Explicit Computation of Performance as a Function of Process Variation

Lou Scheffer
Cadence

ABSTRACT

Each manufactured chip is a little bit different, and designers want as many as possible of these chips to work. Process variation is a function of many variables, as the width, thickness, and inter-layer thickness can vary independently for each layer on a chip, as can temperature and voltage. Currently designers cope with this by picking a few subsets of these conditions, called process corners, and analyzing at these conditions. However, it's easy to show this approach is both too conservative (the specified conditions will seldom occur) and not conservative enough (it misses errors that can occur due to process variation). We present a unified theory of process variation that includes inter-chip variation, intra-chip deterministic variation (such as caused by proximity effects and metal density), and intra-chip statistical variation. Using this mechanism, we can explicitly compute performance as a function of process variation. This allows us to compute less pessimistic timing numbers and address yield optimization in the design process.

Categories and Subject Descriptors

J.6 [**Computer Applications**]: Computer Aided Engineering

General Terms

Algorithms, Performance, Design, Verification

Keywords

Static timing, Process variation, Statistical timing, Yield

1. INTRODUCTION

Not all chips are created equal. Building a chip is a sequence of hundreds of operations, each of which will occur a little differently on each chip despite enormous effort to make them repeatable[8].

This variation occurs in many ways. First, there is variation from chip to chip - often one chip is significantly faster than another. Some of these variations, such as gate delay, are tightly correlated. If one gate is fast, then so are all the others on the same chip.

Variations in interconnect follow a different pattern. One machine, in one action, lays down all the metal (or dielectric) for a layer of a chip. A different machine, or the same machine at a different time, lays down the next layer. Thus the layers are not correlated to each other, but they are correlated across the chip. If metal-1 is thick, then metal-1 will be thick across the whole chip, but this implies nothing about the thickness of metal-2.

There is also variation within a single chip[7]. Some of this is determined by the layout - for example the metal or inter-layer dielectric (ILD) thickness may depend on the the local metal density - and some is an artifact of manufacturing, such as cross-chip gradients.

Finally, all the above variations have a random component as well, due to manufacturing imperfections. This statistical component will vary depending on the distance between two objects on a chip. Two components close to each other on a chip are likely to be closely matched, whereas two components further away will have larger differences.

Process variation occurs in a highly multi-dimensional space. Interconnect alone occupies $3N$ dimensions, where N is the number of routing layers. For each routing layer there are three main variables - metal thickness, metal width, and inter-layer dielectric thickness. (Note that the metal width and metal spacing do *not* vary independently - their sum, the pitch, is extremely well controlled, so they are precisely anti-correlated.) Cell delays add at least three more dimensions, historically P (process), V (voltage), and T (temperature). P, intended to represent the cell speed, is a composite of more fundamental variations such as threshold voltage and oxide thickness. V and T are operating conditions and not manufacturing variances, but share many of the same characteristics and can benefit from the same analyses. This paper looks mainly at interconnect variation but the same approach applies to the other sources of variation as well.

2. DETERMINISTIC VARIATION

How can we take these different sources of variation into account? The variations in each parameter are the sum of two kind of effects - the deterministic variation, which can in theory be predicted from the layout, and the random variation, which cannot. For example, the final width of a metal line is determined partially by interactions with neighboring lines (the predictable part) and partly by the exact exposure and etching conditions on this particular chip (the unpredictable part).

Accounting for deterministic local variation will improve any analysis. It is particularly important before statistical analysis since the deterministic effects are both reproducible and highly correlated. Therefore treating these effects statistically will result in either significant errors or relatively little improvement, or both.

Deterministic effects can be modelled straightforwardly, at least in flat extraction. For example,

- Compute the nominal width of each conductor from the neighboring width/spaces. The residual expected after OPC should be used for this correction.
- Compute the nominal thickness of each conductor and ILD from the local density map for that layer.
- Derive the R and C values from the local width and thickness.

Hierarchical extraction will require deferred evaluation in some form. If we extract a block in isolation, for example, the average local metal density is not known for features near the sides or corners. One solution is to keep nominal values and derivatives, and perform correction once the environment is known. Keeping derivatives is examined in detail in section 4 for the purpose of computing inter-chip variations, but can also be used to enable hierarchical correction for deterministic effects.

2.1 Previous work using corner cases

Since working in 30 or so dimensions is difficult, designers have made various approximations. First, deterministic effects have been largely ignored, and the resulting variation lumped into the random component (though this is changing[6, 2]). Next, the number of process variation combinations is reduced to a small number of "process corners". A process corner sets all relevant variables to an extreme (usually 3σ) value. This corresponds to a corner of a hypercube in the real process space. If the process space has N_D dimensions, there are 2^{N_D} process corners - far too many to analyze (much less the interior points). Since timing is the most important result, designers choose a subset of these corners where they expect the cells and/or interconnect to be particularly fast or slow.

2.1.1 Analysis at fast/slow corners

This approach assumes that the most extreme delay cases will be the worst. One worst case will be with slow cells and slow interconnect; this will be used for checking for setup time problems. Fast cells and fast interconnect will be used for checking for hold problems. The two interconnect cases are obtained by using two extractions - one at fast interconnect corner, one at the slow interconnect corner. The two cell models are evaluated at the worst and best combination of PVT (process, voltage, and temperature). This approach does not address intra-chip variation at all, as it assumes all nets and cells scale exactly the same way.

2.1.2 4 and 6 corner analysis

What about intra-chip variation, which is smaller than inter-chip variation, but is still present? Analog designers have looked at this in detail[9], but in digital designs simpler methods are used. The major worry is that problems may arise if the clock and data signals do not track precisely.

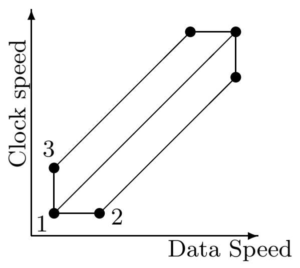

Figure 1: 6 corner cases

One technique, called *4 corner analysis*, examines setup and hold times in fast and slow process conditions. In each of these four cases, the launching clock skew, the logic delay, and the receiving clock skew are set to their most pessimistic values.

Another technique for addressing this problem is called *6 corner analysis*. It depends on classifying every gate and net as clock or data. The six corner cases that are then analyzed are (refer to Figure 1):

- (1) clock and data both maximally slow
- (2) clock maximally slow; data almost as slow
- (3) data maximally slow; clock almost as slow

and the 3 corresponding cases with fast instead of slow. A complete timing analysis is done in each case. This approach assumes that delays within the clock network will track, but clock and data may respond somewhat differently to process variation. This method (and the 2 and 4 corner methods) assume that the various branches of the clock network of the clock network are affected similarly by process variations. This is not a serious restriction if the clock network is built from the ground up, but is hard to enforce in an era of included IP blocks, each with their own clocking structure.

2.1.3 Problems with corner analysis

One big problem with worst case analysis is that it is too conservative. It is extremely unlikely to have a 3σ variation on each of 30 or so dimensions. Without further analysis we cannot tighten this bound, however, since it is possible that the timing of at least some critical paths are determined by only one process variable (say metal-1 resistance). Then a 3σ variation could in fact occur with significant probability.

The other problem is that corner analysis is not conservative enough, or in other words it can miss real errors. Take, for example, a flip flop where the data line is dominated by metal-1 delay and the clock line by metal-2 delay. Then the worst case for setup is when metal-1 is slow and metal-2 is fast. The worst case for hold time is when metal-1 is fast and metal-2 is slow. Neither of these two cases is found by either a 2, 4, or 6 corner analysis.

Another source of missed errors is shown in the example below. The worst corner for a given timing constraint may differ on each net, and even for each input on each net. Corner analysis with any practical number of corners may miss some of these errors.

2.2 Previous statistical approaches

The problems with corner analysis are well known. One proposed way around them is statistical timing analysis, as

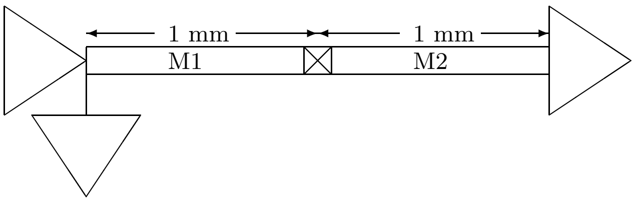

Figure 2: Example

in [4]. This is quite different than the approach proposed here, though the end goal is the same. In statistical timing analysis, probability distributions are propagated through the timing graph. In this work, functions that describes the delay as a function of process variation are propagated. This has two main advantages - correlations between signals that depend on the same process parameter are maintained, and the contributors to the final distribution are available for troubleshooting or optimization.

The work closest in spirit to the current work is [3]. Here they calculate sensitivities (linear as in this work) to three process variables (Vdd, V_T, and a ΔC on each net), and sum them along paths. This model does not look in any detail at interconnect - they simply add a single lumped value to every net in the design. There is no attempt to analyze the layout and find the actual sensitivities and their correlations. Therefore, unlike this work, they cannot hope to replace corner analysis.

3. EXAMPLE AND MOTIVATION

The example shown in Figure 2 illustrates many of the problems induced by process variation. It contains a net with one driver and two receivers. The driver has a 1 Kohm output impedance (R_s = 1 KΩ) and the line is 2 mm long, the first half metal-1, the second metal-2. There are two inputs on the net, one at the driver and one at the far end. Each input has an input C_l of 40 ff. We wish to calculate the delay of this line under variation of 2 parameters, the delta width of layers 1 and 2. For simplicity, we'll use the Elmore delay, and for concreteness we'll use values typical of a 130 nm copper process. Let R_1, C_1 and R_2, C_2 be the resistance and capacitance of the first and second portion of the line. Assuming the nominal line has a width w = 150 nm, a square cross section, and the conductivity of copper, then a wire on metal-1 has resistance

$$R = 746 \frac{w}{w + \Delta w1} \frac{\Omega}{\text{mm}} \tag{1}$$

For the capacitance, we'll assume 180 ff/mm total, half to the layers above and below, and half to the neighbors on the same layer. The half that couples to the neighbors will scale inversely as the space to the neighbor, nominally assumed to be 150 nm away and with the same Δw. A wire segment on metal-1 will then have capacitance

$$C = (90 + 90 \frac{w}{w - \Delta w1}) \frac{\text{ff}}{\text{nm}} \tag{2}$$

with similar expressions for R and C of wires on metal-2. We'll assume the deltas have a gaussian distribution, with a standard deviation of 10% of the width, or 15 nm.

Table 1 shows how the R, C, and delay (RC) of a 1 mm line vary as the line widths vary. The R and C values vary over a much wider range than their product, since they vary in opposite directions. If metal-1 is narrow, for example, R increases while C decreases. The slowest interconnect happens when Δw is negative. In this case the R goes up more than the C comes down, so this generates the longest wire delays. Conversely the fastest interconnect occurs when Δw is positive. The C is higher, but R is lower by more than enough to compensate. In all cases the interconnect delay changes by a much smaller percentage than the change in width. This implies that keeping only a distribution of values for each component will not work - we need to keep track of the correlation.

Table 1: R, C and Timing with process variation

ΔW(nm)	R(ohms)	C(ff)	RC(ps)
0	746	180	134.3
-45	1065(+43%)	159(-12%)	169.3(+26%)
45	574(-23%)	219(+22%)	125.7(-6.4%)

First, look at the worst case delay at the driven end of the net. We show that a traditional two corner analysis is too conservative. The Elmore delay at the output of the driver is

$$d = R_s(2C_l + C_1 + C_2) \tag{3}$$

where C_1 and C_2 are functions of the process parameters. The slowest and fastest corners can be found by maximizing/minimizing C_1 and C_2, since we are not varying R_s in this example. If we set both C_1 and C_2 to their 3σ values, we get a minimum delay of 399 ps and a max of 517 ps. Unless $\Delta w1$ and $\Delta w2$ are completely correlated, however, these values are needlessly conservative. If metal-1 and metal-2 have identical uncorrelated gaussian distributions, for example, the distribution will be $\sqrt{2}$ narrower than the estimate above. This corresponds to a smaller delay range of [416,500] ps, a 29% narrower spread. This also points out that we need some new process data (the degree of correlation between metal-1 and metal-2 widths) that was not needed in the 2 corner approach.

Next, we look at the delay at the end of the line, and show the worst case is missed. This can't be fixed by without adding more process corners since the front of the line, and the end, have different worst corners! The Elmore delay at the end of the line is

$$d = R_s(2C_l + C_1 + C_2) + R_1(\frac{C_1}{2} + C_2 + C_l) + R_2(\frac{C_2}{2} + C_l) \tag{4}$$

If we set $\Delta w1 = \Delta w2 = +45$ nm, the usual fast corner, the result is 813 ps. If we set $\Delta w1 = \Delta w2 = -45$ nm, the usual slow corner, the result is is 822 ps. So can we conclude that for all process conditions the delay must be in the range [813,822] ps? Somewhat surprisingly, the answer is no - this range does not include either the fastest or slowest case! In fact the nominal delay ($\Delta w1 = 0$ and $\Delta w2 = 0$) is 768 ps, well ouside the range. How can this happen? The total delay has two components - the delay of the driver and the delay through the wire. The cell delay increases with increasing C, as observed at the near input. But the wire delay decreases with increasing C, since the larger C means less R, and by more than enough to compensate. The sum of these two delays has a local minimum, which is bracketed by the two "worst case" corners. Furthermore the combination $\Delta w1 = -45$ nm and $\Delta w2 = 45$ nm has a much greater delay, about 903 ps. This happens because wire R is the dominant contributor to the delay near the driver, but wire C dominates the delay near the end.

3.1 Motivation

How can we deal with this? Here's one possible solution. First, we must ask the extractor to generate not only the nominal values, but the way they change with process parameters (a derivative, for example, can express this change to first order). For this example we will do this from the analytic expressions (1) and (2). Then we can calculate how the delays will change with process parameters. For example, for the near end load, use equation (3) to find:

$$\frac{\partial d}{\partial w1} = \frac{\partial d}{\partial C_1}\frac{\partial C_1}{\partial w1} = R_s\frac{\partial C_1}{\partial w1} = +0.600 \text{ ps/nm}$$

and likewise for the delay as a function of $\Delta w2$:

$$\frac{\partial d}{\partial w2} = \frac{\partial d}{\partial C_2}\frac{\partial C_2}{\partial w1} = R_s\frac{\partial C_2}{\partial w2} = +0.600 \text{ ps/nm}$$

This confirms that maximizing the widths maximizes the delays, and we can use the approximation

$$d = d_{NOMINAL} + \frac{\partial d}{\partial w1}\Delta w1 + \frac{\partial d}{\partial w2}\Delta w2 \qquad (5)$$

to compute the distribution of delays from the distributions of widths $w1$ and $w2$, and hence eliminate the pessimism of two corner analysis.

The far end delay has a more complex expression, but the principle is the same. Both R_1 and C_1 are a function of $w1$, so we need to use the chain rule on equation (4) to get

$$\begin{aligned}\frac{\partial d}{\partial w1} &= \frac{\partial d}{\partial C_1}\frac{\partial C_1}{\partial w1} + \frac{\partial d}{\partial R_1}\frac{\partial R_1}{\partial w1} \\ &= (R_s + R_1/2)\frac{\partial C_1}{\partial w1} + (C_1/2 + C_2 + C_l)\frac{\partial R_1}{\partial w1} \\ &= -0.716 \text{ ps/nm}\end{aligned}$$

A similar calculation for delay as a function of $w2$ yields

$$\frac{\partial d}{\partial w2} = +0.402 \text{ ps/nm}$$

From the signs of the derivatives it's clear the longest delay will be with $\Delta w1$ negative and $\Delta w2$ positive, so the correct worst case corner is identified.

The advantages of this approach are

- You can use the derivatives to get a first order approximation of the delay at any process conditions.
- It's easy to find the worst case corners (from the signs of the derivatives)
- You can plug in a distribution for process parameters and easily generate the resulting distribution of delays. No assumptions are needed on the shapes of the distributions - we do not need to assume they are gaussian.
- These delays can be added, subtracted, and compared without losing any correlations that may exist.

4. KEEPING DERIVATIVES

As seen in the example above, the basic idea is that each quantity of interest (such as a capacitance or a delay) is described by a nominal value and a description of how it changes with each process variable. To first order, this change in value is just a vector of derivatives, with one entry for each process variable. There are several ways this vector of derivatives can be computed; first we show how this data could be used if we had it.

As an example, look at timing analysis. We wish to show that the data signal of a flip-flop arrives neither too late (a *setup* violation) nor too early (a *hold time* violation) over all process conditions.

First, we note that the arrival time of a data signal will be a continuous function of the process variations $\vec{P}$, with components p_1 through p_N:

$$A(\vec{P}) = A(p_1, p_2, \cdots, p_n)$$

We first approximate this with a Taylor series expansion around the nominal process conditions, $p_1^{nom}, \cdots, p_N^{nom}$:

$$A(p_1, p_2, \cdots, p_n) \approx A(p_1^{nom}, \cdots, p_N^{nom}) + \sum_1^N a_i \; \Delta p_{i,a}$$

where $\Delta p_{i,a}$ is the deviation of p_i from the nominal process point for the net 'A'. Since $A(p_1^{nom}, \cdots, p_N^{nom})$ is the normal arrival time under nominal process conditions, we will write this as A_{nom} and get

$$A(p_1, p_2, \cdots, p_n) \approx A_{nom} + \sum_1^N a_i \; \Delta p_{i,a} \qquad (6)$$

Now, what is the deviation from nominal of the process parameters for net A? This is composed of three parts - a deterministic portion which we assume has already been accounted for, a global portion, G_i, and a statistical portion $S_{i,a}$.

$$p_{i,a} = G_i + S_{i,a}$$

In this case, G_i is the global deviation of the ith process parameter (i.e., metal is thick on this chip), and $S_{i,a}$ is the statistical deviation of parameter i for net A.

The clock arrival time has a similar approximation:

$$C(p_1, p_2, \cdots, p_n) \approx C_{nom} + \sum_1^N c_i \; \Delta p_{i,c}$$

The coefficients c_i and a_i will (in general) have different values depending on the layer of the routes, relative amounts of wire delay and cell delay in the path, and so on.

4.1 Relative times

Except for primary inputs and outputs, the designer does not care about the arrival time of each signal independently, but only the relative delay between various signals on the chip. In particular, the data must arrive at specified times with respect to the clock on the receiving element. We can subtract the clock time from the data time to get the difference:

$$\begin{aligned}A(\vec{P}) - C(\vec{P}) &= A_{nom} - C_{nom} + \underbrace{\sum_1^N (a_i - c_i)\; \Delta G_i}_{global} \\ &\quad + \underbrace{\sum_1^N (a_i S_{i,a} \ominus c_i S_{i,c})}_{statistical}\end{aligned}$$

Where the $\ominus$ indicates the difference is statistical. For a given net pair, the second sum evaluates to a single random variable with a distribution that depends on the similarity of the nets. Two adjacent lines on the same layer will yield a very narrow distribution, but two lines that are far apart will have a larger difference. [1]

For the setup check we want to ensure this difference is in the legal range, not too long (violates performance spec) nor too small (violates hold spec). This translates into

$$A(\vec{P}) - C(\vec{P}) < T_{tmax}$$

and

$$A(\vec{P}) - C(\vec{P}) > T_{hold}$$

Each of these conditions, for each flip-flop, defines a fuzzy hyper-plane in process space. On one side if the hyperplane the chip will (probably) work, and on the other side it will (probably) not. The thickness of the fuzzyness is determined by the random component of process variation - since this component will usually be small compared to the global effects, the hyperplane (in general) is not very fuzzy. Furthermore, the more closely matched the clock and data are in terms of location, layer assignment, wire widths, and so on, the more tightly correlated the signals will be, and the less fuzzy the hyperplane.

Since the worst conditions may (and probably will) be different for each flip-flop on the chip, we end up with at least twice as many hyperplanes as flip-flops (one setup and one hold constraint from each). Taken together, all the hyperplanes from all the flip-flops determine an N_D dimensional convex polytope.

The chip works provided its process condition is inside this polytope, so the polytope determines the parametric yield. This is the integral over process space of (probability of this process condition) times (probability chip works under this process condition). The point on the surface of the polytope that is closest to the origin will define the most likely condition under which the chip will fail.

5. APPLICATIONS

The main result of this technique should be less conservative timing numbers, since we are no longer assuming 3σ variations on all parameters. It will also find some errors that are currently missed.

We can show that this technique always gives a better (less pessimistic) result than corner analysis. We start with a simple case where the delay depends only on 2 process parameters.

$$Delay = D_0 + K_1 \Delta P_1 + K_2 \Delta P_2$$

where ΔP_1 and ΔP_2 have 0 mean and standard deviations σ_1 and σ_2. Then the 3σ corner case will have delay

$$D_0 + 3K_1\sigma_1 + 3K_2\sigma_2$$

Assuming P_1 and P_2 are independent, we can compute the distribution of delay as a standard distribution with a mean of D_0 and a standard deviation of

$$\sigma = \sqrt{(K_1\sigma_1)^2 + (K_2\sigma_2)^2}$$

So the true 3σ value will be

$$D_0 + 3\sqrt{(K_1\sigma_1)^2 + (K_2\sigma_2)^2}$$

or equivalently

$$D_0 + \sqrt{(3K_1\sigma_1)^2 + (3K_2\sigma_2)^2}$$

This is always an improvement since

$$\sqrt{(3K_1\sigma_1)^2 + (3K_2\sigma_2)^2} \leq 3K_1\sigma_1 + 3K_2\sigma_2$$

by the triangle inequality - the left side is the length of the hypotenuse of a right triangle with sides $3K_1\sigma_1$ and $3K_2\sigma_2$. Equality holds only in the case where all the K_i are zero except one. In practice, however, each observable depends one more than one process parameter, and the inequality is strict.

The same argument generalizes to more dimensions, and in general, the more parameters an observable depends on, the bigger the improvement. If an observable depends on N parameters, then improvement will range from 0 to $1 - 1/\sqrt{N}$, depending on the correlation of the parameters. For two parameters, the maximum improvement is about 29%, for 3 parameters 42%, and so on. Because of this effect, path delays will in general have a bigger percentage improvement than the individual stage delays, since they will depend on more parameters (barring fortuitous parameter cancellation).

Also, with this technique we can compute the yield due to timing effects. Start with the paths in order of their nominal timing, longest paths first. Find the odds the first path fails, then find the odds that the second path fails subject to the condition that the first is OK. Find the odds that the third fails subject to the condition that the first two are OK, and so on. Terminate when each new path is adding only a negligible probability of failure.

The order in which we consider the paths does not matter except for efficiency, since we are measuring a weighted integral over the volume of a polytope. A different order of paths just corresponds to doing the defining cuts in a different order; the integral is unchanged.

It's also possible to estimate the improvement in parametric yield from removing a critical path. This can be done by removing the hyperplane associated with the path and see how much the (weighted) volume of the polytope increases. This technique could in principle allow the user to trade off a slightly increased chip size versus fewer, less critical, paths.

6. COMPUTING DELAY AS A FUNCTION OF PROCESS VARIATION

How can we generate delays as a function of process parameters, as in Equation 6? There are two basic approaches that could be used. The first we'll call the "experimental" approach. Here we simply run extraction, delay calculation, and timing verification many times, using different values

[1]This makes the implicit assumption that after the deterministic and chip-to-chip errors are removed, the remaining intra-chip variation is purely random and uncorrelated. This is not exactly true, but the penalty for this simplification will be small if the intra-chip variation is much less than to the inter-chip variation. What if this assumption proves false as processes shrink? Many of the most significant forms intra-chip variation can be handled by increasing the number of process parameters. From[8], most of the intra-chip interconnect variation can be expressed as a gradient. Instead of metal-2 thickness, therefore, we might have metal-2 thickness, metal-2 X gradient, and metal-2 Y gradient. Then correlations caused by the gradients will be taken into account correctly, at the cost of more complex calculations and (likely) less efficient pruning in timing analysis.

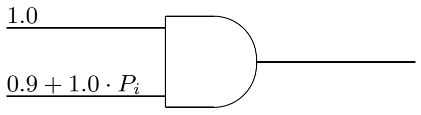

Figure 3: Difficult case for experimental approach

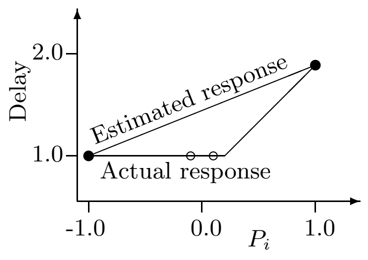

Figure 4: Resulting timing

of the P_i, and try to deduce the coefficients A_i in equation 6. Alternatively, we can try the "computational" approach. Here we start with an extractor that computes not only nominal component values, but how they change with process variation. From this data, we can compute net delays, and then timing, as a function of the process parameters.

6.1 The experimental approach

The most straightforward way to find timing as a function of process variation is to simply vary one p_i at a time, then re-run coefficient generation, extraction, delay calculation, and timing analysis. The advantage is that no changes to these programs are required. The main disadvantages are massive CPU and data requirements, and possible inaccuracy due to the non-differentiable nature of timing analysis.

Assuming the coefficient generation has already been done ahead of time, each analysis takes about $2N_D$ times the effort of a single extraction through timing run. Although time consuming, this technique at least gives good results for extraction and delay calculation, where the results are continuous and differentiable functions of the parameters. It does not work as well for timing analysis, since the answers here contain min() and max() functions, and hence are not differentiable. For example, suppose we have the gate of Figure 3, where the specified arrival times are a function of process variable P_i. Suppose that P_i varies from -1.0 to 1.0. If we use a small variation in P_i in our experiment (shown by the open circles in Figure 4), we will conclude that P_i has no effect on the output time. If we use the whole range, as shown by the filled circles, we will conclude the time varies from 1.0 at minimum P_i to 1.9 at the maximum. This is correct as it stands, but interpolation between the two data points gives a wrong estimated response.

6.2 The computational approach

A different approach is to keep the variation with process parameters through each step of the process. This is potentially both more efficient and more accurate, but requires changes to many tools and data formats.

Coefficient generation can keep the variation of coefficients with process variables. This may be slow to derive but is only done once per process. Then extraction can compute the variation of the Rs and Cs with process variables. Next, delay calculation can compute the interconnect and gate delays as a function of process. Finally, the timing verifier can sum these delays to get the final arrival times, and then compare against constraints. Each step is more complex, but it only takes one run to get the full performance over all process variations.

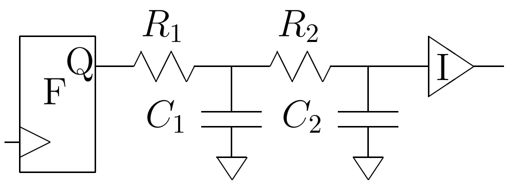

Figure 5: Example for chain rule analysis

Adding or subtracting two variables with derivatives is straightforward - just operate component by component. More complex operations require the *chain rule.* This lets us compute the derivative of a function whose inputs are also changing. For example, the delay $Delay$ of a gate depends on the process conditions $\vec{P}$, and also on the input slope I and output load C, which are also functions of $\vec{P}$. We can write

$$Delay(\vec{P}) = D(\vec{P}, I(\vec{P}), C(\vec{P})\ )$$

where D gives the delay for a particular $\vec{P}$, I, and C. From this we can compute the derivatives

$$\frac{\partial Delay}{\partial P_i} = \frac{\partial D}{\partial P_i} + \frac{\partial D}{\partial I}\frac{\partial I}{\partial P_i} + \frac{\partial D}{\partial C}\frac{\partial C}{\partial P_i}$$

Figure 5 gives an example of how this works. In this diagram, the values of all components are functions of process variation. For example, in the diagram, we write R_1, but we really mean $R_1(\vec{P})$. The first step is to calculate the delay at pin Q of the flip-flop as a function of process variation. Presuming we have characterized the delay at Q as a function of $\vec{P}$, we need to find the effective C value of the network (which will also be a function of $\vec{P}$). If C_{in} is the input capacitance of instance I, and DP the driver parameters of F (both as a function of $\vec{P}$), then the driver load C_D is

$$C_D = C_{Eff}(DP, R_1, C_1, R_2, C_2, C_{in})$$

Using the chain rule, we can calculate the dependence of C_D on $\vec{P}$:

$$\begin{aligned}\frac{\partial C_D}{\partial P_i} &= \frac{\partial C_{eff}}{\partial DP}\frac{\partial DP}{\partial P_i} + \frac{\partial C_{eff}}{\partial R_1}\frac{\partial R_1}{\partial P_i} + \frac{\partial C_{eff}}{\partial C_1}\frac{\partial C_1}{\partial P_i} \\ &\quad + \frac{\partial C_{eff}}{\partial R_2}\frac{\partial R_2}{\partial P_i} + \frac{\partial C_{eff}}{\partial C_2}\frac{\partial C_2}{\partial P_i} + \frac{\partial C_{eff}}{\partial C_{in}}\frac{\partial C_{in}}{\partial P_i}\end{aligned}$$

Now we can find the delay D at the output of flip-flop F, as a function of process variation, by using the chain rule again:

$$\frac{\partial D}{\partial P_i} = \frac{\partial D_{ff}}{\partial P_i} + \frac{\partial D}{\partial C_{eff}} \cdot \frac{\partial C_{eff}}{\partial P_i}$$

The slope S at the Q output of F, as a function of $\vec{P}$, is computed in a similar manner. Then we find the delay and slope at the input of I ([5] concentrates on this step), and then the delay and slope at the output of I. We continue in the manner until we come to the timing sinks.

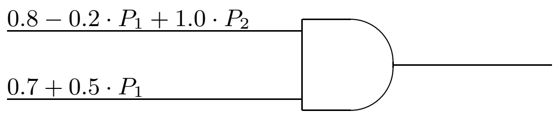

Figure 6: Two hyperplanes needed

6.2.1 Propagating functions

Propagating delays with process variation through multi-input gates requires special care. For example, consider a 2 input AND gate. The output delay is determined by the later arriving of the two inputs. It is possible that the last arriving input depends on the process variation, as shown in Figure 6 where P_1 and P_2 have ranges $[-1, 1]$. (Here we will assume we are using a first order approximation to the real process variation, and hence each response surface is a hyperplane.)

Attempting to handle this exactly may result in a combinatorial explosion. Unless the two inputs have identical process derivatives (very unlikely) there will be some conditions under which each input arrives last, so the number of hyperplanes needed to describe the output timing at least doubles with every multi-input gate. To prevent a data explosion, the timing verifier must prune as it goes.

Some conditions for dropping hyperplanes are obvious. For each input, there is a corresponding inequality that must be satisfied when this input is the last to arrive. If this inequality cannot be satisfied (no solution within the maximum expected process variation) then the corresponding hyperplane can be dropped. A slightly more complex calculation can also discard hyperplanes that are possible but of sufficiently low probability.

Beyond this, there is a wide variety of pruning and approximation strategies that could be used. In general, the more aggressive the pruning, the less data growth, but the more pessimism that is introduced. As an example of an aggressive pruning strategy, we can always express the output timing as a single hyperplane that dominates the hyperplane derived from each input. (This is the moral equivalent of the MAX() function in normal timing analysis.) It's pessimistic for most process conditions, but preserves at least some process variability and prevents data explosion. In the example above we could replace both input hyperplanes by $1.6 - 0.2 \cdot P_1 + 0.2 \cdot P_2$ if we are concerned with maximum delays.

An exactly analogous problem happens when propagating required times backward through the network, except it is caused by nets with multiple fanouts instead of gates with multiple fan-ins. The solutions are exactly the same - we discard the conditions that cannot happen, and devise a pruning strategy to cope with the rest of the conditions.

6.3 Memory and computation needs

Won't keeping a vector of derivatives for every extracted component take an enormous amount of memory? If we use data compression, perhaps not. At first glance, it looks like we need 30 or so extra floating point values for each component, to represent the derivatives. However, we can normalize each derivative, perhaps as a percentage change of value from a percentage change of a process variable. For example, a 1% change in width might result in a -0.8% change in R value. In this format, most if not all the values will be between -1 and 1. These can be converted to fixed point, and stored in a byte to 1% accuracy, probably sufficient for this application. Then we need only 30 extra bytes for each component, a factor of 4 better.

We can do still better, though, by dropping the terms with near 0 values. The properties of a wire on layer metal-2, for example, will depend strongly on only 4 process parameters, metal-2 thickness and width, and the thickness of the two surrounding interlayer dielectrics. If we assume all the other variations are negligible, we could represent this as 6 bytes - a start index, a stop index, and 4 values. This is only 6 bytes extra per component - an eminently practical amount of data. For reduced networks this particular compression won't work as well, since a given component in the reduced form may have contributions from many layers, but there are many fewer components in the reduced form.

The time tradeoff is currently unclear. The individual calculations are more complex, but there will be many fewer runs (6 times fewer than with 6 corner analysis, for example). In theory, we can simply replace every add, subtract, multiply, divide, and function call with a more complex operation that keeps first derivatives. If there are N process variables, then each addition and subtraction will require N operations. A multiply requires $2N$ scalar multiplies and N additions. A divide requires N multiplies, $2N$ divides, and a N subtractions. Each function call (such as sin(), ln(), or sqrt()) requires an underlying derivative calculation, N multiplies, and N additions. These numbers can probably improved by taking advantage of sparseness.

Thus if we assume the floating point computations themselves take 10% of the run time of a timing verifier, and we have 20 to 30 process variables, then we would expect the timing analysis portion of a program to slow down by a factor of 2 or 3. This is a extremely crude estimate and needs to be confirmed by experiment.

7. EXPERIMENTAL RESULTS

Here we describe experiments designed to test some of the key assumptions of the calculations described above. We used the "experimental" approach above since it can be done with off the shelf programs. First, we checked to see if net capacitances are really a linear function of the process variations. Next, we quantified the improvement from using a statistical sum rather than 3σ corners. We used total net C for this measurement but have reason to believe this will extend to timing as well. Finally, for every net C in one design, we checked to see how many process parameters had a significant effect on the value. This helps indicate how well data compression will work.

7.1 Test of linearity

This technique relies on the variations of component values and timing being near linear in process variations. The effect of the process variations are not really independent, though, so the "hyperplanes" are really curved hypersurfaces. For the interconnect this effect should be small. Both capacitance and resistance can be approximated by functions of the form $1/d$, where d is the relevant dimension. Over a $\pm 20\%$ range, though, $1/d \approx 1.021 - 1.035d$ with an RMS error of only 1.7%. So we'd expect the linear approximation to hold within a few percent.

Our first experiment tests this linear approximation. Starting with a nominal 0.25μ process, the test cases changed

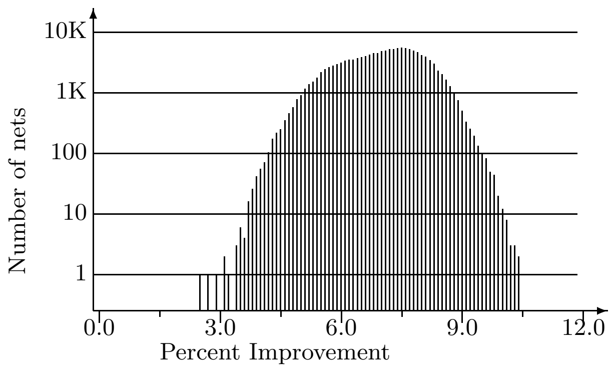

Figure 7: Histogram of Improvement

three physical parameters - width and thickness of M2, and the M1-M2 inter-layer spacing. The three parameters were changed by 0 and ±20 % in all combinations (27 cases) plus a few more combinations with 10% variation for a total of 35 cases in all. For each case, *coeffgen* and *HyperExtract* [1] were used to find the total C for each signal in a 144,284 net test case. Individual net capacitances varied by as much as 27% between the cases. For every net, a least-squares fit tried to model all 35 extracted values as a nominal value plus 3 derivatives. Of the 144,284 nets, the biggest deviation from linear was $[-0.75, +0.82]\%$. Since total C is the dominant component of delay for short wires, and since critical paths have mostly short wires, there is reason to hope that the most critical delays will be near-linear functions of the process parameters. Similar experiments (with a different set of process variations) were reported in [3], where they also found near-linear variation.

7.2 Test of amount of improvement

Next, we calculated the improvement in worst case C by varying each of 17 process parameters one at a time, to find the dependence of each C on each process variable. Assuming the process variables were uncorrelated, we then computed the statistical distribution of each C and compared the 3σ value to the value computed at the 3σ worst case corner. The histogram of these results is shown in Figure 7. As expected, no nets were worse, and some nets improved as much as 10.4%. The average improvement was 7.01%. Path delays should show a bigger improvement than individual capacitances, from the argument of section 5. Therefore a total improvement on the order of 10% seems likely, although this would need to be confirmed by experiment.

7.3 Sparseness of derivatives

These experiments also confirmed the sparseness of the derivatives. On the average, only 8.75 of the 17 process parameters had more than a 1% effect on each total net capacitance. At the 3% level only 5.84 parameters had an effect. This was true even though the variable considered was lumped C, which will (on the average) have contributions from many layers. A distributed model, and inclusion of R, would presumably reduce these numbers even further.

8. LIMITATIONS OF THIS RESEARCH

Due to experimental limitations, only the variation of total net capacitance was checked for linearity. A full variational timing analysis would need to include cell delays, too. The variations of these cell delays as a function of process variation (and voltage and temperature) may not be as linear as those of capacitance. Furthermore, the experiments did not look at the effects of net R, which we know to be significant in some cases.

The experiments only varied the process parameters over a ±20% range. Real process data might require a larger range and hence be more non-linear.

Also, although the final results (delays and slopes) are surely near-linear functions of the process parameters, this may not be true of intermediate forms such as moments, poles, and zeros. New delay calculation algorithms such as [5] that can propagate derivatives may be required.

There is a question of whether even the foundries measure and/or keep the correlation data needed to take full advantage of this algorithm. There is also a question of whether the correlations are stable, or constantly change as the process is tweaked.

9. CONCLUSIONS

More accurate timing through a detailed treatment of statistical variation is most likely possible - the assumptions seem reasonable, and neither the memory or processing time seems excessive. To do this, we would need new versions of extraction, delay calculation, and timing analysis. The payoff will be less pessimistic timing numbers, new possible tradeoffs involving timing yield, and fewer missed errors.

10. REFERENCES

[1] Cadence. *HyperExtract Parasitic Extractor User Guide.*

[2] W. Dai and J. Hao. Timing analysis taking into account interconnect process variation. In *IEEE International Workshop on Statistical Methodology*, pages 51–53, 2001.

[3] A. Gattiker, S. Nassif, R. Dinakar, and C. Long. Timing yield estimation from static timing analysis. In *International Symposium on Quality Electronic Design*, pages 437 – 442, 2001.

[4] H.-F. Jyu, S. Malik, S. Devadas, and K. Keutzer. Statistical timing analysis of combinational logic circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1(2):126 – 137, June 1993.

[5] Y. Liu, L. T. Pileggi, and A. J. Strojwas. Model order-reduction of rc(l) interconnect including variational analysis. In *Proc. 36th Design Automation Conference*, pages 201–206, 1999.

[6] V. Mehrotra. *Modeling the Effects of Systematic Process Variation on Circuit Performance.* PhD thesis, MIT, 2001.

[7] S. Nassif. Within-chip variability analysis. In *International Electron Devices Meeting Technical Digest*, pages 283–286, 1998.

[8] B. Stine, D. Boning, and J. Chung. Analysis and decomposition of spatial variation in integrated circuit processes and devices. *IEEE Transactions on Semiconductor Manufacturing*, 10(1):24–41, Feb 1997.

[9] T.-H. Yeh, J. Lin, S.-C. Wong, H. Huang, and J. Sun. Mis-match characterization of 1.8 v and 3.3 v devices in 0.18 micron mixed signal cmos technology. In *2001 Microelectronics Test Structures*, pages 77–82, 2001.