Gradient-Based Optimization of Custom Circuits Using a Static-Timing Formulation

A. R. Conn* I. M. Elfadel* W. W. Molzen, Jr.* P. R. O'Brien†
P. N. Strenski* C. Visweswariah* C. B. Whan*

*IBM Thomas J. Watson Research Center
Route 134 and Taconic
Yorktown Heights, NY 10598

†IBM Electronic Design Automation
11400 Burnet Road, M. S. 9460
Austin, TX 78758

Abstract

This paper describes a method of optimally sizing digital circuits on a static-timing basis. All paths through the logic are considered simultaneously and no input patterns need be specified by the user. The method is unique in that it is based on gradient-based, nonlinear optimization and can accommodate transistor-level schematics without the need for pre-characterization. It employs efficient time-domain simulation and gradient computation for each channel-connected component. A large-scale, general-purpose, nonlinear optimization package is used to solve the tuning problem. A prototype tuner has been developed that accommodates combinational circuits consisting of parameterized library cells. Numerical results are presented.

1 Introduction and previous work

Circuit optimization by transistor and wire sizing is an important part of designing custom high-performance digital circuitry. There are two well-known methods of circuit optimization, *dynamic tuning* and *static tuning.* In the former method [1, 2, 3], the user must specify input patterns for simulation and only those measurements that are actuated during the simulation can be optimized. Hence there is a heavy burden on the user to correctly pose the tuning problem. In static tuning, the optimization is on a static-timing basis wherein all paths through the digital logic are considered simultaneously. The requirement of providing input patterns and the burden of stating a meaningful optimization problem are removed from the designer.

One of the best-known static tuners is TILOS [4], in which transistors are modeled by RC equivalent circuits, and the delay of a gate is represented by an Elmore [5] or Penfield-Rubinstein [6] delay model. The resulting sizing problem is shown to be posynomial in transistor and wire widths, and is converted to a convex problem by a simple mapping of variables. A heuristic method of solving this convex problem is employed to obtain the entire delay-area tradeoff curve. Subsequently, an exact solution to the convex problem was proposed in [7]. These methods work quickly and can handle large circuits. However, they suffer from the inaccuracy of approximating a logic gate by an RC circuit and the concomitantly crude delay model, making them unsuitable for custom, high-performance design.

Simulation-based static timing analysis, wherein each subcircuit is analyzed by time-domain simulation, is an ideal framework for path-independent optimization. In such a framework, custom circuitry can be accommodated, and all the benefits of static timing analysis are preserved. This paper presents an optimization technique that

- formulates the static tuning problem in a unique manner,
- is based on nonlinear optimization,
- uses fast transient simulation to evaluate custom circuitry, and
- uses incremental time-domain gradient computation by the adjoint method.

This paper focuses on circuits composed of parameterized library cells. Both datapath and control circuits, whether synthesized or custom-designed, can be optimized. Employing simulation to evaluate circuitry allows extension to general custom circuitry at the transistor-level. The simulation is combined with efficient, incremental, time-domain sensitivity computation in order to provide gradients to the nonlinear optimizer. It is well known that solving large, nonlinear optimization problems is impossible without gradient information or at least some method of approximating gradients.

Section 2 demonstrates the novel formulation by means of a simple example. Details are provided in Section 3. A prototype program called EinsTuner which tunes combinational circuits consisting of parameterized library cells is described in Section 4, and numerical results are discussed in Section 5. Section 6 is devoted to special considerations for custom transistor-level and sequential circuits. Finally, future work and conclusions are presented.

2 Problem formulation by example

Consider the network of Figure 1, consisting of three simple gates G_1, G_2 and G_3. Let $\mathbf{w}$ be the n-vector

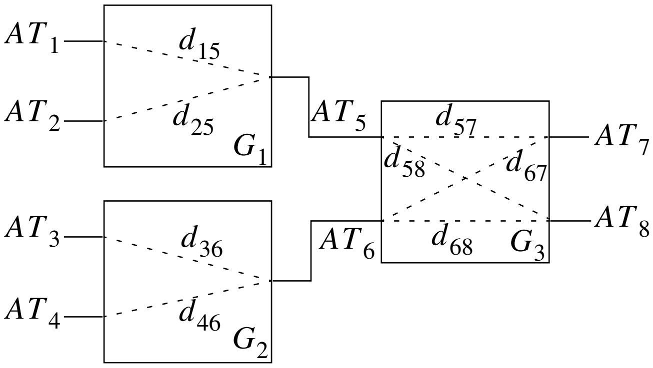

Figure 1: Simple example to illustrate the problem formulation.

of transistor widths to be optimized. Wires are ignored here for simplicity. Assume that we wish to minimize the worst arrival time at the primary outputs subject to an area constraint and simple bounds. The problem can then be stated as follows.

$$
\begin{array}{rl}
\text{minimize} & \max(AT_7, AT_8) \\
\text{s.t.} & AT_7 = \max[AT_5 + d_{57}(\mathbf{w}), AT_6 + d_{67}(\mathbf{w})] \\
\text{s.t.} & AT_8 = \max[AT_5 + d_{58}(\mathbf{w}), AT_6 + d_{68}(\mathbf{w})] \\
\text{s.t.} & AT_5 = \max[AT_1 + d_{15}(\mathbf{w}), AT_2 + d_{25}(\mathbf{w})] \\
\text{s.t.} & AT_6 = \max[AT_3 + d_{36}(\mathbf{w}), AT_4 + d_{46}(\mathbf{w})] \\
\text{s.t.} & \sum_i k_i w_i \leq A \\
\text{s.t.} & L_i \leq w_i \leq U_i.
\end{array}
\tag{1}
$$

In problem (1), the AT variables are the worst-case arrival times at each node of the circuit, the d_{ij} functions are the delays of *propagate segments* from each input pin to each output pin of each gate, A is an area target, and L_i and U_i are the lower and upper bounds on transistor widths (known as simple bounds). Area is modeled by a weighted sum of tunable transistor widths. Unfortunately, problem (1) is non-smooth due to the use of the max function and cannot directly be fed to any standard general nonlinear optimizer. Hence we remap the problem as shown below.

$$
\begin{array}{rlll}
\text{minimize} & z & & \\
\text{s.t.} & z & \geq & AT_7 \\
\text{s.t.} & z & \geq & AT_8 \\
\text{s.t.} & AT_7 & \geq & AT_5 + d_{57}(\mathbf{w}) \\
\text{s.t.} & AT_7 & \geq & AT_6 + d_{67}(\mathbf{w}) \\
\text{s.t.} & AT_8 & \geq & AT_5 + d_{58}(\mathbf{w}) \\
\text{s.t.} & AT_8 & \geq & AT_6 + d_{68}(\mathbf{w}) \\
\text{s.t.} & AT_5 & \geq & AT_1 + d_{15}(\mathbf{w}) \\
\text{s.t.} & AT_5 & \geq & AT_2 + d_{25}(\mathbf{w}) \\
\text{s.t.} & AT_6 & \geq & AT_3 + d_{36}(\mathbf{w}) \\
\text{s.t.} & AT_6 & \geq & AT_4 + d_{46}(\mathbf{w}) \\
\text{s.t.} & \sum_i k_i w_i & \leq & A \\
\text{s.t.} & L_i & \leq & w_i \;\leq\; U_i.
\end{array}
\tag{2}
$$

In reference to problem (2), the following points are to be noted.

- z is an auxiliary variable introduced to solve the problem. The arrival times at non-primary-inputs are variables of the optimization problem, just like the transistor widths. Input arrival times are provided in the form of timing assertions. So the optimization is carried out in the space $\mathbf{w} \in \Re^n; z \in \Re; AT_i, i = 5, 6, 7, 8 \in \Re^4$.
- (2) expresses a standard smooth nonlinear optimization problem without discontinuities in the first derivatives (provided the d_{ij} functions are continuously differentiable in $\mathbf{w}$).
- At the solution, by definition z is at its minimum value, while all the constraints are feasible, therefore indicating that we have solved the intent of the original problem. A gradient-based nonlinear optimizer will, of course, find a local minimum. There is extensive literature beginning with TILOS [4] that establishes the convexity of the static tuning problem under simplified modeling assumptions. On convex problems, converging to a local minimum guarantees global optimality. To the extent our more accurate transient-simulation-based modeling possibly destroys convexity, we seek to improve the circuit from its start point by moving it at least to a local minimum. One may also apply downhill optimization starting from the results of convex optimization using approximate delay models.
- At the solution, the constraints along all critical paths will be tight, and the arrival times along these paths will be correct. A timing run may be performed after the tuning to correctly determine the arrival times on non-critical paths.
- The formulation of the timing constraints is very similar to RITUAL [8], a wirelength minimization program.
- For the illustration above, details such as slews, slew-dependencies and separate rising and falling arrival times have been omitted for simplicity. Our formulation and software, however, take them into account as described in Section 3.
- The problem can equally be formulated as a minimization of area subject to a timing requirement, or the minimization of a weighted sum of critical path delay (or negative slack) and area.
- Not every delay function d_{ij} depends on every transistor width, a fact that is exploited in the next section and is significant for the optimizer.
- Simultaneous wire and transistor optimization fits into this general framework.

3 Problem formulation in detail

While the previous section described the concept in relation to a simplified situation, this section will lay out the problem formulation in detail.

3.1 Variables

Each node of the circuit has four variables associated with it. For node i, the four variables are the rising arrival time AT_i^r, the falling arrival time AT_i^f, the rising slew S_i^r and the falling slew S_i^f. Slew is considered to be

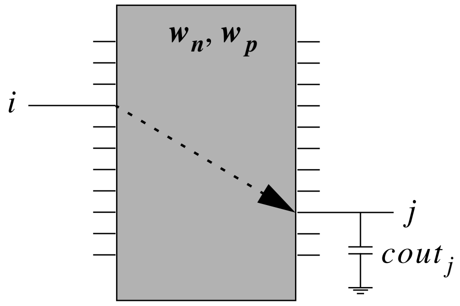

Figure 2: Generic channel-connected component.

a 0%-100% measurement (100%-0% for falling signals) of an idealized ramp waveform, but the definition can easily be changed to suit other timing methodologies. Next, we have the auxiliary variable z. Assume that we are dealing with parameterized gates in which each channel-connected component (CCC) has two design variables, w_n and w_p, denoting the width of all the NFETs and all the PFETs in the CCC, respectively. Again, the formulation can easily be generalized to a full-custom situation.

3.2 Timing constraints

Figure 2 shows a generic multi-input multi-output CCC. For the propagate segment from pin i to pin j, the constraints are shown below, assuming that the segment is an inverting segment. Depending on the type of gate, each propagate segment is classified as either an inverting segment, a non-inverting segment or both, and the constraints are listed appropriately. The entire network is traversed and constraints such as the ones listed below are gathered for every propagate segment.

$$\begin{aligned} AT_j^r &\geq AT_i^f + d_{ij}^r(w_n, w_p, cout_j, S_i^f) \\ AT_j^f &\geq AT_i^r + d_{ij}^f(w_n, w_p, cout_j, S_i^r) \\ S_j^r &\geq s_{ij}^r(w_n, w_p, cout_j, S_i^f) \\ S_j^f &\geq s_{ij}^f(w_n, w_p, cout_j, S_i^r). \end{aligned} \tag{3}$$

Note that a superscript of r implies a rising signal, delay or slew, while a superscript of f refers to the corresponding falling quantities. The nonlinear delay functions are denoted by d_{ij}, and the s_{ij} terms represent the nonlinear slew functions. The fanout capacitance at pin j is $cout_j$, which is a function of the sizes of the fanouts of pin j augmented by any wire capacitances on that net. As in most static timers, fanout capacitance is approximated by a lumped, linear capacitance which is a function of the transistor and wire sizes of the immediate fanouts. The gate itself is modeled at the transistor level.

A subtle point to note is that the slew constraints expressed above are conservative in nature, since the slew that is propagated to the next stage is the worst of all computed pin-to-pin slews. The propagate segment causing the worst slew may or may not coincide with that causing the worst arrival time. Many static timers pick the slew of the propagate segment causing the worst arrival time as the slew that is chosen for propagation to the next stage of logic. However, since the latter assumption can be optimistic in some situations (which goes against the grain of a late-mode static timer) and because it causes discontinuities in the formulation, we choose the more conservative approach shown in equation (3).

3.3 Additional constraints and the objective function

There are a number of additional constraints required for successful optimization.

- If we are interested in minimizing critical delay, for each primary output j, two additional constraints
$$\begin{aligned} z &\geq AT_j^f - RAT_j^f \\ z &\geq AT_j^r - RAT_j^r \end{aligned} \tag{4}$$
are required, where RAT indicates the required arrival time of rising and falling signals at the primary outputs. Of course, if all the required arrival times are uniform, then those terms can be dropped from the constraints in equation (4) and z interpreted accordingly, without any change in the resulting solution. In addition, an objective function
$$\text{minimize } z \tag{5}$$
is included. If we are not interested in minimizing critical delay, but rather minimizing area subject to system timing requirements, then for each primary output, two additional constraints
$$\begin{aligned} AT_j^f &\leq RAT_j^f \\ AT_j^r &\leq RAT_j^r \end{aligned} \tag{6}$$
are required, and the z variable is unnecessary.
- Area is typically expressed as the weighted sum of some or all of the tunable transistor and wire sizes. Area can either be constrained or minimized. Keeping the area in check generally (but not always) keeps the power consumption at reasonable levels.
- Arrival times and slews on primary inputs are set equal to the assertions provided by the designer.
- Required slews are expressed as simple inequality constraints involving a single variable.
- Input loading constraints are expressed as some (usually simple) function of the fanout widths of the primary input being less than a required maximum loading.
- The objective function can consist of just area, or just the quantity z or some weighted combination thereof. The weighting can be varied and the problem re-run to determine a tradeoff curve.
- Additional constraints for dynamic logic and sequential circuits are discussed in Section 6.

4 Implementation details

4.1 Overview

The circuit optimization formulation of the previous sections was implemented in a prototype tool called EinsTuner. The software architecture of the program is

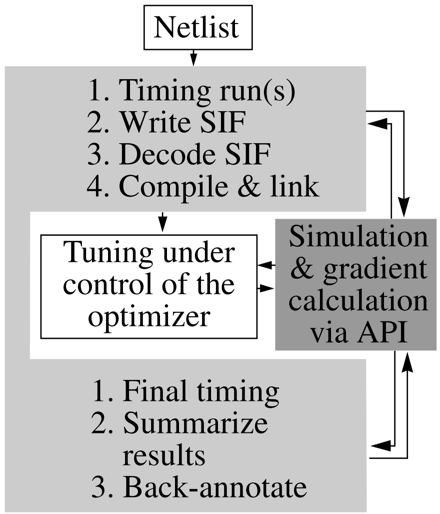

Figure 3: Software architecture of EinsTuner.

shown in Figure 3. The netlist is fed to the "outer layer" (lightly shaded box of Figure 3) of the EinsTuner program which performs an initial timing run. Then the tuning problem is expressed in the SIF (Standard Input Format) nonlinear optimization language [9]. The resulting SIF file is decoded to produce several problem-specific FORTRAN files. These files are compiled and the resulting objects linked with the optimizer objects and the "back-end" of EinsTuner to create a custom executable. The back-end consists of routines to evaluate nonlinear functions such as delays and slews and the gradients thereof and supply them to the optimizer on demand. The evaluation is carried out by invoking a circuit simulator through an application programming interface (API).

The custom executable is then invoked to carry out the actual inner optimization. Upon completion, transistor sizes are snapped to a technology-imposed grid, a final timing run is performed, and the required files for back-annotation of the new sizes are generated by the outer layer.

4.2 Problem formulation

Several aspects of the problem formulation are noteworthy. While any reasonable start point and any reasonable simple bounds should theoretically converge to the same tuned circuit, we should keep in perspective that we are asking the optimizer to solve problems in 1,000- or even 10,000-dimensional space in a few hundred iterations. The situation is further complicated by the presence of numerical noise in the computed delays and slews. In fact, any simulation-based data is inherently noisy. Hence many choices were made in the problem formulation to make the situation conducive to the optimizer being as aggressive as possible, and converging in as few iterations as possible.

- Units, scale factors and weight factors were chosen with great care so that the resulting problem was well-scaled.
- A concerted effort was made to keep the optimizer within a "physical range" of variables. Slews at non-primary-inputs were constrained to be within technology-specific lower and upper slew bounds.
- To increase optimization efficiency, a special method was used to determine the bounds and start points of arrival times and slews. A quick "mock timing" run is conducted before formulating the problem to determine a lower bound on all arrival times. Weighted combinations of the actual and lower bound arrival times are used as initial values, thus starting the optimization in an infeasible state. This choice was found to produce swift and aggressive optimization.
- Fanout capacitances were treated as "internal variables" [9] to reduce the dimensionality of the problem.
- Consider an optimization problem

$$\begin{array}{ll} \min\limits_{x,z} & z \\ \text{s.t.} & z \geq c_i(x), i = 1, 2, \ldots, n. \end{array} \tag{7}$$

  We know from the Kuhn-Tucker optimality conditions that the Lagrange multipliers corresponding to the constraints must sum up to -1 at the solution. Hence we initialized the Lagrange multipliers to $-1/n$, where n is twice the number of primary outputs of the network in our case.
- Where possible, inequalities were converted to equalities. For example, the arrival time and slew constraints for any 1-input gate (such as an inverter) can be written with equality constraints.

4.3 Simulation and gradient computation

EinsTuner is based on simulation and gradient computation of each CCC by the fast event-driven simulator SPECS [10, 11]. Whenever a transistor size or fanout capacitance of a CCC is updated, the optimizer automatically calls SPECS to re-compute the delays, slews, and gradients thereof. SPECS uses table models for device i-v characteristics. Specialized integration techniques, event-driven simulation and simplified device models enable SPECS to be 70x faster than <tool name>, an IBM-internal SPICE-like simulator at a relative stage timing accuracy of 5%. Path delays, however are predicted more accurately.

The key feature of SPECS exploited in EinsTuner is a mature gradient computation capability that has been extensively used in a dynamic tuner [12, 2, 3]. SPECS computes incremental time-domain sensitivity information by both the adjoint and direct methods. The adjoint method is used in EinsTuner. The sensitivity of a time-domain measurement (such as delay, slew, power or noise) can be computed with respect to any number of parameters (such as transistor widths, input slews or fanout capacitances) in a single adjoint analysis. Each adjoint analysis is a small incremental overhead on the nominal simulation. Gradients with respect to transistor widths include chain ruling and combining gradients with respect to diffusion capacitances whose values depend on those widths. In tuning the benchmark circuit c2670, for example, an estimated 15 million time-domain gradients were computed. Without a fast, accurate and reliable time-domain sensitivity engine, it would not have been possible to create a tool such as EinsTuner.

SPECS's "group adjoint" gradient capabilities [3] are utilized. For example, each slew measurement is coded as a single "group" expressed as a linear combination of individual measurements, such as 10% and 90% crossing times. Thus the sensitivity of slew with respect to all sensitivity parameters is computed by one adjoint analysis instead of two.

For each channel-connected component, the circuit is "constructed" by means of calls to a simulation application programming interface (API). The simulation conditions for all the propagate segments are concatenated in time and a single simulation of the CCC is performed to compute all the delays and slews. Simultaneously, the gradients of the delays and slews are computed with respect to each transistor size, input slew and fanout capacitance. All these gradients are cached in local arrays until requested by the optimizer. Much care was expended in the memory management of the API. Finally, various measures were taken to minimize the noisiness of the data provided by the simulator. Reduced noisiness in the data was crucial to achieving convergence on some of the larger benchmark circuits.

4.4 Nonlinear optimization

We use the large-scale, general-purpose nonlinear optimization package LANCELOT [9, 13, 14] with several special modifications for EinsTuner. LANCELOT uses an augmented Lagrangian merit function and employs a trust-region based algorithm. The merit function consists of a Lagrangian and a penalty term consisting of a weighted sum-of-squares of the constraints. Simple bounds are accommodated easily and efficiently by means of projections. The optimizer can be configured to use different preconditioners, to solve the inner bounded quadratic problem approximately or accurately, and so on.

LANCELOT allows one to exploit *group partial separability* in the problem structure. For example, in the EinsTuner problem formulation, the nonlinear contribution to each delay or slew constraint depends on only a few variables. This sparsity is communicated to LANCELOT via the SIF file and exploited by the optimizer, a key enabler of being able to solve large problems in relatively few iterations.

Since simulation and gradient computation are expensive compared to the optimization algorithms, it is worth going to great lengths in the optimizer to try to reduce the number of iterations required to achieve convergence. This principle was applied in several ways to speed up the optimization. All the slack variables internally introduced by LANCELOT to convert inequalities to equality constraints and the z variable occur exclusively linearly in the objective function and constraints. Thus they occur at most quadratically in the merit function, since the penalty term in LANCELOT's merit function squares the constraints. After each regular step of LANCELOT, a *second step* [15] can be computed analytically that updates these variables so as to further minimize the merit function. This two-step updating leads to fewer iterations.

Several steps were taken to encourage the optimizer to be aggressive, such as forcing a large initial trust-region radius and revising the criteria for trust-region management. As a result, the optimizer often takes large steps that cause the circuit to "fail," meaning that one of the measured signals at the output of a CCC fails to switch in a reasonable time. In such a situation, the simulator sends a special return code to the optimizer. The optimizer skips the rest of the iteration, reduces the trust-region radius and tries again. In our opinion, failure recovery is a necessary ingredient of efficient circuit tuning.

Prior to the simulation-based version of EinsTuner, a version that models delays and slews of gates by means of analytic equations was developed. In this environment, exact gradients can be provided to the optimizer and the data is not noisy. This software prototype was shown to consistently converge to within arbitrarily small gradient and constraint tolerances with default initializations and stopping criteria, thus validating the formulation of the problem. Therefore, if accurate and convex analytic delay models are available, the EinsTuner formulation can easily obtain the global optimum. Further, the success of this prototype bodes well for being able in the future to mix transistor-level modeling with analytic delay rules.

Simulation data is inherently noisy. In analytic problems, if the optimizer takes a small step, one expects a good match between the optimizer's model of the n-dimensional space and reality. However, this safety net does not exist in the case of simulation-based data. Several optimization choices were made to deal with noise. In particular, a special stopping criterion was developed to detect that no further significant improvement is readily available because we have a step size at which the change in the data is dominated by noise.

5 Numerical results

EinsTuner was tested on a number of combinational benchmark circuits, including two actual circuit designs from a high-performance microprocessor. All tests were conducted on an IBM PowerPC 43P 200 MHz workstation except where specified.

Other than the two actual designs (`iqia` and `incrementer`) and two artificially generated problems (`inv3` and `a3_3`) the testing procedure was as follows.

1. The design was synthesized from the ISCAS-85 suite of combinational benchmarks into an implementation consisting of restricted library cells using an internal logic synthesis tool.

Table 1: Sizes of benchmark problems.

Name	Problem size					Area
	# Gates	# Transistors	# Nodes	# Variables	# Constraints	μm
inv3	3	6	4	23	15	69.7
c17	7	28	12	63	61	345.6
a3_3	9	34	12	67	75	549.9
full_adder	14	46	17	97	97	307.8
iqia	22	72	30	165	147	762.2
c8	180	584	208	1,193	1,203	3,187
c432	299	960	335	1,939	1,929	2,477
incrementer	408	1,554	472	2,705	3,237	8,777
c880	481	1,582	541	3,127	3,213	5,914
c1355	665	2,180	706	4,155	4,425	9,130
c499	667	2,216	708	4,167	4,497	9,240
c2670	837	2,796	992	5,643	5,665	9,358

2. Each gate was treated as a parameterized cell with one variable controlling the width of the NFETs and one variable controlling the width of the PFETs. In some gates (such as OAI21) not all NFETs or PFETs were identical. Nonetheless, all NFETs and PFETs were each ratio-ed to a single parameter.
3. The schematic was sized by employing a simple gain-based heuristic that involves traversing the graph from the primary outputs to the primary inputs. A gain factor of 4.0 was used to convert the fanout capacitance seen at each node of the network into a fanin capacitance. The fanin capacitance was then converted into total equivalent fanin gate width by means of a technology factor. A β (PFET to NFET width parameter) ratio specific to the type of gate was used to apportion the equivalent transistor width between the NFETs and PFETs. The area of the resulting heuristically-sized schematic was computed.
4. EinsTuner was then configured to minimize critical delay, subject to staying within the same area as the heuristic sizing.

The purpose of applying the heuristic was so that EinsTuner results could be compared to reasonably-tuned circuits. More importantly, however, EinsTuner was able to solve the optimization problems as indicated by the smallness of the projected gradient and infeasibilities at the solution.

In the case of the remaining benchmarks, identical steps to the above were followed, but instead of a heuristic initial sizing, the real initial sizes were used. The optimization was then constrained to tune at constant area. These designs had already been well-tuned prior to the application of EinsTuner.

Table 1 shows the size of the various benchmarks. Note that the largest problem, while still being a modest-sized 2,796-transistor circuit, had over 5,600 variables and over 5,600 constraints, a moderately large problem by nonlinear optimization standards.

Table 2 shows the actual numerical results. The critical path delay obtained by the heuristic method and the formal optimization, and the percentage improvement are shown in the third major column. In the case of the two actual designs, the original delay is shown in the heuristic column for convenience, but no heuristics were employed. An important measure of the success of the optimization is revealed by the smallness of the infeasibilities. The worst ("W") and average ("A") infeasibilities of the arrival time ("AT") and slew constraints are shown in the table. Every single constraint of every single problem was satisfied to within 1 ps or less. The number of iterations for even the largest problems is 300 or less, and the projected gradient is brought down dramatically from the start point. While it is possible to try to bring down the projected gradient further, our stopping criteria kicked in and terminated the optimization in the interests of efficiency. We have indications that any further progress would be small relative to the cost of carrying it out. Finally, the CPU time is shown in the right-most column. The largest problem ran for over a day! Several methods of reducing the long run times of EinsTuner are enumerated in Section 7.

Profiling results on some smaller problems indicate that 43% of the CPU time is spent in transient simulation, 36% in gradient evaluation and 20% in the nonlinear optimization routines. On c2670, SPECS simulated 0.25 million CCCs and computed 15 million gradients, and LANCELOT executed almost half a million conjugate gradient iterations!

Three additional benchmarks with 1,529, 2,145 and 5,325 gates are presently being tuned with EinsTuner, but the results are not available in time for this preliminary version of the paper.

In order to make additional comparisons, we have recently implemented a static tuning method similar to TILOS [4]. Instead of using a crude RC model, our version accurately simulates CCCs in the time-domain, but otherwise it has the same basic algorithm as TILOS, i.e., start with minimum area, and then iteratively add area in small increments to the most sensitive and timing-critical portions of the circuit. Initial results lead us to believe that our formulation using nonlinear opti-

Table 2: Numerical results.

Name	#Tx	Critical path delay			AT infeas.		Slew infeas.		#	Proj. grad.		CPU
		Heur. (ps)	Final (ps)	%	W (ps)	A (ps)	W (ps)	A (ps)	its.	Beg.	End	time (s)
inv3	6	174.7	126.8	27.4	2e-3	3e-4	1e-3	3e-4	70	120	7e-2	35.8
c17	28	241.1	178.8	25.8	2e-2	2e-3	9e-2	4e-3	58	290	0.3	152.4
a3_3	34	317.8	245.4	22.8	0.2	2e-2	0.5	3e-2	48	380	0.29	159.0
full_ adder	46	398.3	305.8	23.2	0.15	7e-3	6e-3	3e-4	83	290	0.21	339.0
iqia	72	274.4	242.9	11.5	2e-2	1e-3	4e-3	1e-4	115	330	7e-2	699.0
c8	584	917.7	719.1	21.6	0.3	2e-3	0.5	4e-3	123	380	2.9	6816
c432	960	1708.6	1369.3	19.9	0.4	5e-3	0.6	1e-2	214	380	5.8	18621
incre- menter	1554	630.3	595.3	5.6	0.1	8e-4	0.5	2e-3	129	380	6.5	25466
c880	1582	1976.9	1430.3	27.6	0.3	2e-3	1.0	3e-3	166	380	5.7	31710
c1355	2180	1502.0	1193.4	20.5	0.4	2e-2	1.0	9e-3	248	280	9.1	67929
c499	2216	1524.7	1200.7	21.3	0.4	8e-3	0.9	4e-3	283	370	9.4	78889
c2670	2796	1341.9	1066.2	20.5	0.3	1e-3	0.8	3e-3	283	380	7.0	104521

mization can achieve significantly better solutions than currently available heuristic methods. For example, as shown in Table 2, the critical path delay for `iqia` was improved by 11.5%. In comparison, the best improvement we were able to achieve using our own TILOS-like method was only 7%.

6 Circuit generality

The EinsTuner implementation described in this paper only accommodates combinational circuits consisting of parameterized library cells. Because the approach is simulation-based and includes a general-purpose nonlinear optimizer, it is readily extended to arbitrary transistor-level custom designs and sequential circuits. The software prototype will gain in generality by incorporation into a transistor-level timer and use of the timer's graph to generate the list of constraints. This section describes the additional considerations necessary.

6.1 Arbitrary transistor-level circuits

Two existing techniques will allow the extension of EinsTuner to arbitrary custom circuits. First, pattern-matching of the transistor topology in a CCC using graph isomorphism algorithms allows the recognition of a wide variety of gates [16], including dynamic logic such as self-resetting CMOS or domino gates. Once the gate-type is recognized, a pre-stored set of timing constraints is added to the SIF file for each gate of a particular type. The constraints for dynamic logic can include special timing requirements relating the arrival of the precharge signal to the data signals or special relationships between the forward and reset paths in the case of self-resetting CMOS.

Second, for topologies that cannot be recognized by pattern-matching, a state-traversal algorithm can be used to set up the propagate segments for the CCC [17]. The end result of either pattern-matching or state-traversal is a list of propagate segments (based on which a list of constraints can be generated), and the rules for simulating the CCC. Side-path loading and initialization of internal nodes of the CCC to actuate the worst-case pin-to-pin delay for each propagate segment is an important part of this analysis.

6.2 Sequential circuits

Sequential elements such as latches must first be recognized either by attribution of the netlist, or by pattern-matching. Depending on the type of sequential element, pre-compiled rules are followed to generate timing constraints and to simulate the element. For example, a latch will generate an additional set-up constraint. Special considerations are required for edge-triggered and transparent latches. For any type of latch, the required additional constraints must be added to the SIF file and the "back-end" configured to simulate the sequential element to evaluate each propagate segment.

7 Future work

In addition to the extensions described in the previous section, several avenues of future work suggest themselves. The long run times of EinsTuner can be ameliorated on many fronts. The direct method of sensitivity analysis may prove to be more efficient for the smaller (and most commonly encountered) CCCs. Employing a programming interface [18] to communicate with the optimizer will improve efficiency. The "adjoint Lagrangian" [3, 2] mode of gradient computation can be employed to compute all the gradients needed for a single CCC by means of a single adjoint analysis, which would dramatically reduce the CPU time for gradient computation. Automatic criticality- and topology-based pruning can be used to reduce the number of timing and slew constraints without loss of accuracy. Failure recovery can be implemented more efficiently to avoid the overhead of possible repeated failures. All CCCs of a certain type (say, NAND2 gates) can be simulated one after the other by "building" the circuit once and then repeatedly analyzing the gate after resetting transistor sizes each time. CCC evaluations are independent of one another and can be parallelized. Finally, SPECS simulation can be employed with a coarser device table model, thus trading some accuracy for speed (the results in this paper used 50 mV table model segments). More accurate models can be adaptively employed as convergence is approached.

Much can be done to make the nonlinear optimization more effective. Dealing with noisy data and determining good stopping criteria are topics of ongoing, but difficult, research. Two-step updating [15] can be applied to all arrival time variables, since they appear only linearly in the timing constraints.

In the future, one could envision taking noise constraints into account during optimization, using the mapping of semi-infinite noise constraints into equality constraints as in [19]. Simultaneous early and late-mode optimization could be employed to "fix" any fast-path problems. Tuning of wires along with transistors dovetails nicely into the formulation. Inclusion of timing constraints at several process corners and minimizing the worst negative slack across all the process corners can be used to improve parametric manufacturability. Where analytic delay rules exist as a function of transistor widths, they can be mixed with custom circuitry for the purposes of optimization.

8 Conclusions

This paper presented a unique formulation of the circuit optimization problem based on static timing analysis. By using large-scale, nonlinear optimization and fast transistor-level simulation and gradient computation, a wide range of circuits can be accurately optimized. Tuning of a number of datapath and control benchmark circuits has been demonstrated.

9 Acknowledgments

The authors gratefully acknowledge the work of Payam Heydari of the University of Southern California for his extensive work on SPECS sensitivity computation, which is a key ingredient of the EinsTuner project. David Ling provided SPECS device models for all the tuning runs and was involved with the project from the start. EinsTuner benefited greatly from the JiffyTune dynamic tuner development and the authors acknowledge the contributions of Ruud Haring and Chai Wah Wu at IBM Research. In particular, Ruud Haring provided the capcalc program which is used to model fanout capacitances in EinsTuner. We would like to thank Jeff Soreff of IBM East Fishkill for discussions on various aspects of timing and tuning. Jerry Kaminsky and David Ling provided managerial support and encouragement. Dan Brand, Ruud Haring, Payam Heydari, Prabhakar Kudva, Ruchir Puri and Phillip Restle reviewed the paper and gave us valuable suggestions and feedback.

References

[1] 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.

[2] 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.

[3] 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. Accepted for publication; to appear.

[4] 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.

[5] 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.

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

[7] 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.

[8] A. Srinivasan, K. Chaudhary, and E. S. Kuh, "RITUAL: A performance driven placement algorithm for small cell ICs," *IEEE International Conference on Computer-Aided Design*, pp. 48–51, November 1991.

[9] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer Verlag, 1992.

[10] 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.

[11] C. Visweswariah and J. A. Wehbeh, "Incremental event-driven simulation of digital FET circuits," *Proc. 1993 Design Automation Conference*, pp. 737–741, June 1993.

[12] 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.

[13] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, "Global convergence of a class of trust region algorithms for optimization with simple bounds," *SIAM Journal on Numerical Analysis*, vol. 25, pp. 433–460, 1988. See also same journal, pp. 764–767, volume 26, 1989.

[14] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, "A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds," *SIAM Journal on Numerical Analysis*, vol. 28, no. 2, pp. 545–572, 1991.

[15] A. R. Conn, L. N. Vicente, and C. Visweswariah, "Two-step algorithms for nonlinear optimization with structured applications," Research Report RC 21198(94689), IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598, June 1998. Submitted to *SIAM Journal on Optimization*.

[16] M. Ohlrich, C. Ebeling, E. Ginting, and L. Sather, "SubGemini: identifying subcircuits using a fast subgraph isomorphism algorithm," *Proc. 1993 Design Automation Conference*, pp. 31–37, June 1993.

[17] J. P. M. Silva and K. A. Sakallah, "GRASP–A new search algorithm for satisfiability," *IEEE International Conference on Computer-Aided Design*, pp. 220–227, November 1996.

[18] C. C. Douglas, D. A. George, and M. E. Henderson, "Object classes for numerical analysis," in *Proceedings of the second annual object-oriented numerics conference*, pp. 32–49, Rogue Wave Software, Inc., Corvallis Oregon, April 1994.

[19] A. R. Conn, R. A. Haring, and C. Visweswariah, "Noise considerations in circuit optimization," *IEEE International Conference on Computer-Aided Design*, November 1998. Accepted for publication.