MIXED EQUATION-SIMULATION CIRCUIT OPTIMIZATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Metha Jeeradit

November 2011

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Mark Horowitz, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Stephen Boyd

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Boris Murmann

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost Graduate Education

Abstract

One of the main challenges in designing analog circuits today is to cope with the complex behaviors of modern devices. For example, the drain current of a deep-submicron MOSFET device can no longer be predicted by the long-channel models, due to various effects including carrier velocity saturation, vertical-field induced mobility degradation, drain-induced barrier lowering (DIBL), and narrow channel effects, to name only a few. In addition, the characteristics of each individual device can be further affected by random variability and proximity effects. As a result, it is extremely difficult to design an optimal circuit at the presence of these complex phenomena without relying on computational aids, e.g. circuit optimizers. However, many circuit optimizers based on global optimization techniques such as simulated annealing or evolutionary algorithms may also have difficulties unless they are provided with good initial guesses.

This research focuses on building a circuit optimizer that leverages designer's intent to help the simulation-based optimizers find solutions quickly. The technique is inspired by continuation techniques in numerical analysis where a difficult problem is solved by constructing an easier problem first and gradually refining its solution to that of the hard problem. In a circuit optimization context, the designer's simplified equations for the circuit serve as the easier problem. These simplified design equations are easy to write as they need not be completely accurate and have solutions that are easy to reason about.

While these design equations may not generate a good initial design for optimizing real circuits directly, they can help guide the optimizer towards an optimal solution by constructing and solving a series of intermediate problems that gradually change from the initial problem based on design equations to the real circuit problem with accurate device models. Assuming that the solutions to these intermediate problems change only gradually along this path, each problem can be solved very efficiently via local optimization methods, using the solution to the previous problem as a starting point. Although more problems have to be solved, since each is solved very efficiently, the overall computational costs can be lower than that of solving the final problem directly. We apply our circuit optimization approach to several design examples and investigate the importance of the initial designer's equations by comparing our optimizer's performances against those of a local optimizer and a generic continuation-based optimizer.

We extend this basic idea to leverage hierarchy to help optimize large circuits, and demonstrate this technique with a PLL design example. Besides the runtime benefit, performing optimization hierarchically is also intuitive and more in line with how a circuit designer would approach designing a large circuit problem like the PLL.

Acknowledgement

For the longest time, this just didn't seem like it was going to happen. And were it not for the supports, advices, and encouragements that I have received from various people over the years, this probably wouldn't have happened. I would like to thank all these great people who have made my Ph.D. at Stanford possible.

First and foremost, I would like to thank my adviser, Prof. Mark Horowitz, for being a great mentor and for giving me the freedom to work on what I wanted to. Mark has always been very helpful, kind and generous, and I always feel that I can go and talk to him on just about anything. He also has this uncanny ability to understand my problems quickly — even when I sometimes do not understand what problems I am having myself — and would usually be able to suggest solutions to their problems that have helped saved me a lot of sleepless nights. I feel incredibly lucky to have Mark as my adviser.

I would like to also thank my reading and Oral committee members. I would like to thank Prof. Boris Murmann for his expertise and insights on analog circuits design, and for always keeping his door opened so that I can randomly drop by unannounced to get his advice. I would like to thank Prof. Stephen Boyd for his expertise and insights into various optimization techniques. I would also like to thank Prof. Ada Poon and Prof. Philip Wong, for agreeing to be on my Oral committee on a fairly short notice. I want to thank my former adviser Prof. Thomas Lee who has been incredibly patient with me while I was exploring various options during my time as a part-time student.

I want to also thank Prof. Jaeha Kim, who has also been of a tremendous help in just about everything — from providing expertise on analog circuit design to helping me grow as a person. His support and guidance to me have been beyond amazing even since the time when he was my boss and mentor at Rambus to the time that he was my mentor at Stanford. I feel incredibly fortunate to have known and to have worked with such a great person. I owe him my deepest gratitude.

I want to also thank everyone in Mark's group, both past and present — James Mao, Byongchan Lim, Sabrina Liaos, Frances Lau, Megan Wachs, Pete Stevenson, Stephen Richardson, Sameh Galal, Ofer Shacham, Zain Asgar, Limor Bursztyn, Andrew Danowitz, Kyle Kelley, Krishna Malladi, Kahye Song, Gordon Wan, Jim Weaver, Valentin Abramzon, Bita Nezamfar, Donguk Moon, Mehmet Ozan, Hwang Ho Choi, Tao Yu, Suyao Ji, and many others — for their friendship, fun group trips, and stimulating discussions. I want to also thank Mark's former and current wonderful administrative assistants, Teresa Lynn and Mary Jane Swenson, for always making sure that things are run smoothly.

My experience at Rambus also played a big part in shaping my research on analog circuit optimization. I would like to thank my former bosses — Carl Werner, Jared Zerbe, Nhat Nguyen, Bruno Garlepp, and Kevin Jones — and their group members, Yohan Frans, Fred Chen, Andrew Ho, Jason Wei, Reza Navid, Brian Leibowitz, Haechang Lee, Amita Agarwal, Simon Li, William Stonecypher, Qi Lin, Aida Varzaghani, Amir Amirkhany, Jafar Savoj, Aliazam Abbasfar, and many others, for the fun experiences on designing chips as part of a big team.

I want to also thank Rambus for sponsoring the early part of my Ph.D. program when I was a part time student and Texas Instruments, especially Martin Izzard and Tom Vrostos, for sponsoring this project in the latter part of my Ph.D. program through its completion.

And last but certainly not least, I would like to also thank my family — my Dad, my Mom, my brother Jackerin (Jack), and my fiancée and my best friend Wirulda

Pootakham (Nik). Dad, Mom, and Jack, thanks for your unconditional love and support. I couldn't have done this without you.

Nik, thanks for everything. Thanks for knowing when to push me and when to support me. Thanks for always finding and searching for fun things to do together. Without you, my Stanford experience just wouldn't have been the same.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The main tasks facing a circuit designer during a typical circuit design process are 1) selecting an appropriate circuit topology that the designer deems most suitable for their target application, 2) sizing the underlying circuits in the chosen topology and verifying that its performance meet the application' design specifications, and 3) realizing the layout of the sized schematic and re-verifying its performance.

To help designers quickly evaluate different circuit topology options, they need a fast and intuitive circuit sizing methodology. The traditional approach for accomplishing this is to first form simple analytical equations to estimate critical performance parameters based on the designer's understanding of how the circuit should behave. The circuit is then simulated based on the device sizes obtained from optimizing these simplified equations. If the resulting circuit performance does not meet the design specifications, the designer re-adjusts their equations to come up with new sizes before simulating again. This cycle is then repeated until either the design specifications are met or the designer becomes convinced that this topology cannot be made to work in the target application.

While traditional approach to circuit sizing has worked well in the past, growing device complexity as you scale down technology has made it very time consuming. If the discrepancies between the device models the designers used to form their simplified equations and the actual device models used in simulation are sufficiently large, the iterative procedure between hand calculations (forming and solving simplified analytical equations) and simulations can become a blind search.

Even if the circuit designer chooses to reuse a circuit topology from another technology node where its design has been proven to work, its circuit sizing procedure can also be as time consuming. There are two main reasons for this. First, because the process non-idealities and limitations vary from process to process, the topology that may be suitable in one process technology may not be suitable in another. For example, a cascoded amplifier may work well in a 0.18μm process with a 3V supply but may fail in a 45nm process with a 1V supply due to headroom limitations. Second, the design may have worked in a previous process because it has been sized for a different performance objective than what is being targeted for in the current application. Perhaps the original design was sized for low power while the new application may have high performance as its main objective. Clearly, we need to improve both the way circuit sizing is done and the way the designer's knowledge is transferred from one technology process to another.

A natural step to improving the circuit sizing methodology is to automate it. Automation tries to accomplish two goals: speed and convenience. The approaches taken by existing circuit optimizers are usually either fast or convenient but not both. For them to be fast, the optimizers usually need a large amount of input from the designers, usually in the form of very accurate design equations, making them difficult to create. On the other hand, the optimizers that are easy to use will usually need to treat the optimization problem as a black box since they require very little input from the designers and thus have very little information about the underlying circuit, making the optimization process very slow. Chapter 2 will examine the advantages and disadvantages of these existing circuit optimizers in more detail.

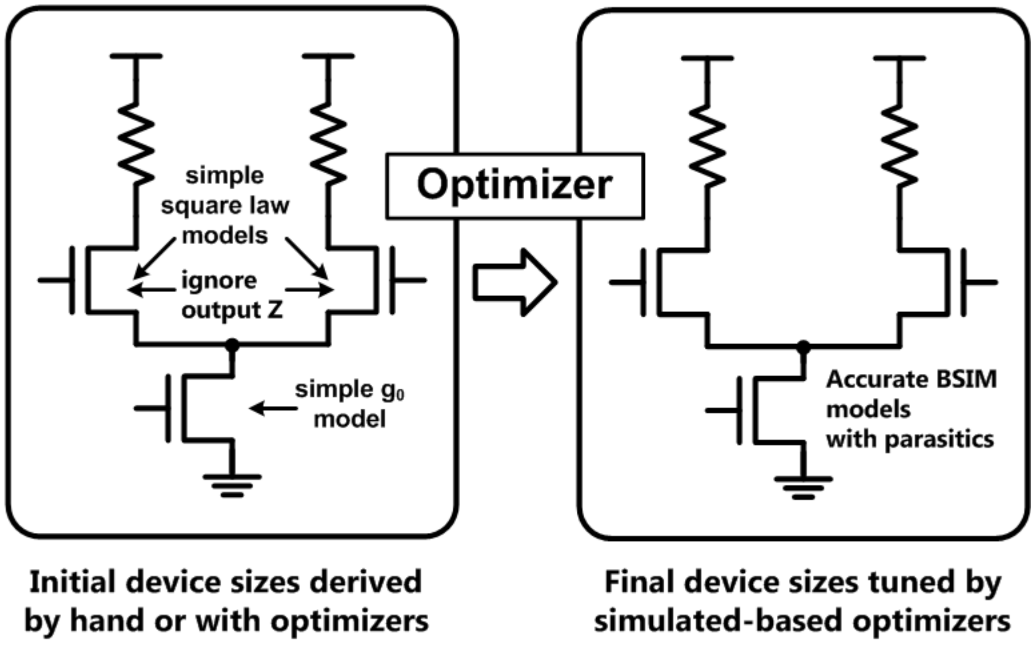

simple
square law
models
ignore
output Z
simple g_0
model
Optimizer
Accurate BSIM
models
with parasitics
Initial device sizes derived
by hand or with optimizers
Final device sizes tuned by
simulated-based optimizers

Figure 1.1: High-level view of our optimization framework. Designers design their circuits using their simplified equations and let optimizers take care of the non-idealities by moving the initial design point in the equation space to the final design point in simulation space.

The main focus of this work is on creating an improved circuit sizing methodology through a circuit optimization framework that provides a good compromise between speed and convenience. It also tries to retain the intuitive analysis of the traditional circuit sizing approach while coping with the differences between these models and the real transistor's behaviors.

Our framework in Figure 1.1 accomplishes these goals by first letting the designers size their circuits in the traditional way and then relying on the optimizer to handle the device process non-idealities. In our methodology, the designers first use simplified equations that best represent their design intent to initially size their circuits. The optimizer then leverages these simplified equations to help it move the initial design point in the equation space to the final design point in the simulation space using a numerical continuation technique called homotopy.

Conceptually, we can view our approach as slowly introducing non-idealities into the designer's assumed device models and iteratively refining our solution as we move towards the simulation space. The main assumption here is that the discrepancies between the two models are second order, and without these non-idealities, the circuit would have worked exactly the way the designers had intended. Chapter 3 will describe our approach in more details before going through a few design examples in Chapter 4. Chapter 5 will then analyse the performance of our optimizer.

Because the approach described in Chapter 3 can still be quite slow for circuits with a large number of parameters or with a long simulation time, Chapter 6 will investigate how we can extend our methodology to size these circuits through hierarchical optimization. The extended approach is demonstrated through a PLL optimization example. Chapter 7 will then summarize and conclude our work and discuss possible future directions.

Chapter 2

What Makes a Good Circuit Optimizer?

As discussed in the introduction, a good circuit optimizer should be:

1. Intuitive to circuit designers, to give them confidence and understanding into why the resulting device sizes have been chosen,
2. Fast enough to allow the circuit designers to explore different circuit topologies, and
3. Easy to use such that the overhead in setting up new optimization runs is small.

This chapter examines prior and existing circuit optimization approaches against these criteria to help us understand their strengths and weaknesses. A more detailed survey of existing circuit optimizers can be found in [14].

Previous and current circuit optimizers largely take one of two approaches [14]: equation-based or simulation-based approach. In an equation-based approach, shown in Figure 2.1, the focus is on having a very fast optimization process by first creating very accurate design equations and then performing the optimization in the equation space. The main drawback of this approach is that the resulting equations can become complex and difficult to derive. Thus, they can provide little insight, losing

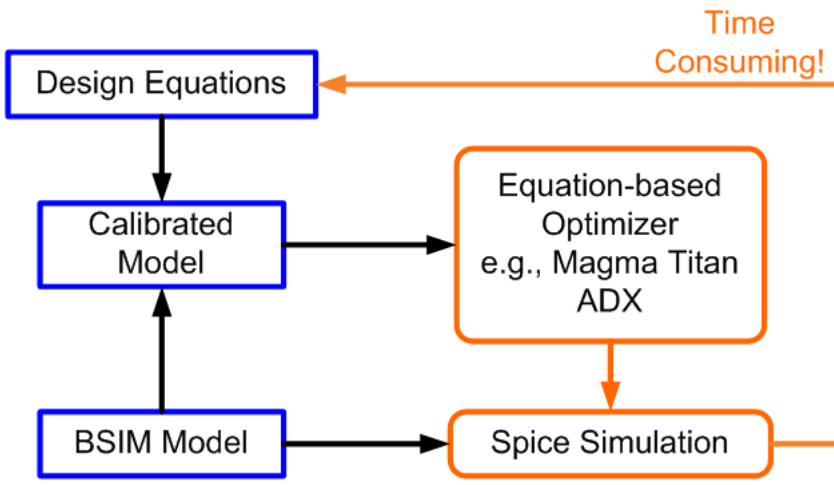


Figure 2.1: Equation-based Circuit Optimization Approach.

some of their merits as analytical formulas. In simulation-based approach shown in Figure 2.2, the focus is more on making the tool convenient for users by leveraging generic optimization algorithms. The user only has to provide the simulation decks and optimization objectives to the optimizer and the tool will perform the optimization using non-linear algorithms such as simulated-annealing, stochastic pattern searches or evolutionary algorithms. The drawback here is that these methods are very resource-intensive.

In the 1980s when computational resources were scarce, early approaches to device sizing automation [8, 18] were equation-based. These approaches were not only fast, which is to be expected of equation-based approaches, they were also very accurate because the effects of the non-idealities in the old process technologies were not as pronounced as the ones in current process technologies making it relatively easier then to create accurate design equations. They were also intuitive because the sizes were derived using easy-to-understand design scripts. However, these early approaches took the view that only experienced designers should design circuits so the tasks for writing the equations for the design scripts were reserved only for these experienced designers, making the optimizers very easy to use for regular users.

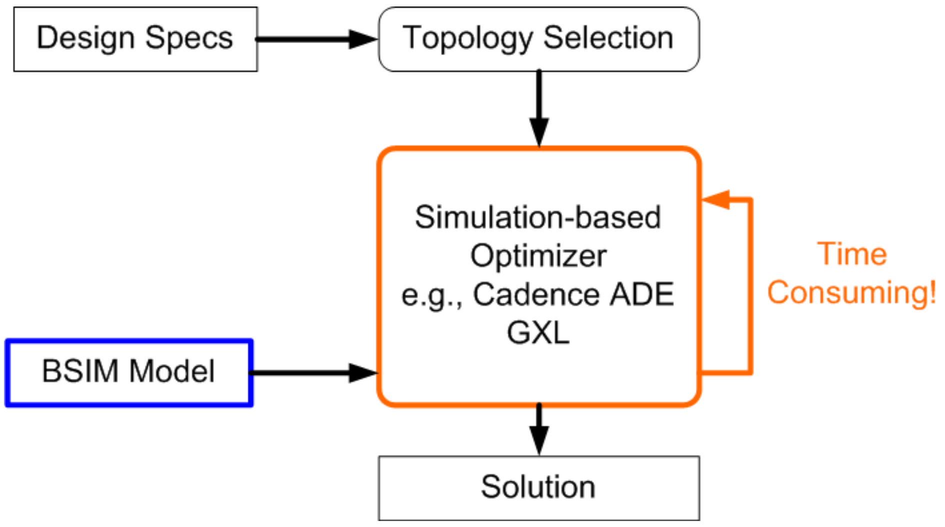


Figure 2.2: Simulation-based Circuit Optimization Approach.

Hence, these 'optimizers' (more appropriately called synthesis tools) were packaged as knowledge-based optimizers where the design equations (also known as formal descriptions) for each circuit derived by experienced designers were stored in a library along with its design plan or design scripts for determining its device sizes for a given set of design specifications. Furthermore, if an optimization run fails, these optimizers will attempt to tell you why it failed and will try to give hints as to how you may be able to adjust the specifications to make the design feasible. The goal was to make analog circuit design easily accessible to digital and junior analog circuit designers.

While these early tools seemed to have all the makings of a good circuit optimizer, they were not widely adopted as the experienced designers found it difficult and time-consuming to contribute new designs to the library. One of the main reason for this is because "analog design heuristics are very difficult to formalize in a general and context-independent way" [14], hence each new circuit topology required its own custom design plan with little reuse from other design plans and the effort required for creating a design script was found to be roughly four times more than what is needed to actually design the circuit once [27]. Furthermore, designers also needed to have some programming knowledge in order to incorporate their design plans into the library making the process of adding new circuits to the library too complex even for experienced circuit designers to use. While these tools did have some commercial success (MIDAS [27], AZTECA [26], CATALYST [52], BLADES [9], ISAID [51]), most of the circuits used by the customers in these tools were the ones that were already provided by the tool developers [14].

2.1 Equation-based Optimizers

To make the tools more practical, more recent equation-based optimizers let users directly input the design equations that should be reasonably accurate in order to obtain solutions that are close to the actual optimums in simulation. Since the optimization engine in these tools does not rely on any simulation, having accurate equations is essential to the accuracy of the final optimization results. This requires including many sophisticated effects of the devices making it difficult to write the equations. This process can be eased either by using template languages [21] or by automatically generating equations via pilot simulations [16]. Nevertheless, the resulting equations can become too complex to provide any useful insights, losing their merits as analytical formulas. Furthermore, many optimizers in this category [6, 7, 13, 16, 19, 20, 21, 22, 23, 24, 25, 31, 50] make the optimization process efficient by forcing the equations to be convex or quasi-convex. However, because not all design characteristics are easily captured in convex equation forms, creating accurate design equations can be difficult. This is especially the case when the design objectives involve optimizing large-signal or transient characteristics. Because of the manual effort involved in creating accurate design equations, switching designs to a different circuit topology, or even to the same design in different process technology, can also incur significant overhead as new equations are needed specific to the new topology or to a different set of non-idealities in the new process technology. As with knowledge-based optimizers, the main problem with these equation-based optimizers is the ability to derive accurate design equations quickly.

To help overcome this problem, attempts have been made to automate generating design equations, largely via symbolic analysis tools such as SYMBA [53] and RAINER [59], with limited success. These tools faced scalability problems, generated large uninterpretable expressions, and were limited to only linearized circuits in frequency domains. Because the number of terms in exact symbolic solutions scale exponentially with circuit sizes, heuristic simplification and pruning algorithms were required to make the symbolic analysis more manageable. Initial approaches such as ISAAC [15], SYNAP [47], and ASAP [10], mimicked what designers would do during hand-calculations by only retaining important circuit elements and generating expressions only for the dominant contributions. However, they were still only applicable to circuits with less than 15 transistors making them impractical. Later approaches like SYMBA and RAINER, use more sophisticated simplification algorithms which made them applicable to larger circuits. Nevertheless, the expressions generated by these tools were still large and unwieldy, restricting their usefulness to designers.

More recent researches on symbolic analysis and generation try to tackle both the accuracy and scalability problems concurrently. For example, Xu et al. in [57] demonstrated a hierarchical symbolic analysis technique that can analyze a 44-transistor circuit exactly. Other researches in this area explore different applications of symbolic analysis such as providing sensitivity analysis and visualization [37, 48], or implementing a symbolic circuit simulator [5].

2.2 Simulation-based Optimizers

Instead of focusing on finding faster ways to generate accurate design equations, simulation-based optimizers remove the need for the equations altogether and directly optimize the circuit based on numerical simulation results [16, 28, 32, 36, 38, 39, 45, 58].

Since there is no special structure (e.g. convexity) to exploit, most optimizers in this category rely on global nonlinear optimization techniques such as simulated annealing [16], stochastic pattern search [45], geostatistics technique [58], evolutionary algorithms [36] or a combination of algorithms [38]. All these methods are either slow or resource-intensive. For instance, Phelps, et al. [45], reported that optimizing a two-stage operational amplifier with 32 devices may take more than 10^4 simulations using simulated annealing.

These approaches usually require little work from a designer and initially appear "easy" to use. There is also very little work needed to switch to a different design topology. However, these methods are often slow, as many simulations are run, and their convergence may still be dependent on a good initial point, so the overall user experience is generally not good.

2.3 Mixed Equation-Simulation Optimizers

A good compromise, it seems, lies somewhere between the two existing approaches. An early attempt at this mixed equation-simulation approach was the ASTRX/OBLX tool [42]. In place of simulation, ASTRX/OBLX uses Asymptotic Waveform Evaluation (AWE), a model reduction technique, to evaluate circuit objectives and performances. Because AWE is a black-box modeling technique — mainly applicable to relatively linear circuits — it was not much of an improvement on equation-based optimizers in terms of providing design insights. Moreover, the underlying optimization engine still relied on simulated annealing algorithm and, as a result, also did not provide much improvement on speed over simulation-based optimizers.

Based on the characteristics of the existing circuit optimizers, we contend that:

1. For circuit optimizers to be fast, they need to leverage designer's knowledge (through design equations) to help guide the optimizer to the final solution.

2. For circuit optimizers to be intuitive, these design equations should not be too complex and mainly contain the dominating contributions of relevant circuit elements.

3. For circuit optimizers to be easy to use, the optimizers should do most of the work.

Our proposed circuit optimizer takes up the mixed equation-simulation approach with these goals in mind which will be described in more details in the next chapter.

Chapter 3

Designer-Centric Circuit Optimization

As discussed in the previous chapter, we would like our circuit optimizer to be intuitive and fast like equation-based optimizers but require little equation setup like simulation-based optimizers. Designers should be able to design their circuits using simple and intuitive equations that allow them to reason and understand how their circuits behave with respect to their design variables while the underlying optimization engine should be responsible for taking care of the non-idealities that are not present in the design equations by making sure that the final design point meets the design specifications in the target process technologies.

Suppose you would like to estimate the output voltage of the differential amplifier shown in Figure 3.1. As a circuit designer, you would not use the exact equations from the device manual to estimate the current of the input devices as they can be incredibly complex and hard to understand. Instead, supposed you knew that the input voltages will be small — and hence so will the gate overdrives of the input devices — then you would prefer to use a simple long-channel transistor model to help you estimate the saturation current more easily. And because the circuit uses resistive loading, you would also ignore the output resistances of the input pair in the long-channel model. Non-idealities that were not modeled in the current design

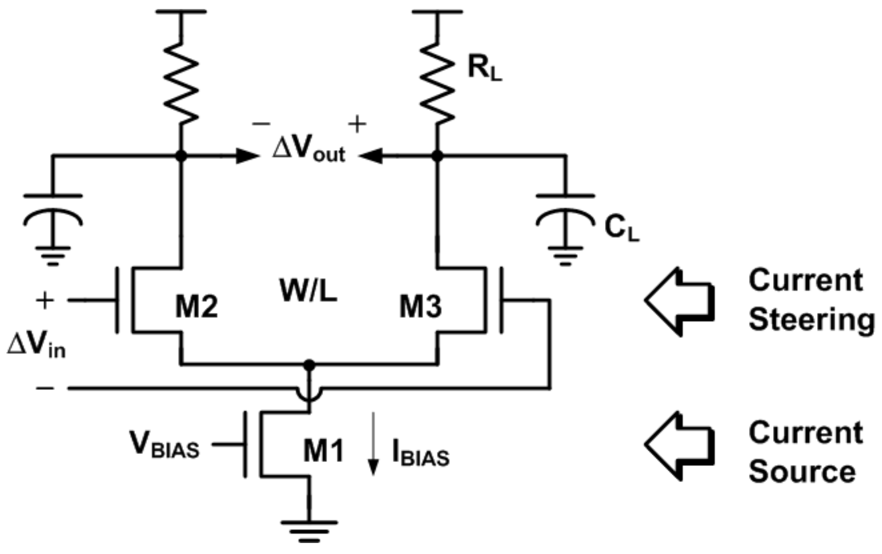


Figure 3.1: Designer's intent in a differential amplifier.

equations such as channel-length-modulation (CLM), drain induced barrier lowering (DIBL), and substrate-current-induced-body-effect (SCBE) effects, would then ideally be taken care of by the optimizer during its optimization process.

The circuit design process can be intuitive when designers are able to use assumptions about their circuit behaviors to keep the design equations simple and representative of their design intent. Designers pick a circuit architecture that they think is best suited to the needs of their target application and each device in the circuit, in turn, fulfills a particular purpose to help satisfy those needs.

Suppose your target application led you to choose the differential amplifier architecture in Figure 3.1 where you would then need to maximize its gain-bandwidth product. You can use your knowledge of how each device in the amplifier is intended to work to help you derive simplified gain and bandwidth equations. More specifically, as the role of the tail transistor is to be a current source and the input devices are to be transconductors, we can use a simple ideal current source (I_{BIAS}) to model the tail transistor and a simple long-channel model to model the input pair. We can then derive the following gain and bandwidth equations in terms of the input device's width (W) and length (L):

$$Gain = G_m \cdot R_L = \sqrt{2 \cdot \mu C_{ox} \cdot \frac{W}{L} \cdot I_{BIAS}} \cdot R_L \tag{3.1}$$

$$Bandwidth = \frac{1}{2\pi \cdot R_L \cdot (C_L + C_j \cdot W)} \tag{3.2}$$

where μ is the carrier mobility, C_{ox} is the gate oxide capacitance per area, C_j is the drain junction capacitance per unit width, and R_L and C_L are the load resistance and capacitance, respectively.

Because the simplified design equations naturally exclude many non-idealities, the actual behavior of the circuit in simulation will not behave exactly the way the simplified design equations predict. Figure 3.2 shows the gain and bandwidth of the differential amplifier as predicted by the design equations and by the simulation (using BSIM4 device model) for various input device sizes. Although the design equations and the simulation data exhibit similar gain and bandwidth trends, there are significant differences in the actual values between the two models. Consequently, the design point obtained from optimizing the simplified equations may be far from the actual optimum and may not even meet the design specifications in the simulation space. Hence, we will need the optimizer to help us move the initial design point obtained from optimizing our ideal design space to the final design point that meets the design specifications in the simulation space.

For the rest of this chapter, we will first provide a brief background on the homotopy methods and then describe our circuit optimization algorithm and its issues.

3.1 Homotopy Methods

Homotopy is a numerical continuation technique that helps solve hard non-linear problem by first solving an easier problem and then gradually converting the easier

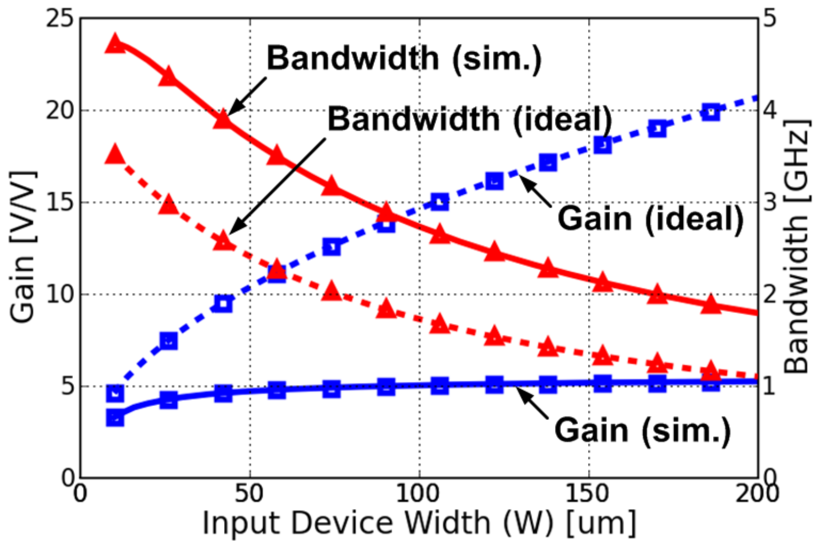


Figure 3.2: Gain and bandwidth of the differential amplifier as predicted by the designer's model and by the BSIM4 simulation model.

problem to the harder problem through a series of intermediate problems. The idea is that if the two successive intermediate problems are close enough, they should also have close solutions and we can therefore leverage the known solution of the preceding intermediate problem to help solve the next intermediate problem. An example use of this homotopy technique in the circuit simulation field is in improving the convergence of DC analysis [40] where the trivial DC solution of a circuit with the supply voltage VDD equal to zero is solved first. The DC solution is then iteratively refined while the supply voltage VDD is gradually ramped up to its desired value. A good analogy of the homotopy technique is the transient simulation. To determine a node voltage at a certain time point, you first start the transient simulation at time $t = 0$ and then progresses the simulation until t reaches the desired time point while the simulator iteratively refines the node voltage value along the way.

In general, a homotopy path between the easy and the hard problems is controlled by a mapping parameter. The supply voltage VDD is the mapping parameter in the DC convergence analysis example while the time t is the mapping parameter in the transient simulation analogy.

Traditionally, homotopy methods have been used to solve systems of non-linear equations and were capable of solving systems with over 100 variables since the 1980s [54]. They were also capable of solving optimization problems by first converting them to systems of non-linear equations and then applying homotopy methods on them. A general algorithm for applying a homotopy method to an optimization problem typically consists of:

1. Convert the optimization problem to a system of n nonlinear equations with n unknowns, i.e., $F(x) = 0$ where $x \in \Re^n$ and $F : \Re^n \to \Re^n$.
2. Construct a homotopy map $H(x, \lambda) = 0$ that satisfies $H(x, 0) = E(x)$ (a known easy system with known solution) and $H(x, 1) = F(x)$ (the hard system from step 1 above) where λ is the homotopy mapping parameter.
3. Track the zero curve (i.e., refine the solution of $H(x, \lambda) = 0$) while varying λ from 0 (the solution of the easy system) to 1 (the solution of the hard system).

The third step — zero curve tracking — is an established process with a mature software package available [56] while the second step — constructing a robust homotopy map — is least understood and even considered an art [55].

A commonly used homotopy map in the second step linearly interpolates between the hard system $F(x)$ and the easy system $E(x)$:

$$\lambda \cdot F(x) + (1 - \lambda) \cdot E(x) = 0 \tag{3.3}$$

When the easy system in the above equation (3.3) is of the form $E(x) = (x - x^0)$, where x^0 is an arbitrary initial point given by the user, the method is called Fixed-Point homotopy. This is one of the most widely used form of homotopy (along with Newton homotopy [60]) because it only requires the user to provide an initial point and not the whole easy problem. However, not every easy problem with an arbitrary initial point can efficiently guide the convergence to the final solution and, as our examples in a later chapter will show, we can usually guide the optimizer more efficiently using user design equations than using general fixed-point homotopy.

3.1.1 Intent-Based Homotopy

We call our proposed approach intent-based homotopy because we use circuit designer's equations that represent their intent as the easy problem $E(x)$ in the homotopy map (3.3). These design equations usually follow the general trends of the real circuit behaviors albeit being inaccurate in the absolute quantities, like the gain and bandwidth of the differential amplifier example in Figure 3.2. If a solution cannot be reached with our intent-based homotopy then it is likely that the circuit is unsuitable to perform the intended operation for the target technology (e.g., the circuit may not have enough voltage headroom) which by itself is also a valuable feedback to the designers.

Another difference between our approach and the traditional homotopy approach to solving an optimization problem is that we do not first convert the problem to a system of non-linear equations. Instead, because solutions of successive problems in homotopy are close to one another, we directly leverage existing local optimization packages to help solve each intermediate problem using the solution of the preceding problem as the initial guess.

To create each intermediate problem, we use the linear homotopy map in Equation (3.3) to generate homotopy functions $H(x) = \lambda \cdot F(x) + (1 - \lambda) \cdot E(x)$ for each performance constraint in the optimization problem. In circuit optimization context, each $E(x)$ function represents the designer's equation for each performance metric such as power equation as a function of transistor sizes. We can easily evaluate these functions for any point in the design space with negligible cost. The $F(x)$ functions represent simulated values for the same performance metrics which can be quite expensive to evaluate at any given design point.

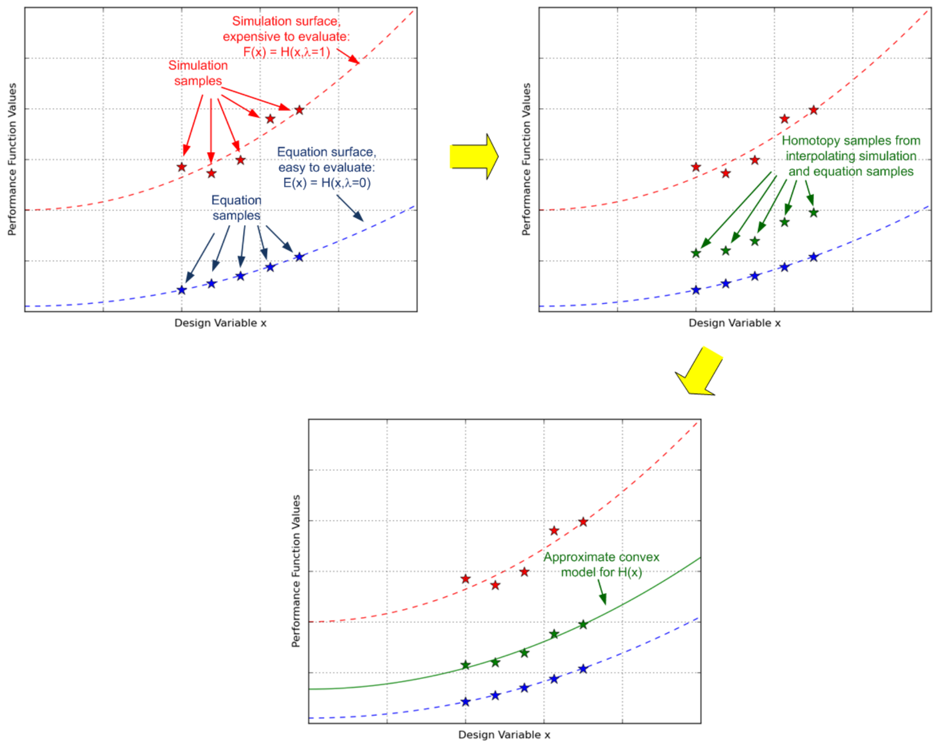


Figure 3.3: Approximate problem generation. First, we evaluate $E(x)$ and $F(x)$ functions on randomly generated design points around an initial point (top left graph). Next, we interpolate the easy and hard samples from the previous step to generate homotopy samples (top right graph). We then fit these homotopy samples to an approximate (convex) model (bottom graph) for each performance constraint function to generate an approximate representation of the problem.

Hence, instead of solving each actual intermediate problem directly, we solve an approximate representations of each problem. Figure 3.3 demonstrates how we form this approximate representation. First, we generate homotopy data samples by evaluating $E(x)$ and $F(x)$ functions on randomly generated design points (around an initial point) and then interpolating their values. These samples are then fitted to approximate (convex) models to generate an approximate representation of the problem.

We can now provide a high-level intent-based homotopy circuit optimization algorithm:

1. Solve intent-based equations to get a starting point x^0 and advance λ.
2. Form next intermediate optimization problem using intent-based homotopy and locally solve it using previous solution x^{k-1} as initial point.
3. Advance λ.
4. Goto 2 and repeat until $\lambda = 1$.

Section 3.4 will go over this algorithm in more details after we first discuss a few algorithmic issues in the next two sections.

3.2 Constrained Optimization

Most practical circuit design problems have design constraints that may either be internally or externally imposed. The internal design constraints exist to ensure circuit functionality such as making sure a current source in the circuit has enough saturation margin or that the output impedance of a supply regulator is low enough so that it can sink or source its load current. The external design specifications are usually derived from the application specifications such as a jitter requirement for an oscillator or a gain requirement for an amplifier.

The initial solution obtained from solving the simplified design equations, more often than not, will not meet all of these constraints in the simulation space which poses a serious problem to designers and optimizers. For example, it has been reported [17] that for a two-stage operational amplifier design, roughly 90% of the design time is spent on tweaking sizes in simulation to meet the design specifications even after obtaining the initial design point from the design equations. When given an infeasible initial design point, most optimizers either take an additional step to first find a feasible point [4] or just simply discard the given initial point [3].

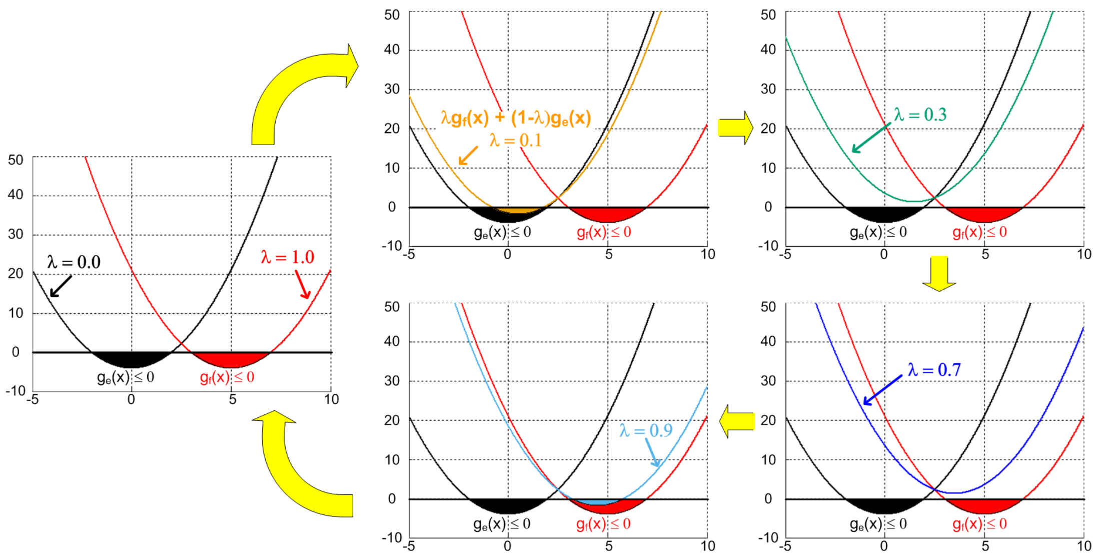


Figure 3.4: Infeasible intermediate constraint spaces. Starting from the left graph, $g_e(x) \leq 0$ is the initial easy constraint and $g_f(x) \leq 0$ is the final hard constraint. $g_e(x) = x^2 - 4$ and $g_f(x) = (x-5)^2 - 4$. The feasible region for each intermediate constraint is shaded. Between $\lambda = 0.2$ and $\lambda = 0.8$, the intermediate constraint spaces are empty as evident by $\lambda = 0.3$ and $\lambda = 0.7$ graphs.

Our optimizer can also fail when the initial design point is infeasible in simulation space if the homotopy map given in Equation (3.3) is used directly. For example, as shown in Figure 3.4, suppose the constraints for the easy and the hard problems are $x^2 - 4 \leq 0$ and $(x-5)^2 - 4 \leq 0$ respectively, then the intermediate constraints parameterized by lambda are $x^2 - 10\lambda x + (25\lambda - 4) \leq 0$. For λ values between 0.2 and 0.8 the intermediate constraints have empty feasible space and the homotopy methods can no longer trace the solution after $\lambda = 0.2$.

The reason why some of the intermediate constraints have empty feasible space is because the feasible regions for the easy and the hard problems are non-overlapping. Since a homotopy solution path connects a point inside the feasible region of the easy problem to a point inside the feasible region of the hard problem, it will need to cross over a region where both constraints are infeasible. As interpolating any two infeasible points also result in an infeasible point, the infeasible region between the easy and the hard problems can never become feasible in any of the intermediate problems. Hence, there is no way for a point that starts inside the feasible region of the easy problem to move across to the feasible region of the hard problem.

If you somehow have knowledge of a feasible point in the simulation space, one way of addressing this issue is to adjust the easy problem so that this point is also feasible in its constraint space. This can be done by simply applying a relaxation amount based on the easy constraint value at this point to the easy constraint function.

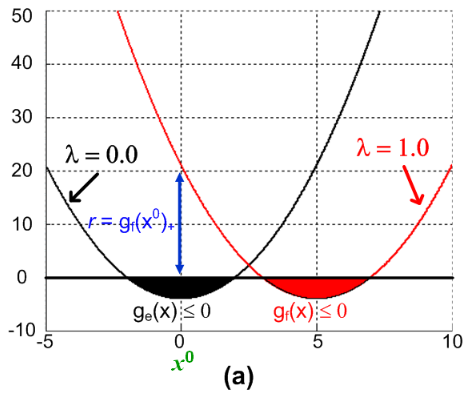
More specifically, let $g_e(x) \leq 0$ and $g_f(x) \leq 0$ be the constraints for the easy and the hard problems that have non-overlapping feasible spaces, and suppose there exists x^f such that $g_f(x^f) \leq 0$, then applying a relaxation amount $r = g_e(x^f)$ (which is always positive due to the non-overlapping property) to the original easy constraint function will make the feasible region of the new easy constraint function $g_e^*(x) \leq 0$ (where $g_e^*(x) = g_e(x) - r$) overlap with the feasible region of the hard constraint because x^f is now feasible in both the new easy and the hard constraints.

In practice, we are unlikely to know of a feasible point in simulation space a priori, so we use the solution to the easy problem instead to help estimate the initial relaxation amount. This heuristic constraint relaxation technique is similar to [44].

Again, let $g_e(x) \leq 0$ and $g_f(x) \leq 0$ be the constraints for the easy and the hard problems respectively, and let x^0 be the solution to the easy problem, we can relax the intermediate constraints as:

$$\lambda \cdot g_f(x) + (1-\lambda) \cdot g_e(x) \leq (1-\lambda) \cdot g_f(x^0)_+ \tag{3.4}$$

In other words, we use the violation amount of the initial point in the hard constraint, $g_f(x^0)_+$ (i.e., the positive part of $g_f(x^0)$) as our initial relaxation amount i.e., $r = g_f(x^0)_+$. This relaxation is gradually removed as we move towards the hard problem as illustrated in Figure 3.5. Obviously, if the hard problem itself has empty



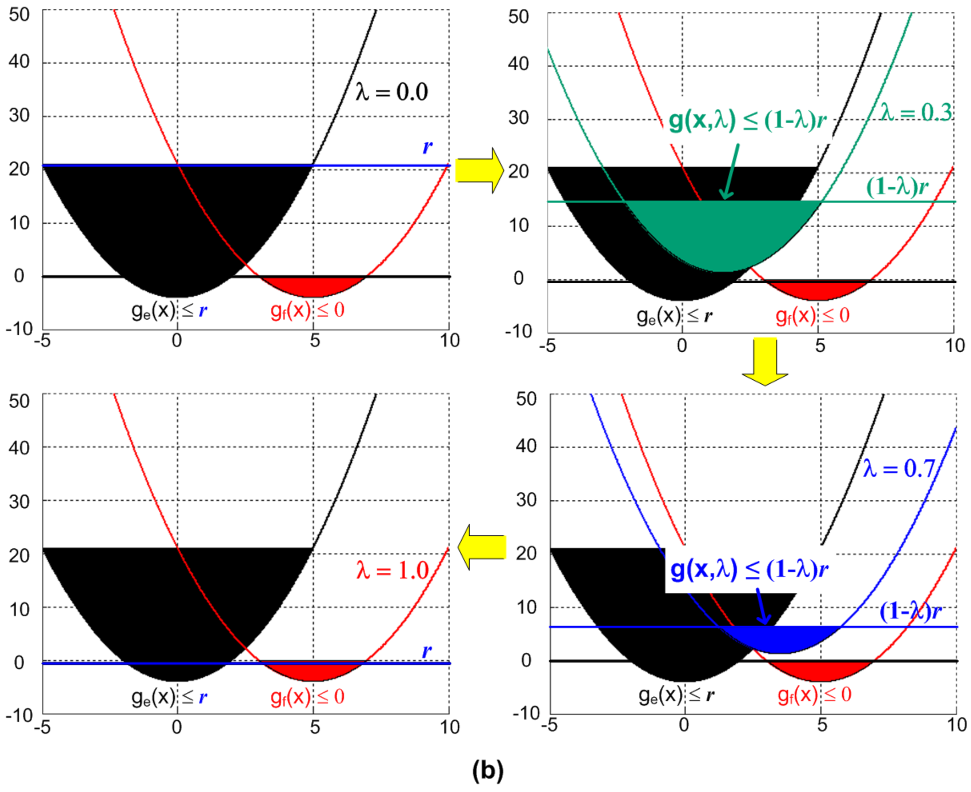


Figure 3.5: Constraint Relaxation. (a) Initial relaxation amount is equal to the initial violation of the initial point in the hard constraint $r = g_f(x^0)_+$. (b) Evolution of the relaxed constraint space through homotopy interpolation from $\lambda = 0.0$ (top left) to $\lambda = 0.3$ to $\lambda = 0.7$ then to $\lambda = 1.0$ (bottom left). The relaxation amount is gradually driven to zero as we move from the easy problem to the hard problem.

feasible space, then relaxation cannot help. Therefore, if the algorithm cannot advance its solution, it is highly indicative that the circuit cannot be realized as intended for the target design specifications.

3.3 Convergence of Homotopy Methods

The path from the easy problem to the hard problem may be fraught with difficulties as it may encounter many path irregularities. As the previous section shows, the path may not exist from the easy problem to the hard problem even though the hard problem is solvable because intermediate problems may become infeasible. The path may start from one solution of the easy problem and then loops back to another solution of the easy problem. The path may split or get stuck in an infinite spiral. See [60] for a more comprehensive study of possible path singularities.

Fortunately, the convergence theory for homotopy methods have been studied extensively and the main criteria for the guaranteed existence of a homotopy path can be informally summarized as [55]:

1. All intermediate constraints must not have empty feasible space.
2. The easy problem must yield a single, unique solution.
3. The solution path must have no singular points.

The first condition ensures that a path exists between the solutions of the easy and the hard problems. We use a heuristic constraint relaxation technique described in the previous section to help avoid this problem, assuming that the hard problem has non-empty feasible space.

The second condition ensures that once we trace the path from the initial problem, it does not come back and re-cross $\lambda = 0$ hyperplane but steadily progresses toward reaching the final solution at $\lambda = 1$. One of the reason why the fixed-point homotopy map is popular is because there is only one solution at $\lambda = 0$ given by $x = x^0$ guaranteeing that we meet this second criteria.

In our case, since we do not impose upon the user to provide an initial problem with only one, unique solution, we can meet the second condition by superposing a fixed-point homotopy on the intent-based homotopy for the objective function. Suppose that our design equation objective is to minimize $f_e(x)$ and our simulation-based objective is to minimize $f_s(x)$, then we can construct our new superimposed objective for the intermediate problems as:

$$\text{minimize } \lambda \cdot f_h(x, \lambda) + (1 - \lambda) \cdot \parallel x - x^0 \parallel^2 \tag{3.5}$$

where $f_h(x, \lambda) = \lambda \cdot f_s(x, \lambda) + (1 - \lambda) \cdot f_e(x)$ is the basic linear-mapped homotopy of the easy and the hard problems, and $\parallel x - x^0 \parallel^2$ is the equivalent fixed-point form of the easy problem in the optimization context. If x^0 is chosen to be the minimum of $f_e(x)$ (unlike in a general fixed-point homotopy where x^0 can be any arbitrary point), then the superimposed objective will only have one solution at $x = x^0$. In practice, however, we have not found a need to use this feature so it is turned off by default in our optimizer as it also increases optimization time.

The third condition for path existence is to rule out strange behaviors such as path bifurcation, path splitting or path turning points [60]. In real-numbered homotopy systems, the only singularities that can occur along the solution path are quadratic turning points [33, 35]. This simply means that the solution path may not be monotonic with respect to any of the design parameters or the mapping parameter λ. Consequently, optimizers that advance the homotopy mapping parameter λ monotonically may encounter serious difficulties as the solution for the next intermediate problem may actually be quite far away. To deal with these singular quadratic points, we can use a different controlling parameter — typically the arc length of the path s — to help advance the intermediate problem and then determine the next λ based on the current arc length. This is what we also do in our adaptive stepping algorithm that will be described in more details in the next section.

3.4 Circuit Optimization Algorithm Details

As shown in Section 3.1.1, our intent-based homotopy circuit optimization algorithm consists of two nested loops. The outer homotopy loop drives the progression from the easy problem based on the simple design equations to the final problem based on simulation. The inner local optimization loop optimizes the intermediate problems using previous solution as the initial guess.

3.4.1 Inner Local Optimization Loop

We implemented our inner local optimization loop based on sequential convex programming (SCP) [11] which solves a non-convex optimization problem by solving a sequence of approximate convex problems. We fit data samples to convex functions to generate an approximate convex problem and then use optimization package MOSEK (http://www.mosek.com) to efficiently solve the approximate problem. As SCP technique does not require derivatives of the objective and constraint functions, we can apply our algorithm on problems that have non-smooth or even discontinuous surfaces.

Because each approximate problem is just that — an approximation — each approximate problem also has an associated trust region T^k representing the set of design points where we trust the models used to form the problem in iteration k. The size of this trust region is updated adaptively as we progress through the sequence of approximate convex problems in the SCP algorithm.

How the trust region T^k updates depend on how well the optimizer is doing in each iteration of the local optimization loop. One way of determining how well it is doing is to look at the fitting error. If the fit is good, you increase the trust region and if it is bad you make the trust region smaller.

Another way is to look at how much progress the optimizer is making in each iteration. If it seems that it is moving too slow towards the local optimum, the trust region is increased and vice versa. A good heuristic that is commonly used [44] to determine optimizer progress is to look at how the predicted decrease in the penalty function value compares to the actual decrease in the penalty value.

Let $\phi(x^k)$ be the penalty function associated with simulation problem at iteration k and $\hat{\phi}(x^k)$ be the penalty function associated with the convex approximation at iteration k, then the predicted and the actual decrease are given by [1]:

$$\begin{aligned} \Delta_{predicted} &= \phi(x^{k-1}) - \hat{\phi}(x^k) \\ \Delta_{actual} &= \phi(x^{k-1}) - \phi(x^k) \end{aligned} \tag{3.6}$$

If the actual decrease is more than (some fraction of) the predicted decrease, it indicates that the model did a good job in predicting the decrease in ϕ so the trust region is subsequently enlarged. Conversely, if the actual decrease was less, then it indicates that the optimizer did a bad job in predicting the decrease in ϕ so the trust region is subsequently shrunk.

3.4.2 Adaptive Stepping Algorithm

To help advance the homotopy mapping parameter λ, we use the arc length of the path, s, as our controlling parameter to help us deal with singular turning points mentioned in Section 3.3 and use the following adaptive stepping algorithm to determine how far along the arc we want to advance at each iteration step.

Let T^k be the trust region associated with the approximate models generated at each outer homotopy iteration, we can estimate the next amount of arc length Δs to move along the path by solving a linear sub-problem:

$$\begin{array}{ll} \text{maximize} & \Delta s \\ \text{subject to} & x^k + \frac{dx}{ds} \cdot \Delta s \in T^k \end{array} \tag{3.7}$$

where x^k is the solution of the current intermediate problem and $\frac{dx}{ds}$ is the solution gradient with respect to the arc length parameter. What this does is to find the largest amount of arc length Δs possible that we can advance along the path while keeping the design point within the trust region T^k of the current intermediate problem. Based on the next predicted arc length, the next homotopy mapping parameter can then be derived as:

$$\lambda^{k+1} = \lambda^k + \frac{d\lambda}{ds} \cdot \Delta s \tag{3.8}$$

Let the solutions to $H(x, \lambda) = 0$ be the solution path connecting the solutions of the easy and the hard problems at $\lambda = 0$ and $\lambda = 1$ respectively, then the gradients $\frac{dx}{ds}$ and $\frac{d\lambda}{ds}$ can be found by differentiating $H(x, \lambda) = 0$ and solving it:

$$\begin{bmatrix} \nabla_x H(x, \lambda) & \frac{\partial H(x,\lambda)}{\partial \lambda} \end{bmatrix} \cdot \begin{bmatrix} \frac{dx}{ds} \\ \frac{d\lambda}{ds} \end{bmatrix} = 0 \tag{3.9}$$

As $\nabla_x H(x, \lambda)$ and $\frac{\partial H(x,\lambda)}{\partial \lambda}$ can be analytically obtained from the approximate convex models without requiring extra simulations, there is negligible extra computation cost in implementing this adaptive stepping algorithm.

3.4.3 Final Circuit Optimization Algorithm

Our final intent-based homotopy circuit optimization algorithm below adds the constraint relaxation described in Section 3.2 and the adaptive stepping algorithm described in Section 3.4.2 to the general algorithm presented in Section 3.1.1:

1. Solve intent-based equations to get x^{intent}.

2. Estimate constraint relaxation r based on simulated value at x^{intent} i.e., $r = g_f(x^{intent})$ for each constraint function.

3. Solve relaxed initial problem below and use its solution as the initial point for

the first intermediate problem in step 4).

$$\begin{array}{ll} \text{minimize} & f_e(x) \\ \text{subject to} & g_e^i(x) \le r^i \quad i = 1, \ldots, m \end{array} \tag{3.10}$$

4. Construct next intermediate problem using linear homotopy map in Equation (3.3).

5. Locally solve each intermediate problem using previous optimal point as initial point.

6. Estimate next λ based on the adaptive stepping algorithm in Section 3.4.2.

7. Repeat steps 4-6 until $\lambda = 1$.

We can reduce the total number of simulations required by the SCP technique by recycling some of the simulation results obtained from the previous iterations. Suppose each iteration step of the inner local optimization loop (step 5 in the algorithm above) requires M samples to construct an approximate convex problem. Let T^k be the trust region associated with the approximate models generated at each iteration k, then if a simulation sample from the previous iteration $k-1$ falls within the current trust region T^k, then we can reuse this sample as part of the M samples needed to construct the current approximate problem.

Chapter 4

Design Examples

We will show through a series of examples in this chapter how our proposed optimizer allows the designers to design their circuits using their simplified design equations while achieving a locally optimal solution in the simulation space.

Each example will start off with the design equations that are used to help guide our intent-based circuit optimizer followed by a discussion on the optimizer results. In the results section, we will discuss the discrepancies between the design equations and the simulation results, and how the optimizer takes care of the non-idealities that are not modeled in the design equations.

Our four examples will explore different aspects of our optimizer. We will start off with a simple ring oscillator design example that shows how designers can use relatively simple design equations to help guide our optimizer. Next, we will demonstrate how our constraint relaxation technique works in practice with a replica-compensated regulator design example. The StrongARM sense-amplifier latch design example will show how difficult it is to write very accurate equations for some circuit behavior — input-referred-noise in this case — and how our optimizer can be more robust than a local optimizer when the initial design point, obtained from solving the design equations, is infeasible. The last example, a charge-pump design, will demonstrate a usage of statistical constraints in our design objectives. All the design examples

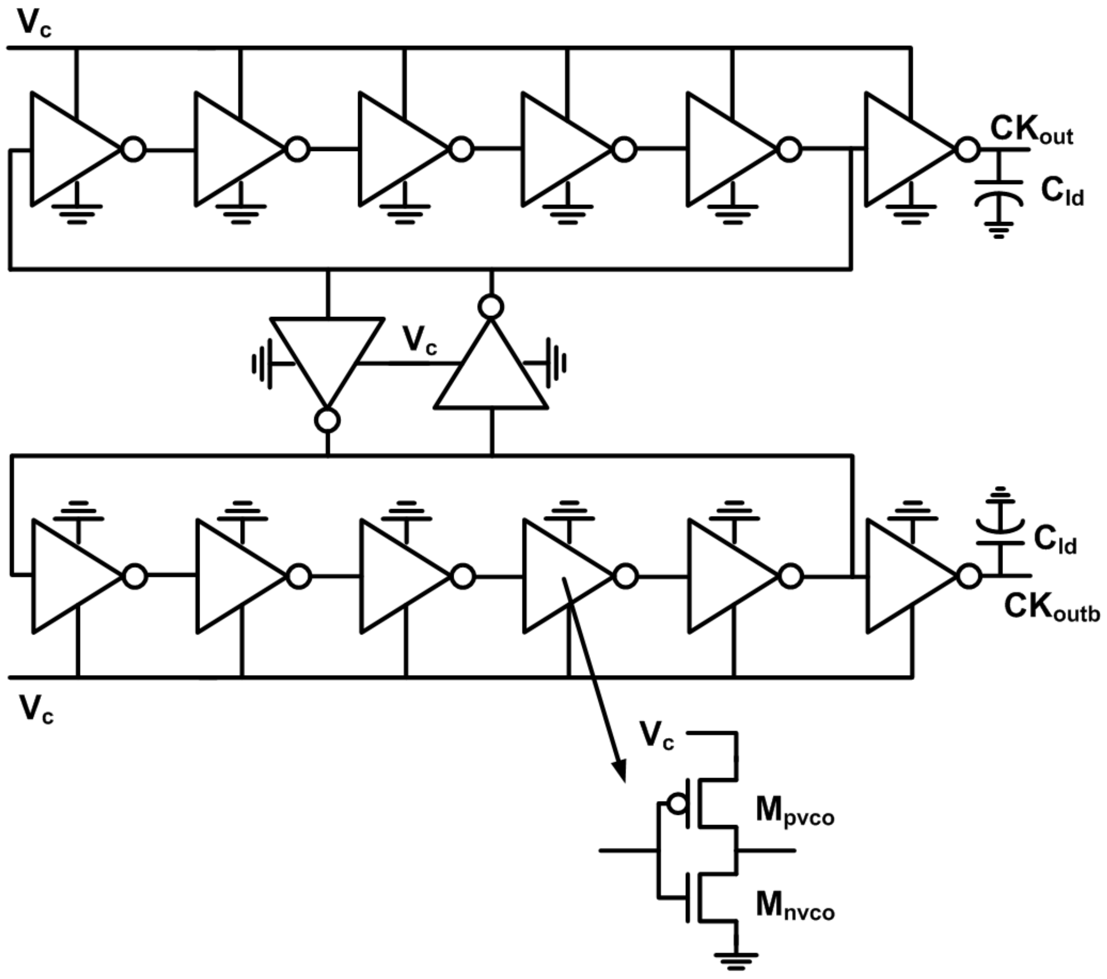


Figure 4.1: 5-stage Differential Voltage-Controlled Ring Oscillator.

in this section were done in 65nm PTM technlogy (http://ptm.asu.edu). Variants of these examples also appeared in [29] which were done in TSMC 0.18μm technology.

4.1 Low Power, Low Phase Noise, Voltage Controlled Oscillator

Our first design example is a 2-variable, 5-stage differential voltage-controlled ring oscillator (VCO) with an output buffer stage, as shown in Figure 4.1.

Problem Description

The design objective for our VCO is to minimize its energy consumption in each cycle measured at the nominal control voltage $V_c = V_{dd}/2$ (0.55V) subject to the following constraints:

- **Output Frequency Range**. The VCO output frequency should span 1GHz-2GHz range over the 0.4-0.8V control voltage tuning range.
- **Output Phase Noise**. The VCO output phase noise measured at 1MHz offset should be less than -94dBc/Hz.
- **Transistor Size Ratio**. The pMOS-to-nMOS size ratio of the inverter element should be less than 5.

The frequency and phase noise constraints are usually derived from the application specifications while the size ratio constraint is internally imposed to ensure that the resulting VCO sizes are reasonable.

The two design variables are the widths of the pMOS and nMOS transistors, W_p and W_n, in each VCO delay element.

Design equations

As frequency is the inverse of delay, we can use an RC model for the transistors to first estimate the delay of each inverter before summing up twice its delay around the ring and then inverting to obtain the VCO oscillation frequency. In this RC model, the switch on-resistance is inversely proportional to the transistor width while the node capacitance at each output of the inverter consists of gate and drain capacitances that are proportional to the transistor width i.e.,

$$\begin{aligned} R_n &= R_{sqn}(V_c) \cdot \frac{L_n}{W_n} \\ R_p &= R_{sqp}(V_c) \cdot \frac{L_p}{W_p} \\ C_{node} &= (C_g + C_j) \cdot (W_n + W_p) \end{aligned} \tag{4.1}$$

where R_{sqn} and R_{sqp} are the equivalent sheet resistances of the nMOS and pMOS transistors in $\Omega/\Box$ while C_g and C_j are gate and junction capacitances per unit width in $F/\mu m$ respectively. These four parameters can be calibrated for each control voltage V_c for a given technology.

The oscillation frequency is then given by:

$$\begin{aligned} f_{osc} &= \frac{1}{2 \cdot N \cdot t_d} \\ t_d &= 0.5 \cdot (R_n + R_p) \cdot C_{node} \end{aligned} \tag{4.2}$$

where $N = 5$ is the number of stages in our ring oscillator.

As the power consumption of the VCO is mostly dynamic and proportional to the total capacitance C_{total} being charged and discharged in each cycle, we can use the same RC model to estimate the power consumption. The total capacitance C_{total} consists of the $2N$ node capacitances (as there are two rings) that are being discharged twice in a cycle plus the load capacitance, i.e., $C_{total} = 4 \cdot N \cdot C_{node} + C_{ld}$. The power consumption is then given by:

$$Power = C_{total} \cdot V_c^2 \cdot f_{osc} \tag{4.3}$$

where V_c is the input control voltage.

The energy consumption per cycle is then given by:

$$Energy = \frac{Power}{f_{osc}} \tag{4.4}$$

For the phase noise, we assume that it is inversely proportional to the power based on the Leeson's model [34]:

$$PN = \frac{K_{pn}}{Power} \tag{4.5}$$

where K_{pn} is a proportionality constant that can be calibrated for a given technology process.

Results

As this is a simple 2-variable problem, we can sweep the whole design space in simulation and compare each performance function to its model counterpart. Figure 4.2 and 4.3 show comparisons between the design equations and the simulation results for energy and phase noise (measured at 1MHz offset) over the design variable space respectively. Even though our design equations were not very complex, they matched up with the simulation results very well.

Figure 4.4 shows the solution path through the energy vs. phase noise space. Since our equations underestimated energy and overestimated phase noise, the initial optimal point from the design equations has higher energy and lower phase noise than the final optimal point in simulation.

Table 4.1 shows the simulated results for the initial design point and final design point. We comfortably met the frequency range specification so it played a negligible role in the optimization process. As the energy and phase noise are roughly constant for a given total capacitance (see Equation (4.4) and (4.5)), there may be many multiple optimal solutions to our problem (along the hyperplane where $W_p + W_n$ is constant). Hence, we also constrained the pMOS-to-nMOS ratio to keep the size in a practical range. Note that we could have added a minimum constraint on the size ratio as well but since the optimizer gravitated towards large ratio size, it was not necessary.

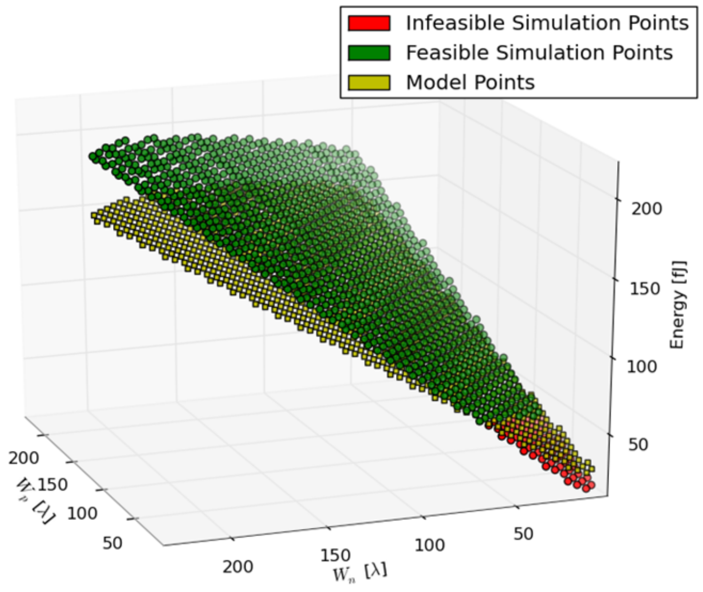


Figure 4.2: Energy comparison between design equations and simulation results over the design variable space.

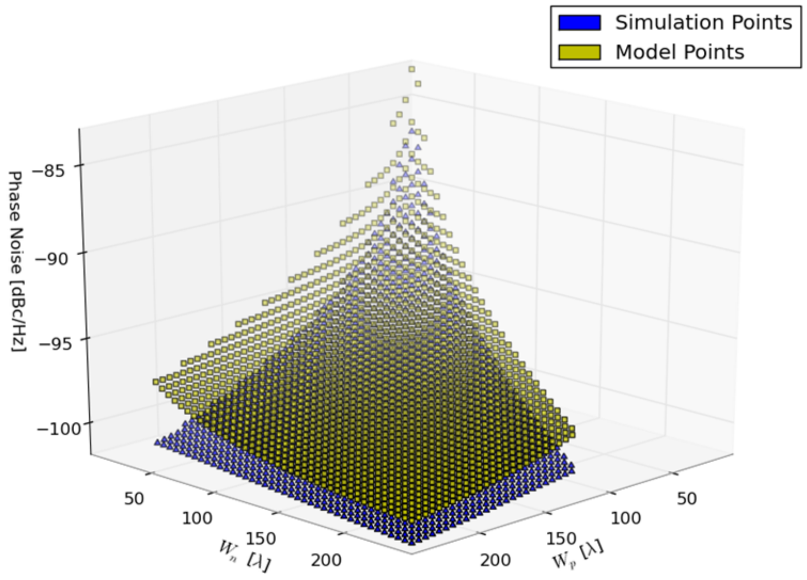


Figure 4.3: Phase noise at 1MHz offset comparison between design equations and simulation results over the design variable space.

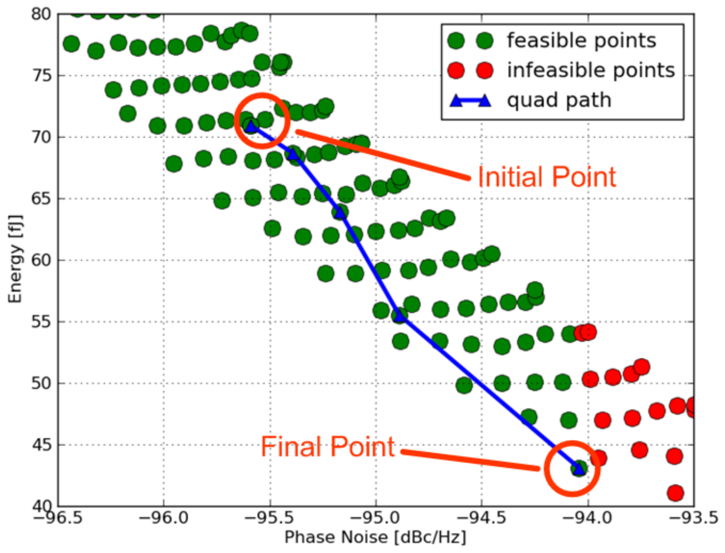
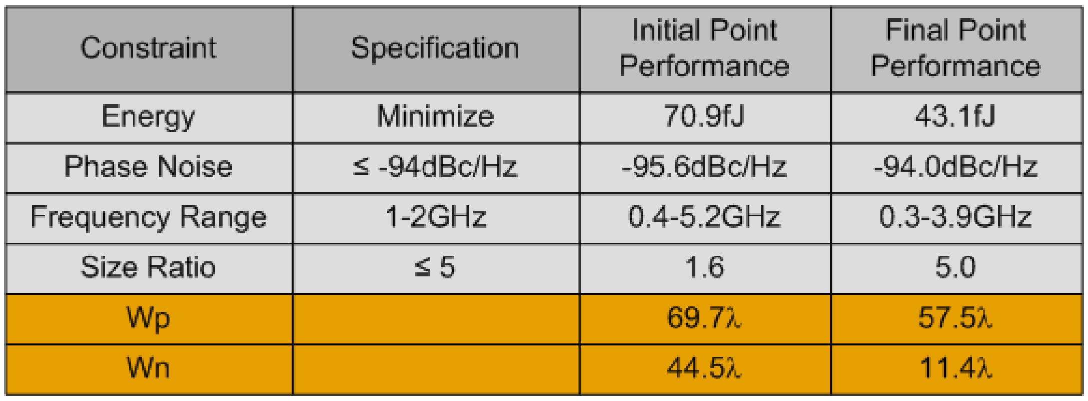


Figure 4.4: Homotopy Solution Path for the VCO design example through Energy vs. Phase Noise simulation space.



Constraint	Specification	Initial Point Performance	Final Point Performance
Energy	Minimize	70.9fJ	43.1fJ
Phase Noise	≤ -94dBc/Hz	-95.6dBc/Hz	-94.0dBc/Hz
Frequency Range	1-2GHz	0.4-5.2GHz	0.3-3.9GHz
Size Ratio	≤ 5	1.6	5.0
Wp		69.7λ	57.5λ
Wn		44.5λ	11.4λ

Table 4.1: Simulation Result for the initial design point and the final design point.

4.2 Replica-Compensated Supply Regulator

Our next design example is a 7-variable replica-compensated supply regulator [2] shown in Figure 4.5 which will be used in a PLL design example in Chapter 6 to drive our VCO from the previous section.

Problem Description

The design objective here is to minimize its power consumption subject to the following constraints:

- **Supply Sensitivity**. The maximum supply sensitivity (over frequency) should be less than -24dB.
- **Regulator Bandwidth**. The overall regulator bandwidth should be greater than 500MHz which is roughly 20 times greater than the PLL loop bandwidth.
- **Effective Amplifier Bandwidth**. The effective amplifier bandwidth (for the transfer function from node V_{in} to V_{bp}) should be greater than 1.5GHz. This is 3 times greater than the regulator bandwidth (or roughly 60 times greater than PLL loop bandwidth).
- **Output Current**. The regulator should be able to source at least 120μA of load current at $V_{in} = V_{dd}/2 = 0.55V$.

The supply sensitivity constraint is usually derived from the application specifications. For our example, the supply sensitivity of -24dB translates into about 0.06% change in regulated output voltage V_{reg} for every 1% change in supply voltage V_{dd}.

The bandwidth and output current constraints are internally imposed to ensure the functionality of the regulator when it is used inside a PLL. The first two bandwidth constraints ensure that the regulator will not degrade PLL stability assuming PLL loop bandwidth is around 25MHz.

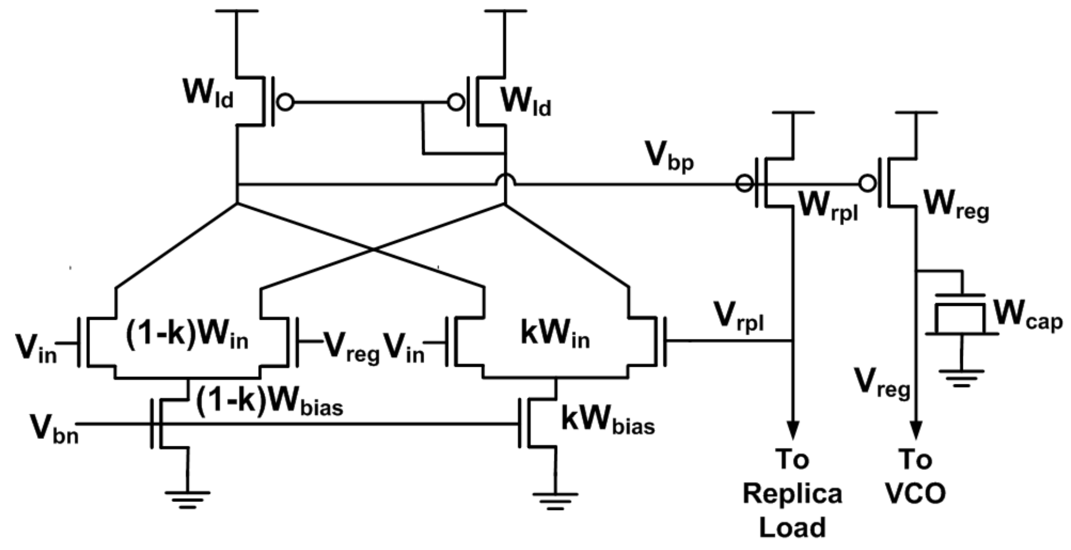


Figure 4.5: Replica-Compensated Supply Regulator.

The output current constraint is to ensure that the regulator is capable of sourcing the VCO load current. From the VCO example in Section 4.1, the VCO consumes roughly 65μW at $V_c = 0.55V$ which translates to about 120μA of current that the regulator has to source.

The seven design variables include six transistor widths (W_{ld}, W_{in}, W_{bias}, W_{rpl}, W_{reg}, W_{cap}) and the weighting allocation between feed-forward and feedback path (k).

Design Equations

Most of the design equations here are based on [2]. This section provides a brief summary of the design equations relevant to our design problem.

As our regulator circuit is purely analog, we expect the power consumption to be simply the sum of the static current in each branch multiplied by the supply voltage.

$$Power = (I_{bias} + I_{rpl} + I_{reg}) \cdot Vdd \tag{4.6}$$

where $I_{bias} = (W_{bias}/W_{ref}) \cdot I_{ref}$, and I_{rpl} and I_{reg} are the current through M_{rpl} and M_{reg}, respectively. The expressions for the currents can be obtained using the simple long-channel device model: $I = (W/2L) \cdot \mu C_{ox} \cdot (V_{gs} - V_{th})^2$.

As there are two main poles in the regulator — the pole due to the amplifier stage and the pole due to the output stage driving a large load capacitance (e.g., from ring oscillator) — the regulator bandwidth can be dominated by either of these poles. However, as we want a high amplifier bandwidth to track out supply noise, our intent is to have the output pole be the dominant one. Assuming this is the case, the regulator bandwidth will then be approximately set by the output pole ω_o (pole due to parasitic capacitances at V_{reg}) as:

$$\omega_{bw} \approx \frac{(1 + A_a A_o)}{(1 + kA_a A_o)} \cdot \omega_o \approx \frac{\omega_o}{k} \tag{4.7}$$

where A_a and A_o are the open-loop DC gains of the amplifier stage and the output stage respectively. In other words, the replica feedback reduces the amplifier gain by roughly $1/k$, which means, for unity gain feedback, the output pole pushes out by $1/k$ as well. k is one of the design variables representing the weighting allocation between feed-forward and feedback path.

The expression for the effective amplifier bandwidth is hard to derive exactly due to the frequency dependent feedback of the local replica. However, as we intend the replica pole to be much higher than the output pole so that all the amplifier gain can be applied to suppress the supply noise, we can approximate the effective amplifier bandwidth as the open-loop amplifier bandwidth multiplied by one plus the total DC feedback gain i.e.,

$$\omega_{aeff} = (1 + kA_{rep}A_a) \cdot \omega_a \tag{4.8}$$

where A_{rep} represents the DC gain of the replica output stage (from V_{bp} to V_{rpl}), A_a and ω_a are the open-loop DC gain and bandwidth of the amplifier, respectively.

The supply sensitivity of the regulator mainly comes through the output device M_{reg} where its open-loop sensitivity S_{vdd} is equal to the resistive divider ratio between the VCO resistance and the output device resistance. However, due to amplifier feedback, the sensitivity is suppressed by one plus the feedback gain $A_a A_o$. Note that because both the main path and the replica path in the amplifier stage respond to supply noise, the amplifier will use all of its gain to suppress the supply noise, and is independent of the k factor. The maximum sensitivity then occurs near ω_{bw} where the amplifier gain drops by ω_a/ω_{bw}. Hence the worst-case sensitivity can be approximated by:

$$S_{max} = S_{vdd} \cdot \frac{\omega_{bw}}{\omega_a \cdot (1 + A_a A_o)} \tag{4.9}$$

The output current can be obtained using long-channel device model applied to the output PMOS transistor.

Results

Originally, we optimized the regulator without the output current constraint and found that the PLL had serious trouble locking with this supply regulator. This is because the regulator was not able to source enough current required by the VCO in its operating frequency range, which resulted in the regulated output voltage saturating out at a voltage lower than required by the VCO.

More specifically, Figure 4.6 shows the input-output transfer functions of the supply regulator for the ideal case and for the optimized regulator without the output current constraint. In the ideal case, the output V_{reg} is a direct copy of the input V_c. However, our regulator saturated out at around 0.5V while Figure 4.7 shows that the control voltage range required for the VCO to operate between 1-2GHz is roughly from 0.5-0.6V. Hence, our PLL failed to lock with this regulator because the maximum regulator output voltage is outside the VCO operating range. This failed example served to remind us of the "Garbage In Garbage Out" principle — that the resulting design from performing the optimization is only as good as how we constrained the problem.

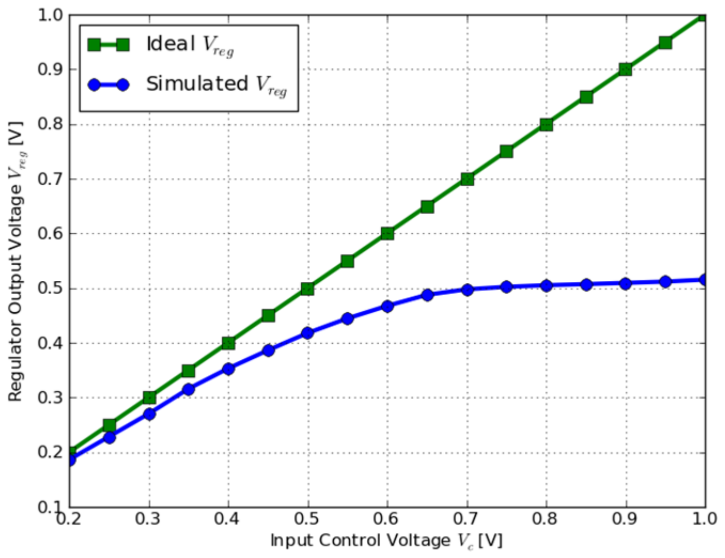


Figure 4.6: Input-Output transfer function for the bad regulator design without output resistance constraint.

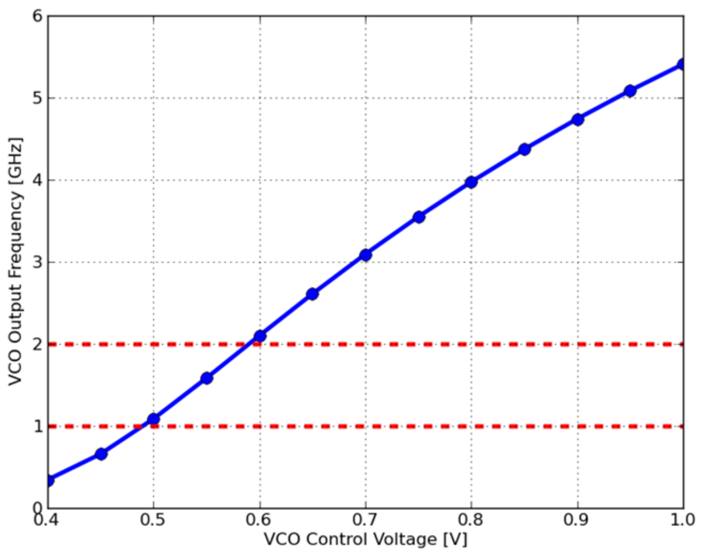


Figure 4.7: VCO tuning range curve with the output frequency range specification indicated as two dashed-lines.

Constraint	Specification	Initial Point Performance	Final Point Performance
Power	Minimize	130μW	174μW
Supply Sensitivity	≤ -24dB	-16.0dB	-23.8dB
Regulator Bandwidth	≥ 500MHz	1.1GHz	526MHz
Amplifier Bandwidth	≥ 1.5GHz	1.1GHz	4.6GHz
Output Current	≥ 120μA	122μA	120μA
W_{ld}		135λ	49λ
W_{in}		350λ	359λ
W_{bias}		90λ	108λ
W_{rpl}		64λ	75λ
W_{reg}		259λ	174λ
W_{cap}		396λ	562λ
k		0.18	0.42

Table 4.2: Simulated performance comparison between the initial and the final designs for the supply regulator circuit.

Table 4.2 summarizes regulator's simulated performance for both the initial and the final design points (with the output current constraint). The initial design did not meet supply sensitivity and effective amplifier bandwidth constraints. Figure 4.8 shows that the supply sensitivity of the initial design was not met because of a peaking around regulator bandwidth frequency (1.1GHz), which was significantly reduced in the final design. This is in agreement with our assumptions as, in deriving equation (4.9), we had expected the worst-case sensitivity to occur around the regulator bandwidth frequency. This peaking behavior can also be seen in the regulator AC transfer function for the initial design point as shown in Figure 4.10. Figure 4.9 shows that the optimizer meets the final amplifier bandwidth specification by reducing the amplifier gain from the initial design point and thus extends out the bandwidth.

To make sure that we no longer have the output voltage saturation problem, Figure 4.11 shows the input-output transfer function again for the new optimized design with

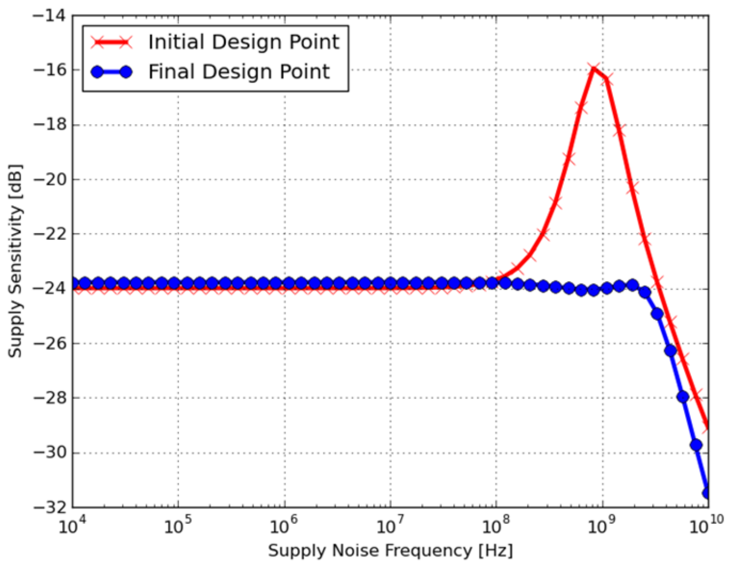


Figure 4.8: Simulated regulator's supply sensitivity AC transfer function for both the initial and the final designs.

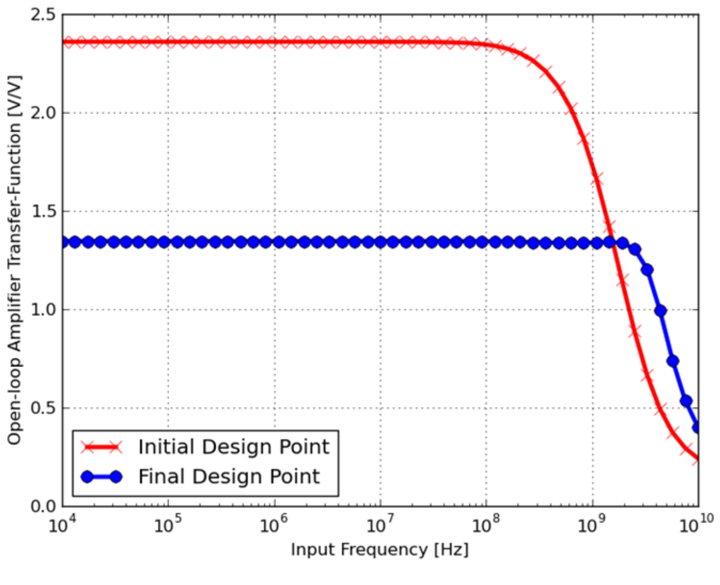


Figure 4.9: Simulated regulator's AC transfer function for the amplifier stage (with replica feedback) for both the initial and the final designs.

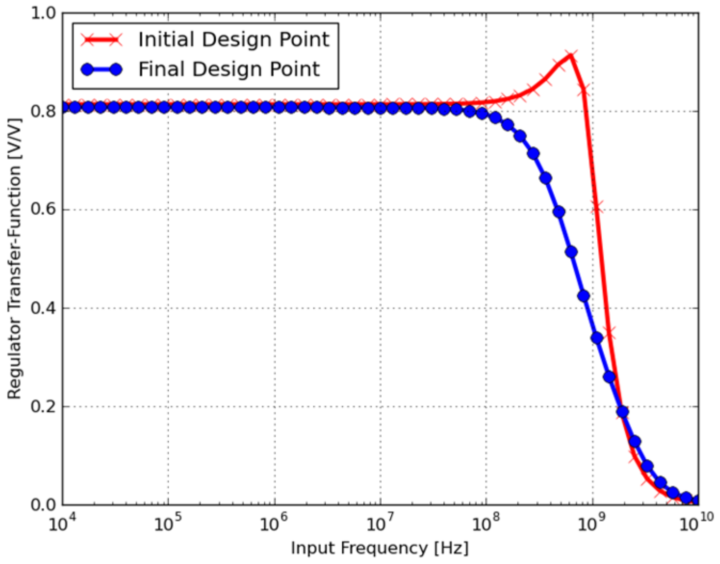


Figure 4.10: Simulated regulator's AC transfer function for both the initial and the final designs.

the output current constraint. Our regulated output voltage here can comfortably cover the required VCO input range of 0.5-0.6V.

Figure 4.12-4.14 shows optimizer solution paths from the initial design point ($\lambda = 0$) to the final design point ($\lambda = 0$) for power, supply sensitivity, and the effective amplifier bandwidth constraints. We can see that even though our initial design was infeasible in simulation problem, the intermediate designs remained feasible in all the relaxed intermediate problems where the relaxation was gradually removed as we moved towards the final problem. Because the supply sensitivity was comfortably met in our initial relaxed problem, the optimizer started off by focusing on reducing the power (Figure 4.12) until the supply sensitivity hit the constraint boundary (Figure 4.13) roughly around $\lambda = 0.6$, at which point the homotopy power started to increase back up to maintain feasibility.

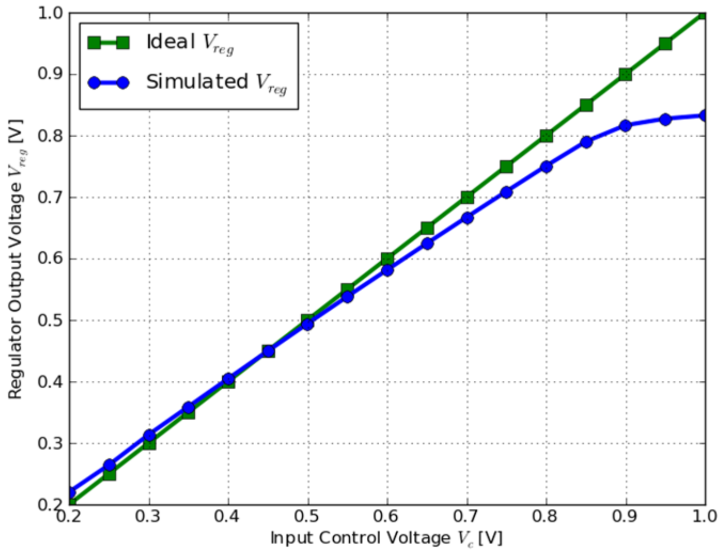


Figure 4.11: Regulator's Input-Output transfer function for the final design.

Based on the homotopy value and simulated value at $\lambda = 0.0$, Figure 4.13 and 4.14 tell us that we overestimated effective amplifier bandwidth and underestimated supply sensitivity in our design equations. This is likely because we overestimated the gain factors in our design equation by using simple long-channel model. Equation (4.8) and (4.9) show that both the effective amplifier bandwidth and supply sensitivity are dependent on the gain factors. Note that because ω_a is inversely proportional to A_a for a given power, the effective amplifier bandwidth is only dependent on the g_m/C factor of the amplifier stage and not on its output resistance. Hence, the error in the effective amplifier bandwidth estimation is likely due to errors in estimating the g_m of amplifier stage and A_{rep} while the error in the supply sensitivity is likely due to errors in estimating g_m of the amplifier stage and A_o. Figure 3.2 from Chapter 3 shows that we can overestimate gain (and hence g_m because R_L is fixed in Equation 3.1) by a factor of 4 based on this model. This is roughly 12dB which is close to the discrepancy in our supply sensitivity results.

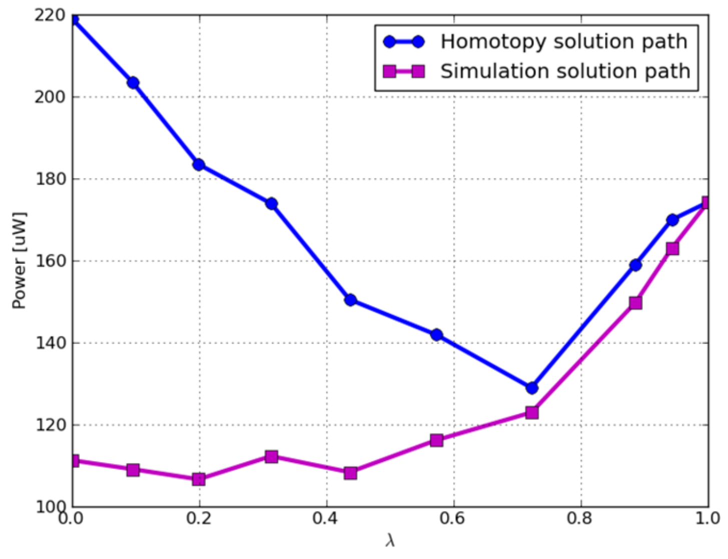


Figure 4.12: Power solution path from the initial design point ($\lambda = 0$) to the final design point ($\lambda = 1.0$) for the supply regulator design.

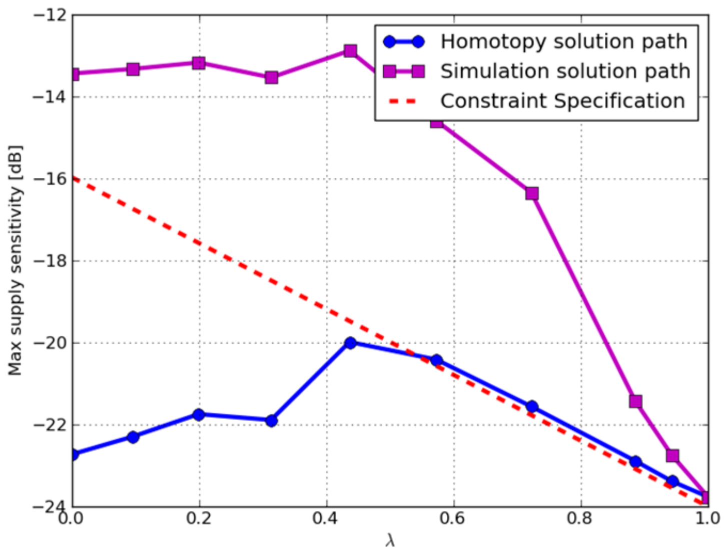


Figure 4.13: Supply sensitivity solution path from the initial design point ($\lambda = 0$) to the final design point ($\lambda = 1.0$) for the supply regulator design.

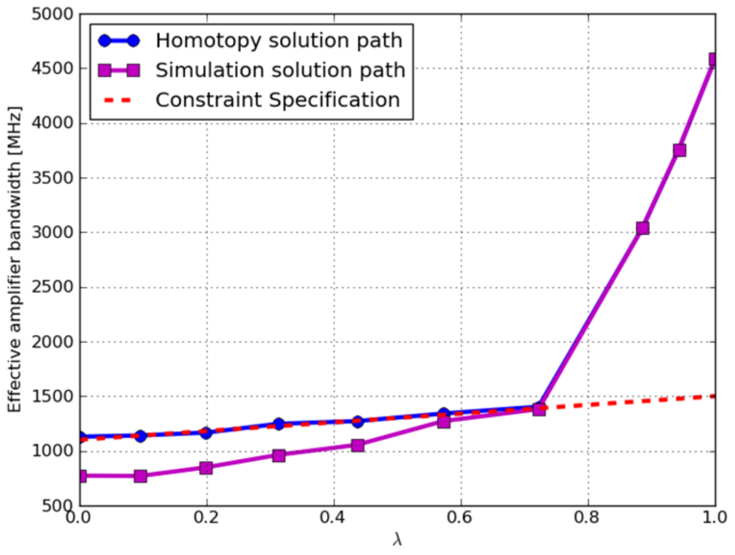


Figure 4.14: Effective amplifier bandwidth solution path from the initial design point ($\lambda = 0$) to the final design point ($\lambda = 1.0$) for the supply regulator design.

4.3 Clocked Comparator

Our next design example is a 7-variable StrongARM comparator shown in Figure 4.15. This example will show that it can be very difficult to write accurate design equations for some circuit performance behavior. More specifically, writing accurate input-referred-noise equation for fully-dynamic StrongARM comparator is very difficult and has only recently been done analytically in [41]. Yet as we will see, the solution provided by the complex analytical expression underestimated the noise by almost a factor of 2 in our example and also did not meet the design specifications.

Problem Description

The design objective for the StrongARM comparator is to minimize its power consumption subject to the following constraints:

- **Clock-to-Output Delay**. The *clk-q* delay should be less than 100ps.

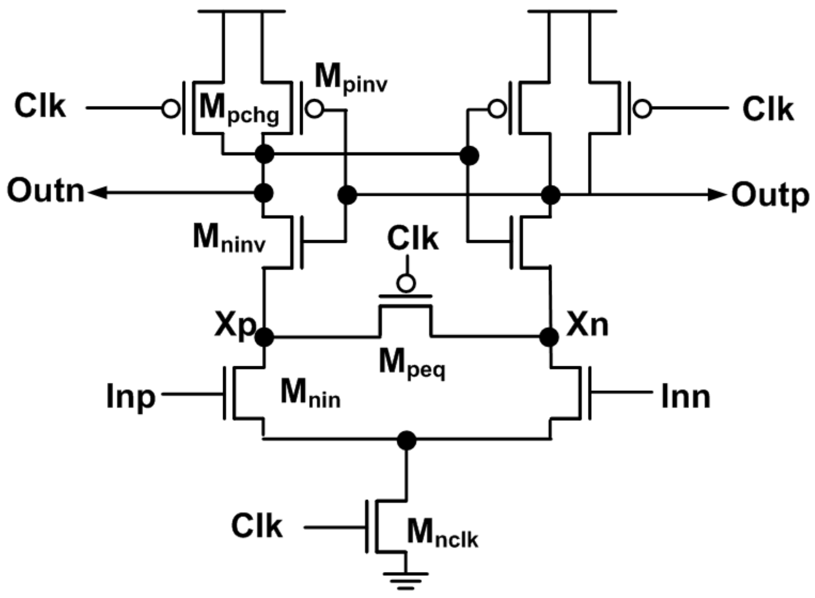


Figure 4.15: StrongARM Comparator.

- **Input-Referred-Noise**. This should be less than 75μV rms.

Both of these two constraints are externally imposed by the application. The seven design variables include six transistor widths (W_{nclk}, W_{nin}, W_{ninv}, W_{pinv}, W_{peq}, W_{pchg}) and the input common-mode voltage (V_{cm}).

Design Equations

Power consumption is given by its dynamic power dissipation:

$$Power = \alpha \cdot C_{total} \cdot Vdd^2 \cdot f_0 \tag{4.10}$$

where $C_{total} = C_{tail} + C_x + C_{out} + C_{clkdrv} + C_{inputdrv}$ is the total capacitance being discharged in one cycle, f_o is the operating frequency, and α is the average activity factor. C_{clkdrv} and $C_{inputdrv}$ are the total capacitances driven by the input and clock buffers (not shown) in each cycle respectively including the gate capacitances of the M_{nin} and M_{peq} devices.

The *clk-q* delay is quite complex to estimate due to the cross-coupled transistors so we will ignore their feedback effect to simplify our equations and use RC model to model the delay through each transistor:

$$t_{clkq} = ln(2) \cdot (R_{nclk} \cdot C_{tail} + (R_{nclk} + R_{nin}) \cdot C_x + (R_{nclk} + R_{nin} + R_{ninv}) \cdot C_{out}) \tag{4.11}$$

As mentioned earlier, we will use slightly modified equations from [41] to model our input-referred noise. The two main differences between our circuit and the one in [41] are:

1. We do not have precharge transistors for nodes X_p and X_n, and
2. We have an extra equalization transistor M_{eq} which we can ignore its noise contribution as this transistor is off for the majority of the relevant time period.

The modified input-referred-noise equations are:

$$\sigma_{in}^2 = \sigma_{nin}^2 + \sigma_{ninv-pinv}^2 + \sigma_{pchg}^2 \tag{4.12}$$

$$\sigma_{nin}^2 = \frac{2 \cdot kT \cdot \gamma}{C_X \cdot F} \tag{4.13}$$

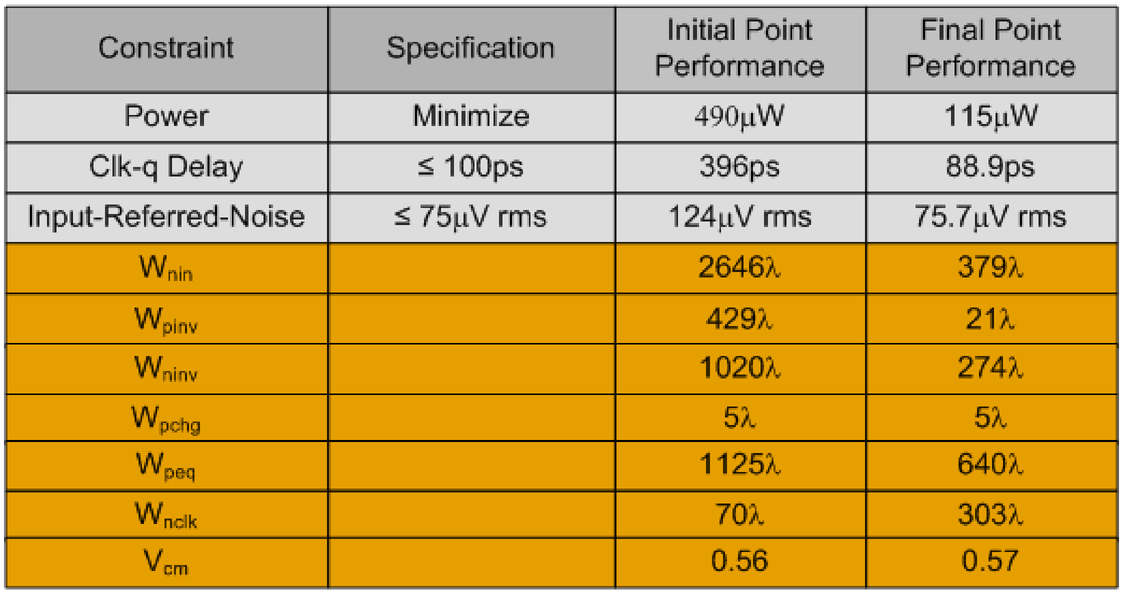
$$\sigma_{ninv-pinv}^2 = \frac{kT \cdot \gamma}{2 \cdot C_X \cdot F^2 \cdot H} + \frac{kT \cdot \gamma \cdot C_{out}}{8 \cdot C_X^2 \cdot F^2 \cdot H^2} \tag{4.14}$$

$$\sigma_{pchg}^2 = \frac{kT}{2 \cdot C_{out} \cdot F^2} + \frac{kT}{2 \cdot C_X \cdot F^2 \cdot H} + \frac{kT \cdot C_{out}}{8 \cdot C_X \cdot F^2 \cdot H^2} \tag{4.15}$$

$$F = \frac{2 \cdot V_{th} \cdot \rho}{-1 + 2 \cdot \sqrt{(V_{dd} - V_{th}) \cdot (V_{cm} - V_{th}) \cdot \rho}} \tag{4.16}$$

$$H = \frac{V_{dd} - V_{cm}}{V_{th}} \cdot \frac{\frac{W_{ninv}}{L_{ninv}} \cdot V_{th}^2}{\frac{W_{ninv}}{L_{nin}} \cdot (V_{cm} - V_{th})^2 - \frac{W_{ninv}}{L_{ninv}} \cdot V_{th}^2} \tag{4.17}$$

where $\rho = \frac{W_{nin}/L_{nin}}{W_{nclk}/L_{nclk}}$, k is the boltzmann constant, T is the temperature, γ is the transistor noise factor, V_{cm} is the input common-mode-voltage, V_{th} is transistor threshold voltage, V_{dd} is the power supply, C_X is the total capacitance at node C_{Xp} or C_{Xn}, and C_{out} is the total output capacitance. The expressions for F and H were further simplified from [41] by ignoring the body effect of the transistors.



Constraint	Specification	Initial Point Performance	Final Point Performance
Power	Minimize	490μW	115μW
Clk-q Delay	≤ 100ps	396ps	88.9ps
Input-Referred-Noise	≤ 75μV rms	124μV rms	75.7μV rms
W_{nin}		2646λ	379λ
W_{pinv}		429λ	21λ
W_{ninv}		1020λ	274λ
W_{pchg}		5λ	5λ
W_{peq}		1125λ	640λ
W_{nclk}		70λ	303λ
V_{cm}		0.56	0.57

Table 4.3: Simulated performance comparison between the initial and the final designs for the StrongARM comparator circuit.

A rough explanation of what these equations mean is as follows. Equation (4.12) sums up the input-referred-noise contributions from three main sources: the input transistor, the two cross-couple transistors, and the precharge transistors. Each thermal noise source is of the usual form kT/C in equations (4.13)-(4.15) and the additional factors in these equations are there to help refer the noises back to the input node.

Results

Table 4.3 compares the StrongARM comparator simulated performance for both the initial design point and the final design point. The initial point did not meet both the *clk-q* and input-referred-noise specifications while the final design point was very close to the limits of both specifications.

Figure 4.16-4.18 show the optimizer solution paths from the initial design point ($\lambda = 0$) to the final design point ($\lambda = 1$). We can see that from the ($\lambda = 0$) point in

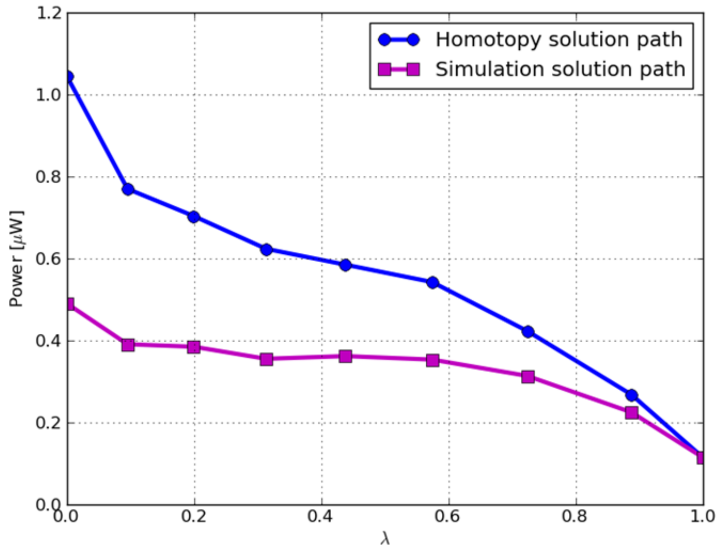


Figure 4.16: Power solution path from the initial design point ($\lambda = 0$) to the final design point ($\lambda = 1.0$) for the StrongARM comparator design.

Figure 4.18 that despite our efforts to write accurate equations for the input-referred-noise, the simulated noise was almost a factor of 2 higher than what our design equation predicted.

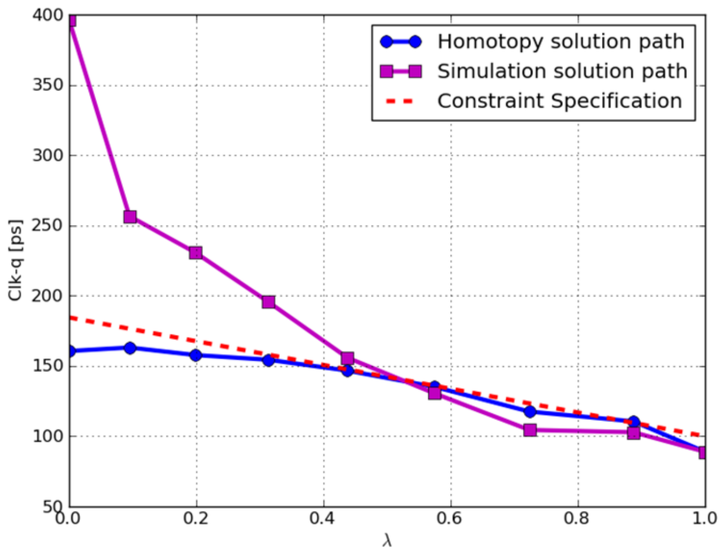


Figure 4.17: *Clk-q* delay solution path from the initial design point ($\lambda = 0$) to the final design point ($\lambda = 1.0$) for the StrongARM comparator design.

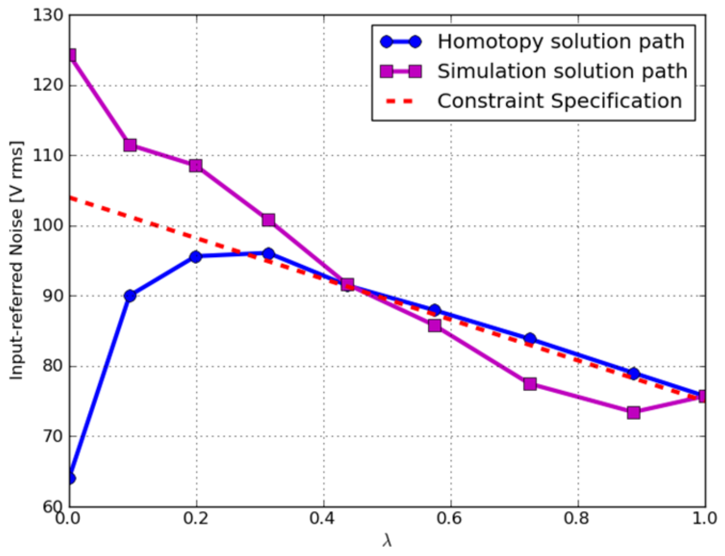


Figure 4.18: Input-referred-noise solution path from the initial design point ($\lambda = 0$) to the final design point ($\lambda = 1.0$) for the StrongARM comparator design.

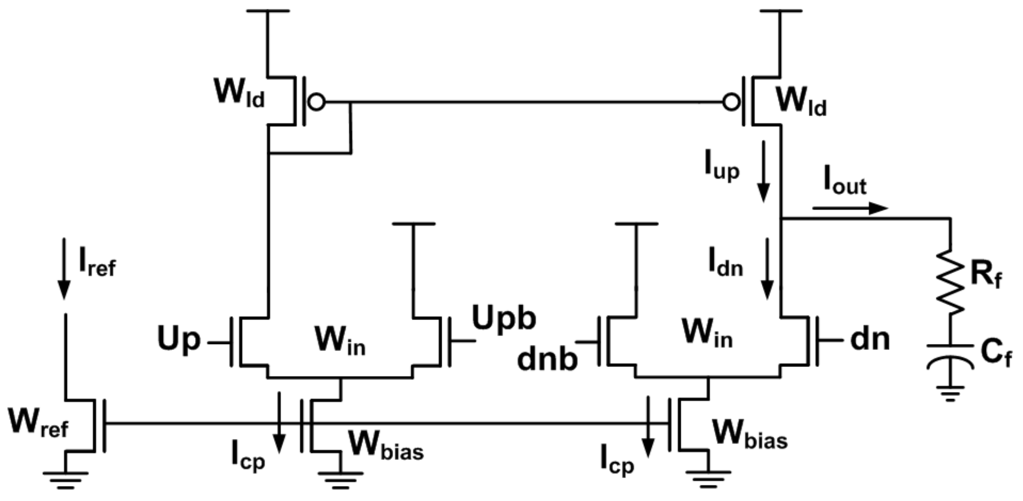


Figure 4.19: Charge-Pump and Filter.

4.4 Charge-Pump

Our next design example is a 5-variable charge-pump and filter circuit shown in Figure 4.19. This example will show how to include statistical constraints in the optimization. We want to constrain the charge-pump mismatch and offset statistically. Specifying statistical constraints in our optimizer is really not very different from specifying any other constraints — you still write design equations to summarize the statistical constraints and perform Monte Carlo simulations for the simulation problem. This charge-pump will be used inside a PLL that will be described in Chapter 6.

Problem Description

The design objective for our charge-pump and filter circuit is to minimize its power consumption subject to the following constraints:

- **Up-Down Current Mismatch**. The combined systematic mismatch and 1-σ random mismatch should be less than 10%.
- **Charge Injection Current Overshoot**. The amount of current injected into the output node at each cycle due to charge-injection when the input switches

should result in less than 30% overshoot compared to nominal charge-pump current.

- **Input-referred Offset**. The input-referred timing offset between the up and down input signals should be less than 5ps.
- **Voltage Ripple**. Maximum voltage ripple caused by charge-pump mismatch should result in less than -40dBc of spur when this charge-pump is used inside a PLL. Assuming a VCO gain of 10GHz/V, this requires the ripple on the control voltage be smaller than 2mV based on spur equation (6.7) at 1GHz operating frequency.
- **Supply Sensitivity**. The nominal charge-pump current sensitivity to supply noise should be less than 2% per 1% change in supply.
- **PLL Natural Frequency**. When this charge-pump is used inside a PLL, the resulting PLL natural frequency based on the loop filter sizes should be less than $0.05 \cdot \omega_{ref}$ to ensure PLL stability where $\omega_{ref} = 2\pi f_{ref}$ with $f_{ref} = 500$MHz.
- **PLL Damping Factor**. When this charge-pump is used inside a PLL, the resulting PLL damping factor based on the loop filter sizes should be greater than $1/\sqrt{2}$ to ensure that PLL has sufficient phase margin.

The first three constraints — mismatch, current overshoot, and offset — all have the same goal of reducing the overall offset since charge-pump offset can be caused by current or timing mismatch or charge-injection. The extra mismatch and charge-injection constraints are there to ensure that the final offset value is not dominated by just one single effect. These three constraints are usually internally imposed to ensure charge-pump functionality.

The voltage ripple and supply sensitivity are usually externally imposed as they can have a significant effect on the jitter performance when used inside a PLL. As the voltage ripple is caused by charge-pump mismatch, its constraint is related to the first three constraints on mismatch, overshoot, and offset. The difference is that the first three constraints are imposed primarily for functionality while the ripple constraint is there for jitter performance reasons.

The last two constraints — PLL natural frequency and PLL damping factor — are there because the sizes of the loop filter have direct effect on these PLL parameters. These two constraints can be externally or internally imposed depending on the application.

The five design variables include three transistor widths (W_{in}, W_{ld}, W_{bias}) and the resistor and capacitor in the loop filter (R_f, C_f).

Design Equations

Power is given by a sum of the static currents in the three branches multiplied by the supply voltage:

$$Power_{cp} = (2 \cdot I_{cp} + I_{ref}) \cdot V_{dd} \tag{4.18}$$

where $I_{cp} = (W_{bias}/W_{ref}) \cdot I_{ref}$ is the bias current in both differential pairs and I_{ref} is the reference current.

The current mismatch consists of two components: systematic mismatch $\mu_{\Delta I_{out}}$ and random mismatch $\sigma_{\Delta I_{out}}$. We will assume that systematic mismatch is caused primarily by channel-length modulation of the pMOS current mirror. We will derive random mismatch based on Pelgrom's transistor mismatch model [43].

For the systematic mismatch, we first determine the gate voltage of the pMOS current mirror and then derive the up current (I_{up}) based on long-channel transistor model:

$$V_{cp} = \sqrt{\frac{I_{cp}}{\frac{W_{ld}}{2L_{ld}} \cdot \mu_p C_{ox}}} + V_{th} \tag{4.19}$$

$$I_{up} = I_{cp} \cdot \left(\frac{1 + \lambda_p \cdot (V_{dd} - V_c)}{1 + \lambda_p \cdot V_{cp}} \right) \tag{4.20}$$

where μ_p is the pMOS transistor mobility, C_{ox} is the oxide capacitance, V_{thp} is the transistor threshold voltage, λ_p is the channel-length modulation factor, and V_c is the output bias voltage. The down current is simply equal to the bias current I_{cp}. The systematic mismatch is then given by:

$$\begin{aligned} \mu_{\Delta I_{out}} &= \left| \frac{I_{up} - I_{dn}}{I_{dn}} \right| && (4.21) \\ &= \lambda_p \cdot \left| \frac{V_{dd} - V_c - V_{cp}}{1 + \lambda_p \cdot V_{cp}} \right| && (4.22) \end{aligned}$$

The Pelgrom's model accounts for the threshold voltage (V_{th}) and current gain ($\beta = \frac{W}{2L} \cdot \mu C_{ox}$) variations as follows:

$$\begin{aligned} \sigma_{\Delta V_{th}} &= \frac{0.6V \cdot T_{ox}}{\sqrt{WL}} && (4.23) \\ \sigma_{\Delta \beta} &= \frac{2\%}{\sqrt{WL}} && (4.24) \end{aligned}$$

where T_{ox} is the oxide thickness, W and L are the width and length of the transistor respectively. The drain current variation in each transistor can then be derived as:

$$\sigma^2_{\Delta I_d} = \frac{\sigma^2_{\Delta \beta}}{\beta} \cdot I^2_{cp} + gm^2 \cdot (\sigma^2_{\Delta V_{gs}} + \sigma^2_{\Delta V_{th}}) \tag{4.25}$$

We can find random current mismatch for our charge-pump by applying equations (4.23) - (4.25) to derive current variation in each transistor and then refering and summing them at the output branch. This is given by:

$$\begin{aligned} \sigma^2_{\Delta I_{out}} &= \sigma^2_{\Delta I_{up}} + \sigma^2_{\Delta I_{dn}} && (4.26) \\ \sigma^2_{\Delta I_{up}} &= \sigma^2_{\Delta \beta_{ld}} \cdot I^2_{cp} + gm^2_{ld} \cdot (\sigma^2_{\Delta V_{thld}} + \sigma^2_{\Delta V_{gsld}}) && (4.27) \\ \sigma^2_{\Delta I_{dn}} &= \tfrac{1}{4} \cdot \left(\sigma^2_{\Delta \beta_{bias}} \cdot I^2_{cp} + gm^2_{bias} \cdot \left(\sigma^2_{\Delta V_{thbias}} + \sigma^2_{V_{thref}} + \sigma^2_{\Delta \beta_{ref}} \cdot \tfrac{I_{ref}}{\beta_{ref}} \right) \right) && (4.28) \end{aligned}$$

The total mismatch is then given by:

$$\Delta I_{mismatch} = \mu_{\Delta I_{out}} + \sigma_{\Delta I_{out}} \tag{4.29}$$

Extra current due to charge injection is given by the product of the input gate capacitance and maximum $\frac{dv}{dt}$ of the input voltage signal:

$$\begin{aligned} \Delta I_{inject} &= C_g \cdot \frac{dv}{dt} \\ &= W_{in} \cdot C_{ox} \cdot \frac{Vdd}{t_{rf}} \qquad (4.30) \end{aligned}$$

where t_{rf} is the rise and fall time of the input voltage signal.

To determine the input-referred timing offset, we make use of the fact that the net charge delivered to the output must be zero i.e., $t_{up} \cdot I_{up} = t_{dn} \cdot I_{dn}$. The nominal values for t_{up} and t_{dn} are given by the minimum pulse width (t_{min}) of the up and down voltage signals. Hence, the input-referred offset is given by:

$$\begin{aligned} \sigma_{toffset} &= |t_{up} - t_{dn}| \qquad (4.31) \\ &= t_{min} \cdot \left|1 - \frac{I_{up}}{I_{dn}}\right| \qquad (4.32) \\ &= t_{min} \cdot \Delta I_{mismatch} \qquad (4.33) \end{aligned}$$

For the voltage ripple, our main concern is its effect on the PLL output spur at the reference frequency. This ripple is caused by current mismatch, timing mismatch, and charge-injection in the charge-pump. The charge-pump output current in a PLL locked state due to these charge-pump non-idealities is a periodic signal and will usually have a profile over a reference period similar to that shown in Figure 4.20. For example, suppose there is a current mismatch between the up and down currents, $I_1 = I_{up} - I_{dn}$, representing the mismatched current that is on for a small period of time due to the minimum pulse width of the input up and down signals (t_1). I_2 will represent the smaller down current that is on for t_2 amount of time to cancel out the charge dumped on the control voltage node by the mismatched current.

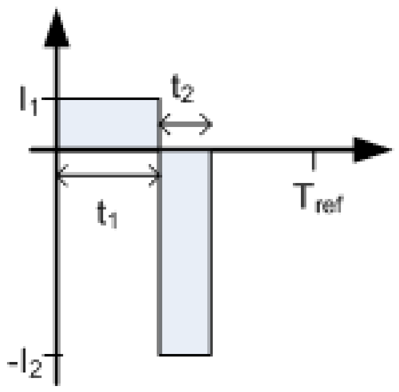


Figure 4.20: Charge-pump output current due to charge-pump non-idealities in a PLL locked state over a reference period. For example, due to current mismatch, $I_1 = I_{up} - I_{dn}$, representing the mismatched current that is on for a small period of time due to the minimum pulse width of the input up and down signals (t_1). I_2 will represent the smaller down current that is on for t_2 amount of time to cancel out the charge dumped on the control voltage node by the mismatched current.

To estimate the ripple, we first decompose the periodic charge-pump output current into a discrete Fourier series:

$$I_{out} = \sum_{k=1}^{\infty} c_k \cdot e^{jk\omega_{ref}t} \tag{4.34}$$

and find the Fourier coefficient at the reference frequency to get the magnitude of the mismatch current. This decomposition can be done separately for each non-ideality effect resulting in three Fourier coefficients (at the reference frequency), one for each mismatch source. These components are given by [49]:

$$c_{1i} = \pi \cdot \Delta I_{mismatch} \cdot I_{cp} \cdot \left(\frac{t_{min}}{T_{ref}}\right)^2 \tag{4.35}$$

$$c_{1t} = 2\pi \cdot \Delta t_{mismatch} \cdot I_{cp} \cdot \left(\frac{t_{min}}{T_{ref}^2}\right) \tag{4.36}$$

$$c_{1c} = \pi \cdot \Delta I_{inject} \cdot I_{cp} \cdot \left(\frac{t_{rise}}{T_{ref}}\right)^2 \tag{4.37}$$

where c_{1i}, c_{1t}, and c_{1c} are the Fourier coefficients associated with current mismatch, timing mismatch, and charge-injection respectively. t_{min} is the minimum pulse width of the up and down signals. t_{rise} is the rise (or fall) time of the up and down signals. T_{ref} is the period of the reference signal. The voltage ripple is then given by:

$$\begin{aligned} \Delta V_{ripple} &= \Delta I_{cp} \cdot Z_f \\ &= (c_{1i} + c_{1t} + c_{1c}) \cdot Z_f \end{aligned} \tag{4.38}$$

where Z_f is the loop filter impedance $= R_f + \frac{1}{j\omega C_f}$.

A change in the supply voltage affects the source and gate voltages of the pMOS transistors hence their V_{gs} will remain the same (assuming the gate node is perfectly coupled to V_{dd}) while their V_{ds} will vary. Consequently, the change in current due to the change in supply is mainly because of the finite output resistance of the pMOS current mirror transistor i.e., $\frac{dI}{d_{Vdd}} = gds_{ld}$ where gds_{ld} is the transistor drain-source conductance. The current sensitivity is then given by:

$$\begin{aligned} S_I &= \frac{dI_{cp}/I_{cp}}{dV_{dd}/V_{dd}} = \frac{dI_{cp}}{dV_{dd}} \cdot \frac{V_{dd}}{I_{cp}} \\ &= gds_{ld} \cdot \frac{V_{dd}}{I_{cp}} \end{aligned} \tag{4.39}$$

The equations for the last two constraints, PLL's natural frequency (ω_n) and damping factor (ζ), use a second-order system to model the PLL. The resulting equations are given by:

$$\omega_n = \sqrt{K \cdot \omega_z} \tag{4.40}$$

$$\zeta = \tfrac{1}{2} \cdot \sqrt{\frac{K}{\omega_z}} \tag{4.41}$$

$$K = \frac{K_{pfd} \cdot K_{vco} \cdot R_f}{N} \tag{4.42}$$

$$\omega_z = \frac{1}{R_f \cdot C_f} \tag{4.43}$$

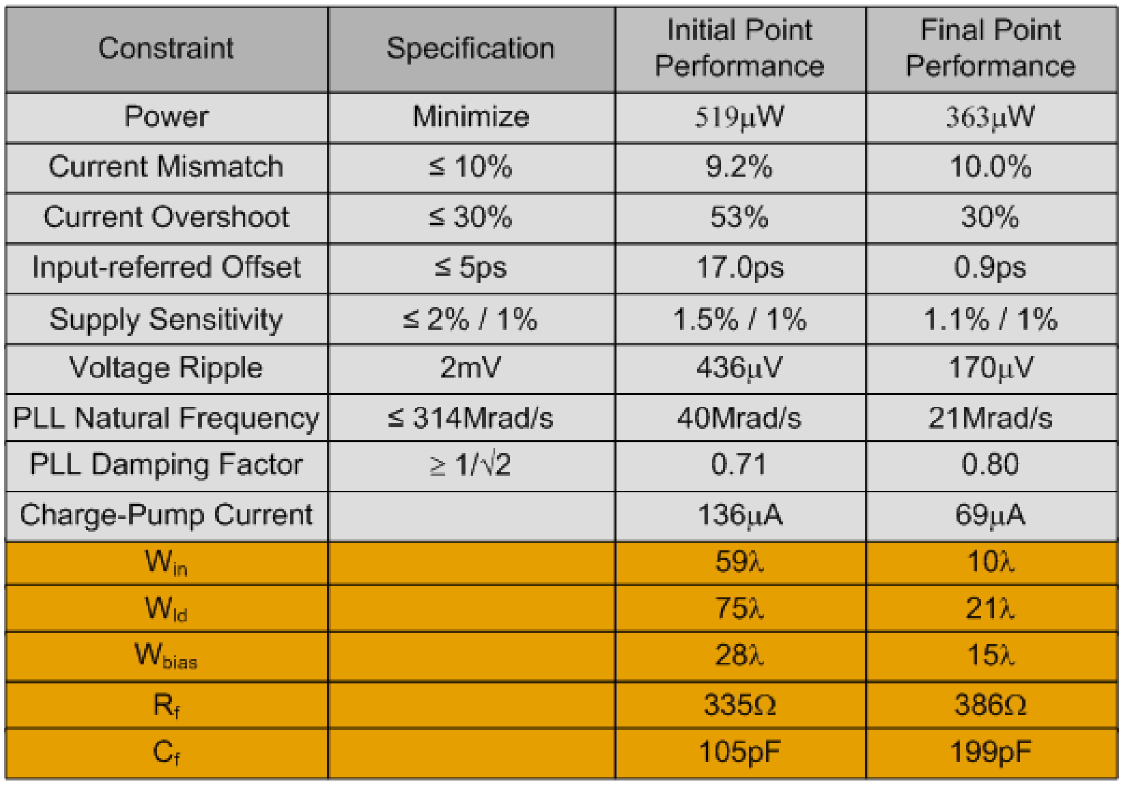
where K_{pfd} is the phase-frequency-detector gain, K_{vco} is the VCO gain, and N is the PLL divider ratio.

Results

Table 4.4 compares the charge-pump and filter simulated performance for both the initial design point and the final design point. The initial design point did not meet the current overshoot and the input-referred offset specifications. Figure 4.21 and 4.22 show the transient waveforms of the up and down current for the initial design and final design respectively. We can see a pronounced overshoot in the down current in both designs (53% in the initial design and 30% in the final design) due to charge-injection through the gate capacitance of the down input transistor. There is no overshoot in the up current as the current signal from the up differential pair is filtered by the pMOS current mirror.

Figure 4.23 compares the charge vs. input arrival time transfer function between the two design points. The initial design needed an offset of 17ps to have zero net charge at the output while the final design only needed 0.9ps.

Figure 4.24 compares the output current histogram between the two design points. Although the final design has smaller systematic mismatch (μ) and smaller random mismatch (σ) than the initial design, the magnitudes of the mismatches relative to the charge-pump current are similar because the final design also has smaller charge-pump current. The total mismatch percentage is around 10% in both cases.



Constraint	Specification	Initial Point Performance	Final Point Performance
Power	Minimize	519μW	363μW
Current Mismatch	≤ 10%	9.2%	10.0%
Current Overshoot	≤ 30%	53%	30%
Input-referred Offset	≤ 5ps	17.0ps	0.9ps
Supply Sensitivity	≤ 2% / 1%	1.5% / 1%	1.1% / 1%
Voltage Ripple	2mV	436μV	170μV
PLL Natural Frequency	≤ 314Mrad/s	40Mrad/s	21Mrad/s
PLL Damping Factor	≥ 1/√2	0.71	0.80
Charge-Pump Current		136μA	69μA
W_{in}		59λ	10λ
W_{ld}		75λ	21λ
W_{bias}		28λ	15λ
R_f		335Ω	386Ω
C_f		105pF	199pF

Table 4.4: Simulated performance comparison between the initial and the final designs for the charge-pump and filter circuit.

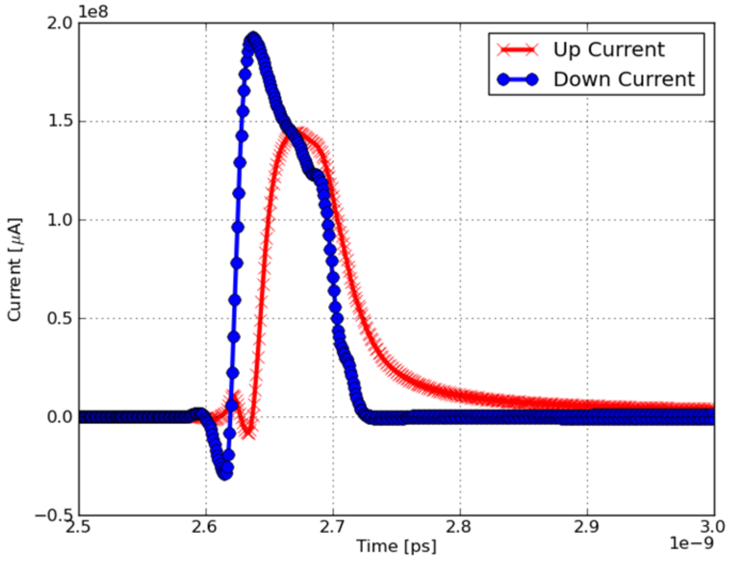


Figure 4.21: Transient up and down currents for the initial design.

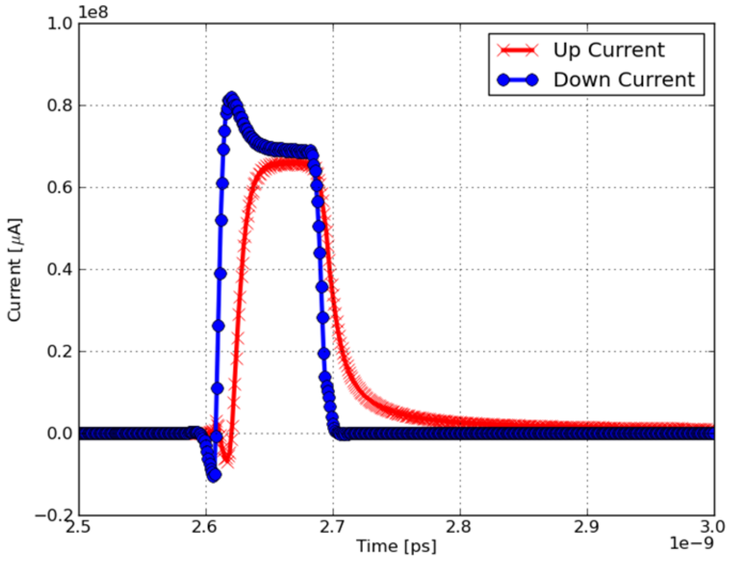


Figure 4.22: Transient up and down currents for the final design.

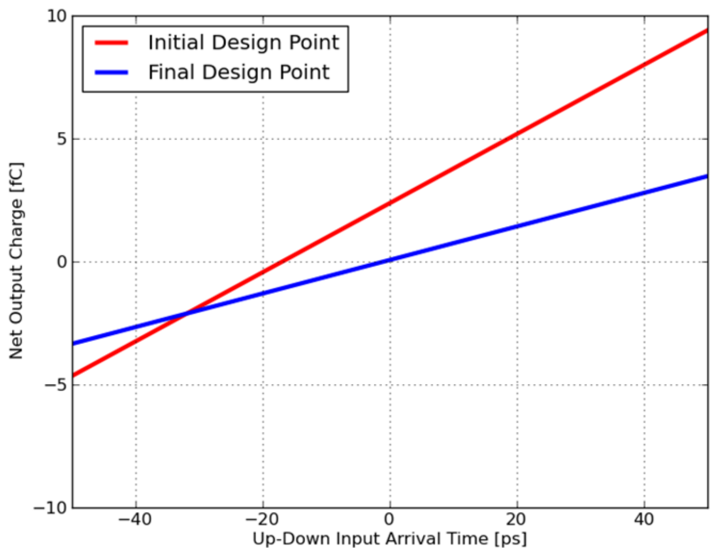


Figure 4.23: Comparison of charge-pump transfer function between initial and final designs.

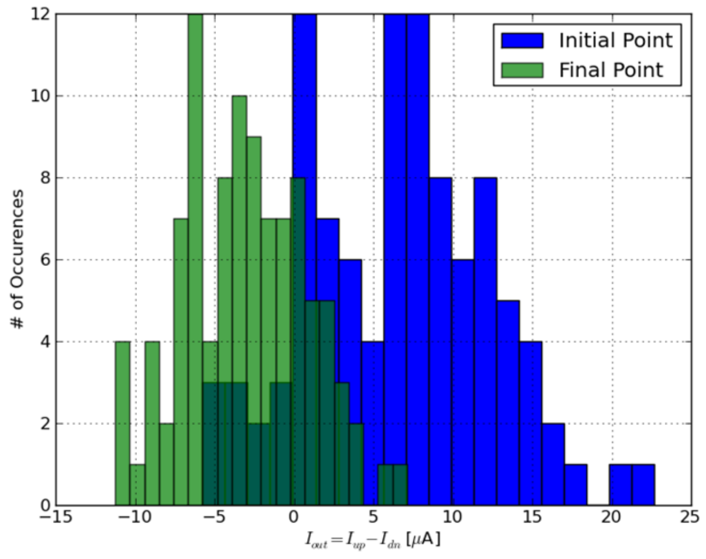


Figure 4.24: Comparison of output current histogram between initial and final designs.

4.5 Discussion

From the design examples that we have presented, we have shown that:

- Designing with our optimizer can be intuitive. We can use rough design equations, formed from our first order understanding of the circuit, to help guide our optimizer. Because these equations have inherent assumptions, they will not perfectly match their corresponding simulation behaviors and it is the optimizer's job to take care of the non-idealities that are not modeled in our equations.

- Our intent-based homotopy optimizer is not a substitute for experienced analog circuit designers as the design problems in equation space can still be quite complex to solve. Rather, the goal of our optimizer is to improve designer's productivity by not requiring the designers to take into account every single non-ideality effect to try to come up with a design solution. Designers still need to rely on their experiences and their expertise to model the circuits in order to generate rough initial sizes.

- Our optimizer can handle infeasible initial design point by applying constraint relaxation technique to the initial design problem. The relaxed amount is gradually driven to zero as we approach the final design problem.

- Some circuit behavior such as input-referred-noise of the StrongARM comparator can be very difficult to model. Even though we wrote very accurate design equations for the input-referred-noise, the resulting initial design still did not meet the noise specification.

- As the charge-pump design example showed, we can incorporate statistical constraints just like any other constraints. The optimization time for these examples will usually be long as we now need to run many Monte Carlo simulations for each sample data point.

Chapter 5

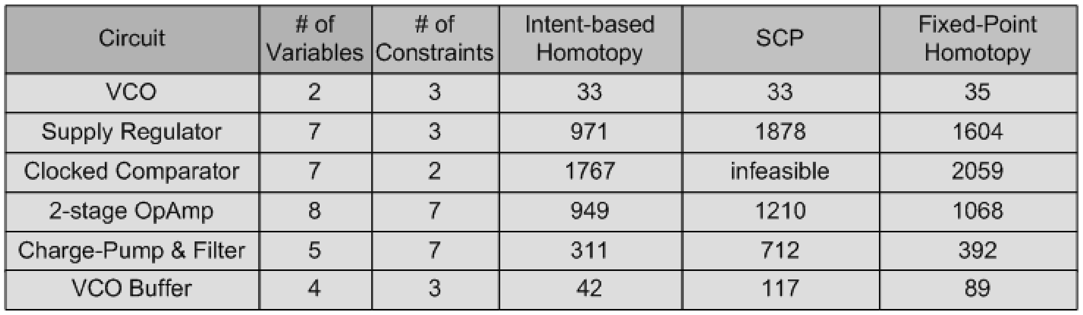
Optimizer Analysis

In this chapter, we will analyze various factors that affect the performance of our optimizer such as the importance of the initial design equations and the scalability of the data-fitting functions used in our optimizer.

5.1 Importance of Initial Design Equations

Having shown *how* our intent-based homotopy optimizer works with several design examples, we would like to also understand *why* it works. Could it be that the initial design equations yielded a very good initial point and we could have just solved the problems using local optimization? Or could it be because of the smooth homotopy interpolation and we could have just used any arbitrary problems that have same initial optimal points as the initial problems in applying the homotopy method?

To answer the first question, we compare our intent-based homotopy optimizer against a local optimizer using the same initial point. We use SCP as our local optimizer reference comparison which is the same one used in the inner loop of our intent-based homotopy optimizer. Columns 4 and 5 in Table 5.1 compares the number of simulations needed in our intent-based homotopy optimizer and the local SCP optimizer. We can see that though our optimizer performs better in many cases, SCP optimizer can sometimes perform equally well. However, the key advantage is that



Circuit	# of Variables	# of Constraints	Intent-based Homotopy	SCP	Fixed-Point Homotopy
VCO	2	3	33	33	35
Supply Regulator	7	3	971	1878	1604
Clocked Comparator	7	2	1767	infeasible	2059
2-stage OpAmp	8	7	949	1210	1068
Charge-Pump & Filter	5	7	311	712	392
VCO Buffer	4	3	42	117	89

Table 5.1: Optimizer performance comparison between intent-based homotopy, SCP, and fixed-point homotopy. The last three columns on the right show the number of simulations required by each optimization algorithm.

our optimizer is also more robust. The SCP optimizer cannot solve all problems; in the StrongARM clocked comparator example, the SCP optimizer got stuck in an infeasible region.

Figure 5.1 shows the solution paths of our local SCP optimizer starting from two different points in the StrongARM comparator optimization and the solution path of our intent-based homotopy optimizer through a Power-Noise-Delay space. The two different starting points for the local optimizer are the point obtained from solving the initial design equations and the point obtained from solving the initial relaxed design equations (using the constraint relaxation techinque described in Section 3.2).

We can see that the local optimizer got stuck in an infeasible region for both cases. Figure 5.2 and 5.3 show 2D projections of the solution path in Power-Delay and Power-Noise spaces respectively. We can see from these two figures that the local optimizer was able to eventually find a design point that meets the *clk-q* specification but was unable to reduce the input-referred-noise less than about 90μV.

Having shown the advantages of homotopy over local optimization, we need to compare our intent-based method to other homotopy approaches. To investigate this, we compare our intent-based homotopy optimizer against a general fixed-point homotopy optimizer using the same starting point. These results are shown in the

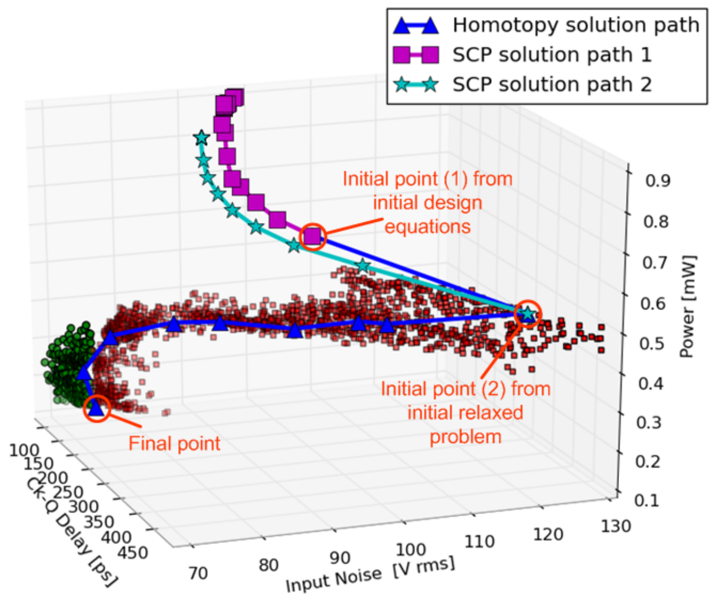


Figure 5.1: Comparison of the solution paths of the local SCP optimizer and our intent-based homotopy optimizer through Power-Delay-Noise space for the StrongARM comparator design. The two initial points (1) and (2) for the local optimizers are from solving the initial design equations and from solving the relaxed initial design equations respectively. Our intent-based homotopy optimizer starts from initial point (1).

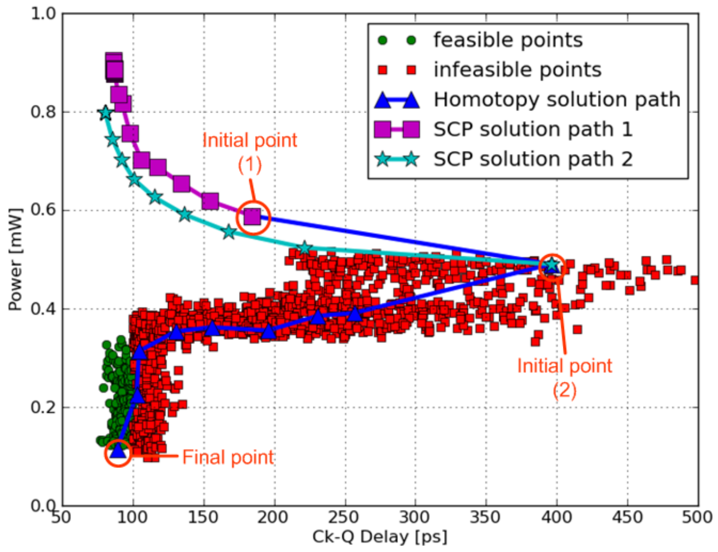


Figure 5.2: 2D projection of the solution paths of the local SCP optimizer and our intent-based homotopy optimizer through Power-Delay space.

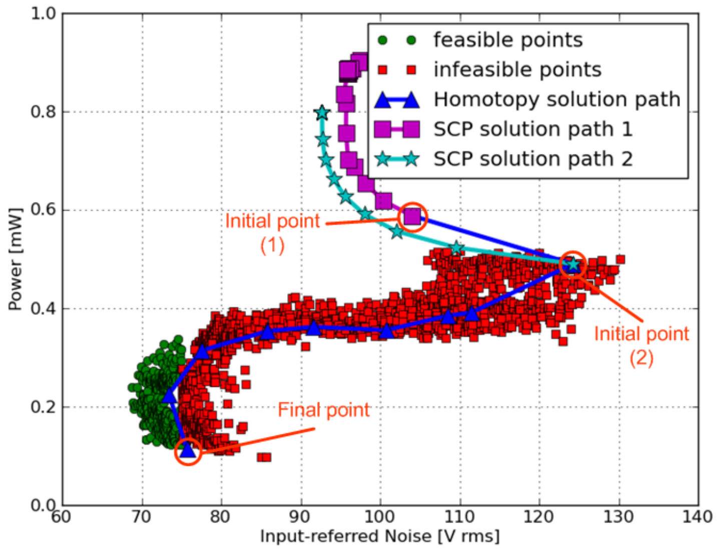


Figure 5.3: 2D projection of the solution paths of the local SCP optimizer and our intent-based homotopy optimizer through Power-Noise space.

last column of Table 5.1. The fixed point homotopy only requires an initial point and not the design equations from the user. We can see that, for our examples, while the fixed-point homotopy optimizer is as equally robust as our optimizer, it generally takes longer to optimize the circuits suggesting that the initial design equations do play a significant role in guiding the optimizer to the final solution.

In summary, while our intent-based homotopy optimizer does not guarantee a globally optimal point, it can be more robust than using a local optimizer and can guide the optimizer more efficiently than using a general fixed-point homotopy technique.

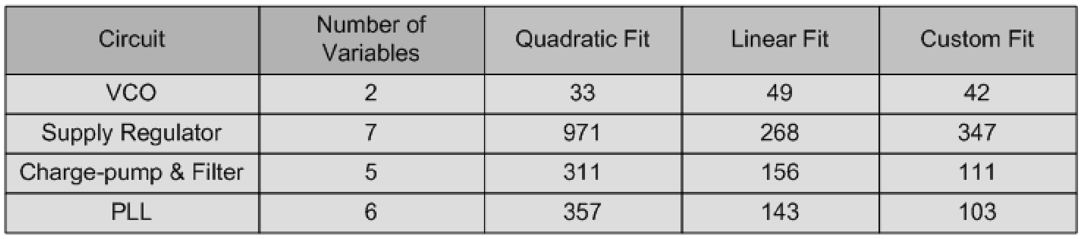
5.2 Optimizer Performance Analysis

In this section, we examine various factors that affect the performance of our optimizer. We first analyze how various different fitting functions used to form the approximate convex problems in the local optimization loop affect the overall computational efficiency of our optimizer. Then we examine how our optimizer would fare if it had a perfect initial design equation.

5.2.1 Computational Efficiency

As most of our optimization time is spent simulating circuits, the number of samples required to create the approximate problem at each local optimization step plays a significant role in the computational efficiency of our optimizer. The standard formulation of the SCP optimization described in the previous section creates and solves a quadratic model. In this section, we use three different types of fitting functions: quadratic, linear and user-assigned to explore this question. The quadratic and the linear fitting functions obviously scale quadratically and linearly with the number of parameters respectively.

The available user-assigned fitting functions are limited to functions that are both convex and scale linearly with the number of parameters. For our experiment, we limit these to linear, inverse and log functions. The user can select from these functions



Circuit	Number of Variables	Quadratic Fit	Linear Fit	Custom Fit
VCO	2	33	49	42
Supply Regulator	7	971	268	347
Charge-pump & Filter	5	311	156	111
PLL	6	357	143	103

Table 5.2: Comparison of the number of simulations taken by the optimizer for different fitting functions.

what they think will best fit their for each objective and constraint functions in their design problem.

We chose these functions since using linear and user-assigned functions might perform better than the original quadratic function because these functions require fewer simulations to create the model. On the other hand, because the quadratic function can provide a better fit to a general surface and will therefore usually has larger trust region, this might result in the optimizer taking less number of iterations and ultimately use less total simulation per each local optimization run.

Table 5.2 shows the comparison of the number of simulations used by each fitting function for several design examples. We can see that using the linear and the user-assigned fitting functions perform better than using the quadratic fitting function. Hence the benefit of simpler model fitting of the linear and the user-assigned functions outweighs the cost of having a larger number of iterations per solution. This can also be seen in Figure 5.4 where we plot the ratio between the number of simulations used by the quadratic fitting function (N_{quad}) to the the number of simulations used by the linear fitting function (N_{linear}). We can see that this ratio scales roughly linearly (or worse) with the number of variables. This data indicates that the quadratic fit does require fewer iterations but that advantage does not strongly depend on the number of variables. Thus, even for small number of variables, using linear fitting function is much more efficient.

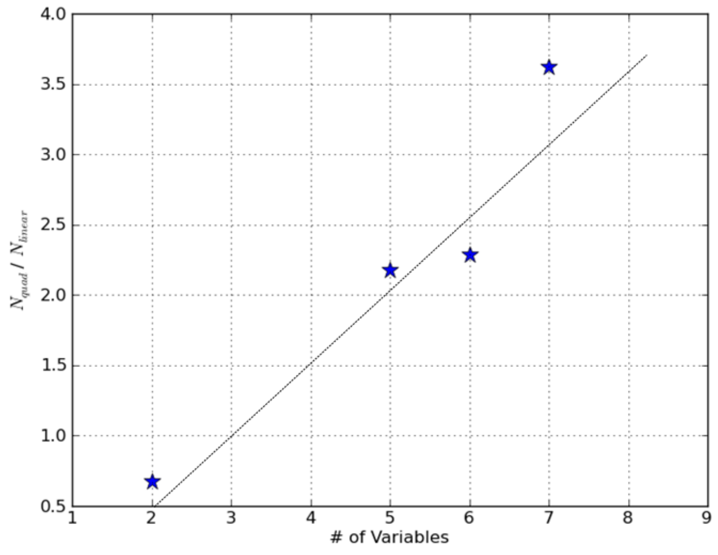


Figure 5.4: Ratio between the number of simulations used by the quadratic fitting function method (N_{quad}) to the number of simulations used by the linear fitting function method (N_{linear}).

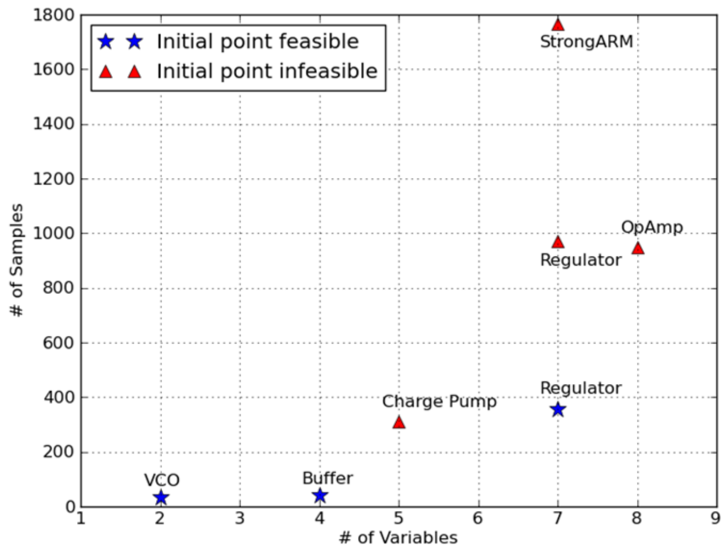


Figure 5.5: Number of samples vs. number of variables categorized by the feasibility of the initial point. These numbers are for optimization done with quadratic fitting functions. The constraints in the regulator example presented in Section 4.2 were also modified to generate an example with feasible initial point.

Regardless of the fitting functions used, the number of simulations needed for each optimization run should not be affected by the number of constraints as the same set of samples are reused to fit each constraint function. However, the difficulty of the constraints does play a significant role. If the constraints are difficult to meet or even initially infeasible by the initial design point, the number of simulations needed may be increased as suggested by Figure 5.5 that shows the number of samples vs. the number of variables categorized by the feasibility of the initial point using quadratic fitting function.

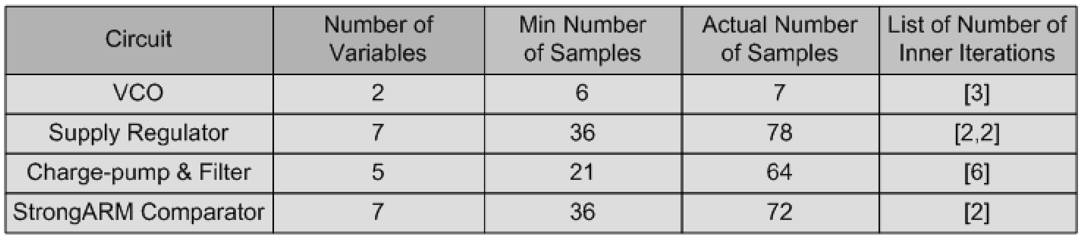
5.2.2 Perfect Initial Design Equations

In this section, we examine what is the best possible performance of our optimizer: what would happen if we had a perfect initial problem to use in our intent-based homotopy for each design problem. Of course we cannot do this directly because the perfect initial design equations would need to model the simulation space perfectly. Hence, figuring out the exact analytical form of these equations is impractical.

Instead, we first optimize each problem once (using intent-based homotopy) to pre-determine the actual solution to the simulation problem and then we reuse this solution and the same simulation problem as our new initial problem for the intent-based homotopy. This new initial simulation problem serves as a proxy for our perfect initial problem. Obviously, solving this artificial perfect initial problem will be slow as it is a simulation problem but its main purpose is to help us investigate how many iterations the optimizer would take in the best case scenario.

In this ideal case, the optimizer should take just one step at the outer level ($\lambda = 0$ to $\lambda = 1$) and one iteration and the inner level. Hence, the ideal number of simulations would just be the number of simulations required to fit the data sample once.

Table 5.3 summarizes the number of simulations needed using this methodology with the quadratic fitting function for several design examples. The "Min Number of Samples" column tells us the least number of samples the optimizer need in an ideal case. We can see from the last column ("List of Number of Inner Iterations") that, for most cases, our optimizer behaved ideally at the outer homotopy loop, taking just one homotopy step. The inner loops, however, usually take a few iterations and this is because the local optimizer needs a few iterations to figure out when it has achieved convergence.



Circuit	Number of Variables	Min Number of Samples	Actual Number of Samples	List of Number of Inner Iterations
VCO	2	6	7	[3]
Supply Regulator	7	36	78	[2,2]
Charge-pump & Filter	5	21	64	[6]
StrongARM Comparator	7	36	72	[2]

Table 5.3: Number of simulations taken by the optimizer with perfect initial problems. The "Min Number of Sample" column indicates the minimum number of samples required by the optimizer to create one approximate convex problem. The "List of Number of Inner Iterations" column gives a list of the number of inner iterations. The size of this list is the number of outer homotopy iterations. For example, the VCO optimization took only one outer iteration, and in this outer iteration, it took three inner iterations.

5.2.3 Summary

From the analyses that we have performed in this chapter, we have found that:

- Our optimizer can be more robust than a local optimizer. The StrongARM comparator design example showed that a local optimizer may get stuck in an infeasible region when given an initially infeasible point whereas the homotopy optimizers were able to find a feasible solution.

- Using linear models in the approximate convex problems in the local optimization loops can lead to a better optimizer performance than using quadratic models. This is because the benefit of needing less number of simulations to fit the linear models outweigh the cost of having a larger number of outer iterations per solution.

- Design equations can play a significant role in guiding the optimizer to the final solution. The comparison of our optimizer with a generic fixed-point homotopy showed that we can use design equations to more efficiently guide the optimizer. Our experiments with perfect initial equations showed that more accurate equations can also help improve the optimizer performance by reducing the total number of simulations required. These results are encouraging, since it shows that when the design equations are close, our method requires very few additional simulations to validate the result. But it also shows that checking the convergence of the solution is, in general, expensive. This fact will become important when we look at the hierarchical optimization in the next Chapter.

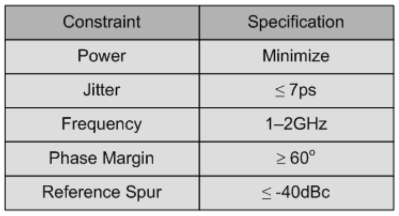
Chapter 6

Hierarchical Optimization

As the performance of the flat optimization technique described in Chapter 3 scales with both the number of design variables and the simulation time, it can become impractical to design circuits with long simulation times or with large numbers of parameters. An 18-variable PLL design took 519 simulations or roughly 70 hours to optimize using this flat optimization technique (with linear fitting function). Clearly, we need a more efficient way to design large circuits if we want to give circuit designers opportunities to explore different architectures. In this chapter, we show, through a PLL example, how we can design large circuits much more efficiently using hierarchical optimization.

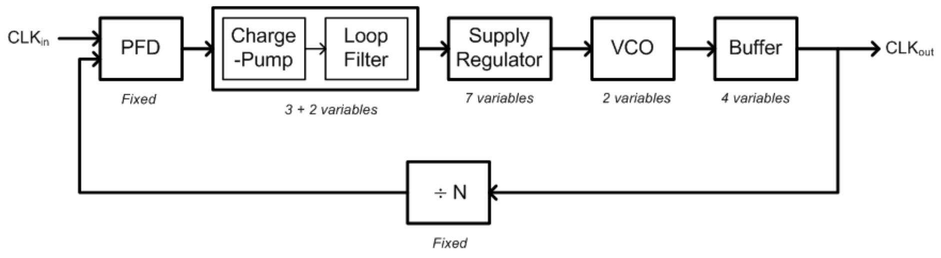
6.1 Methodology

As with any general hierarchical optimization schemes, we first divide our top-level circuit into several sub-blocks and design the top-level circuit using certain properties of the sub-blocks as the design variables. In our approach, the sub-block design specifications are the sub-block properties that we use as the top-level variables. Each sub-block is responsible for determining their own transistor sizes by performing their own intent-based homotopy optimizations as discussed in the previous chapters.



Constraint	Specification
Power	Minimize
Jitter	$\leq$ 7ps
Frequency	1–2GHz
Phase Margin	$\geq 60^{o}$
Reference Spur	$\leq$ -40dBc

(a) PLL Constraints and Specifications



(b) 18-variable PLL Block Diagram

Figure 6.1: 18-variable PLL Specifications and Block Diagram

Consider an 18-variable PLL optimization problem as shown in Figure 6.1(a) where we want to minimize its power subject to jitter, frequency range, reference spur, and phase margin constraints. We may divide the PLL into several sub-blocks as shown in Figure 6.1(b) consisting of phase-frequency detectors (PFD), charge-pump and loop filter, supply regulator, voltage-control oscillator (VCO), VCO buffer, and frequency divider. The sizes of the transistors in the PFD and the frequency divider blocks are fixed in our example so these blocks do not have any design variables. The rest of the sub-blocks are optimized using intent-based homotopy under their own design specification constraints. We optimized these sub-blocks (except for the VCO buffer) in Chapter 4.

6.1.1 Classifying Sub-Block Specifications

To determine which sub-block specifications to use as top-level design variables, we first classify the sub-block specifications based on their relationship to the top-level specifications as follows:

- ***Internal specifications***. These are constraints that exist to ensure the functionality of the sub-blocks when used inside a top-level block.

  Although changing these sub-block specifications may have some effects on the top-level specifications, the main job of these constraints are to ensure the internal functionality of the sub-blocks.

  For example, in the supply regulator block, we need to ensure that the supply-regulator, when used inside a PLL, does not degrade the stability of the PLL so we need to set its bandwidth constraint to be much higher than the PLL loop bandwidth. This minimum regulator bandwidth constraint is an internal specification to the regulator sub-block.

  The maximum output current specification of the same supply regulator block is also an internal specification because this constraint exists to ensure that it can drive the VCO.

- ***Pass-through specifications***. These are constraints that are passed through directly from top-level specifications.

  As the PLL frequency range is directly controlled by the VCO frequency range, the top-level frequency range specification is considered a pass-through specification which is passed directly down to the VCO sub-block

- ***Property specifications***. These are specifications that are also used as optimization variables at the top-level.

  VCO jitter specification is an example of a property specification where it is also used as a top-level parameter due to its effect on the top-level jitter specification.

  Regulator supply sensitivity is also another example where we expect the regulator supply sensitivity to have direct effect on the top-level jitter performance.

Table 6.1 summarizes the different types of specifications for the sub-blocks used in our PLL example with the property specifications highlighted.

Constraint	Specification	Type	Reason
Power	Minimize	Objective	
Input Offset	$\leq 5\text{ps}$	Internal	Ensure PLL Robustness
Current Mismatch	$\leq 10\%$	Internal	Help reduce offset
Current Overshoot	$\leq 30\%$	Internal	Help reduce offset
Supply Sensitivity	$\leq 1\% / 1\%$	Property	Affect PLL jitter
Voltage Ripple	$\leq 2\text{mV}$	Property	Affect PLL spur
ω_n	$\leq 0.05\omega_{ref}$	Internal	Ensure PLL stability
ζ	$\leq 1/\sqrt{2}$	Property	Affect PLL phase margin

(a) Charge-pump & Filter Specification Types

Constraint	Specification	Type	Reason
Power	Minimize	Objective	
Eff. Internal Amp BW	$\geq 1.5\text{GHz}$	Internal	Ensure PLL stability
Regulator BW	$\geq 500\text{MHz}$	Internal	Ensure PLL stability
Output Current	$\geq 120\mu\text{A}$	Internal	Ensure low drop-out
Supply Sensitivity	$\leq -24\text{dB}$	Property	Affect PLL jitter

(b) Regulator Specification Types

Constraint	Specification	Type	Reason
Power	Minimize	Objective	
Frequency	1–2GHz	Pass-through	Affect PLL frequency
Phase Noise	$\leq -95\text{dBC/Hz}$	Property	Affect PLL jitter

(c) VCO Specification Types

Constraint	Specification	Type	Reason
Power	Minimize	Objective	
Gain	≥ 4	Internal	Ensure full-rail signal to divider
Delay	$\leq 55\text{ps}$	Internal	2nd order effect on phase margin
Supply Sensitivity	$\leq 165\text{ps/V}$	Property	Affect PLL jitter

(d) VCO Buffer Specification Types

Table 6.1: Specification Types for the PLL sub-blocks

6.1.2 Top-Down Optimization

There are two main ways of performing the optimization at the top level. One approach would be to solely rely on the top-level equations provided by the designers (see Section 6.1.4) and optimize the top-level problem in the equation space. The sub-blocks are then optimized once using the final top-level design variables as their design specifications. The main benefit of this approach is that we can significantly reduce the total optimization time as our hierarchical optimization approach now neither depends on the number of top-level parameters nor the top-level simulation time. The main drawback is that our accuracy is now dependent on the accuracy of the top-level equations. Note that with this technique, we only remove the dependencies on the number of parameters and simulation time at the top level but not at the sub-block level because we still use intent-based homotopy to optimize the sub-blocks. However, the sub-blocks' optimization times are usually much faster making the potential reduction in total optimization time substantial.

An alternative approach would be to rely on homotopy, just like in the flat optimization approach, to help move the solution from the designer's equation space to one in simulation space. The main benefit is that we will have simulation-accurate optimization result, just like in the flat approach. However, the optimization time will take much longer as we still depend on both the number of top-level parameters and the top-level simulation time. Each top-level sample point that the optimizer takes is also now more complex.

While a sample point in a flat optimization is equivalent to a simulation of the circuit, a sample point in our hierarchical optimization in this alternative approach consists of a top-level optimization and the sub-block optimizations. Because the performance objective and constraint functions are evaluated at the top-level, a sample point for our top-level optimization will require a flat top-level simulation.

In our example, this means running a whole PLL simulation to determine the actual PLL's power, jitter, reference spurs and phase margin for each sample point.

Each top-level PLL simulation will in turn need the flat optimization variables — the underlying transistor sizes of various sub-blocks in the PLL — as input to the simulator. To determine these flat optimization variables, we optimize the sub-blocks using the current values of the top-level variables as the sub-block specifications. Hence, each hierarchical simulation consists of a flat top-level simulation and all sub-block optimizations.

Although we use intent-based homotopy on the first optimization runs of the sub-blocks, subsequent sub-block optimizations may rely on local optimization technique because the specifications should change gradually as we move the top-level design from ideal space to simulation space.

6.1.3 Hierarchical Optimization Algorithm

Let's briefly summarize the steps of our hierarchical optimization algorithm:

1. Identify sub-block properties based on designer's top-level circuit model equations.
2. Reformulate top-level optimization problem using these properties as top-level optimization variables.
3. Optimize the top-level problem either in the equation space or with intent-based homotopy as described in the previous section. If intent-based homotopy is used to drive the top-level optimization, goto Step 4. Otherwise we are done.
4. For each top-level sample required in Step 3, optimize the sub-blocks with either intent-based homotopy or local optimization using the solution of the top-level problem as their specifications, and run a top-level simulation.
5. Repeat from Step 3 until done.

6.1.4 PLL Top-Level Model

To help decide which sub-block specifications are property specifications, we need to know how the sub-block specifications affect the top-level performance functions. This is done in our approach using the circuit designer's model equations. This section describes the top-level equations for the PLL's performance constraints using the sub-block specifications as its design parameters.

Our PLL has five constraint functions: power, jitter, reference spurs, frequency range and phase margin. Since the frequency range is a pass-through specification, we do not need to write a top-level model for it and just pass this specification down directly to the VCO sub-block.

The six top-level parameters (the sub-block property specifications highlighted in Table 6.1) are the supply sensitivity, voltage ripple and the damping factor in the charge-pump and filter block (cp), the supply sensitivities in regulator (reg) and VCO buffer block (buf), and the phase noise of the VCO block (measured at 1MHz offset frequency). We expect the sub-blocks' supply sensitivities and the VCO phase noise to primarily affect jitter, the voltage ripple to primarily affect the reference spur, and the damping factor to primarily affect the phase margin. The model equations for the four non-pass-through constraint functions below reflect this expectation.

- ***Power***. Power is a sum of the individual sub-block powers which also include a fixed-component due to the PFD and the divider. For the VCO sub-block, we know based on Leeson's model [34] that power is inversely proportional to phase noise (PN). For the other variable sub-blocks, we model their powers to be inversely proportional to their supply sensitivities (SS). We use this model to help reduce the number of top-level parameters. The power expression is thus given by:

$$\begin{aligned} Power_{PLL} &= Power_{fixed} + Power_{vco} + Power_{reg} + Power_{cp} + Power_{buf} \\ &= Power_{fixed} + \frac{Kp_1}{PN_{VCO}} + \frac{Kp_2}{SS_{cp}} + \frac{Kp_3}{SS_{reg}} + \frac{Kp_4}{SS_{buf}} \end{aligned} \tag{6.1}$$

where Kp_{1-4} are inverse proportionality constants that can be estimated by fitting the sub-block data.

- ***Jitter***. To determine the contribution to the output jitter from each noise source, we multiply each noise source power spectral density by its noise transfer function to the output to get the output noise spectral density due to each noise source. Then we can use Wiener-Khinchine Theorem [12] to obtain the output jitter due to each noise source:

$$Jitter^2_{srci \to out} = \int N_{srci}(f) \cdot H_{srci \to out}(f) df \tag{6.2}$$

where $N_{srci}(f)$ is the i^{th} noise source, $H_{srci \to out}(f)$ is the noise transfer function from the i^{th} noise source to the output, and $Jitter^2_{srci \to out}$ is the square of the output jitter contribution due to the i^{th} noise source.

Assuming the noise sources are independent, we can sum their contribution to the total output jitter in a root-sum-square fashion.

$$Jitter^2_{out} = \sum Jitter^2_{srci \to out} \tag{6.3}$$

Since we only have access to the top-level parameters and not the noise sources themselves, we need to write the individual jitter equation (6.2) in terms of the VCO phase noise and other sub-blocks' supply sensitivities.

For the VCO sub-block, we want to write its PLL jitter contribution in terms of its phase noise measured at 1MHz offset frequency. To do this, we assume that the VCO output phase noise is dominated by white noise sources (such as device thermal noise or white supply noise) and thus has a $1/f^2$ shape in its power spectrum. We can then write its jitter equation as:

$$\begin{aligned} Jitter^2_{vco \to out} &= \int PN_{vco}(f) \cdot H_{vco \to out}(f) df \\ &= PN_{1MHz} \cdot \int \left(\frac{1MHz}{f} \right)^2 \cdot H_{vco \to out}(f) df \end{aligned} \tag{6.4}$$

where PN_{1MHz} is the VCO phase noise measured at 1MHz offset frequency and is used as a top-level parameter, $H_{vco\to out}(f)$ is the noise transfer function from the VCO output to the PLL output. We can simplify this equation further by assuming that the integral term is constant with respect to our design parameters.

For the charge-pump, regulator and buffer sub-blocks, we assume that the main noise source in each of these blocks is the supply noise and thus write their PLL jitter contribution in terms of their supply sensitivities. To do this, we first decompose each of the noise transfer function into a multiplication of two transfer functions: one from the supply noise to the block output and the other from the block output to the PLL output. For example, the jitter equation for the regulator sub-block is given by:

$$\begin{aligned} Jitter^2_{reg\to out} &= \int N_{vddreg}(f)\cdot H_{vddreg\to out}(f)df \\ &= \int N_{vddreg}(f)\cdot\{H_{vddreg\to reg}(f)\cdot H_{reg\to out}(f)\}\,df \qquad (6.5) \end{aligned}$$

where $N_{vddreg}(f)$ is the supply noise source, $H_{vddreg\to reg}(f)$ is the (square of) regulator's supply sensitivity and is a top-level parameter, and $H_{reg\to out}$ is the noise transfer function from the regulator output to the PLL output. We can then simplify this equation further by assuming that the supply noise source and the noise transfer function are constant with respect to our design parameters. Similar expressions can be derived for the charge-pump and buffer sub-blocks.

We can now use equation (6.3) to combine these individual jitter contribution to get the total output jitter:

$$\begin{aligned} Jitter^2_{out} &= Jitter^2_{fixed} + Jitter^2_{vco\to out} + Jitter^2_{reg\to out} \\ &\quad + Jitter^2_{cp\to out} + Jitter^2_{buf\to out} \\ &= Jitter^2_{fixed} + Kj_1\cdot PN_{1MHz} + Kj_2\cdot SS^2_{cp} + Kj_3\cdot SS^2_{reg} \\ &\quad + Kj_4\cdot SS^2_{buf} \qquad (6.6) \end{aligned}$$

where Kj_{1-4} are the proportionality constants, PN_{1MHz} is the VCO phase noise at 1MHz offset, $SS_{cp/reg/buf}$ are the supply sensitivities for the charge-pump, regulator and VCO buffer respectively.

- ***Reference Spur***. The non-idealities in the charge-pump cause periodic ripples on the control voltage of the VCO which translate into a spur at the reference frequency offset in the PLL output power spectral density. Assuming a narrow band FM modulation, the expression for reference spur is given by [49]:

$$Spur_{ref} = 20 \cdot log \left(\frac{K_{vco} \cdot \Delta v_{ripple}}{2 \cdot \omega_{ref}} \right) \tag{6.7}$$

  where K_{vco} is the VCO gain, Δv_{ripple} is the voltage ripple on the control signal and is also a top-level design parameter, and $\omega_{ref} = 2\pi f_{ref}$ ($f_{ref} = 500$MHz).

- ***Phase Margin***. For a second order system, the phase margin is related to the damping factor by:

$$PM = \arccos \left(\frac{1}{(2\zeta)^2} \right) \tag{6.8}$$

  where ζ is the damping factor which is also a top-level design parameter.

Circuit	Flat approach (with linear fit)	Hierarchical approach with only equations at top-level	Hierarchical approach with homotopy at top-level		
			Quadratic Fit	Linear Fit	Custom Fit
PLL	519 sims / 70 hrs	0 sims / 2 hrs	357 sims / 2.3 days	143 sims / 23.1 hrs	103 sims / 16.8 hrs

Table 6.2: Number of top-level simulations and total optimization time comparison between the flat approach, hierarchical approach with only equations at the top level and hierarchical approach with homotopy at the top-level. Note that the number of simulations do not include the sub-blocks data but the optimization times do.

6.2 Results

Table 6.2 compares the PLL optimization times between the flat optimization approach and the two hierarchical approaches. The flat optimization technique with linear fitting function took about 70 hours. The hierarchical approach with the top-level problem relying solely on their equations took less than 2 hours. The hierarchical approach with homotopy took about 17 hours in the best case (custom fitting function) and about 55 hours in the worst case (quadratic fitting function). From these results, we can see that the first hierarchical approach that relies solely on the equations at the top-level had a significant improvement over the other two approaches: 35x faster than the flat approach and 8.5x faster than the other hierarchical approach.

The question is how much accuracy did we sacrifice from using the faster hierarchical approach? Table 6.3 summarizes the initial and final design performances of the homotopy-based approach. The initial performance also represents the performance of the equation-only approach (at the top level). This initial point was already feasible so the optimizer focused on decreasing the power at the expense of more jitter. The solution paths for power and jitter can be seen in Figures 6.2 and 6.3. We can see that using the more accurate approach only provided about 5% improvement. Clearly, the benefit of the faster approach outweigh the cost in its accuracy, at many stages in the design. In particular, it allows the designer to “quickly” get a feeling for the overall optimization leading to faster debug and tuning loops.

		Specification	Initial	Final
Constraints	Power	Minimize	1.08mW	1.03mW
	Jitter	≤ 7ps	5.6ps	7.0ps
	Frequency	1–2GHz	0.3–3.9GHz	0.3–3.9GHz
	Phase Margin	≥ 60°	75°	77°
	Reference Spurs	≤ -40dBc	-43.9dBc	-44.1dBc
Top-Level Variables	VCO Spec	Phase Noise	-91.8dBc/Hz	-93.1dBc/Hz
	Regulator Spec	Supply Sensitivity	-20.5dB	-15.5dB
	Buffer Spec	Supply Sensitivity	0.17ps/mV	0.23ps/mV
	Charge-Pump Specs	Supply Sensitivity	5% / 1%	4.7% / 1%
		Voltage Ripple	2.0mV	2.8mV
		Damping Factor	0.78	0.7

Table 6.3: Simulated performance comparison between the initial and the final designs for the PLL circuit.

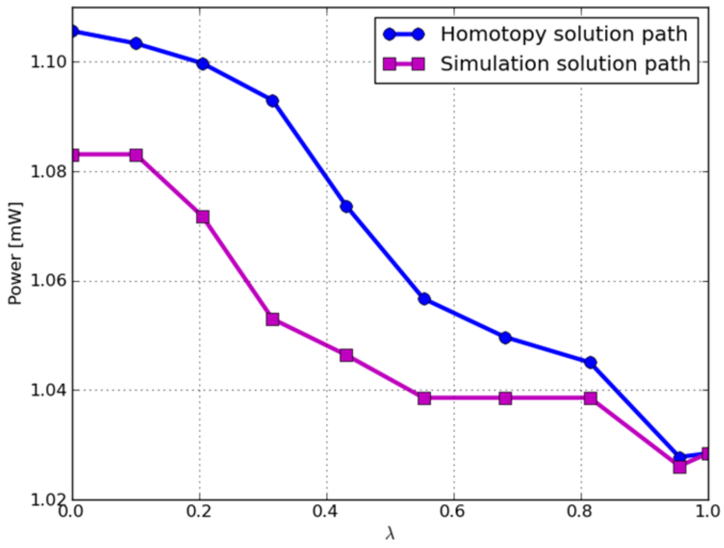


Figure 6.2: Power solution path from the initial design point ($\lambda = 0$) to the final design point ($\lambda = 1.0$) for the PLL design example.

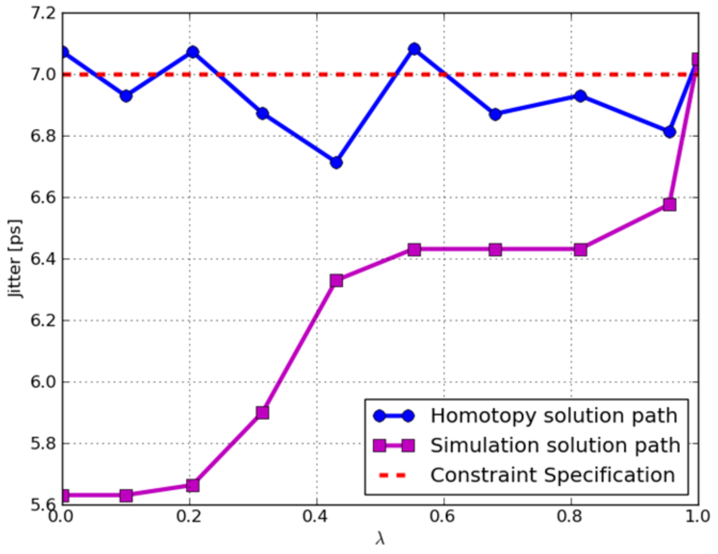


Figure 6.3: Jitter solution path from the initial design point ($\lambda = 0$) to the final design point ($\lambda = 1.0$) for the PLL design example.

The reason why there is very little improvement in the more accurate approach is because while the optimization algorithm of the faster approach is decoupled from top-level simulations, it still requires the sub-blocks to be simulated in the sub-blocks' optimization runs. If the final sub-block design specifications (that are top-level variables) in the faster approach are not unreasonable, the sub-block optimization ensures that the PLL is already well-designed. This is in contrast with an equation-only flat optimization approach where its accuracy will be entirely dependent on the accuracy of the equations and the initial point you obtain can be very far away from the final optimized point. This can be seen by looking at the simulated performance of the initial point in the examples in Chapter 4.

To understand why the more accurate hierarchical approach is slow, Table 6.4 provides the break-down of the number of simulations and optimization times spent inside each block (using quadratic fitting function at the top-level). The "Initial #

Block	# of Variables	Property Specifications	Initial # of Simulations	Total # of Simulations	Time Spent [Hours]
PLL	6 (top-level)			143	19.0
CP + Filter	5	SS_{cp}, Δv_{ripple}, ζ	228	378	2.1
Supply Regulator	7	SS_{reg}	280	2016	1.3
VCO	2	PN_{1MHz}	38	191	0.5
VCO Buffer	4	SS_{buf}	40	60	0.2
				Total	23.1

Table 6.4: Breakdown of the number of simulations and optimization time for the PLL design example using linear fitting function.

of Simulations" column gives the number of simulations spent in initially optimizing each sub-block using intent-based homotopy. These numbers also represent the number of simulations used in the faster hierarchical approach where the sub-blocks were optimized only once. Subsequent calls to sub-block optimizations utilize local optimization as explained in Section 6.1.2. We can see that 82% of the time is spent simulating the top-level PLL.

6.3 Benefits and Limitations

Besides the runtime benefit mentioned in the earlier section, performing optimization hierarchically is also more intuitive and more in line with how a circuit designer would tackle a big block like a PLL. In a traditional approach to a PLL design, circuit designers would first make sure that the individual sub-blocks like the PFD, VCO, charge-pump, etc. work well according to their specifications first before putting them together to be used in a more time-consuming set of top-level PLL simulations.

The question that often arises during a traditional PLL design approach is how to set the design specifications of the sub-blocks. Since we first ask the designers to categorize the different sub-block specifications, this forces the designer to think carefully about the relationship between the sub-block performance and the top-level performance. The sub-block optimizers will spend more time focusing on the objective and constraints that have direct impact on the top-level performance while ensuring that the sub-block is functional.

One of the main current limitations of our hierarchical implementation is that we cannot use a multi-valued variable in our top-level equations because sub-block specifications can only be single-valued. For example, the top-level jitter contribution from the VCO in our PLL example is more accurately written in terms of the VCO's phase noise spectrum instead of a phase noise value at a certain offset. We had to make some assumptions about the shape of the phase noise spectrum in order to write our equations.

Chapter 7

Conclusions

The main contributions that we have made in this work include:

1. Investigating applying homotopy algorithms to circuit optimization problems while making use of designers' equations to help guide the optimizer. This intent-based homotopy approach was faster than a generic homotopy approach or a local optimization approach and more robust than local optimization. Furthermore, our approach also allow designers to improve their productivity as they can use rough design equations to come up with initial sizes for the circuits and not have to worried about all the unimportant non-ideality effects.

2. We explored hierarchical extensions of our flat approach to optimize larger circuits and found, for our example, that we can rely solely on equations at the top-level to optimize our circuits while we use the intent-based homotopy approach to optimize the underlying sub-blocks.

7.1 Future Work

As can be seen in our design examples, the optimization times for our optimizer are in general still quite slow compared to equation-based approach. This is because the basis of our research started off from the simulation-based optimization approaches' point of view and then we asked ourselves how we can improve upon these approaches, i.e., how we can make them faster. And we did find that we can make them faster but, unfortunately, no where near as fast as equation-based approaches.

The reason why simulation-based optimization approaches cannot be fast is because there is a factor affecting the optimization time that the optimizer cannot control: circuit simulation time. If a circuit takes many minutes or hours to simulate, it is almost impossible to make this approach anywhere near as fast as equation-based approaches.

This raises an interesting question for future work on whether we could do better by starting from equation-based optimization approaches' point of view and then tackle their weaknesses which are mainly ease of use and accuracy. Researches in symbolic analysis and generation, as mentioned in Chapter 2, attempt to address these issues to some extent by trying to automatically generate equations for you. However, their main drawback is that they are mainly applicable to linear circuits.

One potential interesting way of overcoming this drawback is to first figure out the linear intent of the circuit and apply the above technique to this linear part of the circuit. In our design methodology group, we believe that almost all circuits have a linear intent, but it may be in some other transformed domains rather than the direct voltage or current domains [30]. For example, the PLL takes in a voltage input and produces a voltage output, and the relationship between those two voltages are highly non-linear but that is not the main intent of the PLL. After all, it is a *phase* locked loop, so its intent lies primarily in the phase domain. Hence, if you view the PLL from a phase perspective, then the PLL circuit is linear. If these linear intents can be leveraged in some ways by the symbolic analysis and generation tools, then we can make these tools more broadly applicable and useful to a larger community of circuit designers.

Another interesting future work direction would be to investigate various ways to speed up the performance of hierarchical optimization. There are two main approaches to doing this. First, with the top-level homotopy approach, we can have designers create better top-level models so that the top-level homotopy loop will take less number of simulations.

Second, instead of relying solely on the top-level equations, we can try to make the underlying simulation engine faster which can be especially beneficial for design problems with time-consuming simulations like our PLL example. This can be done through use of behavioral top-level models. However, this method relies heavily on the designer's knowledge of the circuit to ensure that the resulting behavioral models are reasonably accurate compared to real simulation. This is the approach taken in [46, 61].

Bibliography

[1] N. M. Alexandrov, J. E. Dennis, R. M. Lewis, and V. Torczon. A trust-region framework for managing the use of approximation models in optimization. *Structural Optimization*, 15:16–23, February 1998.

[2] E. Alon, J. Kim, S. Pamarti, K. Chang, and M. Horowitz. Replica compensated linear regulators for supply-regulated phase-locked loops. *IEEE Journal of Solid-State Circuits*, 41:413–424, February 2006.

[3] J. April, F. Glover, J. P Kelly, and M. Laguna. Practical introduction to simulation optimization. In *Proceedings of the 2003 Winter*, volume 1, pages 71–78, December 2003.

[4] S. Boyd and L. Vandenberghe. *Convex optimization.* Cambridge University Press, 2004.

[5] W. Chen and G. Shi. Implementation of a symbolic circuit simulator for topological network analysis. In *IEEE Asia Pacific Conference on Circuits and Systems*, pages 1368–1372, December 2006.

[6] D. M. Colleran, C. Portmann, A. Hassibi, C. Crusius, S. S. Mohan, S. Boyd, T. H. Lee, and M. Hershenson. Optimization of phase-locked loop circuits via geometric programming. In *IEEE Custom Integrated Circuits Conference*, pages 377– 380, September 2003.

[7] J. L. Dawson, S. P. Boyd, M. Hershenson, and T. H. Lee. Optimal allocation of local feedback in multistage amplifiers via geometric programming. *IEEE*

Transactions on Circuits and Systems I: Fundamental Theory and Applications, 48(1):1–11, January 2001.

[8] M. DeGrauwe and W. Sansen. A synthesis program for operation amplifiers. *IEEE International Solid-State Circuits Conference*, XXVII:18–19, February 1984.

[9] F. El-Turky and E. E Perry. BLADES: an artificial intelligence approach to analog circuit design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(6):680–692, June 1989.

[10] F.V. Fernandez, A. Rodriguez-Vazquez, and J.L. Huertas. Interactive AC modeling and characterization of analog circuits via symbolic analysis. *Analog Integrated Circuits and Signal Processing*, 1(3), November 1991.

[11] C. Fleury. Sequential convex programming for structural optimization problems. In *Optimization of Large Structural Systems*, pages 531–553, 1993.

[12] William A. Gardner. *Introduction To Random Processes*. Mcgraw-Hill (Tx), 2 sub edition, January 1990.

[13] G. Gielen, G. Debyser, K. Lampaert, F. Leyn, K. Swings, G. Der Van Plas, W. Sansen, D. Leenaerts, P. Veselinovic, and W. van Bokhoven. An analogue module generator for mixed analogue/digital asic design. *International Journal of Circuit Theory and Applications*, 23(4):269–283, July 1995.

[14] G. G. E. Gielen and R. A. Rutenbar. Computer-aided design of analog and mixed-signal integrated circuits. *Proceedings of the IEEE*, 88(12):1825–1854, December 2000.

[15] G. G. E. Gielen, H. C. C. Walscharts, and W. M. C. Sansen. ISAAC: a symbolic simulator for analog integrated circuits. *IEEE Journal of Solid-State Circuits*, 24(6):1587–1597, December 1989.

[16] G. G. E. Gielen, H. C. C. Walscharts, and W. M. C. Sansen. Analog circuit design optimization based on symbolic simulation and simulated annealing. *IEEE Journal of Solid-State Circuits*, 25(3):707–713, June 1990.

[17] R. Haaglund. *An optimization-based approach to efficient design of analog circuits*. PhD thesis, Linkping Studies in Science and Technology, 2006.

[18] R. Harjani, R. A. Rutenbar, and L. R. Carley. OASYS: a framework for analog circuit synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(12):1247–1266, December 1989.

[19] J. P. Harvey, M. I. Elmasry, and B. Leung. STAIC: an interactive framework for synthesizing CMOS and BiCMOS analog circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(11):1402–1417, November 1992.

[20] M. Hershenson. Design of pipeline analog-to-digital converters via geometric programming. In *IEEE International Conference on Computer Aided Design*, pages 317–324, November 2002.

[21] M. Hershenson. CMOS analog circuit design via geometric programming. In *Proceedings of the American Control Conference*, volume 4, pages 3266–3271, July 2004.

[22] M. Hershenson, S. P. Boyd, and T. H. Lee. GPCAD: a tool for CMOS op-amp synthesis. In *IEEE international conference on Computer-aided design*, pages 296–303, 1998.

[23] M. Hershenson, S. P. Boyd, and T. H. Lee. Optimal design of a CMOS op-amp via geometric programming. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(1):1–21, January 2001.

[24] M. Hershenson, A. Hajimiri, S. S. Mohan, S. P. Boyd, and T. H. Lee. Design and optimization of LC oscillators. In *IEEE International Conference on Computer-Aided Design*, pages 65–69, 1999.

[25] M. Hershenson, S. S. Mohan, S. P. Boyd, and T. H. Lee. Optimization of inductor circuits via geometric programming. In *Proceedings of 36th Annual Design Automation Conference*, pages 994–998, 1999.

[26] N. C. Horta and J. E. Franca. Automatic synthesis of data conversion systems using symbolic techniques. In *Proceedings of the 38th Midwest Symposium on Circuits and Systems*, volume 2, pages 877–880 vol.2, August 1995.

[27] J. H. Huijsing, R. J. van de Plassche, and W. M. C. Sansen. *Analog circuit design: operational amplifiers, analog to digital convertors, analog computer aided design.* Springer, 1993.

[28] Cadence Design Systems Inc. Virtuoso neocircuit: Circuit sizing and optimization, May 2009.

[29] M. Jeeradit, J. Kim, and M. Horowitz. Intent-leveraged optimization of analog circuits via homotopy. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 1614–1619, March 2010.

[30] J. Kim, M. Jeeradit, B. Lim, and M. Horowitz. Leveraging designer's intent: A path toward simpler analog CAD tools. In *IEEE Custom Integrated Circuits Conference*, pages 613–620, September 2009.

[31] H. Y. Koh, C. H. Sequin, and P. R. Gray. OPASYN: a compiler for CMOS operational amplifiers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(2):113–125, February 1990.

[32] M. Krasnicki, R. Phelps, R. A. Rutenbar, and L. R. Carley. MAELSTROM: efficient simulation-based synthesis for custom analog cells. In *Proceedings of the 36th annual IEEE Design Automation Conference*, pages 945–950, 1999.

[33] H. Lamure and D. Michelucci. Solving geometric constraints by homotopy. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):28–34, March 1996.

[34] D. B. Leeson. A simple model of feedback oscillator noise spectrum. *Proceedings of the IEEE*, 54(2):329–330, February 1966.

[35] T. Y. Li and X. Wang. Solving real polynomial systems with real homotopies. *Mathematics of Computation*, 60(202):669–680, April 1993.

[36] B. Liu, Y. Wang, Z. Yu, L. Liu, M. Li, Z. Wang, J. Lu, and F. V. Fernandez. Analog circuit optimization system based on hybrid evolutionary algorithms. *VLSI Journal Integration*, 42(2):137–148, February 2009.

[37] D. Ma, G. Shi, and A. Lee. A design platform for analog device size sensitivity analysis and visualization. In *IEEE Asia Pacific Conference on Circuits and Systems*, pages 48–51, December 2010.

[38] T. McConaghy and G. Gielen. Globally reliable Variation-Aware sizing of analog integrated circuits via response surfaces and structural homotopy. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(11):1627–1640, November 2009.

[39] F. Medeiro, F. V Fernandez, R. Dominguez-Castro, and A. Rodriguez-Vazquez. A statistical optimization-based approach for automated sizing of analog cells. In *IEEE international conference on Computer-aided design*, pages 594–597, 1994.

[40] R. C. Melville, L. Trajkovic, S. -C. Fang, and L. T. Watson. Artificial parameter homotopy methods for the DC operating point problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(6):861–877, June 1993.

[41] P. Nuzzo, F. De Bernardinis, P. Terreni, and G. Van der Plas. Noise analysis of regenerative comparators for reconfigurable ADC architectures. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 55(6):1441–1454, July 2008.

[42] E. S. Ochotta, R. A. Rutenbar, and L. R. Carley. Synthesis of high-performance analog circuits in ASTRX/OBLX. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(3):273–294, March 1996.

[43] M. J. M. Pelgrom, A. C. J. Duinmaijer, and A. P. G. Welbers. Matching properties of MOS transistors. *IEEE Journal of Solid-State Circuits*, 24(5):1433–1439, October 1989.

[44] V. M. Perez, J. E. Renaud, and L. T. Watson. Homotopy curve tracking in approximate interior point optimization. *Optimization and Engineering*, 10(1):91–108, March 2008.

[45] R. Phelps, M. Krasnicki, R. A. Rutenbar, L. R. Carley, and J. R. Hellums. ANACONDA: robust synthesis of analog circuits via stochastic pattern search. *Proceedings of the IEEE Custom Integrated Circuits*, pages 567–570, 1999.

[46] R. A. Rutenbar, G. G. E. Gielen, and J. Roychowdhury. Hierarchical modeling, optimization, and synthesis for System-Level analog and RF designs. *Proceedings of the IEEE*, 95(3):640–669, March 2007.

[47] S. J. Seda, M. G. R. Degrauwe, and W. Fichtner. A symbolic analysis tool for analog circuit design automation. In *IEEE International Conference on Computer-Aided Design*, pages 488–491, November 1988.

[48] G. Shi and X. Meng. Variational analog integrated circuit design via symbolic sensitivity analysis. In *IEEE International Symposium on Circuits and Systems*, pages 3002–3005, May 2009.

[49] K. Shu and E. Sanchez-Sinencio. *CMOS PLL synthesizers: analysis and design.* Springer, 2005.

[50] K. Swings and W. Sansen. DONALD: a workbench for interactive design space exploration and sizing of analog circuits. In *Proceedings of the conference on European design automation*, pages 475–479, 1991.

[51] C. Toumazou and C. A. Makris. Analog IC design automation: part i. automated circuit generation: new concepts and methods. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(2):218–238, February 1995.

[52] J. C. Vital and J. E. Franca. Synthesis of high-speed A/D converter architectures with flexible functional simulation capabilities. In *IEEE International Symposium on Circuits and Systems*, volume 5, pages 2156–2159, May 1992.

[53] P. Wambacq, F. V. Fernandez, G. Gielen, and W. Sansen. Efficient symbolic computation of approximated small-signal characteristics. *IEEE Journal of Solid-State Circuits*, pages 461–464, May 1994.

[54] L. T. Watson. Computational experience with the ChowYorke algorithm. *Mathematical Programming*, 19(1):92–101, December 1980.

[55] L. T. Watson. Theory of globally convergent Probability-One homotopies for nonlinear programming. *SIAM Journal on Optimization*, 11(3):761, 2001.

[56] L. T. Watson, S. C. Billups, and A. P. Morgan. ALGORITHM 652: HOMPACK: a suite of codes for globally convergent homotopy algorithms. *ACM Transactions on Mathematical Software*, 13(3):281–310, September 1987.

[57] H. Xu, G. Shi, and X. Li. Hierarchical exact symbolic analysis of large analog integrated circuits by symbolic stamps. In *16th Asia and South Pacific Design Automation Conference*, pages 19–24, January 2011.

[58] G. Yu and P. Li. Yield-aware analog integrated circuit optimization using geostatistics motivated performance modeling. *IEEE International Conference on Computer-Aided Design*, pages 464–469, November 2007.

[59] Q. Yu and C. Sechen. A unified approach to the approximate symbolic analysis of large analog integrated circuits. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 43(8):656–669, August 1996.

[60] W. Zangwill. *Pathways to solutions, fixed points, and equilibria.* Prentice-Hall, 1981.

[61] J. Zou, D. Mueller, H. Graeb, and U. Schlichtmann. A CPPLL hierarchical optimization methodology considering jitter, power and locking time. *Proceedings of the 43rd annual Design Automation Conference*, pages 19–24, June 2006.