A Framework for Analog Circuit Optimization

by

Piotr Mitros

Submitted to the Department of Electrical Engineering and Computer
Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2004 [June 2004]

© Piotr Mitros, MMIV. All rights reserved.

The author hereby grants to MIT permission to reproduce and
distribute publicly paper and electronic copies of this thesis document
in whole or in part.

Author ....
Department of Electrical Engineering and Computer Science
February 17, 2004

Certified by..

L. Råfael Reif
Professor
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students
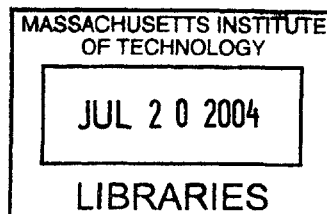
A Framework for Analog Circuit Optimization

by

Piotr Mitros

Submitted to the Department of Electrical Engineering and Computer Science
on February 17, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

This thesis presents a system for optimization of analog circuit topologies and component values. The topology is optimized using simulated annealing, while the component values are optimized using gradient descent. Local minima are avoided and constraints are kept through the use of coordinate transformations, as well as the use of default starting points for component values. The system is targeted for use in 3D integrated circuit design. The architecture is extendable, and is designed to eventually include capabilities for automated layout and mixed-signal design.

Thesis Supervisor: L. Rafael Reif
Title: Professor

Acknowledgments

I would like to thank Professor Rafael Reif for his patience with me as I moved into his group, and for his support of this project. I would also like to thank Professor Gerald Jay Sussman for his valuable consultations on the topic of this thesis, and his continued guidance in years past.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

We present a system used for optimizing analog integrated circuit topologies and component values. The system differs from existing circuit generation and layout programs in a number of key areas. First, this system requires minimal additional training on the part of the user, since its usage is similar to that of tools familiar to engineers. Second, the system scales well to typical-sized circuits. Finally, the system is targeted primarily at 3D integrated circuit design, which presents a unique set of challenges beyond those inherent in traditional 2D design. The eventual goal of the project is to create a CAD system capable of partially automating the design and layout of 3D mixed signal systems.

Simulated annealing is used for topology optimization, while a numerically efficient form of gradient descent is used for component value optimization. Constraints are implemented and local minima are avoided by the use of a combination of coordinate transformations, default component values, and carefully chosen objective functions.

1.1.1 Motivation

3D integrated circuits consist of a stack of wafers, each with a layer of devices, with dense vertical interconnects. There exist a number of processes for developing 3D

13

integrated circuits. Depending on the process, the vertical dimensions of the inter-connects may be comparable to the horizontal dimensions of the interconnects. As a result, there is no significant penalty due to wires going from wafer to wafer. In the case of digital circuits, this may be exploited to significantly reduce interconnect length, and therefore increase performance or reduce power consumption. An au-tomated system for 3D digital circuit layout was developed by Shamik Das [3] and demonstrated a 28-51% reduction in wire length.

We believe 3D IC technology can achieve significant gains in the analog and mixed signal domains as well. However, we believe the major gains are more likely to come from access to multiple types of devices, rather than simply from tighter layout. As an example, a software radio 3D IC would likely include an RF die for the front-end, an analog die for the ADC, a digital layer for DSP/network-layer processing, as well as possibly a power device layer for driving an antenna or for power conversion. Due to the nature of 3D IC processes, it is feasible to have fine-grained distribution of components to the optimal layer within a small section of the circuit. Thus, in addition to shorter wires, we can gain benefits from from 3D IC in several ways.

First, within individual components, we may be able to benefit from the variety of devices. In designing an operational amplifier, we can choose to use high r_0 devices in the VAS in order to avoid cascodes, therefore providing large output swing. At the same time, we can use faster, lower-capacitance devices in the input stage, in order to move the secondary poles out.

Second, we can have high coupling between analog and digital circuits. For in-stance, in the software radio case, the digital logic may be able to monitor ADC distortion based on known properties of the signal, and provide feedback that adjusts bias levels of the ADC to reduce this distortion.

Finally, surprisingly, we may actually be able to achieve economic gains through the use of 3D IC technology. Especially in the mixed signal domain, we may be able to achieve circuits with equivalent performance to traditional ones, but using primarily significantly simpler, cheaper processes. For instance, the original Pentium processor was implemented on a BiCMOS process, although it used very few bipolar transistors.

One of the major design goals for the Pentium II was to move to a significantly more economic pure-CMOS process. Using 3D IC technology, we could place most of the design on a low-cost CMOS dies, while placing the few, specialized components on a single BiCMOS, EEPROM, or otherwise specialized layer. Even in high-performance designs, significant portions of the circuit are not in the critical path. Those we could move to a more mature process, with, again, significantly reduced cost and improved yield.

We must also pay attention to distribution between layers in order to avoid wasted die space, and therefore high cost — in most processes, the die dimensions cannot vary between layers. As a result, in a chip with one layer that is considerably more populated than another, it may make sense to shift non-critical components between layers in order to reduce die size. In general, when shifting components between layers, it makes sense to reevaluate topology — for instance, a cascode may not be necessary on a layer with higher Early voltage.

The eventual goals of the project are to design a tool chain capable of end-to-end 3D IC mixed signal design and layout, and to design a circuit demonstrating the level of performance increase from 3D IC over conventional 2D technologies. In this paper, we present a system for optimizing component placement between layers, as well as the choice of topology, and of component values.

1.1.2   Design Constraints

One goal of this tool is to be very practical. A number of circuit optimizers or generators require a significant amount of training on the part of the user, since they rely on interfaces and file formats that the general engineering community is unfamiliar with. A number of systems must be given large amounts of additional knowledge about the specific topologies they optimize, and as a result, inputing large numbers of topologies is impractical ([4], [5]). This system, in contrast, uses a variant on the SPICE file format, which most engineers are familiar with. It is capable of operating with no additional knowledge about the circuit, beyond what would be available in a SPICE file (although the system is capable of using additional information, if available). Thus,

while some of the system's more advanced features require learning details specific to the system, the core functionality is available with minimal additional training, and there is a smooth learning curve to the more advanced functionality.

In spite of significant research, automatic analog circuit generation and layout programs still create circuits significantly inferior to those designed by high-caliber analog engineers. Our tool allows engineers to manually design or to place arbitrary design constraints on arbitrary portions of the circuit. As a result, the tool can be used to automate the design of non-critical portions of the circuit, leaving the critical portions to the engineer, and so integrates well with existing industry design flow.

This system is also designed to scale to larger circuits than many typical academic systems.

However, to achieve these goals, we tackle a much more constrained problem than the one tackled by most automated circuit design tools. Specifically, most automated tools can generate arbitrary topologies. Ours, however, merely optimizes topologies based on components designed and provided by engineers. The system cannot create fundamentally new circuits, although it may compose existing ones in interesting ways.

1.1.3 Paper Organization

Chapter 2 introduces the architecture of the system, describes practical usage, and gives an overview of the file formats. Chapter 3 describes the simulated annealing algorithm used for discrete topology optimization. It requires knowledge of the data structures introduced in chapter 2. Chapter 4 presents the component value optimization algorithm. The coordinate transformations in chapter 4 require knowledge of chapter 2, but otherwise, chapter 4 stands alone. Chapter 5 gives an overview of the test setup of the system. Finally, chapter 6 concludes with areas of future research and enhancement to the system.

Chapter 2

System Architecture

2.1 Overview

The system represents circuits as a hierarchy of components. As an example of the representation used, the hierarchy for one operational amplifier is shown in figure 2-1. This representation contrasts with those used in traditional automatic circuit generation programs in that most traditional simulated annealing systems view circuits in terms of connections between components, rather than as a hierarchy [7] (although a few systems with hierarchical representations exist [5]). As a result, they can represent arbitrary or nearly-arbitrary circuits, the vast majority of which do not work. Our hierarchical representation has the property that nearly every possible circuit works, although the circuits vary in performance. This property is due to the requirement of additional input from the user, in the form of pre-designed components from which to build hierarchies. The property that nearly every generated system works allows us to scale to significantly larger systems.

Each circuit element has associated with it a vector of continuous values. For instance, in the case of a transistor, this might correspond to the gate length and width. In the case of a compound component like a current mirror, this would correspond to the properties of all of its subcomponent elements (although the correspondence would not necessarily be one-to-one).

The program runs in batch mode. The input to the program is a set of files de-
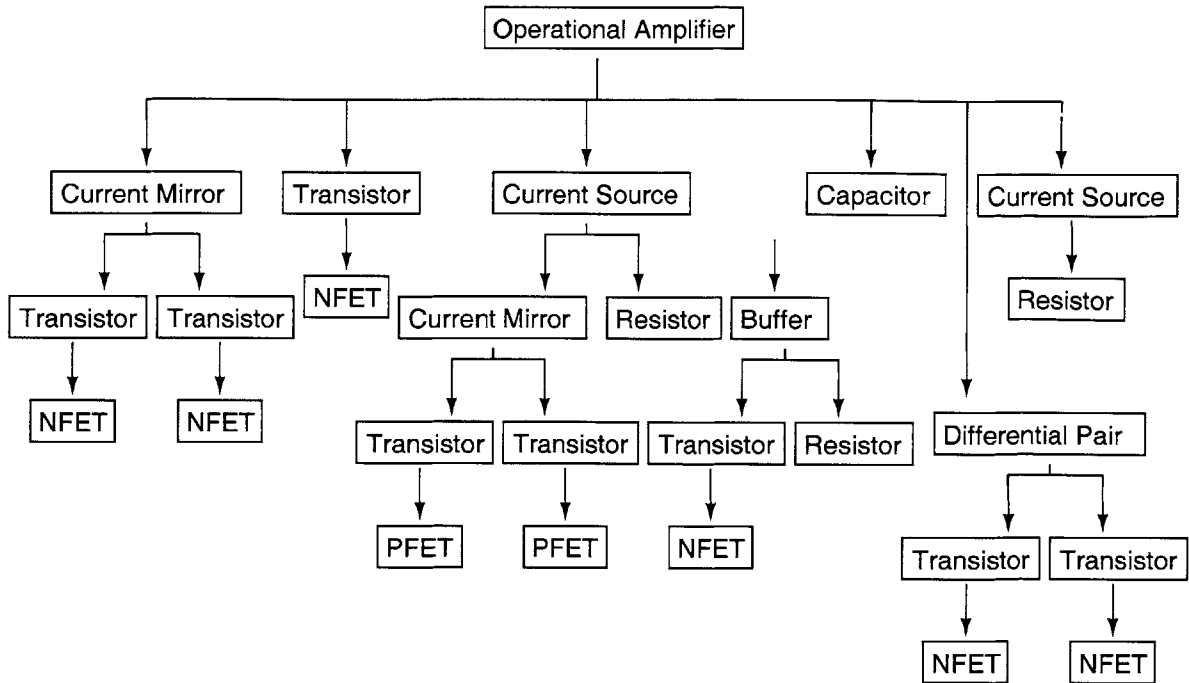
Figure 2-1: Operational Amplifier Representation

scribing the possible topologies of the circuit to be optimized and its subcomponents. The output is a SPICE file of the final circuit. For more advanced usage, the program is designed to be modified — it has an architecture for incorporating new fundamental and compound types of circuit components, and the objective function is specified in the source code.

2.1.1 Circuit Element Data Structure

Each CIRCUITELEMENT consists of an automatically generated ID (unique to the component type), a human-readable (potentially non-unique) name, and a vector of floating point numbers representing component values (widths and length of transistors, etc.).

Special component types exist for fundamental components, such as resistors, capacitors, and transistors. In addition, there is a COMPOUNDCIRCUITELEMENT type, which represents an aggregation of components. Each subelement is given a name. In addition to the fields inherited from CIRCUITELEMENT, COMPOUNDCIRCUITELE-

MENT adds a MAP of subelement names to strings indicating subelement types, and a second MAP from names to the actual subelements. The first MAP is used so that the global system knows what sorts of elements it can populate the second MAP with.

2.2 Component Generator

The system has a library that is capable of generating components based on a component type. There may be multiple designs for a component of a given type (for instance, the N_CURRENT_MIRROR type might be implemented as a plain current mirror, a cascode current mirror, a Wilson current mirror, and as a variety of other designs). Similarly, a component may have multiple types (in most cases, a general type, such as N_CURRENT_MIRROR, as well as several specific types, such as N_WILSON_CURRENT_MIRROR).

Each type of component has an associated generator[1] object capable of creating instances of that component. These generators are registered in a MAP of STRINGS to VECTORS of element generators. New elements may be created with:

```
circuitElement*getElementOfType(string type, int index);
int getElementCount(string type);
```

GETELEMENTOFTYPE returns a new element based on its type (the index is used to differentiate between multiple elements of the same type), while GETELEMENT-COUNT returns the number of elements of a given type.

The C++ startup order guarantees that global variables will be set to zero prior to execution of any code, including the constructors of global variables. We relied on this property of C++ to implement the element generator subsystem in such a way that it requires no initialization. We created a series of stub classes and macros that add generators through the use of constructors of dummy global variables. As a result, components may be added to the system simply by adding files to the build process — there is no global initialization that instantiates and adds the individual

---

[1]This is called an object factory in some OO programming models.

elements. Given a new component class, a corresponding generator can be created and added to the component generator subsystem with a simple macro:

```
ADD_GENERATOR g1(new elementGeneratorAdapter0<newComponent>, "Name");
```

Here, ELEMENTGENERATORADAPTER0 is a template that creates an appropriate element generator. Similar templates are provided for elements whose constructors take arguments. NAME is the name under which the component is registered. ADD_GENERATOR creates an object, with no data or functions aside from a constructor, whose constructor registers the generator with the component generation subsystem.

2.3   File Formats

2.3.1   Circuit Elements

The most common component type is the AUTOELEMENT. This is a subclass of COMPOUNDCIRCUITELEMENT that is automatically generated from a file. This type of file is the primary interface that an engineer would use to input elements into the library. A sample AUTOELEMENT description file is shown below:

```
' Basic Operational Amplifier


NAME: Operational Amplifier 1


TYPE: OPAMP
TYPE: SIMPLE_OPAMP1


SUBELEMENT: P_CURRENT_SOURCE I1
SUBELEMENT: P_CURRENT_SOURCE I2
SUBELEMENT: P_DIFF_PAIR DIFF1
SUBELEMENT: N_CURRENT_MIRROR M1
```

```
SUBELEMENT: N_TRANSISTOR Q1

SUBELEMENT: BUFFER B1

SUBELEMENT: CAPACITOR C1 2400


-- SPICE MODEL

.SUBCKT NAME  1 2 3 4 5

XI1 1 2 6 I1

XI2 1 2 8 I2

XDIFF1 1 2 3 4 9 7 6 DIFF1

XMIRROR 1 2 9 7 M1

XQ1 8 7 2 Q1

XB1 1 2 8 5 B1

XC1 7 8 C1

.ENDS NAME
```

Here, the NAME field gives a human-readable name, primarily used for debugging purposes. The TYPE field registers the component to be included in the map of that type. In the example given, if another component calls for an OPAMP, the element generator may create an element of the type from the file shown above. By convention, elements also usually include at least one unique type, in this case SIMPLE_OPAMP1, in case another component wants to include those elements explicitly. The SUBELEMENT lines map component types to names used within the actual circuit. In the case of capacitor C1, a default value for the first element of the describing vector is specified. The use of this will be explained in chapter 4. Finally, the SPICE MODEL section gives the actual model of the circuit. Here, NAME will be replaced with the component name, when the component is generated. The strings for subelements will, likewise, be replaced with the names of those subcomponents in the actual SPICE file.

There are two ways of handling matched components. The long way is to generate and use compound component types, such as MATCHED_TRANSISTOR. As a shorthand notation, it is also possible to include the same element twice in the SPICE model section under the same name (this is handled correctly for total circuit area

calculations and elsewhere).

2.3.2 Subelement List

The subelement list file contains the filenames of all of the included AUTOELEMENTs, one per line.

2.3.3 Process File

The process file is divided into sections. Each section has a tag delimiting the beginning and ending:

```
--BEGIN-SECTION-NAME--

...

--END-SECTION-NAME--
```

Where SECTION-NAME may currently be either SHEET-RESISTANCES, CAPACITANCE-MATRIX, or TRANSISTOR-PARAMETERS.

The SHEET-RESISTANCES section contains a mapping of layers to resistance (in $\Omega/\square$), with each line containing the name of a layer, followed by a numeric resistance. The TRANSISTOR-PARAMETERS is in the same format, but contains transistor parameters (minimum dimensions, etc.). Finally, the CAPACITANCE-MATRIX contains a list of layers, followed by an upper-triangular matrix containing interlayer capacitances. A sample section is shown below:

```
--BEGIN-CAPACITANCE-MATRIX--
Bulk N+ P+ Poly Poly2 M1 M2 M3 N_W
0    428  726   87     0    33   17   10   42
          0     0    2456    0    36   17   12    0
                0    2366    0     0    0    0    0
                       0   883   62   16    9    0
                             0    53    0    0    0
                                   0   31   13    0
```

$$\begin{array}{ccc} 0 & 32 & 0 \\ & 0 & 0 \\ & & 0 \end{array}$$

```
--END-CAPACITANCE-MATRIX--
```

A single circuit may have devices based on several processes, as would be found in mixed-signal 3D integrated circuits.

Chapter 3

Topology Optimization

3.1   Notation

Let us define the set Ω to be the set of possible circuit topologies. Let $f : \Omega \implies \Re$ be the objective function we are trying to minimize over Ω.

3.2   Global Optimization of Topology

The global topology is optimized using simulated annealing [6]. The general idea behind simulated annealing is that we try to construct a near-optimal solution through a series of random changes to the system. In general, we prefer to keep changes that decrease the objective function. To avoid local minima, however, we allow a small number of changes that increase it. The actual algorithm is inspired by annealing — for every configuration of the system, we treat the objective function as the energy of the system. At every point in time, we have a temperature for the system. At high temperatures, we perturb the system by large amounts, and more frequently accept configurations that have a higher energy than the original. As the system evolves, we reduce temperature, and so make smaller perturbations, and show a stronger preference for ones with lower energy.

The basic simulated annealing algorithm is as follows:

1. Start with a system $S \in \Omega$ and a temperature T.

2. Perturb the system S to a new system $S' \in \Omega$. At large temperatures, make large perturbations. At low temperatures, make small perturbations.

3. Define the weights $w = f(S)$ and $w' = f(S')$. Compute the probability $p = e^{\frac{w-w'}{kT}}$. With probability p, assign $S \longrightarrow S'$. Notice that if $w' \leq w$, this assignment always happens. If $w' > w$, it will occasionally happen, but with declining probability based on the difference between w and w'.

4. Decrease the temperature according to the annealing schedule.

5. Repeat until we reach some end condition.

In our implementation, the initial system consists of a randomly constructed circuit. Each perturbation consists of replacing a subset of the subcomponents of the circuit with new, randomly generated components.

It is implemented as a template function:

```
template<class T1> circuitElement*perturb(circuitElement *ce,
                            int level, T1 f, double scale);
```

Here, LEVEL is the current depth in the circuit tree. F is a function that takes the current LEVEL, and returns a boolean value indicating whether or not we should replace an element. It is a template type, so we can implement F as either a true function, or as an arbitrary function object.

As used in our algorithm, we pass as F a function object that contains the current temperature. It calculates $p = (1 - T)^l$, where T is the temperature and l is the level, as well as a random number r between 0 and 1. If p is less than r, it returns TRUE; otherwise, it returns FALSE. This way, at high temperatures, we have frequent and global perturbation. As temperature decreases, the perturbations become less frequent and more localized to lower levels.

In addition to simulated annealing, we also implement primitives for use in genetic algorithms [9] and other forms of optimization (calculating distance between elements, etc.). At this time, however, the primitives are not used.

26

Chapter 4

Component Value Optimization

4.1 Optimization of Component Parameters

4.1.1 Gradient Descent Algorithm

For gradient descent, we use a variant on Powell's Direction Set Method.

The naïve approach to gradient descent relies on computing the gradient of the slope, minimizing in the direction of greatest descent, and repeating. This yields very poor results in practice — by the Projection Theorem, the vector chosen for every step is orthogonal to the previous one. The problem with this is shown in figure 4-1. If we are in a valley, minimizing in 2 dimensions, and the initial step happens to fall at a 45° angle to the valley, the successive step will fall at 45° in the opposite direction, and each successive step will also fall at a 45° angle. As a result, the overall minimization takes a very large number of steps.

In this example, we would, instead, like to minimize in the direction of the valley. Let us define n to be the number of dimensions. The intuition behind Powell's method is that, assuming the shape and direction of the valley is fairly constant, each set of n orthogonal steps will be similar to the one before (this can be seen by looking at pairs of steps in figure 4-1). An example of Powell's algorithm is shown in figure 4-2.

Stated more formally, let us begin at point p_0 and minimize in the n orthogonal directions. Call the resulting point p_n. Minimize again, and call the resulting point
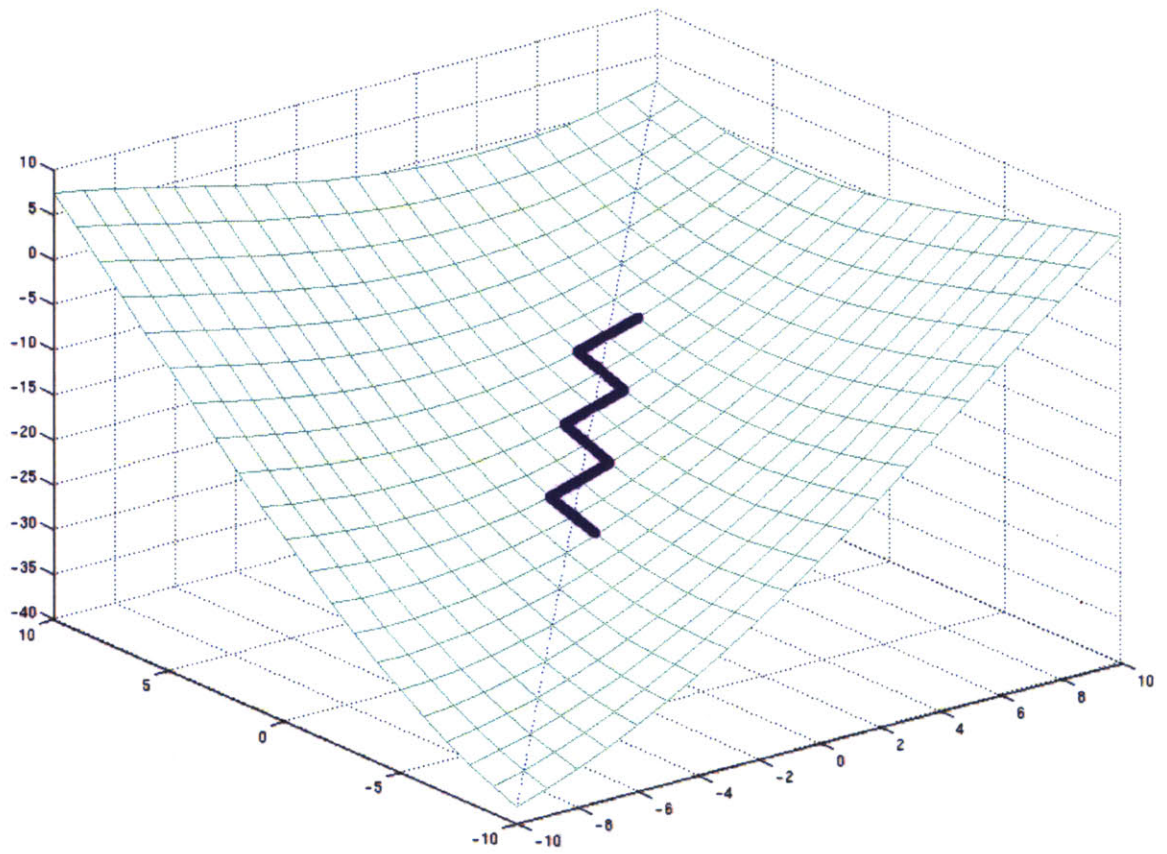
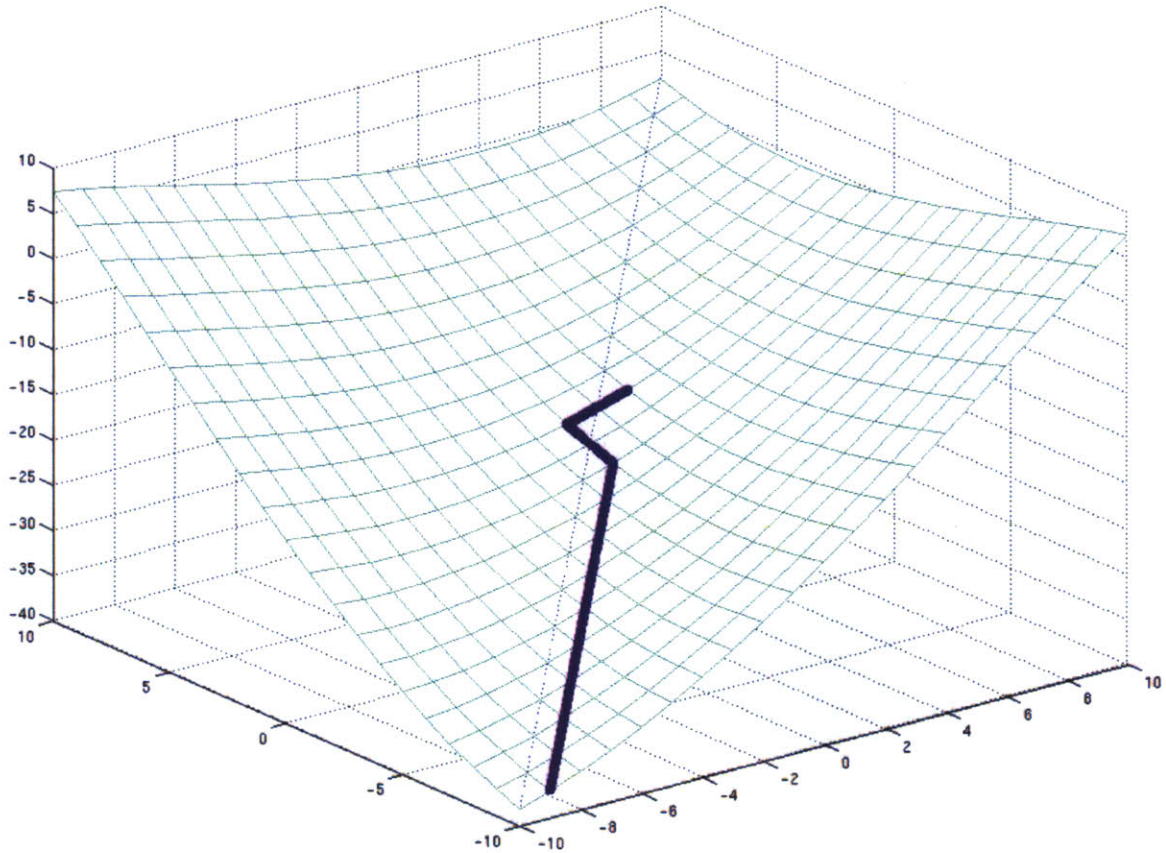Figure 4-1: Example of Naïve Gradient Descent on Valley

Figure 4-2: Example of Powell's Algorithm on Valley

p_{2n}. Then, $p_{2n} - p_n \approx p_n - p_0$, so $p_{mn} \approx m \cdot (p_n - p_0)$. As a result, after a single set of n independent minimizations, we define a vector $v = p_n - p_0$, and run the optimization in the direction v.

At the end of each iteration, we replace a vector in the original direction set with the new vector v. In this way, if we are still in the same valley, in the next iteration, the original set of orthogonal vectors acts as a correction term to v. Formally, let $v_0...v_{n-1}$ be the original set of independent vectors. We assign $v_n \longleftarrow v = p_n - p_0$, and then shift $v_i \longleftarrow v_{i+1}$. In this way, the new $v_0...v_{n-2}$ form a correction factor for the amount by which v is incorrect. Brent [1] shows that in n-dimensional space, in general, we need n iterations of the algorithm (n^2 line minimizations) in order to reach the bottom of a quadratic form.

This approach has the problem that the vectors build up dependence, and after a

number of iterations, we may be minimizing over a subset of the global space. There are a number of solutions to this problem. We take the approach of resetting the vectors after some preset number of iterations.

Formally, the overall algorithm is as follows:

1. Let $v_0...v_{n-1}$ be a set of n orthogonal vectors. Let p_0 be the initial point.

2. Let $p \longleftarrow p_0$. For $i = 0...n - 1$, let $p \longleftarrow \text{linemin}(p, v_i)$

3. Let $v_n \longleftarrow p - p_0$. Let $p_0 \longleftarrow \text{linemin}(p, v_n)$

4. For $i = 0...n - 1$ let $v_i \longleftarrow v_{i+1}$

5. If number of inner iterations not exceeded (default n), return to step 2

6. If number of outer iterations not exceeded, return to step 1

In practice, we found that this algorithm was able to reach very close to the minimum in a small number of iterations of line minimization. Nevertheless, due to the number of steps in each line minimization, the overall system did not scale well to large problems. The algorithm is implemented as a template, and so can operate on arbitrary vector types, so long as the +, *, = and [] operators are implemented, as well as the RESIZE(INT) function.

4.1.2   Local Minima, Constraints, and Changes of Variables

Raw gradient dissent suffers from problems with local minima. In particular, in the case of circuits, most circuits will have local minima where devices are outside of their desired region of operation.

For instance, starting with a simple common-emitter amplifier, as shown in figure 4-3, we find that if R_1 and R_2 are chosen such that the transistor is biased outside of its active region, most of the signal feeds through the collector-base capacitance of the transistor. As a result, if we optimize for a given gain greater than 1, the optimizer acts to maximize this capacitive coupling, and so will increase the resistor
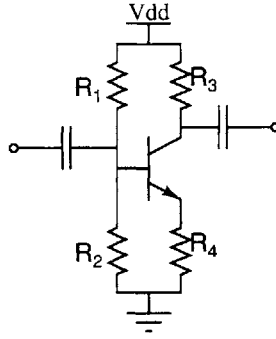
Figure 4-3: Common-Emitter Amplifier with Degeneration

values to as close to infinity as possible. Since the capacitive coupling and the signal coming from the gain of the transistor are directly out of phase, the optimizer acts to minimize gain from the transistor.

This problem has traditionally been addressed using systems, such as simulating annealing or genetic algorithms, which occasionally take random steps uphill. We do not, however, believe that this approach can scale to large circuits. Specifically, let us assume that given random component values, each active component has some probability p of being biased correctly. With n active components, the probability of the overall circuit functioning drops to p^n.

As a result, while the simulated annealing supports modification of continuous component values, in the final version of the algorithm, we disabled this functionality. Instead, we used a simpler, more manual algorithm. In all the cases we encountered, the minima came from the circuit entering regions where it fundamentally does not work — if the circuit does work, there is usually a direct path of descent to the optimal value. We address this problem through several techniques to guarantee the circuit stays within a valid region of operation.

The primary technique in our tool-box is to allow the designer to specify default values for components. This simple technique eliminates the vast majority of local minima. For compound components, such as a current or voltage sources, the architecture allows the engineer to quickly subclass a new CIRCUITELEMENT type that does a coordinate transformation, so the higher-level components can specify default

31

values in a simple format.

In the few cases where this approach fails, the architecture allows several other techniques. First, we can revert to simulated annealing. For isolated cases, probably the best way to do this is to simply to create several components with different default values and let the discrete optimizer pick the working one. We still suffer from p^n scaling, but the approach previously outlined may make n sufficiently small that this will not be a problem.

The next series of techniques rely on modifying the objective function of the overall circuit to avoid local minima. These primarily rely on having the expense of non-functional circuits (and potentially other local minima) be much higher than that of poorly performing circuits. We provide a library of mathematical functions that allow simple construction of complex weight functions. These are listed in appendix A. This should be used sparingly, since over-reliance on this class of techniques can yield very poor numerical performance.

In the most extreme cases, the designer can override the CIRCUITELEMENT type to force more complex constraints.

4.2   Constraints

The manufacturing process imposes some constraints on device size. For instance, transistors have minimum lengths and widths. The system imposes these constraints through coordinate transforms. The basic classes for transistors, resistors, and capacitors map the describing vector to a hyperbola. The hyperbola was chosen because, away from the minimum, the describing vector still maps directly to component values. Near the minimum, it gives a smooth function, and so does not result in the numerical instability in the way that, for instance, $|x| + y_{\mathrm{min}}$ would. The curvature of the hyperbola has not been optimized. Less curvature would result in a worse mapping of vector values to component values and would likely slow convergence for components away from the minimum. More curvature would increase gradient minimization time for components near the minimum, and if taken to an extreme, may

result in numerical instability.

4.3 Line Minimization

We rely on a standard golden-ratio bisection algorithm for minimization of a function in one variable. Given a starting point, we search geometrically in both directions until we find a set of three points such that the middle point is smaller than the remaining two points. In the general case, a component should not be able to grow indefinitely, since fundamentally, all variables eventually map back to area, which should be included in the objective function.

The line minimization function is implemented as a template, and so theoretically can be applied to arbitrary data types, with an arbitrary type of operation over which to minimize.

The performance of the line minimization algorithm was the dominant bottleneck in the performance of the program, and we are currently experimenting with more intelligent algorithms.

4.4 Further Readings

Numerical Recipes [6] offers a good overview of techniques for function minimization with and without derivatives. Brent [1] offers rigorous proofs about the properties of the algorithms used.

Chapter 5

System Results

In a survey of research papers, the testing methodology was suspect — namely, in all but one case, the author of the paper would show a chart with two columns, one with desired specs, and the other with the specs of the circuit that the program generated (the one exception used a simple filter design from an introductory circuits course, and compared to student performance). In all cases, the test program met the required specs. It was, however, impossible to determine how ambitious the target specs were, how close to optimum the final design was, nor how the program compared to previous attempts. Due to the fact that no proper testing methodologies are available, we follow in this tradition.

We tested the system on a number of small to medium scale circuits. Convergence on small circuits was very rapid (the degenerated common-emitter amplifier mentioned previously and similar circuits) — several hundred simulations at most.

On a fairly complex operational amplifier design, with several dozen degrees of freedom in continuous parameters, and a large library of possible subcomponents, the overall system was impractical. The gradient descent took several hours per design[1]. Since each step in the simulated annealing requires a gradient descent, using the overall algorithm was unreasonably slow.

As a result, for medium-scale circuits, we tested the gradient descent algorithm independently of the simulated annealing. Our test setup was an operational amplifier

---

[1]Test system was a Pentium III/866 notebook.

design project from MIT's advanced graduate honors analog MOS LSI course[2]. This design problem was chosen for a number of reasons:

- The circuit only functions within a tight region of design parameters. To achieve the DC signal gain, the output stage must have functioning cascodes. However, the ± 1V output swing specification, combined with the ± 1.25 power rail leaves a very narrow region of operation for the cascodes. Similarly, the resistor in the compensation network must remain reasonably matched to the g_m of the output stage, or else the system fails to achieve adequate phase margin. This tightness of the specs results in a large number of narrow valleys outside of which the circuit ceases to function; this is the sort of problem which automated circuit optimizers tend to be worst at.

- The specs were fairly tight. A fair portion of the class did not successfully meet all specs. As a result, we had external evidence that the specs formed a reasonably challenging baseline against which to evaluate the circuit.

Here, we used the operational amplifier shown in figure 5-1. This configuration has 33 degrees of freedom. We began the simulation in a known-working but rather poor configuration — namely, we used a very large compensation capacitor to guarantee stability, and uniform, large, square devices to avoid short-channel effects. The weight function for all variables except phase margin was:

$$f(x) = \begin{cases} a \cdot e^{-r*b} & : \quad x > -.25 \\ r^2 & : \quad x \le -.25 \end{cases}$$

with $r = (x - x_0)/x_0$, $x_c = 0.25$, $b = \frac{2}{x_c}$, $a = x_c^2 \cdot e^{-b \cdot x_c}$.

This function behaves as least squares when each variable was far away from the target. Near the target, it switched to a decaying exponential, in order to give a small additional bonus for performance beyond the target. The constants are chosen such as to make the function and its derivative continuous at the transition from square-law to exponential.

---

[2]6.775: Design of Analog MOS LSI

Figure 5-1: Test Operating Amplifier Topology

For phase margin, we used the objective function:

$$f(x) = \begin{cases} \frac{(x-x_0)^2}{x_0} + e^{20-x} & : \quad x < 60 \\ 0 & : \quad x \geq 60 \end{cases}$$

The addition of the exponential gave a wall that guaranteed the circuit did not fall out of the desired region of operation. The overall weight function was a sum of the individual weight functions of all associated variables.

Due to simulation time, we simplified the specs from the original problem in the class. First, we pruned several "automatic" specs, namely power supply rejection ratio and common mode rejection ratio. If these are specified at fixed frequencies, as was the case with this problem, it is trivially possible to send these to values arbitrarily close to zero. To do this, we can couple the power rails and common mode into either inverting or non-inverting nodes. The gradient descent will set the level of coupling such as to give zero gain from the input to output. Furthermore, this

37

Specification	Target	Final Result
Power Dissipation	$< 100\mu W$	$91\mu W$
DC Signal Gain at $V_{OUT} = 0V$	> 20000	26527
DC Signal Gain at $V_{OUT} = 1V$	> 10000	17172
DC Signal Gain at $V_{OUT} = -1V$	> 10000	18559
Unity Gain Frequency	$> 50MHz$	$63MHz$
Phase Margin	$> 60°$	$60°$

Table 5.1: Gradient Descent Results

topology tends to give fairly good CMRR and PSRR naturally, so we did not expect these to be limiting factors.

We also removed specifications for settling time and step response. These require lengthy transient simulations, and can be fairly accurately approximated from bandwidth and phase margin. Finally, we removed specs for noise and input common mode range, since these specs were not the limiting factor on the topology.

As is shown in table 5.1, the gradient descent was able to exceed all specs. It took 5,000 iterations of SPICE to initially meet specs, although the final specs shown are after 60,000 iterations (although the values did not change significantly during the last half of that run).

As mentioned, due to performance constraints, the simulated annealing algorithm was tested on a library of operational amplifiers and operational amplifier subcomponents without gradient descent. It was clear that it was improving the system, although due to the lack of optimization on component values, it is difficult to give an objective metric of performance. A library of subcomponents used in testing simulated annealing is given in appendix B. The top-level circuits were variants on the operational amplifier shown in figure 5-1, with and without the addition of an output buffer.

5.1 Theoretical Performance Scaling

It is easy to show SPICE performance cannot scale better than linearly in the size of the circuit. In practice, it is often considerably worse, and furthermore, complex

circuits require larger numbers of simulations. For instance, the test operational amplifier required a minimum of three simulations in order to demonstrate performance over a range of output levels, and would require nine in order to evaluate all parameters in the original problem. If we were to extend the problem to calculating across process corners, temperature variations, and device variations, evaluating the objective function would require dozens of simulations. A large body of work exists on approximating SPICE results with neural networks and other approximate models [10]. However, these systems base results on changing parameter values with a fixed topology. It is not clear if or how they could be applied to differing topologies. While this could conceivably increase the speed of individual gradient descents, due to the initial number of SPICE runs needed to train the model, we are sceptical of the possibility of order-of-magnitude performance increases, although it is certainly a worthwhile area of experimentation.

Powell's algorithm theoretically requires n^2 iterations to reach the minimum of a general quadratic form [1]. In practice, we found that a reasonable approximation to the minimum was reached considerably more quickly in most cases, and the actual number of iterations was equal to a small constant times n. However, in a best case, it still takes a minimum of n iterations to determine the path of steepest descent.

Line minimization requires a logarithmic number of operations in the ratio between distance to minimum and tolerance.

Chapter 6

Conclusion

6.1 Future Improvements

This program is the first step in the development of a larger system used for automating optimization and layout of mixed signal circuits, specifically targeted at use in 3d integrated circuit design. As a result, there is a large set of tasks remaining.

6.1.1 Layout

Eventually, the system will also need to include support for automatic layout. Simulated annealing is one of the more common techniques for automatic layout of analog circuits [8]. Adding basic support for automatic layout would consist of the following steps:

- Adding a method to the generic CIRCUITELEMENT class that returns a layout, based on some externally accessible parameters.

- Adding support to the simulated annealing algorithm for optimizing the layout-related parameters.

The major difficulty with implementing automatic layout is developing a reasonable representation of a circuit — one that is likely to produce realistic layouts, have a reasonable number of degrees of freedom, and allows convergence to a near-optimal

layout reasonably quickly. Specifically, statistically common configurations should follow the appropriate rules of analog layout. Although a large body of academic research exists in this area, automated layout still lags behind manual layout, and so is still primarily limited to the low-performance digital domain.

In the same way as the architecture described here currently supports large, manually designed blocks, the architecture also naturally supports manually laid out subcomponents in critical portions of the circuit. It should also be relatively easy to add constraints to the layout by subclassing from the appropriate CIRCUITELEMENT class. As a result, critical sections of the layout may be implemented manually by a layout engineer.

6.1.2 Mixed Signal Integration

Basic mixed signal integration is nearly automatic. Adding support for mixed signal circuits would require adding CIRCUITELEMENT types for digital elements, modifying output to include non-SPICE for non-analog elements, and creating an objective function evaluator for the overall system.

6.1.3 Subcomponent Optimization

The performance of this system scales better than that of most academic systems, but very large circuits may still be prohibitively slow. One way around this problem is to preoptimize subcomponents. While theoretically a good idea, in practice, this approach is fairly difficult to implement — namely, circuits have very large numbers of unknown variables. Determining what to optimize for, and which components are "better" than others is difficult, and the library of preoptimized components very quickly grows to be unreasonably large.

6.1.4 More Intelligent Subcomponent Selection

Simulated annealing and genetic algorithms can run much more efficiently if, rather than choosing a component at random, they have some idea as to which components

work well. This way, they can choose different subcomponents with different probabilities. The simple way to implement this would consist of allowing the designer to specify default or probable component values, in much the same way as we permit defaults for describing vector values.

A more ambitious approach would consist of developing machine learning algorithms to determine what components work well in what contexts, or alternatively, what substitutions generally make sense. However, creating a usable definition of "context," and useful groupings of components is difficult. The concepts are very vaguely defined, and as a result is unclear what the proper approach to the problem is.

6.1.5 Improvements in Line Minimization

Our current line minimization algorithm is based on the simple golden ratio bisection algorithm. More sophisticated line minimization algorithms exist — generally, they rely on some sort of extrapolation from existing data points, falling back to golden ratio bisection for cases where extrapolation is not converging well.

Although it would make sense to implement a quadratic extrapolation technique immediately, many circuit equations are not well-modeled by a quadratic. We may be able to do better by implementing a larger set of extrapolators, and either choose one based on a machine learning algorithm, or simply directly chose the one that best matches existing known points in the line.

6.1.6 Faster Circuit Modeling

Gradient descent can be significantly improved if we can use existing data-points to generate an accurate model of the circuit. This model can be used with our algorithm instead of simulation for a modest performance increase. More ambitiously, given such a model, we may be able to calculate the optimal solution directly. Several approaches to generating models exist, commonly relying on neural networks [10] or posynomial models [2].

6.1.7 Three Dimensional Integration

At the moment, the system supports multiple transistor types. However, since it is not yet capable of layout, it does not recognize the cost of switching between layers. In some types of 3d processes, this cost is fairly expensive, and needs to be well-modeled. This should come as a natural consequence of integrating layout. However, integrating this sort of information into the simulated annealing algorithm, outside of the objective function, may result in improvements in speed. Specifically, adding a penalty to switching layers in the PERTURB function may potentially result in faster convergence.

6.2 Conclusion

We have demonstrated a system capable of automatic optimization of analog integrated circuits. The system is currently capable of working on small to medium size circuits. We believe, however, that it can be extended to large-scale circuits as well. Primarily due to the length of the edit/compile/run cycle, the parameters used in gradient descent are still very suboptimal. Optimizing these should give a large improvement in performance. Implementing some level of extrapolation in gradient descent should also give a significant improvement in performance. Finally, avoiding a full gradient descent in between steps in simulated annealing, but rather reoptimizing only the portions of the circuit effected by the perturbation can give an order-of-growth improvement in speed.

Furthermore, the circuit has direct paths of extension for use with 3D IC layout, as well as mixed signal design. Although these are significant endeavors, we believe the system provides a natural, flexible architecture within which to implement them.

Appendix A

Library of Functions

The system includes a library of functions to assist in constructing objective functions.

A.1 Brick Wall

This function is used to prevent a parameter from exiting a certain range.

$$f(x) = \begin{cases} \frac{1}{x} & : \ x > 0 \\ \text{MAX\_FLT} & : \ x \leq 0 \end{cases}$$

A.2 Limiting Functions

These functions can be used to limit a parameter to fall within a given range, and are typically used if we want to guarantee that one parameter will always be more important than another.

$$f(x) = e^{-x^2}$$

$$f(x) = 1 - e^{-x^2}$$

$$f(x) = e^{-x^6}$$

$$f(x) = 1 - e^{-x^6}$$

$$f(x) = \frac{\arctan(x)}{2\pi}$$

A.3   Approximate Least Squares

This approximates square-law, until we reach close to the desired value. At that point, it switches to exponential to continue to give small gains for beating the specification.

$$f(x) = \begin{cases} a \cdot e^{-r*b} & : \quad x > -x_c \\ r^2 & : \quad x \le -x_c \end{cases}$$

with $r = (x - x_0)/x_0$, $x_c = 0.25$, $b = \frac{2}{x_c}$, $a = x_c^2 \cdot e^{-b \cdot x_c}$.

Appendix B

Component Library

For simulated annealing, we include a library of pre-made components. Although these are shown with bipolar transistors, in most cases, they allow an arbitrary transistor of the same polarity as a subcomponent. In most cases, we only include some subset of the components in each run.
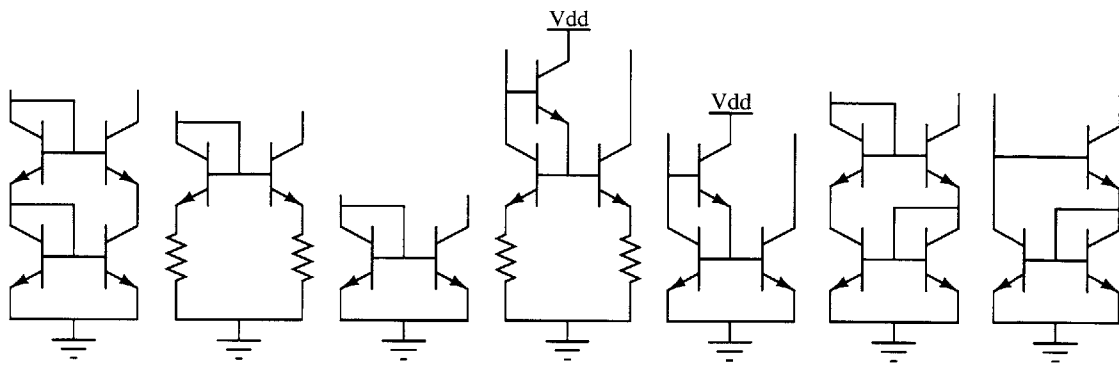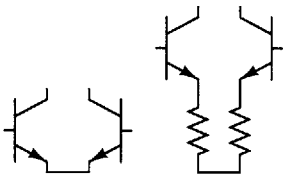
B.1 Current Sources
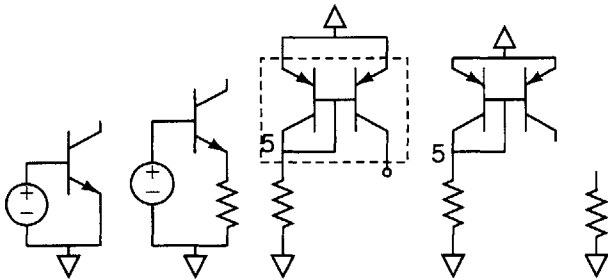
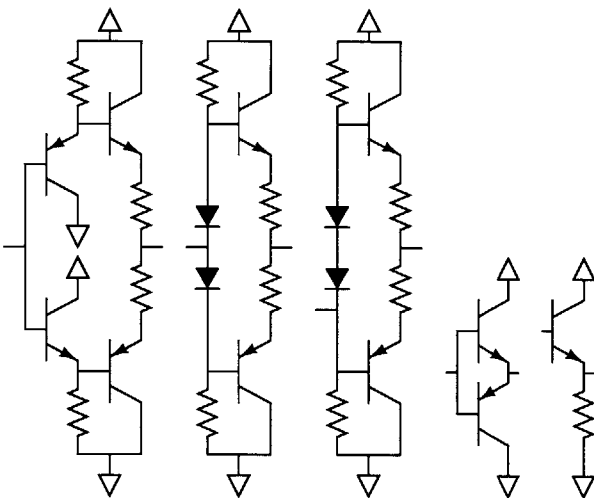## B.2 Voltage Sources
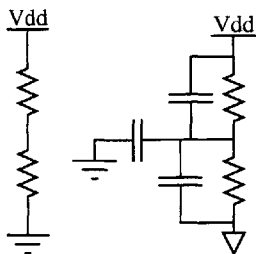
B.3 Current Mirrors

B.4 Differential Pairs

B.5 Buffers

Bibliography

[1] Richard P. Brent. *Algorithms for Minimization Without Derivatives*. Prentice-Hall, Englewood Cliffs, NJ, 1973.

[2] W. Daems, G Gielen, and W Sansen. Simulation-based generation of posynomial performance models for the sizing of analog integrated circuits. *IEEE Transactions on Computer-Aided Design*, 22(5):57–534, May 2003.

[3] S. Das, A. Chandrakasan, and R. Reif. Design tools for 3-d integrated circuits. *In Proc. ASP-DAC*, pages 53–56, January 2003.

[4] F. El-Turky and E. Perry. Blades: An artificial intelligence approach to analog circuit design. *IEEE Transactions on Computer-Aided Design*, 8(6):680–692, June 1989.

[5] R. Harjani, R. Rutenbar, and L. Carley. Oasys: A framework for analog circuit synthesis. *IEEE Transactions on Computer-Aided Design*, 8(12):1247–1266, December 1989.

[6] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*, chapter 10. Cambridge University Press, second edition, 1992.

[7] T. Sripramong and C. Toumazou. The invention of cmos amplifiers using genetic programming and current-flow analysis. *IEEE Transactions on Computer-Aided Design*, 11(21):1237–1251, November 2002.

[8] M. Vecchi and S. Kirkpatrick. Global wiring by simulated annealing. *IEEE Transactions on Computer-Aided Design*, 2(4):215–222, October 1983.

[9] Patrick H. Winston. *Artifical Intelligence*, chapter 25, pages 507–528. Addison-Wesley, Reading, Massachusetts, third edition, 1992.

[10] G. Wolfe and R. Vemuri. Extraction and use of neural network models in automated synthesis of operational amplifiers. *IEEE Transactions on Computer-Aided Design*, 22(2):198–212, February 2003.