A Convex Optimization Approach to Transistor Sizing for CMOS Circuits

Sachin S. Sapatnekar*, Vasant B. Rao*, and Pravin M. Vaidya**

* Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1101 W. Springfield Avenue, Urbana, IL 61801.

** Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield Avenue, Urbana, IL 61801.

Abstract

The transistor sizing problem of minimizing the circuit area, subject to the circuit delay being less than a given specification is formulated as a convex programming problem. An efficient convex programming algorithm is then used to obtain the exact *solution. Experimental results on a variety of circuits show that, for a given delay specification, this approach is able to produce circuits with significantly smaller area when compared with TILOS [1,2].*

1. Introduction

A general digital circuit may be treated as a collection of combinational subcircuits that lie between latches. To satisfy a given clocking requirement for the circuit, each combinational subcircuit, that takes its inputs from a set of latches at the time of a clock transition, must produce valid signals at its set of output latches before the next clock transition. In order to do so, the sizes of certain transistors in the circuit need to be increased to more than the minimum allowable size. The *transistor sizing problem*, for each combinational stage, is to

$$\text{minimize } Area$$
$$\text{subject to } Delay < \Delta,$$

where both *Area* and *Delay* are certain functions of the sizes of transistors and Δ is a delay specification for the combinational stage that depends on the clock width.

Many of the approaches to the transistor sizing problem that have been suggested so far recognize the fact that the problem can be mapped on to a convex optimization problem, and hence any local minimum is a global minimum. However, none of these approaches takes full advantage of convexity to generate an optimum solution to the problem. The goal of this research is to efficiently generate an exact solution to the transistor sizing problem. This is done by modeling the delay and area as *posynomial* [8] functions of transistor sizes. A simple variable transformation is then used to transform posynomials to convex functions. Finally, an efficient convex programming algorithm [4] is used to solve the resulting optimization problem. A waveform-independent delay estimator that computes rise and fall delays separately, accounts for the slope of waveforms, and handles general pass transistor networks, is described in Section 2. The optimization algorithm, which specifically takes advantage of the fact that the objective and constraint functions are convex, is described in Section 3.

2. The Delay Estimation Algorithm

2.1 Overview

Consider a combinational CMOS circuit with a set of primary input nodes and primary output nodes. The circuit is first divided into *channel-connected components*, where each component is a set of transistors that are connected by drain and source nodes. All transitions at the primary inputs of the combinational circuit are controlled by a clock; therefore it is reasonable to assume that they all occur at the same time. A technique known as PERT (Program Evaluation and Review Technique) [5] is used to compute the maximum overall rise and fall delays between primary inputs and primary outputs of the circuit. A trace-back method is then used to obtain a *critical path*. The numbers t_h and t_l are assigned to each output node of each component in the circuit, which correspond to the total rise and fall delay from the primary inputs. Additionally, the output transition is modeled as a ramp-like function, with Δ_h and Δ_l being the rise and fall transition times respectively.

Initially, the values of t_h, t_l, Δ_h and Δ_l are known only at the primary inputs (where they have been set to zero). The PERT method places all components, whose inputs are only primary inputs, in a queue and schedules these for evaluation. The evaluation of a scheduled component involves finding the values of t_h, t_l, Δ_h and Δ_l at each output node of the component, based on the corresponding values at the input nodes of the component, and the Elmore delay [6] (explained in Section 2.3 below) of an RC network representing the worst-case scenario for the component. Next, all unevaluated components, for which the t_h and t_l values have been computed at each input, are placed at the end of the evaluation queue. The procedure ends when all components have been evaluated.

2.2 RC Model of a MOS transistor

A MOS transistor is modeled as a voltage-controlled switch with an on-resistance R_{on} between drain and source, and three grounded capacitances C_d,

C_s, and C_g at the drain, source, and gate terminals, respectively. The resistance and capacitances associated with a MOS transistor of width x are of the form :

$$R_{on} = \frac{d_0}{x} \; ; \; C_d = C_s = d_{11}x + d_{12} \; ; \; C_g = d_{21}x + d_{22}$$

The precise transistor model is the same as that in [3].

2.3 Evaluation of a Component

Each transistor in a component scheduled for evaluation could have an input waveform corresponding to a steady logic 0, a steady logic 1, a logic 0 to logic 1 transition, or a logic 1 to logic 0 transition. It is required that the worst-case Elmore delay at an output node of the component be found over all possible input combinations. The algorithm for finding the worst-case fall delay at an output node o of a component is described below. The worst-case rise delay at o can be found analogously.

The component is represented by an undirected weighted graph G with an edge between the drain and source nodes of each transistor in the component. Edge weights are given by the resistance R_{on} of the corresponding transistor. The V_{DD} node and all incident edges are then removed from the graph. Let $t_{h,max}$ denote the maximum value of t_h among all input nodes of the component and suppose this occurs at the gate node of a n-type transistor corresponding to an edge $e_{\max}$. It is assumed that the worst-case path is the *largest resistive path* (LRP) (i.e. the path of largest weight) between o and ground that passes through $e_{\max}$. Since this is equivalent to the longest path problem in a graph, which is NP-Hard [5], we have developed a heuristic to perform this task. This heuristic is exact for series-parallel graphs of CMOS AOI gates, and can be outlined as follows.

```
$T$ = maximum weighted spanning tree in $G$ containing $e_{\max}$
      such that the path $P$ between $o$ and ground in $T$ contains $e_{\max}$
$maxW$ = sum of weights of edges in $P$
$LINK$ = edges in $G-T$
for each edge $e \in LINK$ {
  $T_1 = T \cup e$
  $P_1$ = max weight $o$-to-ground path in $T_1$ through $e_{\max}$
  $W$ = sum of weights of edges in $P_1$
  if ( $W > maxW$ ) {
    $e'$ = any edge in $P - (P \cap P_1)$
    $T = T_1 - e', P = P_1, maxW = W$
  }
}
```

Now, consider any spanning tree T_w of the graph G, and let R_{ij} denote the resistance of the intersection between the paths to ground from nodes i and j in T_w. The Elmore delay [6] between o and the ground node in the RC-tree represented by T_w is given by

$$\sum_{j \in T_w} R_{oj} C_j \tag{2.1}$$

where C_j is the capacitance to ground at node j in T_w. Therefore, in order to find a tree that contains the LRP and which maximizes the Elmore delay, certain edges must be added to the LRP in such a way that R_{oj} is maximized for every node j in the graph. The algorithm to construct the worst-case tree T_w from the LRP starts by taking T_w to be the LRP itself. For a node $n_1 \notin T_w$ the algorithm finds a node $n_2 \in T_w$ that is farthest from the ground node and is connected to n_1 by a path that does not intersect T_w. This path is then added to T_w and the procedure is repeated until all nodes of G are included in the tree T_w. The worst-case fall delay at o is then computed using (2.1). Finally, the value of t_l for output node o is computed by adding $t_{h,max}$ and a term (see [7] for exact formulas) related to the rise-time Δ_h at the input node corresponding to the worst-case Elmore fall delay. Also, the value of the transition time at o is taken to be $\Delta_l = 2\,(t_l - t_{h,max})$.

The value of t_h, the worst-case rise delay at each output node of the component can be found in an analogous manner. The weighted graph representing the component is constructed as before except that the ground node is removed instead of the V_{DD} node. The rest of the procedure to find the worst-case Elmore rise delay is identical to that of the fall delay with the role of the ground node replaced by the V_{DD} node; the roles of t_h and Δ_h are exchanged with those of t_l and Δ_l in the fall delay case.

2.4 Area and Delay Functions

Let n denote the number of transistors in a combinational circuit and let $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ be an n-dimensional vector of the transistor widths. The total area of the circuit is taken to be

$$Area(\mathbf{x}) = \sum_{i=1}^{n} x_i \tag{2.2}$$

which is a posynomial in $\mathbf{x}$. The overall delay $D(\mathbf{x})$ through the critical path, can be shown to be a posynomial [1,7,8] of the form

$$D(\mathbf{x}) = \sum_j \gamma_j \prod_{i=1}^{n} x_i^{\alpha_{ij}} \tag{2.3}$$

where $\gamma_j > 0$ and $\alpha_{ij} \in \{-1,0,1\}$.

3. The Convex Programming Algorithm

Under the variable transformation $x_i = e^{z_i}$, the posynomial area and delay functions of (2.2) and (2.3) become convex functions

$$Area(\mathbf{z}) = \sum_{i=1}^{n} e^{z_i} \text{ and } D(\mathbf{z}) = \sum_j \gamma_j e^{\sum_{i=1}^{n} \alpha_{ij} z_i} \tag{3.1}$$

Hence, the original transistor sizing problem now becomes a *convex programming* problem of minimizing a convex function $Area(\mathbf{z})$ over a convex set of constraints $S = \{\mathbf{z} \in R^n : D(\mathbf{z}) \le \Delta\}$. Let $\mathbf{z}_{opt}$ be the exact solution to this problem.

An efficient convex programming algorithm proposed by Vaidya in [4] maintains a polytope P given by

$$P = \{\mathbf{z} : A\,\mathbf{z} \geq \mathbf{b}\} \tag{3.2}$$

such that $\mathbf{z}_{opt}$ lies within P. where $A \in R^{m \times n}$ and $\mathbf{b} \in R^m$. Here, m denotes the number of linear inequality constraints describing the polytope. The initial polytope P may be selected to be an n-dimensional box $\{\mathbf{z} : \ln(x_{\min}) \leq z_i \leq \ln(x_{\max})\}$, where $x_{\min}$ and $x_{\max}$ are the user-specified minimum and maximum allowable transistor sizes, respectively.

The algorithm proceeds iteratively as follows. First, a *center* $\mathbf{z}_c$ deep in the interior of the current polytope P is found by using Newton's method [9] to minimize the following *log-barrier function*

$$F(\mathbf{z}) = -\sum_{i=1}^{m} \ln(\mathbf{a}_i^T\mathbf{z} - b_i) \tag{3.3}$$

where $\mathbf{a}_i^T$ denotes the i-th row of A. Note that near the boundary of the polytope, $F(\mathbf{z})$ tends to infinity and its value decreases as one moves "deeper" into the interior of the polytope. Also, the value of $F(\mathbf{z})$ is undefined outside the boundary of the polytope. Moreover, F is a convex function of $\mathbf{z} \in P$, with its gradient and Hessian respectively given by

$$\nabla F(\mathbf{z}) = -\sum_{i=1}^{m} \frac{\mathbf{a}_i^T}{(\mathbf{a}_i^T\mathbf{z} - b_i)} \in R^n \tag{3.4}$$

$$\nabla^2 F(\mathbf{z}) = \sum_{i=1}^{m} \frac{\mathbf{a}_i\,\mathbf{a}_i^T}{(\mathbf{a}_i^T\mathbf{z} - b_i)^2} \in R^{n \times n} \tag{3.5}$$

The Newton's method generates iterates of the form $\mathbf{z}_{k+1} = \mathbf{z}_k + t^*\xi_k$, for $k = 0,1,2,\cdots$, until convergence, where

$$\xi_k = -\,[\nabla^2 F(\mathbf{z}_k)]^{-1}\,[\nabla F(\mathbf{z}_k)]^T \tag{3.6}$$

is the *Newton direction* at $\mathbf{z}_k$, and t^* is the point that minimizes the one-dimensional function $\phi(t) = F(\mathbf{z}_k + t\,\xi_k)$ and is obtained by performing a *one-dimensional line-search* [9].

After finding the center $\mathbf{z}_c$ of the existing polytope P, an *oracle* is invoked to determine whether or not the center $\mathbf{z}_c$ lies within the constraint region S. The oracle is merely a routine that invokes the delay estimator described in Section 2, with the transistor sizes $x_{c,i} = e^{z_{c,i}}$, to determine whether or not the delay requirement is met. If the point $\mathbf{z}_c$ lies outside S, it is possible to find a *separating hyperplane* passing through $\mathbf{z}_c$ that divides the polytope P into two parts, such that S lies entirely in the part satisfying

$$\mathbf{c}^T\,\mathbf{z} \geq \mathbf{c}^T\mathbf{z}_c \tag{3.7}$$

where $\mathbf{c} = -[\nabla D(\mathbf{z})]^T$ is determined by the oracle. The symbol ∇ represents the gradient operator. If, however, the point $\mathbf{z}_c$ lies within the feasible region S, i.e., the transistor sizes $x_{c,i} = e^{z_{c,i}}$ satisfy the delay constraints, then there exists a hyperplane that divides the polytope into two parts such that $\mathbf{z}_{opt}$ is contained in one of them. This hyperplane satisfies the equality in (3.7) with $\mathbf{c} = -[\nabla Area(\mathbf{z})]^T$. In either case, the inequality (3.7) is added to the current polytope to give a new polytope that has roughly half the original volume [4]. The process is repeated until the polytope is sufficiently small.

The gradient of the delay has contributions from transistors that lie on the critical path of the circuit, or act as capacitive loads for transistors on the critcal path; the delay gradient component corresponding to all other transistors is zero. In contrast, the area function is such that it contributes a non-zero value to each component of its gradient.

The process of computing a Newton direction of (3.6) involves the inversion of an $n \times n$ Hessian matrix which takes $O(n^3)$-time and can prove to be rather expensive. This expense can be cut down by maintaining the inverse of an approximate Hessian $\hat{H}$ via rank-one updates [9], and using an approximate Newton direction $\hat{\xi}_k = -\hat{H}^{-1}(\nabla F(z_k))^T$ instead of ξ_k in the line search. It must be noted that using an approximate Newton direction instead of the exact one essentially does not affect the convergence properties of the center-finding algorithm or the overall convex programming algorithm [4]. The procedure used for the efficient computation of $\hat{\xi}_k$ involves rank-one update schemes, used in conjunction with a preconditioned conjugate gradient method [10]. Due to limitations of space, it is not possible to descibe these schemes in further detail in this paper.

4. Results

The algorithms described in the previous sections have been implemented in a tool called CONTRAST (Convex Optimization-based Novel TRAnsistor Sizing Tool) for CMOS circuits. A prototype of TILOS, as described in [1,2], with the delay model described in Section 2 of this paper, has also been implemented for comparison with our method. The TILOS parameter, BUMPSIZE [2] is taken to be 1.1.

A sample of the results generated by CONTRAST and TILOS is shown in Table I. For each circuit, the number of transistors, n, the delay specification Δ, and the circuit area and the execution time on a Sun Sparcstation of CONTRAST and TILOS are listed. The technology parameters correspond to a 0.9μ process with a minimum transistor size $x_{\min} = 1.8\mu$. The loading capacitance at each primary output is taken to be the gate-capacitance C_g of a CMOS inverter with each transistor of size 50μ. Under relatively lax delay constraints, TILOS comes close to the optimal solution generated by CONTRAST. However, when the delay specification is tight, CONTRAST generates solutions that give as much as a 40% reduction in area over TILOS.

Table I : A Comparison of Results from CONTRAST and TILOS

Circuit	n	Δ (ns)	CONTRAST		TILOS	
			Area	Time	Area	Time
inv6	12	2.5	330.9	3.4s	382.3	3.4s
		2	940.9	5.4s	1653.3	4.9s
inv10	20	5	325.5	8.3s	352.0	6.4s
		4	967.1	13.7s	1357.7	10.4s
tree	28	3	244.2	19.2s	251.0	8.6s
		2	759.2	18.2s	823.1	14.9s
add2	52	15	141.5	51.7	157.1	12.6
		12	188.8	74.7s	225.6	20.3s
		10	224.5	95.0s	347.5	28.6s
add8	208	50	756.1	19.1m	954.4	5.9m
		40	1136.5	17.1m	1941.0	9.8m
add32	832	250	2243.4	522.6m	2634.7	72.2m
		200	3276.0	614.6m	3937.8	126.2m

Table II shows the variation of transistor sizes in a 7-stage inverter chain. As Δ (given in ns) is made tighter, it is seen that the transistor sizes tend to increase more drastically from the input to the output stage. The size of the input stage is restricted owing to the contribution of the resistance of the source that drives the first stage.

Table II : Variation of p/n Transistor Sizes along a 7-inverter Chain

Δ	Stg 1	Stg 2	Stg 3	Stg 4	Stg 5	Stg 6	Stg 7
20	1.9/1.8	2.1/2.0	3.2/1.9	2.2/2.0	3.0/1.9	3.4/2.6	17.5/8.2
15	2.3/1.8	3.8/2.9	5.7/2.9	5.0/3.9	7.6/3.7	9.6/7.3	40.3/19.2
13	2.6/1.8	5.9/4.5	12.4/6.4	12.8/10.0	20.0/10.8	27.1/21.7	86.1/44.9
12.5	2.6/1.8	6.5/5.1	16.5/8.6	23.9/17.5	46.4/24.4	63.4/47.1	161.6/87.1

An 8-stage inverter chain was sized using CONTRAST and TILOS under various delay specifications. A comparison of the area obtained from the two is presented in Table III. It shows that TILOS performs fairly well when the difference between the areas of the unsized circuit (i.e., when all transistors have been set to $x_{\min}$), and the optimally sized circuit is small. As this condition becomes weaker, or in other words, Δ becomes tighter, the performance of CONTRAST is found to be increasingly superior to that of TILOS.

Table III : CONTRAST vs TILOS on a Chain of 8 Inverters

Δ	6.0ns	4.0ns	3.5ns	3.25ns	3.1ns	3.0ns
CONTRAST	89.8	250.4	415.6	591.3	760.4	921.9
TILOS	93.3	271.9	482.6	747.2	1066.2	1512.5

Finally, the performance of CONTRAST's delay estimator on a chain of 8 inverters (Inv8) and a 2-bit adder using complex AOI gates (Add2), in relation with SPICE delay values, is shown in Table IV. Each column in the table corresponds to a different set of transistor sizes in the circuit. The values can be seen to be in good agreement.

Table IVa : CONTRAST's Delay Estimator vs SPICE (Circuit Inv8)

CONTRAST	1.84e-8	1.50e-8	9.97e-8	4.99e-9	3.99e-9
SPICE	1.84e-8	1.47e-8	1.03e-8	5.20e-9	4.10e-9

Table IVb : CONTRAST's Delay Estimator vs SPICE (Circuit Add2)

CONTRAST	2.56e-8	1.99e-8	1.50e-8	1.20e-8	9.98e-9
SPICE	2.32e-8	1.64e-8	1.43e-8	1.19e-8	1.02e-8

Acknowledgements

This work has been supported in part by JSEP under contract N00014-90-J-1770 and in part by NSF under contracts CCR-9057-481 and CCR-9007-195.

REFERENCES

[1] J. P. Fishburn and A. E. Dunlop, "TILOS : A Posynomial Programming Approach to Transistor Sizing," *Proc. ICCAD-85*, Santa Clara, CA, pp. 326-328, Nov 1985.

[2] A. E. Dunlop, J. P. Fishburn, D. D. Hill, and D. D. Shugard, "Experiments using Automatic Physical Design Techniques for Optimizing Circuit Performance," *Proc. 32nd Midwest Symposium on Circuits and Systems*, Urbana, IL, Aug 1989.

[3] J. Shyu, J. P. Fishburn, A. E. Dunlop, A. L. Sangiovanni-Vincentelli, "Optimization-Based Transistor Sizing," *IEEE J. Solid-State Circuits*, Vol 23, pp. 400-409, Apr 1988.

[4] P. M. Vaidya, "A New Algorithm for Minimizing Convex Functions Over Convex Sets," *Proc. IEEE Foundations of Computer Science*, Oct 1989, pp. 332-337; also to appear in *Mathematical Programming*.

[5] S. Even, *Graph Algorithms*, Computer Science Press, 1979.

[6] J. Rubenstein, P. Penfield, and M. A. Horowitz, "Signal Delay in RC Tree Networks," *IEEE Trans. Computer-Aided Design*, Vol. CAD-2, No. 3, pp. 202-211, Jul 1983.

[7] N. Hedenstierna, and K. O. Jeppson, "CMOS Circuit Speed and Buffer Optimization" *IEEE Trans. Computer-Aided Design*, Vol CAD-6, pp. 270-281, Mar 1987.

[8] J. G. Ecker, "Geometric Programming : Methods, Computations and Applications," *SIAM Review*, Vol. 22, No. 3, pp. 338-362, July 1980.

[9] D. G. Luenberger, *Linear and Non-linear Programming*, Addison-Wesley, 1984.

[10] G. H. Golub, and F. H. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, 1989.