

Uncertainty-Aware Circuit Optimization

Xiaoliang Bai
University of California at San Diego
La Jolla, CA 92037
xibai@ece.ucsd.edu

Philip N. Strenski
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
strensk@us.ibm.com

Chandu Visweswariah
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
chandu@watson.ibm.com

David J. Hathaway
IBM Microelectronics
Essex Junction, VT 05452
dhathawa@us.ibm.com

ABSTRACT

Almost by definition, well-tuned digital circuits have a large number of equally critical paths, which form a so-called “wall” in the slack histogram. However, by the time the design has been through manufacturing, many uncertainties cause these carefully aligned delays to spread out. Inaccuracies in parasitic predictions, clock slew, model-to-hardware correlation, static timing assumptions and manufacturing variations all cause the performance to vary from prediction. Simple statistical principles tell us that the variation of the limiting slack is larger when the height of the wall is greater. Although the wall may be the optimum solution if the static timing predictions were perfect, in the presence of uncertainty in timing and manufacturing, it may no longer be the best choice. The application of formal mathematical optimization in transistor sizing increases the height of the wall, thus exacerbating the problem. There is also a practical matter that schematic restructuring downstream in the design methodology is easier to conceive when there are fewer equally critical paths. This paper describes a method that gives formal mathematical optimizers the incentive to avoid the wall of equally critical paths, while giving up as little as possible in nominal performance. Surprisingly, such a formulation *reduces* the degeneracy of the optimization problem and can render the optimizer more effective. This “uncertainty-aware” mode has been implemented and applied to several high-performance microprocessor macros. Numerical results are included.

1. INTRODUCTION AND MOTIVATION

In the quest for high performance, much research effort has gone into using optimization methods to solve the transistor sizing problem [2, 1, 10, 12, 4, 3]. It has long been speculated that aggressive optimization of a circuit drives the design into a corner of the process space, causing its yield to suffer.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

39th Design Automation Conference New Orleans, Louisiana, June 10-14, 2002.

Copyright 2002 ACM ??? ...\$5.00.

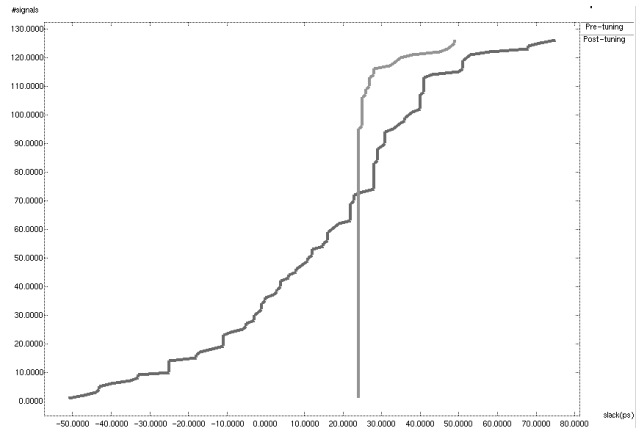


Figure 1: Pre- and post-tuning slack histogram.

One approach to solve this problem is to model the random and systematic effects that impact yield and use them to estimate and/or maximize yield. This task is obviously an important but daunting one. Another approach is to change the design so that it is as insensitive as possible to variations and uncertainties. Instead of attacking the slack of just the most critical path or paths to the exclusion of all other considerations, what is required is an optimization strategy that targets the entire *distribution* of slacks.

A well-tuned circuit has a large number of equally critical paths, which manifests itself as a *wall* in the slack histogram. Fig. 1 shows a slack histogram of a S/390 microprocessor execution-unit macro before and after tuning. A point (x, y) in the histogram implies that there are y signals with a slack of x or worse. The process of optimizing the circuit typically involves stealing transistor width from the off-critical paths and speeding up the critical path. The obvious outcome is a large number of equally critical paths, as shown in Fig. 1. The slack of the circuit has improved from -51 ps to +24 ps in this example, but 95 signals are now equally critical.

Unfortunately, the basis upon which the circuit is optimized has its limitations. Static timing makes a number of simplifying assumptions. The prediction of parasitics may be less than perfect. Model-to-hardware correlation problems often arise. The design may undergo further downstream tweaking and restructuring. Clock skew may not be

correctly taken into account. Unpredictable process variations may cause delays to change from their nominal values. In the face of all this uncertainty, having a wall of critical paths is undesirable. In [14], the authors list all the sources of uncertainty and advocate defining a “window of uncertainty” in the slack histogram. As long as the slack of any signal is in the window, there is no guarantee that the circuit will operate at the required frequency. Hashimoto et al. [11] go further and show that it is possible for careful optimization of a circuit to actually *degrade* the performance due to manufacturing variations. The impact of the height of the wall is demonstrated in a dramatic fashion in [11].

Consider an example of a mathematical optimizer tuning a circuit. To begin with, assume two paths are equally critical. As these paths are speeded up, their slacks become equal to the third critical path; now the optimizer must simultaneously work on *three* equally critical paths. As this process continues and optimality is approached, there are diminishing returns. Nonetheless the optimizer will not hesitate to make the number of equally critical paths larger even to gain a fraction of a pico-second. In fact, the less critical paths which are being down sized to provide device width (and perhaps reduced loading) for critical paths are *more* exposed to process variations because of small devices and large slews. Thus the quest for optimality in the limiting slack while excluding other considerations can be dangerous.

Without attempting to model the actual uncertainties, this paper proposes a method to avoid the wall of equally critical paths during optimization. A penalty is added to the objective function to give the optimizer an incentive to avoid a tall wall. The chosen form of the penalty has several good properties that are crucial to implementing a working solution to this problem. The resulting circuits have more appealing slack histograms, while paying a negligible price for the better slack distribution. In the face of uncertainty, the circuit with the better slack distribution offers both better performance and better insensitivity to variations. After optimization, if the required slack is still not met, restructuring is much easier when the number of equally critical paths is manageable. An interesting by-product of the new formulation is that the inherent degeneracy in the problem formulation is reduced, causing the optimizer to be more effective.

The outline of the rest of the paper is as follows. Section 2 presents the penalty function to implement uncertainty-aware tuning. Properties of the penalty function and choice of parameters are discussed. Section 3 extends this formulation to accommodate arrival time pruning. Numerical results and slack histograms are shown in Section 4. Alternative formulations of the optimization problem are discussed in Section 5 before the paper concludes with Section 6.

2. UNCERTAINTY-AWARE TUNING

2.1 Background

Assuming delay minimization, the optimization problem is formulated as

$$\begin{aligned}
 & \text{minimize} && z \\
 \text{s.t.} & && z \geq AT_i + RAT_i && i \in PO \\
 \text{s.t.} & && AT_j \geq AT_i + d_{ij}(x, s_i) && j \in (IN \cup PO), \\
 & && && i \in fanin(j) \\
 \text{s.t.} & && s_j \geq s_{ij}(x, s_i) && j \in (IN \cup PO), \\
 & && && i \in fanin(j) \\
 \text{s.t.} & && \sum_{i \in T} A_i(x_i) \leq A_{target} \\
 \text{s.t.} & && \sum_{j \in T} L_{ij}(x_j) \leq L_{target_i} && i \in PI \\
 \text{s.t.} & && \beta_{lower_i} \leq \beta_{smallest}(x) && i \in G \\
 \text{s.t.} & && \beta_{largest}(x) \leq \beta_{upper_i} && i \in G \\
 \text{s.t.} & && X_{lower_i} \leq x_i && i \in T \\
 \text{s.t.} & && x_i \leq X_{upper_i} && i \in T \\
 \text{s.t.} & && s_i \leq S_{upper_i} && i \in (IN \cup PO)
 \end{aligned} \tag{1}$$

where the following notation is employed.

$$\begin{aligned}
 z &= \text{auxiliary variable that represents cycle time} \\
 AT &= \text{arrival time variables } \textit{after} \text{ launching} \\
 &\quad \text{clock edge} \\
 RAT &= \text{required arrival times at POs } \textit{before} \\
 &\quad \text{capturing clock edge}^1 \\
 s &= \text{slew variables} \\
 x &= \text{transistor widths} \\
 d_{ij} &= \text{nonlinear delays of timing graph edges} \\
 s_{ij} &= \text{nonlinear slews of timing graph edges} \\
 A_i &= \text{contribution of transistor } i \text{ to circuit area} \\
 L_{ij} &= \text{contribution of transistor } j \text{ to loading of PI } i \\
 PO &= \text{set of primary outputs} \\
 PI &= \text{set of primary inputs} \\
 IN &= \text{set of internal nodes} \\
 T &= \text{set of tunable transistors} \\
 G &= \text{set of tunable gates} \\
 \beta_{smallest} &= \text{smallest P to N path-strength ratio} \\
 \beta_{largest} &= \text{largest P to N path-strength ratio} \\
 A_{target} &= \text{area target} \\
 L_{target} &= \text{input loading limits on PIs} \\
 \beta_{lower} &= \text{lower bounds on } \beta \text{ ratio} \\
 \beta_{upper} &= \text{upper bounds on } \beta \text{ ratio} \\
 X_{lower} &= \text{lower bounds on } x \\
 X_{upper} &= \text{upper bounds on } x \\
 S_{upper} &= \text{upper bounds on slews.}
 \end{aligned} \tag{2}$$

The above formulation has some interesting properties [4]. Note that the arrival times and slews are variables of the problem. The delays and slews of each arc, the β ratios, the input loading and the contribution of each transistor’s width to the total area are the nonlinear elements of the formulation. At the solution, the arrival times and slews on the critical path will have the “timing-correct” value, all others may not (they can take one of several equally correct values). Thus off-critical delay and slew constraints and variables are often redundant or degenerate. The formulation as shown above assumes downstream propagation of the largest slew of any input edge incident on a timing point, i.e., what is commonly referred to as “max-slew propagation.”

The above formulation typically creates the situation where a tuned circuit has a large number of equally critical signals, as shown in the post-tuning slack histogram of Fig. 1. The

¹By convention, required arrival times are either expressed *before* a capturing clock edge or *after* a launching clock edge.

reason is that the optimizer will strive to decrease the effective cycle time z until no further improvement is possible, thus causing a large number of paths to be equally critical.

2.2 Penalty function

To prevent this situation, we propose addition of a penalty term corresponding to each primary output. As the arrival time of the signal approaches criticality, we would like this penalty to grow in size. In delay minimization, we unfortunately cannot judge criticality in an absolute sense, since the effective cycle time (z) of the circuit changes as the optimization progresses. Hence, we formulate the penalty for each primary output arrival time in relation to the effective cycle time z . Define a *separation*

$$d_i = z - AT_i - RAT_i \quad i \in 1, 2, \dots, N \quad (3)$$

where N is the cardinality of the set PO . This separation represents the criticality of each primary output as the optimization evolves, and is relative to the most critical path at any stage of the optimization. We would like this separation to be as large as possible for all primary outputs. Of course, at least one of the primary outputs will define the effective cycle time, and the separation for that one will be zero.

The basic idea is to change the objective function from z to

$$z + k \sum_{i=1}^N P(d_i) \quad (4)$$

where k is a weighting constant and P a suitably chosen penalty function such that the optimizer strives to balance reduction in z with increase in desirable separation.

2.3 Choice of penalty function

There are several requirements on the penalty function. First, since it will be used in a numerical optimizer, it is required to be continuous and smooth. Second, the penalty should be a monotonically decreasing function of d_i , reducing to a negligibly small value after a “required” separation is achieved. Third, its value should be well defined at both zero and negative values of separation (this rules out, for example $P = 1/d_i$). Finally, we would like to control the rate at which the penalty function drops off as the separation increases, since this will provide a knob to adjust for expected amounts of uncertainty.

A simple choice that satisfies all these criteria is

$$P(d_i) = e^{-d_i/\sigma} \quad (5)$$

leading to an objective function which is

$$z + k \sum_{i=1}^N e^{-(z-AT_i-RAT_i)/\sigma}. \quad (6)$$

Of course, several other choices of the penalty function are possible, but this simple choice works well in practice. The penalty starts at k for the most critical primary output and reduces by a factor e for each σ of separation that a primary output obtains relative to the most critical path. Beyond about 3σ or 4σ , relatively speaking, further separation is no longer valuable.

2.4 Choice of k and σ

The purpose of the penalty function is to introduce *downward* pressure on all primary output arrival times relative to

z . Unfortunately, this results in *upward* pressure on z . Thus the parameters of the penalty function must be chosen in such a way as to make sure that there is overall downward pressure on z . In other words, the optimizer should not artificially increase z to be higher than the actual cycle time of the circuit just to obtain a reduction in the separation penalty terms. We term this situation “lift-off” of z .

One (conservative) way to ensure this criterion is to make sure the gradient of the penalty function with respect to z is always positive. Differentiating (6) with respect to z , we obtain

$$1 - \frac{k}{\sigma} \sum_{i=1}^N e^{-d_i/\sigma} > 0. \quad (7)$$

The biggest possible value of the summation is N (*all* primary outputs equally critical) and the smallest value is 1 (one primary output critical, all others separated by a lot). Thus we will be safe provided

$$1 - \frac{kN}{\sigma} > 0 \text{ or } k < \frac{\sigma}{N}. \quad (8)$$

Therefore if k is less than σ/N , lift-off is impossible. If it is more than σ , lift-off is guaranteed! In between these two extremal values, the behavior depends on the relative arrival times in a particular test case.

For a circuit with 50 primary outputs, considering both rising and falling arrival times, we have 100 timing points. Thus, if we choose $\sigma = 10$ ps, k should be less than 0.1. In practice, k can be chosen slightly larger without deleterious effects. However, satisfying (8) guarantees us that z will not lift off.

It turns out that lift-off is not really harmful since as soon as z artificially increases, the separation term for the most critical signal is smaller than k and all the other separation terms also decrease, so the gradient of the objective function with respect to z quickly increases till z will not increase further. Suppose the amount of lift-off in z at optimality is l , then the objective function can be rewritten as

$$l + z' + k' \sum_{i=1}^N e^{-(z'-AT_i-RAT_i)/\sigma} \quad (9)$$

where $z' = z - l$ and $k' = ke^{-l/\sigma}$ and therefore we obtain a solution to a different problem with a smaller k but no lift-off.

2.5 Area minimization and tradeoff modes

The formulation in (1) can easily be modified to create a tradeoff mode which minimizes a weighted sum of area and effective cycle time or an area mode which minimizes area subject to timing constraints. In the tradeoff mode, the area constraint is just moved into the objective function with an appropriate weight factor. In area mode, area is the objective function. In this case, z is replaced by the actual desired cycle time of the circuit being optimized (optionally offset by a desired positive slack), and therefore is a constant. Nonetheless, all of the above analysis and the choice of the penalty function for these two additional modes are unchanged.

3. ARRIVAL TIME PRUNING

As was mentioned earlier, the formulation in (1) suffers from a high degree of redundancy and degeneracy. Off-critical ar-

rival time and slew constraints can be tight, yet have zero Lagrange multipliers since they do not influence the objective function. One way in which this problem is addressed is by arrival time pruning [15]. Arrival time pruning results in a mixed path/block formulation. In fact, for small circuits and “non-bushy” circuits, *every* arrival time variable is often pruned, leading to a completely path-oriented formulation. From an uncertainty-awareness point of view, this pruning has benefits, but raises new problems.

The benefit is that the “separation” is not just forced at the primary outputs, but at the source of all sub-paths from unpruned internal arrival times to the primary outputs. The disadvantage is that the separation variable d_i of (3) can no longer be defined in a straightforward manner since the arrival time at the primary output might have been eliminated due to pruning. Indeed, a typical primary output arrival time constraint will be of the form

$$z \geq AT_1 + d_{1,2} + d_{2,3} + \dots + d_{i-1,i} + RAT_i \quad (10)$$

where the d values are the delays along the sub-path from the internal node 1 to the primary output i . Hence the separation variable for this minimax constraint is

$$d = z - AT_1 - d_{1,2} - d_{2,3} - \dots - d_{i-1,i} - RAT_i \quad (11)$$

which is not only nonlinear unlike (3), but a function of a large number of variables. This does not lend itself to a penalty function as described in the previous section.

We make the observation that most optimizers convert inequalities to equalities by the addition of a slack variable. Although this is not to be confused with slack as defined by a static timing analysis tool, the conceptual meaning of the two slacks is similar. Thus (10) would become

$$\begin{aligned} z - AT_1 - d_{1,2} - d_{2,3} - \dots - d_{i-1,i} - RAT_i - u &= 0 \\ u &\geq 0 \end{aligned} \quad (12)$$

where u is an auxiliary slack variable introduced by the optimizer. In this case, we can define the penalty function in terms of this slack variable as $P(u)$! The procedure would be simply to keep track of the slack variables of all the minimax constraints (of the form $z \geq \text{something}$) and then add a penalty to the objective function which is $k \sum_{i=1}^N e^{-u_i/\sigma}$ where N is the number of minimax constraints. Since u_i is always positive, each penalty term is at most k when $u_i = 0$ and then decays quickly as the slack variable increases in value. The key difference is that the number of penalty terms in this method can be much larger than the method of the previous section, so the same care must be taken in the choice of k and σ to make sure that there is eventually sufficient downward pressure on z . Also note that *two-step updating* [8] which takes advantage of the fact that slack variables appear in the merit function in a particular analytic form is no longer applicable to the slack variables that serve this second purpose as separation variables.

With this new formulation, there is downward pressure on *every* primary output arrival time, since the critical ones need to reduce to allow reduction of z , and the others need to decrease to allow increased separation! The downward pressure is transitively passed all over the network, thus reducing the arrival time and slew degeneracy of the formulation. Further, unlike the formulation of (1), the optimizer’s final results will match the arrival times and slews that a static timer would predict on the tuned circuit.

4. RESULTS

The methods described in both the previous sections have been implemented in a circuit optimization package called EinsTuner, which has been applied to several generations of PowerPC and S/390 microprocessors. The main components of EinsTuner are a static, transistor-level timer EinsTLT [13], a fast time-domain simulator and gradient calculator SPECS [16, 9] and a general-purpose nonlinear optimization package LANCELOT [7, 5, 6]. The problem is formulated as in (1), including arrival time pruning [15]. The delays and slews of individual logic blocks are computed by running a time-domain simulation under the covers; gradients with respect to transistor widths, input slews and output loads are obtained by the adjoint method. The resulting optimization problems can be very large and the resulting run time quite long. Recently, a circuit with 47,000 transistors was tuned. The optimization problem had 140,000 variables and 110,000 constraints and took over 4 days of CPU time to solve.

The straightforward implementation of uncertainty-aware tuning in EinsTuner was achieved by modifying the pruning algorithm to leave the primary outputs unpruned and adding a penalty function as in (4). The *group function* facility of LANCELOT [7] was exploited to express the penalty function. This method will be referred to as method 1.

In the second method, all arrival times were allowed to be pruned. The slack variables of the resulting minimax constraints were used as separation variables to form the penalty function, again expressed as a group function. Minimax constraints were submitted to the optimizer as equality constraints and the slack variables and their bounds were explicitly managed by EinsTuner. Thus LANCELOT did not know that these were indeed slack variables, and did not attempt to apply two-step updating to them. This method will be referred to as method 2.

This section presents three different types of results. First, slack histograms are shown to demonstrate the avoidance of the “wall,” and the effect of varying k and σ are shown. Second, the performance of a circuit with and without uncertainty-aware tuning is compared in the face of variations. Third, numerical results are presented which show that uncertainty-aware tuning can be accomplished with little to no loss of circuit performance.

4.1 Slack histograms

A 38-bit PowerPC incrementer was tuned with and without uncertainty-aware tuning. This is a difficult circuit on which to achieve separation because of the symmetry and regularity of the critical paths. The resulting slack histograms are shown in Fig. 2. The slack histograms in this section were produced using method 1 of uncertainty awareness; method 2 results look qualitatively similar. Although the uncertainty-aware tuning has a limiting slack that is 8 ps worse than the nominally tuned circuit, the height of the wall has been reduced in more than half.

As k and σ are changed, the amount of separation can be controlled, but at progressively higher cost in the limiting slack. Fig. 3 shows the effect of varying k at a fixed σ and slack histograms obtained by varying σ while holding k fixed are shown in Fig. 4. These experiments were performed on the same S/390 execution-unit macro as Fig. 1.

4.2 Degradation due to variations

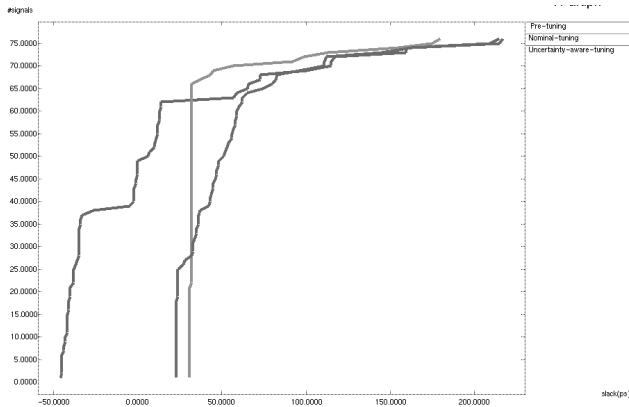


Figure 2: Slack histogram for nominal and uncertainty-aware tuning.

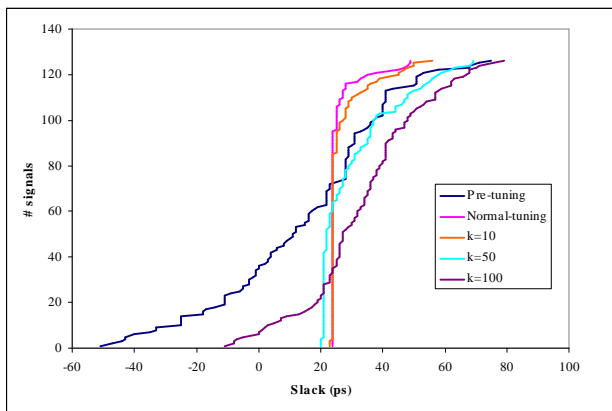


Figure 3: Effect of varying k .

It is clear that “separation” can be obtained at a relatively small cost in the limiting slack. In many cases, there is almost no price to pay in terms of the limiting slack, as will be shown in the subsequent section. But is the uncertainty-aware circuit really better? A simple Monte Carlo experiment was fashioned to gain insight into this question. Ideally, the nominally tuned circuit and uncertainty-aware circuit would be manufactured and the parametric yields of the two chips compared. Obviously, such an experiment is impractical.

Instead, each point in the slack histograms of the 38-bit PowerPC incrementer (Fig. 2) was *randomly perturbed* according to a 6σ -truncated Gaussian distribution of the form

$$sample = \mu + \sigma\sqrt{2}g(K(2r - 1)) \quad (13)$$

where μ is the mean of the distribution (the mean arrival time was left unchanged), σ^2 is the variance, g is the inverse erf function, K is $erf(3/\sqrt{2})$ and r is a random number between 0 and 1. In other words, the arrival time of each primary output was assumed to have a truncated Gaussian distribution with no correlation to other primary outputs. While this is clearly untrue in real life, the crude experiment serves to capture the intent of uncertainty-aware tuning. For the nominally tuned circuit, 10,000 circuits were “manufactured” by means of a Monte Carlo analysis whereby each primary output arrival time was randomly perturbed. The

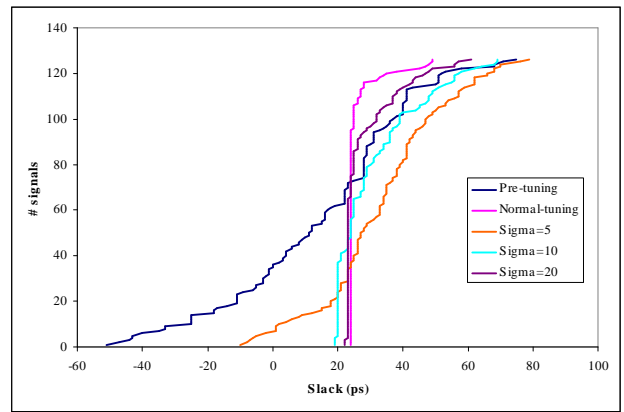


Figure 4: Effect of varying σ .

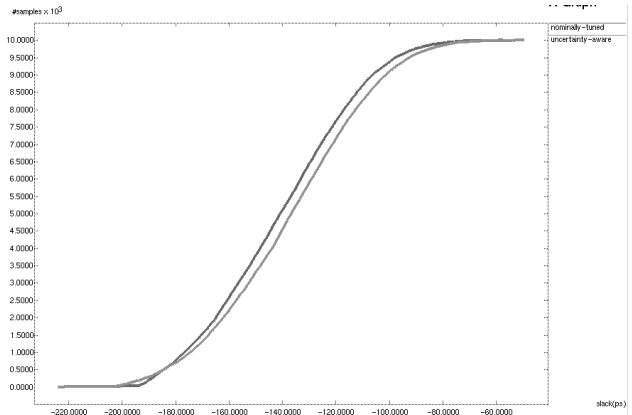


Figure 5: Estimated slack histograms after random perturbation for 38-bit PowerPC incrementer.

resulting circuits were “sorted” in order of increasing performance and the resulting distribution in the performance of the chips is shown in Fig. 5. The same analysis was repeated for the uncertainty-aware circuit, with the same random number seeds for each new “chip.” The results are superposed in Fig. 5. The perturbation caused both results to perform worse than the nominal circuit, which is the well-known behavior of the max function in the face of variations [11]. Nonetheless, the reduction of the steepness of the wall in the uncertainty-aware circuit made it the better circuit in the face of variations. It can be seen that for a given performance, the uncertainty-aware circuit has a higher parametric yield, and for a given parametric yield, the uncertainty-aware circuit allows a higher chip frequency. Even when the magnitude of the yield improvement is small, it can make a huge difference in reducing costs and enhancing profitability.

The same experiment was repeated on the execution unit circuit of Fig. 1 and the resulting performance distributions are shown in Fig. 6. Thus even for these regular datapath circuits, benefits can be realized from separation.

4.3 Numerical results

A set of 39 test cases ranging in size from 6 to 13,046 transistors were tuned in three different ways: nominal tuning in

Table 1: Comparison of nominal and uncertainty-aware optimization.

Test case	# of FETs	Slack (ps)			
		Pre-tuning	Nominal	Method1	Method2
ioaareg_3t2n	22	-2.5	7.3	6.9	6.4
inv3	6	720.0	758.0	759.0	759.0
ibmTechGates	22	83.3	132.0	132.0	131.5
c17	28	683.0	738.0	738.0	738.0
iiiff	26	-161.0	-62.4	-61.1	-60.1
a3_3	34	634.0	718.0	718.0	718.0
andy_graph	34	634.0	651.0	654.0	650.0
full_adder	46	632.0	687.0	687.0	687.0
iqia	72	645.0	668.0	668.0	668.0
tgmux	62	-106.5	-65.9	-66.1	-66.4
alu1b	102	583.0	618.0	617.0	618.0
gain_experiment	162	-118.5	-41.4	-41.4	-41.6
rtt_incr32l	716	-129.1	-41.4	-46.5	-48.0
ibbx_compare	824	514.0	554.0	554.0	554.0
exdmsk	882	609.0	683.0	680.0	680.0
ioaareg_dl	772	-22.1	39.1	38.0	36.1
c8	584	432.0	590.0	588.0	587.0
exdcgn_mac	1270	-16.5	30.3	31.1	30.8
epdx3p_mac	802	-54.5	-0.8	0.3	1.1
ifti_incr38	880	454.0	530.0	530.0	529.0
vadd_byte	1428	-77.7	6.2	7.3	7.7
epdlzd_mac	1346	-30.6	101.6	100.9	99.7
rtincrp	1725	-221.9	-115.9	-118.5	-117.5
exdalmx_mac	1792	-0.5	40.3	40.2	40.2
incrementer	1554	447.0	511.0	507.0	509.0
ifbreg_ge56	1400	5.0	424.0	422.0	417.0
exdcdr_mac	1856	-407.9	-325.7	-329.2	-320.1
rtbg58cmp	1500	-42.1	29.9	30.3	29.4
ioperdf_cmpr_al64p	2256	-12.5	164.9	165.6	164.8
idopcmp_mac	3448	10.4	50.7	52.5	51.0
idagiqu_smry_gr_wr_tune	2332	-0.5	33.7	33.2	33.0
exdadd_mac2	4258	-388.3	56.3	58.1	57.0
exdadd_mac	4258	-265.8	-85.9	-86.6	-87.4
iooacrd_mac	5184	-200.7	-200.1	-200.1	-200.1
epdinc_mac	7199	-46.2	83.9	84.6	84.4
rtoutmx_mac	7805	-95.1	-10.9	-5.3	0.7
exdblu_mac	8054	-619.6	-504.5	-500.5	-504.2
idfast_mac	8836	-480.7	-473.7	-473.5	-479.7
ibbhtdf_rlm	13046	11.7	62.6	60.7	64.8

delay minimization mode, uncertainty-aware tuning method 1 and uncertainty-aware tuning method 2. All other constraints such as the area limit, β ratio constraints, slew limits and input loading constraints were maintained the same in all three cases. Table 1 shows the results. In most cases, the loss of performance due to uncertainty-aware tuning is very small. As discussed previously, the new method eliminates some redundancy and degeneracy in the formulation, causing the optimizer to be more effective as can be seen in some test cases, particularly the large ones. Thus added slack separation can be obtained at little or no performance cost.

5. ALTERNATIVE FORMULATIONS

This section proposes two different formulations to make the

uncertainty awareness more effective. It can be argued that the objective of providing variation tolerance is only important at the primary outputs and latch points of a circuit. But we can only achieve this if we have some separation, or positive timing slack, on all internal points in the fan-in cone of each primary output or latch point. And if we assume that all delay edges in the design are equally subject to variation, we can achieve maximum variation tolerance by ensuring that we have separation at the maximum number of internal delay edges, where separation on a delay edge from i to j means that $AT_i + d_{ij} < RAT_j$. Here, RAT_j is defined relative to the clock launching edge, unlike the primary output required arrival time assertions which are defined as times before the capturing clock edge. In method 1 described above, we require separation only on the worst path feeding each primary output. By reusing the slack vari-

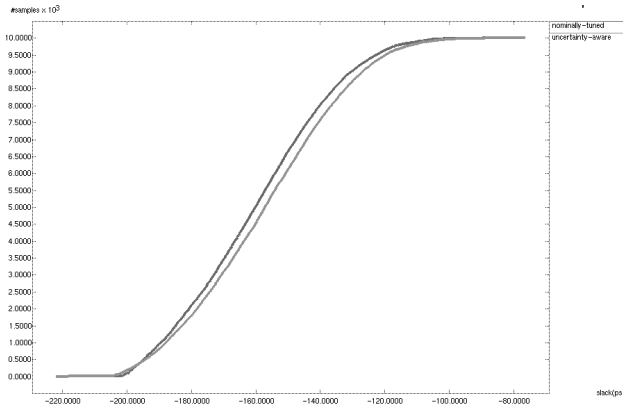


Figure 6: Estimated slack histograms after random perturbation for S/390 execution unit macro.

ables of the minimax constraints in method 2, we are able to extend this separation pressure to all sub-paths terminating on a primary output. This section discusses methods to give the optimizer incentive to encourage separation at *all* timing points of a circuit.

One way to achieve this goal is to introduce a required arrival time variable (*RAT*) at each timing point. These required arrival times are defined relative to the launching clock edge, as above. The *RAT* variables form a parallel set of constraints that are otherwise identical to those of the arrival time variables. However, these constraints are “anchored” by the z variable rather than by the arrival time assertions at the primary inputs (no constraints are introduced relating the *RAT* variables to the *AT* assertions at the primary inputs). Thus the *AT* variables are constrained to grow “up” from the primary input *AT*s, while the *RAT* variables are constrained to grow “down” from the z variable. The difference between the *RAT* and *AT* variables at a timing point is the timing slack, and by treating this as a separation variable and adding a corresponding penalty term to the objective function, we can ensure that downward pressure is applied to every *AT* variable and upward pressure is applied to every *RAT* variable. Along the critical path, the separation will be zero, and on every off-critical path, the separation will correspond to the positive slack that a static timer would predict, relative to the most critical path. The only drawback is that this method would lead to a substantial increase in the number of variables and constraints.

Another way of achieving the same goal is to define a separation for every unpruned arrival time in the circuit relative either to z or to some constant arrival time that is chosen to be larger than z . Then a penalty term can be introduced to ensure downward pressure on each arrival time such that wherever it is possible to achieve separation, the optimizer will have incentive to do so.

An advantage of the former (*RAT*-variable) approach using an exponential or similarly decaying penalty term is that the maximum pressure is applied to the timing points with the smallest timing slack, and thus the optimizer is given more incentive to increase the slack of the worst paths. In contrast, if we apply a separation penalty relative to a single common value, a decaying penalty function is inappropriate, since the natural separation of each *AT* variable from this

common value will vary significantly, leading to uneven separation pressures. Thus a linear penalty term is used (e.g., a simple sum of all *AT* variables in the problem), and the optimizer may increase the separation at a number of points with already large positive timing slacks at the cost of more valuable separation at a smaller number of nearly-timing-critical points. An advantage for the optimizer of both of these approaches is that they completely eliminate degeneracy in the timing and slew constraints.

Fig. 7 gives a visual comparison of the original tuning problem and the application of different variations of uncertainty-awareness. The left vertical edge of each figure represents the primary input arrival time assertions, and the right side represents the z variable which we are trying to minimize. Open circles represent *AT* variables, solid lines represent delays, and dashed lines represent the constraint slacks. The bold solid lines represent the critical path. The double-headed arrows associated with some arrival time variables represent degeneracies in these variables. The zig-zag lines represent separations for which penalty terms will be added to the objective function. Finally, the filled circles and the heavy dashed lines connecting them represent *RAT* variables and the constraints between them.

In Fig. 7-a we see a typical result of a conventional tuning approach, where all constraint slacks are forced to zero and all paths have been made exactly equal, yielding a slack wall. Although we show primary output *AT*s for clarity, note that these would most likely have been pruned. In Fig. 7-b we see a case where, due to other constraints which create upper bounds on delays, some paths end up with positive slacks. In this case we avoid the slack wall, but end up with degeneracy in the problem, where the slack values to the right and left of the degenerate arrival time variables can be arbitrarily traded off against each other. In Fig. 7-c we show method 1 uncertainty-awareness, in which a separation term is introduced between each primary output *AT* and z , and we have removed some of the problem degeneracy. Note that the separation term for the critical primary output has been compressed to zero but is shown to illustrate that it remains in the problem formulation; if this were not the case we would have “lift-off.” In Fig. 7-d we show method 2 uncertainty-awareness, in which primary output *AT*s have been pruned and a separation term has been introduced for each primary output constraint, further reducing the problem degeneracy. In Fig. 7-e we show the application of a separation term between each *AT* variable and a common reference (in this case z). Note that all degeneracy has been removed, but that the magnitude (lengths) of the separations vary widely. Finally, in Fig. 7-f we show the introduction of *RAT* variables and constraints between corresponding *AT* and *RAT* variables. Note that the chain of required arrival time constraints begins at the outputs (z) and grows backward, and that the constraints between the primary inputs and the *RAT* variables are omitted. The pairs of *AT* and *RAT* variables between which separation terms appear represent the same timing point, and the delays between corresponding pairs are actually the same delay values; they are duplicated in the figure for illustrative purposes.

6. CONCLUSIONS AND FUTURE WORK

Mathematical optimization of digital circuits causes a “wall” of equally critical paths, which is unsuitable to downstream

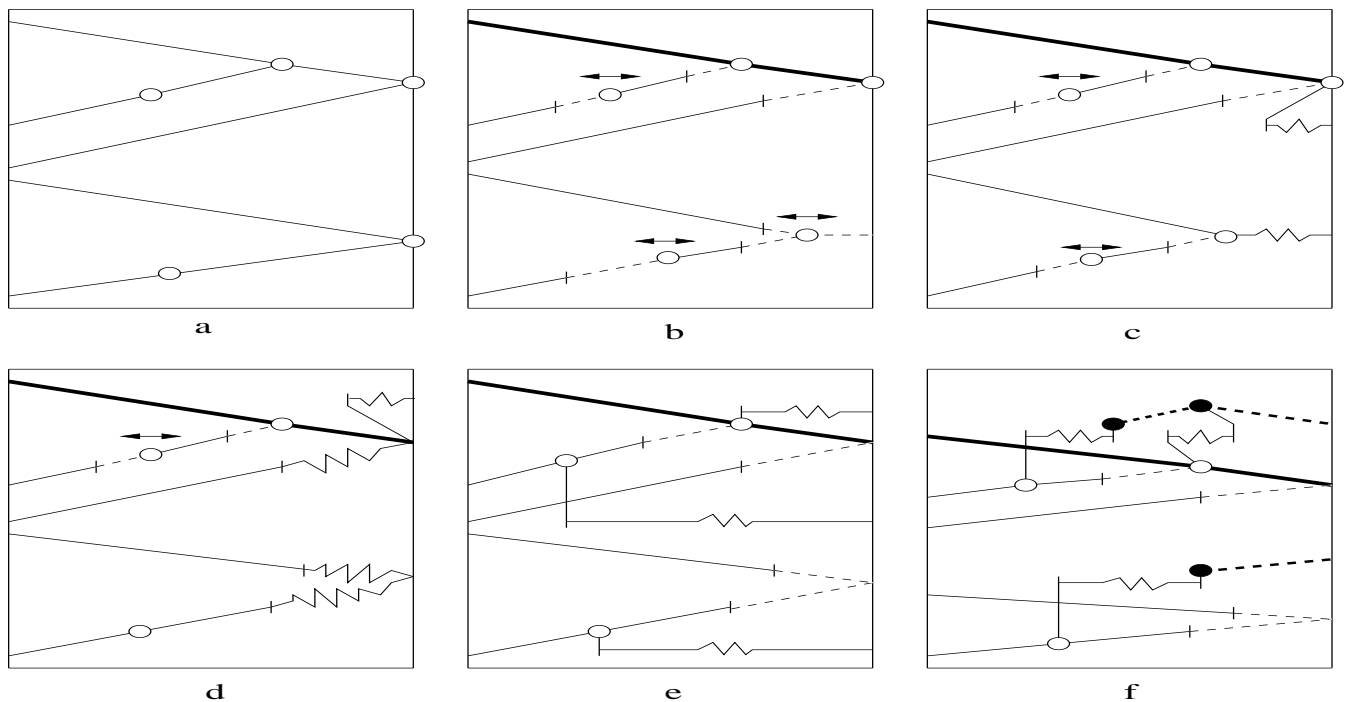


Figure 7: Formulations to improve the effectiveness of uncertainty-aware tuning.

restructuring in the design methodology and leaves a circuit extremely vulnerable to manufacturing variations. This paper proposed and demonstrated a simple yet effective method of avoiding the wall at little or no cost in circuit performance. In fact, the new method makes the optimizer more effective by eliminating degeneracy in the problem. The method was extended to the situations when arrival times are pruned by making the slack variables of the optimizer play a dual role as separation variables. Detailed numerical results on high-performance microprocessor macros were presented to show the effectiveness of the method.

7. ACKNOWLEDGMENTS

The authors gratefully acknowledge discussions with Sani Nassif, Jim Warnock and Jochen Jess, and the help of members of the EinsTuner team, particularly Jun Zhou, Mei-Ting Hsu, Cindy Washburn and Andy Conn.

8. REFERENCES

- [1] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli. A survey of optimization techniques for integrated-circuit design. *Proceedings of the IEEE*, 69(10):1334–1362, October 1981.
- [2] R. K. Brayton and R. Spence. *Sensitivity and optimization*, volume 2 of *CAD of Electronic Circuits*. Elsevier Scientific Publishing Co., Amsterdam, The Netherlands, 1980.
- [3] C.-P. Chen, C. N. Chu, and D. F. Wong. Fast and exact simultaneous gate and wire sizing by Lagrangian Relaxation. *IEEE Transactions on Computer-Aided Design of ICs and Systems*, 18(7), July 1999.
- [4] A. R. Conn, I. M. Elfadel, W. W. Molzen, Jr., P. R. O'Brien, P. N. Strenski, C. Visweswariah, and C. B. Whan. Gradient-based optimization of custom circuits using a static-timing formulation. *Proc. 1999 Design Automation Conference*, pages 452–459, June 1999.
- [5] A. R. Conn, N. I. M. Gould, and P. L. Toint. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM Journal on Numerical Analysis*, 25:433–460, 1988. See also same journal, pp. 764–767, volume 26, 1989.
- [6] A. R. Conn, N. I. M. Gould, and P. L. Toint. A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572, 1991.
- [7] A. R. Conn, N. I. M. Gould, and P. L. Toint. *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer Verlag, 1992.
- [8] A. R. Conn, L. N. Vicente, and C. Visweswariah. Two-step algorithms for nonlinear optimization with structured applications. *SIAM Journal on Optimization*, 9(4):924–947, September 1999.
- [9] 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*, 10(2):171–183, February 1991.
- [10] J. P. Fishburn and A. E. Dunlop. TILOS: A posynomial programming approach to transistor sizing. *IEEE International Conference on Computer-Aided Design*, pages 326–328, November 1985.
- [11] M. Hashimoto and H. Onodera. Increase in delay uncertainty by performance optimization. *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 379–382, 2001.
- [12] 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*, CAD-7(4):501–519, April 1988.
- [13] V. B. Rao, J. P. Soreff, T. B. Brodnax, and R. E. Mains. EinsTLLT: transistor-level timing with EinsTimer. *Proc. TAU*, December 1999.
- [14] S. E. Rich, M. J. Parker, and J. Schwartz. Reducing the frequency gap between ASIC and custom designs: a custom perspective. *Proc. 2001 Design Automation Conference*, pages 432–437, June 2001.
- [15] C. Visweswariah and A. R. Conn. Formulation of static circuit optimization with reduced size, degeneracy and redundancy by timing graph manipulation. *IEEE International Conference on Computer-Aided Design*, pages 244–251, November 1999.
- [16] C. Visweswariah and R. A. Rohrer. Piecewise approximate circuit simulation. *IEEE Transactions on Computer-Aided Design of ICs and Systems*, 10(7):861–870, July 1991.