Accurate and Precise Computation using Analog VLSI, with Applications to Computer Graphics and Neural Networks

Thesis by

David B. Kirk

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1993

(Defended March 16, 1993)

Caltech-CS-TR-93-[0008]

Acknowledgements

Thanks to both Alan Barr and Carver Mead for ideas, encouragement, and support for this project. This thesis is a synthesis based on inspiration from work being done in both Alan Barr's Graphics Lab and Carver Mead's Physics of Computation Lab. In particular, Al has been especially helpful to me in choosing a productive path from myriad possibilities.

Many, many thanks to my collaborators, Kurt Fleischer, Lloyd Watts, and Douglas Kerns for their dedication and interest in making our joint projects successful. Special thanks to Kurt, who has been a frequent and important collaborator. Thanks to John LeMoncheck for help getting our physical chip testing setup together, and generally helping me to get chips built. I am also indebted to all of the members of the graphics lab, for moral support, tool support, technical advice, and in general making the lab a productive place to work.

This work was supported in part by an AT&T Bell Laboratories Ph.D. Fellowship, and by grants from Apple, DEC, Hewlett Packard, and IBM. Additional support was provided by NSF (ASC-89-20219), as part of the NSF/DARPA STC for Computer Graphics and Scientific Visualization. All opinions, findings, conclusions, or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

Abstract

This thesis develops an engineering practice and design methodology to enable us to use CMOS analog VLSI chips to perform more accurate and precise computation. These techniques form the basis of an approach that permits us to build computer graphics and neural network applications using analog VLSI. The nature of the design methodology focuses on defining goals for circuit behavior to be met as part of the design process.

To increase the accuracy of analog computation, we develop techniques for creating *compensated* circuit building blocks, where compensation implies the cancellation of device variations, offsets, and nonlinearities. These compensated building blocks can be used as components in larger and more complex circuits, which can then also be compensated. To this end, we develop techniques for automatically determining appropriate parameters for circuits, using constrained optimization. We also fabricate circuits that implement multi-dimensional gradient estimation for a gradient descent optimization technique. The parameter-setting and optimization tools allow us to automatically choose values for compensating our circuit building blocks, based on our goals for the circuit performance. We can also use the techniques to optimize parameters for larger systems, applying the goal-based techniques hierarchically. We also describe a set of thought experiments involving circuit techniques for increasing the precision of analog computation.

Our engineering design methodology is a step toward easier use of analog VLSI to solve problems in computer graphics and neural networks. We provide data measured from compensated multipliers built using these design techniques. To demonstrate the feasibility of using analog VLSI for more quantitative computation, we develop small applications using the goal-based design approach and compensated components. Finally, we conclude by discussing the expected significance of this work for the wider use of analog VLSI for quantitative computation, as well as qualitative.

Contents

Index of Figures

Ch. 6. Applying the Goal-based Design Methodology

Chapter 1

Introduction

This thesis is a small step toward a long-term goal of producing a new type of hardware for computer graphics and neural networks. Chapters 1 and 2 describe the philosophical basis of our work, including a brief review of previous work. Chapters 3 and 4 describe numeric representation and thought experiments for circuit techniques to improve the precision of analog computation. Chapters 5, 6, and 8 describe research and experiments that we have done, including measurements from chips that we have built.

As a part of our discussion, it is important to describe the motivation for our work. In computer graphics, a need exists for faster calculations in many areas, including geometric modeling, rendering, physically-based modeling, constraint calculations, and virtual environments. Computation speed also limits the size and complexity of neural networks that can be built and simulated. There also exist applications in other areas of endeavor that require interaction with the real world in real time, such as speech recognition and synthesis, vision, robotics, and interactive input and display devices.

In each case, we need to compute values of mathematical functions, solutions to linear systems of equations, roots of nonlinear equations, solutions of differential equations, solutions to constrained minimization problems, and other large problems.

We also desire techniques and computational media that scale well for problems that are large in dimension or complexity. In order to effectively address these computational needs, we require quantitative computation at a high rate of speed. This thesis proposes techniques for improving the quantitative performance of analog VLSI sufficiently that we can consider using analog VLSI for some of the applications described above.

We present a world view for computation and simulation that includes goal-setting and satisfaction behavior as a key component. The goal-setting is part of a design methodology for making accurate and precise analog circuits. We present designs and test results for circuits which implement the goal-based design ideas. We also describe techniques for choosing parameters for these circuits, both externally and on-chip. Finally, we present some small systems which are built using these circuits and the techniques hierarchically. We also provide some zeroth-order analysis of the expected performance of the analog circuits as compared to a typical digital computer realization. In conclusion, we also discuss the expected significance of this work for future applications involving quantitative computation using analog VLSI.

1.1 Why Use Analog VLSI?

Currently available digital hardware is simply not fast enough for the real-time tasks described above, and massive parallelism in digital computers has so far failed to deliver on its great promise in many contexts. Why is there any reason to believe that analog computation will be more fruitful? And, why should we consider analog VLSI? Part of the reason is that analog computation can be very fast: analog circuits can compute approximate solutions to differential equations, constraint problems, etc., very rapidly. Some problems are inherently analog, and their discretization is an artifact of the dominance of digital computers. In addition, we would like to take advantage of some of the benefits of VLSI fabrication. For instance, integrated circuit technology scales well with problem size, in that large and complex circuits can be fabricated on a single chip. Also, VLSI design and fabrication techniques are developing rapidly. Analog VLSI can be a very efficient use of silicon and power, producing compact and low-power implementations.

There has been increasing interest recently in using analog VLSI [Mead 89] for a variety of computational tasks. Mead and others have pursued the paradigm of using individual analog transistors to model components of neural systems. One of Mead's insights is that rather than developing an entirely new manufacturing technology for producing analog VLSI chips, we can produce analog CMOS VLSI chips using standard digital CMOS VLSI processes. The key element in this strategy is to produce designs that are tolerant to the device variations that are present in a digital production process.

Much of Mead's work has been in an area where actual biological systems (esp. individual neurons, synapses, and networks of neurons) are used as metaphors for the design of analog circuits. The architecture of biological neural systems guides the construction of the analog VLSI circuits. Mead and others have pursued the paradigm of using individual analog transistors and small circuits to model components of neural systems. An important component of this work lies in the attempt to mimic the adaptation that real biological neurons are able to do. Modeling analog VLSI simulations of neural systems requires producing circuits that are intrinsically adaptive.[1] Another component of this design philosophy is the exploration of architectures and circuits that are tolerant of device variations, and perform computations collectively.

Other research has focused on increasing the accuracy and precision of computation with analog VLSI, and on developing a design methodology for creating analog VLSI circuits which can

[1] One view of the desirability of "intrinsically adaptive" circuits is that real biological neurons are inaccurate computational elements, and yet must adjust to real inputs. To produce a reasonable analog VLSI model of neurons' behavior, the circuits must be intrinsically adaptive as well.

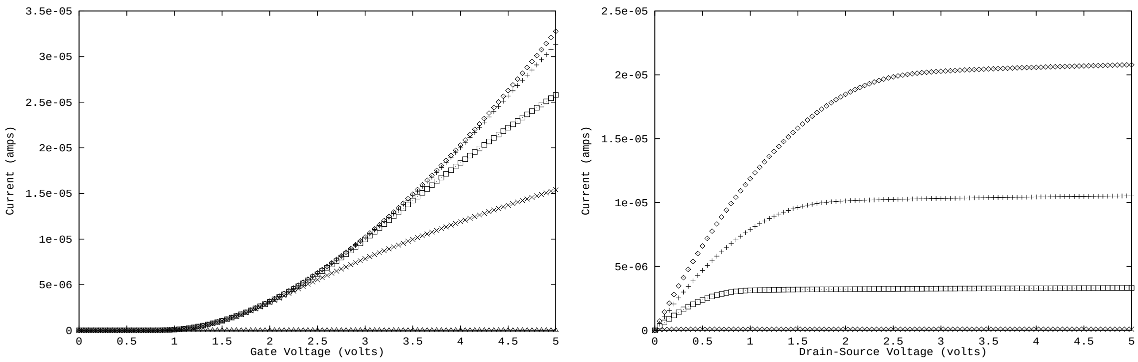

Figure 1.1: a) shows the drain current of a single transistor, as the gate voltage is varied from 0 to 5 volts. The family of curves represents varying the difference between the source and drain voltage. b) shows the drain current as the source-to-drain voltage difference is varied from 0 to 5 volts. The family of curves represents varying the gate voltage. The analog VLSI multipliers discussed in Chapter 6 operate in the nearly linear region to the right of a) and to the left of b). □

be adjusted to perform to the desired accuracy [Kirk 91]. This work can be characterized as using adaptation and optimization to harness analog VLSI for more "conventional computing" applications. This approach is attractive because analog transistors provide a rich computational substrate. Figure 1.1a) shows the current flowing through an analog transistor as its gate voltage is varied. Figure 1.1b) shows the current as the source-to-drain voltage is varied, while holding the gate voltage constant. These figures are meant as a qualitative demonstration of the variety of current-voltage responses available from a single transistor. Note the regions of qualitatively different behavior, such as roughly linear, exponential, and quadratic I-V relation.

1.1.1 Why Not Analog VLSI

Given all of these exciting features of analog computing, why don't we compute everything with analog circuits? And, furthermore, why not analog VLSI? With current technology, offsets and other variations in individual devices and circuit components can easily vary by 10-20%. So, if we are constructing our circuits from discrete analog components, we can hand-select them to minimize variations. This quickly becomes impractical for any large system, and we would like to have greater integration, in terms of the number of components on a chip. If we construct our circuit in analog VLSI, however, we then cannot replace or rearrange the components within the chip.

Although there are, as described above, exciting regions of nearly linear and exponential behavior, analog devices are plagued by nonlinearity and nonlogarithmy. Analog chips are ultimately imperfect and difficult to control. We must also be prepared for noise in our circuits, as an artifact of a direct signal representation. Consequently, analog circuit design has a reputation as being "black magic."

The resulting effects are problems with accuracy and precision in analog computation.

1.1.2 Our Approach: Kill the "Why-nots"

A reasonable approach to exploiting a technology with advantages and drawbacks is to try to use the good parts and avoid the bad. We propose to devise circuits and techniques to improve accuracy and precision in analog VLSI. We then proceed to develop these techniques into a design methodology. This design methodology is then used to construct some example applications.

1.2 Describing the Physical World with Mathematical Models and Goals

The research described in this thesis is motivated by a goal-based view of modeling and simulation. A model can be posed in terms of a goal or set of goals, and the purpose of a simulation can be to meet some goal or set of goals. Goal-setting and goal satisfaction are reasonable and desirable ways to achieve success in parts of a simulation.

We can create a goal-based model for an analog chip's behavior, and we can use this model to control the analog VLSI chip and improve the accuracy and precision of the chip's performance. So, goal-based modeling operates at several levels, both within our simulation of the chip's behavior and within the actual circuits.

We wish to use analog VLSI to implement approximations to mathematical models, since we believe that the physical world can be described by a set of mathematical models, often including differential equations. Accordingly, we proceed by choosing some portion of the physical world to model, and formulating a set of equations to model it.

We subscribe to a structured modeling approach described in [Barzel 92]. We choose some portion of the world to model, and decide which aspects of that portion of the world are of interest. We then formulate equations which describe the behavior of the portion of interest.

We then solve the equations (techniques discussed in Sec. 1.4 and Sec. 1.3), often in the context of a simulation of the model. We evaluate the results in terms of accordance with model intent: did the simulation provide the desired information? We may then potentially reformulate the model, to redo the simulation in order to learn more. Our *explicit goals* for the simulation help us to evaluate our results.

The equations which we have formulated to describe the physical world may be used as the basis for a computational simulation. Parameters of the simulation may be derived from the scope of the region of interest of the physical world. The simulation goals are derived from the model equations.

We wish to choose a solution technique that has the potential to meet the stated goals, and choose a computational substrate on which to implement the chosen solution technique. One way to solve the equations for our simulation is using analog computation. In theory, analog computation is ideal, because analog circuits are excellent substrate for solving ODE's - the normal operation of an analog circuit continuously solves ODE's! Part of the challenge is to construct a circuit that solves the ODE's that we wish to solve. This idea will be developed further in Sec. 1.4.

We prefer to approach the simulation using "direct" methods. A direct method is one which guarantees solution of the problem, by solving the problem explicitly, rather than implicitly. A direct method implies that there is a high probability that satisfying the goal will produce a solution to the problem:

$$\textit{satisfaction of direct method goal} \equiv \textit{solution to problem} \tag{1.1}$$

An important part of the use of direct methods is to understand the problem that you are trying to solve, and formulate goals that are guaranteed to solve that problem. We would like to contrast this approach with an alternate approach involving the use of indirect methods.[2]

We encourage the use of direct methods in modeling and simulation, to guarantee that the actual goals will be met. The explicit choice of goals is an exercise that has many benefits over tacit choice of goals. Using this approach to guide the design of a model increases the chances that the model will provide the desired effects. We attempt to use this approach in the work presented in this thesis, choosing goals which guarantee success; the goals are identical to our objectives in creating the model. We wish to avoid tacit or implied goals.

1.3 Advantages and Disadvantages of Simulation using Digital Computers

One method which can be used to effect a simulation of the model equations is to solve the equations on a digital computer. For the case of differential equations, solution implies integration though time. A number of software packages or libraries provide this capability in some form. We can also use general purpose tools, rather than special one-time-use programs, and it is convenient to pursue the simulation in this way.

This approach has a number of advantages, as well as some disadvantages. One of the greatest advantages of digital simulation is that by and large, digital computation is a proven technology. Digital computers have been manufactured for a number of years. They work. In fact, they work

[2]Let us assume that we wish the floor to be clean. If we have stated the goal explicitly and directly as "we wish the floor to be clean," then it is easy to see that satisfaction of the stated goal will actually result in satisfaction of the true goal.

Alternatively, we could state the goal as "we wish to mop the floor," (confusing the goal with a method for pursuing the goal), or "we wish the floor to be wet," (confusing the goal with one of the likely consequences of the satisfaction of the goal). Satisfaction of one of these stated goals does not guarantee that the actual goal will be satisfied.

repeatably and (often) identically. They have routinely been used to design chips, simulate circuits, even write dissertations.

Another important advantage of performing simulations using digital computers is that simulations will automatically get faster over time. How can this be so? As long as some care has been taken to write a portable program, the simulation can be run on almost any (uniprocessor) digital computer. Over time, it has been seen that the performance of digital computers increases by roughly constant factors per year. This geometric increase in simulation performance is available without any algorithmic or modeling expense. With additional effort, the program can be run on a multiprocessor computer system, and there is the potential for nearly N-fold speedup, where N is the number of processors. In practice, for many types of problems, the N-fold speedup is not achieved, however, and is often not very closely approached.

Even with all of the advantages described above, digital simulation of the model equations can still be painfully slow, particularly in the case of integrating differential equations. The main reason for this is that the differential equations are meant to describe a continuous process which is being approximated by a sequence of discrete steps, since digital computers do not operate in continuous time.

We may often spend lots of time and lots of work for simple things. An example of this is seen in attempts to solve stiff differential equations. We also have only coarse control of error tolerances, specified over the range of a problem, whereas we may be much more concerned with details of one particular part of the simulation. The overall lack of speed in digital computing may cause unwanted approximations in the model or the accidental neglect of important details.

Another potential problem with digital computation is quantization errors. Digital representation is inexact, in the form of discrete bits. The floating point representation of numbers makes the errors worse since the absolute precision varies with the size of the signal. We can use fixed point fractional arithmetic, but the range of our representation is then limited. In either case, sequences of digital arithmetic operations must be re-quantized at each step. The process of repeated quantization causes roundoff errors to accumulate throughout the process of the computation. This accumulation can cause instability when roundoff occurs at a critical point in computation. Integration over time accumulates roundoff errors, and differentiation may cause errors, too (particularly, a small difference between large numbers).

1.4 Advantage and Disadvantages of Simulation using Analog Computation

Another alternative for a computational medium for simulation is to use adaptive analog VLSI. As discussed in earlier sections, analog VLSI has many potential benefits, including speed, scale and efficiency of integration, and low power. Using analog VLSI, we can directly simulate more details of our model by designing those details into our circuits. The potential drawbacks are limited accuracy and precision, and problems with repeatability of our computation.

We can compare a very large computation, such as a climate simulation on a Cray supercomputer to an analog implementation. The digital simulation may be solved using finite elements, discrete time steps, and must forego consideration of many details to scale the problem down to a manageable size. Using an analog circuit to directly model parameters and simulate in continuous time may produce a result that is both qualitatively and quantitatively different. Which approach is better? It's hard to say. Analog VLSI is similar in some ways to old analog computers, and so a consideration of their strengths and weaknesses may be helpful.

There is a long history of analog computing. Engineers and scientists solved many problems in the 1940s and 50s with analog computers. Most analog computers (ACs) were special purpose, built to solve one problem only, although some were configurable general purpose machines. The configuration process was physical, though, requiring connections and components to be changed by hand. ACs were expensive, physically large, and generally unreliable.

In addition, just as when using analog VLSI today, accuracy was a problem for ACs. Some of the same reasons apply; individual components' variation from nominal values, nonlinearities, and non-ideal behavior prevented significant quantitative computation. Analog computers often produced results that varied with the physical environment: temperature changes affected the computation results. Subsequently, analog computers were devoured by digital computers in the marketplace. Reasons for the decline of analog computing included analog computers' large size, expense, low reliability, and limited accuracy and precision. The old analog computers could mostly be used for qualitative computation.

Analog VLSI, implemented in a state-of-the-art CMOS technology, can compute many functions quite rapidly, although approximately. Due to the levels of integration that are possible today, we can produce compact, low power implementations in analog VLSI. In addition to the analogy to historical analog computers, there are some key differences. It is difficult to build a good inductor using analog VLSI, and so we must operate with only resistances, capacitances, and transistors. Even large resistances are difficult to produce in a standard CMOS VLSI process.

As stated earlier, some of the main drawbacks of analog VLSI are the problems of accuracy and precision. We must accept statistical variations of device parameters, which produce "non-ideal" devices (non-linear, non-exponential, or non-whatever-we-wanted). Since we are using representations where the absolute level of a signal contains the information, we are susceptible to contamination by noise, and must be concerned about precision and repeatability of computations. Adaptive circuits can help to solve some of these problems, and we will discuss this further in the following sections.

Due to the problems with accuracy and precision at the scale of individual components, it can be very difficult to design and produce a useful large-scale analog system. It is very hard to design a large deterministic system with nondeterministic components. How can one solve this conundrum? One potential solution might be to construct our circuit using only ideal devices. Unfortunately, with currently available VLSI fabrication processes, this is not reasonable to consider. We can, however, make better devices by using some well-known design techniques. An example of such a technique is common-centroid layout, which can be used to reduce the effects of production variations such as dopant concentration gradients or oxide thickness. [Allen 87] and [Millman 79] describe circuit and layout techniques to reduce the effects of some common production variations.

Even with clever layout tricks, we still need a reliable way to control analog circuits containing only a few components, so we can use them consistently in larger circuits. We also would like to control larger scale circuits, to have them perform according to our desires. We propose to use various automatic optimization techniques to compensate for undesirable device and circuit behaviors.

1.5 Using Goals for Analog VLSI Computation

We propose to use goal-based (teleological) modeling to control analog circuits and cause them to meet our expectations for circuit performance. Our goals for analog circuits will typically be a computation that we would like to perform accurately, with a specified level of precision. We first define goals for the desired circuit behavior. We then propose an analog circuit to perform the desired task. Through analog circuit simulation, we can verify that the circuit can perform the task, at least in an approximate fashion.

We can then use goals to guide a constrained optimization process to make the circuit perform our desired task as well as possible, given the actual devices. The optimization process is a way of controlling the accuracy of an analog circuit. This technique will be discussed in more detail in Ch. 5.

A model is provided for the proper behavior of each circuit or circuit element. The provided model is the goal which we desire that the actual component will meet. We require:

- circuit components (nonideal)
- a means of controlling or adjusting the circuits
- goal-directed criteria for making adjustments

The design process becomes the combination of designing a circuit *and* devising explicit goals for how it should behave. We have the expectation that we can use the goals to guide the adjustment of the circuit subsequent to its fabrication. By making explicit demands on circuit performance, we have a metric by which we can measure our satisfaction with the circuit realization in silicon. Consequently, we have a concrete measure of error, based on the extent of goal satisfaction. We can therefore improve the quality of quantitative computation using analog VLSI.

In order to make the goal-based design process tenable, we need to be flexible in our definition of goals, to allow the meeting of our goals and the actual capabilities of our circuits. We will therefore attempt whenever possible to define goals in terms of circuit signals. We can interpret results in terms of a model for whatever we desire to compute.

The process of goal-based modeling can be applied hierarchically. Individual circuit components can be compensated using a set of goals and circuitry for compensation. These compensated components can then be composed into a larger circuit, which can be again adjusted to perform such that it meets some larger-scale goals. The satisfaction of the various levels of goals can occur simultaneously, or as a sequential hierarchical process.

These goal-based techniques facilitate modular circuit and system design strategies for analog VLSI similar to commonly used techniques for design of digital VLSI systems. Realistically, it's more complex than that, since even compensated analog components are still imperfect. This work is, however, the first step on the path towards the goal of being able to reliably design and produce analog circuits to perform complex computations.

1.6 Thesis Roadmap

Ch. 2 discusses accuracy and precision, the difference between them, and why this is important to analog computation. We also discuss some issues of complexity in simulation, particularly using analog computing. Ch. 3 discusses how we represent numbers and functions as analog signals, and also how we extract numbers (answers) from analog signals.

Ch. 4 describes several circuit building approaches for increasing the precision of analog computation. Ch. 5 describes techniques for using constrained optimization to increase the accuracy of analog computation. We also introduce the goal-based design methodology in Ch. 5. In Ch. 6, we use the goal-based design methodology to construct some compensatable components (specifically, a family of multipliers). We also describe the hierarchical compensation process used to combine three compensated multipliers to compute a dot product. In Ch. 7, we use the components developed in Ch. 6 to construct a small system. The system is a collection of circuits that use gradient descent to solve a system of constraint equations which orthonormalize an approximate rotation matrix.

In Ch. 8, we describe some key differences between continuous and discrete optimization strategies. We present an implementation of circuits that perform on-chip continuous multi-dimensional optimization through gradient estimation, descent, and annealing.

Ch. 9 discusses the contributions of this thesis as a whole, and identifies future areas of potentially productive work that may proceed from this thesis.

Chapter 2

Approaching Quantitative Computation in Analog VLSI

The intent of this chapter is to describe one way to think about computation, and why we believe it to be appropriate for some types of quantitative analog computation for computer graphics and neural networks. We also discuss accuracy and precision and how these qualities guide our thinking about computation with analog VLSI. Although there are numerous ways to think about computation, we focus on those that have been useful in developing the strategies described in this thesis.

Discrete information theory and complexity of discrete computation has long been used to assist in algorithm development for digital computers. Given the current level of interest in analog computation in general and analog VLSI in particular, there is a surprising dearth of research results in the area of analog information and complexity theory. The field of analog information theory is fragmented, and has yet to produce any truly strong results. We wish to explore analog information theory and continuous complexity to guide us in our work designing analog circuits.

A rigorous treatment of analog information theory is beyond the scope of this thesis, but we feel that we must briefly describe the landscape of existing work in this area. To motivate our discussion of analog complexity, we discuss a few results in that area, and draw some of our own conclusions. We begin by describing what we are hoping to achieve, and the process by which we go about it.

2.1 Goals and Representations for Computation

An appropriate first step in performing any computation is the choice of goals. The main question that we are asking is: What to compute? We must first explicitly decide what it is that we wish to compute, before we can begin to make other decisions about the computation, let alone perform the actual computation itself.

Many researchers have determined that integrating goal-setting behavior into the design process has many benefits [Ullman 92]. We encourage this approach, and desire to make the choice of goals explicit, not tacit. Explicit and detailed enumeration of design goals is a powerful tool for producing successful projects. We don't mean to imply that most designers create products without having goals beforehand. We simply wish to emphasize that there is great value in making the goal-setting process as explicit and detailed as possible. The more detailed and well-defined the goals are, the better we are able to evaluate our results.

It is helpful to explicitly record goals at the initiation of a project and update them as the project evolves and the goals change. Explicit goals help to provide a yardstick for the success of the project, and to keep the project on-track. Goal-setting is an intrinsic part of our design approach, so it is forced to be explicit, and that is not a bad thing. The existence of explicit goals makes it clear to us where we are succeeding, and where we are failing in our objectives.

The goals determine what the end product of the computation will be. Making an explicit goal for a computation is like making a business plan for a corporation. The goals and plans determine the path to be taken. The goal-setting process forces the details to be worked out in advance.

After the goals have been chosen, one must choose a representation or framework in which to express the goals, and organize the computation. We must also be concerned about being able to measure the satisfaction of the goals within the chosen representation. And, we must be concerned about our ability to pose the problem accurately within the chosen representation. It is also important that the representation permit efficient computation toward the goals, unless we have infinite resources.

Thus, the complexity of the computation is also an important issue, both in terms of the innate complexity of the problem as well as the complexity of computation on the chosen substrate, within the chosen representation for the problem.

2.2 Complexity of Analog Computation

Although we will not present any new results in complexity theory, we believe that a brief review of some existing work will set the stage for our discussion of analog computation. We hope to see future research address the problem of developing a better measure of information in analog signals. It would then be possible to develop a measure of complexity based on that definition of information. Such a measure of analog complexity would be very useful in comparing the potential benefits of analog computation as opposed to digital computation. In the absence of such criteria, we will make a preliminary comparison of the computations of some analog circuits with their digital counterparts. We will make use of some work in the area of discrete information theory and complexity to help us to think about analog complexity.

Discrete information theory deals with ideal mathematical models, and not real physical sources and real physical channels. Nonetheless, the theory is useful in that it provides a framework within which to consider more detailed models of real systems. The theory also helps to give us a feel for the type of cost/performance tradeoffs that are involved in real computation problems. As an example, *mutual information*, as defined in [Gallager 68], serves to provide us with a measure of how much the occurrence of a particular event tells us about the probability of occurrence of some other event. Mutual information thus provides us with a tool to assist us in predicting the probable behavior of a system, given partial knowledge. We can use information theoretic tools to build an understanding of space-time and other resource tradeoffs to be made in computation. We are particularly interested in tradeoffs between accuracy, precision, and resources such as time and circuit size (silicon area).

Traub's Information-Based Complexity (IBC) studies the computational complexity of infinite dimensional problems [Traub 88]. Such problems arise in continuous mathematical models as are often used in engineering, particularly when integration, optimization, and/or differentiation are involved. Since digital computers can represent only a finite set of numbers, then many continuous inputs and outputs can only be approximated. The input is often contaminated by the discretization, so the original continuous mathematical problem may be only approximately solved.

IBC considers the tradeoff between the cost and the quality of the approximation. If we consider analog computation to be continuous in nature, then IBC has some relevance. For the simulation of continuous models, analog computation may be more suitable for obtaining an acceptable approximation with a small cost.

2.2.1 Complexity for "continuous" Analog Signals

For continuous signals, a measure of information content and complexity should be affected by the overall "size" of the signal. A truly continuous mathematical signal has effectively infinite bandwidth, and therefore by some measures it has infinite information capacity. It may be possible to develop an information theory for infinite capacity channels, but it is not directly useful for analog computation. Any physically realizable signal has a finite bandwidth and therefore finite information capacity.

For a real physical circuit, there is a limited information processing capacity. Each signal is composed of a finite number of electrons. Therefore, we can apply discrete information theory to analog signals using electrons as an ensemble. We believe that this may be an important direction for future research. At the high end of the bandwidth, there is a limit due to the slew rate limits of the circuit. At the low end of the bandwidth, there is a limit due to the number of electrons and the consequent number of transitions that can occur.

Although we have stated that there is a dearth of work in the area of analog complexity theory, there is some work. Vergis et al [Vergis 86] discusses the complexity of mechanical analog computation for a very restricted class of problems. Their analysis explicitly rules out quantum mechanics, other probabilistic behavior, and nonlinear devices as not applicable to their analysis. By eliminating nonlinearities, Vergis explicitly excludes those things that we are most interested in exploiting.

Vergis proves for this restricted class of devices that analog computers cannot solve NP-complete problems using less than polynomial resources, where resources include time, space, and material. Vergis describes problems that are "easy" to solve using digital computers, but are inherently difficult for analog computers. We assert that there exist problems, especially those of a continuous nature, for which the converse is true.

An example of such a problem is the classical physical N-body problem, with gravitational or electrical forces. The naive algorithm for solving for the interaction forces has a computational complexity of N^2, where N is the number of bodies. An improved, hierarchical algorithm has a complexity of order $N \log N$, due to the properties of the spatial hierarchy. An approximate solution can be obtained in linear time [Greengard 88]. A mechanical analog computer can calculate the exact solution in real (linear) time. The exact mechanical analog computer that we would choose is the actual N body mechanical system! From this example, we can draw the following conclusion: for simulation of continuous mathematical models, there may be special advantages for analog computation. The N-body problem, when analyzed in these terms, also brings up an interesting related question. For a circuit or system that operates in continuous time, what does it mean to require polynomial resources? Can we consider time to be infinitely divisible? In this thesis, we won't attempt to answer any of these questions, but they give some insight as to why the analysis of analog complexity is hard.

2.3 The Importance of Precise and Accurate Computation

Although we might prefer exact computation, we must usually settle for an approximation. Given that we accept approximate computation, there are two important issues that should concern us: accuracy and precision. These concepts describe the quality of the approximation. We describe accuracy and precision first in terms of definitions, and then in terms of computation and simulation. Figure 2.1 provides one view of the difference between accuracy and precision.

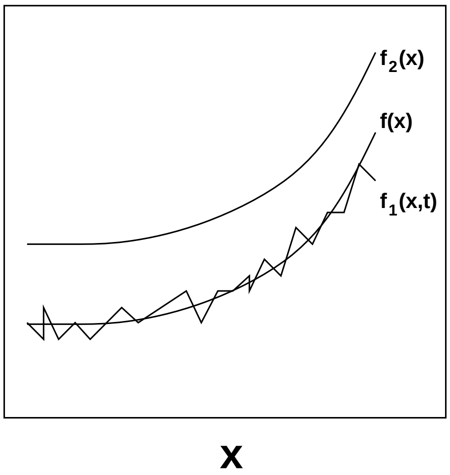

Figure 2.1: The difference between accuracy and precision. The curve $f(x)$ is approximated by the curves $f_1(x,t)$ and $f_2(x)$. Although $f_1(x,t)$ is more accurate than $f_2(x)$, as an approximation to $f(x)$, it may be that $f_2(x)$ is more precise. □

In Fig. 2.1, the function that we wish to represent is $f(x)$. One representation, $f_2(x)$, repeatably produces identical values with successive measurements. Although the function $f_2(x)$ is very precise (perhaps infinitely so), it does not accurately represent $f(x)$. In Fig. 2.1, $f_1(x,t)$ does not repeatably produce the identical response with successive measurements. $f_1(x,t)$ represents the function $f(x)$ plus some additive noise contribution:

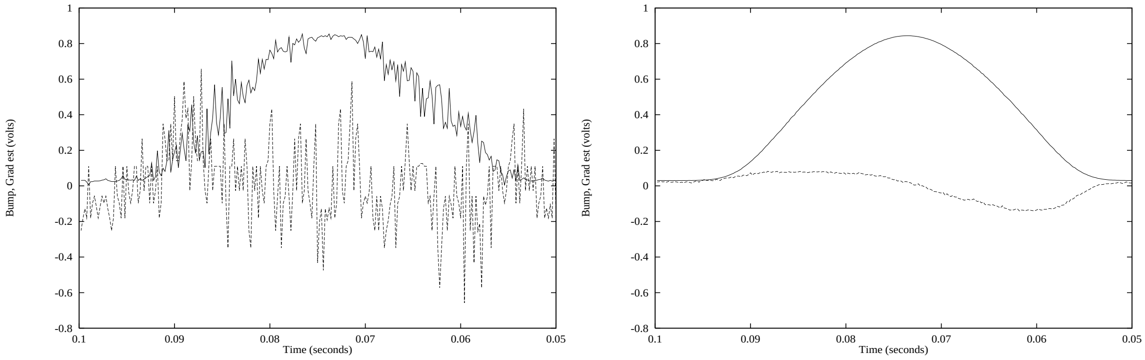

Figure 2.2: An example of a potentially accurate but imprecise measurement (Chip data for 1D Gradient Estimate). The curves above are data measured from the gradient estimation chip described in Ch. 8. (left) raw data, and (right) average of 1024 runs. Although the curves on the left are an imprecise representation of a bump function and its gradient, we see from the averaged curves on the right that the representation is moderately accurate: we can see the bump function and its qualitative gradient estimate. □

$$f_1(x,t) = f(x) + \alpha n(t). \tag{2.1}$$

The function $f_1(x,t)$ is a more accurate representation of $f(x)$ than is $f_2(x)$, but $f_1(x,t)$ is less precise. Averaging together successive measurements could improve the precision of $f_1(x,t)$, and we will discuss this technique in Ch. 4.

Fig. 2.2 shows some data measured from a chip that we built which is described in Ch. 8. The data is presented here as an example of a measurement that is somewhat accurate, but very imprecise.

Precision can be defined as "the degree of agreement of repeated measurements of a quantity" and, as such, is a quantification of how much useful information is present in a measurement. A commonsense definition for precision is "knowing what you've got, to some degree of exactness." Precision is based on a determination of how much useful information you have.

Accuracy can be defined as "the degree of conformity to some recognized standard value" and, as such, is a quantification of how closely a measurement agrees with our expectations. A commonsense definition for accuracy is "getting what you want." Accuracy is based on some concept of what result you should get.

As a practical matter, accuracy is often expressed as a binary judgement: either a representation or computation is accurate, or not. Precision is measured and discussed in many different ways, however. One way of measuring precision is as a ratio of the smallest representable signal to the largest representable signal. We could also be more concerned about the repeatability of measurements, and therefore express precision as the size of the noise floor, or its ratio to our signal range. Finally, in discrete computations, we might choose to think of precision in terms of the smallest amount by which we can reliably increment a signal, and still perceive the altered signal as different. Each of these ways of considering precision is appropriate for different ways of thinking about computation. When we are discussing precision in the remainder of this thesis, we will be careful to describe which definition we are using.

2.3.1 Feynman's Treatment of Accuracy and Precision in Physics

Feynman [Feynman 82] discusses some of the problems associated with simulation of physics using computers. Using a general model of discrete (digital) computation, Feynman describes the simulation of "discrete" physics, a fiction created for the purposes of the discussion. He also describes some simulation problems that are ill-suited to be solved using discrete computation. Among these problems are those that are continuous in nature and defy discretization. Specific examples are probability based problems which suffer irrecoverable loss of precision in the discretization, and thus do not accurately reflect the original model. Quantum mechanical simulation on digital computers can also lead to paradoxical and contradictory solutions.

2.3.2 Accuracy and Precision as Goals

Accuracy and precision are not important to us as goals unto themselves. Accuracy and precision are important to us inasmuch as they help us to satisfy the broader goals of our computation; i.e., to ensure that our simulation reflects what we are attempting to model. This is primarily an issue of accuracy, with precision a subordinate goal. We are more concerned, in general, that our simulation retain the desired character of the thing being modeled, than the precision of the result. However, given that the model is accurate, we also desire as much precision as is practical, so that we may know as much about the model as we are able.

We will pose goals for the accuracy of our computations, and will meet them with the precision with which we are able to measure the satisfaction of the goals. Thus, the precision is largely a side-effect. In addition, we can choose a representation for our signals in which the precision is maximized in the region of the goals. Issues of signal representation are discussed in more detail in Ch. 3.

2.4 Summary

We have presented a cursory overview of some important ideas in information theory and complexity. We have also provided a few examples of work toward extending these ideas to analog complexity, and a glimpse of the difficulties involved.

The nature of physical computation involves compromises and approximations, and we can consider these approximations in terms of accuracy and precision of the computation. In this chapter, we have provided definitions for accuracy and precision, and examples of the application of those concepts. In the next chapter, we will discuss the representation of numbers and functions in both analog and digital computers. In Ch. 4, we will discuss techniques for increasing the precision of analog computation. We will consider accuracy in more detail in Ch. 5.

Chapter 3

Representing Numbers and Functions in Analog and Digital Computation

This chapter discusses some ways to think about how numbers and functions can be represented and used, both in digital and analog computing. We first briefly mention features of representing numbers in digital computing, and then describe how analog representations may differ. We also discuss techniques for mapping mathematical numbers and functions into analog circuit quantities for the purpose of analog computing. Finally, we describe how to extract information from an analog computation, in order to return to the mathematical world to interpret our results.

3.1 Representing Numbers in Digital Computation

In digital computation, we can build precision progressively by assembling hierarchies of bits. The first, lower-order bit represents a quantity half of the size of the next higher bit. By stringing together a sequence of N bits, we can represent 2^N unique numbers. There may be an implied (tacit) offset and scale factor involved in the interpretation of the bits.

For integers, the tacit scale factor is one. We may also assume or provide some other scale factor, to represent numbers over some range. The combination of bits then provides some amount of precision within that range. The key feature of the approach of stringing together bits is that each additional bit divides the size of the uncertainty in half.

We may also attempt to use strings of bits to represent real numbers in a digital computer. One problem with this attempt is that the real numbers are infinite, while the string of bits represents only a finite subset. We must choose some way to map the real numbers onto the subset that we can represent with our bit string. One mapping that is commonly used is floating point. In addition to the original string of bits, there may be an explicit scale factor.

The scale factor enables us to represent numbers at various scales. One drawback of this representation is that the absolute precision varies with different scale factors. This difference in absolute precision can cause problems in computation. For real numbers, we can think in terms of using operators such as $+, -, *, /$, etc. We must approximate these operations as applied to our discrete representation of numbers for digital computation.

We now turn to examining the problems involved in representing numbers in analog, and compare and contrast with digital representations.

3.2 Representing Numbers in Analog Computation

We'll avoid attempting to represent integers exactly using analog signals and proceed to consider the representation of the real numbers. A number, as represented in analog, may be a voltage, a current, or some other electrical quantity.

We mostly consider a direct representation of a number as a single physical quantity, rather than as an aggregate as in the use of a string of digital bits, although we describe a thought experiment using "analog bits" in Sec. 4.1.2. Ch. 4 discusses signal mappings which preserve precision, and considers multi-wire, multi-signal mappings.

Unlike the rigid nature of bits, analog signals may assume any value in some range, to the precision permitted by the size of the charge on an electron, or other physical parameters. Rather than choosing a one or zero (on/off) as in a digital representation, we are interested in the value. Unlike digital, it is difficult to error-correct an analog signal, or provide a "restoring force" to protect the integrity of the signal, since it is exactly the tenuous level of the signal that interests us. We are therefore much more susceptible to measurement errors, noise, and other distortions of the signal. We must also be concerned about the repeatability of the measurements. Any time-varying noise that is introduced into the computation or the measurement affects our precision. So, instead of arbitrary, but fixed, discretizations (digital), we are faced with arbitrary and variable errors (analog).

3.3 Representation of Functions in Analog and Digital Computation

Functions can be thought of as "numbers that change over time," or that change as some other parameter changes. For functions, we can think in terms of operators and functions such as $y' = f(y)$, $g(y)$, etc. In digital computation, functions are subject to the discretization of the numeric representation (floating point, for example). Digital implementations of functions are also subject to the discretization and errors caused by the algorithm or step size used to calculate the function.

Analog circuits compute functions of time and other input parameters as an essential part of their operation. Analog circuits are susceptible to errors based on the numeric representation, but the errors are likely to be different than those in the digital domain. It is difficult to say which representation, analog or digital, is more susceptible to discretization errors without considering a specific problem. Analog circuits are certainly more susceptible to errors due to noise and device variations, however.

It's more difficult to choose a metric for the accuracy of a function, than for a number. Correctness is evaluated in terms of a *gestalt*, or within the context of the total range, rather than in terms of individual numerical values. Among the difficult issues are deciding which parts of the domain and range of the function are most important. This topic will be discussed in more detail in Ch. 5. We will also consider issues of choosing a signal representation to improve precision for an analog computation in Ch. 4.

It is hard to map the concept of bits and building precision by strings of static signals (ones and zeros) onto analog computation. For analog circuits, it is meaningful to think of the units of computation as functions to be composed together to form a hierarchy. A computation is embodied in the connection of the functions, which together compute the desired result. Accuracy is provided by the proper choice of the building blocks to implement the functions, and in their use.

In Ch. 4, we will explore several ways of getting more precision from analog circuits. The issue of precision in analog computation is often related to measurement error, and producing repeatable measurements. Consequently, many of our discussions in Ch. 4 will focus on reducing noise susceptibility. We will consider device variations and offsets more in the context of improving the accuracy of analog computation, which we will describe in Ch. 5. Within the world view of computation as a composition or hierarchy of functions, we can use goal-based modeling to improve the accuracy and precision of computation.

3.4 Solving Differential Equations more Precisely

Much research in computation involves attempts to solve differential equations more precisely. As discussed in Ch. 2, there are reasons why this will always be difficult for certain types of problems, especially when using discrete numeric representations and digital computers. Given that we are attempting to model and simulate some "thing," one of the first abstractions that is made is from the model to the differential equations which describe its behavior. As described in [Barzel 92], the next conceptual step lies in the solution of the equations.

A key component of the solution is the numeric technique chosen to use in order to solve the equations, as well as the type of computation to use (analog or digital). Often, problems are formulated as differential equations, which are then solved using some discrete step technique implemented on a digital computer. An alternative approach is to cast the differential equation into an analog circuit. We hope that we can obtain more accurate solutions of differential equations using analog circuits.

Mahowald and Douglas [Mahowald 91] described an implementation of a "silicon neuron" which faithfully models some of the electrical behavior of biological neurons. The electro-chemical behavior of neurons is quite complex and requires significant processing time to simulate on a digital computer. Due to the processing time required, it is unreasonable to attempt to simulate a large population of neurons on a single digital computer. Mahowald thus set out to produce an analog VLSI analog of a neuron that could be used to simulate a large network of neurons.

The approach that was used is extremely interesting; the physical properties of the transistors are used to directly simulate the physics of the electrochemical reactions of the actual neurons. Each of the ion currents, sodium, potassium, and chlorine, is modeled using a few transistors. These rates are exponential functions of the reversal voltages and the membrane voltage, and are thus quite suitable for implementation using transistors. This direct modeling of a set of differential equations using analog transistors is an an excellent example of the potential of analog VLSI to solve differential equations for many other applications.

It is possible to generalize from the ideas behind Mahowald's neuron, to produce components that can be configured and used to solve other problems involving differential equations.

3.5 Mapping Numbers and Functions to and from Analog Signals

Analog circuits don't compute with real numbers. Analog circuits compute using current and voltage values that change over time. Everything in analog circuits is charge-based, including current and voltage. Current is charge movement over time. Voltage is a potential difference with respect to some reference, based on a quantity of charge. Electrical charge is available only in discrete quanta; charge is not continuous. Even so, we can often use the continuous approximation to think about the operation of circuits.

With the present device geometries and subthreshold transistor operation, the number of electrons involved is relatively small. Given a certain computation speed and permissible power level, there are concrete limitations to the possible precision. There are 1.602×10^{-19} Coulombs/electron, and 1 Ampere is 1 Coulomb/second. So, 1 Ampere consists of 6.24×10^{18} electrons/second. 1 nano-Ampere consists of 6.24×10^{9} electrons/second. For a computation at the nano-Ampere level, completed in 1 microsecond, we have on the order of 6.24×10^{3} electrons to work with. At this level, a difference of one electron per microsecond changes the current by 0.16%!

This small number of electrons poses a potential problem for analog circuits in the near future, as device sizes continue to shrink, and operating voltage ranges narrow as well. In fact, given the analysis in the previous paragraph, it is not surprising that noise and precision are problems with analog computing with current geometries, particularly operating at the subthreshold level. The small number of electrons involved in an analog computation indicates that our choice of signal representation will be of paramount importance in determining the quality of our analog computation.

We must find ways to reliably and repeatably map from real numbers and real-valued functions to circuit quantities: we call this process the *number to signal mapping function*, "G."

We must worry about the useful range, in voltage or current, within which a circuit can operate, as well as what precision is needed for the signal. Explicit consideration of these factors as part of the circuit/computation design can increase the precision and accuracy of our results. We can then use an analog circuit to compute something of value. After performing the computation, we must find some way to extract the useful information from the results of the computation. This extraction involves computing a robust inverse mapping G^{-1}, which we can use to convert voltage and current signals back to numbers and mathematical functions (or floating point or fixed point digital representation, for digital computers).

We must first choose the range of the representation. Some issues that should concern us as we make this choice relate to characteristics of accuracy and precision requirements. If we expect the signal to vary "linearly" over some fixed range, or "exponentially" over a larger range of scales, we will choose differing signal representations. These decisions are related to choices made in digital computation about number representation using fixed point or floating point.

One of our biggest decisions relates to absolute/relative error criteria and the range of signal. Voltages may range from some ground reference to Vdd, which is typically a 5 volt range, and our absolute precision is typically on the order of millivolts. So, a useful computing range for voltage is about 3 orders of magnitude.

The useful computing range for currents spans more orders of magnitude than that of voltage. Our current signal noise floor is on the order of pico-Amperes, and we can consider computing with currents as large as micro-Amperes or milli-Amperes. So, we have a useful computing range of 6-9 orders of magnitude. Thus, as a rule of thumb, currents are a more appropriate representation for signals that have a larger range of operation.

We have other choices to make in determining our signal representations. For example, we must choose the limits of the computation representation. Often, we will want to choose some other value than zero (0 volts or 0 amperes) to represent a mathematical zero for the purposes of our computation. The value of "zero" that is appropriate for integrating a gradient signal over time is not necessarily 0 volts, and this issue is discussed in Ch. 7. The choice of "zero" is frequently important if we wish to represent negative quantities, particularly with voltages, since we cannot easily use voltages below ground (0V) in a silicon chip. We can choose to represent values with respect to some reference other than zero, providing a differential signal. We also may not know the ideal value for our reference in advance, and will have to discover it as part of our optimization process. We will see why unknown reference values are important, and some techniques for determining appropriate reference values in Ch. 8.

As in the previous chapter, we will prefer to think of functions as "numbers that vary over time" or that vary as some other parameter(s) change. We will, therefore, assume that all of the issues pertaining to mapping numbers to signals are also relevant to mapping functions to analog signals.

There is some analogy in the operation of our number to signal mapping function G to the oft-lamented "scaling problem" in the analog computers from the 1940s and 50s [Tomovic 62] [Soroka 54]. The scaling problem deals with choosing analog signals to represent numbers. We are addressing a similar need to map a desired computation onto some substrate. Thus, the scaling problem is a subset of the problem we are addressing. An important aspect of our work is that we will be concerned with accuracy and the quantitative character of our computation, and we will attempt to enforce it through the use of goal-based design techniques.

In addition to the classical analog computing scaling, to make the computation *doable* using our computation substrate, we wish to choose a representation which allows our goal-based design methodology to improve the accuracy of our computation. As an auxiliary goal, we are interested in a mapping that provides the greatest precision in those regions where we are most concerned about the results. These issues relate to the desire to use analog for quantitative computation in addition to qualitative computation. We will be using the optimization process to help us to more closely approach an "optimal" scaling and representation, rather than one that merely works. So, in some ways, the work in this thesis is an extension of work done on the scaling problem in analog computers.

As discussed at the beginning of Sec. 3.5, we use some general rules of thumb to get us started thinking about signal mappings. Currents can provide greater accuracy over orders of magnitude (relative). Thus, currents are good for "log-like" signals that span many orders of magnitude. Voltages can provide greater accuracy over smaller ranges, and voltage errors often manifest themselves as additive noise, or *absolute* error.

Although these properties are not absolute, they serve as a good guideline to start the signal mapping process.

3.5.1 Extracting Numbers from Signals

After mapping a number or function to an analog electronic signal and performing some computation, we need to interpret the resulting signals to obtain the result of our computation.

The inverse of our number to signal mapping function, G^{-1}, is defined by our choices for the number to signal mapping function G, and the computations that we have performed. G^{-1} is not strictly a mathamatical inverse of our G function, unless the circuit computation is the identity function. G^{-1} is a convenient label, however, since the signal to number mapping function is *conceptually* an inverse operation.

The scaling and representation choices that we have made (using our various number to signal mapping functions) in forming our analog inputs provide implicit scaling and offsets in the computation. Other signal representations and transformations within the analog circuit add additional scale factors, offsets, and distortions. So, the inverse function G^{-1} is implied by the aggregation of all of these scalings and offsets. The inverse function is required to "undo" those parts of the number to signal mapping function and analog processing that are not part of the desired computation. The inverse functions may be implicit in the measurement of an output voltage or current. There are also explicit components, in terms of applying offsets or scaling, to produce the proper outputs.

All of the choices are made in the selection of the input mapping functions, and the inverses are constrained by the earlier choices.

We can "correct" errors in a hierarchical fashion, with the aid of our G and G^{-1} mappings. For instance, in Ch. 6, we will describe the process of correcting errors in an analog multiplier, and then correcting errors in the use of a collection of multipliers to compute a dot product, and so on.

3.6 Summary: Finding Accuracy and Precision within Analog VLSI

Most of the material in this thesis relates to finding more accuracy and precision within analog VLSI. We will begin with imperfect, variable devices, and examine techniques for creating designs which are either tolerant of the defects, or can be adjusted to compensate for the errors.

As discussed in Sec. 1.1, analog VLSI transistors are a rich computation substrate. We would like to be able to make use of the compact, fast computation capabilities of analog VLSI devices. We are interested in computing with continuous, time-varying signals, usually arising from some real-world source. The slew-rate limits give a bound on what value a signal can take at a particular time, thus are related to Lipschitz conditions. We do not take advantage of the Lipschitz analogy in this thesis, but believe that it presents an opportunity to be investigated in future research.

By choosing continuous computation, we are trading some benefits for some costs. A major benefit of continuous computation is that a continuous signal is smooth, without the discontinuities inherent in a discrete representation. This smoothness allows and encourages a direct representation, for example, of computational time as real time. One drawback of this direct representation is that there is no way to differentiate noise from the actual signal. Therefore, care must be taken to choose a representation in which the expected scale of the noise is not detrimental to the desired computation. Ch. 4 describes some circuits that may be used to increase precision by reducing susceptibility to noise.

Chapter 4

Precision from Analog VLSI, or, "knowing what you've got"

We wish to obtain great advantages from the fact that analog signals are closer to a continuous representation than are digital signals. These advantages come with an associated cost in terms of noise and related precision problems. In Ch. 3, we described some preliminary ideas about representing numbers and functions as analog signals.

We now consider approaches for obtaining greater precision in analog computation. Specifically, we present a thought experiment involving several multi-wire signal representations for analog computation. We also provide a zeroth-order analysis of the expected performance of these multi-wire representations. Finally, we discuss hybrid techniques which combine digital and analog components to attempt to capture the best features from both analog and digital. We also briefly describe a few example approaches for computing using hierarchies of functions, and for solving differential equations using analog VLSI.

4.1 Multi-wire Analog Signal Representations

As mentioned above, one technique for increasing the precision of representation of an analog signal is to use multiple wires to describe the signal. We describe three different techniques, and offer a brief analysis of each approach as it performs on a sample computation.

4.1.1 Signal Duplication

One of the simplest ways that one may increase the precision of analog VLSI computation is to duplicate the computation, and average the results. We use tools from Monte Carlo estimation theory in analyzing the approach of duplicating computation. The duplication can be performed either by integration over time, or by actually performing the calculation multiple times discretely, perhaps with multiple copies of the circuit.

For integration over time, it is sufficient to integrate over a long enough time span such that the time scale of the variations is usually much smaller than the time scale of the integration. In practice, this can be costly in terms of the performance of the computation. The time required for integration may require that the computation is slowed to an unacceptable level. Nonetheless, using small circuit capacitances for smoothing of noise is an effective technique, and we utilize it in Ch. 8.

Let us turn now to the consideration of duplicating the entire circuit, and averaging the results. In our analysis, we make use of some results from Monte Carlo estimation theory.

Given a set of samples $x_1, x_2, \ldots, x_n$, we define the *sample variance*[1] σ^2 as

[1] There are several possible ways to define sample variance, and we have chosen a definition which is, in general, an unbiased estimator.

$$\sigma^2 = \sum_{i=1}^{n} (x_i - \bar{x})^2 / n \tag{4.1}$$

where $\bar{x}$ is the sample mean.

So, if we assume that the samples are *independent*, and *normally distributed* around the desired result, the sample mean $\bar{x}$ will approach the desired result as n, the number of samples, approaches infinity. In theory, duplicating and averaging the results of analog circuits might seem to be an easy way to gain more precision. In fact, the theory described above indicates that the *standard deviation* of the averaged result (σ_{averaged}) would be reduced by a factor of $\sqrt{n}$:

$$\sigma_{\text{averaged}} = \sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2 / n} = \frac{\sigma}{\sqrt{n}} \quad . \tag{4.2}$$

Unfortunately, given real circuits, we cannot count on reaching this limit. The problem lies in the assumptions made in the previous paragraph, in the words "independent" and "normally distributed." Much of the noise in analog circuits on a particular chip may in fact be correlated, since it can be caused by noise from the power supply, for example. Also, even if some device variations are normally distributed, the effects of those variations on the computation may be distributed very differently.

We can, however, expect that the duplication of analog circuits will reduce the amount of noise, since at least *some* of the noise will be uncorrelated. Likewise, if we choose our designs so that the center of the distributions of possible device variations produces our desired result, the average of a collection of computed analog results can be expected to exhibit greater precision.

Let us assume that some fraction of the noise is correlated between the multiple copies of our circuit, and that the remaining noise is uncorrelated, and ignore for the moment the effects of non-normal distributions of effects from device variations. We will denote the fraction as a parameter α, which can range from 0 to 1. The correlated noise will introduce a *bias*, or systematic error, which will not be ameliorated by the duplication of the circuits. We will assume that, excluding the effects of the noise, the circuits compute x flawlessly, without any device variations. We therefore represent the x_i as

$$x_i = \bar{x} + \alpha c_i + (1 - \alpha) u_i \tag{4.3}$$

where c_i is the noise which is correlated between copies of the circuit, $\bar{x}$ is in this case the actual mean, rather than the sample mean as above, and u_i is the uncorrelated noise. Note that we assume that $c_i = c_j$ for any pair of i and j. We also assume that the uncorrelated noise u_i is zero-mean.

We now represent the noise-based sample variance as

$$\sigma^2 = \sum_{i=1}^{n} (\bar{x} + \alpha c_i + (1 - \alpha) u_i - \bar{x})^2 / n \tag{4.4}$$

or, simplifying,

$$\sigma^2 = \sum_{i=1}^{n} (\alpha c_i + (1 - \alpha) u_i)^2 / n \tag{4.5}$$

and, we represent the resulting standard deviation as

$$\sigma = \sqrt{\sum_{i=1}^{n} (\alpha c_i + (1 - \alpha) u_i)^2 / n} \quad . \tag{4.6}$$

In the limit as n grows large, the standard deviation will now converge to the scale of αc_i, the correlated noise contribution.

(4.7)
$$lim_{n \to \infty} \sigma = \alpha c_i \quad .$$

So, if $\alpha = 0.5$, we could expect the precision to increase to a limit of twice as good as without duplication. Certainly, as part of a design using this strategy, we would need to consider diminishing returns, and choose an appropriate amount of duplication, short of infinite.

4.1.2 "Analog Bits" and Coordinate Charts

Another approach to increasing precision using multiple wires could involve partitioning the signals. There are a few ways that we may consider to perform the partitioning. The first is analogous to the hybrid analog/digital circuits that will be described in Sec. 4.2, and is also similar to the encoding of a number into bits. The encoding is based on successive approximations: the first wire carries a rough (analog) representation of the signal, and the next wire carries a rough representation of the error from the first signal, and so on.

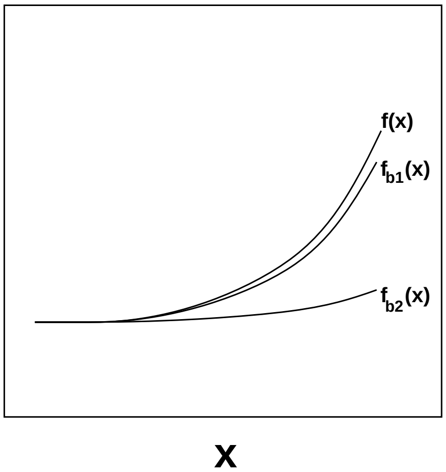

Figure 4.1: A multi-wire analog signal representation, using "analog bits." $f(x)$ is approximated by $f_{b1}(x) + f_{b2}(x)$, over the full range of x. □

Figure 4.1 describes an example of the "analog bits" representation. $f_{b1}(x)$ is a rough approximation of the desired signal $f(x)$. $f_{b2}(x)$ is an attempt to represent the error produced by using

$f_{b1}(x)$ to approximate $f(x)$. Our "analog bits" representation will use $f_{b1}(x) + f_{b2}(x)$ to represent $f(x)$. These wires can represent signals at different scales, as with bits, although the scale factor does not have to be two. Another way to think of the analog bits representation is that the functions $f_{b1}(x)$ and $f_{b2}(x)$ are successive approximations of the function $f(x)$. [Kerns 92b] discusses multi-wire signal representations of this type, and provides some discussion of implementation issues.

The second partitioning scheme that we will consider is a larger departure from digital encodings. We can divide the range of our representation into two or more parts, and use separate parts of the circuit to handle the separate parts of the signal.

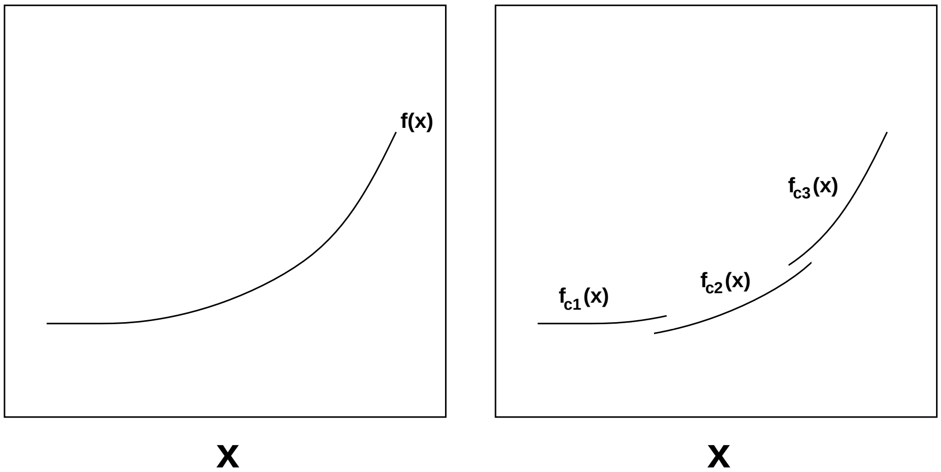

Figure 4.2: A multi-wire analog signal representation, using coordinate charts and a partition of unity. $f(x)$ is approximated by one of $f_{c1}(x)$, $f_{c2}(x)$, or $f_{c3}(x)$, at various part of the range of x. ($f_{c2}(x)$ is offset vertically for clarity) □

We will call this representation a coordinate chart based range decomposition technique, and will refer to the circuitry for processing each subrange as a "range block" or simply, a block. This range decomposition technique is based on ideas from differential geometry, including coordinate charts. Over the range of the function $f(x)$, we can use several component functions to represent $f(x)$. The individual mappings from the component functions to the function $f(x)$ are called *coordinate charts*. The blending function which permits combining the coordinate charts to represent the function $f(x)$

is called a *partition of unity*. The collection of coordinate charts that, combined using a partition of unity, represent a function is called an *atlas*.

For each portion of the range of the independent variable x, there is at least one of the functions $f_{c1}(x)$, $f_{c2}(x)$, or $f_{c3}(x)$ defined. Taken together, the three functions approximate the target function $f(x)$. In order for this range decomposition technique to be an improvement over using a single circuit to represent $f(x)$, the combination of $f_{c1}(x)$, $f_{c2}(x)$, and $f_{c3}(x)$ need to be a better representation, in this case, by providing greater precision.

There are a few reasons for us to believe that this greater precision may be possible. First, by judicious choice of the ranges of the input x for each of $f_{c1}(x)$, $f_{c2}(x)$, and $f_{c3}(x)$, we can ensure that the range of the functions is reduced from the full range of $f(x)$. We can therefore apply (as part of our number to signal mapping "G" function) a separate scale factor for each of $f_{c1}(x)$, $f_{c2}(x)$, and $f_{c3}(x)$. Thus, we can use the extra available range to increase the precision of calculation within each range block. This is most effective if the errors are primarily additive in nature. A similar range decomposition technique is often used in software for numerical computation of functions in libraries.

It is reasonable to expect that a piecewise composition of a function can more accurately represent that function. A cost of this coordinate chart based range decomposition approach is that we must separately engineer the computation paths for each range of the independent variable x, to maximize our precision, and prevent "hitting the rails." So, in addition to requiring duplicates of the circuits, we now also require duplication of the design work, at least to the extent of simulating different operating ranges.

We also require extra circuitry for detecting the appropriate range block(s) to use, blending between the chosen range blocks, and routing the proper signal to our outputs. This leads to another potential weakness of the coordinate chart based range decomposition technique. Given a selection of independent input variables, we can easily imagine a product space developing, for the range blocks from each set of coordinate charts. Then, to perform a computation using functions based on the coordinate chart representation, we would have a geometric growth in the number of circuits required. Therefore, care must be taken to re-use circuit blocks whenever possible, to control the growth of the circuits.

4.1.3 Range Decomposition Block Implementation

We will describe an implementation that provides some of the underpinnings of the range decomposition representation scheme, and proceed with some thought experiments about how such circuits might be used. The circuit fragments that we describe in this section implement a portion of a partition of unity. They implement a blending of a set of component functions.

First, we'll discuss the implementation of one of the underlying mechanisms for this representation.

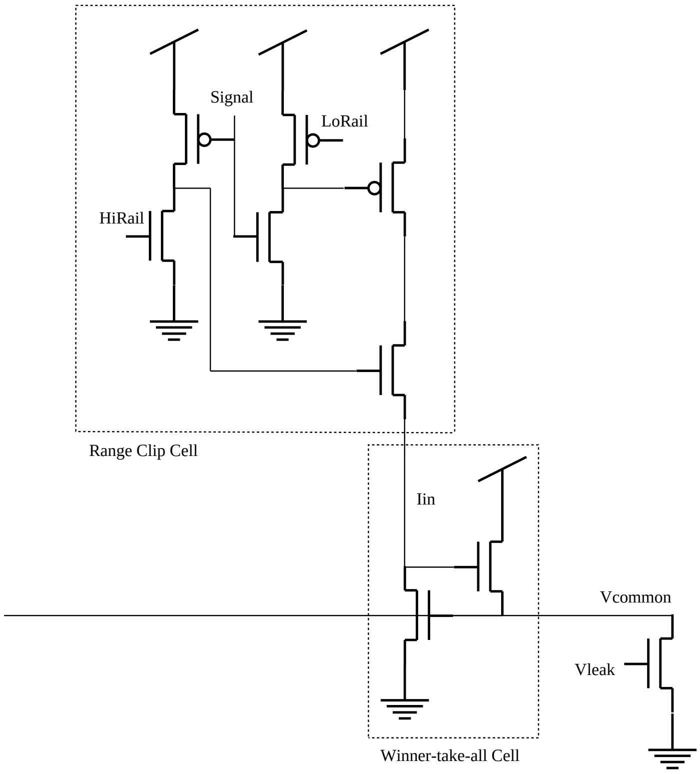

Figure 4.3: A single block of a precision enhancing circuit. □

Figure 4.3 shows a circuit diagram of one block of the coordinate chart based range decomposition scheme. The signal for the coordinate chart is a voltage connected to the *Signal* input. The *HiRail* and LoRail inputs determine the active range of the coordinate chart element. If the signal is within the range set by *HiRail* and LoRail, then there is a current at *Iin*, the input to the winner-take-all circuit. For example, assuming equal strength P and N transistors, if *HiRail* is set to 2.0 Volts and LoRail is set to 3.5 Volts, then the signal is within the range of the coordinate chart when *Signal* is between 1.5 and 3.0 Volts. *HiRail* sets the upper limit of the coordinate chart range to (5.0 - *HiRail*), given a 5 Volt power supply. Likewise, *LoRail* sets the lower limit of the coordinate chart range to (5.0 - *LoRail*).

Figure 4.4 shows a circuit diagram of how multiple blocks of the range decomposition scheme would be connected together. Each "precision block" implements one coordinate chart, with its own *Signal*, *HiRail*, and LoRail to set the range. The voltage on each *Iin* wire encodes the winner(s), which indicate which coordinate chart(s) is/are in range. These *Iin* signals can be used to control the blending of the coordinate charts.

We require a circuit to priority encode the N precision range blocks, and to select which block(s) is/are active. The circuit should identify which coordinate chart or charts are "in range," and disable the rest.

We choose to construct a "winner-take-all" based selection circuit. The WTA circuit [Lazzaro 89] selects a winner from among N currents, and the largest current is the winner. It encodes the winner by driving the voltage high on the input (current) wire. A changing input voltage is somewhat of a problem for this application, since that change in voltage affects the input currents, but the problem is easily solved by isolating the actual input currents from the WTA winner selection. We add a few additional transistors to isolate the input currents from the output voltages. So, we have an 8-transistor per precision stage selection circuit, as shown in Fig. 4.3.

Given a set of (coordinate chart) voltage values, the coordinate chart circuit does the following:

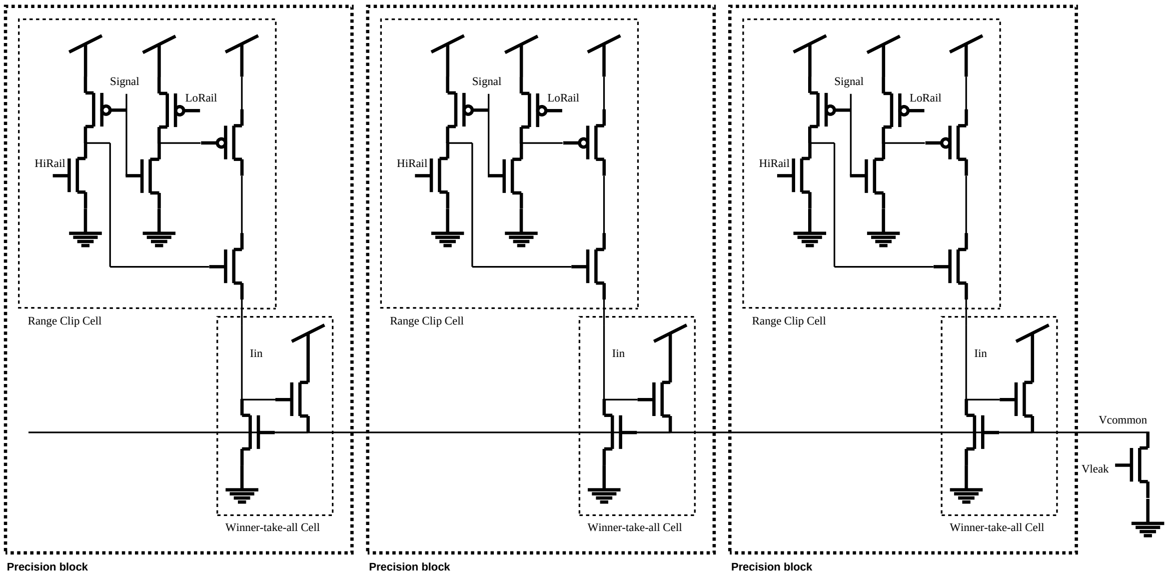

Figure 4.4: A 3 stage precision enhancing circuit. □

- If ALL of the voltages are outside of the operating range (settable, but, for example, 1-4 volts), the behavior is undefined: anyone can win, based on individual transistor offsets.

- if exactly one voltage is in range, it wins decisively (winner output voltage = 3V, loser output voltages = 10nV).

- if more than one voltage is in range, they are weighted exponentially by how close they are to the center of the range.

Based on these behaviors, it seems that the ranges must overlap in order for the representation to work without discontinuities. For this circuit, the encoding of the coordinate chart blending will be a voltage level wire (*Signal*) and an enable wire (*Iin*) for each coordinate chart-based precision stage. Our representation is reasonably compact. We use two two-transistor amplifiers to do the limit checks, two transistors to isolate the inputs, and two transistors for the winner-take-all.

4.1.4 An Example Problem for the Multi-wire Signal Representations

We perform a thought experiment, applying the three precision enhancing schemes to an example problem. We then provide a rudimentary comparison of the three approaches. Let us assume that we wish to compute a function like

$$f(x) = e^x \sin x \tag{4.8}$$

in some region $x > 0$.

Let us assume further that we have additive noise in our circuits, and that a fraction α of the noise is correlated between circuit components (due to power supply, perhaps). We will, as in Sec. 4.1.1, describe the noise function $\eta(t)$ as being composed of a correlated component $c(t)$ and an uncorrelated component $u(t)$

$$\eta(t) = \alpha c(t) + (1 - \alpha)u(t) \tag{4.9}$$

and we add subscripts to indicate noise from different circuits or subcircuits. We also assume that the correlated noise sources $c_i(t)$ are, in fact, correlated, so that $c_i(t) = c_j(t)$ for any i and j.

In our analysis, we consider the effects of various values for the "noise correlation coefficient" α. To simplify the analysis, we do not consider multiplicative noise or individual transistor variations. We refer to the computed functions as $f_s(x,t)$ for the straightforward single circuit approach, $f_d(x,t)$ for the duplicated signal approach, $f_b(x,t)$ for the "analog bits" approach, and $f_c(x,t)$ for the coordinate chart based approach.

Straightforward Approach

So, a straightforward implementation (assuming that we can construct a circuit that accurately approximates our function $f(x)$), will produce the following result

$$f_s(x,t) = f(x) + \beta\eta_1(t) \tag{4.10}$$

where β is the scale of the noise, and $\eta_1(t)$ is an instance of the noise as a function of time, as described above, in Eq. 4.9.

Signal Duplication Approach

For the signal duplication scheme, the expected value of the computed function is also a function of the number of duplicated circuits n:

$$\begin{aligned} f_d(x,t) &= f(x) + \tfrac{(\alpha c(t)\sqrt{n} + (1-\alpha)u(t))\beta}{\sqrt{n}} \\ &= f(x) + \alpha\beta c(t) + (1-\alpha)\tfrac{\beta u(t)}{\sqrt{n}} \end{aligned} \tag{4.11}$$

remembering, from above, that a fraction $(1-\alpha)$ of the noise is uncorrelated, so that only that fraction of the noise is reduced by the duplication.

"Analog Bits" Approach

For the "analog bits" approach, we compute two functions, $f_{b1}(x,t)$ and $f_{b2}(x,t)$, which we add together to produce $f_b(x,t)$.

$$f_b(x,t) = f_{b1}(x,t) + f_{b2}(x,t) \quad . \tag{4.12}$$

We desire that $f_b(x,t)$ approximate $f(x)$. So, we begin by letting

$$f_{b1}(x,t) = f_s(x,t) \tag{4.13}$$

since $f_s(x,t)$ is already a reasonable approximation.

In Sec. 4.1.2, we defined $f_{b2}(x,t)$ as a representation of the error between $f(x,t)$, and $f_{b1}(x,t)$. So, we can use $f(x)$ (which we are given) and $f_{b1}(x)$ (which we can measure) to choose $f_{b2}(x,t)$.

$$f_{b2}(x,t) \equiv f(x) - f_{b1}(x,t) = f(x) - (f(x) + \beta\eta_1(t)) \quad . \tag{4.14}$$

We might choose the composition of $f_{b2}(x,t)$ based on a feedback and optimization scheme, perhaps using an external digital computer. We will discuss such optimization techniques in more detail in Ch. 5. Unfortunately, when we construct $f_{b2}(x,t)$, we get a different noise function $\eta_2(t)$, instead of $\eta_1(t)$. Thus,

$$f_{b2}(x,t) = f(x) - (f(x) + \beta\eta_2(t)) = -\beta\eta_2(t) \quad . \tag{4.15}$$

So, we base $f_{b2}(x,t)$ on the "error" between $f(x)$ and $f_{b1}(x,t)$, which reduces to the scaled noise function $\beta\eta_2(t)$, meaning that we would like to subtract out the noise (a perfectly reasonable goal). The resulting function $f_b(x,t)$ is

$$\begin{aligned} f_b(x,t) &= f_{b1}(x,t) + f_{b2}(x,t) \\ &= (f(x) + \beta\eta_1(t)) + (f(x) - (f(x) + \beta\eta_2(t))) \\ &= f(x) + \beta\eta_1(t) - \beta\eta_2(t) \\ &= f(x) + (1-\alpha)\beta(u_1(t) - u_2(t)) \end{aligned} \tag{4.16}$$

since $f_{b2}(x,t)$ cancels out the correlated noise $c(t)$ from $f_{b1}(x,t)$. The expected value of the "analog bits" representation is

$$E[f_b(x,t)] = f(x) + (1-\alpha)\beta u(t)/\sqrt{2} \tag{4.17}$$

since the two uncorrelated noise functions $u_1(t)$ and $u_2(t)$ are, by definition, not correlated.

Because a portion α of the noise is correlated and $(1-\alpha)$ is not, the "analog bits" approach in this case produces a result whose quality is highly dependent on the value of α. If α is approximately equal to one half, the precision of the analog bits approach is approximately equal to the naive approach, $f_s(x,t)$, as can be seen from Eq. 4.16. If, however, more of the noise is correlated, the analog bits approach would produce a precision improvement, since $f_{b2}(x,t)$ will cancel out more noise than it adds.

Coordinate Chart Approach

It is more complicated to analyze the coordinate chart based approach, and we make a few more assumptions. There are several possible benefits of the coordinate chart representation, but we only exploit one of them in our thought experiment. For a real circuit, it is reasonable to assume that a piecewise representation, as in the coordinate chart approach, permits us to more accurately compute the function. In [Toumazou 90], Gilbert discusses the merits of piecewise function approximation for computing trigonometric functions in analog, specifically ECL, circuits. Since we have stated that our circuit already accurately computes $f(x)$ (with noise added), our thought experiment does not explore the accuracy improvements that are possible through this part of the coordinate chart approach.

The other part of the coordinate chart approach is more applicable to precision improvements. Since the coordinate chart based approach subdivides the input domain, we can choose sub-domains of the independent variable x which reduce the output range of $f(x)$. We then increase the scale factor applied to the computation of $f(x)$ within our limited input domains. For example, for the function $f(x) = e^x \sin x$, the output range of $f(x)$ is small near zero. So, for x small ($< \delta$), we can use a function $f_{c1}(x)$ which has an offset d_1 and a scale factor, $k_1 > 1$. For $x > \delta$, we can use a function $f_{c2}(x)$, which has a different offset d_2 and scale factor k_2. For the purpose of this analysis, we will assume that the offsets d_i are perfect, and do not introduce any additional noise or imprecision.

Since the noise $\eta(t)$ is additive, the scale of the noise may not change.[2] So, since the absolute scale of the signal has been increased and the scale of the noise remains constant, we have increased the precision by changing the relative scales. If we assume that we can rescale the computation of $f(x)$ so that the signal size is twice as great ($k_1 = k_2 = 2$), then we have doubled the precision of

[2] We are assuming that scaling the computation of $f(x)$ does not also scale the noise, and this would be true if the noise is measurement noise, but may not be true for other possible noise sources.

the computation.

$$f_c(x) = \begin{cases} f_{c1}(x) = k_1 f(x) + d_1 + \beta\eta_1(t) & \text{if } x < \delta \\ f_{c2}(x) = k_2 f(x) + d_2 + \beta\eta_2(t) & \text{otherwise} \end{cases} \tag{4.18}$$

Comparing the Approaches

Using Eq. 4.11, Eq. 4.16, and Eq. 4.18, we calculate the predicted precision improvement for the three multi-wire signal representations, for a collection of values of α, the fraction of noise that is correlated. We present the results in terms of the relative scale of the noise given the representation, compared to the single circuit approach. Note that numbers greater than one indicate that the technique actually makes the precision worse, and 1.00 indicates no net change. Smaller numbers are better.

representation	Relative Scale of Noise				
	$\alpha = 0$	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$	$\alpha = 1$
$f_s(x,t)$	1.00	1.00	1.00	1.00	1.00
$f_d(x,t)$ $n = 2$	0.71	0.78	0.85	0.93	1.00
$f_d(x,t)$ $n = 4$	0.50	0.63	0.75	0.87	1.00
$f_d(x,t)$ $n = 8$	0.35	0.51	0.68	0.84	1.00
$f_b(x,t)$	1.41	1.06	0.71	0.35	0.00
$f_c(x,t)$	0.50	0.50	0.50	0.50	0.50

4.2 Using Analog and Digital VLSI Together

Another paradigm for computation involves using analog and digital components together. By combining the two approaches, it is possible to obtain some of the benefits from both. (Of course, it is also possible to combine the drawbacks of both, but hopefully we can avoid that outcome!) Since there are many ways to combine analog and digital computing and we only consider a few, we emphasize that we do not consider the treatment in this thesis to be a definitive treatise on hybrid analog/digital representations. We merely provide a few examples to widen the view from purely analog representations.

For solving a differential equation, we can use the same basic idea as Mahowald, using simple circuits of only a few transistors to approximate some function. The analog circuit doesn't have to be particularly accurate or precise. It must only compute its approximation quickly and deterministically, as its inputs change.

We can then add digital circuitry to compute cooperatively with the analog circuitry. The digital computation can serve as a taskmaster for the analog computation. We can use some digital evaluation of the error to guide the analog computation. An example of this is as follows:

- an analog circuit solves a differential equation over time, although perhaps inaccurately or with low precision.
- A digital circuit (or even a microprocessor) can monitor the progress of the analog circuit, checking its output against some metric.

As the analog simulation begins to diverge excessively from the desired solution, the digital circuit can restart it from a new position, or simply provide control inputs.

In addition to using analog/digital hybrids to control a large computation such as solving a differential equation, we can use the same technique on a more local scale. Some small amount of digital circuitry can be used to calibrate and choose parameter settings for the analog circuitry. The hybrid may be thought of as a unit, and also as a multi-wire representation of a single signal.

The hybrid analog/digital circuit can potentially be more accurate and precise than the purely analog version, and may run faster and be smaller than a purely digital implementation. We can also explore this concept on a macroscopic scale, using an external digital computer to set parameters for analog circuit operation. Ch. 5 describes the use of this technique to increase the accuracy of the performance of a few simple circuits.

In addition, we might imagine that this hybrid analog/digital correction process operates continuously through time. We can then image a computation such as a control system, which operates with real-time, real-world inputs, and can correct its own errors as it operates. Ch. 8 describes some experiments using feedback and an entirely analog circuit to estimate gradients and perform descent in multiple dimensions. The gradient descent circuit is an example of this concept of "in-flight error correction."

4.3 Summary

In this chapter, we have described several multi-wire signal representation techniques, designed to increase the precision of analog computations. We have also presented a brief thought experiment using these representations to compute a function, and analyzing, to first order, the expected precision improvement over a naive single wire signal representation.

In the next chapter, we will turn our thoughts to issues of accuracy. We will describe some ideas for designing and compensating analog circuits for greater accuracy for some designated computational task. We will also present the results of some experiments that we have performed to test our ideas. Finally, we will draw some conclusions from our experiments and propose a design methodology that can reliably improve the accuracy of analog computation.

Chapter 5

Accuracy from Analog VLSI, or, "getting what you want"

5.1 Accuracy through Goal-based Design

This chapter discusses an approach for producing analog VLSI circuit elements which compute functions accurately. The basis of the approach is *teleological*, in that it operates by setting goals to be met by the circuit. We describe techniques for choosing the goals, as well as optimization methods to satisfy the goals.

An interesting related issue becomes apparent during the course of the exposition in this chapter. The issue is, "how does one express goals?" If the goal is to produce a particular number, then the goal pursuit and satisfaction involve comparing two numbers. The goal evaluation becomes potentially more difficult if the goal relates to a behavior, which may be expressed as the set of values of a function. An example of such a behavior is the frequency response of a circuit. Further complications arise when the behavior is a function of time, and therefore cannot be evaluated as a time-independent curve. An example of such time-dependent behavior is a neuron's response to some stimulus, a spike train which changes over time.

In this chapter, we discuss the choice of goals and optimization strategies for static values and time-independent curves, but identify the problem of evaluation of time-dependent behaviors as a challenging research topic, for future work.

5.2 Constrained Optimization Applied to the Parameter Setting Problem for Analog Circuits

We use constrained optimization to select operating parameters for two circuits: a simple 3-transistor square root circuit, and an analog VLSI artificial cochlea. This automated method uses computer controlled measurement and test equipment to choose chip parameters which minimize the difference between the actual circuit's behavior and a specified goal behavior. Choosing the proper circuit parameters is important to compensate for manufacturing deviations or adjust circuit performance within a certain range. As biologically-motivated analog VLSI circuits become increasingly complex, implying more parameters, setting these parameters by hand will become more cumbersome. Thus an automated parameter setting method can be of great value [Fleischer 90]. We wish to provide an "electronic screwdriver" that we can use to adjust operating parameters. Automated parameter setting is an integral part of a *goal-based engineering design methodology* in which circuits are constructed with parameters enabling a wide range of behaviors, and are then "tuned" to the desired behaviors automatically.

Constrained optimization methods are useful for setting the parameters of analog circuits. We present two experiments in which an automated method successfully finds parameter settings which cause our circuit's behavior to closely approximate the desired behavior. These parameter-setting experiments are described in Sec. 5.5. The difficult subproblems encountered were (1) building the electronic setup to acquire the data and control the circuit, and (2) specifying the comparison of the goal to the measured circuit behavior in a mathematical form suitable for the optimization tools. We describe the necessary components of the electronic setup in Sec. 5.3, and we discuss the selection of optimization technique toward the end of Sec. 5.5.

Automated parameter setting can be an important component of a system to build accurate analog circuits. The power of this method is enhanced by including appropriate parameters in the initial design of a circuit: we can build circuits with a wide range of behaviors and then "tune" them to the desired behavior. In Sec. 5.8, we describe a comprehensive design methodology which embodies this strategy.

5.3 Implementation

We have assembled a system which allows us to test these ideas. The system can be conceptually decomposed into four distinct parts:

circuit: an analog VLSI chip intended to compute a particular function.

target function: a computational model quantitatively describing the desired behavior of the circuit. This model may have the same parameters as the circuit, or may be expressed in terms of biological data that the circuit is to mimic.

error metric: compares the target to the actual circuit function, and computes a difference measure.

constrained optimization tool: a numerical analysis tool, chosen based on the characteristics of the particular problem posed by the circuit (may vary with the experiment).

Figure 5.1 shows a schematic depiction of the constrained optimization tool used in a system with the error metric, target function, and a chip.

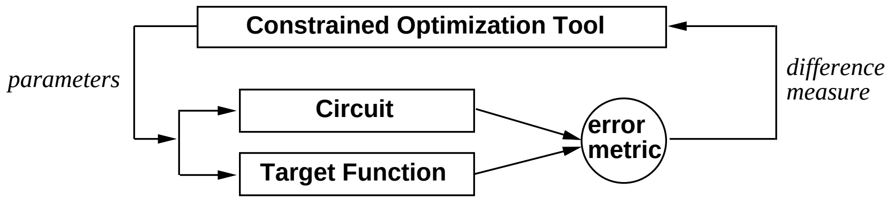

Figure 5.1: The constrained optimization tool uses the error metric to compute the difference between the performance of the circuit and the target function. It then adjusts the parameters to minimize the error metric, causing the actual circuit behavior to approach the target function as closely as possible. □

5.4 A Generic Physical Setup for Optimization

A typical physical setup for choosing chip parameters under computer control has the following elements: an analog VLSI circuit, a digital computer to control the optimization process, computer programmable voltage/current sources to drive the chip, and computer programmable measurement devices, such as electrometers and oscilloscopes, to measure the chip's response. The parameter setting system acts as an "electronic screwdriver" which allows the parameter adjustments to be made under computer control, according to some evaluated performance metric.

The combination of all of these elements provides a self-contained environment for testing chips. The setting of parameters can then be performed at whatever level of automation is desirable. In this way, *all* inputs to the chip and all measurements of the outputs can be controlled by the computer. In this environment, it becomes very convenient to write small programs to control testing and parameter setting of analog VLSI chips. The programs combined with a log of the data taken provide a valuable record of the test procedure and operating characteristics of the chip(s).

5.5 The Experiments

We perform two experiments to set parameters of analog VLSI circuits using constrained optimization. The constraints arise from the fact that we have limitations on the allowable voltage/current parameters. We may also impose other constraints, as well. The first experiment is a simple one-parameter system, a 3-transistor "square root" circuit. The second experiment uses a more complex time-varying multi-parameter system, an analog VLSI electronic cochlea. The artificial cochlea is composed of cascaded second-order section filters.

5.5.1 Square Root Experiment

In the first experiment we examine a "square-root" circuit [Mead 89], which actually computes $ax^{\alpha}+b$, where α is typically near 0.4. We introduce a parameter (V) into this circuit which varies α indirectly. By adjusting the voltage V in the square root circuit, as shown in Fig. 5.2, we can alter the shape of the response curve.

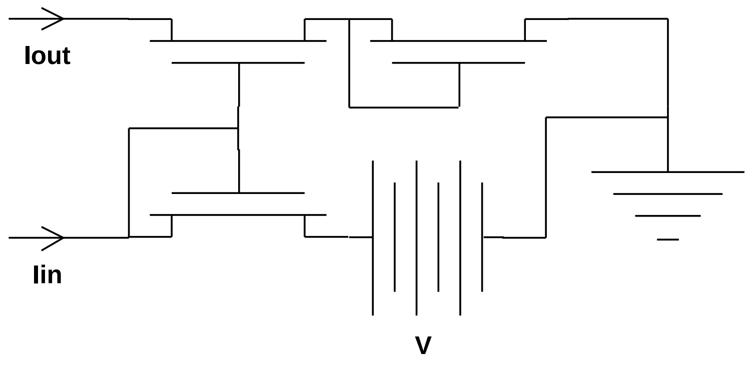

Figure 5.2: Square root circuit. This circuit produces an output current that is approximately the square root of the input current. Adjustment of the parameter V alters the shape of the input-output response curve over the operating range. □

We have little control over the values of a and b in the square root circuit, so we choose an error metric which optimizes α, targeting a curve which has a slope of 0.5 in log-log I_{in} vs. I_{out} space. Since $b << a\sqrt{x}$, we can safely ignore b for the purposes of this parameter-setting experiment. The entire optimization process takes only a few minutes for this simple one-parameter system, as implemented on a Motorola 68000-based machine.

Figure 5.3 shows the trajectory of the input-output current relation, as the parameter V is varied. Each curve represents the output current as a function of the input for a single value of V. Each

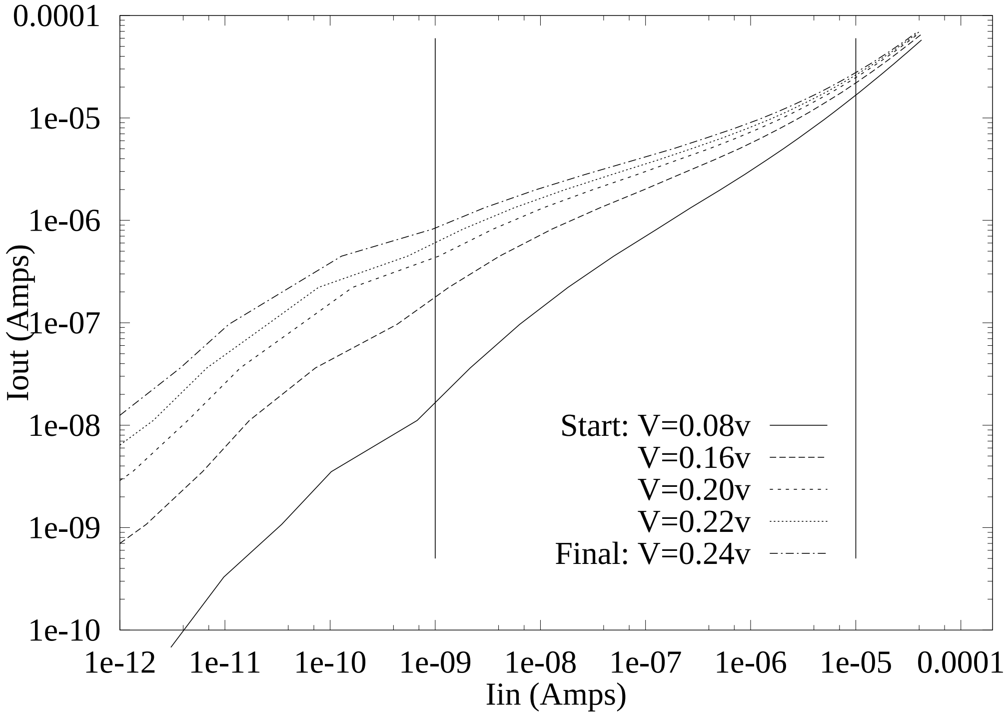

Figure 5.3: Trajectory of input-output response for optimization of square root circuit. The parameter V is varied to produce the "best" response. The set of curves depict the optimization process as the voltage V is varied to produce a more "square-root-like" response. Each curve represents the output response over the entire input range for a particular value of the parameter V. The metric that we used to measure the quality of the square root is a slope of 1/2 in log-log space. □

distinct value of V produces a different curve. The set of curves depict the optimization process as the voltage V is varied to produce a more "square-root-like" response. The metric that we used to measure the quality of the square root is a slope of 1/2 in log-log space.

Figure 5.4 shows the final results of the square root computation, with the circuit (see Fig. 5.2) output normalized by the parameters a and b.

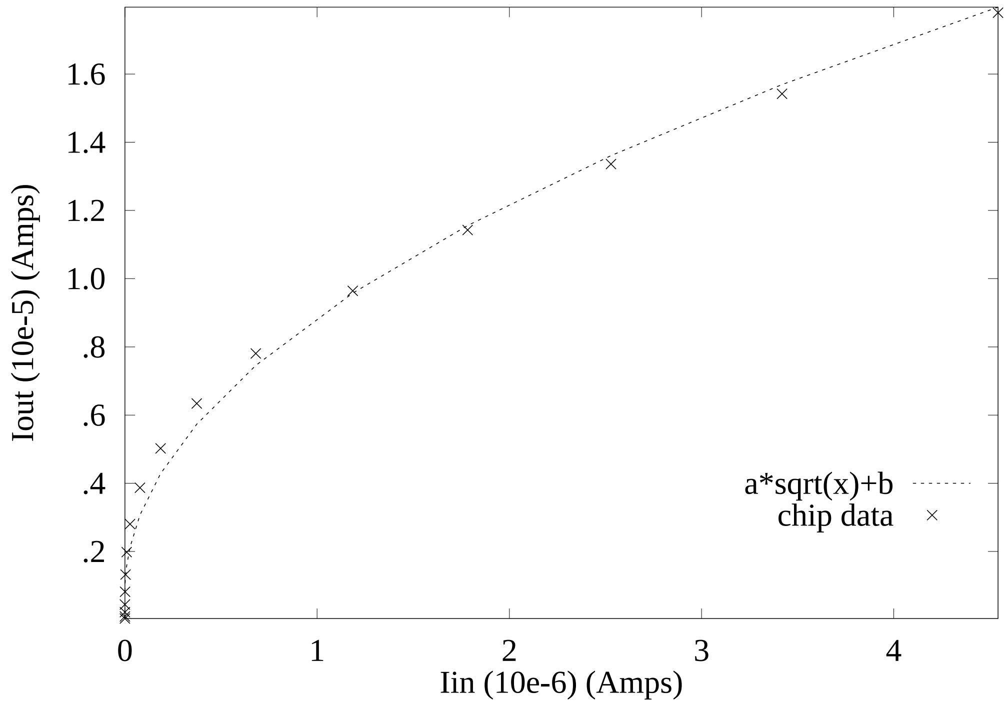

Figure 5.4: Results of optimization of square root circuit. Parameters were chosen automatically to produce a "best fit" of the circuit response to a mathematical square root. □

5.5.2 Analog VLSI Cochlea

As an example of a more complex system on which to test the constrained optimization technique, we chose a silicon cochlea, as described by [Lyon 88]. The silicon cochlea is a cascade of lowpass second-order filter sections arranged such that the natural frequency τ of the stages decreases exponentially with distance into the cascade, while the quality factor Q of the filters is the same for each section (tap). The value of Q determines the peak gain at each tap.

To specify the performance of such a cochlea, we need to specify the natural frequencies of the first and last taps, and the peak gain at each tap. These performance parameters are controlled by bias voltages V_{τ_L} V_{τ_R}, and V_Q, respectively. The parameter-setting problem for the analog VLSI cochlea circuit is to find the bias voltages that give the desired performance. This optimization task

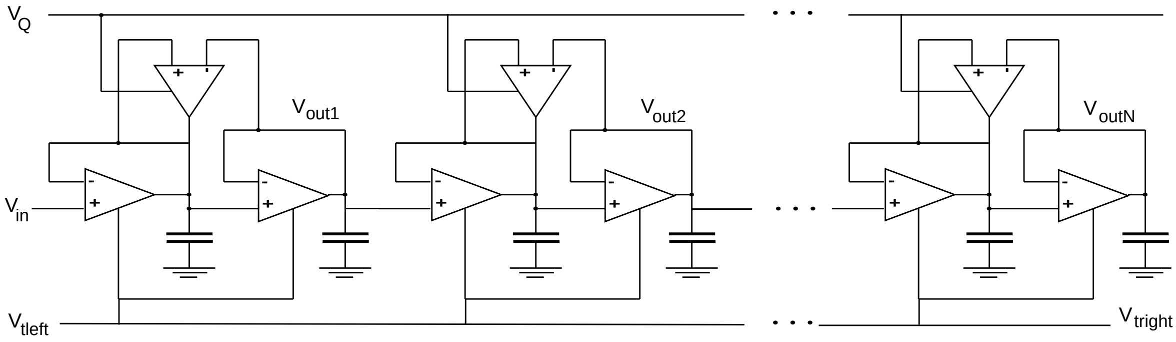

Figure 5.5: Cochlea circuit □

is more lengthy than the square root optimization. Each measurement of the frequency response takes a few minutes, since it is composed of many individual instrument readings.

Cochlea Results

The results of our attempts to set parameters for the analog VLSI cochlea are quite encouraging.

Figure 5.6 shows the trajectories of the error metrics for the first and last tap of the cochlea. Most of the progress is made in the early steps, after which the optimization is proceeding along the valley of the error surface, shown in Fig. 5.8. Also note in Fig. 5.9 that the gradient is small for the parameters for the low frequency cutoff, since the error surface is relatively flat. The effects of the interaction between the large gradient from Fig. 5.8 and the small gradient from Fig. 5.9 can be seen in the small oscillations along the trajectory in Fig. 5.6. A continuous gradient descent process would not suffer from these same problems.

Figure 5.7 shows both the target frequency response data and the frequency responses which result from our chosen parameter settings. The curves are quite similar, and the differences are at the scale of measurement noise and instrument resolution in our system.

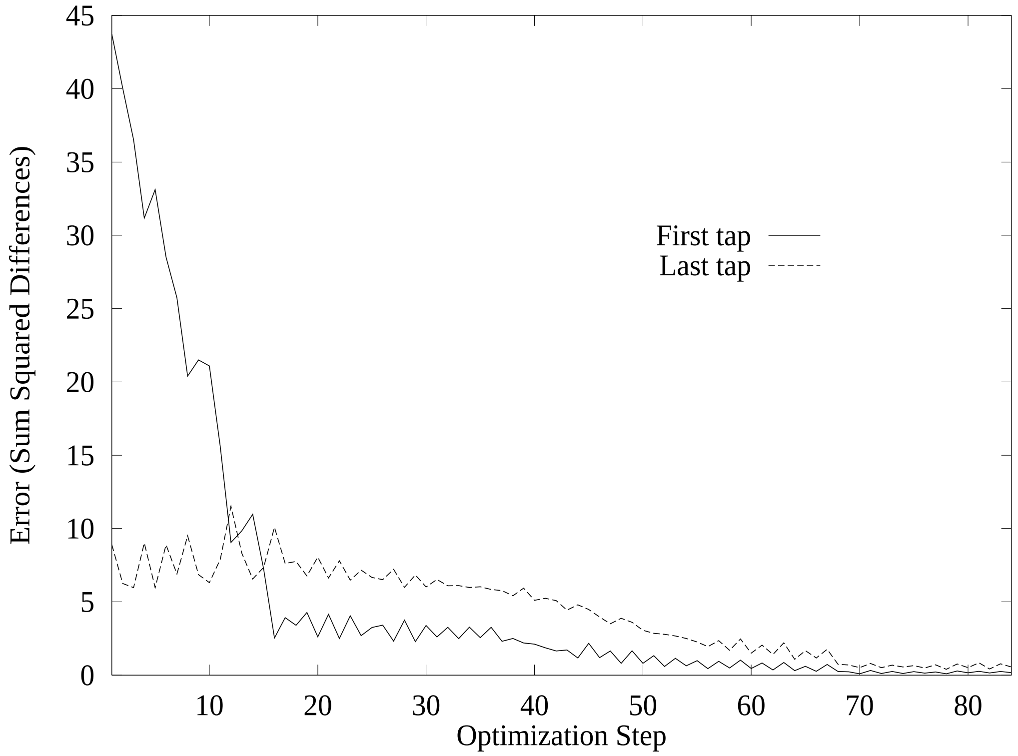

Figure 5.6: Error metric trajectories for gradient descent on cochlea □

Cochlea Optimization Strategies

We explored several optimization strategies for finding the best parameters for the electronic cochlea. We discuss two of the techniques:

gradient descent: assume that we know nothing except the input/output relation of the chip. Then we can estimate the gradient for gradient descent by varying the inputs. More robust numerical techniques, such as conjugate gradient, can also be helpful when the energy landscape is steep, with narrow valleys.

special knowledge: use a priori knowledge of the effect of each knob to guide the optimization. We use the knowledge that Vq controls the gain, and that Vtl and Vtr mostly control the left

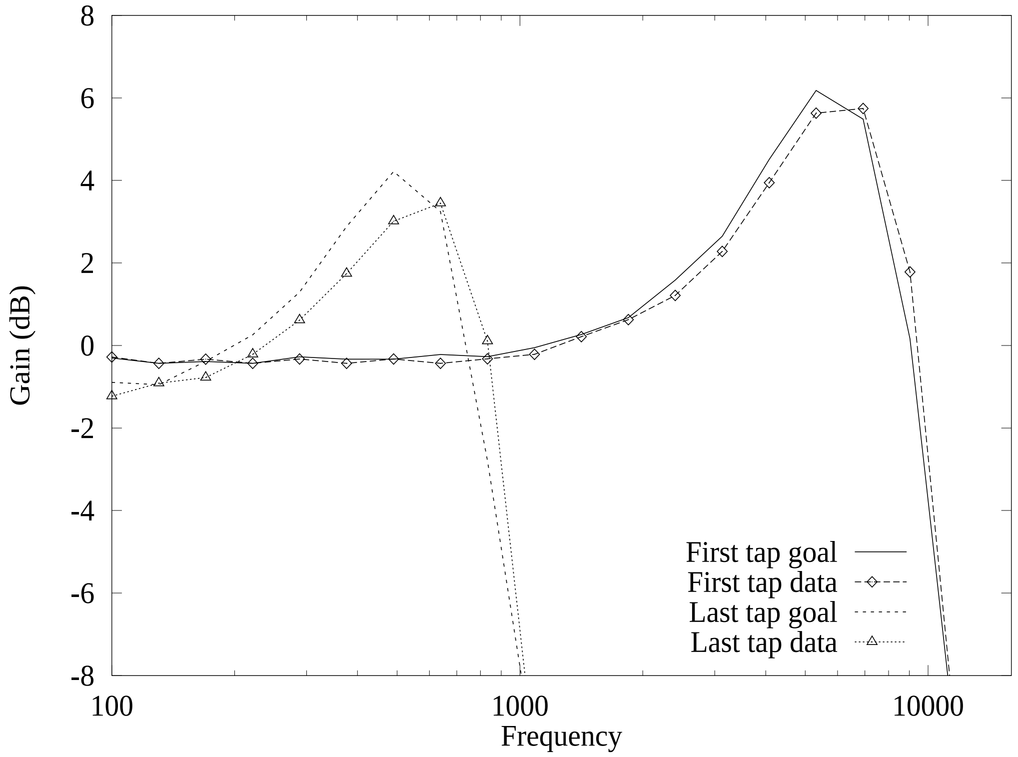

Figure 5.7: Target frequency response and gradient descent optimized data for cochlea □

and right frequency cutoffs. Using special knowledge allows for much faster optimization, but is a less general technique.

We found the gradient descent technique to be reliable, although it did not converge nearly as quickly as the "special knowledge" optimization. This corresponds with our intuition that any special knowledge we have about the circuit's operation will aid us in setting the parameters. Anytime that we possess a priori knowledge about the operation of a circuit, that information is a powerful aid to efficient parameter setting. Special knowledge can help us to avoid searching unimportant regions of the parameter space.

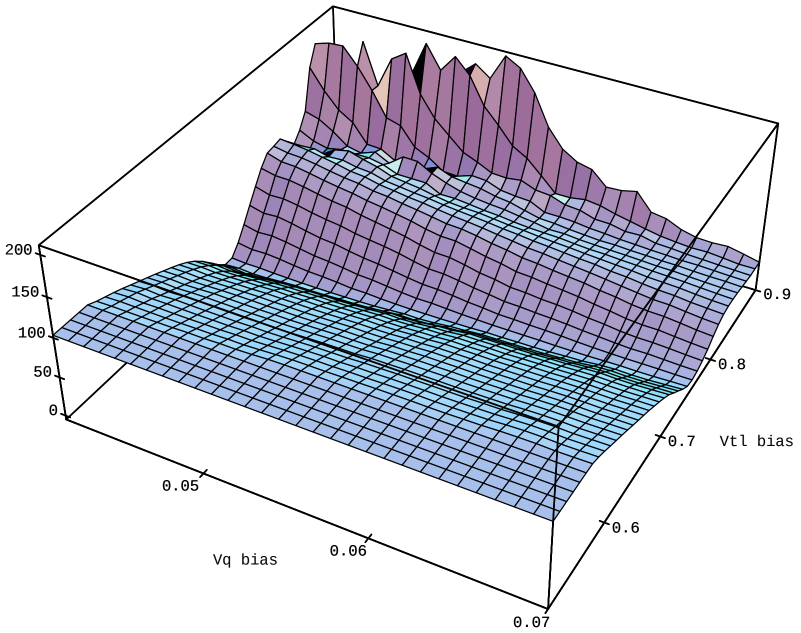

Figure 5.8: The error surface for the error metric for the frequency response of the first (high frequency cutoff) tap of the cochlea. Note the deep, narrow, angled valley in the error surface. Our target (the minimum) lies near the far left, at the deepest part of the valley. The voltages shown on the horizontal axes in this figure are relative to reference values, so should not be construed as absolute parameter settings for operation of Lyon's cochlea. □

5.6 Choosing An Appropriate Optimization Method

One element of our system which has worked without much difficulty is the optimization. However, more complex circuits may require more sophisticated optimization methods. A wide variety of constrained optimization algorithms exist which are effective on particular classes of problems (gradient descent, quasi-Newton, simulated annealing, etc) [Platt 89, Gill 81, Press 86, Fleischer 90], and we can choose a method appropriate to the problem at hand. Techniques such as simulated annealing can find optimal parameter combinations for multi-parameter systems with complex behavior, which gives us confidence that our methods will work for more complex circuits.

We don't claim that the parameter setting technique described here can automatically be extended to any complex circuit. In fact, there certainly exist circuits with many operating parameters

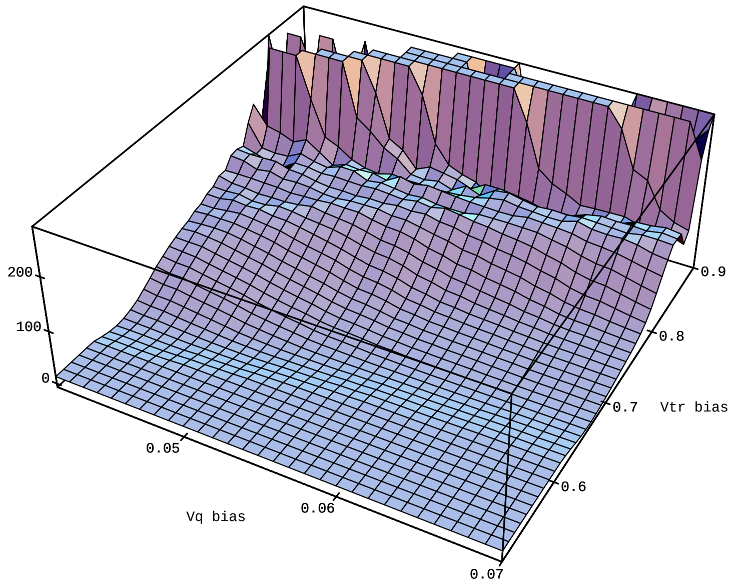

Figure 5.9: The error surface for the error metric for the frequency response of the last (low frequency cutoff) tap of the cochlea. Note the shallow valley in the error surface. Our target (the minimum) lies near the far left, at the deepest part of the valley, although the minimum is difficult to find, due to the shallow slope of the error surface. □

for which our technique would perform poorly. However, we believe that there is great potential for extending the parameter setting technique to increasingly complex circuits. We hope that future research will explore techniques for decomposing complex parameter setting problems into collections of simpler problems which we already know how to solve.

The choice of error metric may also need to be reconsidered for more complex circuits. For systems with time-varying signals, we can use an error metric which captures the time course of the signal. We can deal with hysteresis by beginning at a known state and following the same path for each optimization step. Noisy and non-smooth functions can be improved by averaging data and using robust numerical analysis techniques which are less sensitive to noise. Certainly, there exists a need for more research into the development of appropriate error metrics for parameter setting for complex circuits.

5.7 Conclusions

The constrained optimization technique works well when a well-defined goal for chip operation can be specified. We compare automated parameter setting with adjustment by hand: consider that humans often fail in the same situations where optimization fails (e.g., multiple local minima). In contrast, for larger dimensional spaces, hand adjustment is very difficult, while an optimization technique may succeed more frequently.

We expect to integrate the technique into our chip development process, and future developments will move the optimization and learning process gradually into the chip. It is interesting to note that our gradient descent method "learns" the parameters of the chip in a manner similar to backpropagation. Seen from this perspective, this work is a step on the path toward robust on-chip learning. Clearly, more work is needed to permit the evaluation of complex error metrics entirely onchip.

In order to use the constrained optimization technique, there are two moderately difficult problems to address. First, one must assemble the equipment to set circuit input parameters and record results from the circuit under computer control (e.g., voltage and current sources, electrometer, digital oscilloscope, etc). We must also provide software interfaces to the equipment, so that we can write programs to probe and measure the circuits. This is a one-time cost since a similar setup can be used for many different circuits. We can also use the setup for general-purpose chip testing, optimization, and calibration. The use of computer-controllable instruments encourages us to write reusable tools and programs for chip testing. We have found the process of chip testing by writing software to be very robust and an efficient use of the designer's time.

The second and more difficult issue is how to specify the target function of a circuit, and how to compute the error metric. For example, in the simple square-root circuit, one might be more concerned about behavior in a particular region of input, or perhaps along the entire range of operation. Care must be taken to ensure that the combination of the target model and the error metric accurately describes the desired behavior of the circuit.

The existence of an automated parameter setting mechanism opens up a new avenue for producing accurate analog circuits. The goal of *accurately* computing a function differs from the approach of providing a cheap (simple) circuit which loosely *approximates* the function [Gilbert 68] [Mead 89]. By providing appropriate parameters in the design of a circuit, we can ensure that the desired function is in the domain of possible circuit behaviors (given expected manufacturing tolerances). Thus we define the domain of the circuit in anticipation of the parameter setting apparatus. The optimization methods will then be able to find the best solution in the domain, which could potentially be accurate to a high degree of precision.

5.8 The Goal-based Engineering Design Technique

The results of our optimization experiments suggest the adoption of a comprehensive *Goal-based Engineering Design Technique* that directly affects how we design and test chips. The technique implies the use of an "electronic screwdriver" to permit the adjustment of operating parameters under computer control. We use computer-controlled instruments to effect parameter changes in the circuits. The approach encourages a habit of writing programs to be used both with a chip simulation and for optimization of the fabricated chip.

Our results change the types of circuits we will try to build. The optimization techniques allow us to aggressively design and build ambitious circuits and more frequently have them work as expected, meeting our design goals. The technique also encourages us to formulate detailed and explicit goals for circuit operation, rather than less precise tacit goals. As a corollary, due to the expectation of increased accuracy, we can confidently attack larger and more interesting problems.

The technique is composed of the following four steps:

1) goal-setting: identify the target function, or behavioral goals, of the design

2) circuit design: design the circuit with "knobs" (adjustable parameters) in it, attempting to make sure that the desired (target) circuit behavior is in gamut of the actual circuit, given expected manufacturing variation and device characteristics.

3) optimization plan: devise an optimization strategy to explore the parameter settings. This includes capabilities such as a programmable digital computer to control the optimization, software to control the parameter setting process, and computer-driven instruments which can apply voltages/currents to the chip and measure voltage/current outputs.

4) optimization: use the optimization procedure to select parameters to minimize the deviation of the actual circuit performance from the target function. The optimization may make use of *special knowledge* about the circuit, such as "I know that this knob has effect x," or personal interaction, such as "I know that this is a good region, so explore here."

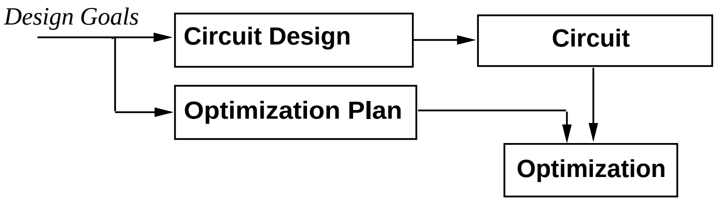

Figure 5.10: The goal-setting Process □

The goal-setting process, as shown in Fig. 5.10, produces design goals that influence both the circuit design and the form of the optimization plan. It is important to produce a match between the design of the circuit and the plan for optimizing its parameters.

Chapter 6

Applying the Goal-based Design Methodology

This chapter describes the use of the goal-based design methodology. We describe how to use the approach at the lowest level, to create circuit building blocks that can be combined together to construct systems. As part of the description, we briefly summarize the operation of two example building blocks. The first example is an offset-compensated amplifier, due to Mead and Allen, described in [Mead 90] and [Mead 91]. The second example is developed as part of this thesis, a family of compensated multipliers. In Ch. 7, we use these two building blocks to construct a computer graphics application, the solution of a rotation matrix constraint.

6.1 Circuits with "knobs"

The heart of the goal-based design methodology is the strategy of designing circuits with "knobs." By this we mean that the circuits have built-in adjustments which can be used to improve the circuit performance. We can then begin our design with more imperfect and variable components, knowing that they will be "dialed-in" to better performance later. This process can be applied hierarchically, producing larger systems or applications out of the compensated building blocks. Although many electronic circuits are designed with trimming or compensating capability [O'Dell 88], this strategy has not been consistently used to design systems using analog circuits.

It is the assertion of this thesis that a powerful design paradigm is created by *always* designing components with knobs, and planning the compensation process as part of the production of the design. An area which we identify for future research is to incorporate the consideration of possible device variations and compensatory knob selections into the design and circuit simulation process. Potential avenues of research include using interval analysis [Alefeld 83] [Snyder 92] to chart expected values of variations, and using a simulation to validate a range of possible knob selections before fabrication of the circuits.

6.2 Constructing a Compensated Amplifier

As a brief introduction to the idea of using knobs for compensation, we'll describe a compensated amplifier [Mead 90] [Mead 91]. The compensated amplifier design allows a simple amplifier design to be adjusted to perform more closely to the desired behavior.

The basic design is the well-known CMOS transconductance amplifier (transamp) Fig. 6.1. The transamp has some undesirable shortcomings. The first of these is that the operating range is very small, in that the output spans the operating range if the inputs, V_1 and V_2, differ by more than a few tens of millivolts. The second shortcoming that we wish to discuss is far more serious, and it relates to transistor variations. Since the five transistors may have different operating parameters

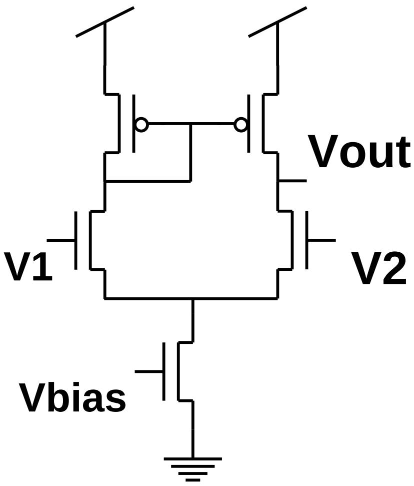

Figure 6.1: A schematic diagram of a transconductance amplifier. □

(due to fabrication process variations), the "center" of the transamp's operating range may not occur when $V_1 = V_2$. If the transamp is to be used as a follower or a comparator, these offsets can seriously impair the usefulness of the transamp.

In order to help to address the shortcomings of the transamp, Mead and Allen suggest adding one or two capacitive input structures with floating gates as shown in Fig. 6.2 and Fig. 6.3, respectively.

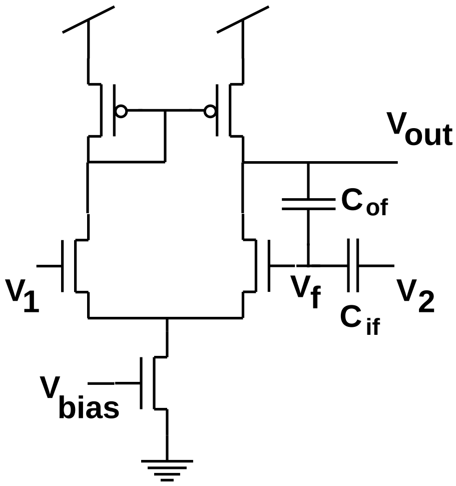

Figure 6.2: A schematic diagram of a transconductance amplifier with a capacitive input divider for offset compensation. □

There are several advantages to this approach of capacitive input offset compensation. First, one

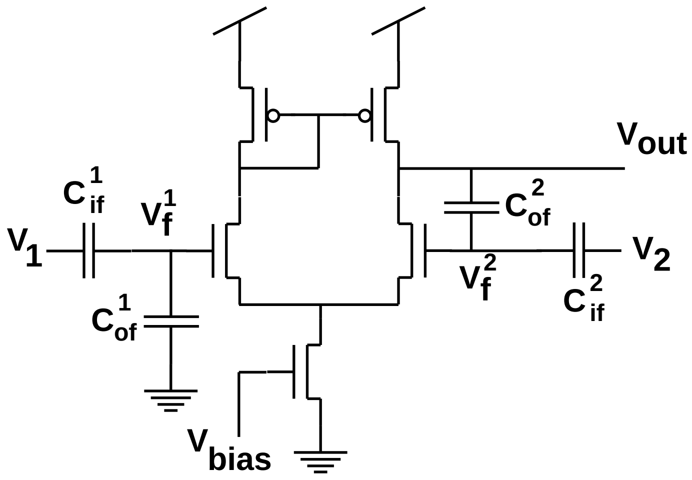

Figure 6.3: A schematic diagram of a transconductance amplifier with two capacitive input dividers for offset compensation. □

can divide the inputs to increase the output linear range. Second, at the same time, one can cancel offsets by adjusting the amount of charge on the floating gate.

The floating gate is completely electrically isolated; polysilicon is surrounded by silicon dioxide. So, we can expect that the charge stored there will remain effectively indefinitely. This permanance of charge on a floating gate is especially attractive for a device or design compensation application, since we will often wish to compensate for errors once, and retain the same setting. Methods for adding/removing charge include hot electron injection [Anderson 90], Fowler-Nordheim tunneling [Thomsen 91], and UV-induced electron transfer [Kerns 92b]. Kerns [Kerns 92b] describes some issues related to fabricating and using floating-gate compensated amplifiers, particularly by using UV-induced electron transfer to adjust the charge on floating gates. In addition, Kerns describes a small system (2nd-order-section delay line) built from a sequence of compensated amplifiers. We will describe the operation of the compensated amplifier as an aid to understanding the compensated multipliers developed in this thesis.

6.2.1 Equations of Operation

For the singly compensated wide-range transconductance amplifier, as shown in Fig. 6.2, we have the following relation between output voltage and the amplifier input voltages:

$$V_{out} = A(V_1 - V_f) \tag{6.1}$$

where A is the gain of the amplifier.

We wish to eliminate the floating gate voltage and solve the equation to provide V_{out} in terms of V_1 and V_2. The details of the derivation are in Appendix B. The resulting relation is given in Eq. 6.2.

$$V_{out}\left[1 + (1/A)(1 + C_{if}/C_{of})\right] = \quad V_1(1 + C_{if}/C_{of}) - V_2(C_{if}/C_{of}) \tag{6.2}$$

Note that the effect of the input V_1 is at a different scale than the effect of the input V_2. We can correct this by providing the capacitive division at *both* inputs. For the doubly compensated wide-range transconductance amplifier, as shown in Fig. 6.3, we have the following relation between output voltage and the input voltages:

$$V_{out} = A(V_f^1 - V_f^2) \quad . \tag{6.3}$$

where, again, A is the gain of the amplifier.

We now also have two sets of capacitors, for the two floating nodes. So, we have a pair of relations for the capacitive division of the inputs:

$$C_{of}^1(V_{out} - V_f^1) = C_{if}^1(V_f^1 - V_1) \tag{6.4}$$

$$C_{of}^2(V_{out} - V_f^2) = C_{if}^2(V_f^2 - V_2) \tag{6.5}$$

As above, the derivation is presented in Appendix B. The resulting relation is:

$$V_{out} = \frac{1}{B}(V_1 - V_2) \tag{6.6}$$

where $B = C^1_{of}/C^1_{if} = C^2_{of}/C^2_{if}$, and we have assumed that AB is much larger than $(1 + B)$. The derivation in Appendix B follows the style of [Mead 91].

The response of the doubly compensated amplifier has the advantage of similar scale of the inputs, V_1 and V_2, as well as the capability to correct for input offsets. By appropriate scaling of the capacitors, an almost arbitrary scaling of the inputs can be provided.

6.3 Differential Multipliers

Using what we have learned from constructing offset compensated amplifiers, we can improve the performance of a class of multipliers. We will consider multipliers that have the following general input-output relation:

$$P = k(x - x_0)(y - y_0) \tag{6.7}$$

where P, x, x_0, y, and y_0 may be voltages or currents, and k is some constant. We will be concerned with techniques for improving and controlling the performance of such multipliers, in terms of offsets (errors in controlling x_0 and y_0), and nonlinearities (variations in k over the operating range). Figure 6.4 shows an example of such a generic multiplier, in this case, having a voltage output.

6.3.1 Two-transistor Multiplier

We can actually perform quite a nice multiply operation with two transistors [Denyer 81][Denyer 83][Tsividis 86]. Figure 6.5 shows the configuration of the two transistors.

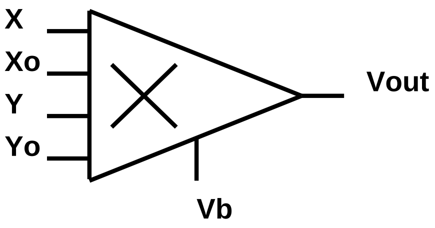

Figure 6.4: A schematic diagram of a differential multiplier. □

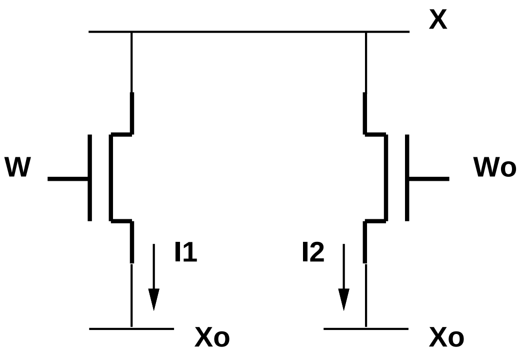

Figure 6.5: A two-transistor multiplier. □

In the ohmic region of operation, we can approximate the drain current in a transistor as:

$$I_{ds} \approx (V_{gs} - V_T)V_{ds} + \frac{1}{2}V_{ds}^2 \quad . \tag{6.8}$$

Therefore, the currents in the transistors in Fig. 6.5 are approximated as follows:

$$I1 = (w - x_0 - V_T)(x - x_0) + \frac{1}{2}(x - x_0)^2 \tag{6.9}$$

$$I2 = (w_0 - x_0 - V_T)(x - x_0) + \frac{1}{2}(x - x_0)^2 \quad . \tag{6.10}$$

If we subtract $I2$ from $I1$, we get, to a first-order approximation:

$$I_{diff} = (w - w_0)(x - x_0) \quad . \tag{6.11}$$

6.3.2 Four-transistor Multiplier

We can cancel out more higher order nonlinearities by using a 4 transistor multiplier design, as shown in Fig. 6.6 [Tsividis 86]. The currents in the transistors in Fig. 6.6 are as follows:

$$\begin{aligned} I1 &= (w - x_0 - V_T)(x - x_0) + \tfrac{1}{2}(x - x_0)^2 \\ &\quad +(w_0 - x_0 - V_T)((2x_0 - x) - x_0) + \tfrac{1}{2}((2x_0 - x) - x_0)^2 \end{aligned} \tag{6.12}$$

$$\begin{aligned} I2 &= (w_0 - x_0 - V_T)(x - x_0) + \tfrac{1}{2}((x - x_0)^2 \\ &\quad +(w - x_0 - V_T)((2x_0 - x) - x_0) + \tfrac{1}{2}(2x_0 - x) - x_0)^2 \quad . \end{aligned} \tag{6.13}$$

Therefore, the difference current is:

$$I_{diff} = 2(w - w_0)(x - x_0) \quad . \tag{6.14}$$

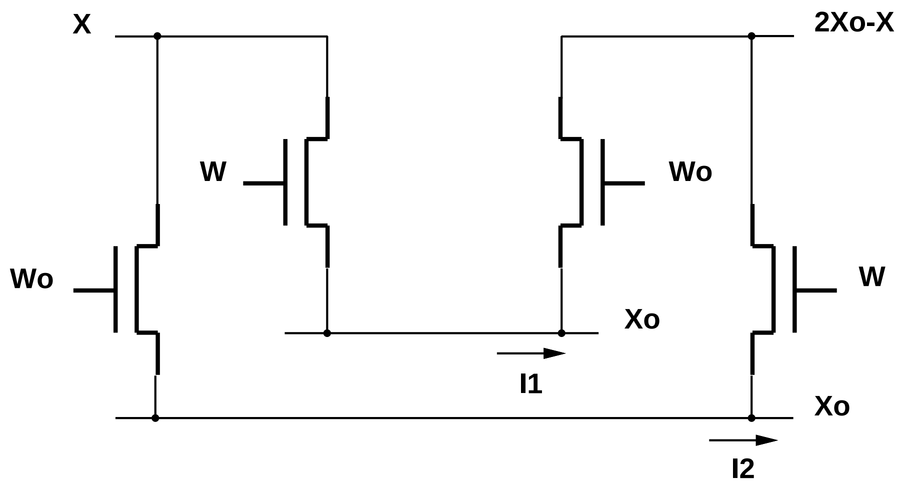

Figure 6.6: A schematic diagram of a four-transistor multiplier. □

We can use a larger proliferation of transistors to cancel out higher order deviations from linearity, as well. For the purposes of describing our compensation technique, we need only to apply the technique to the two transistor version.

6.3.3 Gilbert Multiplier

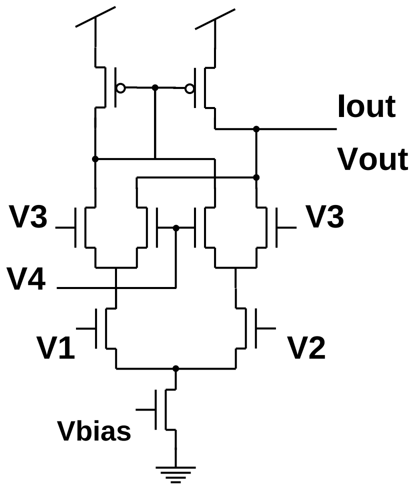

Figure 6.7: A schematic diagram of a Gilbert transconductance multiplier. □

Figure 6.7 shows a schematic diagram of a Gilbert transconductance multiplier.

The Gilbert multiplier is quite a powerful design in ECL, but has some problems in its CMOS realization. Some of these problems are described in [Babanezhad 85], along with some additional circuitry to address the problems that are identified.

Our primary concern is the cancellation of input offsets, and we assume that similar techniques can be applied to most robust and nearly linear multiplier designs. In the following section, we describe the design, operation and chip-measured results of a compensation technique as applied to a two-transistor multiplier. We will not present the same level of detail for additional compensated multiplier designs.

6.4 Constructing a Compensated Multiplier

We will now describe how to construct a multiplier that has a "knob" for the correction of input offsets, starting with one of the multipliers described above.

In the case of the two transistor multiplier, one of the parameters, the source-drain voltage, is already well controlled. However, the offsets can be corrected by relating the values of x_0 to the generation of values for x, so that when we wish a zero, $x = X_0$. The larger offsets are primarily due to variations in the gate offsets between the two transistors. These offsets can be controlled by adjusting the value of W_0.

Figure 6.8 shows a schematic diagram of a compensated version of a two-transistor multiplier. By adjusting the charge on the floating gate V_f, we can compensate for the offset voltage caused by differences between the two transistors.

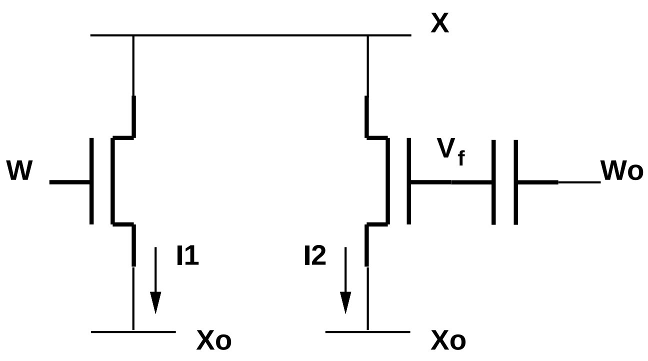

Figure 6.8: A two-transistor multiplier with floating gate compensation capability. By adjusting the quantity of charge on the floating gate, the voltage V_f can be adjusted, relative to the reference voltage W_0. This adjustment can be used to compensate for any offsets caused by differences between the two transistors. □

The 4 transistor multiplier is sufficiently similar in concept to the 2 transistor version that we do not describe its compensation here. A compensated 4 transistor multiplier would have better linearity. The compensation technique can be applied equally well to other better (more nearly linear) multipliers, including the Gilbert transconductance multiplier described above. Appendix B describes the derivation of the expected response of a compensated Gilbert multiplier.

Before we present results measured from our multiplier chips, it is useful to review the number to signal mapping function G and the signal to number mapping function G^{-1}, as described in Ch. 3. For each of the voltage inputs to our multiplier, we require a mapping from the desired numeric input (in the mathematical domain) to the actual voltage signal to be applied to the multiplier.

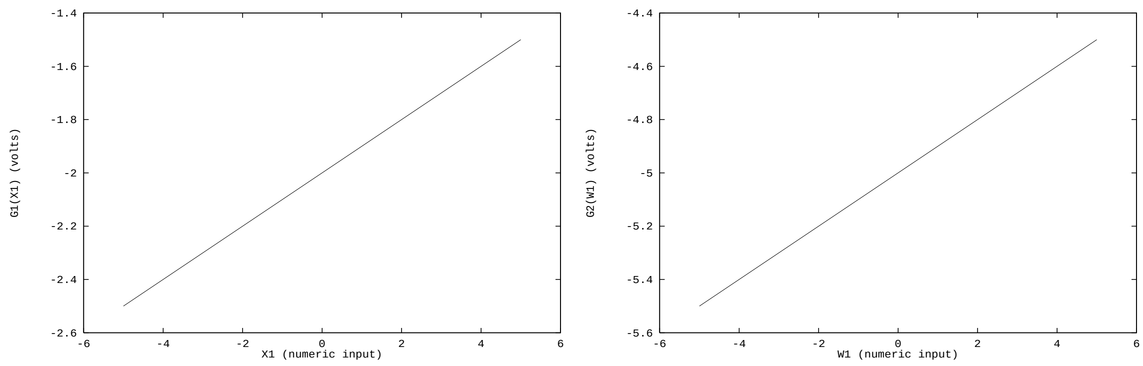

Figure 6.9: Two examples of the input number to signal mapping function, G_1 and G_2. We use the linear input mapping to attach a numeric meaning to the electrical signals used to drive the multiplier. We could also apply a more complex nonlinear input mapping function to compensate for offsets or nonlinearities in the multiplier operation. □

Figure 6.9 describes the two number to signal mappings $G_1(X)$ and $G_2(W)$ that we used for the multiplier. We also require a mapping from the physical output voltage produced by the multiplier to a numeric output (in the mathematical domain), so that we can interpret our results. Figure 6.10 describes the output signal to number mapping that we used for the multiplier.

The use of the two input mappings, $G_1()$ and $G_2()$, and the output mapping, $G_3()$, allows us to think of the multiplier as multiplying *numbers* instead of simply electrical signals. The functions $G_1()$ and $G_2()$ map the numeric inputs to electrical signals, which are multiplied together using the analog multiplier. The function $G_3()$ maps the output signal back to a numeric representation. So, the combination of $G_1()$, $G_2()$, the analog multiplier, and $G_3()$ allows us to multiply the numeric range of $[-5, 5]$ by the numeric range $[-5, 5]$, producing a number in the range $[-25, 25]$.

Fig. 6.11 shows the results of the raw two transistor multiplier before compensation. The graph plots one of the voltage inputs, $G_1(X)$, of the multiplier on the horizontal axis, vs. the difference current output on the vertical axis. In our representation for $G_1(X)$, -2.5 volts represents a large negative number, -2 volts represents mathematical zero, and -1.5 volts represents a large positive number. We use our number to signal mapping function G to map a numeric range to the 1 volt input range of the multiplier. The three curves are produced from three values for the other voltage

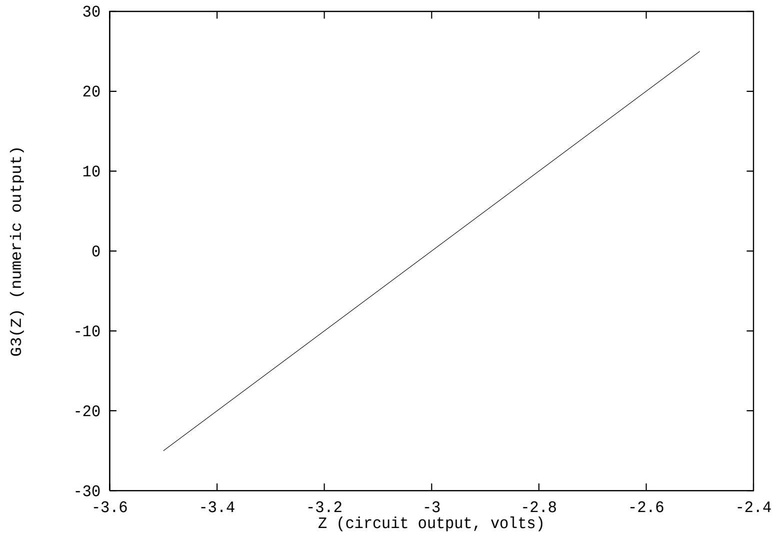

Figure 6.10: An example of the output signal to number mapping function, G_3. We use the linear output mapping to attach a numeric meaning to the electrical signals produced at the output of the multiplier. As with the input mappings G_1 and G_2, we could also apply a more complex nonlinear output mapping function to compensate for offsets or nonlinearities in the multiplier operation. □

input to the multiplier, $G_2(W)$, which ranges from -5.5 volts to -4.5 volts, where -5 volts represents mathematical zero. Note the nonzero offsets, as evidenced by the nonzero slope line formed by the square symbols. That line represents the results of multiplying zero by a set of other quantities, so it should be horizontal, at zero.

Fig. 6.12 shows the output from a compensated two-transistor multiply circuit. As above, the graph plots one of the voltage inputs, $G_1(X)$, of the multiplier on the X axis, vs. the difference current output on the Y axis. Also as before, in our representation for $G_1(X)$, -2.5 volts represents a large negative number, -2 volts represents mathematical zero, and -1.5 volts represents a large positive number. The three curves are produced from three values for the other voltage input to the multiplier, $G_2(W)$. Note that the “zero” line (again delineated by the square symbols), is much closer to horizontal at zero, due to the effects of the compensation.

Fig. 6.13 shows the compensated voltage-in, voltage-out multiplier performance. The signal

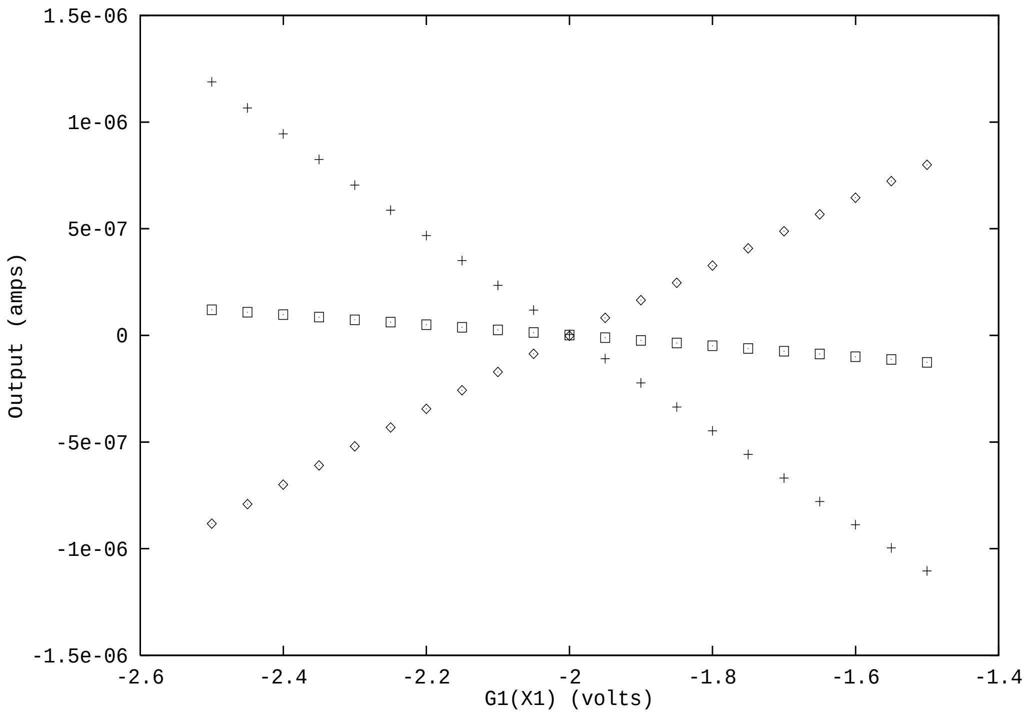

Figure 6.11: The results of an *uncompensated* multiply operation (actual measured chip data). The analog multiplier circuit has not been adapted to compensate for input offsets and other device variations. The graph plots one of the voltage inputs, $G_1(X)$, of the multiplier on the horizontal axis, vs. the difference current output on the vertical axis. In our representation for $G_1(X)$, -2.5 volts represents a large negative number, -2 volts represents mathematical zero, and -1.5 volts represents a large positive number. The three curves are produced from three values for the other voltage input to the multiplier, $G_2(W)$. Note the nonzero offsets, as evidenced by the nonzero slope line formed by the square symbols. That line represents the results of multiplying zero by a set of other quantities, so should be horizontal, at zero. □

presented in this figure is an intermediate value in the hierarchical constraint computation. Its nonlinearity and nonzero offset characteristics reflect the fact that this output contains biases to compensate for variations in the next stage of computation. These curves represent the sum of the multiplier output and the compensation input for a subsequent computational element, as chosen by an external optimization process.

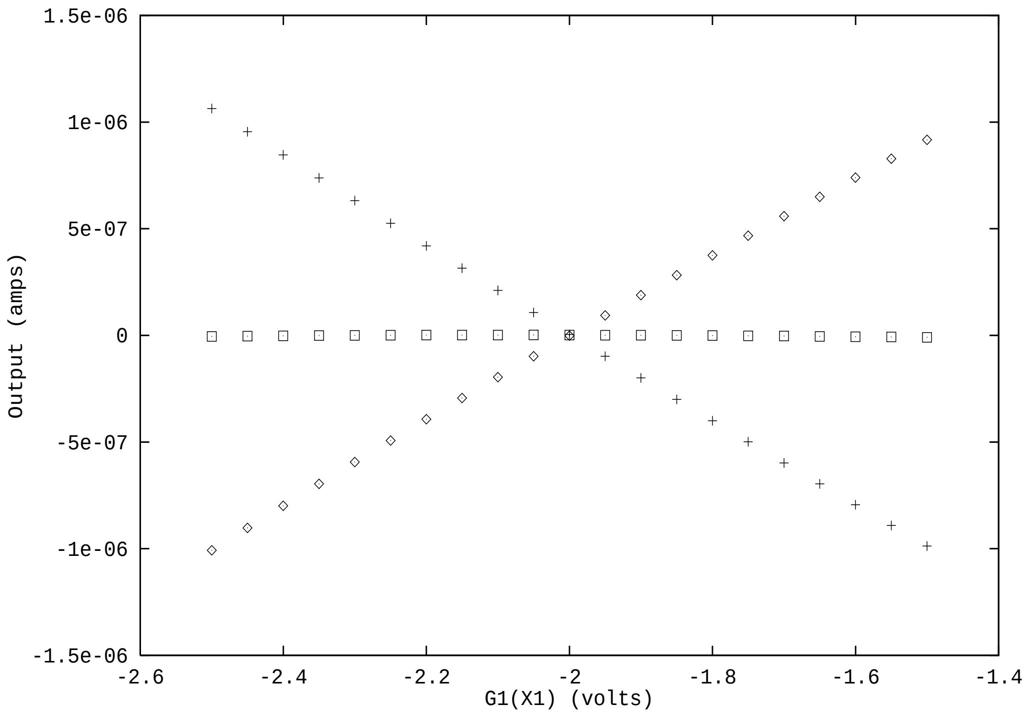

Figure 6.12: The output of a compensated compensated two transistor multiply circuit (actual measured chip data). The graph plots one of the voltage inputs, $G_1(X)$, of the multiplier on the horizontal axis, vs. the difference current output on the vertical axis. In our representation for $G_1(X)$, -2.5 volts represents a large negative number, -2 volts represents mathematical zero, and -1.5 volts represents a large positive number. The three curves are produced from three values for the other voltage input to the multiplier, $G_2(W)$. The input offset error is less than 1 mV (0.1% of the input range), and the maximum deviation from linearity (relative error) is 2.2%. Note that the "compensation" process employed here corrects only for the input offset error, but does not affect the nonlinearity. If greater linearity is desired, then an alternative multiplier design (or compensation schedule) should be chosen. □

6.4.1 Constructing a Well-Behaved Inner-Product (dot product) Circuit

We can connect together three of the compensated multipliers to produce an inner (or dot) product. Since the multipliers each require two current sense amplifiers (one for I+ and one for I-), we might implement a dot product using 6 amplifiers. As suggested by [Denyer 81], we can share one sense amplifier between all of the I+ amplifiers, and a second one between all of the I- amplifiers, so we only need two.

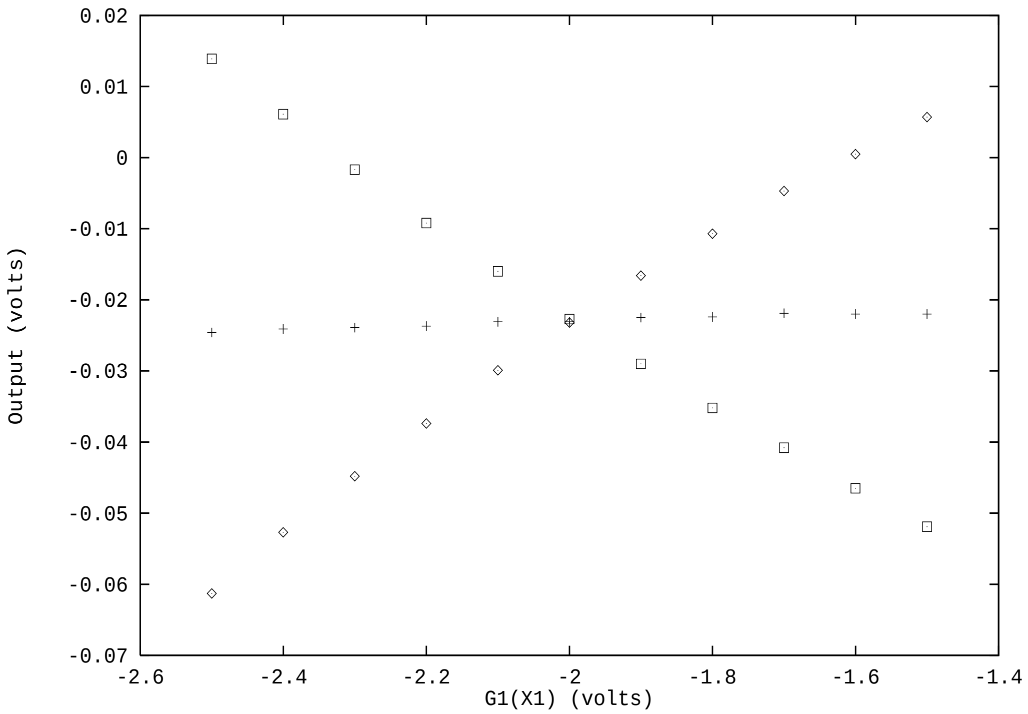

Figure 6.13: The output of a multiplier, after nearly linear current-to-voltage conversion (actual measured chip data). The output voltage is measured relative to a reference voltage for the current sense amplifier. □

Figure 6.14 describes the use of three compensated two-transistor multipliers to construct a well-behaved dot product. We can also use a compensated amplifier for the subtraction of I+ - I-, thus providing another knob for the hierarchical compensation.

Fig. 6.15 shows data measured from a chip that implements the three multiply components of a compensated dot product. Note that the offset correction is very accurate, but that the linearity is somewhat less accurate.

Although in this case the hierarchical compensation process is straightforward, we will describe the sequence of actions in some detail to clarify the process for more complex circuits. We proceed from lower-level (more elementary) computations, to higher level computations, involving ever greater numbers of circuit components.

First, we compensate the individual multipliers within the dot product circuit. Each multiplier

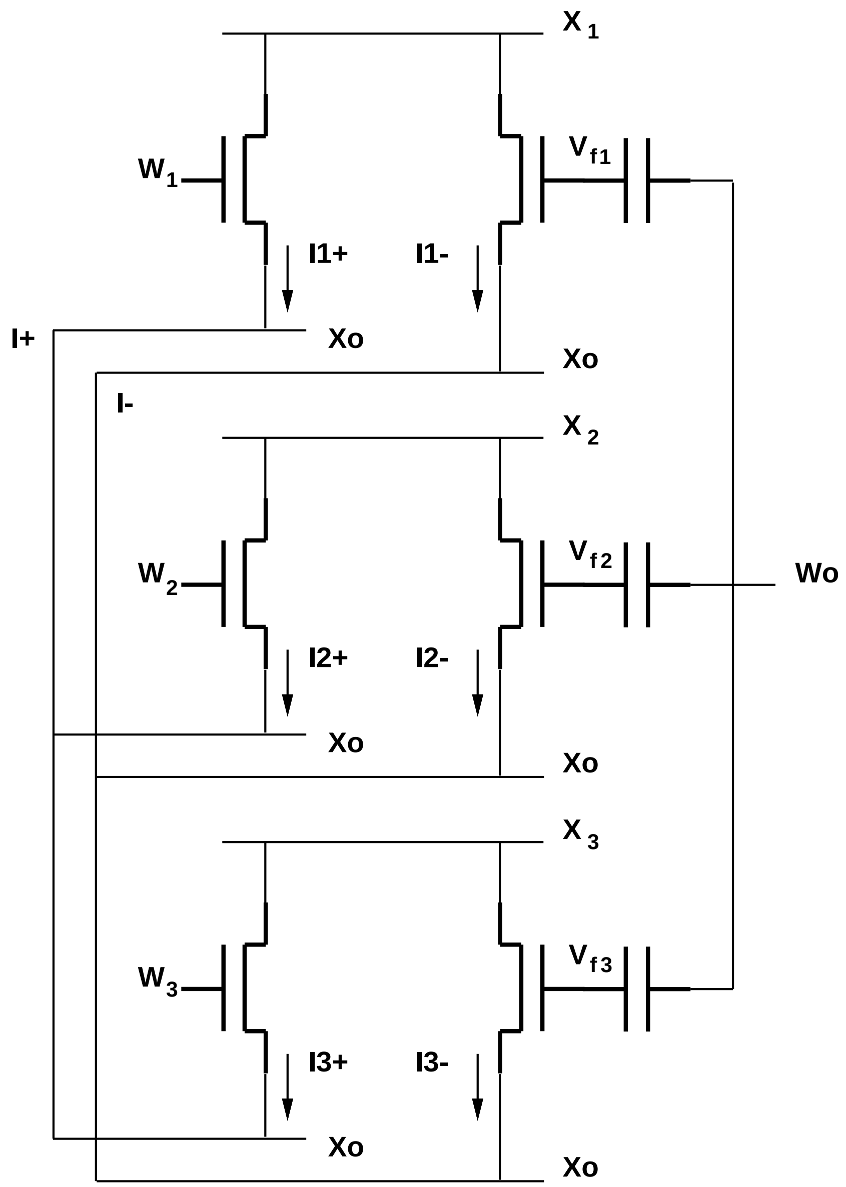

Figure 6.14: A set of three two-transistor multipliers with floating gate compensation capability, connected to form a dot product calculation. By adjusting the quantity of charge on the three floating gates, the voltages V_{f1}, V_{f2}, and V_{f3} can be adjusted, relative to the voltage $W0$. These adjustments can be used to compensate for any offsets caused by differences between the two transistors in each multiplier. □

should produce a "zero" output for all cases where a numeric zero result is desired. Using our number

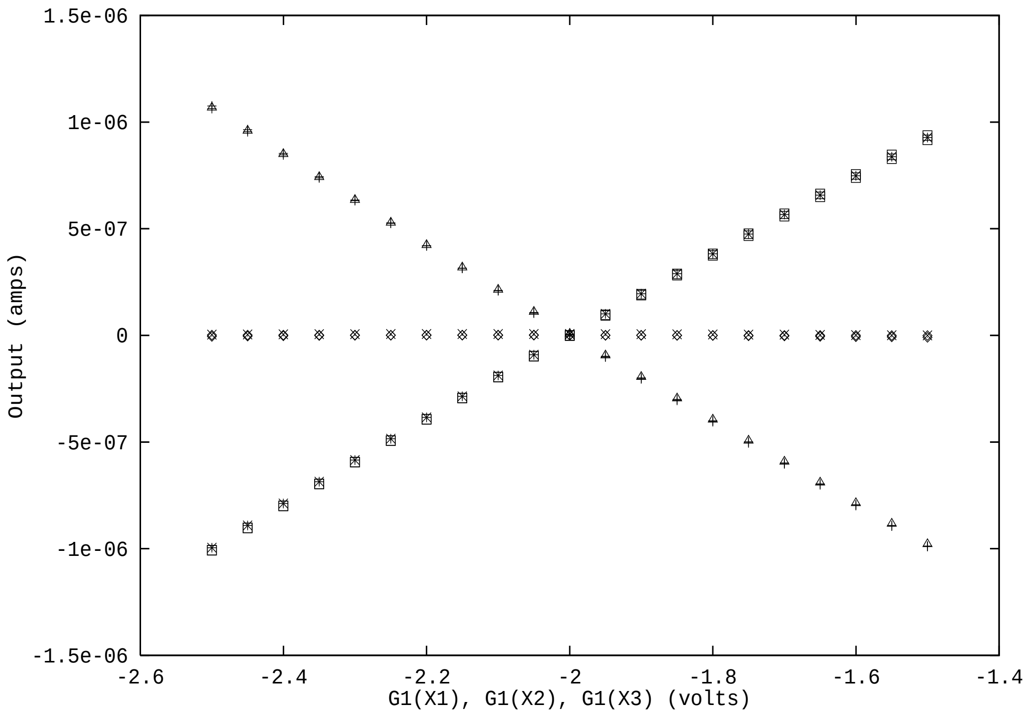

Figure 6.15: The 3 components of a 3D dot product (actual measured chip data). The characteristics of the three multiply operations are similar, with respect to the input offset magnitudes and shape of nonlinearities. The graph plots the voltage inputs (X_1, X_2, and X_3) of each multiplier on the X axis, vs. the difference current outputs on the Y axis. In our representation for X_1, X_2, and X_3, -2.5 volts represents a large negative number, -2 volts represents mathematical zero, and -1.5 volts represents a large positive number. The three curve groups represent three values for the other voltage inputs to the multipliers, W_1, W_2, and W_3. □

to signal input mapping functions $G_1()$ and $G_2()$, and our output signal to number mapping function $G_3()$, we want to enforce the following constraints as closely as possible:

(6.15)
$$5 * 0 = 0$$

(6.16)
$$0 * 5 = 0$$

$$-5 * 0 = 0 \tag{6.17}$$

$$0 * -5 = 0 \tag{6.18}$$

$$0 * 0 = 0 \quad . \tag{6.19}$$

We use a sum of squares error metric to weight the constraints equally, and penalize large errors more than small ones. Using gradient descent, we can adjust the single parameter for each multiplier to find the value that most closely solves these constraint equations. We choose these particular constraints because we believe that errors about zero are more critical than errors at the extremes of the operating range. These choices are appropriate for the application discussed in Ch. 7. If we are more concerned about other regions of operation, we can choose constraint appropriately.

After each of the multipliers has been compensated, we consider the performance of the dot product as a whole. We are concerned now with how the multipliers' individual errors interact as part of the dot product calculation. Consequently, we choose as our constraints unit vectors which are either perpendicular or parallel, and thus should produe either zero or one, respectively, at the output. Examples of such constraints are:

$$0 * 0 + 0 * 0 + 1 * 1 = 1 \tag{6.20}$$

$$\sqrt{2} * \sqrt{2} + \sqrt{2} * \sqrt{2} + 0 * 0 = 1 \tag{6.21}$$

$$\sqrt{3} * \sqrt{3} + \sqrt{3} * \sqrt{3} + \sqrt{3} * \sqrt{3} = 1 \tag{6.22}$$

$$1 * 0 + 0 * 1 + 0 * 0 = 0 \tag{6.23}$$

$$\sqrt{2} * \sqrt{2} + \sqrt{2} * (-\sqrt{2}) + 0 * 0 = 0 \tag{6.24}$$

and so on. Again, we use a sum of squares error metric, and a gradient descent process to choose the optimal parameters. There are also three parameters to be adjusted, one for each of the three multipliers. Also, since the multipliers have each already been compensated to perform reasonably well (by some metric), we make only small adjustments at this stage. After this gradient descent process is complete, we have produced a compensated dot product circuit, which performs much more accurately than the uncompensated version.

In Ch. 7, we will use a set of hierarchically compensated dot product circuits (as described above), to calculate an error metric and gradients, to enforce a constraint.

6.4.2 Constructing a Well-Behaved Matrix Multiply

Although we have not built a two-dimensional matrix multiply from the compensated multiplier and dot-product, we have built all of the necessary components. The extension of the ideas of the dot product to a matrix multiply is straightforward and so is not described explicitly as a separate project. We proceed instead to a more challenging application of the compensated multiplier, applying constraints to produce a unit orthogonal rotation matrix. This task includes the pieces of the matrix multiply as parts of its operation.

6.5 Summary

Building circuits with “knobs” makes them more useful as building blocks for larger applications. We have described several examples of compensated circuit building blocks, providing increased accuracy and consistency over the uncompensated versions. We have also described how to combine these circuit building blocks using hierarchical compensation techniques to make more interesting and powerful computational primitives, such as dot products and two-dimensional matrix multipliers.

The next logical step is to use the compensated circuit building blocks to construct more complex applications. In Ch. 7, we will describe how to use these techniques to cosntruct a small constraint solution system for computer graphics.

Chapter 7

Implementing Rotation Constraints in Analog VLSI

Now that we have built compensated multiplier circuits and have used them to construct well-behaved multiply and dot-product functions, we can use these robust building blocks to construct a more interesting application.

We describe an algorithm for producing a 3x3 rotation matrix from 9 input values that form an approximate rotation matrix, and we describe the implementation of that constraint in analog VLSI. This constraint is useful when some source [e.g., sensors, a modeling system, other analog VLSI circuits], produces a potentially "imperfect" matrix, which is to be used as a rotation. The 9 values are continuously adjusted over time to find the "nearest" true rotation matrix, based on a least-squares metric. With appropriate design methodology, adaptive analog VLSI is a fast, accurate, and low-power computational medium. The implementation is potentially interesting to the graphics community because there is an opportunity to apply adaptive analog VLSI to many other graphics problems.

7.1 Analog VLSI for Constraint Satisfaction

This chapter has two main purposes. First, we demonstrate the implementation of a nontrivial constraint technique in analog VLSI. Second, we describe the technology of adaptive analog VLSI to the computer graphics community. We believe that analog VLSI has great potential as a computation substrate for implementing rendering and modeling primitive operations. We hope that the realization of this potential will open up exciting new areas of research in computer graphics.

There has been increasing interest recently in using analog VLSI [Mead 89], for a variety of computational tasks. Mead and others have pursued the paradigm of using individual analog transistors to model components of neural systems. Related research has focused on increasing the accuracy and precision of computation with analog VLSI and on developing a design methodology for creating analog VLSI circuits which can be adjusted to perform to the desired accuracy [Kirk 91]. These techniques make analog VLSI more tractable for quantitative computation.

This is not the first appearance of analog computation in computer graphics. There is a history of analog implementation in computer graphics. Certainly, there is some amount of analog hardware in almost every graphics system, at least in the form of a D/A (digital-to-analog) converter in the path to the video monitor. Even in liquid crystal displays on laptop computers, there is probably some analog circuitry in the display path. There have also been more extensive uses of analog, however. For instance, Adage implemented matrix multiplication for the purpose of performing coordinate transformations in analog circuitry, although not in VLSI.

There is also a history of digital VLSI acceleration in computer graphics: geometry engines [Clark 82], hardware frame buffer assists [Rhoden 89], vector generators [Barkans 90], systems [Voorhies 88]

[Fuchs 89], etc. Most high-performance graphics workstations have a substantial amount of special purpose chips to provide the kind of interactive performance that we have come to expect. Most, if not all of this silicon is dedicated to rendering tasks. Even the geometry engine is used primarily in the display pipeline.

It is important to note that in these discussions, we have chosen a *particular* constraint to demonstrate the general technique of implementing a constraint in analog VLSI. There are many other examples of useful constraint computation that could be formulated in a similar fashion [Platt 89] [Barzel 92], and also could be implemented in analog hardware. The particular constraint that we have chosen to implement is meant to be representative of a large set of possibilities. Our example raises the exciting prospect of implementation of extensive hardware *modeling* assists in analog VLSI. There are also many rendering tasks which are appropriate for analog VLSI hardware implementation, but we won't discuss them in this thesis.

7.1.1 The Rotation Matrix Constraint

The constraint technique that we have chosen is the ortho-normalization of a rotation matrix. We have chosen the 3x3 matrix formulation because it is easier to perform coordinate transformations with the same underlying computational modules that are used to implement the constraints. In Sec. 7.3, we describe several computational blocks that we can also use to construct coordinate transformation hardware. The matrix formulation is also complex enough to be interesting as an example problem for hardware.

The rotation matrix constraint is particularly useful as part of a real-time computer graphics system (see Fig. 7.6). For virtual reality applications, a sensor may be used to produce a 3D orientation, in the form of a 3x3 rotation matrix. Sensors are often flawed, noisy, or otherwise inaccurate and do not provide sufficient and reliable information for producing an accurate rotation matrix. In such cases, we then wish to continuously produce a "best estimate" rotation matrix, based on the sensor measurements.

A similar task exists in robotics applications. We might have a sensor which can detect the position of an end effector of a robot arm, and also a measure of the control inputs. In practice, a robot arm is often controlled by providing joint angle control inputs. However, the control may be inaccurate, and there may be "slop" in the joints. We may want to then compute an estimate of the actual joint angles, which, if the arm segments are rigid, must be pure rotations.

There are also many applications to physically-based modeling. When solving constraint equations for motion of rigid bodies, we may produce values that are inaccurate due to accumulating arithmetic roundoff errors, integration step size, or approximations in our model. When combined to form a rotation matrix to describe the orientation of a body, the errors may cause the introduction of scaling or skewing into the matrix. The constraint technique described in this section allows us to automatically adjust for these errors.

In Sec. 7.2, we describe the constraint algorithm that we use to produce the rotation matrix. In Sec. 1.1, we introduced in more detail the technology used for the implementation (analog VLSI), and explained why we believe that it has great potential to be useful for computer graphics. In Sec. 7.3, we present a block diagram description of the constraint chip.

7.2 The Constraint Algorithm

Our goal is to produce a 3x3 rotation matrix containing no scale or skew components, given 9 numbers which are already nearly a rotation matrix.

For a mathematically perfect rotation matrix M,

$$MM^T = I \tag{7.1}$$

where I is the identity matrix.

We define the function $f()$:

$$f(M) = (MM^T - I) : (MM^T - I) \tag{7.2}$$

where the double-dot operator (:) denotes the sum of products of terms of the two matrices, producing a scalar result, analogous to the dot product of two vectors. We can rewrite Eq. 7.2 as:

$$f(\underline{\underline{M}}) = \sum_{j=1}^{3}\sum_{k=1}^{3}((\sum_{i=1}^{3} M_{ij}M_{ik}) - \delta_{jk})((\sum_{\ell=1}^{3} M_{\ell j}M_{\ell k}) - \delta_{jk}) \quad . \tag{7.3}$$

where δ_{ij} indicates the identity matrix ($\delta_{ij} = 1$ when $i = j$ and 0 otherwise).

When M is a rotation matrix (or reflection), $f(M)$ in Eq. 7.2 is equal to zero, and when M is not purely a rotation matrix, $f(M) \neq 0$. Since $f(M)$ is always greater than or equal to zero, M is a rotation matrix when $f(M)$ is minimized.

We perform gradient descent to minimize the function $f()$, as follows:

$$M'(t) = -\epsilon \; \nabla f(M(t)) \tag{7.4}$$

where epsilon is a parameter which determines the speed of the descent. Appendix 1 describes the derivation of our gradient calculation method in detail.

The analog VLSI implementation does not suffer from many of the problems of a digital implementation, since analog circuits can operate in continuous time. For instance, in a digital implementation, Euler's method could be used to solve Eq. 7.4. With large step sizes, Euler's method frequently becomes unstable. With small step sizes, Euler's method may converge slowly or not at all. There are other techniques, such as the conjugate gradient method, for improving the performance in digital implementations. The continuous nature of an analog implementation, however, avoids this type of problem entirely.

As the computation proceeds, two kinds of changes are occurring. First, the imperfect input matrix may be changing over time. Second, based on our optimization process, the output matrix will be changing to fulfill our rotation matrix constraint. Since the analog VLSI circuit operates very quickly, and in continuous time, the optimization can occur at a much finer time scale than the changing of the input matrix.

It is possible to make analog circuits more quantitatively useful, by designing *compensatable* circuit building blocks that can be adjusted to perform more closely to some performance metric. For example, let us assume that our goal is to build a "perfect" analog multiplier. In analog VLSI, we can easily build a circuit which computes an "imperfect" multiply-like operation, but the "perfect" multiply is more elusive. We can design a multiplier that is monotonic within some input range, and operates in four quadrants (the sign of the output is correct for all combinations of inputs' sign). Without extreme care in the design, however, the "multiplier" would have a number of drawbacks. The circuit's response might deviate significantly from the desired linear function of its inputs

$$f(x, y) = x * y \quad . \tag{7.5}$$

The "multiplier" would also, very likely, have nonzero input offsets.[1]

A compensated multiplier might have adjustable parameters which would allow for the improvement of the linear range of behavior, as well as the cancellation of input offsets. A description of how to design, build, and optimize compensatable components is presented in detail in this thesis, in Sec. 6.4. Also, in Sec. 6.4, we present some measurements from chips implemented and compensated using these techniques.

7.3 Applying Analog VLSI to the Constraint Problem

Now that we have described the desired constraints (in Sec. 7.2) and the substrate technology of adaptive analog VLSI (Sec. 1.1), we explain, at a block diagram level, how we use analog VLSI to

[1] Input offsets are present for an analog multiplier $f(x,y) \approx x * y$ when $f(x,0) \neq 0$ or $f(0,y) \neq 0$.

solve the constraint problem. These block diagrams represent a hierarchical decomposition of the chip that we built.

As one might guess from the form of the equations of the derivation in Sec. 7.2, the circuit architecture is a nested, structured hierarchy of dot products, with some additional computation.

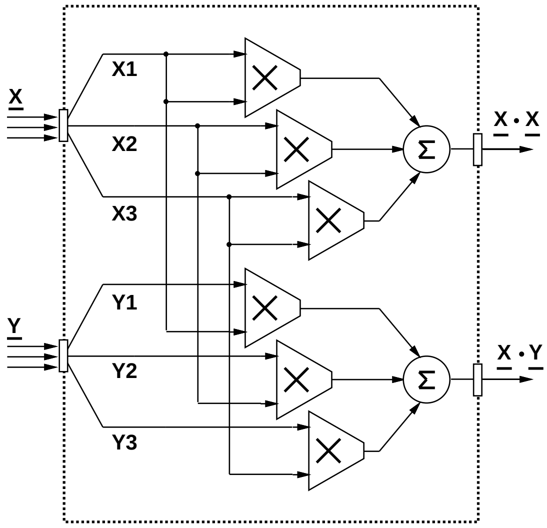

Figure 7.1: A functional block containing two dot products. The inputs are the matrix column vectors $\underline{X}$ and $\underline{Y}$, and the outputs are the scalars $X \cdot X$ and $X \cdot Y$. □

We can think of the nine input values, the "imperfect" rotation matrix, as the three 3D basis vectors, $\underline{X}$, $\underline{Y}$, and $\underline{Z}$, (the three columns of the matrix). We can see by examining Eq. D.7 in Appendix D that the computation of the various components of the gradient, η_{pq}, requires dot products of the matrix basis vectors. Appendix 1 describes the calculation of η_{pq}. Fig. 7.1 shows a functional block which computes two of the six basis vector dot products that are required.

Fig. 7.2 shows a set of three functional blocks (from Fig. 7.1) which together compute the six 3D basis vector dot products that are required to form the gradient, η_{pq}, as shown in Eq. D.7 in Appendix D. The details of the circuit, the device layout, and the compensation procedure for the multiplier and dot product blocks are presented in this thesis.

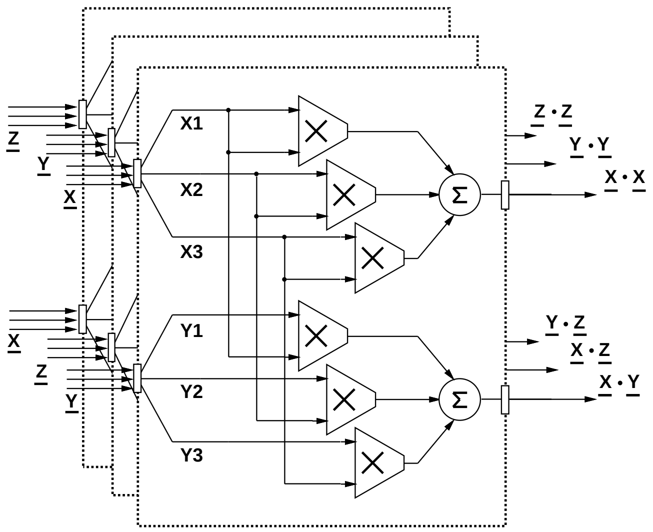

Figure 7.2: A collection of three dot product blocks. With the 3D basis vector inputs $\underline{X}$, $\underline{Y}$, and $\underline{Z}$, they compute the six dot products required to enforce the constraints. □

Fig. 7.3 shows the use of the basis vector inputs and three of the dot product results to produce the gradient components for one of the basis vectors, in this case, $\underline{X}$.

Fig. 7.4 shows a set of three constraint blocks, from Fig. 7.3, which together compute all of the components of the gradient for the correction of the imperfect matrix. The combination of these three constraint blocks and the three dot product blocks from Fig. 7.2 forms the gradient calculation hardware. X_1', X_2', X_3', Y_1', Y_2', Y_3', Z_1', Z_2', and Z_3' are the nine derivative components. Together, they form the gradient, which we use to optimize the components of the matrix $\underline{\underline{M}}$. Descending along the direction of the gradient produces a matrix which fulfills our constraints.

We use the derivative terms from Fig. 7.4 to add or subtract from the original input values of the matrix, $\underline{\underline{M}}$. Since the circuits are analog and operate in continuous time, we can integrate these corrections on capacitors, and use the gradient components to set the level of current to add/subtract. Thus, this circuit structure can be used to continuously track and correct a (potentially flawed) matrix that changes over time. Fig. 7.5 shows the connections required to provide the feedback from the calculated gradient components to modify the input matrix components. The gradient

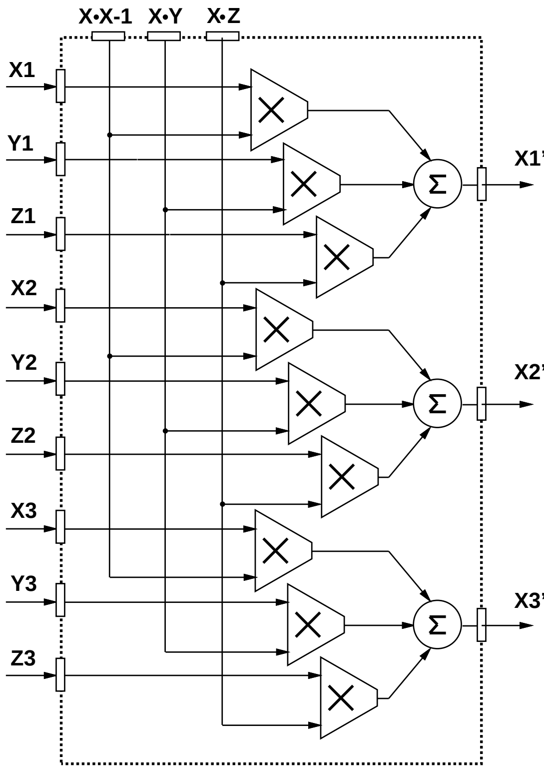

Figure 7.3: A basis vector constraint block, using the outputs from the dot product constraint block. This computational element implements the rotation matrix constraint for one of the three matrix column vectors. □

calculation occurs in continuous time, using the analog VLSI hardware. The input can change continuously, or discretely (using the "reset" input in Fig. 7.5), and the constraint solution tracks the input. There is an implicit "zero" involved in the integration process and the reset operation. This choice of a "zero" signal to represent a mathematical zero gradient relates to our number to signal mapping function G, as described in Sec. 3.5. In our implementation, the error feedback summation is achieved by adding or subtracting charge from a capacitor. The "zero" level for gradient feedback is chosen based on the reference level for the integrator, so that at some quiescent reset level, no current will be added or removed from the summation capacitor.

Fig. 7.6 shows a schematic view of the the rotation matrix constraint solution box connected as part of a system. Given a source of approximate rotation matrices $M^{in}(t)$, the constraint enforce-

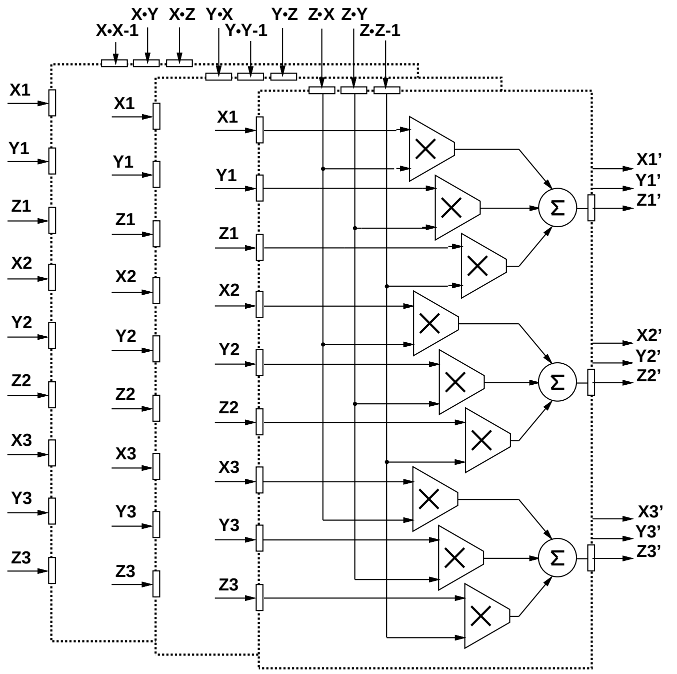

Figure 7.4: A collection of three constraint blocks. The combination of these three constraint blocks and the three dot product blocks forms the gradient calculation hardware. □

ment produces rotation matrices $M^{out}(t)$, which can be used for modeling, rendering, or control applications.

7.4 Results

We have designed, implemented, fabricated, and tested chips which contain compensated multipliers, dot products, and constraint blocks, as described in Fig. 7.1 through Fig. 7.4. The design is modular (similar to the structure of the figures), so that we are confident that the overall system will work, given the test results. We have tested all of the components, and present the data (results measured from chips implementing the compensated multiplier and dot product with hierarchical compensation were presented in Sec. 6.4). We also present a software simulation of the constraint process in action, using the constraint technique described herein.

Fig. 7.7 shows the results of a simulation of our constraint technique in action. The outer curved

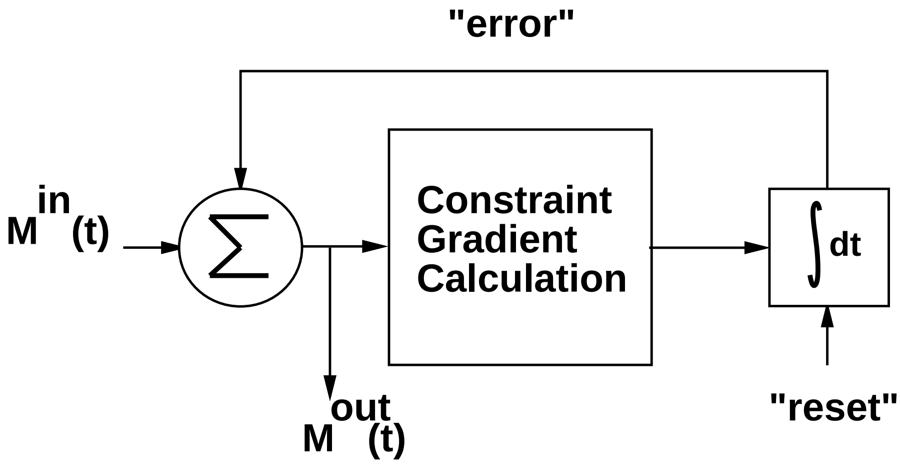

Figure 7.5: An example of a gradient descent process, as employed to enforce the rotation matrix constraint. The feedback from the gradient calculation modifies the effective inputs to the constraint gradient calculation box. As the constraint is satisfied, the output, $M^{out}(t)$ settles to a rotation matrix, if $M^{in}(t)$ is not changing, or is changing at a slower time scale. Note the "reset" input to the integrator box. If the input matrix $M^{in}(t)$ changes discontinuously, we want to restart the constraint optimization from the new matrix, and we can accomplish this using the integrator reset. □

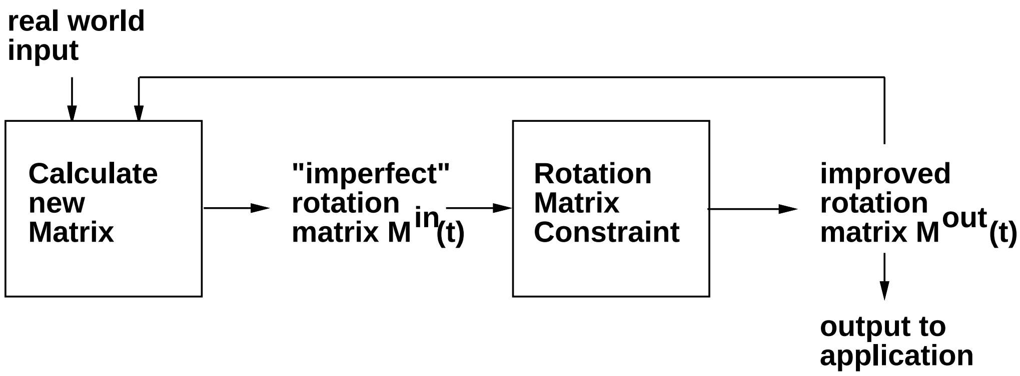

Figure 7.6: A system-level view of the rotation matrix constraint enforcement, and how the result of applying the constraint might be used. □

octant represents the manifold of a set of points transformed using the input imperfect matrix. The inner, more spherical shape represents the same points (and more) transformed through the constrained rotation matrix. The lines drawn between the two shapes represent the constraint optimization path taken by our algorithm, as described in Eq. 7.4.

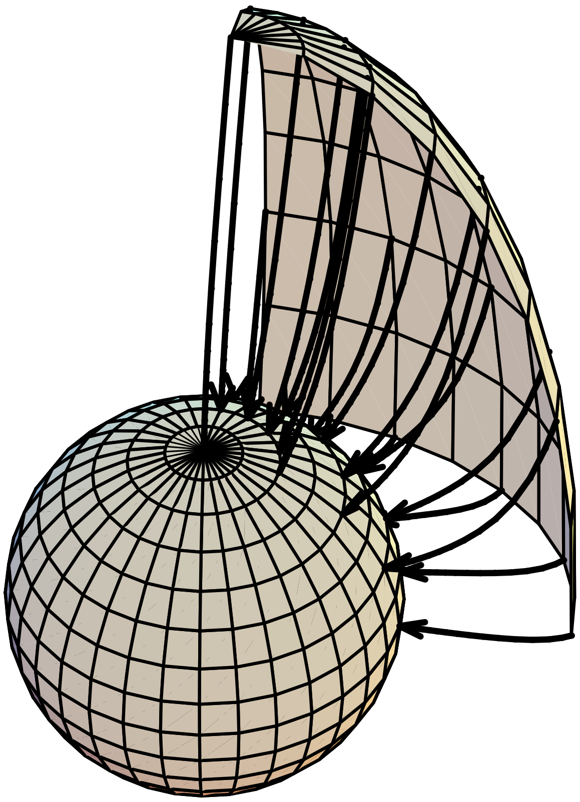

Figure 7.7: The results of a software simulation of our constraint technique in action. The outer curved octant represents the manifold of a set of points transformed using the input imperfect matrix. The inner, more spherical shape represents the same points (and more) transformed through the constrained rotation matrix. The lines drawn between the two shapes represent the constraint optimization path taken by our algorithm, as described in Eq. 7.4. □

We do not have measured data for the complete constraint chip in operation at this time. We will discuss the expected performance, however, and compare it to a digital computer implementation. We assume that a typical "hot-box" workstation processor or digital signal processor (DSP) chip can run at approximately 100 million floating point operations per second (mflops). Not counting assignments or memory references, and using Eq. D.11, we estimate that we require approximately 75 operations to calculate the gradient for one time step. If we also assume that we will use Euler's method to perform gradient descent, we expect that convergence may require on the order of 100's of time steps. So, for each new approximate rotation matrix, we expect that the time required to produce the orthonormalized matrix will be approximately 75 microseconds.

For the analog implementation, we base the performance analysis on previously reported speeds for the two transistor multiplier [Denyer 81]. Denyer reports that product rates in excess of 2 Mhz are easily achievable. Since the multipliers and constraint solution circuitry operate in parallel and in continuous time, we expect much faster convergence than in the digital discrete case described above, on the order of a few microseconds. So, the expected performance of the analog VLSI constraint solution can be roughly 25 times the performance of an aggressive digital implementation. We also can, by using other multiplier designs, reach even greater performance rates for the analog VLSI solution. Additional benefits of the analog VLSI solution include reduced power (vs. a workstation, or even a DSP), and much lower cost. The analog VLSI rotation constraint solution is implemented on a MOSIS tiny chip.

7.5 Summary

In this chapter, we have described a constraint technique for producing orthogonal, unit scale rotation matrices from "imperfect" inputs. The technique is potentially useful in a system which produces a sequence of approximate rotation matrices over time. One example of such a system involves producing rotation matrices from approximate inputs from interactive devices. Another possible approach produces approximate rotation matrices from angular velocity $\omega(t)$, via the following relation:

$$\underline{\underline{M}}'(t) = \underline{\omega}(t) \times \underline{\underline{M}}(t) \tag{7.6}$$

In Eq. 7.6, we show a vector-matrix cross product. By this expression, we mean the following:

$$M'_{ip} = \sum_{j=1}^{3}\sum_{k=1}^{3} \epsilon_{ijk}\omega_j M_{kp} \qquad i = 1,2,3, \quad p = 1,2,3 \tag{7.7}$$

where ϵ is defined as

$$
\begin{aligned}
\epsilon_{123} \;=\; \epsilon_{231} \;=\; \quad \epsilon_{312} &= \quad 1 \\
\epsilon_{321} \;=\; \epsilon_{213} \;=\; \quad \epsilon_{132} &= \; -1 \\
\textit{for all other } i,\ j,\ k, \quad \epsilon_{ijk} &= \quad 0 \ .
\end{aligned}
\tag{7.8}
$$

Such a system would produce an approximate rotation matrix at each time step, which could be corrected by the constraint technique described in this chapter. Additional potential applications are covered briefly in Sec. 7.1.

We also described the emerging and evolving technology of adaptive analog VLSI and speculate on its possible value to the field of computer graphics. In the example of the rotation system above, an analog VLSI rotation matrix constraint solver could enforce the rotation constraint *continuously* as the matrix is updated.

Interpreting this result with a broader view, we have demonstrated the implementation of a nontrivial constraint in analog VLSI. This is significant because it implies a future of implementing "hardware for modeling" in the form of hardware constraint solution. Current digital implementations of constraint systems cannot compute real time constraint solutions for models containing more than a few bodies. The advent of adaptive analog VLSI presents an exciting opportunity to consider building hardware to accelerate modeling to a level of performance commensurate with that of digital rendering hardware. We believe that Analog VLSI has the potential to be a significant tool for computer graphics.

Chapter 8

Toward On-chip Learning: Optimization, Adaptation, and Annealing

This chapter describes some experiments with on-chip optimization. We present an approach and results from an implementation of gradient estimation and on-chip gradient descent and annealing to optimize a multi-dimensional scalar function. In Sec. 8.1, we discuss alternative optimization techniques, and evaluate the benefits and potential flaws in our optimization approach. We also discuss potential applications of our gradient estimation and descent technique.

Applications include learning in a neural network system, which could be almost entirely on-chip. Key components of the system, such as references and evaluations of error metrics, will require some external (offchip) computation. Future research should improve the quality and reliability of onchip error metric and reference evaluation. We also anticipate the use of the gradient estimation and descent technique for setting parameters for arbitrary circuits. Other important applications relate to Ch. 6, for using gradient estimation, descent and annealing to assist in enforcing constraints.

8.1 Styles of Optimization

In Ch. 5, we described an optimization process using a digital computer as an external aid to constrained optimization. We used the computer-controlled instruments as an "electronic screwdriver" to effect adjustments under computer control. Aside from the drawback of requiring an external digital computer to perform the optimization on the chip parameters, there are other issues related to digital techniques for optimization that we discuss in this section.

Arguably, the simplest technique for optimization on a digital computer implementation is Euler's method. Euler's method has many well-known problems, including a sensitivity to step size. For multi-dimensional optimizations where the various parameters are at different scales, we may see oscillation, which is at best inefficient for swift convergence, and is sometimes disastrous. Techniques such as the conjugate gradient method [Gill 81] can improve performance for "difficult" optimization tasks.

Even when using sophisticated programs with adaptive step sizes, the choice of an appropriate step size can continue be a problem. One cost of a sophisticated technique is the time required to perform the additional computation. Another cost of sophisticated techniques is the complexity of the software: a large investment is required to produce a robust implementation of a complex technique. A mitigating positive factor is that these techniques can be cast as reusable software libraries, so the investment may be shared over multiple projects. Even with all of the algorithmic cleverness we can muster, convergence is still often slow, particularly for large-dimensional problems, since the processing time scales as N at best, where N is the number of parameters (dimensions) for the problem (N^2 for the conjugate gradient method).

Another alternative to consider is continuous time optimization, along with a parallel implementation. One major advantage of the continuous time approach is that the process is smooth, avoiding problems with step size. A direct and unavoidable cost of the smoothness is the susceptibility to noise. Section 8.2 describes a technique for continuous time optimization, and actually uses noise as part of the technique.

Many of the learning techniques described in the neural network literature [Rumelhart 86] are "model-based," in that the algorithms for updating neural connections are influenced by the knowledge of the "model," or, the network structure involved. In Ch. 5, we described a "special knowledge" optimization for the cochlea circuit, which allowed us to use the knowledge of what the effects of each input knob would be. While it is wasteful to ignore a priori information about the model, or system, often this knowledge is not available, or it varies with time. "Model-free" optimization is a more challenging problem, and has been the focus of our efforts.[1] In both the discrete and continuous case, we assume that we know very little about the structure of the problem or circuit; we can only know its inputs and outputs.

We also consider problems which are distinctive in that we already know something about the function, or model. We can then use our a priori knowledge to assist us in the optimization. This approach relates to some early work in on-chip optimization, using static connectivity circuits to solve particular linear and nonlinear programming problems [Pyne 56] [Chua 84]. One can assemble static circuits composed of resistors, capacitors, and amplifiers, which can solve nonlinear programming problems [Chua 84]. This is an example of model-based optimization, since the connection scheme is implied by the model. We are more interested in problems of a dynamic nature, where not only

[1] We enclose "model-free" in quotation marks since in some sense no representation is entirely free of some type of model. For example, in our "model-free" optimization, our model is that we are computing a deterministic function, which reliably produces consistent outputs, given identical inputs.

the values of parameters of the problem may change, but their relations, as well. We wish to be able to perform optimization on some arbitrary function, without requiring a priori knowledge of the internal structure of the problem. The next section describes an approach to on-chip optimization which operates in continuous time and in a "model-free" fashion.

8.2 Analog VLSI Implementation of Multi-dimensional Gradient Descent

We describe an analog VLSI implementation of a multi-dimensional gradient estimation and descent technique for minimizing an on-chip scalar function $f()$. The implementation uses noise injection and multiplicative correlation to estimate derivatives, as in [Anderson, Kerns 92]. One intended application of this technique is setting circuit parameters on-chip automatically, rather than manually [Kirk 91]. Gradient descent optimization may be used to adjust synapse weights for a backpropagation or other on-chip learning implementation. The approach combines the features of continuous multi-dimensional gradient descent and the potential for an annealing style of optimization. We present data measured from our analog VLSI implementation.

This work is similar to [Anderson, Kerns 92], but represents two advances. First, we describe the extension of the technique to multiple dimensions. Second, we demonstrate an implementation of the multi-dimensional technique in analog VLSI, and provide results measured from the chip. Unlike previous work using noise sources in adaptive systems, we use the noise as a means of estimating the gradient of a function $f(\underline{y})$, rather than performing an annealing process [Alspector 88]. We also estimate gradients continuously in position and time, in contrast to [Umminger 89] and [Jabri 91], which utilize discrete position gradient estimates.

It is interesting to note the existence of related algorithms, developed at approximately the same time [Cauwenberghs 93] [Alspector 93] [Flower 93]. The main difference between our approach and the related algorithms is that our implementation operates in continuous time, with continuous differentiation and integration operators. The other approaches realize the integration and differentiation processes as discrete addition and subtraction operations, and use unit perturbations. [Cauwenberghs 93] provides a detailed derivation of the convergence and scaling properties of the discrete approach, and a simulation. [Alspector 93] provides a description and simulation of the use of the technique as part of a neural network hardware architecture. [Flower 93] derived a similar discrete algorithm from a node perturbation perspective in the context of multi-layer feedforward networks. We can limit our technique to discrete, unit perturbations (instead of noise), discrete differentiation (subtraction), and discrete integration (addition), to make our approach more closely resemble [Cauwenberghs 93] [Alspector 93] [Flower 93]. Consequently, we believe that our technique is more general.

Our work is similar in spirit to [Dembo 90] in that we don't make any explicit assumptions about the "model" that is embodied in the function $f()$. The function may be implemented as a neural network and its error metric for learning, and the multiple parameters are the input dimensions of the function. In that case, the gradient descent is on-chip learning of the parameters of the network, learning appropriate weight values.

We have fabricated a working chip containing the continuous-time multi-dimensional gradient descent circuits. This chapter includes chip data for individual circuit components, as well as the entire circuit performing multi-dimensional gradient descent and annealing.

8.3 The Gradient Estimation Technique

Anderson and Kerns [Anderson, Kerns 92] describe techniques for one-dimensional gradient estimation in analog hardware. The gradient is estimated by correlating (using a multiplier) the output of a scalar function $f(v(t))$ with a noise source $n(t)$, as shown in Fig. 8.1. The function input $y(t)$ is additively "contaminated" by the noise $n(t)$ to produce $v(t) = y(t) + n(t)$. A scale factor B is used to set the scale of the noise to match the function output, which improves the signal-to-noise ratio. The signals are "high-pass" filtered to approximate differentiation (shown as d/dt operators in

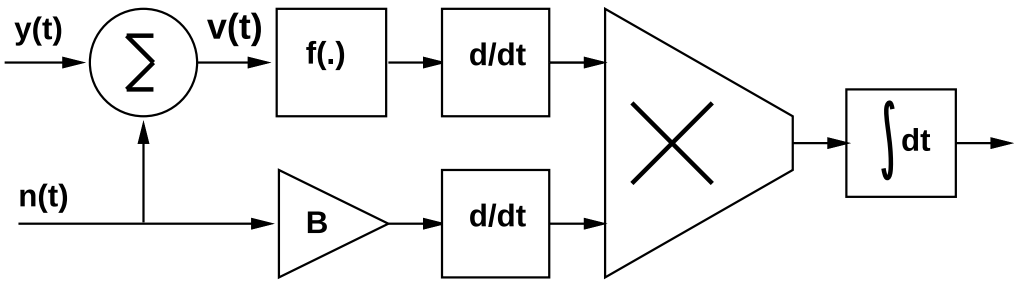

Figure 8.1: Gradient estimation technique from [Anderson, Kerns 92] □

Fig. 8.1) directly before the multiplication. The results of the multiplication are "low-pass" filtered to approximate integration.

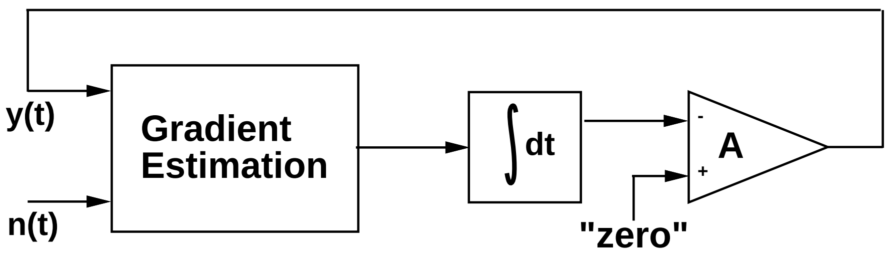

Figure 8.2: Closing the loop: performing gradient descent using the gradient estimate. □

The gradient estimate is integrated over time, to smooth out some of the noise and to damp the response. This smoothed estimate is compared with a "zero" reference, using an amplifier A, and the result is fed back to the input, as shown in Fig. 8.2. The contents of Fig. 8.1 are represented by the "Gradient Estimation" box in Fig. 8.2. The "zero" reference in Fig. 8.2 is an example of a required external reference, as mentioned in Sec. 3.5. At some point en *every* onchip optimization process, we require at least one absolute, accurate reference.

We have chosen to implement the multi-dimensional technique in analog VLSI. We will not reproduce here the one-dimensional analysis from [Anderson, Kerns 92], but summarize some of the more important results, and provide a multi-dimensional derivation. [Anderson 92] provides a more detailed theoretical discussion.

8.4 Multi-dimensional Derivation

The multi-dimensional gradient descent operation that we are approximating can be written as follows:

$$\underline{y}'(t) = -k\nabla f(\underline{y}(t)) \tag{8.1}$$

where $\underline{y}$ and $\underline{y}'$ are vectors, and the solution is obtained continuously in time t, rather than at discrete t_i. The circuit described in the block diagram in Fig. 8.1 computes an approximation to the gradient:

$$\nabla f = \frac{\partial f}{\partial y_i} \approx \int \left(\frac{d}{dt} f\left(\underline{y}(t) + \underline{n}(t)\right) n_i'(t) \right) dt \quad . \tag{8.2}$$

We approximate the operations of differentiation and integration in time by realizable high-pass and low-pass filters, respectively. To see that Eq. 8.2 is valid, and that this result is useful for approximating Eq. 8.1, we sketch an N-dimensional extension of [Anderson 92]. Using the chain rule,

$$\frac{d}{dt} f\left(\underline{y}(t) + \underline{n}(t)\right) = \sum_j \left(y_j'(t) + n_j'(t)\right) \frac{\partial f}{\partial y_j} \quad . \tag{8.3}$$

Assuming $n_j'(t) \gg y_j'(t)$, the rhs is approximated to produce

$$\frac{d}{dt} f\left(\underline{y}(t) + \underline{n}(t)\right) = \sum_j n_j'(t) \frac{\partial f}{\partial y_j} \quad . \tag{8.4}$$

Multiplying both sides by $n_i'(t)$, and taking the expectation integral operator $E[\,]$ of each side,

$$E\left[n_i'(t) \frac{d}{dt} f\left(\underline{y}(t) + \underline{n}(t)\right) \right] = E\left[n_i'(t) \sum_j n_j'(t) \frac{\partial f}{\partial y_j} \right] \quad . \tag{8.5}$$

If the noise sources $n_i(t)$ and $n_j(t)$ are uncorrelated, $n_i'(t)$ is independent of $n_j'(t)$ when $i \neq j$, and the sum on the right has a contribution only when $i = j$,

$$E\left[n_i'(t)\frac{d}{dt}f\left(\underline{y}(t)+\underline{n}(t)\right)\right] = E\left[n_i'(t)n_i'(t)\frac{\partial f}{\partial y_i}\right] \tag{8.6}$$

$$E\left[n_i'(t)\frac{d}{dt}f\left(\underline{y}(t)+\underline{n}(t)\right)\right] \approx \alpha\frac{\partial f}{\partial y_i} = \alpha\nabla f \quad . \tag{8.7}$$

The expectation operator $E[\ ]$ can be used to smooth random variations of the noise $n_i(t)$. So, we have

$$\nabla f = \frac{\partial f}{\partial y_i} \approx \frac{1}{\alpha}E\left[n_i'(t)\frac{d}{dt}f\left(\underline{y}(t)+\underline{n}(t)\right)\right] \quad . \tag{8.8}$$

Since the descent rate k is arbitrary, we can absorb α into k. Using Eq. 8.8, we can approximate the gradient descent technique as follows:

$$y_i'(t) \approx -\hat{k}\ E\left[n_i'(t)\frac{d}{dt}f\left(\underline{y}(t)+\underline{n}(t)\right)\right] \quad . \tag{8.9}$$

8.5 Elements of the Multi-dimensional Implementation

We have designed, fabricated, and tested a chip which allows us to test the noise injection approach to multidimensional gradient estimation and descent. The chip implementation can be decomposed into six distinct parts:

noise source(s): an analog VLSI circuit which produces a noise function. An independent, correlation-free noise source is needed for each input dimension, designated $n_i(t)$. The noise circuit is described in [Alspector 91]. As mentioned in Sec. 8.2, we could use a more constrained input such as clocked unit perturbations, ramps, etc.

target function: a scalar function $f(y_1, y_2, \cdots, y_N)$ of N input variables, bounded below, which is to be minimized [Kirk 91]. The circuit in this case is a 4-dimensional variant of the bump circuit described in [Delbrück 91]. In the general case, this $f()$ can be any scalar function or error metric, computed by some circuit. Specifically, the function may be a neural network learning performance metric, and parameters to be set are then the network's weights.

input signal(s): the inputs $y_i(t)$ to the function $f()$. These will typically be onchip values for an onchip learning application, or real-world inputs for an interaction or sensing task. For the purposes of this implementation, the inputs are provided from off-chip.

multiplier circuit(s): the multiplier computes the correlation between the noise values and the function output. Offsets in the multiplication appear as systematic errors in the gradient estimate, so it is important to compensate for the offsets. Linearity is not especially important, although monotonicity is critical. Ideally, the multiplication will also have a "tanh-like" character, limiting the output range for extreme inputs.

integrator: an integration over time is approximated by a low-pass filter

differentiator: the time derivatives of the noise signals and the function are approximated by a high-pass filter.

The N inputs, $y_i(t)$, are additively "contaminated" with the noise signals, $n_i(t)$, by capacitive coupling, producing $v_i(t) = y_i(t) + n_i(t)$, the inputs to the function $f()$. The function output is differentiated, as are the noise functions. Each differentiated noise signal is correlated with the differentiated function output, using the multipliers. The results are low-pass filtered, providing N partial derivative estimates, for the N input dimensions, shown for 4 dimensions in Fig. 8.3.

The function $f()$ is implemented as an 4-dimensional extension of Delbrück's [Delbrück 91] bump circuit, described in detail in Appendix C. For learning and other applications, the function $f()$ can implement some other error metric to be minimized.

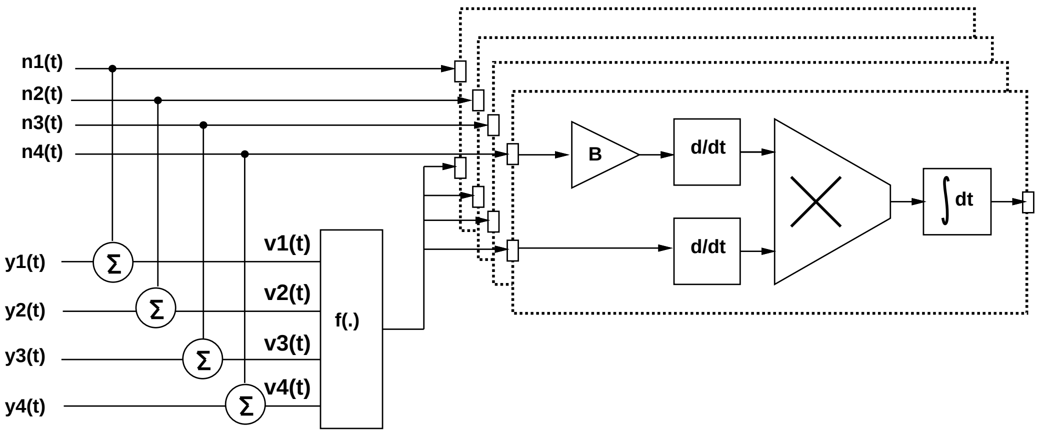

Figure 8.3: Block diagram for a 4-dimensional gradient estimation circuit. □

8.6 Chip Results

We have tested chips implementing the gradient estimation and gradient descent techniques described in this chapter. Figure 8.5 shows the gradient estimate, without the closed loop descent process. Figure 8.6 shows the trajectories of two state variables during the 2D gradient descent process. Figure 8.7 shows the gradient descent process in operation on a 2D bump surface, and Fig. 8.8 shows how, using appropriate choice of noise scale, we can perform annealing using the gradient estimation hardware.

In Appendix C, we present data from a simulation of a 4-dimensional implementation of the N-d bump circuit. Results from [Delbrück 91] indicate that the bump circuit simulation is very similar to what can be expected from the chip.

We also fed the noise data into the 4-d bump circuit simulation, and differentiated and correlated the results, in order to simulate the entire system. We chose random initial conditions for the 4 state variables, and used the gradient estimates to perform gradient descent. Figure 8.4 shows two parameter choices for that simulation. Figure 8.4a) shows the simulation for small magnitude of noise and very short integration time for the gradient estimate. The trajectories of the 4 state variables, y_1, y_2, y_3, and y_4, are shown over time as they each approach 1.5 volts, which minimizes

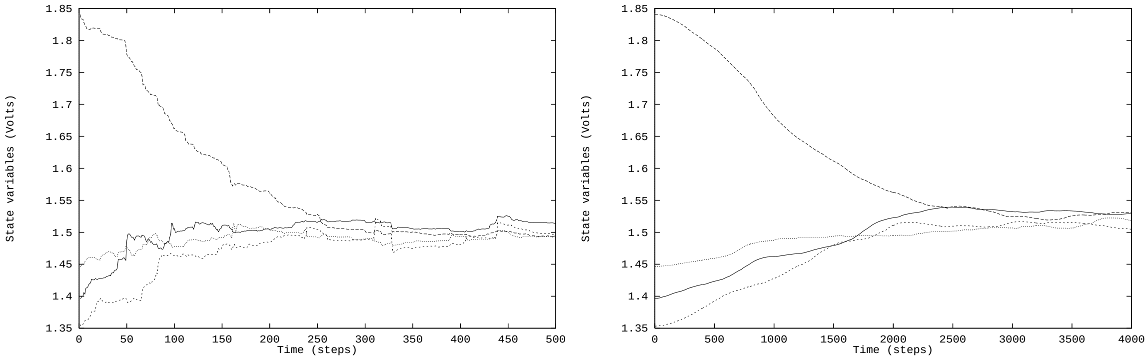

Figure 8.4: Gradient descent simulation results for (a) small noise magnitude and short integration time, and (b) larger noise magnitude and long integration time. Each graph shows the optimization path taken by the 4 state variables over time. Each variable's target value is 1.5 volts. □

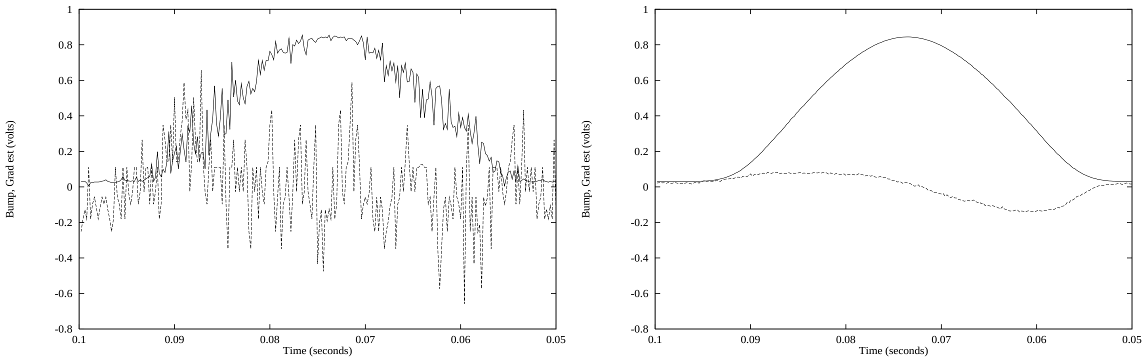

Figure 8.5: Measured Chip Data: 1D Gradient Estimate. Upper curves are 1D bump output as the input $y(t)$ is a slow triangle wave. Lower curves are gradient estimates. (left) raw data, and (right) average of 1024 runs. Note that while the gradient estimate is qualitatively correct, the magnitude is not symmetric about zero. This is due to the fact that the correlating multiplier that we used is not symmetric in its response. □

the scalar function. Figure 8.4b) shows a longer integration time with larger scale noise input (noise magnified 5x).

8.7 Summary

We have implemented an analog VLSI structure for performing continuous multi-dimensional gradient descent, and the gradient estimation uses only local information. The circuitry is compact

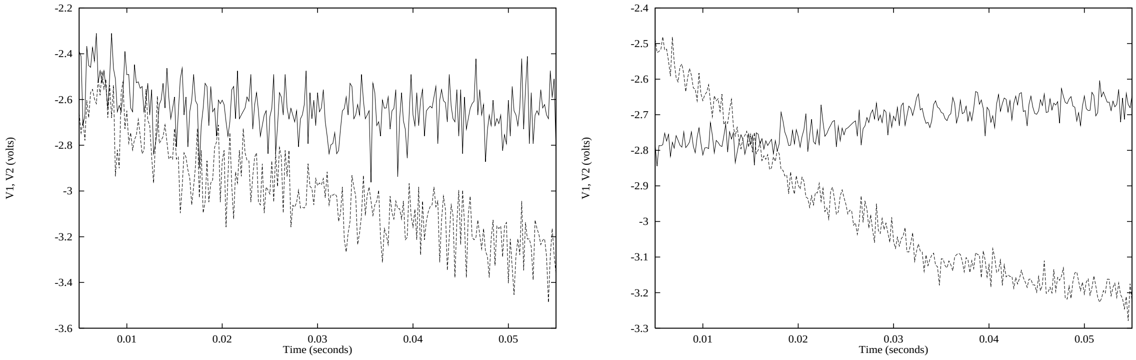

Figure 8.6: Measured Chip Data: 2D Gradient Descent. The curves above show the function optimization by gradient descent for 2 variables. Each curve represents the path of one of the state variables $\underline{y}(t)$ from some initial values to the values for which the function $f()$ is minimized. (left) raw data, and (right) average of 8 runs. □

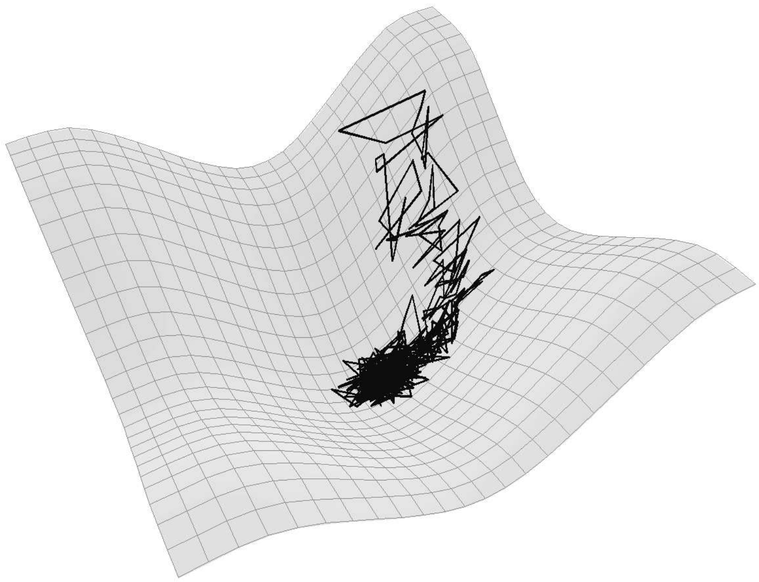

Figure 8.7: Measured Chip Data: 2D Gradient Descent. Here we see the results for 2D gradient descent on a 2D bump surface. Both the bump surface and the descent path are actual data measured from our chips. □

and easily extensible to higher dimensions. This implementation leads to on-chip multi-dimensional optimization, such as is needed to perform on-chip learning for a hardware neural network.

The system performs robustly, and converges quite rapidly for the examples that we have exam-

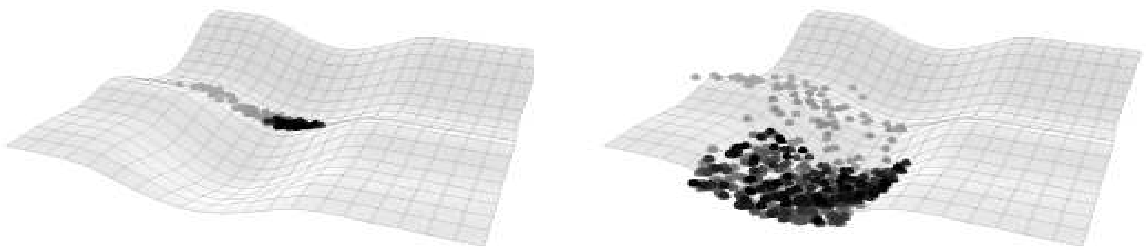

Figure 8.8: Measured Chip Data: 2D Gradient Descent and Annealing. Here we see the effects of varying the amplitude of the noise. The dots represent points along the optimization path. At left, with small magnitude noise, the process descends to a local minimum. At right, with larger magnitude, the descent process escapes to the global minimum. A schedule of gradually decreasing noise amplitude could reduce the probability of getting caught in undesirable local minima, and increase the probability of converging to a small region near a more desirable minimum, or even the global minimum. □

ined. Although the data shown illustrates convergence on a timescale of roughly 50 milliseconds, we have measured convergence rates on the order of microseconds, using large amplitude noise. Since the convergence rate is related to the noise input clock and the capacitances in the circuit, we believe that even greater performance is possible with careful design. Future research should explore the potential for greater performance using the gradient estimation and descent technique described herein.

We compare the performance of the gradient estimation and descent circuit to a digital implementation, running on a high-performance workstation. Using the same assumptions as in Sec. 7.4, we can imagine a digital implementation of a simple one-dimensional optimization converging in 75 microseconds. As the dimensionality of the problem increases, the convergence time for the digital implementation degrades linearly, as N, the number of dimensions, increases.

The real advantage of the analog VLSI gradient estimation and descent becomes apparent in a large multi-dimensional optimization. The time required for convergence in the analog implementation scales as $\sqrt{N}$, where N is the number of dimensions. The performance degradation occurs as a signal to noise ratio reduction as more components of the gradient are estimated using the same perturbed function evaluation. In low-dimensional problems, we apparently have "signal to spare" since the convergence for a 2-dimensional problem is not measurably slower than for a 1-dimensional problem.

An additional benefit of the analog implementation is provided if we also implement the multi-dimensional function in analog VLSI as well. Our gradient estimation approach operates by evaluating that function continuously over time. In the digital implementation, we must evaluate the function N times, once for each dimension, at each step.

As described in Sec. 8.6, we can also perform a kind of annealing style of optimization by choosing large noise amplitude. We have successfully used this technique to demonstrate an escape from a local minimum, as shown in Fig. 8.8. We see from this figure that when the noise amplitude is small, the optimization process descends to a local minimum. As shown at the right side of Fig. 8.8, when we use a larger amplitude of noise, the descent process escapes to the global minimum. One drawback of the use of larger amplitude noise is that the descent process converges to a larger neighborhood about the minimum. This suggests that future work might focus on adding a schedule to the scale of the noise input. A gradually decreasing scale for the noise amplitude may produce results similar in quality to simulated annealing. We anticipate future work to provide a theoretical analysis for the annealing process performed by our implementation, in order that our approach may be compared to simulated annealing.

Our approach also has some drawbacks, however. The gradient estimation is sensitive to the input offsets in the multipliers and integrators, since those offsets result in systematic errors. Although we discussed the choice of "zero" values briefly in Sec. 8.3, and earlier, in Sec. 3.5, this is an important problem that deserves more attention. Since we used uncompensated multipliers and amplifiers in the gradient estimation circuitry, the "zero" values for the feedback described in Fig. 8.2 were selected by hand (actually, using a digital computer program, but off-chip, at any rate). Future work could increase the ease of use of the gradient estimation approach by producing circuits which adapt to these offsets. We believe, however, that there will always exist the need for some sort of external reference.

Another drawback of the gradient estimation technique is that we add noise to the input signals. If smooth signals are required, then we must apply some additional circuitry for integration and smoothing to present the conditioned signal to the outside world.

Finally, we hope that with only small additional circuit complexity, the performance of analog VLSI circuits can be greatly increased by permitting them to be intrinsically adaptive. On-chip implementation of an approximate gradient descent technique is an important step in this direction.

It is especially appropriate to consider the implementation of learning for neural networks using analog VLSI. Since we must discover the offsets of the analog circuits in order to perform offset compensation, there is already a learning process involved in our use of the analog circuits. For a neural network implementation, we also require a learning process in order to set the weights. We can, in an analog neural network implementation, lump together the process of learning the network weights and the process of learning the circuit offsets. In combining the two learning processes, we remove a great part of the stigma of the variability of the components available for analog computation. We look forward to future work using the gradient estimation and descent technique to develop robust onchip learning applications.

Chapter 9

Conclusions

Results of this thesis include several concrete projects and some generally applicable techniques for constructing adaptive analog VLSI systems. We recap the specific projects described in these thesis and draw some conclusions from the work. We also discuss the significance of these projects in terms of their contribution to the fields of computer graphics, neural networks, and analog circuit design. Based on the results that we present, a set of areas for future work are identified.

9.1 Engineering Design Methodology for Analog VLSI

The conclusions that we draw from Ch. 5 are that the quality and performance of analog VLSI designs can be enhanced by the adoption of a goal-based engineering design methodology. A key component of the goal-based technique is that the goals for the system and sub-systems should be planned explicitly (not tacitly) as part of the design process. As part of this approach, a complete design consists of not only the circuit and layout, but metrics for expected performance, adjustable parameters in the circuit design to permit improved accuracy and precision, and an optimization plan which modifies the parameters to enhance the system performance.

Chapter 7 describes a project that uses the goal-based technique as explicitly part of the design procedure. The results of the work on rotation matrix constraints indicate that the goal-based technique can have wide application in many systems involving constraint satisfaction. Although we chose to demonstrate a constraint technique as applied to a particular field, the general ideas are likely to be applicable not only in computer graphics, but also in other fields, such as robotics and other disciplines requiring simulation.

9.2 Toward Harnessing Analog VLSI for Quantitative Computation

Another result from Chapters 5, 6, and 7 is that the use of constrained optimization and a goal-based design technique can produce analog VLSI circuits and systems that perform quantitatively better than designs created without the use of these ideas.

To increase the accuracy of analog computation, we developed techniques for creating *compensated* circuit building blocks, where compensation implies the cancellation of device variations, offsets, and nonlinearities. These compensated building blocks are used as components in larger and more complex circuits, which are then also compensated. To this end, we developed techniques in Ch. 5 for automatically determining appropriate parameters for circuits, using constrained optimization. We also fabricated circuits that implement multi-dimensional gradient estimation for a gradient descent optimization technique. The gradient estimation and descent techniques and related circuits were described in Ch. 8. The parameter-setting and optimization tools allowed us to automatically choose values for compensating our circuit building blocks, based on our goals for the circuit performance. We also used the techniques to optimize parameters for larger systems, applying the goal-based techniques hierarchically, as described in Ch. 6.

9.3 Future Work

In addition to the work completed in this thesis, we can identify a few areas of potentially fruitful research, based on the work describe herein.

9.3.1 Design for Testability in Analog Circuits

The process of using goal-based design techniques requires that many internal circuit signals are available externally, or at least, available to some measurement circuitry. The combination of this availability and the explicit (rather than tacit) elucidation of performance goals implies an improved testability for such circuits. The desired performance of circuits can be easily compared with the actual performance, making it easier to test and validate the fabricated design.

9.3.2 Automatic Configuration and Adaptation in Analog Circuits

An additional benefit can be realized by the integration of the compensatable circuit blocks with on-chip optimization techniques also described in this thesis. Together, the two techniques can help to produce self-configuring analog circuits. A need was identified and described in Chapters 6 and 8 for increased automation of the process of adaptation to offsets and other variations. Of particular interest are techniques for automatic selection of "zero" values for gradient feedback optimization schemes. One important area for future research envolves the extension of the adaptation process to require fewer external reference and error metric evaluation signals. We are also interested in the application of our design methodology to larger and more complex circuits, implying the development of more sophisticated techniques for specification of hierarchical goals.

A related desire is an increasingly automated procedure for producing mappings from numbers and mathematical functions to signals in analog VLSI, embodied in our G function. In our analog computation experiments, the signal representations and number to signal mappings (and inverse mappings) were all chosen "by hand," explicitly. The design of more quantitative analog computing systems can be made easier and more automatic by the development of a procedural methodology for mapping numbers to signals.

9.3.3 Low-power Computer Graphics Subsystems

Following on the development path begun in Ch. 7, one can imagine using constraints implemented in analog VLSI more extensively in computer graphics systems. This is particularly attractive for parallelism, speed, and the low-power nature of analog computation. We can use analog for constraint satisfaction in modeling, as well as lighting, rendering and other display tasks. Since computer graphics display has comparatively low precision requirements (screen resolution is 10 bits and color resolution is typically 8 bits), it seems that applying analog VLSI to rendering is a nice stepping stone to a greater use of analog computing for graphics.

9.3.4 On-chip Learning for Neural Networks

The optimization experiments in Ch. 8 show a potential for some very exciting analog VLSI learning circuits for neural networks. Since the gradient estimation and descent does not depend on a particular network architecture, it is amenable to many styles of neural computation. Furthermore, since the convergence time for the noise injection and correlation technique scales only as $\sqrt{N}$ in the number of parameter dimensions, N, the technique may be applicable to systems with a very large numbers of parameters.

Also, as described in Sec. 8.6, we used the gradient estimation technique to perform an annealing style of optimization by choosing large noise amplitude. We have successfully used this technique to demonstrate an escape from a local minimum, as shown in Fig. 8.8. When the noise amplitude was small, the optimization process descended to a local minimum. When we used a larger amplitude of noise, the descent process escaped to the global minimum. One drawback of the use of larger amplitude noise was that the descent process converged to a larger neighborhood about the minimum. This suggests that future work might focus on adding a schedule to the scale of the noise input.

A gradually decreasing scale for the noise amplitude may produce results similar in quality to simulated annealing. We believe that an annealing approach may be used successfully to enhance the performance of the currently popular backpropagation techniques that are often applied to neural network training.

Appendix A

Equations of Operation of a Compensated Amplifier

For the singly compensated wide-range transconductance amplifier, as shown in Fig. 6.2, we have the following relation between output voltage and the input voltages:

$$V_{out} = A(V_1 - V_f) \tag{A.1}$$

and the following expression for the capacitances:

$$C_{of}(V_{out} - V_f) = C_{if}(V_f - V_2) \tag{A.2}$$

(A.3)
$$C_{of}V_{out} - C_{of}V_f = C_{if}V_f - C_{if}V_2$$

(A.4)
$$C_{of}V_{out} + C_{if}V_2 = C_{if}V_f + C_{of}V_f$$

(A.5)
$$\frac{C_{of}V_{out} + C_{if}V_2}{C_{if} + C_{of}} = V_f \quad .$$

Substituting back into Eq. A.1, we get:

(A.6)
$$V_{out} = A\left(V_1 - \frac{C_{of}V_{out} + C_{if}V_2}{C_{if} + C_{of}}\right)$$

(A.7)
$$V_{out}(C_{if} + C_{of}) = \quad AV_1(C_{if} + C_{of}) - AC_{of}V_{out} - AC_{if}V_2$$

(A.8)
$$V_{out}(C_{if} + C_{of} + AC_{of}) = \quad AV_1(C_{if} + C_{of}) - AC_{if}V_2$$

(A.9)
$$V_{out} = \frac{AV_1(C_{if} + C_{of}) - AC_{if}V_2}{C_{if} + C_{of} + AC_{of}}$$

(A.10)
$$V_{out}\left[1 + (1/A)(1 + C_{if}/C_{of})\right] = \quad V_1(1 + C_{if}/C_{of}) - V_2(C_{if}/C_{of}) \quad .$$

For the doubly compensated wide-range transconductance amplifier, as shown in Fig. 6.3, we have the following relation between output voltage and the input voltages:

$$V_{out} = A(V_f^1 - V_f^2) \quad . \tag{A.11}$$

We now also have two sets of capacitors, for the two floating nodes. So, we have a pair of relations for the capacitive division:

$$C_{of}^1(0 - V_f^1) = C_{if}^1(V_f^1 - V_1) \tag{A.12}$$

$$C_{of}^2(V_{out} - V_f^2) = C_{if}^2(V_f^2 - V_2) \quad . \tag{A.13}$$

Solving for V_f^1, we get:

$$-C_{of}^1 V_f^1 = C_{if}^1 V_f^1 - C_{if}^1 V_1 \tag{A.14}$$

$$C_{if}^1 V_1 = C_{if}^1 V_f^1 + C_{of}^1 V_f^1 \tag{A.15}$$

$$V_f^1 = \frac{C_{if}^1 V_1}{C_{if}^1 + C_{of}^1} \tag{A.16}$$

and, correspondingly:

$$V_f^2 = \frac{C_{of}^2 V_{out} + C_{if}^2 V_2}{C_{if}^2 + C_{of}^2} \quad . \tag{A.17}$$

Substituting back into Eq. A.11, we get:

$$V_{out} = A\left(\frac{C^1_{if}V_1}{C^1_{if}+C^1_{of}} - \frac{C^2_{of}V_{out}+C^2_{if}V_2}{C^2_{if}+C^2_{of}}\right) \quad . \tag{A.18}$$

If we assume that the ratio $C^1_{if}/C^1_{of} = C^2_{if}/C^2_{of} = 1/B$, then

$$V_{out} = A\left(\frac{V1}{1+B} - \frac{V_{out}}{1+1/B} - \frac{V2}{1+B}\right) \tag{A.19}$$

$$\frac{V_{out}}{A} + \frac{V_{out}}{1+1/B} = \frac{V1-V2}{1+B} \tag{A.20}$$

$$\frac{V_{out}(1+B)(A+1+1/B)}{A(1+1/B)} = V1 - V2 \tag{A.21}$$

$$\frac{V_{out}(1+B)(AB+B+1)}{A(B+1)} = V1 - V2 \tag{A.22}$$

$$V_{out}\left(B + \frac{B+1}{A}\right) = V1 - V2 \quad . \tag{A.23}$$

If we assume that AB is much larger than $(1+B)$, then we have

$$V_{out} = \frac{1}{B}(V_1 - V_2) \quad . \tag{A.24}$$

Appendix B

Equations of Operation of Compensated Gilbert Multiplier

We could construct a compensated multiplier by adding capacitive input structures to a Gilbert transconductance multiplier or a wide-range transconductance multiplier [Mead 89]. A straightforward version has no feedback; there is merely capacitive division on the inputs.

For the doubly compensated transconductance multiplier, we have the following relation between output voltage in terms of four of the node voltages:

$$I_{out} = tanh(V_f^1 - V_f^2) tanh(V_f^3 - V_f^4) \tag{B.1}$$

and, assuming that we remain in the linear ranges of the tanh functions, we have:

(B.2)
$$I_{out} = A_1(V_f^1 - V_f^2)A_2(V_f^3 - V_f^4)$$

since the operation of the Gilbert multiplier is similar to the product of responses of two amplifiers.

We also have four sets of capacitors, for the four floating nodes. So, we have a quartet of relations for the input capacitive divisions:

(B.3)
$$C_{of}^1(0 - V_f^1) = C_{if}^1(V_f^1 - V_1)$$

(B.4)
$$C_{of}^2(0 - V_f^2) = C_{if}^2(V_f^2 - V_2)$$

(B.5)
$$C_{of}^3(0 - V_f^3) = C_{if}^3(V_f^3 - V_3)$$

(B.6)
$$C_{of}^4(0 - V_f^4) = C_{if}^4(V_f^4 - V_4) \quad .$$

Using Eq. B.3 and solving for V_f^1, we get:

(B.7)
$$-C_{of}^1 V_f^1 = C_{if}^1 V_f^1 - C_{if}^1 V_1$$

(B.8)
$$C_{if}^1 V_1 = C_{if}^1 V_f^1 + C_{of}^1 V_f^1$$

(B.9)
$$V_f^1 = \frac{C_{if}^1 V_1}{C_{if}^1 + C_{of}^1}$$

and, correspondingly:

(B.10)
$$V_f^2 = \frac{C_{if}^2 V_2}{C_{if}^2 + C_{of}^2}$$

(B.11)
$$V_f^3 = \frac{C_{if}^3 V_3}{C_{if}^3 + C_{of}^3}$$

(B.12)
$$V_f^4 = \frac{C_{if}^4 V_4}{C_{if}^4 + C_{of}^4} \quad .$$

Substituting back into Eq. B.2, we get:

(B.13)
$$V_{out} = A_1 \left(\frac{C_{if}^1 V_1}{C_{if}^1 + C_{of}^1} - \frac{C_{if}^2 V_2}{C_{if}^2 + C_{of}^2} \right) A_2 \left(\frac{C_{if}^3 V_3}{C_{if}^3 + C_{of}^3} - \frac{C_{if}^4 V_4}{C_{if}^4 + C_{of}^4} \right) \quad .$$

If we assume that the ratio $C_{if}^1/C_{of}^1 = C_{if}^2/C_{of}^2 = C_{if}^3/C_{of}^3 = C_{if}^4/C_{of}^4 = 1/B$, then

(B.14)
$$V_{out} = A_1 \left(\frac{V_1}{1+B} - \frac{V_2}{1+B} \right) A_2 \left(\frac{V_3}{1+B} - \frac{V_4}{1+B} \right)$$

(B.15)
$$V_{out} = A_1 A_2 \left(\frac{V_1 - V_2}{1+B} \right) \left(\frac{V_3 - V_4}{1+B} \right)$$

(B.16)
$$V_{out} = \frac{A_1 A_2}{(1+B)^2}(V_1 - V_2)(V_3 - V_4) \quad .$$

In this design, the relation of the output range to the input range is determined by the ratio

(B.17)
$$\frac{V_{out}}{(V_1 - V_2)(V_3 - V_4)} = \frac{A_1 A_2}{(1+B)^2} \quad .$$

We anticipate that it is possible to design a version of the capacitive input structures for the Gilbert multiplier using feedback, but we have not explored such a design.

Appendix C

Description of Operation of N-Dimensional Bump Circuit

This appendix describes the operation of an extension to Delbrück's bump-anti-bump circuit [Delbrück 91]. Delbrück's circuit computes a smooth similarity measure of two voltages. As such, it computes a basis function in one dimension. Delbrück's circuit is shown in Fig. C.1.

The equations of operation of the bump circuit are shown in Eq. C.1.

$$I_{out} = \frac{I_0}{1 + \frac{4}{w} cosh^2(\frac{\kappa \Delta V}{2})} \quad . \tag{C.1}$$

If we wish to produce a basis function in N dimensions, where N is greater than one, we must

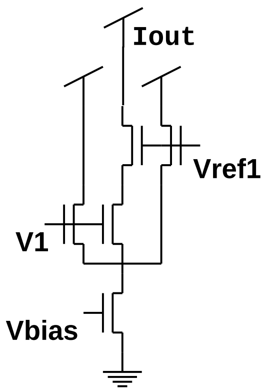

Figure C.1: Delbrück's bump circuit. □

extend the circuit, or develop another circuit entirely. Delbrück's circuit is very well-behaved, so it is desirable to extend it, if we can.

For learning and other applications, the N-dimensional basis function can implement a smooth error metric for function minimization. It can also be used as a basis function for function interpolation neural networks that operate by summing a collection of basis functions.

One way to compute an N-dimensional basis function from a collection of 1-dimensional basis functions is to multiply them together. We can use the output current from a single bump circuit (Fig. C.1) as the bias current for another bump circuit. This variant is shown in Fig. C.2. The main problem with this circuit is that with the long stack of paired transistors in the center leg, we eventually run out of range (too many diode drops). An additional problem is that the center transistors do not have the same source and drain voltages in each "dimension" (V1, V2, V3, V4), and so the bumps are dissimilar in the various dimensions.

The problem with the stack of transistors can be alleviated by flipping alternate stages (dimensions) upside-down, and using pfets instead of nfets for the odd-numbered stages. This variation is shown in Fig. C.3. This version also has the disadvantage that the odd-numbered stages will behave differently than the even-numbered stages, since they are pfets.

This problem, too, can be eliminated by adding an additional current mirror between the stages,

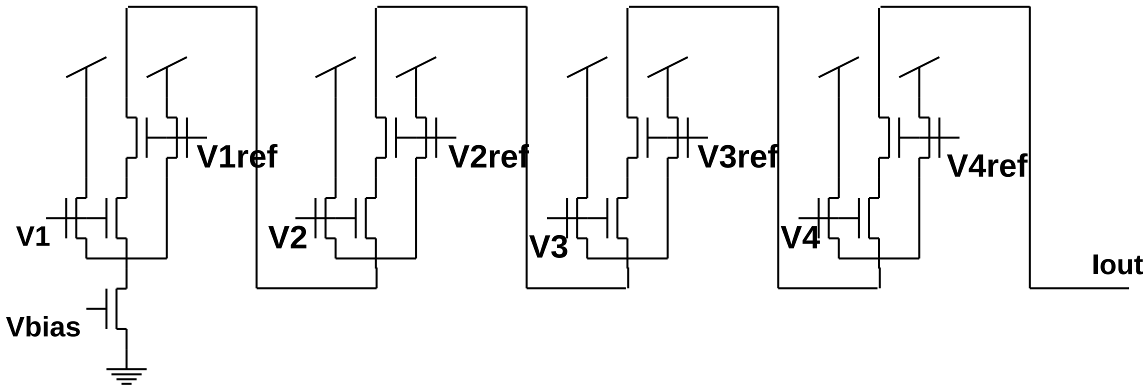

Figure C.2: A stack of Delbrück's bump circuits. □

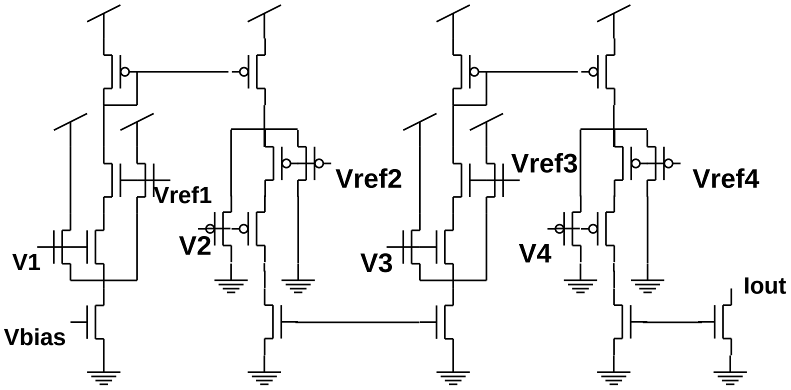

Figure C.3: A simple 4-dimensional extension of Delbrück's bump circuit, using alternating P and N bump stages. □

which permits all 4 (in this example) stages to be nfets, with similar source and drain voltages. This variant is shown in Fig. C.4.

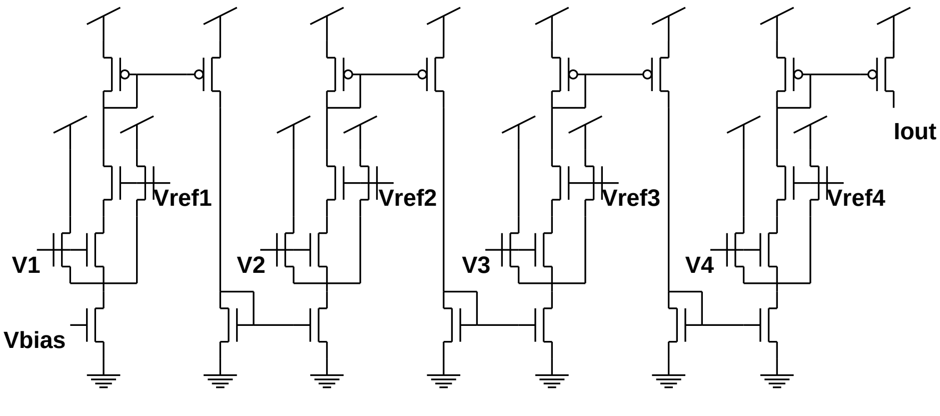

Figure C.4: An improved 4-dimensional extension of Delbrück's bump circuit, using pairs of current mirrors. □

The circuit shown in Fig. C.4 implements a 4-dimensional basis function, centered at (Vref1, Vref2, Vref3, Vref4), addressed by (V1, V2, V3, V4). Each of the inputs will behave similarly. The circuit can be extended to an arbitrary number of dimensions by adding the appropriate number of additional stages. So, the circuit grows linearly in size as the dimensionality is increased. A simulation of the operation of the 4-D bump circuit (Fig. C.4) is shown in Fig. C.7.

As we consider the evaluation of basis functions in higher dimensional spaces, we can foresee another, more subtle problem. In a low-dimensional space, say, two, a basis function occupies a significant amount of volume in that space. As the dimensionality increases, unless the basis functions are extremely wide, they occupy an increasingly small fraction of the volume of the space. This is because the parameters of any one dimension can cause the output to be small (if V1 is far from Vref1, the output current Iout will be small regardless of the values of V2, V3, etc.).

This effect may be undesirable, particularly if we are using the basis function to represent an error metric function. We get no signal if any parameter is out of range. With the addition of a small amount of circuitry and a few more operating parameters, this effect, too, can be ameliorated. Figure C.5 shows the circuit for 4 dimensions. We have added a means for copying the current from successive stages, as well as having individual bias currents for each stage (dimension). The output current is read as a sum of (I1, I2, I3, I4).

The amount of current applied for each bump circuit is controllable via a scale factor on the current mirrors, Vscale. We can also "close the loop" by connecting Vcopy to Vcopyin, thus allowing the current to be copied from the last stage back to the first. This allows us to remove some of the dependence on the order of connection of the individual bump circuits. This circuit produces a response somewhere between that of the purely multiplied basis functions, as in the circuit of Fig. C.4, and the response of a set of separate bumps added together.

A further variation of the circuit in Fig. C.5 is shown in Fig. C.6. The amount of current copied is again controllable via a scale factor on the current mirrors, Vscale. However, Vscale does not

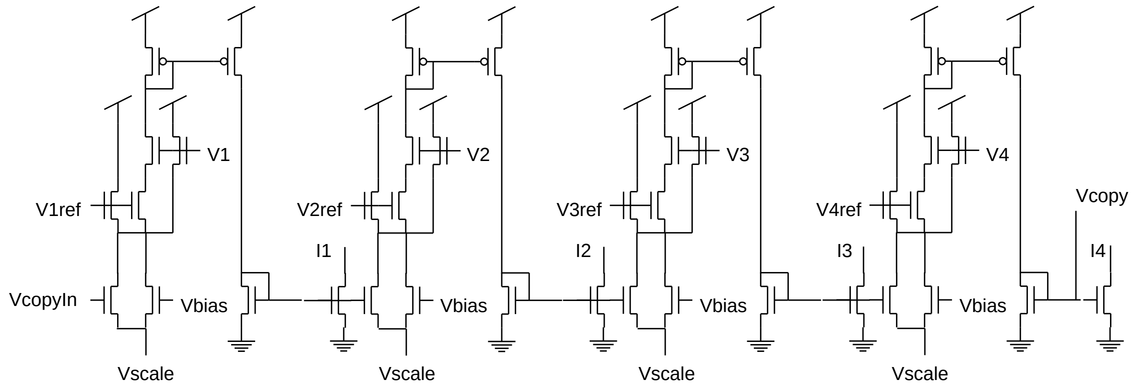

Figure C.5: A variant of the 4-dimensional extension of Delbrück's bump circuit, which allows mixtures of multiplying and adding the basis functions in each dimension. □

affect the amount of current injected by the Vbias transistors. This allows us to smoothly vary the circuit response between multiplying the functions together, and adding them.

Consider the following two cases. If Vscale is raised significantly above GND, then the various bump circuits are largely isolated; i.e., little current is copied. This means that I1, I2, I3, and I4 are very similar to individual bump currents. Adding them together produces an output current which is similar to 4 1-D bumps. If Vscale is small, or even below ground, then the copied current becomes significant. Finally, if only the first Vbias is nonzero and the rest of the Vbias values are near zero, then most of the current is copied from stage to stage. This produces a current, I4, that is very similar to the result of the circuit in Fig. C.4.

So, by manipulation of the various parameters in the circuit shown in Fig. C.6, we can produce responses between that of multiplying and adding the basis functions in the various dimensions. This has the effect of allowing a partial projection of the N-dimensional basis function onto a lower dimensional space.

Figure C.7 shows a simulation of a 4-dimensional implementation of the N-d bump circuit. The four curves represent the response as one of the 4 input parameters is varied at a time. Results from [Delbrück 91] indicate that the bump circuit simulation is very similar to what can be expected from the chip.

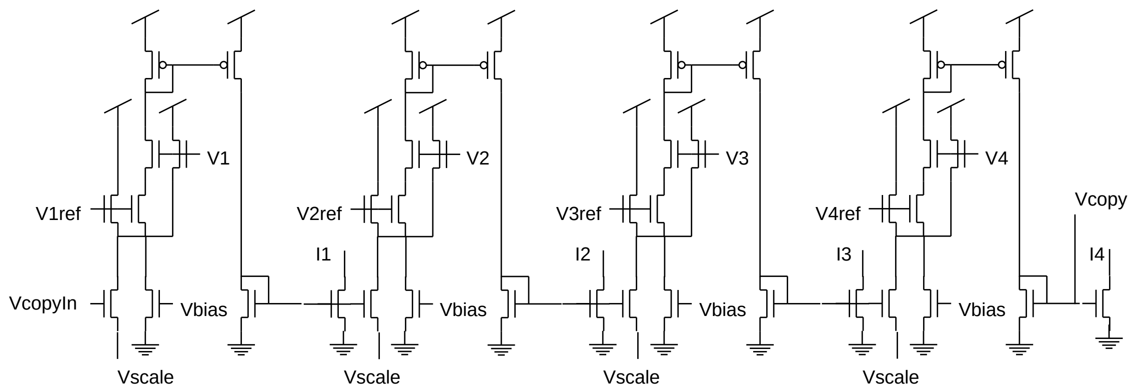

Figure C.6: Another variant of the 4-dimensional extension of Delbrück's bump circuit, which allows independent scaling of the basis in each dimension. □

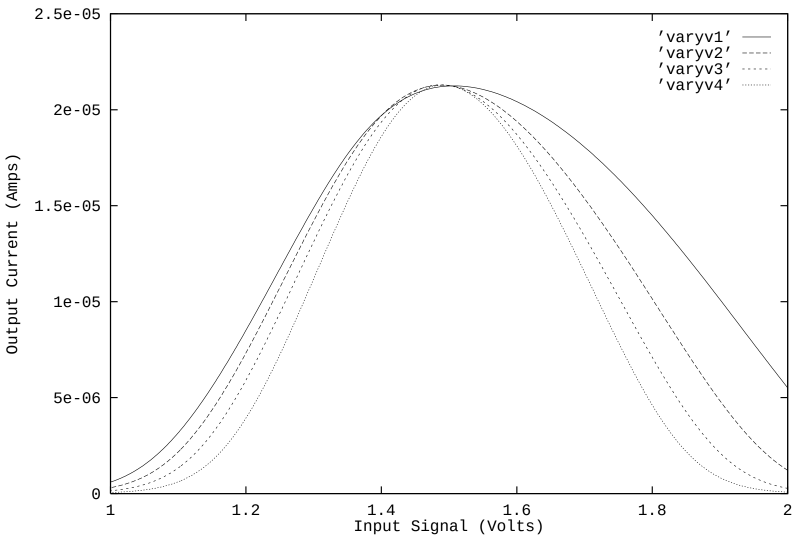

Figure C.7: Response of N-d bump circuit as input parameters are varied (analog circuit simulation). The curves *varyv1* through *varyv4* show the effects of varying inputs $V1$ through $V4$, respectively, one at a time. The reference values V_{ref1} through V_{ref4} (see figure 3) are held at 1.5 Volts. □

Appendix D

Derivation of Rotation Constraint Equations

The expression $A : A$ can be written:

(D.1) $$A : A = \sum_{j=1}^{3} \sum_{k=1}^{3} A_{jk} A_{jk} \quad .$$

In Ch. 6, we defined the function $f()$, which we reproduce here:

(D.2) $$f(M) = (MM^T - I) : (MM^T - I) \quad .$$

So, we can rewrite Eq. D.2 as:

(D.3)
$$f(\underline{\underline{M}}) = \sum_{j=1}^{3}\sum_{k=1}^{3}((\sum_{i=1}^{3} M_{ij}M_{ik}) - \delta_{jk})((\sum_{\ell=1}^{3} M_{\ell j}M_{\ell k}) - \delta_{jk}) \quad .$$

where δ_{ij} indicates the identity matrix ($\delta_{ij} = 1$ when $i = j$ and 0 otherwise).

In order to use Eq. D.3 to enforce a constraint, we would like to pose it in a form which allows us to do some sort of optimization. More specifically, in order to perform a gradient descent operation, we require the gradient. So, we compute the gradient, using Einstein Summation Notation (ESN):

(D.4)
$$\begin{aligned} \nabla f &= \frac{\partial f}{\partial M_{pq}} \\ &= 2(M_{ij}M_{ik} - \delta_{jk})(\delta_{lp}\delta_{jq}M_{lk} + M_{lj}\delta_{lp}\delta_{qk}) \\ &= 4(M_{iq}M_{ik} - \delta_{qk})M_{pk} \quad . \end{aligned}$$

Equation D.4: □

We wish to use Eq. D.4 to perform gradient descent to minimize the function $f()$, as follows:

(D.5)
$$M'(t) = -\epsilon\ \nabla f(M(t))$$

where epsilon is a parameter which determines the speed of the descent.

We define η as the gradient of $f()$:

(D.6)
$$\eta_{pq} = 4(M_{iq}M_{ik} - \delta_{qk})M_{pk} \quad .$$

We can also simplify $M_{iq}M_{ik}$ by introducing $\underline{B_1}$, $\underline{B_2}$, and $\underline{B_3}$ as basis vectors of the matrix $\underline{\underline{M}}$, and D_{ij} as the dot product of $\underline{B_i}$ and $\underline{B_j}$:

(D.7)
$$\begin{aligned} \eta_{pq} &= 4(B_q \cdot B_k - \delta_{qk})M_{pk} \\ &= 4(D_{qk} - \delta_{qk})M_{pk} \quad . \end{aligned}$$

Equation D.7: □

Since the dot products are symmetric, there are only 6 unique D_{qk} terms: the 3 diagonal terms, D_{11}, D_{22}, and D_{33}, and the three unique cross terms, D_{12} (or D_{21}), D_{23} (or D_{32}), and D_{13} (or D_{31}).

So, the following set of equations describe a discrete form of the gradient descent process:

$$M_{pq}^{new} = M_{pq}^{old} - \epsilon\ \eta_{pq} \tag{D.8}$$

and we can absorb the 4 from Eq. D.7 into ϵ, since ϵ is an arbitrary constant.

We have the following set of 9 equations for the components of the gradient:

$$\begin{aligned}
\eta_{11} &= (D_{11}-1)M_{11} + D_{12}M_{21} + D_{13}M_{31} \\
\eta_{12} &= D_{21}M_{11} + (D_{22}-1)M_{21} + D_{23}M_{31} \\
\eta_{13} &= D_{31}M_{11} + D_{32}M_{21} + (D_{33}-1)M_{31} \\
\eta_{21} &= (D_{11}-1)M_{12} + D_{12}M_{22} + D_{13}M_{32} \\
\eta_{22} &= D_{21}M_{12} + (D_{22}-1)M_{22} + D_{23}M_{32} \\
\eta_{23} &= D_{31}M_{12} + D_{32}M_{22} + (D_{33}-1)M_{32} \\
\eta_{31} &= (D_{11}-1)M_{13} + D_{12}M_{23} + D_{13}M_{33} \\
\eta_{32} &= D_{21}M_{13} + (D_{22}-1)M_{23} + D_{23}M_{33} \\
\eta_{33} &= D_{31}M_{13} + D_{32}M_{23} + (D_{33}-1)M_{33} \quad .
\end{aligned} \tag{D.9}$$

Equation D.9: □

We can define $\underline{B_1} = \underline{X}$, $\underline{B_2} = \underline{Y}$, and $\underline{B_3} = \underline{Z}$, so we can now write the discrete time step gradient descent optimization as:

$$\begin{aligned}
\underline{X}^{new} &= \underline{X}^{old} - \epsilon\ \eta_{p1} \\
\underline{Y}^{new} &= \underline{Y}^{old} - \epsilon\ \eta_{p2} \\
\underline{Z}^{new} &= \underline{Z}^{old} - \epsilon\ \eta_{p3} \quad .
\end{aligned} \tag{D.10}$$

Equation D.10: □

and, we can now write η in terms of $\underline{X}$, $\underline{Y}$, and $\underline{Z}$:

$$
\begin{aligned}
\eta_{11} &= (D_{11}-1)X_1 - D_{12}Y_1 - D_{13}Z_1 \\
\eta_{12} &= D_{21}X_1 - (D_{22}-1)Y_1 - D_{23}Z_1 \\
\eta_{13} &= D_{31}X_1 - D_{32}Y_1 - (D_{33}-1)Z_1 \\
\eta_{21} &= (D_{11}-1)X_2 - D_{12}Y_2 - D_{13}Z_2 \\
\eta_{22} &= D_{21}X_2 - (D_{22}-1)Y_2 - D_{23}Z_2 \\
\eta_{23} &= D_{31}X_2 - D_{32}Y_2 - (D_{33}-1)Z_2 \\
\eta_{31} &= (D_{11}-1)X_3 - D_{12}Y_3 - D_{13}Z_3 \\
\eta_{32} &= D_{21}X_3 - (D_{22}-1)Y_3 - D_{23}Z_3 \\
\eta_{33} &= D_{31}X_3 - D_{32}Y_3 - (D_{33}-1)Z_3 \quad .
\end{aligned}
\tag{D.11}
$$

Equation D.11: □

Glossary

* Indicates neologisms or uncommon interpretations of common terms.

accuracy Accuracy can be defined as "the degree of conformity to some recognized standard value" and, as such, is a quantification of how closely a measurement agrees with our expectations. A commonsense definition for accuracy is "getting what you want." Accuracy is based on some concept of what result you should get.

atlas An atlas is a set of coordinate charts, which, taken together with their associated partition of unity, are used to represent a manifold.[1]

cochlea The cochlea is an organ of hearing which converts acoustic mechanical energy into neural signals. It serves to separate sound information at different frequencies. [Lyon 88] described an analog VLSI circuit that behaves as an "electronic cochlea," separating information of differing frequencies in an input electronic signal.

compensated* A circuit is said to be *compensatable* if the circuit has been design to include parameters that can be set to improve the performance of the circuit. The circuit is said to be *compensated* if the parameters have been set to adjust the circuit's performance to meet some goal, perhaps the cancellation of input offsets.

[1] Atlas, partition of unity, coordinate chart, and differential geometry are defined in an intuitive fashion. For more rigorous definitions, please see [Thorpe 79].

coordinate chart A coordinate chart is a parameterization from a set of input coordinates to an output N-surface. See [Thorpe 79].

differential geometry A field of mathematics which studies N-dimensional surfaces and their properties. See [Thorpe 79].

direct method A direct method is one which guarantees solution of the problem, by solving the problem explicitly, rather than implicitly. A direct method implies that there is a high probability that satisfying the goal will produce a solution to the problem.

explicit model An explicit model is an object or a concept that is described denotationally to a fine level of detail. Any feature of the model that is important is stated clearly and directly. Any feature that is not stated is by definition not an important part of the model.

goal-based (or goal-oriented) As applied to a strategy or a methodology, goal-based indicates that the strategy is formulated with the intent to pursue some goal or purpose, with the expectation of the eventual satisfaction of that goal.

input offset An input offset is a particular type of imperfection that may exist in an analog circuit. An input offset is present when the output of the circuit for a given set of inputs is the same as the expected output for a set of inputs that are perturbed from the provided set.

model-based An optimization or learning scheme is model-based if the procedure utilized to proceed with the optimization or learning uses knowledge about the model being optimized.

partition of unity A partition of unity defines a set of real-valued blending functions used to combine the output from several coordinate charts. The blending functions add to 1, or unity. See [Thorpe 79].

precision Precision can be defined as "the degree of agreement of repeated measurements of a quantity" and, as such, is a quantification of how much useful information is present in a measurement. A commonsense definition for precision is "knowing what you've got, to some degree of exactness." Precision is based on a determination of how much useful information you have.

sample variance The sample variance of a sample $\{x_1, \cdots, x_n\}$ whose mean is $\bar{x}$, is defined to be $\sum_{i=1}^{n}(x_i - \bar{x})^2/n$, which is a maximum likelihood estimator if the distribution is normal [James 76].

standard deviation For a random variable, the standard deviation is defined to be the positive square root of the variance [James 76].

tacit model A tacit model is a description of an object or a concept that does not explicitly state important details, but leaves those details as implications. An example: A car. Those parts of the car that I have not described are tacitly assumed.

teleological An item that is teleological has characteristics that relate to a goal or purpose.

transconductance amplifier A transconductance amplifier produces a transconductance which relates its current output to its voltage inputs [Mead 89].

transconductance multiplier A transconductance multiplier produces a transconductance which relates its current output to the product of its two differential inputs (four voltages) [Mead 89].

References

[Aarts 90] Aarts, Emile, and Jan Korst, "Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing," John Wiley and Sons, New York, 1990.

[Abu-Mostafa 88] Abu-Mostafa, Yaser S., editor, "Complexity in Information Theory," Springer-Verlag, New York, 1988.

[Alefeld 83] Alefeld, G., and J. Herzberger, "Introduction to Interval Computations," Academic Press, New York, 1983.

[Allen 87] Allen, Phillip E., and Douglas R. Holberg, "CMOS Analog Circuit Design," Holt, Rinehart and Winston, Inc., Fort Worth, TX, 1987.

[Alspector 93] Alspector, J., R. Meir, B. Yuhas, and A. Jayakumar, "A Parallel Gradient Descent Method for Learning in Analog VLSI Neural Networks," in *Advances in Neural Information Processing Systems*, Vol. 5, Morgan Kaufman, San Mateo, CA, 1993.

[Alspector 91] Alspector, J., J. W. Gannett, S. Haber, M. B. Parker, and R. Chu, "A VLSI–Efficient Technique for Generating Multiple Uncorrelated Noise Sources and its Application to Stochastic Neural Networks," IEEE Transactions on Circuits and Systems, Vol.38, no.1, pp.109–123, January, 1991.

[Alspector 88] Alspector, J., B. Gupta, and R. B. Allen, "Performance of a Stochastic Learning Microchip," in *Advances in Neural Information Processing Systems, vol. I,* Denver Colorado, Nov. 1988. D. S. Touretzky, ed., Morgan Kauffman Publishers, 1989, pp. 748–760.

[Anderson 90] Anderson, Janeen D. W., and C. A. Mead, "MOS Device for Long-Term Learning," U.S. Patent #4,953,928.

[Anderson, Kerns 92] Anderson, Brooke P., and Douglas Kerns, "Using Noise Injection and Correlation in Analog Hardware to Estimate Gradients," submitted to IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications.

[Anderson 92] Anderson, Brooke P., "Low-pass Filters as Expectation Operators for Multiplicative Noise," submitted to IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications.

[Anderson 92b] Anderson, Brooke P., "Various Algorithms for Optimization and Learning in Adaptive Systems," Ph.D. Thesis, California Institute of Technology, October, 1992.

[Babanezhad 85] Babanezhad, Joseph N., and Gabor C. Temes, "A 20-V Four-Quadrant CMOS Analog Multiplier," IEEE Journal of Solid-State Circuits, Vol. SC-20, No. 6, pp. 1158–1168, December 1985.

[Barkans 90] Barkans, Anthony C., "High Speed High Quality Antialiased Vector Generation," Computer Graphics, Vol. 24, No. 4, August, 1990, pp. 319–326.

[Barzel 92] Barzel, Ronen, "Physically-Based Modeling for Computer Graphics: A Structured Approach," Academic Press, Cambridge, MA, 1992.

[Blum 89] Blum, Lenore, Mike Shub, and Steve Smale, "On a Theory of Computation and Complexity over the Real Numbers: NP-Completeness, Recursive Functions and Universal Machines," Bulletin of the American Mathematical Society, Volume 21, Number 1, July 1989.

[Bult 86] Bult, Klaas, and Hans Wallinga, "A CMOS Four-Quadrant Analog Multiplier," IEEE Journal of Solid-State Circuits, Vol. SC-21, No. 3, pp. 430–435, June 1986.

[Cauwenberghs 93] Cauwenberghs, Gert, "A Fast Stochastic Error-Descent Algorithm for Supervised Learning and Optimization," in *Advances in Neural Information Processing Systems*, Vol. 5, Morgan Kaufman, San Mateo, CA, 1993.

[Chandrasekhar 60] Chandrasekhar, S., "Radiative Transfer," Dover Publications, New York, 1960.

[Chua 84] Chua, Leon O., and Gui-Nian Lin, "Nonlinear Programming Without Computation," IEEE Transactions on Circuits and Systems, Vol. CAS-31, No. 2, February 1984, pp. 182–188.

[Clark 82] Clark, James, "The Geometry Engine: A VLSI Geometry System for Graphics," Computer Graphics, Vol. 16, No. 3, July, 1982, pp. 127–133.

[Delbrück 91] Delbrück, Tobias, "'Bump' Circuits for Computing Similarity and Dissimilarity of Analog Voltages," Proceedings of International Joint Conference on Neural Networks, July 8-12, 1991, Seattle Washington, pp. I-475–479. (Extended version as Caltech Computation and Neural Systems Memo Number 10.)

[Dembo 90] Dembo, A., and T. Kailath, "Model-Free Distributed Learning," IEEE Transactions on Neural Networks, Vol. 1, No. 1, pp. 58–70, 1990.

[Denyer 81] Denyer, Peter B., John Mavor, "MOST Transconductance Multipliers for Array Applications," IEEE Proceedings, Volume 128, Pt. I, Number 3, pp. 81–86, June 1981.

[Denyer 83] Denyer, Peter B., Colin F. N. Cowan, John Mavor, Christopher B. Clayton, and John L. Pennock, "A Monolithic Adaptive Filter," IEEE Proceedings, Volume 128, Pt. I, Number 3, pp. 81–86, June 1981.

[DeWeerth 91] DeWeerth, Steven P., "Analog VLSI Circuits for Sensorimotor Feedback," Ph.D. Thesis, California Institute of Technology, May, 1991.

[Feynman 82] Feynman, Richard, "Simulating Physics with Computers," International Journal of Theoretical Physics, Vol. 21, Nos. 6,7, 1982.

[Fleischer 90] Fleischer, Kurt, John Platt, and Alan Barr, "An Approach to Solving the Parameter Setting Problem," IEEE/ACM 23rd Intl Conf on System Sciences, January 1990.

[Flower 93] Flower, B., and M. Jabri, "Summed Weight Neuron Perturbation: An $\mathcal{O}(n)$ Improvement over Weight Perturbation," in *Advances in Neural Information Processing Systems*, Vol. 5, Morgan Kaufman, San Mateo, CA, 1993.

[Fuchs 89] Fuchs, Henry, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel, "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," Computer Graphics, Vol. 23, No. 3, July, 1989, pp. 79–88.

[Gallager 68] Gallager, R., "Information Theory and Reliable Communication," John Wiley, New York, 1968.

[Gilbert 68] Gilbert, B., "A Precise Four-Quadrant Multiplier with Sub-nanosecond Response," *IEEE Journal of Solid-State Circuits,* SC-3:365, 1968.

[Gill 81] Gill, P. E., W. Murray, and M. H. Wright, "Practical Optimization," Academic Press, 1981.

[Goodman 68] Goodman, Joseph W., "Introduction to Fourier Optics," McGraw-Hill, New York, 1968.

[Gray 84] Gray, Paul R., and Robert G. Meyer, "Analysis and Design of Analog Integrated Circuits," John Wiley and Sons, Inc., New York, 1984.

[Gregorian 86] Gregorian, Roubik, and Gabor C. Temes, "Analog MOS Integrated Circuits for Signal Processing," John Wiley and Sons, New York, 1986.

[Greengard 88] Greengard, Leslie, "The Rapid Evaluation of Potential Fields in Particle Systems," M.I.T. Press, Cambridge, MA, 1988.

[Harris 91] Harris, John, "Analog Models of Early Vision," Ph.D. Thesis, California Institute of Technology, October, 1991.

[Jabri 91] Jabri, M., S. Pickard, P. Leong, Z. Chi, and B. Flower, "Architectures and Implementations of Right Ventricular Apex Signal Classifiers for Pacemakers," IEEE Neural Information Processing Systems 1991 (NIPS 91), Morgan Kaufman, San Diego, 1991.

[James 76] James, Glenn, et al., "Mathematics Dictionary," Van Nostrand Reinhold Company, New York, 1976.

[Jayant 84] Jayant, N. S., and Peter Noll, "Digital Coding of Waveforms," Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.

[Kerns 92a] Kerns, Douglas, "A Compact Noise Source for VLSI Applications," submitted to IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications.

[Kerns 92b] Kerns, Douglas, "Experiments in Very Large Scale Analog Computation," Ph.D. Thesis, California Institute of Technology, September, 1992.

[Kirk 91] Kirk, David B., Kurt Fleischer, Lloyd Watts, and Alan Barr, "Constrained Optimization Applied to the Parameter Setting Problem for Analog Circuits," Neural Information Processing Systems 4, Morgan Kaufman, Palo Alto, CA, 1992.

[Kirk 92] Kirk, David B., Douglas Kerns, Kurt Fleischer, and Alan Barr, "An Analog VLSI Implementation of Gradient Descent," Neural Information Processing Systems 5, Morgan Kaufman, Palo Alto, CA, 1993.

[Korn 56] Korn, Granino A., and Theresa M. Korn, "Electronic Analog computers," McGraw-Hill Book Company, Inc., New York, 1956.

[Lazzaro 89] Lazzaro, John, Sylvie Ryckebusch, Misha Mahowald, and Carver A. Mead, "Winner-take-all Networks of $O(n)$ Complexity," Neural Information Processing Systems 1, Morgan Kaufman, Palo Alto, CA, 1989.

[Leighton 83] Leighton, Frank Thomson, "Complexity Issues in VLSI," MIT Press, Cambridge, MA, 1983.

[Lyon 88] Lyon, R. A., and C. A. Mead, "An Analog Electronic Cochlea," IEEE Trans. Acous. Speech, and Signal Proc., Volume 36, Number 7, July, 1988, pp. 1119–1134.

[Maher 89] Maher, Mary Ann, "A Charge-Controlled Model for MOS Transistors," Ph.D. Thesis, California Institute of Technology, May, 1989.

[Mahowald 91] Mahowald, Misha, and Rodney Douglas, "A Silicon Neuron," Nature, Vol. 354, No. 26, December 1991, pp. 515–518.

[Mahowald 92] Mahowald, Misha, "VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function," Ph.D. Thesis, California Institute of Technology, Caltech-CS-TR-92-15, May, 1992.

[Mead 89] Mead, C. A., "Analog VLSI and Neural Systems," Addison-Wesley, 1989.

[Mead 90] Mead, C. A., Allen, Timothy P., "Subthreshold CMOS Amplifier with Offset Adaptation," U.S. Patent #4,935,702.

[Mead 91] Mead, C. A., Allen, Timothy P., "CMOS Amplifier with Offset Adaptation," U.S. Patent #5,068,622.

[Millman 79] Millman, Jacob, "Microelectronics: Digital and Analog Circuits and Systems," McGraw-Hill Book Company, New York, 1979.

[NAG] NAG Fortran Library, Numerical Algorithms Group, 1400 Opus Place, Suite 200, Downers Grove, IL 60515

[Naka 91] Naka, Ken-ichi, Hiroko M. Sakai, "The Messages in Optic Nerve Fibers and their Interpretation," Brain Research Reviews, 16 (1991) pp. 135–149, Elsevier Science Publishers.

[O'Dell 88] O'Dell, T. H., "Electronic Circuit Design: Art and Practice," Cambridge Universiy Press, Cambridge, England, 1988.

[Platt 89] Platt, J. C., "Constrained Optimization for Neural Networks and Computer Graphics," Ph.D. Thesis, California Institute of Technology, Caltech-CS-TR-89-07, June, 1989.

[Press 86] Press, W., Flannery, B., Teukolsky, S., Vetterling, W., "Numerical Recipes: the Art of Scientific Computing," Cambridge University Press, Cambridge, 1986.

[Pyne 56] Pyne, Insley B., "Linear Programming on an Electronic Analogue Computer," Trans AIEE, Part I, 75 (1956) pp. 139–143.

[Rhoden 89] Rhoden, Desi, and Chris Wilcox, "Hardware Acceleration for Window Systems," Computer Graphics, Vol. 23, No. 3, July, 1989, pp. 61–67.

[Rubenstein 81] Rubenstein, Reuven Y., *Simulation and the Monte Carlo Method,* John Wiley and Sons, New York, 1981.

[Rumelhart 86] Rumelhart, D. E., and J. L. McClelland, Eds., "Parallel Distributed Processing: Explorations in the Microstructure of Cognition,", Vol. 1, M.I.T. Press, Cambridge, MA, 1986.

[Salam 88] Salam, Fathi M. A., and Mark L. Levi, "Dynamical Systems Approaches to Nonlinear Problems in Systems and Circuits," Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

[Scott 60] Scott, Norman R., "Analog and Digital Computer Technology," Mcgraw-Hill Book Company, New York, 1960.

[Siegel 81] Siegel, Robert, and John R. Howell, "Thermal Radiation Heat Transfer," Hemisphere Publishing, New York, 1981.

[Sivilotti 90] Sivilotti, Massimo, "Wiring Considerations in Analog VLSI Systems, with Application to Field-Programmable Networks," Ph.D. Thesis, California Institute of Technology, October, 1990.

[Snyder 92] Snyder, John, "Interval Analysis for Computer Graphics," Computer Graphics, Vol. 26, No. 2, pp. 121–130.

[Soclof 85] Soclof, Sidney, "Analog Integrated Circuits," Prentice-Hall, Inc., Englewood Cliffs, NJ, 1985.

[Soroka 54] Soroka, Walter W., "Analog Methods in Computation and Simulation," McGraw-Hill Book Company, New York, 1954.

[Thomsen 91] Thomsen, Axel, and Martin A. Brooke, "A Floating-Gate MOSFET with Tunneling Injector Fabricated Using a Standard Double-Polysilicon CMOS Process," IEEE Electron Device Letters, Vol. 12, No. 3, pp. 111–113, March, 1991.

[Thorpe 79] Thorpe, J. A., **Elementary Topics in Differential Geometry,** Springer-Verlag, New York, 1979.

[Tomovic 62] Tomovic, Rajko, and Walter J. Karplus, "High-Speed Analog Computers," Dover Publications, New York, 1962.

[Toumazou 90] Toumazou, C., F. J. Lidgey, and D. G. Haigh, "Analogue IC Design: the Current Mode Approach," Peter Peregrinus Ltd., London, United Kingdom, 1990.

[Traub 88] Traub, J. F., G. W. Wasilkowski, H. Wozniakowski, "Information-based Complexity," Academic Press, New York, 1988.

[Tsividis 86] Tsividis, Yannis, Mihai Banu, and John Khoury, "Continuous-Time MOSFET-C Filters in VLSI," IEEE Transactions on Circuits and Systems, Vol. CAS-33, No. 2, pp. 125–140, February 1986.

[Ullman 92] Ullman, David G., "The Mechanical Design Process," McGraw-Hill, New York, 1992.

[Ullman 84] Ullman, Jeffrey D., "Computational Aspects of VLSI," Computer Science Press, Rockville, MD, 1984.

[Umminger 89] Umminger, Christopher B., and Steven P. DeWeerth, "Implementing Gradient Following in Analog VLSI," Advanced Research in VLSI, MIT Press, Boston, 1989, pp. 195–208.

[Vergis 86] Vergis, Anastasios, Kenneth Steiglitz, and Bradley Dickinson, "The Complexity of Analog Computation," Mathematics and Computers in Simulation 28 (1986) 91-113, North-Holland.

[Voorhies 88] Voorhies, Douglas, D. Kirk, and O. Lathrop, "Virtual Graphics," Computer Graphics, Vol. 22, No. 4, August, 1988, pp. 247–253.