# Constrained Circuit Optimization Using Logical Effort

George Chen, *Member, IEEE,* Jo Ebergen, *Member, IEEE,* and Jiyang Cheng

***Abstract*—We present a simple and efficient algorithm that computes the trade-off curve for path delay and total area of a circuit path. Each point on the trade-off curve represents the minimum area for a given delay constraint or, conversely, the minimum path delay for a given area constraint, where gate sizes are the only variables. The graph of the area-versus-delay function permits the designer to choose the best combination of transistor area and path delay by choosing the proper gate sizes. A similar algorithm efficiently computes the trade-offs between path delay and energy dissipation of the path. The algorithms permit any single circuit path, including those that contain loops. For such cyclic paths, the path delay is the cycle time and the constraint for the path delay is the constraint for the cycle time. The derivation of the algorithm is based on the theory of Logical Effort and Langragian relaxation.**

***Index Terms*—Logical Effort, Optimization, Area, Energy.**

## I. Introduction

TIMING closure is challenging to manage and achieve in advanced microprocessor design due to the sheer size of the chip and the multitude of designers involved. In a typical advanced microprocessor design, tens, if not hundreds, of individual blocks are hierarchically assembled. After assembling the blocks, the design group, typically a central "integration" team, runs static timing analysis to find paths that don't meet the timing requirements.

A path which fails may pass through two or more blocks owned by different engineers and assembled by a third. These engineers then attempt to find a solution for the violating path, whether by resizing gates, introducing more buffers, or possibly a more expensive architectural change.

One method that can be used in analyzing such violations is Logical Effort. This technique was first introduced over 15 years ago as a method to easily minimize the delay through a series of logic gates [1] to allow designers to analyze circuit topologies for speed. Since then, designers have used logical effort in the design of combinational and asynchronous circuits [2]–[6] .

Within the CAD domain, tool developers have started to harness logical effort for optimization: [7] examines technology mapping when the libraries and associated delay calculations are based upon logical effort, [8] uses logical effort as the delay model in exploring heuristic gate sizing algorithms with discrete-sized cells, and [9] describes LEOPARD, an area-delay optimization tool.

G. Chen and J. Cheng are with Processor CAD group at Sun Microsystems, Sunnyvale, CA.

J. Ebergen is with Sun Labs at Sun Microsystems, Menlo Park, CA.

Manuscript received TBD; revised TBD.

Our work extends and combines these domains by providing a simple analytical model, based on Logical Effort, that can be used to study area-delay or energy-delay tradeoffs along a single path. This technique can be used to assist designers in solving paths with long interconnect, typically those that cross block boundaries. By means of a number of simple examples we show that generating a trade-off curve for single-path delays is extremely easy, and can be done almost on the back of an envelope. Our generation of trade-off curves avoids pitfalls of starting in infeasible points [10] or scaling issues [10], [11]. Our method allows circuits with feedback.

There are many papers related to constrained optimization. Most of the related work deals with the more general problem of optimization of combinational circuits, which have multiple paths between inputs and outputs and have no feedback loops. We only deal with single paths, but those paths may contain feedback loops. The paper on TILOS [12] was one of the first optimization methods to solve the problem of minimization of total transistor area subject to certain delay constraints. Several papers have proposed improved and more accurate methods to solve the constrained optimization problem for multi-path networks, in particular [10], [13], and more recently [14]. Chen and others propose a solution to the constrained optimization problem for multi-path networks in [15] using Lagrangian relaxation, a technique we use as well. Tennakoon and Sechen suggest an improvement in [11] to the method in [15]. Because these methods deal with multiple paths between inputs and outputs instead of a single path, these methods are more complicated. The method of Lagrangian relaxation is used also in [16], [17] for solving a constrained minimization problem involving wire sizing.

## II. Review and Notation

For our delay model we use a simple Elmore RC delay model that is similar to the delay model in Logical Effort [1]. In this section we briefly derive this model from the Elmore RC model and explain how it differs from the Logical Effort model.

In Figure 1(a), gate 0 drives gate 1 and a fixed load represented by $C_L$. Figure 1(b) represents our equivalent delay model. $C_{p0}$ represents the intrinsic or parasitic capacitance for gate 0. $C_{in1}$ represents the input capacitance for gate 1. The Elmore RC model yields the following delay $d_0$ for gate 0.

$$d_0 = k(R_0C_{p0} + R_0C_L + R_0C_{in1})$$

for some constant $k$.

In our delay model we assume that the drive resistance of a gate scales inversely proportional with the size of a gate and

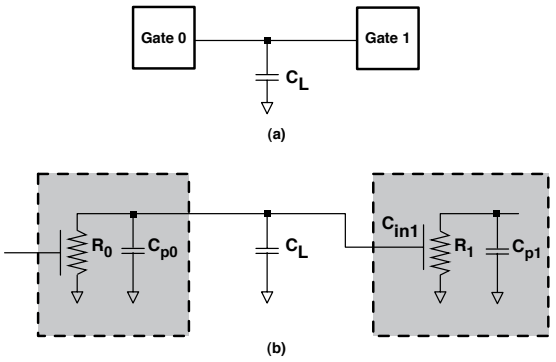


Fig. 1. The delay model. (a) Gate 0 drives gate 1 and load $C_L$ (b) Equivalent model with R's and C's

that the input and intrinsic capacitances scale linearly with the size of the gate. Thus, if $x$ represents the scale factor of a gate, $r$ represents the drive resistance of that gate of size 1, $c_{in}$ represents the input capacitance of that gate of size 1, and $c_p$ represents the intrinsic capacitance of that gate of size 1, then $R(x) = r/x$, $C_{in}(x) = x * c_{in}$, and $C_p(x) = x * c_p$. Besides being called the scale factor, $x$ is also called the size of the gate or the drive strength of the gate. With these assumptions the delay for gate 1 in Figure 1 can rewritten as

$$d_0 = k((r_0/x_0)(x_0 c_{p0}) + (r_0/x_0)C_L + (r_0/x_0)(x_1 c_{in1}))$$

Subsequently we simplify and normalize this expression using the drive resistance $r_{inv}$ and input capacitance $c_{inv}$ of a unit inverter.

$$d_0 = (k r_{inv} c_{inv})[(r_0 c_{p0}/r_{inv} c_{inv}) + (r_0 C_L/x_0 r_{inv} c_{inv}) + (r_0 x_1 c_{in1}/x_0 r_{inv} c_{inv})]$$

The next simplifying assumption is that the drive resistances of all gates of unit size are equal. In other words $r_i = r_{inv}$ for any gate $i$. Let us furthermore define $\tau = k r_{inv} c_{inv}$. We get

$$d_0 = \tau[c_{p0}/c_{inv} + (C_L/x_0 c_{inv}) + (x_1 c_{in1}/x_0 c_{inv})]$$

Define $p_0 = c_{p0}/c_{inv}$, $g_1 = c_{in1}/c_{inv}$ and measure delays in units of $\tau$, then

$$d'_0 = p_0 + (C'_L + g_1 x_1)/x_0 \quad (1)$$

The value $p_0$ represents the parasitic delay of gate 0 expressed in units of $\tau$. The value $p_0$ is equal to the amount of intrinsic capacitance of gate 0 of unit size relative to the input capacitance of an inverter of unit size. The value $g_1$ represents the logical effort of the input of gate 1. The value $g_1$ is equal to the amount of input capacitance of gate 1 of unit size relative to the input capacitance of an inverter of unit size. Both $p_0$ and $g_1$ can be seen as time constants expressed in units of $\tau$.

The traditional Logical Effort [1] method expresses the gate delay for gate 0 in units of $\tau$ as

$$d'_0 = p_0 + g_0 * h \quad (2)$$

where $d'_0$ is the normalized delay of gate 0, $p_0$ is the intrinsic delay of gate 0, $g_0$ is the logical effort of gate 0, and $h$ is the electrical effort equivalent to the output capacitance divided by input capacitance of gate 0. In other words,

$$h = (C'_L + g_1 x_1)/g_0 x_0$$

Although expressions 1 and 2 express the same delay, we use the notation introduced in 1 for the following.

As in Logical Effort, we normalize delay and capacitance in all our equations to the delay and capacitance of a unit-size inverter. We measure delays in terms of $\tau = k r_{inv} c_{inv}$. We express capacitance in terms of $\kappa = c_{inv}$, which is the input capacitance of a unit-size inverter.

The advantage of using Logical Effort is that expressions for delay as in (1) mostly stay the same when changing from one technology to another. For example, the values for the logical efforts $g$ and parasitic delays $p$ mostly remain constant for all gates. The only values that change are $\tau$ and $\kappa$. This abstraction from a particular technology allows comparisons between circuits independent of the implementation technology.

## III. The Problem

Consider the situation in Figure 2, a path consisting of $N + 1$ gates with $N + 1$ side loads. For the moment we ignore resistive wires; we include those later. Before we can

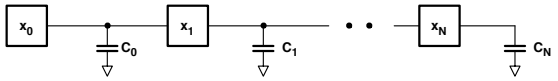


Fig. 2. A single circuit path consisting of $N + 1$ gates and $N + 1$ side loads

formulate the constrained minimization problems, we have to find expressions for the path delay $D(x)$, path area $A(x)$, and dynamic energy dissipation $E(x)$ of the path.

The gate delay $d_i$ of gate $i$ depends on the sizes $x_i$ as follows for $0 \leq i \leq N$

$$d_i = p_i + \frac{C_i + g_{i+1} x_{i+1}}{x_i} \quad (3)$$

where $x_{N+1} = 0$ and $g_{N+1} = 0$. The path delay $D(x)$ is simply the sum of all gate delays:

$$D(x) = \sum_{i=0}^{N} (p_i + C_i/x_i + g_{i+1} * x_{i+1}/x_i) \quad (4)$$

Again, in this formula $x_{N+1} = 0$ and $g_{N+1} = 0$.

Finding an expression for the total transistor area of a path is a bit simpler than finding an expression for path delay. As a measure for the total transistor area of a path, we add all transistor widths of the gates and require that all transistors are of the same (minimum) length. In order to normalize area measurements, we define the unit for area as the total transistor width of a minimum-sized inverter. We denote this area by $\alpha$.

Because the total transistor width of a gate is proportional to the total input capacitance of a gate, the total transistor width of a single gate is proportional to $\sum_j (g_{j,i} * x_i) = (\sum_j g_{j,i}) * x_i$, where $j$ ranges over all inputs of gate $i$. Let us denote the sum of all logical efforts of inputs of gate $i$, also called the total logical effort of gate $i$, by $a_i$, that is, $\sum_j g_{j,i} = a_i$. As an

example, the total logical effort of an inverter is 1. With this definition, a measure for the total transistor area of all gates on the path as a function of the vector of sizes $x$ is

$$A(x) = \sum_{i=0}^{N} a_i * x_i \tag{5}$$

Finding an expression for the total dynamic energy dissipation of a path is also easy. When gate $i$ drives its output capacitance, it dissipates in its drive resistance an energy proportional to the total output capacitance. The constant of proportionality is $\epsilon = 0.5 c_{inv} V_{dd}^2$, which is our unit of energy. In our energy measurements we only look at dynamic energy dissipation and ignore static energy dissipation due to leakage currents for example. We also assume that the short-circuit energy dissipation is small compared to the dynamic energy dissipation due to charging and discharging capacitances. Provided that input slews are not too large, this is a valid assumption [18]. The total energy dissipated by the path as a function of the vector $x$ of sizes, where each gate drives its output once, is proportional to the sum of all capacitances on the path, where the constant of proportionality is $\epsilon$.

$$E(x) = \sum_{i=0}^{N} (p_i * x_i + g_{i+1} * x_{i+1} + C_i) \tag{6}$$

$$= \sum_{i=0}^{N} ((p_i + g_i) * x_i + C_i) - g_0 x_0 \tag{7}$$

where $g_{n+1} = 0$ and $x_{n+1} = 0$.

With expressions for path delay, path area, and path energy dissipation, we can formulate our problems formally. The problem statement for area minimization under a delay constraint becomes

*Find the vector of sizes $x$ that minimizes $A(x)$ subject to the constraint that $D(x) \leq D_0$.*

The problem statement for delay minimization under an area constraint becomes

*Find the vector of sizes $x$ that minimizes $D(x)$ subject to the constraint that $A(x) \leq A_0$.*

Similarly, the problem statement for energy minimization under a delay constraint becomes

*Find the vector of sizes $x$ that minimizes $E(x)$ subject to the constraint that $D(x) \leq D_0$.*

The problem statement for delay minimization under a energy constraint becomes

*Find the vector of sizes $x$ that minimizes $D(x)$ subject to the constraint that $E(x) \leq E_0$.*

## IV. Convex Optimization

Because each of the functions $D(x)$, $A(x)$, and $E(x)$ are convex functions, each of the constrained minimization problems is a well-known convex optimization problem [19]. There are many general techniques to solve these optimization problems. We use the technique called Lagrangian relaxation. The problem for area minimization

*Find $x$ that minimizes $A(x)$ subject to $D(x) \leq D_0$.*

is equivalent to the Lagrangian relaxation problem

*Find $x, \lambda > 0$ that minimize*

$$A(x) + \lambda(D(x) - D_0) \tag{8}$$

The parameter $\lambda$ is the Lagrange multiplier [19].

To construct the area-delay trade-off curve, we have to find, for a series of delay constraints $D_0$, the minimum area $A$. Because this minimum area is a function of constraint $D_0$, we denote this area as $A_{min}(D_0)$. For each value of $D_0$ we can apply Lagrangian relaxation and find the values of $x_0, \lambda_0 > 0$ that minimize $A(x) + \lambda(D(x) - D_0)$. The value of $A(x_0)$ is then the value $A_{min}(D_0)$ we are looking for. Of course, when choosing $D_0$, we have to be careful to choose a value $D_0$ that is at least the minimum path delay $D_{min}$, otherwise the problem becomes infeasible.

Instead of minimizing $A(x) + \lambda(D(x) - D_0)$ over all $x$ and $\lambda$ for each choice of $D_0$, we apply a slightly different method. For each $\lambda$, we compute the value $x_\lambda$ that satisfies

$$\nabla A(x) = -\lambda \nabla D(x) \tag{9}$$

where $\nabla$ denotes the gradient in $x$, the vector of first-order derivatives to $x$. Equation (9) expresses the condition that the first order derivative to $x$ of $A(x) + \lambda(D(x) - D_0)$ is 0. The value $x_\lambda$ minimizes the function $A(x) + \lambda D(x)$, for each choice of $\lambda$. Because the function $A(x) + \lambda D(x)$ is convex, any local minimum is the global minimum, and as a consequence the value of $x_\lambda$ is unique.

In order to also satisfy that the first-order derivative to $\lambda$ of $A(x) + \lambda(D(x) - D_0)$ is 0, we need to satisfy

$$D(x) = D_0 \tag{10}$$

Thus, if for each $\lambda$, we choose $D_0$ such that $D_0 = D(x_\lambda)$, then for $x = x_\lambda$, equation (10) is also satisfied. In other words, for each choice of $\lambda$, $x_\lambda$ minimizes $A(x)$ subject to $D(x) \leq D(x_\lambda)$. Consequently, the points $(A(x_\lambda), D(x_\lambda))$ are the points on the area-delay trade-off curve for all choices of $\lambda > 0$.

Let us contrast the two methods for solving the constrained minimization problem. The traditional method minimizes $A(x) + \lambda(D(x) - D_0)$ over all $x$ and $\lambda$ for each choice of $D_0 \in [D_{min}, \infty)$, where $D_{min}$ is the minimum path delay. Our proposed method consists of calculating $x_\lambda$ satisfying (9) for each choice of $\lambda \in [0, \infty)$. There are two reasons why we prefer our method. First, the range for $\lambda$ is $[0, \infty)$ for every problem, whereas the range for $D_0$ depends on the minimum path delay $D_{min}$, which may be different for every problem. In order to obtain a feasible solution, we must choose a value for $D_0$ that is larger than the minimum path delay. The second reason is that solving (9) for a given $\lambda$ turns out to be very easy, as explained in a next section.

To solve our constrained optimization problems, we must find the gradients in $x$ for the delay, area, and energy function $D$, $A$, and $E$ respectively. Fortunately, this is easy. The only variables for an acyclic path are sizes $x_1$ through $x_n$. Recall that for an acyclic path, size $x_0$ is fixed. Taking the gradient of (5), we obtain for $0 < i \leq N$

$$(\nabla A)_i = \frac{\partial A}{\partial x_i} = a_i \tag{11}$$

Similarly, for the total energy, by taking the gradient of (7), we find $0 < i \leq N$

$$(\nabla E)_i = \frac{\partial E}{\partial x_i} = p_i + g_i \tag{12}$$

These gradients are constant vectors represented by logical effort values. For the gradient of $D$ we find from (4)

$$(\nabla D)_i = \frac{\partial D}{\partial x_i} = (g_i/x_{i-1} - (C_i + g_{i+1} * x_{i+1})/x_i^2) \tag{13}$$

where $g_{N+1} = 0$ and $x_{N+1} = 0$. Substituting (13) and (11) into (9), we find that the following equations for $0 < i \leq N$

$$a_i = -\lambda * (g_i/x_{i-1} - (C_i + g_{i+1} * x_{i+1})/x_i^2) \tag{14}$$

where $x_0$ is a fixed value, $g_{N+1} = 0$, and $x_{N+1} = 0$. In a later section we shall describe how to solve these equations efficiently for $x$.

## V. A Graphical Illustration

Let us briefly explain our method and equation (9) by means of a figure. We illustrate graphically the point that minimizes the area $A(x)$ under the constraint $D(x) \leq D_0$. Figure 3 gives a contour plot of a delay function $D(x)$ of two variables $x_1$ and $x_2$ for some simple circuit. For the particulars of the circuit we refer to section IX, although those specifics may be ignored for this illustration. The delay contours are a series of nested deformed circles. Each delay contour indicates all points with the same path delay. The increments of the contour values are 10% of the minimum value. Notice that the path delay around the minimum 12.6 is very flat. The gradient $\nabla D$ in a point on a delay contour is a vector that points in the direction of steepest increase, and the length of the vector indicates the size of the increase. The gradient in a point is always perpendicular to the tangent of the delay contour.

The area function is a linear function of vector $x$ and its contours are straight lines. We have shown one area contour in Figure 3 by a bold black line. Area contours with a higher value are further from the origin and area contours with a smaller value are closer to the origin. The gradient of the area function is a constant vector for all points $(x_1, x_2)$.

Let us say that the delay constraint is $D(x) \leq 13.9$. We want to find the minimum area $A(x)$ satisfying this delay constraint. We can find this point graphically by finding the contour for the area function that is closest to the origin and intersects the contour $D(x) = 13.9$. In other words, we have to find the area contour that is closest to the origin and that *touches* the contour $D(x) = 13.9$. In the point where both contours touch, their gradients are colinear, but point in opposite direction, as illustrated in Figure 3. This point is uniquely defined by the equation

$$\nabla A(x) = -\lambda \nabla D(x) \quad \text{for some } \lambda > 0 \quad \text{and} \quad D(x) = 13.9$$

Alternatively, if you fix $\lambda$ instead of $D_0$, there is a unique value $x_\lambda$ that satisfies $\nabla A(x) = -\lambda \nabla D(x)$. The point $A(x_\lambda), D(x_\lambda))$ defines a point on the area-delay trade-off curve.

A similar reasoning applies for minimizing the total energy dissipation $E(x)$ of a path under a delay constraint for the path. Like $\nabla A(x)$, $\nabla E(x)$ is a constant vector consisting of logical effort values for all values of $x$. We find for each $\lambda$, the value $x_\lambda$ that satisfies

$$\nabla E(x) = -\lambda \nabla D(x) \tag{15}$$

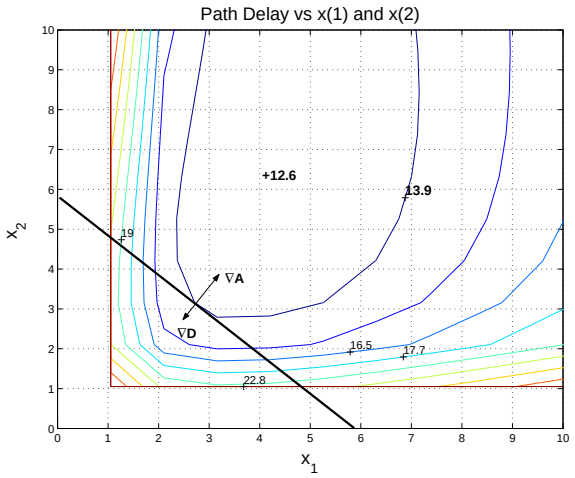


Fig. 3. Illustration of gradients of $A(x)$ and of $D(x)$ and $\nabla A(x) = -\lambda \nabla D(x)$

## VI. Plotting The Area vs. Delay Curve

We can plot the area $A(x_\lambda)$ versus delay $D(x_\lambda)$ for all values of $\lambda$. An example plot appears in Figure 4 for a finite range of values for $\lambda$. This plot shows the trade off between the path delay and the total area. The designer can find the minimum area for each path delay or the minimum delay for each area constraint. For the chosen combination of path delay

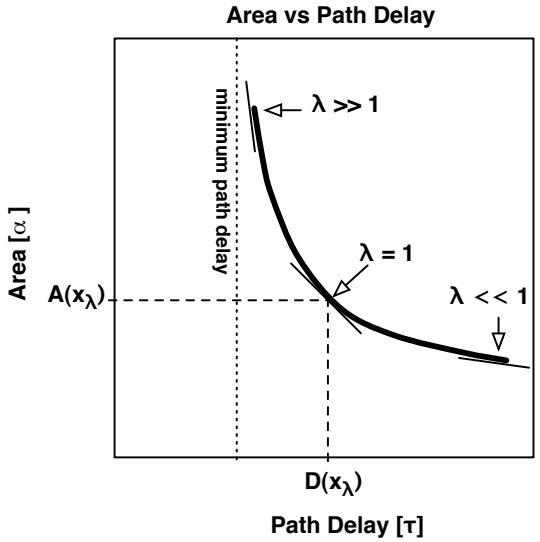


Fig. 4. An example plot of area $A(x_\lambda)$ versus delay $D(x_\lambda)$ for a finite range of values for $\lambda$. The value of $-\lambda$ is the tangent of the point $(D(x_\lambda), A(x_\lambda)$ on the area-versus-delay function. For each $\lambda$, $A(x_\lambda)$ is the minimum area for delay constraint $D(x_\lambda)$ for the path. $D(x_\lambda)$ is the minimum path delay for area constraint $A(x_\lambda)$ for the path.

$D(x_\lambda)$ and area $A(x_\lambda)$, the vector $x_\lambda$ gives the sizes of the gates on the path.

The tangent of the area-versus-delay function in the point $(D(x_\lambda), A(x_\lambda))$ is $-\lambda$. In each point on the area-versus-delay curve, the value of $\lambda$ indicates the trade off between area and delay. A small change $\delta$ in $D(x_\lambda)$ creates a change $-\lambda * \delta$ in $A(x_\lambda)$. Conversely, a small change $\delta$ in $A(x_\lambda)$ creates a change $-\delta/\lambda$ in $D(x_\lambda)$. For example, in the point that corresponds to $\lambda = 1$, we can trade one unit $\tau$ of extra delay for one unit $\alpha$ of less area.

Note that for $\lambda$ approaching infinity, $x_\lambda$ approaches the value that minimizes the path delay $D(x)$ without any constraint. For acyclic paths, the value for $x_\lambda$ that achieves the minimum path delay is finite, and thus the area for minimum path delay is finite. For cyclic paths, the minimum cycle time is unattainable for finite values of $x_\lambda$. For cyclic paths, the area goes to infinity as the cycle time approaches the minimum cycle time. The minimum cycle time is an asymptote for the area-versus-delay function.

Because the delay function around the minimum path delay is always very flat, the area-versus-delay curve always has a long steep part close to the minimum path delay, as illustrated in Figure 4. This part indicates that for a small change in path delay there can be a large change in area. In other words, a small decrease in path delay close to the minimum comes with a high cost in terms of increase in area.

## VII. Finding $x_\lambda$ By Coordinate-Wise Descent

For a given $\lambda$, we can find the vector $x_\lambda$ that satisfies equation (9) by coordinate-wise descent with exact line search [19]. Although there are many other methods to find $x_\lambda$, we have found that the method of coordinate-wise descent is simple and converges very rapidly to the solution. Coordinate-wise descent has been applied in many other papers on circuit optimization by means of gate or wire sizing ([15], [17], [20], [21], [11]).

Initially the algorithm can start at any vector $x$. In absence of any good choice of start vector, you can use the vector $x_i = 1$ for all $0 < i \leq N$.

For each $i$ from $N$ to 1, the algorithm updates coordinate $x_i$ using equation (9), where we assume that all variables except $x_i$ are fixed. After each update of a variable in $x_N$ through $x_1$, the function $A(x) + \lambda D(x)$ decreases or remains the same. After several rounds of updates, the variables $x_N$ through $x_1$ rapidly approach the minimum of the function. When the relative change in each $x_i$ is less than a prescribed tolerance, the algorithm terminates.

The algorithm calculates each update to variable $x_i$, $0 < i \leq N$, as follows. Solving (14) for $x_i$, we find

$$x_i = \sqrt{\frac{C_i + g_{i+1} * x_{i+1}}{(a_i/\lambda) + g_i/x_{i-1}}} \tag{16}$$

We call the right-hand side of this equation the update function for variable $x_i$.

$$update_i(x_{i+1}, x_{i-1}, \lambda) = \sqrt{\frac{C_i + g_{i+1} * x_{i+1}}{(a_i/\lambda) + g_i/x_{i-1}}}$$

The algorithm is as follows. Here $x^{(k)}$ is the $k$-th approximation of $x_\lambda$.

$x^{(0)} := \mathbf{1}$;
$k := 0$;
**repeat** $k := k + 1$;
  **for** $i = N$ **to** 1 **do**
    $x_i^{(k+1)} := update_i(x_{i+1}^{(k+1)}, x_{i-1}^{(k)}, \lambda)$
  **end**
**until** $||x^{(k+1)} - x^{(k)}|| < tol * ||x^{(k+1)}||$

Note that if you want to start the algorithm with $x_i = 0$, for all $0 < i \leq N$, the above formulas must be rewritten to avoid division by 0.

When the path is cyclic, some updates are slightly different. First, size $x_0$ of gate 0 is no longer fixed, but variable. Consequently, the indices are from 0 through $N$. Second, because each of the gates 0 through $N$ is part of a cycle, the addition in indices is modulo $N + 1$.

When we want to find the minimum energy dissipation of a path under a delay constraint, we find for each $\lambda$ the value $x_\lambda$ that satisfies

$$\nabla E(x) = -\lambda \nabla D(x)$$

where $(\nabla E)_i = p_i + g_i$ for $0 < i \leq N$. This condition is similar to that for area minimization. In other words, to find $x_\lambda$ for energy minimization, we only need to substitute $p_i + g_i$ for $a_i$ in the algorithm for area minimization.

## VIII. Constraints For Sizes

So far we have not put any constraints on the gate sizes. Gate sizes often have two types of constraints. First, gate sizes can assume only values between a lower bound $L_i$ and upper bound $U_i$, which may be specific to each gate $i$.

$$L_i \leq x_i \leq U_i$$

Second, gate sizes often assume only discrete values, rather than continuous values. How can we incorporate these two constraints? This section discusses several ways to deal with these constraints following ideas proposed in [20], [22].

Before we deal with the two constraints, we observe a property of the update function. The update function $update_i$ for variable $x_i$, is a monotonic function in $x_{i+1}$, $x_{i-1}$, and $\lambda$. This means that for two points $p0 = (x0_{i+1}, x0_{i-1}, \lambda 0)$ and $p1 = (x1_{i+1}, x1_{i-1}, \lambda 1)$

$$\text{if} \quad p0 \leq p1 \quad \text{then} \quad update_i(p0) \leq update_i(p1)$$

The monotonicity property still holds if we change the update function to

$$update'_i(x_{i+1}, x_{i-1}, \lambda) = \min\left(U_i, \max\left(L_i, \sqrt{\frac{C_i + g_{i+1} * x_{i+1}}{(a_i/\lambda) + g_i/x_{i-1}}}\right)\right)$$

for each $0 < i \leq N$. The modified update function makes sure that the value after each update stays within the interval $[L_i, U_i]$.

The consequence of the monotonicity property for the modified update function is that for each value of $\lambda$ and start vector $x_i^{(0)} = L_i$, for each $0 < i \leq N$, the approximation algorithm produces a monotonically increasing sequence $x^{(0)} \leq x^{(1)} \leq x^{(2)} \leq ...$ This sequence converges to $x_{\lambda,i}$, unless for some $i$, $x_i^{(k)}$ remains stuck at its lower bound $L_i$ or its upper bound $U_i$.

Similarly, we can start the approximation algorithm at the upper bounds, in which case the algorithm produces a monotonically decreasing sequence of sizes. In either case, each size $x_i$ remains within its interval $[L_i, U_i]$. This handles the constraints for the upper and lower bounds.

We can exploit the monotonicity property even further, as proposed [21]. If we want to construct the trade off curve between area and delay, for example, we must compute the values $x_\lambda$ for a sequence of values $\lambda 0, \lambda 1, \lambda 2, ...$ If this sequence of $\lambda$'s is a monotonic sequence, then you can use the vector $x_{\lambda 0}$ as the start vector for approximating $x_{\lambda 1}$, and $x_{\lambda 1}$ as the start vector for approximating $x_{\lambda 2}$, and so on. Depending on whether the sequence of $\lambda$'s is an increasing or decreasing sequence, we must choose the lower bounds $L_i$ or the upper bounds $U_i$, respectively, as the initial start vector. This choice of start vectors of each $\lambda$ speeds up the computation of the trade-off curve even further.

In this paper we consider all gate sizes to be continous. To deal with discrete values for the sizes, we can round the gate sizes in $x_\lambda$ to discrete values. Because of the rounding, we may choose sub-optimal gate sizes. How much we loose in optimality depends on the resolution of the discrete gate sizes in the neighborhood of the optimal values and the gradients of the delay, area, and energy functions in the same neighborhood.

Alternatively we can compute lower and upper bounds for the discrete gate sizes that achieve the optimum. We compute the lower bounds for the optimum by always rounding downwards the update value for each $x_i$ to nearest discrete gate size in a monotonically increasing sequence. We compute the upper bounds for the optimum by always rounding upwards the update value for each $x_i$ in a monotonically decreasing sequence. Often the gap between lower and upper bound for each gate size is very small.

## IX. A Small Example

Figure 5 shows a string of three inverters with one side load. The side load is 10($\kappa$), the final load is 10($\kappa$) as well, and $x_0 = 1$.

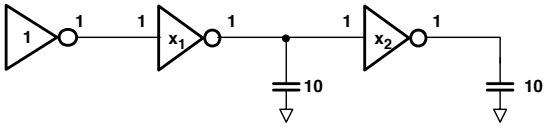


Fig. 5. A string of three inverters with one side load and one final load

The functions $D(x)$ and $A(x)$ are as follows:

$$D(x) = (1 + x_1/1) + (1 + 10/x_1 + x_2/x_1) + (1 + 10/x_2)$$

$$A(x) = 1 + x_1 + x_2$$

Let us compute the area-vs-delay function for this example. For each $\lambda$ we have to find $x_\lambda$ satisfying

$$\nabla A(x) = -\lambda \nabla D(x)$$

where $\nabla A(x) = [1, 1]$. Because all gates are inverters, we have $g_i = 1$ and $a_i = 1$ for $i = 1, 2$. The update functions for $x_1$ and $x_2$ are

$$x_1 = \sqrt{\frac{10 + x_2}{(1/\lambda) + 1}}$$

$$x_2 = \sqrt{\frac{10}{(1/\lambda) + 1/x_1}}$$

Let us first calculate the minimum path delay. For this purpose we choose $\lambda = \infty$. The update functions now simplify to

$$x_1 = \sqrt{10 + x_2}$$

$$x_2 = \sqrt{10 * x_1}$$

Table I gives the successive values for $(x_1, x_2)$ during the execution of the coordinate-wise descent algorithm. After only six steps, or three vector values, the algorithm yields values for $(x_1, x_2)$ that are less than 1% from the actual minimum. The path delay is minimal for $(x_1, x_2) = (4.05, 6.36)$ with a path delay of approximately 12.66$\tau$. The area at the minimum is approximately (1+4.05+6.36) = 11.4$\alpha$. The energy dissipation by the path at the minimum is approximately (1 + 2*4.05 +10 + 2*6.36 + 10) = 41.8$\epsilon$. Figure 6 graphically illustrates the coordinate-wise descent algorithm for $\lambda = \infty$ for this example.

| current value | next value | $x^{(0)}$ | $x^{(1)}$ | $x^{(2)}$ | $x^{(3)}$ |
|---|---|---|---|---|---|
| $x_2$ | $\sqrt{x_1 * 10}$ | 1 | 3.16 | 6.02 | 6.35 |
| $x_1$ | $\sqrt{1 * (10 + x_2)}$ | 1 | 3.63 | 4.00 | 4.04 |

TABLE I
SUCCESSIVE VALUES FOR $(x_1, x_2)$ OF THE COORDINATE-WISE DESCENT ALGORITHM WHEN $\lambda = \infty$.

Let us see how fast the algorithm converges to the final values of $x_1$ and $x_2$ when we choose $\lambda = 1$. Table II gives the simplified update functions and the successive values for $(x_1, x_2)$ during the execution of the coordinate-wise descent algorithm. The result after each step, viz., each update, appears in the next column. After only four steps, or two vector values, the algorithm yields values for $(x_1, x_2)$ that are less than 1% from the actual solution. For $\lambda = 1$, $x_\lambda = (2.52, 2.68)$ and

| current value | next value | $x^{(0)}$ | $x^{(1)}$ | $x^{(2)}$ | $x^{(3)}$ |
|---|---|---|---|---|---|
| $x_1$ | $\sqrt{(10 + x_2)/2}$ | 1 | 2.47 | 2.52 | 2.52 |
| $x_2$ | $\sqrt{10/(1 + 1/x_1)}$ | 1 | 2.24 | 2.67 | 2.68 |

TABLE II
SUCCESSIVE VALUES FOR $(x_1, x_2)$ OF THE COORDINATE-WISE DESCENT ALGORITHM WHEN $\lambda = 1$.

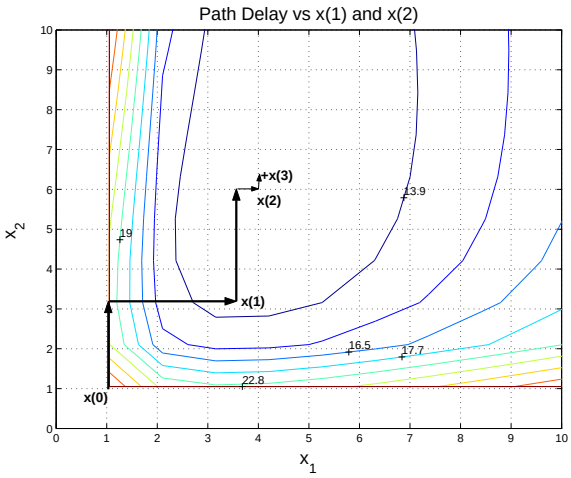


Fig. 6. A graphical illustration of the coordinate-wise descent algorithm with $\lambda = \infty$ and a contour plot of the path delay as a function of the gate sizes $x_1$ and $x_2$.

$D(x_\lambda) \approx 14\tau$, $A(x_\lambda) \approx 6.2\alpha$, and $E(x_\lambda) \approx 31.4\epsilon$. We can calculate that by paying about 13% in extra path delay, the area of the path goes down by more than 45%. The energy goes down by almost 25%.

The area-vs-delay function appears in Figure 7. The sequence of values for $\lambda$ is $\lambda_i = a^i * 0.1$ for $i = 0, 1, .., 15$ where $a^5 = 10$.

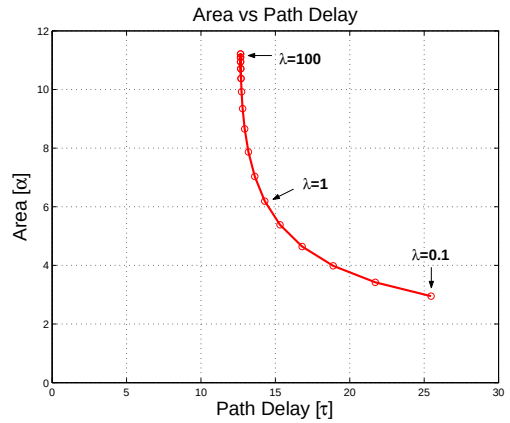


Fig. 7. The area-vs-delay function for the example circuit

## X. An Example Including Loops

In this example we consider a network with cycles. The network is a chain of C-elements connected in a ring as shown in Figure 8. Such chains of C-elements are often used in asynchronous circuits [23]. We are asked to find the energy-versus-delay function for one segment of this chain of C-elements.

If we need to consider every gate in this chain of C-elements, the task can become quite large. For our analysis purposes, however, it is sufficient to consider the segment of Figure 9, where terminals labeled A connect to each other and terminals labeled B connect to each other. We have labeled

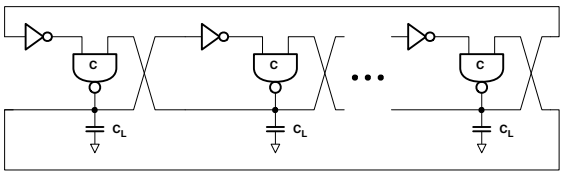


Fig. 8. A chain of C-elements connected in a ring

each input and output of a gate with the logical effort or parasitic delay of that terminal.

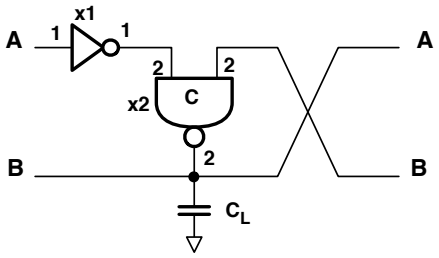


Fig. 9. A segment from a chain of C-elements connected in a ring

The delay function $D(x)$ for this circuit is the cycle time of the cycle containing one inverter and two C-elements. The delay function is

$$D(x) = (1 + 2 * x_2/x_1) + 2 * (2 + (C_L + x_1 + 2 * x_2)/x_2)$$

Rather than computing the energy function of a complete chain of C-elements, we calculate the energy function of a single segment only. The energy spent during one cycle in a single segment consisting of one inverter and one C-element is

$$E_{segment}(x) = 2 * x_1 + 6 * x_2 + C_L$$

The gradients for these functions are

$$\nabla D = \begin{bmatrix} 2/x_2 - 2 * x_2/x_1^2 \\ 2/x_1 - 2 * (C_L + x_1)/x_2^2 \end{bmatrix}$$

$$\nabla E = \begin{bmatrix} 2 \\ 6 \end{bmatrix}$$

The update functions obtained from $\nabla E = -\lambda \nabla D$ are

$$x_1 = \sqrt{\frac{2 * x_2}{2/\lambda + 2/x_2}}$$

$$x_2 = \sqrt{\frac{2 * (C_L + x_1)}{(6/\lambda + 2/x_2)}}$$

If we want to know the minimum cycle time of this circuit, $\nabla D = 0$ must hold. This condition implies that $x_1^2 = x_2^2$ and $x_2^2 = x_1 * (C_L + x_1)$. For fixed $C_L > 0$, this condition can only be satisfied for $x_1, x_2 \to \infty$ and $x_1/x_2 \to 1$. The minimum cycle time in that case is $1 + 2 + 2 * (2 + 1 + 2) = 13\tau$. This minimum cycle time is independent of the value of $C_L$.

The minimum cycle time of $13.0\tau$ is unattainable: the energy dissipation becomes prohibitive as the cycle time

approaches the minimum cycle time. Suppose we want to minimize the cycle time of the chain of C-elements and the energy dissipation of each segment. We choose to give equal weight to cycle time and energy dissipation. That is, we want to find the gate sizes $x$ such that the gradients in the energy function and cycle time are the same, but of opposite sign. In other words, any small change in gate sizes $x_1$ and $x_2$ that causes an increase or decrease in cycle time is counter balanced by the same decrease or increase, respectively, in energy. For our example we take $C_L = 10$.

Because we give equal weight to cycle time and energy, we choose $\lambda = 1$. We must solve $\nabla D = -\nabla E$. Solving this equation by coordinate-wise descent, we find the values in Table III. We find that for $(x_1, x_2) = (1.06, 1.76)$, the value

| current value | next value | $x^{(0)}$ | $x^{(1)}$ | $x^{(2)}$ | $x^{(3)}$ |
|---|---|---|---|---|---|
| $x_1$ | $\sqrt{(2*x_2)/(2+2/x_2)}$ | 1 | 1.01 | 1.06 | 1.06 |
| $x_2$ | $\sqrt{2*(10+x_1)/(6+2/x_2)}$ | 1 | 1.65 | 1.75 | 1.76 |

TABLE III
SUCCESSIVE VALUES FOR $(x_1, x_2)$ OF THE COORDINATE-WISE DESCENT ALGORITHM.

for energy $E$ is $24.3\epsilon$ and the value for cycle time $D$ is $24.88\tau$. The gate delay for the inverter is $4.32\tau$ and the delay for the C-element is $10.28\tau$.

The complete energy-vs-delay function appears in Figure 10. The sequence of values for $\lambda$ is $\lambda_i = a^i * 0.1$ for $i = 0, 1, .., 10$ where $a^5 = 10$.

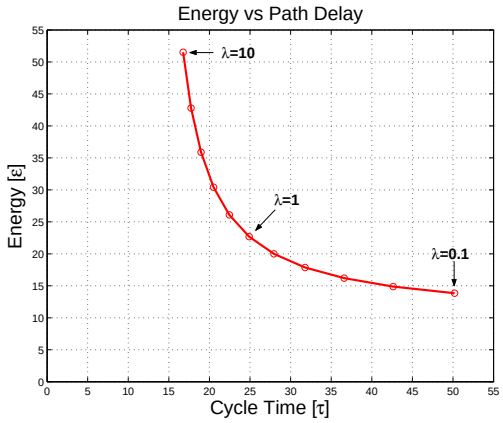


Fig. 10. The energy-vs-delay function for the example circuit

## XI. INCLUDING RESISTIVE WIRES

We can generalize our model by including resistive wires between gates. Figure 11 illustrates a path with side loads and resistive wires. We represent wires between gates by means of $\pi$ models, and we have merged any side loads to the wire load before and after the resistor. The variables $x_{i-1}, x_i, x_{i+1}$ represent gate sizes, $C_{i-1,0}, C_{i-1,1}, C_{i,0}, C_{i,1}$ represent fixed capacitances, and $R_{i-1}, R_i$ represent fixed resistances. We define $C_i$ as the total side load of stage $i$,

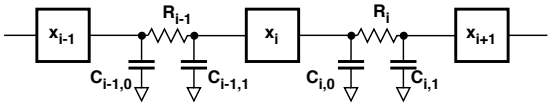


Fig. 11. A path with side loads and resistive wires between gates.

i.e., $C_i = C_{i,0} + C_{i,1}$. When the wire load $C_w$ is the only side load, then $C_{i,0} = C_{i,1} = C_w/2$.

Recall that the units for time and capacitance are $\tau$ and $\kappa$ respectively. Thus the units for resistance are $\tau/\kappa$.

Because of the addition of resistive wires, we must add the delays due to the resistance of the wire to the path delay. Using the Elmore delay model, we obtain (for stage $i$)

$$d_i = p_i + (g_{i+1} * x_{i+1} + C_i)/x_i + R_i * C_{i,1} + R_i * g_{i+1} * x_{i+1}$$

The term $R_i * C_{i,1} + R_i * g_{i+1} * x_{i+1}$ is the extra delay due to the resistance of the wire. The total path delay becomes

$$\begin{aligned} D(x) &= \sum_{i=0}^{N}((p_i + R_i * C_{i,1}) + \\ &\quad (C_i + g_{i+1} * x_{i+1})/x_i + R_i * g_{i+1} * x_{i+1}) \end{aligned}$$

where $g_{N+1} = 0$ and $x_{N+1} = 0$.

For the area function we consider the contributions of the fixed wires to be fixed. So the wires contribute only a fixed constant to $A(x)$. The modified delay function and area functions are still convex functions in $x$, and thus our optimization approach still applies.

For the gradient $\nabla D$ we obtain for $0 < i \leq N$

$$\begin{aligned} (\nabla D)_i &= \frac{\partial D}{\partial x_i} \\ &= (g_i/x_{i-1} - (C_i + g_{i+1} * x_{i+1})/x_i^2) + R_{i-1} * g_i \end{aligned}$$

Because the modified area function has changed only by a fixed constant, the gradient $\nabla A$ remains the same.

We need to solve $\nabla A = -\lambda \nabla D$, where the delay function is modified. For $0 < i \leq N$ this equation boils down to

$$a_i = -\lambda * ((g_i/x_{i-1} - (C_i + g_{i+1} * x_{i+1})/x_i^2) + R_{i-1} * g_i)$$

where $g_{N+1} = 0$ and $x_{N+1} = 0$. Solving for $x_i$ yields

$$x_i = \sqrt{\frac{C_i + g_{i+1} * x_{i+1}}{(a_i/\lambda) + g_i/x_{i-1} + R_{i-1} * g_i}}$$

This equation gives the update to coordinate $x_i$ in each iteration. This update function is very similar to the one without resistive wires: the only extra term is $R_{i-1} * g_i$.

## XII. SIMULATION RESULTS

To test this algorithm and determine the level of accuracy provided by logical-effort models, we created a testcase of a chain of five inverters in 65 nm technology. The first gate was fixed as 8x (being 8 times the gate size of a 1x minimum inverter). The remaining gates were sized using the logical effort algorithm described. See Figure 12.

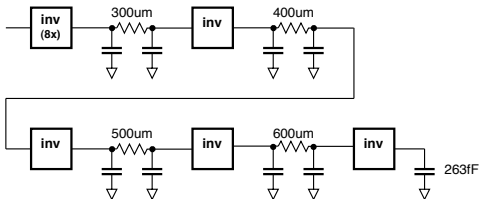


Fig. 12. Simulated testcase with long wires in 65nm technology.

We first solved for the set of gate sizes that would produce the absolute minimum delay (i.e. $\lambda = \infty$). These sizes were then put into the schematic and a SPICE simulation performed. We found the logical effort model was about 5% off from the SPICE results.

Then, we constrained the path delay to a value (larger than the previously found minimum, of course) and solved for the best set of sizes corresponding to the minimal area. These sizes were then SPICE'd and the result compared to the original constrained value. We again found that the logical-effort method gave a fair estimate of the delay, this time to about 9% of the SPICE value. Table IV summarizes these results. In general, the delays are within 10% of SPICE and can be used to provide a good starting point for additional optimizations.

| Test | LE Delay (ps) | SPICE (ps) | Difference | Area ($\alpha$) |
|---|---|---|---|---|
| Minimum Delay | 274 | 260 | 5.4% | 320.5 |
| Delay Constrained | 344 | 315 | 9.2% | 113.0 |

TABLE IV
SUMMARY OF SPICE SIMULATIONS.

We also applied our algorithms to a number of long circuit paths in an upcoming microprocessor design from SUN Microsystems Inc. For each path we calculated the delay $D_0$ and area $A_0$ for the path for the gate sizes chosen by the designer. We then calculated with our algorithm the delay and area for the sizes that achieve minimum path delay and for the sizes that achieve minimum area subject to the constraint that the path delay is at most the path delay of the designer's choice of sizes, i.e, $D(x) \leq D_0$. Table V summarizes our results for four paths. The results confirm that the minimum path delay

| | Min path delay | | Designer's choice | | Constrained Min | |
|---|---|---|---|---|---|---|
| | D [ps] | A [$\alpha$] | D [ps] | A [$\alpha$] | D [ps] | A [$\alpha$] |
| path 1 | 89 | 851 | 102 | 216 | 107 | 186 |
| path 2 | 177 | 670 | 259 | 170 | 262 | 126 |
| path 3 | 171 | 923 | 230 | 269 | 221 | 192 |
| path 4 | 97 | 1485 | 110 | 165 | 113 | 128 |

TABLE V
SUMMARY OF RESULTS FOR FOUR CIRCUIT PATHS

can come at a high cost in terms of gate area and that we may improve significantly in area while achieving approximately the same delay as for the designer's choice of gate sizes.

## XIII. CONCLUSION

We have described an efficient algorithm that can be used for area-delay and energy-delay optimizations. This algorithm takes advantage of the simplicity of the logical effort model to allow one to quickly compute the trade-offs between delay and area or energy. As a result, designers can experiment with sizing trade-offs without resorting to time-consuming SPICE simulations.

## ACKNOWLEDGMENT

The authors would like to thank Ilyas Elkin for providing support and examples for our algorithms.

## REFERENCES

[1] I. Sutherland *et al.*, *Logical Effort: Designing Fast CMOS Circuits*. San Francisco, CA: Morgan Kaufmann Publishers, Inc., 1999.
[2] N. Burgess, "Logical effort analysis of a media-enhanced adder," in *Conference Record of the Thirty-Seventh Asilomar Conference on Circuits and Systems, 2003*, vol. 1, Nov. 2003, pp. 344–348.
[3] D. Harris, "Logical effort of higher valency adders," in *Conference Record of the Thirty-Eight Asilomar Conference on Circuits and Systems, 2004*, vol. 2, Nov. 2004, pp. 1358–1362.
[4] P. Celinski, S. Al-Sarawi, D. Abbot, S. Cotofana, and S. Vassiliadis, "Logical effort based design exploration of 64-bit adders using a mixed dynamic-cmos/threshold-logic approach," in *Proc. IEEE Computer Society Annual symposium on VLSI, 2004*, Feb. 2004, pp. 127–132.
[5] I. Sutherland and J. Lexau, "Designing fast asynchronous circuits," in *Seventh International Symposium on Asynchronous Circuits and Systems, 2001*, Mar. 2001, pp. 184–193.
[6] S. Laberge and R. Negulescu, "An asynchronous fifo with fights: case sstudy in speed optimization," in *The 7th IEEE International Conference on Electronics, Circuits, and Systems, 2000*, Dec. 2000, pp. 755–758.
[7] B. Hu, Y. Watanabe, and M. Marek-Sadowska, "Gain-based technology mapping for discrete-size cell libraries," in *Proceedings Design Automation Conference, 2003*, June 2003, pp. 574–579.
[8] S. Karandikar and S. Sapatnekar, "Logical effort based technology mapping," in *IEEE/ACM International Conference on Computer Aided Design*, Nov. 2004, pp. 419–422.
[9] P. Rezvani, A. Ajami, M. Pedram, and H. Savoj, "Leopard: a logical effort-based fanout optimizer for area and delay," in *IEEE/ACM International Conference on Computer-Aided Design*, Nov. 1999, pp. 516–519.
[10] J.-M. Shyu, A. Sangiovanni-Vincentelli, J. Fishburn, and A. Dunlop, "Optimization-based transistor sizing," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 2, pp. 400–409, Apr. 1988.
[11] H. Tennakoon and C. Sechen, "Gate sizing using lagrangian relaxation combined with a fast gradient-based pre-processing step," in *Proc. IEEE/ACM International Conf. Computer-Aided Design (ICCAD)*, Nov. 2002, pp. 395–402.
[12] J. Fishburn and A. Dunlop, "TILOS: A posynomial approach to transistor sizing," in *Proc. IEEE/ACM International Conf. Computer-Aided Design (ICCAD)*, Nov. 1985, pp. 326–328.
[13] S. Sapatnekar, V. Rao, P. Vaidya, and S.-M. Kang, "An exact solution to the transistor sizing problem for cmos circuits using convex optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 11, pp. 1621–1634, Nov. 1993.
[14] A. Wächter, C. Visweswariah, and A. R. Conn, "Large-scale nonlinear optimization in circuit tuning." *Future Generation Comp. Syst.*, vol. 21, no. 8, pp. 1251–1262, 2005.
[15] C.-P. Chen, C. Chu, and D. Wong, "Fast and exact simultaneous gate and wire sizing by lagrangian relaxation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 7, pp. 1014–1025, July 1999.
[16] C.-P. Chen, H. Zhou, and D. Wong, "Optimal non-uniform wire-sizing under the elmore delay model," in *Proc. IEEE/ACM International Conf. Computer-Aided Design (ICCAD)*, Nov. 1996, pp. 38–43.
[17] C. Chu and D. Wong, "Greedy wire-sizing is linear time," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 4, pp. 398–405, Apr. 1999.
[18] H. J. Veendrick, "Short-circuit energy dissipation and its impact on the design of buffer circuits," *IEEE Journal of Solid-State Circuits*, vol. SC-17, no. 1, pp. 468–473, Feb. 1984.

[19] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
[20] J. Cong and L. He, "Theory and algorithm of local-refinement-based optimization with application to device and interconnect sizing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 4, pp. 406–420, Apr. 1999.
[21] J. Cong and K.-S. Leung, "Optimal wire sizing under the distributed elmore delay model," in *Proc. IEEE/ACM International Conf. Computer-Aided Design (ICCAD)*, Nov. 1993, pp. 634–639.
[22] J. Cong and C.-K. Koh, "Simultaneous driver and wire sizing for performance and power optimization," *IEEE Transactions on VLSI Systems*, vol. 2, no. 4, pp. 408–425, Dec. 1994.
[23] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, June 1989.

**George Chen** received the B.S. degree in electrical engineering from the California Institute of Technology , Pasadena, CA, in 1987, the M.S. and PhD. degrees in electrical engineering from Stanford University in 1988 and 1993, respectively. Since 2000, he has been a member of Sun's Processor CAD group managing and developing analysis tools and flows for processor designs.

**Jo Ebergen** Jo Ebergen received his PhD in Mathematics and Computing Science from Eindhoven University of Technology in 1987. From 1988 to 1996 he taught at the Computer Science Department at Waterloo University in Canada. Since 1996 he works for Sun Microsystems Laboratories, where he is a distinguished engineer working on experimental architectures, clockless circuits, formal verification, and circuit optimizations. Jo is a member of IEEE and a distinguished engineer of ACM.

**Jiyang Cheng** Jiyang Cheng received the B.S. degree in Electrical Engineering from University of Arizona in 1999 and the M.S. degree in Electrical Engineering from Stanford University.

From 1999 to 2001, He was a telecommunication analyst in NASA's Jet Propulsion Laboratory. He was involved in several Mars missions including Mars Reconnaissance Orbiter, Mars Exploration Rovers mission.

From 2003, He was a member of technical staff at Sun Microsystems. His current research interests include signal integrity based static timing analysis and timing based noise analysis. His experience includes SRAM design, ASIC design, noise and static timing tool design.