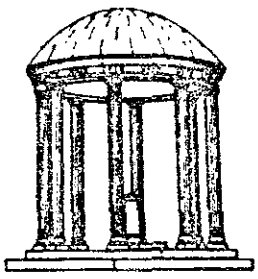
Algorithms for the Electrical Optimization of Digital MOS Circuits

Kye S. Hedlund, Assistant Professor



The University of North Carolina at Chapel Hill
Department of Computer Science
New West Hall 035 A
Chapel Hill, N.C. 27514

This work supported in part by grants from International Business Machines, Inc. and the Microelectronics Center of North Carolina. (Submitted for publication).

Abstract

This work addresses the problem of automating the electrical optimization of digital MOS circuits. Improvements to a circuit's speed, area, and power consumption are sought through modifications to the transistor sizes in the circuit; no changes in the circuit structure, number of gates or clocking are introduced. Linear algorithms are presented for computing optimal transistor sizes to minimize delay, area or power. These algorithms have been incorporated into a prototype electrical optimization tool, EO, that assists the designer in the performance tuning of MOS VLSI circuits. The designer provides the circuit layout and the requirements on its speed, power consumption, and area. EO identifies the slowest paths through the circuit and computes the optimal transistor sizes to achieve the performance objectives. This frees the designer to deal more directly with the speed, power, and area characteristics of the circuit rather than the details of its implementation. Due to the fast interactive response of the tool, the designer can experiment with a large number of different transistor sizing options to determine the best balance of delay vs. area, delay vs. power, and power vs. area. The transistor sizing algorithms compute *optimal* points so that the designer is guaranteed of making the most efficient possible tradeoffs of scarce resources. When compared to manual designs, the circuits produced by EO are typically faster or have substantially lower power consumption and area.

1. Introduction

In addition to being functionally correct, a chip must meet constraints on its speed, power, and area. The packaging technology places an upper bound on the power dissipation per package. Manufacturing technology limits the size of a chip that can be reproduced with acceptable yield, and system performance requirements dictate the required speed of the individual chips.

Each of these three basic quantities can be traded off against each other: speed may be increased by consuming more power, in nMOS power consumption is decreased by lengthening pullup transistors thus increasing circuit area, etc. Much of the art of the design process is in balancing the tradeoffs: How to achieve highest speed using the least power and area?

This paper presents algorithms for automating the electrical optimization of digital MOS circuits. Improvements to a circuit's speed, area, and power consumption are sought

through modifications to the transistor sizes in the circuit; no changes in the circuit structure, number of gates, or clocking are introduced. The algorithms presented here work for both nMOS and CMOS circuits. The algorithms are fast thus allowing interactive computation of optimal transistor sizes.

The two algorithms discussed in this paper compute optimum transistor sizes that:

1) Minimize the delay through the circuit subject to bounds on the sizes of the individual transistors (thus limiting any increase in circuit area)

2) Minimize power consumption (nMOS) or area (CMOS) subject to bounds on both the maximum delay through the circuit and on the transistor dimensions

A simple example will illustrate the nature of the electrical optimization problem. Consider a chain of three CMOS inverters (Figure 1). For simplicity, assume that all transistors have the same width and length. Let the width of both the n and p channel transistors in gate 2 be w_2, and let D be the total delay through the three gates. Consider the effect of increasing w_2 while keeping the size of the transistors in gates 1 and 3 fixed.

Increasing w_2 causes the magnitude of the output current of gate 2 to increase. Thus the time required, d_2, for gate 2 to drive its output signal will decrease (Figure 1b). However, increasing w_2 also increases the capacitive load on the output of gate 1 thus slowing the output transition of the first gate. At some point, $w_2 = A$, these two effects will be balanced. This is the point of minimum delay (with respect to w_2). The delay minimization algorithm must find the optimum point when varying the transistor sizes of within all gates in the circuit. This optimum point must be found within a search space whose dimensions equals the number of gates.

When minimizing area or power, the designer specifies a target delay, $D_{\mathrm{MAX}} \geq D_{\mathrm{MIN}}$. A vector of transistor sizes is computed so that the sum of the transistor sizes in the circuit is minimized (Figure 1b, point B) subject to the constraint that the total delay does not exceed D_{MAX}. The algorithm must minimize a linear function (area or power) subject to a nonlinear constraint (total delay) in a multi-dimensional search space.

A prototype tool, EO, for optimizing MOS circuits has been implemented. This tools computes optimum transistor sizes to meet the performance requirements specified by the designer. The tool assists the designer in optimally distributing power to circuit elements and in spending circuit area to gain speed or reduce power consumption. Since the tool can be run interactively even for large circuits, the designer can rapidly evaluate many design alternatives choosing the best tradeoff of speed, area, and power.

Advantages of this work over some previous approaches ([Agul77], [Glas84], [Lee84], [Mats85a], [Mats85b], [Rueh77], [Trim83a], [Trim83b]) include: fast interactive response instead of lengthy batch-mode runs, guaranteed optimal results rather than heuristic approximations, optimization over sets of paths rather than just a single path, optimization for both polarities (0,1) of all input signals, and the capability to put limits on the maximum and minimum transistor sizes.

The primary disadvantage of this approach is that SPICE level accuracy of the delay estimation is not achieved. A premium has been placed on computational speed so that interactive response is achieved. This dictates simplified device models and thus lower accuracy in predicting delay. The delay estimates are typically accurate to within ±20% of SPICE. It is anticipated that this accuracy will suffice for many circuits. Variations in processing parameters can cause the circuit performance to vary by amounts greater than 20%. Work is underway to incorporate an improved delay model accurate to ±6-8%. Where greater accuracy in delay prediction is required, EO can be used for initial sizing followed by fine tuning via SPICE simulations.

The next section describes a prototype electrical optimization tool. Section 3 briefly presents the transistor and circuit models used to predict the electrical behavior of a circuit. The main research results are in sections 4 and 5: the optimization algorithms and an analysis of the algorithm performance on circuits extracted from real chips. The conclusions and directions of future research are in the final section.

2. EO – An Automated Electrical Optimization Tool

With current CAD tools, performance tuning of a circuit requires a long, iterative procedure. An electrical model of the circuit is extracted from its mask representation. This model is simulated on input waveforms chosen by the designer to (hopefully) expose the worst case behavior of the circuit. This simulation output is manually analyzed. If there are performance problems, the designer makes an educated guess at correcting the problem and repeats the extraction and simulation process.

The process of designing a high performance circuit consists of iterating a simulate—analyze—adjust loop until adequate performance is obtained. There are several disadvantages to this approach.

1) It is labor intensive. Considerable designer time is required to analyze the simulation results and make minor circuit adjustments.

2) Only relatively small cells may be simulated. Electrical simulators such as SPICE can accurately predict delays, but they are compute-intensive. Fast turnaround or interactive runs are limited to small circuits.

These limitations on maximum circuit size make it impractical to simulate large portions of a design. Only single cells or small collections of cells can be simulated. Unfortunately, electrical nodes often cross cell boundaries making it difficult to predict from simulation results the behavior of a cell in its actual on-chip environment.

Critical path analysis addresses these problems. A path analysis tool [Joup83, Oust83] finds the slowest paths in a circuit. The tool is fast enough (due to simple transistor models) to analyze an entire chip thus the accuracy of the results do not depend on the designer's judgment of which paths to simulate or the choice of input waveforms. However, critical path analysis still does not address the most fundamental problem in the current approach to performance optimization.

3) Critical path analysis tools can point out problems but they do not indicate how to solve them. The designer is still caught in the loop of repeatedly simulating, analyzing, and adjusting the circuit.

When performance problems are found with critical path analysis, a variety of techniques can be used to improve circuit performance: inserting buffers, changing clocking, precharging highly capacitive nodes, etc. Once the circuit design has been finalized, the designer must match the drive capability of transistors to their output loads; large transistors are used to drive highly capacitive nodes and smaller transistors are assigned to low capacitive outputs. Because transistor sizing is so tedious, designers tend to use "rules of thumb" to select between a small number of sizing options.

An automated electrical optimization tool, EO, has been implemented that computes optimum transistor sizes. This relieves the designer of the time consuming process of making an educated guess, simulating and refining the guess. Instead, the designer explores optimal points in the design space searching for the right combination of circuit performance. When compared to manual designs, the optimal circuits computed by EO often are somewhat faster and have substantially lower power consumption and area.

A prototype version of an automated electrical optimizer was implemented in the spring of 1984. This first version, PLAOPT, was restricted to the optimization of PLAs [Hedl84, Hedl85]. A subsequent version, EO, was expanded to handle general nMOS circuits not containing pass transistors or super buffers. (These restrictions come from limitations in the user interface that generates the optimization model of the circuit. The optimization

algorithms are fully general and capable of handling all circuit constructs and both nMOS and CMOS technology.)

EO has been used primarily as a vehicle for algorithm development. It has a rudimentary textual user interface. Critical path analysis is performed by the Crystal2 program [Oust83] by John Ousterhout of the University of California at Berkeley. The C language code for Crystal2 is used (with minor modifications) as a module in EO.

Figure 2 shows a stylized version of an interactive session with EO.[2] The designer runs EO on the circuit level representation of the nMOS circuit MainCntrl.[3] The input to EO has been extracted from the mask representation and contains information about parasitic capacitances and resistances. The first table displays the critical path through the circuit along with the incremental delays at the six nodes of the critical path. For each gate on the path, the pullup length and pulldown width are displayed. The designer has selected between two sizing options (Wpd = 4.0 or 8.0) for each of the gates. This is representative of manual transistor sizing.

The designer then specifies (in lambda units [Mead80]) the maximum pulldown width and minimum pullup length. The minimum path delay subject to these constraints is computed. The middle table displays the incremental delays and associated transistor sizes to achieve minimal delay. Note that the transistors in tow of the gates have been increased to their maximum allowed size. Under the given transistor size constraints, path delay can be reduced by 60.0% at a cost of more than tripling the static power consumption along the path.

In this example, the designer judges, based on system requirements, that a path delay of 45ns is sufficient. The "Minimum Power" command is used to size transistors for a path delay longer than the minimum possible delay. The designer enters the requested delay, 45.0 (ns), and EO computes the transistor sizes to achieve this delay with minimum power consumption. Note that all transistors in this 45ns circuit are larger than in the original circuit but that none are at the maximum allowed size.

[2] Some command formats are slightly different than indicated in the figure, and critical path analysis is not gracefully integrated into EO. Performing the operation "Critical Path" actually requires a separate computer run.

[3] This is a subcircuit of a head tracker chip designed and fabricated by Gary Bishop at the University of North Carolina at Chapel Hill.

Several observations about this interactive session should be made:

1) The tool has fast interactive response. An optimization command typically requires no more than 10 to 20 second of CPU time (VAX 11/750). This allows the designer to rapidly explore optimum tradeoffs of speed, area and power.

2) The optimum transistors sizes for both "Minimum Delay" and "Minimum Power" vary over a considerable range. This has been observed for a large number of circuits, and this supports the claim that manually designed circuits are often far from optimum (because designers typically select from a small number of sizing options).

3) The designer guides the optimizer by selecting the set of paths over which optimization is performed. For simplicity, this example shows the selection of a single path. However, the algorithms can perform optimization over multiple, intersecting paths. In actual usage, the designer would make more complex selections of paths. For example "optimize over all paths whose delay exceeds 50.0ns."

4) Designers incrementally improve their circuits. This mode of operation is built into EO. A designer wanting to speedup a 120ns circuit to 90ns would first speed the circuit to (say) 100ns. Due to interactions between the many paths through the circuit, the set of critical paths may be changed by this delay reduction. A second Critical Path command would find the key paths in this 100ns circuit. A second optimization command would reduce the delay from 100ns to 90ns over the new set of critical paths.

The current implementation of EO's user interface has several limitations:

- Crystal is not integrated into the optimizer
- pass transistors and super buffers are not modeled by the user interface
- transistor characteristics are coded into the program making it technology specific.

It should be emphasized that the optimization algorithms are fully general and do not suffer from these limitations.

Work is underway at the University of North Carolina and the Microelectronics Center of North Carolina to produce a production quality electrical optimizer. The production tool will remove these limitations, provide a graphical interface to the optimizer, and be integrated into the VIVID symbolic design system.

3. Electrical Models

This section describes the models used to predict the electrical behavior of transistors, gates, and paths. The goal is to rapidly and accurately predict the delay and power consumption of circuits. The fundamental equations predicting the behavior of transistors are well known (at least for long channel devices). Simulators such as SPICE accurately predict the behavior of a wide variety of MOS devices and circuit constructs. However, these simulators achieve their high accuracy at a large cost in computational effort. Electrical optimizers based on these detailed device models are exceedingly slow [Nye81].

The goal of the models used in this research is to accurately *and quickly* estimate the circuit delay. Some accuracy is sacrificed for very large decreases in computational cost. The RC model presented here is several orders of magnitude faster than SPICE simulations. Less than 1 msec of computer time (VAX[4] 11/750) is required to estimate the delay of a gate. The average error for estimating total path delay [Oust84] is about 20% for the RC model.[5]

The path is the highest level data structure in the electrical optimizer. A path is a chain of gates plus the polarity of the path input signal. This single path input is assumed to start a sequence of output transitions in the gates along the path.

In each gate, the change in the output is assumed to be triggered by a single input. The transistor affected by this input is called the trigger transistor. All other gate inputs are assumed to be fixed. Within the gate, all electrical pathways from the gate output to Vdd or ground that are in parallel with the trigger transistor are assumed to be open circuited. All transistors in series with the trigger transistor must be turned-on (or else the trigger signal would not change the gate output).

The transistor sizing algorithms associate with each gate a scale factor, S_i, that determines the transistor sizes within the gate. Transistor sizes scale up or down in proportion to the scale factor. In CMOS, the widths of both the n and p channel transistors are directly proportional to S_i (lengths are fixed). In nMOS, the length of the pullup is inversely proportion to S_i while the pulldown width is directly proportional to S_i (pullup width and pulldown length are fixed). Thus changing the scale factor will leave the impedance ratio

[4] VAX is a registered footnote of the Digital Equipment Corporation.

[5] [Oust84] reports 24% for the RC model. The electrical optimizer models transmission gates more accurately thus reducing the error in delay estimation. We estimate the average error to be reduced to 20%.

unchanged. In both nMOS and CMOS, the current supplied or sunk by a gate varies in direct proportion to S_i.

The electrical characteristics of a circuit are summarized by the equations predicting the circuit's delay and either its area (CMOS) or its power consumption (nMOS). In nMOS technology, static power consumption is proportional to sum of the pullup lengths in the circuit

$$P(\vec{S}) = \sum_{i=1}^{N} K \frac{W_{PU_i}}{L_{PU_i}} S_i \qquad (3.1)$$

where K is a constant determined by the fabrication technology.

Area minimization is inherently complex. The relationship between transistor sizes and overall cell area can be a complex function of the layout geometry. Some dense portions of the layout may be highly sensitive to changes in transistor dimensions, whereas other portions of the design may have sufficient unused area to accommodate variations in transistor sizes without affecting cell area.

For practical use, we need a relationship between the transistor sizes and gate area that is independent of the specific circuit layout. Assume that gate area can be approximated by gate height times gate width. The height is proportional to the number of transistors in the gate. Gate width is determined by transistor width, and this is proportional to S_i. This gives the following approximation to the transistor size - gate area relationship

$$A(\vec{S}) = \sum_{i=1}^{N} T_i S_i \qquad (3.2)$$

where T_i is the number of transistors in gate i.

The total delay through a path is simply the sum of the individual gate delays

$$D(\vec{S}) = \sum_{i=1}^{N} d_i \qquad (3.3)$$

To estimate gate delay, a simple RC model is used. In the RC model, the resistance and capacitance is computed for each node and transistor in the path. Figure 3 shows the electrical model of an nMOS circuit path. Extension to CMOS is straight-forward. The resistances and capacitances are separately summed, and their product is the estimate of the gate delay

$$d_i = (\sum R)(\sum C) = (\frac{R_i}{S_i} + R_{w_i})(S_{i+1} C_{G_{i+1}} + C_{W_i}) \qquad (3.4)$$

The capacitance load on the output is the sum of the parasitic wire capacitance between gate i and $i+1, C_{w_i}$, and the input capacitance of gate $i+1$. Since the width of the pulldown transistors is proportional to the gate scale factor, the input capacitance of gate $i+1$ is $S_{i+1}C_{G_{i+1}}$ where $C_{G_{i+1}}$ is the gate capacitance of the original circuit before optimization.

The effective resistance of each transistor is individually computed and added to the parasitic wire resistance, R_{w_i}. For nMOS technology, the gate resistance, R_i/S_i, depends on the sum of the transistor resistances

$$R_i = R_{PU_i} \quad \text{if gate output } = 1$$

$$R_i = R_{PD_i} + R_{ON_i} \quad \text{if gate output } = 0$$

where R_{PU_i} and R_{PD_i} are the effective resistance of a single pullup and pulldown transistor in gate i. R_{ON_i} is the total resistance of the turned-on transistors in series with the trigger transistor (Figure 3). Since the aspect ratio (length/width) of all transistors in the gate decreases in direct proportion to S_i, the effective resistance of the gate is inversely proportional to S_i.

The extension to CMOS is straight-forward. R_i is the sum of the resistances between the output node and either Vdd or ground depending on the output polarity.

Each transistor is modeled as an input capacitor and a switched resistor (Figure 4). The effective resistance of a single transistor is determined by its type (n or p channel, depletion mode, etc.), geometry and the data value it transmits. The dependence on the data value (logical '0' or '1' on the gate output) insures more accurate modeling of transmission gates. A table of effective resistances gives the resistance in ohms per square. This table is indexed by the transistor type and data value, and the table entry is multiplied by the transistor's length/width to give its effective resistance.

The table values of effective resistance can be supplied by the user to define the characteristics of a particular technology or fabrication line. The model is general enough to handle both CMOS and nMOS transistors. The table is generated by running SPICE simulations on a single gate. A step function drives the input, and the effective resistance is the delay divided by the load capacitance.

This table driven approach is a type of macromodeling. The electrical optimizer can treat gates as atomic entities rather than collections of transistors. The electrical characterization of the gates is done once in the computation of the table. The input processor of the electrical optimizer coalesces transistors into gates. Thus the delay computations can

proceed at the gate level rather than the transistor level as in SPICE. Using this higher level of abstraction results in significant speed savings.

There are three main sources of error in the RC model. First, self loading of the output capacitance is not taken into account. Self loading results from the built-in diffusion capacitance of the transistors and is proportional to the transistor width. It could be incorporated into the model by adding the term $S_i C_{I_i}$ to the output capacitance in equation 3.4 where C_{I_i} is the internal capacitance of the trigger transistor of gate i.

Secondly, the lumping of resistances and capacitances tends to overestimate delays since it assumes that all the capacitance must be discharged through all the resistance. This could be improved upon using the Penfield-Rubinstein models for distributed capacitance [Penf81, Rubi83]. However, the electrical optimizer treats transmission gates as separate gates rather than as a part of a complex stage [Oust84, Toku83]. This eliminates the largest source of error when lumping resistances and capacitances.

The most significant source of error is the lack of characterization of waveform shape [Oust84]. The RC model is most accurate when the waveforms rise and fall quickly. In practice, the effective resistance of a transistor depends on the shape of the input signal. A slow rise on the gate input means that the gate turns on slowly. The trigger transistor may do much of its work while only partially turned-on. Thus R_i will be overly optimistic.

4. Algorithms for Transistor Sizing

Algorithms for two operations will be discussed:

a) Minimize the delay through a set of paths

b) Minimize the power consumption or area of a set of paths subject to a bound on the maximum path delay.

The electrical characteristics of a path are modeled by two equations representing the path delay and either its area or its power consumption (equations 3.3, 3.2 and 3.1). The gate scale factors, S_i, are the independent variables. A vector, $\vec{S}$, is sought to minimize either the delay, power, or area. This problem is formulated as a nonlinear optimization problem that is solved subject to constraints.

Note that this expression and the equations for power consumption and area (equations 3.1 and 3.2) both are linear combinations of the S_i. They differ only in the constants. Consequently, the same algorithm will be used for both power and area minimization. We will restrict our discussion to power minimization.

A. Practical Requirements

Any practical algorithm must allow the designer to exercise substantial control over the algorithm and the form of the solution it computes. These constraints may reflect layout geometry restrictions and constraints external to the circuit being optimized. The delay and power minimization algorithms reported here handle the following practical requirements:

- Limits on transistor sizes
- Asymmetric rise and fall times of gates
- Data independent optimization
- Multiple paths

Limits must be placed on the maximum and minimum transistor sizes. This avoids unrealistically small devices and allows the designer to control any increase in circuit area by bounding the maximum size of any transistor. As noted earlier, the sizing in different gates may have different effects on circuit area. This makes it desirable to allow the designer to control the sizing bounds of each gate individually. Some gates may be allowed a wider sizing range than others depending on their relative area and delay sensitivities. The designer is free to choose larger ranges for gates that are especially speed sensitive (i.e. have large output loads) or smaller ranges for the gates that are area sensitive.

Gate delays in nMOS technology are inherently asymmetric; rising transitions take significantly longer than falling transitions. Similarly in CMOS, differences in carrier mobility make n channel transistors typically four times faster than p channel transistors. This can be compensated for by making every p channel device four times the width of its n channel counterpart. However, this is not always practical. Reducing layout area often forces n and p transistors to be the same size. Additionally, most of the transistors in a design are not on the critical path hence it is not necessary to spend layout area to provide oversize p channel transistors. Consequently, the delay model must take into account asymmetries in the rise and fall times of gates for both nMOS and CMOS technology.

Given that a gate may have unequal rising and falling delays, the delay of a path will be dependent on the polarity of the input to the first gate in the path. Recall that we assume that a single input triggers a series of signal changes in the path. Each path must be optimized over both a logical '0' path input and a logical '1' input. Optimizing the circuit for only one of the inputs produces a "picket fence" solution (Figure 5). Transistors are made large for gates with the slower output transition, and small transistors are allocated

to gates with the faster transition. The transistor sizes chosen for a single input polarity result in very slow delay for the input signal of opposite polarity.

In a practical design environment, designers are typically concerned not with the performance of an individual path but of a whole circuit. There are many different paths through even a simple circuit. In some cases the critical path may be obvious (for example, the carry chain in a ripple adder). Often designers have difficulty accurately predicting the slow paths [Oust83]. This makes it essential for the optimization algorithms to handle multiple, perhaps interacting, paths.

These four practical requirements greatly complicate the optimization algorithms. To simplify the presentation of the algorithms, sections B and C present versions of the algorithms capable of handling only transistor size limitations and asymmetric rise/fall times of gates. The extension to all four practical requirements is in section D.

B. Delay Minimization for a Single Path

This section considers the problem of delay minimization restricted to a single path and a single polarity of the path input signal. This is the simplest case, and, as is explained above, it is not sufficient for a practical CAD algorithm. However, the more complex cases of optimization over multiple paths with data independent optimization will be developed from this simplified analysis and algorithm. The algorithm is this section does allow bounds to be placed on the maximum and minimum transistor sizes and takes into account asymmetric rise/fall times of gates.

1. Restricted Model and Problem Formulation

Consider the delay through a single path containing N gates. Arbitrarily select path input '0'. Since the rise and fall times of the gate may be different, the effective resistance of the gate may depend on the polarity of the gate output signal. Let $R_i(0)$ be the transistor resistance of gate i with gate output '0', and $R_i(1)$ corresponds to gate output '1'. The delay through the i^{th} gate in the path is

$$d_i = (R_i(odd(i))/S_i + R_{W_i})(S_{i+1}C_{G_{i+1}} + C_{W_i})$$

where "odd(i)" accounts for the gate output inverting through each successive gate. The total path delay (with '0' path input) is denoted by D_0 and is the sum of the individual gate delays.

$$D_0(\vec{S}) = \sum_{i=1}^{N} d_i$$

The delay minimization problem is formulated as a nonlinear optimization problem with constraints.

$$\text{minimize } D_0(\vec{S}) = \sum_{i=1}^{N} d_i$$
$$\text{subject to } l_i \leq S_i \leq u_i \qquad i = 1, \ldots, N$$

The l_i and u_i are simply derived from the limits on transistor sizes specified by the user. For example, if gate k has a pulldown width of 4λ and the designer specifies the maximum and minimum widths to be 12λ and 2λ respectively, then $u_k = 3.0$ and $l_k = 0.5$.

2. Method of Solution

The nonlinear optimization problem is solved by a standard quasi-Newton method. The algorithm is sketched below (Figure 6) but see [Gill81] for details. The algorithm uses an (approximate) second order model of the delay function, $D_0(\vec{S})$, and makes successive steps reducing the delay until no further delay reduction is possible.

An approximation, B, to the Hessian of D_0 (the matrix of second partial derivatives of D_0) is maintained and updated at every iteration. B is represented by its Cholesky factors, $B = LDL^T$, where L is a unit lower triangular matrix and D is a diagonal matrix with strictly positive diagonal elements. Each update of B requires only a low rank modification of B and hence is computationally efficient.

During the search, some of the S_i may be on their bounds (either $S_i = u_i$ or $S_i = l_i$. The S_i that are not on their bounds are the free variables, N_z. Let $\vec{g}_z$ denote the vector of derivatives of $D_0(\vec{S})$ with respect to the free variables. A search direction, q_z, in the subspace of free variables is determined by solving the set of linear equations

$$B\vec{q}_z = -\vec{g}_z$$

Then α is found such that $D_0(\vec{S} + \alpha\vec{q}_z)$ is approximately minimum. $\vec{S}$ is replaced by $\vec{S} + \alpha\vec{q}_z$ and B is updated. If any S_i reaches its bound during the search along $\vec{q}_z$ it is removed from N_z for the next iteration. Step length control is provided so that α does not force any S_i to violate their bounds.

Stepping from $\vec{S}$ to $\vec{S} + \alpha\vec{q}_z$ may move some S_i off their bounds so that they become free variables. To detect this, Lagrangian multipliers are estimated for all active bounds. If any estimate is sufficiently negative then one of the associated variables is released from its bound and added to N_z. The search at the next iteration is performed over the extended subspace. Otherwise, minimization continues in the current subspace.

3. Algorithm Convergence

The delay optimization algorithm has been run on a number of paths extracted from four different chip designs. In all instances extremely rapid convergence has been observed (Table 1). Typically, 4–6 iterations suffice to find the minimum delay to an accuracy of ± 0.1 on each of the S_i.[7] Monitoring the progress of the iteration shows that the algorithm typically locates a point within about 10% of the minimum in 1–2 iterations. The delay function is flat in the region about the minimum delay so the algorithm spends over half its time exploring the region close to the optimum. This indicates that an algorithm for finding the approximate minimum delay (within 10%) would be even more rapid than the exact minimum delay algorithm reported here.

The algorithm exhibits two desirable features. First, the execution time is insensitive to the starting point of the algorithm. Starting points farther from the optimum simply result in larger step sizes during the first few iterations and don't force additional iterations. The reason for this is that the algorithm uses an approximate second order model of the delay function and this model is an accurate predictor of the function behavior (see below). Second, the running time of the algorithm is approximately linearly proportional to the requested accuracy. The algorithm locates the region near the optimum in a fixed number of iterations and then explores the flat region about the optimum in time proportional to the requested accuracy. This allows the designer to trade runtime for accuracy in a simple and predictable fashion.

The rapid convergence and robustness of the algorithm results from the well-behaved nature of the delay function. The delay function is continuous, smooth, and monotonic. Furthermore,

- $D_0(\vec{S})$ is everywhere convex (see Appendix). This means that it has a single global minimum. There are no local minima for the algorithm to become stuck in.
- $D_0(\vec{S})$ is a quadratic function. This is closely modeled by the (approximate) second order model of the quasi-Newton algorithm. The Taylor series expansion of $D_0(\vec{S})$ is

$$D_0(S_k + \gamma) \approx D_0(S_k) + g_k^T \gamma + \frac{1}{2}\gamma^T B \gamma$$

[7] Many optimum points reported in this paper have been verified by exhaustive search in a bounding box centered about the optimum point reported by the delay minimization algorithm. The convexity of the delay and power functions (see below) guarantees that this will find the global minimum.

where $g_k = \nabla D_0(S_k)$ and B is the approximation to the Hessian of D_0. With D_0 being quadratic, higher terms in this expansion are zero. Thus the approximation is exact to the extent that B approximates the true Hessian. The experimental evidence shows this accuracy to be good.[6]

- In practical circuits, the S_i are of similar scale (i.e. the relative size of the smallest and largest transistors does not vary by several orders of magnitude). This means that the search for the minimum is not prone to hemstitching. Each step in the iteration reduces the delay by a significant percentage of the distance to the optimum point.

C. Area and Power Minimization for a Single Path

This section considers the problem of power minimization restricted to a single path and a single polarity of the path input signal. As noted previously, area minimization is handled with exactly the same algorithm as power minimization. Limitations on the sizes of transistors will be taken into consideration in this section. Extensions to the other practical requirements are made in the following section.

1. Problem Formulation and Method of Solution

Consider minimizing the power consumption of a path while insuring that the path delay does not exceed a fixed bound, D_{MAX}, and that user specified bounds on transistor sizes are satisfied. As with delay minimization, arbitrarily choose path input '0'. Define the power minimization problem to be

$$\text{minimize } P(\vec{S}) = \sum_{i=1}^{N} K \frac{W_{pu_i}}{L_{pu_i}} S_i$$

$$\text{subject to } l_i \leq S_i \leq u_i \quad i = 1, \ldots, N$$

$$\text{and } C(\vec{S}) = D_0(\vec{S}) - D_{MAX} = 0$$

The linear objective function is minimized subject to one nonlinear constraint and N bounds constraints. As with delay minimization, l_i and u_i, are simply determined from the user specified bounds on transistor size.

[6] A full second derivative method using direct computation of the Hessian was also implemented. Its execution times were greater than those for the quasi-Newton approach. This suggests that for the objective function $D_0(\vec{S})$, B is a good approximation to the true Hessian.

The delay constraint is written as an equality rather than an inequality since delay can be traded for power; the point of minimum power consumption will always occur at the maximum allowable delay and never less.

Nonlinear constraints are difficult to handle directly because it is generally impossible to follow the surface of the nonlinear function. Given any point $\vec{x}$ such that $C(\vec{x}) = 0$, in general there is no feasible direction $\vec{q}$ such that $C(\vec{x} + \alpha\vec{q}) = 0$ for all sufficiently small $|\alpha|$.

To handle the nonlinear delay constraint, note that [Gill81, Mats85a] at the point, $\vec{S}^*$, that minimizes power subject to the constraints, the delay and power contours are tangent. Hence their gradients point in the same direction.

$$\nabla P(\vec{S}^*) = \lambda^* \nabla C(\vec{S}^*)$$

where λ^* is the Lagrangian multiplier. Consequently, $\vec{S}^*$ is a stationary point of the Lagrangian function $P(\vec{S}) - \lambda^* C(\vec{S})$

$$\nabla(P(\vec{S}^*) - \lambda^* C(\vec{S}^*)) = 0 \qquad (4.1)$$

Since the Lagrangian function is convex (see Appendix), $\vec{S}^*$ is also a minimum of the Lagrangian function. However, it is important to note that the value of the Lagrangian multiplier, λ^*, is not known in advance.

This suggests the general form of the algorithm for power minimization (see Figure 7): estimate λ^*, find the minimum of a Lagrangian-type function and repeat. At each step, the solution to the minimization subproblem is used to make an improved estimate for λ^* and a point closer to the desired minimum, $\vec{S}^*$. In effect, a subproblem is formulated whose solution approximates the solution of the power minimization problem. This subproblem is solved and a refined subproblem is formulated. This is repeated until the subproblem has a solution sufficiently close to the solution of the power minimization problem.

The primary advantage of this approach is that it transforms a problem with both nonlinear and bounds constraints into a sequence of subproblems involving only bounds constraints. Each of these subproblems is solved by quasi-Newton methods identical to those used for delay minimization. Both theory and experience indicate rapid convergence of the subproblems. Hence, the key to this approach is the formulation of the subproblems.

One disadvantage of using the Lagrangian function above for the minimization subproblem is that the solution to the subproblem is very sensitive to the estimate of the Lagrangian multiplier [Gill81], and no prior knowledge of the Lagrangian multiplier is

known. This means that the number of subproblems required may be large and considerable computation time may be spent solving subproblems whose solutions are not close to the minimum of the power minimization problem. The initial subproblem solutions serve primarily to refine the successive subproblems.

This problem can be avoided by modifying the form of the subproblem. By adding to the Lagrangian function a quadratic penalty term, we form the augmented Lagrangian function

$$L(\vec{S}, \lambda, \rho) = P(\vec{S}) - \lambda C(\vec{S}) + \frac{1}{2}\rho C(\vec{S})^T C(\vec{S})$$

where ρ is a positive scalar.

This objective function has the following advantages over use of the Lagrangian function without augmentation

- Reduced sensitivity to the estimate of λ^*
- The penalty term strongly steers the iterations towards feasible points
- Relatively low sensitivity to ρ

The addition of the penalty term makes the augmented Lagrangian function relatively insensitive to errors in the estimate of the Lagrangian multiplier. Since no advance knowledge of the Lagrangian multiplier is known, the algorithm converges to the optimum point more rapidly. From a body of experimental evidence optimizing paths extracted from real chips, no more than two iterations of the loop in Figure 7 has been required for optimizing a single path. For optimization over multiple sets of paths, two iterations usually suffice but no more than three iterations have been required. Experimentation with various initial estimates of λ^* also substantiate these sensitivity findings.

The quadratic growth of the penalty term forces iterations to remain close to the delay bound. If the starting point of the algorithm is not feasible, the penalty term forces rapid convergence to a feasible point.

A potential disadvantage of this approach is that it is possible for the subproblems to be ill conditioned if ρ is large. In practice it has been observed that the update procedure for ρ (Figure 8) does not lead to ill conditioning. In fact, the progress of the minimization algorithm is relatively insensitive to changes in ρ.

2. Implementation

A more detailed version of the power minimization algorithm is in Figure 8. Its speed depends on the matching of both the algorithm and objective function to the characteristics of the power minimization problem. It should also be emphasized that the speed of the algorithm is critically dependent on the implementation of the algorithm. Implementation tuning produced three orders of magnitude improvement in the algorithm speed. This involved: optimal use of derivative information, selecting subproblem and line search accuracy, setting termination criteria, etc. A few of the more important implementation features are discussed here.

The initial values for λ and ρ where empirically chosen by experimenting over a small set of paths and a modest range of values. The algorithm is not sensitive to their initial values so little effect on the execution time of the algorithm was observed.

The execution time of the algorithm is sensitive to the accuracy of the initial guess for S_i.[9] Although this is to be expected, experimentation with a variety of heuristics for choosing the starting point yielded the following surprising result: the number of iterations required by the algorithm is

- sensitive to the distance of the starting point from the delay bound
- relatively insensitive to the distance on the delay bound from the optimum point.

Choosing a near feasible starting point is important; the power consumption at the starting point is relatively unimportant. The reason for this is unknown, but it is conjectured that the bounds constraints often cause a small step size from one iteration to the next. This forces the algorithm to execute many iterations to locate the delay bound.

The following heuristic is used to generate an initial guess for $\vec{S}$. Recall that the delay minimization algorithm is very fast and takes large initial steps towards the optimum. The iteration sequence of the delay minimization algorithm is used to find two points, $\vec{S}^{(k)}$ and $\vec{S}^{(k+1)}$, whose delay falls above and below the delay bound. Binary search is then used on the line segment between $S^{(k)}$ and $S^{(k+1)}$ to locate a starting point whose delay is close to D_{MAX}.

[9] Note that the algorithm will converge for virtually any starting point (see Appendix). The speed is sensitive to $\vec{S}^{(1)}$, but the convergence of the algorithm has not been found to be dependent on the starting point.

There is no reason for this starting point to have low power consumption. However, once the algorithm is on the delay bound the value of the augmented Lagrangian function is dominated by the power term, $P(\vec{S})$. Thus, the following steps of the iterations solely minimize power without having to first satisfy the delay constraint. Furthermore, the penalty term of the augmented Langrangian function insures that the search direction does not stray far from the delay bound.

This heuristic for the starting point was found to perform substantially better than various fixed choices for S_i (such as $S_i = 1.0\ \forall i$). It is also superior to any of the heuristics tested that find low power solutions not necessarily close to the delay bound.

At each iteration it is necessary to update the estimate of the Lagrangian multiplier, λ^*. At the optimum point, S^*, we know (equation 4.1) $\nabla P(\vec{S}^*) = \lambda^* \nabla C(\vec{S}^*)$. This suggests estimating λ^* by finding a λ that comes as close as possible to making the equality hold at the current iteration point; in other words, find $\lambda^{(k+1)}$ that is the least squares solution to $\nabla P(\vec{S}^{(k)}) = \lambda \nabla C(\vec{S}^{(k)})$. This is a first order estimate based on the current point in the iteration.

Other Lagrangian multiplier estimates were investigated. Experiments were performed with selected values of λ close to λ^* determined by previous runs on the problem instance. This lead to little or no decrease in total execution time. Additionally, it has been noted that very few loop executions in Figure 7 are required. These two results lead to the conclusion that the increased computational cost of more accurate second order estimates of the Lagrangian multiplier is not justified.

Information about the effect of the previous step on the augmented Lagrangian function is used to update the subproblem and determine convergence. At each iteration, the norm of the constraint function is calculated. At the optimum it should be zero. This is scaled by the size of the elements of the Jacobian matrix. This value, newnorm, is a measure of the infeasibility of the delay constraint at the current point. 'oldnorm' is the value at the previous iteration point.

The relative values of the norms (newnorm and oldnorm) are used to adjust the subproblem and its required accuracy. The penalty constant, ρ, is adjusted up or down depending on whether the new point is more or less feasible than the previous point. Smaller values of ρ reduce the condition number of the subproblem. Hence it is desirable to reduce ρ as low as possible while still insuring that the subproblem has a bounded minimum.

As noted earlier, the execution time of the subproblem is roughly directly proportional to the required accuracy. For a fast implementation, the required accuracy of the

subproblem should be proportional to the degree to which the subproblem approximates the original power minimization problem. Decreasing values of the norm dictate more accurate solutions to the subproblems. This is reflected by adjusting "inner_stol" for each subproblem.

D. Multiple Paths and Delay Independence

This section extends the modeling of the delay, area, and power minimization problems to permit optimization over multiple paths through the circuit. The optimization will also be independent of the data values propagating through the circuit; delay, power, or area will be optimized over both polarities of the input signal to each path.

These cases conclude the four practical requirements that the optimization algorithms must handle. The problem models in this section correspond to those implemented in the automated electrical optimization tool, EO.

First, consider data independence. For a single path, $D_0(\vec{S})$ has been defined to be the path delay with a logical '0' value as the input to the path; $D_1(\vec{S})$ corresponds to a '1' path input. The goal is to optimize the circuit behavior independent of the path input. For the delay minimization problem, this means minimizing the objective function

$$\max\ (D_0(\vec{S}), D_1(\vec{S}))$$

This function has a discontinuous derivative arising from shifts in the maximum delay from one input signal polarity to the other. Continuous first derivatives are essential for the speed and convergence of the quasi-Newton algorithm.[10]

The maximum function is approximated by the smooth function, *smax* [Rueh77].

$$smax(x_1, \ldots, x_n) = \frac{1}{\alpha} ln(e^{\alpha x_1} + \ldots + e^{\alpha x_n})$$

where α is the smoothing constant that determines the sharpness of the transition from one delay to another. The optimization problem becomes

$$\text{minimize } smax(D_0(\vec{S}), D_1(\vec{S}))$$

[10] A version of the quasi-Newton algorithm not using direct computation of the derivative was implemented and evaluated. It required approximately 50 times as many function evaluations as compared to using derivative information and thus was prohibitively slow.

This will optimize the delay through a single path independent of the data values in the circuit.

Next, consider multiple paths. The designer's goal is to optimize *circuit* behavior. This requires considering multiple paths from circuit inputs or memory nodes to the circuit outputs. The critical path capability of EO identifies all critical and near critical paths through the circuit. Over this path set, the goal is to minimize or bound the delay through any path in a path set and with any combination of input values to the paths.

Let Q be a set of paths through a circuit. For any path $q \in Q$, let D_{0q} be the delay through path q with logical '0' path input and D_{1q} be the delay with '1' input. The delay minimization problem for multiple paths with data independence becomes

$$\text{minimize } D_Q(\vec{S}) = \underset{q \in Q}{smax}(D_{0q}(\vec{S}), D_{1q}(\vec{S}))$$

$$\text{subject to } l_i \leq S_i \leq u_i \quad i = 1, \ldots, N$$

where N is the total number of gates in all the paths. This minimizes the maximum delay over all paths and all input data values.

For power and area minimization, the objective functions $P(\vec{S})$ and $A(\vec{S})$ are unchanged, but the summation must be performed over all gates in all paths. The delay constraint must, of course, be the maximum delay over all paths. For example, the power minimization problem becomes

$$\text{minimize } P(\vec{S}) = \sum_{i \in G} K \frac{W_{PU_i}}{L_{PU_i}} S_i$$

$$\text{subject to } l_i \leq S_i \leq u_i \quad i \in G$$

$$\text{and } C(\vec{S}) = D_Q(\vec{S}) - D_{\text{MAX}} = 0$$

where G is the set of all gates contained in the path set Q.

The problem formulation for area minimization is similar.

5. Summary of Results

EO has been used to extract, analyze and optimize paths from a number of nMOS chip designs. The algorithms work for CMOS circuits, but they have not been run on any CMOS designs. Tables 1 and 2 show the results of performing delay and power minimization on two sample circuits. The circuit 'Control' is the control logic section from a larger chip, and 'Decode' is an instruction decoder circuit.

For each of the circuits, performance is optimized for a varying number of paths. All paths were selected by critical path analysis. The single path (Number of Paths = 1) is the slowest path in the circuit. Each larger set of paths includes the next smaller set. Thus the path sets for each circuit include successively larger groups of critical and near critical paths. In all three tables, S_i was restricted to the range $0.5 \leq S_i \leq 4.0$ for all i. The tables also show the number of *different* gates appearing in the set of paths.

The results of minimizing delay are summarized in Table 1. This shows the delay in the original circuit before optimization and the delay at the optimum point. The number of search points is one greater than the number of iterations (Figure 7) of the delay algorithm and is thus one measure of computational cost. The CPU time is another more direct measure.

Several conclusions can be drawn from this data:

- Substantial delay reductions were achieved in all circuits analyzed. Total delay could often be cut by a factor of three as compared to the original, unoptimized circuit.
- Low computational effort was required. CPU times were short allowing the interactive response of EO to be very fast. Typical delays are at most a few seconds.

The results of minimizing power are shown in Table 2. The same path sets are used as in the first table. Power was minimized while holding the delay through the slowest path constant (± 1.0%). EO reallocated power between gates in a given path and between paths (slowing down paths that do not limit circuit performance). This causes optimal utilization of a scarce resource (power) while not affecting circuit speed. Power reductions of 10 to 30% are common.

The response time for power minimization was fast. No more than 25 seconds of CPU time were required for fairly large path sets.

In sum, both the delay and power minimization algorithms rapidly compute optimal transistor sizes. This allows EO to provide very fast response time. This interactive capability of EO is a substantial advance over batch-mode tools for optimization. With our limited experience, EO has proved to be a highly effective tool.

The dependence of compute time on problem size is a key characterization of the algorithms. The complex nature of the optimization routines precludes meaningful theoretical results for the power and delay minimization problems. Instead, an empirical evaluation has been conducted.

A single, very long path was selected. It was subdivided into its first 2 gates, first 5 gates, etc. Optimization was performed on each of these increasingly long subsections of the path. The compute requirements as a function of path length are shown in Table 3. For power minimization, the number of subproblems is the number of loop executions in Figure 7; each loop execution requires the minimization of a single subproblem. The number of search points visited to solve each of the subproblems is reported. Their sum is a measure of the compute time required for the problem.

The growth in CPU time is approximately a linear function of the number of gates in the path (Figure 9). The varying contours of the objective functions causes some instances to be more costly and some less; this is typical for many search problems. The pattern is for a linear increase of CPU time with steeper slope for power minimization. This is corroborated by the growth rate of the total number of the number of search points.

Linear growth has been observed for many paths, and it is believed to be typical for these algorithms. However, in general, nonlinear optimization has greater than linear compute time growth. The superior performance of delay or power minimization due to several factors:

- The objective and constraint functions are quadratic and well-behaved. The problem is inherently well conditioned, and the (approximate) second order quasi-Newton methods accurately predict the behavior of the objective and constraint functions.
- The algorithms are well matched to the problems. The use of an augmented Lagrangian function results in very few subproblems (2) and hence is an excellent model of the power minimization problem.

6. Conclusions and Topics for Further Research

Simplified electrical models of transistors and gates are used to estimate the performance of digital MOS circuits. Using these models, linear time algorithms compute the optimum transistors sizes to minimize delay, area or power. Based on these models and algorithms, an interactive electrical optimization tool was implemented. During the past two year this tool has been used to analyze a variety of industrial and university nMOS circuits.

The major conclusions of this research are:

1) Automatic computation of optimal transistor sizes produces circuits that are often substantially better than those produced by manual design. Optimal circuits are typically faster and have substantially improved area and power consumption. This of

course depends on the amount of manual effort the designer has invested. The tedium of manual transistor sizing typically forces the designer to choose from a small set of sizing alternatives. Optimal circuits exhibit a much wider range of sizes and thus are often prohibitively difficult to discover manually.

2) Optimal transistor sizes can be rapidly computed. The experimental evidence shows the algorithms to be approximately linear. Compute times are typically 2-20 seconds even for fairly large sets of paths containing 30 to 40 gates. The speed of the algorithms allows interactive computation of transistor sizes.

3) An interactive tool to compute optimal transistor sizes increases designer productivity and improves circuit quality. The initial experience with EO has been very favorable. The response time of the tool is very fast, and the basic user interface fairly simple.

A surprising result is that nonlinear optimization techniques are sufficiently fast to handle even large circuits. The high speed and fast response of EO results from three factors:

- The optimization problem has been carefully formulated to be well behaved. Also, the algorithms and their implementations have been carefully tailored to the specific problems at hand. As a result, little searching of the state space is required.
- Simplified transistor models allow the problem to be formulated as a relatively simple set of tridiagonal, quadratic equations. These can be solved with high computational efficiency. Simulators such as SPICE require two to three orders of magnitude more computation time making the optimization problem prohibitively costly.
- The electrical models of the paths to be optimized is represented by a simple and compact data structure thus allowing rapid access. The full circuit is represented by a complex graph structure composed of interconnected paths, gates, transistors and nodes. From this, a compact optimization model is extracted. It consists of a simple set of arrays containing resistances, capacitances, etc. These arrays can be rapidly accessed with a minimum of indirect addressing. This substantially reduces the overhead of the optimization algorithms. Direct access of the graph structure would be approximately 50% slower. In addition, the optimization modeler coalesces individual transistors and nodes into gates. Thus the optimizer works at a higher level of abstraction.
- EO is well matched to the activities performed by designers. The goal is to form a man-machine combination that is more effective then either alone. To accomplish this, the basic mode of operation built into EO is that of making incremental circuit

improvements based on guidance from the designer; the designer specifies path sets and performance requirements. Incremental improvement is a fundamental engineering method. User guidance in part prevents the CAD tool from becoming overwhelmed with the complexity of VLSI circuits. Typically, only 1-10%of the gates in a large circuit need be optimized at any one time.

There are numerous topics for future research. The primary topic is investigating more accurate models for delay prediction. A slope model [Oust84] that incorporates the effect of waveform slope has been added to the most recent version of EO. It computes path delays to within 6-8% of SPICE.

Additional work is underway on analytical analysis of the problem: What simplifications must be made in order to derive closed for solutions to the optimization problems?

The development of a production quality electrical optimizer was begun in June 1985. Planned are extensions to CMOS technology, a graphical interface and integration into a symbolic layout system.

Acknowledgements

Many people have contributed to this work. Conversations with Gershon Kedem have been most enlightening. Richard Craddock, Rick Gross, William Hargrove, and John Poulton read a draft of this paper and made numerous helpful comments. Leigh Pittman did a superb job preparing the figures and the manuscript. Last but not least, John Ousterhout's work on Crystal has contributed greatly to this research. The critical path analysis and trigger model have been borrowed from Crystal.

References

Agul77 Agule, B.J., Lesser, J.D., Ruehli, A.E. and Wolff, P.K. "An Experimental System for Power/Timing Optimization of LSI Chips," *Proceedings of the Fourteenth Design Automation Conference* (1977).

Gill81 Gill, P.E., Murray, W. and Wright, M.H. *Practical Optimization*, Academic Press (1981).

Glas84 Glasser, L.A. and Hoyte, L.P.J. "Delay and Power Optimization in VLSI Circuits," *Proc. 21st Design Automation Conf.* (1984), 529-535.

Hedl85 Hedlund, K.S. "Electrical Optimization of PLAs," *Proc. 22nd. Design Automation Conf.* (June 1985).

Hedl84 Hedlund, K.S. "Models and Algorithms for Transistor Sizing in MOS Circuits," *IEEE International Conf. on Computer-Aided Design* (Oct. 1984).

Joup83 Jouppi, N.P. "Timing Analysis for nMOS VLSI," *Proc. 20th Design Automation Conf.* (1983), 411-418.

Lee84 Lee, C.M. and Soukup, H. "An Algorithm for CMOS Timing and Area Optimization," *J. of Solid-State Circuits,* SC-19, 5 (Oct. 1984), 781-787.

Mats85a Matson, M.D. "Optimization of Digital MOS VLSI Circuits," *1985 Chapel Hill Conf. on VLSI* (May 1985).

Mats85b Matson, M.D. *Macromodeling and Optimization of Digital MOS VLSI Circuits* (Feb. 1985), Ph.D. Thesis, MIT.

Nye81 Nye, W., et al. "DELIGHT: An Optimization-Based Computer-Aided Design System," *Proceedings IEEE International Symposium on Circuits and Systems*, (April 1981), 851–855.

Okaz84 Okazaki, K., Moriya, T. and Yahara, T. "A Multiple Media Delay Simulator for MOS LSI Circuits" *Proc. 20th Design Automation Conf.* (1984), 279-285.

Oust83 Ousterhout, J.K. "Crystal: A Timing Analyzer for nMOS VLSI Circuits," *Third Caltech Conference on Very Large Scale Integration*, (Jan. 1983), 57-70.

Oust84 Ousterhout, J.K. "Switch-Level Delay Models for Digital MOS VLSI," *Proc. 21st Design Automation Conf.* (June 1984), 542-548.

Penf81 Penfield, P. Jr. and Rubinstein, J. "Signal Delay in RC Tree Networks," *Proc. 18th Design Automation Conf.* (1981), 613-617.

Rubi83 Rubinstein, J., Penfield, P. Jr. and Horowitz, M.A. "Signal Delay in RC Tree Networks" *IEEE Trans. on CAD/ICAS* **CAD-2** 3 (July 1983), 202-211.

Rueh77 Ruehli, A.E., Wolff, P.K. and Goertzel, G. "Analytical Power/Timing Optimization Techniques for Digital Systems," *Proceedings of the Fourteenth Design Automation Conference* (1977).

Tamu83 Tamura, E., Ogawa, K. and Nakano, T. "Path Delay Analysis for Hierarchical Building Block Layout System," *Proc. 20th Design Automation Conf.* (1983), 411-418.

Toku83 Tokuda, T. et.al. "Delay-Time Modeling for ED MOS Logic LSI," *IEEE Trans. on Computer-Aided Design,* **CAD-2**, 3 (July 1983), 129-134.

Trim83a Trimberger, S.M. *Automated Performances Optimization of Custom Integrated Circuits*, (Jan. 1983), Ph.D. Thesis, Cal Tech.

Trim83b Trimberger, S.M. "Automated Performance Optimization of Custom Integrated Circuits," *VLSI 83*, (Aug. 1983), 99–108.

Appendix - Theoretical Results

This section establishes the convexity of the objective functions used in power and delay minimization. Convexity guarantees that a single global minimum exists.

Theorem 1 – $D_0(\vec{S})$ is convex.

Proof—To establish this, we must show that $G(\vec{S})$, the Hessian of $D_0(\vec{S})$, is positive definite. In other words,

$$x^T G(\vec{S}) x > 0 \qquad \text{for any nonzero vector } x$$

By definition, the S_i form an orthogonal basis for the search space. Let y_i be the unit vector in direction S_i, $y_i = < 0, \ldots, 1, \ldots, 0 >$ with the one in the i^{th} position. We need show only

$$y_i^T G(\vec{S}) y_i > 0 \qquad i = 1, \ldots, N$$

But

$$y_i^T G(\vec{S}) y_i = \frac{\partial^2 D}{\partial S_i^2} = \frac{2}{3} \frac{R_i(odd(i))}{S_i^3} (S_{i+1} C_{G_{i+1}} + C_{W_i})$$

which is strictly positive. Consequently, $G(\vec{S})$ is everywhere positive definite. □

The convexity of $D_0(\vec{S})$ establishes that the delay minimization problem for a single path without bounds on the S_i has a single global minimum. D_1 differs from D_0 only by substituting even(i) for odd(i). Thus $D_1(\vec{S})$ is also convex.

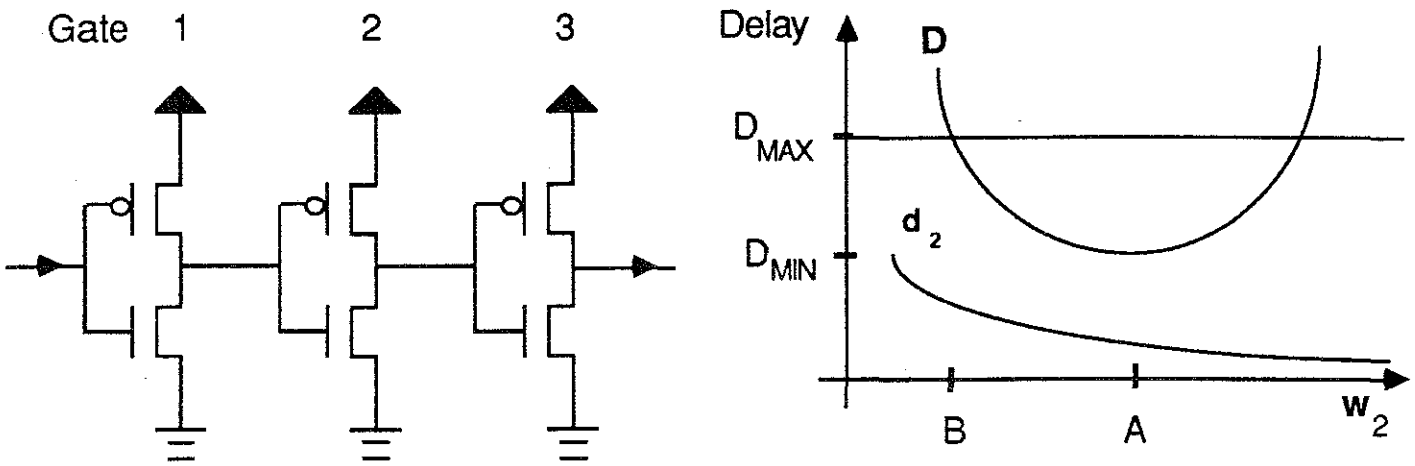
Next we show convexity of the objective function for multiple paths.

Theorem 2— If f_j are convex functions then smax f_j is convex.

Proof — As in Theorem 1, we must show

$$y_i^T (\text{smax } f_j) y_i = y_i^T (\frac{1}{\alpha} ln \sum_j e^{\alpha f_j}) y_i > 0 \quad i = 1, \ldots, N$$

smax f_j is convex iff $\sum_j e^{\alpha f_j}$ is convex. Since $e^{\alpha f_j}$ is monotonic, it must be convex if f_j is convex. □



a) Three CMOS Inverters

b) Delay Behavior

Figure 1 - Example of Effect of Transistor Sizing on Delay (D = total delay through all 3 gates; d_2 = output delay of gate 2)

```
%eo MainCntrl.sim

*** MainCntrl:  111 nodes  97 depletion  408 enhancement

eo> Critical Path
     Transition   Delay        Lpu    Wpd
     InSt2* -> 0   1.4 (ns)
       jbar -> 1  13.5 (ns)    2.0    8.0
       jbuf -> 0   4.6 (ns)    2.0    8.0
        r16 -> 1  15.1 (ns)    4.0    4.0
      zObar -> 0   2.2 (ns)    4.0    4.0
         MO -> 1  31.0 (ns)    2.0    8.0
path   pMO delay  67.8 (ns)

eo> Set max pdwidth 32
eo> Set min pulength 0.5
eo> Minimum Delay pMO

                        Before Opt      After Opt   After/Before
    Maximum Delay        67.8 (ns)      27.0 (ns)      0.399
Power Consumption        13.1 (mW)      45.0 (mW)      3.43

     Transition   Delay        Lpu    Wpd
     InSt2* -> 0   2.6 (ns)
       jbar -> 1   5.7 (ns)    0.50   32.0
       jbuf -> 0   4.0 (ns)    0.80   20.0
        r16 -> 1   4.9 (ns)    1.00   16.0
      zObar -> 0   1.4 (ns)    1.67    9.6
         MO -> 1   8.4 (ns)    0.50   32.0
path   pMO delay  27.0 (ns)

eo> Minimum Power pMO
*** enter (flt pt) Maximum Allowable Delay (ns) = 45.0

                        Before Opt      After Opt   After/Before
    Maximum Delay        67.8 (ns)      45.0 (ns)      0.661
Power Consumption        13.1 (mW)      17.3 (mW)      1.317

     Transition   Delay        Lpu    Wpd
     InSt2* -> 0   1.5 (ns)
       jbar -> 1  12.4 (ns)    1.82   8.8
       jbuf -> 0   4.8 (ns)    2.50   6.4
        r16 -> 1   8.8 (ns)    1.11   7.2
      zObar -> 0   1.9 (ns)    3.10   5.2
         MO -> 1  15.6 (ns)    1.00  16.0
path   pMO delay  45.0 (ns)

eo> Quit
[ 39.7 VAX 11/750 CPU seconds ]
```

Figure 2 - Interactive Session with Electrical Optimizer

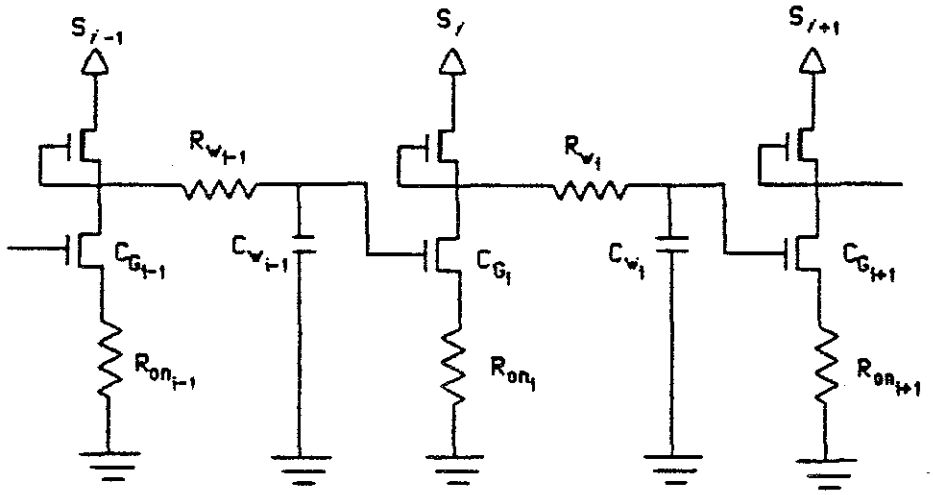


Figure 3 - Electrical Model of a Path (nMOS)

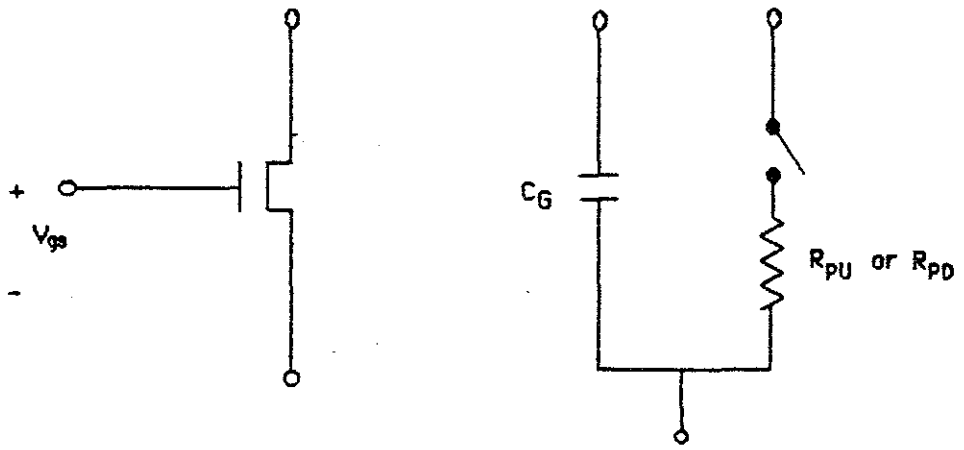
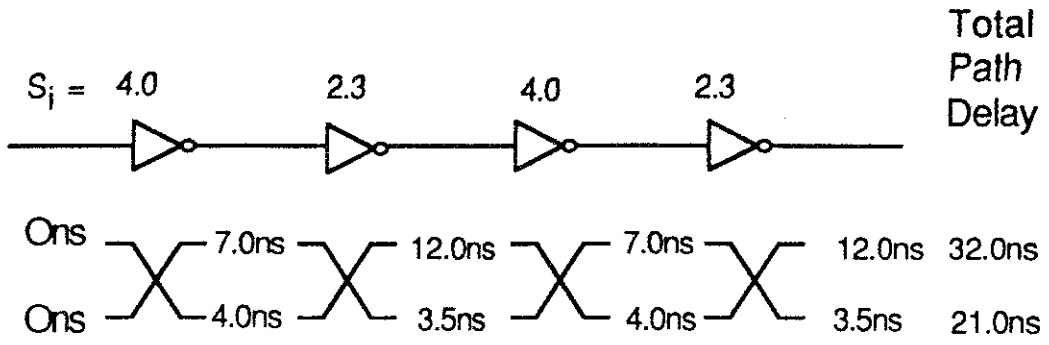
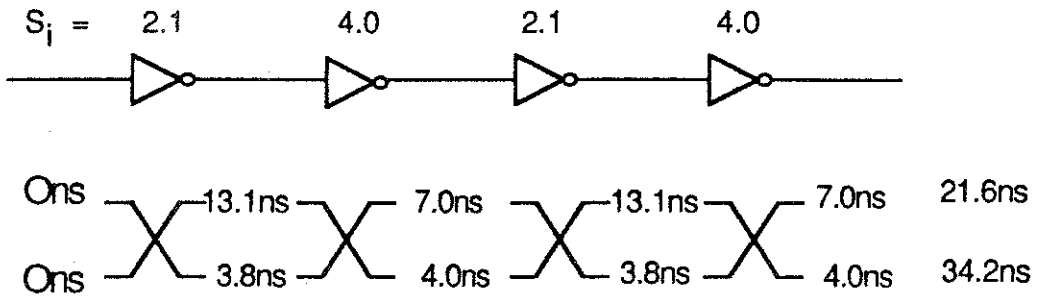


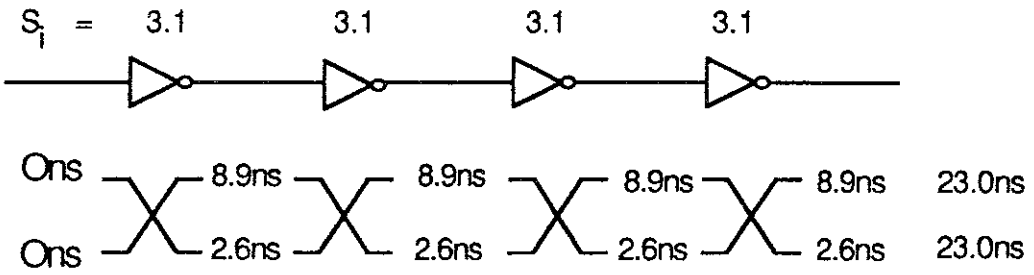
Figure 4 - Electrical Model of a Transistor



a) Circuit optimized for '0' path input



b) Circuit optimized for '1' path input



c) Circuit optimized for both polarities of path input

Figure 5 - Dependence of Delay Optimization on Input Signal Polarity (gate scale factors appear above gates; nMOS inverters with impedance ratio = 4.0)

Compute l_i and u_i based on transistor size limitations

$S_i = 1.0 \qquad i = 1, \ldots, N$

B_1 = identity matrix

$k = 1$

repeat

compute search direction $\vec{p}_k$ by solving $B_k \vec{p}_k = \nabla D_0(\vec{S}_k)$

find step length, α_k, that approximately minimizes $D_0(\vec{S}_k + \alpha_k \vec{p}_k)$

make step length α in direction $\vec{p}_k$, $\vec{S}_{k+1} = \vec{S}_k + \alpha_k \vec{p}_k$

update B_k

$k = k + 1$

until convergence

Figure 6—Outline of Quasi Newton Algorithm for Delay Minimization Along a Single Path.

Compute l_i and u_i based on transistor size limitations

Initialize λ and ρ

Make an initial guess of S_i

repeat

minimize $L(\vec{S}, \lambda, \rho) = P(\vec{S}) - \lambda C(\vec{S}) + \frac{1}{2}\rho C^T(\vec{S})C(\vec{S})$

subject to $l_i \leq S_i \leq u_i \qquad i = 1, \ldots, N$

update λ

update ρ

until convergence

Figure 7—Outline of Algorithm for Power or Area Miminization

Compute l_i and u_i based on transistor size limitations

Initialize scalars $\lambda^{(1)} = 1.0$ and $\rho^{(1)} = 10.0$

Make an initial guess of vector $\vec{S}^{(1)}$

stol = user requested acuracy of S_i

inner_stol = required accuracy of S_i for subproblem = stol

newnorm = 0.01

$k = 0$

repeat

$k = k + 1$

required accuracy of subproblem = inner_stol =

max($\frac{\text{stol}}{\rho}$, min(oldnorm, inner_stol))

minimize $L(\vec{S}^{(k)}, \lambda^{(k)}, \rho^{(k)}) = P(\vec{S}^{(k)}) - \lambda^{(k)} C(\vec{S}^{(k)}) + \frac{1}{2}\rho^{(k)} C(\vec{S}^{(k)^T}) C(\vec{S}^{(k)})$

subject to $l_i \leq S_i \leq u_i \quad i = 1, \ldots, N$

by the quasi-Newton algorithm

choose $\lambda^{(k+1)}$ to minimize $\|\lambda \nabla C(\vec{S}^{(k)}) - \nabla P(\vec{S}^{(k)})\|_2^2$

oldnorm = newnorm

newnorm = $\sqrt{\frac{C(\vec{S}^{(k)})^T C(\vec{S}^{(k)})}{\|\nabla C(\vec{S}^{(k-1)})\|_2^2}}$

$\rho^{(k+1)} = 10\rho^{(k)} \frac{\text{newnorm}}{\text{oldnorm}}$

until convergence

Figure 8 — More Detailed Summary of Algorithm for Power or Area Minimization

Table 1 - Performance of Electrical Optimizer on Delay Minimization

Delay Minimization						
Circuit	Number of Paths	Number of Gates	Delay (ns) Before Opt	Delay (ns) After Opt	Number of Search Points	CPU Time (sec)
Control	1	5	69.6	26.1	9	0.8
	4	12	69.6	26.3	7	1.2
	12	33	69.6	26.2	13	5.5
	37	36	69.6	26.5	10	7.5
Decode	1	5	58.7	19.9	6	0.8
	6	20	59.1	21.0	13	1.6
	10	31	59.1	21.0	13	4.6
	16	46	59.1	21.0	15	6.1

Table 2 - Performance of Electrical Optimizer on Power Minimization (Delay Constant)

Power Minimization					
Circuit	Number of Paths	Number of Gates	Power (mW) Before Opt	Power (mW) After Opt	CPU Time (sec)
Control	1	5	13.1	10.5	1.6
	4	12	29.5	24.1	3.2
	12	33	80.4	52.0	13.3
	37	36	88.6	61.9	24.4
Decode	1	5	13.1	10.2	1.4
	6	20	45.9	39.8	9.1
	10	31	70.5	54.4	20.0
	16	46	112.3	87.5	26.5

CPU times are measured on a VAX 11/750 with floating point accelerator.

Table 3 - Compute Time vs. Problem Size for Delay and Power Minimization on a Single Path

	Power Minimization				Delay Minimization	
Number of Gates	CPU Time(sec)	Number of Subproblems	Search Points Per Subproblem		CPU Time(sec)	Number of Search Points
2	1.8	2	5	3	0.5	3
5	1.4	2	11	3	0.6	6
10	2.0	2	14	3	0.8	6
15	3.9	2	27	5	1.0	7
20	5.8	2	30	4	1.2	7
25	8.2	2	28	15	3.1	16
30	7.1	2	24	2	3.5	14

CPU times are measured on a VAX 11/750 with floating point accelerator

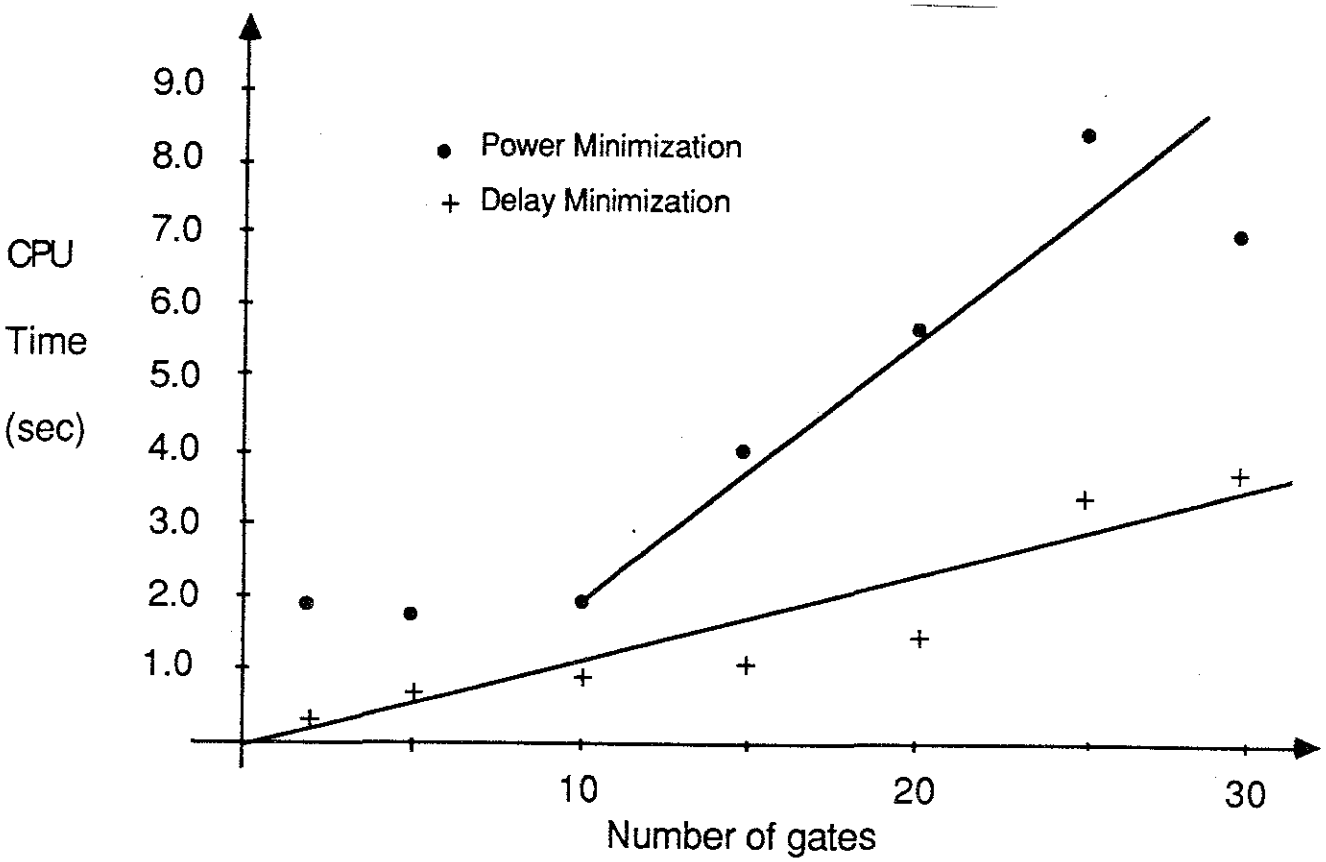


Figure 9 - Delay and Power Minimization on a Single Path as a Function of the Number of Gates (see Table 3)