Optimized is Not Always Optimal

The Dilemma of Analog Design Automation

Juergen Scheible
Electronics & Drives
Reutlingen University
Reutlingen, Germany
Juergen.scheible@reutlingen-university.de

ABSTRACT

The vast majority of state-of-the-art integrated circuits are mixed-signal chips. While the design of the digital parts of the ICs is highly automated, the design of the analog circuitry is largely done manually; it is very time-consuming; and prone to error. Among the reasons generally listed for this is often the attitude of the analog designer. The fact is that many analog designers are convinced that human experience and intuition are needed for good analog design. This is why they distrust the automated synthesis tools. This observation is quite correct, but this is only a symptom of the real problem.

This paper shows that this phenomenon is caused by very concrete technical (and thus very rational) issues. These issues lie in the mode of operation of the typical optimization processes employed for the synthesizing tasks. I will show that the dilemma that arises in analog design with these optimizers is the root cause of the low level of automation in analog design. The paper concludes with a short review of proposals for automating analog design.

CCS CONCEPTS

• Hardware ⟶ Electronic design automation ⟶ Methodologies for EDA • Hardware ⟶ Very large scale integration design ⟶ Analog and mixed-signal circuits ⟶ Analog and mixed-signal circuit synthesis.

KEYWORDS

Analog design automation, optimization algorithms, procedural automation, analog layout synthesis, EDA

ACM Reference format:

Juergen Scheible. 2022. Optimized is Not Always Optimal - The Dilemma of Analog Design Automation. In *Proceedings of 2022 International Symposium on Physical Design (ISPD '22), March 27-30, 2022, Virtual Event, Canada,* ACM, New York, NY, USA, 8 pages https://doi.org/10.1145/3505170.3511042

ISPD '22, March 27–30, 2022, Virtual Event, Canada.

ACM ISBN 978-1-4503-9210-5/22/03.
https://doi.org/10.1145/3505170.3511042

1 Introduction

1.1 Trends: Moore's Law and the Design Gap

Microelectronics has grown at a breathtaking pace since the first ICs were designed over 60 years ago. It has changed our lives beyond recognition and will continue to do so. This technological evolution has been made possible by rapid progress in semiconductor technology, which has enabled ever more functions to be integrated on an IC by on-going miniaturization. The top black line in Fig. 1 (left-hand scale) shows the exponential growth of IC devices on a computer chip. This is known as "Moore's law". Chips can be designed and built nowadays with feature sizes of some few nanometers, which can contain up to 100 billion transistors – so to speak computer farms in thumbnail format.

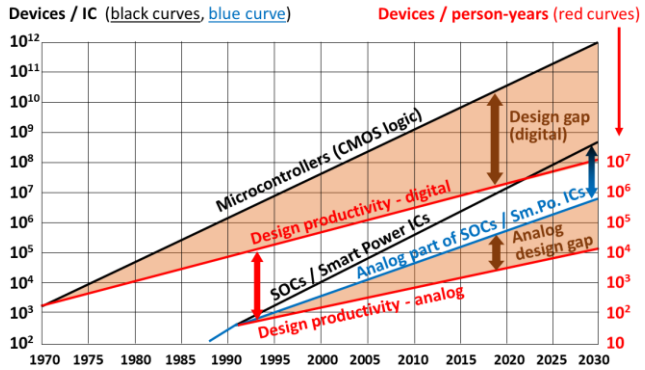

Figure 1: Moore's law, design productivities and design gaps for logic ICs in CMOS technology and SOCs and Smart Power ICs in BCD technologies

Not only do ICs need to be fabricated, they have to be designed first. This is why enormous efforts are made in the specialist field of electronic design automation (EDA) to design ever more powerful design tools for IC design engineers so that they are able to master the extremely challenging complexity they have to face. The success of these efforts can be seen in design productivity. In fact, this success can be quantified by dividing the functionality integrated on a chip (measured in the number of devices) by the required design effort (measured in person-years).

How the design productivity for logic chips has progressed (right-hand scale) is shown by the upper red line in Fig. 1. There is an exponential growth rate as well. At just over 20% per year, it is

much greater than with other technical disciplines and therefore quite respectable. This growth rate does however still lag considerably behind Moore's law. This is where the well-known design gap comes from, with the result that the development costs for a logic IC have a tenfold increase every 15 years.

1.2 The Analog Design Gap

Today's microelectronic systems contain digital logic as well as multiple analog subcircuits. The analog parts manage the internal power supply and interface to the outside world to convert sensor signals at their inputs and to control actors at their outputs. It has long been standard practice to combine such analog parts with digital logic on a chip. That's why the great majority of today's chips are mixed-signal ICs. If the power output stages are also included, the chip is called a "System on chip" (SOC), or if the chip consists mainly of power transistors it is called a "Smart Power" IC. The key figures for these types of SOCs, mentioned above, which have been produced since the 1990s in BCD mixed-signal designs, for automobile electronics, for example, are shown in the bottom part of Fig. 1.

The design of digital subcircuits for these ICs is highly automated, while their analog counterparts are still mainly designed manually. It can be seen from Fig. 1 that analog design productivity lags behind digital design productivity by multiple orders of magnitude (see red double arrow). The result is that most of the design effort (often > 90%) goes into the analog parts in an SOC, although these analog parts contain only a very small portion of the functionality (typically some few percent, see blue curve).

Analog circuit and layout design is therefore by far the main cost factor in today's dominant mixed-signal designs. This problem is known as the "analog design gap" [6]. A key takeaway here is: although the digital circuit parts (the parts that follow Moore's law) are growing to a greater extent than the analog parts (see blue-black double arrow), the dominance of analog circuit and layout design as a cost driver is so great that the analog design gap has become the bottleneck in state-of-the-art microelectronics.

1.3 Paper Structure and Goal

The basis for successful automation in digital design are optimization techniques. In the paper I want to show why this powerful "all-purpose weapon" in EDA repeatedly fails in analog design. There are very rational, technical reasons for this, which, in my view, are the main cause of the analog design gap.

In order to understand the causes of the analog design gap, we need to know how automation works in IC design and what the crucial difference between analog circuit design and digital circuit design is in this regard. I will address the second question in Sect. 2. The first question will be dealt with in the two sub-sequent sections. Some of the principle EDA strategies are described in Sect. 3. I will discuss the structure and mode of operation of optimization algorithms in Sect. 4. The experienced reader can skip these two sections covering the EDA basics without any difficulty. Using this knowledge, we will address a key dilemma impacting all optimization algorithms in Sect. 5. This leads to a performance limit, which I call the "optimization horizon" and which, in my view, is the main cause of the analog design gap. Procedural techniques and machine learning are other options that are available instead of optimizers. In Sect. 6, I will give a short résumé of the ideas put forward in the paper and add a few notes on alternative approaches to close the analog design gap.

2 Digital vs Analog – The Crucial Difference

At a first glance, it is difficult to understand why analog circuit design is so much less automated than digital design, as both types of circuit are made up of the same types of devices (mainly transistors), which are electrically connected through interconnects. They are also located on the same semiconductor base. Hence, they are made of exactly the same materials. Looking at the matter from this perspective, all one sees is that the number of devices and nets on the digital side is much greater than on the analog side. The digital devices are therefore much more "complex". Logically enough, you would expect the design of the digital devices to require much more effort? But what we find in the real world is the exact opposite.

To understand this apparent paradox, we need to look at digital and analog circuits from a different perspective – that is, the functional perspective. Continuous-time and continuously valued signals are processed by analog circuits. Digital signals are, by contrast, both discrete-valued and discrete-time signals. This is a fundamental difference and has serious implications.

Electronic signals are impacted by multiple unwanted, but largely unavoidable affects. Let's have a brief look at them. The electrical behavior of the devices depends greatly on temperature (temperature variations on chips are often greater than 100 kelvin); on fabrication tolerances (typical variations are in double-digit percentages); or on mechanical stress in the semiconductor crystal (triggered, for example, by the expansion or contraction of different materials or by the chip package). There are also parasitic electrical effects caused by unwanted interactions between the circuit elements via the substrate or between the IC wires. In addition, there are other sources of noise from within and from outside the system, which cause additional disturbances.

Analog as well as digital subcircuits are impacted by all these effects. However, analog circuits are much more sensitive to such perturbations, as every change in current or voltage due to these effects alters the required signal and thus degrades the signal quality. The analog designer needs to be aware of all these factors and to take them into account by taking appropriate measures when designing the circuit and the layout in order to achieve the required signal quality under all circumstances. This is a formidable challenge when faced with such a large number and variety of noise sources, although the individual analog circuit blocks are normally only comprised of a few dozen devices – the circuit blocks are "small" in other words. The difficulty with analog design does not lie in the size of the circuits, but rather in the mastering of the variety of influences to which they are susceptible.

Digital circuits are, by contrast, much less susceptible to these effects, as they work exclusively with discrete signals. Generally, they are dealing with binary signals with only two different values. (These values are typically the dual digits "0" and "1" or the logical values "true" and "false".) This means the electrical design requires

only two voltage levels. The only prerequisite is that the different levels can be clearly identified. This is achieved by defining a forbidden zone between the levels and by waiting until all signal nodes have returned to one of their permissible logic states after an event before triggering read processes. This is done by setting the clock rate.

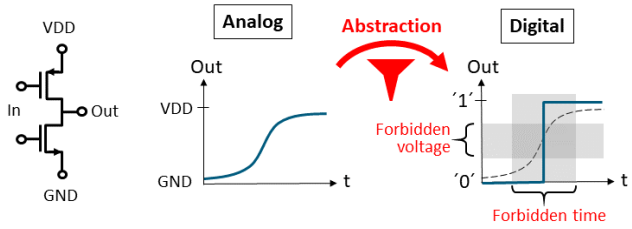

Figure 2: A comparison between analog and digital waveforms

This rigid abstraction is shown in Fig. 2 as an example of a simple series circuit from an NMOS and a PMOS transistor. This circuit can be used in analog technology as an amplifier, a push-pull stage or a half-bridge; in digital electronics it is known as an inverter. Digital circuits can thus be practically immunized against the above noise to assure error-free operation. Another advantage is that because the transistors in digital circuits need only operate as simple On/Off switches, they must not meet as many requirements as (linear) analog circuits.

Analog designers and digital designers are right – but in different ways – when they say their design problems are very complex. Therefore, when talking about complexity, I prefer to distinguish between (1) *quantitative* complexity, as observed in digital designs, referring to the sheer number of design elements (aka "More Moore"), and (2) *qualitative* complexity, as found in analog designs, and rooted in the diversity of the requirements to be considered (aka "More than Moore").

3 Strategies and Techniques in Electronics Design

Two strategies have proved to be very successful when faced with complex design challenges (not only in electronic design): the "divide-and-conquer" principle and the use of computer programs, i.e., automation. Both strategies are applied in many situations.

3.1 Divide and Conquer

The task of designing a microelectronic system consists of converting the required electrical functions (documented in a system specification) to the fabrication data (layout) needed to fabricate the masks. This task is so challenging that it cannot be performed in a single step. It is therefore split into multiple subtasks. The result is a design flow made up of design steps, which are executed serially. A host of different design flows comprising multiple design steps have evolved to suit the specific design task at hand (digital, analog, mixed signal) [7].

Multiple benefits accrue from this task sharing approach: the subtasks are less complex and allow members of the design team to specialize in individual tasks. This type of specialization also promotes computer support, as it simplifies the design of automation procedures. Another advantage is that clearly defined intermediate results, which can be checked for correctness, are obtained with the design steps. Design flaws can thus be identified early and fixed.

Another type of division occurs in the structural domain. The system to be designed is split into smaller parts. The result is circuit blocks, mostly in multiple hierarchy levels, which are easier for humans and for algorithms to deal with. Partitioning in subcircuits delivers great improvement in compute time due to the growth in runtime complexity for algorithms [4]. The great advantage here is that the circuits and layouts for the system parts can be largely designed independently of one another and in parallel. This significantly shortens the design timeline.

3.2 Basic Principles of Design Automation

3.2.1 Synthesis and analysis

There are differences between synthetizing techniques and analyzing techniques in EDA. Synthetizing techniques realize individual or multiple back-to-back design steps, i.e., they produce new design results or intermediate results. The successful application of a synthesis technique helps progress a design flow. One could also say that synthetizing techniques add value.

Analyzing techniques, on the other hand, are deployed to verify the correctness of design results or intermediate results, that is, they are verifying techniques. They enable the decision as to whether a design step was successful or not. In the latter case, the design step must be repeated (either partially or completely). If the step was successful the flow can move on to the next design step. It is crucial that errors in the design process are detected as early as possible, as the costs for clearing a fault increase by about one order of magnitude with each design step.

If we compare analog and digital design in connection with these two techniques, we find the following big difference: very powerful, optimization-based synthesis techniques are available for digital design in contrast to analog design. This is why digital design productivity beats analog design productivity by multiple orders of magnitude. But low analog design productivity does not mean that there is no automation in the analog circuit design. The tools for simulating analog circuits at the SPICE level and the techniques for electrical verification (LVS) and checking design rules (DRC) in layouts are absolutely necessary analysis tools in the analog design flow. Without these powerful quality requirements analog circuit design would long be impossible.

Synthetizing techniques are relevant for our examination.

3.2.2 Synthesis techniques

3.2.2.1 Optimizers. Synthetizing techniques in design automation are based on the principle of optimization. Optimization algorithms, also called "optimizers", are "all-purpose weapons" in EDA. Optimizers search for a solution to a specific problem based on a defined objective function, which describes the quality of a solution. Its eponymous property is that it finds the best solution for which a

good value will be assigned to the defined objective function. It normally examines multiple possible solutions. Depending on the method the number of solution options can be small or very large (e.g., a stochastic search algorithm such as simulated annealing). Some optimizers calculate a guaranteed optimum mathematically (e.g., quadratic placement [5]). We shall discuss characteristic properties of optimizers in Sect. 4.

3.2.2.2 Generators. Procedural techniques are another tool category. Procedural techniques do not perform optimization, but run scripts containing actions predefined by experts in the field. They are also known as generators. A generator produces a predefined design result while an optimizer searches for a new solution for a design problem at runtime. Generators are therefore knowledge based.

3.2.2.3 Neural Networks. In recent times, we have witnessed increased use of machine learning and deep learning techniques for synthesizing tasks in EDA. Neural networks are first trained and then a set of output data is produced from a set of input parameters. For example, there are attempts to obtain the dimensioning parameters for an operational amplifier at the output of a trained neural network by inputting the desired performance parameters of the amplifier to the network. The neural network contains expert design knowledge of the required widths and lengths of the transistors. Many more applications have been described in recent years, which are often used to supplement optimization-based or procedural techniques. These approaches need to mature further before they find their way into industrial application. But this transition will definitely take place.

4 Some Theory Behind Optimization Methods

4.1 The Purpose of Optimizers

Optimization provides a solution to a so-called optimization problem. There are a set of possible solutions for an optimization problem and an objective function that grades the solutions. The optimization aims to find a solution from the set of solutions for which the objective function assumes a value that is as large (or small) as possible.

To solve an electronic design task by optimization, we first need to define the optimization goal. This goal must be quantifiable so that the quality of different solution options can be assessed and compared. Take for example a placement problem where one might want to minimize the chip area A or the sum L of all (estimated) wire lengths (aka total wire length). It is often the case that multiple optimization goals are aimed for simultaneously. These goals are then bundled in a compound objective function $Q_{mod} = f(A, L)$ in this example case. A popular simple approach is to build the objective function from the weighted sum of individual objective goals [5]. For the above example, this would be

$$Q_{mod} = w_A \cdot A + w_L \cdot L \quad (1)$$

The weighting of the two sub-goals is then easily set with the weights w_A und w_L. The index "mod" signifies that the quality refers to a model here. This will be explained in the next section.

4.2 Model Generation and Solution Space

Before an optimization can be run on a computer, a mathematical model of the real design problem hast to be created (Fig. 3 on top). The result of this model generation is the optimization problem, consisting of the objective function Q_{mod} and a data structure. An algorithm is required as well that will process the data structure and execute the optimization. These elements need to be accurately matched in the model.

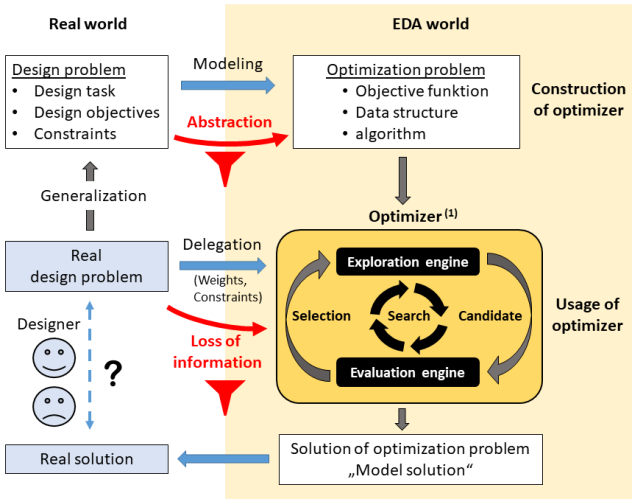

Figure 3: The model generation procedure and the structure and mode of operation of an optimizer. (1) in accordance with [12]

Let us look at the data structure first. It should contain all relevant characteristics of the design problem needed for finding a solution. The data structure should also be constructed in such a way that the objective function can be efficiently calculated and that solution options can be effectively generated.

Relevant properties of a design problem include the parameters that describe the solution. I will call these the solution parameters s_i. In our placement problem this would be the pairs of coordinates (x_j, y_j) we are looking for. This means that j = 1, …, n for n devices to be placed and hence i = 1, …, 2n (two solution parameters per device).

In addition, all information needed to calculate the objective function is required. For our example, this means that information on the electrical nets is needed for the objective function (1) so that the estimated total wire length can be calculated. Other properties may be of relevance for the search for solutions depending on how the optimization method works. This means that these properties would have to be modeled as well. We will come back to this later.

The model produces a virtual solution space S spanned by the solution parameters s_i. Each s_i defines an axis of S. Solution spaces are typically extremely high dimensional. Every point in S represents a potential solution. The solution space in our exemplary placement problem has 2n dimensions. Two dimensions of such a solution space are visualized in Fig. 4.

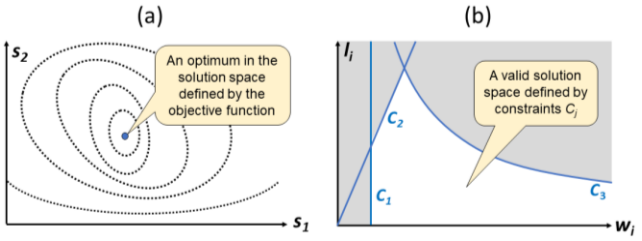

Figure 4: Two-dimensional sections of a solution space. The objective function has constant values along the broken lines (a). Constraints define a valid solution space (b)

In our example, axes s_1 and s_2 could be the possible x- and y-coordinates for a device we wish to place or one coordinate for each of two devices to be placed.

4.3 Structure of an Optimization Algorithm

The optimization algorithm searches the solution space S for the best solution with regard to the quality, that is, the solution with the best Q_{mod} value. This point is also called the global optimum for an optimization problem. One can visualize the values Q_{mod} (s_1, …, s_{2n}) as a "mountain range" across S, where the quality of a particular solution is represented by the height value. Some contour lines for this mountain range are indicated in Fig. 4a. We search for a point in the mountain range that is either as high or as low as possible, depending on the definition of Q_{mod}.

The structure and the mode of operation of an optimizer are shown in the beige box in Fig. 3. It consists of an exploration engine and an evaluation engine. The exploration engine explores the solution space S and selects (or generates) candidate solutions, which are assessed by the evaluation engine in accordance with the objective function. Based on this assessment, the exploration engine decides which solution option should be evaluated next. This procedure is executed iteratively resulting in a directed search for increasingly "better" solutions in S.

Iterations are performed until a termination criterion is reached. This is typically a runtime limit or a minimum solution quality. This is an algorithmically found solution, which I call the "model solution". The devices in our example would be placed at the locations in the layout that are defined by the solution parameters in the model solution. The "real-world solution" is only found with this operation. This transfer operation may appear trivial. We will see later that it is very important to distinguish between the model solution (i.e., the solution in the mathematically modeled solution space) and the real-world solution for a design task (this is the result of a design step).

4.4 Constraints

Typically, specific boundary values have to be adhered to in every design step. These are called constraints. Constraints can arise from the chip fabrication process, from the specified electronic circuit function or from the design methodology [7]. Constraints define whether a point in the solution space S represents a valid solution or not. Constraints often appear as boundary surfaces that divide S into a "valid" and an "invalid" (i.e., forbidden) space. The valid area of S is also called a search space.

We will illustrate these procedures now with a practical example – a routing task. A set of straight interconnect segments needs to be determined for every wiring net in this task. We define the following solution parameters for interconnect segments: coordinate of a segment end, segment length, orientation (angle), segment width and routing layer. A section of the solution space for this type of interconnect segment i with the two solution parameters segment width w_i and segment length l_i is shown in Fig. 4b. Forbidden parts of the space are colored gray. The technological constraint specifying a minimum width is shown as C_1. A maximum IR drop could be requested for the segment in order for the circuit to function properly. This yields a maximum track resistance, which is converted to a maximum permissible value for the ratio l_i/w_i by means of the sheet resistance. This yields the constraint C_2. If one wanted to constrain the parasitic capacitance for operational reasons, the segment area would need to constrained accordingly and the result would be a constraint of the form C_3. Design-methodology constraints are also possible. You could, for example, constrain the solution parameter "orientation" (not shown in Fig. 4b) so that only horizontal, vertical and diagonal segments are allowed.

5 The Limits of Optimization

5.1 Computational Efficiency

The computational efficiency of an optimizer is determined by how well the evaluation engine and the exploration engine cooperate. The following have a direct bearing on this efficiency: (1) for how many points in the solution space can the evaluation engine calculate the objective function per unit of time and (2) how successful the exploration engine is at searching the solution space – in other words, how many trials does it need to approach a good solution.

The first factor is easy to quantify. The second factor is a qualitative aspect of the search algorithm, which can also be estimated by means of tests on benchmark examples. "Good" – or globally optimal model solutions in the best case – can be generated with high computational efficiency.

The contribution of the exploration engine to the computational efficiency is clearly a function of the size of the solution space (i.e., the cardinal number of set S); it is also impacted by the composition of the "quality mountains". The global optimum can definitely be found quickly with a simple gradient technique if there is only one peak in the mountain range, for example. This is also the case if the number of peaks is so small that enough attempts can be made at a successful search for a solution. Solution spaces are usually so complex in practice, however, that such simple approaches using "greedy" algorithms fail for real-world tasks. There are a whole host of smart algorithms in EDA that attempt to get over this problem. Many of them address very specific design problems. There are other general approaches, like simulated annealing, which is compute intensive because it is stochastics based.

The constraints play a very important role here as well. Fig. 4b shows how they bound the search space. This has a beneficial effect on computational efficiency. Design-methodology constraints are a case in point here: they have led to very unique design styles. I would like to mention two of them. The number of possible cell locations is reduced to a simple grid in gate arrays. The placement problem is thus reduced to an assignment problem, whose solution space is smaller by some orders of magnitude than with an unconstrained placement. The designer is constrained to HV routing with uniform wire widths and pin positions at only two channel edges in standard cell design. This is despite the fact that the solution space from the point of view of the available technology options offers much more degrees of freedom for the design and could be larger. Adherence to these constraints has meant, however, that very efficient custom channel routers for this design style could be designed and built.

Now you might think that the more constraints there are the better computational efficiency is. You might be right in many cases – as shown in the examples – but unfortunately things are not that simple. Constraints can have a very negative impact on computational efficiency (i.e., on the time required to find a solution). This will become clear in the following cases and examples.

The mathematical model in the case of design-methodology constraints is designed so that the exploration engine "automatically" remains in the search space. This is not the case with functional constraints. In this case, the exploration engine normally checks in every iteration whether the solution option violates a constraint or not, i.e., whether it is in the valid search space. Take the example in Fig. 4b. The quotient and the product of width and length are calculated for every interconnect wire under investigation to test the given constraints. This requires extra compute time, which can be quite significant and which increases with every additional constraint.

I mentioned in Sect. 4.2 that other properties may need to be included in the model along with the design problem properties explained there. This is always the case when there are constraints that cannot be described as a function of the solution parameters (as in the example we just outlined above). If we take into account the overlaps between devices in the placement problem in Sect. 4.2, we would have to model the surface areas or the exact dimensions of the devices as well. To prevent overlaps, we can then exclude right from the start solution options with overlapping devices from the algorithm so that only valid placement solutions are generated. This would be an additional set of constraints, for which we would need this additional information. Another strategy would be to allow overlaps, but to include them in the objective function so that the optimizer would try to minimize the sum of overlaps. No new constraints would then be created, but the computational overhead of the evaluation engine would be greater. (In the latter case, a solution may then have to be followed up with a so-called "legalization" step to eliminate any residual overlaps).

Here is another example of how additional constraints are generated: if we wish to include the resulting power losses and their effects on the electronic circuit in a layout optimization when designing an SOC, the necessary calculation bases would have to be included in the problem model. Further parameters would be needed to add extra dimensions to the solution space so that the new boundary values can be modeled. This would increase the size of the solution space as well as the computational overhead needed for checking the new constraints.

In addition to these technical issues, there are functional constraints, which are not necessarily known and that need to be defined separately. And normally they cannot be automatically passed to the optimizers. This type of "constraining" work typically has to be done manually and is very time-consuming. This is often the reason why analog designers shy away from using optimizers: it is not worth the work involved.

5.2 Modeling Accuracy and Model Complexity

I would like to point out at this stage that the (modeled) optimization problem is not the same as the (real-world) design problem. The mathematical modeling procedure, whereby a design problem is represented as an optimization problem, is to certain extent always an abstraction of the design problem (see top part of Fig. 3). Therefore, information is always lost when building a model, and thus also when using an optimizer (see bottom part of Fig. 3). The result is that not all aspects of the real-world design problem go into the optimization. A designer could be very disappointed when the model solution is transformed back from the optimization result, even if Q_{mod} has a very high value or if it is the global optimum even.

I would now like to present two examples for that. Let's look at two simple solutions for a placement problem. If we model the nets using spring forces (quadratic placement, force placement [5]), the net structure is represented globally. However, the devices are typically very tightly placed with multiple overlaps. If no other measures are taken (e.g., fixed cells at the periphery of the placement area), all cells will be placed at the same point. And that represents the global optimum! The layouter would not be happy with this result – to say the least. The principle issue with this approach is that devices are only modeled as mass points. If devices are placed with the Mincut technique [5], they are spread very well across the placement area due to partitioning and assignment of the devices to sections of the surface space. The downside here is that Mincut progressively looses the global view of the nets as iterations progress. So we see: "Optimized is not always optimal (from the designer's perspective)".

Naturally, we could get rid of these drawbacks by including the missing aspects in the model of the design problem, in other words, by increasing the modeling accuracy. This is exactly what is happening in EDA research and in the EDA industry. In case of the two very primitive basic algorithms shown above, there is a vast array of extensions and upgrades to them out there. Multiple authors have also profitably combined them and have produced powerful practical placement tools for digital layouts for the industry. Model complexity has increased with these on-going upgrades.

5.3 Real Optimization Quality

I want to introduce the term "real optimization quality" Q_{real}. The intention is that the term Q_{real} is a measure of how "good" a real-world solution produced by an optimizer for a design problem is. How could this be measured? One could examine how well a real-world solution achieves the performance parameters in the system spec (e.g., by post-layout simulation) in order to obtain a value for Q_{real}. However, this could only be done for the final result of a design process. For intermediate results of individual design steps, probably only a subjective appraisal of the designer would be possible. But I'm not interested in such an evaluation, instead I want to show that the quantities "computational efficiency" and "model complexity" described in the prior sections and which I label E and M respectively are closely related.

Experience shows that a high Q_{real} can only be achieved with a very detailed model, that is, with a high M. In other words, Q_{real} increases as M increases. That said, an optimizer must have sufficient E (i.e., it must be able to assess a sufficient number of promising solution options), otherwise a high M would be useless. We can express this relationship as follows:

$$Q_{real} = E \cdot M \tag{2}$$

This formula is not to be understood as a mathematically exact expression, as I have only explained E and M verbally and have also refrained from defining units. It shows the basic relationship between the two quantities: the more complex (and therefore the more precise) a real-world design problem is modeled and the more efficient the search algorithm, the greater Q_{real} will be – i.e., the better the quality of the real-world design result in the eyes of the designer.

5.4 The Optimizer Dilemma: The Optimization Horizon

There is another relationship between E and M. We have seen how the compute requirements of an optimizer grows if we introduce further characteristics of a design problem into the model of the optimization problem. This means that as M increases E is weakened. Put mathematically: E and M are negatively correlated. If we show the EDA optimizers as points on a diagram with the axes M and E, we get a cloud as depicted in Fig. 5.

From this it follows that you always face a dilemma when designing or choosing an optimizer to solve a design problem. If M is low (that is, the level of abstraction is high), there is a good probability of finding a global optimum in the solution space or to find a solution that is nearly optimum (i.e., high Q_{mod}), as E is high (a simple algorithm with a fast evaluation engine and a "benign" solution space). The downside is that despite a high Q_{mod}, a low Q_{real} is expected because of the loss of information due to the high degree of abstraction of the optimizer's model. If, on the other hand, you choose a technique with high M (low abstraction, i.e., an accurate model of the problem), E will be low. Consequently, there is less probability of the optimizer finding a solution with high Q_{mod} (although in this case a high Q_{mod} would mean a good Q_{real} due to the high M). The metric Q_{real} is bounded here as well.

The takeaway is that the real-world optimization quality Q_{real} cannot be increased at will. The quality of the real-world design result that can be achieved by optimization has a definite limit. This calls to mind the "event horizon" beyond which we cannot see (from the general theory of relativity). This inspired me to call this optimization boundary the "optimization horizon" (Fig. 5).

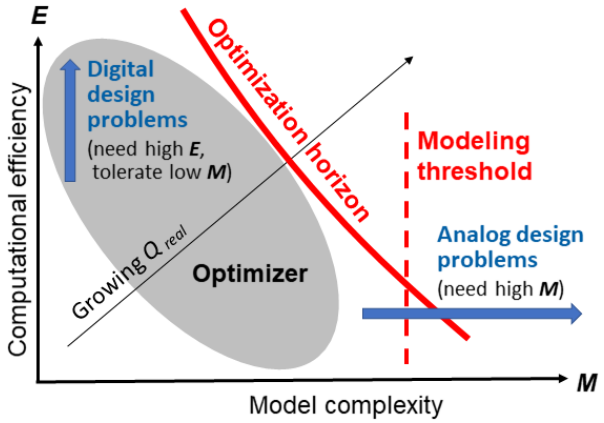

Figure 5: Relationship between the model complexity and computational efficiency of optimizers, and the limits of optimization

The optimization horizon is clearly not a fixed quantity, given that Eq. (2) is not a formal, mathematical equation. This boundary will definitely be pushed back by new IT techniques and new computer generations. It will never disappear though.

5.5 The Productivity Gap Between Analog and Digital Design

The enormous design productivity gap between analog and digital design can be explained in the light of the above observations.

Digital design, as explained, benefits from the restriction to discrete-valued and discrete-time signals. This rigid standardization means a massive reduction of the degrees of freedom in electronic design and simplifies the mathematical modelling. The key point here is that this abstraction does not take place during modeling, but *before* that in the real world. The real-world design problem is already abstracted! (see Fig. 2 on the right). The real-world design problem is so massively simplified that all relevant properties can be successfully included in an optimization model. Another way of putting it: a low M is tolerable. The parameter E can thus be high. This is precisely what is needed to be able to handle the extreme *quantitative* complexity in digital design.

This brings us to the fundamental difference to analog design. The high *qualitative* complexity practically prohibits the design problem from being "simplified" with standardizations. By contrast here, all disturbances need to be *included* in the design process to effectively suppress their effects with special measures in the circuit design (especially with the use of differential circuits) and in the physical design (especially by matching [8]). The aspiration of the analog designers is to proactively use all degrees of freedom in order to come as close as possible to this objective. Their expertise lies in

their practical experience and is essentially knowledge of the specific design problem constraints. This specialist knowledge is implicitly brought to bear in manual design. The key advantage here is that as there is no need to formalize the work, and there is no risk of losing information through modeling.

This points us to the root cause of the design productivity gap: the strength of optimization methods, which arises in digital design from the abstraction procedure, becomes a weakness in analog design. This explains, in my opinion, why optimization techniques are ill-suited for analog design – in fact, I would almost go so far as to say that they resist a (good enough) solution by optimization. To put it another way: practical analog design problems require a Q_{real} that is beyond the optimization horizon.

Optimization-based automation is anathema to many practical analog designers. There are good reasons for this. The only exception worth mentioning is the use of optimization for design centering. The compute power involved is huge, as the evaluation engine must utilize simulations ("simulation in the loop"). Although these tools can also be used for circuit design to determine "optimal" nominal values, they are rarely used for this purpose. Most analog designers prefer to draw on their intuition and experience instead.

Other experts for analog EDA are of the opinion that optimizers theoretically are able to synthesize analog circuits. But they feel the necessary modeling is practically impossible [4]. This could be called the "modeling threshold", which I have drawn in Fig. 5 as well. In the end its effect is the same as for the optimization horizon.

6 Conclusion and Alternative Approaches

This treatise should not give the impression that optimizers are absolutely unsuited for analog design problems. There are certainly individual cases where it works, but these are too rare for industrial acceptance There will be also a shift in the optimization horizon as a result of advances in computing technology. However, the challenges in analog design will also continue to increase in semiconductor processes going forward. I'm afraid optimization will always pull the short straw in this race. Furthermore, I'm convinced that we will never see the day when the analog parts of chips are designed solely by means of an optimization-based design flow with a degree of automation that is comparable to a digital flow.

An automated analog flow can be achieved by taking a look at further automation techniques. Procedural techniques are particularly promising here. They have received increased attention in recent years as well. One such approach is BAG [2] and further developments of it [1], which is a hot research topic in multiple European EDA-research laboratories at the moment. Template-based layout-generators are presented in [11]. We ourselves have developed the EDP tool (Expert Design Plan), a procedural approach for dimensioning analog circuits that has already found its way into industrial application [14].

Combining procedural and optimization techniques is a formidable challenge, as they work with contrarian automation paradigms. A very good analysis on this subject is to be found in [9]. This work also introduces an attempt at a solution to the problem called SWARM, where smart layout generators cooperate as agents, resulting in a self-organizing layout (refer to [10] for a compact description). In [13] and [7], I outline how the different automation paradigms could be merged in a top-down-meets-bottom-up design flow.

Basically, I agree with those authors, who say that a smart combination of multiple automation approaches is the best way to go. MAGICAL [15] and ALIGN [3] are impressive recent examples of such approaches. They both offer a fully automated analog physical design, incorporating human expert knowledge, as well as analytical, heuristic and machine learning techniques.

REFERENCES

[1] Eric Chang, Jaeduk Han, Woorham Bae, Zhongkai Wang, Nathan Narevsky, Borivoje Nikolić, Elad Alon. 2018. BAG2: A Process-Portable Framework for Generator-Based AMS Circuit Design. In *Proc. of IEEE Custom Integrated Circuits Conference (CICC)*, April 8-11, 2018, San Diego, CA, USA. https://doi.org/10.1109/CICC.2018.8357061.

[2] J. Crossley, A. Puggelli, H.-P. Le, B. Yang, R. Nancollas, K. Jung, L. Kong, N. Narevsky, Y. Lu, N. Sutardja, E. J. An, A. L. Sangiovanni-Vincentelli, and E. Alon. 2013. BAG: A designer-oriented integrated framework for the development of AMS circuit generators. In *Proc. of 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, April 18-21, 2013, San Jose, CA, USA, 74–81. https://doi.org/10.1109/ICCAD.2013.6691100.

[3] Tonmoy Dhar, Kishor Kunal, Yaguang Li, Meghna Madhusudan, Jitesh Poojary, Arvind K. Sharma, Wenbin Xu, Steven M. Burns, Ramesh Harjani, Jiang Hu, Desmond A. Kirkpatrick, Parijat Mukherjee, Soner Yaldiz, Sachin S. Sapatnekar. 2021. ALIGN: A System for Automating Analog Layout. In *IEEE Design & Test*, Vol. 38 (2), April 2021, 8-18. https://doi.org/10.1109/MDAT.2020.3042177.

[4] Helmut E. Graeb, personal communication.

[5] Andrew B. Kahng, Jens Lienig, Igor L. Markov, Jin Hu. 2011. *VLSI Physical Design: From Graph Partitioning to Timing Closure.* (1st. ed.), Springer, Dordrecht. https://doi.org/10.1007/978-90-481-9591-6.

[6] Jens Lienig and Juergen Scheible. 2020. *Fundamentals of Layout Design for Electronic Circuits.* Springer, Cham, 1-29. https://doi.org/10.1007/978-3-030-39284-0.

[7] Jens Lienig and Juergen Scheible. 2020. Methodologies for Physical Design: Models, Styles, Tasks, and Flows. In *Fundamentals of Layout Design for Electronic Circuits.* Springer, Cham, 127–164. https://doi.org/10.1007/978-3-030-39284-0_4.

[8] Jens Lienig and Juergen Scheible. 2020. Special Layout Techniques for Analog IC Design. In *Fundamentals of Layout Design for Electronic Circuits.* Springer, Cham, 213–255. https://doi.org/10.1007/978-3-030-39284-0_6.

[9] Daniel Marolt. 2018. *Layout Automation in Analog IC Design with Formalized and Nonformalized Expert Knowledge.* Ph.D. Dissertation, University of Stuttgart, Germany. http://dx.doi.org/10.18419/opus-10231.

[10] Daniel Marolt, Juergen Scheible, Goeran Jerke, Vinko Marolt. 2016. SWARM: A Self-organization Approach for Layout Automation in Analog IC Design. In *Int. Journal of Electronics and Electrical Engineering (IJEEE)*, 2016, Vol. 4 (5), 374-385. https://doi.org/10.18178/ijeee.4.5.374-385.

[11] Benjamin Prautsch, Uwe Hatnik, Uwe Eichler, Jens Lienig. 2021. Template-Driven Analog Layout Generators for Improved Technology Independence. In *Proc. of 16th GMM/ITG-Symposium ANALOG 2018.* Sep. 13-14, 2018. Munich, Germany, 1–6. https://ieeexplore.ieee.org/document/8576850.

[12] Rob A. Rutenbar. 2006. Design Automation for Analog: The Next Generation of Tool Challenges. 2006. *1st IBM Academy Conference on Analog Design, Technology, Modeling and Tools.* IBM T.J. Watson Research Labs, Sep. 2006, http://users.ece.cmu.edu/~rutenbar/pdf/rutenbar-iccad06tut.pdf.

[13] Juergen Scheible, Jens Lienig. 2015. Automation of Analog IC Layout – Challenges and Solutions. In *Proc. of Int. Symp. on Physical Design (ISPD'15)*, Mar 29 – Apr 1, 2015, Monterey, CA, USA, 33-40, http://dx.doi.org/10.1145/2717764.2717781.

[14] Matthias Schweikardt, Yannick Uhlmann, Florian Leber, Juergen Scheible, Husni Habal. 2019. A Generic Procedural Generator for Sizing of Analog Integrated Circuits. In *Proc. of the 15th Conf. on Ph.D. Research in Microelectronics and Electronics (PRIME 2019)*, July 15-18, 2019, Lausanne, Switzerland, 17-20. http://dx.doi.org/10.1109/PRIME.2019.8787743.

[15] Biying Xu, Keren Zhu, Mingjie Liu, Yibo Lin, Shaolan Li, Xiyuan Tang, Nan Sun, David Z. Pan. 2019. MAGICAL: Toward Fully Automated Analog IC Layout Leveraging Human and Machine Intelligence: Invited Paper. In *Proc. of 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 4-7, 2019, Westminster, CO, USA, 1-8. https://doi.org/10.1109/ICCAD45719.2019.8942060.