from A to F, the complex gate should not be formed. For purposes of comparison in this example, it is assumed that all inverters that must be added to some inputs of the complex gate (for the designs of Figure 9(a) and Figure 9(b) to be equivalent) are pushed back to the previous stages.



Path	Rise Time (ns)	Fall time (ns)	Avg. Time (ns)
A->F	8.3	7.7	8.0
C->F	5.5	4.0	4.75

(a) Active Transistor Area: 96um^2



Path	Rise Time (ns)	Fall time (ns)	Avg. Time (ns)
A->F	15.6	5.0	10.3
C->F	3.4	5.0	4.2

(b) Active Transistor Area: 72um^2

All transistors have size $12um^2$

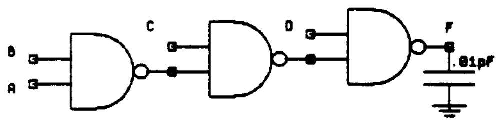
Figure 9: Area/Delay Considerations for 2-Level Complex Gates

Gates along the non-critical paths are combined into complex gates in the fourth phase. The worst case path to each output is processed first to prevent the initial combining from being along the short paths. Figure 10 shows the delay and area for a complex gate with more transistors than Figure 9. It demonstrates that complex gates with more transistors are slower for some paths, but yield a savings in transistor area. Note that the complex gate of Figure 10(b) could be made just as fast as its individual gate implementation. This would require larger transistor sizes, however, and hence the complex gate consumes a larger layout area in order to match the delay. Therefore, paths with much smaller delays than the critical path can have complex gates with more transistors.

Algorithm 2: Complex Gate Generation

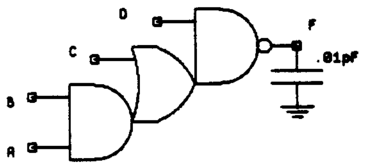
```
{combining gates into complex gates}
Let t be sink node and s be source node in the graph;
   Ci be the input capacitance of vertex i associated with all the fanout pins;
   Si be the slack of vertex i (required delay - actual delay);
   C_high and C_low be the fanout constraints;
   S_small and S_large be the slack constraints;
   Q be a set of vertex.

PROCEDURE combining(V,Q)
BEGIN
  FOR (i = t to s in Q)
  BEGIN
   IF (mark[i]=false)
   BEGIN
```



Path	Rise Time (ns)	Fall time (ns)	Avg. Time (ns)
A->F	12.0	10.5	11.25
C->F	8.3	7.7	8.0
D->F	5.5	4.0	4.75

(a) Active Transistor Area: $144um^2$



Path	Rise Time (ns)	Fall time (ns)	Avg. Time (ns)
A->F	15.44	8.56	12.00
C->F	11.11	3.56	7.33
D->F	5.11	3.33	4.22

(b) Active Transistor Area: $96um^2$

All transistors have size $12um^2$

Figure 10: Area/Delay Considerations for 3-Level Complex Gates

```
  IF (Ci > C_high OR Si < S_small)
   combine gates into two level three input complex gate;
  ELSE IF (Ci > C_low OR Si < S_large)
       combine gates into two level four input complex gate;
      ELSE
         combine as many gates as possible into complex gate;
  mark[i] = true;{where i is in V}
 END;
END;
```

■

4.3. Transistor Sizing

The fourth phase is transistor sizing. Transistor sizing is performed first on the critical path to ensure optimal sizes for speed. Transistors in gates along non-critical paths can then be sized. The N and P transistor sizes are chosen to achieve equal rise and fall times for the gate. The sizing algorithm proceeds from outputs to inputs basing size upon the capacitive load that must be driven. To achieve nearly optimal sizes for speed we examine the gate to be sized and its preceding stage. The graph of a gate's transistor size plotted against total path delay is a convex curve [He87]. We examine the effect of sizing on both gates, thus considering two curves. The point where these two curves intersect is the transistor size chosen. Further details of the transistor sizing algorithm can be found in [WuVG89].

Algorithm 3: Combine-Then-Size Strategy

```
{This algorithm is to combine gates into complex gates, and then to size the transistors to obtain
optimum speed}

Let V,E be the vertex and edge sets of the graph G;
   where V is the set of all gates in G;
   and E is the set of all connections between gates in G;
Let Vcritical be the vertex set of the critical path;

BEGIN
  restructure_longest_path(V,Vcritical);
  find_critical_path(V,Vcritical);

  {combine gates along critical path}
  combining(V,Vcritical);
  {combining gates along non-critical paths}
  combining(V,V);

  {size transistors along critical path}
  transistor_sizing(Vcritical,V);
  {size transistors along non-critical paths}
  transistor_sizing(V,V);
END
```
■

5. Results

Our synthesis tool is currently running on SUN 3 workstations under the UNIX operating system. Synthesized designs with complex gates and sized transistors are passed to LES for layout generation and then to GDT[1] [BuMa85] for simulation and comparison. We have run a number of examples and compared our results with those of MISII (OCTTOOLS release 2.0). They are shown in Table 1. To achieve a fair comparison, the output of MISII (which does not size transistors) was run through our transistor sizing routine before passing it on to LES. The LES layout was passed on to GDT to perform the simulation. The results for our layout driven synthesis algorithm (LDS) were obtained by first minimizing and factoring the design using MISII, then applying the timing optimization, complex gate formation, and transistor sizing before passing the circuit to LES and GDT. Tables 2 and 3 displays a number of MCNC benchmark examples with comparisons to MISII (OCTTOOLS release 3.1).

[1] GDT is a registered trademark of Silicon Compiler Systems.

Design	Complexity	Area (um^2)			Time (ns)		
	# Gates	MISII	LDS	% improvement	MISII	LDS	% improvement
P147	18	72,341	62,129	14.1	11.84	11.26	4.9
P191	17	62,304	62,708	-0.8	16.55	11.33	31.5
F1	20	76,903	51,000	33.7	17.11	11.50	32.8
F2	29	128,040	81,510	36.3	21.11	12.61	40.3
z4mlc	64	330,038	270,150	18.1	26.28	12.56	52.2

Table 1: Custom Layout Comparison Between MISII the LDS Algorithm

Table 2 compares custom layout results, Table 3 was generated using the MCNC standard cell library. Because of the size of these designs, we did not obtain actual layout results but used estimates for time (which we have found to be within ±10% of the actual value) and the active transistor area (not including the routing area). The following commands were used to perform the logic synthesis in MISII.

```
source script
rlib les_cus.lib
map -m.75
phase -g
speed_up -w 0
map -m.75
```

The script used is the standard script provided with OCTTOOLS release 2.0.

Design	Active Transistor Area (um^2)			Time (ns)		
	MISII	LDS	% increase	MISII	LDS	% improvement
9symml	16,848	16,884	0.2	45.03	41.61	7.7
z4ml	5,340	6,636	24.2	20.16	16.69	17.2
b9	7,764	14,820	91.8	25.87	20.68	20.1
f51m-hdl	8,676	12,168	40.2	33.35	26.30	21.1
f51m	10,944	14,184	29.6	29.65	25.18	15.1
att12	6,564	11,040	68.2	45.95	23.76	48.3

Table 2: Comparison of Custom Layouts Using MCNC Benchmarks

Design	Area			Time		
	MISII	LDS	% increase	MISII	LDS	% improvement
9symml	382	465	21.7	19.6	15.4	21.4
z4ml	119	116	-2.5	10.1	6.6	34.7
b9	237	338	42.6	8.9	8.4	5.6
f51m-hdl	216	284	31.5	10.8	10.7	0.9
f51m	269	295	9.7	11.1	11.3	-1.7
att12	218	256	17.4	14.4	11.6	19.4

Table 3: Comparison of Standard Cell Layouts Using MCNC Benchmarks

We found that it produced superior results for timing than the standard script in OCTTOOLS release 3.1. The technology file les_cus.lib contains delay and area estimations for gates in the SCMOS 3 micron technology. Our algorithm also uses this table to perform delay estimations. Generally synthesis with layout constraints produced designs up to 48 percent faster with an average of 30 percent more area. The area for our designs could be reduced by performing area optimizations along the non-critical paths. Currently no area optimizations are performed. Tables 2 and 3 demonstrate that our algorithm improves performance for both standard cell and custom layout designs.

Figure 11 displays a composite graph showing our ability to perform area/time tradeoffs. Our results are normalized against standard cells whose reference point is shown at time = 1, area = 1. Several points are noted on the graph. Point A shows that synthesis considering layout produces designs with smaller area and faster speed. Point B illustrates the capability to save even more area while having a larger delay than the standard cell synthesis approach. Similarly, Point C shows that speed can be improved further at the expense of area. The dashed section of the curve ends when the minimal transistor sizes are used (PFET size = 2, NFET size = 1). This part of the curve shows expected results as we have not actually tested this. Another observation is that using custom layout, the speed can be kept speed constant while varying the output load (fanout). Of course, there is an associated increase in the area. As

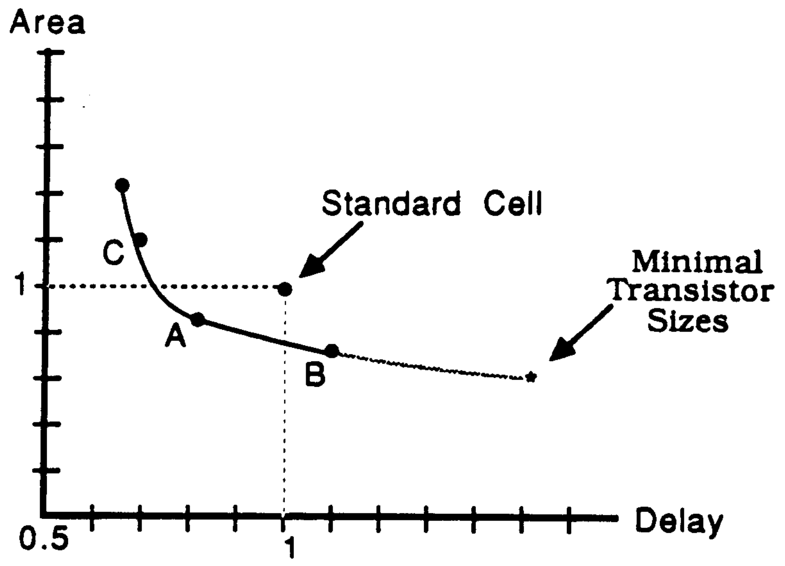


Figure 11: Composite Graph of Experimental Results

the fanout increases in standard cells, however, the speed is decreased.

6. Conclusion

We have noted that higher-quality layouts can be produced by taking layout parameters into account during the synthesis process. Traditional logic synthesis techniques work well for layouts using standard cells but achieve less than optimal results for custom layouts. They fail to consider layout parameters like transistor sizing and complex gate formation. We implemented an algorithm for high-performance CMOS designs that incorporates these parameters and compared our results with those of a traditional logic synthesis system, MISII. Our results demonstrate speed improvements for both custom and standard cell layout Further, layout driven synthesis has greater control over area/time tradeoffs. Future research is needed to refine the interaction of factorization, complex gate formation, and transistor sizing to produce superior results. Improvements should include: (a) re-examining timing after the initial transistor sizing phase and desizing transistors along non-critical paths to reduce area and power, (b) using the combine-and-size strategy instead of our combine-then-size strategy, and (c) performing area optimization along non-critical paths.

7. Acknowledgements This work was supported in part by Applied Micro Circuits Corporation, Magnavox, NSF under grant #MIP-8711025, and the State of California MICRO program. We are grateful for their support.

8. References

[BaAl88] Baltus, D.G., and Allen, J., "SOLO: A Generator of Efficient Layouts from Optimized MOS Circuit Schematics", 25*th DAC*, 1988.

[BeOw87] Beekman, J.A., Owens, R.M., and Irwin, M.J., "Mesh Arrays and LOGICIAN: A tool for their Efficient Generation", 24*th DAC*, 1987.

[BoHa87] Bostick, D., Hachtel, G.D., Jacoby, R., Lightner, M.R., Moceyunas, P., Morrison, C.R., and Ravenscroft, D., "The Boulder Optimal Logic Design System", *ICCAD*, 1987.

[BrRu87] Brayton, R., Rudell, R., Sangiovanni-Vincentelli, and Wang, A., "MIS: A Multiple-Level Logic Optimization System", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-6, No. 6, Nov. 1987.

[BuMa85] Buric, M.R., and Matheson, T.G., "Silicon Compilation Environments", *Proceedings of the Custom Integrated Circuits Conference*, May, 1985.

[Ci87] Cirit, Mehmet A., "Transistor Sizing in CMOS Circuits", 24*th DAC*, 1987.

[FiDu85] Fishburn, J. P., and Dunlop, A. E., "TILOS: A Posynomial Programming Approach to Transistor Sizing", *ICCAD*, 1985.

[GrBa86] Gregory, D., Bartlett, K., de Geus, A., and Hachtel, G., "SOCRATES: A System for Automatically Synthesizing and Optimizing Combinational

Logic", 23*rd* *DAC*, 1986.

[He87] Hedlund, Kye S., "Aesop: A Tool for Automated Transistor Sizing", 24*th* *DAC*, 1987.

[JoTr86] Joyner, W., Trevillyan, Y., Brand, D., Nix, T., and Gundersen, S., "Technology Adaptation in Logic Synthesis", 23*rd* *DAC*, 1986.

[Ke87] Keutzer, K., "DAGON: Technology Binding and Local Optimization by DAG Matching", 24*th* *DAC*, 1987.

[Kim87] Kim, J., "Artificial Intelligence helps cut ASIC Design Time", *Electronic Design*, June 11, 1987.

[Kol85] Kollaritsch, P.W., and Weste, N.H., "TOPOLOGIZER: An Expert System Translator of Transistor Connectivity to Symbolic Cell Layout", *IEEE J. of Solid-State Circuits*, vol. SC-20, no. 3, June 1985.

[KoLu88] Kollaritsch, P., Lusky, S., Prasad, S., and Potter, N., "CLAY: A Malleable-cell Transistor Matrix Approach for CMOS Layout Synthesis", *ICCAD*, 1988.

[LiGa87] Lin, Y-L. Steve, and Gajski, D., "LES: A Layout Expert System", 24*th* *DAC*, 1987.

[ObKa88] Obermeier, Fred W., and Katz, Randy H., "An Electrical Optimizer that Considers Physical Layout", 25*th* *DAC*, 1988.

[TsCe88] Tsareff, C., Cesear, T.M., and Iodice, E., "An Expert System Approach to Parameterized Module Synthesis", *IEEE Circuits and Devices Magazine*, January, 1988.

[VaGa88] Vander Zanden, N., Gajski, D., "MILO: A Microarchitecture and Logic Optimizer", 25*th* *DAC*, 1988.

[WuVG89] Wu, C.H. Allen, Vander Zanden, N., Gajski, D., "An Algorithm for Transistor Sizing in CMOS Circuits", Technical Report #89-04, University of California at Irvine, 1989.

[YeGh88] Yen, H.C., Ghanta, S., and Du, H.C., "A Path Selection Algorithm for Timing Analysis", 25*th* *DAC*, 1988.