Efficient and Accurate Gate Sizing with Piecewise Convex Delay Models

Hiran Tennakoon and Carl Sechen
Department of Electrical Engineering
University of Washington, Seattle WA 98195-2500
hiran@ee.washington.edu, sechen@ee.washington.edu

ABSTRACT

We present an efficient and accurate gate sizing tool that employs a novel piecewise convex delay model, handling both rise and fall delays, for static CMOS gates. The delay model is used in a new version of a gate-sizing tool called Forge, which not only exhibits optimality, but also efficiently produces the area versus delay trade-off curve for a block in one step. Forge includes a realistic delay propagation scheme that combines arrival times and slew-rates. Forge is 6.4X faster than a commercial transistor sizing tool, while achieving better delay targets and uses 28% less transistor area for specific delay targets, on average.

Categories and Subject Descriptors
B.7.2 [Integrated Circuits]: Design Aids.

General Terms
Design, Performance, Algorithms

Keywords
Delay modeling, gate sizing, Lagrangian relaxation, piecewise convex, optimization

1 INTRODUCTION

Automation of the transistor sizing process has been steadily gaining popularity in digital/analog VLSI design. Using modeling techniques, many automation algorithms guarantee globally optimum solutions for given specifications [1]. In digital CMOS design, sizing enables one to evaluate the performance of different cell libraries [2], to address the timing closure problem when the design is placed and routed [3], and to generate area/power versus delay trade-off curves [4]. The usefulness of a transistor or gate sizer depends on three issues: delay model accuracy, optimality and efficiency.

First, the models used must be accurate enough to provide useful solutions to the practical problem. TILOS [5], which was the first algorithmic attempt at transistor sizing, used the popular Elmore delay model [6]. Sapatnekar *et al.* in their work on convex gate level delay models showed that in modern designs the average error of the Elmore model to Hspice is as much as 20.5% [7]. This renders the Elmore model unsuitable for optimization, as solutions will not correlate well with Hspice results. In the same work they present an input slew-rate dependent model with greater accuracy than Elmore [7]. Their approach is to run a gate level characterization using Hspice, over a range of device sizes, input slew-rates, and output loading. Then they fit this measured data to predefined convex functions.

Second, it is important for the sizer to generate solutions at least close to optimum. An important contribution made in TILOS is the observation that Elmore delay is convex under a variable transformation $x = e^z$ [5]. For a convex function any local optimum is also a global optimum. However, their formulation of the problem, and the sensitivity-based heuristic, cannot guarantee optimal results. The complex models used by Sapatnekar *et al.* are convex, and they use an interior-point algorithm to find optimum solutions [9]. However, without correct delay propagation, the practicality of their algorithm is in question. Further, the interior point algorithm is prone to large run-times [10]. Chen *et al.* first formulated the gate-sizing problem in an elegant mathematical form that was solvable with guarantees of optimality [11]. Though they used the Elmore delay model in their work, their formulation of the sizing problem is very significant, as it enabled optimization to be carried out on the entire design rather than optimization of only the critical paths of the design. However, in [12] it is shown that the convergence of the algorithm used is highly sensitive to initial conditions and constant factors.

Third, it is important that the sizing algorithm is efficient, especially given the ever-larger circuits. Approaches such as JiffyTune [13] and EinsTuner [14] use dynamic timing engines to generate circuit sensitivities and thus incur large run-time penalties. While JiffyTune uses dynamic timing analysis, generating exact solutions, EinsTuner uses static timing analysis with the latest-arrival delay propagation method and has additional slew-rate constraints at each node in the design.

Further, most sizing algorithms do not use accurate delay propagation for multi-input gates. Two possible methods of delay propagation for multi-input gates are the latest arrival propagation, and the worst slew propagation. While the first approach is too optimistic, the later is too pessimistic.

This work addresses each of the three above issues. We form our delay models by characterizing a library of gates, with maximum and minimum limits on the sizes of transistors. Each gate in the library has minimum/maximum beta ratio limits. All gates are treated as parameterized cells, with one variable controlling the width of the nMOS transistors, and another controlling the pMOS transistors.

We generate a piecewise convex delay model by efficiently dividing the measured data into smaller regions; thus, we show great improvement in the accuracy and range (in terms of output load and slew-rate) of the model. The delay model is dependent on input slew-rate, transistor sizes, and output loading. While we can only prove convexity on each individual piece, and not the overall model, our sizing algorithm, called Forge, exhibits optimality.

Our static timing analyzer uses a realistic delay propagation method that uses a combination of the arrival times and their slew-rates to determine delay propagation.

The rest of the paper is organized as follows. In Section 2 we will formulate the problem and address algorithmic issues. Then in Section 3 we present the piecewise convex delay model and how it is integrated within the optimization framework. Section 4 will present results for 31 benchmarks, comparing Forge with a commercial transistor sizing tool (CTST). Conclusions will then follow in Section 5.

2 Problem Formulation

We are interested in the generation of the area versus delay trade-off curve for a logic block. The trade-off curve may be generated by a series of optimizations formulated as:

$$\begin{aligned} &\textit{Minimize } f(\mathbf{w}) = \sum w_i \\ &\textit{subject to Delay} \le \textit{Delay}_{target} \end{aligned} \qquad (1)$$

where the area is represented by the sum of transistor widths, w_i. *Delay* is the maximum arrival time at any output. Sweeping the target delay (*Delay$_{target}$*) generates the curve.

This approach is used in [4]. The problem here is how to determine the incremental delay step that the target must be reduced by. Since the absolute best delay is unknown *a priori*, the optimizer may take many small time steps to generate the trade-off curve. Knowing the absolute minimum delay achievable greatly speeds up the process. Our approach is summarized in Figure 1.

Area vs. Delay Curve Generation:

1. Start with minimum sizes
2. Set Delay$_{max}$ = Delay with minimum sizes
3. Solve for minimum delay possible for design
4. Set Delay$_{min}$ = Delay after sizing
5. Set *n* Delay targets between Delay$_{max}$ and Delay$_{min}$
6. Minimize Area for the *n* Delay targets

Figure 1. Approach to generate the area versus delay curve.

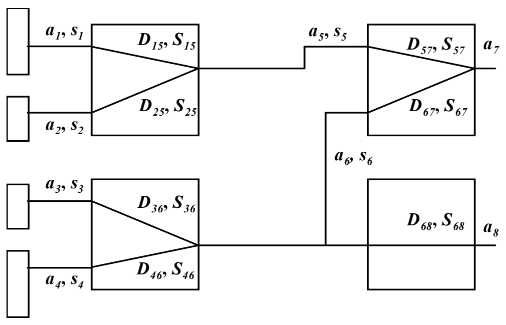

Figure 2. Circuit fragment to aid formulation of the problem for the minimization of the maximum delay. D_{ij}, S_{ij} are the resulting gate input *i* to output *j* delay and output slew-rate, respectively. a_i, s_i are the arrival time and slew-rate at output *i*.

2.1 Minimum Delay Formulation

For the circuit fragment given in Figure 2, the formulation of the minimization of the maximum delay at the primary outputs is given by (2).

$$\begin{aligned} &\textit{Minimize } a_0 \\ &\textit{subject to}: a_7 \le a_0 \qquad a_8 \le a_0 \\ &\quad a_5 + D_{57} \le a_7 \qquad a_6 + D_{67} \le a_7 \\ &\quad a_6 + D_{68} \le a_8 \qquad a_1 + D_{15} \le a_5 \\ &\quad a_2 + D_{25} \le a_5 \qquad a_3 + D_{36} \le a_6 \\ &\quad a_4 + D_{46} \le a_6 \qquad L_i \le w_i \le U_i \end{aligned} \qquad (2)$$

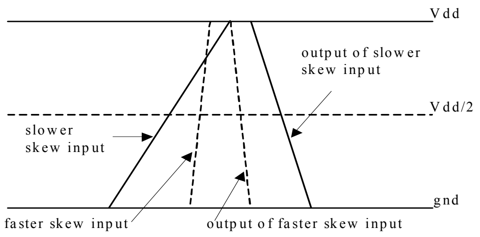

Figure 3. Shows an earlier arriving signal with a large slew-rate causing a later arrival signal at the output than the later arrival at the input.

The additional variable a_0 facilitates the minimization of the maximum of the arrival times a_7 and a_8. L_i and U_i are the lower and upper limits for the transistor widths, and these are simple boundary constraints. Note that for simplicity in the paper (but *not* in the implementation) the formulation here does not include wire capacitances, nor does it differentiate the delay functions in terms of nMOS and pMOS sizes, nor does it distinguish the rise/fall slew-rates.

The formulation given in () uses the latest arrival method to propagate delay. This has been shown to create optimistic delays [8], as shown in Figure 3, where the earlier arriving signal with slower slew results in the latest arrival at the gate output. Propagating the worst slew-rate will result in the delay estimation being too pessimistic [8]. We use a variant of the half-envelope approach from [8], where we combine the delay with ¼ of the slew-rate to decide which signal to propagate. Introducing this to equation ():

$$\begin{aligned} &\textit{Minimize } a_0 \\ &\textit{subject to}: a_7 \le a_0 \qquad a_8 \le a_0 \\ &a_7 = \begin{cases} a_5 + D_{57} & \textit{if } a_5 + D_{57} \ge a_6 + D_{67} \\ a_6 + D_{67} & \textit{if } a_6 + D_{67} > a_5 + D_{57} \end{cases} \\ &a_8 = a_6 + D_{68} \\ &a_5 = \begin{cases} a_1 + D_{15} & \textit{if } a_1 + D_{15} + \tfrac{1}{4}S_{15} \ge a_2 + D_{25} + \tfrac{1}{4}S_{25} \\ a_2 + D_{25} & \textit{if } a_2 + D_{25} + \tfrac{1}{4}S_{25} > a_1 + D_{15} + \tfrac{1}{4}S_{15} \end{cases} \\ &s_5 = \begin{cases} S_{15} & \textit{if } a_1 + D_{15} + \tfrac{1}{4}S_{15} \ge a_2 + D_{25} + \tfrac{1}{4}S_{25} \\ S_{25} & \textit{if } a_2 + D_{25} + \tfrac{1}{4}S_{25} > a_1 + D_{15} + \tfrac{1}{4}S_{15} \end{cases} \\ &a_6 = \begin{cases} a_3 + D_{36} & \textit{if } a_3 + D_{36} + \tfrac{1}{4}S_{36} \ge a_4 + D_{46} + \tfrac{1}{4}S_{46} \\ a_4 + D_{46} & \textit{if } a_4 + D_{46} + \tfrac{1}{4}S_{46} > a_3 + D_{36} + \tfrac{1}{4}S_{36} \end{cases} \\ &s_6 = \begin{cases} S_{36} & \textit{if } a_3 + D_{36} + \tfrac{1}{4}S_{36} \ge a_4 + D_{46} + \tfrac{1}{4}S_{46} \\ S_{46} & \textit{if } a_4 + D_{46} + \tfrac{1}{4}S_{46} > a_3 + D_{36} + \tfrac{1}{4}S_{36} \end{cases} \\ &\qquad L_i \le w_i \le U_i \end{aligned} \qquad (3)$$

In equation (3), for each output except for the primary outputs, the arrival time is equal to the input arrival a_i and gate delay D_{ij} that maximizes the output arrival time, plus ¼ of the output slew-rate S_{ij}. The same holds for the output slew-rates; we pick the slew-rate S_{ij} of the input that maximizes the output delay and slew-rate combination. We take the latest arrival times at the primary outputs. We have a non-smooth problem, as there is an arbitration that determines which delay/slew-rate pair gets propagated. If we assume that the delay and slew-rate are correctly assigned, then we can rewrite (3) as:

$$\begin{aligned} &\textit{Minimize } a_0 \\ &\textit{subject to}: a_7 \le a_0 \qquad a_8 \le a_0 \\ &\quad a_5 + D_{57} \le a_7 \qquad a_6 + D_{67} \le a_7 \\ &\quad a_6 + D_{68} \le a_8 \end{aligned}$$

$$\begin{aligned} a_1 + D_{15} + \tfrac{1}{4}S_{15} &\le a_5 + \tfrac{1}{4}s_5 \\ a_2 + D_{25} + \tfrac{1}{4}S_{25} &\le a_5 + \tfrac{1}{4}s_5 \\ a_3 + D_{36} + \tfrac{1}{4}S_{36} &\le a_6 + \tfrac{1}{4}s_6 \\ a_4 + D_{46} + \tfrac{1}{4}S_{46} &\le a_6 + \tfrac{1}{4}s_6 \end{aligned} \tag{4}$$

For minimizing the worst-case primary output arrival time, we know from the Kuhn-Tucker optimality conditions on the arrival times that the sum of the Lagrange multipliers assigned to the primary outputs must be equal to one, and that the sum of multipliers at a gate's input must equal the sum of the multipliers at the inputs of gates that are driven by that gate [11]. Further, if a set of $\boldsymbol{\lambda}$'s are chosen such that they satisfy the optimality conditions then (4) reduces to:

$$\begin{aligned} &\mathit{Minimize}\ L(\mathbf{w},\boldsymbol{\lambda}) = \lambda_{68}D_{68} + \lambda_{67}D_{67} + \lambda_{57}D_{57} \\ &+ \lambda_{15}\left(D_{15} + \tfrac{1}{4}S_{15}\right) + \lambda_{25}\left(D_{25} + \tfrac{1}{4}S_{25}\right) \\ &+ \lambda_{36}\left(D_{36} + \tfrac{1}{4}S_{36}\right) + \lambda_{46}\left(D_{46} + \tfrac{1}{4}S_{46}\right) \\ &\mathit{subject\ to}: L_i \le w_i \le U_i \end{aligned} \tag{5}$$

Now the problem is reduced to finding the optimum sizes and the optimum values for the Lagrange multipliers $\boldsymbol{\lambda}$. The optimum delays and slew-rates will be selected as shown in equation (3). For more details on applying Lagrangian relaxation to the gate-sizing problem the reader is directed to [11]. Once we obtain the minimum delay bound then we can minimize the area for given delay targets.

2.2 Delay Constrained Area Minimization

Consider the circuit fragment in Figure 2 again. Now we represent the area as the sum of transistor widths and solve the area minimization problem given in (1). By constraining all gate input to output arrival times as in (3) and using the Kuhn-Tucker optimality conditions on the arrival times we have the following formulation for area minimization given delay constraints.

$$\begin{aligned} &\mathit{Minimize}\ L(\mathbf{w},\boldsymbol{\lambda}) = \sum w_i + \lambda_{68}D_{68} + \lambda_{67}D_{67} + \lambda_{57}D_{67} \\ &+ \lambda_{15}\left(D_{15} + \tfrac{1}{4}S_{15}\right) + \lambda_{25}\left(D_{25} + \tfrac{1}{4}S_{25}\right) \\ &+ \lambda_{36}\left(D_{36} + \tfrac{1}{4}S_{36}\right) + \lambda_{46}\left(D_{46} + \tfrac{1}{4}S_{46}\right) \\ &\mathit{subject\ to}: L_i \le w_i \le U_i \end{aligned} \tag{6}$$

Forge starts from the minimum sizes, takes this to be the worst delay, and then sizes for the minimum delay bound, the best delay. Then, Forge sets ten delay targets uniformly between the best and the worst delays, and performs delay-constrained area minimization for these targets. The algorithm computes the entire area versus delay trade-off curve with out restarts. Next, we will discuss problems associated with Lagrangian relaxation based optimization and optimality of the algorithm.

2.3 Algorithmic Issues

When minimizing the Lagrangian, $L(\boldsymbol{w},\boldsymbol{\lambda})$, a known problem is choosing a good step size control mechanism for the sub-gradient techniques that are used to solve for the optimum $\boldsymbol{\lambda}$ [12]. We use the update scheme in [12] for the multipliers.

$$\begin{aligned} \lambda_i &= \lambda_i \times \frac{a_i}{A} && \mathit{for\ each\ primary\ output} \\ \lambda_i &= \lambda_i \times \frac{a_i + D_{ij} + \tfrac{1}{4}S_{ij}}{a_j + \tfrac{1}{4}s_j} && \mathit{for\ all\ other\ gates} \end{aligned} \tag{7}$$

For each primary output, A is the delay target, a_i is the current arrival time, and the multiplier assigned for this constraint is updated by multiplying itself by the ratio of the current arrival time to the required time. For all other gates the multiplier for each gate input i is updated by multiplying itself by the ratio of the current arrival time to input i, plus the input i to output j delay, plus ¼ of input i to output j slew, to the arrival time at output j plus ¼ the slew. Note that correct values for the output arrival time and slew rate have been chosen as given in (3).

Consider the equation below.

$$f(\mathbf{x}) = 10^9 x_1^2 + x_2^2 \tag{8}$$

It can be seen that $f(\mathbf{x})$ is extremely sensitive to changes in x_1 and not x_2. This is a poorly scaled problem and has a large impact on gradient-based approaches [17]. This problem can arise in our case when executing the delay-constrained area minimization phase. There we have the following form:

$$L(\mathbf{w},\boldsymbol{\lambda}) = f(\mathbf{w}) + \sum_i \lambda_i g_i(\mathbf{w}) \tag{9}$$

where f is the sum of sizes, λ_i are the Lagrange multipliers, and g_i are the constraint functions. Note that in the first phase when the algorithm minimizes the maximum delay, f is not present, thus, no scaling is required. If we apply the variable transformation $w = \exp(z)$ then f is convex. Multiplying by α (a positive variable) conserves convexity, and enables scaling between the area terms and the delay terms, as given below.

$$L(\mathbf{w},\boldsymbol{\lambda}) = \alpha f(\mathbf{w}) + \sum_i \lambda_i g_i(\mathbf{w}) \tag{10}$$

```
Initialize $\alpha = \alpha_0$, k = 1, $|\nabla L_0| = \infty$
while $|\nabla L_{k-1}| - |\nabla L_k| \ge$ tolerance
do    $\alpha_{k+1} = \alpha_k/10$
          $k = k+1$
end while
```

Figure 4. Algorithm to determine the scaling factor α. $|\nabla L_k|$ is the norm of gradient of L at iteration k. α_0=1e^{-3}

When the minimum delay bound is found, we determine the α term as follows. Starting with a positive α, we determine the norm of the gradient of (10). Then we progressively decrease α, until the difference between the current norm and the previous is less than a threshold. We found that a tolerance of 0.1 was sufficient. As the algorithm proceeds, since the multiplier values are updated to prevent bad scaling during algorithm execution, we also dynamically adjust α in a manner similar to the update scheme for the multipliers.

$$\alpha = \alpha \times \frac{A}{\max(a_i)} \tag{11}$$

where A is the delay target and the max is taken over all primary output arrival times a_i.

The minimization problems defined in (1) and (3) are termed the primal problems. The primal problems take on the following values:

$$\mathit{for}\ (1) \quad f(\mathbf{w}) = \begin{cases} \sum w_i & \mathit{if\ all\ constraints\ are\ satisfied} \\ +\infty & \mathit{else} \end{cases} \tag{12}$$

$$\mathit{for}\ (3) \quad f(\mathbf{w}) = a_0 \quad \mathit{the\ worst\text{-}case\ arrival\ time}$$

The maximizations of the functions $L(\mathbf{w},\boldsymbol{\lambda})$ defined in (5) and (6) with respect to the multipliers are the Lagrangian dual problems [18]. We have the following relation between the primal and the dual values.

$$f(\mathbf{w}) \ge L(\mathbf{w},\boldsymbol{\lambda}) \tag{13}$$

Further, if the primal and dual values are equal, then that point is a global optimum [18]. We use the above to show in practice that all our solutions are optimum, for the delay models that we use. Delay modeling is the subject of the next section.

3 Delay Modeling

We use a gate-level delay model that models rise/fall delays and rise/fall output slew-rates, for each input to output pair. The model itself is a function of the input rise/fall slew-rates, the nMOS and pMOS sizes, and the output load. As stated before, our cells are parameterized such that we use one variable to represent the nMOS device sizes for a gate and one variable for the pMOS device sizes for a gate.

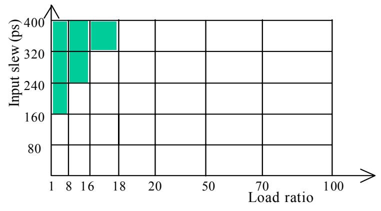

Figure 5. The organization of the piecewise delay models. The slew-rate increases uniformly by 80ps. The load ratio changes non-uniformly. Gray areas indicate pruned regions.

Previous approaches to cell characterization fit the simulation data for the entire range of sizes, input slew-rates, and output loading to one equation [7]. Here we propose an effective method to greatly improve the accuracy of the model by subdividing the data into small regions, and then fitting one function per region. The following points have to be taken into consideration. First, by what criteria do we divide the data set into smaller sets? Second, what are the implications of using a piecewise model for optimization? We will address these questions next.

3.1 Dividing and Conquering the Data Set

As stated in the problem formulation we have transistor size limits and beta ratio limits for gates. Since we are interested in being able to produce the area versus delay curve for a logic block for a wide range of delays, it is crucial to be able to handle a wide range of slew-rates and output loading. We therefore found it effective to divide the measurement data in two dimensions, where one is the input slew-rate and the other is the output load ratio (the latter is analogous to the notion of electrical effort [15]). The electrical effort (or load ratio) for a gate is defined as the output (load) capacitance divided by the capacitance of an input. We used load ratios ranging from 1 to 100. The entire slew-rate range was 20ps to 1.2ns. Thus each region has min and max load ratios and min and max input slew-rates, and the data set for a region includes all possible sizes and all possible beta ratios.

The characterization data is sub-divided into a two-dimensional grid such as that shown in Figure 5. We obtained very accurate fits (as will be shown in Table 2) using a uniform step size of 80ps for the input slew-rate. However, a non-uniform (increasing) step size was needed for the load ratio. We found that the measurement data for the smaller load ratios were harder to fit than the measurement data for the larger load ratios.

Load ratio range	Incremental step
1 – 16	0.8
16 – 20	1
20 – 30	2
30 – 50	5
50 – 70	10
70 – 100	30

Table 1. A non-uniform incremental step in load ratio is used for the characterization.

For smaller load ratios the 50%-50% gate delay becomes non-monotonic (*e.g.* negative) as the input slew-rate substantially increases. With increasing load ratios this behavior diminishes. Table 1 shows the load ratio step sizes we used. As stated earlier for very small load ratios and for large input slew-rates we can have instances when the 50%-50% gate delay becomes non-monotonic (*e.g.* negative) and hence meaningless. This non-monotonic behavior also greatly reduces the accuracy of the fit. Therefore, given the unrealistic delays and poor accuracy, we prevent the sizing algorithm from operating in this portion of the slew-rate/load-ratio space. When carrying out the fitting process, we monitor the average and standard deviation of the error of the fit for a region. If the average error exceeds 5%, we prune that region as well as those directly above it (corresponding to even larger input slew-rates) as shown in Figure 5.

3.2 Delay / Slew-rate Function Forms

Equation (14) gives the general form of the functions we use to fit the delay and slew-rate data. The functional form is a two-term posynomial that depends on the input slew-rate τ, the size of the nMOS devices in the gate w_n, the size of the pMOS devices in the gate w_p, and the output loading. Here $a_i \geq 0$, b_i, c_i, d_i, are real numbers.

$$f(\tau, w_n, w_p, load) = \sum_{i=1}^{2} a_i \tau^{b_i} w_n^{c_i} w_p^{d_i} (load)^{e_i} \quad (14)$$
$$load = \sum kC_j + \sum w_j$$

The output load is the sum of the output wiring load and the output gate loading. Since gate loading is not a pure capacitance (due to nonlinearities and Miller multiplication), we found it much more accurate to characterize the load in terms of microns of poly gate loading instead of capacitance. So that all loading can be represented by a single variable in the characterization tables, the capacitance of a length of wire is also modeled in terms of an equivalent length of poly gate load (ascertained by detailed simulations). The function in (14) is a generalized posynomial, and is convex under the transformation x = exp(z) if $e_i \geq 0$ [7]. We fit simulation data in each region, determined by load ratio and input slew, to equation (14) by using the non-linear curve-fitting package NLREG [16].

3.3 Optimization with a Piecewise Model

In all mathematical approaches to optimization, gradients of functions modeling the problem play a vital role.Since most interesting problems do not have closed form solutions, we must rely on iterative procedures that move from one point in the solution space to another based on gradients with or without second derivative

Gate	Delay		Slew-rate		regions	β_{min}	β_{max}
	$\overline{\text{Error}}$	σ_{Error}	$\overline{\text{Error}}$	σ_{Error}			
INV	0.21%	0.36%	0.17%	0.28%	1411	1	3
NAND2	0.38%	0.70%	0.35%	0.60%	1545	0.5	1.5
NAND3	0.26%	0.48%	0.24%	0.42%	1629	0.33	1
NAND4	0.20%	0.34%	0.19%	0.32%	1474	0.25	1
NOR2	0.53%	0.86%	0.47%	0.76%	1163	1	6
NOR3	0.35%	0.59%	0.29%	0.49%	914	1	9
NOR4	0.20%	0.37%	0.16%	0.28%	703	1	12
AOI21	0.33%	0.58%	0.29%	0.49%	1329	0.5	1.5
AOI22	0.20%	0.36%	0.18%	0.30%	1528	0.5	1.5
OAI21	0.31%	0.56%	0.28%	0.49%	1387	0.5	1.5
OAI22	0.20%	0.37%	0.18%	0.32%	992	0.5	1.5

Table 2 Shown is the average error and standard deviation of the fit over all regions for each gate, for delay and slew-rate. Also shown is the min and max beta ratios allowed, and the number of regions used for each gate.

information at the current point. Thus, the existence ofgradients is vital for the success of any mathematical algorithm.

Consider the case where the descent direction of the function fitted to region 1 points into region 2 and vice versa, for some points in close proximity to the boundary of the regions. A gradient-based algorithm may find itself trapped within this area. Also, if the solution finds itself exactly on the boundary, the gradients are not defined. To overcome this, our piecewise model includes half-overlapping of adjacent regions, and quarter-overlapping of diagonally situated regions.

This is illustrated in Figure 6. We also noted that the largest fitting errors for a region usually occur near the boundaries of a region, and this overlapping approach enables the search algorithm to avoid the boundary portions.

In the sizing algorithm, functions fitted to a particular region are active till either the slew-rate bounds or the load ratio bounds for that particular region are violated. Then an adjacent region's functions are active depending on the new slew-rate and load ratio values. The overlapping of regions allows the solution to venture into another region, and not immediately return to the former region. For example, refer to the dotted path shown in Figure 6 that plots the solutions for a set of algorithm iterations. The initial point is near the top of region 4, then over the next iterations, the

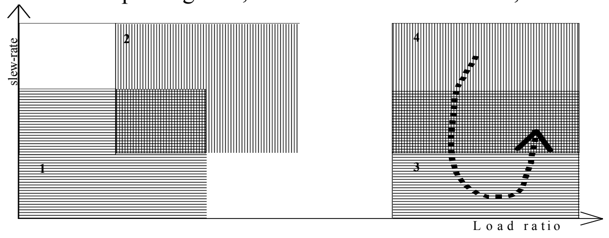

Figure 6. Diagonally placed regions 1 and 2 on the left, overlap by ¼. Adjacently placed regions 3 and 4 on the right overlap by ½. The doted line tracks iterative sizing steps.

load ratio remains the same but the input slew-rate decreases. The solution then crosses the overlap region for regions 3 and 4, still using the function in region 4. Finally, if further reduction in input slew-rate causes the solution to leave region 4 and enter region 3. Subsequently, if the input slew-rate increases, the solution again crosses the overlap region, but this time the function in region 3 is used. In practice we have not observed solutions getting trapped as happens when non-overlapping regions are used. Next we present the results for the delay modeling and area versus delay curve generation.

4 Results

Our library consisted of 11 inverting gates. The characterizations were performed for the 0.18μm TSMC process. Transistor widths were varied from 0.5μm to 12μm, with each gate having its own beta-ratio range as shown in Table 2. With regards to the entire data set, the slew-rate ranges from 20ps to 1.2ns, and load ratios from 1 to 100 (where the actual loading in terms of microns of poly gate loading ranged from 1μm to 2400 μm). Also shown in Table 2 is the number of regions used for each gate.

Table 3 summarizes the results for 31 designs taken from the ISCAS'85 and ITC'99 benchmark sets. At each solution point the transistors are rounded to the nearest tenth of a micron. Then the rise and fall critical paths are extracted and the corresponding Hspice delays are measured. Column 3 gives the average error for a total of 22 extracted paths for the 11 points generated on the trade-off curve, between the measured Hspice values and the Forge calculated values. Comparisons with CTST are also given in Table 3.

Starting with minimum sized designs, CTST is run using both delay requirement mode and cost function mode, with the same beta ratio and transistor size constraints. We use the method given in [2] to obtain the area versus delay trade-off curve. In column 5 we show the average area savings achieved by Forge over CTST, where we only performed the averaging over the target delay ranges that were both achieved by Forge and CTST Column 6 gives the runtime in seconds for Forge, running on a 3.2 GHz Pentium Xeon machine running Linux. Since our version of CTST was only available on a Sun UltraSparc-60 workstation, we also ran Forge on this machine to obtain the runtime improvement factor. Note that for designs b20, b22, and b17 CTST was terminated after failing to complete within 3 days.

Design	Cells	Error	Area Red.	CPU (s)	Impr. runtime
C17	7	2.79%	14.7%	1.5	7.2
b1	12	3.24%	10.8%	1.82	6.5
cm151a	29	3.18%	15.6%	3.16	7.2
cm150a	54	4.20%	38.9%	5.74	7
cu	62	5.23%	11.4%	4.71	7.4
cc	68	4.36%	17.2%	4.7	9.3
b9	116	5.03%	17.6%	7.15	9.7
apex7	286	3.86%	48.3%	21.	7.8
t481	411	3.53%	28.0%	49.1	3.9
C880	503	3.23%	25.0%	63.8	4.6
alu2	530	3.16%	41.2%	100	2.6
C1355	582	3.84%	19.2%	127	6.0
vda	665	4.29%	30.7%	65.3	5.7
C2670	770	6.97%	41.9%	41.9	11.7
rot	804	4.37%	32.1%	64.3	7.8
apex6	883	4.81%	23.7%	113	4.5
dalu	1046	5.21%	16.6%	87.9	6.5
apex5	1055	3.55%	28.6%	123	4.8
frg2	1084	4.83%	32.4%	128	5
k2	1086	4.44%	41.5%	132	4
too_large	1301	4.61%	27.6%	144	4.8
C3540	1812	5.12%	32.0%	272	6.5
C5315	2381	6.19%	23.3%	284	5.4
C7552	2539	4.92%	22.3%	333	5
seq	3198	5.08%	28.4%	346	6.1
i10	3306	6.78%	25.2%	433	7.5
des	4757	4.14%	32.7%	502	7.6
C6288	5489	4.45%	63.5%	1712	6.5
b20	21920	5.29%	n/a	4187	n/a
b22	31635	6.02%	n/a	6210	n/a
b17	44615	6.19%	n/a	7542	n/a
Average		4.61%	28.6%		6.4

Table 2. The number of cells in each design is shown, as is the average % absolute error compared to Hspice, and the average reduction in area compared to the commercial tool's solutions. Runtime and runtime improvement factor compared to the commercial tool are shown as well.

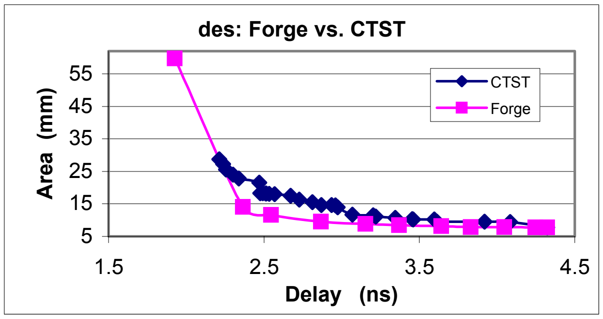

Figure 7. Area versus delay curves for des. Forge finds a 1.14X faster design, and has 32.74% less transistor area.

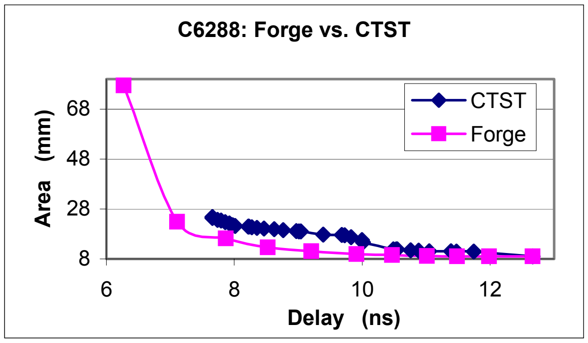

Figure 8. Area versus delay curves for C6288. Forge finds a 1.22X faster design, and has 63.45% less transistor area.

Note that Forge has a significant runtime and area savings advantages over CTST, on average 6.4X faster while averaging 28% less area per design point on the area versus delay trade-off curve.

For all of the benchmarks, and for each of the 11 design points produced by Forge, the difference between the primal and dual function values were less than 1%, and all delay constraints were satisfied within a tolerance of 10ps.

The area versus delay trade-off curves for des and C6288 produced by Forge and CTST are shown in Figures 7 and 8. Note for C6288 that Forge produced a design that was 1.22X faster and the average area savings was 63%. For des Forge solution was 1.14X faster with average area savings of 33%.

5 Conclusions

We presented an efficient and accurate gate sizing tool that employs a novel piecewise convex delay model for static CMOS gates. The model handles distinct rise/fall delays and rise/fall slew-rates. The delay model is used in a new version of a gate-sizing tool called Forge, which not only exhibits optimality, but also efficiently produces the area versus delay trade-off curve for a block in one step. Additional robustness optimizations were added to the Lagrangian relaxation solver used in Forge. Forge includes a realistic delay propagation scheme that combines arrival times and slew-rates. We showed that Forge is 6.4X faster, on average, than CTST, while producing faster circuits and using 28% less transistor area (on average) for the target delay range. Forge produces the complete area-delay trade-off curve for a 44,000-gate design in two hours, on a Pentium Xeon PC (3.2 GHz) running Linux.

6 References

[1] M. delM. Hershenson, S.P. Boyd, and T.H. Lee, “Optimal design of a CMOS op-amp via geometric programming,” IEEE Trans. on Computer-Aided Design, vol. 20, no 1, pp. 1-21, Jan 2001.

[2] M. Vujkovic and C. Sechen “Optimized power-delay curve generation for standard cell ICs,” Proc. Int’l. Conf. on Compter-Aided Des. (ICCAD), pp. 387-394, Nov 2002.

[3] M. Vujkovic, D Wadkins, B Shwartz, and C. Sechen, “Efficient timing closure without timing driven placement and routing,” Proc. Des. Auto. Conf. (DAC), pp. 268-273, June 2004.

[4] M.R.C.M. Berkelaar, P.H.W. Buurman, and J. A. G. Jess, “Computing the entire active area/power consumption versus delay tradeoff curve for gate sizing with a piecewise linear simulator,” IEEE Trans. on Computer-Aided Design, vol. 15, no. 11, pp. 1424-1434, Nov 1996.

[5] J. P. Fishburn and A. E. Dunlop, “TILOS: A posynomial programming approach to transistor sizing,” IEEE Trans. on Computer-Aided Design, pp. 326-328, Nov 1985.

[6] W.C. Elmore, “The transient analysis of damped linear networks with particular regard to wideband amplifiers.” Journal of Applied Physics, vol. 19, no. 1, pp. 55-63, 1948.

[7] K. Kasamsetty, M. Ketkar, and S. S. Sapatnekar, “A New Class of Convex Functions for Delay Modeling and their Application to the Transistor Sizing Problem,” IEEE Trans. on Computer-Aided, vol. 19, no. 7, pp. 779-788, July 2000.

[8] Jim-Fuw Lee, D.L. Ostapko, J. Soreff, C.K. Wong, “On the signal bounding problem in timing analysis”, Proc. Int’l Conf. on Compter-Aided Design, pp. 507-514, Nov 2001.

[9] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya, and S. M. Kang, “An exact solution of the transistor sizing problem for CMOS circuits using convex optimization,” IEEE Trans. on Computer-Aided Design, vol. 12, pp. 1621-1634, Nov 1993.

[10] V. Sundararajan, S. S. Sapatnekar, and K. K. Parhi, “Fast and Exact Transistor Sizing Based on Iterative Relaxation,” IEEE Trans. on Computer-Aided Design, vol. 21, no. 5, pp. 568-581, May 2002.

[11] C. P. Chen, C. C. N. Chu, and D.F. Wong, “Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation,” Proc. Int’l Conf. on Computer-Aided Design, pp. 617-624, Nov 1998.

[12] H Tennakoon, and C. Sechen, “Gate sizing using Lagrangian relaxation combined with a fast gradient-based pre-processing step,” Proc. Int’l. Conf. on Computer-Aided Design, pp. 395-402, Nov 2002.

[13] A. R. Conn, P. K. Coulman, R. A. Harring, G. L. Morril, C. Visweshwariah, and C. W. Wu, “JiffyTune: Circuit optimization using time-domain sensitivities,” IEEE Trans. on Computer-Aided Design, vol. 17, no. 12, pp. 1292-1309, Dec 1998.

[14] A. R. Conn, I. M . Elfadel, W. W. Molzen, Jr, P. R. O’Brien, P. N. Strenski, C. Visweswariah, and C. B. Whan, “Gradient-Based optimization of custom circuits using static-timing formulation,” Proc. Des. Auto. Con. (DAC) pp. 452-459, June 1999.

[15] Ivan E. Sutherland, Robert F. Sproull, David Harris, Logical Effort: Designing Fast CMOS Circuits, First Edition, Morgan Kaufmann, 1999.

[16] J.E. Dennis, D. M. Gay, and R. E. Welsch, “An adaptive nonlinear least-squares algorithm,” ACM Trans. on Math. Software 7,3 Sept. 1981.

[17] Jorge Nocedal, and Stephen J. Wright, Numerical Optimization, Springer-Verlag, 1999.

[18] M. S. Bazaraa, H. D. Sherali, C. M. Shetty, Nonlinear Programming: Theory and Algorithms, Second Edition, John Wiley and Sons, 1993