EPOXY
An Electrical and Physical Layout Optimizer that Considers Circuit Changes

Fred W. Obermeier

Randy H. Katz

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, CA 94720

Research supported by:
AEA Fellowship
NSF DCI-8352227

ABSTRACT

Electrical performance and area improvement are important parts of the overall VLSI design task. Given designer specified constraints on area, delay, and power, EPOXY will size a circuit's transistors and will attempt small circuit changes to help meet the constraints. In addition, the system provides a flexible framework within which to evaluate the effects of different area and electrical models, as well as different optimization algorithms. Since the sum of transistor area is a better measure of dynamic power than cell area, an area model for standard–cell layout is presented. Optimization of a CMOS eight–stage inverter chain illustrates this difference; a typical minimum power implementation is 32.3% larger than the one for minimum area. The combination of a TILOS–style heuristic and augmented Lagrangian optimization algorithm yields quality results rapidly. EPOXY's circuit analysis is from 5 to 56 times faster than Crystal.

Table of Contents

1. Introduction

The task of VLSI IC designers is to produce functional circuits with a specified electrical performance within defined physical limitations. Typically the design process is broken into stages: define the functional specification, generate a circuit that implements the function, and produce a VLSI implementation that meets the performance goals. Transistor sizing is one technique to help achieve the performance objective. EPOXY (the design tool described in this paper) sizes transistors and considers different circuit modifications for meeting the performance goals. It also employs a more accurate area model and provides a flexible framework for comparing different optimization algorithms.

A design's performance is based on electrical information derived from a net–list and layout. From this information, EPOXY constructs symbolic polynomial equations that model the circuit's performance. The optimization techniques use these equations to resize the transistors. If the overall design constraints are still not satisfied, structural changes to the circuit are considered (e.g., inserting buffer stages). The information describing the improved circuit is output as a new net–list and modified layout.

EPOXY provides a flexible framework for evaluating the effects of different electrical and area models, optimization algorithms, and circuit modification heuristics. The system is composed of three parts: performance modeling, optimization, and circuit restructuring as illustrated by *Figure 1.*

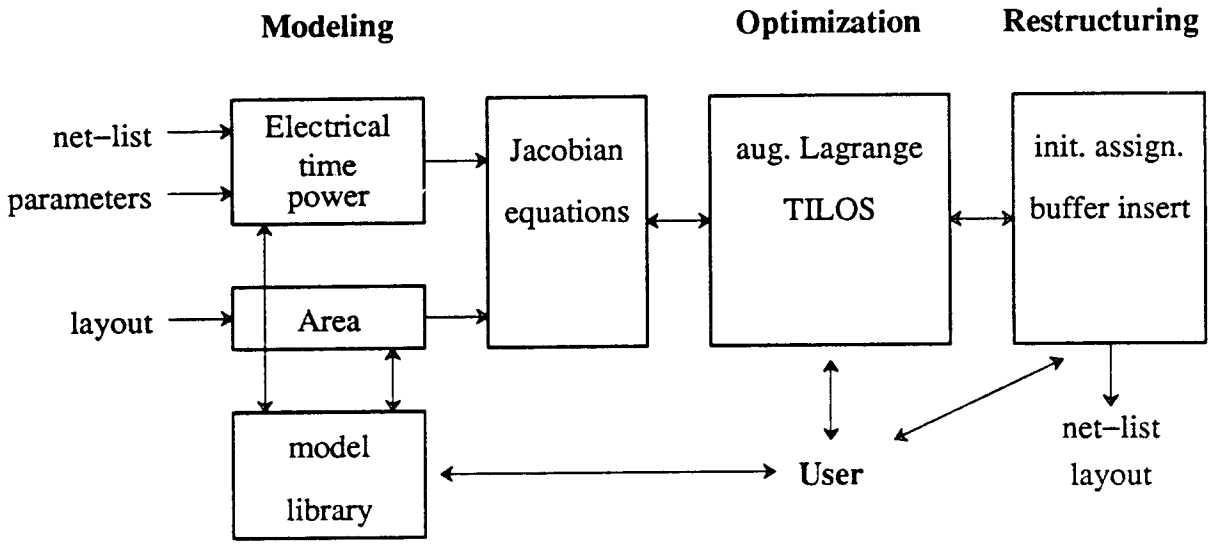


Figure 1. EPOXY system architecture. The net–list, layout, and circuit parameters are converted into symbolic polynomial equations. Additional equations that describe the sparse Jacobian matrix are then derived. An optimization technique is applied to these equations to produce feasible numerical values while minimizing a user specified objective function. Circuit modifications are made by altering the symbolic equations and internal net–list. The optimal numeric values (transistor sizes) and modified circuit layout are returned.

The first section, *modeling*, derives symbolic equations that compute a circuit's electrical performance and area. EPOXY currently employs an accurate area model for standard–cell layout and the distributed RC worst–case electrical models (described in

Section 2).

Symbolic polynomial equations allow circuit performance models to be applied interchangeably so that the effects of modeling accuracy and speed of evaluation on the the overall optimization process can be assessed. That is, model developers can determine the effect of new models on the resulting solution. Better quality optimization can be obtained by changing to more accurate (but more computationally expensive) models within EPOXY's uniform framework.

Abstracting circuit performance by symbolic equations also reduces computation. Any update in the input variables can trigger evaluation of only the effected equations thereby providing model independent incremental simulation. Or these equations can be directly compiled into an optimized object module for each circuit. When common circuit blocks are used, the compiled equations can be rapidly evaluated since the object module can be executed directly. Finally, the expense of computing finite difference approximations of the Jacobian matrix are avoided since the partial differential equations are statically derived.

The next stage, *optimization*, attempts to find an assignment for the input variables of the equations to meet the constraints while minimizing the user defined objective function. The optimization problem is formulated so as to find an assignment for the transistor sizes that satisfies the specified performance requirements. However, the relationship between size and delay is non–linear even for the simple lumped RC model. The system can make use of several different non–linear optimization algorithms. The convergence rate and accuracy for a few non–linear programming algorithms are evaluated in *Section 3*.

Many optimization techniques rely on information provided by a Jacobian matrix. An entry in the Jacobian matrix is defined by a partial derivative equation for each constraint equation with respect to each input. The equations that define the Jacobian matrix are symbolically derived from the performance equations. A parameter file specifies information used in the derivation of the Jacobian matrix such as declared input and output nodes, and transistors of fixed size. The Jacobian matrix is never explicitly stored; rather it is represented by the few derived equations.

The final stage, *restructuring*, alters the optimization problem by making limited circuit changes if constraints cannot be met. These include inserting or removing buffer stages, rearranging transistors within a pull–down or pull–up tree, and splitting large transistors so that cell height and width can be traded off. This level handles the discrete decisions of proposing circuit alternatives while the two other levels determine the best possible implementation for this alternative. (Restructuring techniques are evaluated in *Section 4*.)

Global changes in circuit structure such as boolean resynthesis are best made earlier in the design process. EPOXY deals with the remaining complexity of detailed circuit performance improvement. Therefore, the limited circuit structural changes necessary can be made at a level where the results of these changes can be accurately assessed.

VLSI designs typically incorporate large sections of combinational logic. The equations that model combinational logic can be linearly ordered and evaluated. Circuits with feedback require sets (one for each path) of equations to be solved simultaneously.

Rather than deal with the extra complexity of feedback, the present version of EPOXY restricts its attention to strictly combinational logic. After the equations are generated, they are linearly ordered so that only a single pass is required to provide a valid numerical assignment. Adding feedback requires a more complex equation ordering and solution package.

EPOXY offers a unique approach to performance optimization. Most transistor sizing programs integrate fixed electrical models with a fixed optimization technique (ANDY [Trim83], TV [Joup83], COP [Marp86], TILOS [Fish85]). Other programs that permit a selection of models and optimization algorithms are usually computationally expensive because the two phases are handled by separate programs (DELIGHT [Nye83], APLSTAP [Bray81]). Separation of modeling and optimization routines results in extra storage and communication expense to maintain essentially equivalent information. EPOXY's use of symbolic performance equations combines the execution speed comparable to integrated systems with the flexibility of applying different performance models.

Section 1 details some of the models currently available in EPOXY. The following section (*Section 2*) describes the optimization algorithms implemented within the tool and a comparison of their results. Next (*Section 3*), some circuit alternatives are presented which can be applied in example circuits to improve their performance. A discussion of the general issues such a space/time analysis of the EPOXY system follow in *Section 4*. The final section summarizes the contributions this work offers.

2. Modeling Circuit Performance

A VLSI designer is interested in total area, height, width, maximum delay (time), and total power dissipation. Input parameters include physical parameters such as the placement of transistors, their width and lengths, and parasitic capacitance. Electrical models relate these physical parameters to the electrical performance metrics, i.e., delay and power. The area model provides total height and width and an estimation of layout parasitics.

Abstracting the performance of a circuit by a set of equations gives us the flexibility to evaluate and select the best area and electrical models. The choice of the appropriate model must carefully balance the conflicting requirements of accuracy and computational expense. The effects of modeling accuracy on the resulting optimization are presented below.

The models can be viewed as a set of templates for constructing the equations that compute values for the design metrics (see *Figure 2*). The models are discussed in the following subsections.

Figure 2. A performance model is a template for producing equations that specify design performance metrics, given a circuit description (e.g. a net–list).

2.1. Time

An appropriate design metric for delay must be selected. A VLSI designer is faced with many node delays of potential interest. The worst–case input arrival times and required output response times are usually specified. Therefore, the input node times are fixed (constant) while the output node times are constrained.

Ousterhout has outlined several value–independent models for calculating worst–case delays given layout parasitics and transistor widths and lengths [Oust85]. These include the lumped RC, lumped slope RC, and the distributed slope RC models. The lumped and distributed models differ in how the transistor resistance and capacitance values are combined to produce a worst–case node time. These models determine the effect of each transistor switching independently from all others. "Slope" refers to a more accurate transistor model used for the single switching transistor while all others use the simple linearly–interpolated default transistor model. This more accurate transistor model takes into account the input waveform and fanout effects.

Simple models typically provide less accuracy than more complex models. The simple lumped RC model is usually within 24 percent of SPICE [Vlad81]. Although simple models may be sufficient for ordering critical paths, they are not likely to be accurate enough for optimization which requires absolute comparisons. A few of the electrical modeling issues are outlined in the table below.

Summary of Electrical Model Issues

Model	Eval. Time	Accuracy (Error to SPICE)	Features (Considers)	Problems (Neglects)
Lumped RC	Extremely fast	Average 24%	Continuously differentiable and convex	Overestimates delays Input waveform Fanout effects
Lumped slope RC	Very fast	Average 8%	Input waveform	Fanout effects
Distributed RC	Fast	–	Fanout effects	Input waveform
Distributed slope RC	Slower	Average 6%	Input waveform Fanout effects	
Numerical (SPICE)	Very slow	–		

Equations for the response time of a node are derived using the template: $t_{output\ node} = \max(t_{input\ node} + delay())$ where $t_{input\ node}$ is the time the input changes. A worst–case distributed RC electrical model for *delay*() is currently supported, although other models could be substituted. EPOXY is not restricted to transistor trees as *Figure 3* demonstrates.

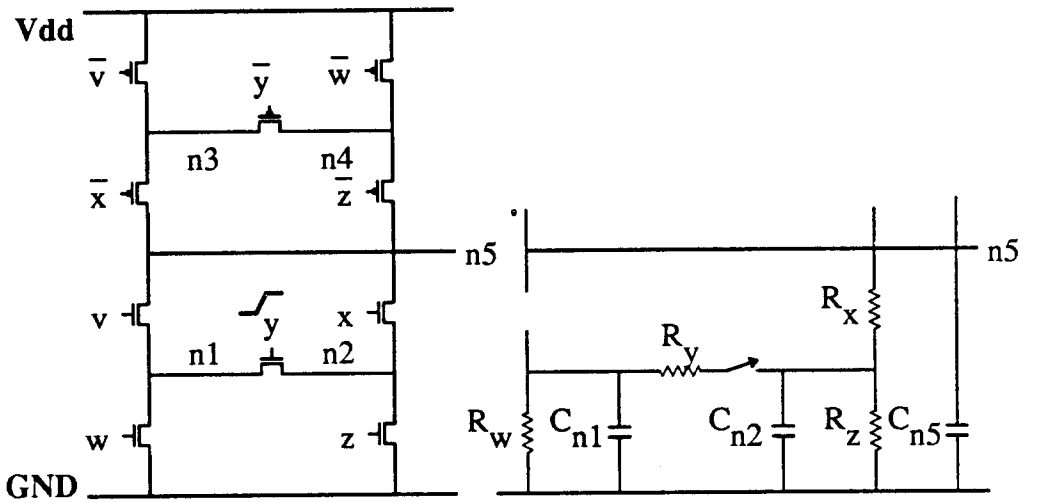


Figure 3. A static CMOS combinational logic gate and the corresponding RC tree for deriving one possible discharge path for node **n5** caused by a rising signal on node **y**, the gate of the transistor (**y**). R_{fet} is the on–resistance of the transistor (*fet*) and C_{node} is the total capacitance on the *node*.

The equations which determine the worst–case response time of node **n5** due to a rising transition on node **y** are:

$$t_{fall,n5} = t_{rise,y} + (R_y + R_w)C_{n2} + (R_x + R_y + R_w)C_{n5}$$

$$t_{fall,n5} = t_{rise,y} + (R_y + R_z)C_{n1} + (R_v + R_y + R_z)C_{n5}$$

The transistor model provides the on–resistances and parasitic capacitances for a given width, length and type (n– or p–channel). EPOXY currently supports a first–order approximation of a transistor's performance. The constants, Kr_{type}, Kg_{type}, and Ksd_{type} are automatically determined by a least squares fit of these modeling equations to SPICE

simulations of small representative circuits. Equal values for the source and drain capacitances ($C_{sd,fet}$) are used.

$$R_{type,fet} = Kr_{type}\left[\frac{l_{fet}}{w_{fet}}\right]$$

$$C_{gate,fet} = Kg_{type}\, w_{fet}\, l_{fet} \qquad C_{sd,fet} = Ksd_{type}\, w_{fet}$$

Finally, node capacitance is determined by adding the parasitic capacitance and the capacitance of attached transistor gates, sources, and drains. The variables appearing in the node capacitance equation are determined by the interconnection of transistors within the circuit. For the example CMOS circuit, the capacitance of node n1 is given below.

$$C_{node} = C_{parasitic} + C_{gate,fet} + C_{sd,fet} + C_{load}$$

$$C_{n1} = C_{parasitic\ n1} + C_{sd,v} + C_{sd,x} + C_{sd,y}$$

One might ask why accurate models are necessary, as long as the ordering of critical paths is maintained. The reason is that aggressive designs are typically forced into the region where a designer must balance minimum delay (maximum speed) with an overall area limitation. A small error in estimating delay to meet a fixed time constraint can have a large impact on overall area. *Figure 4* clearly shows this for the minimization of area (and time) and power for an eight–stage CMOS inverter chain (MOSIS SCMOS λ=0.7μm). A 1ns time error results in a 3,956 μm^2 (8,074 λ^2) error for area minimization and 2,636 μm^2 (5,380 λ^2) error for power minimization. Therefore, if the model underestimates the delay by 16% (1ns), a circuit 100.4% larger than necessary will be reported for a minimum are implementation and 50.1% for a minimum power implementation. Since the lumped RC model has an average error of 25%, transistor sizing tools based on this model such as AESOP [Hedl87] would exhibit even larger area and power errors.

One way to avoid this type of error is to use simpler models to rapidly determine the neighborhood of the potential optimum. Then more accurate models refine the parameter assignment to provide precise values. The following section demonstrates how the optimization code takes advantage of well chosen initial assignments for the input parameters.

2.2. Power

The maximum power constraint for the entire design is usually determined by the IC package. Total power has dynamic and static components. Static power usually dominates for nMOS and pseudo–nMOS design styles, while dynamic power dominates for most other CMOS design styles [West85]. Since dynamic power is a function of the operating frequency, a maximum operating frequency must be defined along with the maximum allowable power.

The static component is determined by the minimum resistance between the power supply signal lines. For NMOS, the smallest resistance occurs when all the transistors are on. Since the static current is mainly determined by the depletion–mode pull–up of each gate, the static maximum power can be approximated by;

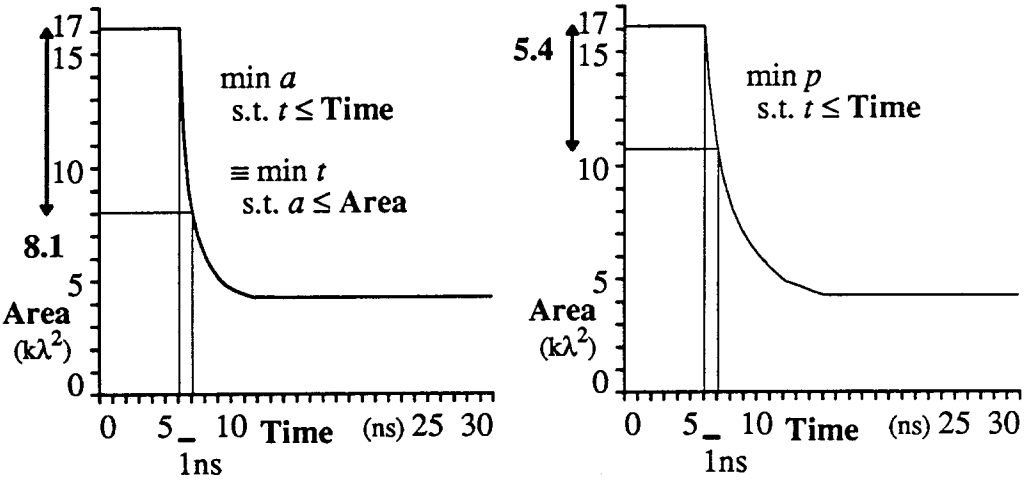


Figure 4. The graphs indicate the area (**Area**) required by an eight–stage inverter chain (MOSIS SCMOS λ=0.7μm) to meet a maximum delay constraint ($t \leq$ **Time**, where t is the response time of the output node and **Time** is the constraint value on the x–axis). The graph on the left shows the minimum area implementation while the one on the right gives the minimum power configuration. The smallest circuit that meets a 7.09ns time constraint requires an area of 8,044λ^2 (3,942 μm^2). The smallest circuit that meets a 6.09ns time constraint requires an area of 16,118λ^2 (7,898 μm^2). Therefore, a 1ns variation translates into an 8,074λ^2 (3,956 μm^2) area difference. Similarly, the smallest power circuit that meet the 7.09ns time constraint requires an area of 10,738λ^2 (5,262 μm^2). This translates into a 5,380λ^2 (2,636 μm^2) difference for a 1ns variation. (Minimization of area subject to a maximum time constraint is equivalent to minimization of time subject to a maximum area constraint. The output has an additional 1pf capacitance load.)

$$power_{static,\,NMOS} = \sum_{gates} \left[\frac{V_{dd}^2}{R_{depletion,\,gate}} \right]$$

For CMOS, the only significant static power component is due to the reverse bias leakage between diffusion regions and the substrate. The power due to leakage current as described by the diode equation is:

$$power_{static,\,CMOS} = V_{dd} \sum_{transistors} i_s \left[e^{\frac{q\,V}{K\,T}} - 1 \right]$$

The saturation current, i_s, is a function of the junction area, but typically 10^{-14} A.

The maximum dynamic power required by a circuit is defined by the capacitance it drives. The power to drive the input nodes comes from external circuitry. C_{driven} is the total capacitance of all nodes that the circuitry drives excluding the capacitance of the supply nodes.

$$power_{dynamic} = C_{driven}\, V_{dd}^2\, frequency_{\max}$$

For the CMOS example, the total power will be approximated by the maximum dynamic power component. The following equations result:

$$C_{driven} = C_{n1} + C_{n2} + C_{n3} + C_{n4} + C_{n5}$$

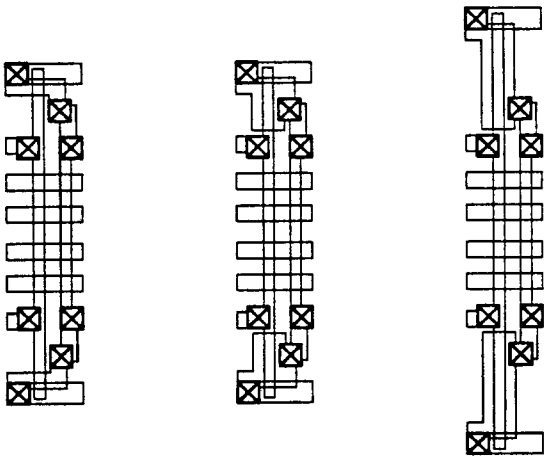
$$power_{CMOS} \approx power_{dynamic}$$

2.3. Area

There are many possible choices for representing the area design metric. A VLSI designer usually needs to generate layout to fit within a given space in an existing design. Or the aspect ratio (ratio of height to width) of a circuit must be maintained so that it can be placed in a realistic package. Therefore, a more appropriate area metric for an optimization program should include cell height and width.

Cell height and width are not determined simply by transistor and routing dimensions alone. The overlap between routing area and total transistor area, significantly impacts the overall area and performance. Therefore, an important element in characterizing layout is how well the area under the routing channel is utilized before the overall dimensions must change. In a structured design environment such as standard cells, this factor can be identified as *free* transistor width. *Free* width is the width that a transistor can grow without affecting the area of the cell. If a transistor width falls within the minimum and *free* width, then the overall cell width will not change. The *free* transistor width is determined by local routing and connections. If one transistor has exceeded its *free* width, then all transistors that share its routing track also benefit from the additional width. This simple factor can rapidly relate transistor size changes to cell height and width changes. *Figure 5* illustrates the effects of *free* transistor width on overall cell area.

Given a circuit architectural style, total height and width can be used as the area metric rather than a simple sum of transistor area. The area model takes into account routing effects on each transistor by determining its *free* transistor width. Since typical standard–cell layout styles employ columns of n– and p–channel transistors, the formulation of the area equations is straightforward.



Transistor width	3 (min)	10 ($w=w_{free}$)	20	λ
Total Trans. Area	12	40	80	λ^2
Total Cell Area	896	896	1176	λ^2

Figure 5. The effect of transistor widths on total cell height and width are illustrated by layout fragments of a standard–cell inverter. The layout on the left contains transistors of minimum width (3 λ, 2.1μm). As long as the transistor width falls below the *free* limit, the layout area will not change (center layout fragment). The layout on the right illustrates the additional cell area required if the transistors increase well above their *free* limit.

$$area = height \; width$$

$$height = \max h_{channels} \qquad h_{channel} = h_{interconnect} + \sum l_{fet}$$

$$width = \sum w_{channels} \qquad w_{channel} = w_{routing,channel} + w_{n,channel} + w_{p,channel}$$

$$w_{type,channel} \geq 0$$

$$w_{type,channel} \geq w_{fet} - w_{free,fet}$$

$$w_{type,channel} \geq w_{power\ buss,channel} - w_{power\ buss,min}$$

An aspect often neglected by transistor sizing programs is that the power busses may need to be widened to accommodate the extra current required by the larger transistors. Wider busses reduce electromigration problems (mainly in NMOS) and transient induced voltage drops (especially in CMOS). Electromigration is related to the average sustained current density, J. When synchronous circuits switch, the power supply lines are required to carry a great deal of current. The transient induced voltage drops are caused by these large currents passing through the resistance drop of the supply lines. The width of the supply lines is no longer a constant; instead it is a function of the loads that the transistors must drive. If the circuit structure is known, a more accurate model can be used to insure the maximum current density is not exceeded. For example, the model for a standard–cell style layout is formulated as:

$$w_{routing,channel} = w_{fixed\ routing,channel} + w_{power\ busses,channel}$$

$$w_{power\ busses,\ channel} \geq w_{\min busses,\ channel}$$

$$w_{power\ busses,\ channel} \geq \left\lceil \frac{power_{static}}{V_{dd}\ J} \right\rceil \qquad \textit{(electromigration)}$$

$$w_{power\ busses,\ channel} \geq \left\lceil \frac{I_{peak}\ R_{sheet}\ length_{buss}}{V_{peak\ tolerated}} \right\rceil \qquad \textit{(reduce supply transients)}$$

Previous transistor sizing tools (AESOP, MOST [Pinc86], TILOS, TV, XTRAS [Kao85]) usually do not model layout height and width. Instead, they use total active transistor area which is actually a better measure of dynamic power than total cell area. The optimization section will show the difference in area, delay and power resulting from a change in the objective function. Typically this can be as much as 32.3% for even the simple eight inverter chain.

Another consequence of choosing a realistic area metric is that there is no longer any need to artificially constrain the maximum size on every transistor as in AESOP. Instead, transistors take on the appropriate size necessary to meet the design constraints.

2.4. Noise Margins

Digital circuits require immunity from noise in order to produce logic results. Noise margins specify the allowable noise voltage on the input which still produce acceptable logic voltage values on the output. The noise margin for a gate is directly related the ratio of the pull up resistance to the pull down resistance [West85]. Since changes in transistor size effect this ratio, we may wish to restrict the ratio to assure an acceptable noise margin. NMOS circuits are much more sensitive to variations in this ratio than CMOS circuits.

$$NM_{min} \leq \frac{R_{pull\ up}}{R_{pull\ down}} \leq NM_{max}$$

2.5. Other issues

The choice of models for any of the design parameters should depend on the accuracy and error margin of the underlying transistor parameters and parasitic values. These values can be based on real layout or they can be estimated. Clearly, highly accurate analysis is of little use if the parameters have a large margin of error. Given a layout style, the parasitic values can also be estimated.

3. Optimization

Area, delay, and power optimization can be formulated as a classical non–linear optimization problem (NLP):

$$\begin{array}{ll} \textbf{min} & f(x) \\ \textbf{subject to} & g(x) \leq 0 \end{array}$$

The objective function, $f(x)$, defines the quality of a configuration represented by the independent vector of variables, x. For electrical performance optimization, a design may be specified by an input parameter assignment such as the widths of transistors. Minimization of a cost function is equivalent to maximizing a corresponding quality function. The constraint functions, $g(x)$, define requirements on the independent variables such as minimum transistor width, overall maximum cell width, and delay constraints.

Performance optimization is inherently non–linear since transistor width (an area metric) is inversely proportional to delay, even for a simple lumped RC electrical model. This non–linear relationship will appear in at least one of the constraint equations. In general, NLP's are difficult to solve.

All optimization techniques used to solve this type of problem share a common approach. An initial assignment is either given by the user or generated by the program. Usually minimum sizes are assigned to all devices. Next the assignment is analyzed by estimating parasitics and then performing a timing analysis or circuit simulation. This is often the slowest step in the optimization process. If the assignment is feasible (meets the constraints) and the objective function has reached a minimum, then the loop terminates. The loop may also terminate if a maximum iteration limit has been reached. Otherwise, another assignment often based on previous assignments is generated and passed to the analysis section, completing the loop.

First we will describe the design space for a CMOS example to illustrate the typical performance tradeoffs available. Then a comparison of some algorithms and heuristics will be presented.

3.1. Performance Envelope

Formulation of the circuit design problem as a general optimization problem allows us to apply a variety of objective functions. All designs of interest to a VLSI designer lie within a performance envelope. *Figure 6* describes the performance envelope for an eight–stage inverter chain.

The graph on the left shows the area required to implement an eight inverter chain to meet a given time constraint. The **X** on the upper left part of the graph marks the performance of the fastest (minimum time) implementation regardless of power consumption and area. The lower right **X** indicates the performance of the implementation with minimum device sizes. This configuration needs the smallest area and power. These two points are connected by curves which give the minimum area and minimum power implementations for a given time constraint. For the area versus time graph, there exists no circuit smaller than the minimum area implementation that meets the time constraint. Circuits that draw more than the minimum power implementation would not be of

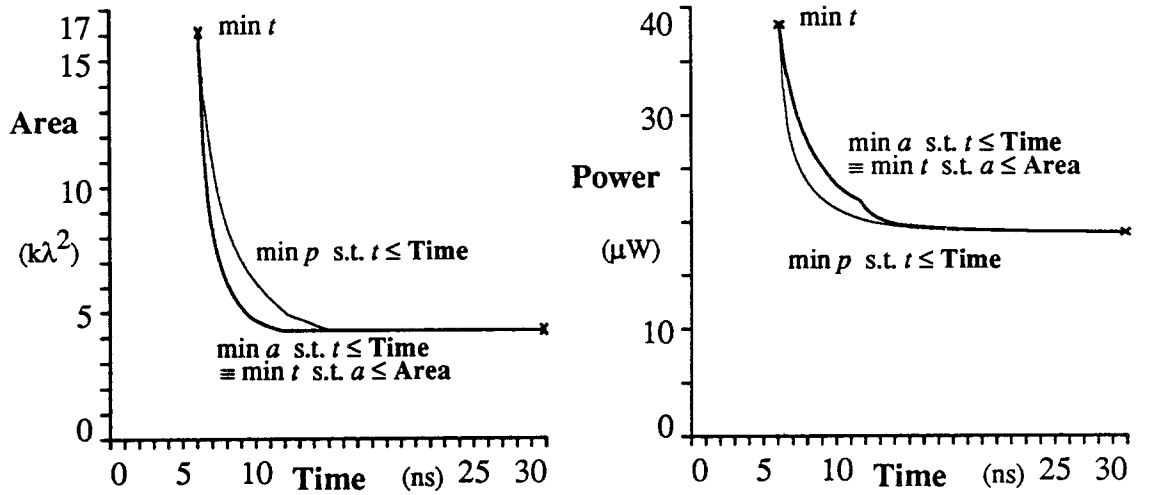


Figure 6. Graphs of the area required (left graph) and power required (right graph) to implement the minimum power (*min p*) and minimum area (*min a*) configurations of an eight–stage CMOS inverter chain subject to a maximum time constraint ($t \leq$ **Time**, where t is the response time of the output node and **Time** is the constraint value on the x–axis). The output has an additional 1 pf capacitance load.

interest. Therefore, the *only* circuits of interest lie between the two curves. The power versus time graph describes a similar situation. Circuits between the two curves represent a tradeoff between the requirements of area, power and time.

These graphs clearly show the effects of the more accurate area model. As the circuit delay decreases, the required area does not change above the minimum size implementation (the flat portion) until at least one transistor exceeds its *free width.* Thereafter, additional area is required for circuit with less delay.

minimization	constraint	area λ^2	(μm^2)	power μW
min power	time ≤ 10ns	6188.35	3032.39	21.22
min area	time ≤ 10ns	4677.26	2291.86	23.48

The table (above) contains data extracted from the graphs of *Figure 6.* For a 10ns implementation, the minimum power implementation is 32.3% larger than the minimum area implementation. Therefore if one uses total transistor area to approximate total cell area, the resulting solution may be as much as 32.3% larger than necessary.

3.2. Comparison of Optimization Techniques

EPOXY's flexible architecture allows a comparison of the different optimization techniques, independent of performance models. We will compare an augmented Lagrangian algorithm similar the one used in COP [Marp86] and a heuristic technique used in TILOS.

3.2.1. Augmented Lagrangian Method

Augmented Lagrangian methods convert a constrained minimization problem into an unconstrained sub–problem [Gill81]. The implemented augmented Lagrangian algorithm handles non–linear inequality constraints by smoothing discontinuous derivates at the solution (if any constraints are active) by the function, $L_A(x,\lambda,\rho)$.

$$\min f(x) \;\; s.t. \;\; g_i(x) \le 0 \;\; \equiv \;\; \max_{\lambda} \min_{x} L_A(x,\lambda,\rho)$$

$$L_A(x,\lambda,\rho) = f(x) + \sum_i \begin{cases} \lambda_i\, g_i(x) + \dfrac{\rho\, g_i(x)^2}{2}, & \text{if } \lambda_i + \rho\, g_i(x) > 0; \quad \textit{active} \\ -\dfrac{\lambda_i^2}{2\rho}, & \text{if } \lambda_i + \rho\, g_i(x) \le 0; \quad \textit{inactive} \end{cases}$$

Note that the function and its gradient are continuous at any point where a constraint changes from *active* to *inactive*. Associated with each constraint equation, $g_i(x)$, is an extended Lagrange multiplier, λ_i. The parameter, ρ, is a penalty factor.

The inner loop, the minimization of $L_A(x,\lambda,\rho)$ with respect to x, is performed by an iterative Golden–rule line search that follows the steepest descent direction, d_j. The input vector, x is updated after each minimization iteration, k.

$$d_j = \frac{\partial f(x)}{\partial x_j} + \sum_j \begin{cases} \dfrac{\partial g_j(x)}{\partial x_j}\left[\lambda_i + \rho\, g_i(x)\right], & \text{if } \lambda_i + \rho\, g_i(x) > 0; \\ 0, & \text{if } \lambda_i + \rho\, g_i(x) \le 0; \end{cases}$$

$$x_j^{k+1} = x_j^k - \alpha\, d_j\,;$$

If the search direction, d_j, is too small (nearly a zero vector) or the traversal in the search direction, α, is too small, an outer maximization iteration is performed by updating λ (iteration count v). If the rate at which the constraints become satisfied is too slow, the penalty factor, ρ is slightly increased. Similarly, if the average value of λ is growing too quickly, ρ is slightly decreased.

$$\lambda_i^{v+1} = \begin{cases} \lambda_i^v + \rho\, g_i(x), & \text{if } \lambda_i^v + \rho\, g_i(x) > 0; \\ 0, & \text{if } \lambda_i^v + \rho\, g_i(x) \le 0; \end{cases}$$

The augmented Lagrangian method converges reasonably fast (linearly) to the optimum solution (if one exists) [Marp86]. Its primary advantage is that progress toward the optimum solution can be guaranteed regardless of the initial solution.

3.2.2. TILOS Heuristic

Another approach to solving the minimization problem is to apply heuristics that encode previous knowledge. Convergence to an optimal value cannot be assured; however, they provide rapid improvement toward a potentially better assignment.. Therefore, they can be used to rapidly locate a feasible solution. Also, heuristics can assure integral

widths and lengths required by lambda–based design styles. Since heuristics are generally problem specific, the types of objective functions permitted would also be restricted.

TILOS is a heuristic transistor sizing program that iteratively increases the transistor width along the worst critical path. This heuristic selects the transistor that reduces the delay most while incurring the least total transistor area increase. The process terminates when all delay constraints are satisfied or when no transistor exists that will decrease the delay. TILOS is essentially restricted to power (sum of transistor area) minimization subject to delay constraints.

We extend the TILOS heuristic for use in general optimization problems by generalizing the notion of critical paths in terms of the underlying performance equations. A set of dependent limiting or failing constraints ($g_i(x) \geq 0.0$) and the equations which interrelate them, will be considered a critical path within the generalized TILOS–style heuristic. Only the input variables to these critical equations can improve the failing constraints. Therefore, while there are failing constraints, the input variable that improves the worst failing constraint the most will be increased. After all constraints are satisfied, the input variable that best improves the objective function while maintaining feasibility (satisfied constraints), will then be increased.

TILOS relies on the general convex nature of the power/delay relationship. However, the point at which other paths become critical causes discontinuities that degrade the results TILOS can produce. One way to overcome this problem is to factor near–critical paths into the sensitivity calculations. For the equation abstraction, this simply means combining Jacobian matrix rows for constraints on the same output variable that are within some ε (e.g. $g_i(x) + \varepsilon \geq 0$).

Since TILOS only increases transistor sizes by a fixed amount, it usually overshoots the minimum possible implementation. If the algorithm were generalized so that decreasing transistors sizes were considered to improve the overall power/delay savings, better solutions should result. However, endless cycling between increasing and decreasing transistors may result. Also, the fixed small increment size usually results in many iterations.

3.2.3. Comparison of Results

To compare these optimization algorithms, we should first establish a set of optimal values for various CMOS minimization problems (for a fixed set of CMOS technology parameters):

Circuit		Performance					
Metric		A	H	W	P	T	Itr
Units		λ^2	λ	λ	uW	ns	
Improvement		(%A)	(%H)	(%W)	(%P)	(%T)	
inv.8	$w=w_{min}$	4288	134	32.0	19.0	29.82	1
C=1pf	min t	11709 (173)	134 (0)	87.4 (173)	32.4 (70)	**6.46** **(-78)**	12414
	min a t≤7ns	**8331** **(94)**	134 (0)	62.2 (94)	31.6 (66)	7.00 (-77)	43103
	min p t≤7ns	11296 (163)	134 (0)	84.3 (163)	**26.9** **(41)**	7.00 (-77)	32572
rand.20	$w=w_{min}$	7,140	140	51	31.8	26.6	1
C=1.0pf	min t	10,276 (44)	140 (0)	73.7 (44)	41.3 (30)	**6.42** **(-76)**	11031
	min a t≤7.11ns	**9,180** **(29)**	140 (0)	65.6 (29)	40.7 (28)	7.11 (-73)	13862
	min p t≤7.11ns	9,709 (36)	140 (0)	69.4 (36)	**40.2** **(26)**	7.11 (-73)	17911

The table lists three optimization problems for each static CMOS circuit: an unconstrained minimization of of delay (min t), and time constrained minimization of area (min a s.t. t≤ ...) and power (min p s.t. t≤ ...). The first row, ($w = w_{min}$), gives the performance of the circuit with all devices of minimum size. Performance is defined as the total cell area (A), height (H), width (W), power (P), maximum delay (T) and number of iterations (Itr) that the augmented Lagrangian algorithm took to produce the result. The numbers inside parenthesis gives the percentage increase or decrease over the minimum sized implementation for each of the design parameters. The extra load capacitance that is added to the outputs is listed as (C=...pf). A maximum frequency of 500 kHz was used to calculate the maximum dynamic power (P) for each of these examples.

The table shows the significant performance improvement transistor sizing offers. It also illustrates the minimum required sacrifice in the other design metrics to achieve the desired performance. For example, $7{,}421\lambda^2$ additional area is required to achieve the fastest implementation, 6.46ns. This translates into a 173 percent increase in area and a 70 percent increase in power for a 78 percent decrease in delay.

These problems present a mix of optimization criteria whose solution is non–trivial. Even thought the eight–stage inverter chain (inv.8) is a small circuit, the values of the resulting performance metrics for each optimization require a greater degree of accuracy than for larger circuits. In other words, a small change in the input for a smaller circuit will have a larger impact on the design metrics than for a larger ones.

A graphical comparison of the convergence rate for the augmented Lagrangian algorithm and the heuristic approach used in TILOS is given in *Figure 7*. The convergence of an area minimization subject to a 10ns time constraint (min a t≤10ns) for the

eight–stage inverter chain is shown in *Figure 7.*

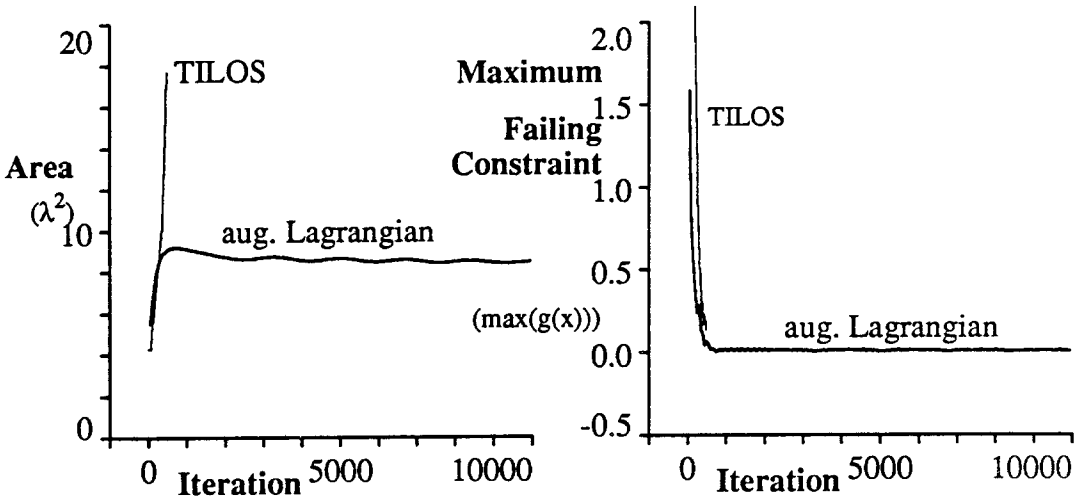


Figure 7. These graphs illustrate the convergence rate for an area minimization subject to a 10ns time constraint of a CMOS eight–stage inverter chain with a 1pf load. The graph on the left gives the worst-case output response time for the inverter chain at every 50 iterations of each optimization routine. On the right, the amount by which the worst constraint fails (max $g(x)$) is also plotted against the iteration number for each optimization algorithm. In fact, the constraints are satisfied rapidly.

Figure 7 shows the convergence rate for the eight–stage inverter chain. Even thought the augmented Lagrangian method satisfies the constraints rapidly, the minimization of the objective function (time) is slower. The heuristic used by TILOS reduces the delay constraint faster; however, it produces a larger implementation.

Figure 8 shows the convergence rate for the power minimization of the eight–stage inverter chain. Again, the generalized TILOS heuristic produces a feasible solution rapidly, but with a much larger power requirement. When used as a starting point for the augmented Lagrangian algorithm, the objective function is reduced first at the expense of the maximum failing constraint. When the results of generalized TILOS heuristic are used as a starting point for the augmented Lagrangian method, results comparable to the original augmented Lagrangian method are eventually obtained. For this example, the augmented Lagrangian achieved similar quality results using fewer iterations than the combination of both algorithms. Using the generalized TILOS heuristic for establishing a initial solution rapidly may not be a good strategy for problems that are very aggressive since the heuristic tends to over-shoot the optimum solution.

The computation to achieve quality results can be reduced further by using simple models to locate the probable region of the global minimum. Then accurate models could refine the search further. Simple models are rapidly evaluated and lead to a fast solution since the design space is convex. Convexity implies that a local minimum is a global minimum. Accurate models take longer to evaluate and represent a more complex design space that may exhibit local minima. This approach also avoids getting trapped in local minima early in the optimization.

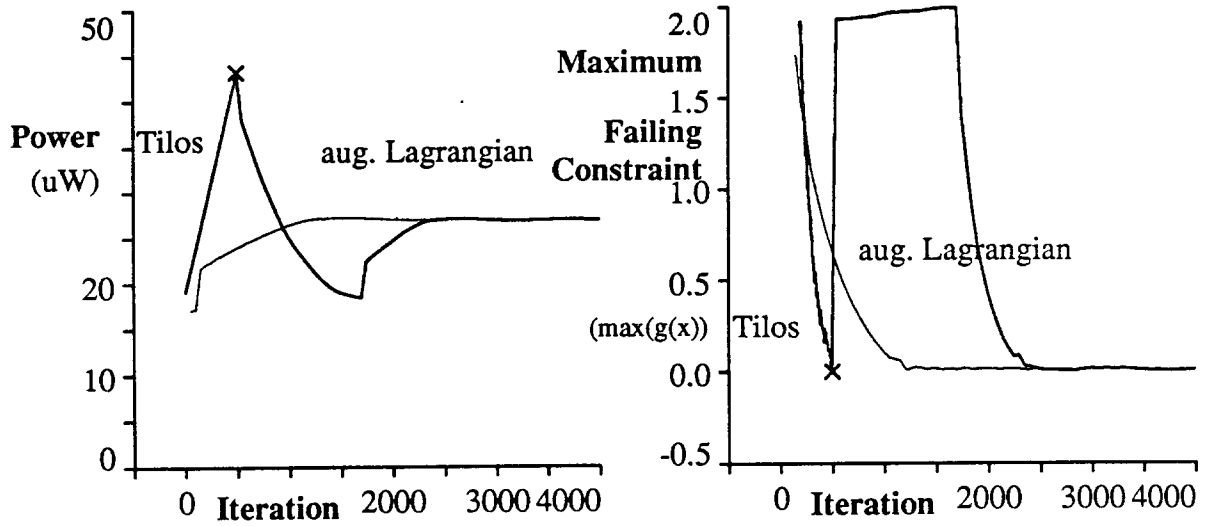


Figure 8. These graphs show the convergence rate for a power minimization subject to a 7ns time constraint for the CMOS eight–stage inverter chain with a 1pf load. The graph on the left shows the maximum power for the inverter chain using the augmented Lagrangian alone (thin line) and when the generalized TILOS heuristic proceeds the augmented Lagrangian (thicker line). The cross indicates the point where the augmented Lagrangian uses the results of the generalized TILOS heuristic for further optimization. The graph on the right gives the value of the worst failing constraint. The convergence criterion for this optimization (0.5%) is not as tight as those used to generate the table of optimized results.

4. Circuit Alternatives and Performance Improvement Heuristics

Optimization techniques involve systematically generating and analyzing different solutions. Given a functional description and desired performance, there are many ways to generate alternatives. The standard approach is to change the widths and lengths of transistors. An alternative is to modify the circuit structure. For example, the number of stages can be changed to drive output capacitances faster. Splitting stages (replicating a logic gate) can reduce the capacitance on a critical path. Other circuit structures such as superbuffers (in nMOS), differential sensing, cascode connections, and clocked drivers can be employed [Glas84].

Even though there may be many implementations that meet the given functional description, many may not meet the performance specification. The optimization section has shown algorithms for handling the transistor sizing aspect of design improvement. A NLP could be formulated to included (integral) circuit structural changes, however, integer NLPs are difficult to solve. Therefore, EPOXY separates the transistor sizing aspects (optimization) from the circuit restructing section.

Heuristics provide a mechanism for encoding previous design knowledge in circuit improvement techniques. These usually alter the circuit structure so that some failing design constraint can be met. Therefore, if a circuit cannot meet the design constraints with transistor sizing, then the circuit structure will be altered.

Many heuristics are motivated by a critical path analysis. Although a critical path usually refers to a limiting delay path from input to output nodes through transistors, the

concept can be generalized to all design metrics. The equation abstract allows EPOXY to uniformly provide critical path information. A critical path is identified as a list of limiting (or unsatisfied) constraints and their variables.

Another advantage of the equation abstraction is that limited circuit modifications result in limited changes in the equations. Since EPOXY constructs performance equations in memory, circuit modifications are quick and equation evaluation times are still small. Therefore, EPOXY modifies the underlying equations instead of the actual circuit.

4.1. Buffer Insertion Heuristics

Circuit delay can be reduced by inserting buffers (inverter pairs). Also, cell area and power may be reduced by the resulting balance of load capacitance among the number of logic levels. Two strategies for inserting buffers are outlined below.

4.1.1. Load Reduction Heuristic

Circuit speed can be increased by reducing the capacitance load on nodes along the critical delay paths. *Figure 9* illustrates two techniques that reduce the effect of this load by isolating non–critical circuitry.

If a node along a critical delay path has a large capacitance load due to non–critical gate inputs, these inputs can be isolated from the critical node by a buffer. The buffer will contribute additional delay to the isolated inputs possibly creating new additional critical paths. Therefore, the only nodes that can be clearly isolated are those that are not within a buffer delay of the critical path.

Although the technique illustrated in *part c* of *Figure 9* does not insert a buffer, it does fall within the load reduction heuristic. Replicating the nand gate will increase the load on nodes **a** and **b** but will decrease the load on node **k** considerably.

Which of these two techniques to use depends on the available slack in delay for the non–critical inputs and the area, height and width limitations that a particular problem imposes. These decision parameters can be determined for each supported technology. The performance metrics for these CMOS circuit fragments are given by the following table (all input nodes switch at 0ns and $freq_{max}$ = 500kHz). Since any additional circuitry may increase the area requirement, circuitry is inserted at the most loaded critical nodes first.

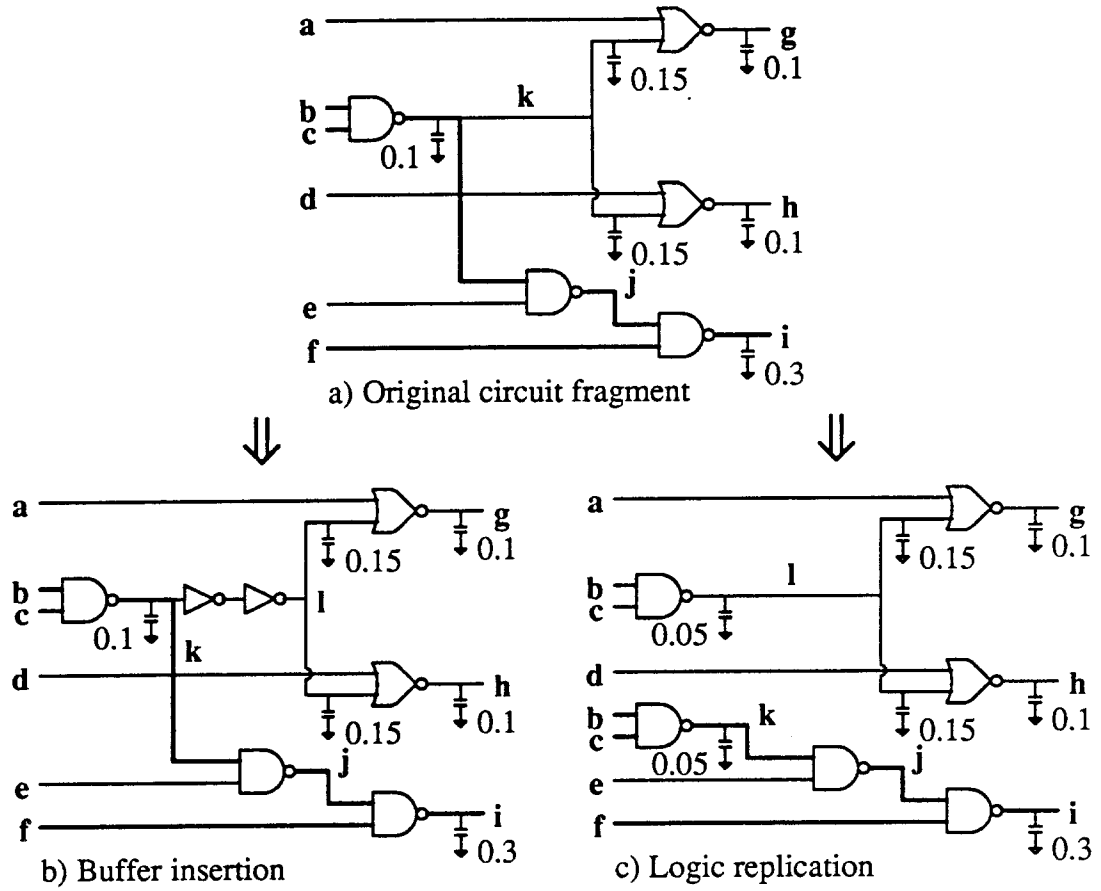


Figure 9. Circuitry along part of a critical delay path is shown in *part a.* (The critical path is highlighted. The parasitic wiring capacitance in pF is also listed.) One way to isolate the non–critical inputs on node *k* is insert a buffer as shown in *part b.* Another is to replicate the logic gate for driving non–critical inputs as *part c* illustrates.

circuit	Area μm^2	Height μm	Width μm	Time (t_i) ns	Power μW
a) original	4269	50.4	84.7	23.2	15.3
b) buffer insertion	5503.7	50.4	109.2	18.9	16.8
c) logic replication	5606.6	56.0	100.1	20.7	18.5

4.1.2. Increased Drive Heuristic

Another buffer insertion strategy is to increase the current drive capability (sourcing and sinking) of a gate with large capacitance loads. Several authors have shown that an optimal number of stages and stage size ratio are required to drive a large capacitance with minimum delay [Mead80] [Neme84] [Hede87].

This strategy works as follows; transistors and nodes along the critical delay path are examined. If space permits (height and width restrictions are not exceeded) buffers are inserted at a high capacitance node which is driven by an input on a low capacitance node. The delay for each stage will be better balanced thereby decreasing the overall

path delay.

4.2. Reordering of Transistors

Reordering transistors along a pull–down or pull–up path can dramatically affect the delay of the output node. Transistors inputs (gate nodes) that switch relatively early should be placed near the supply rails so that the sources and drains of the remaining transistors can be properly set. Alternatively, transistor inputs that switch later should be placed near the output nodes so that the delay of driving successive intermediate node capacitance is reduced.

Rearrangement of transistors must preserve the logical function of the gate (i.e. series and parallel groups). However, dynamic gates impose a restriction on the maximum parasitic capacitance on the non–output nodes. This is to avoid potential charge sharing problems. However, this is not an issue for static CMOS logic.

5. Space / Time Tradeoff

Circuit optimization programs spend most of the time in the analysis portion, since many devices must be reevaluated for each major circuit change. Crystal [Oust85] determines worst–case node delays by constructing the RC paths dynamically. After a node delay is determined, the data structures corresponding to the path are released. Therefore, an optimization tool built directly into Crystal would constantly recreate and destroy the same data structures.

Instead, EPOXY builds a static set of performance equations for the circuit. The models for EPOXY can be built dynamically and interpreted or compiled to form an executable binary. The decision to statically derive the equations instead of a dynamic derivation results in an increase in storage for a decrease in evaluation time. The first table (below) gives statistics for each static CMOS circuit. The second table shows the execution time savings and extra storage requirements for determining the worst–case delay times for all circuit nodes. The time required to perform 1,000 circuit evaluations is given.

Circuit Statistics							
Circuit	nodes	fets	vars	eqs	cnstr.	Jin	%nzJe
inv.8	11	16	594	586	43	30	10.6
inv.10	13	20	782	776	51	36	9.2
rand.20	16	20	559	563	65	36	8.5
adder.1	22	34	1360	1692	108	57	11.5
adder.8	131	224	7891	10321	663	342	2.2

The number of nodes and transistors (fets) for each of the circuits is given. The number of equations (eqs) and variables (vars) EPOXY constructs to evaluate circuit performance using the worst–case distributed RC electrical model and the standard cell area model. The table also includes the size of the Jacobian matrix as the number of constraints (cnstr) and number of Jacobian input variables (Jin). The last column is a measure of the sparsity of the Jacobian matrix. The sparsity of the Jacobian matrix is measured as the maximum percentage of entries (%nzJe) that can be non–zero (i.e., have defining equations).

A Comparison of Crystal and EPOXY									
Circuit	Crystal		EPOXY						
			compile				interpret		
	bytes	sec	bytes	build sec	cc sec	run sec	bytes	build sec	run sec
inv.8	48000	75.4	32768	1.1	8.2	3.0	46150	0.9	38.3
inv.10	56000	97.5	32768	1.4	9.8	3.7	53750	1.1	47.4
rand.20	32000	87.4	40960	1.8	10.8	3.9	54901	1.4	48.2
adder.1	32000	360.6	40960	3.4	21.4	7.7	87826	2.7	86.9
adder.8	104000	7327.6	114688	11.6	151.7	45.9	441029	7.6	508.6

The table (above) gives the execution times for simulating each static CMOS circuit 1,000 times, using Crystal and EPOXY. All times (sec) are for a Sun–3/140‡ with a 68881 co–processor and 8Mbytes of memory runing Sun UNIX† 4.2 Release 3.2. Crystal only lists the maximum heap storage for the circuit. Therefore, the number of bytes reported is rounded to the nearest block size. The compiled version produced by EPOXY gives the size of the executable binary (bytes), time to produce the C file (build), time to compile (cc) and run (run) the program. The interpreted version gives the allocated storage (bytes) for the circuit and its running time (run).

Since this table shows that EPOXY *always* runs faster than Crystal (average of 56 times faster for compiled and 5 times faster for interpreted), we feel that the decision to statically derive the equations is indeed justified. It also shows that compiling the performance equations is always desirable even when the construction (build) and compile (cc) time are included. Compile time can be substantially reduced if object modules are available for frequently used circuits. However, compilation of the equations sacrifices the possibility of modifying the circuit structure since the equations are fixed.

The storage requirements for even the interpreted version of EPOXY suggest that an 8Mbyte Sun–3 can handle a circuit of at least 1000 transistors without paging.

† UNIX is a trademark of Bell Laboratories.

‡ Sun–3 is a trademark of Sun Microsystems, Inc.

6. Conclusions

EPOXY provides a unique flexible framework for evaluating the effects of different electrical and area models, optimization algorithms and circuit modification heuristics. Design performance is determined by generating and evaluating symbolic polynomial equations. When these equations are interpreted, EPOXY is (on average) 5 times faster than Crystal. However, when the equations are compiled, EPOXY achieves an average speedup factor of 56 over Crystal. For the present implementation, a circuit with 1000 transistors should adequately fit within 8 Mbytes.

Accurate models are crucial in performance optimization since a small error in meeting a time constraint (16%) can typically result in much larger area (100% more area) and power dissipation (50%) than necessary.

The sum of transistor areas is not an appropriate measure of total cell area. Rather it is a better approximation of dynamic power. The optimization of an eight–stage inverter chain shows that the minimum power implementation is 32.3% larger than the one for minimum area for a 10ns time constraint. A more accurate area model takes into account local routing and connections.

The flexibility of the optimization routines is demonstrated by applying several different objective functions to derive the performance envelope for a circuit. The envelope defines the maximum region of circuit performance available through transistor sizing. Even though the convergence of the augmented Lagrangian algorithm is slower than heuristic based techniques, the resulting circuits have better performance characteristics.

A list of circuit alternatives is presented for meeting difficult design constraints. These techniques are applied rapidly by altering the performance equations for the circuit.

We intend to add the slope models to EPOXY to provide a richer set of accurate electrical models. Crystal implements the more accurate transistor resistance model as an interpolated table of values. The required partial derivatives of this function can be efficiently implemented by another table which represents a *static* finite difference analysis of these data values.

David Marple provided us with an example formulation of the augmented Lagrangian algorithm. The inner loop minimization loop of the augmented Lagrangian algorithm uses the Golden rule line search. However, a line search based on the Armijo step size can reduce the number of iterations and resulting computation by a factor of 3, and we intend to incorporate this change.

7. Selected Bibliography

[Bray81] R. Brayton, G. Hachtel and A. Sangiovanni-Vincentelli, "A Survey of Optimization Techniques for Integrated-Circuit Design", *Proc. of the IEEE,* 69, 10 , October 1981, pp. 1334-1362.

[Fish85] J. P. Fishburn and A. E. Dunlop, "TILOS: A Posynomial Programming Approach to Transistor Sizing", *IEEE ICCAD*, pp. 326-328, November 1985.

[Gill81] P. Gill, W. Murray and M. Wright, "Practical Optimization", *Academic Press*, London, 1981.

[Glas84] L. A. Glasser and L. P. J. Hoyte, "Delay and Power Optimization in VLSI Circuits", *21st ACM/IEEE DAC Conf.*, June 1984.

[Hede87] N. Hedenstierna and K. O. Jeppson, "CMOS Circuit Speed and Buffer Optimization", *IEEE Transactions on CAD,* CAD-6., 2 , March 1987, pp. 270-281.

[Hedl87] K. Hedlund, "Aesop: A Tool for Automated Transistor Sizing", *Proc. 24th IEEE DAC*, pp. 114-120, June 1987.

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

[Kao85] W. H. Kao, N. Fathi and C. Lee, "Algorithms for Automatic Transistor Sizing in CMOS Digital Circuits", *22nd ACM/IEEE DAC Conference*, pp. 781-784, June 1985.

[Marp86] D. P. Marple and A. E. Gamal, "Area-Delay Optimization of Programmable Logic Arrays", *Advanced Research in VLSI, Proc. of the 4th MIT Conference*, pp. 171-196, April 1986.

[Mead80] C. Mead and L. Conway, "Introduction to VLSI Systems", *Addison-Wesley*, Reading, Mass., 1980.

[Neme84] M. Nemes, "Driving Large Capacitances in MOS LSI Systems", *IEEE Journal of Solid-State Circuits,* sc-19, 1 , February 1984.

[Nye83] B. Nye, A. Sangiovanni-Vincentelli, J. Spoto and A. Tits, "DELIGHT.SPICE: An optimization-Based System for the Design of Integrated Circuits", *IEEE Proc. of the Custom IC Conf.*, pp. 223-238., May 1983.

[Oust85] J. K. Ousterhout, "A Switch-Level Timing Verifier for Digital MOS VLSI", *IEEE Transactions on CAD,* CAD-4, 3 , July 1985.

[Pinc86] J. Pincus, "Transistor Sizing", UCB/Computer Science Dpt. 86/285, UC Berkeley, Computer Science Division (EECS), February 1986.

[Trim83] S. Trimberger, "Automated Performance Optimization of Custom Integrated Circuits", *Proc. International Symposium on Circuits and Systems*, pp. 194-197, 1983.

[Vlad81] A. Vladimirescu, K. Zhang and A. R. Newton, *SPICE Version 2G User's Guide*, Dept. Of Elect. Engin. and Comp. Sciences, UC Berkeley, CA, 94720, Aug. 10, 1981.

[West85] N. Weste and K. Eshraghian, "Principles of CMOS VLSI Design, A Systems Perspective", *Addison-Wesley*, Reading, Mass., 1985.