Contents

PART IV: Optimization and Advanced Flow Techniques

Foreword

Fueled by the continuous downscaling of transistors in the CMOS VLSI technology, integrated electronic circuits are the cornerstone of most applications and devices we use in our daily lives today. From the radio and TV to our smartphone, from our car to the dishwasher, from our laptop to the heart rate monitor we use while jogging, just to name a few. Electronics add intelligence and controllability to objects, and wireless connectivity adds on top mobility and universal connectedness across the globe. But while digital integrated systems are mostly designed in a highly automated manner at higher abstraction levels starting from some form of language-based description, analog and mixed-signal electronic circuits are still today mostly designed at block and circuit level and with little or no automation, be it of course not without individual CAD tools.

Such electronic circuit design is by far not an easy task, not easy to carry out and not easy to learn. The main reasons are the many complex and often conflicting relationships between design variables and circuit performances, the underdetermined nature of the design problem at hand, and the impact of internal and external variations. The many degrees of freedom in the design and the sheer high-dimensional complexity challenge our human brain. As a result, designing a circuit that is optimal in some desired sense and that is guaranteed to meet the targeted requirements and specifications under all fabrication and operational circumstances is not at all trivial. But although it may look as an art to some, it's a skill that actually can be mastered by many by combining experience with analytic insight and systematic design methodology.

While various widely used analog "circuit design" textbooks essentially restrain themselves to presenting and analyzing circuit schematics, this book goes way beyond that and actually presents design from a pragmatic design viewpoint: it describes a systematic variation-aware approach to interactively design fully functional circuits with the help of advanced CAD tools. While EDA tool research is progressing continuously in academia and industry, also for analog and mixed-signal circuits, fully automatic synthesis of analog

circuits in general is probably not to be expected soon as commercial offering. Yet, powerful CAD tools for simulation/analysis as well as for optimization have been developed over the last decades. In combination with a systematic design strategy and the power of today's computers, these can be used to successfully crack the nut of designing a functional electronic circuit.

This book will show you how, and does that in a practical way with several design examples. Large focus is on dealing with the impact of variations due to the fabrication process as well as through the environment. Using advanced statistics and going beyond classical corner and Gaussian analysis, the impact of these variations is analyzed and circuits are optimized for robustness and maximum yield. Though the CAD techniques used are state-of-the-art commercial offerings, the authors are no blind tool believers, and clearly pinpoint the limitations of the tools used. As always, the computer only gives you what you ask for.

Considering its unique focus on a systematic variation-aware and tool-supported approach for designing electronic circuits, this book is recommended reading for practicing design engineers, electronic design engineering students as well as EDA developers. As such, the book illustrates in a pragmatic way that circuit design in today's world is so much more than magic, it is magic that everyone can master!

Prof. Georges G. E. Gielen
University of Leuven, Belgium

Preface

Writing a book, you probably start with an outline, with collecting ideas, with interesting chapters, etc. and if you write an introduction, you often end up too many things, in something too *long*. So here is the two page, shortest possible, introduction and preface. In the subsiding (long) chapter just read further about "why" and "how", about motivation, background information, challenges, trends, etc. Then we really move over to real design, design techniques, to their problems and to new techniques!

People are fascinated by beauty, which has often aspects of maximum simplicity, and endless complexity: nature, arts, sports, technics, philosophy, electronic circuits, and math! The authors really like circuits, but this book is not so much about circuits itself; we focus on *techniques*, on connecting these parts well: circuits and *design techniques* and math; and we will find beautiful and ugly things. To some degree designers also love ugly things and need to find workarounds, and sometimes, but not always, you again end up in something elegant, something yours.

What is the status? Electronic devices are very complex and tricky, but very cheap. This is because essentially they are just *printed*, like this book! The manufacturing of chips is complex, but amazingly efficient and cheap, per device or function even extremely cheap. The biggest invention in design itself is the use of computers and software for *simulations*. So in modern designs people work – usually together in a team – on *virtual* prototypes. And at some point they need to become confident, that the product would also work in reality. This is a challenging task, and many variables have an impact whether a design (component, chip, board, system) fails or succeeds.

Circuit simulation is mathematically something quite special, solving a set of nonlinear equations, like $\boldsymbol{f}(x) = 0$. The real simulation breakthrough was in the 1980s, so what is *beyond* pure simulation?

Dealing with variables and function helps to translate a problem into math and algorithms, but unfortunately we have to deal with many variables, and with functions we simply do not know so well. Usually special *combinations* of variables are critical (like low supply voltage, high temperature, heavy

load capacitance, etc.); and having many variables, means having even more combinations of them, so that at some point also simulation time matters or the full simulation is becoming even unaffordable (like taking years). Imagine you have a budget to run realistically n = 100,000 simulations, "which" simulations you should execute to achieve a certain goal, like "Find the most critical parameter combinations regarding all performances".

On top of such verification tasks, designers want to find the "best" circuit topology and the according component values to make sure that the design works even under these difficult conditions. This is minimization of errors or optimization; and mathematically it is basically the *minimization* of functions.

Dealing with all such problems is possible with modern design software, and this has not only a combinatorial aspect, but also a statistical one. The latter is just because many variable variations are not completely known, having a statistical nature (e.g., production tolerances). Often design techniques are directly related to statistics, like "Verify that simulated production yield is above 99% for all valid environmental parameter combinations" or "Make sure that the standard deviation of the offset voltage is below 10 mV".

Having software for difficult statistical and optimization problems is quite a second "revolution" which has taken place *now* in the industry. Mathematically the step from simulation $\boldsymbol{f}(x) = 0$ to function minimization (e.g., via $\boldsymbol{f}'(x) = 0$) is not so large, indeed there are many similarities, and also statistical methods often pick up optimization algorithms. These beautiful things in math help a bit to find consistency regarding the topics and new methods we address in the book.

Not at all, this means that experienced designers have to through away old "manual" techniques. Actually the opposite is true. Many times we will see that basic math results fit very well to the designer's intuition and how they act, how we anticipate problems. However, often indeed we can further improve amazingly. Two nice basic examples are corner analysis and Monte-Carlo. To some degree these are "optimum" techniques already, but still modern math offers further enhancements like adaptive worst-case corner finders and low-discrepancy sampling. A third example is design tweaking: Often it is done manually by parameter sweeps, but clearly more efficient optimization techniques exist. Not only in math libs, but applicable to complex state-of-the-art chip designs.

That is about 2016, so why many people have not heard about this "revolution", and what we can expect further? Important for researchers, for young students, for investors! Often "time is ready" for certain innovations, so actually many people presented the "first" telephone or "first" electric

light bumb. Gary Nagel's SPICE (Simulation program with integrated circuit emphasis) was extremely successful; and his software picked up many ideas which are available right now at this time, for the math part it was e.g., having sparse matric solvers. SPICE was also easy to use, and the available models were good enough, so everybody learned it, and the success was so large, that even some bugs become a feature (even a reference). The "second revolution" has not yet lead to a kind of "standard" program which designers have to learn, so there is more to read than a manual. At the moment, there is even a "war" on "high-sigma methods"; read why we think it is more a little banter, because also the person in front of the computer screen makes a big impact. However, what is needed for design, the "second revolution" little pieces, and the bigger ones, they are all *present* already and *will remain*! This books is for experienced circuit designers who want to dive deeper, but also for young designers and student in circuits and software. Some will become the next famous personalities in the big industry around electronics.

We have met people who are already working on the next "big thing" in engineering, and what could be this, *technically*? Designers follow a *strategy*, e.g., you need to solve the most critical problems first. However, sometimes it happens that something unexpected occurs, and e.g., other effects become important. So you need to change your strategy. You can also do so in math and computer programs. "Learning" or "decision making" is maybe a bit too much, but creating adaptive, more flexible algorithms is clearly a hot topic since years in research, and it will find a way into real design software too, enabling a true third revolution.

However, now, in this book, there is enough to say for transferring the "second revolution's" techniques to designers, giving a survey, a field guide with many circuit-related examples. Keep yourself up-to-date! We go beyond simulation, go for optimization and statistics; use them in circuit design! This often means getting answers to urgent but difficult design questions, e.g., regarding sensitivities, nonlinearities, design risks quantification, etc. And because already pure optimization, pure advanced statistics are sometimes a bit difficult, we focus on the real interesting, the real circuit-related stuff. This is unique and why we hope to find many readers.

Generic versus commercial. We present many examples and results, and we also add few screenshots from commercial environments to be authentic. This does not imply any judgments, e.g. on tool quality or preferences; and in some cases we unfortunately haven't received the permission to publish our results.

Sometimes the use of screenshots limits unfortunately the printing or display quality a bit (e.g. compared to vector graphics), even if the shot from the tools *itself* is perfect, but we still feel showing that a *direct* tool output *is* available is sometimes important.

Of course, some presented algorithms are almost brand new and will find application in hopefully near-future EDA tools. Often such new methods itself are not that much "present" in the user interfaces; they "are" just a *click* to enable a certain setting.

The real great advantage of modern design environments e.g. against automation by batch files, is that many integrated *debugging* features are available. If e.g. something gets wrong, you can often see it marked red in a table, often with hyperlink to the source of problem, or with a context-sensitive menu providing further options, like cross-probing to schematic, getting a plot, creating a simulation subset related only to the critical parameter combination, etc.

No Fear about Math

For the first time, we deliver circuit design methodology, math, and[1] tools in a well-aligned package. We try to do this according to the golden rule that no scientific method can replace common sense. Actually, anyone *has* a certain gut feeling e.g., on probabilities—like that p is equal to 1/6 for a rolling dice. We want to extend this to treat more difficult problems, like for yield cases with more extreme numbers, or problems with many more variables, including non-uniform or non-normal random variables. We will clarify about judgments on *rare* events and present techniques to treat them correctly.

Besides the book and e-book, there are also two real-time applications, which give you many more insights and a true "feeling" for both optimization and statistics. Complex circuit design environments are simply not designed to give the user a direct quick feedback on anything! Circuit simulations are *time-consuming*, so learning on circuit examples purely based on SPICE in an electronic design environment takes simply too long, and in addition, the graphical tools are often too limited, just because they are optimized for design purposes; and not for algorithm comparisons.

[1] In scientific papers, it is not common to highlight or underline important parts. We will not follow this ill convention, because it makes the understanding harder.

Of course, also the electronic design automation (EDA) R&D departments do not only use the EDA products; of course they use *dedicated* math and statistical software (such as Matlab® or R©). On the other hand, *as much as possible*, we will stick to real design-related examples, because they are simply more convincing compared to simplified, non-circuit simulation-related spreadsheet examples.

Note: Searching for EDA and related keywords gives you often already quite many hits, but CSE (computational science and engineering) is a more general term.

A few words upfront to the math in our book: We try to keep it as simple as possible, but not simpler, because as an engineer you should not rely too much on hope; often you need hard numbers! Most designers are fully aware of the fact that Monte-Carlo results have some randomness, often even some systematic inaccuracies, but *how* large these errors are and *which* techniques are best suited to minimize them is a key question. We focus on concepts, prerequisites, intuition, pictures, examples, and rules of thumb truly linked to circuit design, not on proving mathematical theorems. *Logic merely sanctions the conquests of the intuition (J. Hadamard)! So skilled intuition is our main target.*

On the other hand, it is important to know what is really proven and what only a meaningful assumption is! Marketing material is usually not good in this aspect, and even some mathematical techniques have fancy, a bit too fancy, names: like *maximum likelihood*—can there be something better? or *confidence intervals*—you do not trust them?

In quite many cases, we have to look to the real details; too many scientific articles are not suited as true field guide: They may mention also difficult cases, but *why* problems occur is too often unclear. Usually, this is exactly the most interesting part: If a circuit does not work, you may choose another, but a better idea is to repair, modify, and extend it! As designer in hardware or software, you simply have to do so and to learn.

Engineers are luckily often pragmatic, and you need to be, because you need to apply methods for solving non-trivial problems. People usually transfer things they learned on simple cases to more difficult problems, but sometimes really something essential changes if you move from a one-dimensional problem to two dimensions or even to n dimensions. Examples are helpful, but sometimes also misleading. Even experts can fall into this trap, and simplifying concepts that work well in some cases (like assuming normally distributed data or replacing true parameters by sample estimates, or using

standard confidence interval formulas) can fail completely in other cases, other theoretical cases, but also in real circuit design! This is because analog circuit design can be very complex and highly nonlinear! Concepts working well in one field may fail completely in others, and this triggers the creation of more advanced theories; even when having such an advanced theory or method—like maximum-likelihood estimation (MLE) or bootstrap—people may be overenthusiastic, and limitations may be discovered later—we will give examples for this. Just one is Latin hypercube sampling (LHS), which has been implemented in many simulators for ten years or more. The idea is good, and initial benchmarks had shown a significant speed-up. However, for complex circuit designs, the advantage disappears too often—and there are better methods. This book is not at all on benchmarking different algorithms or tools from different vendors, but of course we do core algorithm comparisons; usually on difficult but still easy-to-understand cases—often these are also representative. Unfortunately, there is no single example test case on which you can show everything—analog examples are often not as easy to scale as digital ones. So, sometimes we also use mathematical examples on which you can indeed prove certain algorithm characteristics, like quadratic convergence of an optimizer.

In fact, this is not at all a mathematical book covering statistics and optimization *in general*. Also we assume that the reader knows about key circuits like amplifiers or filters, but the circuit topology is usually not the primary interest. We need to focus on techniques that work for circuit design, so you will not find detailed discussions on ANOVA, hypotheses, Runge–Kutta integration or linear simplex optimization, yet references will be provided for the readers who need to revisit those concepts in more detail. We show also methods dealing with non-statistical variables, because they are of course important for circuit design and we will see many similarities, e.g., when talking about coverage, sensitivity and correlations.

The authors have both a strong circuit and a mathematical background; remember from school and university that statistics and matrices can be a very boring topic. In this book and by the inclusion of auxiliary software, we want to demonstrate the opposite. Also we always want to relate good manual design techniques to the math background. Learning something challenging is often difficult, but not if you really have to solve your own (hopefully) fascinating highly motivating problems! For instance, what is a designer doing if he/she follows a certain design strategy and how can a design software act in a similar way? Tools can remove boring work from the designer, so that he/she can focus on more fruitful topics, like on exploring different circuit or system

topologies and for preparing your design for a perfect design review. The great thing with computers is that you can often verify amazing things with little setup effort, and you can also use software not only to your direct advantage of solving a specific problem but also for becoming a better, more experienced engineer! For this reason, we do not split the book content in a theory part and examples; instead, we always want to apply algorithms immediately on interesting realistic—often only slightly simplified—design tasks. You should start with such basic understanding to build up more understanding and to be able to explain what the general phenomena are. Actually, the source of all great math is the special case, the concrete example; it is even frequent that every instance of a concept of seemingly generality is in essence the same as a small and concrete special case.

Educated application is one key for success, because there is no "best" algorithm in general for complex tasks like circuit yield optimization. The better the selected method fits to your problem structure, and the better you set convergence parameters and starting points, the shorter the way to success. Custom IC design, statistical analysis, and optimization are difficult topics, so several highly automated and efficient methods have been already used very successfully for more *special* topics like digital circuits (here you can focus on timing and leakage) or memory circuits (here the focus is on read and write capabilities), and here, we may use at least partially analytic expressions instead of running full circuit simulations [Jiajing Wang]. Also in "SPICE model fitting", you can find very efficient algorithms, because you optimize all the time very *specific* models with *fixed* parameter sets to fit for measured data (having also a fixed *structure*, like IV and CV data) and using usually the least-square criteria. Use such methods *if* possible, but here in the book on analog variation-aware design, we typically need more *flexibility* and will describe more generally applicable methods and tools.

In our examples, we need to make a balance: Real ASIC designs are often very complex, and a big mix of techniques is required to solve all problems, so we usually have to apply some simplifications to be able to directly apply and demonstrate new techniques. For instance, the way an optimizer takes in the parameter space is hard to visualize for more than two variables, but luckily almost 99% of the optimization problems can be fully explained with 2-variable examples! Actually, the real advantages of an advanced optimizer become often much more prominent if you apply it to more complex problems, like with more than ten variables. In addition, we wish that the reader is really able to follow and apply the key ideas—the limited book size and often intellectual property (IP) issues unfortunately prevent us to showcase very

complex examples. However, when e.g., looking at an optimizer log file, the user can see that state-of-the art tools really use the described algorithms, and he/she can observe in them the same problems as we demonstrate in our shorter examples. The main difference is only that in big problems, you typically have to fight against multiple problems and the runtimes are much longer!

Hi! This is us, the authors! This is a book, and learning from books is often a bit more difficult, because in long sentences it might be not clear where the key point is. In teaching, you can *stress* words, but in scientific articles, it is not usual to use underlines. So we do not follow this tradition! If we write, e.g., ... almost accurate ..., it could be quite a difference if the focus is on *accurate* or on *almost* or even on both, and if both in which sense? The context for us is usually circuit design, but sometimes it is indeed up to you to decide whether 99% accuracy (or yield) is good or not. Also in medicine, you do not need always a high confidence level, and in urgent cases, better give a heart massage! Modern statistical algorithm come of course with internal accuracy checks, but also in a circuit simulator, the checks may be not done in the way *you* need. In a transient simulation, you may want 0.1 dB accuracy for your FFT, but that is seldom in sync with simulator settings like *reltol*. That is the reason, why engineers do the work, and actually change the world with chips, smartphones, and software—not that many designers, maybe many in electrical engineering, quite many in IC design, some in EDA, and quite a few in statistics and optimization. Luckily, these techniques are a hot field also in other areas.

Front-End Design Flow

To follow this premise of circuit-oriented concept, let us start with a little sketch on an analog design. "Analog design" can have different meanings—our focus is analog design in its *widest* sense, including RF design and mixed-signal and custom digital designs. In a single book, you cannot describe each aspect in full detail from a bare semiconductor wafer to the final tested product. So we focus on what is often called design "front-end" flow, the part that includes defining the circuit in a schematic entry, setting up a simulation testbench, verifying the circuit behavior and checking specifications, plus tweaking the circuit component values; so in our methodological context, "front-end" has not the same meaning as in RF (radio frequency) or sensor design. A very typical approach in analog design is shown in Figure 1.

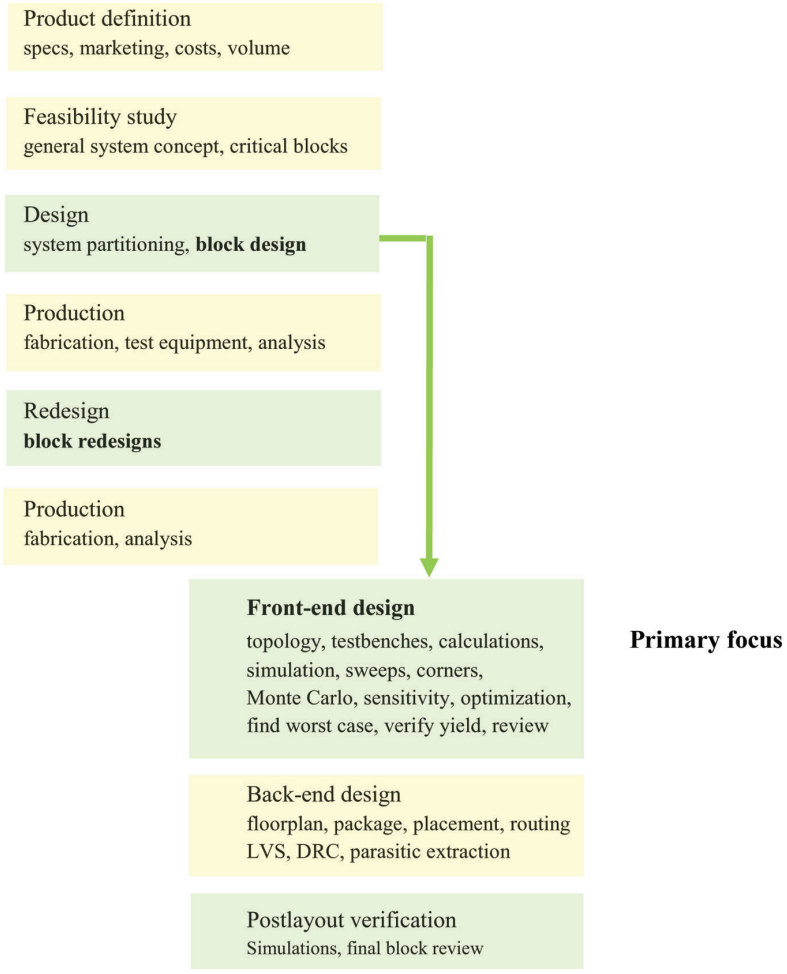

Figure 1 General IC design steps.

Imagine an RF receiver has to be designed, and it might be part of a bigger transceiver chip or even of a full system-on-chip (SoC) containing also bigger digital parts, a PLL synthesizer, ADCs, DACs, etc.

- A system designer (team) has decided on the concept; for example, based on experience and system specs (like IEEE802.11 for WLAN), it has made a certain system partitioning into sub-blocks such as low-noise amplifier (LNA), mixer, variable-gain amplifiers, several filters, an demodulator.

- Next, the system gets refined further, e.g., by creating a detailed frequency plan; with system simulators, MATLAB® and spreadsheets, the system designer will also create a level plan giving an overview of the ranges for wanted signals, interferers, noise, etc. Step by step, the designer can hopefully obtain a consistent and realistic set of sub-block specs (such as gain, area, and power consumption) derived from the top-level system specs. We need to make sure that the system works even in the presence of many known imperfections, such as noise, filter tolerances, group delay ripple, jitter, mismatch, and different kinds of nonlinearities (compression, intermodulation, etc.).
- At some point, a circuit designer has to work on "his" block and related sub-blocks, e.g., an operational amplifier (op-amp) being part of a bigger filter. The block designer either will take an existing circuit or may compose a new op-amp, e.g., based on the input and output voltage range, the supply voltage, gain, speed and noise requirements, etc. For the latter, he would execute several hand calculations for the different component values in the circuit (e.g., based on power budget, g_m, on-resistance, and slew rate). In both cases, he/she needs to verify the circuit behavior in a set of testbenches under all the different environmental conditions. For instance, he/she would do sweeps on temperature, supply voltages, load resistance, load capacitance, etc.
- Usually, some design weaknesses will become visible, like we consume more power than allowed, so the circuit has to be modified, e.g., till the variations are acceptable (like in sync with hand calculations) and till the design is fulfilling all specifications (being "in-spec").
- Of course, we should also perform simulations together with neighboring blocks, like the usual bias generators—if available. At the very end we could end up in a top-level simulation, which is often a mixed-signal simulation (mix of transistor-level blocks and behavioral descriptions).
- Besides simple one-parameter sweeps, we may also do 2D sweeps to account for correlations or pick certain critical corners (combinations of parameters) directly for debugging and design tweaks. In those 2D sweeps or critical corners, the design is often more stressed than in simple sweeps, so tweaking becomes harder and maybe we need to extend the circuit a bit or even need another circuit topology.
- To account for offset voltage, we may do a Monte Carlo (MC) analysis and get a certain standard deviation for it. With corners and MC, we could check the design quite well, and we can calculate the overall most extreme behavior; for example, we can add the offset errors from systematical and statistical imperfections (e.g., PSRR and random offset).

- All in all, at some point we might be completely in spec with enough margin and create a layout ("Back-end" part in Figure 1). From the layout, we can calculate wiring parasitics and we can include them in our simulations. If our initial design margins were large enough, also the postlayout verification would indicate that our design is really "ready for production" (tape-out). Hopefully, we included all important aspects to our verification, so that indeed also the production samples work as expected.

This "simple" flow shows that analog design can be quite efficient. We have many well-established standard techniques, but also some key questions arise immediately; we will address them in our book:

- Imagine your circuit simulations show good results in a corner run and also in a short MC run: *How much does this mean with respect to yield? Is it really ready for tape-out?*
  This has many aspects, and we will guide you (e.g., on MC count) and show you common pitfalls (like assuming normality). We will explain how to get most out of your simulation results, more than just sigma and a histogram!
- Imagine your specifications are very difficult and hard to achieve. *What should be the next steps? Which strategy should you follow?*
  We will create a systematic flow from design start to sign-off verification. We give you an overview on the methods and explain which ones are suited and how to set them up. This also includes optimization techniques (Chapter 8); we tell you when they are useful.
- This leads to the often asked question: *What is the best way to make a design?*
  "Define best!" is probably a good answer. "Best" in the sense that it would always work would lead to a big and very complex flow! "Best" in the sense of efficiency would lead to a flow based on a lot of experience. So "overall best" is for sure a mix of techniques, which makes it sometimes difficult to see a systematic behind the flow. On the other hand, we will show that even when you use methods with strange names (like "high-yield estimation"), you are often still using something close to the techniques you have applied manually from time to time! Generally, there is no need for big changes in your habits.

Actually, one of the nicest questions I ever heard as consultant was this: *How much do I need to overdesign in Cadence to make it working in reality?*

Cadence® is indeed a synonym for custom IC design tools, so technically it would be better to ask for "overdesign in *simulation*"! As analog designer, you know the answer of most questions is "it depends." If you know your laboratory measurement equipment has an uncertainty of 0.5 dB, then you can use this as safety margin and specify it in a datasheet. However, it is quite some work to find out the margins for all the other effects like temperature drift, aging, circuit tolerances, modeling, Monte Carlo variances, and simulator inaccuracies! Ultimately, the whole concept of margins is very efficient but leads to severe errors if applied in a too simplistic way. We will tell about the risks and extend the concept!

Is overdesign a problem? Regarding verification, it is no problem, but for being competitive it could be—especially when talking about bigger key blocks and/or critical subsystems. Actually, it is quite hard to find out whether you are overdesigning or not. You have to run the circuit under extreme but still realistic conditions! Finding those is one key task in variation-aware design, and optimizing the circuit to meet the specs and/or improve on yield is a second key technique. Both are closely related, e.g., in both sensitivities, and search algorithms are important; the more you read and learn about it, the more links you will find, and the more intuition you will get, the better you can design by problem anticipation.

Actually, our op-amp and filter example is just one simple example, and often each described step (or at least one) is far more difficult, especially when designing critical blocks or using the newest silicon technologies. So Figure 2 shows some major trends, and many of them can make circuit design unfortunately more difficult.

One trend among many others [Graeb ITS] in modern IC design is that already the blocks become more and more complex, e.g., to enable multimode operation. In our amplifier/filter example, one may decide to put several stages together—into a kind of subsystem, often including bias circuits and calibration blocks. So the meaning of "block" design can be a bit fuzzy.

Design and Verification

In IC design, many individuals and teams are involved, and they often have different opinions on what should be improved. For example, behavioral modeling is a high-priority problem too, as variability. For both, the benefit

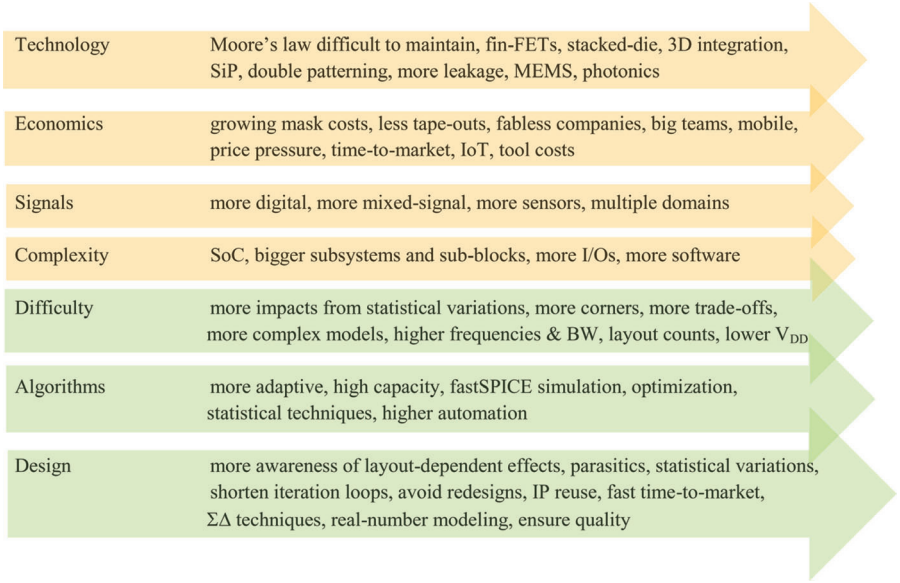

Figure 2 Trends in IC design.

is larger, the earlier you use them. In software the creation of a "design" is quite easy (e.g., due to availability of powerful libraries), but testing can be difficult and very complex; and in digital design the situation is often similar, because reliable standard cells and synthesis tools are already available. So in these communities many people say *only* verification is a problem, we have a "verification gap". This is not completely true for analog and custom designs! Verification is a difficult topic too, as almost all kinds of designs become more and more complex and harder to test. So nowadays, there is pretty much talk about the "verification gap" or "productivity gap." Best don't let you bother and just take the best ideas coming up.

When digital designers talk about verification problems, they typically have complexity in mind, like how to find input test vectors leading to a bug. However, it does typically not mean that they cannot run the testbench with that test vector anymore. However, in analog, mixed-signal and RF, indeed, e.g., a single postlayout simulation pushes even a 1 TByte RAM multicore compute server to the limits! However, there is seldom a verification problem in the sense that you do not *know* what to simulate! On the other hand, for sure also digital designers push their tools to the limits (especially when thinking of HW-SW co-design). Also "new" digital techniques like randomized tests and

coverage-drive verification can make highly sense in context of mixed-signal or even full analog designs.

In analog, there is also a true "automation gap" in topology selection and initial circuit design. Here, a mix of techniques is required. No single commercial tool offers here "push button" solutions—quite opposite to simulation and verification. However, some of these things are also the fun part of analog design in general, so why automate? A tool needs to be really good to convince in that domain! On the other hand, luckily, there are advanced design tools plus powerful compute servers that still allow efficient design plus giving the engineer a great cockpit for steering the design, implementation, and the verification tasks. Further improvements like tool support for parasitic and layout awareness are in trend, giving not only speed but more insights, thus also making the designer better.

We are also not sure whether there is really something like a "productivity gap" or if people need a new smartphone every six month. Software is a big lever to improve your individual productivity, like just designing more transistors (or lines of code) per day, but design *difficulties* are hard to quantify. Terms like "transistors per hour" or "mm^2 chip area per week" are only very rough measures in both front-end design and layout. You can spend weeks on an LNA or months on an RF PA having few transistors only and you might be highly productive, because the person at the competition fails on this problem. Especially in the field of statistical methods and optimization techniques, a breakthrough is observable; so any engineer working in that area should have an interest to become familiar with them. They help you to identify the critical parameters in your design and quantify and minimize their impacts, till you are confident enough to tape-out.

The biggest step forward in analog design in the last 30 years was clearly the introduction of standard circuit simulators, allowing an almost virtual verification, instead of pure breadbording and pure hand calculations. Also some discussions in engineering are also quite old: If there would be "only" a sign-off verification problem, people may say, why not going for a fast production lot, wait four weeks and measure everything—*only* the *silicon* tells the (full) truth!? The big problem with such approach is that the debugging will not be much easier that way, and actually, it is far better to use advanced verification tools, modeling techniques, etc., already from the beginning, for the design phase, not only for sign-off verifications. This way you can improve on product quality, design understanding, reuse IP, and reduce need for costly iterations (especially long, time-consuming iteration loops). In addition, even the silicon is not the truth, e.g., in RF or high-performance ADC design

packaging, external matching networks, supply bypassing, etc., can have a big impact! In addition, the more complex a design is, the more virtual the design and the verification of it will be, naturally, just to reduce risks and costs.

Verification can be indeed treated quite well with systematic mathematical methods, but we feel that this is a too narrow approach, because whether a design is good or not can be often decided much earlier, so it is better to avoid problems early, instead of finding them in a final verification phase. Plan for this as early as possible. Explain how difficult things should be verified in a verification plan, best in conjunction with the target datasheet. Iteration is always required, but keep at least iteration loops *short* and improve your understanding, and the understanding in the whole team.

Therefore, looking at the whole front-end flow as a pure verification problem is a trap! Solving the "verification gap" is often much easier if you do not split the flow into parts, better take a more unified approach and include manual best practices for circuit design. A good way for solving complex problems is to anticipate the "disturbing" influences on the design, to analyze them; often the results from techniques already lead to solutions—step by step, starting with the most urgent problems.

Let us pick up an op-amp example again: At the beginning, the design is almost for sure not working as desired, at least not in the whole operating range. You have to pamper up your baby design and make it work under DC nominal conditions, making it work under wider conditions (over temperature by deciding on a certain bias concept, or for constant performance over a wider supply range by adding cascodes, etc.) and also for your typical input signals, till you end up in a good and robust circuit that works also under the most difficult allowed conditions. In such a flow, the design and your knowledge about it grow in parallel with the verification! This way verification or complexity becomes much less of a problem, and more design insight is a further valuable output.

In digital or software design, there is a trend to split the parts of design and verification, i.e., different people do it. This solves the problem that a designer may "love" his "baby" too much. Actually, the idea that a verification engineer should "hate" the design is good and can lead to better verification, but for analog designs, it should be only a complementary concept. In critical cases better create several "baby designs" and let the best "survive"; once you have made them detailed enough for a fair comparison. Design is often a fight of ideas! Often also a split between front-end design and layout—usually seen in bigger companies—is critical, e.g., misunderstandings on priorities or constraints can lead to unnecessary extra work or just bad layouts.

Layout aspects (or more general the "physical implementation") are important, and the reader is highly encouraged to also read books on technology, layout, and device modeling! Many of these topics will be referenced in our design flow: Modeling is a limiting effect on verification accuracy, and the layout part (unavoidable wiring parasitics and "bad" layouts, respectively) is often a reason for design iterations—unfortunately.

Note: In this book, we concentrate mostly on front-end design not on physical implementation aspects like layout and packaging (or ESD and latch-up), but for sure you also need to include these extra-effects (plus sometimes others like aging), and often, they are as essential as supply voltage or temperature effects. So you should also include them, as soon as possible—not only in final verifications. For some more details, read Chapter 10.

For sure, also simple bad design practices can let design projects fail or at least delay, e.g., last-minute changes without careful re-verification, creating different block versions but not making clear which one to use, hoping that someone else has already verified something, not applying manual checks (often automated run decks for DRC, ESD, latch-up, etc., are incomplete), using models in extreme regions (breakdown, ultralow currents, etc.), or typical interface problems, e.g., between blocks from different departments. Actually, communication is a key part for success. Partly, it can be also supported by EDA tools, e.g., via constraints. Table 1 presents some reasons for re-designs which have been appeared over the years to the authors. Note, that the table focused on first, almost immediate redesigns; in later stages, after a more detailed analysis, problems introduced by variability are much more typical.

At some point, it might be impossible to simulate the design, like a postlayout simulation (maybe still excluding substrate, self-heating effects, aging, ESD, latch-up, etc.) will take a month. In such cases, it would be better to go for production, but being sure to apply at least careful manual checks and calculations. On this, do not underestimate the human factor: We have seen design fails and need for a redesign, just because designers claim "I checked this in Monte Carlo," but due to a small bug in the design kit, it was simply completely impossible to run MC. So the designer was lying so and so.

You may think why everything is so difficult? It should be not a matter of complexity, and it is more on following an important rule that says:

> *"Know what you want, and use what you have. Make no bullshit, do not rely on hope or magic. Double-check your assumptions. There are no stupid questions. Be an engineer!"*

Table 1 Few reasons for chip redesigns

Block	Chip	Class	Comments
Logic part	RF receiver	Last minute change	No full re-verification applied after change. Solution: FIB applied, logic changed in redesign
Bandgap	RF power amplifier	Modeling	Substrate transferred the RF output signal to sensitive bandgap net Solution: Non-Miller-cap frequency compensation of the bandgap, better shielding of pads, FIB and metal redesign
Substrate contacts	RF PA control IC with negative supply	Layout	Contacts connected to gnd instead of V_{EE}. Solution: Layout modified in metal redesign
g_{m}C filter	IR remote control	Variability	Only corner simulation done, no analysis for mismatch Solution: Larger transistors in bias part
Buck converter	Hearing aid chip	LVS	Incorrect LVS setup for self-created metal capacitors. So a short circuit was not detected. Solution: LVS run deck improved, via causing the short removed
RF power amp	RF power amp	Modeling	Amplifier had strong oscillation tendency, modeling for package, substrate, etc. too simplistic. Solution: Modeling improvements and concept change to a narrow-band amplifier
Memory	SoC	Variability	Circuit function correct, but yield too low. Solution: Yield improvement redesign on critical parts.

This book wants to show and explain further advanced methods—beyond simulation and verification—that work for creating analog designs. When we talk about "analog", we include also mixed-signal, high-speed, or custom digital and RF designs—so all kinds of designs where typically a highly flexible design strategy that is required. Not always mathematical and automated techniques can be directly applied, especially for the art part of circuit design, but for sure the described techniques can help good designers and

students to solve more difficult problems and to speed-up the design process in several ways.

We feel also non-math experts, but engineers should know what they are doing when using certain electronic design automation (EDA) software. Users need be able to select and set up a suitable method, because (like circuit simulators) also the new tools come with many options. This book will give you guidance and an overview and should enable the reader to get in touch also with more advanced original papers and more specialized mathematical literatures. Those are clearly recommended if the reader wants to get highly detailed information or creates his own algorithms. This marks also the point where we need to stop in this book and we want to focus on the really important concepts, algorithms, and tools.

The book is written for engineering students and engineers who have already some background in standard techniques like just creating a little design and simulating it, and knowing about technology variations. So we address simulation techniques themselves only in aspects relevant to variations, statistical methods and optimization. We hope we can motivate you, as the reader, to think more about the topics we cover, and you should be able to understand, able to work, and probably also able to tell, to talk, and (at some point) able to teach.

Actually, too often designers end up in SPICE monkeying, hoping the simulator makes the design job. Sure, many things base heavily on models and simulation, but at least you should do it in an efficient way, doing the really required simulations, with some realistic margin and just some acceptable overhead for confidence and understanding.

Often EDA vendors are asked: How much does this tool help to increase the productivity? In some cases, you can indeed quantify this, but we feel—even in the book—we partially focus too much on such a simplified view on engineering. For example, reducing the number of simulations can lead to a measurable speed-up and reduced costs, but actually other aspects are of equal importance, like being able to manage complexity of new system architectures or advanced process nodes, and getting confidence and understanding of your design.

Organization of this Book

The topics we treat are both relevant to scientific researchers, EDA software engineers, electrical engineering students, and circuit designers. You do not need to be an expert in everything; we point you in each chapter also to the literature available for further reading—to more basic and sometimes also to

more advanced references. In the appendix, you can find an ordered list of all references, and we think it is a good mix for getting a deeper introduction and also for digging into the details.

Of course, unfortunately, we have to deal with many abbreviations and special terms, and we collect them in a big list—many are very common, and you should know them.

There is no eat-all-or-nothing, the book has a modular structure (Figure 3): The advanced designer may skip the introduction and the chapter on manual design methods and could jump to the advanced statistical techniques; or the

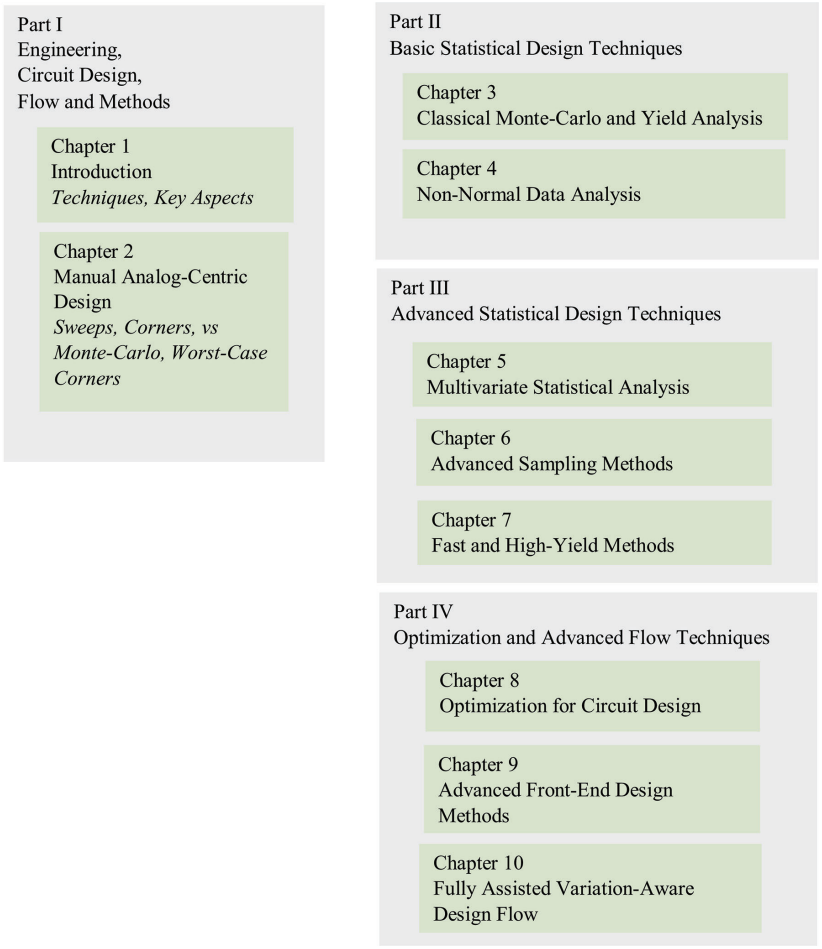

Figure 3 Design techniques and book chapters.

statistics expert could directly go to the optimization chapter, but we think if you read the book page by page, you will not miss clear guidance. Topics that are interesting but not key for further reading are delivered in separate boxes.

We always start with the problem descriptions and discuss different methods to solve them, starting from one of the most straightforward approaches and then moving to more advanced techniques that typically are more accurate or more efficient in difficult cases. The examples we start with are typically quite short because with too complex ones, we would lose the focus too much. Therefore, we shifted several more complex examples into separate subchapters. Of course, also our initial examples are often not so simple in some aspects, e.g., for circuit design, it makes little sense to talk about "linear" optimization, so starting with a quadratic function is quite native, and even such an example can be simple like $x^2 + y^2$ (because here you can optimize both parameters independently and even the simplest optimizer will work very fine) or more difficult like $x^2 - x \cdot y + 100\ y^2$ (here, we have correlations and highly different sensitivities).

The book starts with two chapters on circuit *design* and design *flow*, not so much on circuits. We explain the most important design targets and measures, like yield, corners, and worst-case and Monte Carlo analyses.

In Chapters 3–7 we discuss statistical methods in detail, starting with Monte Carlo and basic result analysis for yield, sample variance, confidence intervals, etc. Next, we extend the method in Chapter 4 to non-normal data analysis, which includes a generalized process capability index C_{GPK}. All these analyses are possible for data obtained both from tests in the foundry or laboratory or from MC simulations, but the simulator has really access to all statistical variables in your design, so you can also do *multivariate* analysis to obtain correlations or to fit complex models to MC data. This is the next step, before we also address MC speed-up techniques and non-MC statistical techniques like the worst-case distance method, which is often much more efficient for high-yield verification. Chapter 7 is a long chapter, just because there is no one-size-fits-all method, almost each method you can break, at least in special difficult test case—and you should understand why.

Chapter 8 is explaining optimization methods. One key part in setting up an optimization is the goal and constraint definition, because often optimizing the circuit under typical conditions is not enough, and at best this could be used as a starting point. Instead, you should really inspect the most critical conditions, and for this, you have to treat both statistical variables and the ranges for your operating conditions like supply and temperature, and then you have to optimize on this. The difficulties and the benefits of such overall optimization

approach are discussed in Chapter 9, together with further advanced front-end design topics. To some degree, we leave here the area where most design environments have fully built-in solutions; so some scripting or manual interaction is required to perform very advanced tasks. Actually, assembling worst-case analysis and optimization is a key part for obtaining a complete and automated or at least a highly assisted variation-aware design flow.

The overall flow aspects, including layout, design-supporting tools, and IP management, are described in Chapter 10. Variation-aware design is not at all only about process variations, mismatch, temperature, and supply effects, and the physical implementation can change the circuit performances sometimes much more! This means layout effects, such as wiring parasitics, may be substrate couplings and nowadays even the neighborhood of each critical transistor matters, so you should include them as soon as possible. Our summary (Chapter 11) includes also an outlook for upcoming further advanced design methods and environments.

Real-Time Apps and Auxiliary Material

Please regard the software as an *integral* part of the book! This does not mean that anything is really missing without these pieces of software, but to really understand what the circuit design is, you have to do circuit design; to learn what statistics can do and what not, you should look at statistical data, at best from circuits!

In some way, EDA design environments are already quite good for learning! The only pity is that circuit simulations can be very time-consuming, so it is almost impossible to get a "feeling", e.g., for how large the variations from one MC run to another could be, just because already *one* "meaningful" MC run can take a day!

How can you learn driving a curve with a speed of 1 mph? You cannot, you need real-time speed to really get the feeling; and for statistics it is very similar. The human eye and the brain has different mechanism to interpret what we see; there is a static and a dynamic part. It is best to use both.

Both apps (see Figure 4) run under Microsoft Windows®, and a detailed documentation is delivered as separate pdf files. With both programs, you can make your own experiments on optimization and statistics, to learn interactively and from many examples just much faster than is possible within a design environment, and to some degree, also more details can be provided.

In the book, usually at the end of each chapter, we provide a collection of questions and answers, and the two apps can support you to answer them.

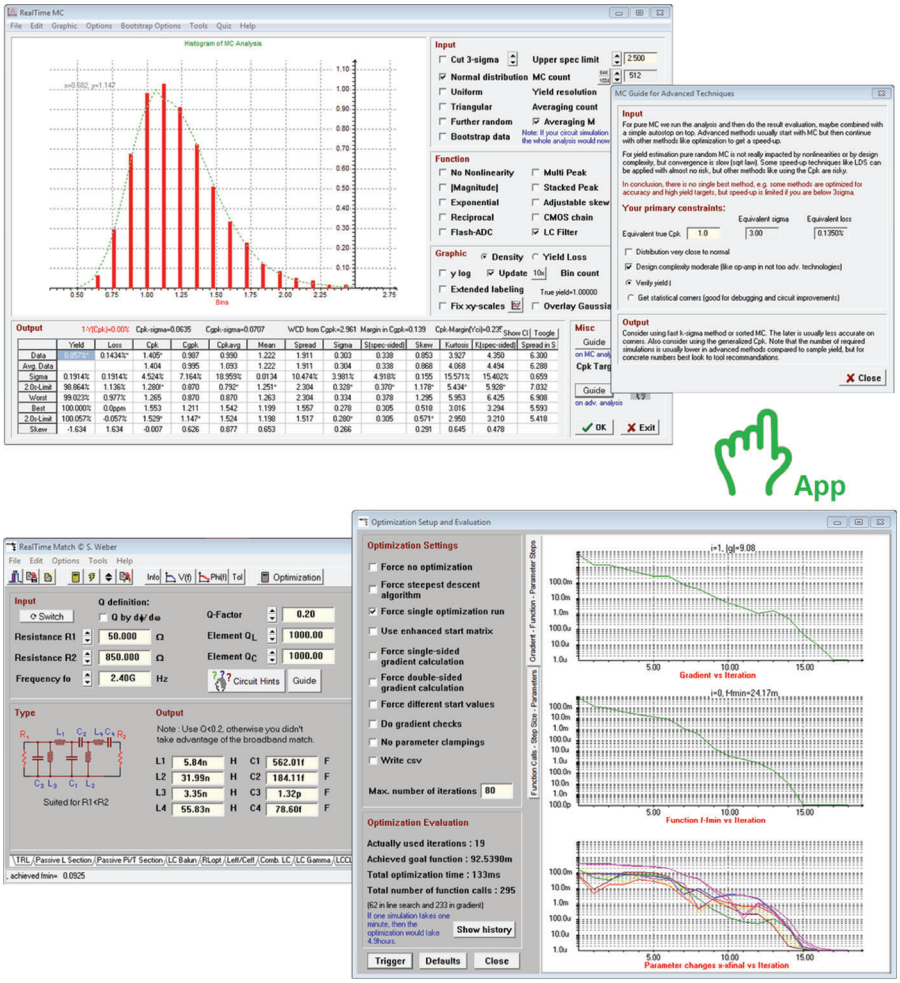

Figure 4 Screenshot of the two learning and design programs RealTime MC and Match.

When you could use one of the apps for solving problems, we often add a little green hand in the book. The newest app versions are available under http://www.riverpublishers.com

App

It could also make sense for more complex experiments to use your IC design environment, and we mark this and connections to further small design tools with a blue hand. At the River Web page you can download supporting material.

www

In addition, we prepared some statistical experiments with Excel®. These you can also download at River Publisher.

In the last chapter, we talk about layout effects and intellectual property (IP), and the creation of it. For that further design programs are available and provided by the River Web page!

These books we highly recommend as a refresher on circuits, math, and statistics:

- Willy. M. C. Sansen, Analog Design Essentials, Springer US, 2007.
- Tony C. Carusone, David A. Johns, Kenneth W. Martin, Analog Integrated Circuit Design, John Wiley, 2012.
- R. E. Walpole, R. H. Myers, S. L. Myers, K. Ye, *Probability & Statistics for Engineers & Scientists*, 9th Edition, Prentice Hall, 2012.

In the different chapters, we point you of course to further, more specific references. Note that especially for mathematical topics, there is a big amount of free and great material available in the Internet, and in the appendix we also give links to such *highly valuable* material. There you can also find links to the homepages of several EDA vendors. Many of them offer a forum and advanced material as download.

Software tools, Math, Statistics and Numerics? Doing math by dealing with data and programming are the best approach to become an expert. In our book we present many spreadsheet examples, not because such tools are best, but because such tools can display data quite well, and the user can see each step applied to the data. Of course, at some point you can push a spreadsheet to the limits, like analyzing 10 million data points; here true programming languages are best, like C, FreePascal or Java. Also in these many libraries for charts and math are available, it is just some work to put all together. The most elegant way for doing math is using math environments like Matlab® or R©. Matlab has the advantage that even a graphical simulation environment is available. R is perfect for all who need access to the true state-of-the-art in math and statistics. Both software packages are quite easy to learn, some things are even easier than in Excel®!