Chapter 2
Fast PVT Verification and Design

Efficiently Managing Process-Voltage-Temperature Corners

Abstract This chapter explores how to design circuits under PVT variation effects, as opposed to statistical process variation effects. Process, voltage, and temperature (PVT) variations are taken into account by individually varying P, V, and T over their allowable ranges and analyzing the subsequent combinations or so-called PVT corners. In modern designs, there can be hundreds or thousands of PVT corners. This chapter reviews design flows to handle PVT variations, and compares them in terms of relative speed and accuracy. It introduces a "Fast PVT" flow and shows how that flow has excellent speed and accuracy characteristics. It describes the Fast PVT algorithm, which is designed to quickly extract the most relevant PVT corners. These corners can be used within a fast and accurate iterative design loop. Furthermore, Fast PVT reliably verifies designs, on average 5x faster than the method of testing all corners on a suite of benchmark circuits. This chapter concludes with design examples based on the production use of Fast PVT technology by industrial circuit designers.

2.1 Introduction

PVT effects are modeled as a set of corners. A PVT corner has a value for each PVT variable. For example, a single PVT corner might have a modelset value of FF, a supply voltage of 1.2 V, a temperature of 25 °C, and power setting of *standby*. A set of PVT corners is created by enumerating the complete combinatorial set of variations. For example, if there are 5 values of modelset, 3 values of voltage, 3 values of temperature, and 4 power modes, there would be a total of $5 \times 3 \times 3 \times 4 = 180$ combinations in the PVT corner set.

There are some cases in custom design where a PVT approach to variation may be used instead of a statistical approach:

T. McConaghy et al., *Variation-Aware Design of Custom Integrated Circuits: A Hands-on Field Guide*, DOI: 10.1007/978-1-4614-2269-3_2,

- Local process variation (mismatch) has negligible effect on the circuit's performance, and modelset corners (FF/SS) accurately bound global process variation's effect on circuit performance. This is often the case with digital standard cells, especially for older process technologies.
- Sufficiently accurate models of statistical process variation are not available. This is typically the case for older process technologies.
- PVT corners have tolerable accuracy in bounding performance, and statistical analysis is deemed too simulation-intensive to complete. For example, many signoff flows mandate a thorough PVT analysis.
- Local process variation affects performance or modelset corners do not bound performance, statistical models are available, but there is no clear statistically-based design methodology. This is sometimes the case with analog design: designers know that FF/SS corners are inaccurate, but do not have a fast statistically-based design flow. If this is the case, a PVT-based flow is *not* appropriate. This book is here to help: the chapters that follow this chapter describe flows to do fast-yet-accurate *statistically*-aware design.

The aim in PVT analysis is to find the worst-case performance values across all PVT corners, and the associated PVT corner that gives the worst-case performance. This is done for each output. In PVT-aware design, the aim is to find a design that maximizes performance or meets specifications across all PVT corners.

Traditionally, it has only been necessary to simulate a handful of corners to achieve coverage: with FF and SS process (P) corners, plus extreme values for voltage (V) and temperature (T), all combinations would mean $2^3 = 8$ possible corners.

With modern process nodes, many more process corners are often needed to properly bracket process variation across different device types. Here, "to bracket" means to find two corners for each spec, one that returns the maximum value and another that returns the minimum value of the respective performance output. Furthermore, transistors are smaller, performance margins are smaller, voltages are lower, and there may be multiple supply voltages. To bracket these variations, more variables with more values per variable are needed.

Adding more PVT variables, or more values per variable, quickly leads to a large number of corners. Even a basic analysis with 4 device types (e.g. NMOS, PMOS, resistor, capacitor) and 4 other variables (e.g. temperature, voltage, bias, load) with 3 values each results in $3^{(4+4)} = 6{,}561$ corners.

The problem gets worse at advanced nodes that have double patterning lithography (DPL), where the RC parasitics among the masks degrade performance. To account for this, a tactic is to treat the bounds on RC parasitic variation as corners by extracting different netlists that represent each possible extreme. This results in a 10–15x increase in the number of corners.

Power verification needs corners for each power mode (e.g. quick boot, cruising, read, write, turbo, and standby), which also increases the total number of corners.

The problem is that simulating each corner can take several seconds, minutes, or even hours for longer analyses. To simulate all possible corners could take hours or even days, which is too time-consuming for most development schedules. Designers may try to cope with this limitation by guessing which corners cause the worst-case performance, but that approach is risky: a wrong guess could mean that the design has not accounted for the true worst-case, leading to a failure in testing followed by a re-spin, or worse, failure in the field.

The technique of finding the worst-case PVT corners is most effectively employed in the context of the design loop, where the engineer is changing circuit design variables such as transistor width (W) and length (L) in order to find the design with the best performance under the worst-case PVT conditions.

What is therefore required is a rapid, reliable methodology to quickly identify the worst-case PVT corners when there are hundreds, thousands, or even tens of thousands of possible corners. We define these attributes as follows:

- ***Rapid***: Runs fast enough to facilitate both iterative design and verification within production timelines.
- ***Reliable***: Finds the worst-case corners with high confidence.

These attributes must be met in the context of a practical flow for PVT-aware design.

This chapter first reviews various design flows to handle PVT variation, including a flow using a Fast PVT corner approach. Next, it considers possible algorithms to implement a Fast PVT corner approach, including a global optimization-based approach that delivers on the rapid and reliable attributes. Finally, it presents results of the chosen approach on real-world production designs. This chapter's appendix provides more details on the global optimization-based Fast PVT approach.

2.2 Review of Flows to Handle PVT Variation

This subsection reviews various flows for handling PVT variation that a designer might consider. These flows include simulating all combinations, guessing the worst-case PVT corners, and a new approach that uses Fast PVT capabilities. This section compares the flows in terms of speed and accuracy.

To illustrate each flow, we provide an estimated number of simulations and design time for a representative real-world circuit. The real-world circuit is the VCO of a PLL on a 28 nm TSMC process technology. It has two output performance measures with specifications of: $48.3 < \text{duty cycle} < 51.7$ %, and $3 < \text{Gain} < 4.4$ GHz/V. Its variables are *temperature*, $V_{ah,vdd}$, $V_{a,vdd}$, $V_{d,vdd}$, and model set (any one of 15 possible sets). All combinations of all values of variables leads to 3375 corners. Since there are two output performance measures, and each output has a lower and an upper bound, there are up to 4 PVT corners that cause worst-case performances. We used a popular commercial simulator.

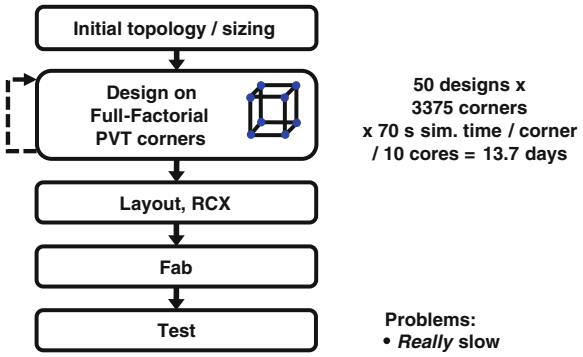

Fig. 2.1 PVT flow: full factorial

Because the flows include changing the design variables, we need a way to compare different approaches in a fair fashion, independent of designer skill and level of knowledge the designer has about the circuit. To do this, we use a simple assumption that in the design loop, the designer considers 50 designs. In our comparisons, we consider time spent with simulations. We do not consider the time spent modifying the design. Also, since multi-core and multi-machine parallel processing is commonplace, for each approach we assume that there are 10 cores running in parallel as our computing resource.

2.2.1 PVT Flow: Full Factorial

Full factorial is the simplest of all flows, shown in Fig. 2.1. In this flow, the designer simply simulates all possible PVT corners at each design iteration. It is comprehensive, and therefore perfectly accurate, to the extent that PVT variation is an accurate model of variation. However, since each of 50 design iterations takes 3375 simulations, it is very slow, requiring 13.7 days of simulation time even when using 10 parallel cores.

2.2.2 PVT Flow: Guess Worst-Case

Figure 2.2 illustrates this flow. After the topology and initial sizing are chosen, the designer uses expertise and experience to guess which PVT corners are likely to cause worst-case performances, without any simulations. Then, the designer simply designs against those corners.

The advantage of this approach is its speed, as it requires no simulations to select corners, and each design iteration only requires simulating the selected corners, which by the designer's reckoning represent respectively the upper and lower specification for each of two outputs. The disadvantages of this method are

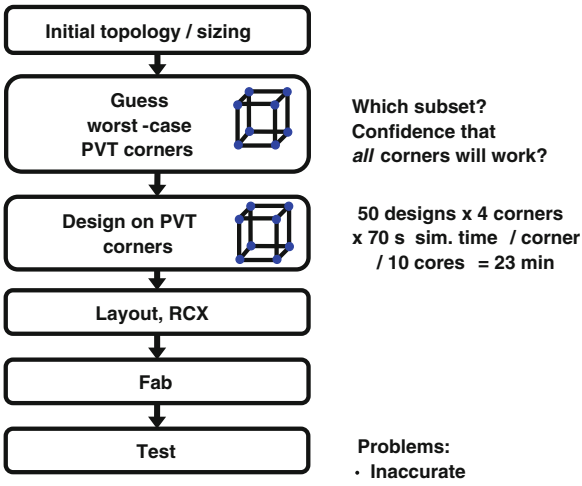

Fig. 2.2 PVT flow: guess worst-case

poor accuracy and reliability. If the designer's worst-case PVT guess corner is wrong, then the known worst-case performance is optimistic. This can be a big issue. For example in power verification, it could mean that the circuit may not meet the required power budget, which in turn translates into poor battery life on the mobile device it is built into.

2.2.3 PVT Flow: Guess Worst-Case + Full Verification

Figure 2.3 shows this flow. It is similar to the previous flow, but adds a step of running all combinations of PVT corners (full factorial) after the design step. This overcomes a key oversight of the previous flow, ensuring that the circuit is verified to the target specs. However, it is possible that the additional verification step finds new worst-case corners that make the design fail specs. To pass at these new PVT corners, the design must be improved at the new corners and verified again. This requires more simulations.

Overall, the flow is more accurate than before, but the full factorial steps are quite slow, resulting in a long overall runtime (14.2 h on our example circuit). Also, if extra design iterations are needed, it could also add substantially to the overall design cycle time, and the long full verification would need to be repeated.

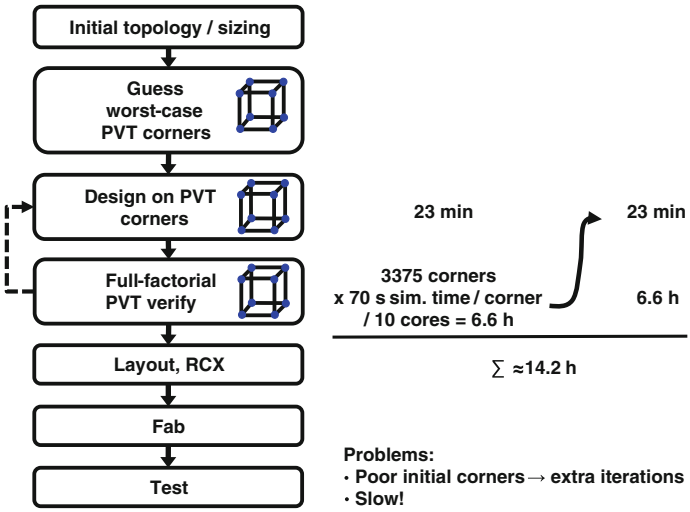

Fig. 2.3 PVT flow: guess worst-case + full verification

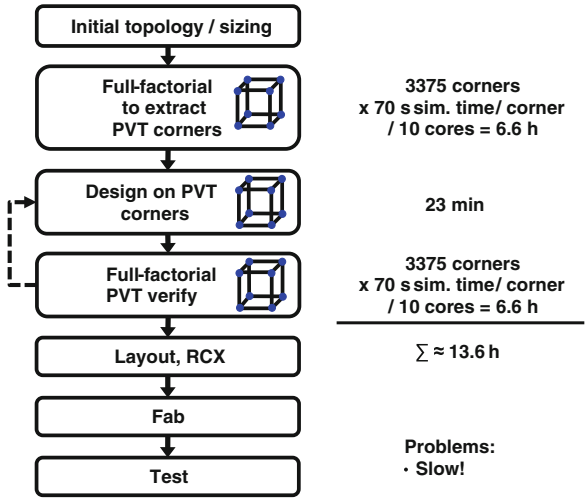

Fig. 2.4 PVT flow: good initial corners + full verification

2.2.4 PVT Flow: Good Initial Corners + Full Verification

Figure 2.4 illustrates this flow. This flow addresses the issue of the previous flow, which is that incorrectly selecting worst-case corners initially can lead to additional design and verification iterations.

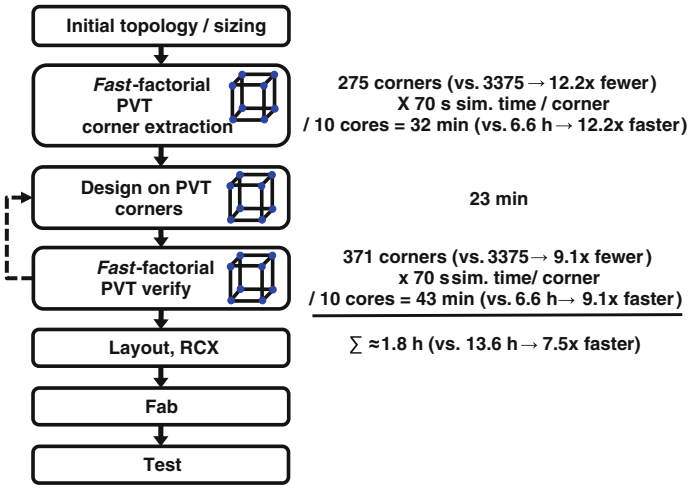

Fig. 2.5 PVT flow: fast factorial PVT (or simply Fast PVT)

The idea with this flow is to simulate all corner combinations at the beginning to identify the correct worst-case PVT corners. Then, the user designs against these corners. Then, in case there are strong interactions between design variables and PVT variables, the user runs full factorial PVT. Usually, PVT corners identified in the beginning will continue to be the worst-case corners; if not, the user designs against the new worst-case corners and verifies again.

Overall, the flow has a comparable runtime and accuracy to the previous flow, but typically only needs one design round because PVT design is done using verified worst-case corners.

This flow is accurate, but still fairly slow because it involves running full factorial PVT twice. On our example circuit, this would still require 13.6 h of simulation time.

2.2.5 PVT Flow: Fast PVT

Figure 2.5 shows this flow. It includes a fast way to handle the two most expensive steps: PVT corner extraction, and PVT verification. It is like the previous flow, but replaces the full factorial PVT steps with *fast factorial* PVT steps.

The flow is as follows. First, the designer completed the initial topology selection and sizing. Then, the designer invokes a fast factorial PVT tool, which we will refer as Fast PVT. Fast PVT efficiently extracts good worst-case PVT corners. The user then designs against these PVT corners. To cover the case where there are strong interactions, he invokes the Fast PVT tool again, verifying that the worst-case PVT corners are found. If needed, he does a second round of design and verification.

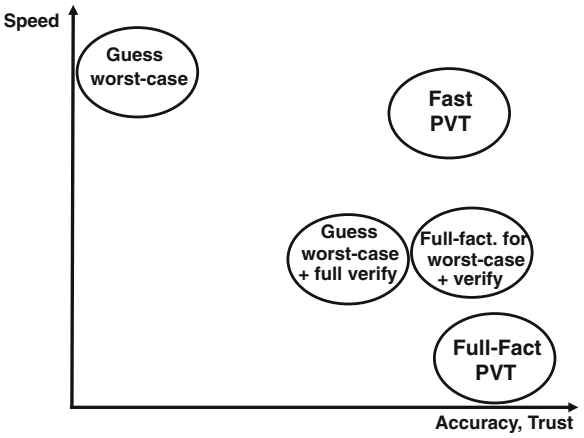

Fig. 2.6 Summary of PVT flows

Figure 2.5, top right, compares the number of simulations needed for corner extraction using full factorial PVT against the number needed using Fast PVT, where Fast PVT uses an algorithm described later in this chapter. Fast PVT finds exactly the same worst-case corners as are found using full factorial PVT. We see that Fast PVT needs 12.2x fewer simulations, and runtime is reduced from 6.6 h to 32 min.

Figure 2.5, center right, compares doing PVT final verification using full factorial with using Fast PVT. Fast PVT again finds exactly the same worst-case corners as full factorial PVT. We see that Fast PVT needs 9.1x fewer simulations, and that runtime is reduced from 6.6 h to 43 min.

The time taken by design iterations is now comparable to the time taken by PVT corner extraction or PVT verification.

In this circuit example, the overall runtime of a Fast PVT-based flow is 1.8 h, which is about 7.5x faster than the 13.6 h required for the previous flow.

We will see later in this chapter how these results are illustrative of Fast PVT's behavior: it will nearly always find the worst-case corners with about 5x fewer simulations than a full-factorial PVT approach.

Subsequent sections of this chapter will review how Fast PVT can be implemented.

2.2.6 Summary of PVT Flows

Figure 2.6 summarizes the different PVT flows described above, comparing them in terms of speed and accuracy. Guessing the worst-case PVT corners without any further verification is the fastest flow, but heavily compromises accuracy. On the flipside, full factorial PVT in the design loop is very accurate but very slow. There are hybrid variants employing full factorial final verification, but replacing full factorial in the loop with corner extraction using initial guessing or full factorial.

However, the full factorial characteristic slows these approaches down. The Fast PVT approach, in the top right, gives the best tradeoff between speed and accuracy, by using a fast factorial approach to extracting corners and to verifying designs across PVT variation.

2.3 Approaches for Fast PVT Corner Extraction and Verification

Fast PVT aims to find the worst-case PVT corners from hundreds or thousands of candidate corners, using as few simulations as possible yet maximizing the designer's chances of finding the truly worst PVT corners. To be suitable for production design, the approach needs to be both rapid and reliable. Engineers and researchers have investigated a number of approaches to do Fast PVT corner extraction. This section summarizes some of the popular approaches and highlights challenges with each method.

2.3.1 Full Factorial

Running all combinations is not fast, but it serves as a baseline for comparison for other methods. As discussed, this can be very time-consuming, taking hours or days. On the positive side, it always returns the true worst-case corners.

2.3.2 Designer Best Guess

Guessing may not produce reliably accurate results, but it serves as a baseline. Here, the designer makes guesses based on experience about what the worst-case corners may be. The advantage of this approach is speed, as guessing requires no simulations. The disadvantage is lack of reliability; a wrong guess can mean failure in testing or in the field. Reliability is strongly dependent on the designer's skill level, familiarity with the design, familiarity with the process, and whether the designer has adequate time to make a qualified guess. In practice, it is difficult to consistently meet all of these goals, which makes a guessing-based approach inherently risky.

2.3.3 Sensitivity Analysis (Orthogonal Sampling, Linear Model)

In this approach, each variable is perturbed one-at-a-time, and the circuit is simulated for each variation. An overall linear response surface model (RSM) is constructed. The worst-case PVT corners are chosen as the ones predicted to give worst-case output values by the linear model. This method is fast, as it only requires $n + 1$ simulations for n variables (the +1 is for typical case). However, reliability is poor: it can easily miss the worst-case corners because it assumes a linear response from PVT variables to output, and assumes that there are no interactions between variables; this is often not the case.

2.3.4 Quadratic Model (Traditional DOE)

In this approach, the first step is to draw $n \times (n\text{-}1)/2$ samples in PVT space using a fractional factorial design of experiments (DOE) (Montgomery 2004), then simulate them. The next step constructs a quadratic response surface model (RSM) from this input/output data. Finally, the worst-case PVT corners are the ones that the quadratic model predicts as worst-case output values. While this approach takes more simulations than the linear approach, it is still relatively fast because the number of input PVT variables n is relatively small. However, reliability may still be poor because circuits may have mappings from PVT variables to output that are more nonlinear than simple quadratic.

2.3.5 Cast as Global Optimization Problem (Fast PVT)

The idea here is to cast PVT corner extraction and PVT verification as an optimization problem: search through the space of candidate PVT corners x, minimizing or maximizing the simulated performance output value $f(x)$. Under that problem specification, the aim is to solve the optimization problem reliably, with as few simulations as possible. The optimization must be *global*: working independently of any initial PVT corner values, and it must not get stuck in local optima. This is a promising idea because it directly addresses the designer task of finding the worst-case PVT corner, and if implemented well, delivers both good accuracy and speed.

Figure 2.7 illustrates how PVT verification can be cast as a global optimization problem. The x-axis represents the possible PVT values, which in this case is just the temperature variable. The y-axis is the performance metric to maximize or minimize, which in this case is the goal to maximize power. The curve is the response of the circuit's power to temperature, found via SPICE simulation. The objective in optimization is to try different x-values, measuring the y-value, and

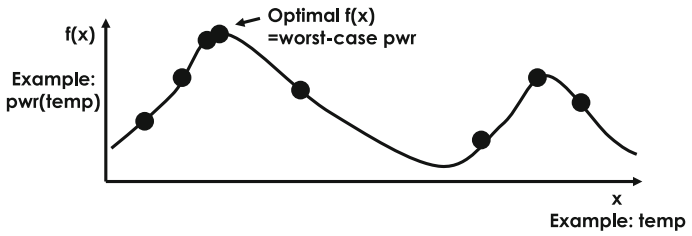

Fig. 2.7 PVT verification cast as a global optimization problem

using feedback to try more x-values, ultimately maximizing the y-value. In this case, different values of temperature are being selected and simulated to find the maximum value of power. The top of the hill in Fig. 2.7 right is a local optimum, as none of the nearby x-values have a higher y-value. We do not want the *Fast PVT* algorithm to get stuck in this local optimum; it should instead find the top of the hill in Fig. 2.7 left, which is the global optimum. In other words, over all possible values of temperature, *Fast PVT* should give the worst-case output performance (in this case, maximum power).

With PVT verification recast as a global optimization problem, we can now consider various global optimization approaches to solve it. Global optimization is an active research area with a long history, spanning techniques with labels like "Branch & Bound" (Land and Doig 1960), "Multi-Coordinate search" (Huyer and Neumaier 1999), and "Model-Building Optimization" (MBO) (Jones et al. 1998). We focus our energy on MBO because it is rapid, reliable, and easy for users to understand. Subsequent sections will validate these claims and describe the approach in general.

In this work, by "Fast PVT", we refer specifically to the solution to the general challenge of finding worst-case corners accurately and efficiently. Fast PVT is then the approach that casts the problem as a global optimization problem, and uses the MBO-based approach to solving the global optimization problem.

2.4 Fast PVT Method

2.4.1 Overview of Fast PVT Approach

The overall approach is to cast PVT corner extraction and PVT verification as a global optimization problem, and solve it using Model-Building Optimization (MBO). MBO-based Fast PVT is rapid because it builds regression models that make maximum use of all simulations so far in order to choose the next round of simulations. MBO uses an advanced modeling approach called Gaussian Process Models (GPMs) (Cressie 1989). GPMs are arbitrarily nonlinear, making no assumptions about the mapping from PVT variables to outputs. Fast PVT is

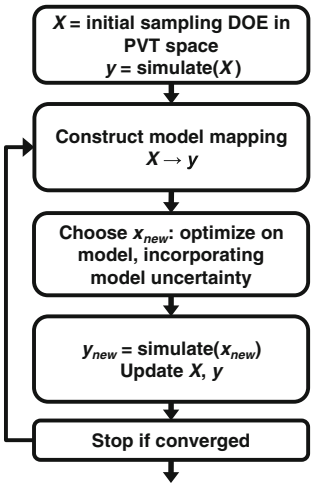

Fig. 2.8 Fast PVT algorithm

reliable because it finds the true worst-case corners, assuming appropriate stopping criteria. Fast PVT is user-friendly and easy to adopt because it lends itself well to visualization, and its algorithmic flow is easy for designers to understand.

As discussed, corner extraction and verification are actually two distinct tasks. Fast PVT has slightly different algorithms, depending on the task:

- ***Corner extraction*** is for finding PVT corners that the designer can subsequently design against. Corner extraction runs an initial design of experiments (DOE), predicts all values using advanced modeling, then simulates the predicted worst-case predicted corners for each output. There is no adaptive component.
- ***Verification*** keeps running where corner extraction would have stopped. It loops to adaptively test candidate worst-case corners while updating the model and improving the predictions of worst-case. It stops when it is confident it has found the worst-case. Verification takes more simulations than corner extraction, but is more accurate in finding the worst-case corners.

Figure 2.8 shows the Fast PVT algorithm. Both Fast PVT corner extraction and verification start by drawing a set of initial samples, *X* (i.e. corners), then simulating them, *y*. A model mapping X → y is constructed.

After that, corner extraction simply returns the predicted worst-case points. Verification proceeds by iteratively choosing new samples via advanced modeling, and then simulating the new samples. It repeats the modeling/simulating loop until the model predicts, with 95 % confidence, that worse output values are no longer possible. When choosing new samples, it accounts for both the model's prediction of the worst-case, as well as the model's uncertainty (to account for model blind spots).

Fast PVT is rapid because it simulates just a small fraction of all possible PVT corners. It is reliable because it does not make assumptions about the mapping from PVT variables to outputs, and explicitly tracks modeling error. Later in this

paper, results on several benchmark circuits will further validate the technique's speed and reliability.

For a detailed explanation of the steps in Fig. 2.8, and theoretical details, we refer the reader to Appendix A. For further details on the GPM modeling approach, we refer the reader to Appendix B.

2.5 Fast PVT Verification: Benchmark Results

2.5.1 Experimental Setup

This section catalogs Fast PVT verification benchmark results on a suite of problems: 13 circuits with a total of 118 outputs, based on industry applications and PVT settings. The circuits include a shift register, two-stage bucket charge pump, two-stage opamp, sense amp, second-order active filter, three-stage mux, switched-capacitor amplifier, active bias generator, buffer chain, and SRAM bit-cell. The number of candidate PVT corners ranges from 130 to 1800. All circuits have devices with reasonable sizings. The device models are from modern industrial processes ranging from 65 to 28 nm nodes.

We performed two complementary sets of benchmarking runs. The methodology for each set is as follows.

Per-circuit benchmarks methodology. First, we simulated all candidate PVT corners and recorded the worst-case value seen at each output; these form the suite of "golden" reference results. Then, we ran Fast PVT verification *once per circuit* (13 runs total) and recorded the worst-case values that were found for each output, and how many simulations Fast PVT took.

Per-output benchmarks methodology. We obtained the "golden" reference results by simulating all PVT corners. Then, we ran Fast PVT verification *once for each output of each circuit* (118 runs total) and recorded the worst-case values that were found for each output, and how many simulations Fast PVT took.

2.5.2 Experimental Results

In all the runs, Fast PVT successfully found the worst-case point. This is crucial: speedup is only meaningful if Fast PVT can find the same worst-case results as a full-factorial (all corners) PVT run.

Figure 2.9 shows the distribution of speedups using Fast PVT verification, on the per-circuit benchmarks (left plot), and per-output benchmarks (right plot). Each point in each plot is the result of a single run of Fast PVT. Speedup is the number of full factorial PVT corners, divided by the number of PVT corners that Fast PVT verification needs to find the worst case outputs and then to stop.

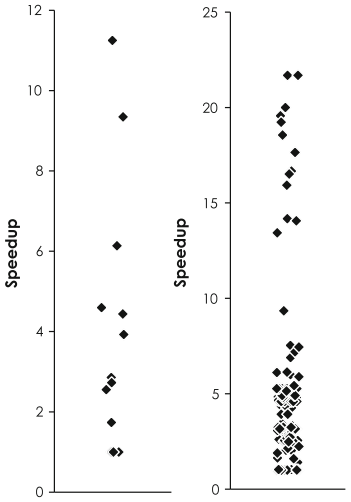

Fig. 2.9 PVT verification results, showing speedup per problem. Left: Speedup on per-circuit benchmarks ($\geq$1 outputs per circuit). Right: Speedup on per-output benchmarks

On the 13 per-circuit benchmark runs (Fig. 2.9 left), Fast PVT found the true worst-case corners 100 % of the time. The average speedup is 4.0x, and the maximum speedup is 11.0x.

On the 118 per-output benchmark runs (Fig. 2.9 right), Fast PVT found the true worst-case corners 100 % of the time. The average speedup is 5.0x, and the maximum speedup is 21.7x.

We found that Fast PVT verification speedups are consistently high when the total number of corners is in the hundreds or more; for fewer corners, the speedup can vary more. A case of 1.0x speedup (i.e. no speed-up) simply means that Fast PVT did not have enough confidence in its models to complete prior to running all possible corners.

2.5.3 Post-Layout Flows

Post-layout flows are worth mentioning. Figure 2.5 shows a PVT-aware flow for designing prior to layout. This flow can be readily adapted to handle post-layout as well. Because post-layout netlists tend to be more expensive to simulate, there are options we can consider, in order of increasing accuracy and simulation time:

- Just simulate on a typical PVT corner. If specs fail, adjust the design to meet them, in layout space or sizing space, with feedback from the simulator on the corner. Finally, if time permits, re-loop with verification.
- Simulate with worst-case pre-layout PVT corners. If specs fail, adjust design as needed.
- Run Fast PVT verification. If specs fail, adjust design as needed.

As an example, we benchmarked Fast PVT on the PLL VCO circuit from Sect. 2.2 in its post-layout form. Fast PVT on the post-layout circuit took 171 simulations to find the true worst-case corners and to achieve verification-level confidence. Comparing this with the technique of simulating all 3375 combinations shows a 19.7x speedup.

2.5.4 Fast PVT for Multiple Outputs and Multiple Cores

The algorithm just described is for a single output, assuming a serial process. It is straightforward to extend the Fast PVT algorithm to handle more than 1 output, as follows. All outputs use the same initial samples. In each iteration, the algorithm for each output chooses a new sample. Model building for a given output always uses all the samples taken for all outputs.

It is also straightforward to parallelize the Fast PVT algorithm. Initial sampling sends off all simulation requests at once, and the iterative loop is modified slightly in order to keep available parallel simulation nodes busy.

2.5.5 Fast PVT Corner Extraction Limitations

Recall that the corner extraction task of Fast PVT runs the initial DOE, builds a model, predicts worst-case corners, and stops. It is not adaptive. Therefore, its main characteristics are that it is relatively fast and simple, but not as accurate as Fast PVT in verification mode in terms of finding the actual worst-case corners.

2.5.6 Fast PVT Verification Limitations

Stopping criteria and model accuracy govern the speed and reliability of Fast PVT verification. The most conservative stopping criteria would lead to all simulations being run, with no speedup compared to full factorial. On the other hand, stopping criteria that are too aggressive would result in stopping before the worst-case is found. Fast PVT verification strikes a balance, by stopping as soon as the model is confident it has found the worst-case. The limitation is that the model may be overly optimistic, for instance if it has missed a dramatically different region. To avoid suffering this limitation, Sect. 2.5.7 provides guidelines on measurements, since having the right measurements can result in improved model quality.

Model construction becomes too slow for >1000 simulations taken. In practice, if Fast PVT has not converged by 1000 simulations, it probably will not converge for more simulations, and will simply simulate the remaining corners. Appendix B has details.

Fast PVT performs best when there are fewer input PVT variables. For >20 variables, modeling becomes increasingly difficult and starts to require >1000 simulations. For these reasons, we do not recommend using Fast PVT with >20 input variables. Appendix B has details.

Fast PVT speedup compared to full factorial is dependent on number of candidate corners: the more candidate corners, the higher the speedup. This also means that if there is a small number of candidate corners (e.g. 50 or less), then the speedup is usually not significant (e.g. 2x, or even 1x).

In summary, while Fast PVT does have some limitations, it nonetheless provides significant benefit for production designs.

2.5.7 Guidelines on Measurements

Fast PVT's speed and accuracy depend on the accuracy of the model constructed, that is the model mapping from PVT input variables to outputs. The greater the model accuracy, the faster the algorithm convergence.

In designing measurements and choosing targets, the user should be aware of these guidelines:

- The outputs can be binary, and more generally, can have discontinuities. Outputs with these behaviors however typically require more simulations to model accurately.
- Some candidate samples can produce output measurement failures, as long as those measurement failures correspond to extreme values of one of the outputs being targeted.
- The outputs cannot contain simulator noise as the primary component of output variation; this situation results in random mappings from PVT variables to output. If this situation occurs, it usually means there is a problem with the measurement. A well-implemented Fast PVT algorithm should automatically detect random mappings from PVT variables to outputs.

2.6 Design Examples

This section presents two design examples based on production use of Fast PVT technology by industrial circuit designers.

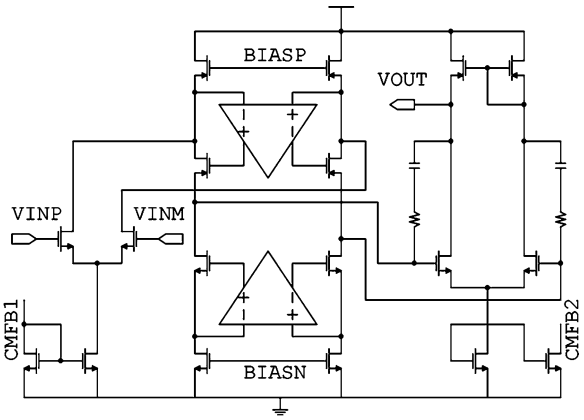

Fig. 2.10 Folded-Cascode amplifier with gain boosting

2.6.1 Corner-Based Design of a Folded-Cascode Amplifier

In this example, we examine the PVT corner analysis of an amplifier circuit. Figure 2.10 shows the amplifier schematic for this example.

Table 2.1 describes the required operating process and environmental conditions for the amplifier.

Performance outputs for the amplifier design include gain, bandwidth, phase margin, noise, power consumption, and area. PVT corner analysis and design are performed using Solido Variation Designer (Solido Design Automation 2012). Simulations are performed with the Cadence® Virtuoso® Spectre® Circuit Simulator (Cadence Design Systems 2012).

First, Fast PVT analysis is used in "corner extraction" mode to establish worst-case corners for each of the outputs of the design. The analysis needs to account for each output individually because the worst-case corners are rarely the same for all outputs. For this design, Fast PVT is configured to find worst-case corners corresponding to minimum gain, bandwidth, and phase margin, and maximum noise and power consumption.

From the 3645 combinations of process, voltage, temperature, bias, and load, Fast PVT corner extraction runs approximately 30 simulations to find a representative set of worst-case corners, resulting in a 122x simulation reduction for this first analysis step over full factorial simulation.

The worst-case corners found by Fast PVT corner extraction result in poor gain and phase margin performance for this design. Therefore, these corners are saved for use in the next step of the flow. Note that worst-case corners for all outputs are saved, not just those for poorly performing outputs. The corners for all outputs need to be included during design iteration in order to ensure that no outputs go out of specification under worst-case conditions when changes are made to the design.

Table 2.1 Required process and environmental conditions for amplifier design

Conditions	Values	Quantity
Process		
MOS	FF, FS, TT, SF, SS	5
Resistor	HI, TYP, LO	3
Capacitor	HI, TYP, LO	3
Total process combinations		45
Environmental conditions		
Temperature	−40, 27, 85	3
Supply voltage	1.45, 1.5, 1.55	3
Bias current[a]	9.5u, 10u, 10.5u	3
Load capacitance	160f, 170f, 180f	3
Total environmental Combinations		81
Total combinations (process and environmental)		3645

[a] Bias current variation affects the BIASN and BIASP voltages shown in the schematic

For the next step in the flow, the corners found during the initial Fast PVT corner extraction are used to examine the sensitivity of the design under worst-case conditions. Several opportunities are available for modifying the design to improve gain and phase margin under worst-case conditions, without trading off too much performance, power, or area. Such opportunities include adjusting the *W/L* ratios of the NMOS and PMOS bias transistors (the transistors with gates connected to BIASN and BIASP), as well as the sizing of the second stage differential pair and capacitors.

Once the sensitivity of the outputs is determined across the worst-case corners, the design is modified and the performance is checked by simulating at each worst-case corner. In this example, this iterative modify/simulate procedure is repeated four times to achieve satisfactory performance across all worst-case corners and to find an acceptable tradeoff between performance, power, and area.

After the design iterations are complete, Fast PVT verification is performed. The analysis confirms that design performance is acceptable across the entire range of PVT combinations. Fast PVT verification runs only 568 simulations to find the worst-case corners for this design out of the total 3645 combinations of process and environmental conditions.

Table 2.2 shows the number of simulations performed for each step in this flow. For comparison purposes, two other approaches were used on this design and are also summarized in the table. The first column summarizes the Fast PVT flow, the second column summarizes the flow of running all combinations and the third column uses the designer's best guess for determining worst-case corners. Although the "best guess" flow uses the least simulations, the resulting design does not perform well across all corners.

Table 2.2 Number of simulations required for Fast PVT design/verification flow

Step	Number of simulations (Fast PVT flow)	Number of simulations (full factorial flow)	Number of simulations (designer "best guess" flow)
Initial corner extraction	30	3,645	5
Design sensitivity across worst-case corners	115	115+[a]	115
Design iteration	20	20+	20
Verification	568	3,645	5
Total	733	7,425+	145[b]

[a] The full factorial flow simulation numbers assume that design sensitivity analysis and design iterations are performed with manually extracted corners from a full factorial PVT analysis. If design iterations are instead performed by running all corners, then the number of simulations would be even larger

[b] The best guess flow does not correctly identify the worst-case corners, and the resulting design performance is not satisfactory in that case

In summary, using the Fast PVT design and verification flow with the amplifier design reduces simulations, while achieving an overall robust design. Note that if there are failures during verification, further iterations are required, which can increase the number of simulations. However, the overall number of simulations in that case is still much less than running full factorial PVT, and in practice, this occurs relatively infrequently.

2.6.2 Low-Power Design

Low-power design is very important in mobile systems-on-a-chip (SoCs), due to the demand for longer battery life in smaller, lighter devices. Power consumption needs to be carefully analyzed across the different components in a mobile SoC to ensure that power consumption is acceptable and that it does not become too large under certain operating conditions.

A challenge in low-power design is that power consumption can vary significantly under different process and environmental conditions. Furthermore, the chip state has a dramatic impact on the power consumption. That chip state interacts with the process and environmental conditions, such that a state that causes little power consumption under one set of conditions may cause much more power consumption under a different set of conditions.

For this reason, it is important to simulate the design under many different process, environment, and state conditions. However, the number of combinations that need to be taken into account for each cell can be very large. This is especially problematic for circuits with long transient simulation times. The challenge is even greater when designing below 28 nm, where additional simulation of RC parasitic corners is required to capture interconnect variability. The analysis must then be

repeated for each cell in the SoC and the results aggregated together. Finally, if power consumption is too great under certain conditions, cells must be redesigned and re-simulated, further increasing the simulation and time burden required for complete power analysis for the chip.

In this example, a variety of conditions need to be taken into account, with the following values for each:

- Temperature: −40, 27, 85 (in °C)
- Voltage: 2.5 V ± 10 %
- Power mode: 0, 1, 2
- Input state: 0, 1, 2
- Bias: 5uA ± 10 %
- Process corners: FF/FS/TT/SF/SS
- RC extraction corners: 1, 2, 3, 4, 5, 6, 7, 8, 9

The total number of combinations required to achieve full coverage of these conditions is 13,122. For a simulation time of one day on a cluster of 200 machines with 200 simulator licenses, it would take over two months to complete one full analysis of this design. It is easy to see how the addition of more variables quickly increases the number of combinations to well above 100,000.

To make the problem tractable, the number of combinations being simulated needs to be reduced. One way to do this is to use design expertise to determine combinations that are unlikely to have adverse power consumption. However, even with this approach, the number of combinations can still be very large. For the remaining combinations, the number of corners to be analyzed needs to be as large as possible.

To achieve this, Fast PVT is used to adaptively simulate and predict the power consumption under all of the required process/environment/state conditions. Fast PVT reduces the total number of simulations for covering the 13,122 combinations to 643, providing approximately a 20x simulation savings.

In summary, power analysis is key to designing successful mobile SoCs, but it is important that variation effects are analyzed across process, environmental, and power state conditions to ensure that the design stays within power constraints. The method chosen to do this must reconcile the tradeoff between thoroughness, available time, and computing resources.

2.7 Fast PVT: Discussion

We now examine Fast PVT in terms of the set of qualities required for a PVT technology outlined in the introduction.

1. ***Rapid***: Fundamentally, Fast PVT can be fast because it learns about the mapping of PVT variables to output performances on-the-fly, and takes advantage of that knowledge when choosing simulations. As the benchmarks

demonstrated, Fast PVT verification has speedups averaging 4–5x; with speedup up to 22x.
2. ***Reliable***: Fast PVT is reliable because it uses SPICE in the loop, nonlinear modeling technology, and takes measures to fill blind spots in the model. In benchmark results, Fast PVT finds the worst-case corners in 100 % of cases across a broad range of real-world problems.

In summary, Fast PVT is simultaneously rapid and reliable, which makes possible a highly efficient and practical PVT design flow.

2.8 Conclusion

In modern semiconductor processes, the need to properly bracket variation has caused the number of possible PVT corners to balloon. Rather than a handful of corners, designers must test against hundreds or even thousands of possible corners, making PVT-based design and verification exceedingly time-consuming.

This chapter described various possible flows to handle PVT variation, and various ways to find worst-case PVT corners. It then presented the Fast PVT approach, which casts PVT verification and corner extraction as a global optimization problem, then solves the problem using model-building optimization (MBO). Benchmark results verified that Fast PVT delivers good speedups while finding the true worst-case PVT corners in all benchmark cases.

Fast PVT enables a rapid PVT design flow, via fast extraction of worst-case PVT corners and fast verification. It reduces overall design time and improves reliability over conventional methods. This in turn promotes the reliable development of more competitive and more profitable products.

Appendix A: Details of Fast PVT Verification Algorithm

Detailed Algorithm Description

We now give a more detailed description of the Fast PVT algorithm. We do so in two parts: first, by showing how we recast the problem as a global optimization problem, then how this problem can be quickly and reliably approached with an advanced model-building optimization technique.

We can cast the aim of finding worst-case corners as a global optimization problem. Consider $\boldsymbol{x}$ as a point in PVT space, i.e. a PVT corner. Therefore, $\boldsymbol{x}$ has a value for the model set or for each device type if separate per-device models are used, V_{dd}, R_{load}, C_{load}, temperature T, etc. We are given the discrete set of N_C possible PVT corners $\boldsymbol{X}_{all} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{NC}\}$, and a SPICE-simulated output

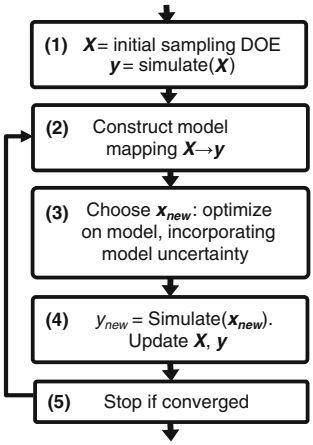

Fig. 2.11 Fast PVT verification algorithm

response to each corner $f(\boldsymbol{x})$. We aim to find $\boldsymbol{x}^*$, the optimal PVT corner which gives the minimum or maximum $f(\mathbf{x})$, depending on the output. Collecting this together in an optimization formulation, we get:

$$\mathbf{x}^* = argmin\,(f(\boldsymbol{x}))$$
$$\text{subject to } \boldsymbol{x} \text{ in } \boldsymbol{X}_{all}$$

Now, given the aims of speed and reliability, the challenge is to solve the global optimization problem with as few evaluations of $f(\boldsymbol{x})$ as possible to minimize simulations, yet reliably find the $\boldsymbol{x}^*$ returning the global minimum, which is the true worst-case corner.

Fast PVT approaches this optimization problem with an advanced model-building optimization approach that explicitly leverages modeling error.

We now detail the steps in the approach, as shown in Fig. 2.11.

Step 1: Raw initial samples: Fast PVT generates a set of initial samples $\boldsymbol{X} = \boldsymbol{X}_{init}$ in PVT space using design of experiments (DOE) (Montgomery 2004). Specifically, the full set of PVT corners is bounded by a hypercube, then DOE selects a fraction of the corners of the hypercube in a structured fashion.

Simulate initial samples: Run SPICE on the initial samples to compute all initial output values: $\boldsymbol{y} = \boldsymbol{y}_{init} = f(\boldsymbol{X}_{init})$.

Step 2: Construct model mapping $\boldsymbol{X} \rightarrow \boldsymbol{y}$: Here, Fast PVT constructs a regressor (an RSM) mapping the PVT input variables to the SPICE-simulated output values. The choice of regressor is crucial. Recall that a linear or quadratic model makes unreasonably strong assumptions about the nature of the mapping. We do not want to make any such assumptions—the model must be able to handle arbitrarily nonlinear mappings. Furthermore, the regressor must not only predict an output value for unseen input PVT points, it must be able to report its confidence in that prediction. Confidence should approach 100 % at points that have previously been simulated, and decrease as distance from simulated points increases.

An approach that fits these criteria is Gaussian process models (GPMs, a.k.a. kriging)(Cressie 1989). GPMs exploit the relative distances among training points and the distance from the input point to training points while predicting output values and the uncertainty of the predictions. For further details on GPMs, we refer the reader to Appendix B.

Step 3: Choose new sample x_{new}: Once the model is constructed, we use it to choose the next PVT corner $\boldsymbol{x}_{new}$ from the remaining candidate corners $\boldsymbol{X}_{left} = \boldsymbol{X}_{all} \backslash \boldsymbol{X}$. One approach might be to simply choose the $\boldsymbol{x}$ that gives minimum predicted output value $g(\boldsymbol{x})$:

$$\boldsymbol{x}_{new} = argmin(g(\boldsymbol{x})) \; subject \; to \; \boldsymbol{x} \; in \; \boldsymbol{X}_{left}$$

However, this is problematic. While such an approach optimizes $f(\boldsymbol{x})$ in regions near where worst-case values have already been simulated, there may be other regions with relatively fewer simulations, which have different simulated values than model predictions. These are model *blind spots*, and if such a region contained the true worst-case value, then this simple approach would fail.

GPMs, however, are aware of their blind spots because they can report their uncertainty. So, we can choose $\boldsymbol{x}_{new}$ by including uncertainty $s^2(\boldsymbol{x})$, where $\boldsymbol{X}_{left}$ is the set of remaining unsimulated corners from $\boldsymbol{X}$:

$$\boldsymbol{x}_{new} = argmin\big(h\big(g(\boldsymbol{x}),\ s^2(\boldsymbol{x})\big)\big) \text{s.t. } \boldsymbol{x} \; in \; \boldsymbol{X}_{left}$$

where $h(\boldsymbol{x})$ is an infill criterion function that combines both $g(\boldsymbol{x})$ and $s^2(\boldsymbol{x})$ in some fashion. There are many options for $h(\boldsymbol{x})$, but a robust one uses least-constrained bounds (LCB) (Sasena 2002). This method returns the $\boldsymbol{x}_{new}$ that returns the minimum value for the lower-bound of the confidence interval. Mathematically, LCB is simply a weighted sum of $g(\boldsymbol{x})$ and $s^2(\boldsymbol{x})$.

Step 4: Simulate new sample; update: Run SPICE on the new sample: $\boldsymbol{y}_{new} = f(\boldsymbol{x}_{new})$. We update all the training data with the latest point: $\boldsymbol{X} = \boldsymbol{X} \cup \boldsymbol{x}_{new}$, and $\boldsymbol{y} = \boldsymbol{y} \cup \boldsymbol{y}_{new}$.

Step 5: Stop if converged: Here, Fast PVT stops once it is confident that it has found the true worst-case. Specifically, it stops when it has determined that there is very low probability of finding any output values that are worse than the ones it has seen.

Illustrative Example of Fast PVT Convergence

Figure 2.12 shows an example Fast PVT verification convergence curve, plotting output value versus sample number. The first 20 samples are initial samples $\boldsymbol{X}_{init}$ and $\boldsymbol{y}_{init}$. After that, each subsequent sample $\boldsymbol{x}_{new}$ is chosen with adaptive modeling. The predicted lower bound shown is the minimum of all 95 %-confidence predicted lower bounds across all unsimulated PVT corners ($\boldsymbol{X}_{left}$). The PVT

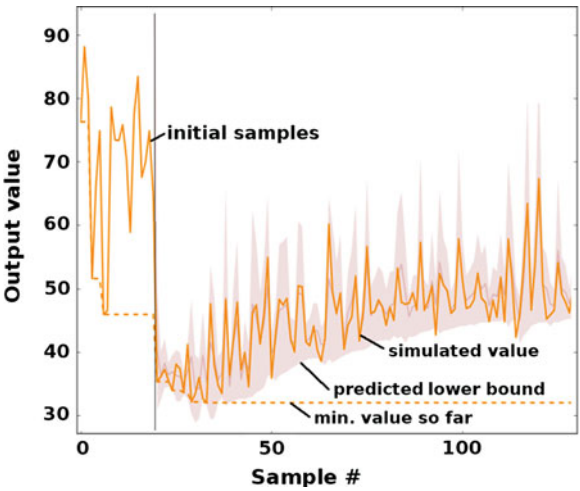

Fig. 2.12 Example of Fast PVT convergence

corner with this minimum value is chosen as the next sample $\boldsymbol{x}_{new}$. That new sample is simulated.

The dashed line in Fig. 2.12 is the minimum simulated value so far. We see that immediately after the initial samples, the first $\boldsymbol{x}_{new}$ finds a significantly lower simulated output value $f(\boldsymbol{x}_{new})$. Over the course of the next several samples, Fast PVT finds even lower simulated values. Then, the minimum value curve flattens, and does not decrease further. Simultaneously, from sample number 20–40, we see that the predicted lower bound hovers around an output value of 30, but then after that, the lower bound increases, creating an ever-larger gap from the minimum simulated value. This gap grows because $\boldsymbol{X}_{left}$ has run out of corners that are close to worst-case, hence the remaining next-best corners are much higher than the already-simulated worst-case. As this gap grows, confidence that the worst-case is found increases further, and at some point we have enough confidence to stop.

Appendix B: Gaussian Process Models

Introduction

Most regression approaches take the functional form:

$$g(\boldsymbol{x}) = \sum_{i}^{NB} w_i g_i(\boldsymbol{x}) + \varepsilon$$

Where $g(\boldsymbol{x})$ is an approximation of the true function $f(\boldsymbol{x})$. There are N_B basis functions; each basis function $g_i(\boldsymbol{x})$ has weight w_i. Error is ε. Because $g_i(\boldsymbol{x})$ can be an arbitrary nonlinear function, this model formulation covers linear models,

polynomials, splines, neural networks, support vector machines, and more. The overall class of models is called generalized linear model (GLM). Model fitting reduces to finding the w_i and $g_i(\boldsymbol{x})$ that optimize criteria such as minimizing mean-squared error on the training data, and possibly regularization terms. These models assume that error ε is normally distributed, with mean of zero, and with no error correlation between training points.

In this formulation, the error distribution remains constant throughout input variable space; it does not reduce to zero as one approaches the points that have already been simulated. This does not make sense for SPICE-simulated data: the model should have 100 % confidence (zero error) at previously simulated points, and error should increase as one draws away from the simulated points. Restating this, the model confidence should change depending on the input point.

Towards Gaussian Process Models (GPMs)

We can create a functional form where the model confidence *depends* on the input point:

$$g(\boldsymbol{x}) = \sum_{i}^{NB} w_i g_i(\boldsymbol{x}) + \varepsilon(\boldsymbol{x})$$

Note how the error ε is now a function of the input point $\boldsymbol{x}$. Now the question is how to choose w_i, $g_i(\boldsymbol{x})$, and $\varepsilon(\boldsymbol{x})$ given our training data $\boldsymbol{X}$ and $\boldsymbol{y}$. A regressor approach that fits our criteria of using $\varepsilon(\boldsymbol{x})$ and handling arbitrary nonlinear mappings, is the Gaussian process model approach (GPMs, a.k.a. kriging). GPMs originated in the geostatistics literature (Cressie 1989) but have recently become more popular in the global optimization literature (Jones et al. 1998) and later in machine learning literature (Rasmussen and Williams 2006). GPMs have such a powerful approach to modeling $\varepsilon(\boldsymbol{x})$ that they can replace the first term of $g(\boldsymbol{x})$ with a constant μ, giving the form:

$$g(\boldsymbol{x}) = \mu + \varepsilon(\boldsymbol{x})$$

In GPMs, $\varepsilon(\boldsymbol{x})$ is normally-distributed with mean zero, and variance represented with a special matrix $\boldsymbol{R}$. $\boldsymbol{R}$ is a function of the N training input points $\boldsymbol{X}$, where correlation for input points $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ is $\boldsymbol{R_{ij}} = corr(\boldsymbol{x_i}, \boldsymbol{x_j}) = exp(-d(\boldsymbol{x_i}, \boldsymbol{x_j}))$, and d is a weighted distance measure $d(\boldsymbol{x_i}, \boldsymbol{x_j}) = \sum_{h=1n} \theta_{\boldsymbol{h}} |\boldsymbol{x_{i,h}} - \boldsymbol{x_{j,h}}|^{ph}$. This makes intuitive sense: as two points $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ get closer together, their distance d goes to zero, and therefore their correlation $\boldsymbol{R_{ij}}$ goes to one. Distance measure d is parameterized by n-dimensional vectors $\boldsymbol{\theta}$ and $\boldsymbol{p}$, which characterize the relative importance and smoothness of input variables. $\boldsymbol{\theta}$ and $\boldsymbol{p}$ are learned via maximum-likelihood estimation (MLE) on the training data.

From the general form $g(\boldsymbol{x}) = \mu + \varepsilon(\boldsymbol{x})$ which characterizes the distribution, GLMs predict values for unseen $\boldsymbol{x}$ via the following relationship:

$$g(\boldsymbol{x}) = \mu + \boldsymbol{r}^{\mathrm{T}}\boldsymbol{R}^{-1}(\boldsymbol{y} - \boldsymbol{1}\mu)$$

where the second term adjusts the prediction away from the mean based on how the input point $\boldsymbol{x}$ correlates with the N training input points $\boldsymbol{X}$. Specifically, $\boldsymbol{r} = \boldsymbol{r}(\boldsymbol{x}) = \{corr(\boldsymbol{x},\boldsymbol{x_1}), \ldots, corr(\boldsymbol{x}, \boldsymbol{x_N})\}$. Once again, this formula follows intuition: as $\boldsymbol{x}$ gets closer to a training point $\boldsymbol{x_i}$, the influence of that training point $\boldsymbol{x_i}$, and its corresponding output value y_i, will become progressively greater.

Recall we want a regressor to not just predict an output value $g(\boldsymbol{x})$, but also to report the uncertainty in its predicted output value. In GLMs, this is an estimate of variance s^2:

$$s^2(\boldsymbol{x}) = \sigma^{2*}\left(1 - \boldsymbol{r}^T\boldsymbol{R}^{-1}\boldsymbol{r} + \left(1 - \boldsymbol{1}^T\boldsymbol{R}^{-1}\boldsymbol{r}\right)^2/\left(\boldsymbol{1}^T\boldsymbol{R}^{-1}\boldsymbol{1}\right)\right)$$

In the above formulae, μ and σ^2 are estimated via analytical equations that depend on $\boldsymbol{X}$ and $\boldsymbol{y}$. For further details, we refer the reader to (Jones et al. 1998).

GPM Construction Time

With GPMs, construction time increases linearly with the number of parameters, and as a square of the number of training samples. In practical terms, this is not an issue for 5 or 10 input PVT variables with up to $\approx$ 500 corners sampled so far, or for 20 input PVT variables and $\approx$ 150 samples; but it does start to become noticeable if the number of input variables or number of samples increases much beyond that.

In order for model construction not to become a bottleneck, the Fast PVT algorithm behaves as follows:

- Once 180 simulations have been reached, it only builds models every 5 simulations, rather than after every new simulation. The interval between model builds increases with the number of simulations (=max(5, 0.04 * number of simulations)).
- If Fast PVT has not converged by 1000 simulations, it simply simulates the rest of the full-factorial corners.

References

Cadence Design Systems Inc. (2012) Cadence® Virtuoso® Spectre® Circuit Simulator, http://www.cadence.com

Cressie N (1989) Geostatistics. Am Statistician 43:192–202

Huyer W, Neumaier A (1999) Global optimization by multilevel coordinate search. J Global Optim 14:331–355
Jones DR, Schonlau M, Welch WJ (1998) Efficient global optimization of expensive black-box functions. J Global Optim 13:455–592
Land AH, Doig AG (1960) An automatic method of solving discrete programming problems. Econometrica 28(3):497–520
Montgomery DC (2004) Design and analysis of experiments, 6th Edition, Wiley, Hoboken
Rasmussen CE, Williams CKI (2006) Gaussian processes for machine learning, MIT Press, Cambridge, MA
Sasena MJ (2002) Flexibility and efficiency enhancements for constrained global optimization with kriging approximations, PhD thesis, University of Michigan
Solido Design Automation Inc. (2012) Variation Designer, http://www.solidodesign.com

http://www.springer.com/978-1-4614-2268-6

Variation-Aware Design of Custom Integrated Circuits: A Hands-on Field Guide

McConaghy, T.; Breen, K.; Dyck, J.; Gupta, A.

2013, XVI, 188 p., Hardcover
ISBN: 978-1-4614-2268-6