An Algorithm for Transistor Sizing in CMOS Circuits

by

Allen C. H. Wu
Nels Vander Zanden
Daniel Gajski

Technical Report 89-04

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

Abstract

This paper describes a novel algorithm for automatic transistor sizing which is one technique for improving timing performance in CMOS circuits. The sizing algorithm is used to minimize area and power subject to timing constraints. We define the transistor sizing problem as a graph problem and use a non-linear optimization technique. The algorithm consists of three separate tasks: critical path analysis, transistor sizing and transistor desizing. The main contribution of the presented algorithm is that the delays of all paths in a given design can be tuned simultaneously to satisfy timing constraints. Furthermore, the minimal transistor area and minimal power dissipation under given timing constraints can be achieved. Experimental results show that this approach has greater control over area/time tradeoffs than traditional sizing algorithms.

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

The task of chip designers is to design a circuit that satisfies both functional requirements and performance constraints. Properly defining transistor size is one technique to improve timing performance. The relationship between transistor sizes and total circuit delay is non-linear. It has been shown by Fishburn and Dunlop [FiDu85] that the transistor sizing problem is convex under the simple lumped RC model. An example of the effect of transistor sizing on delay is shown in Figure 1. Since delay is proportional to gate resistance and load capacitance, increasing transistor size of gate B reduces the resistance and delay of gate B. However, increasing transistor size of gate B also increases the capacitive load(C_b) as well as the delay of gate A. This is a delay balance effect between gate A and gate B. By increasing the transistor size of gate B, the total path delay will decrease until reaching the nadir of the convex (balance point) which is the minimum delay point. The total path delay increases by increasing the transistor size of gate B beyond the balance point. Thus, the objective of the transistor sizing algorithm is to size transistors by finding the optimal point on the convex that satisfies the timing constraints with minimal transistor area.

We formulate the transistor sizing problem into a graph problem using a non-linear optimization technique. This algorithm contains a critical path

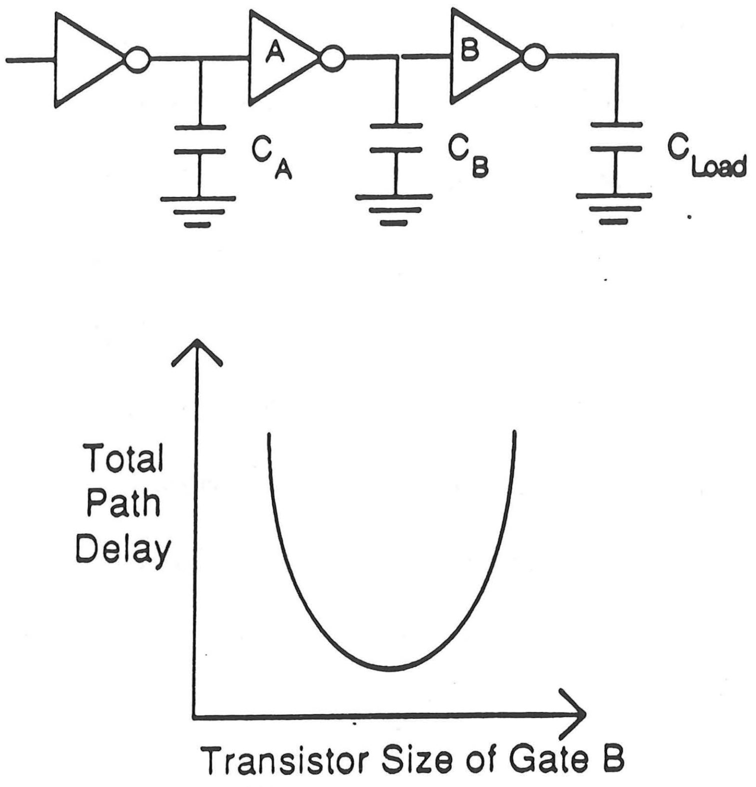

Figure 1 The effect of transistor sizing on delay

analyzer that derives a multistage graph from a design, and then locates the worst critical path using a dynamic programming method. Using a convex optimization technique, the algorithm sizes the PFET and NFET separately. The sizing algorithm tunes the transistors along the worst critical path to satisfy the timing constraint, then sizes the remaining transistors in the design to the optimum values. The sizing algorithm initially tries to over-size the transistors in the design to ensure that all timing constraints are satisfied. Using this method, the sizing algorithm improves the delays of all paths at the same time, and hence does not need to check whether some new path becomes critical. A desizing algorithm then calculates the actual delay of each gate and the delay allowance based on the timing constraints, and then desizes the transistors to minimize the area. The objective of this sizing algorithm is to size transistors so that the minimal transistor area is achieved for the specified timing contraints, and the delays of all paths in a design can be reduced to satisfy timing constraint simultaneously.

The next section decribes some previous work. Section 3 presents the electrical models used to calculate the delay of a circuit and summarizes the optimization algorithms. The experimentla results and conclusions described are in Section 4 and 5.

2.Previous Work

Three approaches have been applied to solve the transistor sizing optimization problem. The first approach[FiDu85][KaFa85] is a heuristic method in which the transistor size is incremented with a step size until all the timing constraints are satisfied. Fishburn and Dunlop[FiDu85] have shown that the transistor sizing problem is convex by using a distributed RC delay model. TILOS is a heuristic transistor sizing program that iteratively increases the transistor size along the worst critical path. This heuristic selects a transistor that reduces the most delay with the minimal increase in area. The process continues until all the timing constraints are satisfied. Thus TILOS minimizes power and area subject to the timing constraints.

XTRAS[KaFa85] used a simple heuristic method to minimize the delay of a circuit. A gate selection routine selects the gate that contributes the most to the delay. Then XTRAS increases the size of selected gate and recalculates the total path delay. This process terminates when the timing requirement is met.

The second approach[Hedl87][ShFi87] is based on the conventional augmented Lagrangian method[Gill81] which requires gradients to be computed for each variable and requires the cost function to be differentiable. The cost function is formulated as a polynomial by modeling the delay as a lumped RC time constant. AESOP[Hedl87] is an interactive transistor sizing tool. It formulates the delay minimization problem into a nonlinear optimization problem with constraints. The nonlinear optimization problem is solved by a quasi-Newton method, and several paths can be solved simultaneously. Using

ASEOP, the user can select sets of paths to be optimized, specify constraints, and interactively evaluate different design and sizing options.

The third approach[PiDe86] uses a simulated annealing method for reducing the delay on many paths simultaneously. MOST is a Prolog program that uses a simulated annealing algorithm to reduce the delay on many paths simultaneously. The implementation makes great use of the delayed binding and backtracking techniques in Prolog. This allows binding the parameters of many paths at same time, thus the delays of many paths can be reduced simultaneously. But the lack of tail-recursion optimization in Prolog limit the maximum circuit size to approximately 100 transistors.

The main problem of the first approach is that only one path(the worst critical path) is optimized at a time. This approach lacks a global view of the whole circuit. Some new paths may become critical after changing the transistor sizes along the worst critical path. Therefore, the algorithm may fail to size the circuit correctly. Using the simulated annealing method, the delays of all paths can be reduced simultaneously. However, it suffers from long execution time and is therefore restricted to small circuits only.

The drawbacks of the second approach are: (1) It solves the problem in an unnecessarily large space by using all the transistors as design variables, and (2)

It lacks an efficient way to get a good initial guess of transistor sizes.

All the three approaches size the logic gates instead of individual transistors; this may produce the problem of non-symmetrical rise time and fall time delays.

3. Models and Algorithms

3.1 Overview

This section describes the models used to estimate the gate delay, and the transistor sizing algorithms for timing optimization in MOS circuits.

The algorithm for transistor sizing consists of three main sections : Critical path analysis, Transistor sizing, and Transistor desizing. The critical path analyzer converts the given design to a multistage graph, and then determines the worst critical path using a dynamic programming method. The transistor sizer optimizes the transistor sizes along the worst critical path to meet timing constraints, and then modifies the rest of the transistors to optimum sizes using convex optimization method.

The transistor desizer then reduces transistor sizes of all paths to minimize the transistor area subject to timing constraints. The transistor desizing alogorithm consists of two phases: delay estimation and transistor desizing. The desizing algorithm estimates the delay allowance of each gate subject to timing constraints, and then desizes the transistors of all gates to the minimal size that satisfies the timing constraints.

3.2 Electrical Models

3.2.1 RC Delay Model

To estimate gate delay, a simple RC delay model[Hedl84] is used to compute the resistance and capacitance of each gate in a path. When critical path analyzer traces a signal path, each transistor in the path is modeled as a fixed resistor driving some output capacitance(Figure 2). There are two transition states in each gate; when the output changes from "1" to "0"(Figure 2(a)) and when the output changes from "0" to "1"(Figure 2(b)). The change in the output is assumed to be triggered by a single input, and all transistors in series with the trigger transistor must be turned on.

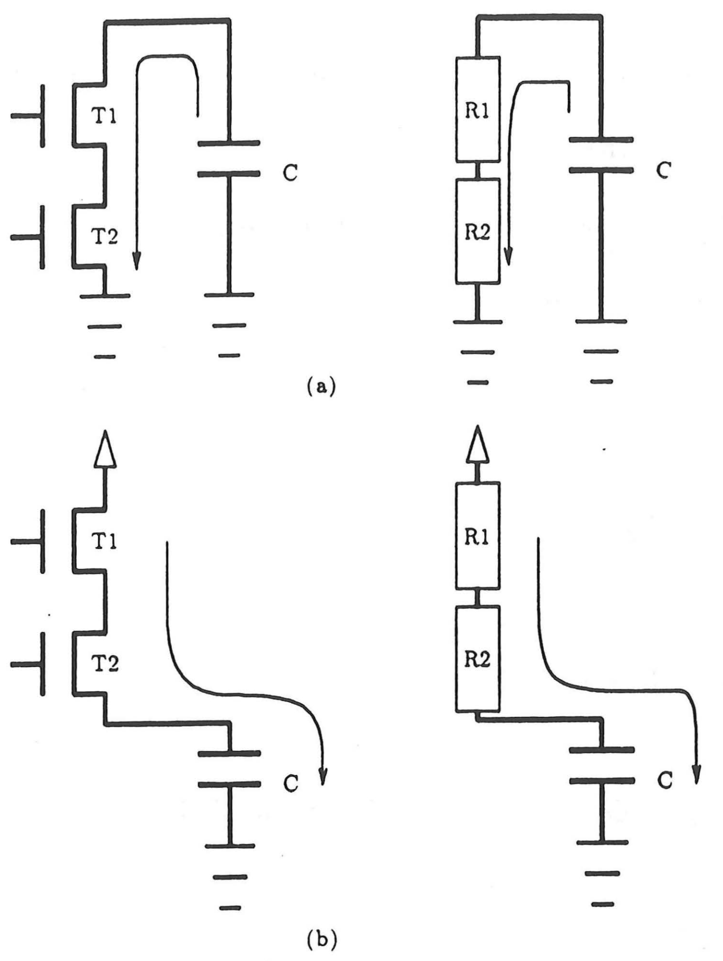

Figure 2 Delay through pullup and pulldown network with RC model

The delay of a gate i is simply the average of the rise and fall time delays :

$$t_{(i)} = (t_{rise(i)} + t_{fall(i)}) \ / \ 2$$

The rise time and fall time delays[Hedl84] are computed as follows(Figure 3):

$$t_{rise(i)} = (R_{pup(i)}/S_{pup(i)} + R_{w(i)}) * C_{(i)}$$

$$t_{fall(i)} = (R_{pdn(i)}/S_{pdn(i)} + R_{w(i)}) * C_{(i)}$$

$$C_{(i)} = C_{pup(i+1)} {}^{*} S_{pup(i+1)} + C_{pdn(i+1)} {}^{*} S_{pdn(i+1)} + C_{w(i)}$$

where

$R_{pup(i)}$ is the sum of the pullup transistor resistances of gate i in series if the output is "1".

$R_{pdn(i)}$ is the sum of the pulldown transistor resistances of gate i in series if the output is "0".

$R_{w(i)}$ is the parasitic wire resistance of gate i.

$C_{(i)}$ is the total capacitive load of gate i.

$C_{pup(i+1)}$ is the PFET gate capacitance of gate i+1.

$C_{pdn(i+1)}$ is the NFET gate capacitance of gate i+1.

$C_{w(i)}$ is the parasitic wire capacitance of gate i.

$S_{pup(i)}$ is the PFET size of gate i.

$S_{pdn(i)}$ is the NFET size of gate i.

The resistance and capacitance of a single pullup and pulldown transistor depend only on the type and width of the transistor. The resistances $R_{pup(i)}$ and $R_{pdn(i)}$ are inversely proportional to the sizes of pullup and pulldown transistors of

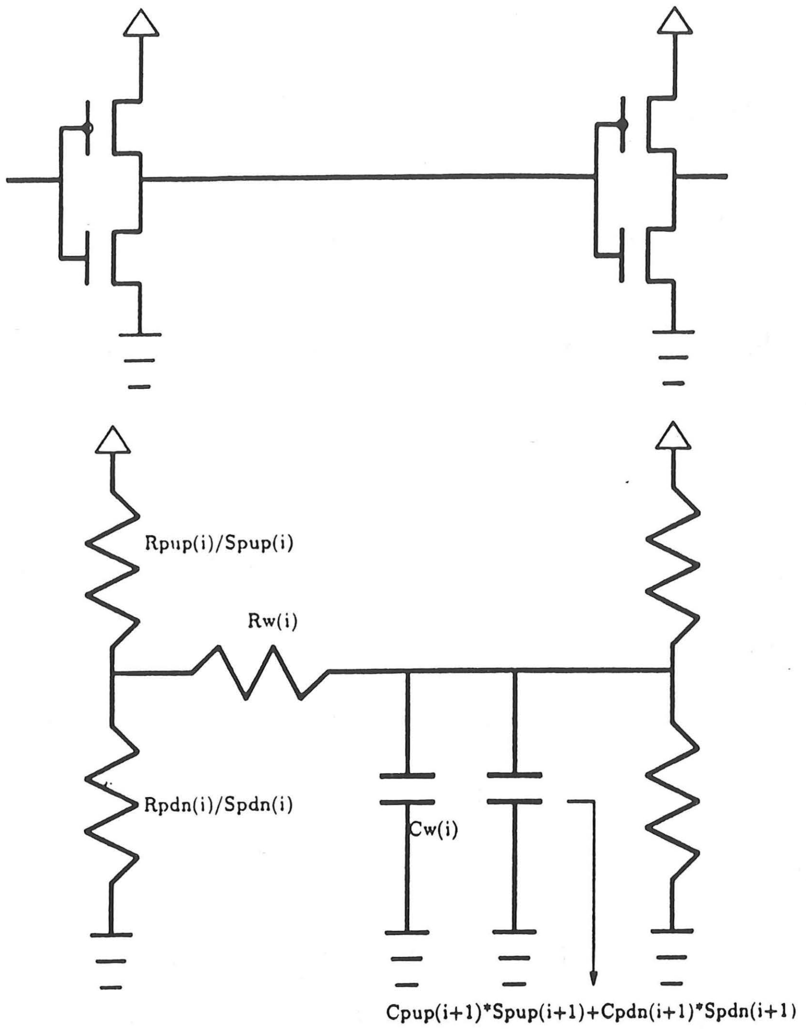

Figure 3 Electrical Model

gate i. The capacitance $C_{pup(i)}$ and $C_{pdn(i)}$ are directly proportional to the sizes of pullup and pulldown transistors of gate i. Therefore, increasing $S_{pup(i)}$ and $S_{pdn(i)}$ decreases the output delay of gate i but increases the capacitive load on gate i-1.

The total delay T through a path is computed by summing up the individual gate delsys in the path. Hence

$$T = \sum t_i$$

where the delay t_i is the individual gate delay in the path.

Our transistor sizing algorithm is implemented before layout takes place. This means that the exact wire lengths for routing are unknown, and some estimates of wires resistances and capacitances must be used. To maintain technology independence, and to reduce the execution time, we used a table driven approach [Hedl87] in which the values of resistance and capacitance are supplied by the user in a table based on different technologies.

3.2.2 Delay Model for Complex Gate

The $\mathbf{R}_{pup}$ and $\mathbf{R}_{pdn}$ of a complex gate are computed based on the longest resistance path along the trigger transistor. For a three input OAI (Figure 4(a)), the effective resistance $\mathbf{R}_{pup}$ is $\mathbf{R}_a+\mathbf{R}_b$ if the trigger transistor is A or B; but the effective resistance $\mathbf{R}_{pup}$ is only $\mathbf{R}_c$ if the trigger transistor is C. For a three input AOI (Figure 4(b)), the effective resistance $\mathbf{R}_{pdn}$ is $\mathbf{R}_a+\mathbf{R}_b$ if the trigger transistor is A or B, and the the effective resistance $\mathbf{R}_{pdn}$ is $\mathbf{R}_c$ if the trigger transistor is C. Thus the delay calculation of a complex gate depends on the input trigger transistor.

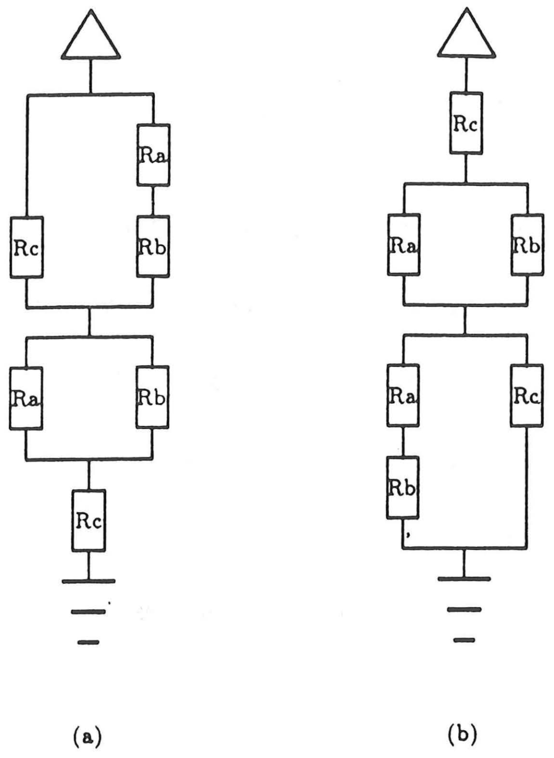

Figure 4 Delay model for complex gate

3.3 Critical Path Analysis

Critical path analysis[KiCl66] was the first approach proposed for timing analysis. Several authors ([Oust85], [Hitc82],[Joup83],[YeGh88]) have reported many different approachs for performing timing analysis. In this section, we decribe an algorithm using a dynamic programming method[HorSa78] to determine the worst critical path.

A multistage directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ is derived from the given design. Each vertex **V** represents a gate in the design. An edge **E** represents the path between two gates. The vertices are partitioned into $k \geq 2$ disjoint stages $\mathbf{V}_i$, $1 \leq i \leq k$ (Figure 5). Let s and t be two vertices in $\mathbf{V}_1$ and $\mathbf{V}_k$ respectively. Dummy source (s) and sink (t) vertices are added for simplicity. Let **delay**(i,j) be the delay of path<i,j>. Let $\mathbf{DELAY}_{total}(s,t)$ be the sum of the delays from s to t. The task of critical path analyser is to find the path with the maximum delay from s to t.

This maximum delay for a k-stage graph problem is computed (using dynamic programming) by starting at the sink node and moving towards the source node one stage at a time. The delay computation depends on two adjacent sets of vertices. To find a maximum delay path from t to s, k - 2 decision sequences may be generated. However, those sequences containing suboptimal distances will not be generated. For example, if the delay of the sequence (G1 -> G3) is larger than the delay of the sequence (G1 -> G2) then the sequence (G1 -> G2) contains a suboptimal distance, and therefore will not be stored for further analysis (Figure 5). Let $\mathbf{DELAY}_{total}(i,j)$ be the maximum delay path from vertex j in $\mathbf{V}_i$ to sink node t. By working backward, we obtain:

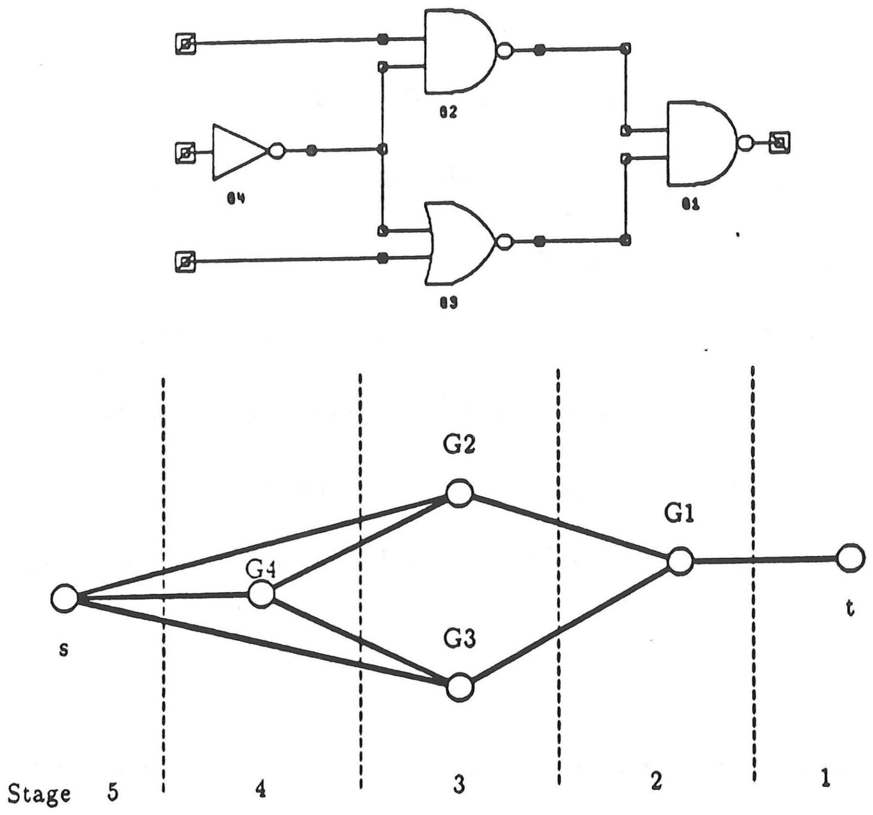

Figure 5 Multistage graph representation

$$\mathbf{DELAY}_{total}(i,j) = \mathrm{MAX}\{\mathbf{delay}(j,m) + \mathbf{DELAY}_{total}(i\text{-}1,m)\}$$

where

$$m \in V_{i-1} \text{ and } (j,m) \in \mathbf{E}$$

The critical path analysis alogrithm consists of the following four phases:

1. Calculate the delay for each vertex initially using unit size.

2. Partition vertices into disjoint sets.

3. Insert the source node, sink node and dummy edges.

4. Determine the worst critical path.

ALGORITHM 1 : Critical path analysis

```
{calculate the delay for each vertex}
Let
  #v be the number of vertices in V;
  $t_{delay}$[i] be the delay time at vertex i;
  $C_{load}$[i] be the output capacitance of vertex i associated with all the fan out pins;
  $R_{peff}$[i] be the sum of the pullup transistor resistance in series at vertex i plus wire
  resistance;
  $R_{neff}$[i] be the sum of the pulldown transistor resistance in series at vertex i plus wire
  resistance;
  $S_{pfet}$[i] be the PFET size of vertex i;
  $S_{nfet}$[i] be the NFET size of vertex i.

procedure gate_delay_calculation(V)
begin
for (i=1 to #v)
  $t_{delay}[i] = (R_{peff}[i] * C_{load}[i])/S_{pfet}[i] + (R_{neff}[i] * C_{load}[i])/S_{nfet}[i]$;
end;
```

```
{Assign vertex stage and determine the longest distance from output to input using
 breadth first search}
procedure graph_stage($D_{max}$=1,E,V)
Let
  Q be a set of vertices,
  d[i] be the stage level of vertex i.
begin
    Q <= v where v is output vertex;
    visited[v] = true;
    d[v] = $D_{max}$;
    while (not EMPTY(Q))
    begin
        v = FRONT(Q);
        DEQUEUE(Q);
        for (each vertex i in V connected to v)
        begin
            if (there are no vertices connected to v)
            ieaf[v] = true;
            if (visited[i] = false) then
            begin
                visited[i] = true;
                d[i] = d[v] + 1;
                if (d[i] > $D_{max}$)
                $D_{max}$ = d[i];
                ENQUEUE(i,Q);
            end;
        end;
    end;
end;


{Insert source node, sink node and dummy edges}
Let #v be the number of vertices in V.
procedure source_and_sink(V,E)
begin
    V <= V + s + t; {where s : source node and t : sink node}
    $t_{delay}$[s] = 0;
    d[s] = $D_{max}$ + 1;
    $t_{delay}$[t] = 0;
    d[t] = 0;
    for (i = 1 to #v)
    begin
        if (d[i] = 1)
            E <= E + <i,t>;
        else if (leaf[i] = true)
            E <= E + <s,i>;
    end;
end;
```

{Find the worst critical path using dynamic programming method}
Let
  i be the vertex stage number which $2 \leq i \leq D_{\max}$;
  D[i,j] be a maximum delay path from vertex j in v_i to sink vertex t and DELAY[i,j] be the delay time of the path;
  d[j,m] be the delay from vertex j to vertex m;

```
procedure critical_path(V, $V_{critical}$)
begin
    $V_{critical} \leq$ t where t is the sink vertex;
    DELAY[i,j] = max(d[j,m] + DELAY[i-1,m]);
    for (all the <s,x> in E)
    begin
        stage = d[x] - 1;
        DELAY[d[s],s] = max(d[s,m] + DELAY[stage,m]);
    end;
    for (i = 2 to $D_{\max}$)
    D[i,j] = max(d[j,m] + DELAY[i-1,m]);
    for (all the <s,x> in E)
    begin
        stage = d[x] - 1;
        D[s,x] = max(d[s,m] + DELAY[stage,m]);
    end;
    {Let critical path be s,$V_{Dmax}$,$V_{Dmax-1}$,...,$V_1$,t}
    Dx <= s;
    for (i = $D_{\max}$+1 to 1)
    begin
        $V_{critical}$ <= $V_{critical}$ + D[i,$D_x$];
        critical[D[i,$D_x$]] = true;
        $D_x$ = D[$D_{\max}$,$D_x$];
    end;
end;
```

3.4 Transistor Sizing Algorithm

As mentioned previously, the relationship between transistor sizes and delay is convex. The transistor sizing computation is based on two adjacent vertices. The capacitive load of the vertex to be sized is the sum of the total gate capacitances that the vertex drives. Using the convex optimization technique, the PFET and NFET sizes in a vertex are sized separately, depending on which transistor (PFET or NFET) contributes the most delay reduction. For instance, if the total delay reduction by increasing the PFET size of a vertex is larger than the total delay reduction by increasing the NFET size of a vertex, the algorithm increases the PFET size of this vertex. The transistor size computation terminates when the minimum delay is obtained. The optimizer first sizes the transistors in the worst critical path to satisfy the timing constraint based on the output capacitive load. Then the optimizer sizes rest of the transistors in the design from output to input to obtain the optimum values. The optimizer tries to over-size the transistors in the design to ensure that the delays of all paths satisfy the timing constraint.

ALGORITHM 2 : Transistor sizing

```
{transistor sizing algorithm}
Let
  Q be a set of vertex;
  $C_{punit}$ be the unit capacitance of PFET;
```

C_{nunit} be the unit capacitance of NFET;
t_{rise}[i] be the rise time delay of gate i;
t_{fall}[i] be the fall time delay of gate i;
t_{p+1} be the delay time after increasing the S_{pfet}[i] by 1;
t_{n+1} be the delay time after increasing the S_{nfet}[i] by 1.

```
procedure transistor_sizing(Q,V)
begin
    for (i = t to s in Q)
    begin
        if (already_sized[i] = false)
        begin
            {initialization}
            $t_{total} = t_{delay}[i] + t_{delay}[i-1];$
            $t_{p+1} = t_{total};$
            $t_{n+1} = t_{total};$
            while ($t_{total}$ is not minimum) {convex optimization}
            begin
                if ($t_{n+1} > t_{p+1}$)
                  $S_{pfet}[i] = S_{pfet}[i] + 1;$
                if ($t_{p+1} > t_{n+1}$)
                  $S_{nfet}[i] = S_{nfet}[i] + 1;$
                $t_{p+1}=(R_{peff}[i-1]/S_{pfet}[i-1]+R_{neff}[i-1]/S_{nfet}[i-1])*(C_{load}[i-1]+C_{punit})$
                      $+(R_{peff}[i]/(S_{pfet}[i]+1)+R_{neff}[i]/S_{nfet}[i])*C_{load}[i];$
                $t_{n+1}=(R_{peff}[i-1]/S_{pfet}[i-1]+R_{neff}[i-1]/S_{nfet}[i-1])*(C_{load}[i-1]+C_{nunit})$
                      $+(R_{peff}[i]/S_{pfet}[i]+R_{neff}[i]/(S_{nfet}[i]+1))*C_{load}[i];$
                $t_{total} = \min\{t_{p+1},t_{n+1}\};$
            end;
        {update delay time of $v_i$ and $v_{i-1}$}
        $t_{rise}[i]=(R_{peff}[i]/S_{pfet}[i])*C_{load}[i];$
        $t_{fall}[i]=(R_{neff}[i]/S_{nfet}[i])*C_{load}[i];$
        $t_{delay}[i]=t_{rise}[i] + t_{fall}[i];$
        $t_{rise}[i-1]=(R_{peff}[i-1]/S_{pfet}[i-1])*C_{load}[i-1];$
        $t_{fall}[i-1]=(R_{neff}[i-1]/S_{nfet}[i-1])*C_{load}[i-1];$
        $t_{delay}[i-1]=t_{rise}[i-1] + t_{fall}[i-1];;$
        end;
    end;
end;
```

3.5 Transistor Desizing Algorithm

In the initial transistor sizing stage, we try to over-size the transistors in the design. The desizing algorithm is then applied to minimize the area and power subject to timing constraints. The desizing algorithm consists two separate phases: delay estimation and transistor desizing. This desizing algorithm first estimates the delay allowance of each gate based on timing constraints, and then desizes the transistors of all gates to the minimal values that satisfies the timing constraints.

We formulate the transistor desizing problem into a graph problem. A multistage directed graph **G = (V,E)** is formed to find the worst critical path at the critical path analysis stage. Each vertex represents a gate in the design, and t_{delay}[i] is the delay of vertex[i]. Both t_{delay}[s] and t_{delay}[t] are zero. Let $t_{estimate}$<i,j>, be the delay of edge<i,j>. We first evaluate the delays at all paths. By applying the timing constraint to the vertex t, $t_{estimate}$<i,j> is computed from t to s using breath first search. The $t_{estimate}$<i,j> computation consists of four basic forms(Figure 6).

(1) A vertex has only one input and one output arc: $t_{estimate}$<i-1,i> is $t_{estimate}$<i,i+1> - t_{delay}[i] (Figure 6(a)).

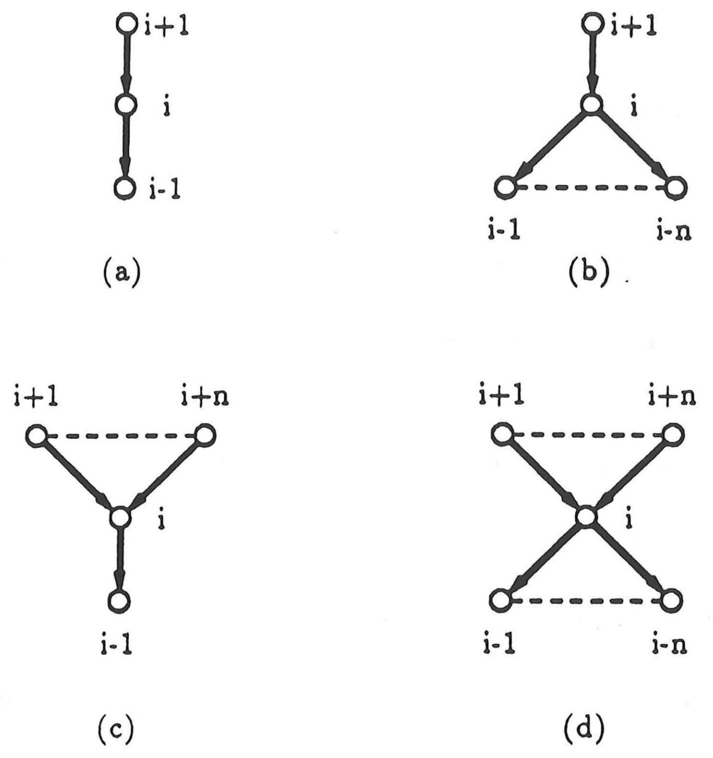

Figure 6 Four graph forms for delay estimation

(2) A vertex has only one input arc and more than one output arc: $t_{estimate}<i-k,i>$, $1 \leq k \leq n$, is $t_{estimate}<i,i+1>$ - $t_{delay}[i]$ (Figure 6(b)).

(3) A vertex has more than one input arc and only one output arc: to consider the worst delay path, the $t_{estimate}<i-1,i>$ is $\min\{t_{estimate}<i,k> - t_{delay}[i]\}$, $i \leq k \leq i+n$ (Figure 6(c)).

(4) A vertex has more than one input and output arc: $\mathbf{t}_{estimate}$<i-k,j>, 1≤k≤n, is min{$\mathbf{t}_{estimate}$<i,k> - $\mathbf{t}_{delay}$[i]}, i≤k≤i+n (Figure 6(d)).

If $\mathbf{t}_{estimate}$<s,j> is larger than zero, the path from input vertex j to sink vetex t is oversized. If $\mathbf{t}_{estimate}$<s,j> is less than zero, the path from input vertex j to sink vertex t is undersized. Since the sizing algorithm tries to oversize all of the paths, the chances of undersizing are very unlikely.

In the transistor desizing stage, the desizer desizes the transistors at all paths from source vertex s to sink vertex t using the information we obtained from the delay evaluation stage. The desizing algorithm is a reverse process of the sizing algorithm. Transistor desizing depends on which transistor, PFET or NFET, contributes less delay reduction.Let $\mathbf{t}_{allowance(i)}$ be the delay allowance of vertex i.

The desizing computation consists of four basic forms as shown in figure 6.

(1) A vextex has only one input and one output arc: if $t_{estimate}$[i,i+1]-$t_{estimate}$[i-1,i] > t_{delay}[i] the v_i is oversized and $t_{allowance}$[i] = $t_{estimate}$[i,i+1]-$t_{estimate}$[i-1,i] Figure 6(a)).

(2) A vertex has only one input arc and more than one output arc: $t_{allowance}$[i] = $t_{estimate}$[i,i+1]-max{$t_{estimate}$[i-j,i]}, 1≤j≤n (Figure 6(b)).

(3) A vertex has more than one input arc and only one output arc: $t_{estimate}[i] = \min\{t_{estimate}[i,i+j]\} - t_{estimate}[i-1,i]$, $1 \leq j \leq n$ (Figure 6(c)).

(4) A vertex has more than one input and output arc: $\min\{t_{estimate}[i,i+j]\} - \max\{t_{estimate}[i-j,i]\}$, $1 \leq j \leq n$ (Figure 6(d)).

If a vertex is already desized to the unit size and there is some delay allowance left, this delay allowance will propagate to the vertices in the higher level. This process teminates when no more transistors can be desized and delays of all paths satisfy the timing constraint.

ALGORITHM 3 : Transistor desizing

```
{delay evaluation of all paths using breath first search}

Let $t_{estimate}[i,j]$ be the delay of edge<i,j>.

**procedure** delay_evaluation($t_{constraint}$,V,$V_{critical}$)
**begin**
   Q <= s where s is source node;
   $t_{estimate}[i,s] = t_{constraint}$;
   **while** (not EMPTY(Q))
   **begin**
      v = FRONT(Q);
      DEQUEUE(Q);
      **for** (each vertex i in V connected to v)
      **begin**
         **if** (i <> sink node t)
         **begin**
            **if** (i has only one input arc from vertex i+1)
```

```
                $t_{estimate}$[i-1,1]=$t_{estimate}$[i,i+1]-$t_{delay}$[i];
            else {i has more than one input arcs (i+1..i+n)}
                $t_{estimate}$[i-1,i] = min($t_{estimate}$[i,i+1..i+n]-$t_{delay}$[i]);
            ENQUEUE(i,Q);
          end;
        end;
      end;
    end;
```

{Desizing Algorithm}

Let $t_{allowance}$[i] be the delay allowance of vertex i.

```
procedure desize(V)
begin
 for (every vertex i in V,do breath first search from sink node t to source node s)
 begin
   if (i-1 is source vertex s)
   begin
     if (i has more one input arc)
       $t_{allowance}$[i] = $t_{estimate}$[i,i+1];
     else
       $t_{allowance}$[i] = min($t_{estimate}$[i,i+1..i+n]);
   end;
 else
 begin
   if (i has one input and one output arc)
     $t_{allowance}$[i] = $t_{estimate}$[i,i+1]-$t_{estimate}$[i-1,i];
   if (i has more than one input arcs and only one output arc)
     $t_{allowance}$[i] = min($t_{estimate}$[i,i+1..i+n])-$t_{estimate}$[i-1,i];
   if (i has only one input arc and more than one output arcs)
     $t_{allowance}$[i] = $t_{estimate}$[i,i+1]-max($t_{estimate}$[i-1..i-n,i]);
   if (i has more than one input and output arcs)
     $t_{allowance}$[i] = min($t_{estimate}$[i,i+1..i+n])-max($t_{estimate}$[i-1..i-n,i]);
 end;
 if ($t_{allowance}$[i] > $t_{delay}$[i])
 begin
   $t_{actual}$ = $t_{delay}$[i];
   while ($t_{allowance}$[i] > $t_{actual}$ AND $S_{pfet}$[i] and $S_{nfet}$[i] are not unit size)
   begin
```

```
        if ($S_{pfet}$[i] <> 1 AND $S_{nfet}$[i] <> 1)
        begin
          $t_{p-1} = t_{fall}$[i]/$S_{nfet}$[i] + $t_{rise}$[i]/($S_{pfet}$[i]-1);
          $t_{n-1} = t_{rise}$[i]/$S_{pfet}$[i] + $t_{fall}$[i]/($S_{nfet}$[i]-1);
          $t_{actual}$ = min($t_{p-1}$,$t_{n-1}$);
          if ($t_{allowance}$[i] > $t_{actual}$)
          begin
            if ($t_{p-1} > t_{n-1}$)
              $S_{nfet}$[i] = $S_{nfet}$[i] - 1;
            else
              $S_{pfet}$[i] = $S_{pfet}$[i] - 1;
          end;
        end;
      else
      begin
        if ($S_{pfet}$[i] > 1)
          $t_{actual}$ = $t_{fall}$[i]/$S_{nfet}$[i] + $t_{rise}$[i]/($S_{pfet}$[i]-1);
        else
          $t_{n-1} = t_{rise}$[i]/$S_{pfet}$[i] + $t_{fall}$[i]/($S_{nfet}$[i]-1);
        if ($t_{allowance}$[i] > $t_{actual}$)
        begin
          if ($S_{pfet}$[i] > 1)
            $S_{pfet}$[i] = $S_{pfet}$[i] - 1;
          else
            $S_{nfet}$[i] = $S_{nfet}$[i] - 1;
        end;
      end;
    end;
    if ($t_{allowance}$[i] > $t_{actual}$)
    begin
      if (i has more than one input arcs)
        for (j=i+1 to i+n)
          $t_{estimate}$[i,j]=$t_{estimate}$[i,j] + ($t_{allowance}$ - $t_{actual}$);
      else
        $t_{estimate}$[i,i+1]=$t_{estimate}$[i,i+1] + ($t_{allowance}$ - $t_{actual}$);
    end;
  end;
 end;
end;
```

4. Results

The previously decribed algorithms are embedded in MILO[VaGa88] which currently runs on SUN 3 workstations under the UNIX operating system. Synthesized designs with sized transistors are passed to LES[LiGa87] for layout generation and then to the GDT [BuMa85] for simulation. The custom layout produced by LES uses 3 micron CMOS technology. We have run a number of examples with varied timing constraints. Table 1 shows the area and delay comparisons between non-optimized designs and optimized designs. The layout area in the table is the total layout area, not transistor area. The delays in the table are measured with Lsim mixed-mode timing simulator. The optimized results show that the delays are 33% to 58% faster and take 4% to 34% more area. Table 1 also shows the comparisons between the required delay reduction and the actual delay reduction. The errors (+3% to -14%) are mainly caused by using the simple RC model and some estimates of wire resistances and capacitances. Futher improvement can be achieved by using the resistance and capacitance parameters extracted directly from the layout.

Table 2 shows the optimized design tradeoffs between area and delay. The designs are optimized by applying varied timing constraints. Figure 7 displays a composite graph showing the tradeoffs between area and delay. The results are normalized against the design using the unit transistor size whose reference point is shown at delay = 100%, area = 100%.

5. Conclusions

We have presented a novel algorithm to formulate the transistor sizing problem into a graph problem associated with a non-linear optimization technique. This algorithm decomposes the transistor sizing process into three interactive phases: critical path analysis, transistor sizing, and transistor desizing. This is different from traditional sizing methods that optimize the given design locally, using one or several paths at a time. Our algorithm optimizes the given design globally, using all of the paths at same time; hence we do not need to check whether some new paths become critical. Therefore, this approach can reduce the delays of all paths to satisfy timing constraints simultaneously. Furthermore, since the PFET and NFET are sized separately, this approach has greater control over area/time tradeoffs than traditional sizing methods.

Using this algorithm, we expect to obtain the minimal transistor area of the design subject to the timing constraints. Since the power dissipation of a design depends on the total transistor area of a design, this approach also produces a design with minimal power subject to the timing constraints.

		area(sq. um)			delay(ns)				
	Trs#	non-opt	opt	%	non-opt	opt	required delay(%)	real delay(%)	error (%)
bcd	42	29,750	39,450	+32.6	16.4	9.3	-40	-43.4	-3.4
add1	44	30,972	41,712	+34.7	19.9	8.2	-50	-58.7	-8.7
f1	60	41,090	51,000	+24.1	26.1	11.5	-55	-55.9	-0.9
random1	62	52,250	62,748	+20.1	24.7	11.4	-40	-53.0	-13.0
random2	72	58,742	62,720	+6.8	25.5	13.5	-50	-47.4	+2.6
random3	76	64,200	68,807	+7.2	32.5	21.5	-30	-33.8	-3.8
f2	96	81,510	86,715	+6.4	26.6	12.6	-50	-52.0	-2.0
random4	100	91,200	94,860	+3.9	30.0	19	-30	-36.7	-6.7
alu2	252	293,314	329,460	+12.3	28.6	12.6	-50	-56.0	-6.0

Table 1

	Trs#		non-opt	opt ----		----		----		---->	
bcd	42	area(sq. um/%)	29,750	34,000	+14.4	39,450	+32.6	44,400	+49.2	50,632	+70.2
		delay(ns/%)	16.4	10.2	-37.6	9.3	-43.4	8.6	-47.4	8.2	-50.0
add1	44	area(sq. um/%)	30,972	36,256	+17.0	41,712	+34.7	45,232	+46.0	48,752	+57.4
		delay(ns/%)	19.9	12.2	-38.6	8.2	-58.7	6.9	-65.4	6.5	-67.3
random5	68	area(sq. um/%)	58,555	62,842	+7.3	64,680	+10.5	70,070	+19.7	72,765	+24.3
		delay(ns/%)	21.2	20.4	-7.3	15.6	-26.3	10.7	-49.5	8.9	-58.0
random6	72	area(sq. um/%)	54,672	61,494	+12.5	64,541	+18.1	71,466	+30.7	74,790	+36.8
		delay(ns/%)	24.2	20.3	-16.1	15.1	-37.6	12.2	-49.5	11.9	-50.8

Table 2

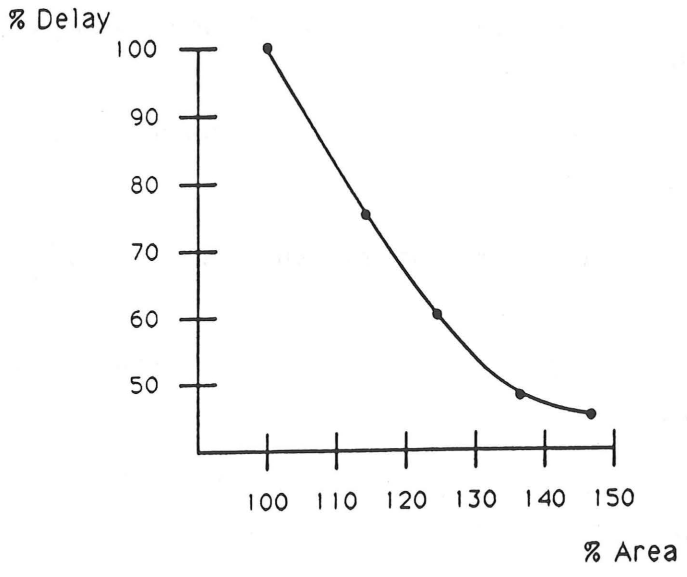

Figure 7 The composite graph of tradeoffs between area and delay

6. Acknowledgements

This work has been supported by NSF (grant no. MIP-8711025), California MICRO program (grant no. 87-134) and grants from Rockwell International and Silicon Systems.

7. References

[BuMa85] M.R. Buric and T.G. Matheson, "Silicon Compilation Environments," Proc. CICC, 1985.

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

[GiMu81] P.E. Gill, W. Murray, and M.H. Wright, Practical Optimization, Academic Press, 1981.

[Hedl84] K. Hedlund, "Models and Algorithms for Transistor Sizing in MOS Circuits," Proc. ICCAD, 1984.

[Hedl87] K. Hedlund, "Aesop: A Tool for Automated Transistor Sizing," Proc. 24th DAC, 1987.

[Hitc82] R.B. Hitchcock, "Timing Verication and the Timing Analysis Program," Proc. 19th DAC, pp. 594-604, 1982.

[HoSa78] E. Horowitz and S. Sahni, "Fundamentals of Computer Algorithms," pp. 203-208, Computer Science Press, Inc., 1978.

[Joup83] N.P. Jouppi, "Timing Analysis for nMOS VLSI," in Proc. 20th DAC pp.411-418 , 1983.

[KaFa85] W.H. Kao, N. Fathi, and C.H. Lee, " Algorithms for Automatic Transistor Sizing in CMOS Digital Circuits," Proc. 22nd DAC, 1985.

[KiCl66] T.I. Kirkpatrick and N.R. Clark, "PERT as an aid to logic design," IBM

J. Res. Develop., vol 10, no. 2,pp. 135-141, Mar, 1966.

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

[Oust84] J.K. Ousterhout, "Switch-level Delay Models for Digital MOS VLSI," in Proc. 21st DAC pp. 542-548, 1984.

[PiDe86] D. Jonathan. Pincus and Alvin M. Despain, "Delay Reduction Using Simulated Annealing," Proc. 23rd DAC, 1986.

[ShFi87] J. Shyu and J.P. Fishburn, "Optimization-Based Transistor Sizing," Proc. CICC, 1987.

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

[YeGh88] H.C Yen, S. Ghanta, and H.C. Du, "A Path Selection Algorithm for Timing Analysis," Proc. 25th DAC, pp. 720-723, 1988.