RC 21191(94676) (05/21/98)
Engineering and Technology
20 pages

IBM Research Report

Electrical and Timing Simulation

Chandu Visweswariah

IBM Research Division
T.J. Watson Research Center
P. O. Box 218
Yorktown Heights, New York

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

Electrical and Timing Simulation

Chandu Visweswariah
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598

In the early days of integrated circuits, a discrete "breadboard" version of the circuit was constructed to test the design before making the first chip. In modern practice, verification of the functionality and performance of a circuit is carried out by means of computer-aided design (CAD) software. Circuit simulation is an essential step before subjecting an integrated circuit design to a costly manufacturing process. This article will introduce the reader to CAD tools used for electrical and timing analysis of circuits, with particular emphasis on digital integrated circuits. The tools model circuit elements, typically by a set of equations, and then solve the resulting equations to predict the circuit's behavior. Simulation is used to verify a circuit's functionality and performance, to study design alternatives and to optimize circuits.

Circuit simulation is a numerically intensive task and the huge amount of simulation required to verify all the details of a complex integrated circuit far outstrips the capability of state-of-the-art simulation algorithms. Although full-chip or exhaustive simulation is rarely feasible, circuit simulation is an essential step in characterizing, designing and optimizing smaller blocks of circuitry, packages and discrete circuits.

Perhaps the best-known circuit simulator is SPICE [1]. In SPICE, electronic devices are modeled by accurate nonlinear equations and the resulting circuit equations are solved using numerical methods. Although the device models may contain inaccuracies and the numerical algorithms solve the equations only to a pre-determined accuracy, this type of simulation will be referred to as "exact simulation" in the rest of this article. Unfortunately, exact simulation cannot be applied to large circuits because the computer resources required for such simulation quickly become inordinate.

In an effort to simulate much larger circuits, "timing simulators" were developed. These simulators make approximations in order to speed up the analysis. Often, they apply only to digital circuits. They run two orders of magnitude faster than exact simulators, but at reduced accuracy (typically producing timing results within 10% of their exact counterparts). Switch-level and logic simulators sacrifice accuracy and the notion of circuit timing to gain even larger speedups in simulation efficiency. Figure 1 shows qualitatively the relative accuracy and efficiency of the different types of simulators. Each type of simulation has a practical limit on the size of circuit that can be accommodated, beyond which the memory or CPU time requirements are exorbitant. For example, it may be possible to simulate a circuit macro containing tens of thousands of transistors using exact simulation methods. In the context of a microprocessor, a full adder may be amenable to exact simulation. However, anything larger like an instruction unit or memory controller can only be handled by timing or switch-level simulation. At the full-chip level, logic simulation is often the only practical alternative. On an entire multi-processor system, logic simulation is barely practical for a limited number of cycles of simulation, and hardware accelerators are used to speed up the process. All of these types of verification with their concomitant

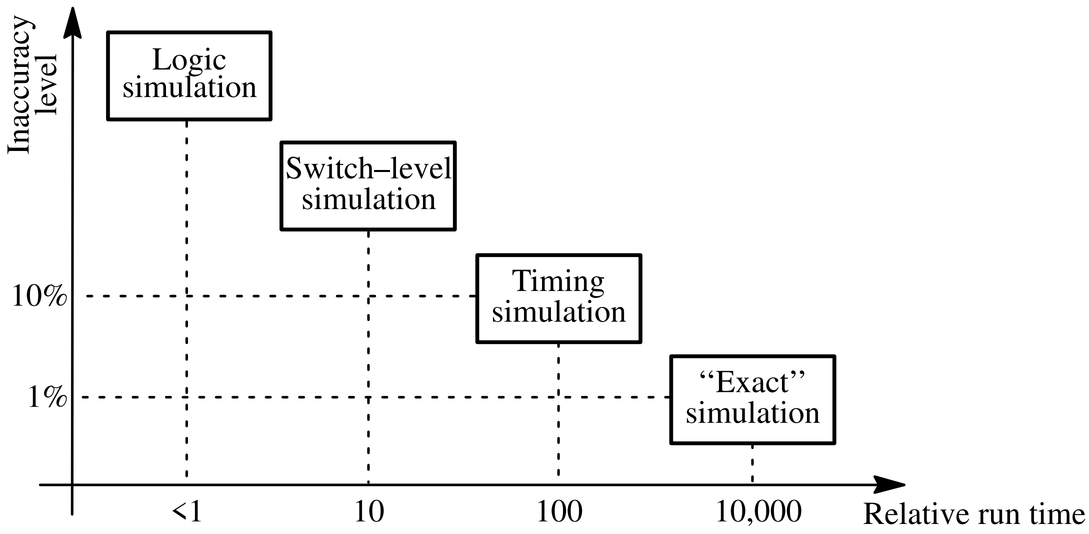

Figure 1: Qualitative depiction of the relative accuracy and efficiency of different types of circuit simulators. For switch-level and logic simulators, there is no notion of timing accuracy.

levels of model abstraction are necessary to prevent bugs in first-pass chip hardware.

Simulation typically requires specification of the input signals and initial conditions to be applied during the simulation. In the context of digital circuits, static timing analysis can be used to circumvent the dependence on being able to predict the input signals that cause worst-case timing behavior. Graph-tracing algorithms are employed to predict conservative bounds on the slowest (and fastest) delays through a circuit.

During the simulation of a circuit, sensitivity analysis can be used to efficiently determine the gradient of circuit response to design parameters such as element values or transistor sizes. These sensitivities can then be used to optimize circuits automatically or improve their yield. Thus a rich variety of techniques have evolved to simulate and optimize circuits at various levels of modeling abstraction.

1 SPICE

SPICE and SPICE-like simulators [1, 2] are the work horses of circuit design. They accept as input a description of the circuit by means of a netlist file, a set of device models which are equations representing the electrical behavior of electronic devices and a set of input signals. The simulator then conducts an analysis of the circuit by solving the circuit equations using numerical techniques. Although these methods can typically only handle a circuit that is a small fraction of an entire chip, they are useful for carefully checking, analyzing and optimizing crucial building blocks that may be replicated several times.

1.1 DC, AC and transient analyses

The simplest type of analysis in SPICE is a DC or operating-point analysis. In this analysis, energy-storage elements such as capacitors and inductors are not considered (or if they exist in the circuit, they are opened and shorted, respectively). Then a set of equations is formulated.

All circuits obey three sets of equations. Kirchhoff's Current Law (KCL) states that the sum of the currents at each node of a circuit is identically zero at any instant of time. In an n-node, b branch circuit, there are n KCL equations. Kirchhoff's Voltage Law (KVL) states that the voltage of each branch is the difference between the node voltages at its terminals, and there are b such equations. Finally, each electronic device must obey its Branch Constitutive Relation (BCR), which is the equation that governs its behavior, such as Ohm's Law for a resistor. Thus we have $(2b + n)$ equations in $(2b + n)$ unknowns, the unknowns being b branch voltages, b branch currents and n node voltages. Each equation involves a small subset of the unknowns, and likewise each unknown appears in just a few equations. Hence the equations are said to be sparse.

In Sparse Tableau Analysis (STA) [3], all $(2b + n)$ equations are formulated and solved simultaneously, while maximally exploiting the sparsity of the equations. In Modified Nodal Analysis (MNA) [4], loosely speaking, the BCR and KVL equations are substituted into the KCL equations, to formulate n equations in the n unknown node voltages, thus leading to a more compact set of equations. In Tree Link Analysis (TLA) or Hybrid Analysis [5], a spanning tree in the graph corresponding to the circuit is chosen. KCL and KVL are written in terms of the fundamental cutsets and fundamental loops (see Appendix A of [6]) of the graph, respectively. Then the circuit equations are formulated with the voltages of the tree branches and the currents of the remaining branches (cotree or link branches) as the basis variables. The term Hybrid Analysis reflects that some of the basis variables are currents and some voltages. Whichever method of equation formulation is used, a set of nonlinear algebraic equations is obtained.

Equations formulated by any of the above methods are solved to obtain the DC solution of the circuit. Newton's method is first used to linearize the system of equations and sparse LU factorization is employed to solve the resulting linear system of equations. To apply Newton's method, the Jacobian (matrix of partial derivatives) of the system matrix must be computed and LU factored at each iteration. The iterative method is repeated until convergence is obtained. DC analysis is at the heart of the various analysis modes offered by exact simulators.

In AC analysis, the circuit equations are formulated in the frequency domain, with each electronic device being represented by a linearized model of complex admittance about its operating point. The resulting equations are solved for various choices of frequency by the same means described above.

In transient analysis, the simulator must determine the behavior of the circuit in the time-domain. A transient analysis typically begins with a DC operating point analysis to determine the initial conditions. Energy storage elements like capacitors and inductors

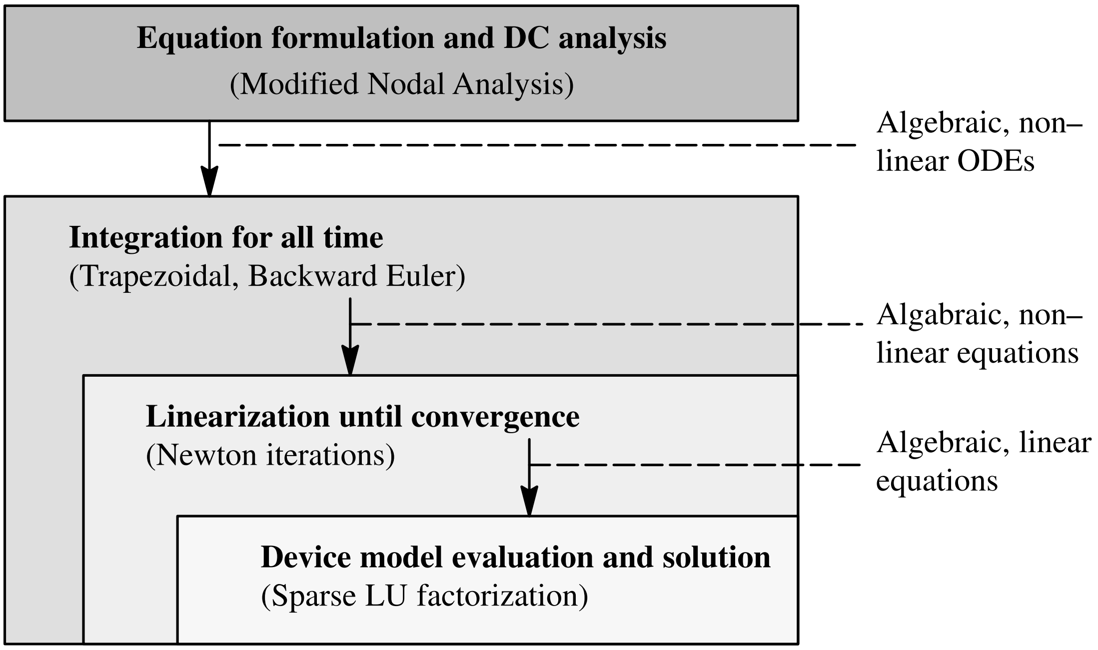

Figure 2: The steps involved in transient simulation.

contribute ordinary differential equations to the system of circuit equations. Thus the simulator must solve a set of nonlinear differential algebraic equations (DAEs). The first step is integration in the time-domain to convert the equations to a set of nonlinear algebraic equations. Integration is achieved by advancing time by a small interval called the time-step, and integrating the currents through capacitors and voltages across inductors, using one of several stable numerical integration algorithms. Circuits that exhibit a wide range of time constants, known as stiff circuits, typically require a large number of time steps. The Trapezoidal rule, Backward Euler and Gear's variable order integration method are popular algorithms for achieving this step. Integration is equivalent to replacing an energy storage element by a suitable "companion model" [6]. The resulting nonlinear algebraic equations are solved as in the case of a DC analysis. Then the "truncation error" which results from the Taylor series approximation inherent in the integration algorithm is estimated. If the error is larger than a pre-determined tolerance, the time-step is reduced in half and another attempt made. In this fashion, time is marched forward until the required simulation results are produced. Such algorithms are called incremental-in-time algorithms, as opposed to event-driven algorithms which will be discussed in a later section of this article. The steps involved in transient simulation are summarized in Figure 2.

Although exact circuit simulation has existed since the late-1960s [7], it is an ongoing topic of research, particularly as applied to analog and communications circuits. Research into speeding up the simulation [8], computing the DC operating point on tricky analog circuits [9], handling nonlinear circuits in the frequency domain [10], accommodating frequency-dependent elements in the time domain [11] and conducting mixed time-frequency simulations [12] are vigorously pursued research topics.

2 Timing simulation

Exact simulation methods have the advantage of being accurate and general, but are computationally burdensome. For even modest-sized circuits (say, 50,000 transistors) and modest simulation intervals (say $1\mu s$) the memory and CPU time requirements they place on computers are unacceptable. Timing simulators were invented in an effort to break this simulation bottleneck. In general, they sacrifice accuracy and generality for a gain of about two orders of magnitude in speed (see Figure 1). The relative timing accuracy of timing simulators is in the 10% range.

Typically applied only to transient simulation of digital FET circuits, timing simulators are based on the following concepts.

- Repeated evaluation of the nonlinear analytic device models in the inner loop of exact simulators is extremely expensive. Timing simulators seek to simplify the device models, thus sacrificing accuracy for speed.
- Exact simulators are incremental in time, thereby being forced to take small time steps if there is activity anywhere in the circuit. Most digital circuits have large subcircuits that are not active at any given time and large sub-intervals of time when the subcircuits are inactive. Timing simulators attempt to exploit this "latency" or "multi-rate nature" by employing event-driven algorithms. These algorithms incur computation only in those sub-intervals of time when the circuit has activity and in only those portions of the circuit that are active. (For a description of relaxation methods, which seek to exploit latency but maintain the accuracy of exact simulators, see "Waveform-Relaxation-Based Circuit Simulation" in this encyclopedia.)
- Simplified linearization or integration techniques are used by timing simulators to gain a speedup over exact simulators.
- Timing simulators have compact representations of the network and individual components in order to be able to store and simulate large circuits.
- These simulators typically partition the circuit into "channel-connected components" (also called "strongly-connected components" or "DC-connected components" in the literature) which are subcircuits consisting of transistors that are source-drain channel-connected, as shown in Figure 3. The boundary of each channel-connected component consists of either gates of transistors, primary inputs of the network, primary outputs of the network, power supply or ground.
- Several timing simulators offer variable accuracy. The user can loosen the accuracy requirements in return for faster execution times.

There is a wealth of literature on timing simulation and the following two paragraphs present a far-from-exhaustive sampling of the best-known simulators that have been developed. See Chapter 11 of [6] for a review of timing simulation. One of the first timing simulators was MOTIS [13] in which table models were used to store $I-V$ characteristics of transistors. The channel currents of transistors were quickly determined by table look-up. These currents

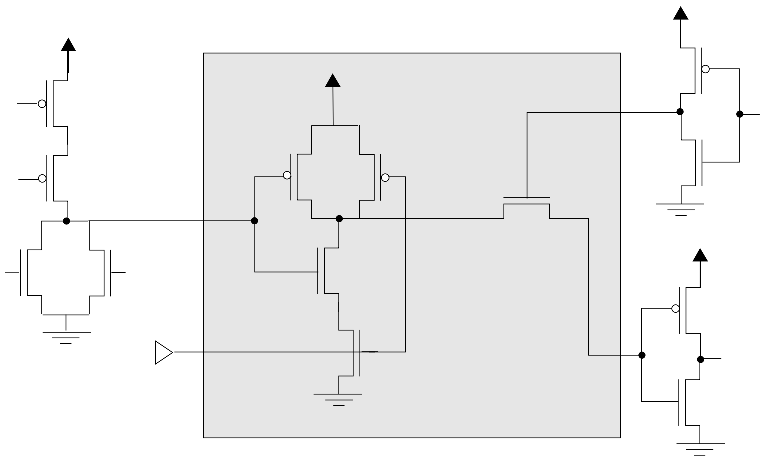

Figure 3: An example of a channel-connected component.

were combined to determine the charging current of the load capacitance of each channel-connected component. Using a secant approximation for the conductance of transistors, the change of voltage at the output of the channel-connected component was computed and propagated to its fanouts in an event-driven fashion. SAMSON [14] is a mixed circuit/logic simulator, which uses event-driven algorithms to exploit latency, but solves each channel-connected component more accurately.

In the late 80s and early 90s, a series of timing simulators were developed that approximated device and circuit quantities by piecewise approximate functions. For example, E-LOGIC [15] uses nodal analysis, but discretizes node voltages into several "states." The time at which each node reaches the next state in its discrete set of voltages is computed, and simulation is thus carried out in an event-driven fashion. To simplify device modeling, the MOS transistor is modeled as a current-limited switch for fast event-driven simulation in [16]. In SPECS [17], device $I-V$ characteristics are approximated by piecewise constant functions, and again the time needed for each device to reach the next segment boundary in its $I-V$ table is computed. TimeMill and PowerMill [18] are timing simulators employing piecewise approximate modeling and event-driven techniques. In ACES [19], piecewise linear functions are employed to model $I-V$ characteristics, modified nodal analysis equations are written for each channel-connected component and explicit integration with adaptive control of the time step are used to achieve efficient simulation.

These fast but approximate timing simulation algorithms have provided designers of digital ICs with additional simulation and power estimation methods that have rapidly become an integral part of the design methodology of modern memory, microprocessor and ASIC chips.

3 Switch-level and logic simulation

Although rich and important topics in their own right, switch-level and logic simulation are briefly addressed in this section for completeness. They apply almost exclusively to digital circuits. In switch-level simulation [20], each transistor is treated as a switch which is either on or off. Additionally, signals are allowed to have one of a discrete number of states, such as 1 (logic high), 0 (logic low) and X (unknown or uninitialized). For example, if the gate of an N-type transistor is high, it is on. From each node of each channel-connected component, all possible conducting paths are traced to the power supply and to ground. Based on heuristics, the "strength" of each path is computed. If a signal has conducting paths exclusively to the power supply or the combined strengths of the paths to the power supply are much larger in magnitude than the strengths of the paths to ground, then the node is assigned a logic high state. Similarly, if the paths to ground are stronger, it is assigned a logic low. If neither of the above is true, the node is assigned an X. Then signal values are propagated to the fanouts of the channel-connected component. Such a computation is repeated for each cycle of simulation after applying new values of the signals at the primary inputs.

Switch-level simulation is efficient because its MOSFET model is so simple. Hence large circuits can be simulated for many time cycles. However, switch-level simulators have some fundamental limitations linked to their simplistic modeling of transistors and time. First, they provide little or no timing information. Second, their handling of analog situations like charge-sharing, glitches or bidirectional signal flow is at best inaccurate.

Logic simulation [21] is one level higher in abstraction. In its simplest form, the circuit is modeled as an interconnection of primitive logic gates, and each gate has a pre-compiled logic behavior. In addition, each gate has a delay model that represents the delay from the arrival of each input to the availability of each output signal. Signal representation is much like switch-level simulators, but logic simulators often have special signal representations for high-impedance states, tristate signals and so on. The simulation algorithm consists of a simple event-driven or selective-trace mechanism. Primary inputs are first assigned their initial values. Gates with exclusively primary inputs are evaluated and their outputs scheduled for update at an appropriate future time. When the outputs are updated, the fanouts of each of these signals are then evaluated and their outputs scheduled for updating. A simple "time wheel" allows coordination of the queue of evaluate and update events. Thus events are repeatedly scheduled and evaluated until the circuit has been simulated for the required number of cycles. Logic simulation is the backbone of digital system verification and permits the verification of large and complex systems by running simulations for a large number of cycles. In fact, logic simulation is such an important step in system verification that various special-purpose hardware engines have been built to speed up the logic simulation process [22].

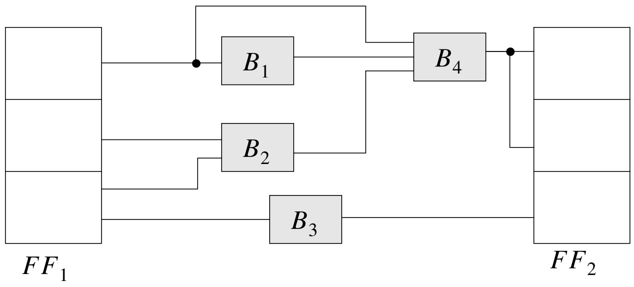

Figure 4: Illustration of static timing analysis.

4 Static timing analysis

SPICE and the timing simulators described above are all dynamic simulators. Input signals are specified in the time-domain, and circuit quantities are computed, starting from initial conditions, for a given interval of time. However, the simulation is only as good as the selection of input signals ("patterns" or "vectors" in digital circuit argot). Digital circuits have numerous paths through them and the simulation verifies the function and timing of only those paths that are sensitized by the input signals.

Often in digital circuits, it is required to compute an upper bound on the delay of all paths from the primary input to the outputs, irrespective of input signals. Such an upper bound is computed by means of static simulation, more commonly known as static timing analysis [23], which is used to characterize the delay of an interconnected set of combinational logic blocks between the flip-flops of a digital circuit. Figure 4 shows a simple circuit consisting of two banks of flip-flops (FF_1 and FF_2) and four combinational blocks (B_1 through B_4). In this example, static timing analysis seeks to predict the earliest time at which FF_2 can be clocked while ensuring that valid signals are being latched into the flip-flops.

Before embarking on static timing analysis, each combinational block's delay is pre-characterized. The delay from each input pin to each output pin is either described as an equation or stored in a look-up table. The delays are functions of such variables as input slope, fanout and output load capacitance. The pre-characterization phase consists of many circuit simulation runs at different temperatures, power levels, loading conditions, and so on. Delay data from these runs are abstracted into a timing model for each block.

The actual analysis is carried out in two phases. In the first phase, the delay of each signal is propagated forward through the combinational blocks, using the pre-characterized delay models. Thus each signal is labeled with a latest arrival time at which its correct digital value can be guaranteed. In the second phase, the required arrival time is propagated backwards from the target bank of flip-flops, FF_2. The required arrival time on a signal is the latest time by which that signal must have its correct value in order for the system to meet timing requirements. The difference between the required arrival time and the actual arrival time of each signal is termed the slack of the signal. After the analysis, all the signals are listed in increasing order of their slack. This analysis yields a wealth of timing information.

Clearly, if there is negative slack on any of the signals, the circuit will not meet the performance requirements. The path with the least (perhaps most negative) slack on all of its signals is the critical path. The nodes along this path will all have the same slack. The slacks also contain clues needed to redesign the circuit to cause it to function correctly. The above analysis can be carried out with a minimum and maximum delay for each block. In that case, a set of early and late arrival times can be computed for each signal. The early mode is computed using the best possible case for the arrival of all input signals to a block and the late mode considers the most pessimistic scenario. Then two sets of slacks are computed for each signal. These slacks yield valuable information about the timing properties of the circuit including possible violations of flip-flop setup or hold times, or possible "fast paths" that might cause spurious switching of the flip-flops.

Delay models are not always available, particularly for custom-designed circuitry. To overcome this problem, the circuit is partitioned into channel-connected components and each component is analyzed by means of dynamic simulation. The dynamic simulation is automatically configured and run under the covers to create a timing model for each channel-connected component on the fly. Depending on whether an early or late mode analysis is being performed, loading and sensitization conditions are chosen for the dynamic simulation to actuate the best-case or worst-case delay through the channel-connected component. Thus static timing analysis is capable of handling circuits without a dependency on pre-characterization or being restricted to cells from a standard library.

Static timing analysis is a highly efficient method of characterizing the timing of digital logic circuits. It can be used to determined the critical path of a circuit and obtain valuable timing information. However, it assumes that all paths in the circuit are active or sensitizable. In reality, however, there are certain paths in logic circuits that are not sensitizable because of the nature of the logic or the manner in which the circuit is exercised. These paths are called false paths. Because it ignores the false-path problem and because late mode analysis makes conservative choices, static timing analysis often predicts pessimistic worst-case delays.

4.1 Special considerations in interconnect analysis

As on-chip dimensions are scaled down, the fraction of total delay contributed by wiring or interconnect increases. Further, on-chip distribution of global signals like power supply, ground or clocks is a thorny problem. Hence interconnect analysis has received special attention in recent years. Frequency-domain methods have been increasingly popular and effective for the analysis of large, lumped, linear networks.

The basic idea is to create a reduced-order model that captures the salient behavior of the original interconnect up to a required frequency. In the earliest attempts [24], the impulse response of the original circuit in the frequency domain, written as a polynomial in the complex frequency s, was matched to a rational polynomial of order q, where q is much less than the order of the original circuit. By means of Padé approximation [25], the coefficients of the rational polynomial approximation (or equivalently, the poles and residues of the reduced-order model) can be determined. The reduced-order frequency-domain model can be used to predict time-domain responses by means of an inverse Laplace transform. The Padé approximation can be performed via a Lanczos process [26] to preserve numerical accuracy and stability. Several variants on this process have been described in the literature to preserve stability, accuracy and passivity of the reduced-order models (see for example [27]). Reduced-order models of large interconnect networks can be plugged into an exact or timing simulator and simulated along with nonlinear drivers and receivers more efficiently than if the entire network were simulated at once. The reduced-order models are also amenable to the sensitivity analysis methods [28] that are the topic of the subsequent section.

5 Sensitivity analysis

In the design of a circuit, let us say we are interested in some response v (perhaps a voltage) and that we pick a value for a design parameter p (perhaps a resistance value or transistor size). Then the sensitivity of the response v to the parameter p, $\frac{\partial v}{\partial p}$, tells us how much the response will change due to a small perturbation in p. Note that such a "small change" sensitivity or gradient is valid only in a small neighborhood around the nominal value of p.

The ability to compute sensitivities efficiently is tremendously useful in circuit design. Sensitivities can be used in tolerance analysis, circuit optimization, computation of periodic steady-state solutions, enhancement of manufacturability and so on. Note that obtaining approximations to the sensitivities by finite differences is too inefficient since it involves rerunning the simulator with small perturbations of the parameters one at a time. The effect of "large changes" in circuits can also be computed [29, 30, 6], but such methods are less efficient and therefore rarely used in practice.

There are two well known methods of computing gradients, the direct method [31] and the adjoint method [32]. The reader is referred to [6] for a tutorial description of the theory behind these two methods. In both methods, a new circuit is formulated whose solution yields the required sensitivities. The new circuit in both cases is topologically identical to the nominal circuit and LU factors computed during the nominal simulation can be re-used to efficiently solve the reformulated circuit. Hence these methods are collectively termed incremental sensitivity analysis.

The direct method is based on direct differentiation of the branch constitutive relations (BCRs) that govern the electrical behavior of the elements of the circuit. For example, consider a resistor governed by Ohm's Law

$$v = iR. \tag{1}$$

Assuming that the parameter of interest is R, we differentiate to obtain

$$\frac{\partial v}{\partial R} = i + R\frac{\partial i}{\partial R} \tag{2}$$

which can be rewritten as

$$\hat{v} = i + R\hat{i} \tag{3}$$

where $\hat{v}$ and $\hat{i}$ are the unknowns in our sensitivity analysis. Therefore, we replace a resistor in the nominal circuit by a resistor of equal value in the sensitivity circuit, but with a voltage source in series. The value of the voltage source is the current through the resistor in the nominal circuit, i, which is known once the nominal circuit has been solved. We thus replace elements in the nominal circuit by appropriate elements, and solve the resulting sensitivity circuit to determine all the $\hat{v}$ and $\hat{i}$ variables simultaneously.

The relations thus derived for each element represent a sensitivity circuit of the same topology as the original circuit but perhaps different circuit elements. The solution of this related circuit yields the sensitivity of all measurements with respect to a single parameter. Fortunately, the (linearized) system matrix of the original and sensitivity circuits are the same at each time instant, and hence the cost of LU factorization can be amortized during the analysis of the sensitivity circuit. In the resistor example above, the extra voltage source appears on the right-hand side of the circuit equations and hence does not change the system matrix of the nominal circuit. Note that the sensitivity circuit must be repeatedly solved as many times as the number of parameters, which is expensive for large numbers of parameters.

The adjoint method is the method of choice for computing gradients of large circuits. While the direct method involves direct differentiation of BCRs, the adjoint method involves differentiation of the matrix of circuit equations. In the circuit context, the adjoint method is best understood as an application of Tellegen's theorem [33]. As in the direct method, an associated circuit called the adjoint circuit is formed. The adjoint circuit has the same topology as the nominal circuit, but possibly different electrical elements. Like the direct method, the LU factors of the nominal circuit can be re-used during the adjoint analysis, modulo some time-point mismatch issues, as discussed below. Control is reversed and time run backwards during the adjoint analysis. Finally, the waveforms of the original and adjoint circuits are convolved to yield the required sensitivities.

The main advantage of the adjoint method is that it yields the gradients of one function with respect to all the parameters in a single adjoint analysis. However, because time is run backwards, the nominal and adjoint analyses cannot be carried out simultaneously. Further, it is not easy to make the time points of the two analyses coincide, leading to a clumsy time-point mismatch problem. The convolution of waveforms is an additional source of computational and memory overhead in the adjoint method.

The single function which forms the sensitivity function in the adjoint method can be any scalar differentiable function of any number of circuit measurements [34]. In particular, if the sensitivities are being computed for the purposes of being supplied to a nonlinear optimizer, then it is quite likely that the optimizer formulates an internal scalar merit function. In such a situation, the gradients of the entire merit function can be computed by means of a single adjoint analysis, irrespective of the number of measurements and the number of parameters!

With either method of computing gradients, chain ruling and combining of gradients is essential. For example, when the width of an MOS transistor varies, the associated intrinsic device capacitances as well as the diffusion capacitances on the source and drain vary. The sensitivity of each measurement with respect to these parasitics must be computed, and then chain ruled and combined to obtain the composite sensitivity with respect to all ramifications of the variation of the parameter of interest.

Incremental sensitivity computation in the case of DC and frequency-domain analyses is practical in the context of exact simulators. In the time-domain, the saving and interpolation of Jacobian matrices and the storage of waveforms for convolution in the adjoint method render incremental sensitivity less tractable. However, in the context of timing simulators, it has been shown that sensitivity computation of large circuits is practical and extremely efficient [35, 34, 36, 6]. Depending on the modeling simplifications used, the associated sensitivity or adjoint circuit can be trivial to solve. In the case of SPECS [17], the associated circuit consists of disconnected capacitors, with impulses of charge transferred between the capacitors at times corresponding to event times in the nominal transient solution. Further, the piecewise nature of the waveforms reduces the cost of otherwise costly convolutions. Fast sensitivity computation on circuits with several thousand transistor width parameters has been reported [37].

6 Circuit optimization

Automatic circuit optimization (or tuning) [38] is an important part of rapidly, repeatably and robustly designing high-performance circuits. The relentless push for ever higher performance, the need to design circuits of greater complexity, the emphasis on custom design and shrinking product cycles have led to an increased interest in optimization techniques. New challenges such as power minimization for portable applications, noise reduction and signal integrity also increase dependence on automatic design methods.

Given a functioning circuit schematic, the circuit tuning problem can be stated as that of optimally assigning values to components (e.g., transistor sizes, wire sizes, resistor values, compensating capacitor values). The performance metrics in digital circuits include (some subset of) delay, transition time, area, power dissipation, signal integrity, additional timing constraints, layout constraints and manufacturability. Most of these metrics are nonlinear functions of the tunable parameters. Each metric can be presented as either an objective function or a constraint. The parameters of the problem are typically transistor and wire sizes, and these parameters are required to lie within simple bounds. Many circuit tuning problems are best stated as minimax problems in which the optimizer is required to minimize the maximum of a finite set of functions. For example, the problem may be stated as minimizing the worst delay across several paths through the logic.

Circuit tuning is best approached by gradient-based nonlinear optimization. In the absence of gradients, large problems cannot be solved and one is typically limited to problems in a few tens of variables. Worse, there is often no guarantee of convergence or optimality in such “gradient-free” techniques. The efficient computation of gradients and even Hessians (matrices of second partial derivatives) is key to effective optimization of large circuits. Note that gradient-based nonlinear optimizers attempt to converge to a feasible and locally optimal point; there is no guarantee of global optimality.

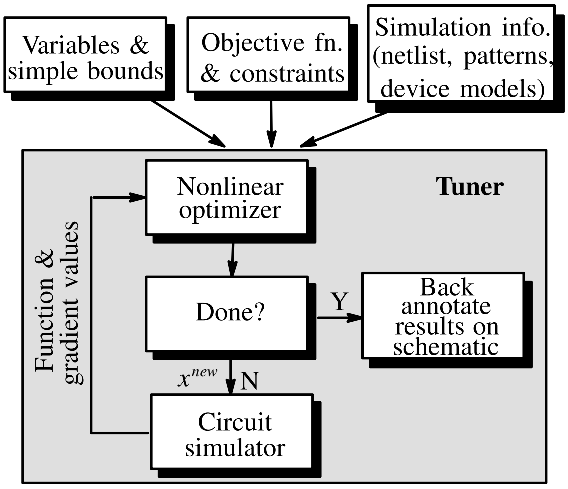

Figure 5: Typical flow of a dynamic tuner.

Circuit optimization techniques fall into three broad categories. The first is dynamic tuning, based on time-domain simulation of the underlying circuit, typically combined with adjoint sensitivity computation. These methods are accurate but require the specification of input signals, and are best applied to small data-flow circuits and "cross-sections" of larger circuits. Efficient sensitivity computation renders feasible the tuning of circuits with a few thousand transistors. Second, static tuners employ static timing analysis to evaluate the performance of the circuit. All paths through the logic are simultaneously tuned, and no input vectors are required. Large control macros are best tuned by these methods. However, in the context of deep submicron custom design, the inaccuracy of the delay models employed by these methods often limits their utility. Aggressive tuning can push a circuit into a precipitous corner of the manufacturing process space, which is a problem addressed by the third class of circuit optimization tools, statistical tuners. Statistical techniques are used to improve yield in the face of inevitable manufacturing variations.

6.1 Dynamic circuit optimization

Dynamic tuning [39, 40, 41, 37] implies circuit optimization based on dynamic time-domain simulation of the underlying circuit. The typical flow of a dynamic tuner is shown in Figure 5. Under the control of a nonlinear optimizer, tunable parameters are set to their initial values and a simulation is performed. The measurements of interest, and the sensitivities of each measurement with respect to all tunable parameters are fed back to the optimizer. Based on this information, the nonlinear optimization package suggests a new solution vector, which is a new assignment of parameter values that is expected to improve the circuit. The iterative process is carried to convergence, or until a user-specified maximum number of iterations is reached.

The parameters in dynamic tuning usually include transistor and wire sizes and they must conform to simple bounds. Ratio-ing of transistor widths to one another must be permitted. Further, grouping of similar structures is useful to ensure that they can share a common layout by maintaining the corresponding transistors of the structures at the same size during the tuning procedure. The measurements in dynamic tuning usually include area (often modeled by the sum of the tunable transistor widths), delay, noise and transition time or slew. The objective function and constraints are expressed in terms of these measurements. Minimax optimization is a useful feature whereby the worst of a set of measurements is minimized. For example, the problem may be stated as minimizing the worst delay of m paths through the circuit, as shown below.

$$\begin{array}{lll} \text{minimize} & \text{maximum} & d_i(x). \\ x \in \Re^n & i \in \{1, 2, \cdots, m\} & \end{array} \tag{4}$$

Note that the optimizer has no *a priori* knowledge of which of these paths exhibits the worst delay. Further, different paths may be critical during different iterations of the optimization.

The main advantage of dynamic tuning is its accuracy. The tuning is realistic since it is based on full-blown transient simulation. Likewise, false paths are avoided in contrast to static tuning methods. If the transistor sizing at any iteration causes failure of a measured signal to switch correctly, the transient simulation is able to detect this situation. In such a case, a non-working circuit has been obtained, usually because of the optimizer taking too aggressive a step. Recovery from this situation is implemented by requiring the optimizer to cut back on its step size and trying again.

However, dynamic tuning suffers from a number of disadvantages. The main disadvantage is that it is specific to the input pattern sensitizations and measurements specified. Unlike static tuning, it is not possible to tune any but the smallest circuit for all possible input patterns and all possible paths through the logic. As with the use of any optimizer, the solution obtained is only as good as the problem specification. Dynamic tuning is particularly vulnerable to designers omitting tacit requirements and then encountering unexpected results. A disciplined approach to accurately expressing all aspects of the problem at hand is essential to making good use of any optimization program!

Dynamic tuning is most often applied to small data-flow circuits in which the critical paths are well known and the input patterns to sensitize these paths are easy to come by. The relative computational inefficiency of these tools also limits the size of circuit that can be tuned. Most dynamic tuners are limited to a few tens of transistors. DELIGHT.SPICE [41] was one of the early practical implementations of a dynamic circuit optimization capability. Recently, a SPECS-based tuner called JiffyTune [34, 37] was reported to tune circuits with over 10,000 transistors. The large capacity was achieved by using a simulator that simplifies device models, employs event-driven simulation and applies the adjoint method to compute sensitivities.

6.2 Static circuit optimization

Static tuning implies circuit optimization based on static timing analysis [23]. One of the earliest static tuners was TILOS [42]. In these approaches [43], transistors are usually modeled by equivalent RC circuits. The actual values of the resistances and capacitances are computed during a pre-characterization procedure. The delay of each channel-connected set of transistors is computed using the Elmore delay model [44, 45], a special and simplified case of the reduced-order models discussed earlier. Alternatively, delay macromodels are used in [46]. Conventional static timing analysis is used to determine the critical path. The delay of the critical path can then be expressed as a function of the widths of transistors and wires. This expression is modeled by a posynomial function (a particular algebraic form, see [47]) of the parameters of the optimization. The observation is then made that by a simple variable substitution, the posynomial function can be converted to a convex function. Thus any local minimum is guaranteed to be a global minimum.

The procedure in TILOS is to start all transistors at their minimum widths, and iteratively bump up the width of the transistor to which the critical path is most sensitive at each step of the algorithm. The procedure is repeated until the lowest critical path delay through the circuit is found. More recently, power optimization has also been proposed in this general framework [48].

The main advantages of static timing analysis are its pattern independence and its speed. Very large circuits can be tuned relatively quickly. All paths are implicitly taken into account because of the underlying static timing basis. The designer is freed of the onus of coming up with input patterns or identifying critical paths. Since the timing of many industrial designs is verified by static timing analysis, there are obvious advantages to carrying out static tuning in that same general framework. Further, interconnect delay can easily be modeled and accommodated into this framework.

Unfortunately, static timing analysis has a number of drawbacks. The most serious one is accuracy. Elmore delays do not provide reasonable accuracy in the context of high-performance sub-micron circuits. Other modeling techniques like "collapsing" each logic gate into an equivalent inverter significantly degrade the accuracy. Unfortunately, the mathematical elegance of mapping the problem into a convex one and the intuitive satisfaction of finding a global minimum are rendered void by the crudeness of the delay approximation. The second major drawback of static tuning is the false-path problem. The optimizer may be hard at work tuning false paths through the circuit, and therefore unable to achieve any performance gains in the paths that really matter. But this problem is no more or less serious than the false-path problem in static timing analysis, and if the circuit "sign-off" is based on static timing analysis, this activity may be legitimate, though wasteful. The third problem with static tuning is the lack of delay models as functions of transistor sizes. Analytic delay models for gates in a library are generally not constructed as a function of transistor sizes, thus making them unusable during optimization. The creation of such models involves an exhaustive and time-consuming SPICE-based characterization process. Finally, starting with all transistors set to their minimum size could lead to circuits that may not even have the correct logical transitions. Dynamic tuners, since they are based on a realistic simulation of the circuit, have the advantage of being able to detect such "non-working" circuits and attempting to recover from them.

Static timing analysis in the context of custom circuits is successful only when each channel-connected component is timed using a dynamic simulator of reasonable accuracy under the covers. For tuning purposes, the fast gradient computation methods mentioned above can then be exploited.

6.3 Design for manufacturability

Yield loss on a fabrication line can be attributed to catastrophic and parametric (or circuit-limited) yield loss. Catastrophic yield loss is due, for example, to dust particles that cause opens or shorts on metal lines. Parametric yield loss, which is discussed in this section, occurs due to inherent manufacturing variations, leading to chips that do not have the required performance characteristics. In sorted designs like microprocessor chips, this degradation can mean that insufficient chips end up in the high-performance, high-profit bin. In non-sorted designs (for example, a bus controller chip), circuits below a performance threshold must be thrown away. Across the chip linewidth variations (ACLVs) constitute the single dominant set of parameters that lead to variations in the performance of the circuit. Statistical tuning is the process of changing design parameters to minimize circuit-limited yield loss.

Aggressive tuning of a circuit often drives it into a corner of the process space, thus causing its yield to suffer. This problem has been studied extensively in the literature and [49] is a good tutorial introduction to the subject. In addition, the books [50, 51] provide a survey of the state-of-the-art as well as extensive pointers to further reading, while [52] is a useful reference on the topic of creating and building statistical models.

The approaches to various aspects of statistical tuning are listed below.

- In *Monte Carlo* analysis, the parametric space is sampled and the design simulated at each sample point. Of course, this method assumes that distributions of the parameters are known. Further, by various principal component and correlation analyses, the number of independent parameters is reduced so as to limit the dimensionality of the space being sampled and therefore the number of simulations required. The results of the simulation runs can be used to determine both the distribution and worst-case behavior of the circuit.

  Designs are often simulated at multiple process corners, which is a simple form of Monte Carlo analysis. It is possible in the context of a dynamic tuner to replicate the nominal objective function(s) and constraints across all process corners, and simultaneously tune at all process corners [34]. Nominal objective functions are transformed into minimax functions across the process corners.

- *Extreme case analysis* is aimed at finding the worst-case behavior of the circuit given a statistical model of the parameter variations. The goal is not to predict the statistical distribution of the performance, but to predict the worst-case. A simple statement of the problem would be, for example, to maximize the delay of a circuit by optimally assigning transistor lengths that are constrained to conform to a pre-characterized distribution.

  A word about relying on extreme case analysis is appropriate here. It is obviously unwise to rely on "best case" assumptions during the design of a chip. Going too far the other way, while providing an easy guarantee of working hardware, is also wasteful. Unrealized performance can represent significant lost revenue. Worse, pessimistic projections may cause other parts of the system to be designed to work at lower specifications. Due to system limitations, it may not then be possible to harness the "surprisingly good" performance provided by chips that were designed using extreme case analysis. In practice, therefore, it is unwise to have a design process that is either overly pessimistic or overly optimistic.

- *Yield prediction and optimization* seek to explicitly model the yield characteristics of a circuit as a response surface. Once this is done, the parametric yield of a circuit in the face of manufacturing variations can be predicted. Further, based on the yield model, the circuit can be modified to maximize the yield.

- *Design centering* methods do not explicitly compute or model yields. Instead, they take the approach that pushing the circuit deeper into the interior of the feasible region in the space of parameter variations will result in a more robust circuit and therefore higher yields. While design centering methods operate on such a geometric model of the feasible region, method-of-moments-based techniques implicitly attempt to move designs away from regions of low yield to regions of high yield without seeking to explicitly compute the feasible region [53].

Despite much research on the topic of statistical tuning, Monte Carlo and extreme case analyses are the most popular approaches; industrial practice consists predominantly of these two methods. The advantages of these methods that are not shared by the other techniques are that they are easy to understand and in a form that is easily accessible to the design engineer.

7 Conclusions

Electrical and timing simulation are crucial components of circuit design. The challenge is to design tomorrow's faster and more complex systems with today's computers and computer-aids. As a result, a wealth of algorithms and techniques for simulation at various levels of abstraction have been developed over the last four decades. As the challenges grow, new approaches have been successful in tackling simulation and verification of larger and more complex integrated circuits. Simulation and optimization of circuits continues to be an active and vibrant research area.

References

[1] L. W. Nagel, "SPICE2, a computer program to simulate semiconductor circuits," Memo UCB/ERL M520, University of California, Berkeley, May 1975.

[2] W. T. Weeks, A. J. Jimenez, G. W. Mahoney, D. Mehta, H. Quassemzadeh, and T. R. Scott, "Algorithms for ASTAP - a network analysis program," *IEEE Transactions on Circuit Theory*, vol. CT, pp. 628–634, November 1973.

[3] G. D. Hachtel, R. K. Brayton, and F. G. Gustavson, "The sparse tableau approach to network analysis and design," *IEEE Transactions on Circuit Theory*, vol. CT, pp. 101–118, January 1971.

[4] C. W. Ho, A. E. Ruehli, and P. A. Brennan, "The modified nodal approach to network analysis," *IEEE Transactions on Circuits and Systems*, vol. CAS, pp. 504–509, June 1975.

[5] P. M. Russo and R. A. Rohrer, "The tree-link analysis approach to the transient analysis of a class of nonlinear networks," *IEEE Transactions on Circuit Theory*, vol. CT-18, pp. 400–403, May 1971.

[6] L. T. Pillage, R. A. Rohrer, and C. Visweswariah, *Electronic circuit and system simulation methods*. McGraw-Hill, 1995.

[7] L. W. Nagel and R. A. Rohrer, "Computer analysis of nonlinear circuits, excluding radiation (CANCER)," *IEEE Journal of Solid State Circuits*, pp. 166–182, August 1971.

[8] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli, "The waveform relaxation method for the time-domain analysis of large scale integrated circuits," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. CAD-1, pp. 131–145, July 1982.

[9] J. S. Roychowdhury and R. C. Melville, "Homotopy techniques for obtaining a DC solution of large-scale MOS circuits," *Proc. 1996 Design Automation Conference*, pp. 286–291, June 1996.

[10] P. Feldmann, R. C. Melville, and D. E. Long, "Efficient frequency-domain analysis of large nonlinear analog circuits," *Proc. Custom Integrated Circuits Conference*, pp. 461–464, May 1996.

[11] S. Kapur, D. E. Long, and J. Roychowdhury, "Efficient time-domain simulation of frequency-dependent elements," *IEEE International Conference on Computer-Aided Design*, pp. 569–573, November 1996.

[12] P. Feldmann and J. Roychowdhury, "Computation of circuit waveform envelopes using an efficient, matrix-decomposed harmonic balance algorithm," *IEEE International Conference on Computer-Aided Design*, pp. 295–300, November 1996.

[13] B. R. Chawla, H. K. Gummel, and P. Kozak, "MOTIS - An MOS timing simulator," *IEEE Transactions on Circuits and Systems*, vol. CAS-22, pp. 901–910, December 1975.

[14] K. A. Sakallah and S. W. Director, "SAMSON2: an event driven VLSI circuit simulator," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 4(4), pp. 668–684, October 1985.

[15] Y. H. Kim, S. H. Hwang, and A. R. Newton, "Electrical-logic simulation and its applications," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, pp. 8–22, January 1989.

[16] G. Ruan, J. Vlach, and J. A. Barby, "Logic simulation with current-limited switches," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, pp. 133–141, February 1990.

[17] C. Visweswariah and R. A. Rohrer, "Piecewise approximate circuit simulation," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 10, pp. 861–870, July 1991.

[18] C. X. Huang, B. Zhang, A. C. Deng, and B. Swirski, "The design and implementation of PowerMill," *Proc. International Workshop on Low Power Design*, pp. 105–110, April 1995.

[19] A. Devgan and R. A. Rohrer, "Adaptively controlled explicit simulation," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. CAD-13, pp. 746–762, June 1994.

[20] R. E. Bryant, "MOSSIM : a switch level simulator for MOS LSI," *Proc. 1981 Design Automation Conference*, June 1981.

[21] B. H. Scheff and S. P. Young, *Gate-level logic simulation*. Prentice-Hall, 1972.

[22] T. Blank, "A survey of hardware accelerators used in computer-aided design," *IEEE Design and Test of Computers*, pp. 21–39, August 1984.

[23] R. B. Hitchcock, Sr., G. L. Smith, and D. D. Cheng, "Timing analysis of computer hardware," *IBM Journal of Research and Development*, pp. 100–105, January 1982.

[24] L. T. Pillage and R. A. Rohrer, "Asymptotic waveform evaluation for timing analysis," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 9, pp. 352–366, April 1990.

[25] J. G. A. Baker, *Essentials of Padé approximants*. Academic Press, 1975.

[26] P. Feldmann and R. W. Freund, "Efficient linear circuit analysis by Padé approximation via the Lanczos process," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 14, pp. 639–649, May 1995.

[27] L. M. Silveira, M. Kamon, I. Elfadel, and J. White, "A coordinate-transformed Arnoldi algorithm for generating guaranteed stable reduced-order models of RLC circuits," *IEEE International Conference on Computer-Aided Design*, pp. 288–294, November 1996.

[28] R. W. Freund and P. Feldmann, "Efficient small-signal circuit analysis and sensitivity computations with the PVL algorithm," *IEEE International Conference on Computer-Aided Design*, pp. 404–411, November 1994.

[29] G. Kron, *Tensor analysis of networks.* John Wiley and Sons, 1939.

[30] A. S. Householder, "A survey of some closed methods of inverting matrices," *SIAM Journal of Applied Mathematics*, vol. 5, pp. 153–169, 1957.

[31] D. A. Hocevar, P. Yang, T. N. Trick, and B. D. Epler, "Transient sensitivity computation for MOSFET circuits," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. CAD-4, pp. 609–620, October 1985.

[32] S. W. Director and R. A. Rohrer, "The generalized adjoint network and network sensitivities," *IEEE Transactions on Circuit Theory*, vol. CT-16, pp. 318–323, August 1969.

[33] B. D. H. Tellegen, "A general network theorem, with applications," *Philips Research Reports*, vol. 7, pp. 259–269, 1952.

[34] A. R. Conn, R. A. Haring, C. Visweswariah, and C. W. Wu, "Circuit optimization via adjoint Lagrangians," *IEEE International Conference on Computer-Aided Design*, pp. 281–288, November 1997.

[35] P. Feldmann, T. V. Nguyen, S. W. Director, and R. A. Rohrer, "Sensitivity computation in piecewise approximate circuit simulation," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 10, pp. 171–183, February 1991.

[36] T. V. Nguyen, A. Devgan, and O. J. Nastov, "Adjoint transient sensitivity computation in piecewise linear simulation," *Proc. 1998 Design Automation Conference*, June 1998.

[37] A. R. Conn, P. K. Coulman, R. A. Haring, G. L. Morrill, C. Visweswariah, and C. W. Wu, "Jiffytune: circuit optimization using time-domain sensitivities," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, 1998. To appear.

[38] C. Visweswariah, "Optimization techniques for high-performance digital circuits," *IEEE International Conference on Computer-Aided Design*, pp. 198–205, November 1997.

[39] R. K. Brayton and R. Spence, *Sensitivity and optimization*, vol. 2 of *CAD of Electronic Circuits.* Amsterdam, The Netherlands: Elsevier Scientific Publishing Co., 1980.

[40] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "A survey of optimization techniques for integrated-circuit design," *Proceedings of the IEEE*, vol. 69, pp. 1334–1362, October 1981.

[41] W. Nye, D. C. Riley, A. Sangiovanni-Vincentelli, and A. L. Tits, "DELIGHT.SPICE: An optimization-based system for the design of integrated circuits," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. CAD-7, pp. 501–519, April 1988.

[42] J. P. Fishburn and A. E. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," *IEEE International Conference on Computer-Aided Design*, pp. 326–328, November 1985.

[43] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya, and S. M. Kang, "An exact solution to the transistor sizing problem for cmos circuits using convex optimization," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. CAD-12, pp. 1621–1634, November 1993.

[44] W. C. Elmore, "The transient analysis of damped linear networks with particular regard to wideband amplifiers," *Journal of Applied Physics*, vol. 19, no. 1, pp. 55–63, 1948.

[45] P. Penfield and J. Rubinstein, "Signal delay in RC tree networks," in *Proceedings of the 2nd Caltech VLSI Conference*, pp. 269–283, March 1981.

[46] M. D. Matson and L. A. Glasser, "Macromodeling and optimization of digital MOS VLSI circuits," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. CAD-5, pp. 659–678, October 1986.

[47] R. J. Duffin, E. L. Peterson, and C. Zener, *Geometric programming - theory and applications.* New York: John Wiley and Sons, 1967.

[48] P. K. Sancheti and S. S. Sapatnekar, "Optimal design of macrocells for low power and high speed," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. CAD-15, pp. 1160–1166, September 1996.

[49] R. Spence and R. S. Soin, *Tolerance design of electronic circuits.* Addison-Wesley Publishing Company, 1988.

[50] S. W. Director and W. Maly, eds., *Statistical approach to VLSI*, vol. 8 of *Advances in CAD for VLSI.* North-Holland, 1994.

[51] J. C. Zhang and M. A. Styblinski, *Yield variablity optimization of integrated circuits*. Kluwer Academic Publishers, 1995.

[52] C. Michael and M. Ismail, *Statistical modeling for computer-aided design of MOS VLSI circuits*. Kluwer Academic Publishers, 1993.

[53] G. Kjellstrom and L. Taxen, "Stochastic optimization in system design," *IEEE Transactions on Circuits and Systems*, July 1981.