PART I

Engineering, Circuit Design, Flow and Methods

1

Introduction: What Makes an Engineer a Good Designer?

We present key measures, elements and problems in design. We discuss the efforts for design and verification tasks; and the inputs and outputs.

In this Chapter 1 we describe the problems of being a design engineer from a still quite general perspective, so many things appear in similar ways also in other fields (like car design). Of course, we have circuit design in our mind, and we describe the typical manual IC-specific design style in Chapter 2.

Design and circuit design is a fascinating topic, and it is a science and also a kind of art—for many amateurs and professionals. There are systematic approaches and there are physical foundations, but usually there is also something "special", especially when designing integrated circuits for high-performance areas like high-speed, high-power, or radio frequencies, but also smaller PCB (printed circuit board) designs, e.g., you often have to minimize the number of components with some "tricks." This is because—almost by definition and in opposite to digital design—there are *many* more things that matter (not only speed, area, and power consumption) and analog circuits are inherently much less error-tolerant.

Analog design is quite an art, because you need creativity to find the right compromises to fulfill many specifications, written and non-written ones. Due to more and more stringent requirements on minimizing size, power consumption, and costs (of course), designs moved in 50 years from the classical simple twenty-transistor op-amp to highly complex, multimode, mixed-signal circuits with billions of transistors (actually memory design is also very close to analog design). On the other hand, the basics have not changed much! By far, not all problems are related to complexity, and small circuits can be most tricky—a small amateur PCB design can fail for similar reasons than a high-end smartphone. It is very tough to be prepared for everything that could go wrong or just varies by nature, like transistor length and width, threshold voltages, load impedances, and gate oxide thickness. Of course, such complex designs are done by very experienced design teams, but if something gets wrong, it is indeed very often due to such variations and/or complexity (e.g., in interfaces and states).

We hope to show that also almost all numerical algorithms are based on "common sense" ("gesunder Menschenverstand")—and also dealing with them, improving them, and even applying them is something creative and fascinating! Common sense fails seldom, just some training is required, and some clarifications.

Styling versus Design. In German these are foreign words, so often both terms are misused. Adding a fancy chrome spoiler to a car, which is no race car, is styling. So something between taste, bad taste and art. Doing it because you need it for a perfect driving behavior is design. Design is closer to science, real construction, problem solving, but of course there is often still pretty much freedom. Also designers have personality, and style; and the solutions from different designers may reflect this. However, to a high degree it is indeed usually possible to clearly state, what is really required or an even optimum solution, and which parts are nice to have. Usually circuits contain not much styling elements, but incorporate quite some art.

What are the key techniques every student, engineer, and designer should know and apply? Of course, learning about circuit design is good, and doing it is even better, but can we be more specific? Two sentences I remember from my professors as a student were as follows:

> *The most important skill to learn at a university is to learn how to learn*
>
> *In IC design you can create almost everything, the problem is always making a reliable design that also works under varying parameters.*

The first statement was to some degree a clear disappointment when I heard it because I want to learn much more, but in our ever-changing environment, this is clearly an important point. It just takes some time to pick that up.

The second quote was quite a surprise, because the technology in 1988 was by far not as advanced as it is today (e.g., most ICs use single metal layer routing, and the bipolar processes had lateral pnp with very low speed and gain)—and I did not have that much experience in how much tolerances can really make your life difficult! As an amateur designer, soldering for an audio amplifier or AM transmitter, you are typically done when the circuit is just running, but that is not the case when giving the circuit to someone else! Often optimism lacks in information.

Both sentences are very important, because as an engineer you make important and costly decisions for your company, and overlooking something can happen easily. Here, professionals are even under more pressure, because you rely much more on virtual techniques; any simulation usually can only answer the questions you prepare for—like "Is the amplifier stable?" In a laboratory the amplifier circuit might just oscillate when you turn on the supply—and you can immediately see it with an oscilloscope. However, in IC design a dedicated testbench is needed, and often different options are available, so a simple question like proving stability, can become quite difficult, especially if you want to go to the limits (like achieving also a large gain and high efficiency).

Just entering a design in a schematic editor and simulating it for a kind of virtual verification is possible since 1970s for professionals (when the circuit simulator SPICE becomes popular) and for amateurs since 1980s (PSpice® came up, running on PCs). In digital design, the flow progress in the following years was amazing: Essentially, nowadays you can create circuits and even whole digital systems with millions of transistors from software because clever programs can synthesize the whole hardware in a given technology, based on few core libraries featuring the basic logic cells.

A Very Short History of Digital Synthesis. In the 1970s, digital designers made designs in quite similar way as analog designers. So they drew schematics with little standard cell sub-blocks (like NAND gates, flip-flops, ALUs, and counters). In the 1980s, for simulation purposes and more compact design description, co-called behavioral languages like VHDL and Verilog have been created. This allowed more complex designs and better documentation. In 1994, Synopsis® created a synthesizing program that has been very quickly adopted by the industry. Recent developments are e.g., regarding verification with new languages like "e" and System Verilog.

Unfortunately, analog synthesis is much harder, because the way from a unit element like a transistor or resistor to a whole block like a programmable amplifier or even to its layout is much longer. Also languages like Verilog-A are not very powerful, and better ones have found no wide application and have no industry standard.

However, in analog, RF and mixed-signal automatic synthesis failed mostly—at least commercially, although for some special areas like DAC or filter design, some kind of synthesis makes indeed sense. On the other hand, further techniques have arrived in real commercial tools and enabled engineers to do things that analog designers dreamed for years. One is statistical design, and the other is optimization—and doing it not only on small academic examples, but also on real professional often highly nonlinear circuit designs!

People working in one area like EDA tools or circuit design can often learn a lot from other fields, even from topics faraway like biology, stock pricing, weather forecast, disaster prediction, or insurances. For instance, lot of attention in many of these fields is on advanced Monte Carlo techniques, whereas for most electrical engineers MC or corner analysis has *not changed much* over the last 30 years!

Not all techniques presented in this book are brand new. For instance, historically optimization is *not* a new topic, and some of the most important algorithms have been developed in the late 1960s and have been applied to electronic design in the 1970s, e.g., for passive RF filter design. However, the option to use such advanced techniques was only present for few academic institutes, and no user-friendly software was really available. One of the earliest commercial successful optimization tools came up in the 1980s and was Super-Compact™, able to simulate and optimize linear RF circuits. We used it intensively for transistor modeling and for wide-band amplifier design. Generally optimization has found widest use in modeling, e.g., for

semiconductor devices—but until now not really much for true complex analog circuits, so we will discuss when optimization makes sense, and how users can make circuits easier to optimize.

A situation perfect for learning is if something gets *wrong*! Of course you can learn from "best practice" examples, but often the real difference between two algorithms becomes visible if something becomes more difficult, so that one fails, whereas the other methods may still work! That is a good starting point for more thinking and for innovations—of course not only for circuit designers, but also for CAD researchers, and for CAD managers. Unfortunately, in EDA environments and in real design situations you have seldom much time to inspect all difficulties regarding algorithms in detail. So in case of problems clear guidance is needed.

A last motivation for reading this book should be this: We all carry around in our head rules and guidelines that give us a sense of intention. The topological map of a big foreign metro will seem "obscure" to a casual visitor, but a *resident* must understand its structure and some details to enable daily travel by memory. Similar to this, engineers have to deal with numerous equations, tools, models, etc., and they must be sorted in one's mind for everyday work. For instance, you do not need to have knowledge about Bessel's function directly in your mind, but should have a feeling for V_{BE} and its behavior versus temperature, versus current, etc., or you should know the meaningful range for current densities in your used technology. Such issues must be at your fingertips, and beyond that, they must be integrated into the *instinctive* fabric—that *is your core being*. You will not get very far on the metro if you need to consult the map each day as you travel to work! Engineers frequently have to make journeys to places far from familiar landmarks. Returning from such, we can return to the challenges of daily work with a new perspective, a little better equipped to examine problems under a brighter light. Do not wait to be told what to do: Do it anyway. Do it soon. Indeed, statistics can be as interesting as circuit design! Optimization has an even closer relationship to design and can be very helpful too. As often in engineering, there are many better ways than "try the same but harder"!

1.1 Key Problems in Circuit Design

In a design project, engineers have to deal with many variables and we have to treat them in a systematic way. Intuitively, you do it mostly, but sometimes confusions can arise. So let us introduce some common simple notations and conventions. In our book, we mark vectors in **bold** face, and we use bold uppercase characters for matrices (like **H** for the Hessian matrix). For random

variables, we follow the convention of using uppercase characters (like X). As performance functions, we use f (or $\boldsymbol{f}$ for multiple performances). For names like resistor instance R_2, we use the normal font, but for variables (usually real or complex numbers), we use italics like $A = g_\mathrm{m} R_\mathrm{L}$. (the m stands for mutual and the L for load—both are names, so no variables).

Note: Sometimes it is hard to say whether a threshold voltage or the supply is fixed or a variable, so we follow our convention when it really matters for understanding—like in flowcharts or equations—but not as slaves. On top of the mentioned conventions, often just all symbols are written in italic style. However, in the context of circuit design these are not always a good ideas.

Example: For an optimization, we usually need to vary multiple parameters x_i, and we can handle this easier by putting them into a vector $\boldsymbol{x}$, e.g., $\boldsymbol{x} = (R_1, C_1, R_2)^\mathrm{T}$ (T stands for transpose, turning the "horizontal" vector into a vertical one. This is sometimes needed for matrix calculations).

A designer has to manage many kinds of parameters $\boldsymbol{x}$ which impact circuit performances $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$:

- Design parameters $\boldsymbol{x}_\mathrm{D}$: They are controllable to the designer, so can be set dedicatedly. We assume they will not change during production, only in the design phase. Examples are the value of resistor R_1, the number of resistor segments in parallel in R_1, the capacitance of a capacitor C_1, and the width or the number of fingers of transistor N_2, and on top of these *nominal* design values, there can be of course variations from process or mismatch!

- Statistical parameters $\boldsymbol{x}_\mathrm{S}$: The resistor R_1 may have a nominal value of 1 kΩ set by R_sheet, length, and width, but in production you may observe statistical variations. Usually, mathematical models are available defining, e.g., the standard deviation of R_sheet. Elements can vary, e.g., due to global statistical variations (like from wafer to wafer), but also even two resistors constructed in the same way and on the same wafer may have different values—within-die variation (WID)—due to the so-called mismatch, so $\boldsymbol{x}_\mathrm{S} = (\boldsymbol{x}_\mathrm{P}, \boldsymbol{x}_\mathrm{MM})$. Often the designer can hardly influence global variations, but mismatch can usually be reduced by increasing the device area. Even for a perfect layout, you have to accept a certain mismatch, unfortunately.

- Conditions, constraints, environmental or operational parameters, range parameters $\boldsymbol{x}_{\mathrm{R}}$: like temperature supply voltage, load resistance, etc. – usually defined as parameter ranges. Also operating modes like temperature, over-current or over-voltage shutdown, power-down, low-gain mode, etc. might be part of $\boldsymbol{x}_{\mathrm{R}}$.

Parameters, Variables, Constants. Often there is quite some confusion about what is what! It actually depends on the *context* and the *analysis you apply*. Surely, ε_0 of vacuum is a physical *constant*, but for other materials ε_{r} it might be a function of temperature or a statistical *variable* even!
Another example is this: We can treat statistics this way, that we assume there is an ideal model from which we get random samples, e.g., in the background there is a normal Gaussian distribution having a fix well-defined mean μ and standard deviation σ. If we take a sample (via measurements or simulations), we can calculate the mean of the sample data, and it might be different from the ideal mean value, just due to chance. So is μ a variable? Having a fix model in mind, it would be no real variable. And what about the mean from the sample? A specific sample is just a sample, it is as it is: once it is, it might be also regarded as a (specific) constant set! Looks strange, but actually this is the way we follow if we do a parameter *estimation* e.g., via maximum likelihood method (ML). Here we take the data is given, so fix. And we search for the model parameters (like μ and σ) which fit best to the data, so we *treat* the parameters as *variables*. Although later we interpret them as fix, e.g., when using the model in a Monte-Carlo analyses.

In many cases it makes also sense to differentiate between (global) variables (like sheet resistance) and e.g., instance-specific parameters (like length of transistor #3 or its threshold variation against the ideal value), but also this is a convention which is usually not followed strictly.

In IC design, there is quite a clear trend that the number of all kind of parameters increases, i.e., design becomes more complex and also the models (Figure 1.1).

In addition, also the impact of variations tends to increase, e.g., an IR drop of 100 mV matters much more in modern low-voltage designs than in older technologies. Also the changes of threshold voltages (e.g., from statistics and temperature) in *relation* to supply or the absolute thresholds become

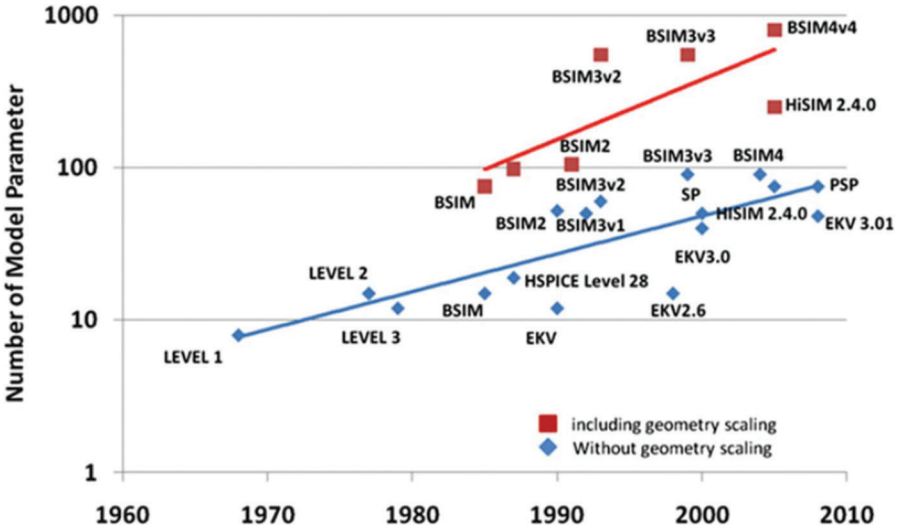


Figure 1.1 MOS model complexity increase [Haase].

more critical (Figure 1.2)—besides several other problems like increased local variations and layout-dependent effects (LDE).

Note: Unfortunately Figure 1.2 shows no units on the y-axis (we just found none!), but the intention is to show that the speed improves by using modern process nodes. This together with smaller area, lower costs and lower power consumption is the major benefit of new technologies. However, unfortunately also the *relative* performance *spread* becomes larger, so harder to manage. Of course, the exact values depend also on technology features, devices sizes, supply voltage tolerances, temperature, etc. What is also not easy to show in

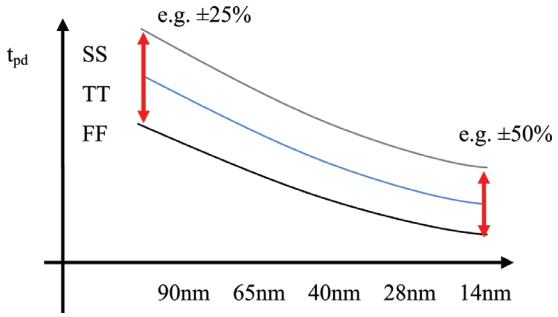


Figure 1.2 Increasing process corner spread on CMOS speed.

a picture, is that e.g., due to clever self-adaptive circuits the spread becomes manageable. And there are tools which can address the problems of variations accurately and efficiently.

In some design environments, and especially in older process development kits (PDK) and model cards, global statistical variations are usually treated not as statistical parameters, but only with fix sets of process corners, like FastMOS (=best-case speed), SlowMOS (worst-case speed), MaxR, MinC, and SlowNMOSFastPMOS (lowest threshold) or just nominal. This is a simplification, because "slow" can only be the worst-case in dedicated terms, like with respect to propagation delay time for a certain class of circuits like CMOS logic but not for other circuits or other measures (like bandwidth and phase margin). The major advantage of process corners is that the designer can directly pick them, simulate, and get at least an approximated worst or best case. In many design environments, you also have different setups available, so you can decide whether you want to treat process variations as corners or via MC. Best use both methods for understanding and efficiency. Actually, also classical logic design was already done in a variation-aware sense, but it excluded statistical variations almost completely.

More Statistical Methods? In principle, we can treat statistical variables $\boldsymbol{x}_\mathrm{S}$ with combinatorial methods—which makes sense with discrete random variables, like coins—or we may use Monte Carlo. Actually, there are good attempts to use statistical techniques also for range parameters (corners) $\boldsymbol{x}_\mathrm{R}$ or for design variables $\boldsymbol{x}_\mathrm{D}$. The idea of *randomized verification* for corners is quite clever in cases where the number of *directed* tests would be huge, like in big digital or software systems! Random methods for design variables can make sense for difficult optimization problems to achieve global convergence. In the near future, more and more statistical methods will come up—also in analog design.

1.1.1 Brute-Force Design—No Way!

If you want to address the general problem of "design" mathematically and want to describe it in high detail, we would have to deal with all performances f collected in vector $\boldsymbol{f}$ as a function of all variables $\boldsymbol{x} = \boldsymbol{x}_\mathrm{D}, \boldsymbol{x}_\mathrm{S}, \boldsymbol{x}_\mathrm{R})^\mathrm{T}$.

Note: This "art of design" is actually only a subtasks, although a very important and time-consuming one. Usually, there is a kind of exploration

phase upfront, which consists also of testing different circuits and composing/extending circuits. And afterward, there is also a longer sign-off phase, and there the focus is on verification ($\boldsymbol{x}_\mathrm{D}$ almost fix). However, often there is no clear separation, neither in project time, nor in the tools; there are many overlaps and iterations. This is almost a characteristic for analog design (Figure 1.3).

Unfortunately, the performance function $\boldsymbol{f}(\boldsymbol{x})$ can be extremely complex. In a clever testbench, we might be able to get all f with a single circuit simulation like a transient analysis driving the circuit to all modes, but even then we can typically only cover one single point $\boldsymbol{f}(\boldsymbol{x})$ (also called sample) of that function; already this can take a minute or an hour. As circuit simulation is often the most time-consuming (automated) part of the design, overall efficiency can be often measured in *many* simulations needed to achieve the targets. In fact, simulators are quite complex and have dozens of analyses and hundreds of options, whereas the classical methods on top—like parameter sweeps or Monte Carlo—have little *internal* runtime and a simpler setup.

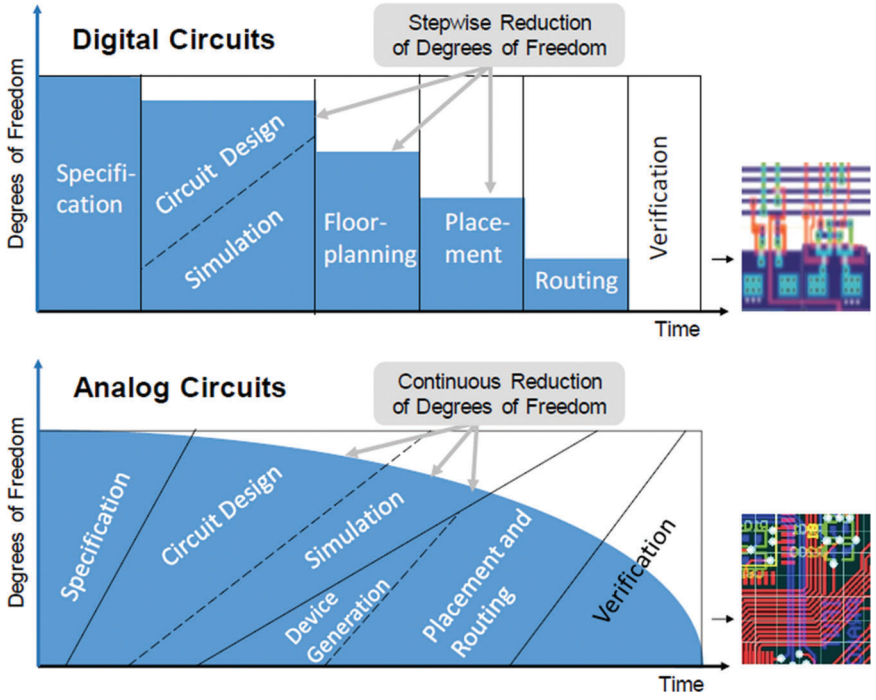


Figure 1.3 Degree of freedom in digital and analog flows [Scheible2015].

The major difference is probably that designers are very familiar with simulator options, because settings like *gmin*, *reltol*, or *maxstep* can be directly linked to electrical measures, so the other tools on top often come sometimes with something you are less familiar with.

To get a feeling for the circuit, designers usually apply many hand calculations and do many sweeps. To cover nonlinearities accurately enough, the sweeps should be dense enough. Especially temperature behavior is often nonlinear, so you would set up a sweep with 10–100 points. Also for the supply voltage, it is often good to hit the transition, when problem starts to appear, accurately enough. Such sweeps are perfect for understanding, but pure sweeps of *one* parameter at a time do not often show well the complete behavior, because of correlations, or mathematical due to *mixed* terms like $x_1 \cdot x_2$ (here the impact of x_1 on f depends on x_2).

Example #1: CMOS logic delay usually increases with temperature due to lower mobility μ. However, at low supplies V_{DD}, this effect can change because the negative TC of the threshold V_{TO} starts to become more important, and at very low supplies (like for hearing aid applications), also the overall TC might be negative, instead of positive! So the usually helpful picture of increasing delay versus temperature gets wrong, just because delay, temperature, and supply are highly correlated and nonlinear. A one-parameter sweep can be captured in a vector for input and output values, but two-parameter correlations need to be captured in a matrix. If we look to 5 discrete values for both parameters, we end up in $5 \cdot 5 = 5^2$ combinations.

Unfortunately, even if you would run all 25 two-parameter combinations, you might still miss some critical cases, because more than two parameters also can form such correlation group! And we do not know exactly which parameter correlations to treat.

Example #2: If we would like to inspect all combinations in our design (like an op-amp), we would have to treat 20 design parameters, 100 statistical parameters, and 5 operational range parameters. For each parameter, we may want to run 5 values, so to get a full picture, we end up in all-in-all $5^{20} \cdot 5^{100} \cdot 5^5$ combinations to simulate. Even if one simulation takes only a second to get all f in $\boldsymbol{f}$, we would end up in a simulation time of more than 7E79 years. Doing this and looking at all results, we would have the guarantee to find the best design values for the given circuit topology, and its behavior under all conditions. For pure verification (i.e., for fix $\boldsymbol{x}_{\mathrm{D}}$), we would only have to cover the two last parts, so we need $5^{100} \cdot 5^5$ simulations or 7E65 years and even brute-force verification (without exploiting any assumptions on the design) is almost impossible.

The biggest part in our example is the statistical part taking 5^{100} points, so using a dedicated statistical technique like Monte-Carlo can already give some speed-up! We will do so and will also discuss the risks. However, even if we have a clever statistical method, we would have to run it for all the range parameter combinations and for design also at each design point. What about numbers? For verification using the sample yield being in the order of 3σ or approximately 99.8%, we need rougly 3,000 MC points for 95% confidence; so we still end up in $3{,}000 \cdot 5^5$ simulations or 3.5 months for *pure verification*. Only such exhaustive or brute-force methods would really give a kind of guarantee for any arbitrary complex and nonlinear design. As this is hopelessly inefficient, we need better methods which really exploit the structure of $\boldsymbol{f}$ by finding in which variables we have high sensitivities, strong nonlinearities, and correlations. This way we can avoid "uninteresting" simulations providing us almost redundant results. We need to compose a clever search strategy that leads us quickly to the design limits. Luckily, this is possible because many circuit design problems are similar.

Of course any such efficient design strategy has both parts which can be applied in general (like doing sweeps) but also adaptive parts (like we need to find out which variables are important and form a group with strong impact on a certain output f). Usually, the variables with the highest nonlinearity cause most pain, e.g., temperature characteristics are often difficult, but even more extreme cases can occur. For instance, you may want no monotony errors in a DAC, but to check this, you may really need to simulate each bit, because such errors may take place anywhere. In such cases, best create a *dedicated* testbench, maybe one with autostop if we have found a monotony error or using an algorithm which starts at a place with the highest fail probability (e.g., around half-input, when the MSB would toggle).

A Very Short History of Statistics and Numerics. Using statistical methods to invest on card games and coin flipping is very old, but in opposite to other mathematical areas like geometry, statistics as science is quite young! For instance, a clear judgment why least-square techniques should be used for fits, and when not, was just given in 1921 by R. A. Fisher. The way statistics are often taught based on the axioms of Kolmogorov dates to 1933! The correct confidence interval method for the mean of a normal distribution was given in 1908 by W. S. Gosset,

under pseudonym "Student"! Of course bigger breakthroughs are done by C. F. Gauss in the nineteenth century, e.g., he solved many difficult physical problems by applying least squares, problems on which, e.g., Leonhard Euler still failed. The central limit theorem has a longer history starting in the eighteenth century, but proof has taken time as well. Monte Carlo techniques came up in 1940s when numerical computers came up more and more. First quasi-Newton optimization algorithms have been invented in the late 1950s. Bootstrap techniques have been created in the late 1970s. The popular latin hypercube sampling method has been described in 1979 by McKay. Advanced worst-case distance methods are even newer. Matrices are an elegant method to collect numbers and equations, and the term came up in 1850 by J. J. Sylvester.

1.2 Engineering Techniques

Engineers are discoverers, hunters and gatherers, seldom dancers, or actors. A first key technique—and maybe even the most important one—is knowing what you want to do and being able to apply your knowledge.

1.2.1 Ground Work and Anticipation

The circuit behavior is usually defined by physical relations like the Ohm's law or the transfer characteristic of a MOSFET or an amplifier. For a block, this usually ends up in a set of equations like the total gain is $A_{\mathrm{tot}} = \prod A_{\mathrm{stage}}$ with $A_{\mathrm{stage}} = g_{\mathrm{m}} R_{\mathrm{L}}$. In nonlinear cases, such equations might be hard to solve for obtaining the element values, so often simplifications are needed, e.g., based on Taylor series. In an ideal op-amp-based amplifier (having an infinite open-loop gain), the (closed-loop) voltage gain is defined by the feedback resistor ratio, like $A_{\mathrm{stage}} = -R_2/R_1$ (Figure 1.4).

Obviously, a design is more robust if it relies on *ratios* instead of *absolute* values, but sometimes it is not so clear, e.g., because the loop gain might be not as high as desired, so that on top of the (resistor) ratio mismatch error, other effects could be present, and even dominating—"bad luck." Also "good luck" is possible, e.g., you may find a clever bias concept to make g_{m} proportional to $1/R_{\mathrm{L}}$ to *cancel* out the absolute variations even in a simple transistor amplifier stage. Via hand calculations, you can typically obtain only some start values for the circuit elements, e.g., for those inside the amplifier or for the RC values of a filter, and finding the really best-suited values requires

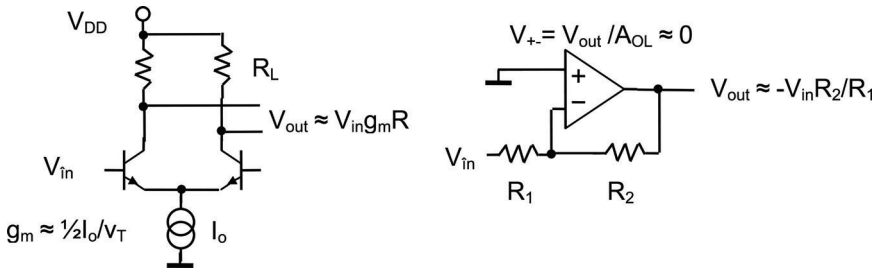


Figure 1.4 Typical transistor amplifier stage and op-amp as inverting voltage amplifier.

some tweaking and resimulations or even multiple prototypes and redesigns, respectively.

1.2.2 Iterative Refinement

Our example clearly shows that iterative refinement is a key technique too: System design may start with simple budget sheets, often entered in spreadsheet programs. At some point, you want to include more effects and running quick simple simulations, e.g., in Mathworks® MATLAB®. Later, in a real circuit design environment, you can switch part by part to more complex models—based on Verilog-A—or to transistor-level circuits which also take loading effects into account. Last you create full layouts, extract parasitic elements, and run really time-consuming sign-off performance verifications, whereas the functional verification and the testbench creation are usually done at a much higher abstraction level. At the end, you can decide whether the design is good enough to make tape-out, i.e., creating an expensive mask set and fabricating the design.

Of course modeling is a key part of the design process and is partly done by modeling experts. Modeling is very helpful in testbench creation, in debugging, and also in the specification phase because having a testbench with models is a kind of "executable specification." It is very helpful for circuit implementation to see how each block should act in the system context, like what are the input signals and the desired outputs. Such "executable specs" help a lot regarding team communication and give also a good status overview. Ultimately, this gives high confidence already in *early* design stages, because it allows to have always something that works and can be demonstrated. All these points are often even more important than the simple simulation speed-up you may get with simpler models compared to transistor-level simulations. For this reason, start the modeling early in a project. Read a bit more about modeling in Section 1.3.2.

1.2.3 Composition in Design

Besides refinement, also composition is important: building of complex systems or blocks by simpler elements. Analog circuit design is a bit like Lego®, and digital is even almost 100% Lego! For instance, you may start directly with a known op-amp circuit topology and optimize just the parameter values, or you may construct a new op-amp:

- Decide on the input stage type according to the input common-mode voltage range (for ground-sensing op-amps, you could use a PMOS input, but no NMOS, and for rail-to-rail signals, you typically need both types or a level shifter) and bandwidth requirements.
- Decide on the number of stages to fulfill the overall gain requirements.
- Choose an output stage based on drive requirements, technology limitations, output voltage range, etc.
- Further decisions could be related to use either a simple class A concept or more power-efficient class AB stages (or even switched-mode amplifiers).

Construction often comes with decisions, and these might be tough to make, because you have to work out each solution to some degree till you are able to make decisions. Decisions are much harder to automate than pure parameter refinements! And analog designers use a lot of different Lego keystones—some are small like a differential pair, and others are complex like a PLL or ADC.

Of course, design tweaking and composition methods are usually in competition, but can also complement well. For instance, if you design a second-order LC lowpass, you know you can get 40 dB/dec, so a certain attenuation for the fifth harmonic. However, in reality, the elements have self-resonances, and with good luck, you can exploit the series inductance of your SMD capacitor and get a much better damping for HD_5! In this case, an optimizer might have found a similar solution, but designer's knowledge could outperform any optimizer in such simple case—but often not in more complex case.

1.2.3.1 Construction vs. optimization

Exploiting the problem structure is usually the key for design efficiency. Optimizers can *partly* act in this way, because they follow a certain *strategy* which can be mathematically even quite optimal (see Chapter 8). In this book, we address *parameter* optimization based on a fix circuit topology, because we want to talk about methods that work in commercial EDA tools.

In several academic papers, true synthesis techniques for analog have also been reported. Usually, these are based on a circuit library and optimization and construction techniques. Construction techniques, which are often rule or knowledge-based, are important too and can be more efficient regarding the number of simulations than pure optimization.

Note that there is, in principle, no clear difference between optimization for component parameters only and optimization which would include topology optimization. You could just give all of your circuits an integer number, and let the optimizer optimize on both this integer and the usual component parameters! However, this is (by far) not the best way, because it just does not exploits the problem structure and nonlinear mixed real integer optimization but is very difficult, thus creating a big burden for the optimizer!

As mentioned, construction is often regarded as an alternative to optimization, e.g., you could try to code [Berkely] your design strategy from spec to circuit for each circuit type—like two-stage op-amp with Miller compensation, NMOS input, folded cascade stage, and PMOS class A output—in a script (e.g., in a programming language like Perl or SKILL®), maybe even including the layout. Unfortunately, such scripts are obviously much harder to create and usually quite limited (at least without optimization), e.g., regarding the specs, you can address as input, and maintenance is a problem too. In addition, it is not easy to make such scripts technology-independent—although interesting approaches at least exist, e.g., by doing the sizing according to $g_\mathrm{m}/I_\mathrm{D}$ technique and by the inclusion of optimization or lookup tables [Iskander2013] (Table 1.1).

It is an interesting question if such fully automated methods will be available in "analog", would they be really well *adopted* by designers? And what about competing methods which may focus on more design *insight*? One current prominent example is the mismatch contribution analysis (see Chapter 5)! Essentially, the whole idea of "awareness" is based not only on "automation" but mainly on avoiding long iteration loops and for getting more insights: for variations, for parasitics, for layout-dependent effects, electromigration, etc.! Also tools like IP management systems have strong user-specific aspect: Any IP system is only as good as the users and administrators are in structuring and maintaining it. Analog designs will probably never be as "simple" as logic design.

Already in existing environments, many companies have made clever extensions to let the designers work in a convenient way, like offering property editors not only for editing but also with immediate feedback for design

Table 1.1 Construction versus optimization-supported flow

	Construction-Based Design	Optimization-Supported Design
Topology definition	Typically in a script	By designer in schematic
Parameters to design	Defined in script	Defined by designer, e.g., supported via contribution analysis
Rules for sizing	Defined in script, e.g., according to $g_\mathrm{m}/I_\mathrm{D}$ method and circuit-specific calculations	To fulfill block performance, support by sizing rules and many other ones (see Chapter 2)
Flexibility	Limited, need script changes	High
Speed	High, because circuit-specific calculations can highly avoid SPICE simulations	Low
Suitability for high-performance designs	Limited, especially if you want to avoid SPICE simulations	Yes

parameters, like for transistors you get immediately after entering width W and length L also a value for the threshold voltage standard deviation based on the process matching constants or for capacitors by setting W and L you will get not only the capacitance but e.g., also the parasitic substrate capacitance and the parasitic series resistor. Implementing more (like giving a layout preview, or displaying key performances like f_T, f_res, Q factor, S_{11}, MAG, noise density, and maximum allowed current—whatever makes sense) is often no big thing. Information at your fingertips is often just work—or a talk with your CAD team! In modern design environments, most customers use only roughly 65% of all tool features they buy for and individuals often even less, so training and continuous improvement is essential.

1.2.4 Team Work and Divide-and-Conquer

In bigger projects, many engineers work together. Usually, some experienced system designers decide on system specs and system partitioning. Once the system topology is defined, we can derive block specifications; however, there is some flexibility in doing that like you can obtain an overall amplification of 1,000 by using either 1 or 2 or 3 amplifiers in a chain. This limits the application of the classical "divide-and-conquer" approach—as maybe number one general design technique. Besides its limitations, in general, this approach is *extremely* successful in chip design because it enables working on

many blocks *in parallel*. Each of the blocks will then be designed in quite a similar way, using similar tools. Also in block design, we often apply divide-and-conquer, e.g., when we split the verification into corner runs, and for treating mismatch we use MC.

1.2.5 Automation and Tools

Automation is an important method as well. Designers love their circuit "babies", and they love and hate tools. Designers hate repeating uninteresting tasks, and usually, it pays out to automate things. I remember the old days where you can just run a single simulation, look to the waveforms, take some notes, and tweak the design. After some time, you inspected the more critical corners on temperature T and supply voltage and made a little table on a sheet of paper. Of course, a circuit is work in progress, so you changed it a bit later, and so the table went out of date and becomes quickly inconsistent! Already using a simulator was some kind of automation, and also setting up a clever testbench is a key part of your everyday work. Nowadays, you can also easily automate your result evaluation interactively with a "few" mouse clicks, with built-in calculator and assistances—sent by heaven. This makes also the application of more advanced techniques much easier, like Monte Carlo analysis.

Automation is not only to support lazy people or just to enable design. Being efficient, creating affordable products, and fighting not only for the best but also for economic products are must for engineers. So automation is a strong driver to reduce costs and becoming more and more important, because the risk for failure is always present—redesigns are becoming even more expensive in modern technologies (e.g., due to increasing mask costs), and unexpected redesigns are one major reason for missing the design-in time window.

You may anticipate many problems—maybe a dozen—like a chess player, but surely at some point in design (still many), things may get wrong. Then, you become a hunter for bugs and you have to debug and improve. In this case, you typically do not know completely what is happening, but you should have a working hypothesis and create tests to check it. In theory, there is no difference between theory and practice, and in practice, there is. This is usually because in "real" problems, we often just have a mix of problems.

In a design project, progress means removing the unknowns step-by-step or at least quantifying their range and minimizing their impacts till

you are confident enough that you can tape-out. This comes not only from simulation and verification plans! You should understand *what* is causing the limited PSSR of your circuit, and with analytical methods like small-signal equivalent circuits, you can calculate your expectations and verify them using power supply sweeps. The same you can often do for other parameters like temperature as well as for other design metrics like offset voltage. Later you will check for critical corner combinations like low-V_{DD} and slow technology corner, best with a sweep on temperature on top. Or you will set up an MC analysis to check for the production yield. And if your yield target is high, you may switch to special high-yield estimation techniques; and over-all analog design bases also heavily on experience. For luck all many tools do not only provide automation, but many can bring also much insight to the designer. So it is not only "tool speed-up versus costs" that matters.

Last but not least: I remember, in a big tool demo provided by our leading experts at the end, this question came up:

> *OK, we saw that great demo, but what else do you still need to do? Can we use it already?*

The answer was nice too:

> *Next step is to enable designers that they can do what I have shown!*

Indeed, in making real EDA tools, this aspect is important as well, because if something is difficult to use or confusing, analog designers will not use it. This also points out well that education and training are indeed key points in becoming and staying a good engineer! In fact, some techniques like Monte Carlo are quite old, but still there is a lot of confusion in MC result interpretation! So from time to time, engineers should stop in following the usual habits and focus on things which may look boring or confusing at first glance.

1.2.6 Re-Use in Designs

A last "last but not least" might be "do not reinvent the wheel." If you already know a good solution from experience, then it is often best to reuse it and to focus on other problems. Why applying optimization on a low-performance circuit with known design strategy and well-defined construction steps? Luckily, a big part of design work can be already simplified by pure reuse, e.g., using existing Verilog-A models or testbenches for standard circuits

like bandgap, op-amp, ADCs, or voltage regulators. Also analog designs can reuse some circuit blocks like digital gates and flip-flops, or you may use layout macros for differential pairs or current mirrors, etc. Further examples of reuse are the use of macro compilers (for scalable memory blocks, etc.), sharing verification templates in the team which collect known critical corner conditions. In Chapter 10 we will further describe IP and re-use techniques.

1.2.7 Summary

The sweetspot of EDA tools is usually accuracy (like being able to treat very detailed and complex models), and capacity (doing calculations fast). However, tools are not very good in following most of these *manual* approaches (Figure 1.5). For instance, only slowly advanced partitioning techniques are available in EDA tools, usually for becoming even faster and to enable application to extremely complex systems. Simple examples are Fast-SPICE simulators and parameter screening techniques in an optimizer for calculating worst-case distances.

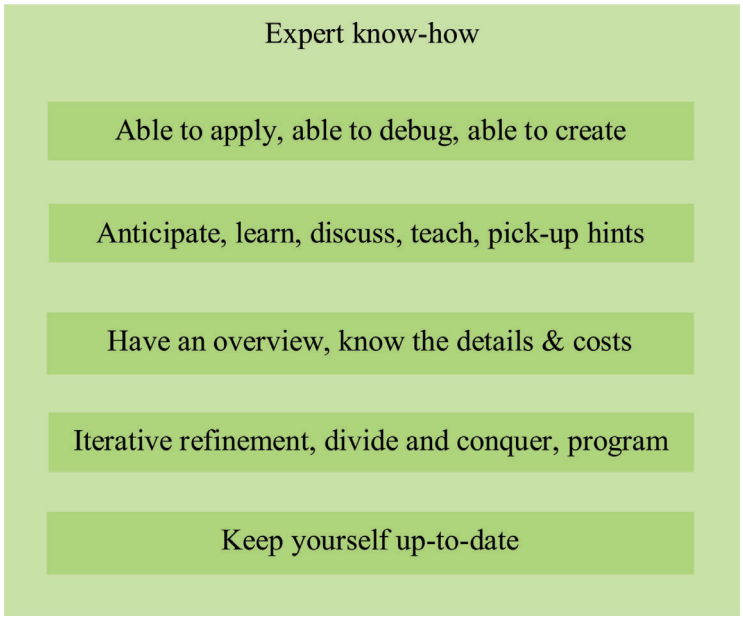


Figure 1.5 Engineering core competences.

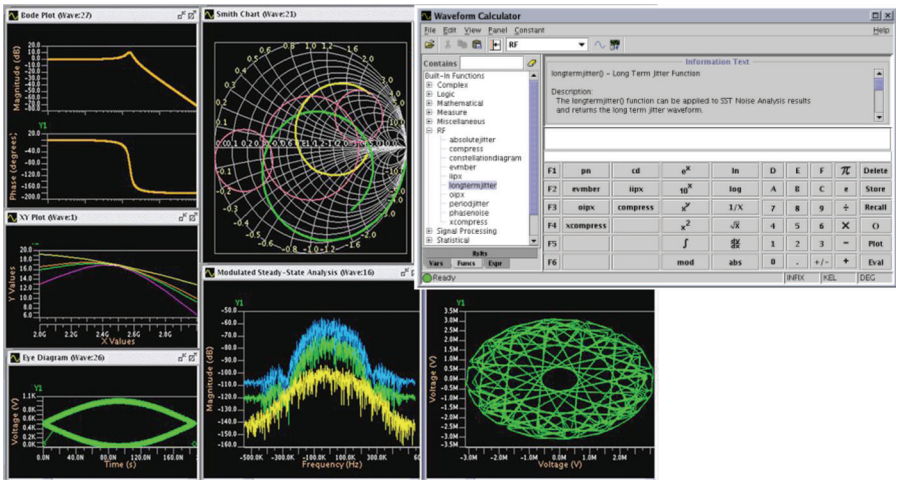


Figure 1.6 Simulation plots quickly created from a special calculator (EZWave^{TM}, Courtesy Mentor Graphics).

Usually the decision about te next design step and which circuit should be used is up to the designer, not to the tool. At best, an optimizer can optimize multiple predefined circuits in parallel, and then it can hand out the best solution found. Only in academic research, true circuit topology optimization indeed exists already. Debugging of circuits and construction are still almost beyond the scope of EDA tools, but of course all the software is also designed to highly support these tasks. For instance, special calculators are available to derive standard circuit performance measures (like 3dB-bandwidth or 10%–90% risetime, and much more) quickly from simulation data (Figure 1.6, not shown is the comfortable graphical stimuli editor). So, since roughly 1985 IC design is a clever mix of manual and semi-automated techniques.

All over the world, engineers have made tools to support you in solving problems and these use the same engineering techniques as described. Often you just have to read the software manuals or ask the EDA vendor for a product update presentation.

1.3 Key Elements and Aspects in Circuit Design

Let us now take a look to further elements in design, specific to circuit and IC design. A native starting point is of course a datasheet.

1.3.1 Datasheets, Conditions, and Trade-Offs

The datasheet is the key document for any electrical device, as target datasheet is often the base for future products and discussions. Here, you promise certain functionality and characteristics. The circuit simulations during the design phase help to find the best-suited circuit and also allow a virtual verification based on simulation models, but for this we need clear guidance for efficient work. Usually, the datasheet reports both the typical performance and the guaranteed minimum performance; and it defines a bunch of testbenches. Often a performance can be defined in different ways, e.g., in terms of power in Watt or in dBm. Usually, the designer set up tests up in a convenient way, e.g., fitting to measurement equipment and to get numbers easy to handle. The latter is also important for numerical algorithms, e.g., the period of an oscillator could be infinite, just in case that the oscillator does not work. To avoid infinite numbers, better use the oscillator frequency $f = 1/T$. Of course, terms of "pass" versus "fail" and for the yield, the unit does not matter at all, but for other kind of data analysis or for optimization, it does!

Of course, a circuit should not only work at nominal conditions but also provide correct operation in a certain *range* of important environmental parameters such as temperature, supply voltage, and load capacitance. In older environments often designers spend many hours to collect simulation data and to create spec *compliance* tables for reviews and for documentation, but since several years this is a feature provided automatically (in the user interface and e.g., as HTML as CSV file) in most EDA environments (Figure 1.7). With context-sensitive menus or additional buttons also many more options are available, such as backannotations to schematic, plotting window access, sorting and filtering features, log file access, selection of a subset of corners for debugging, automatic datasheet generation, etc.

Often there is confusion about which performances are required under which conditions; a small change can have a big impact on whether the design is easy to create or almost impossible! For instance, a small change in the input voltage range of a DC–DC converter could impact the whole topology (buck vs boost vs buck-boost) and pin-out. Clarify these points *early* and *explicitly* in a verification plan, e.g., as appendix to the target datasheet.

When designing a product, you have to make many trade-offs, e.g., you can make a product cheaper using a simple process technology (like pure digital CMOS process), but this can make the design (much) more difficult, because older processes offer usually only moderate bandwidths.

	Variable	Nominal					PVT0	PVT1	PVT2	PVT3	PVT4	PVT5
	Vcc	1.5					1.7		1.9		1.7	
	gpdk090.scs	NN					SS				FF	
	Temperature	27					-40	100	-40	100	-40	100
Testbench	**Output**	**Value**	**Spec**	**Pass/Fail**	**Min**	**Max**	**Value**	**Value**	**Value**	**Value**	**Value**	**Value**
Test1	BW05dB	8.25M	> 5M	pass	7.94M	16.5M	11.95M	8.519M	16.55M	10.38M	9.889M	7.936M
	BW1dB	12.5M		n.a.	12.2M	94.8M	19.52M	13.76M	94.84M	71.29M	15.04M	12.24M
	BW3dB	101.8M	> 20M	pass	77.9M	110.4M	110.4M	82.84M	103.9M	77.87M	103.4M	85.95M
	BWfromtr	19.0M		n.a.	17.5M	45.1M	37.88M	41.78M	44.28M	45.1M	23.4M	17.54M
	HD2	66.2	> 60	**4 fails**	48.35	71.7	71.71	66.42	67.39	62.11	57.61	52.22
	HD3	91.58	> 70	pass	73.4	110.9	100.2	100.8	110.9	88.35	82.84	75.91
	Icc	143u	(100u;1m)	pass	141.8u	167u	144.8u	141.8u	150.1u	145.3u	158.1u	151.9u
	Vinoise(1M)	606n	< 1u	pass	265.3n	817n	460.2n	686.6n	513.2n	715.7n	265.3n	770.7n
	LoopGain DC	50.83	> 50	**6 fails**	29.9	55.4	55.41	50.21	49.92	44.65	40.54	34.98
	Peaking	971m	< 1	**4 fails**	0	4.57	3.449	4.572	3.812	4.411	125.5f	661.1m
	PM	73.9	> 70	**4 fails**	65.3	73.9	69.59	70.22	65.34	67.03	72.64	72.47
	PSRR(10M)	4.698	> 0	pass	3.69	6.21	6.132	3.921	6.211	3.859	5.404	3.692
	PSRR(DC)	46.57	> 30	**1 fail**	28.9	49.1	49.15	45.28	45.11	40.94	38.62	33.67
	t05percent	50.12n		n.a.	36.2n	74.9n	46.54n	68.64n	49.69n	74.94n	39.21n	56.44n
	t1percent	44.9n	< 40n	**fail**	31.2n	60.8n	31.24n	57.33n	38.03n	60.76n	35.65n	41.92n
	trise	18.4n	< 30n	pass	7.76n	19.96n	9.24n	8.377n	7.904n	7.76n	14.96n	19.96n
	Voffset	1.22m	(-25m;25m)	pass	1.18m	7.48m	1.178m	2.383m	1.868m	3.426m	2.229m	4.24m
	Summary	**1 fail**		**fail**			**1 fail**	**2 fails**	**3 fails**	**4 fails**	**2 fails**	**3 fails**

Figure 1.7 Typical compliance table for a corner analysis (first 6 corners only).

For a given technology, you can often select a high-speed parallel architecture, but that usually consumes more power and occupies a larger chip area. In many cases, some overdesign with respect to performance is possible, so that you will be on the safer side (e.g., on noise, offset, and distortion), but for critical blocks, this is harder and leads usually to the use of more power consumption and chip area, so your design will not be competitive! Underdesign is risky, maybe you can still keep the specs, but the yield may drop significantly or you will be out-of-spec and need a redesign.

1.3.1.1 Trade-off examples

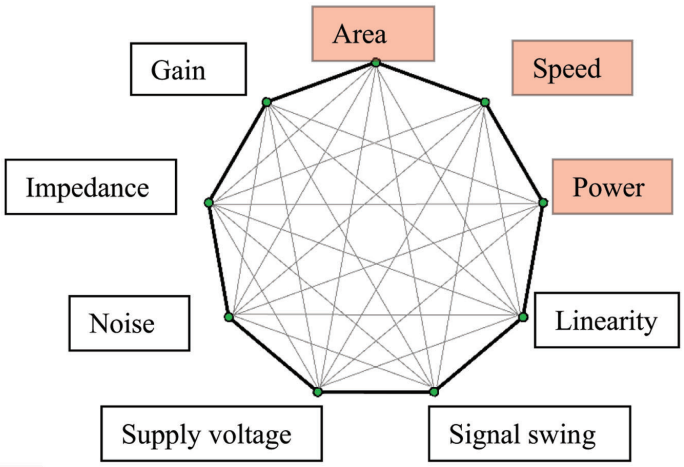
In analog, mixed-signal, or RF, there are generally many compromises, requiring much experience and careful well-organized work is required. Figure 1.8 shows the major trade-offs, but there can be even more (like costs and stability) or some need to be split up (like distortion into odd and even order, or speed into rise time, fall time, delay, bandwidth, and settling time).

Note: The green connections in Figure 1.8b show which performances have *positive* correlation (like unity-gain frequency and power), and the red ones show negative correlations (like phase margin versus gain). But look up, also the positive correlations often compete, because it also matters whether we have upper or lower spec limits.

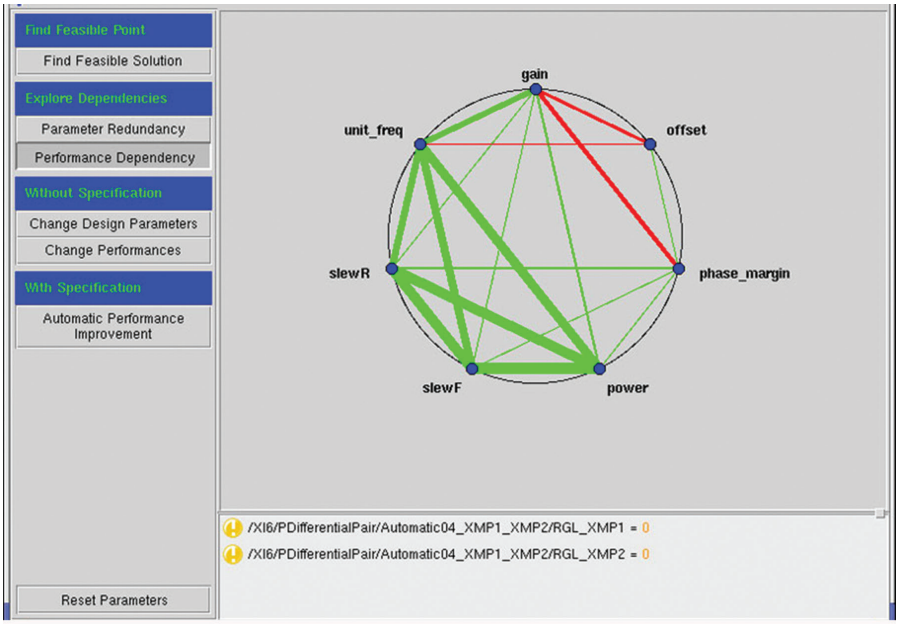
Trade-off examples:

- Low noise is often a key requirement and is often directly related to bias currents (so power) and device area (especially for flicker noise).
- Also linearity and output range are related to bias currents and of course also to supply voltages.
- Low offset voltages (and good DC accuracy in general) require large area devices to minimize mismatch, but this increases the chip area, and it also leads to speed restrictions (or increasing power).
- Often high DC accuracy and low distortion come in sync, but at higher frequencies, they can also compete due to reduced loop gains.
- If you want a certain output impedance (often required for RF circuits), it may give severe restrictions on the supply voltage or your impedance transformation networks, which unfortunately need some area, limit the bandwidth, and reduce efficiency.

In Chapter 2, we pick up the trade-off topic when discussing the typical manual design flow and transistor sizing.



(a)



(b)

Figure 1.8 Typical design trade-offs (red=digital) and circuit-specific tool output (Courtesy of MunEDA, red=fighting specs).

Testing versus "Guaranteed by Design" Chips would be very expensive if really everything would be *tested* in hardware under all environmental conditions. Therefore, production tests are usually done only at room temperature and some critical conditions. For this reason, most datasheets are split, e.g., in a part describing the performance at 25°C with usually quite tight tolerances and parts for the characteristics over a wider range of T, V_{DD}, R_{L}, etc. Detailed tests under these wider conditions are usually done only from time to time, in laboratory, not during production. This way many specs are not really guaranteed by 100% testing, but "by design." Only for very expensive components, it is affordable to really perform a near-100% production test, like for military or spacecraft applications.

1.3.1.2 Datasheet contents

Datasheet for commercial products could also serve well as a reference for blocks on an integrated circuit. Let us do so by inspecting the datasheet of a commercial high-performance operational amplifier (excerpt, Courtesy of Texas Instruments, for the complete information go to http://www.ti.com/lit/ds/symlink/opa1612.pdf).

In the same way we can also create a design documentation e.g., of an op-amp block in an ASIC. For instance, we can look to several commercial examples, or we may use a datasheet template generator (see Figure 1.9).

An official target datasheet is usually quite complete from the pure customer viewpoint (at least you have to convince the customer), but some key characteristics for yourself are usually missing like yield and worst-case corners. Also it is usually not defining chip area, block shapes, bonding diagrams, and second-order effects like substrate noise. A real complete datasheet in the "IC-design sense" is good for documentation purposes, but also to support other designers in your team, to make a designer review or just to learn. This way also the reuse of the block can be made much easier. Some specs are typically not interesting for customers, but very important to know internally. Actually, for the customers, maybe the guaranteed minimum performance matters, just to fulfill system specs, but in other applications, it may matter if your performance variations, e.g., in an ADC, are due to temperature or supply voltage or due to mismatch, and for pure ADC design, maybe just the total variation matters. However, if you want later to reuse the design for a multichannel or IQ ADC application, the mismatch is usually more critical, compared to temperature effects. For such reason, documentation can

TEXAS INSTRUMENTS
OPA1611, OPA1612
SBOS450C – JULY 2009 – REVISED AUGUST 2014

OPA161x SoundPlus™ High-Performance, Bipolar-Input Audio Operational Amplifiers

1 Features

- Superior Sound Quality
- Ultralow Noise: 1.1 nV/√Hz at 1 kHz
- Ultralow Distortion: 0.000015% at 1 kHz
- High Slew Rate: 27 V/µs
- Wide Bandwidth: 40 MHz (G = +1)
- High Open-Loop Gain: 130 dB
- Unity Gain Stable
- Low Quiescent Current: 3.6 mA per Channel
- Rail-to-Rail Output
- Wide Supply Range: ±2.25 V to ±18 V
- Single and Dual Versions Available

2 Applications

- Professional Audio Equipment
- Microphone Preamplifiers
- Analog and Digital Mixing Consoles
- Broadcast Studio Equipment
- Audio Test And Measurement
- High-End A/V Receivers

3 Description

The OPA1611 (single) and OPA1612 (dual) bipolar-input operational amplifiers achieve very low 1.1-nV/√Hz noise density with an ultralow distortion of 0.000015% at 1 kHz. The OPA1611 and OPA1612 offer rail-to-rail output swing to within 600 mV with a 2-kΩ load, which increases headroom and maximizes dynamic range. These devices also have a high output drive capability of ±30 mA.

These devices operate over a very wide supply range of ±2.25 V to ±18 V, on only 3.6 mA of supply current per channel. The OPA1611 and OPA1612 op amps are unity-gain stable and provide excellent dynamic behavior over a wide range of load conditions.

The dual version features completely independent circuitry for lowest crosstalk and freedom from interactions between channels, even when overdriven or overloaded.

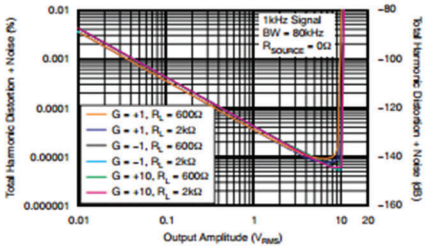
Both the OPA1611 and OPA1612 are available in SOIC-8 packages and the OPA1612 is available in SON-8. These devices are specified from –40°C to +85°C.

Device Information(1)

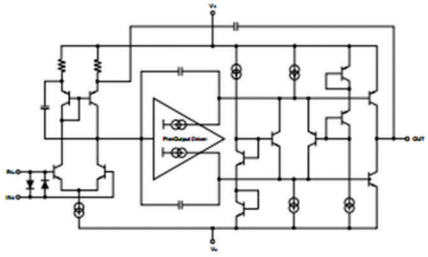
PART NUMBER	PACKAGE	BODY SIZE (NOM)
OPA1611	SOIC (8)	4.90 mm × 3.91 mm
OPA1612	SOIC (8)	4.90 mm × 3.91 mm
	SON (8)	3.00 mm × 3.00 mm

(1) For all available packages, see the orderable addendum at the end of the datasheet.

THD+N Ratio vs Output Amplitude



Functional Block Diagram



6 Specifications

6.1 Absolute Maximum Ratings

over operating free-air temperature range (unless otherwise noted)(1)

		MIN	MAX	UNIT
Supply voltage	V_S = (V+) – (V–)		40	V
Input voltage		(V–) – 0.5	(V+) + 0.5	V
Input current (all pins except power-supply pins)			±10	mA
Output short-circuit(2)			Continuous	
Operating temperature	(T_A)	–55	+125	°C
Junction temperature	(T_J)		200	°C

(1) Stresses beyond those listed under *Absolute Maximum Ratings* may cause permanent damage to the device. These are stress ratings only, which do not imply functional operation of the device at these or any other conditions beyond those indicated under *Recommended Operating Conditions*. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.
(2) Short-circuit to V_S / 2 (ground in symmetrical dual supply setups), one amplifier per package.

6.2 Handling Ratings

			MIN	MAX	UNIT
T_{stg}	Storage temperature range		–65	+150	°C
$V_{(ESD)}$	Electrostatic discharge	Human body model (HBM), per ANSI/ESDA/JEDEC JS-001, all pins(1)	–3000	3000	V
		Charged device model (CDM), per JEDEC specification JESD22-C101, all pins(2)	–1000	1000	
		Machine model (MM)	–200	200	

(1) JEDEC document JEP155 states that 500-V HBM allows safe manufacturing with a standard ESD control process.
(2) JEDEC document JEP157 states that 250-V CDM allows safe manufacturing with a standard ESD control process.

6.3 Recommended Operating Conditions

over operating free-air temperature range (unless otherwise noted)

	MIN	NOM	MAX	UNIT
Supply voltage (V+ – V–)	4.5 (±2.25)		36 (±18)	V
Specified temperature	–40		+85	°C

6.4 Electrical Characteristics: V_S = ±2.25 V to ±18 V

At T_A = +25°C and R_L = 2 kΩ, unless otherwise noted. V_{CM} = V_{OUT} = midsupply, unless otherwise noted.

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
AUDIO PERFORMANCE						
THD+N	Total harmonic distortion + noise	G = +1, f = 1 kHz, V_O = 3 V_{RMS}		0.000015%		
				–136		dB
IMD	Intermodulation distortion	SMPTE/DIN two-tone, 4:1 (60 Hz and 7 kHz), G = +1, V_O = 3 V_{RMS}		0.000015%		
				–136		dB
		DIM 30 (3-kHz square wave and 15-kHz sine wave), G = +1, V_O = 3 V_{RMS}		0.000012%		
				–138		dB
		CCIF twin-tone (19 kHz and 20 kHz), G = +1, V_O = 3 V_{RMS}		0.000008%		
				–142		dB
FREQUENCY RESPONSE						
GBW	Gain-bandwidth product	G = 100		80		MHz
		G = 1		40		MHz
SR	Slew rate	G = –1		27		V/µs
	Full-power bandwidth[1]	V_O = 1 V_{PP}		4		MHz
	Overload recovery time	G = –10		500		ns
	Channel separation (dual)	f = 1 kHz		–130		dB
NOISE						
	Input voltage noise	f = 20 Hz to 20 kHz		1.2		μV_{PP}
e_n	Input voltage noise density[2]	f = 10 Hz		2		$nV/\sqrt{Hz}$
		f = 100 Hz		1.5		$nV/\sqrt{Hz}$
		f = 1 kHz		1.1	1.5	$nV/\sqrt{Hz}$
I_n	Input current noise density	f = 10 Hz		3		$pA/\sqrt{Hz}$
		f = 1 kHz		1.7		$pA/\sqrt{Hz}$
OFFSET VOLTAGE						
V_{OS}	Input offset voltage	V_S = ±15 V		±100	±500	µV
dV_{OS}/dT	V_{OS} over temperature[2]	T_A = –40°C to +85°C		1	4	µV/°C
PSRR	Power-supply rejection ratio	V_S = ±2.25 V to ±18 V		0.1	1	µV/V
INPUT BIAS CURRENT						
I_B	Input bias current	V_{CM} = 0 V		±60	±250	nA
		VCM = 0 V, DRG package only		±60	±300	nA
	I_B over temperature[2]	T_A = –40°C to +85°C			350	nA
I_{OS}	Input offset current	V_{CM} = 0 V		±25	±175	nA
INPUT VOLTAGE RANGE						
V_{CM}	Common-mode voltage range		(V–) + 2		(V+) – 2	V
CMRR	Common-mode rejection ratio	(V–) + 2 V ≤ V_{CM} ≤ (V+) – 2 V	110	120		dB
INPUT IMPEDANCE						
	Differential			20k || 8		Ω || pF
	Common-mode			10^9 || 2		Ω || pF

(1) Full-power bandwidth = SR / (2π × V_P), where SR = slew rate.
(2) Specified by design and characterization.

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
OPEN-LOOP GAIN						
A_{OL}	Open-loop voltage gain	(V–) + 0.2 V ≤ V_O ≤ (V+) – 0.2 V, R_L = 10 kΩ	114	130		dB
		(V–) + 0.6 V ≤ V_O ≤ (V+) – 0.6 V, R_L = 2 kΩ	110	114		dB
OUTPUT						
V_{OUT}	Voltage output	R_L = 10 kΩ, A_{OL} ≥ 114 dB	(V–) + 0.2		(V+) – 0.2	V
		R_L = 2 kΩ, A_{OL} ≥ 110 dB	(V–) + 0.6		(V+) – 0.6	V
I_{OUT}	Output current		See Figure 27			mA
Z_O	Open-loop output impedance		See Figure 28			Ω
I_{SC}	Short-circuit current			+55		mA
				–62		mA
C_{LOAD}	Capacitive load drive		See Typical Characteristics			pF
POWER SUPPLY						
V_S	Specified voltage		±2.25		±18	V
I_Q	Quiescent current (per channel)	I_{OUT} = 0 A		3.6	4.5	mA
	I_Q over Temperature[3]	T_A = –40°C to +85°C			5.5	mA
TEMPERATURE RANGE						
	Specified range		–40		+85	°C
	Operating range		–55		+125	°C
θ_{JA}	Thermal resistance, SOIC-8			150		°C/W

(3) Specified by design and characterization.

6.5 Typical Characteristics

At T_A = +25°C, V_S = ±15 V, and R_L = 2 kΩ, unless otherwise noted.

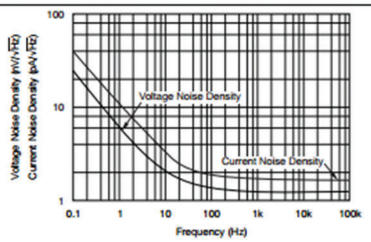


Figure 1. Input Voltage Noise Density and Input Current Noise Density vs Frequency

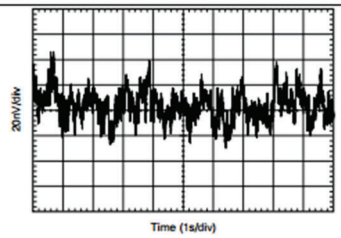


Figure 2. 0.1-Hz to 10-Hz Noise

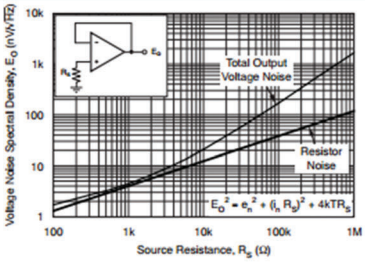


Figure 3. Voltage Noise vs Source Resistance

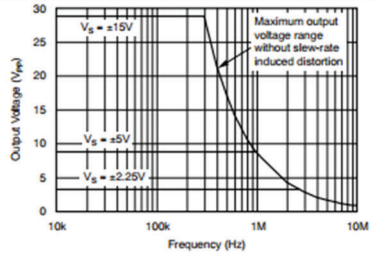


Figure 4. Maximum Output Voltage vs Frequency

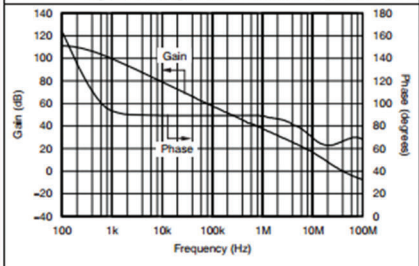


Figure 5. Gain and Phase vs Frequency

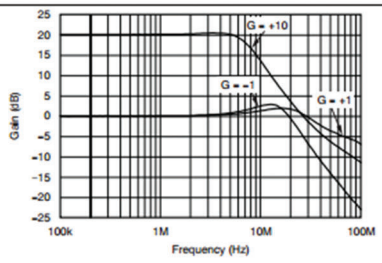


Figure 6. Closed-Loop Gain vs Frequency

Typical Characteristics (continued)

At T_A = +25°C, V_S = ±15 V, and R_L = 2 kΩ, unless otherwise noted.

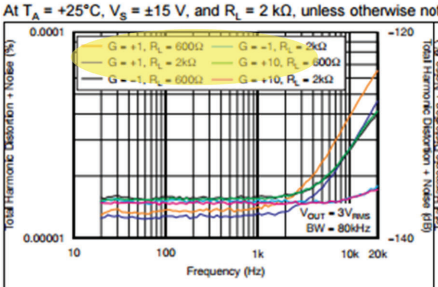


Figure 7. THD+N Ratio vs Frequency

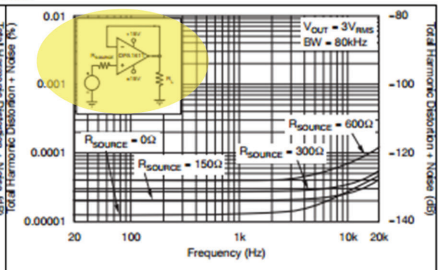


Figure 8. THD+N Ratio vs Frequency

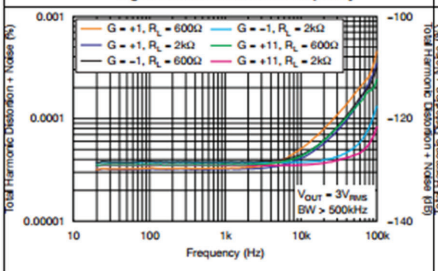


Figure 9. THD+N Ratio vs Frequency

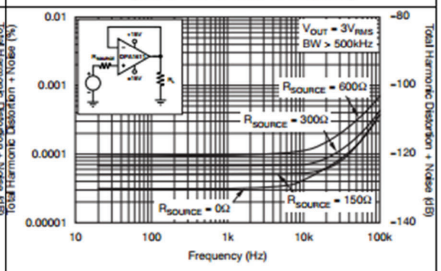


Figure 10. THD+N Ratio vs Frequency

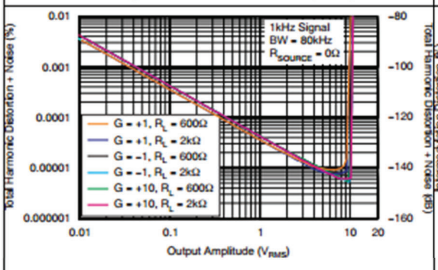


Figure 11. THD+N Ratio vs Output Amplitude

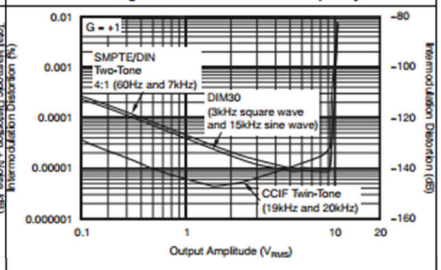


Figure 12. Intermodulation Distortion vs Output Amplitude

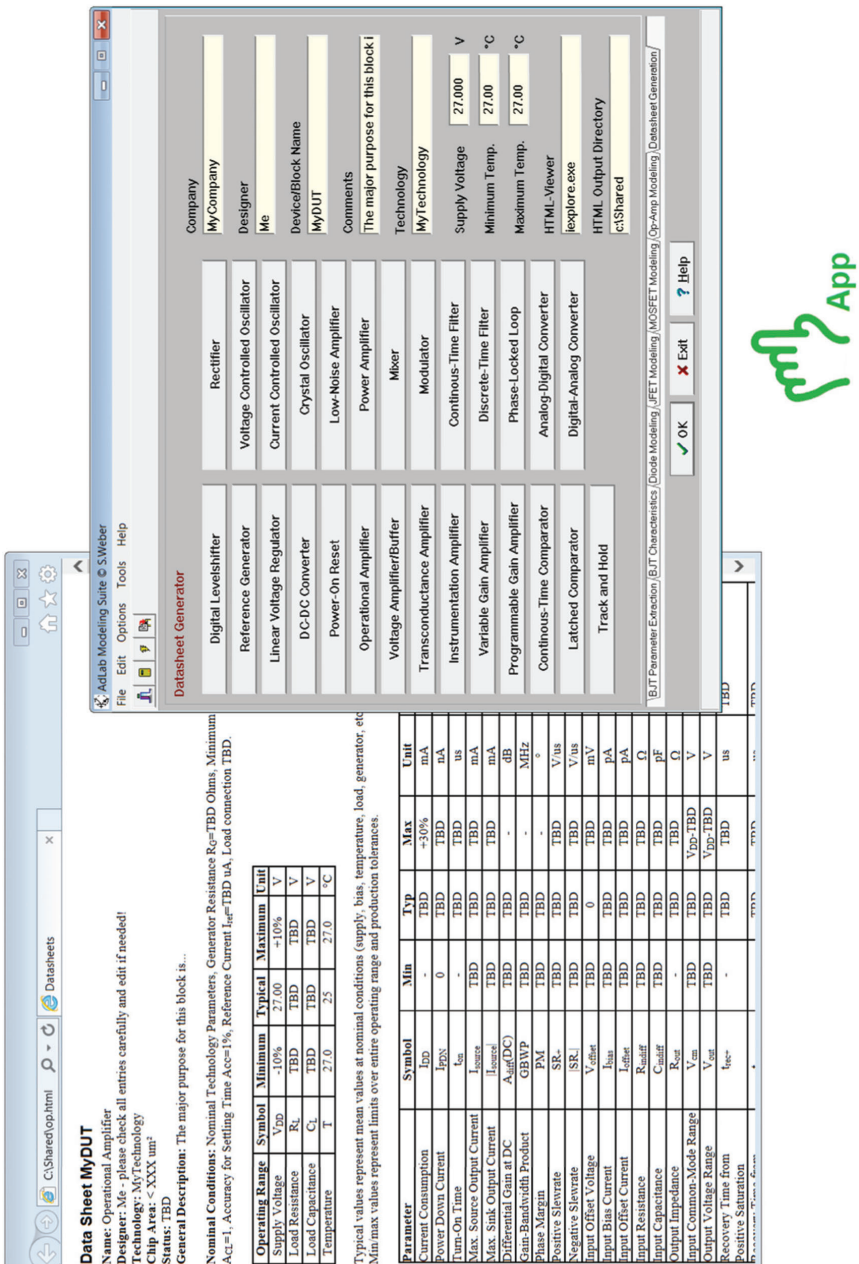


Figure 1.9 Freeware datasheet generator and its HTML output.

be hardly too good! Also it is convenient to see *where* improvements make sense, e.g., where you have already followed best practices and reached the state-of-the-art.

Often not all specs are fully confirmed by the customer, or you may want to add internal specifications, for documentation purposes or to avoid design iterations. For instance, in a system, only the overall offset or noise figure may matter, but to understand the design, it could be also interesting to know about the offset voltage generated in each amplifier stage. In addition, you may want to limit layout-depending effects on offsets. Or you have a spec on bandwidth, and by anticipating critical nets, you may want to limit the parasitics at several internal nets.

1.3.2 Modeling Is Key

This is not a book about modeling or about simulation, but very often a project failure is due to bad or even "lack" of modeling. Actually, if you do "nothing", assume "no model", then you typically implicitly assume a too ideal model, just a bad model. Even if you have no good model(s) e.g., for device mismatch or package inductance, it is a stupid idea to assume no mismatch or no inductance! It is indeed a good method to start with something almost ideal, but then also check the design with the use of realistic models; do it soon, and step by step.

Already when started using simulation techniques in the 1960s, many things rely on modeling (Table 1.2), and of course also for hand calculations you would use models. Actually, mathematically any function can be interpreted as a model, there might be a strong physical background, like for structural models, or even no direct meaning at all, just a fit. In this chapter, we focus more on the first type of models, but for some design methods like corner or sensitivity investigations also pure mathematical models, pure response models have their benefits.

Luckily, the device models have been improved a lot over the years, and partially, you can trust them more than measurements. On the other hand, *more and more* things rely on modeling, not only the simulator results, but also the way the outputs vary, e.g., in a Monte Carlo analysis. So not only accurate IV + CV and noise modeling is essential, but also accurate *statistical* modeling! Luckily, also the MC models have been improved a lot over the years. In fact, the more physically based the model is, the easier the statistical modeling will be, e.g., in the simple old bipolar Gummel–Poon model,

Table 1.2 Different model type for circuit design

Type	Tool	Comment
Device models	Circuit simulator	e.g., classical SPICE models, built-in to the simulator
Statistical models	Circuit simulator e.g., for Monte-Carlo simulation, dedicated statistical tool	Describing the parameter variations of the device models
Behavioral models	Circuit simulator	e.g., Verilog-A models for blocks to get a speed-up over transistor-level simulations or to test ideas or system performance quickly and without having a full implementation
Auxiliary models	e.g., for substrate, package, parasitics, aging, etc.	e.g., SPICE subcircuits

the parameter "IS" is quite difficult to model because it is *not* related to a single physical property! The opposite is true e.g., for the oxide thickness of a MOSFET—here, we can expect much less impacts and correlations with other parameters like doping concentration, bandgap voltage, or sheet resistances. For this reason, the accuracy of most models found in modern PDKs is quite good, although for sure some deviations to reality exist. For instance, often a uniform, normal, or lognormal distribution is assumed. Often this fits to a simplified physical theory, but frequently it is only a meaningful or just acceptable fit to measurement data.

The foundries monitor the process continuously by making process control measurements (PCM). The results will be double-checked in simulation (Figure 1.10 from [Pieper2008]) by just using the same testbenches as in the fab, e.g., on sheet resistance, capacitances, saturation currents, and small circuits like ring oscillators. Based on PCM results the foundry can make sure that only good wafers will be delivered to the customer. Often it is good to be in tight contact with the technologists. I remember in a new process the fab had problems with the current gain β of the new vertical pnp device, but luckily our new circuit was robust enough to work accurate even with a very low β. So instead of throwing away the wafers, we were able to deliver our customer.

Variations may come for different physical reasons, so process variations in general can be classified as random and non-random (e.g., temperature or age),

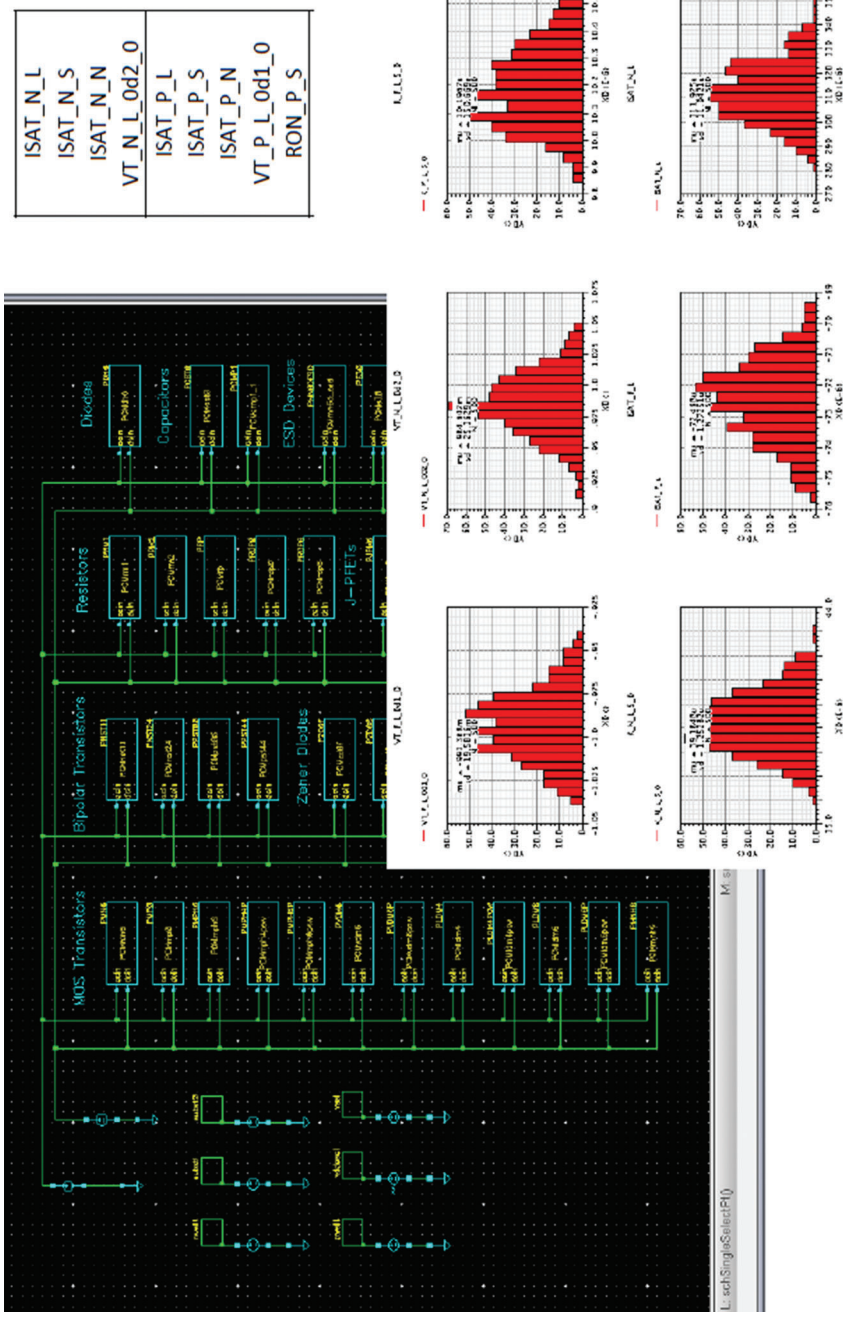


Figure 1.10 Typical foundry PCM testbench and histograms (courtesy Infineon Technologies).

and statistical effects are usually split into intra-die and inter-die variations. For circuit designers, this level of classification is usually enough and models for this are usually available from the foundry in the process development kit PDK. Actually, for quality investigations, also a deeper split into lot-to-lot, wafer-to-wafer, and die-to-die variations makes sense. Statistical variations might be independent or correlated (Figure 1.11).

Usually, an additive law is assumed for parameters:

$$p = p_0 + p_{\text{process}} + p_{\text{mismatch}} \tag{1.1}$$

where p_0 is the nominal value of the parameter (but it might be a function of temperature), p_{process} models the global variations and is shared among all instances on your chip, and p_{mismatch} is intra-die variation specific to instance. Physically, the mismatch depends on the distance between the instances and also on layout details, but as in front-end design, the layout is often not yet defined and these details are typically ignored. They are also not that large *if* you follow good layout practices, like having the same orientation for devices which should match well.

Typically, process variations on threshold voltage V_{TO} are not much depending on device sizes and are often larger than mismatch variations, but the latter become larger for smaller devices. Knowing this, designers can create quite accurate circuits if they can manage that global process variations cancel out! This is done in structures like differential pairs or current mirrors, so that in these now the mismatch dominates. Another key technique to reduce variations is calibration, e.g., one time (in production test), dynamically (e.g., switching to a calibration mode), or sometimes even in the background.

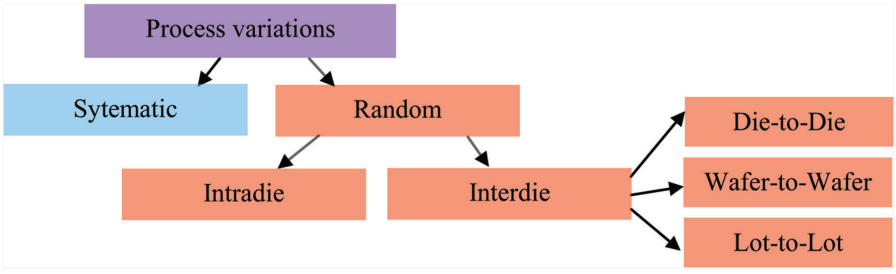


Figure 1.11 Typical classification for circuit design.

Note: For some parameters like leakage currents, often a multiplicative law is used and lognormal statistical parameters. This makes an analysis typically not more difficult; it actually even helps because the voltage across a PN junction follows typically a logarithmic law so that it would show a normal Gaussian distribution because the exponential function in the lognormal distribution and the logarithmic junction behavior would cancel each other! For external components, it is usually more realistic to assume a uniform distribution, not a Gaussian, although sometimes discrete elements have also very strange distributions, e.g., if you buy $\pm 5\%$ SMD components, it is not unlikely that the $\pm 1\%$ samples are sorted out and sold for a higher price!

Designers should check all models carefully, because sometimes one kind of resistor or transistor is only "better" (e.g., on mismatch or temperature coefficient) due to bad and too simple modeling! Usually, "special" things like noise, mismatch, or breakdown are not treated well in seldom used or special components (like native or low-V_{TO} transistors or coils). A further problem is often that more extreme devices like very small or very big ones are modeled not as good as typical devices. One example for this could be mismatch modeling, and often the simple $\sqrt{\text{A}}$-law is assumed and implemented in the model files (Figure 1.12), more complex, more accurate models are usually only available in advanced technologies like 28 nm CMOS or lower (although of course highly advanced models could be also created for older technologies).

Discrete versus Chip Design. In principle there is no big difference between a discrete design, e.g., using SMD components, and IC design, but if you really exploit the advantages of each you can end up in many differences. If you need high accuracy elements, you can choose e.g., discrete components with tighter tolerances, spending a bit more money for critical parts. In IC design you have to live with quite large process variations and some area and component-type dependent mismatch. So for discrete designs Equation (1.1) becomes easier: We have no real process *tracking* for element tolerances, but of course we *have* absolute tolerances causing also mismatch, e.g., between two SMD resistors. In discrete designs, the manufacturer usually guarantees a certain maximum tolerance (like $\pm 5\%$), whereas e.g., in typical IC technologies we have e.g., $\pm 15\%$ from technology and $\pm 1\%$ from mismatch (being usually differential normal distributed). IC designers can often build very good

pair stages, whereas in discrete designs two packaged transistors (or resistors, capacitors, etc.) never match very well. Of course discrete designers have much more freedom regarding element choice, e.g., we can choose a dual-transistor or a transistor array, or even a full op-amp in an 8-pin package. In some aspects IC designers are much more limited, e.g., on-chip inductors cannot really compete with SMD coils on Q-factor or current handling capability. It is also hard to create IC technologies which have both very small transistors (for optimum logic and memory implementation with lowest costs) and e.g., power elements (e.g., able to handle 40 Volts or more). At some point a very universal IC technology would become too expensive, so that e.g., a multi-chip system make much more sense, often also regarding design time, flexibility, time-to-market, etc. Another aspect is design methodology: of course discrete designs are quite easy to breadboard, but in IC design intensive simulations are almost a MUST for verifications.

```
library mos090
section stat_mis
parameters
+ pvt_mc=0
+ pu0_mc=0
+ plw_mc=0

statistics {
 mismatch {
  vary pvt_mc  dist=gauss std=1/1
  vary pu0_mc  dist=gauss std=1/1
  vary plw_mc dist=gauss std=1/1
  }  }
endsection stat_mis

…..

inline subckt pmos1v  (D G S B)
parameters l=0.1u w=10u M=1 nrd=s1v_hdif_pe/w nrs=s1v_hdif_pe/w  as=1p ad=1p ps=1u pd=1u
+ varvt =  .0029   // 1 sigma Vt mismatch variation in unit of v-um
+ geo_fac =  0.7071 / sqrt(l*w*M*1e12) mm_delvt = varvt * geo_fac * pvt_mc
+ mm_mu0 = 1-(.005 * geo_fac * pu0_mc ) mm_dl= 2e-03  * geo_fac * l * plw_mc
…..
```

Figure 1.12 Transistor model card (typical older process, part for mismatch modeling marked bold).

Many outcomes rely on modeling, so often some small extra-margin might be included for this (like make wires wider than needed according to EM and IR drop requirements or let circuit work to 20% higher clock frequency)—this helps a bit to be prepared for the unknown.

As mentioned, also circuits can be modeled, for example, we may create simplified equation-based models and use these for early simulations and planning on system behavior. In this book, we do not focus on modeling, but using a modeling language clearly helps a designer to solve his problems efficiently. Luckily, model reuse is often easier than circuit design reuse! For instance, the same model might be used for an LNA, PA, or just any amplifier. And even if you need a very complex and accurate model, you may still end up in a single LNA model, and it can represent transistor-level models of many kinds and many technologies. One advantage is that optimization with such models is much faster, because less parameters are involved; you can directly optimize on key parameters like gain, NF, and IP3, which is much easier then tweaking the element values to achieve the desired performance. Figure 1.13 show a Verilog-A model of a voltage reference, also here we can define e.g., the noise level directly as parameter, without changing other parameters.

1.3.3 Design, Debugging, and Tools

A designer should have clear opinions on what he wants to achieve and how. Coming to that point requires of course some discussions and experience, but then there are still quite many things that could go wrong. One interesting aspect in tools is that often they are useful for much more than only one specific task—if you know them well.

Designers do experiments, collect data, and decide for further experiments based on the results of the previous experiments. In circuit design, statistics play a role, and also in math, such approach is known, the so-called design of experiments (DOE). DOE covers techniques like parameter sweeps, corner analysis, and Monte Carlo, but of course a big part of design is also intuition and problem anticipation.

Good debugging capabilities (in laboratory and on computer) are very essential, and using iterative refinement and divide and conquer helps a lot because often the error is easiest to identify if you are at the transition from something that works to something that does not work. Actually the word "engineer" comes from the Latin word *ingeniator*, meaning a keen-witted artificer. In the circuit tweaking phase, the designer learns a lot about the circuits, the system, the testbenches, and the technology by doing many parameter sweeps. If you make the sweeps extreme enough, you always have to debug

```
`include "../../veriloga.inc"

// Author: Stephan Weber, Munich
// Version/Origin: New model
// Status : Initial model [/] In work [X] Fully qualified [/]
// Description : Bandgap block
// Limitations : TC are only present in parametric sweep, not in DC sweep
// Testbench Schematic : test_bg


module bandgap(en, out,VCC);
input (* integer inh_conn_prop_name="vcc" ;
         integer inh_conn_def_value= "\\vcc! " ; *) VCC ;

input en; output out; electrical out, en, VCC;

parameter real vref = 1.2 from [0:inf);
parameter real rout = 1k  from (0:inf);
parameter real vnoise = 10e-9  from (0:inf);
parameter real fc = 1k from [0:1G];     // voltage noise flicker corner frequency
parameter real fc2 = 10M from (0:inf);  // voltage noise roll-off frequency
parameter real trise=1u from (0:inf);
parameter real tfall=0.1u from (0:inf);
parameter real tondel=2u  from [0:inf); // tdoff is 0s
parameter real vthres=1.5;
parameter real psrr=80 from [20:inf);   // at DC
parameter real fpsrr=100k from (0:inf); // roll-off frequency for psrr
parameter real tcvout=0 from [-10m:10m];  // TC in V/K
parameter real tc2vout=-0.12u from [-1u:1u];  // Square law in V^2/K
parameter real icc=100u from (0:inf);
parameter real ileak=1n from (0:inf);
parameter integer fullhints=1 from [0:1];

electrical  noise, noisef, psr;

real      Vout, VrefT, tdel,Cnoise,Cpsr,rPSRR, Vnom, i, v, pwr;
integer   enstate;

 analog function real set_vout;
   input state, vactive; integer state; real vactive;
   begin
        if(state>0) set_vout = vactive;
        else set_vout=0;
   end
 endfunction
 analog function real set_delay;
   input state, tondel; integer state; real tondel;
   begin
        if(state>0) set_delay = tondel;
        else set_delay=0;
   end
 endfunction

analog begin

   @(initial_step) begin
         i=icc;
         v=V(VCC);
         pwr=i*v;

      VrefT=vref+($temperature - `TNOM)*tcvout+pow($temperature - `TNOM,2)*tc2vout;
      Vnom=`vcc_min/2+`vcc_max/2;
      enstate=(V(en)> vthres) && (V(VCC)>=`vcc_min);
      Vout=set_vout(enstate,VrefT);
      tdel=set_delay(enstate,tondel);

      // Check voltages at initial step:
      if (V(VCC)<`vcc_min) $strobe("%M: Warning! Supply TOO LOW in BG at initial step");
      if (V(VCC) >`vcc_max) $strobe("%M: Warning! Supply TOO HIGH in BG at initial step");
      // Noise roll-off
      Cnoise=1/(2*`PI*fc2);
    // psrr
    rPSRR=pow(10,-psrr/20);
    Cpsr=rPSRR/(2*`PI*fpsrr);
    if(fullhints) $strobe("%M: Temperature influence vref-Vrefnom %f5.3",VrefT-vref);
 end
```

```
      @ (cross(V(en)- vthres, 1)) begin
      enstate=V(VCC)>=`vcc_min;
      Vout=set_vout(enstate,VrefT);
      tdel=set_delay(enstate,tondel);
      end;
      @ (cross(V(VCC)- `vcc_min, 1)) begin
      enstate=V(en)>=vthres;
      if(enstate) $strobe("%M: BG turned on by Vcc>Vccmin and EN=H.");
      Vout=set_vout(enstate,VrefT);
      tdel=set_delay(enstate,tondel);
      end;

      @ (cross(V(en)- vthres, -1)) begin
      enstate=0;
      Vout=set_vout(enstate,VrefT);
      tdel=set_delay(enstate,tondel);
      end;

      @ (cross(V(VCC)- `vcc_min, -1)) begin
      $strobe("%M: Warning! Supply TOO LOW in BG.");
      enstate=!`reset_on_vccmin;
      Vout=set_vout(enstate,VrefT);
      tdel=set_delay(enstate,tondel);
      end;

      @ (cross(V(VCC)- `vcc_max, 1)) $strobe("%M: Warning! Supply TOO LARGE in BG");

       V(noise) <+ white_noise(vnoise*vnoise, "BG noise")+flicker_noise(vnoise*vnoise*fc,1, "BG 1/f noise");
      // RC lowpass filter for noise roll-off
      `CAPG(noisef,Cnoise);
      `RES(noise,noisef,1);
      // RC highpass filter for psrr
      `CAP(VCC,psr,Cpsr/1M);
      `RESG(psr,1M);
      V(out) <+ I(out)*rout + transition(Vout, tdel, trise, tfall) + V(noisef) + (V(VCC)-Vnom)*rPSRR + V(psr);
      I(VCC) <+ V(VCC)*`gmin + (icc-ileak)*transition(Vout/VrefT, tdel, trise, tfall) + ileak;
      $pwr(I(VCC)*V(VCC));

end

endmodule
```

Figure 1.13 Verilog-A model for a bandgap reference cell.

something! Of course, sometimes, you also have to debug not only circuits, e.g., you may need to check whether this is a model problem or a simulator accuracy problem. For this, inspect log files and tighten the simulator accuracy.

Modeling is also perfect for debugging, e.g., the "assumption" that the gain is lower due to package inductance by 1 dB is often meaningful, but of course it is much better to include the package to your testbench. This way your setup reflects the idea directly (even if you forget the assumption) and even much more accurately!

Mistakes can be costly, so to be able to make decisions for difficult problems, you need high trust. A good technique is "always double-check." Do not rely too much on thinking or "obvious" things: Imagine there is a design problem, and you measure ten samples in laboratory. Maybe the variations are not large, but to conclude that the samples behave like in a nominal simulation is risky. If your samples are from one production lot only, you may have significant process deviations on top of the usual mismatch. So it still can make sense to run, e.g., a short Monte Carlo process and mismatch analysis

to clarify the performance problem; first get an overview before making conclusions!

Also tools like simulators double-check things internally, e.g., by applying multiple convergence criteria, and often you can double-check further by making a "golden run" using very tight accuracy settings (like $reltol = 1\ e$-9). Sometimes this is difficult (e.g., due to convergence problems, maybe caused by floating nets, and high-Q elements) or time-consuming, so your deep expertise is required, like treating not only *reltol*, but also tweak more advanced options (like *maxstep*, *minstep*, the integration method or whatever).

In the advanced techniques described in the book, it is absolutely the same, e.g., just run MC twice with different settings or inspect the reported confidence intervals and inspect the log files in detail.

In statistics and optimization, there are luckily only a few icy places where you need to look up carefully, probably confidence intervals are one (Chapter 3.5) and we will tell you! A good method is usually doing an analysis in a different way, e.g., checking transient results against what you expect from AC behavior or double-check yield calculated from sample yield and process capability index C_{PK} (Section 3.6.2).

Often you have to decide which to trust more—and that depends on many things—e.g., phase margin PM gives you a number to quantify stability, but a single number cannot fully represent all kind of instabilities in a nonlinear system, so double-check with transient analysis, S-parameters, manual calculations, waveform inspections, etc.—exploit what you have; and try to get what is missing.

The good thing is that tool problems are often related to circuit problems! So most designers apply such techniques anyway to some degree and extend it hopefully. You should never really stop: Some outputs of analysis are for sure almost trivial and check for what you directly want to verify, like that a unity-gain buffer really reproduces the input signal—easy to check in a transient analysis. The more experience you have, the more you can do: Check also overshoot and distortions, and look maybe to the differential input voltage to check whether the loop gain is high enough forcing a low difference. Check the recovery behavior: Is your circuit coming back quickly to correct operation in case of overdrive? It is hard to be aware upfront of everything, e.g., the filter cutoff frequency sensitivity to RC elements should be one to one (like 10% in R gives a shift of 10% in frequency), but in high-Q filters, this will change even depending on topology. Also the sensitivity to other parameters is not always easy to predict, e.g., because op-amp loop gain might not be really large anymore at the frequency of interest.

Maybe some problems remain, and you have to discuss them with an expert and have to train yourself further. Do not *only* learn from your own mistakes.

Almost all kinds of tools have a direct obvious output, like a histogram from a Monte Carlo analysis or the performance variations in a corner analysis or parameter sweep. However, there is often much more, unfortunately sometimes "hidden" in log files or menus. If a design is bad, you may want an optimization, but often other methods are more efficient: Many advanced simulators feature an analysis to check the impact of mismatch (or transistor parameters) to DC operating point. The result of such analysis could be a ranking list of the instances causing the biggest performance changes and the total variation, e.g., in the output voltage of a reference generator. A designer can do a lot with that information: If, for example, the top 4 transistors dominate the mismatch and we make their area $4\times$ bigger and we can expect an improvement of the overall mismatch by almost $2\times$! So we can improve directly without using an optimizer!

Also automated optimizers and high-yield estimation (HYE) methods provide much more benefits than just improving the circuit or verifying the yield—in addition, you get valuable design information for your understanding and for more efficient work. We will tell you because this way advanced designers have often even much *more* benefits from advanced tools than less experienced ones. For instance, sensitivity analysis results are available as a "by-product" of more advanced analysis like worst-case corner search or just a Monte Carlo analysis. In opposite to simulator built-in analysis, those have often the advantage of higher flexibility, like being not limited to DC or AC behavior, but valid for any kind of output (like noise figure (NF), total harmonic distortion (THD), or third-order intercept point (IP3)).

To some degree, statistical analyses are often not done because statistics is so interesting, but also because it is one important piece for enabling sensitivity-driven design. But watch out, and this is not for free, e.g., it is quite easy to calculate the sensitivity of a certain performance metric with respect to a certain transistor width (like W_1), but this does not mean that you as a designer can do really much with it, because in a low-offset differential pair, nobody would usually change the width W of only one of the two transistors forming the pair! What you really need is the sensitivity to W_1 being in synch with W_2 and that is no netlist information available to the simulator. Also many simulators can provide sensitivities to many transistor parameters, but you as a designer cannot really change the technology or just one individual transistor

parameter like mobility or current gain β without compromising others (there is, e.g., a trade-off between β and early voltage V_{EA}). As a designer, you have to formulate your questions in testbenches—and this is always some significant work. In addition, there is also much design work *beyond* pure sensitivity in general, because the sensitivity cannot catch circuit modifications like adding a buffer chain or a cascade or a capacitor for more frequency compensation flexibility.

In conclusion? In this book, we give you guidance on many methods, and sometimes the very basic methods—like simply simulating really all variable combinations or doing a huge Monte Carlo analysis—are extremely inefficient, so maybe we condemn them too often and we stress the disadvantages too much in the readers mind? We are fully aware that sometimes only almost-brute-force methods are completely foolproof, and indeed, you can always construct test cases, where "too" clever methods would fail! Running all combinations allows to find the worst-case safely, but an automated search can be much faster. On the other hand, having really the results from all combinations would also offer to find the best one, which is not of much help for verification, but is indeed helpful for starting laboratory investigations or for keeping your design alive and improving it further.

Murphy's Law versus RTFM? Besides doing the setup and decision making, designers are also challenged by tool bugs and limitations, sometimes. For a transistor-level simulator, hundreds of options may solve problems, or cause them. Luckily, most statistical or optimization techniques feature much less options, e.g., for Monte-Carlo, you may need to decide what do you want to save, which random seed you want, and how many run points, but not much more! Also more advanced methods do not need a big setup luckily, and mostly, this is because—even and especially the most advanced—algorithms came with a lot of internal automatically adjusted options. With the options available directly in the user interface, you typically set a certain compromise between speed and accuracy, e.g., by setting stopping criteria.

However, of course something could go wrong like an optimization gives no progress or a high-yield estimation algorithm is not able to provide an accurate solution. In these cases, read the log files carefully, try to follow the hints, and read the fantastic manual. Actually, 20 years ago, software documentation was sometimes horrible, but nowadays the products are quite mature and well-documented, featuring many examples, screenshots,

and even demo databases or videos. Ask the service team, and often you are not the first one having this problem. If something gets really wrong, double-check not only for the direct tool output message window, but look also to the general log window, to messages in the Unix shell, etc. Best go back to an easier testbench till you get something that works. From that, you can extend the setup again, step by step, till you narrow down the problem and locate it—like problem happens with a certain version or is only present in a certain device or type of distribution, etc. Of course, there is a general problem in documentation: A manual is typically focused on explaining all the different features, so it is often not really solution-oriented! However, often there is further material like app notes, videos and white papers giving more background explanations and examples.

Also the simple methods may come with further options, e.g., the overhead in running all combinations can be used to derive internal error limits, which might be not available to that level in highly advanced methods which would really only do the absolute minimum number of necessary simulations.

There is no free lunch! "Greedy" methods can fail, often in a quite spectacular way! We will get some examples later. On the other hand, the design challenges are often so large that indeed, brute-force methods would be far too inefficient (like we would need more than one million simulations to run) and too simple basic techniques (like doing only one-dimensional sweeps and ignoring all correlations) would become highly inaccurate. Then, mixed approaches and iterative techniques become attractive, but still it is good to know what their benefits are and how they mitigate the remaining risks.

Trust and Error Limit. Tools often report error limits, this gives trust. And actually a pure point estimate is not enough, you should really have also an error estimate. "Error Estimate", seldom you can get more; and such error estimates can have different quality! You would be in a perfect situation, if someone gives you a true guarantee, like $900\,\Omega < \mathrm{R} < 1\,\mathrm{k}\Omega$. If you buy an SMD component, you get almost such hard limits. Unfortunately, in statistics you typically have only statistical "limits", like "The chance that R is larger than 1kΩ is below 0.1%". These kind of limits are better than nothing, but not as good as hard limits. In addition, in many cases estimations depend on model assumptions, but it is hard to say if a

a model is really valid or not. There is a gray area! So not only the estimate (e.g., on yield) but also the method to estimate its tolerance might be not 100% correct. Try to understand how the error estimation works in the specific algorithm. Using Taylor series is one general method, but often you can hardly know how many terms you need to include for error calculations; and usually error estimation works only with low risk for interpolations, not for extrapolations.

Check if model assumptions are meaningful, do not misuse special methods, e.g., check by eye inspection if the data is Gaussian if you use confidence intervals on the mean based on the Student-t distribution. Of course, multiple errors can be present, and they may add up significantly. Here double-checking is best. If someone is promising that a certain method is "trustable" and "verifiable", he often promises too much (at least in a mathematical or legal sense), or he forgets to mention the prerequisites.

1.3.4 Simulation Aspects

Of course you need to be able to simulate your circuits, usually on transistor level to design your circuits with computer support. This is standard since 1980s. For complex analysis, the runtime could unfortunately cause problems; usually more in design automation and for advanced analysis, then e.g., for pure interactive manual design and debugging plus waveform inspection. Therefore, e.g., advanced statistical methods need to be efficient regarding the number of simulations to get a certain output, like sensitivity or the standard deviation of your circuit performances.

These aspects are quite obvious, like a 10-corner simulation may take 10× more times than a 100-corner simulation. The good thing is that also most advanced methods have still quite a moderate internal runtime, but one aspect is often overlooked—accuracy! For instance, it can be already challenging to get accurate enough transient results, e.g., for a DFT output with low-noise floor or to get the overshoot really accurately. For instance, reading out the maximum output voltage $\max(V_{\mathrm{out}})$ can be impact by tiny spikes or small shifts in the simulation steps the simulator takes. Usually, designers can manage such problems for their verifications, with careful testbench setup, but for some advanced analysis, you need indeed truly a higher accuracy. This is mainly the case for comparisons, like for gradient calculations by finite differences for an optimization. Having a too large numerical noise could prevent to find the best circuit solution, could prevent getting accurate sensitivity results.

Later we will see that optimization is also one step for finding worst-case distances (WCD), so also some advanced and very useful statistical techniques need really a good testbench setup.

Not only transient analyses are critical regarding accuracy, but also an AC analysis can create numerical noise (even if the related DC solution is already very accurate), e.g., by using too few frequency points and when looking to characteristics like bandwidth, peak frequency, deepness of a notch, or filter passband ripple! Therefore, always inspect your results manually with a waveform viewer and read out the performances manually; also double-check the accuracy setting (like tighten the error limits and double the number of points and check how much the results change).

All in all, quite a big part of engineers' work is spent on making good testbenches, universal testbenches, and tests for debugging, up to a full optimization setup which really captures all performances correctly and reliably.

1.3.5 Total Yield and Partial Yield

The sample yield is easy to calculate as the number of good samples n_{pass} divided by the total sample count n. You can calculate it not only for each specification, but also for all specifications together. In both cases, yield is only well defined if you have enough pass and fail samples to guarantee a "stable" statistic!

Of course, changing one design parameter like resistor R_1 may improve the regarding performance A but may make performance B worse; not only performance matters, and with respect to costs, the production yield has similar importance as performance itself.

Note: The real production yield is also impacted by layout defects like broken vias. Here, in the book we focus on what the front-end designer can do to improve the yield. The term "design for yield" or "design for manufacturing" (DFM) can be used in different ways. In layout, yield improvements are possible too, e.g., by avoiding single vias, following more rigid design rules, etc. Also note that in statistics and in production, the term "sample" often has slightly different meanings: In production, a sample is usually a single piece, but in statistics, also a certain set of samples (or several MC points) are regarded as sample. Note, because also such sets of random samples are random samples, not fully representing the whole statistic.

To simulate the production yield of our design (defined by $\boldsymbol{x}_{\mathrm{D}}$) in a computer, we can mimic the fabrication and its production tolerances (defined by $\boldsymbol{x}_{\mathrm{S}}$) with a Monte Carlo simulation. For instance, we can generate a set of $\mathrm{n} = 1{,}000$ designs, each having different statistical parameters. To be in spec, we need to run many simulations on each design to cover all *combinations* of operating (range) parameters $\boldsymbol{x}_{\mathrm{R}}$ like temperature, load resistance, and supply. If we want to verify at least three values for each of the r range parameters, we need to do three-corner simulations. For realistic designs, this may lead quickly to >250-corner simulations, to be executed on *each* MC sample (coming from our virtual production), so overall to >250,000 simulations. This is a simple but very time-consuming way to check the design. If you want to improve your design on yield with given performance specs you even need to tweak your design and the step with >250,000 simulations is required for *each* individual design, which ends-up in a very slow over-all progress!

On the other hand, truly only this extremely exhaustive flow has no systematic errors. In general, the Monte-Carlo simulation effort for design is given by:

$$\begin{aligned}\#\text{simulation} = \#\text{design combinations to inspect} \cdot \#\text{tests/simulations} \\ \text{per test} \cdot \#\text{corners}^{\text{sweep-points for each corner}} \cdot \ \text{MC points}\end{aligned} \tag{1.2}$$

Note: If you as a designer make a very clever testbench setup, you might be able to treat multiple corners already in one simulation. This is often done for important parameters, like doing a DC sweep on temperature or supply voltage! However, "too clever" testbenches are often harder to manage, to extend or less handy for debugging.

Mathematically (see Chapter 3), the overall yield is defined as volume integral over the product of the indicator function and the joint pdf. The indicator function gives a 1 in the pass (or acceptability) region (the region where all performances are in-spec) and 0 in the fail regions.

Even if we exclude the condition parameters, it is typically a very difficult and highly nonlinear function of a huge number of statistical variables; the more performances we have to check, the more difficult the spec-to-failure boarder (and the yield integral) will look like. Later we will give you some pictures and equations.

Do you hate buzz words? Like "new," "breakthrough,," "brute-force MC"? You are right, e.g., why taking a stupid brute-force method as "reference"? In this case, however, there are indeed good reasons to do so. One is that using fancy adaptive or empirical methods as reference would lead to very fuzzy comparisons. Also there is often no better standard way, and new adaptive methods are often simply not available to all authors! In addition, e.g., a full-factorial analysis is a brute-force "stupid" method, but it leads to very well-defined measures: If the number of variables is given and the individual values, you can directly calculate the total number of all combinations and your simulation effort. Also for MC, something like this is possible. On top, you can also often quantify the remaining inaccuracies of such methods. Often the user has to decide for the setup of two different more advanced statistical methods, which might be hard to understand and unfortunately not really well documented. In this case, go one step back and inspect MC as reference; then relate both advanced algorithm against MC for the manifold aspects. Whenever possible, we try to give also references to manual best practices for design, but there is unfortunately no "gold standard" for more advanced methods including those based on design experience, intuition, and common sense!

When checking production samples against the spec limits, we can calculate the yield; each performance leads to a certain *partial* yield. Only if a sample is in-spec for all performances, we can ship that sample to the customer as a good sample. The total or overall yield is lower or equal to the lowest partial yield. It is well known that the partial yield for spec1 and spec2 can be 50%, but the overall yield might be 0 to 50%—depending on correlations. Typically, we have both "fighting" specs (often bandwidth or rise time versus phase margin—see Figure 1.14), where we have almost to add the yield losses and almost redundant specs (like bandwidth and rise time), where we can almost just use the minimum partial yield. So the "compromise" of assuming no correlation, giving 25% is often not so unrealistic, luckily.

Of course, for too difficult and too many competing specs, the design becomes completely infeasible! Luckily, for high yields (and you typically aim for this), the total yield uncertainty from correlation relaxes a lot: e.g., $Y_1 = Y_2 = 99.8\%$ can lead to $Y_{\mathrm{tot}} = 99.6\% \ldots 99.8\%$ which is often an acceptable accuracy. In such cases, the non-correlated case (99.6004%) is anyway very close to the worst-case. In Chapter 5, we will address the difficulty of performance correlations in more detail.

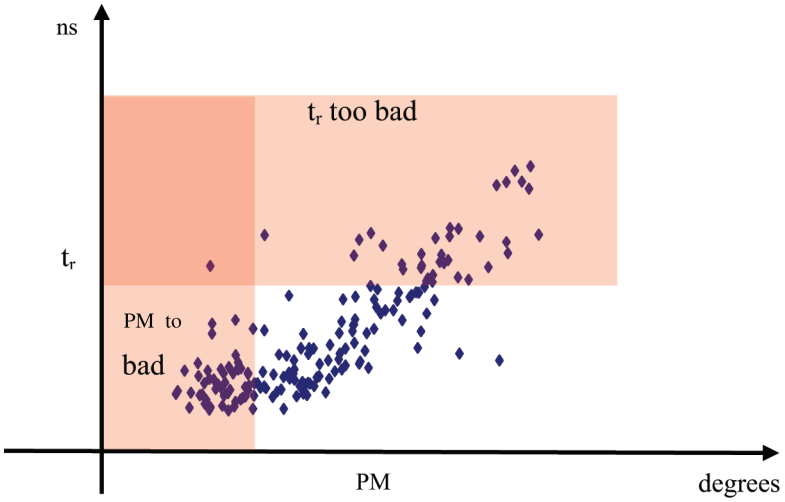


Figure 1.14 Yield for two fighting performances (phase margin and rise time).

Big chips should be still fabricated with a high yield like 90%, but to guarantee this, *each block* needs to have a much higher yield like 99.9%. If we have replicated blocks in our design, like digital standard cells, memory cells, or subcells of a high-resolution DAC or ADC, we need even much higher block yields, which often cannot be verified efficiently with standard MC methods (Figure 1.14).

As you can see, dealing with yield numbers can be a bit difficult, especially if we want to address yield yields, like 99.999%. For this reason, it is very common to express the yield in terms of sigma for a yield-equivalent normal Gaussian distribution.

One problem is unfortunately that sometimes we have single-sided spec and sometimes double-sided ones, and for a single spec placed at 3sigma, the equivalent yield would be app. 99.85%, but for a double-sided spec ± 3 sigma, we would have two times the loss, so $Y = 99.7\%$. The latter number is used a bit more frequently, but most real specs are single-sided, e.g., for the yield or for PSRR, you are only interested in avoiding production samples with a too low yield or PSRR! Instead of using "sigma", you can also use the C_{PK}, and we will discuss it in detail in Chapter 3. The C_{PK} is only valid for normal Gaussian data, and in Chapter 4, we extend the idea and explain the generalized C_{PK}.

Figure 1.15 shows the pdf of a normal distribution with readouts for yield. If the sigma of a design is fix, then one good way to improve on yield is to

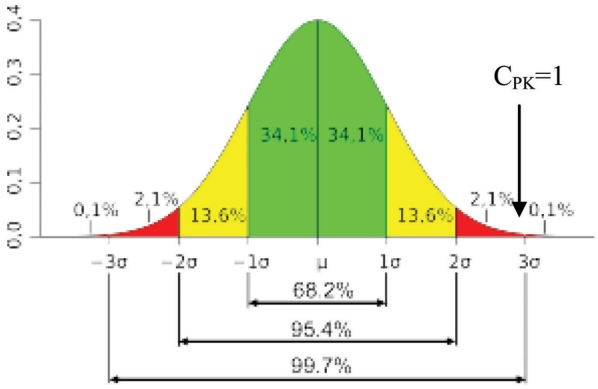


Figure 1.15 Yield parts for a normal Gaussian distribution.

Table 1.3 Yield in terms of sigma and C_{PK}

Spec Setting* or Yield in Sigma	C_{PK}	Rule of Thumb	Single-Sided Yield Loss	Double-Sided Yield Loss
0 sigma	0	50% fails	50%	100%
1 sigma	0.33	about 1 failure in 6	15.9%	31.8%
2 sigma	0.67	about 1 failure in 50	2.3%	4.6%
3 sigma	1	about 1 in 700	0.14%	0.27%
4 sigma	1.33	1 in 30 thousand	0.003%	0.006%
5 sigma	1.67	1/3 in a million	290 ppb	590 ppb
6 sigma	2	1 in a billion	1 ppb	2 ppb

*Distance of spec to mean for using a normal Gaussian distribution.

"center" the design. This way you can minimize the total loss according to upper and lower spec limits; better have a balance than too much loss on one of both spec limits. Another way is of course to try to make the design more robust and to reduce the sigma, so yield optimization is more than only design centering (Table 1.3).

How many sigmas do you need? Please start to like "sigma", it allows dealing with less extreme numbers, and it provides you a better feeling for statistics. If your plan a high-volume production, a good chip-level yield makes life (e.g., testing) much easier (and cheaper). So maybe Y = 95% is a realistic target, maybe even 99%. However, if your design contains 1,000,000 memory cells or more, then we need for each cell a real high yield, easily six sigma. For blocks which are placed only once on the

chip the situation is more relaxed, but still a chip can contain easily a hundred blocks, and already one block fail can lead to a bad chip sample; so each block should still have a yield in the order of 4 to 5 sigma. So to some degree we have applications where high-yield verification is a must, and also cases where low-yield methods like Monte-Carlo fit well. However, as shown, also the intermediate region is important, and already for 4 sigma Monte-Carlo might be impractical (read the chapter on confidence intervals and yield verification), at least if your circuit requires time-consuming simulations.

A further important questions is also how accurate your MC estimates (for yield, mean, standard deviation, etc.) should be. Usually it does not matter so much if the standard deviation of your offset voltage is 5 mV or 6 mV, so sometimes 20% error in terms of sigma might be still acceptable. However, for an ADC or DAC too large mismatch can quickly cause severe errors like missing codes or non-monotonic behavior. In pure Monte-Carlo analysis all estimates have a certain tolerance; and tighter tolerances require more simulation effort. Find more details in Chapter 3.

1.3.6 Robust Designs

You are typically happy if your design is in specification over the full operating region. But how to achieve it? By far the best way is to make the design robust by construction and not to rely on pure simulation and verification techniques!

In analog circuit, we represent signals directly by physical natures (I, V, C, etc.), so they are much more sensitive to manufacturing process and environmental parameter than digital circuits. Design robustness requires the systematic elimination (or at least minimization) of sensitivities to all those parameters. This is only possible by careful choice of the circuit and system architecture, circuit topology, and very careful implementation. This is time-consuming and requires accurate device modeling, and good understanding for the circuit operation and the technology behind. Many problems need to be anticipated, so that a timely project execution and verification are feasible.

A big trend in making analog circuits robust is using clever mixed-signal techniques, e.g., $\Sigma\Delta$ ADC and PLLs. Those were only a first step and a lot of innovations can be further expected, because pure analog techniques tend to become more difficult or just too expensive compared to clever mixed techniques.

In analog circuits, some old tricks may not work so well anymore, e.g., due to reduced supply voltages, or you may need to use a technology optimized for digital, giving less flexibility as an analog-oriented BiCMOS process. Often innovations are coming from both: new restrictions and new opportunities. In Chapter 2, we will give several examples. More complex examples and an excellent overview on ADC design (but not only this) can be found in [Murmann]. Performance gains in circuits are not only coming from CMOS scaling (triggered by the down-sizing of transistor dimensions in new technologies), but also coming from great innovations and surprising concepts. Sometimes the improvement is *not* in making a *better* op-amp but just using *no* op-amp anymore (like replacing them by oscillators or comparators or charge multipliers).

Some of the general techniques for yield improvement are visualized in Figure 1.16; (a) shows a non-optimized design which is "in spec" at nominal conditions, but it fails on performance f_2 at the worst-case corner. In (b), we accepted the variations, but we improved overall performance, and this might be difficult but often possible by spending more area or current (assuming a big spec margin here). In (c), we reduced the variations in performance f_2, but it

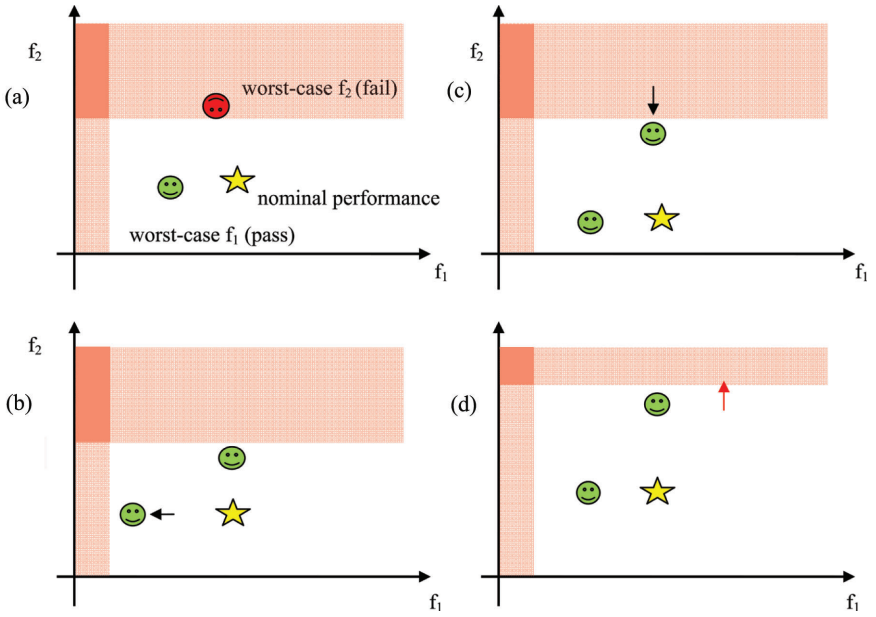


Figure 1.16 Different yield improvement strategies for two performances f_1 and f_2.

often comes with sacrificing other performances or the performances spread in other performances! What is also often possible is to ask for a spec relaxation (Figure 1.16(d)). The ideal case of reducing the *variations* in general (so that it is not needed to make the nominal performance extremely good) is usually also quite difficult to realize; e.g., you may need more chip area to reduce mismatch variations. Sometimes there is no other solution then just spending more area or current, often this is the case for critical parts, and e.g., we may need to compensate the area increase by using smaller transistors in less critical parts. Later we will give further examples, and one solution is of course just to try another circuit variant.

Note: Worst-case (WC) refers not only to environmental conditions, also to statistical variations. A nice circuit example is a Butterworth filter, having a *maximum flat* passband gain. If we design a Butterworth g_mC filter at nominal conditions and process corner (NN) it can happen, that e.g., far too many Monte-Carlo samples have a large undesired filter ripple. So if we really need a flat response for almost all samples and conditions, we actually should design our filter this way that also these extreme MC samples—also being a kind of worst-case corner – are in spec. And this is usually only possible by limiting the mismatch impacts and by reducing the filter Q factors. So at the WC we would get a Butterworth behavior (quite high Q factors) and at nominal we get a filter closer to a Bessel filter (quite low Q factors). Having an eye on nominal performance for understanding *and* on WC for being in spec is the perfect method for achieving robust designs efficiently. Doing this we could see *early* enough that our filter is almost impossible to design, and we may need indeed another circuit, e.g., and to increase the filter order.

1.4 Design Flow Inputs and Outputs

Some elements in the custom IC design flow we already mentioned, beside schematic, specifications, testbenches, layouts, etc., there are also many other documents important for you and your customers, like a guarantee for a certain life time or a limited number of bad devices in the delivery.

Especially for reuse purposes, a (much) more detailed design-oriented datasheet—more a real design documentation—is usually desirable. In addition, make a presentation to your colleagues, and describe well the circuit and its tricks (Table 1.4).

Table 1.4 Custom IC elements

What	Requirements	Created by	Comment
Datasheet	Product idea, electrical and mechanical specifications	Designer, marketing, customer	
Design documentation	Datasheet plus additional information (e.g., on tricky parts, on sensitivities)	Designer	
Process developments kit PDK	Featuring component libraries, simulation models, layout cells, run decks to check design rules, etc.	Foundry	Is technology-specific, and number of rules and complexity increases more and more
System topology	Datasheet	System designer	e.g., checked with Excel® and MATLAB®
Floorplan	Datasheet, chip size estimate, pin positions, block size estimations	Lead designer, lead layouter	
Schematics	Inputs and outputs, circuit function	Designers, using a schematic entry	For circuits and testbenches
Netlists	Schematic	Automatic, triggered by designer	Usually in SPICE format or a similar one
Postlayout netlists	Layout, LVS results	Automatic, triggered by layouter	Tools offer also table outputs, backannotation of parasitics into schematic, etc.
Layouts	Schematic, layout hints, e.g., in OA format	Layouter or designers, using a layout editor	Hints can be provided verbally, as comments or as constraints
Bond plan	Package and die drawing	Lead designer	To be send to fab
LVS report	Schematic and layout, LVS run decks to extract devices	Layouter	To make sure that what you layouted is fit to schematic
DRC report	DRC run deck, layouts	Layouter	To check that design can be manufactured
GDS	Layout	Layouter	Defining the coordinates for all elements to be created at each layer

(*Continued*)

Table 1.4 (Continued)

What	Requirements	Created by	Comment
Evaluation board	Laboratory test hardware (and software)	Designers and application engineer	
Test circuit/program	Production test hardware and software	Designers and test engineers	Test is usually related to production tests, but verification is usually referred as part of design
Quality plan/report	Checklists, etc.	Designers and quality engineers, etc.	This is clearly a team effort. Often it is required to follow certain norms like ISO 9000 on quality management
Third-party IP	Usually, you get only a minimum on documentation on files, like GDS, SPICE netlist, and datasheet	IP vendor, foundry, etc.	Often used for digital standard cells, memories, IO cells, etc., but can be also a major part like ADC, PLL, DDR3 interface, etc.

Figure 1.17 is giving a picture for different design and analysis methods according to the different variable types. For circuit simulations, all three types matter, whereas some other techniques like DRC and LVS run are usually only done for a fix design defined by $\boldsymbol{x}_{\mathrm{D}}$. Note, that the complete space $\boldsymbol{x} = (\boldsymbol{x}_{\mathrm{R}}, \boldsymbol{x}_{\mathrm{S}}, \boldsymbol{x}_{\mathrm{D}})^{\mathrm{T}}$ can be huge and the performance functions $\boldsymbol{f}$ depend on all three types, so doing a special analysis, like a corner run is capturing only a little subspace, which might be not fully representative. So mathematically, doing only these basic analysis is working without really having the eyes fully open. Actually doing only isolated analyses in *one* kind of variable, would be mathematically only acceptable if the circuit would be have according to Equation (1.3).

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}_{\mathrm{R}}(\boldsymbol{x}_{\mathrm{R}}) + \boldsymbol{f}_{\mathrm{S}}(\boldsymbol{x}_{\mathrm{S}}) + \boldsymbol{f}_{\mathrm{D}}(\boldsymbol{x}_{\mathrm{D}}) \tag{1.3}$$

In this case e.g., the sensitivities $\delta\boldsymbol{f}/\delta\boldsymbol{x}_{\mathrm{D}}$ would not depend on $\boldsymbol{x}_{\mathrm{S}}$ and $\boldsymbol{x}_{\mathrm{R}}$, but this is clearly unrealistic.

Let us see in Chapter 2 how open the eyes are in a typical manual IC design flow.

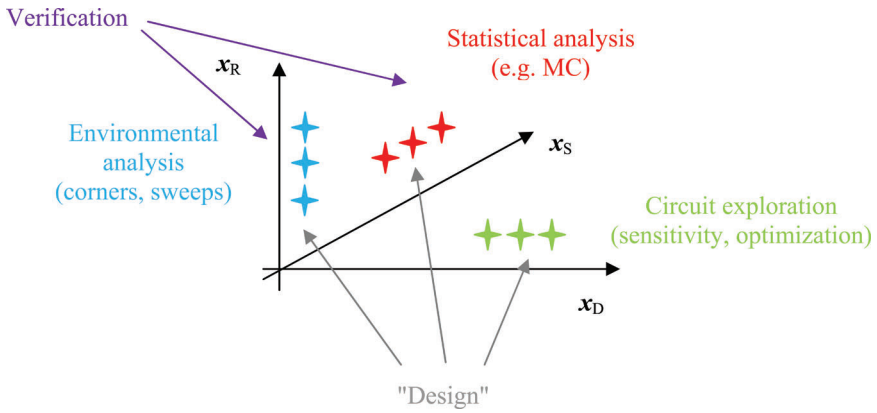


Figure 1.17 Flow input and outputs.

What is in a PDK? To make real designs to be manufactured, you need library elements, These represent the chosen technology, and coming usually from your foundry. So in modern complex technologies, a process development kit contains quite a bunch of material. From the technology library, like cmos90rf, you can pick cells (actually the symbol e.g., for a certain NMOS transistor, a certain resistor type, etc.) and create your circuit blocks in a schematic entry. To run simulations, the process development kit also includes simulator models, like Gummel-Poon models for bipolar transistors.

Also layout views are part of the PDK, and such layout cells are usually *parametrizable*, because in opposite to a SMD transistor, chip designers can e.g., choose the width and length of their elements in a quite large range to optimize circuit characteristics. Such layout cells are called programmable cells (pcells), and each contains the geometric construction statements for the different chip layers to form a specific component (like a high-voltage transistor).

Also available are e.g., rule decks. Using them we can make sure that the design becomes really manufacturable, e.g., all designed element need to be separated by at least a certain minimum distance, to avoid e.g., problems with short circuits, leakage, etc.

The tools picking up the PDK content are typically coming from an EDA vendor. Some required tools are even available for free, like the original old SPICE simulator. However, usually commercial implementation offer more features (like special analysis types) or higher performance

(like faster matrix solution, parallel processing, etc.), plus service (not only around the tools itself, but e.g., also regarding design IP, methodology, hosting, etc.).

1.5 Questions and Answers

1. How complete should a datasheet be?
   In the style of official datasheets, there are huge variations, some provide only the absolute minimum, and some are readable like a book on learning circuit design! For good examples, look to the old datasheets of OP-07, the famous PMI low-noise high-precision operational amplifier or to newer products of leading manufacturers. Often the devil is in the details, e.g., some performance specs require an accurate testbench description. For instance, the distortion might be small as inverting amplifier, but much larger in unity-gain configuration due to common-mode distortion. In addition, load and frequency will have a significant impact.
2. Assume the error in your yield calculation is 0.3σ, and what is the error in yield loss?
   The relationship is highly nonlinear, e.g., 2.3 × loss error at 3σ, 6 × at 6σ.

3. Could it happen that in a full MC analysis the sigma of a reference voltage is 2× smaller than from a production?
   This can happen as it also can happen that two results of an MC analysis are not identical! For instance check whether at least the MC simulation confidence interval hit what you get in production. In addition, the production in one fab and few lots (see Figure 1.18) might not show all the allowed tolerances, which you may see over the whole product lifetime or at other fabs using the same process technology! Usually, the limits a fab has to guarantee also come with some margin, and bad wafers will be thrown away, so often process parameter distributions look Gaussian, but with cuts or like multiple narrow Gaussian distributions shifted against each other. Of course, you should design for high long-term yield!
4. Look at the Texas Instruments op-amp datasheet, and how many specs are included? Is this typical? Compare to Figure 1.6!

There are more than twenty specs, which are quite a lot for a simple block with seven pins. Few important characteristics like saturation voltages, and recovery times are not included.

5. Discuss when a design is "good"!
   This is an important question, because only when we can formulate this, we could think of a true design automation.

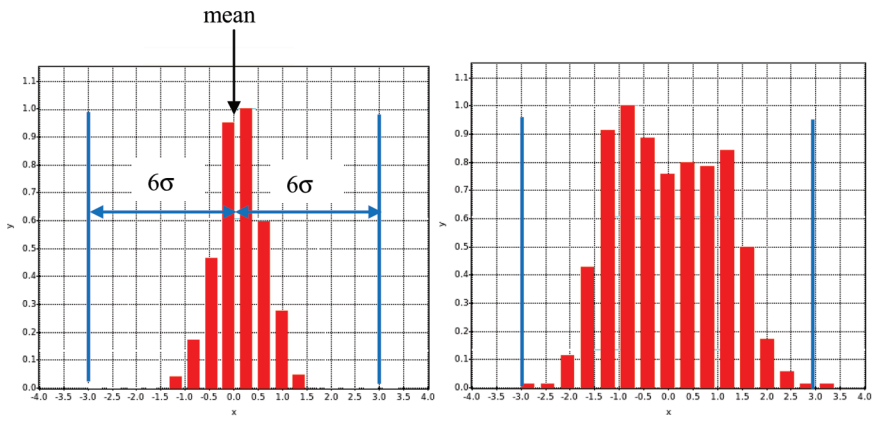


Figure 1.18 Typical short-term and long-term distributions in a fab [Pieper2008].