USC-SIPI REPORT #189

Automated Synthesis of Analog MOS VLSI Circuit Modules

by

David Jan-Chia Chen

December 1991

Signal and Image Processing Institute
UNIVERSITY OF SOUTHERN CALIFORNIA
Department of Electrical Engineering-Systems
Electrical Engineering Building
University Park/MC-2564
Los Angeles, CA 90089 U.S.A.

Acknowledgements

I would like to express my deepest thanks to my research advisor Professor Bing Sheu for his guidance and support throughout the course of my Ph.D. research work. I wish to extend my sincere appreciation to Professor Clarence Crowell and Professor Dominic Cheung for serving on my dissertation committee. I would also like to thank them along with Professors Melvin Breuer and Alan McCurdy for serving on my qualifying examination committee. Professor Dominic Cheung is with the Department of Comparative Literature.

I am very grateful to Professor Hans Kuehl, Chairman of EE-Electrophysics Department; Professor Melvin Breuer, Chairman of EE-Systems Department; Professor Leonard Silverman, Dean of Engineering School; and Ms. Ramona Gordon, Senior Administrative Assistant, for providing me the opportunity to pursue my Ph.D. studies under such a great research environment at USC. Encouragement from Dr. George Lewicki, the former Director of the MOSIS Service, Dr. Cesar Pina, the present Director of the MOSIS Service, and other members of the USC/Information Sciences Institute at Marina del Rey, CA is also highly appreciated.

The help and friendship of many fellow graduate students in the VLSI Signal Processing Laboratory have contributed to this work. I thank Antony Fung for his contribution to the early development of the prototype Op-Amp design expert system. Antony Fung moved on to pursue his medical degree at U. C. Davis. I also thank Dr. Ji-Chien Lee for his assistance in writing some software modules for the primitive cell generators. Discussions on artificial neural network design with Dr. Bang Lee were very useful. Many thanks are due to Dr. Wen-Jay Hsu and Sudhir Gowda for tutoring me on the public-domain electronic-CAD tools. I am also thankful to Ming Hsu and Je-Hurn Shieh for teaching me how to use TROFF and other desktop publishing tools. I appreciate

Oscal T.-C. Chen, Chia-Fen Chang, Joongho Choi, Han Yang, Wai-Chi Fang, and Dr. Chung-Ping Wan for their various discussions and assistance.

Contemporary work of fellow researchers from both industry and academia have greatly stimulated my research interests in analog IC design automation. In particular, the active contributions to this field from researchers at Carneige-Mellon University, University of California, Berkeley, and the Centre Suisse d'Electronique et de Microtechnique S.A. (CSEM), Switzerland are greatly acknowledged.

I also wish to thank many people in Sharp Digital Information Products, Inc. in Irvine, CA; especially Mr. T. Kojima, Vice President, and Mr. Gus Kinoshita, Director of Communications Engineering, for their encouragement and support. Finally, I would like to thank my wife, Ming, our children, Tony and Tiffany, and our parents for their understanding as well as the sacrifice put up by them through all these years.

Partial results of this research was presented at IEEE Workshop on Analog Circuit Engineering at Princeton, NJ in April 1989; at IEEE International Conference on Computer Design at Cambridge, MA in October 1988, 1989, 1990; and at IEEE Custom Integrated Circuit Conference at San Diego, CA in May 1991.

This research has been studied since Fall 1986 while I keep a full-time position at Sharp Digital Information Products, Inc. The Interactive Television Program at the School of Engineering has been extremely valuable for most of my course work. The research was partially supported by DARPA under Contract No. MDA972-90-C-0037 and by Faculty Research and Innovation Fund No. 22-1502-9759 to Professor Bing Sheu from University of Southern California and Contributions from TRW, Inc. and NKK Corp.

Table of Contents

List of Tables

List of Figures

Abstract

Recent advances in VLSI technologies have allowed the integration of information processing subsystems on one chip, including both analog and digital parts. In addition to the conventional telecommunications and computer networking applications, new applications of mixed analog-digital VLSI are being unveiled in the fields of image processing and neural-based flexible information processing. The need for computer-aided design tools to reduce the dominant analog circuit design time and cost for such processor chips has become imminent. This thesis presents advanced methods for automatic synthesis of analog MOS VLSI circuit modules. A design system that consists of a knowledge-based synthesis tool and a constraint-based layout generator has been developed to automate the design of key analog MOS circuit modules. An expert system has been developed to assist in an iterative analog design process. This integrated-circuit design expert system, which interfaces with a circuit simulator, is capable of optimizing circuit topologies as well as circuit element geometries to better satisfy a given set of performance specifications. The constraint-based layout generator uses analog circuit recognition and critical-net analysis techniques to identify crucial portions of an analog circuit and systematically derive proper layout constraints for analog circuit performance optimization. Constraint-driven floorplanning and routing techniques are developed to generate high-quality full-custom analog circuit layouts which incorporate the layout constraints. This layout tool is capable of quickly generating a correct and high-performance custom layout with a selectable aspect ratio for a wide variety of analog circuit modules. Extensions to higher-level subsystem layout synthesis using hierarchical floorplanning and module generation techniques are also described. Automatic layout generation techniques for single- and multi-layer neural networks have been developed. Experimental results on several CMOS Op-Amps, a voltage comparator, and a 16-neuron Hopfield neural network are

described. These results indicate that this new analog circuit synthesis approach provides two advantages: it is quite general and yet effective, and can be used by system designers who are inexperienced in analog circuit design.

Chapter 1

Introduction

1.1 Symbols and Nomenclature

Fueled by the continued evolution of the computer-aided design (CAD) tools, the field of integrated circuit (IC) design has been growing rapidly in recent years. Unfortunately, as with any high-technology field, a large amount of technical jargon has evolved, which must be mastered by anyone wishing to be conversant with those working in the area. For this reason, the current terminology used in the IC field has been adopted in this thesis. Because of the connection with artificial intelligence, a number of words are used that tend to personify software tools and properties. However, for readers who are not familiar with the subject, a glossary of the key technical terms is also provided in Appendix E. A list of the symbols is included in Appendix D.

1.2 Motivation

Recently, very large scale integration (VLSI) technologies have advanced rapidly with the continual shrinking of the minimum geometrical feature size towards the deep submicron level. Meanwhile, analog MOS circuit techniques [1-9] have also dramatically improved over the past decade to a state making possible realization of high-performance analog designs using standard 5V CMOS technologies. As a consequence, an electronic system containing complex analog and mixed analog-digital circuit functions can now be integrated on

a single VLSI chip [10-16] to achieve the full economic gain of microelectronic technologies. Typical examples are found on today's advanced signal processing systems for telecommunications [10-12], hard-disks [13], video and image processing [14-16], robotics, automative electronics, and intelligent sensors, among others. In addition, exciting new applications of analog VLSI are being unveiled in the field of neural computing [17-19], which has been recognized as a major enabling technology for future information processing.

While the digital parts of such VLSI chips can nowadays be rapidly synthesized in semicustom styles such as the standard-cell approach, most analog subsystems still need to be handcrafted by analog circuit specialists due to the lack of effective CAD tools for analog ICs [20]. As a result, the design time and cost involved for the analog circuitry portion often constitutes a bottleneck to the overall design of the microelectronic chip. This holds true even when the analog circuitry comprises only a small fraction of the total chip area. Therefore, to support the growing analog IC design trends in today's system-on-a-chip environment, the development of effective analog electronic-CAD tools, especially the automatic design synthesis and layout generation tools, is of vital importance.

1.3 Analog IC Design Automation

Work in analog IC design automation has only recently received increased attention from the research community [21-45]. Although these initial research efforts have shown some encouraging results, they are just a beginning. More systematic, flexible, and effective techniques are yet to be explored.

1.3.1 Design synthesis

Analog integrated-circuit design is commonly perceived as one of the most knowledge-intensive design tasks. In order to deal with the complexity arising out of different voltage levels in analog circuit designs, CAD tools are needed that are "smarter" than currently available tools used for digital circuit designs. With an increasing interest in its applications, artificial intelligence (AI) has shown significant impact in fields such as medicine and science. Complex engineering design tasks such as analog IC designs that require large amounts of expert knowledge can also adopt the AI approach by using expert systems to assist in a design. In recent years, several studies have been reported on the expert system-based approach to analog IC designs, especially on design synthesis of operational amplifiers (Op-Amps) [21-28]. Although implemented differently, most knowledge-based analog IC design programs have taken the multiple-fixed-architecture approach. In this approach, design decisions are made with a limited number of fixed circuit choices in the database. Since the operation range over which analog circuits are designed is wide and indefinite, fixed design choices impose considerable limits upon the flexibility and thus the applicability of existing analog circuit design tools.

1.3.2 Automatic layout generation

The layout automation for analog ICs is much more difficult to deal with than their digital counterparts. To effectively address the diversity of analog circuit designs, a technology-independent custom layout style based on flexible module generation techniques is necessary. Special care needs to be taken to minimize electrical performance degradation due to device mismatching,

parasitics, and noise coupling in the analog circuit layout. The physical layout also needs to accommodate large variations in device sizes while maintaining a compact area. Furthermore, it is desirable that the layout can be quickly adjusted for different physical aspect ratios. Finally, the layout tool must be user-friendly for system designers who typically have limited analog circuit knowledge.

Research in analog layout automation is also receiving increased attention from the IC community [32-45]. Most earlier attempts typically relied on a semicustom style and resulted in very limited success [32-34]. The new custom layout synthesis approaches [35-41], however, use one of two methods. One uses a fixed floorplan based on predefined slicing trees. This provides rather limited flexibility. The other method relies on a general simulated annealing program adapted from the digital design world. This often results in inadequate analog performance. Most layout tools are designed to handle Op-Amps only. Many of the above analog-specific performance considerations have not been addressed systematically. Moreover, some tools require analog circuit layout constraints to be manually specified by the users as part of the input and thus are not sufficiently automated to perform a satisfactory netlist-to-layout synthesis.

1.4 Scope and Organization of the Thesis

This thesis describes new automatic analog module synthesis and layout generation techniques for analog and mixed analog-digital VLSI design. A system that automates the design of common analog MOS integrated circuit modules has been developed. The system consists of two parts: a knowledge-

based design synthesizer [29-31], which transforms module specifications into sized schematics, and a constraint-based layout generator [42-45] which transforms sized schematics into mask geometry, as illustrated in Fig. 1.1.

Figure 1.2 depicts a hierarchical view of typical mixed-signal VLSI circuit configuration. Two classes of mixed-signal VLSI circuits are defined: one is for implementing the conventional signal processing systems; the other is for realizing neural networks, a growing field. The analog section in the conventional signal processing VLSI is typically composed of several common analog subsystems including A/D converters, D/A converters, switched-capacitor filters (SCFs), continuous-time filters (CTFs), and other nonfilter subsystems such as phase-lock loops (PLLs), automatic gain controllers (AGCs), waveform generators, etc. On the other hand, the analog circuit structure of a neural VLSI is less complex which normally contains neuron blocks and synapse matrices. Note that at the module level, all the circuit modules can be shared by both classes of VLSI subsystems. These circuit modules include Op-Amps, comparators, voltage-controlled oscillators (VCOs), multipliers, and other active and passive circuit modules. These modules are considered as the most important building blocks for analog subsystems and their circuit performance are the most critical to the system performance. The modules can be further decomposed into simpler primitive cells and then down to the individual device level.

The scope of this thesis is mainly to explore automatic design synthesis and layout generation techniques for analog IC modules. Special emphasis has been placed on the physical design aspect of analog circuit module generation. In

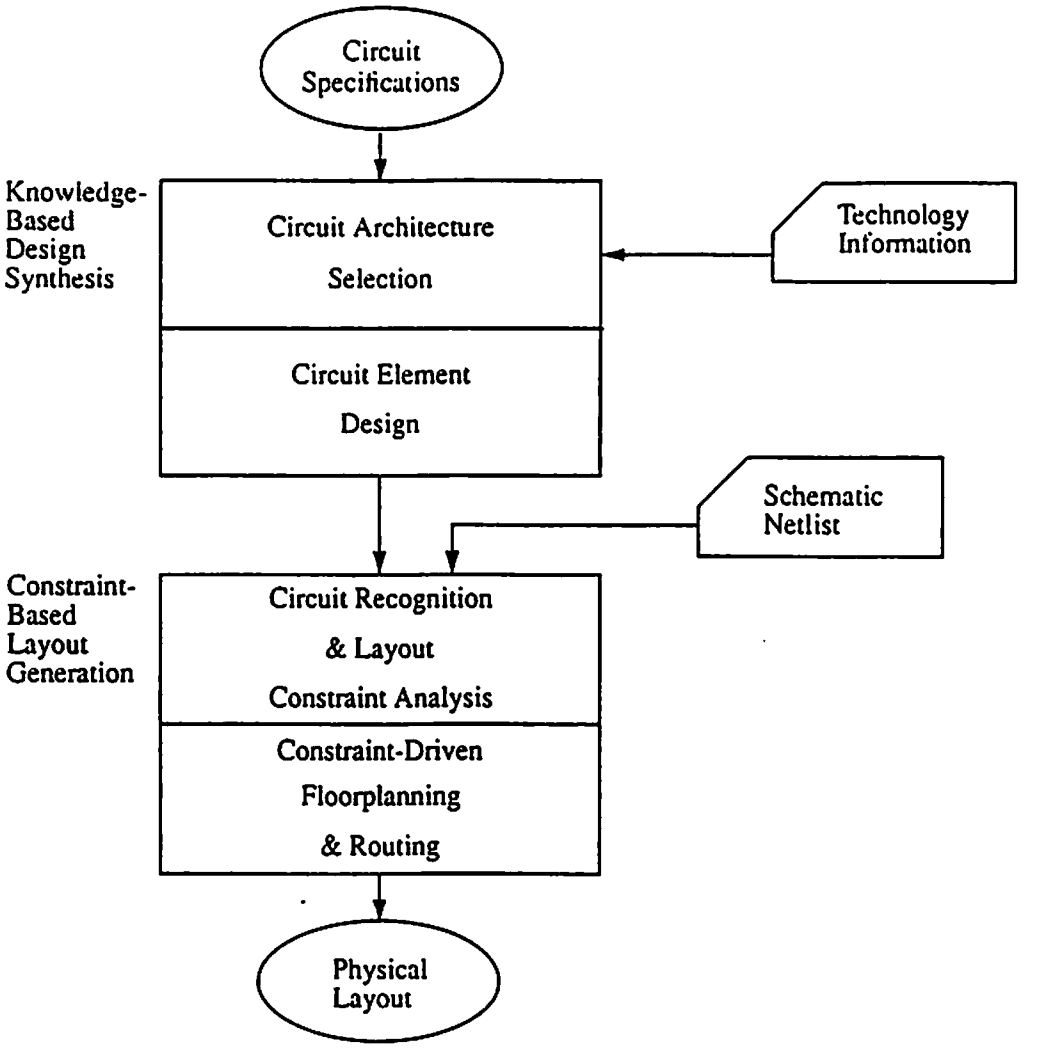


Fig. 1.1 An analog IC synthesis system composed of a knowledge-based design synthesis tool and a constraint-based layout generator.

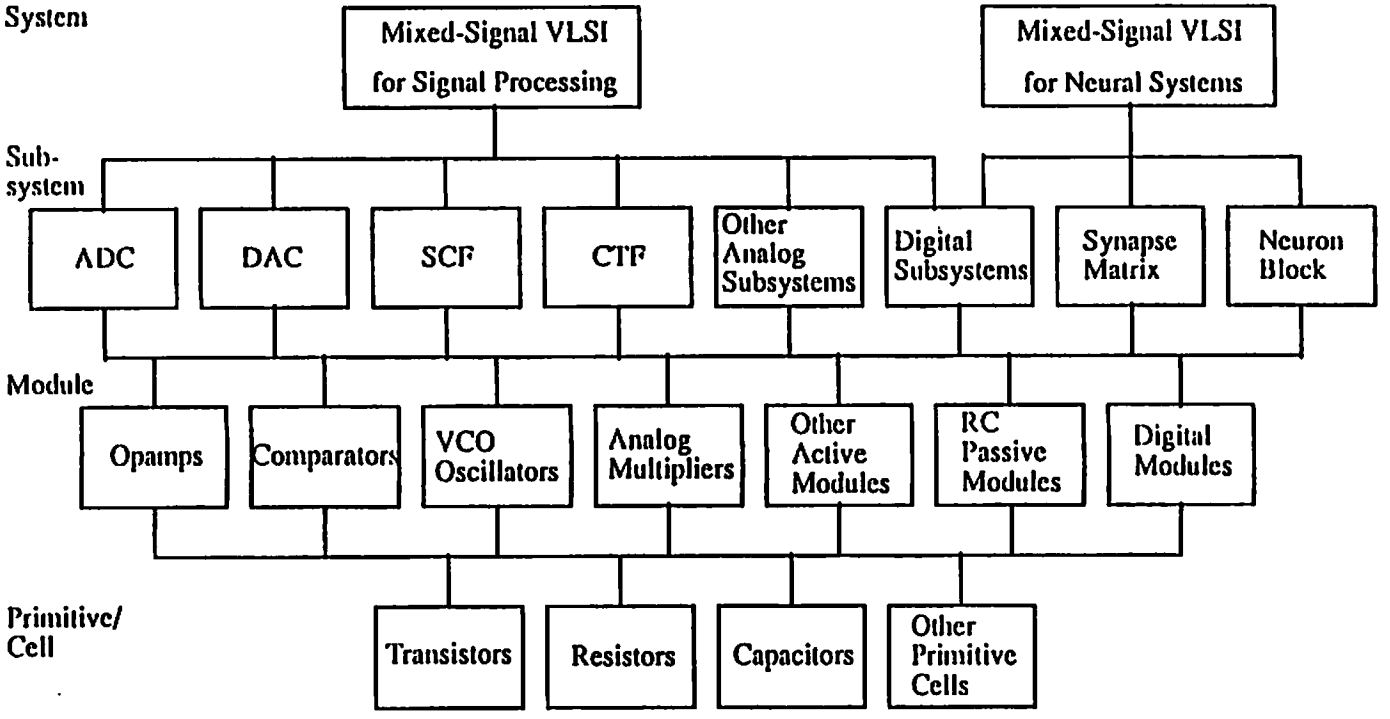


Fig. 1.2 A hierarchical view of mixed-signal VLSI circuit configuration.

addition, some of the subsystem layout issues and strategies for the mixed-signal VLSI circuits are addressed.

In Chapter 2, a self-configuration technique for expert system-based analog IC design [29-31] is presented. A rule-based expert system has been developed for assisting in a CMOS operational amplifier design. The implementation of the expert system is described and experimental results are presented.

In Chapter 3, the analog circuit layout considerations are examined and automatic analog IC layout systems are reviewed. An overview of the constraint-based method to systematically address the challenges for analog IC layout automation [42-45] is presented. Given a SPICE-like netlist of an analog circuit module, this method is capable of quickly generating a high performance and reasonably compact custom circuit layout to desired aspect ratios.

Chapter 4 describes automatic analog circuit recognition and critical net analysis techniques used for the layout constraint analysis. The rule-based analog circuit recognition technique is used to recognize all the critically matched device pairs from the input netlist and identify critical analog circuit nodes that have key significance for analog circuit layout concerns. Several recognition examples of common analog MOS circuit modules are given. The critical net analysis technique is described which takes the recognized circuit information and internally generates proper layout constraints for a given analog circuit module.

In Chapter 5, the constraint-driven analog circuit floorplanning and routing techniques are described. An efficient floorplanning technique based on a zone-

sensitivity partitioning algorithm and slicing structures has been developed which can derive a slicing tree and determine the optimal component shape for the module to satisfy both the electrical and physical aspect ratio constraints. Generated layout results of several circuit modules are presented.

In Chapter 6, layout synthesis strategies for analog VLSI subsystems are discussed. A general layout methodology for conventional analog MOS signal processing subsystems using hierarchical floorplanning and module generation techniques is described. Special layout considerations and strategies for automatic layout of analog artificial neural networks are also discussed. Efficient layout generation techniques for single- and multi-layer neural networks are developed. The generated layout results of an experimental neural network is presented.

Chapter 7 presents concluding remarks and suggests future extensions of this research. In Appendix A, a real-world example of mixed analog-digital VLSI implementation for telecommunication applications [12] is presented. For readers who are not familiar with the neural circuits, Appendix B covers some basic concepts in VLSI implementation of artificial neural networks. Finally, a partial listing of the prototype program for custom analog MOS circuit module generation is included in Appendix C.

Chapter 2

Knowledge-Based Analog IC Design Synthesis

One major reason that CAD tools available for analog IC designs are yet to reach the level of performance of those for digital designs is that analog designs, unlike digital designs, which only need to deal with two discrete logic levels (0 and 1), have to deal with a continuous spectrum of voltage levels. This added complexity due to continuously varying voltages requires CAD tools for analog circuit designs to be more "knowledgeable" about a design task than those used for digital circuit designs.

Knowledge-based approaches to analog IC designs have attracted much attention in recent years [21-31]. Since fast progress is currently being made in analog IC design techniques, a heuristic approach seems more useful than ever. In this chapter, an expert system assisted analog design methodology is presented. With the expert system acting as a design assistant, designs are carried out iteratively. The iterative design concept is used to contrast the analog and digital design processes from the coventional design automation point of view. A prototype expert system has been developed to serve as the analog design assistant. It adopts the flexible architecture approach and is thus capable of self-modifying the circuit topology. The implementation of an expert system for CMOS operational amplifier design is described.

2.1 Analog vs. Digital IC Design Processes

The design process of an analog or a digital IC can be partitioned into three phases: architecture selection, circuit topology refinement and transistor geometry scaling. Although each part of the design process can be considered 'distinct', they are by no means isolated design steps. As a matter of fact, there are strong interactions among them. Decisions made at one design phase may affect decisions made at other design phases. For example, a circuit topology that looked promising during the topology refinement phase of the design may have to be changed due to unsatisfactory performance after the transistors are sized. Similarly, a circuit architecture may need to be replaced if desired performance cannot be achieved by altering detailed circuit topology and sizing transistor geometries.

In a digital IC design process, the major design task is to search for a circuit architecture that will satisfy the functional requirements. Detailed circuit topology refinements are carried out for speed, power and area improvements. Transistor size plays a relatively minor role in a digital design process when pre-designed circuit cells are used as building blocks for the final circuit. For a digital VLSI design, the performance of the circuit depends mainly on the inter-coupling between the architecture selection and topology refinement. Circuit performance is improved by fine-tuning the circuit topology within a pre-determined architectural frame as well as changing the architecture based on topology refinement feedback. Since the specification of a digital design is usually functional and the number of performance variables is limited, a final design solution

can often be achieved without careful sizing of transistor geometries or complex trade-off evaluations.

The design process of an analog circuit follows the similar design phases for a digital circuit. A circuit architecture is first determined based on the evaluation of the overall design specifications. Detailed circuit topology refinement and transistor sizing are then performed to attain the desired circuit performance. The interaction between topology refinement and transistor geometry sizing plays a crucial role in the analog design process. Unlike digital designs where cells of discrete transistor sizes are often used as building blocks, analog designs deal with a continuous spectrum of transistor sizes. Variations in individual transistor size may result in significant changes in circuit performance. Based on the information from transistor sizing, the circuit topology sometimes requires modifications to improve the circuit performance. In such cases, the transistors in the modified circuits need to be sized again. Evidently, the strong coupling between the two design phases is essential to a successful analog design process. Design modifications can be made not only within a fixed architectural frame but also across various architectural design planes. Figure 2.1(a) illustrates the concept of a generic analog design process. The analog design space can be simplified into design planes. Different design planes represent different circuit architectures within which circuit topology refinement and transistor sizing can be carried out. While the design planes are weakly coupled, strong interaction exists between topology refinement and transistor sizing on each design plane. With the discrete topology refinement choices, the circuit topology cannot be altered arbitrarily. Only identifiable circuit primitives such as

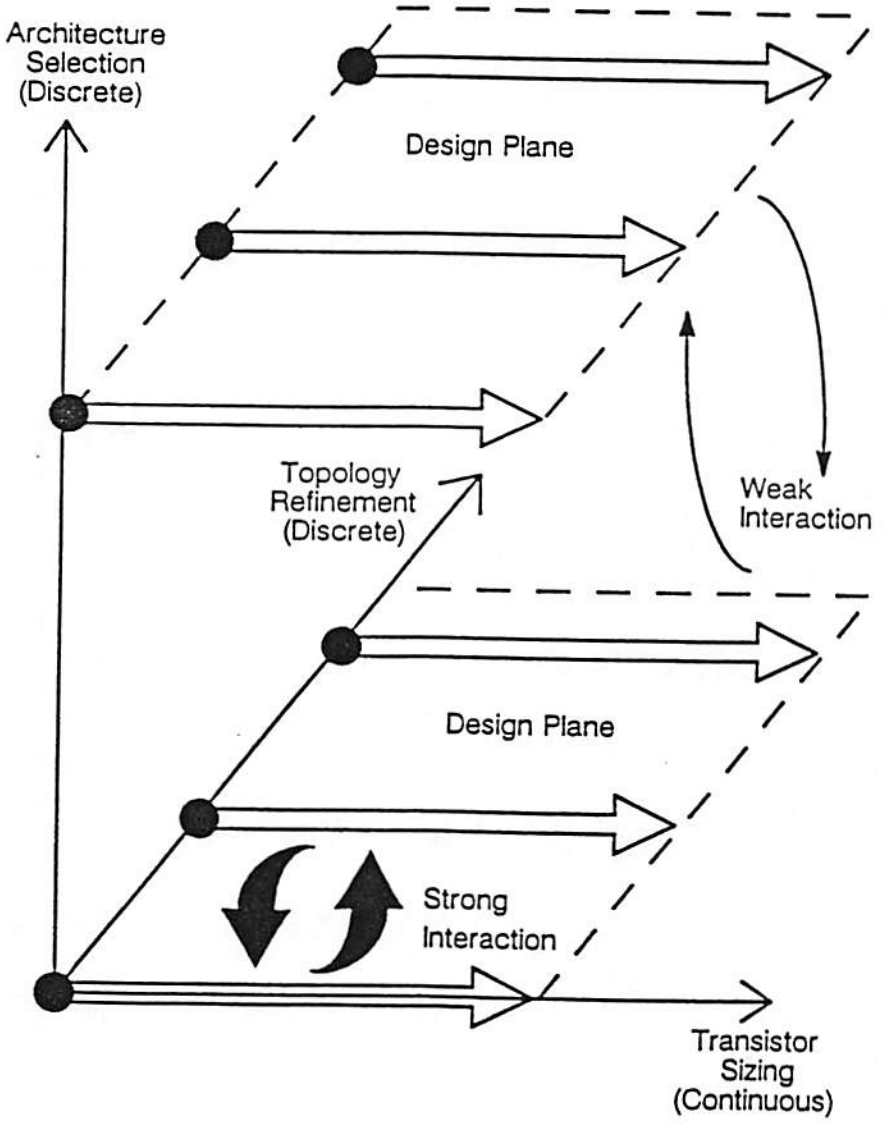


Fig. 2.1 Generic analog integrated-circuit design process.
(a) Strong interaction exists between topology refinement and transistor sizing on each design plane.

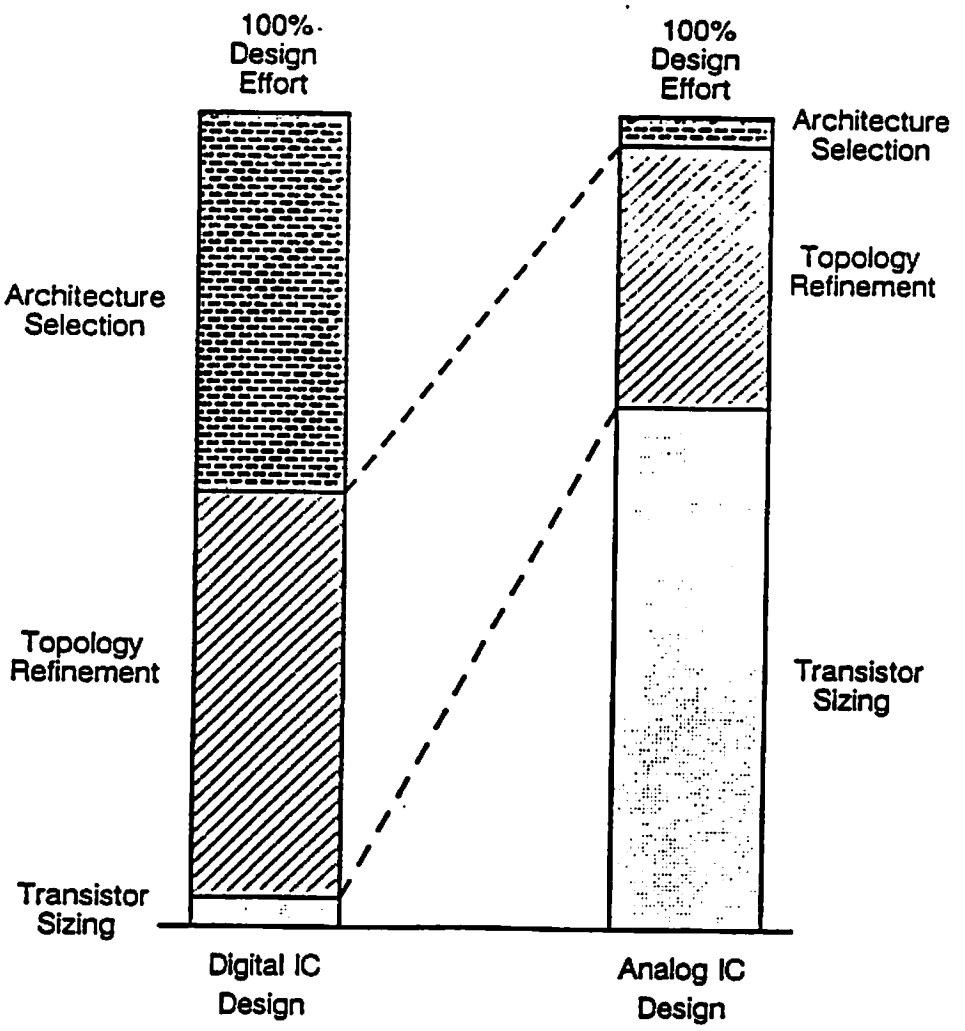


Fig. 2.1 (continued)
(b) Comparison of relative design efforts for digital and analog IC designs.

a current source and an active load can be replaced as a complete unit within a circuit module. However, transistor geometries do not have to take on discrete values. They can be sized with any value in the continuum of size-range limited only by the resolution of the layout generation facility and by the physical size specification of the circuit module. Figure 2.1(b) constrasts the relative design efforts needed for analog and digital IC design processes. For digital IC designs, most design efforts are spent on architecture selection and topology refinement. For analog IC designs, transistor sizing takes up the major portion of the design efforts. The exact weighting of each design phase is design dependent.

2.2 Iterative Analog IC Design Using Expert Systems

Due to the nonlinear nature of analog design processes, iterations are often required to fine-tune the transistor geometries and thus to converge on a design that achieves the desired performance. A design process usually starts with approximated analyses using simplified analytical design equations. An initial design solution is generated based on the result of such an analysis. As complex as analog circuit operations are, approximated analyses cannot accurately predict the actual performance of a circuit. Detailed simulations using a simulator with accurate component models are thus necessary to verify a design. The circuit simulator is able to provide complete analyses of circuit operations as well as good predictions of the actual circuit behavior. Without a circuit simulator, analog circuits cannot be designed with good accuracy.

Incorporation of a reliable circuit simulator in the analog design process is essential not only in actual design practice, but also in developing a computer-aided analog design environment. Since its introduction in 1975, the SPICE circuit simulator [46-48] has been widely accepted as a circuit simulator for analog circuit designs. Given appropriate model parameters, it has been proven to be a reliable tool in simulating actual circuit behaviors. Today, even the most experienced analog IC designers still find it necessary to confirm their designs with the SPICE simulator due to the simplified design equations used in the manual design process. As powerful as the SPICE simulator is, it is, nevertheless, a "passive" CAD tool. For analog circuit designs, it analyzes the circuit operations and evaluates the circuit performance only when it is presented with a complete circuit. It does not automatically compare the achieved results with the performance specifications. It does not suggest possible ways of improving the circuit. Put this another way, SPICE does not "actively" involve in the decision steps of the circuit design process. The designers always need to "interpret" the SPICE results before making improvements to the design. To an inexperienced analog IC designer, this can be a difficult and time consuming task. As more and more system-level designers are required to cover the circuit-level design tasks, analog CAD needs not only a circuit simulator like SPICE, but also a tool that contributes "actively" to the circuit design process. However, what analog CAD lacks is a tool to act as human experts in decision making, not a tool to eliminate the service of the reliable and proven SPICE. Thus, in a computer-aided analog design environment, tools that are used to assist in decision making

during the design process must frequently communicate and consult with SPICE for accurate analyses and for achieving desired designs.

To facilitate the analog IC designs, we have developed a new design methodology that utilizes expert systems to assist in the design process. Figure 2.2 shows the behavioral description of this new methodology. Its converging nature resembles the Newton-Raphson's algorithm [49] for solving nonlinear equations. According to the Newton-Raphson's algorithm, a solution for a non-linear equation is quadratically reached through tangential extrapolations of intermediate results. In the expert system assisted design process, the system first provides a reasonable initial design guess. Recommendations from the expert system then "extrapolate" along the tangent of the projected point on the performance curve to its intersection with the design axis. The intersection point becomes the next design input to the circuit simulator. Results from the circuit simulator project the intersection point on the design axis back onto a point which is closer to the design goal on the performance curve than the previous point. The expert system, upon receiving the simulated results from the circuit simulator, then acts again to provide design recommendations for the next iterative cycle. With an intelligently generated initial solution, a design that satisfies all performance requirements ought to emerge in an efficient manner. Notice that such a design is not always achievable. Physical limitations that exist in a given fabrication technology could prohibit a design ever meeting all the specifications. In that case, either the fabrication technology or circuit specification has to be adjusted to produce a feasible design. Alternatively, a binary-tree search algorithm can be used for extrapolations in which a solution is

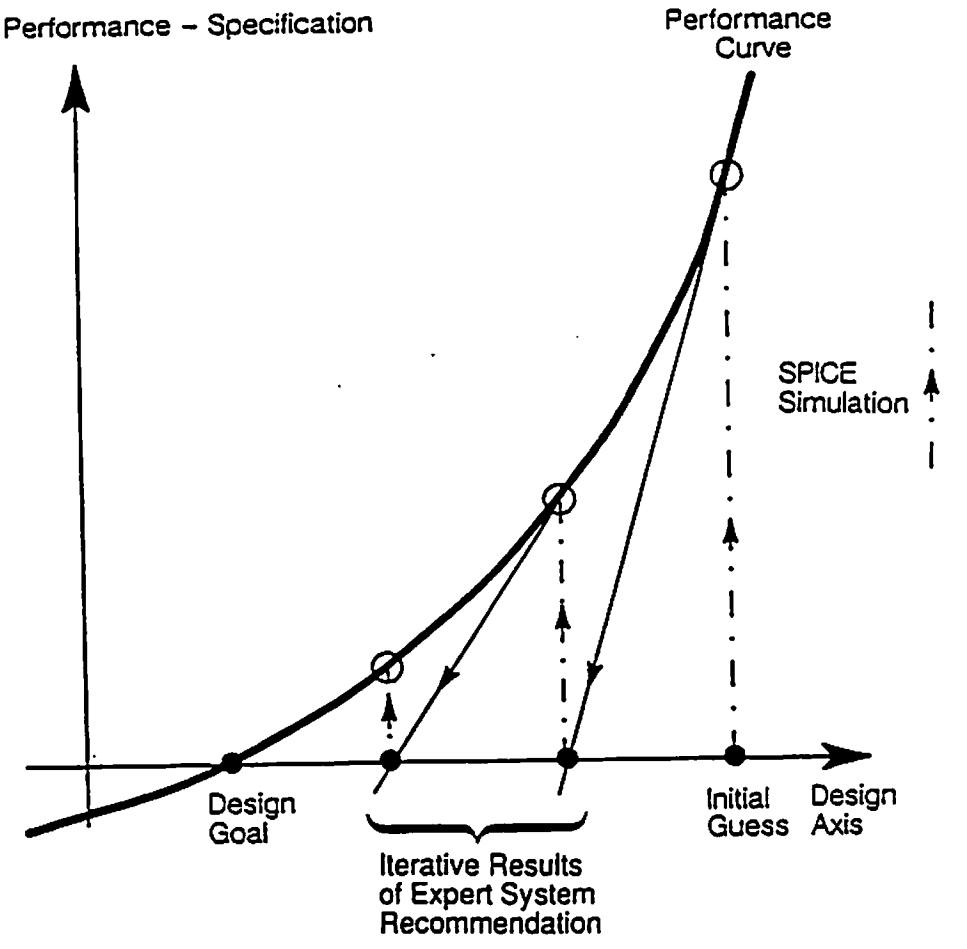


Fig. 2.2 Schematic illustration of a converging iterative design process using an expert system as an analog circuit design assistant and a circuit simulator.

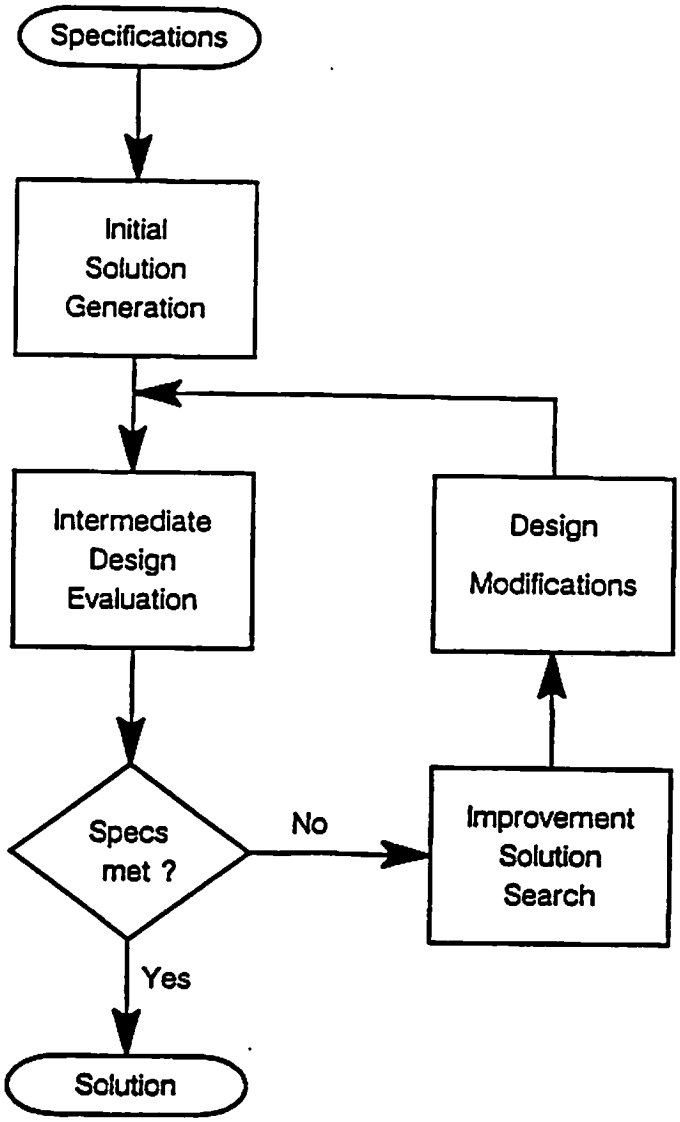


Fig. 2.3 Iterative analog IC design process model.

reached through successive approximations of intermediate results [88]. This method generally assures the convergence of a final solution.

Figure 2.3 shows the iterative analog design process model. Every design starts with a set of performance specifications. An initial design is generated based on these specifications. The achieved performance is compared with the specifications after the design goes through evaluations. If all specifications are met, the design is accepted as a solution. However, if not all specifications are satisfied, ways to improve the performance have to be identified and the design has to be modified. The modified design has to be evaluated and the circuit performance needs to be checked again. This kind of evaluation and modification cycle, shown in the figure by the feedback loop, continues until a desired solution is reached. In the case of very tight specifications, the design loop is halted and a warning is issued as the knowledge of possible improvements is exhausted.

The iterative design process can be realized through the integration of SPICE and a design expert system. While SPICE is used as an evaluation and verification tool for intermediate design solutions, the expert system acts as a design assistant providing design improvement recommendations. Unlike the one-shot approach in which SPICE is used only at the end of a design process, the iterative design approach utilizes simulation results at final as well as intermediate design steps. Consequently, high performance designs can be attained reliably.

2.3 Flexible Architecture Design by Self-Configuration

Previous approaches taken to synthesize analog circuits can be categorized into two groups: the multiple-fixed-architecture approach [24,25] and the subcircuit assembly approach [23,26]. For the multiple-fixed-architecture approach, a number of fixed circuits are stored in the design knowledge base. Based on the specifications for a new design, one of the circuits is selected at the beginning of the design process. Transistor geometries are then optimized to achieve required circuit performance. During the design process, the chosen circuit configuration cannot be changed in any way. Transistor geometries are the only design variables once a circuit is selected. As one can see, the achievable circuit performance range is limited by the number of fixed circuits stored in the design database. In order to achieve high performance designs for various analog circuit applications, a large number of circuits may be required in the database. Even then, the design process is static and the design flexibility is extremely limited.

The subcircuit assembly approach hierarchically partitions a circuit design task into several subcircuit design tasks. Subcircuit units are assembled to form the complete circuit. The aim of the assembly approach is to optimize the trade-offs among performance specifications at an early stage of a design. Although the approach is more flexible than the multiple-fixed-architecture approach due to the added design freedom at the topology level, transistor geometries are still the only design variables once a circuit topology is determined. No modification can be made to the circuit topology at a later stage of the design even if the earlier analyses gave a suboptimal circuit configuration.

For both approaches, the design process is carried out entirely by the expert system using approximate analytic equations. Circuit simulators are not used for evaluations at any intermediate stage of the design. Although the final design is verified through a circuit simulator, the achievable accuracy is limited by the equations implemented in the expert system. Since analog circuits are usually required to satisfy complicated design specifications, approximate analytic models and fixed circuit topology impose considerable constraints on the flexibility and thus the applicability of the analog tools adopting these approaches.

We have developed a flexible architecture approach to knowldege-based analog IC design synthesis [29-31]. This approach mimics real design practice in the sense that a chosen circuit topology can be altered while transistor geometries are sized. The expert system is capable of making choices among fixed circuit topologies as well as replacing portions of the circuit during a design process for performance improvements. Figure 2.4 shows the design flow of the approach. Unlike the other two approaches mentioned earlier, the flexible architecture approach optimizes circuit topology at the same time as transistor geometries. Circuit topology modification are based on SPICE simulation results. They are considered only when the desired performance cannot be achieved by sizing the transistors of the selected circuit topology.

The flexible architecture approach features the self-configuration technique which uses circuit primitives such as current sources and active load as "replacement parts" in a design. If the initial circuit configuration selected among the fixed circuits cannot satisfy the performance requirements, the solution space is searched and another solution is identified for trial toward the desired improve-

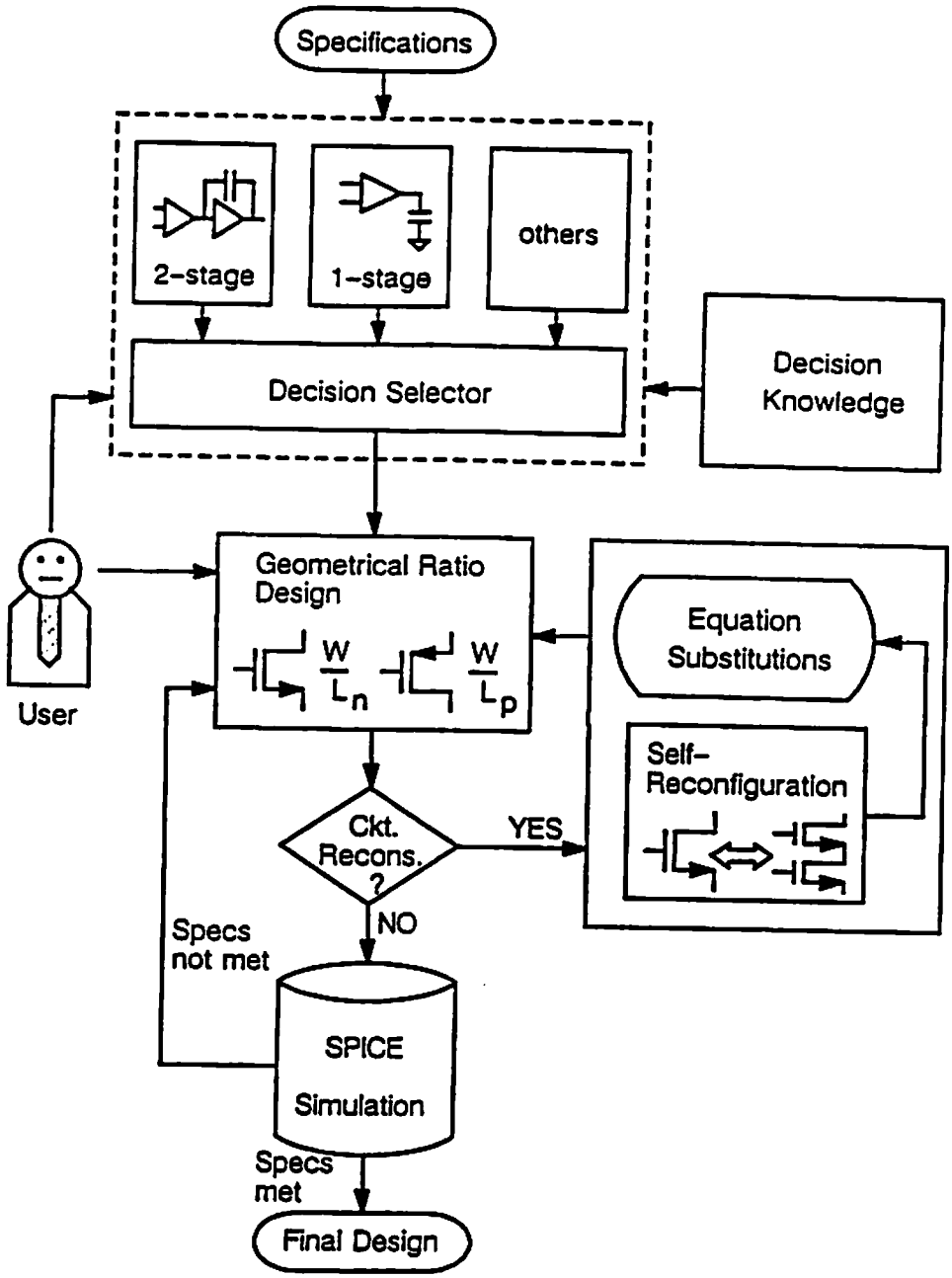


Fig. 2.4 Design flow of the flexible architecture approach for operational amplifier synthesis.

ments. If the solution requires circuit topology modification, appropriate circuit primitives are used for replacements. For example, since the gain of an operational amplifier (Op-Amp) is directly proportional to the output resistance of the input stage, one solution to increase the gain of an Op-Amp is to cascode the input stage [50]. Thus, if a circuit with a non-cascode input stage is unable to satisfy the gain requirement, cascode input pair as well as cascode active load can be used to replace the original input stage for higher voltage gain. The cascode input stage replacement solution is part of the design knowledge for amplifier gain improvement. Once the circuit is modified, subsequent analyses of the circuit would be carried out with the cascode input stage in place of the original input stage. Here, instead of requiring a full circuit topology with cascode input stage to be stored in the design database, the chosen circuit configuration is modified to yield the desired improvement. A two stage Op-Amp with simple input stage and one with a cascode input stage are shown in Figs. 2.5(a) and (b) respectively. The circuit primitive replacement scheme may be used for any functional block that can be identified in the chosen circuit. The expert system possesses the knowledge to perform replacement only if it would contribute to the circuit performance improvements desired.

The self-configuration technique is realized through equation substitutions in the design knowledge base. The equations corresponding to the replaced portion of the circuit in a design are substituted by the equations of the replacing subcircuits for subsequent design analyses. Figure 2.6 illustrates the circuit primitive replacement and equation substitution procedure of the self-configuration tech-

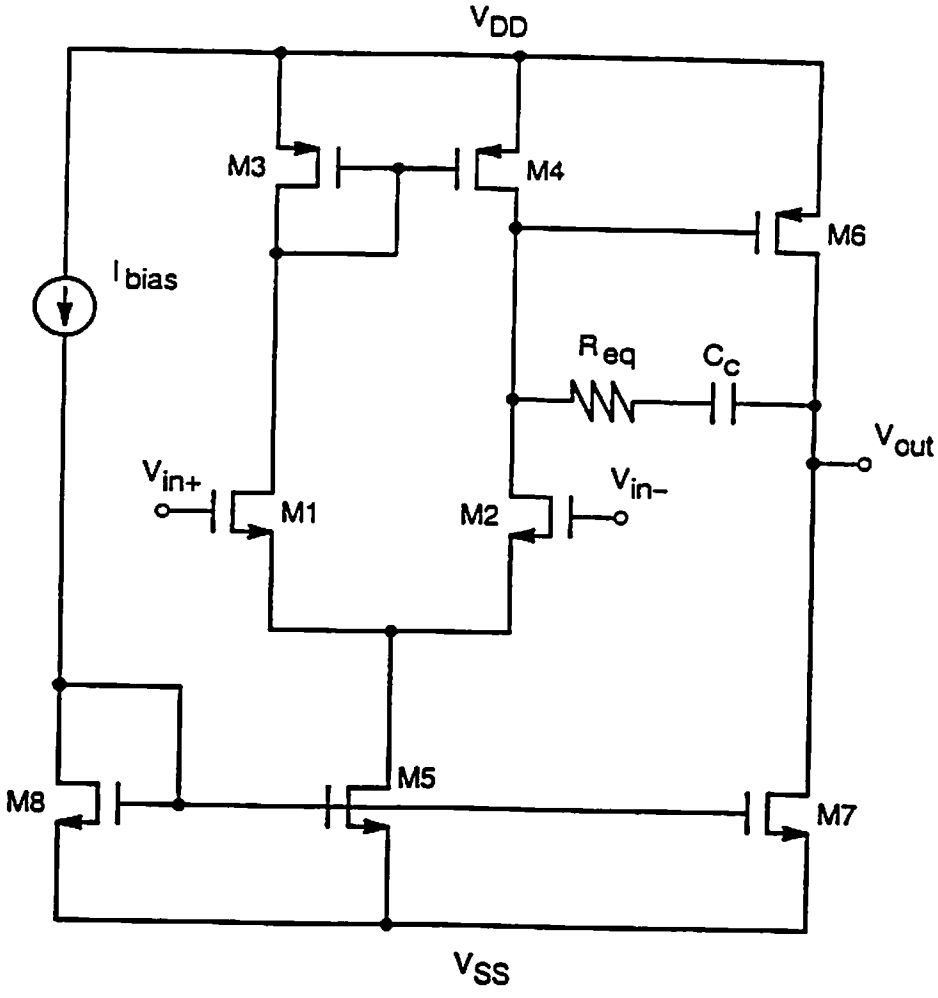


Fig. 2.5 Simplified schematic diagram of a basic two-stage CMOS operational amplifier.
(a) with simple input stage.

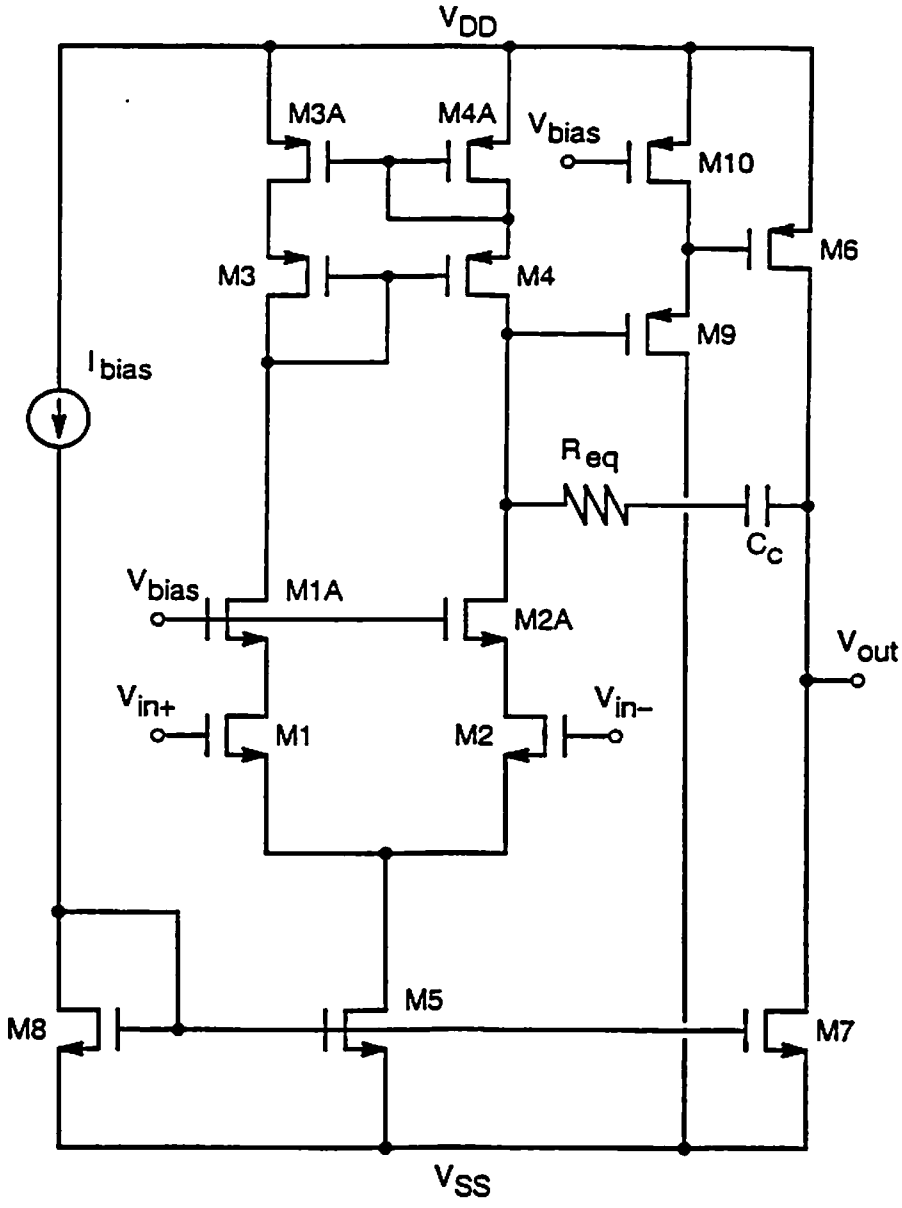


Fig. 2.5 (continued)
(b) with fully cascode input stage.

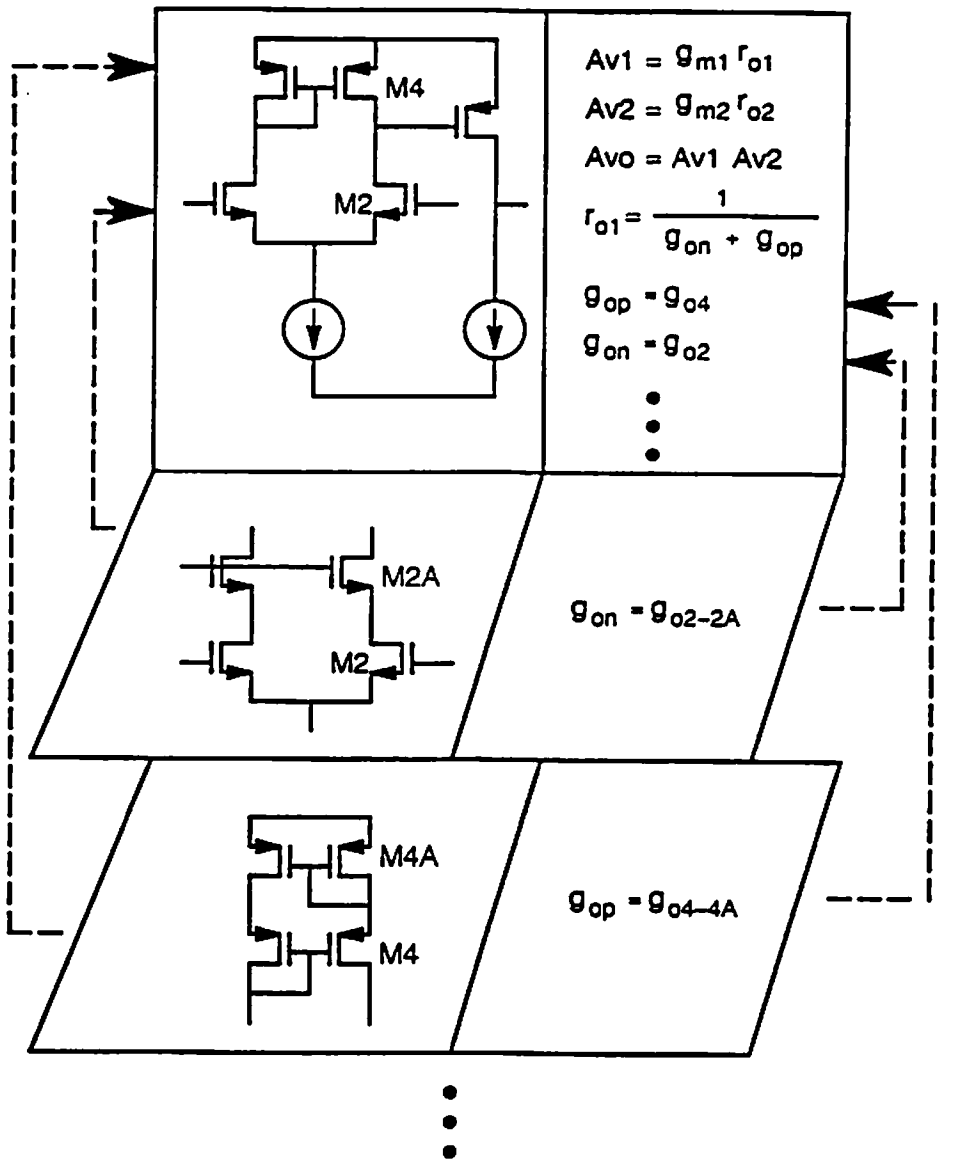


Fig. 2.6 Circuit primitive replacement and equation substitution for self-configuration.

que. When a fixed Op-Amp circuit is chosen at the beginning of a design, its corresponding design equations are used to generate the initial design solution. If the circuit was modified during the design process, i.e. when a portion of the circuit is replaced, relevant equations are updated to reflect the change. Further analyses would then be carried out with the new set of design equations.

The use of equation substitution can be explained by the following example. For the two-stage CMOS Op-Amp shown in Fig. 2.5(a), the voltage gain of the first stage can be expressed as

$$A_{v1} = g_{m1} \cdot r_{o1} \tag{2.1}$$

where g_{m1} is the transconductance of the transistor M1 and r_{o1} is the total output resistance of the input stage and can be approximated as

$$r_{o1} = \frac{1}{g_{o4} + g_{o2}}. \tag{2.2}$$

Here, g_{o4} and g_{o2} are the output conductances of the transistors M4 and M2 respectively. In order to get more gain out of the amplifier, the basic input stage can be replaced by cascode devices. Figure 2.5(b) shows the schematic diagram of an Op-Amp with a cascode input pair. In this case, the voltage gain becomes

$$A_{v1} = \frac{g_{m1}}{\dfrac{g_{o4}}{g_{m4A} \cdot r_{o4A}} + \dfrac{g_{o2}}{g_{m2A} \cdot r_{o2A}}} \tag{2.3}$$

where g_{m2A} and r_{o2A} are the transconductance and output resistance of the transistor M2A, g_{m4A} and r_{o4A} are the transconductance and output resistance of

the transistor M4A. It is evident from Eqs. (2.2) and (2.3) that the self-modifying replacement of the basic input pair by cascode devices merely require the substitutions of $\frac{g_{o2}}{g_{m2A} \cdot r_{o2A}}$ for g_{o2} and $\frac{g_{o4}}{g_{m4A} \cdot r_{o4A}}$ for g_{o4}.

2.4 An Expert System for CMOS Op-Amp Design

As one of the most important building blocks in analog IC design, the operational amplifier has found its way into various analog circuit applications. Many analog IC design issues can be addressed through the design of an operational amplifier.

A typical set of performance specifications of an Op-Amp design includes open-loop voltage gain (A_{vo}), power dissipation (P_d), input range (V_{ip}), output swing (V_{op}), offset voltage (V_{os}), common mode rejection ratio (C_{MRR}), power supply rejection ratio (P_{SRR}), unity-gain bandwidth (f_o), phase margin (ϕ_M), input-referred noise (V_{noise}) , settling time (t_{set}) and slew rate (S_r). The fact that many of these specifications are highly interrelated makes the operational amplifier design ever so challenging. Figure 2.7 shows the interrelationship among some crucial performance objectives. It is evident that improvement made on one of the performance specifications may very well result in deterioration of the other specifications during a design.

Let's use the CMOS operational amplifier example shown in Fig. 2.5(a). The open-loop voltage gain of such a basic two stage Op-Amp can be expressed as

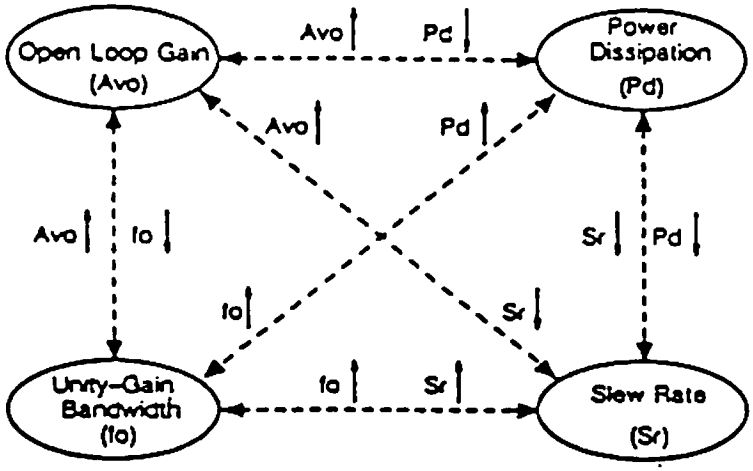


Fig. 2.7 Interrelationship among performance objectives of an operational amplifier.

$$A_{vo} = G_{m1}R_{o1}G_{m2}R_{o2} \tag{2.4}$$

where G_{m1} and R_{o1} are the transconductance and output resistance of the first stage, G_{m2} and R_{o2} are the transconductance and output resistance of the second stage. Since the G_m's are directly proportional to the square roots of the biasing currents and the R_o's are inversely proportional to the biasing currents, the voltage gain (A_{vo}), as a result, is inversely proportional to the square roots of the biasing currents. Thus, a decrease in the current level would yield a higher voltage gain. However, both the slew rate (S_r) which can be expressed as

$$S_r = \frac{I_{ss}}{C_c} \tag{2.5}$$

and the unity-gain bandwidth (f_o) which can be expressed as

$$f_o = \frac{G_{m1}}{C_c} \tag{2.6}$$

would suffer because of this current level reduction. Here, I_{ss} is the bias current through transistor M5 and C_c is the compensation capacitor. Apparently, trade-off stands as one of the major design considerations. The design of an operational amplifier, therefore, requires not only fundamental understanding of circuit operations, but also expert knowledge of optimizing performance tradeoffs.

An expert system for Circuit-level AMPlifier design (CAMP) has been developed to assist the design of operational amplifiers. It consists of a knowledge base which contains domain knowledge that is necessary for solving problems associated with a given design task and an inference engine that knows

how to manipulate the knowledge to reach a design solution. Design knowledge is represented in CAMP as facts and rules. Facts are fixed data in the knowledge base and rules use those available facts for making decisions, deriving other facts and drawing conclusions. It is not enough for an expert system to have all the required domain knowledge but not knowing when or how to effectively apply the knowledge. The inference engine carries the responsibility of making effective use of the knowledge. It decides on how and in what order the rules are to be used to infer new knowledge. With sufficient knowledge in the knowledge base, the inference engine is able to reason its way to a design solution that parallels a human expert solution.

Implemented in Turbo Prolog [51,52], the design knowledge of CAMP is represented by logic-based method where analyses are carried out through the use of predicate logic. Constants and equations are stored as rules and predicates. Rules are written in the IF-THEN format. The expression

IF A and B THEN C

shows that the goal C is proven true if both subgoals A and B were proven true. Rules can be used to express relationships among facts as well as relationships among rules. The CAMP program contains approximately 95 rules in 1700 lines of code. Predicates are used to express fixed relationship between objects. For example, the fact that object X increases object Y can be expressed through the predicate "increases" as

increases (*X* , *Y*).

In the CAMP program, knowledge stored as facts include performance specifications and constant parameters. The specification for the open-loop voltage gain of an Op-Amp (A_{vo}) is stored in the knowledge base as

$$spec\,(avo\ ,\ 5500).$$

The predicate "spec" is defined to relate the specification value for the open-loop voltage gain, in this case 5500, to the symbol avo. Similar predicates are defined for other specification values. A set of default values are coded in the expert system. However, the user can change any of them at the beginning of each design session. Constant parameters are defined in a similar way. Users are to provide the values from a set of SPICE model parameters.

Equations are used in rules for inferring conclusion or new knowledge. A rule is activated if the conditional body of the rule were satisfied. A design is carried out through inference chaining of related rules. Each design decision is based on previous decisions and available facts. For example, to establish the relationship between the unity-gain frequency (f_o) and the width-to-length ratio ($\frac{W}{L}$) of transistor M1 in Fig. 2.5(a), the following two relationships are used for inference chaining:

$$g_{m1} = 2\,\pi\,f_o\,C_c \tag{2.7}$$

and

$$(\frac{W}{L})_1 = \frac{g_{m1}{}^2}{2\,k_p{}'\,I_{ss}} \tag{2.8}$$

where g_{m1} is the transconductance of M1, C_c is the compensation capacitance,

k_p' is the transconductance coefficient and I_{ss} is the biasing current through M5. In other words, the determination of $(\frac{W}{L})_1$ to achieve the specified unity-gain frequency requires the knowledge of bias currents and transconductance which in turn requires the compensation capacitance value. Because many factors contribute to a design decision, inferences not only ease the implementation of the decision making process but also provide a more dynamic solution to incorporate gradually-formed knowledge for complex design decisions.

Various improvement solutions are implemented for each performance objective. Improvement solution space is searched whenever an initial design cannot satisfy certain performance requirements. For example, the common-mode-rejection-ratio (CMRR) of the two-stage Op-Amp shown in Fig. 2.5(a) can be expressed as

$$CMRR = 2 \frac{g_{m1} \cdot g_{m3}}{g_{o5} \cdot g_{o1}} \tag{2.9}$$

where g_{m1} and g_{m3} are the transconductances of M1 and M3 and g_{o5} and g_{o1} are the output conductances of M5 and M1 respectively. To increase the CMRR of the circuit, g_{m1} and g_{m3} can be increased or g_{o5} and g_{o1} can be decreased. Thus, possible improvement solutions include increase of M5 length which decreases g_{o5}, increase of input pair length which decreases g_{o1} and cascode current source which effectively decreases g_{o5}. The final decision depends on how much the achieved performance deviates from the desired value. Deviation of each performance objective is assigned a percentage error factor,

$$Er\_X = \frac{Achv\ X - X}{X} \tag{2.10}$$

where "Achv" stands for "achieved". A correction factor is then defined for each percentage error factor Er_X as

$$CF = \frac{1}{1 + Er\_X}. \tag{2.11}$$

The correction factor of each performance objective is used to determine an improvement solution as well as the exact adjustment that needs to be made. Solutions that do not alter the current circuit topology are given higher priorities. Circuit topology modification is performed only when there exists no good improvement alternative in transistor sizing and when the estimated performance after the modification meets the specification. Thus, in the case of the unsatisfactory CMRR, the device lengths of the current source and/or the input pair are increased before the option of cascoding the current source is considered.

Figure 2.8 shows the four-step design process of CAMP. Step-I of the design is a quick evaluation of the feasibility of given performance specification values. The evaluation is conducted by a set of approximate equations. If the specifications do not seem achievable, CAMP comments on the limitations. If they were feasible, CAMP enters step-II of the design. Note that device parameters extracted from fabrication process information must be provided for proper evaluations.

The step-II of the design addresses the issue of the dc characteristics of the circuit. In this phase, CAMP attempts to provide proper dc biasing currents and voltages for the circuit. The circuit simulator SPICE is used to evaluate and

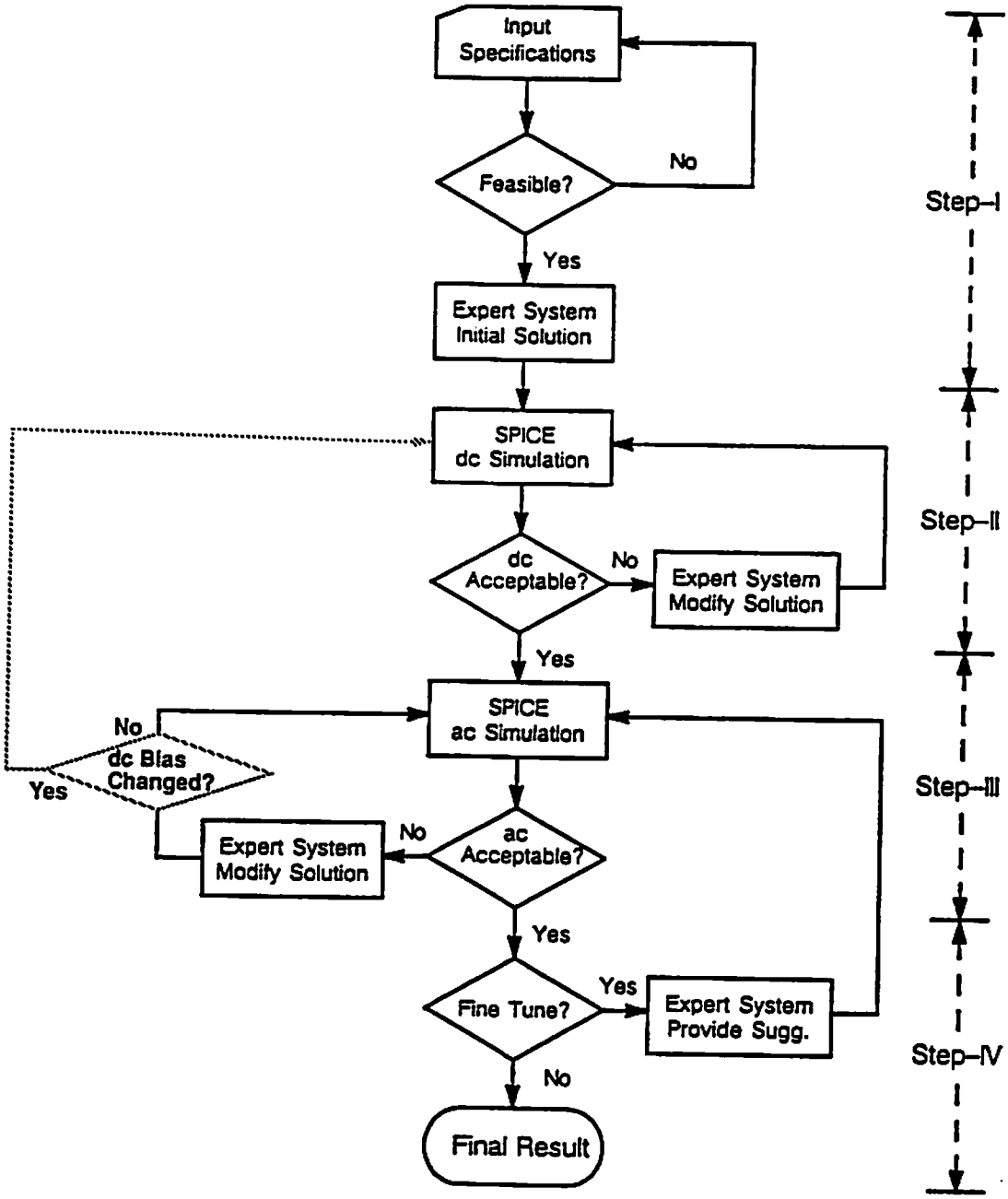


Fig. 2.8 The four-step design process of the CAMP expert system.

confirm the design generated by CAMP. The SPICE result is fed back to CAMP for necessary iterations.

Once the circuit is biased properly, step-III of the design process addresses the issue of the ac characteristics of the circuit. Here, SPICE is used to evaluate the small signal performances of the CAMP generated design. Similar to the situation in step-II, several iterations are necessary to achieve desired performances. Note that if the dc biasing currents are changed significantly during the step-III design process, then the dc characteristics of the modified circuit needs to be re-checked in each iteration. This can be done by adding a feedback path (shown in a dashed line in Fig. 2.8) from step-III to step-II of the design process [88].

The step-IV is a very important step in the Op-Amp design process. CAMP assists the circuit designer in fine-tuning the circuit performance. At this step of the design, the designer has the option to select any of the specifications for further improvement. CAMP would provide information for possible design modifications and trade-offs in related performance objectives.

2.5 Experimental Results

As an intelligent assistant to Op-Amp design, CAMP shows satisfactory performance. Table 2.1 shows the specifications and the achieved performance of an experimental Op-Amp design. The last column of the table shows the circuit performance after the self-configuration to improve the common-mode-rejection-ratio. The input stage current source of the circuit was modified to a cascode

Table 2.1 Op-Amp specification and achieved performance.

Performance Parameters	Specification Values	Achieved Performance (simple input stage current source)	Achieved Performance (cascode input stage current source)
Open loop voltage gain (dB)	60	74.3	75
Unity-gain bandwidth (MHz)	1.0	1.6	1.6
Phase margin (Deg.)	45	50	50
Power dissipation (mW)	2.0	0.98	0.94
Common-mode rejection ratio (dB)	80	74	92
Slew rate (V / us)	2.0 (3 V step)	3.1 rising 3.8 falling	2.9 rising 2.8 falling
Common-mode input range (V)	+ 4.0 - 4.0	+ 4.1 - 4.3	+ 4.1 - 4.3
Power supply rejection ratio PSRR + (dB)	40 @ DC	42.6 @ 1 Hz 42.5 @ 5 KHz	43 @ 1 Hz 43 @ 5 KHz
Power supply rejection ratio PSRR - (dB)	40 @ 5 KHz	42.6 @ 1 Hz 42.6 @ 5 KHz	43 @ 1 Hz 43 @ 5 KHz

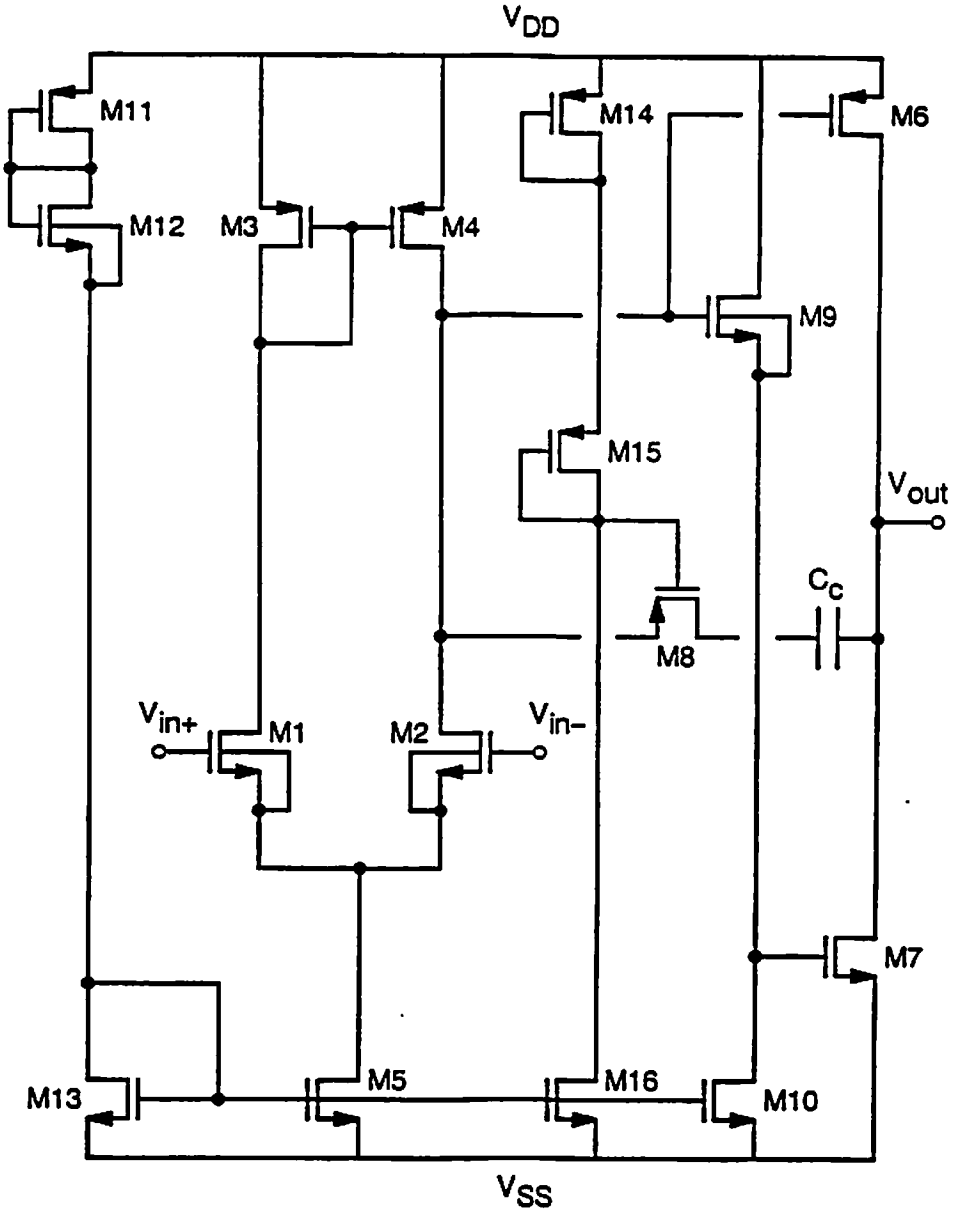


Fig. 2.9 Schematic diagram of a two-stage operational amplifier. (a) with simple biasing current source.

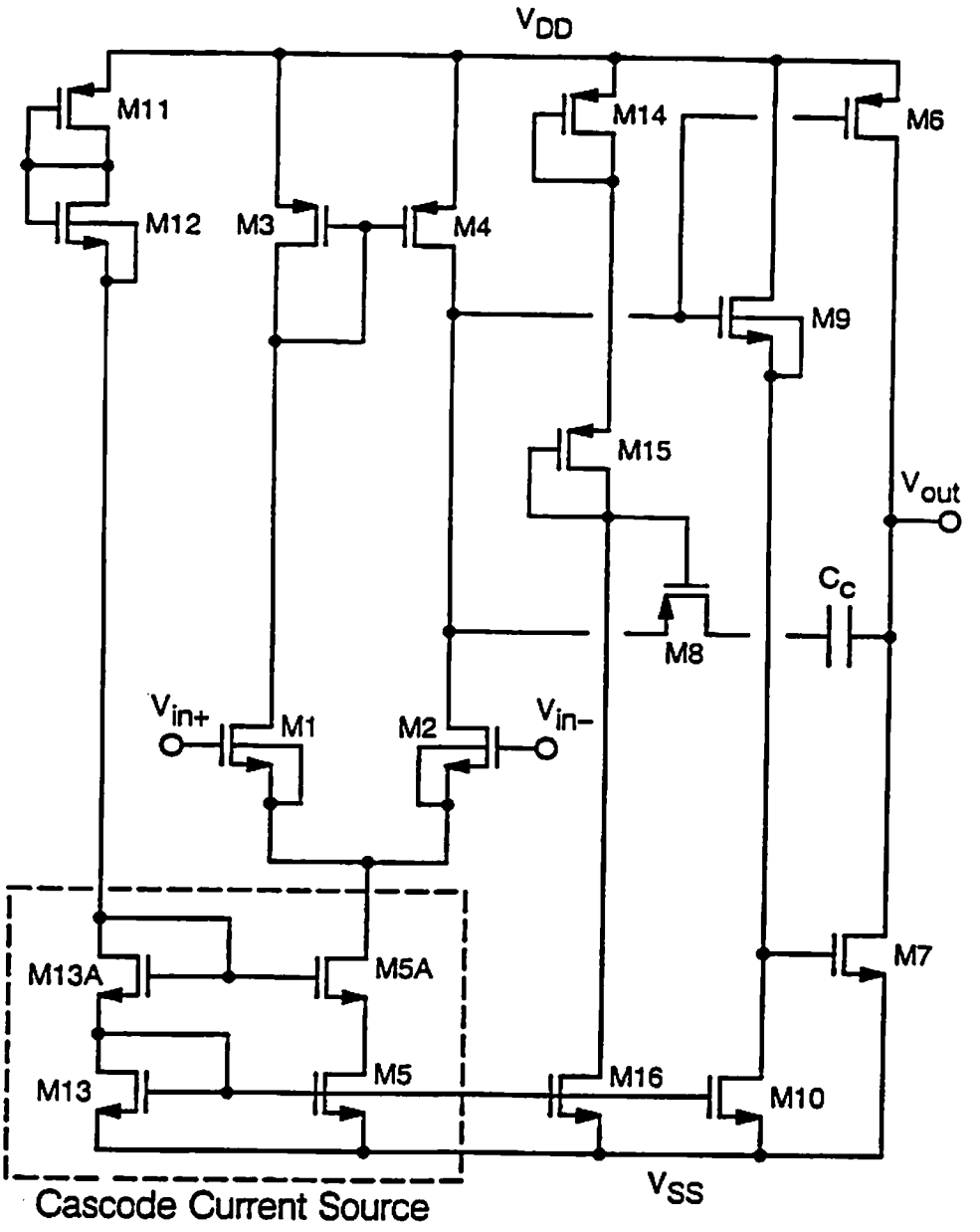


Fig. 2.9 (continued)
(b) with cascode biasing current source for common-mode rejection improvement.

current source in this case. Figure 2.9(a) and (b) show the schematics of the starting circuit and modified circuit respectively. For this particular design, 3 iterations were required before the self-modification and 2 more iterations were needed to finalize the design. SPICE is used for intermediate design evaluations during the design process as well as for final performance evaluation. Iterations between SPICE and CAMP contribute to the fast emergence of the final design. Without the strong interactions with SPICE, an accurate and reliable design would not be achieved efficiently. Table 2.2 lists the transistor sizes for the final circuit.

Table 2.2 Drawn device sizes.

Device	W (um)	L (um)	Device	W (um)	L (um)
M1	146	16	M9	3	54
M2	146	16	M10	25	3
M3	23	16	M11	3	3
M4	23	16	M12	3	123
M5	30	3	M13	11	3
M5A	30	3	M13A	11	3
M6	4	3	M14	3	3
M7	15	3	M15	3	3
M8	4	3	M16	22	3
Cc	5 pF				

Chapter 3

Constraint-Based Analog IC Layout Synthesis

3.1 Analog Layout Considerations

The field of layout automation for analog ICs has progressed at a much slower rate than has been the case for digital circuits. The main difficulty lies in the fact that the layout requirements for analog circuits are generally more intricate and difficult to deal with. This can be attributed to the diverse, sensitive, and complicated nature of analog circuits. The analog circuits are generally quite diverse in a wide variety of circuit functions and topologies with a wide performance range. Take the operational amplifier module for example: there exists dozens of circuit architectures [50] based on various design choices, such as one-stage or two-stage, class-A or class-AB, cascode or without cascode, static- or dynamic-biasing, with or without low-impedance output stage, single-ended or fully-differential output. Unlike the digital domain where the usage of a fixed standard cell library laid out in semicustom styles is generally sufficient, in the analog domain it is not only difficult, but also impractical to store a rich enough set of analog circuit library cells to cover the wide spectrum of possible applications. Furthermore, such cell libraries tend to become obsolete as soon as the process technology or design rules change. As a consequence, parameterized generators that are technology independent and can operate at the device, module, and up to the subsystem levels are more flexible and thus are more suit-

able for analog IC applications. Therefore, the development of such generators is a key step toward automatic layout of analog VLSI circuits.

Performance issue is the major concern for analog IC layout automation. Unlike digital circuits, the performance of the analog circuits is heavily dependent on fine details of physical layout and device behavior. The key considerations are the effects of device mismatching, parasitics, and noise coupling on analog circuit layout. These undesirable effects can lead to various kinds of performance degradations for analog circuits. For instance, the poor matching of critical analog devices [53,54] in the layout will induce large offset and poor accuracy performance at the circuit and subsystem levels. Excessive interconnect parasitics on a critical node will also deteriorate the transient and frequency responses of an amplifier. Furthermore, any noise coupled into the sensitive analog circuit layout area will significantly reduce the achievable circuit dynamic range. These performance constraints can be very stringent especially for some high performance system applications.

The area efficiency of the produced layout is also a concern. One of the main difficulties in analog IC layout is the fact that device sizes in the same circuit can vary over a wide range. In fact, as large as two orders of magnitude variations in device sizes is not uncommon. These large devices usually have to be laid out in variable shapes to result in a more compact analog circuit layout. Another important consideration is in the user interface. One particular concern here is the fact that VLSI system designers typically do not have extensive analog IC knowledge. Therefore, the layout tool itself must possess ample expert knowledge to generate circuit layout without requiring special help from the

users. Finally, to match the quick turnaround requirement in the VLSI design environment, the computation efficiency of the layout tool must also be optimized.

3.2 Review of Automatic Analog Layout Generators

Recently, analog circuit layout synthesis has received increased attention from the research community [32-45]. Most earlier attempts typically relied on a semicustom style using analog standard cell libraries [32-34]. However, as explained before, this method had achieved very limited success. More recently, recognizing the limitations of the standard cell approach, several custom analog layout methodologies based on the module generation concept have emerged [35-41]. These approaches either use simple fixed-topology floorplans [35,41], dedicated slicing trees [37-39], or rely on a bottom-up general simulated annealing program [36,40]. Prototype analog IC layout tools such as Berkeley's OPASYN [39], CMU's ANAGRAM [40], and CSEM's ILAC [38] are developed.

OPASYN uses a top-down fixed-floorplan approach based on predefined slicing trees for placement [55] and a net-by-net switchbox routing procedure. While this approach is workable and can produce a good result, the main drawback is that a specific slicing tree must be manually designed by human expert designers for each of the generic circuit topologies and pre-stored in the database for placement. It can handle only certain topologies of Op-Amp building blocks.

As a result, this tool is considered only semi-automated and provides very limited flexibility.

ILAC also adopts the slicing structures, but relies on a simulated annealing technique for placement optimization [56] and a scanline-based incremental channel router. Comparing to OPASYN, this approach shows a higher degree of flexibility. However, the tool is not fully automated, either. To obtain a satisfactory performance, it still requires an analog IC expert to manually specify analog circuit layout constraints such as matching device pairs for each circuit as part of the inputs. Moreover, since this floorplan optimization technique is adopted from the digital design world [56] where the area and wire length are the only concerns, most of the analog-specific performance requirements cannot be effectively addressed. Consequently, the achievable performance with this approach can be limited.

ANAGRAM uses a general simulated annealing method based on arbitrary (i.e. non-slicing) structures for placement and a line expansion algorithm for routing. Comparing to OPASYN and ILAC, this approach gives the highest degree of flexibility, but often at the expense of performance. One major limitation is that allowing the arbitrary layout structures, in addition to the versatile analog circuit layout constraints resulting from variable shapes of the cells, different terminal locations, geometrical matching, parasitics, noise coupling, etc., has created too many degrees of freedom for the conventional simulated annealing algorithms to handle efficiently. In ANAGRAM, the annealing cost function includes terms to minimize total area and wire length, but incorporates only very little analog circuit layout constraints. As a consequence, the performance of the resulting

layout is often not well optimized. Their generated layout results are usually less predictable and look quite different from those handcrafted by expert designers.

Some disadvantages common in all three systems are as follows. First, performance considerations are not addressed adquately and systematically in these systems which often lead to much less optimized layout results. Second, they do not have enough expert analog layout knowledge built into the system and thus are not really tailored to be used by system designers on a standalone basis. And finally, more work is yet to be done to extend the analog circuit layout generator from the module level into the higher analog subsystem level.

Recognition of the shortcomings of the above approaches has led to a new approach to the automatic custom layout generation for analog ICs [42-45] which is described next.

3.3 The Constraint-Based Approach

Our approach is quite distinguished from existing methods. It is based on automatic circuit recognition and layout constraint analysis techniques by systematically capturing the analog IC layout expertise of human engineers. Instead of relying on a general optimization approach commonly used to solve the digital IC layout problems, we believe that the most effective solution for automatic analog circuit layout is to directly mimic the actual layout design practice used by human analog IC experts. The typical analog circuit layout design flow used by human experts is illustrated in Fig. 3.1. The layout generation starts with a sized schematic of an analog circuit module. The designer first reduces the

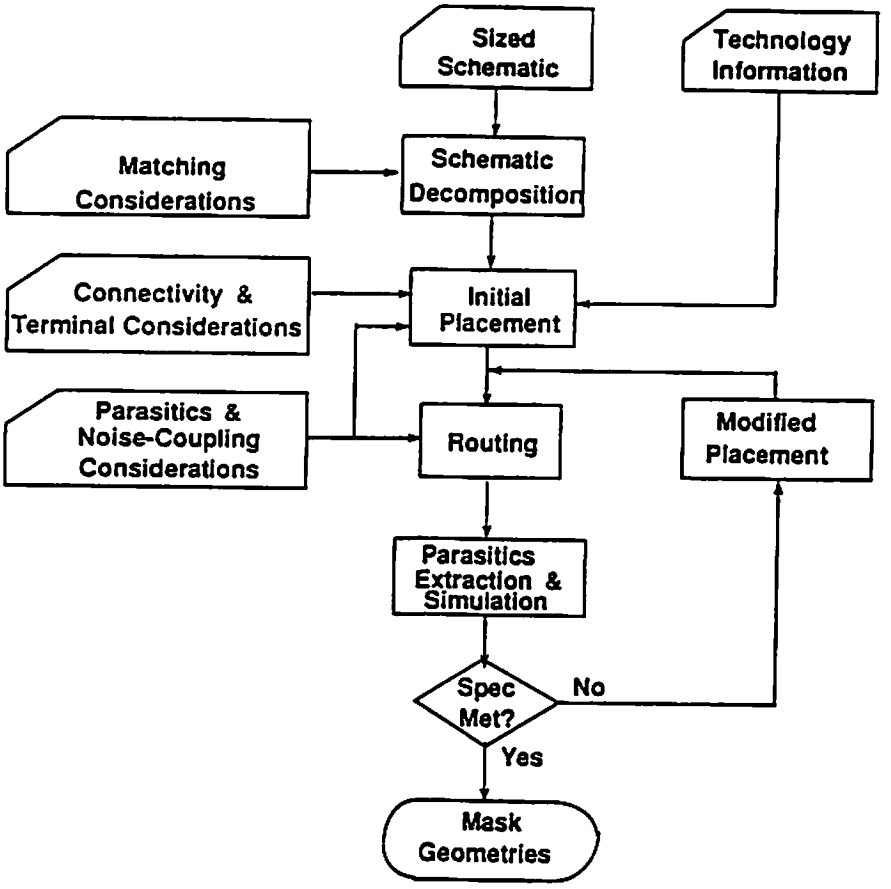


Fig. 3.1 Analog circuit layout design flow - a human expert model.

complexity of the problem by identifying matched device pairs in the circuit and decomposing the schematic into a set of smaller cells based on these matched devices. Considering layout design rules, connectivity, terminal locations, parasitics, noise coupling, and so forth, the expert designer then works out a sound topological arrangement for all the cells in the circuit and sketches a rough placement of these cells. After the initial placement of the cells, the designer routes the nets and at the same time compacts the resulting layout. When the layout is done, parasitic elements such as parasitic capacitances and resistances from wirings are extracted from the mask geometries. These parasitic elements are then added to the input circuit schematic and the resulting circuit is simulated for verification. If all the performance and area specifications are met, the layout design is completed. Otherwise, the current layout must be modified to reduce the excessive parasitic elements to improve performance or to obtain better area utilization if necessary.

The basic strategies used in our approach to achieve high performance are first to incorporate the major component matching requirements within the circuit primitives, then to internally generate layout constraints to minimize the interconnect parasitics and noise couplings. Figure 3.2 shows the flowchart of the automatic analog circuit layout synthesis method. It consists of five major processing steps: circuit primitive recognition, critical net analysis, floorplanning, primitive cell generation, and routing. The system performs a complete netlist-to-layout synthesis task. It starts with an analysis of the circuit before actual layout generation. Given the schematic netlist of an analog circuit module (which can be simply extracted from the SPICE input deck of the module), the

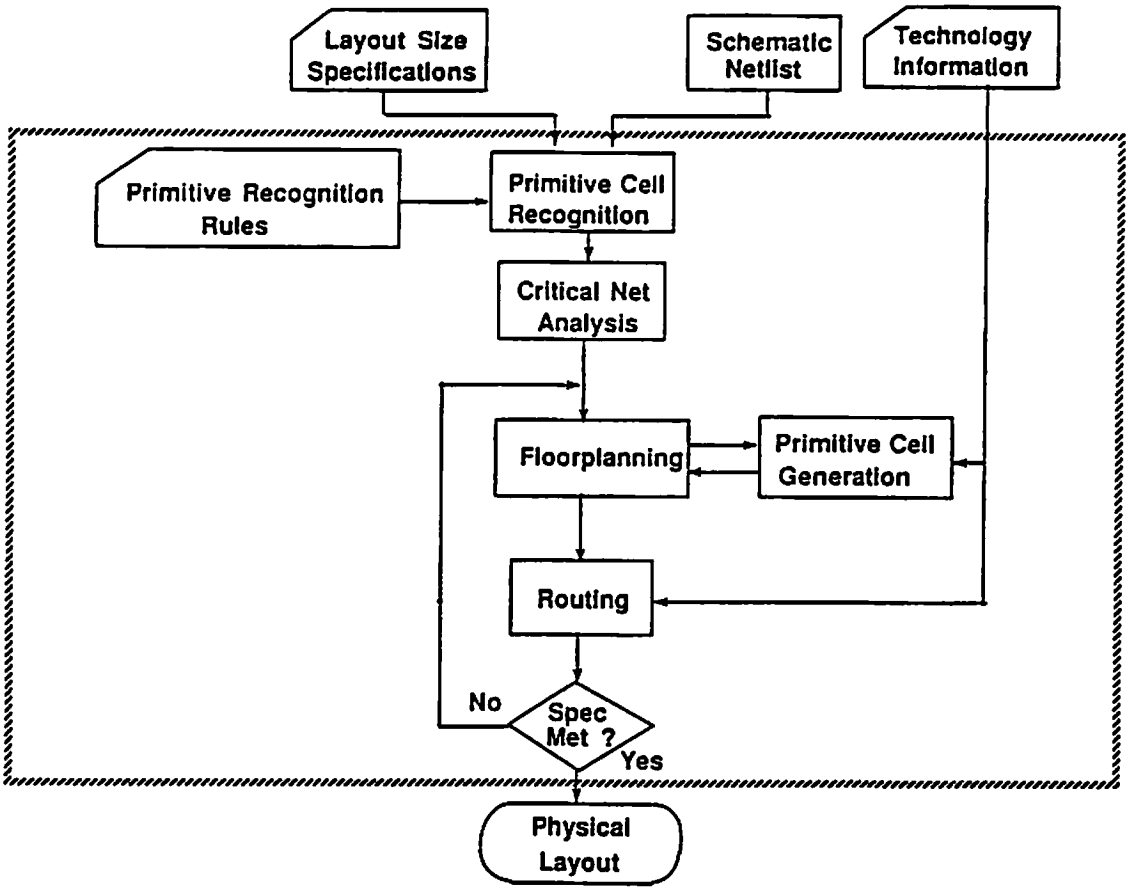


Fig. 3.2 An automatic custom layout synthesis methodology for analog ICs.

system first identifies critical analog circuit nodes and then recognizes matched device pairs in the circuit. The recognizable analog circuit primitives can range from a single transistor to a cascode current mirror. Based on the recognized circuit information, the system then carries out a critical net analysis for the circuit and internally generates proper layout constraints. The weighted constraints are then fed into the constraint-driven analog floorplanning and routing procedures and incorporated in the final layout. The next two chapters describe each processing step in detail.

Chapter 4

Circuit Recognition and Layout Constraint Analysis

4.1 Analog Circuit Recognition

Circuit recognition is an important step to design automation. A digital circuit recognition technique was proposed for symbolic circuit verifications [57] in which the functionality of logic components in a static CMOS circuit can be easily recognized using a small set of Boolean or logic circuit expressions. This is possible because a digital circuit schematic is normally composed of few logic primitives such as NOT, NAND, and NOR. Such a recognition technique, however, would not work directly on analog circuits. This is mainly because that analog designs, unlike digital designs which only need to deal with two discrete logic levels (0 and 1), have to deal with a continuous spectrum of voltage levels. Consequently, an analog circuit can be designed in an almost infinite number of variations, which in turn makes the analog circuit recognition rather difficult.

Analog circuit recognition [29] is a crucial step toward automatic synthesis of high-quality custom analog IC layout. If crucial portions of an analog circuit can be recognized and given special layout processing, then component mismatching, parasitics, and noise coupling that would result in degraded analog circuit performance can be minimized. To serve this purpose, we have developed a unique rule-based analog circuit recognition technique [42] from a layout synthesis perspective. This technique is designed to recognize matched device pairs in an analog circuit (such as differential pairs and current mirrors)

so that they can be treated as special units in the layout to achieve excellent matching, while decomposing the circuit schematic into a small set of predefined circuit primitives. In addition, it can identify critical circuit nodes in the schematic that are specially sensitive to interconnect parasitics and noise coupling in the layout.

The analog circuit recognition is realized through the application of recognition rules. Rules are defined for critical circuit nodes and circuit primitives. The input to the recognition program is a SPICE-like netlist extracted from the circuit schematic and the output is a set of recognized circuit nodes and circuit primitives. The recognition process consists of three steps. First, the critical circuit nodes are identified. Second, the applicable recognition rules are applied to recognize analog circuit primitives. Third, the circuit primitive information is then assembled for primitive cell generation.

4.1.1 Critical analog circuit nodes

An analog circuit schematic can be described in terms of a number of nodes, circuit primitives, and their interconnection patterns. Table 4.1 lists a set of circuit nodes used for analog MOS circuit recognition. The first part contains basic circuit nodes extracted from the input netlist. They consist of power (P), input (I), output (O), and bias (B) pins as well as internal component nodes such as transistor gate (G), drain (D), source (S), and substrate (U) terminals, capacitor plate (C), and resistor terminal (R). A node that connects two or more transistor terminals is defined as a junction node. For analog MOS circuits, four critical junction nodes can be identified [42]: diode-connected node, high-impedance node, source-coupled node, and current-mirroring node. Figure 4.1

Table 4.1 Circuit nodes for analog MOS primitive recognition.

Basic Circuit Nodes	Symbols
Power Source Node	P
Input Node	I
Output Node	O
Bias Node	B
Transistor Gate Terminal	G
Transistor Drain Terminal	D
Transistor Source Terminal	S
Transistor Substrate Terminal	U
Capacitor Plate	C
Resistor Terminal	R
Junction Nodes	Symbols
High-Impedance Node	HI
Diode-Connected Node	DC
Source-Coupled Node	SC
Current-Mirroring Node	CM

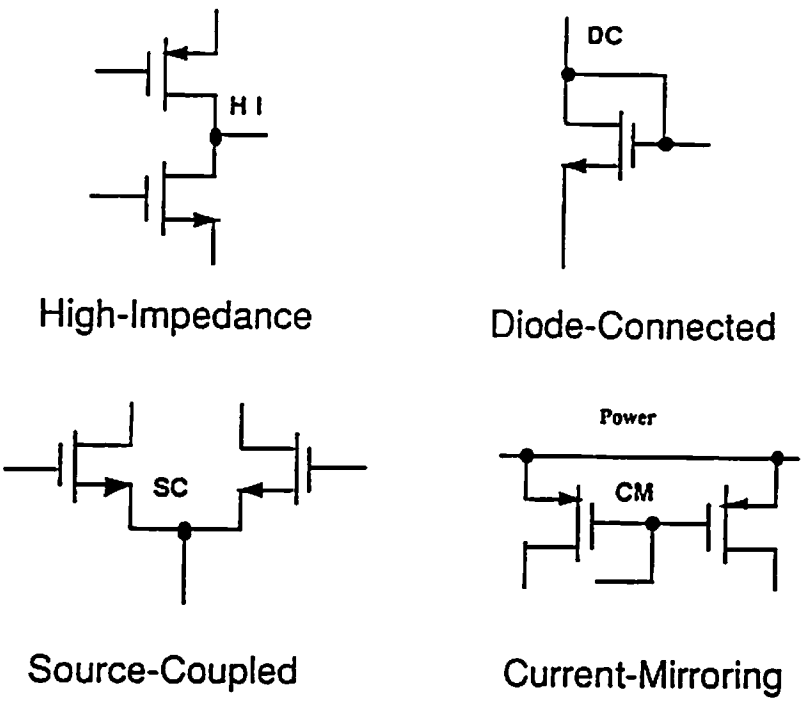


Fig. 4.1 Four critical analog MOS circuit nodes.

shows example schematics of these critical circuit nodes which usually require special layout care.

A diode-connected (DC) node is defined as a junction node that connects the gate terminal of a transistor to its drain terminal (which is equivalent to a diode connection), i.e.,

$$DC = G(M_1) \cdot D(M_1). \tag{4.1}$$

where M_i denotes a transistor, G(M_i) is the gate terminal and D(M_i) is the drain terminal of the transistor. Since a DC node is normally used to bias an analog circuit, it requires special layout care in device matching and parasitics.

A high-impedance (HI) node is identified as a junction node that connects drain terminals of multiple transistors, i.e.,

$$HI = D(M_1) \cdot D(M_2) \cdots D(M_n). \tag{4.2}$$

If an HI node merges with a DC node which has a lower impedance, then the result is a DC node. An HI node normally represents a dominant pole of an amplifier circuit and thus the associated parasitics and noise coupling should be minimized.

A source-coupled (SC) node is defined as a junction node that connects source terminals of multiple transistors, i.e.,

$$SC = S(M_1) \cdot S(M_2) \cdots S(M_n). \tag{4.3}$$

If an SC node merges with a power source node, then the result is simply a power source node. An SC node is normally located in the center of a differential amplifier circuit where the input signal is very sensitive to crosstalk

noise. In addition, component matching of the differential pair is very critical.

A current-mirroring (CM) node is actually an extension of a common-gate (CG) node which is identified as a junction node that connects gate terminals of multiple transistors, i.e.,

$$CG = G(M_1) \cdot G(M_2) \cdots G(M_n). \tag{4.4}$$

A CM node is a CG node with the source terminals of the associated transistors tied to a same power source node. The CM node is typically used to link a current mirror circuit, while the CG node usually inserts a cascode stage for an amplifier circuit. Both require good device matching.

As will be evidenced later, the use of these circuit nodes can significantly simplify the rules required for recognizing the analog circuit primitives.

4.1.2 Analog circuit primitives and recognition rules

The recognition technique uses a special set of analog circuit primitives ranging from a single transistor to a cascode current mirror to achieve high performance and flexibility. Figure 4.2 shows schematics of seven generic circuit primitives which are parameterizable and recognizable by the prototype system. Note that all the typical matched device pairs in analog MOS circuits such as the input differential pair, current mirror, and ratioed current source are included in these circuit primitive structures to ensure good matching performance. Each circuit primitive can be parameterized in different transistor types and physical sizes. The number of devices in the multiple-transistor circuit primitives is also programmable. In addition, each primitive can be constrained by a group matching parameter to allow a balanced match between selected circuit primitives.

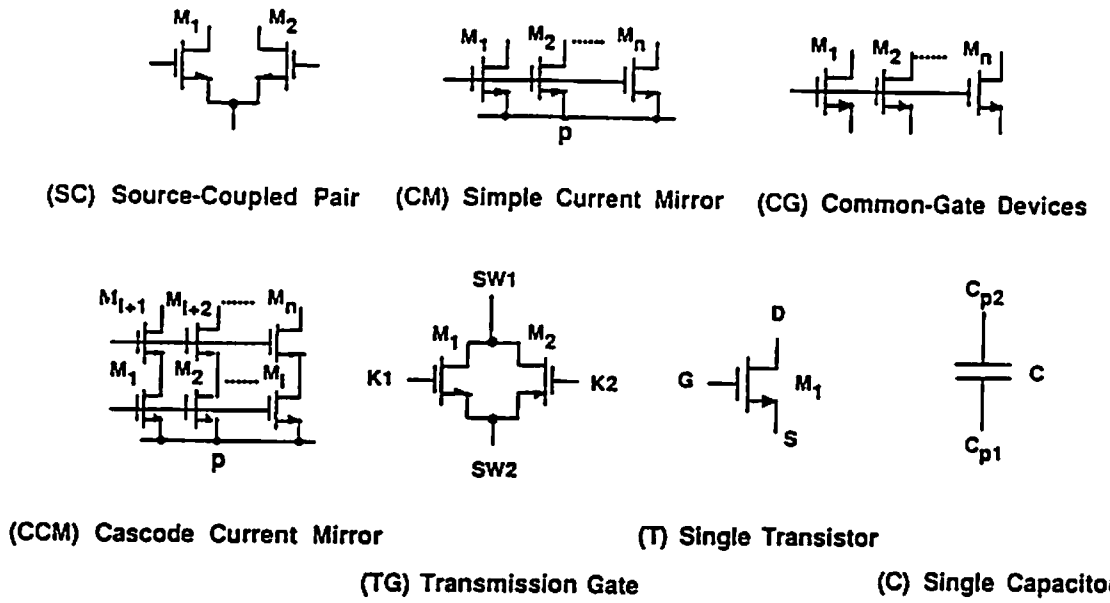


Fig. 4.2 A set of recognizable analog MOS circuit primitives.

To recognize these circuit primitives for various analog CMOS circuits, the following seven basic rules are used. In addition, extended rules can be applied to impose the extra layout care required for certain special circuit primitives.

R1: <source-coupled device primitive (SC)>

$$SC\{M_1, M_2, ..., M_n\}:$$

$$\left[S(M_1) = S(M_2) = \cdots = S(M_n) = SC\right] \wedge$$

$$\left[t(M_1) = t(M_2) = \cdots = t(M_n)\right], \quad n \geq 2. \tag{4.5}$$

Here M_i denotes a transistor and $t(M_i)$ represents the transistor type, either p or n. This rule states that if the source terminals of transistors M_1-M_n, which have the same transistor type, are connected to form an SC node, then these transistors are recognized as a source-coupled device primitive. An important extension of this primitive would be an input differential pair primitive typically found in an Op-Amp. In this case, the added condition is that the gate terminals of the transistor pair must be connected to the circuit input nodes, i.e.,

R1.1: <input differential pair primitive (SC-IN)>

$$SC\text{-}IN\{M_1, M_2\}:$$

$$\left[R1(n=2)\right] \wedge \left[G(M_1) = I\right] \wedge \left[G(M_2) = I\right]. \tag{4.6}$$

Almost any analog circuit needs a current mirror primitive. The key circuit node to look for in this primitive is the CM node, i.e.,

R2: <simple current mirror primitive (CM)>

$$CM\{M_1, M_2, ..., M_n\}:$$

$$[G(M_1) = G(M_2) = \cdots = G(M_n) = CM] \wedge$$

$$[t(M_1) = t(M_2) = \cdots = t(M_n)], \quad n \geq 2. \tag{4.7}$$

Similarly, this basic rule can be extended to further identify an input-stage load primitive typically found in a two-stage Op-Amp. The extended rule requires that the drain terminal of one transistor be a DC node, while the drain terminal of the other transistor must be an HI node which is not tied to an output node, i.e.,

R2.1: <Op-Amp input-stage load primitive (CM-OPIL)>

$$CM-OPIL\{M_1, M_2\}:$$

$$[R2(n=2)] \wedge [D(M_1) = DC] \wedge$$

$$[D(M_2) = HI] \wedge [D(M_2) \neq O]. \tag{4.8}$$

Common-gate device primitive is another widely used primitive, especially in cascode amplifier circuits. In most cases, the primitive forms the cascode device itself. The recognition rule is similar to R2:

R3 : <common-gate devices primitive (CG)>

$$CG\{M_1, M_2, ..., M_n\}:$$

$$\left[G(M_1) = G(M_2) = \cdots = G(M_n) = CG\right] \Lambda$$

$$\left[t(M_1) = t(M_2) = \cdots = t(M_n)\right], \quad n \geq 2. \tag{4.9}$$

By stacking the simple current mirror and the common-gate device primitive together, a cascode current mirror primitive can be formed, i.e.,

R4: <cascode current mirror primitive (CCM)>

$$CCM\{M_1, M_2, \ldots, M_n\}:$$

$$\left[CM\{M_1, M_2, \ldots, M_i\}\right] \Lambda \left[CG\{M_{i+1}, M_{i+2}, \ldots, M_n\}\right] \Lambda$$

$$\left[D(M_j) = S(M_{j+1}),\ j = 1, 2, \ldots, i\right] \Lambda$$

$$\left[t(M_1) = t(M_2) = \cdots = t(M_n)\right], \quad n = 2i \geq 4. \tag{4.10}$$

In this way, the parasitics associated with the cascode node, which typically represents a non-dominant pole of a cascode amplifier [50], can be minimized.

A CMOS transmission gate, which is composed of a PMOS transistor and an NMOS transistor, is typically used as a switch in the switched-capacitor analog circuits. The recognition rule for that is quite simple:

R5: <CMOS transmission gate (TG)>

$$TG\{M_1, M_2\}:$$

$$\left[D(M_1) = D(M_2)\right] \Lambda \left[S(M_1) = S(M_2)\right] \Lambda$$

$$\left[t(M_1) \neq t(M_2)\right]. \tag{4.11}$$

After applying the above recognition rules to an analog circuit, most of the devices that require special matching arrangement should have been recognized. Thus, the rest of transistor elements in the circuit can be treated as single transistor primitives. These primitives can be identified from the transistor elements that are not part of the already-recognized multiple-transistor primitives after application of rules R1-R5, i.e.,

R6: <single transistor primitive (T)>

$$T\left\{M_1\right\} = M_i;\ \ M_i \notin \left\{SC, CM, CG, CCM, TG\right\}. \tag{4.12}$$

For some important single-transistor primitives, such as the output driver transistor for a two-stage Op-AMP, other special layout attention is also required. Since in that case the transistor is typically connected between the two high-impedance nodes of the Op-Amp, this recognition rule can be extended as follows:

R6.1: <Op-Amp output driver primitive (T-OPOD)>

$$T\text{–}OPOD\left\{M_1\right\}:$$

$$\left[R6\right] \wedge \left[D(M_1) = HI\right] \wedge \left[D(M_1) = O\right] \wedge$$

$$\left[G(M_1) = HI\right]. \tag{4.13}$$

A capacitor primitive is indispensable for switched-capacitor circuits. The corresponding rule is:

R7: <single capacitor primitive (C)>

$$C: \left[C_{p1} = C\right] \wedge \left[C_{p2} = C\right]. \tag{4.14}$$

Note that these rules can handle frequently-used analog CMOS circuit modules. As can be seen, both the recognition rules and circuit primitives are designed to be flexible such that they can be easily extended to handle new types of analog circuits. For example, a resistor primitive and some bipolar transistor primitives can easily be added to handle the BICMOS analog designs as well.

To handle fully differential circuits or some circuits where good thermal matching is required, additional recognition steps are used to identify the symmetrical matching constraints needed to impose between certain circuit primitives [45]. For fully differential circuits, the recognized differential primitive pairs are further partitioned into two symmetrical primitive groups to optimize the differential circuit performance. For circuits where good thermal matching is required, proper constraints on balanced match between the sensitive circuit primitives and the recognized high-current circuit primitives, which are typically located in the output driver stage of an analog circuit module, can also be generated.

4.1.3 Examples

To illustrate the effectiveness of the recognition technique for analog circuit layout considerations, let's consider the following examples of commonly-used analog IC modules including a two-stage CMOS Op-Amp, a single-stage folded cascode Op-Amp, a voltage comparator, an analog multiplier, a voltage-controlled oscillator, a sense amplifier, and a fully-differential CMOS Op-Amp.

Example 1 -- Two-stage CMOS Op-Amp

Figure 4.3(a) shows the schematic of a popular two-stage CMOS Op-Amp [58]. Transistors M_1-M_7 form the two-stage core amplifier with M_8-M_{11} and capacitor C_c providing a tracking RC compensation. By applying seven recognition rules to this circuit, the Op-Amp schematic can be decomposed into seven recognized circuit primitives including critically matched device pairs as shown in the schematic. For example, transistors M_1 and M_2 are recognized through rule R1.1 as an input differential pair primitive and are to be closely matched. The four n-channel transistors at the bottom ($M_5/M_7/M_9/M_{12}$) are recognized through rule R2 as a current mirror primitive and thus all four transistors are to be matched to maintain proper biasing. Finally, transistor M_6 is recognized through rule R6.1 as an output transistor primitive whose gate and drain terminals are both connected to high impedance nodes and thus the associated parasitics and noise coupling are to be minimized. Figure 4.3(b) shows the SPICE input file of the Op-Amp used as an input to the circuit recognition program. The generated output file showing the recognized circuit primitives as well as the critical circuit nodes is listed in Fig. 4.3(c).

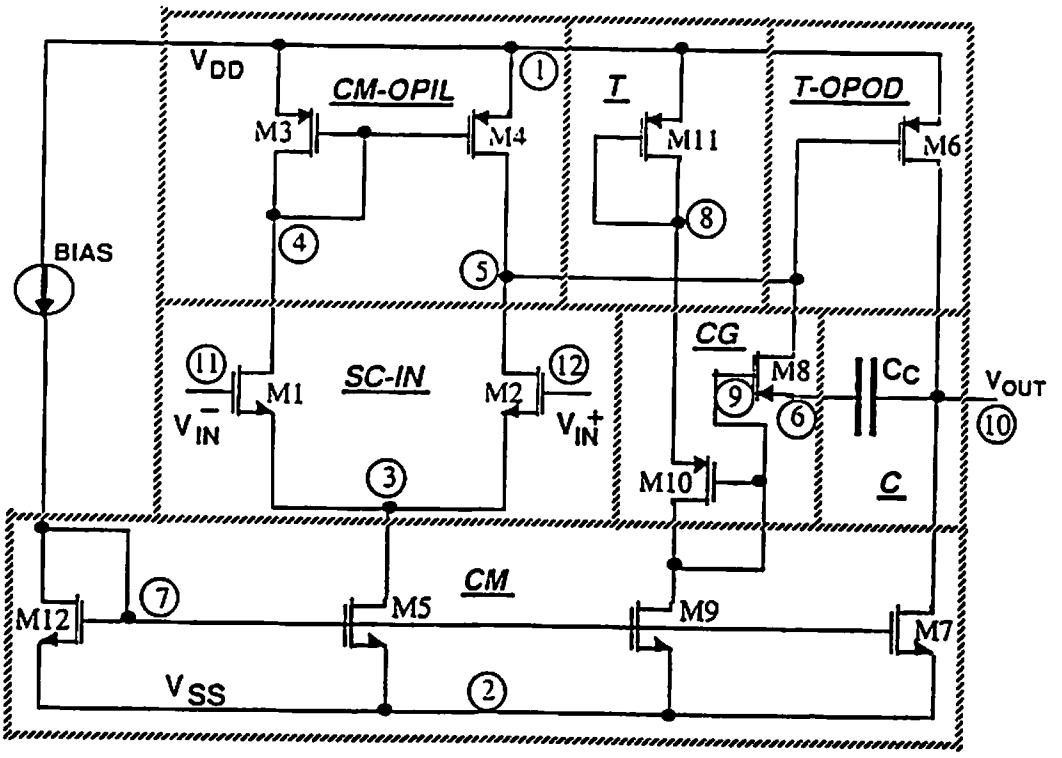


Fig. 4.3 Circuit recognition of a two-stage CMOS Op-Amp.
(a) Schematic of recognized circuit.

```
*** Example 1 - Basic Two-Stage CMOS Opamp ***
*
* Input Stage
*
M1   4 11  3  3  N W=60U  L=2U
M2   5 12  3  3  N W=60U  L=2U
M3   4  4  1  1  P W=60U  L=3U
M4   5  4  1  1  P W=60U  L=3U
M5   3  7  2  2  N W=40U  L=4U
*
* Output Stage
*
M6  10  5  1  1  P W=600U L=3U
M7  10  7  2  2  N W=200U L=4U
*
* Frequency Compensation
*
M8   6  9  5  1  P W=108U L=3U
M9   9  7  2  2  N W=25U  L=4U
M10  9  9  8  1  P W=12U  L=3U
M11  8  8  1  1  P W=75U  L=3U
*
CC 6 10 5P
*
* Bias
*
M12  7  7  2  2  N W=20U  L=4U
IBIAS 1 7 10U
*
VDD 1 0 +5V
VSS 2 0 0V
VIN- 11 0
VIN+ 12 0
*
CL 10 0 20P
*
```

Fig. 4.3 (continued)
(b) SPICE input file.

```
*** Example 1 - Basic Two-Stage CMOS Opamp ***

** Basic Circuit Nodes **
P  : {1,2}
I  : {11,12}
O  : {10}
B  : {7}
C  : {6,10}

** Critical Junction Nodes **
DC : {4,7,8,9}                    /* diode-connected node */

HI : {5,10}                       /* high-impedance node */

SC : {3}                          /* source-coupled node */

CM : {4,7}                        /* current-mirroring node */

** Recognized Primitive Cells **
SC-IN(n) : {M1,M2}                /* input differential pair */

CM(n) : {M5,M7,M9,M12}            /* current mirror */

CM-OPIL(p) : {M3,M4}              /* Op-Amp input-stage load */

CG(p) : {M8,M10}                  /* common-gate devices */

T(p) : {M11}                      /* single transistor */

T-OPOD(p) : {M6}                  /* Op-Amp output driver */

C : {CC}                          /* single capacitor */
```

Fig. 4.3 (continued)
(c) Recognition output file.

Example 2 -- Single-Stage Folded Cascode Op-Amp

Figure 4.4(a) shows the schematic of a single-stage folded cascode Op-Amp [2] commonly used for high-speed switched-capacitor circuit applications. This Op-Amp features a cascode gain stage, M_1-M_{10}, with a high-swing cascode biasing circuit, M_{11}-M_{19}. Transistors M_{20}-M_{24} perform the differential-to-single-ended conversion for the output while providing an offset compensation. By applying four recognition rules to this circuit, a total of nine primitive cells including all the matched current mirrors and cascode devices are recognized as shown in the schematic. Figure 4.4(b) shows the SPICE input netlist of the Op-Amp. The recognition output file is listed in Fig. 4.4(c).

Example 3 -- Voltage Comparator

Figure 4.5(a) shows the schematic of a common voltage comparator with hysteresis [6]. Transistors M_1-M_6 form the differential input stage of the comparator with M_{11}-M_{12} supplying the tail current. The output stage is composed of M_7-M_{10}. All the five matched device pairs in the circuit can be identified using two recognition rules. The SPICE input netlist of the comparator is shown in Fig. 4.5(b) and the recognition result is listed in Fig. 4.5(c).

Example 4 -- Analog Multiplier

Figure 4.6(a) shows the schematic of a Gilbert multiplier [59], which is an important building block for conventional analog signal processing subsystems as well as neural networks. The core of the four-quadrant multiplier, including transistors M_1-M_{12}, consists of three differential pairs with four voltage inputs, V_{IN1}-V_{IN4}. Transistors M_{13}-M_{16} form the output stage of the multiplier. By

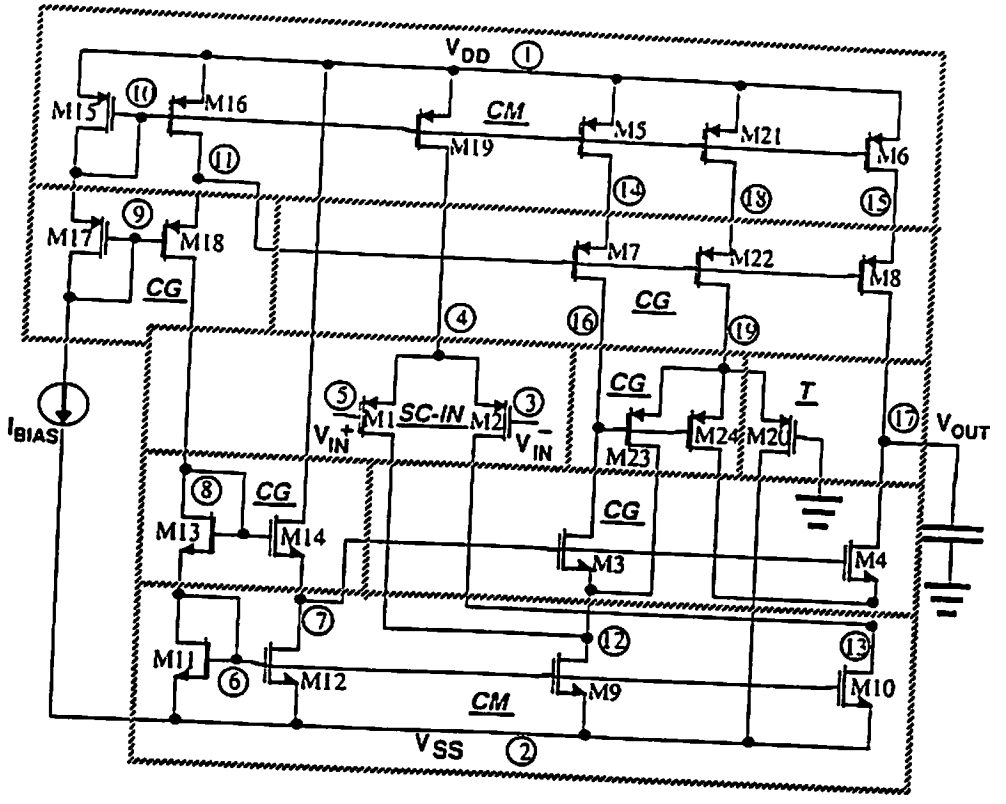


Fig. 4.4 Circuit recognition of a single-stage folded cascode Op-Amp.
(a) Schematic of recognized circuit.

```
*** Example 2 - Single-Stage Folded Cascode Op-Amp ***
*
* Gain Stage
*
M1  12  5  4  4  P W=600U  L=3U
M2  13  3  4  4  P W=600U  L=3U
M3  16  7 12  2  N W=600U  L=4U
M4  17  7 13  2  N W=600U  L=4U
M5  14 10  1  1  P W=1200U  L=3U
M6  15 10  1  1  P W=1200U  L=3U
M7  16 11 14 14  P W=1200U  L=3U
M8  17 11 15 15  P W=1200U  L=3U
M9  12  6  2  2  N W=1350U  L=4U
M10 13  6  2  2  N W=1350U  L=4U
*
* Bias Circuit
*
M11  6  6  2  2  N W=600U   L=4U
M12  7  6  2  2  N W=600U   L=4U
M13  8  8  6  2  N W=75U    L=4U
M14  1  8  7  2  N W=600U   L=4U
M15 10 10  1  1  P W=1200U  L=3U
M16 11 10  1  1  P W=1200U  L=3U
M17  9  9 10 10  P W=225U   L=3U
M18  8  9 11 11  P W=1200U  L=3U
M19  4 10  1  1  P W=2400U  L=3U
*
* Offset Compensation
*
M20  2  0 19 19  P W=600U   L=3U
M21 18 10  1  1  P W=1200U  L=3U
M22 19 11 18 18  P W=1200U  L=3U
M23 12 16 19 19  P W=600U   L=3U
M24 13 16 19 19  P W=600U   L=3U
IBIAS 9 2 100U
CL 17 0 2P
*
VDD 1 0 +2.5V
VSS 2 0 -2.5V
VIN+ 5 0 0
VIN- 3 0 0 AC 1
```

Fig. 4.4 (continued)
(b) SPICE input file.

```
*** Example 2 - Single-Stage Folded Cascode Op-Amp ***

** Basic Circuit Nodes **
P  : {1,2}
I  : {3,5}
O  : {17}
B  : {9}
C  : {}

** Critical Junction Nodes **
DC : {6,8,9,10}                          /* diode-connected node */

HI : {16,17}                             /* high-impedance node */

SC : {4,19}                              /* source-coupled node */

CM : {6,10}                              /* current-mirroring node */

** Recognized Primitive Cells **
SC-IN(p) : {M1,M2}                       /* input differential pair */

CM(n) : {M9,M10,M11,M12}                 /* current mirror */
CM(p) : {M5,M6,M15,M16,M19,M21}

CG(n) : {M3,M4}                          /* common-gate devices */
CG(n) : {M13,M14}
CG(p) : {M7,M8,M22}
CG(p) : {M17,M18}
CG(p) : {M23,M24}

T(p) : {M20}                             /* single transistor */
```

Fig. 4.4 (continued)
(c) Recognition output file.

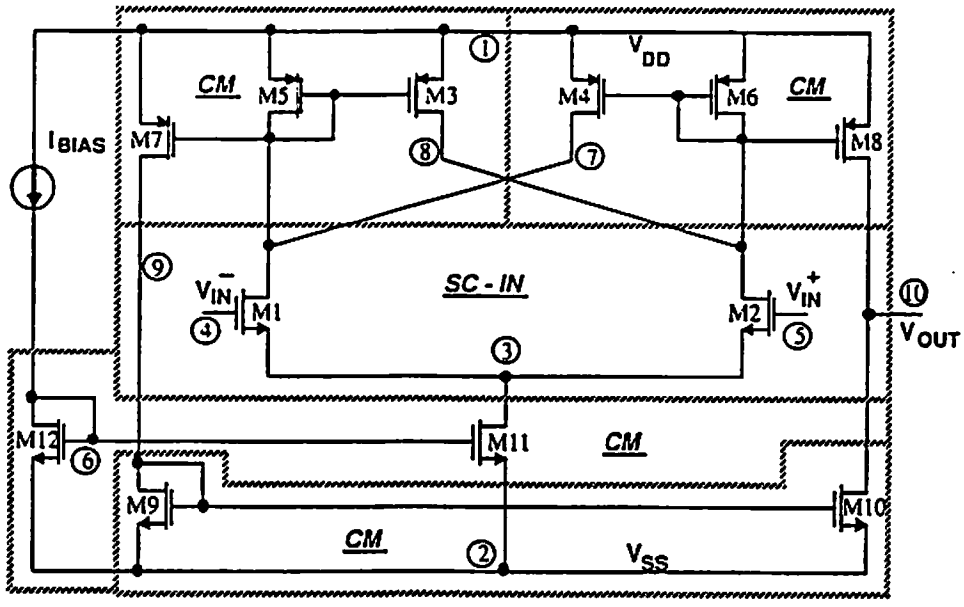


Fig. 4.5 Circuit recognition of a voltage comparator.
(a) Schematic of recognized circuit.

```
*** Example 3 - Voltage Comparator ***
*
*
M1   7  4  3  2  N W=30U  L=2U
M2   8  5  3  2  N W=30U  L=2U
M3   8  7  1  1  P W=20U  L=4U
M4   7  8  1  1  P W=20U  L=4U
M5   7  7  1  1  P W=30U  L=4U
M6   8  8  1  1  P W=30U  L=4U
M7   9  7  1  1  P W=60U  L=4U
M8  10  8  1  1  P W=60U  L=4U
M9   9  9  2  2  N W=60U  L=4U
M10 10  9  2  2  N W=60U  L=4U
M11  3  6  2  2  N W=40U  L=4U
M12  6  6  2  2  N W=20U  L=4U
*
IBIAS 1 6 20U
*
VDD 1 0 5V
VSS 2 0 0V
VIN- 4 0
VIN+ 5 0
*
VOUT 10 0
*
```

Fig. 4.5 (continued)
(b) SPICE input file.

```
*** Example 3 - Voltage Comparator ***

** Basic Circuit Nodes **
P  : {1,2}
I  : {4,5}
O  : {10}
B  : {6}
C  : {}

** Critical Junction Nodes **
DC : {6,7,8,9}                    /* diode-connected node */

HI : {10}                         /* high-impedance node */

SC : {3}                          /* source-coupled node */

CM : {6,7,8,9}                    /* current-mirroring node */

** Recognized Primitive Cells **
SC-IN(n) : {M1,M2}                  /* input differential pair */

CM(n) : {M9,M10}                    /* current mirror */
CM(n) : {M11,M12}
CM(p) : {M3,M5,M7}
CM(p) : {M4,M6,M8}
```

Fig. 4.5 (continued)
(c) Recognition output file.

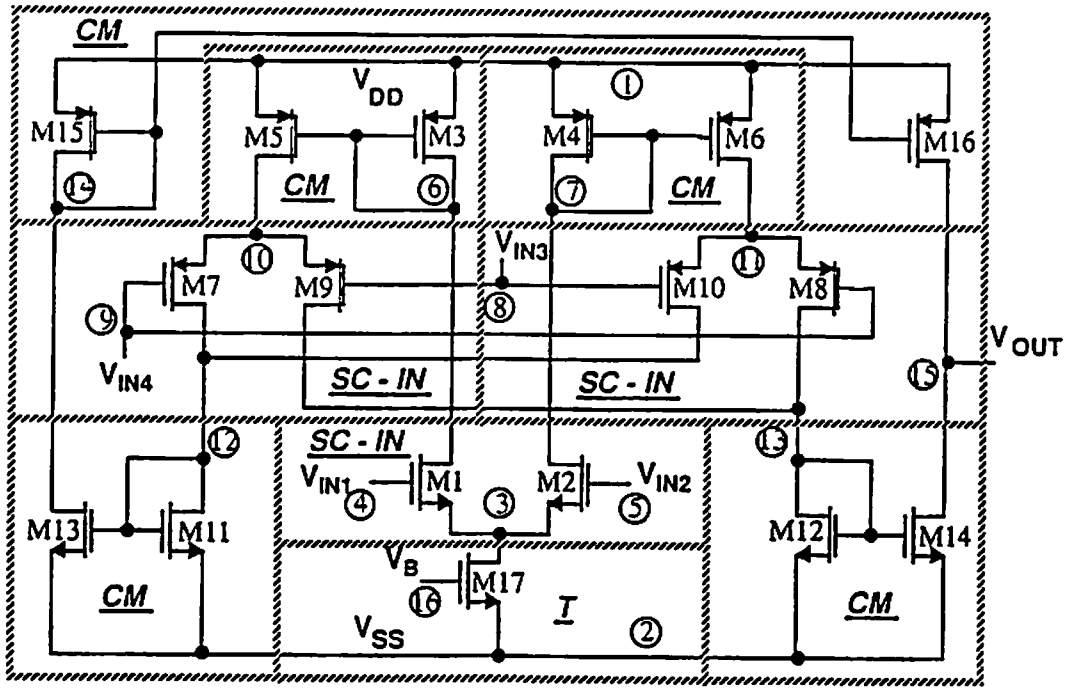


Fig. 4.6 Circuit recognition of an analog multiplier.
(a) Schematic of recognized circuit.

```
*** Example 4 - Analog Multiplier ***
*
* Input/Gain Stage
*
M1   6  4  3  2  N
M2   7  5  3  2  N
M3   6  6  1  1  P
M4   7  7  1  1  P
M5  10  6  1  1  P
M6  11  7  1  1  P
M7  12  9 10  1  P
M8  13  9 11  1  P
M9  13  8 10  1  P
M10 12  8 11  1  P
M11 12 12  2  2  N
M12 13 13  2  2  N
*
* Output Stage
*
M13 14 12  2  2  N
M14 15 13  2  2  N
M15 14 14  1  1  P
M16 15 14  1  1  P
*
VBIAS 16 0
*
VDD 1 0 +5V
VSS 2 0 0V
*
VIN1  4 0
VIN2  5 0
VIN3  8 0
VIN4  9 0
*
VOUT 15 0
*
```

Fig. 4.6 (continued)
(b) SPICE input file.

```
*** Example 4 - Analog Multiplier ***

** Basic Circuit Nodes **
P  : {1,2}
I  : {4,5,8,9}
O  : {15}
B  : {16}
C  : {}

** Critical Junction Nodes **
DC : {6,7,12,13,14}                    /* diode-connected node */

HI : {15}                              /* high-impedance node */

SC : {3,10,11}                         /* source-coupled node */

CM : {6,7,12,13,14}                    /* current-mirroring node */

** Recognized Primitive Cells **
SC-IN(n) : {M1,M2}                     /* input differential pair */
SC-IN(p) : {M7,M9}
SC-IN(p) : {M8,M10}

CM(n) : {M11,M13}                      /* current mirror */
CM(n) : {M10,M12,M14}
CM(p) : {M3,M5}
CM(p) : {M4,M6}
CM(p) : {M15,M16}

T(n) : {M17}                           /* single transistor */
```

Fig. 4.6 (continued)
(c) Recognition output file.

applying the appropriate recognition rules, a total of nine primitive cells including all the matched differential pairs and current mirrors are recognized from the input netlist, as shown in Figs. 4.6(b) and (c).

Example 5 -- Voltage-Controlled Oscillator (VCO)

Figure 4.7(a) shows the schematic of a CMOS VCO [6], which is a key circuit module for analog phase-locked loop subsystems. The oscillator function is implemented by transistors M_{13}-M_{16} and capacitor C_{osc}. A comparator with hysteresis, which consists of transistors M_1-M_{12}, is used to assist the voltage control. The frequency of the oscillator is actually controlled by the input voltage, V_{in}. Given the input netlist (Fig. 4.7(b)) of the VCO, a total of ten primitive cells can be identified along with the critical circuit node information as given in Fig. 4.7(c).

Example 6 -- Sense Amplifier

Figure 4.8(a) shows the schematic of a high-speed sense amplifier [60] based on a half-V_{DD} bit-line sensing scheme [61], which is commonly used in modern CMOS DRAM designs. The sense amplifier consists of an NMOS cross-coupled latch (M_1 and M_2) and a PMOS cross-coupled pair (M_3 and M_4) with a pair of switch transistors inserted in between. By using the analog circuit recognition rules described, the latch can be recognized as a source-coupled pair primitive and the switch-transistor pair can be recognized as a common-gate device primitive (Fig. 4.8(b) and (c)). Note that by identifying these devices as matched pairs, the mismatch between two halves of the sense amplifier can be minimized in the layout and thus can achieve better sensitivity performance.

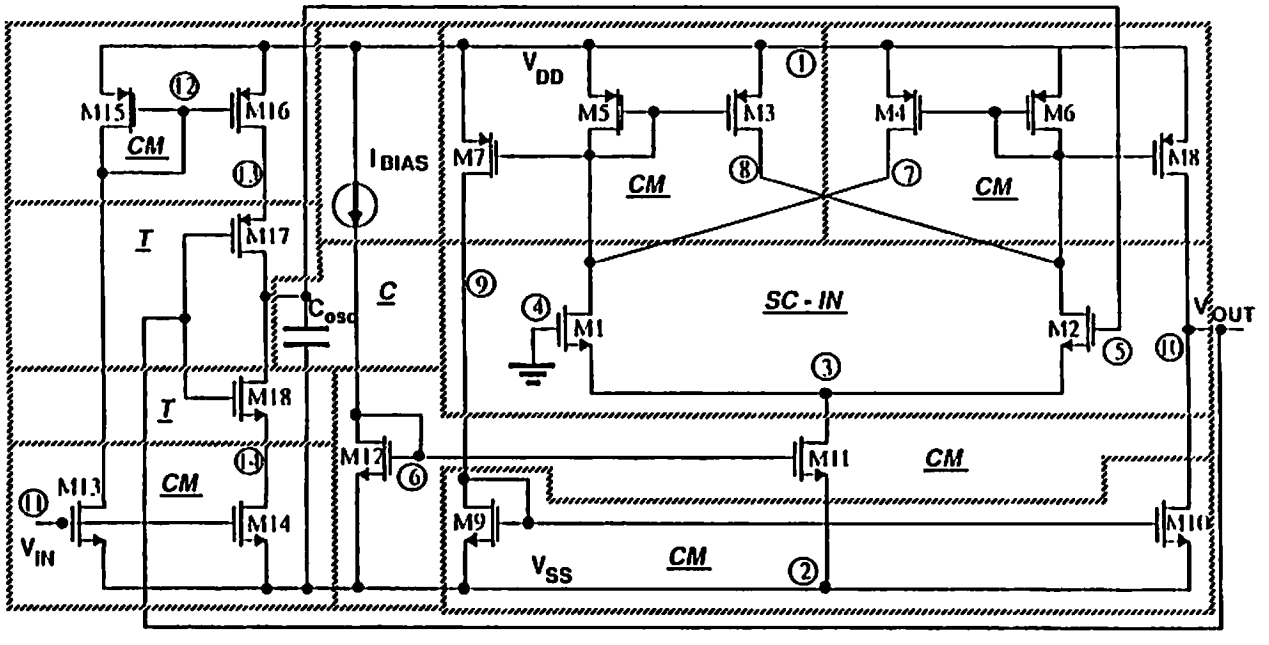


Fig. 4.7 Circuit recognition of a voltage-controlled oscillator.
(a) Schematic of recognized circuit.

```
*** Example 5 - Voltage-Controlled Oscillator ***
*
* Comparator
*
M1   7  4  3  2  N W=30U  L=2U
M2   8  5  3  2  N W=30U  L=2U
M3   8  7  1  1  P W=20U  L=4U
M4   7  8  1  1  P W=20U  L=4U
M5   7  7  1  1  P W=30U  L=4U
M6   8  8  1  1  P W=30U  L=4U
M7   9  7  1  1  P W=60U  L=4U
M8  10  8  1  1  P W=60U  L=4U
M9   9  9  2  2  N W=60U  L=4U
M10 10  9  2  2  N W=60U  L=4U
M11  3  6  2  2  N W=40U  L=4U
M12  6  6  2  2  N W=20U  L=4U
*
* Oscillator
*
M13 12 11  2  2  N
M14 14 11  2  2  N
M15 12 12  1  1  P
M16 13 12  1  1  P
M17  5 10 13  1  P
M18  5 10 14  2  N
*
COSC 5 2
*
IBIAS 1 6 20U
*
VDD 1 0 5V
VSS 2 0 0V
VGND 4 0 2.5V
*
VIN 11 0
*
VOUT 10 0
*
```

Fig. 4.7 (continued)
(b) SPICE input file.

```
*** Example 5 - Voltage-Controlled Oscillator ***

** Basic Circuit Nodes **
P  : {1,2}
I  : {11}
O  : {10}
B  : {4,6}
C  : {5}

** Critical Junction Nodes **
DC : {6,7,8,9,12}                    /* diode-connected node */

HI : {10}                            /* high-impedance node */

SC : {3}                             /* source-coupled node */

CM : {6,7,8,9,11,12}                 /* current-mirroring node */

** Recognized Primitive Cells **
SC-IN(n) : {M1,M2}                     /* input differential pair */

CM(n) : {M9,M10}                       /* current mirror */
CM(n) : {M11,M12}
CM(n) : {M13,M14}
CM(p) : {M3,M5,M7}
CM(p) : {M4,M6,M8}
CM(p) : {M15,M16}

T(n) : {M18}                         /* single transistor */
T(p) : {M17}

C : {COSC}                           /* single capacitor */
```

Fig. 4.7 (continued)
(c) Recognition output file.

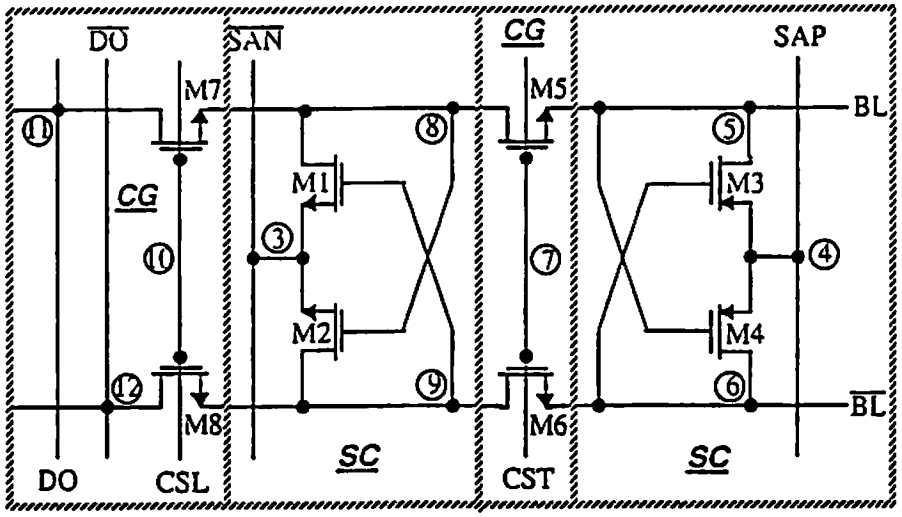


Fig. 4.8 Circuit recognition of a sense amplifier.
(a) Schematic of recognized circuit.

```
*** Example 6 - Sense Amplifier ***
*
*
M1   8  9  3  2  N
M2   9  8  3  2  N
M3   5  6  4  1  P
M4   6  5  4  1  P
M5   8  7  5  2  N
M6   9  7  6  2  N
M7  11 10  8  2  N
M8  12 10  9  2  N
*
VDD 1 0 +5V
VSS 2 0 0V
*
VBL 5 0
VBLB 6 0
VSANB 3 0
VSAP 4 0
VCSL 10 0
VPHIT 7 0
VDQ 11 0
VDQB 12 0
*
```

Fig. 4.8 (continued)
(b) SPICE input file.

```
*** Example 6 - Sense Amplifier ***

** Basic Circuit Nodes **
P  : {1,2}
I  : {}
O  : {}
B  : {}
C  : {}

** Critical Junction Nodes **
DC : {}                          /* diode-connected node */

HI : {}                          /* high-impedance node */

SC : {3,4}                       /* source-coupled node */

CM : {}                          /* current-mirroring node */

** Recognized Primitive Cells **
SC(n) : {M1,M2}                  /* differential pair */
SC(p) : {M3,M4}

CG(n) : {M5,M6}                  /* common-gate devices */
CG(n) : {M7,M8}
```

Fig. 4.8 (continued)
(c) Recognition output file.

Example 7 -- Fully-Differential CMOS Op-Amp

Figure 4.9(a) shows the schematic of a fully-differential CMOS Op-Amp [62] featuring a single-stage, class-AB amplifier with chopper-stabilized differential inputs/outputs and a dynamic common-mode feedback (CMFB) circuit. This complicated Op-Amp is a useful building block for low-power and high-performance analog CMOS circuit applications. The circuit structure of the Op-Amp can be partitioned into the input stage (M_1-M_{12}), output branch (M_{13}-M_{20}), dynamic bias (M_{21}-M_{28}), dynamic CMFB (M_{29}-M_{42} and C_1-C_4), input choppers (M_{43}-M_{46}), and output choppers (M_{47}-M_{54}). Note that this fully differential circuit is symmetric with respect to the center line of the schematic. After recognizing all the primitives in the circuit as in the single-ended cases, the recognizer searches for the identical primitive pairs in the recognized list and further partitions the differential primitive pairs into two symmetrical primitive groups. Given the input netlist shown in Fig. 4.9(b) and with additional rules and primitives required, all the primitives in this circuit example can be recognized as shown in Fig. 4.9(c). The recognized differential circuit primitive pairs are to be placed symmetrically during the physical layout generation to preserve the matched circuit performance.

4.2 Critical Net Analysis

The accuracy requirements of analog placement and routing are generally more extensive than those for digital design [42]. One main challenge for analog circuits is net parasitics on critical circuit nodes which could directly affect the small-signal and transient characteristics of the circuit performance. The

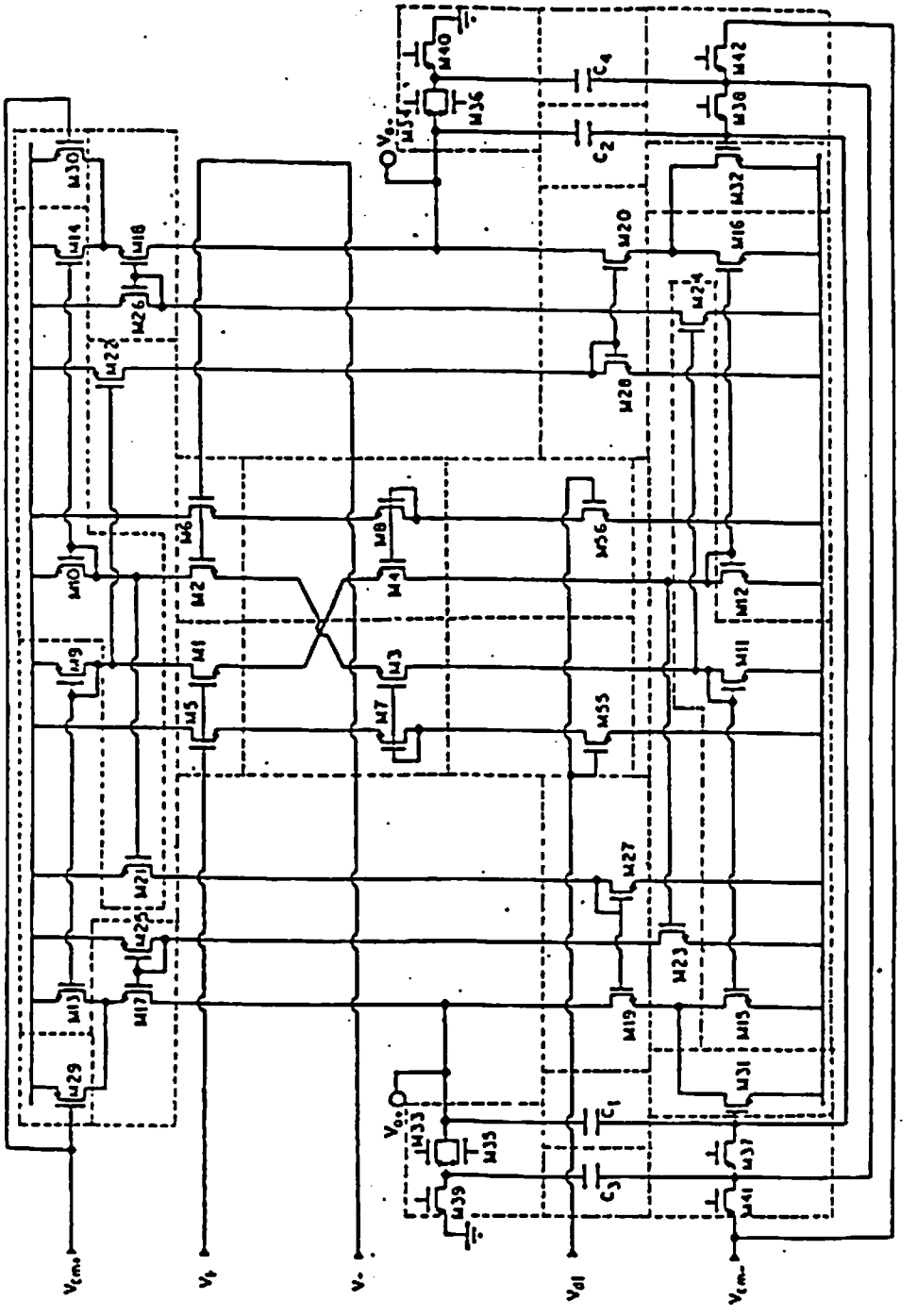


Fig. 4.9 Circuit recognition of a fully-differential CMOS Op-Amp.
(a) Schematic of recognized circuit.

```
*** Example 7 - Fully Differential CMOS Opamp ***
*
*  Chopper-stabilized, single-stage, dynamically-biased
*  cascode, class AB fully differential opamp
*  with dynamic CMFB
*
*  Input Stage
*
M1   5  1  6  6  N W=20U  L=2U
M2  25 21 26 26  N W=20U  L=2U
M3   4  3 26 40  P W=20U  L=2U
M4  24 23  6 40  P W=20U  L=2U
M5  40  1  2  2  N W=20U  L=2U
M6  40 21 22 22  N W=20U  L=2U
M7   3  3  2 40  P W=20U  L=2U
M8  23 23 22 40  P W=20U  L=2U
M9   5  5 40 40  P W=8U   L=2U
M10 25 25 40 40  P W=8U   L=2U
M11  4  4 50 50  N W=4U   L=2U
M12 24 24 50 50  N W=4U   L=2U
*
*  Output Branch
*
M13  8  5 40 40  P W=10U L=2U
M14 28 25 40 40  P W=10U L=2U
M15  7  4 50 50  N W=5U  L=2U
M16 27 24 50 50  N W=5U  L=2U
M17 10  9  8 40  P W=10U L=2U
M18 30 29 28 40  P W=10U L=2U
M19 10 11  7  7  N W=4U  L=2U
M20 30 31 27 27  N W=4U  L=2U
*
*  Dynamic Bias
*
M21 11 25 40 40  P W=10U L=2U
M22 31  5 40 40  P W=10U L=2U
M23  9 24 50 50  N W=4U  L=2U
M24 29  4 50 50  N W=4U  L=2U
M25  9  9 40 40  P W=4U  L=4U
M26 29 29 40 40  P W=4U  L=4U
M27 11 11 50 50  N W=4U  L=10U
```

Fig. 4.9 (continued)
(b) SPICE input file.

```
M28 31 31 50 50  N W=4U  L=10U
*
*  Dynamic CMFB
*
M29  8 61 40 40  P W=12U L=2U
M30 28 61 40 40  P W=12U L=2U
M31  7 63 50 50  N W=4U  L=2U
M32 27 63 50 50  N W=4U  L=2U
M33 15 56 16 50  N W=4U  L=2U
M34 36 56 35 50  N W=4U  L=2U
M35 15 57 16 40  P W=4U  L=2U
M36 36 57 35 40  P W=4U  L=2U
M37 63 56 64 50  N W=4U  L=2U
M38 64 56 63 50  N W=4U  L=2U
M39 16 55 45 50  N W=4U  L=2U
M40 45 55 36 50  N W=4U  L=2U
M41 64 55 62 50  N W=4U  L=2U
M42 62 55 64 50  N W=4U  L=2U
C1 15 63 1P
C2 35 63 1P
C3 16 64 0.5P
C4 36 64 0.5P
*
*  Input Choppers
*
M43  1 52 14 40  P W=4U  L=2U
M44 34 52 21 40  P W=4U  L=2U
M45 21 51 14 40  P W=4U  L=2U
M46 34 51  1 40  P W=4U  L=2U
*
*  Output Choppers
*
M47 10 54 15 50  N W=4U  L=2U
M48 35 54 30 50  N W=4U  L=2U
M49 10 52 15 40  P W=4U  L=2U
M50 35 52 30 40  P W=4U  L=2U
M51 30 53 15 50  N W=4U  L=2U
M52 35 53 10 50  N W=4U  L=2U
M53 30 51 15 40  P W=4U  L=2U
M54 35 51 10 40  P W=4U  L=2U
*
```

Fig. 4.9 (continued)
(b) SPICE input file (continued).

```
*  Input Stage Bias
*
M55  3 41 50 50  N W=4U   L=2U
M56 23 41 50 50  N W=4U   L=2U
VBDF 41 45
*
*  CMFB Bias
*
VBCM+ 61 45
VBCM- 62 45
*
*  Power Supplies
*
VDD 40 0 +2.5V
VSS 50 0 -2.5V
VBAG 45 0 0V
*
*  Clock Signals
*
VKP1 55 0
VKP2 56 0
VKP2B 57 0
VKPA 51 0
VKPAB 53 0
VKPB 52 0
VKPBB 54 0
*
*  Input Signals
*
VIN+ 14 45
VIN- 34 45
*
*  Output Load
*
CL1 15 45 2P
CL2 35 45 2P
*
```

Fig. 4.9 (continued)
(b) SPICE input file (continued).

```
*** Example 7 - Fully Differential CMOS Opamp ***

** Basic Circuit Nodes **
P  : {40,50}
I  : {14,34}
IS : {1,21}
O  : {15,35}
OS : {10,16,30,36}
B  : {41,45,61,62}
K  : {51,52,53,54,55,56,57}
C  : {15,16,35,36,63,64}

** Critical Junction Nodes **
DC : {3,4,5,9,11,23,24,25,29,31}        /* diode-connected node */

HI : {10,30}                            /* high-impedance node */

SC : {40,50}                            /* source-coupled node */

CM : {4,5,24,25,61,63}                  /* current-mirroring node */

** Recognized Primitive Cells (before differential partition) **
P2-3n : {M11,M15,M24},                  /* CM-DC(n) */
        {M12,M16,M23}
P2-3p : {M9,M13,M22},                   /* CM-DC(p) */
        {M10,M14,M21}
P2-4n : {M55,M56}                       /* CM-B(n) */
P2-4p : {M29,M30}                       /* CM-B(p) */
P2-5n : {M31,M32}                       /* CM-SC(n) */
P3-1n : {M1,M5},                        /* CG-IN(n) */
        {M2,M6}
P3-1p : {M3,M7},                        /* CG-IN(p) */
        {M4,M8}
P3-2n : {M19,M27},                      /* CG-DC(n) */
        {M20,M28}
P3-2p : {M17,M25},                      /* CG-DC(p) */
        {M18,M26}
P5    : {M33,M35},                      /* TG */
        {M34,M36}
P5-2  : {M47,M49}, {M51,M53},           /* TG-OUT */
        {M48,M50}, {M52,M54}
```

Fig. 4.9 (continued)
(c) Recognition output file.

```
P6-3n : {M37}, {M39}, {M41},      /* T-SW(n) */
        {M38}, {M40}, {M42}
P6-4p : {M43}, {M45},             /* T-IS(p) */
        {M44}, {M46}
P7    : {C3}, {C4}                /* C */
P7-1  : {C1}, {C2}                /* C-LD */

** Recognized Primitive Cells (after differential partition) **
* Primitive Group 1 *
P2-3n1   : {M11,M15,M24}            /* CM-DC(n1) */
P2-3p1   : {M9,M13,M22}             /* CM-DC(p1) */
P2-4n1/2 : {M55}                    /* 1/2CM-B(n1) */
P2-4p1/2 : {M29}                    /* 1/2CM-B(p1) */
P2-5n1/2 : {M31}                    /* 1/2CM-SC(n1) */
P3-1n1   : {M1,M5}                  /* CG-IN(n1) */
P3-1p1   : {M3,M7}                  /* CG-IN(p1) */
P3-2n1   : {M19,M27}                /* CG-DC(n1) */
P3-2p1   : {M17,M25}                /* CG-DC(p1) */
P51      : {M33,M35}                /* TG(1) */
P5-21    : {M47,M49}, {M51,M53}     /* TG-OUT(1) */
P6-3n1   : {M37}, {M39}, {M41}      /* T-SW(n1) */
P6-4p1   : {M43}, {M45}             /* T-IS(p1) */
P71      : {C3}                     /* C(1) */
P7-11    : {C1}                     /* C-LD(1) */

* Primitive Group 2 *
P2-3n2   : {M12,M16,M23}            /* CM-DC(n2) */
P2-3p2   : {M10,M14,M21}            /* CM-DC(p2) */
P2-4n2/2 : {M56}                    /* 1/2CM-B(n2) */
P2-4p2/2 : {M30}                    /* 1/2CM-B(p2) */
P2-5n2/2 : {M32}                    /* 1/2CM-SC(n2) */
P3-1n2   : {M2,M6}                  /* CG-IN(n2) */
P3-1p2   : {M4,M8}                  /* CG-IN(p2) */
P3-2n2   : {M20,M28}                /* CG-DC(n2) */
P3-2p2   : {M18,M26}                /* CG-DC(p2) */
P52      : {M34,M36}                /* TG(2) */
P5-22    : {M48,M50}, {M52,M54}     /* TG-OUT(2) */
P6-3n2   : {M38}, {M40}, {M42}      /* T-SW(n2) */
P6-4p2   : {M44}, {M46}             /* T-IS(p2) */
P72      : {C4}                     /* C(2) */
P7-12    : {C2}                     /* C-LD(2) */
```

Fig. 4.9 (continued)
(c) Recognition output file (continued).

other challenge is the undesirable signal crosstalk between a sensitive analog circuit net and a noisy net which could cause a degradation to the circuit dynamic range performance. To address these two layout issues, a critical net analysis technique has been developed. It first conducts a sensitivity analysis for all the nets in the circuit and then performs a net constraint analysis to generate appropriate layout constraints.

4.2.1 Net sensitivity classification

According to the sensitivity to layout constraints, all the recognized circuit nodes can be classified into seven categories: power (P), bias (B), sensitive (S), less-sensitive (LS), insensitive (IS), neutral (N), and noisy (NY) nets, as listed in Table 4.2. The most sensitive nets in an analog circuit are the ones associated with the input nodes which tend to carry small signals. The internal high-impedance nodes and cascode nodes form the less-sensitive nets. The large voltage-swing nodes such as output nodes are classified as insensitive, while digital clock nodes can be quite noisy. The analog ground and other inactive nodes are classified as neutral nets.

4.2.2 Net constraint analysis

Based on the net sensitivity information, the analyzer then assigns net priorities and generates proper layout constraints. A net distance constraint is assigned for each circuit net with different priorities as listed in Table 4.3. The highest priority is denoted by HH and the lowest priority is denoted by LL. The circuit node with the highest-priority distance constraint requires a very short wire connection to result in minimum parasitics. Although the detailed

Table 4.2 Sensitivity classification for analog circuit nets.

Sensitivity classes	Typical examples
Power (P)	analog VDD and VSS
Bias (B)	dc biasing nodes
Sensitive (S)	signal input nodes
Less-Sensitive (LS)	internal high-impedance and cascode nodes
Neutral (N)	analog ground and other inactive nodes
Insensitive (IS)	large signal output nodes
Noisy (NY)	digital clock and control signals

Table 4.3 Priority assignment for analog circuit net distance constraints.

Net	S	LS	IS	N	NY
Priority	HH	H	M	L	LL

Table 4.4 Priority assignment for analog circuit net spacing constraints.

Net1 \ Net2	S	LS	IS	N	NY
S	LL	M	H	L	HH
LS	M	LL	M	L	H
IS	H	M	LL	L	M
N	L	L	L	LL	L
NY	HH	H	M	L	LL

priority assignments may depend on the recognized circuit type and topology, the high-impedance sensitive nodes are normally assigned with the highest priority, while the inactive nodes or the noisy nodes are assigned the lowest.

Next, a prioritized net spacing constraint for each pair of circuit nodes is also generated according to Table 4.4. The spacing constraint between the sensitive and noisy nodes is assigned to a high priority. Therefore, large spacing is reserved for those nodes to minimize the undesirable wire crossover or adjacent wire crosstalk.

The weighted layout constraints are then fed into the constraint-driven floorplanning and routing procedures to generate a high-performance and design-rule-correct analog circuit layout.

Chapter 5

Constraint-Driven Floorplanning and Routing

The physical layout process starts with the floorplanning. Because there exist various physical constraints due to variable shapes and pin locations as well as electrical constraints due to device matching, parasitic, and noise coupling in analog circuit layout, most well-known algorithms developed for digital circuits such as min-cut [63] and simulated annealing [64] are considered rather inconvenient or inefficient for analog circuit floorplanning.

To address the special analog circuit layout requirements, an efficient and constraint-driven analog floorplanning technique based on slicing structures and a macro-cell style has been developed. The macro-cell layout style is used to accommodate the various shape requirements for analog circuit modules. The slicing structure is used to optimize the component shape and to make the routing easier. Based on a zone-sensitivity partitioning algorithm [44], this floorplanner can take the layout constraints from the previous circuit recognition and analysis steps and automatically derive a high-performance slicing tree for the circuit module. In addition, by incorporating Stockmeyer's algorithm [65], this floorplanner can determine the optimal component shape for the circuit module to satisfy the user-specified physical layout size constraints and to achieve high area efficiency. The complete floorplanning and optimization process, as illustrated in Fig. 5.1, can be divided into five steps: 1) top-down slicing tree generation, 2) bottom-up shape constraint estimation, 3) iterative topology refinement, 4) top-down optimal shape selection, and 5) bottom-up primitive cell generation

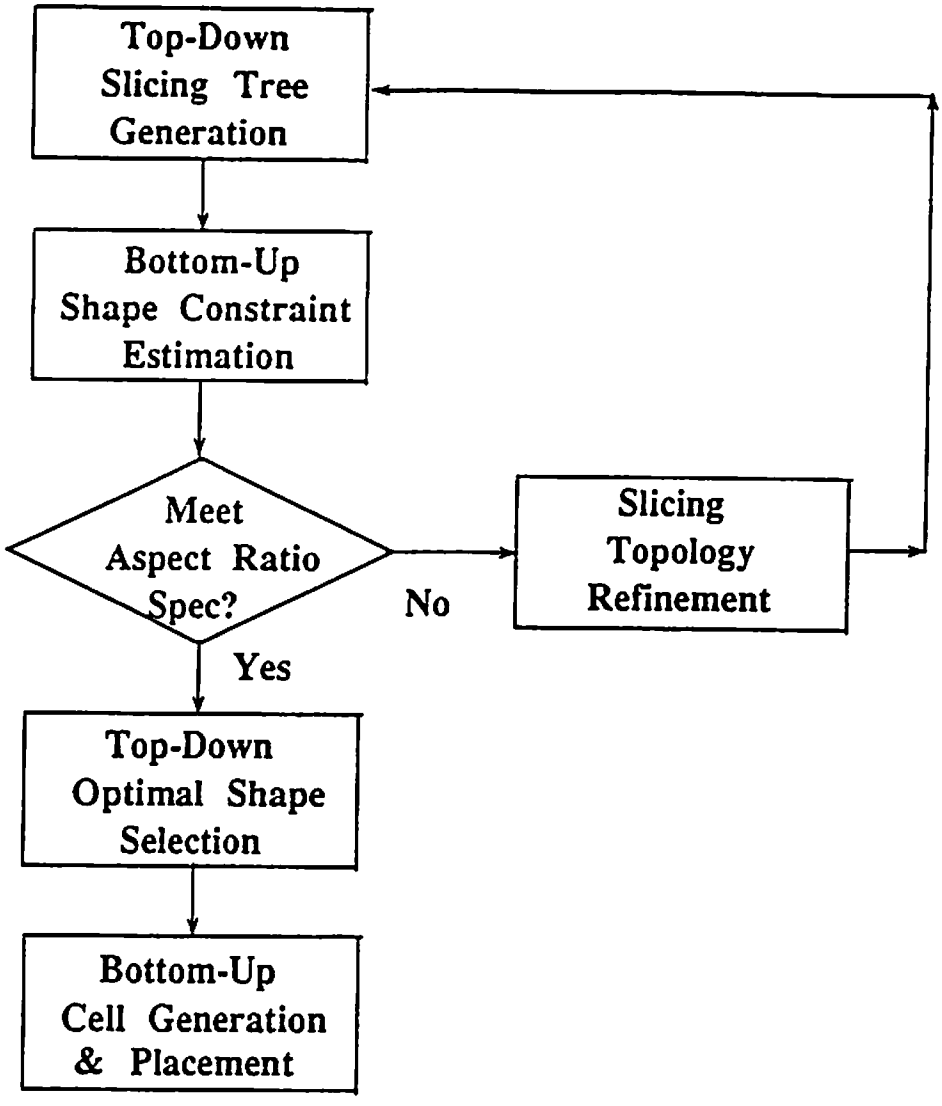


Fig. 5.1 A constraint-driven floorplanning and optimization procedure for analog circuits.

and placement.

5.1 Sensitivity-Based Floorplanning

Given the user-specified circuit module aspect ratio and the circuit schematic netlist, the first step of the floorplanning process is to derive the slicing tree, i.e. to determine relative positions of all the slicing structures in the module in a top-down fashion. Unlike other layout tools where a fixed slicing floorplan is prearranged for each restricted analog circuit topology, this floorplanner is capable of generating high-quality floorplans for a wide variety of analog circuit modules.

The floorplanning starts by selecting an initial vertical slicing topology based on the recognized circuit information obtained from the previous analysis steps. All of the transistor elements in the circuit can be partitioned into 'source' and 'dangling' components. The source components are defined as the transistor elements whose source terminals are directly tied to a power bus, while the dangling elements are the ones that have no direct power connections. The power busses are arranged to run horizontally across the top and bottom of the module. Depending on the circuit complexity level of the module, the initial floorplan is divided into one to four vertical slices shown in Fig. 5.2. In general, one- and two-slice topologies are suitable for simple analog circuit modules without cascode or dangling devices. Analog circuit modules with single cascode are assigned with a three-slice topology, while the circuit with multiple cascodes is started with a four-slice floorplan. Floorplans with five or more vertical slices are seldom used and thus are not incorporated. Based on these

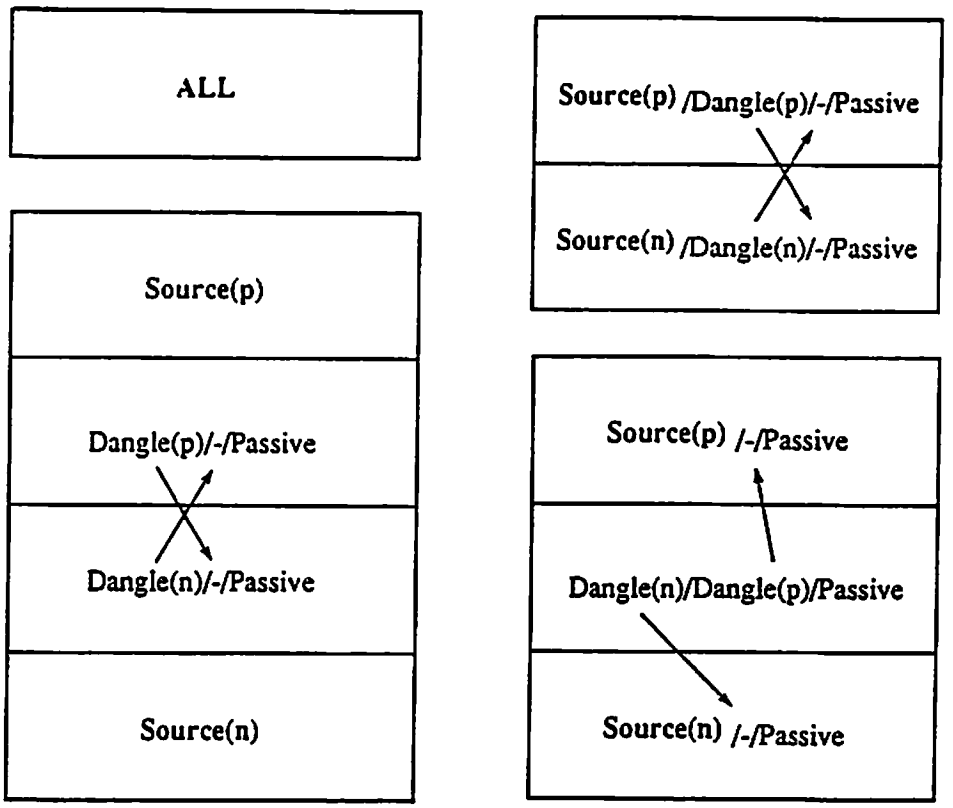


Fig. 5.2 Four possible vertical slicing topology arrangements.

source and dangling properties of the components, all primitive cells in the module are assigned to each vertical slice to satisfy the vertical connectivity constraints. After the initial vertical slicing floorplan is completed, the dangling elements are then allowed to switch between slices (possible moves are indicated by arrows shown in Fig. 5.2) to improve the overall area usage of the floorplan.

In a subsequent slicing step, each vertical slice is further partitioned into horizontal slices. The relative positions between the primitives are mainly determined by the constraints on the net sensitivity and the given terminal positions. As shown in Fig. 5.3, each vertical slice is partitioned into multiple layout zones in accordance with the sensitivity levels of circuit primitives. Unless otherwise constrained by the user-specified terminal locations, the most sensitive zones are assigned to the far left (or right) position, while the most noisy zones are to the opposite side, and with the neutral zones located in the middle. The sensitivity level of each primitive can be calculated as a mean value of all the net sensitivities linked to the primitive. By comparing the sensitivity levels for all the circuit primitives allocated in each vertical slice, the relative positions between the primitives can be determined. The same exact procedure is used to position the devices within the multiple-transistor primitives. After all the positioning movements have been settled down, the resulting layout topology can be assembled into a slicing tree representation. Note that as a result of this zone-sensitivity partitioning, in the final slicing topology the circuit primitives with close sensitivity levels will tend to cluster together, while the components with significantly different sensitivities will be driven away from each other. For instance, the circuit primitives in the input stage of an amplifier will tend to be trapped together

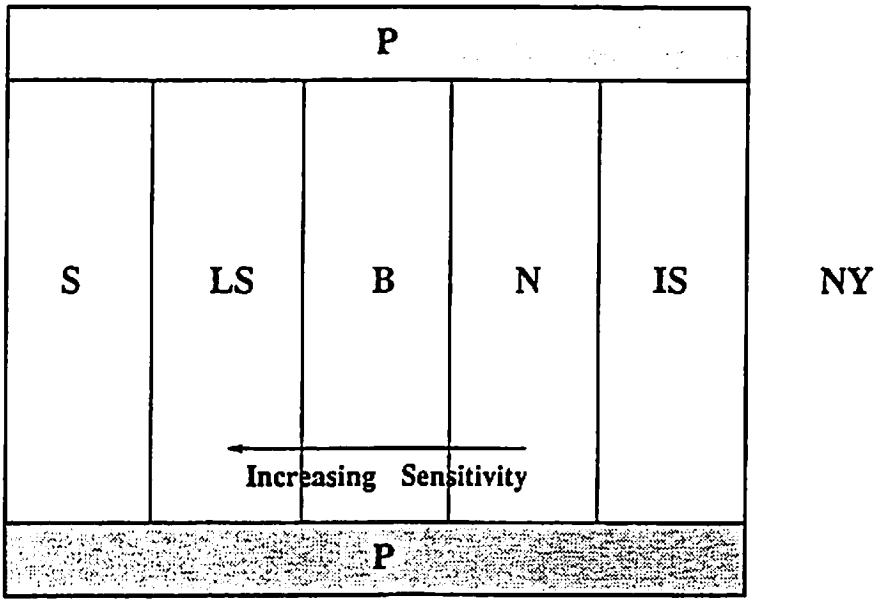


Fig. 5.3 A sensitivity-based horizontal slicing floorplan.

on one side of the floorplan, while the output stage is pulled away to the opposite side. Thus by using this sensitivity-based floorplanning scheme, which has essentially been used by human experts in the analog IC layout practice for years, high-quality analog circuit floorplans can be automatically generated for a wide variety of analog circuit modules.

Additional floorplanning steps are taken to incorporate the symmetrical matching constraints between circuit primitives [45]. For fully differential circuits, the recognized balanced circuit primitives on each half circuit are placed symmetrically with respect to each other. Similarly, for thermal matching considerations, the symmetrical placement of the sensitive circuit primitives can be arranged with respect to the high-current circuit primitives.

5.2 Physical Shape Optimization

After the slicing tree is generated, the problem to be solved is to determine the optimal shape and orientation of the individual circuit primitives based on the user-specified module aspect ratio or size constraints. An elegant algorithm due to Stockmeyer [65] can be used to solve this problem in a polynomial time.

First, a shape constraint relation needs to be defined. Given a rectangular cell with a width equal to X and a height equal to Y, the shape constraint relation R of the cell is given [55] by

$$R = \left\{ (x,y) \mid y \geq h,\ x \geq w \right\} \tag{5.1}$$

as illustrated in Fig. 5.4. The shaded area in the figure represents the set of all

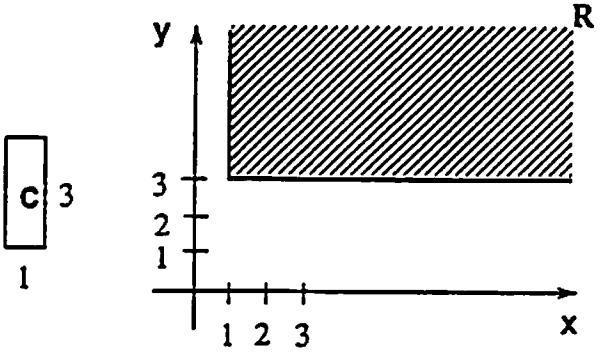


Fig. 5.4 The shape constraint relation for primitive cell C.

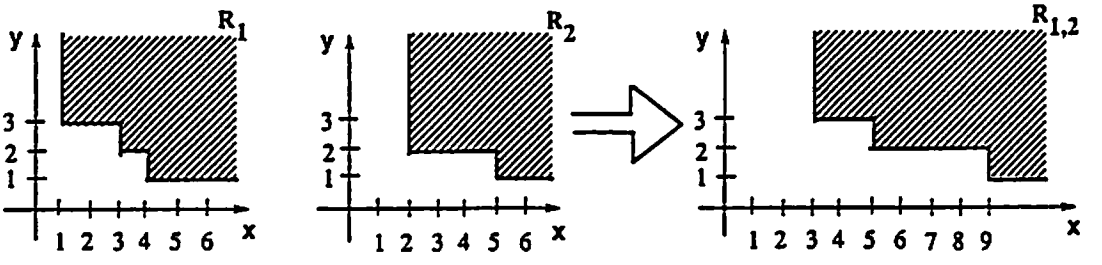


Fig. 5.5 Summing the shape constraint relations for a horizontal assembly.

pairs satisfying Equation 5.1. Given the shape constraint relations of two rectangles, R_1 and R_2, a combined shape relation $R_{1,2}$ can also be obtained for the enclosing rectangle. If the two rectangles are placed side by side, then $R_{1,2}$ can be expressed as

$$R_{1,2} = \Big\{(x,y) \mid (x_1,y) \in R_1,\ (x_2,y) \in R_2,\ \cdots,$$

$$(x_n,y) \in R_n,\ x = x_1 + x_2 + \cdots + x_n\Big\} \tag{5.2}$$

as illustrated in Fig. 5.5. The height of the enclosing rectangle is constrained to inherit the maximum of the heights of the two rectangles, while the width is given by the sum of the widths of the two rectangles. Similarly, a dual process can be applied to position two rectangles on top of each other.

In the shape constraint estimation step, a list of the possible shapes of each primitive is first estimated by calling the corresponding primitive generator with devize size information. Based on this information, each primitive cell can be given a shape constraint relation. Then by using Stockmeyer's algorithm, the shape constraint relations of the primitive cells can be composed bottom-up until the shape constraint of the entire module is obtained at the top of the slicing tree as illustrated in Fig. 5.6. During the bottom-up traversal, the routing area assigned with each cut line is also estimated based on the total number of nets acrossing through and is included as an offset during the composition of the shape constraint relations.

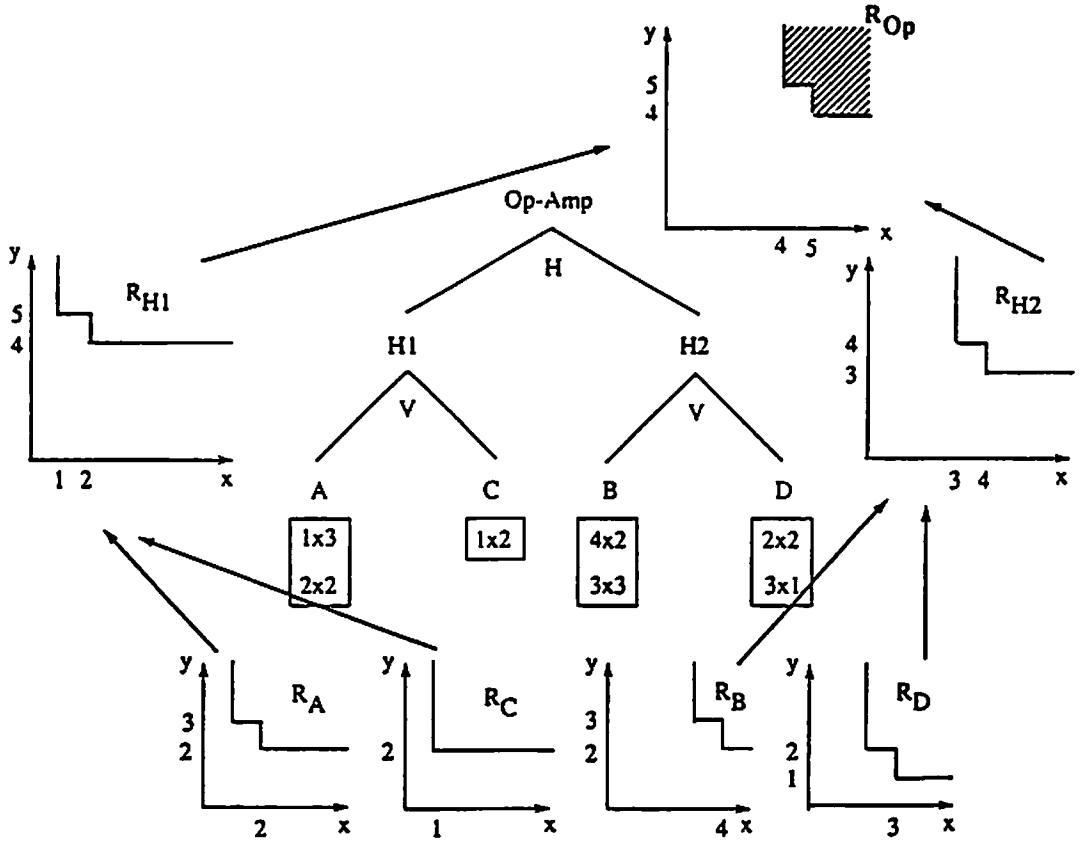


Fig. 5.6 Illustration of bottom-up module shape constraint estimation.

Then in a subsequent, top down module shape selection step, the user-specified aspect ratio or size constraints are converted into actual X- and Y-dimensions of the entire floor. This constraint is then propagated to its descendant nodes until the X- and Y-dimensions of all the primitives are determined as shown in Fig. 5.7. If the composed aspect ratio is not satisfied, an iterative refinement step is also permitted to go back to slightly modify the slicing tree topology for further improvement. The basic idea is that if the composed aspect ratio is turned out too small, then the system will decrease the number of vertical slices in the module. On the other hand, if the composed aspect ratio is too large, then the system will increment the number of vertical slices. Therefore, the optimization process for the aspect ratio and the slicing tree becomes a closed loop, and hence the overall optimal layout can be ensured. Once the final module aspect ratio is determined, this information is passed top-down to each sub-tree and eventually, to each primitive to determine its optimal X and Y dimensions. Finally, the actual layout of the primitives are generated and placed in a bottom-up fashion according to the optimal shape and the final slicing topology obtained.

5.3 Primitive Cell Generation

The primitive cell generators operate in two different modes. First, during the module shape estimation process, they quickly estimate and report to the floorplanner all the possible shapes of the primitive based on the given device size information. Then, after the final floorplan is determined, these cell generators are called again to produce the actual cell layout based on the particular

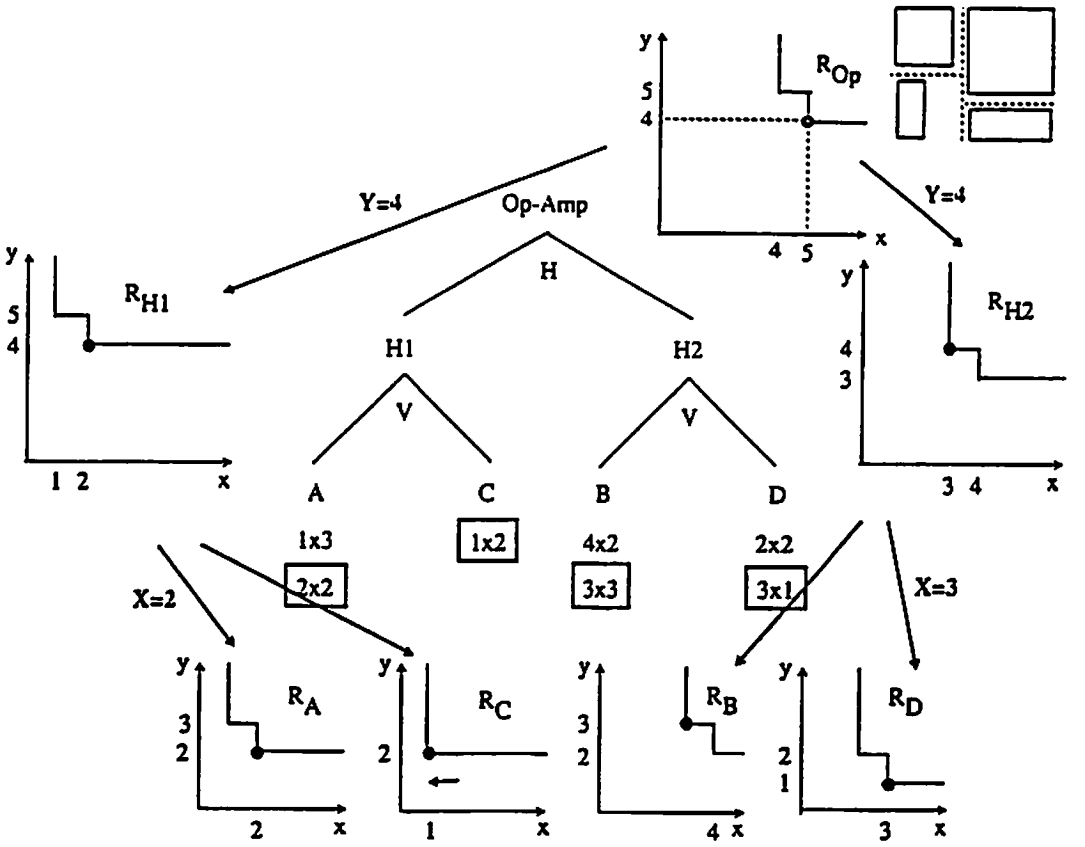


Fig. 5.7 Illustration of top-down optimal primitive shape selection.

shape function assigned.

The detailed layout of the primitive cells is generated in a procedural manner by a set of leaf cell generators. Each generic primitive cell (as shown in Fig. 4.2 for analog MOS circuits) has its own leaf cell generator. All the leaf cell layout generators are parameterizable in transistor type, size, shape, orientation, and number of devices. Thus, given the sized schematic netlist and the recognized primitive cell information, each circuit primitive can be custom-generated based on the individual shape function assigned during floorplanning. Figure 5.8 shows the generated layout examples of source-coupled pair and current mirror primitives with variable shapes. Note that the matched devices in the primitive cells are laid out with the same structure, same shape, same orientation, and minimum distance to achieve the optimum matching. The variable shapes of the primitives are realized with split-geometry comb-like layout structures. This ability to vary the shapes of the cells helps the floorplanner to optimize the area usage and meet the user-specified module shape constraints. Furthermore, each primitive can be constrained by a group matching parameter to allow a balanced thermal or differential matching between selected circuit primitives. All the cell generators are programmed using generic mask layers and design rules which can be driven by any specific technology file supplied externally. Therefore, it permits the layout generator to quickly produce correct mask layout independent on the process technology used.

5.4 Routing

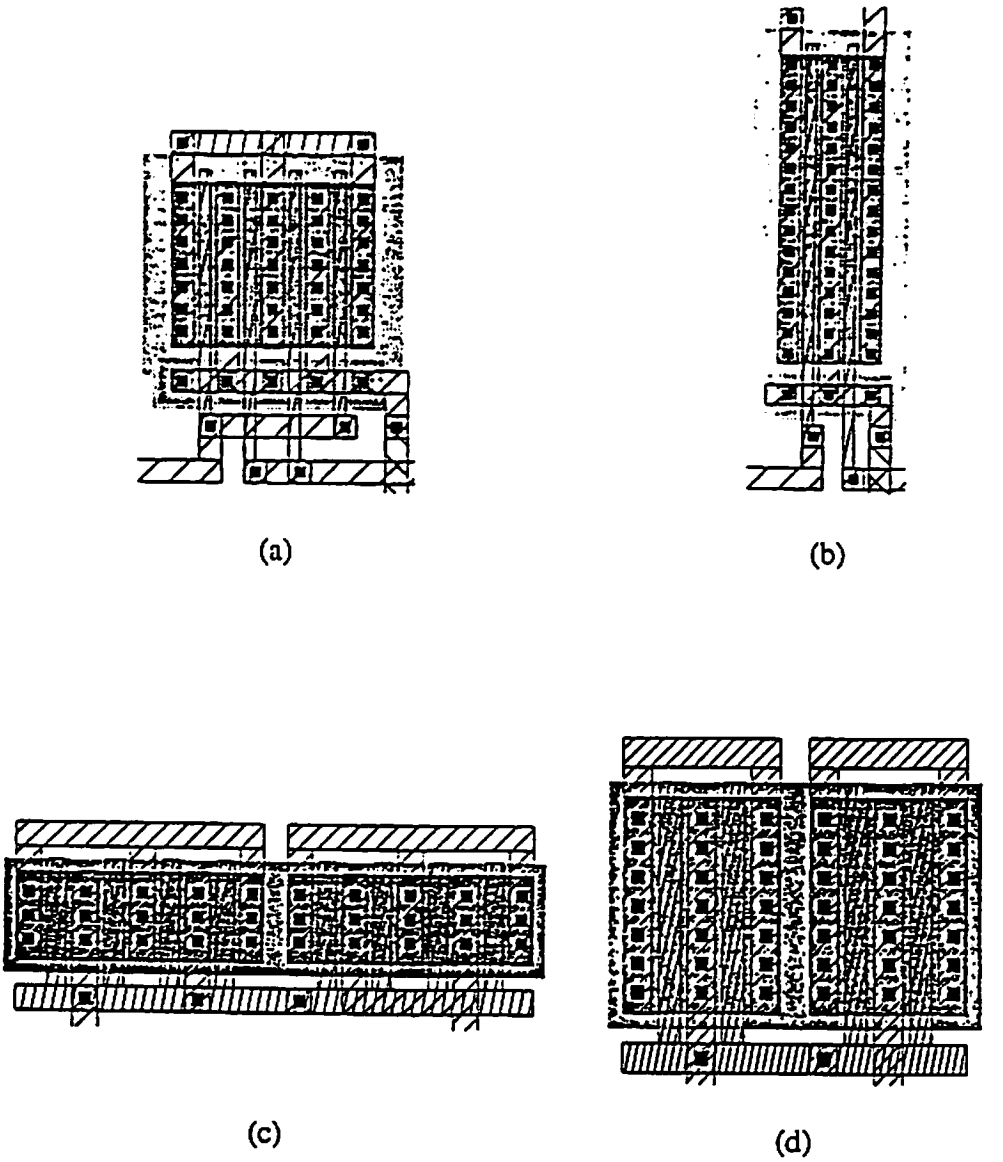


Fig. 5.8 Generated layout examples of primitive cells with variable shapes.
(a) Source-coupled pair with aspect ratio of 0.75.
(b) Source-coupled pair with aspect ratio of 0.3.
(c) Current mirror primitive with aspect ratio of 2.5.
(d) Current mirror primitive with aspect ratio of 1.

The concerns of analog circuit routing are generally more extensive than those of digital circuit routing. One of the main concerns for analog circuits is routing parasitics on some critical circuit nodes which will affect the ac and transient characteristics of the circuit performance. Another concern is the undesirable signal crosstalk between a sensitive net and a noisy net such as a large-swing output signal or a digital clock signal which can cause a degradation to the circuit dynamic range performance. Fortunately, as discussed during floorplanning, one half of these layout constraints have already taken care by the floorplanner which makes the routing easier. In addition, the use of slicing structures also helps to simplify the routing requirements. As a result, a robust switch-box router such as Mighty [66], which can handle obstacles, allow sensitive nets to be routed before other nets, and maximize the use of metal layers, can be used.

A constraint-driven analog circuit routing strategy [42] based on Mighty router is developed. The routing process is implemented in multiple steps. The routing order of the nets in each circuit is driven by the priorities of the net sensitivity constraints generated in the previous analysis step. The sensitive nets such as input signals are usually assigned with the highest priority and get routed first, while the noisy nets such as the large-swing output signals or digital clock signals are the lowest priority and get routed last. The main advantage of this priority-based multiple-step routing method is that the pre-routed nets in the high-priority classes can automatically become the fixed obstacles for the nets in the lower-priority classes, and thus can prevent any undesired signal coupling during the routing process.

5.5 Experimental Results

Based on the described layout synthesis method, a prototype analog circuit module layout generator has been implemented in C. The system runs under UNIX on a Sun 4/60C workstation. The generated layout can be viewed in the Magic environment [67].

A wide variety of analog IC building blocks have been used to demonstrate the layout methodology. The recognition program has been successfully applied to recognize many popular analog CMOS circuit modules including operational amplifiers, comparators, voltage multipliers, voltage-controlled oscillators, and artificial neural networks. For operational amplifiers, the program has been tested for various architectures and circuit topologies including single-stage transconductance amplifier, folded-cascode amplifier, two-stage amplifier, and fully differential amplifiers. Recognition results on a two-stage CMOS Op-Amp, a folded-cascode CMOS Op-Amp, and a voltage comparator are shown in Figs. 5.9(a), 5.10(a), and 5.11(a), respectively, with their corresponding generated slicing floorplans shown in Figs. 5.9(b), 5.10(b), and 5.11(b). The generated layout results of the two Op-Amps are shown in Figs. 5.9(c) and 5.10(c), while the generated comparator layouts with the input aspect ratios of 1 and 1.5 are illustrated in Figs. 5.11(c) and 5.11(d), respectively. In addition, experimental results of a fully-differential CMOS Op-Amp are shown in Fig. 5.12. Note that in these layouts, different slicing topologies and various component shapes are utilized to satisfy the layout constraints. All the matched devices in the circuits are laid out as primitive cells to achieve optimal matching. Moreover, the wire lengths of critical nets such as the input and cascode nets are kept short to

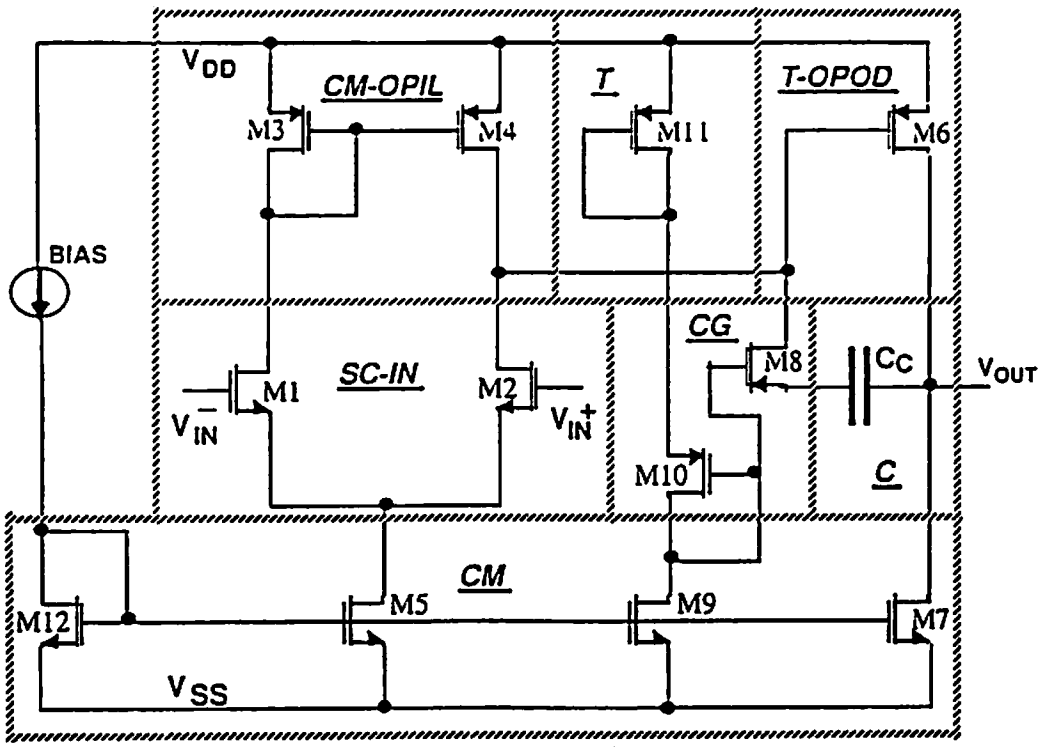


Fig. 5.9 Experimental results of a two-stage CMOS Op-Amp.
(a) Schematic of recognized circuit.

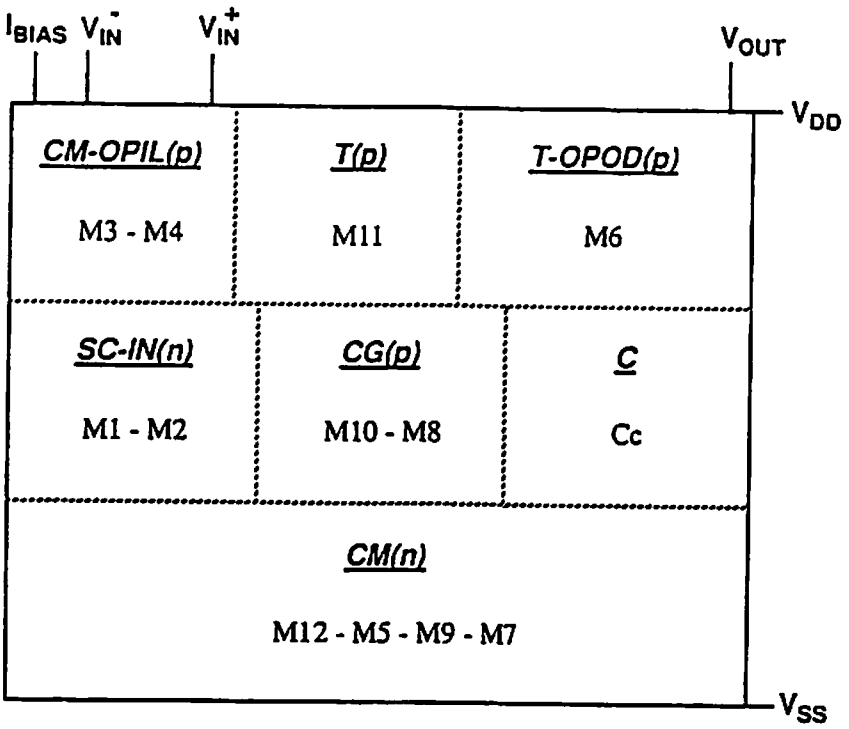


Fig. 5.9 (continued)
(b) Generated slicing floorplan.

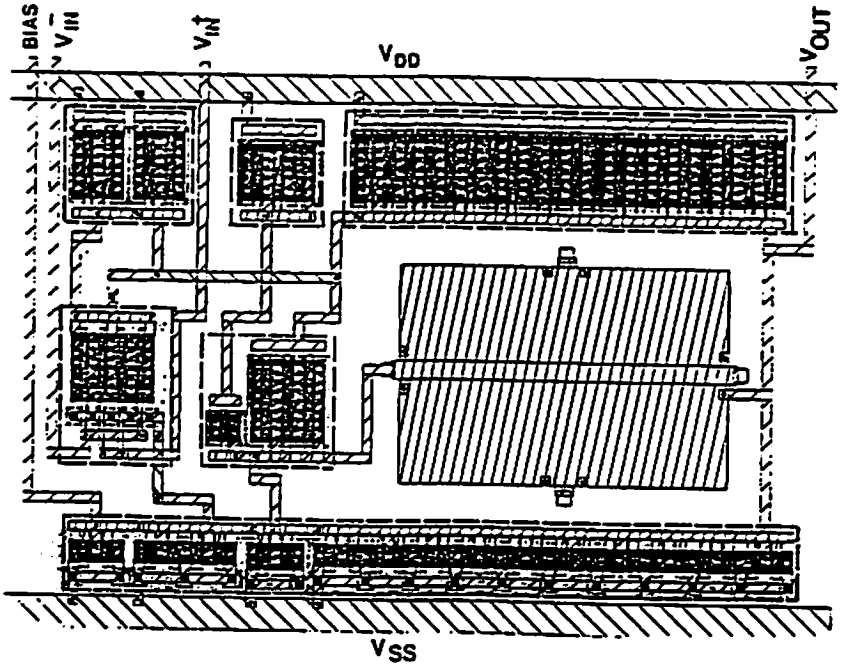


Fig. 5.9 (continued)
(c) Generated layout.

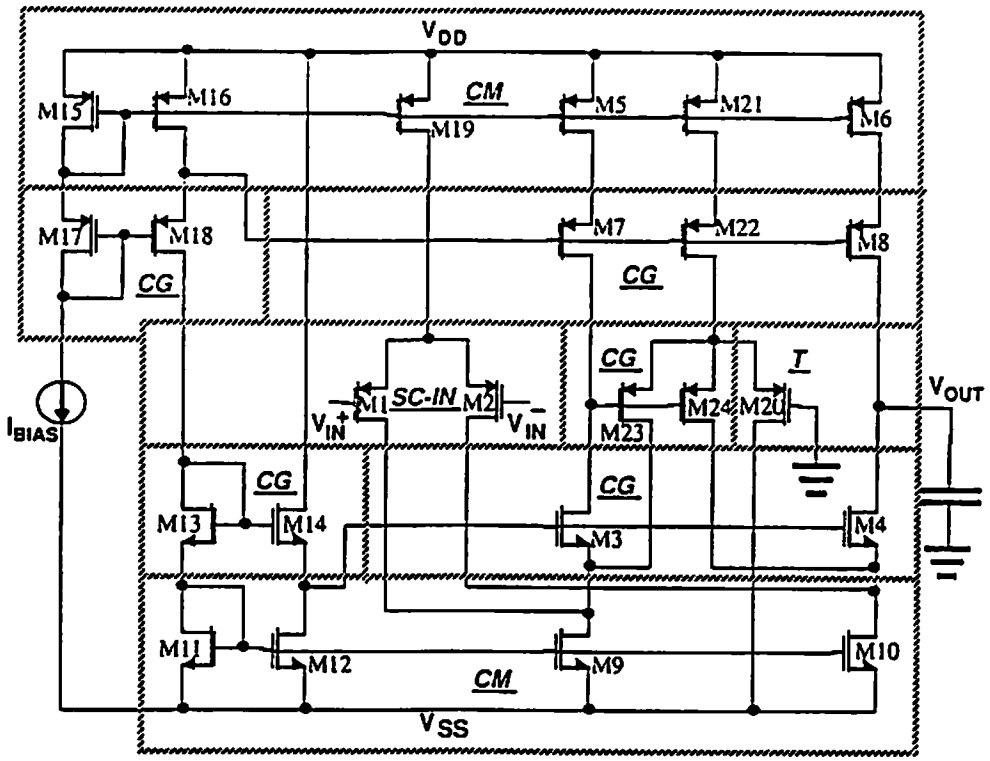


Fig. 5.10 Experimental results of a single-stage folded cascode Op-Amp.
(a) Schematic of recognized circuit.

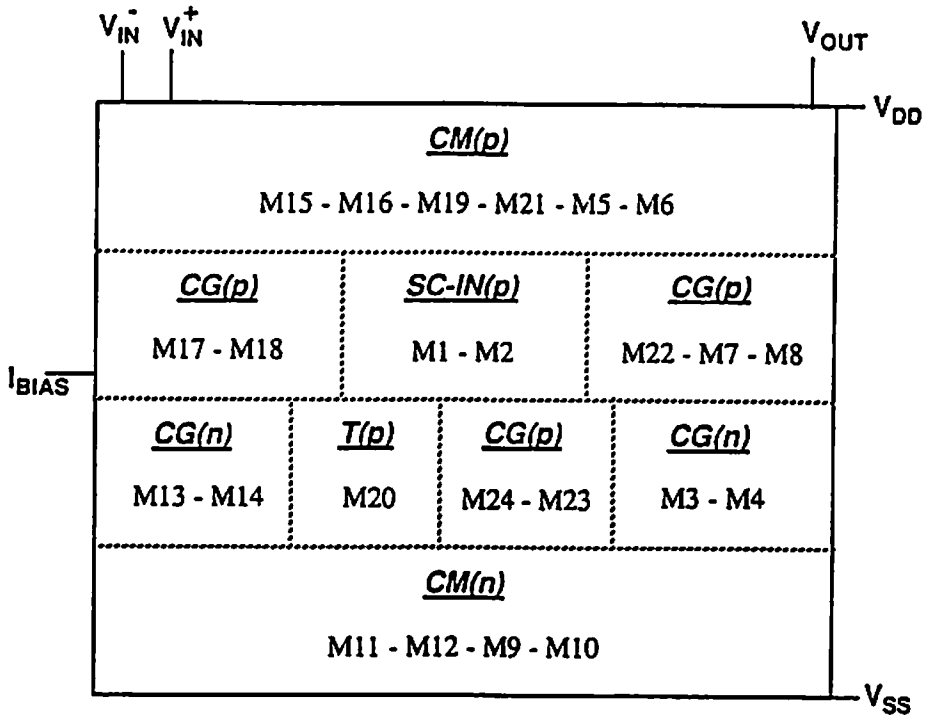


Fig. 5.10 (continued)
(b) Generated slicing floorplan.

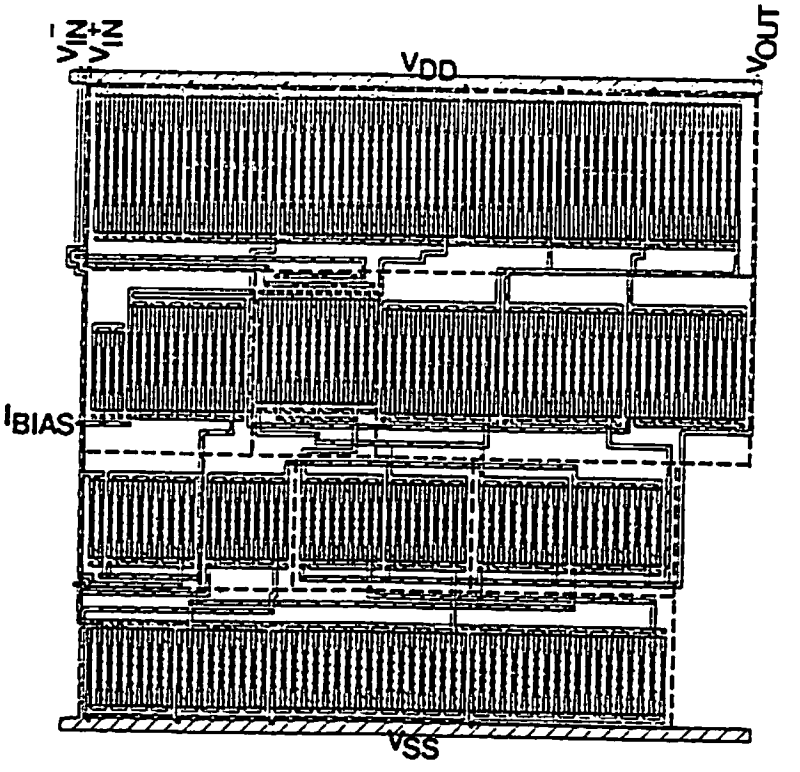


Fig. 5.10 (continued)
(c) Generated layout.

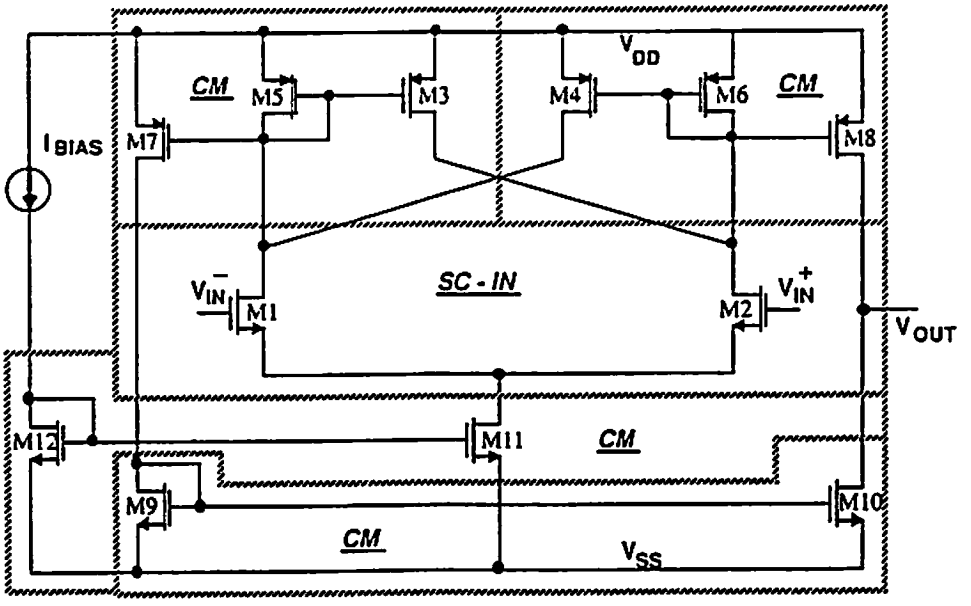


Fig. 5.11 Experimental results of a voltage comparator.
(a) Schematic of recognized circuit.

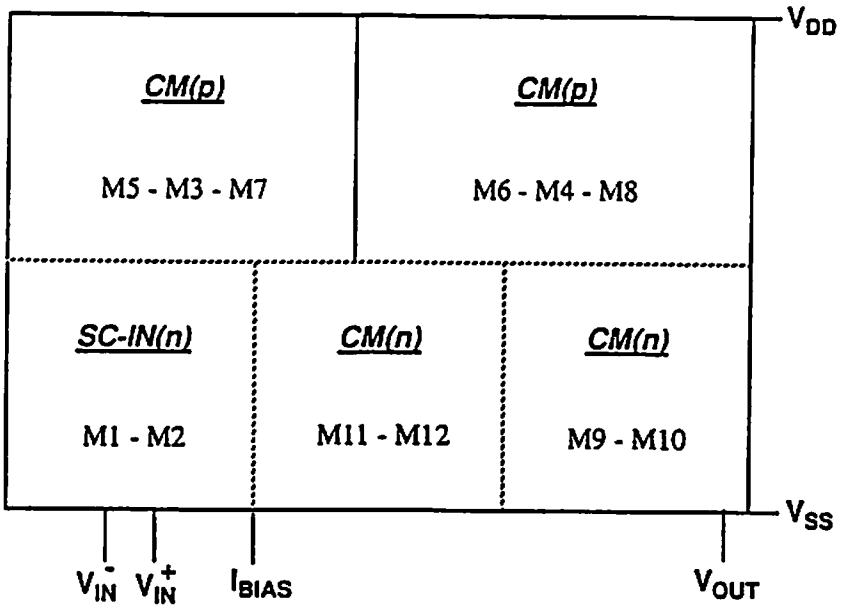


Fig. 5.11 (continued)
(b) Generated slicing floorplan.

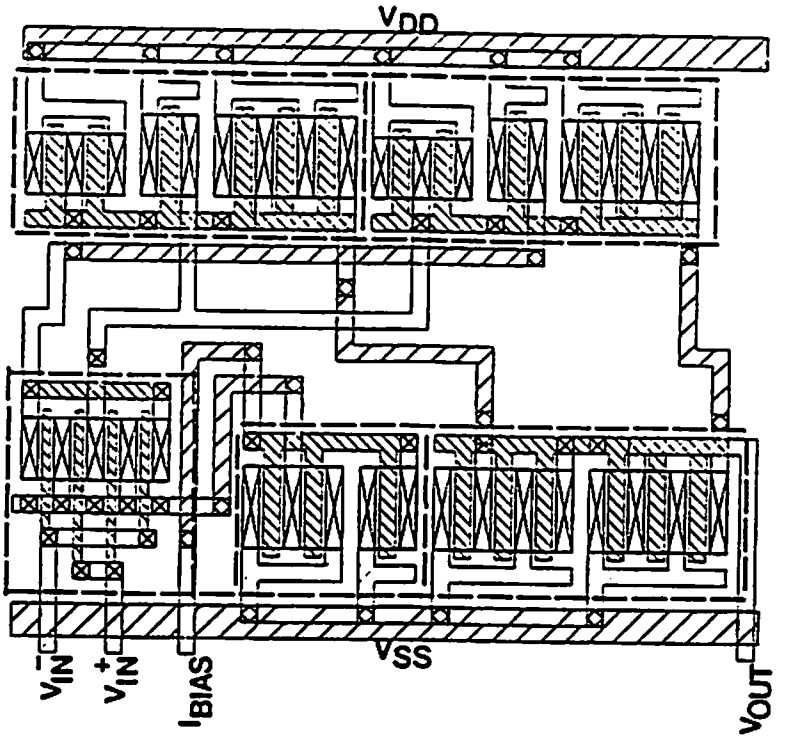


Fig. 5.11 (continued)
(c) Generated layout with input aspect ratio of 1.

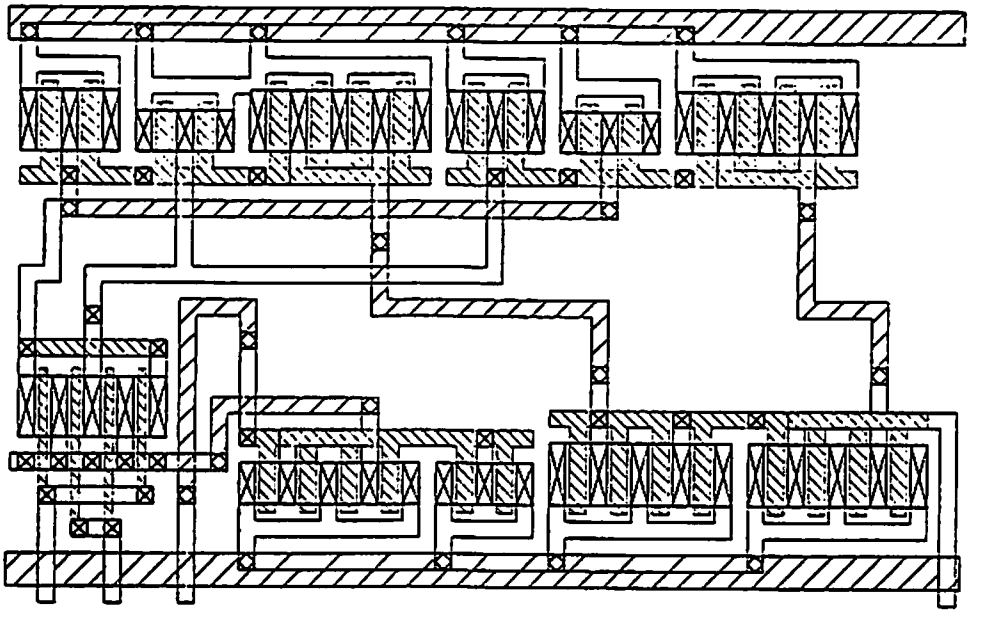


Fig. 5.11 (continued)
(d) Generated layout with input aspect ratio of 1.5.

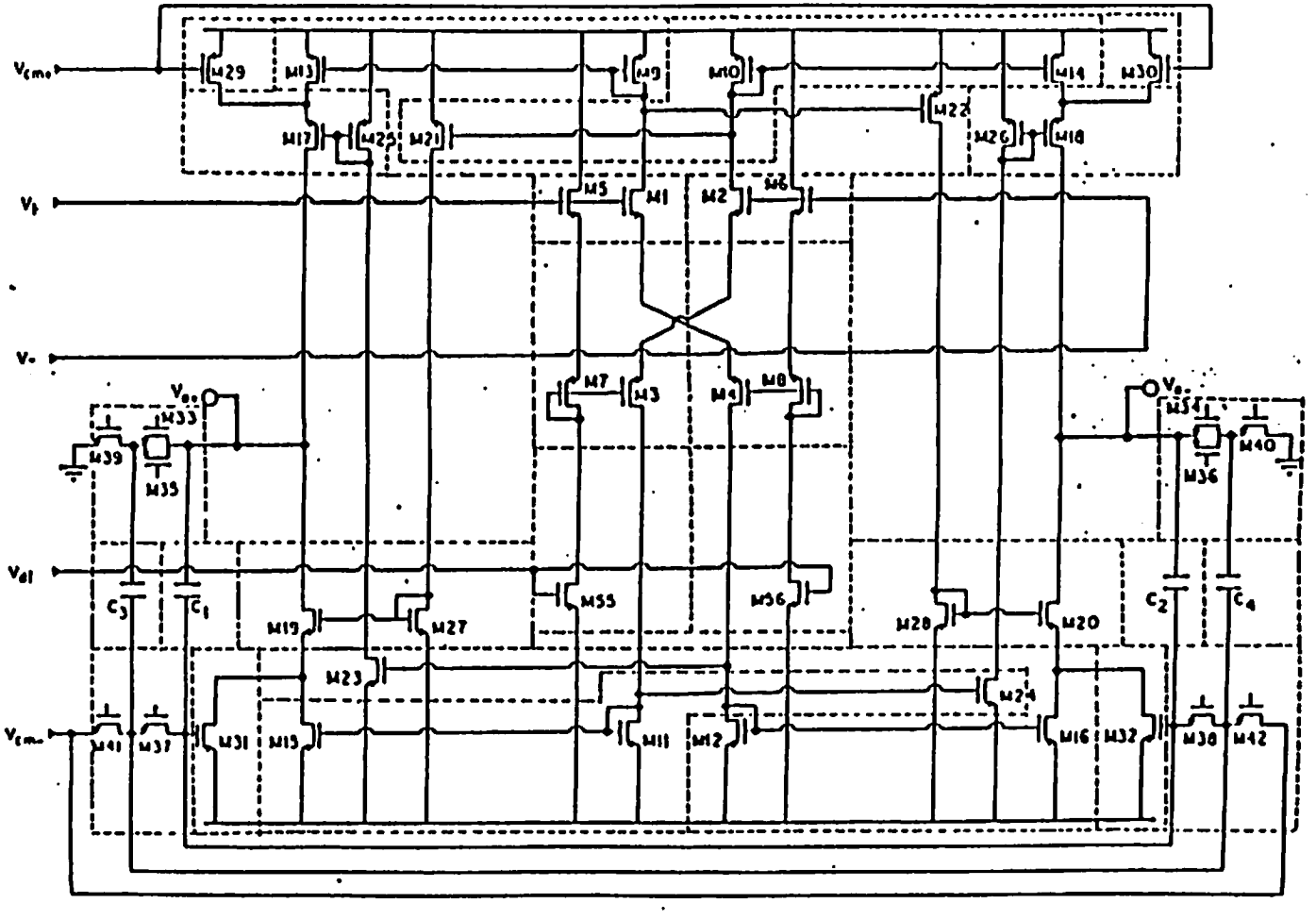


Fig. 5.12 Experimental results of a fully-differential CMOS Op-Amp.
(a) Schematic of recognized circuit.

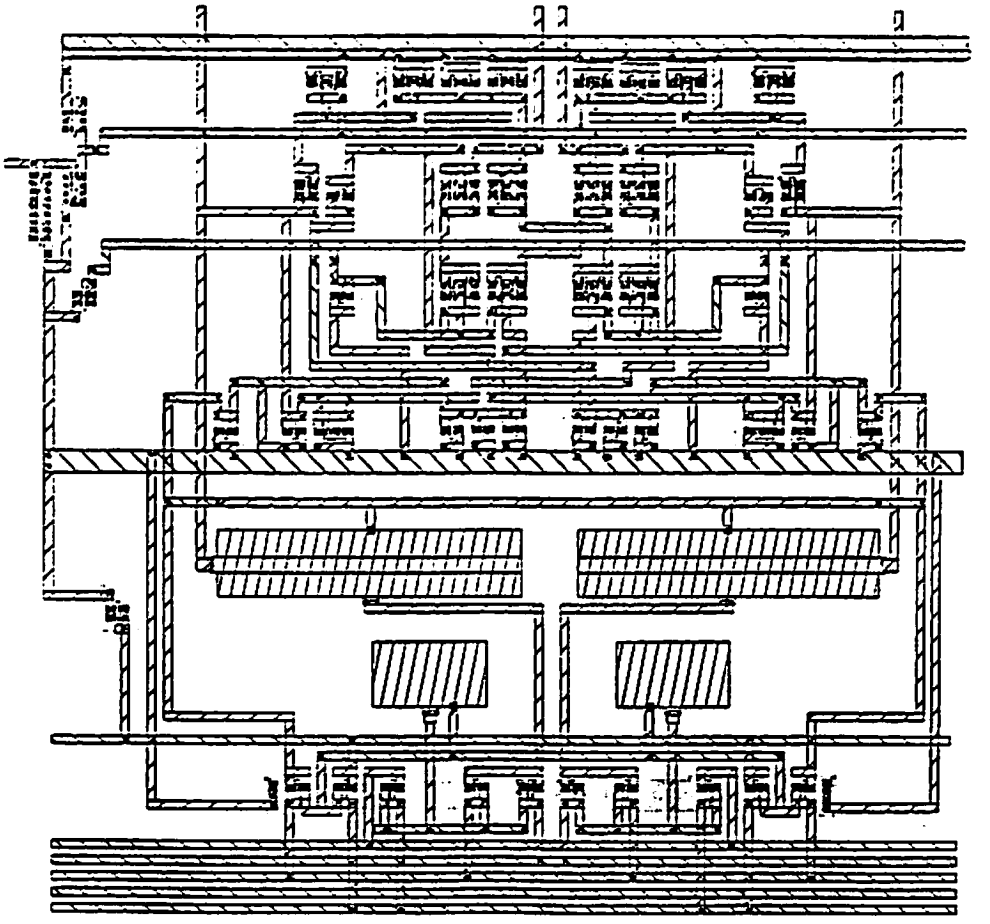


Fig. 5.12 (continued)
(b) Generated layout.

minimize associated parasitics. The sensitive input stage is properly isolated from the large-swing output stage to minimize noise coupling.

The generated layout of a two-stage CMOS Op-Amp was submitted to MOSIS fabrication using a 2-μm double-polysilicon CMOS process [68]. Figure 5.13 shows the microphotograph of the fabricated Op-Amp. Table 5.1 summarizes the measured Op-Amp circuit performance. Comparing the SPICE simulation results without including parasitic capacitances of interconnects indicates that the automatically generated layout using this approach can indeed achieve a very satisfactory performance with only very small layout-introduced device mismatch and parasitic effects.

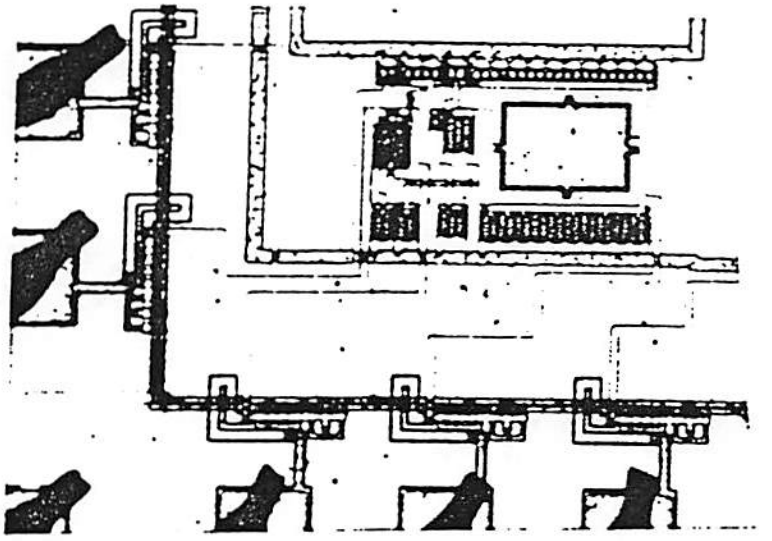


Fig. 5.13 Microphotograph of the two-stage CMOS Op-Amp.

Table 5.1 Performance summary of the two-stage CMOS Op-Amp.

Parameters	Units	SPICE Results	Chip Results
DC Open-Loop Gain	dB	72	69
Unity-Gain Frequency	MHz	4.4	4.2
Phase Margin	Deg.	64	58
Slew Rate	V/uS	4.0	3.7
Input Offset	mV	0.2	2.3
Output Voltage Swing	V	4.5	4.3
Power Supply Current	mA	0.16	0.16

Chapter 6

Layout Synthesis Strategies for Analog VLSI Subsystems

To make analog circuit layout tools more useful to mixed-signal system designers, it is desirable to extend the capabilities of the tools from the module level into the analog subsystem level. Thus the system designers can directly deal with large macromodules such as switched-capacitor filters and A/D converters to further reduce the chip design cycle time.

With a simple extension of the hierarchy, the constraint-based layout methodology described can also be applied to perform subsystem-level layout synthesis using the generated circuit modules as the primitive cells (see Fig. 3.2). Due to the use of slicing structures, the floorplanning can be done in a hierarchical manner to allow a global optimization of the area usage by representing the entire subsystem floorplan as a slicing tree, while decomposing the problem solving into each module level of the hierarchy. This hierarchical layout methodology is suitable for the conventional analog MOS subsystems as well as the newly emerging analog VLSI neural networks.

The physical assembly process for analog VLSI subsystems can be partitioned into three major processing steps: hierarchical floorplanning and placement, module generation, and block routing as shown in Fig. 6.1. In the following the layout synthesis strategies for both the conventional analog MOS subsystems and the analog VLSI neural networks will be addressed. For readers who are interested in real-world implementation of the mixed analog-digital signal processing VLSI, a telecommunication IC example is provided in Appendix A.

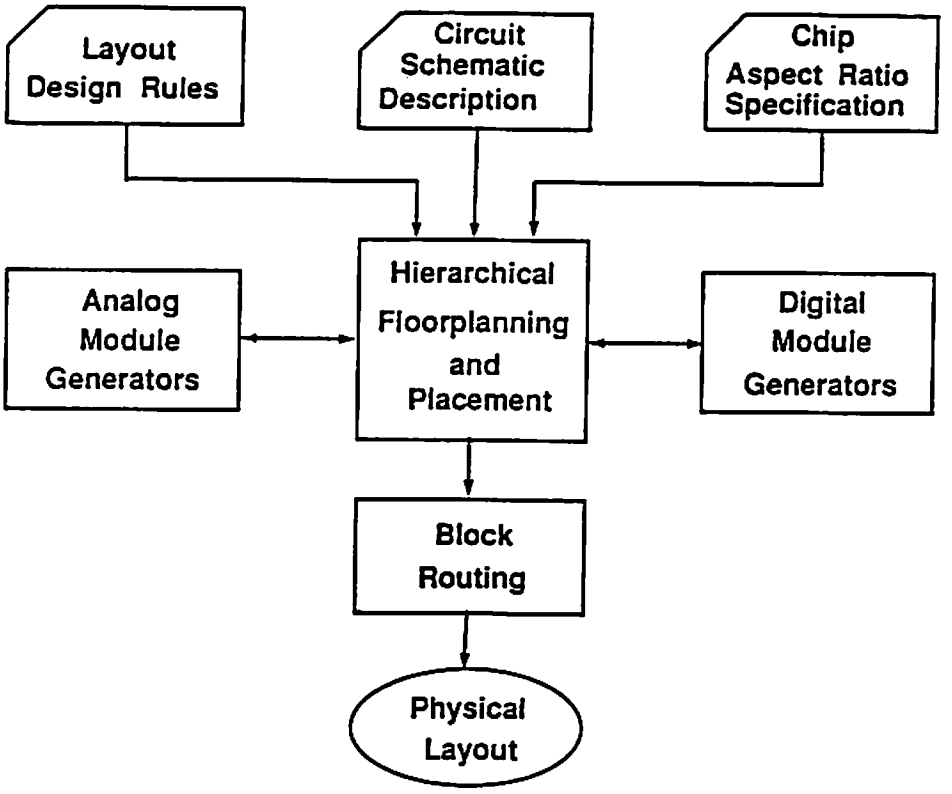


Fig. 6.1 A mixed analog-digital VLSI layout system.

For readers who are not familiar with the VLSI neural circuit implementation, a brief introduction to the subject is also included in Appendix B.

6.1 Layout Synthesis of Conventional Analog MOS Subsystems

6.1.1 Hierarchical floorplanning

Figure 6.2 shows two layout floorplans for traditional analog MOS subsystems such as switched-capacitor filters and data converters. The fixed floorplan shown in Fig. 6.2(a) was recently proposed by H. Yaguthiel et al. of Berkeley [69] for automatic layout of switched-capacitor filters using the standard cell approach. And the floorplan shown in Fig. 6.2(b) is an extended hierarchical version based on our new constraint-based module generation method [44]. The new floorplan is divided into four vertical slices of circuit modules and three routing channels using zone-sensitivity partitioning. Based on the circuit structure and sensitivity levels to overall performance, four types of circuit modules are defined: active, passive, switch, and control modules. The active modules, such as Op-Amps and comparators, are considered as the most sensitive area in the subsystem floorplan. The passive modules, such as capacitors and resistors, are considered the next most sensitive. The less-sensitive analog circuit switches are included in the switch modules. The control modules, which contain digital control logic and clock generators, are the most noisy sources. By using the sensitivity-based floorplanning algorithm, the relative positions of the modules can be determined (as illustrated in Fig. 6.2(b)) to minimize the layout-introduced parasitics and noise coupling in the subsystem level. The construction of the slicing subtrees at the module level is then performed by calling the

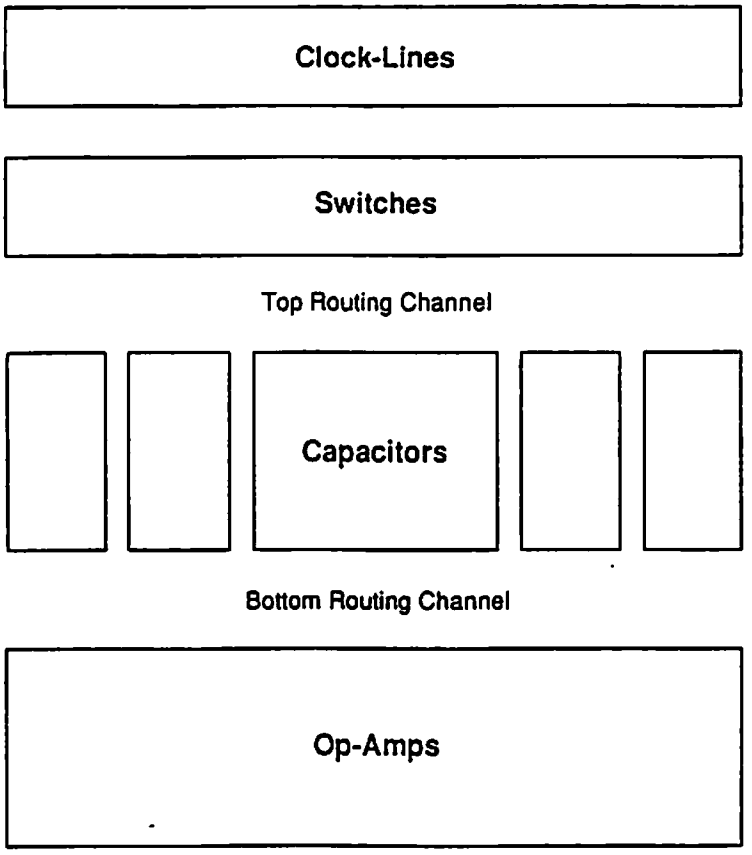


Fig. 6.2 Layout floorplans for traditional analog MOS subsystems. (a) A fixed floorplan for switched-capacitor filters based on the standard cell approach.

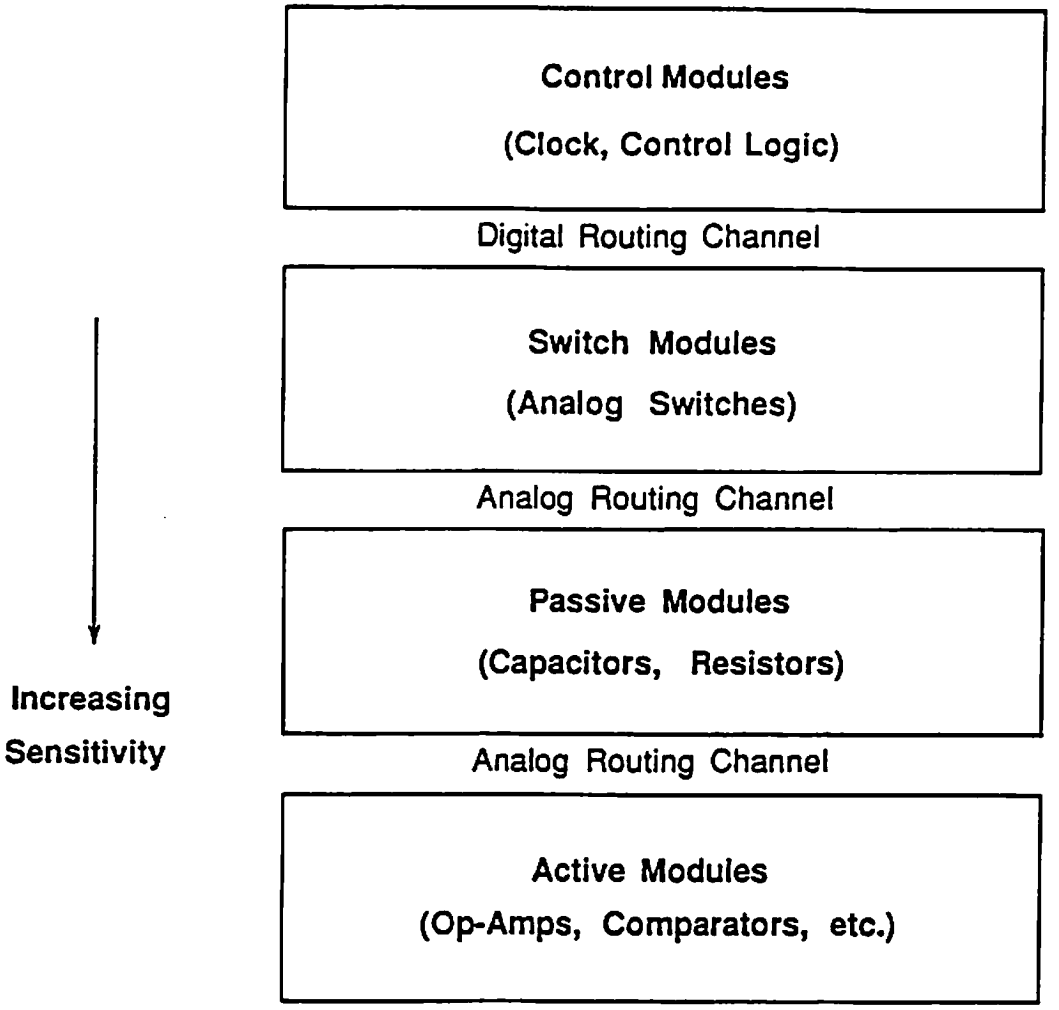


Fig. 6.2 (continued)
(b) A hierarchical floorplan using the sensitivity-based module generation approach.

module generator. Note that this floorplan fits very well into the overall layout methodology based on slicing structures. The key advantage of this approach is that one layout generator can operate from the device, primitive cell, module, and all the way up to the subsystem level within the same hierarchy and therefore can make the overall layout optimization for the subsystem very efficiently. Compared to the previous standard cell-based methods [69-71], this hierarchical layout generation approach can provide a better flexibility and performance.

6.1.2 Constraint-Driven Analog IC Module Generation

The layout generation of analog circuit modules plays a crucial role in determining the overall chip layout quality of the mixed-signal systems, either in terms of performance or area utilization. To satisfy the various layout constraints imposed at the circuit module level, a highly flexible analog IC module layout generator, such as the one described in Chapter 5, is required. Thus as demonstrated before, given the schematic netlist and constraints on the module shapes and pin locations derived from the subsystem-level floorplanning, the constraint-driven module generator can then be called upon to produce the high-quality analog circuit module layout.

6.1.3 Block Routing

To handle the irregular routing regions as well as the mixed-signal routing requirements encountered on the subsystem level, a two-dimensional switch-box router such as Mighty [66] is necessary. The constraint-driven routing strategy based on prioritized net sensitivities can be applied to handle the block routing between modules to minimize the undesired parasitics and noise coupling in the

subsystem layout. The block routing process is implemented in multiple steps. First, all the nets connected between blocks are classified into three categories: sensitive, insensitive, and noisy nets. Sensitive nets include the pure analog signal nets that connect between the active and passive analog modules. The nets that connect between the passive analog and switch modules are classified as an insensitive class, while the nets associated with the control modules are considered as a noisy class. Each class of nets is routed separately in order of assigned priority. The sensitive nets are assigned with the highest priority and get routed first, while the noisy nets are assigned with the lowest priority and get routed last. Therefore, the undesired parasitics associated with the sensitive analog circuit nets and noise crosstalk between mixed signals can be minimized in the layout to achieve high subsystem performance.

6.2 Layout Synthesis of Analog VLSI Neural Networks

For the new class of analog VLSI neural networks (see Appendix B for some introductory material on the subject), extended layout synthesis strategies should be taken. In contrast with the traditional analog circuit architectures, an analog VLSI neural system typically comprises a massively interconnected network of simple analog signal processors. Hence, the size of each analog module and the routing efficiency between modules have a great impact on the total subsystem area. Moreover, due to the matrix configuration of artificial neural networks, additional constraints on the matching of X and Y dimensions around the matrix also need to be taken into considerations.

This section describes automatic layout generation techniques for analog VLSI neural networks based on the hierarchical, constraint-based layout methodology [43]. Special layout requirements for the neural network implementation are addressed. This method is flexible enough to allow a rapid generation of a compact and high performance neural network layout to desired aspect ratios. Experimental layout results on a 16-neuron fully-connected neural network [72] is also presented.

6.2.1 Special layout considerations

There are two major concerns to be considered when dealing with the automatic layout of analog VLSI neural systems. The first concern is due to performance considerations. As mentioned before, because of the sensitive nature of analog integrated circuits, the dependence of circuit performance on layout is in general much more critical for analog circuits than that for digital circuits. The key considerations are the effects of device mismatching, parasitics, and noise coupling on analog circuit layout which can lead to various kinds of performance degradations for traditional analog circuits if proper cares were not taken. Fortunately, some of these constraints can be greatly relaxed in neural networks. This is mainly because the self-learning capability of the neural system makes the computation precision of individual neurons less critical.

The other major concern is due to the area efficiency of the physical layout. In fact, this issue becomes more important for analog VLSI neural systems than that for the conventional analog circuits. The following are the primary physical layout constraints for neural system implementation [43]:

1) Module size. Because of fine-grained parallelism of neural networks, the compactness of each analog module layout has a great impact on the total chip size.

2) Routing area. Because of massive interconnections of neural networks, maximization of the routing efficiency is very important.

3) Variable shapes. Because of widely varying device sizes in analog circuits, variable component shapes need to be accommodated while maintaining the compact module size.

4) Matrix layout. Because of matrix configuration of neural networks, many additional constraints on the matching of X and Y dimensions around the matrix, vertical and horizontal signal feedthrough, and input/output terminal locations need to be taken into considerations.

6.2.2 Neuron and synapse matrix layout techniques

Efficient layout generation techniques for various neural networks have been developed based on the constraint-based, hierarchical layout methodology. Both neuron and synapse modules are designed to be parameterizable. Figure 6.3 shows a neuron module which can be parameterized either as an input neuron only, an output neuron only, or an input/output neuron, depending on the given neural network topology. There are three signal pins assigned for each I/O function. Two pins are assigned for internal matrix routing: one is for neuron-to-synapse connection and the other for neuron-to-neuron interconnect. The latter can be used as an internal feedback channel for Hopfield-type neural networks. Another neuron I/O pin is made available to outside interface which can be used

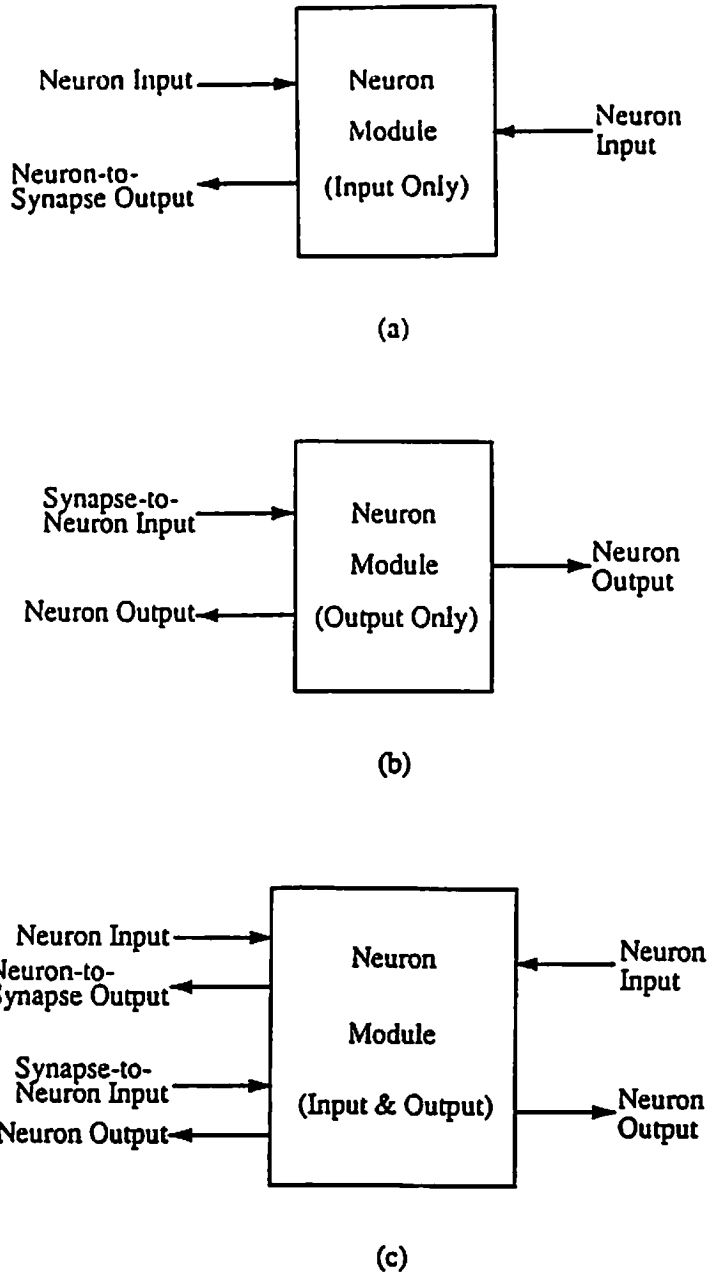


Fig. 6.3 Parameterized neuron module I/O configurations.
(a) Input neuron only.
(b) Output neuron only.
(c) Input and output neuron.

either for feedforward connections between the multi-layer networks or for external feedback connections in Hopfield networks.

The I/O pin configurations of the synapse module is also made parameterizable to improve the routing efficiency of the matrix layout for various neural networks. As illustrated in Fig. 6.4, the matrix routing channels of each synapse module can be parameterized into five different I/O configurations to obtain high area efficiency for any given network topology.

Figure 6.5 shows one of the typical neuron and synapse circuit modules [73]. Given the schematic netlist and constraints imposed on the pin locations and module shapes, the described general-purpose analog module generator can be called upon to automatically produce the geometrical layout that meets the constraints. Because of the great deal of flexibility provided by the module generation and I/O parameterization for the basic neuron and synapse building blocks, various neural networks ranging from application-specific [74,76] to general-purpose [75,77] networks can be automatically generated with high efficiency.

6.2.3 Network floorplanning and routing

Due to its memory-like circuit structure, the physical layout of analog artificial neural networks is most efficiently realized in a matrix style. Figure 6.6 shows a general layout floorplan for single-layer neural networks using parameterizable neuron and synapse modules. A parameterized Hopfield neural network [43] is shown in Fig. 6.7, where the input neurons are configured at the bottom and the output neurons are placed on the right. Figure 6.8 shows the

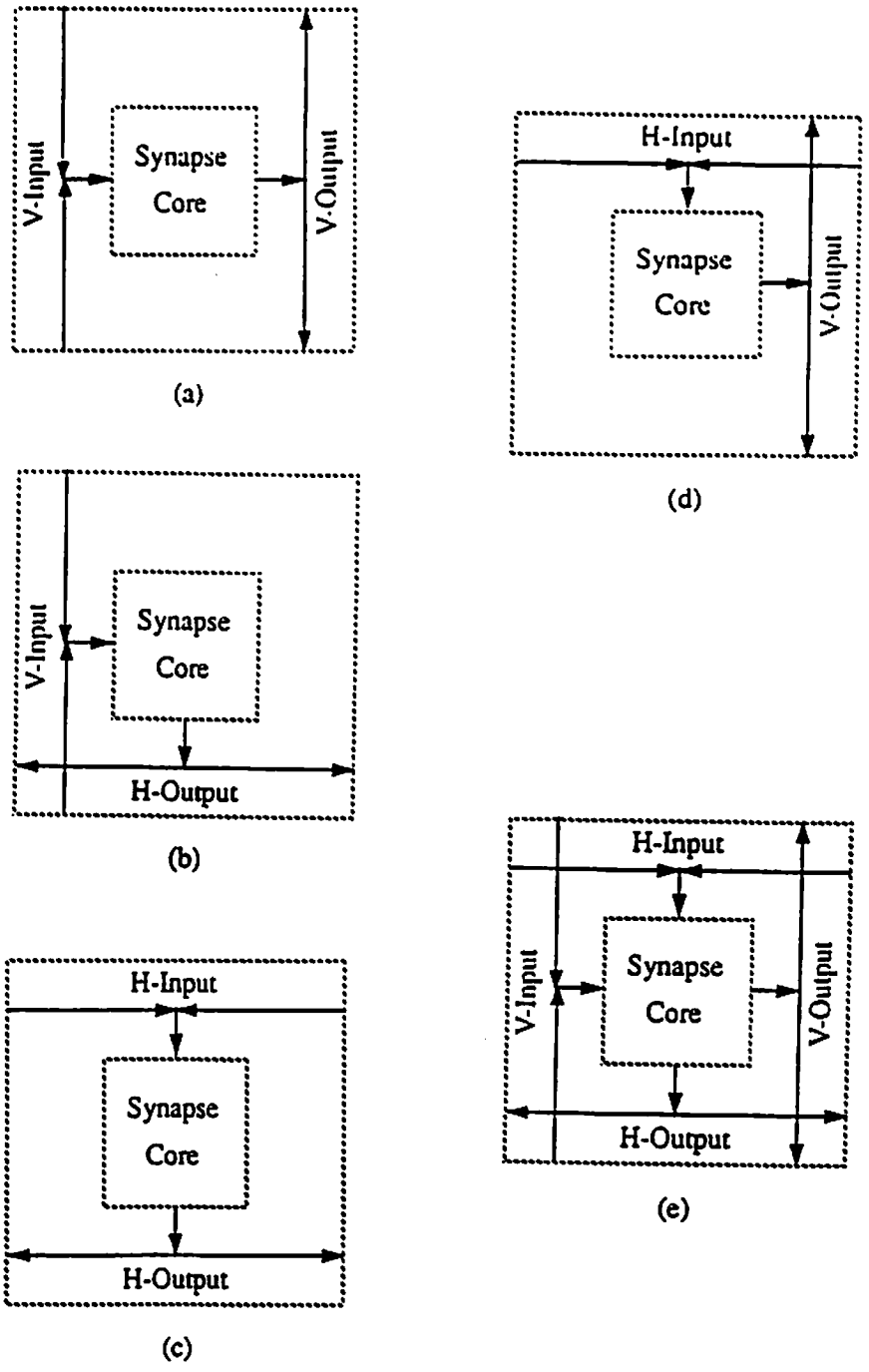


Fig. 6.4 Parameterized synapse matrix routing configurations.
(a) V-input & V-output.
(b) V-input & H-output.
(c) H-input & H-output.
(d) H-input & V-output.
(e) H/V-inputs & H/V-outputs.

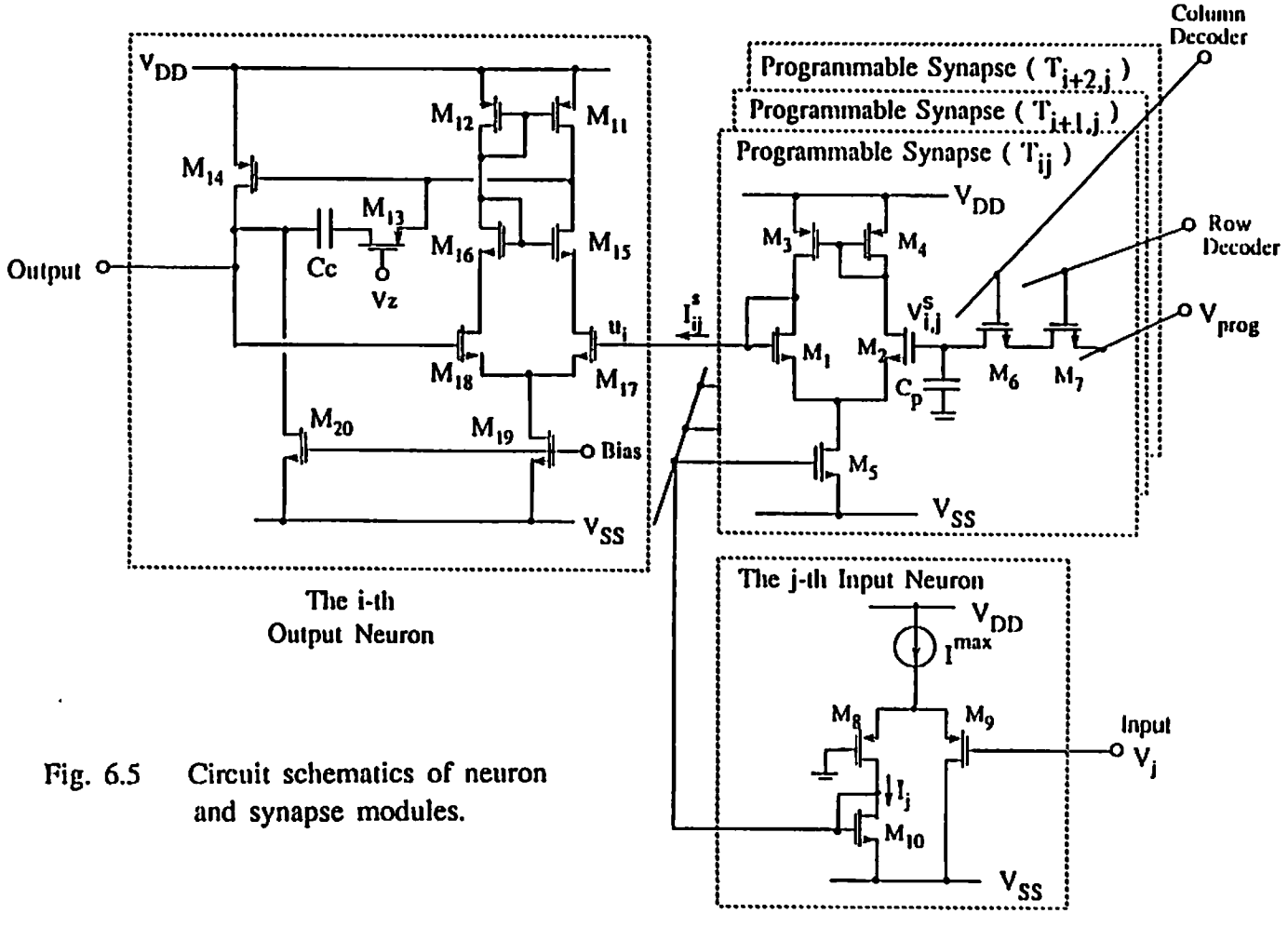


Fig. 6.5 Circuit schematics of neuron and synapse modules.

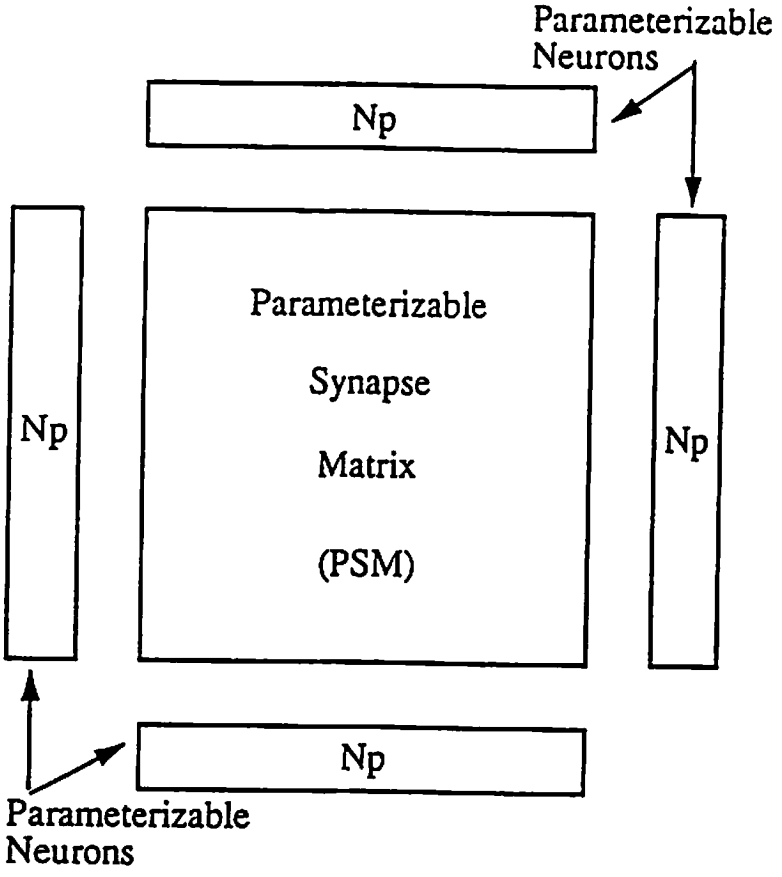


Fig. 6.6 A general layout floorplan for single-layer neural networks using parameterizable neuron and synapse modules.

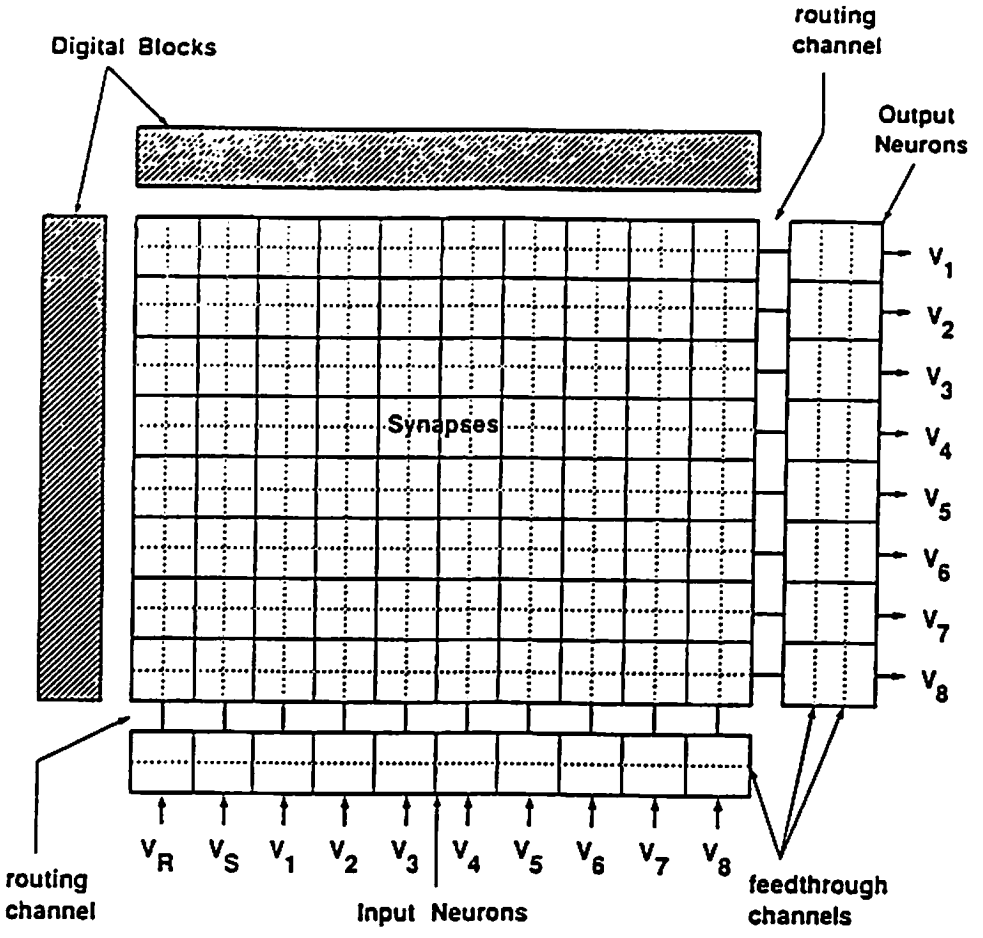


Fig. 6.7 A parameterized floorplan for a Hopfield neural network.

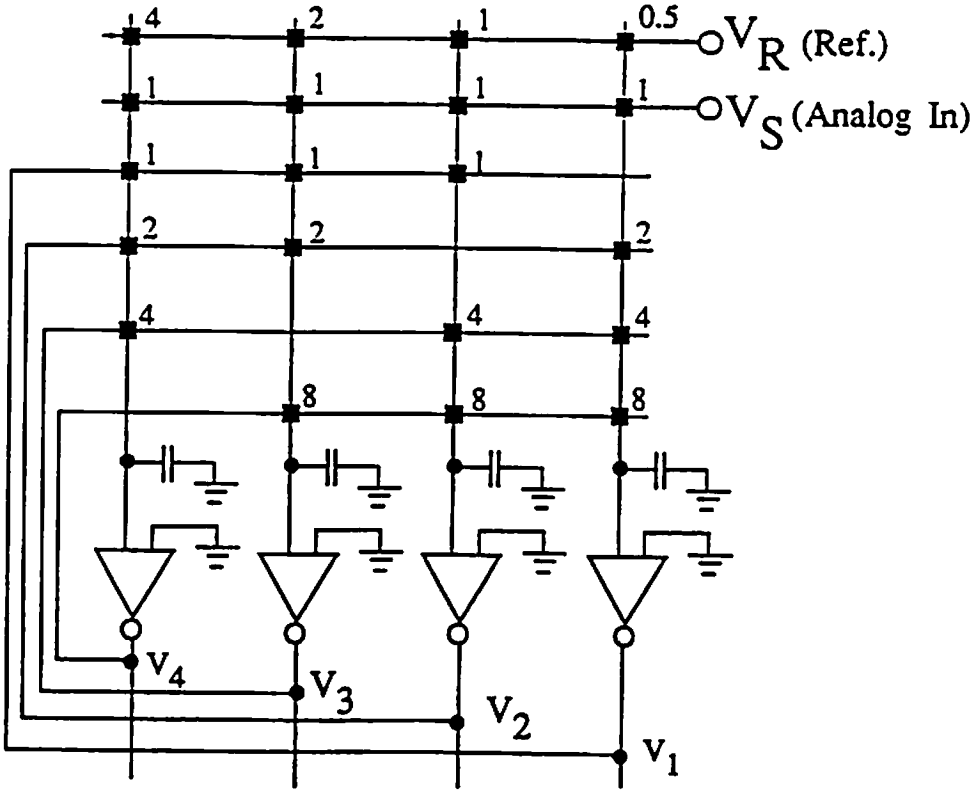


Fig. 6.8 Schematic diagram of a multiple-neuron Hopfield neural network.

top-level schematic diagram of the Hopfield network. As shown in Fig. 6.7, the top-level floorplan is first partitioned by two cut lines into three blocks: synapse, neuron input, and neuron output blocks. The two cut lines not only determine the relative positions of the blocks, but also define the routing channels to be used for signal connections between blocks. Next, each block is divided into a number of identical modules. Note that the interconnection of these identical neuron or synapse modules is done by abutment to minimize the massive connectivity problem. In addition, the module size matching constraint that the one dimension of the central synapse module needs to equal that of the neuron input module, while the other dimension needs to equal that of the neuron output module, is specially imposed during the physical shape optimization process. Figure 6.9 shows the entire slicing tree representation for an 8-neuron Hopfield network using the neuron and synapse modules of Fig. 6.5 [73].

As discussed before, the performance constraints of individual neurons can usually be greatly relaxed in neural networks due to the self-learning capability of the system. To take this advantage, different routing strategies are used to improve the module area efficiency. At the module level, a simple channel router is used to achieve a compact area realization, while a switch-box router is used for block routing on the system level to attain more flexibility and control. This is accomplished by simply changing the net routing order. By ordering the net routing in accordance with the bottom-up traversal of the slicing tree, a switch-box router [66] can be effectively used as a channel router to improve the routing efficiency in the module. Consequently, a more compact module layout can be achieved.

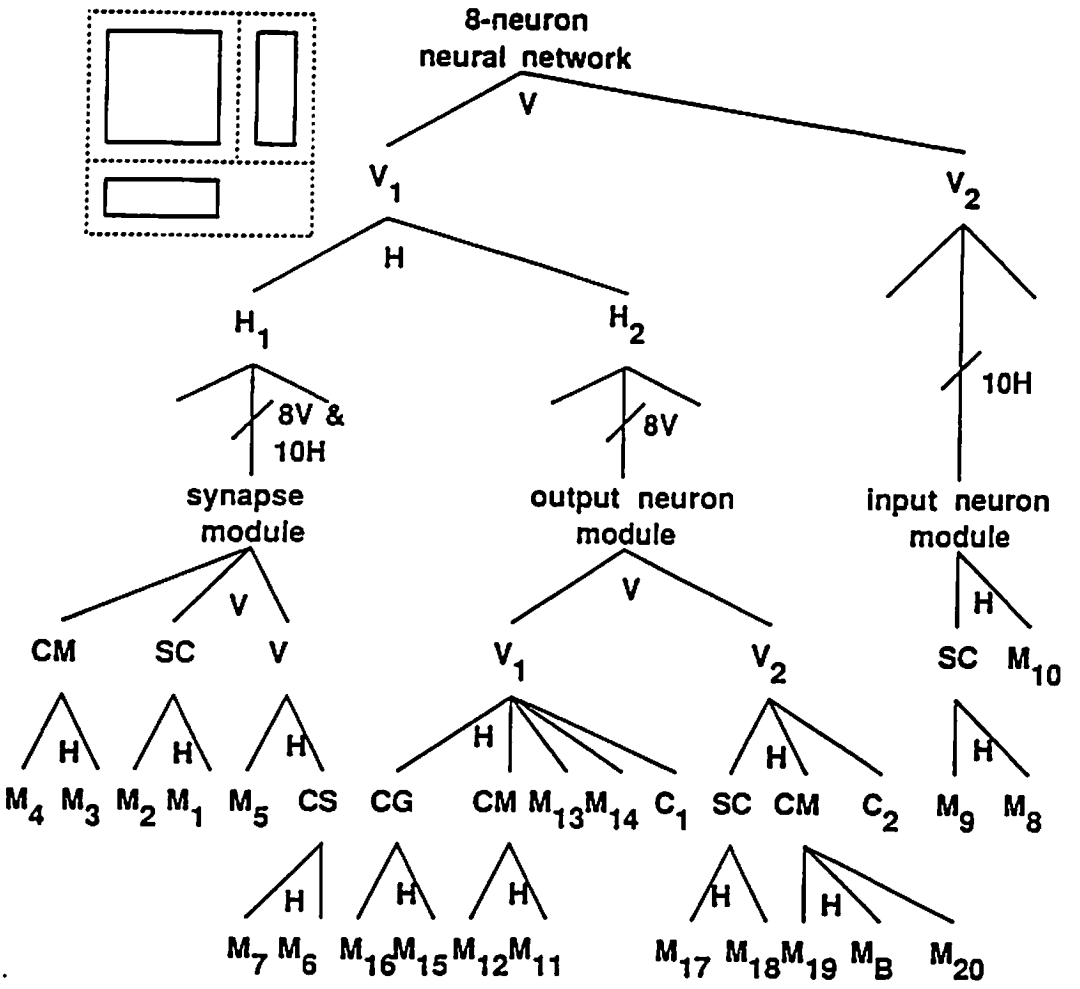
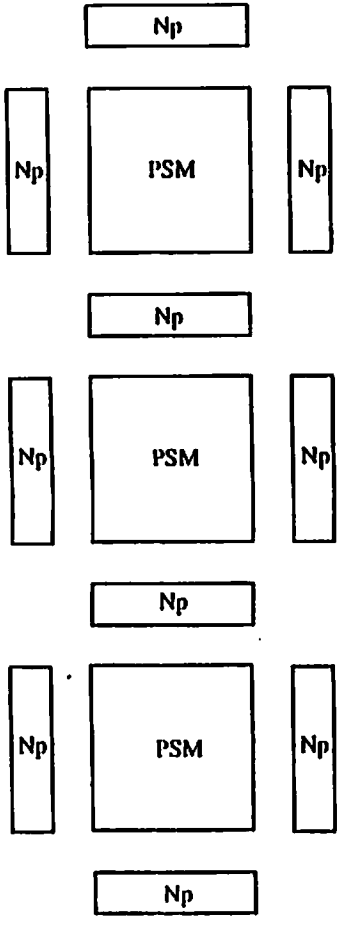


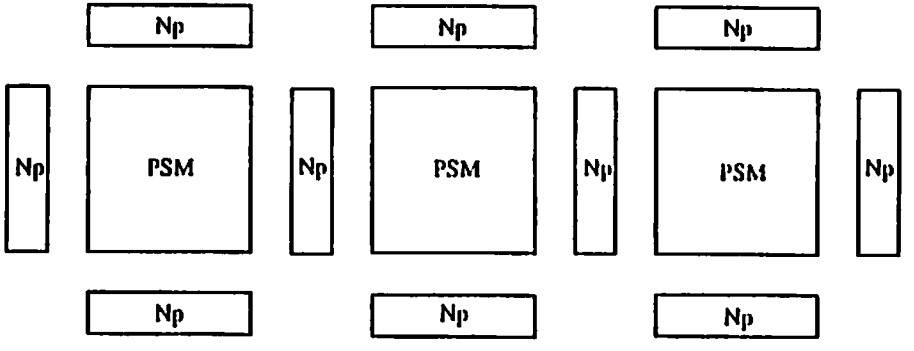
Fig. 6.9 A slicing tree representation for the 8-neuron Hopfield network using the neuron and synapse modules of Fig. 6.5.

To handle the irregular routing regions as well as the mixed-signal routing requirements encountered on the neural subsystem level, a priority-based mutliple-step routing method based on Mighty switch-box router [66] is employed. First, all the nets connected between blocks are classified into three categories: sensitive, insensitive, and noisy nets. Sensitive nets typically include the pure analog signal nets that connect internally between the synapse block and neuron blocks. The nets that carry the large voltage-swing or converted digital signals such as the nets connected externally to the neuron output and input blocks are defined as an insensitive class, while the nets associated with the digital blocks are considered as a noisy class. Each class of nets is routed separately in order of assigned priority to prevent any undesired signal coupling during the mixed-signal routing process.

Both the chip floorplanning and block routing problems become more difficult when dealing with multi-layer neural networks. With two- or three-layer networks, the floorplan can be parameterized in either the single-row or single-column format as shown in Fig. 6.10. The hidden-neuron layer usually needs to be placed in the middle section, while the input and output neurons can have more flexibility. Figure 6.11 shows two possible floorplans for a four-layer feedforward neural system. For the illustration purpose, the input neurons and output neurons of the hidden layers are separated. In the floorplan shown in Fig. 6.11(a), each layer of the networks is placed with the same orientation to preserve a regular signal flow pattern. However, due to the long feedforward connection paths required between layers, the routing area is not effeciently utilized. In addition, because the analog and digital blocks are mixed up together,



(a)



(b)

Fig. 6.10 Parameterizable floorplans for two- or three-layer neural networks.
(a) Vertical style.
(b) Horizontal style.

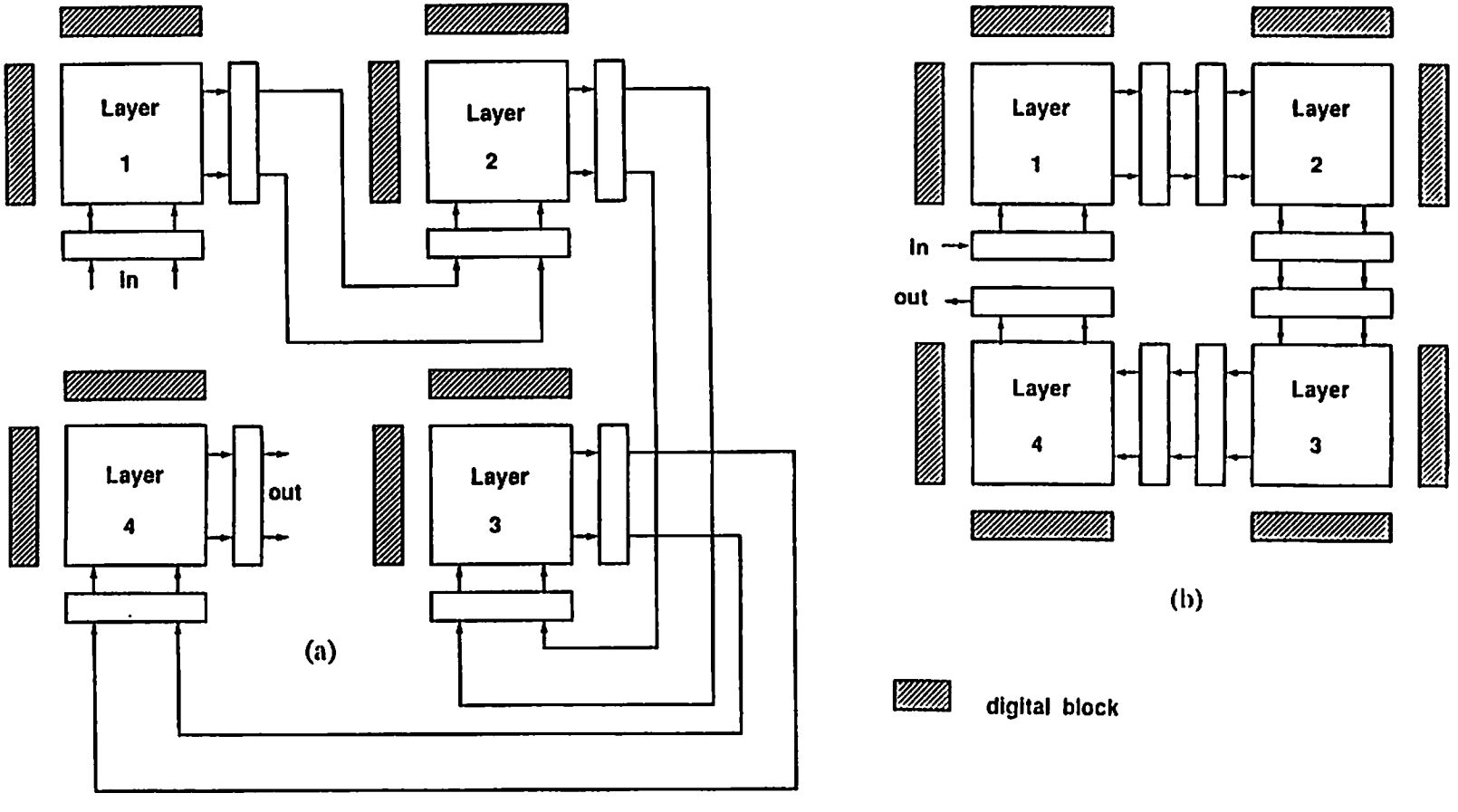


Fig. 6.11 Parameterizable floorplans for four-layer neural networks.
(a) Floorplan A.
(b) Floorplan B.

the routing problem would become more difficult. With a simple change in the orientation of the three layers of networks, an alternative floorplan can be obtained as shown in Fig. 6.11(b). Note that in this floorplan, the analog and digital blocks and their associated routing areas are adquately separated. As a result, a minimal routing area can be obtained.

6.2.4 Experimental results

Based on the described layout methodology, several neural circuits have been experimentally processed by the prototype layout generator. Figure 6.12 shows the generated layout of the neuron and synapse modules of Fig. 6.5. The complete layout result of a 16-neuron fully-connected Hopfield network is shown in Fig. 6.13. Note that in the module layout, all the matched devices are laid out as primitive cells to achieve optimal matching. In addition, different slicing topologies and different component shapes are utilized to satisfy the individual module layout constraints, while the cross matrix constraints between modules are satisfied. The layout-introduced parasitic capacitances and resistances can then be extracted and simulated with the input circuit schematics using SPICE for performance verification.

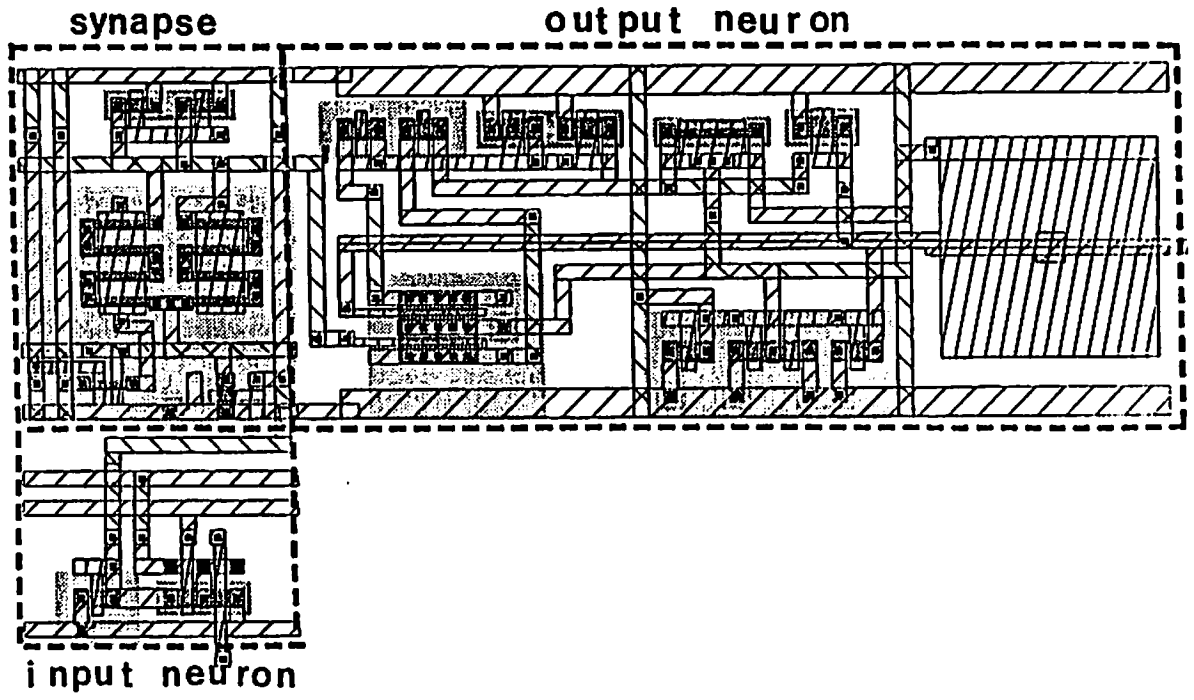


Fig. 6.12 Generated layout of the neuron and synapse modules of Fig. 6.5.

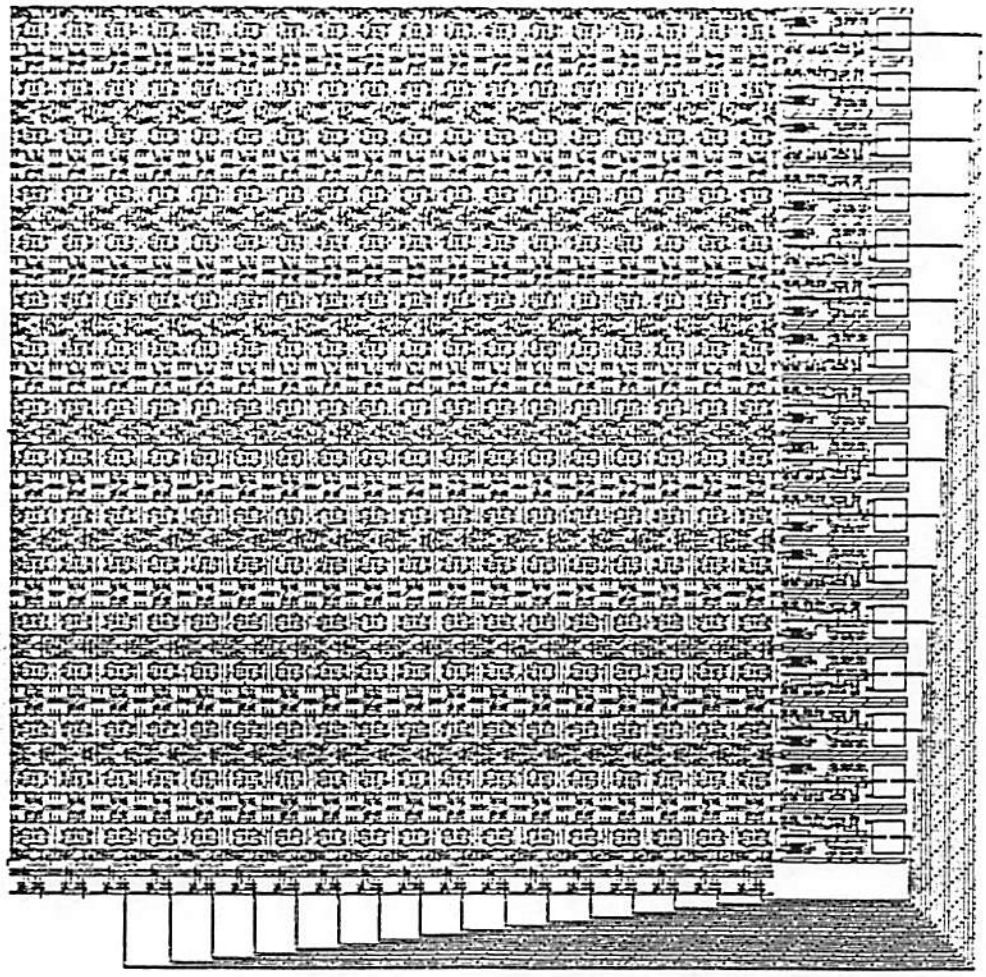


Fig. 6.13 Generated layout of a 16-neuron Hopfield neural network.

Chapter 7

Conclusions and Future Directions

Efficient methods for automatic design synthesis and layout generation of analog integrated circuit modules have been presented. A system that consists of a knowledge-based top-down design tool and a constraint-based bottom-up layout generator has been developed to automate the design of common analog MOS VLSI circuit modules. Extensions to higher-level subsystem layout synthesis using hierarchical floorplanning and module generation techniques have also been described for two different classes of mixed-signal VLSI applications. One is for conventional signal processing systems and the other is for newly emerging neural networks.

A new expert system assisted analog IC design methodology is introduced. Through iterations between a circuit simulator and an expert system, the methodology offers a reliable means of achieving high performance analog circuit designs. The self-reconstructing technique for the flexible architecture approach has been described. It is realized through equation substitutions during a design process. A prototype analog circuit design synthesis tool for CMOS Op-Amps has been developed. The organization and implementation of the tool are described in detail. The experimental results show that the design system, by adopting the iterative design methodology and the flexible architecture approach, is powerful and efficient in carrying out analog circuit design tasks. The capability of the design synthesis system can be extended to analog signal processing

modules other than operational amplifiers.

A custom layout synthesis method for analog IC modules using automatic circuit recognition and layout constraint analysis techniques is described. Special analog IC layout requirements are analyzed and effective solutions are discussed. Through the use of an analog circuit recognition technique, circuit primitives and critical analog circuit nets are systematically processed before layout generation. Weighted analog circuit layout constraints are internally generated and incorporated in the final layout to minimize undesired parasitic effects. An effective analog circuit floorplanning technique based on a zone-sensitivity partitioning algorithm is developed to derive a slicing floorplan incorporating the electrical as well as the physical constraints. Subsystem-level layout synthesis strategies for conventional analog signal processing systems are discussed. Efficient layout generation techniques for analog VLSI neural networks have been developed. A prototype layout generator has been implemented based on the described analog IC layout methodology. Generated layout results of several common analog CMOS circuit modules as well as analog neural networks have been demonstrated. Measurement results from test circuits have shown that the automatically generated analog circuit layout based on this methodology can indeed produce a satisfactory circuit performance with negligible degradation due to the layout-introduced parasitic effects. This layout synthesis approach is quite general and can be applied to handle a wide variety of analog circuit modules as well as analog VLSI subsystems owing to its self analysis capablitity for high-quality analog circuit layout.

More effort is required to extend this work from applications to IC modules to more complex analog and mixed-signal VLSI systems. Some of the analog subsystem layout issues have already been addressed in this dissertation. By using the sensitivity-based hierarchical slicing floorplanning and the constraint-based module generation techniques described in Chapter 6, a powerful analog subsystem layout framework can be quickly established. In this framework, the layout generator can operate from the subsystem, module, primitive cell, and all the way down to the device level of the hierarchy and thus can produce better layout results with a great deal of flexibility.

More challenging problems still remain in the design synthesis area for analog IC subsystems. Unlike the layout synthesis where a general tool can be applied to virtually all the analog circuits using the bottom-up approaches, the design synthesis of analog IC modules and subsystems can only be implemented from the top down and thus requires one specific design tool for each analog circuit function as well as for each circuit topology. For example, a design synthesis tool for A/D converters cannot be used to design SC filters. Even for the same A/D circuit function, a sigma-delta A/D synthesizer cannot be used to design a charge-redistribution type of A/D converter, and vice versa. Therefore, multiple design synthesis tools will be required for analog and mixed-signal IC design. To restrict the problem to a more manageable size, the research efforts could concentrate on the common and frequently used analog circuit subsystems. In addition, use of fuzzy logic reasoning for analog VLSI design synthesis needs to be explored to reduce the number of design rules in the expert system program.

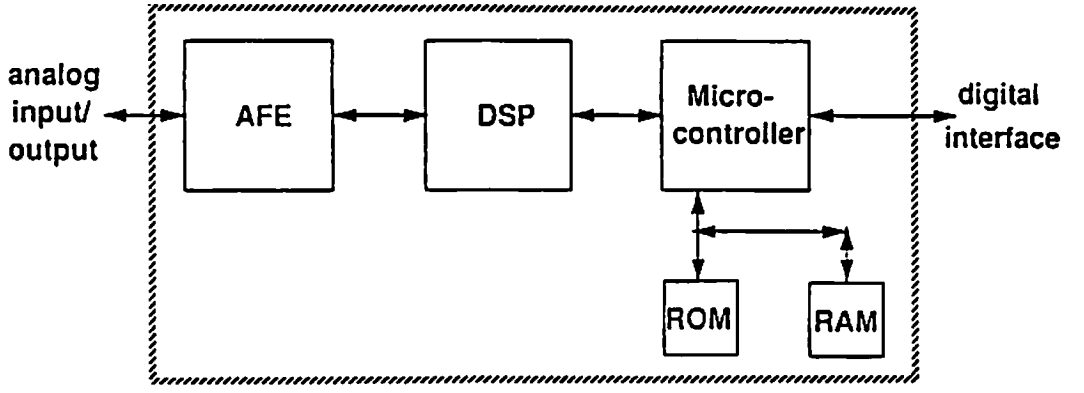
Appendix A

A Mixed Analog-Digital Signal Processing VLSI for Telecommunications**

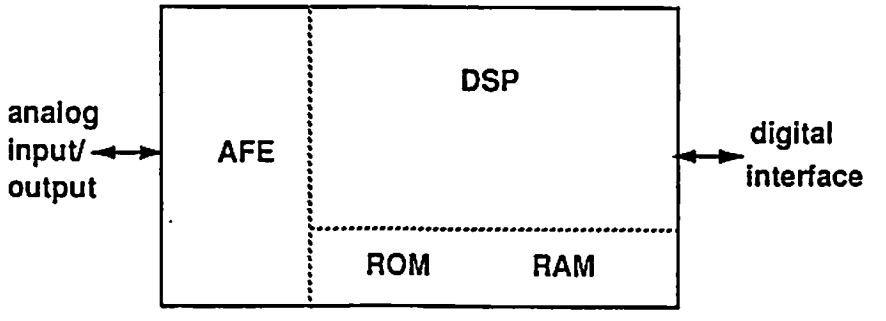
I. Introduction

The use of mixed analog and digital signal processing techniques has made it easier to realize high-speed voiceband telecommunications system applications (such as data and facsimile modems) which can support multiple standards [78-82]. With advanced CMOS VLSI technologies, such voiceband systems can now be highly integrated in a multi-chip set composed of a digital signal processor (DSP), an analog front-end (AFE), and a microcontroller with additional memories as shown in Fig. A.1(a). While the demands for lower-cost and smaller-size systems are continuously pushing for higher levels of integration, until now, the highest integration reported for high-speed voiceband systems is still limited to a two-chip set containing a single-chip DSP and a single-chip AFE. The main barriers to achieving the ultimate single-chip solution can be attributed to the difficulties in designing the analog circuits to operate with a single 5V supply (instead of the conventional 10V supply) and integrating it with the DSP on a single VLSI chip, while still maintaining the adequate system performance and a reasonable die area.

** The work presented in this appendix was accomplished by David J. Chen and other members in the Communication VLSI Design Group in Sharp Digital Information Products, Inc., Irvine, CA. Partial results of this work was presented at IEEE Custom Integrated Circuits Conference at San Diego, CA in May 1991.



(a)



(b)

Fig. A.1 A mixed analog-digital signal processing system for telecommunications.
(a) A multi-chip configuration.
(b) A single-chip solution.

In this appendix, a single-chip solution which has successfully integrated all the necessary AFE, DSP, and microcontroller functions on a 81 mm^2 chip for implementing high-speed, multi-standard voiceband systems [12] is presented. The chip is fabricated in 1.0-μm CMOS technology and consumes 375 mW from a single 5V supply. A well-balanced combination of analog and digital signal processing techniques is used to achieve the design objectives.

II. Circuit Description

The chip is divided into two sections: DSP and AFE, as illustrated in Fig. A.1(b). The DSP section contains a DSP and on-chip memories. The DSP is designed with a custom architecture and an instruction set tailored for voiceband applications to allow a great deal of flexibility for accommodating multiple standards. In addition to handle the major signal processing tasks, it can also perform microcontroller functions. The analog section provides all the necessary analog front-end interface and signal conditioning for the DSP. In addition, it implements some selected signal processing functions to reduce substantially the computational requirements of the DSP and to achieve a more efficient utilization of the silicon area.

(A) Analog Front-End Section

The block diagram of the AFE section is shown in Fig. A.2(a). The circuit can be further divided into analog input and output channel blocks. Since the performance requirements of the input channel is generally more stringent than those of the output channel for telecommunication applications, different design

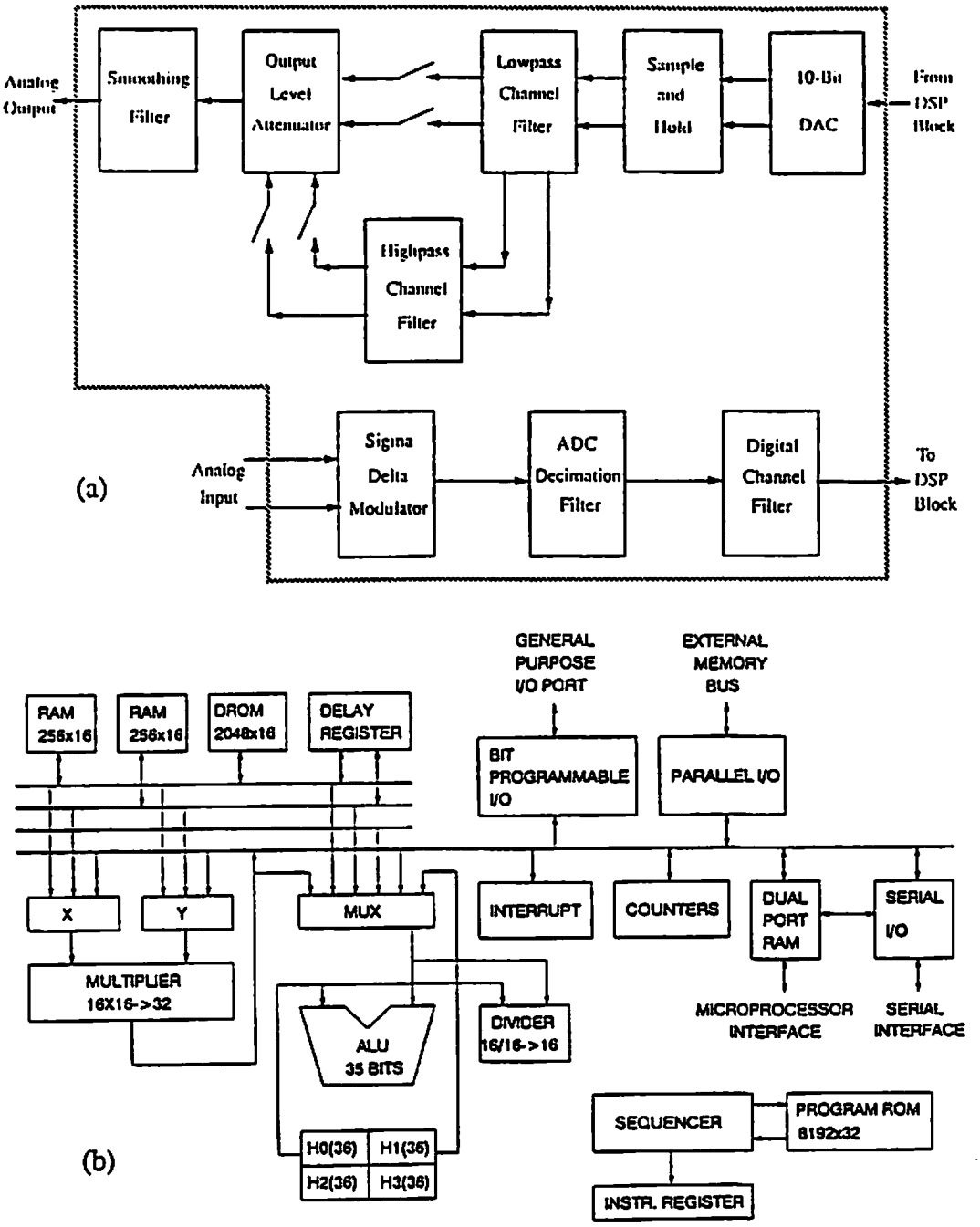


Fig. A.2 Block diagram of a mixed analog-digital MOS VLSI for voiceband telecommunications.
(a) Analog front-end section.
(b) Digital signal processor section.

strategies were taken. In the input channel, a 16-bit oversampling A/D converter is designed with all-digital channel filtering to provide the large dynamic range performance and flexibility required for processing various input signal conditions. In the output channel, a 10-bit D/A converter is provided with mostly-analog filtering to minimize the chip area. To allow a single-chip integration with the DSP circuits, the analog circuits are designed to operate with a single 5V power supply. A fully differential architecture is used in all the internal analog circuits to help increase the dynamic range and improve the power supply rejection.

In the input channel, the 16-bit oversampling A/D converter is implemented using a second-order sigma-delta modulation principle [83]. The analog modulator is designed with switched-capacitor integrators using an oversampling clock rate of 5.184MHz to ensure a sufficient signal-to-noise ratio performance margin for various voiceband applications. To meet the high-speed requirements, the operational amplifiers are designed with a folded-cascode amplifier topology [50] with dynamic common-mode feedback, as shown in Fig. A.3. The 1-bit output stream of the modulator is then decimated and filtered by the two-stage digital decimation filter. The first-stage filter, which is realized with a 3rd-order comb filter, removes the high-frequency noise and decimates the signal down to four times of the Nyquist sampling rate. The second digital filter, which is a FIR filter, performs the input channel filtering, while downsampling the signal further to the Nyquist rate. Both the decimation ratio and channel frequency response can be programmed by DSP to handle different voiceband operations.

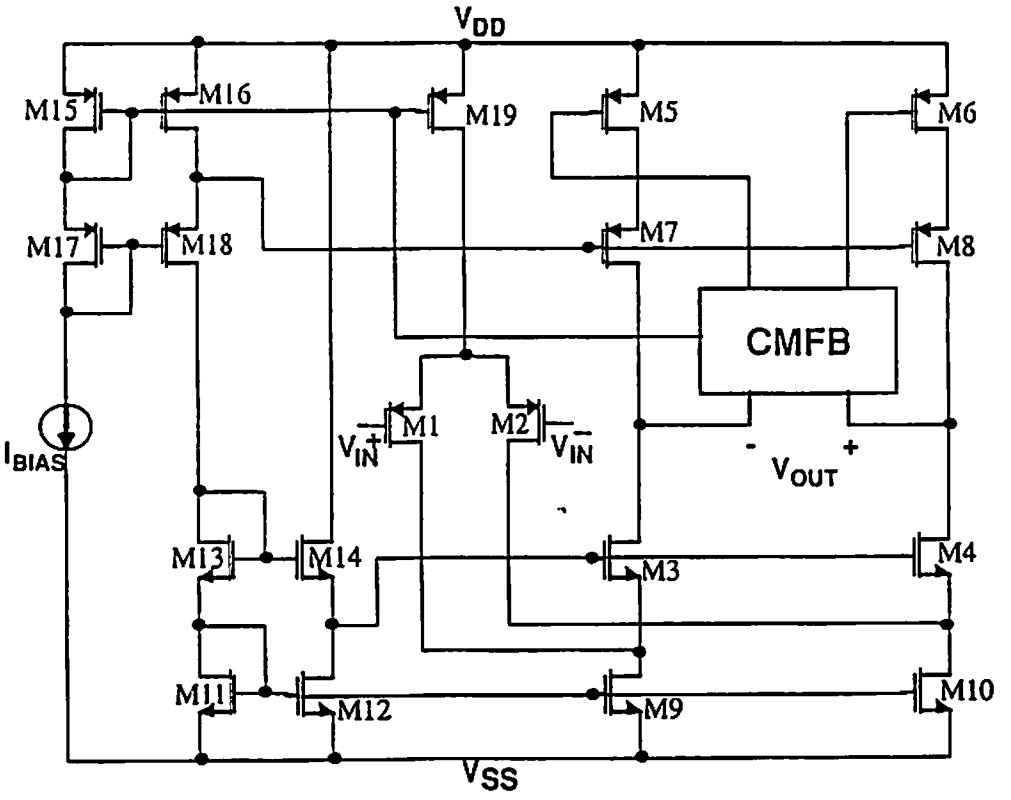


Fig. A.3 Schematic of the high-speed fully-differential Op-Amp.

Instead of using a sigma-delta approach, the D/A converter in the output channel is designed with a fully-differential charge-redistribution [84] technique using switched capacitors to obtain a better area efficiency. The converted analog signal then goes through a sinx/x-compensated sample-and-hold circuit, a programmable switched-capacitor channel filter, a 4-bit programmable output level attenuator, and a 2nd-order continuous-time output smoothing filter. The programmable switched-capacitor filter, which consists of a 6th-order elliptic lowpass filter and a 5th-order elliptic highpass filter, can be configured to produce a number of channel filter characteristics required for various voiceband operations.

(B) Digital Signal Processor Section

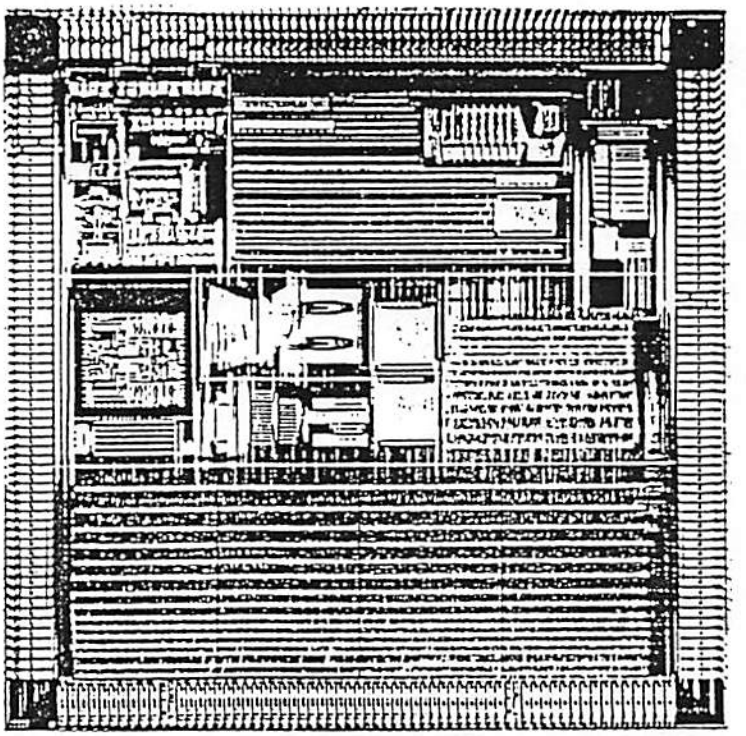
Figure A.2(b) shows the block diagram of the DSP section. The DSP is operated with an instruction cycle of 96.5 ns to provide an adequate processing speed for handling high-speed and multi-standard voiceband operations. It includes a 16x16 hardware multiplier which is designed to perform a multiply in one machine cycle. The ALU performs 35-bit fixed-point arithmatic operations. The on-chip memories include 8192 x 32 bits of instruction ROM, 2048 x 16 bits of data ROM, and two banks of 256 x 16 bits data RAM. It also allows the access of up to 64k bits of external RAM and supports external DSP in a multi-chip configuration. A serial I/O block is included to generate all the input and output channel clocks using seperate digital phase-locked loops.

III. Implementation and Performance Results

This mixed-signal device has been fabricated in an 1.0μm double-metal CMOS process. The chip size with an 8K instruction ROM is about 81 mm^2. Figure A.4 shows microphotograph and floorplan of the chip. The pure analog circuit section of the chip is laid out manually in full custom style, while the layout of digital filters and the majority of the DSP section is done by an automatic CAD tool using standard cells. Note that in the floorplan, the quite analog circuit section is arranged in the top left corner of the chip to obtain a good separation from the noisy DSP section. Moreover, the low-frequency analog output channel block (shown at the bottom of the analog section) is properly isolated from the high-frequency analog input channel. In addition, many substrate shieldings and guard rings are used around the analog circuit layout area to further reduce any noise coupling from the digital circuits.

The measured frequency response of the analog output channel is illustrated in Fig. A.5. Three examples of the possible channel filter operations are shown. The 1st photo shows a full-channel lowpass filter response for half-duplex data transfer operations. The 2nd photo displays a half-channel lowpass filter response. This filter function is required in transferring data in the lower split-channel for a full-duplex operation on a 3KHz channel bandwidth. The 3rd photo shows a bandpass filter response of the output channel. This allows the data transfer in the upper split-channel during the full-duplex operation. The measured signal-to-noise ratio of the analog input channel is shown in Fig. A.6. More than 80dB dynamic range is obtained. Table A.1 summarizes the chip characteristics.

(a)



(b)

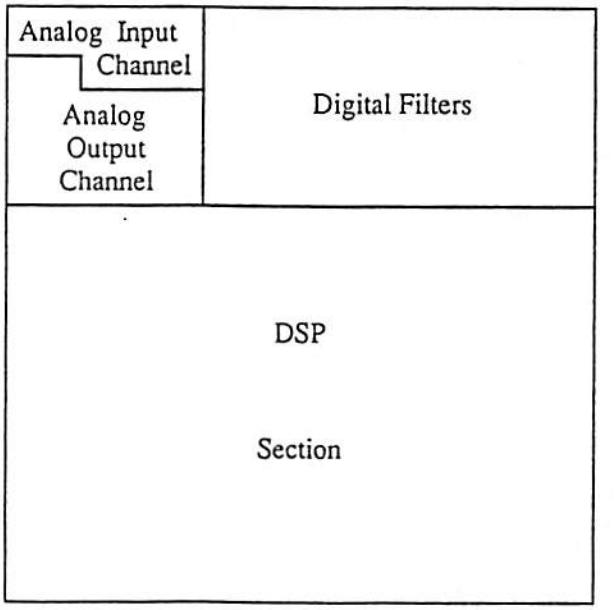


Fig. A.4 Microphotograph and floorplan of the mixed-signal chip.
(a) Die photo.
(b) Chip floorplan.

(a)

(b)

(c)

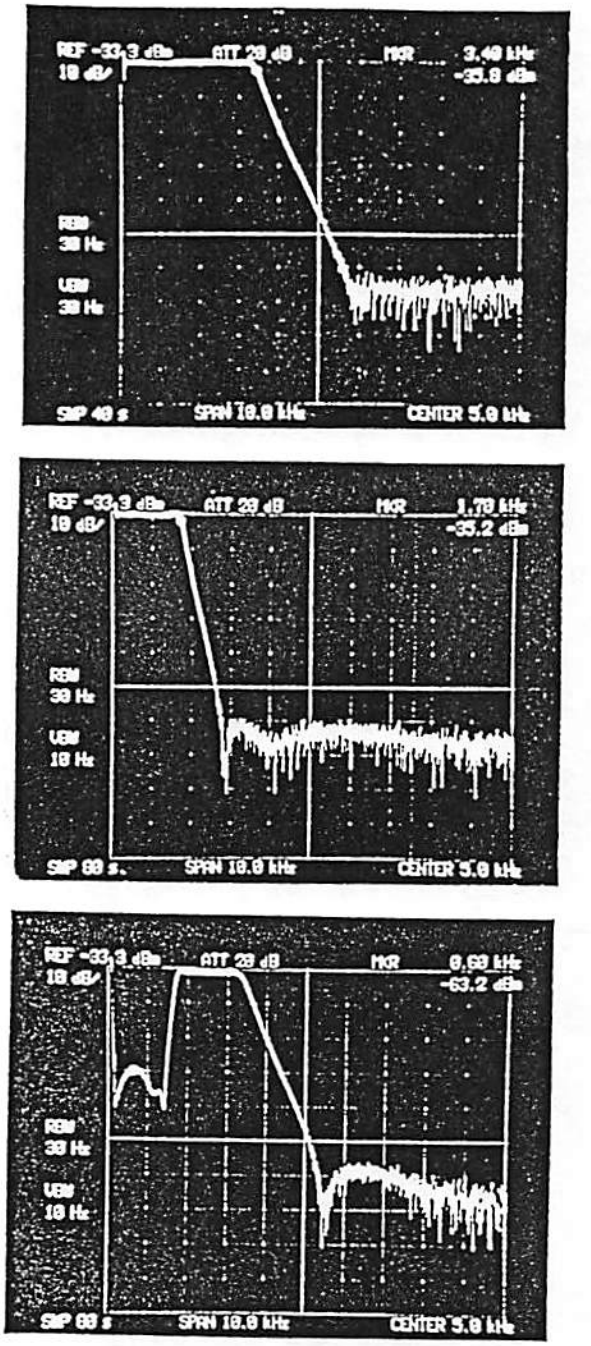


Fig. A.5 Measured frequency response of analog output channel.
(a) The full-channel lowpass filter response.
(b) The lower split-channel lowpass filter response.
(c) The upper split-channel bandpass filter response.

Table A.1 Summary of the mixed-signal chip characteristics.

Power Supply	5.0V
Power Consumption	375 mW
Chip Size (with 8K IROM)	81 mm^2
Transistor Count	500,000
Process	1.0μm Double Metal CMOS
Instruction Cycle	96.5 nS
Dynamic Range	88 dB

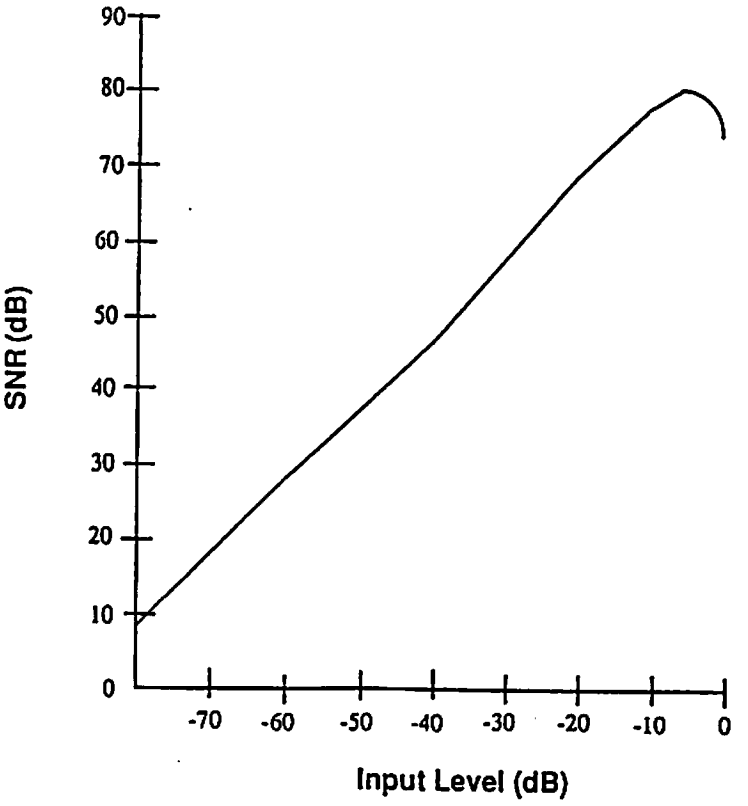


Fig. A.6 Measured signal-to-noise ratio of analog input channel.

IV. Conclusions

A single-chip solution for implementing high-speed voiceband telecommunication systems using mixed analog-digital signal processing techniques has been presented. The chip contains a 96.5nS DSP, a high performance AFE, and an 8K instruction ROM in 81 mm^2 using 1.0-μm CMOS technology. It requires only a single 5V supply and dissipates 375 mW. With sufficient on-chip memories, this device is capable of implementing most of the voiceband system applications in a single-chip solution. And with more processing speed obtainable from the continuously scaled CMOS VLSI technology in the near future, this mixed-signal processor is very promising in realizing audio system applications in a single chip as well.

Appendix B

Basic Concepts in Analog VLSI Implementation of Artificial Neural Networks

The implementation of artificial neural networks using VLSI technologies has recently received much attention [17-19]. In particular, the analog VLSI approach is extremely attractive for implementing neural network algorithms in terms of hardware size, power, and speed [85]. Several analog CMOS neural network realizations have been reported [74-77] that contain up to several hundreds of neurons and thousands of synapses on a single chip.

The secret of immense computational power in the neural networks is revealed as the parallel processing done by neurons and synapses. While each neuron performs simple analog processing at a low speed, the rich connectivity among neurons through synapses provides powerful computational capabilities for the large quantity of data. The data are processed asynchronously in the continuous time domain and spread globally into all network elements. In addition to the parallel processing nature, the neural network has self-learning capability which is done by changing the weights of synapses between neurons. The self-learning capability makes neural networks useful at the situation that training data are insufficient and the fault tolerance of a system is necessary. Moreover, the relatively imprecise network elements can be compensated with the self-learning scheme. As a result, the immense computational power and self-learning capability give neural networks excellent prospects for solving complex problems in image processing, pattern recognitions, fuzzy logic, machine vision,

inexact reasoning, and adaptive control. The following covers some basic concepts in implementing VLSI artificial neuron models.

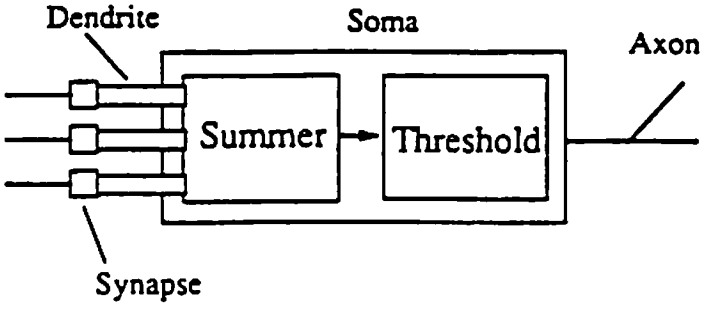
(A) Basic Neuron Models

A biological neuron basically consists of a cell body, dendrites, and axons as shown in Fig. B.1(a). The cell body, which is called *soma*, performs complicated chemical processes, such as summation and firing with respect to a threshold level. The input signals for a cell body are transmitted through the dendrite, while the output signals are carried to other cells through axons. The electrical signal of an axon connects to a dendrite through a specialized contact, which is called *synapse*. In general, the neuron performs a simple threshold function. When the potential inside the cell body is larger than the threshold value, the neuron fires. Here, the normal firing rate is quite low which typically requires a few milli-seconds. It is known that the human brain has approximately 10^{11} neurons and 10^4 synapses.

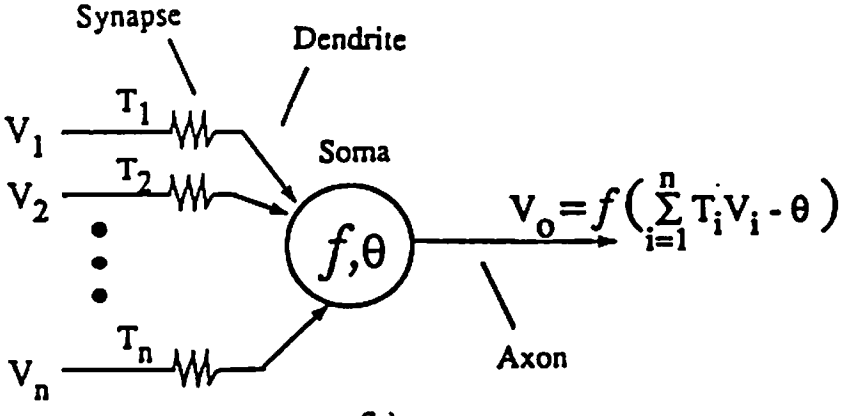
In the artificial neural network, the neuron and synapse are configured as processing element and connection strength, respectively. Figure B.1(b) shows a simplified artificial neuron model. The basic computation of the artificial model involves a weighted sum over the entire set of neuron inputs. The result of the weighted sum minus a threshold value is amplified to produce the new neuron output. Mathematically, this can be described as

$$V_o = f\left[\sum_{i=1}^{n} T_i V_i - \theta\right]. \tag{B.1}$$

where the V_i are the neuron input voltages, T_i are the synapse connection



(a)



(b)

Fig. B.1 Neuron model.
(a) Biological neuron model.
(b) Artificial neuron model. (The V_i are the neuron input voltages, T_i are the synapse weights, and θ is a constant threshold value.)

weights, and θ is a contant threshold term. The $f(\cdot)$ is a nonlinear function, which can be thought of as an amplifier.

(B) Basic Neural Architectures

The various features of the artificial neural network are determined by the function of neuron and the interconnection patterns. The grouped neurons which are arranged into a disjointed structure is called a *layer*. Figure B.2 shows several basic neural architectures which include single-layer/multi-layer and feed-forward/feedback networks. A single-layer network typically consists of a neuron input stage, a synapse connection matrix, and a neuron output stage. With a feedback path added to the single-layer network, the resulting recursive network becomes the well-known Hopfield neural network [72]. The condition for the synapse weight in the Hopfield network is very restrict (being symmetric and no self-feedback terms), while that for the neuron transfer function is very relaxed (only monotonically increasing). Due to the simple architecture and clearly proved dynamics of the Hopfield network, many hardware implementations have been widely used in real-world applications such as associative memory and the solution of certain engineering optimization problems. Finally, the multi-layer neural network, which is typically vitalized by a back-propagation learning rule, is a feed-forward network with hidden layers located in the middle stages.

(C) VLSI Neuron Implementation

The neural network algorithms for the software computation cannot be directly used to realize VLSI neural networks. Basic elements of the VLSI artificial neural networks consist of amplifiers as neurons and resistors as

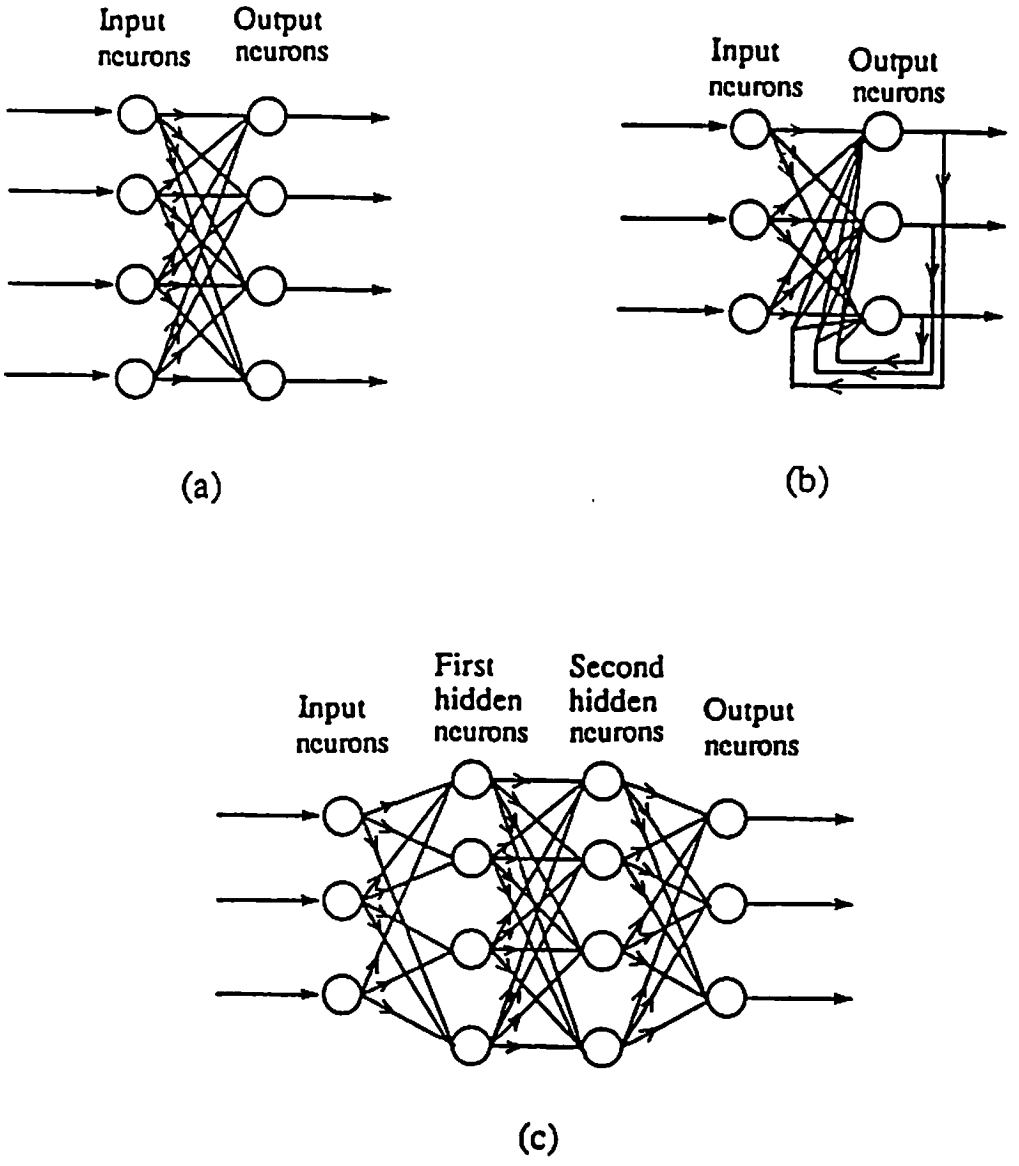


Fig. B.2 Basic neural network architectures.
(a) Single-layer feedforward network.
(b) Single-layer network with feedback -- Hopfield neural network.
(c) Multi-layer feedforward network.

synapses. The neuron transfer function for the software computation is an exact mathematical function, while that for the VLSI implementation is an approximate function. In addition, perfect match and wide dynamic ratio of the synapse weights are difficult to obtain in VLSI technologies.

One important factor in VLSI neural network design is to increase the amplifier gain. Figure B.3 shows the simple circuit realization of a neural model using resistors. The synapse weight T_i is realized with a conventional resistor R_i (i.e., $T_i = 1/R_i$). For a negative synapse weight, a resistor and an inverted neuron output are used. Note that the neuron input voltage V_i is assumed to be an independent voltage source and the internal voltage source resistance is included in R_i. Several VLSI chips [86,87] have been reported using this direct resistor realization. The amplifier output voltage is determined by

$$V_o = f\left[\frac{\sum_{i=1}^{n} V_i/R_i}{1/R_{eq} + \sum_{i=1}^{n} 1/R_i}\right]. \tag{B.2}$$

Here, $f(\cdot)$ is the transfer function of a neuron and R_{eq} is the input impedance of the neuron. On the other hand, the neuron output in software computation is determined by

$$V_o = f\left[\sum_{i=1}^{n} V_i/R_i\right]. \tag{B.3}$$

In the direct resistor realization method, large amplifier gain is needed due to the effect of the denominator in (B.2). In addition, the relative synapse values are more important than the absolute values in the VLSI implementation. In VLSI

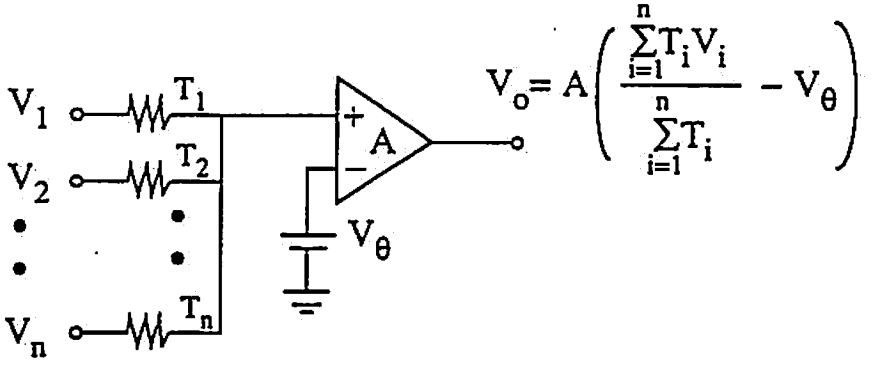


Fig. B.3 A VLSI neuron with direct resistor implementation.

technologies, the relative tolerance in the device is much easier to be achieved than the absolute tolerance.

Appendix C

Prototype Program for Analog MOS Circuit Module Layout Generation

```
{************      SLAM -- Development Version 1.0   ************}
{************       By David Jan-Chia Chen at USC    ************}
{************             June 1991                  ************}

/*************************************************************************/
/**** Analog MOS Circuit Recognition Module (Pascal Version) *******/
/*************************************************************************/
program circuit_recognition (input,output);

const
  b1    = 0;
  maxm  = 30;
  maxdc = 10;
  maxhi = 10;
  maxsc = 10;
  maxcm = 10;

var
  transistor: array [1..maxm] of
                record
                  m : integer;
                  d : integer;
                  g : integer;
                  s : integer;
                  u : integer;
                  sw: 0..1
                end;
  p1, p2, b2, c1, c2, o, i1, i2, p2m1, p2m2 : integer;
  dc : array [1..maxdc] of integer;
  hi : array [1..maxhi] of integer;
  sc : array [1..maxsc] of integer;
  cm : array [1..maxcm] of integer;
  p4m : array [1..maxm] of integer;
```

```
  nmrofm   : 0..maxm;
  nmrofdc  : 0..maxdc;
  nmrofhi  : 0..maxhi;
  nmrofsc  : 0..maxsc;
  nmrofcm  : 0..maxcm;
  nmrofp4m : 0..maxm;

procedure m1;                    {basic circuit note extraction}
  var
    one_char: char;
    entry, v1, v2, i: integer;
  begin
  for i := 1 to 60 do
    begin
    read(one_char);
    write(one_char)
    end;
  readln;
  writeln;
  writeln;
  entry := 0 ;
  while not eof do
    begin
    read(one_char);
    case one_char of
      '*' : readln;
      ' ' : readln;
      'M' : begin
            entry := entry + 1;
            with transistor[entry] do
              begin
              readln(m,d,g,s,u);
              sw := 0 ;
              writeln('M',m:2,d:4,g:4,s:4,u:4)
              end
            end;
      'V' : begin
            read(one_char);
            if one_char = 'D' then
               readln(one_char,p1)
            else if one_char = 'S' then
```

```
                readln(one_char,p2)
              else
                begin
                read(one_char);
                read(one_char);
                if one_char = '+' then
                   readln(i1)
                else
                   readln(i2)
                end;
          end;
      'I' : begin
          read(one_char);
          read(one_char);
          read(one_char);
          read(one_char);
          readln(v1,v2);
          if v1 = p1 then
             b2 := v2
          else
             b2 := v1;
          end;

      'C' : begin
          read(one_char);
          if one_char = 'L' then
             readln(o)
          else
             readln(c1,c2);
          end
    end
    end;
  nmrofm := entry ;
  writeln('P1 = ', p1:4,'   P2 = ', p2:4);
  writeln('I1 = ', i1:4,'   I2 = ', i2:4);
  writeln('C1 = ', c1:4,'   C2 = ', c2:4);
  writeln('B1 = ', b1:4,'   B2 = ', b2:4);
  writeln('O  = ', o:4)
  end;

procedure m2;                    {dc node recognition}
```

```
  var
    entry, entry2 : integer;
    dropit : boolean ;
  begin
  nmrofdc := 0 ;
  for entry := 1 to nmrofm do
    with transistor[entry] do
      if d = g then
         begin
         dropit := false ;
         for entry2 := 1 to nmrofdc do
           if d = dc[entry2] then
             dropit := true;
         if dropit = false then
            begin
            nmrofdc := nmrofdc + 1 ;
            dc[nmrofdc] := d
            end;
         end;
  writeln;
  for entry := 1 to nmrofdc do
    writeln('dc(',entry:2,') = ',dc[entry]:4);
  end;

procedure m3;                    {hi node recognition}
  var
    entry, entry2, count : integer;
    dropit : boolean;
  begin
  nmrofhi := 0 ;
  for entry := 1 to nmrofm do
    begin
    count := 0 ;
    for entry2 := 1 to nmrofm do
      if transistor[entry].d = transistor[entry2].d then
         count := count + 1 ;
    if count > 1 then
      begin
      dropit := false ;
      for entry2 := 1 to nmrofdc do
        if transistor[entry].d = dc[entry2] then
```

```
            dropit := true;
        if dropit = false then
          for entry2 := 1 to nmrofm  do
            if transistor[entry].d = transistor[entry2].s then
              dropit := true;
        if dropit = false then
          for entry2 := 1 to nmrofhi do
            if transistor[entry].d = hi[entry2] then
              dropit := true;
        if dropit = false then
          begin
          nmrofhi := nmrofhi + 1 ;
          hi[nmrofhi] := transistor[entry].d
          end;
        end;
      end;
    writeln;
    for entry := 1 to nmrofhi do
      writeln('hi(',entry:2,') = ',hi[entry]:4);
    end;

procedure m4;                          {sc node recognition}
    var
      entry, entry2, entry3 : integer ;
      dropit : boolean ;
    begin
    nmrofsc := 0 ;
    for entry := 1 to nmrofm do
      begin
      entry2 := entry + 1 ;
      while entry2 <= nmrofm do
        begin
        if transistor[entry].s = transistor[entry2].s then
          begin
          entry2 := nmrofm + 1 ;
          dropit := false ;
          for entry3 := 1 to nmrofsc do
            if transistor[entry].s = sc[entry3] then
              dropit := true;
          if dropit = false then
            begin
```

```
            nmrofsc := nmrofsc + 1 ;
            sc[nmrofsc] := transistor[entry].s
            end;
          end
        else
          entry2 := entry2 + 1;
        end;
      end;
    writeln;
    for entry := 1 to nmrofsc do
      writeln('sc(',entry:2,') = ',sc[entry]:4);
    end;

procedure m5;                    {cm node recognition}
    var
      entry, entry2, entry3 : integer ;
      dropit : boolean;
    begin
    nmrofcm := 0 ;
    for entry := 1 to nmrofm do
      begin
      if (transistor[entry].s = p1) or
         (transistor[entry].s = p2) then
        begin
        entry2 := entry + 1 ;
        while entry2 <= nmrofm do
          begin
          if (transistor[entry].s = transistor[entry2].s) and
             (transistor[entry].g = transistor[entry2].g) then
            begin
            entry2 := nmrofm + 1 ;
            dropit := false ;
            for entry3 := 1 to nmrofcm do
              if transistor[entry].g = cm[entry3] then
                 dropit := true;
            if dropit = false then
              begin
              nmrofcm := nmrofcm + 1 ;
              cm[nmrofcm] := transistor[entry].g
              end;
            end;
```

```
        entry2 := entry2 + 1;
        end;
      end;
    end;
  writeln;
  for entry := 1 to nmrofcm do
    writeln('cm(',entry:2,') = ',cm[entry]:4);
  end;

procedure m6;                    {primitive p1 recognition}
  var
    entry, entry2, entry3  : integer;
    dropit : boolean;
  begin
  for entry := 1 to nmrofm do
    if transistor[entry].g = i1 then
      for entry2 := 1 to nmrofm do
        if (transistor[entry2].g = i2) and
           (transistor[entry].s = transistor[entry2].s) then
          begin
          dropit := true ;
          for entry3  := 1 to nmrofsc do
            if transistor[entry].s = sc[entry3] then
              dropit := false;
          if dropit = false then
            begin
            writeln;
            write('p1 : (m',transistor[entry].m:2);
            writeln(', m',transistor[entry2].m:2,')');
            transistor[entry].sw := 1 ;
            transistor[entry2].sw := 1
            end;
          end;
  end;

procedure m7;                    {primitive p2 recognition}
  var
    entry, entry2, entry3, count  : integer;
    dropit : boolean;
  begin
  writeln;
```

```
for entry := 1 to nmrofm do
  begin
  count := 1 ;
  for entry3 := 1 to nmrofm do
    if (entry <> entry3) and
       (transistor[entry].g = transistor[entry3].g) then
       begin
       entry2 := entry3 ;
       count := count + 1 ;
       end;
  if count = 2 then
      begin
      dropit := true ;
      for entry3  := 1 to nmrofcm do
        if transistor[entry].g = cm[entry3] then
          dropit := false;
      if dropit = false then
        begin
        dropit := true ;
        for entry3  := 1 to nmrofdc do
          if transistor[entry].d = dc[entry3] then
            dropit := false;
        end;
      if dropit = false then
        begin
        dropit := true ;
        for entry3  := 1 to nmrofhi do
          if (transistor[entry2].d = hi[entry3]) and
             (hi[entry3] <> o ) then
            dropit := false;
        end;
      if dropit = false then
        begin
        write('p2 : (m',transistor[entry].m:2);
        writeln(', m',transistor[entry2].m:2,')');
        transistor[entry].sw := 1 ;
        transistor[entry2].sw := 1 ;
        p2m1 := transistor[entry].m ;
        p2m2 := transistor[entry2].m ;
        end;
      end;
```

```
    end;
  end;

procedure m8;                    {primitive p3 recognition}
  var
    entry, entry2, count, e1, m1 : integer;
    dropit : boolean;
  begin
  writeln;
  for entry := 1 to nmrofcm do
    begin
    count := 0 ;
    for entry2 := 1 to nmrofm do
      begin
      if transistor[entry2].g = cm[entry] then
        begin
        count := count + 1;
        if count = 1 then
          begin
          m1 := transistor[entry2].m;
          e1 := entry2
          end;
        if count = 2 then
          begin
          dropit := false ;
          if ((m1 = p2m1) and (transistor[entry2].m = p2m2)) or
             ((m1 = p2m2) and (transistor[entry2].m = p2m1)) then
             dropit := true ;
          if dropit = false then
            begin
            write('p3 (cm',entry:2,') : m',m1:2,', m');
            write(transistor[entry2].m:2);
            transistor[e1].sw := 1 ;
            transistor[entry2].sw := 1
            end;
          end;
        if count > 2 then
          begin
          write(', m',transistor[entry2].m:2);
          transistor[entry2].sw := 1
          end;
```

```
        end;
      end;
    writeln
    end;
  end;

procedure m9;                    {primitive p4 recognition}
  var
    entry, entry2, entry3, count : integer;
    dropit : boolean;
    nmrofp4g : 0..maxm;
    p4g : array [1..maxm] of integer;
  begin
  nmrofp4g := 0 ;
  nmrofp4m := 0 ;
  for entry := 1 to nmrofm do
    begin
    count := 1 ;
    for entry2 :=  1 to nmrofm do
      begin
      if (entry <> entry2) and
         (transistor[entry].g = transistor[entry2].g) and
         (transistor[entry].g <> b1) then
        begin
        dropit := false ;
        for entry3 := 1 to nmrofcm do
          if transistor[entry].g = cm[entry3] then
            dropit := true;
        if (dropit = false) and (count = 1) then
          for entry3 := 1 to nmrofp4g do
            if transistor[entry].g = p4g[entry3] then
              dropit := true;
        if dropit = false then
          begin
          count := count + 1;
          if count = 2 then
            begin
            writeln;
            write('p4 : m',transistor[entry].m:2);
            nmrofp4g := nmrofp4g + 1 ;
            p4g[nmrofp4g] := transistor[entry].g ;
```

```
            nmrofp4m := nmrofp4m + 1 ;
            p4m[nmrofp4m] := transistor[entry].m ;
            transistor[entry].sw := 1
            end;
          write(', m',transistor[entry2].m:2);
          nmrofp4m := nmrofp4m + 1 ;
          p4m[nmrofp4m] := transistor[entry2].m ;
          transistor[entry2].sw := 1
          end;
        end;
      end;
    end;
  writeln
  end;

procedure m10;                    {primitive p5 recognition}
  var
    entry, entry2 : integer;
    dropit : boolean ;
  begin
  writeln;
  dropit := true ;
  for entry := 1 to nmrofhi do
    if hi[entry] = 0 then
      dropit := false;
  if dropit = false then
    for entry := 1 to nmrofm do
      if transistor[entry].d = 0 then
        begin
        dropit := true ;
        for entry2 := 1 to nmrofhi do
          if (transistor[entry].g = hi[entry2]) and
             (transistor[entry].g <> 0)          then
            dropit := false ;
        if dropit = false then
          begin
          writeln('p5 : m',transistor[entry].m:2);
          transistor[entry].sw := 1
          end;
        end;
  end;
```

```
procedure m11;                    {primitive p5c recognition}
  var
    entry, entry2 : integer;
    dropit : boolean ;
  begin
  dropit := true ;
  for entry := 1 to nmrofhi do
    if hi[entry] = 0 then
      dropit := false;
  if dropit = false then
    for entry := 1 to nmrofm do
      if transistor[entry].d = 0 then
        begin
        dropit := false;
        for entry2 := 1 to nmrofcm do
          if transistor[entry].g = cm[entry2]  then
            dropit := true  ;
        if dropit = false then     {check duplicate with p5}
          for entry2 := 1 to nmrofhi do
            if transistor[entry].g = hi[entry2]  then
              dropit := true  ;
        if dropit = false then     {check duplicate with p4}
          for entry2 := 1 to nmrofp4m do
            if transistor[entry].m = p4m[entry2]  then
              dropit := true  ;
        if dropit = false then
          begin
          writeln('p5c: m',transistor[entry].m:2);
          transistor[entry].sw := 1
          end;
        end;
  end;

procedure m12;                    {primitive p6 recognition}
  var
    entry, entry2 : integer;
  begin
  writeln;
  for entry := 1 to nmrofm do
    begin
    if transistor[entry].s = c1 then
```

```
      for entry2 := 1 to nmrofhi do
        if transistor[entry].d = hi[entry2] then
          begin
          writeln('p6 : m',transistor[entry].m:2,', cc');
          transistor[entry].sw := 1
          end;
    if transistor[entry].s = c2 then
      for entry2 := 1 to nmrofhi do
        if transistor[entry].d = hi[entry2] then
          begin
          writeln('p6 : m',transistor[entry].m:2,', cc');
          transistor[entry].sw := 1
          end;
    if transistor[entry].d = c1 then
      for entry2 := 1 to nmrofhi do
        if transistor[entry].s = hi[entry2] then
          begin
          writeln('p6 : m',transistor[entry].m:2,', cc');
          transistor[entry].sw := 1
          end;
    if transistor[entry].d = c2 then
      for entry2 := 1 to nmrofhi do
        if transistor[entry].s = hi[entry2] then
          begin
          writeln('p6 : m',transistor[entry].m:2,', cc');
          transistor[entry].sw := 1
          end;
    end;
  end;

procedure m13;                    {primitive p7 recognition}
  var
    entry, entry2 : integer ;
    eqdc : boolean ;
  begin
  writeln;
  for entry := 1 to nmrofm do
    with transistor[entry] do
      if sw = 0 then
        begin
        eqdc := false;
```

```
        for entry2 := 1 to nmrofdc do
          if g = dc[entry2] then
            eqdc := true;
        if (g = b1) or
           (g = b2) or
           (eqdc = true ) then
          begin
          writeln('p7 : m',m:2);
          sw := 1
          end;
        end;
  end;

procedure m14;                    {check unrecognized transistors}
  var
    entry : integer ;
  begin
  writeln;
  for entry := 1 to nmrofm do
    with transistor[entry] do
      if sw = 0 then
        writeln('unrecognized transistor : m',m:2);
  end;

begin {main routine}
  m1;                   {basic circuit note extraction}
  m2;                   {dc node recognition}
  m3;                   {hi node recognition}
  m4;                   {sc node recognition}
  m5;                   {cm node recognition}
  m6;                   {circuit primitive recognition}
  m7;
  m8;
  m9;
  m10;
  m11;
  m12;
  m13;
  m14;
end.
```

```
/*****************************************************/
/***************  netlist.c  ************************/
/*****************************************************/

#include "netlist.h"

  main()
{
  int                   row,column;
  double                 sqrt();
  struct BLOCK             sop;
  int                   i,j,k,m,p;
  int                   arg,index,rownet[3][20];
  int                   count[3];
  int                   wmax,lmax,row_L[3];
  char                   input_string[30];
  FILE                   *fp1,*fopen();
  struct INDIVIDUAL_TX   tx[100];
  struct PRIMITIVE_CELL  cell[60];
  int                   tx_num,cel_num,mos;
  float                 aspect,area;
  float                 blk_area();
  struct NETBOX           *net[100];
  char                  *malloc();
  LINK                    head[50],temp;
  int                   Ncount,count_list();
  int                   net_length(),length;
  int                   net_name[50];
/*********************************************/
  sop.name="sop";
/*********************************************/
  if ((fp1=fopen("recog_module","r")) ==NULL){
    printf("0rror: Cannot open recog_module");
    exit(1);
  }
  while (fscanf(fp1, "%s", input_string) != EOF) {
   if(strcmp(input_string, "tx_num") == 0){
      fscanf(fp1, "%d", &tx_num);
      for(i=0; i<tx_num; i++){
      fscanf(fp1, "%s", (tx[i].name));
      fscanf(fp1, "%d", &(tx[i].drn.net));
```

```
  fscanf(fp1, "%d", &(tx[i].gat.net));
  fscanf(fp1, "%d", &(tx[i].src.net));
  fscanf(fp1, "%d", &(tx[i].sub.net));
  fscanf(fp1, "%d", &(tx[i].L));
  fscanf(fp1, "%d", &(tx[i].W));
  }/* end of for i */
}/* end of if */
if(strcmp(input_string, "net_num") == 0){
  fscanf(fp1, "%d", &(sop.net_num));
  for(i=0; i<sop.net_num; i++){
  fscanf(fp1, "%d", &(net_name[i]));
  }
}
if(strcmp(input_string, "input_net") == 0){
  fscanf(fp1, "%d", &(sop.input_net_num));
  for(i=0; i<sop.input_net_num; i++){
  fscanf(fp1, "%d", &(sop.input_net_list[i]));
  }
}
if(strcmp(input_string, "output_net") == 0){
  fscanf(fp1, "%d", &(sop.output_net_num));
  for(i=0; i<sop.output_net_num; i++){
  fscanf(fp1, "%d", &(sop.output_net_list[i]));
  }
}
if(strcmp(input_string, "sensitive_net") == 0){
  fscanf(fp1, "%d", &(sop.sensitive_net_num));
  for(i=0; i<sop.sensitive_net_num; i++){
  fscanf(fp1, "%d", &(sop.sensitive_net_list[i]));
  }
}
if(strcmp(input_string, "cel_num") == 0){
  fscanf(fp1, "%d", &(sop.cel_num));
  for(i=0; i<sop.cel_num; i++){
  fscanf(fp1, "%s ", (sop.cell[i].name));
  fscanf(fp1, "%c", &(sop.cell[i].type));
  fscanf(fp1, "%d", &(sop.cell[i].tx_num));
  for(j=0; j<(sop.cell[i].tx_num); j++){
    fscanf(fp1, "%d", &mos);
    sop.cell[i].tx[j]=tx[mos];
  }/* end of for j */
```

```
      }/* end of for i */
    }/* end of if */
    if(strcmp(input_string, "matrix") == 0){
      fscanf(fp1, "%d", &(sop.row));
      fscanf(fp1, "%d", &(sop.column));
      for(j=0; j<sop.column; j++){
      for(i=0; i<sop.row; i++){
        fscanf(fp1, "%d ", &(sop.matrix[i][j][0]));
        for(k=1; k <= sop.matrix[i][j][0]; k++){
          fscanf(fp1, "%d ", &(sop.matrix[i][j][k]));
        }/* end of for k */
      }/* end of for i */
      }/* end of for j */
    }/* end of if */
    if(strcmp(input_string, "aspect") == 0){
      fscanf(fp1, "%f", &aspect);
    }
  }/* end of while */
/*********************************************************/


/*********************************************************/
  for(i=0; i < sop.net_num;i++){
    Ncount = 0;
    for(j = 0;j<sop.cel_num;j++){
      for(k = 0;k<sop.cell[j].tx_num;k++){
      if(sop.cell[j].tx[k].drn.net == net_name[i]){
        Ncount++;
        head[Ncount]  =  (LINK) malloc(sizeof(ELEMENT));
        head[Ncount]->x = sop.cell[j].tx[k].drn.x;
        head[Ncount]->y = sop.cell[j].tx[k].drn.y;
        head[Ncount]->dsg = 1;
        head[Ncount]->cel = j;
        head[Ncount]->tx  = k;
      }
      if(sop.cell[j].tx[k].gat.net == net_name[i]){
        Ncount++;
        head[Ncount]  =  (LINK) malloc(sizeof(ELEMENT));
        head[Ncount]->x = sop.cell[j].tx[k].gat.x;
        head[Ncount]->y = sop.cell[j].tx[k].gat.y;
        head[Ncount]->dsg = 3;
```

```
        head[Ncount]->cel = j;
        head[Ncount]->tx  = k;
      }

      if(sop.cell[j].tx[k].src.net == net_name[i]){
        Ncount++;
        head[Ncount]  =  (LINK) malloc(sizeof(ELEMENT));
        head[Ncount]->x = sop.cell[j].tx[k].src.x;
        head[Ncount]->y = sop.cell[j].tx[k].src.y;
        head[Ncount]->dsg = 2;
        head[Ncount]->cel = j;
        head[Ncount]->tx  = k;
      }
      }
    }
    net[i] = head[1];
    for(m = 1;m<Ncount;m++) head[m]->next = head[m+1];
    head[Ncount]->next = NULL;
  }
/*****************************************/
  for(i = 0;i < sop.net_num;i++){
    temp = net[i];
    do{
      printf("%d,%d,%d,%d0,net_name[i],temp->dsg,temp->cel,temp->tx);
      temp = temp->next;
    } while(temp !=  NULL);
  }
/*****************************************/

}/* end of main */



/* Function*/


/* Count a list recursively. */

int count_list(head)
LINK  head;
{
```

```
  if (head == NULL)
    return(0);
  else
    return(1+ count_list(head->next));
}

/*****************************************************/
```

```
/*******************************************************/
/******************  netlist.h  ********************/
/*******************************************************/

#include  <stdio.h>
#include  <ctype.h>
#include  <strings.h>
#include  <sys/file.h>
#include  <sys/types.h>

#define    dc2py      1 /* diff contact to poly gate (edge to edge) */
#define    mc         4 /* metal contact */
#define    halfmc     2 /* 0.5 * mc */
#define    wc2dc      4 /* well contact to diff contact (edge to edge) */
#define    pyw        2 /* poly extensin to diff */
#define    mc2mc      4 /* metal contact spacing (edge to edge) */
#define    dc2dc      4 /* diff contact to diff contact (edge to edge) */
#define    tox    430.0 /* oxide thickness */

#define    MAX(x,y)  (((x) >= (y)) ? (x) : (y))
#define    MIN(x,y)  (((x) <= (y)) ? (x) : (y))
#define    MOD(x)    (((x) >= 0) ? ((int)((x/8.0)+0.5))
                : ((int)((x/8.0)-0.5)))

struct POINT {
  int  x;
  int  y;
};

struct LINE {
  struct POINT  p1;
  struct POINT  p2;
  int           material;
};

struct RECT {
  int  llx;
  int  lly;
  int  urx;
  int  ury;
};
```

```
struct FORM {
  int t1;
  int t2;
  int t3;
  int t4;
  int t5;
  int t6;
};

struct TRAN {
  char          *name;
  int           timestamp;
  int           pinnum;
  int           diode;
  struct POINT dm,gat,src;
  struct RECT   box;
};

struct CELL {
  char          *name;
  int           timestamp;
  int           pinnum;
  int           txnum;
  struct POINT dm[10],gat[10],src[10];
  struct RECT   box;
};

struct PIN {
  int                   net;
  int                   priority;
  int                   x;
  int                   y;
};
struct INDIVIDUAL_TX {
  char                  name[10];
  int                   W;
  int                   L;
  int                   par;
  struct PIN            dm,gat,src,sub;
};
struct PRIMITIVE_CELL {
```

```
  char                name[10];
  char                type;
  int                 tx_num;
  int                 W;
  int                 L;
  struct INDIVIDUAL_TX tx[10];
  struct POINT          origin;
  int                 orient;
};

struct BLOCK {
  char                *name;
  int                 timestamp;
  int                 net_num;
  int                 input_net_num;
  int                 input_net_list[20];
  int                 output_net_num;
  int                 output_net_list[5];
  int                 sensitive_net_num;
  int                 sensitive_net_list[5];
  int                 cel_num;
  int                 W;
  int                 L;
  int                 row,column;
  int                 matrix[3][3][3];
  struct PRIMITIVE_CELL cell[60];
};

struct NETBOX {
  struct NETBOX       *next;
  int x;
  int y;
  int tx;
  int dsg;                          /* drain=1 source=2 gate=3 */
  int cel;
};

typedef struct NETBOX  ELEMENT;
typedef ELEMENT *      LINK;
```

```
/*********************************************************/
/*********************  macro.h  *************************/
/*********************************************************/

#include  <stdio.h>
#include  <ctype.h>
#include  <strings.h>
#include  <sys/file.h>
#include  <sys/types.h>

#define    dc2py      1 /* diff contact to poly gate (edge to edge) */
#define    mc         4 /* metal contact */
#define    halfmc     2 /* 0.5 * mc */
#define    wc2dc      4 /* well contact to diff contact (edge to edge) */
#define    pyw        2 /* poly extensin to diff */
#define    mc2mc      4 /* metal contact spacing (edge to edge) */
#define    dc2dc      4 /* diff contact to diff contact (edge to edge) */
#define    tox   430.0 /* oxide thickness */

#define    MAX(x,y)  (((x) >= (y)) ? (x) : (y))
#define    MIN(x,y)  (((x) <= (y)) ? (x) : (y))
#define    MOD(x)    (((x) >= 0) ? ((int)((x/8.0)+0.5))
                : ((int)((x/8.0)-0.5)))

struct POINT {
  int  x;
  int  y;
};

struct LINE {
  struct POINT  p1;
  struct POINT  p2;
  int           material;
};

struct RECT {
  int  llx;
  int  lly;
  int  urx;
  int  ury;
};
```

```
struct FORM {
  int t1;
  int t2;
  int t3;
  int t4;
  int t5;
  int t6;
};

struct TRAN {
  char          *name;
  int           timestamp;
  int           pinnum;
  int           diode;
  struct POINT drn,gat,src;
  struct RECT   box;
};

struct CELL {
  char          *name;
  int           timestamp;
  int           pinnum;
  int           txnum;
  struct POINT drn[10],gat[10],src[10];
  struct RECT   box;
};

struct BLOCK {
  char          *name;
  int           timestamp;
  int           pinnum;
  int           celnum;
  struct POINT vdd,vss,in[20],out[20],drn[20],gat[20],src[20];
  struct RECT   box;
};
```

```
/*************************************************/
/*****************  modgen.c  ********************/
/*************************************************/

#include "macro.h"

main()
{
  char *name;
  int   i,signalin,signalin_list[20];
  int   signalout,signalout_list[10];
  int   net,pin,list[200][4];
  int   sen,sen_list[10];
  int   right,lower,left,upper;
  struct BLOCK     b1,module();

  name="bk1.mag";
  printf("Please enter the input signal number:\n");
  scanf(" %d",&signalin);
  printf("Please enter the input signal name:\n");
    for(i=0;i<signalin;i++){
      scanf(" %d",&signalin_list[i]);
    }
  printf("Please enter the output signal number:\n");
  scanf(" %d",&signalout);
  printf("Please enter the output signal name:\n");
    for(i=0;i<signalout;i++){
      scanf(" %d",&signalout_list[i]);
    }

  printf("Please enter the internal net number:\n");
  scanf(" %d",&net);
  printf("Please enter the pin number:\n");
  scanf(" %d",&pin);
  for(i=0;i<pin;i++){
    printf("Please enter the net list of #%d pin:[net_num,drn(1)/src(2)/
            gat(3),cel_num,tx_num]\n",i);
    scanf(" %d,%d,%d,%d",&list[i][0],&list[i][1],&list[i][2],&list[i][3]);
  }
  printf("Please enter the sensitive net number:\n");
  scanf(" %d",&sen);
```

```
  printf("Please enter the sensitive net name:h");
  for(i=0;i<sen;i++){
    scanf(" %d",&sen_list[i]);
  }
  printf("Please enter the right_offset:h");
  scanf(" %d",&right);
  printf("Please enter the lower_offset:h");
  scanf(" %d",&lower);
  printf("Please enter the left_offset:h");
  scanf(" %d",&left);
  printf("Please enter the upper_offset:h");
  scanf(" %d",&upper);

  b1=module(name,signalin,signalin_list,signalout,signalout_list,net,list,
     pin,sen,sen_list,right,lower,left,upper);
}


struct BLOCK module(fname,insignal_num,insignal_net,outsignal_num,
       outsignal_net,net_num,netlist,pin_num,sen_num,sen_net,right_offset,
       lower_offset,left_offset,upper_offset)
char  *fname;
int    insignal_num,outsignal_num,net_num,pin_num,sen_num;
int    right_offset,lower_offset,left_offset,upper_offset;
int    netlist[200][4],insignal_net[20],outsignal_net[10],sen_net[10];
{
  FILE           *fp1,*fopen();
  int            i,j,cel_num,degree,refer,offset;
  int            timestamp;
  int            borderllx,borderlly,borderlrx,borderlry;
  int            borderurx,borderury,borderulx,borderuly;
  extern time_t  time();
  char           *infile,bk_name[10],cel_name[80][10],m_name[80][10];
  char           change,dir,align;
  struct POINT   coordinate,place();
  struct FORM    transform[100],rotate(),upsidedown();
  struct RECT    box[100];
  struct CELL    *cel_pt[100];
  struct CELL    gen_difin(),gen_cm(),gen_cap(),cel[100];
  struct BLOCK   bk;
  int            dummy,vias_num,wires_num,mighty_does_work;
```

```
  char            cel_type[10],input_string[20];
  struct POINT    vias[200];
  struct LINE     wires[500];
/*****************************************************************/
  for(j=0;j<10;j++){
          if(fname[j] != '.') bk_name[j]=fname[j];
        else {
          bk_name[j]=' ';
          break;
        }
      }
  bk.name=bk_name;
  bk.pinnum=0;
  printf("How many primitive cells are included in this block (%s):\n",bk.name);
  scanf("%d",&cel_num);
  bk.celnum=cel_num;
  printf("Please tell me the first cell name [~.mag], its llx,lly will be
         align to (0,0):\n");
  scanf(" %s",cel_name[0]);
  printf("What type is this cell ? [difin/tx/cm/cap]:\n");
  scanf(" %s",cel_type);
  if( strcmp(cel_type, "difin") == 0 ) {
    cel[0]=gen_difin(cel_name[0]);
  }
  if( strcmp(cel_type, "cm") == 0 ) {
    cel[0]=gen_cm(cel_name[0]);
  }
  if( strcmp(cel_type, "cap") == 0 ) {
    cel[0]=gen_cap(cel_name[0]);
  }
  box[0]=cel[0].box;
  cel_pt[0] = &cel[0];
  transform[0].t1 = 1;
  transform[0].t2 = 0;
  transform[0].t3 = 0;
  transform[0].t4 = 0;
  transform[0].t5 = 1;
  transform[0].t6 = 0;
/*****************************************************************/
  printf("Do you want do (rotation), (upsidedown), or (no change)
         on this cell? [r/u/n]:\n");
```

```
  scanf(" %c",&change);

if (change=='r'){
  printf("How many degrees? [0/90/180/270]:h");
  scanf(" %d",&degree);
  transform[0]=rotate(cel_pt[0],degree);
}
if (change=='u'){
  transform[0]=upsidedown(cel_pt[0]);
}
  transit(cel_pt[0],0,0);
  transform[0].t3 = transform[0].t3 - box[0].llx + 0;
  transform[0].t6 = transform[0].t6 - box[0].lly + 0;

/*********************************************************************/
for(i=1;i < cel_num;++i){
  printf("Please enter the next cell name:");
  scanf("%s",cel_name[i]);
  printf("What type is this cell ? [difin/tx/cm/cap]:h");
  scanf(" %s",cel_type);
  if( strcmp(cel_type, "difin") == 0 ) {
    cel[i]=gen_difin(cel_name[i]);
  }
  if( strcmp(cel_type, "cm") == 0 ) {
    cel[i]=gen_cm(cel_name[i]);
  }
  if( strcmp(cel_type, "cap") == 0 ) {
    cel[i]=gen_cap(cel_name[i]);
  }
  box[i]=cel[i].box;
  cel_pt[i] = &cel[i];
  transform[i].t1 = 1;
  transform[i].t2 = 0;
  transform[i].t3 = 0;
  transform[i].t4 = 0;
  transform[i].t5 = 1;
  transform[i].t6 = 0;
  printf("Do you want do (rotation), (upsidedown), or (no change)
          on cell [%s]? [r/u/n]:h",cel_name[i]);
  scanf(" %c",&change);
  if (change=='r'){
```

```
    printf("How many degrees? [0/90/180/270]:\n");
    scanf(" %d",&degree);
    transform[i]=rotate(cel_pt[i],degree);
  }
  if (change=='u'){
    transform[i]=upsidedown(cel_pt[i]);
  }

  printf("Enter the following placement informations for cell [%s]:
        \n",cel_name[i]);
  printf("reference cell name, direction [E/W/N/S], align [l(lower)
        /u(upper)/l(left)/r(right)], offset:\n");
  scanf("  %d,%c,%c,%d",&refer,&dir,&align,&offset);
  coordinate = place(cel_pt[i],cel_pt[refer],dir,align,offset);
  transform[i].t3 = transform[i].t3 - box[i].llx + coordinate.x;
  transform[i].t6 = transform[i].t6 - box[i].lly + coordinate.y;
}
/*****************************************************************/
  bk.box.llx = cel_pt[0]->box.llx;
  bk.box.lly = cel_pt[0]->box.lly;
  bk.box.urx = cel_pt[0]->box.urx;
  bk.box.ury = cel_pt[0]->box.ury;
  bk.pinnum  = cel_pt[0]->pinnum;
  for(j=1;j<cel_num;j++){
    bk.box.llx = MIN(bk.box.llx, cel_pt[j]->box.llx);
    bk.box.lly = MIN(bk.box.lly, cel_pt[j]->box.lly);
    bk.box.urx = MAX(bk.box.urx, cel_pt[j]->box.urx);
    bk.box.ury = MAX(bk.box.ury, cel_pt[j]->box.ury);
    bk.pinnum  = bk.pinnum + cel_pt[j]->pinnum;
  }

  borderllx=bk.box.llx - left_offset*(mc+mc2mc);
  borderlly=bk.box.lly - lower_offset*(mc+mc2mc);
  borderlrx=bk.box.urx + right_offset*(mc+mc2mc);
  borderlry=bk.box.lly - lower_offset*(mc+mc2mc);
  borderurx=bk.box.urx + right_offset*(mc+mc2mc);
  borderury=bk.box.ury + upper_offset*(mc+mc2mc);
  borderulx=bk.box.llx - left_offset*(mc+mc2mc);
  borderuly=bk.box.ury + upper_offset*(mc+mc2mc);
```

```
/*********************************************************************/
  infile="mighty_in";
  if((fp1=fopen(infile,"w")) == NULL){
    printf("0an't open file name = %sh",infile);
    exit(1);
  }
/*********************************************************************/
  fprintf(fp1,"number_of_nets %dh",insignal_num+outsignal_num+net_num+2);
/*********************************************************************/
  fprintf(fp1,"rectagoncorners 4h");
  fprintf(fp1,"%d %dh",MOD(borderllx),MOD(borderlly));
  fprintf(fp1,"%d %dh",MOD(borderlrx),MOD(borderlry));
  fprintf(fp1,"%d %dh",MOD(borderurx),MOD(borderury));
  fprintf(fp1,"%d %dh",MOD(borderulx),MOD(borderuly));
  bk.vdd.x = MOD(borderllx)+(MOD(borderurx)-MOD(borderllx))/2;
  bk.vdd.y = MOD(borderury);
  bk.vss.x = MOD(borderllx)+(MOD(borderurx)-MOD(borderllx))/2;
  bk.vss.y = MOD(borderlly);
/* input/output signals come out from top */
/*  bk.in[0].x = MOD(borderllx)+10;
  bk.in[0].y = MOD(borderury);
  bk.in[2].x = MOD(borderllx)+20;
  bk.in[2].y = MOD(borderury);
  bk.in[1].x = MOD(borderllx)+30;
  bk.in[1].y = MOD(borderury);
  bk.in[6].x = MOD(borderllx)+40;
  bk.in[6].y = MOD(borderury);
  bk.in[7].x = MOD(borderllx)+50;
  bk.in[7].y = MOD(borderury);
  bk.in[8].x = MOD(borderllx)+60;
  bk.in[8].y = MOD(borderury);
  bk.in[4].x = MOD(borderllx);
  bk.in[4].y = MOD(borderury)-20;
  bk.in[3].x = MOD(borderllx);
  bk.in[3].y = MOD(borderury)-40;
  bk.in[5].x = MOD(borderllx);
  bk.in[5].y = MOD(borderury)-60;
  bk.in[9].x = MOD(borderllx)+20;
  bk.in[9].y = MOD(borderlly);
  bk.in[10].x = MOD(borderllx)+40;
  bk.in[10].y = MOD(borderlly);
```

```
  bk.in[11].x = MOD(borderllx)+60;
  bk.in[11].y = MOD(borderlly);
  bk.in[12].x = MOD(borderllx)+80;
  bk.in[12].y = MOD(borderlly);
  bk.out[0].x = MOD(borderurx)-20;
  bk.out[0].y = MOD(borderury);
  bk.out[1].x = MOD(borderurx)-40;
  bk.out[1].y = MOD(borderury);
*/
  bk.in[0].x = MOD(borderllx)+5;
    bk.in[0].y = MOD(borderury);
    for(i=1;i<insignal_num;i++){
      bk.in[i].x = bk.in[i-1].x+5;
      bk.in[i].y = MOD(borderury);
    }
  bk.out[0].x = MOD(borderurx)-5;
  bk.out[0].y = MOD(borderury);
  bk.out[1].x = bk.vdd.x-10;
  bk.out[1].y = bk.vdd.y;
  bk.out[2].x = bk.vdd.x+10;
  bk.out[2].y = bk.vdd.y;
  bk.out[3].x = bk.vdd.x-20;
  bk.out[3].y = bk.vdd.y;
  bk.out[4].x = bk.vdd.x+20;
  bk.out[0].y = bk.vdd.y;

/*  for(i=1;i<outsignal_num;i++){
    bk.out[i].x = bk.out[i-1].x-5;
    bk.out[i].y = MOD(borderury);
  } */
/*****************************************************************/
  fprintf(fp1,"number_of_pins %dn",pin_num+insignal_num+outsignal_num+2);
  fprintf(fp1,"1 %d %d 2h",bk.vdd.x,bk.vdd.y); /* net#1 is vdd */
  fprintf(fp1,"2 %d %d 2h",bk.vss.x,bk.vss.y); /* net#2 is vss */
  for(i=0;i<insignal_num;i++){
    fprintf(fp1,"%d %d %d 1h",insignal_net[i],bk.in[i].x,bk.in[i].y);
  } /* input signal net */
  for(j=0;j<outsignal_num;j++){
    fprintf(fp1,"%d %d %d 1h",outsignal_net[j],bk.out[j].x,bk.out[j].y);
  } /* output signal net */
  for(i=0;i<pin_num;i++){
```

```
    if (netlist[i][1]==1) /* drain port */
      fprintf(fp1,"%d %d %d 1h",netlist[i][0], MOD(cel_pt[netlist[i][2]]
      ->drn[netlist[i][3]].x), MOD(cel_pt[netlist[i][2]]->drn[netlist[i][3]].y));
    if (netlist[i][1]==2) /* source port */
      fprintf(fp1,"%d %d %d 1h",netlist[i][0], MOD(cel_pt[netlist[i][2]]
      ->src[netlist[i][3]].x), MOD(cel_pt[netlist[i][2]]->src[netlist[i][3]].y));
    if (netlist[i][1]==3) /* gate port */
      fprintf(fp1,"%d %d %d 1h",netlist[i][0], MOD(cel_pt[netlist[i][2]]
      ->gat[netlist[i][3]].x), MOD(cel_pt[netlist[i][2]]->gat[netlist[i][3]].y));
  }
/*****************************************************************/
  if(sen_num != 0){
    fprintf(fp1,"sensitive_nets %dh",sen_num);
    for(i=0;i<sen_num;i++) fprintf(fp1,"%d ",sen_net[i]);
    fprintf(fp1,"h");
  }
/*****************************************************************/
  fprintf(fp1,"obstacles %dh",cel_num * 8);
  for(i=0;i<cel_num;i++){
    fprintf(fp1,"%d %d %d %d 1h", MOD(cel_pt[i]->box.llx), MOD(cel_pt[i]
      ->box.lly), MOD(cel_pt[i]->box.urx), MOD(cel_pt[i]->box.lly));
    fprintf(fp1,"%d %d %d %d 2h", MOD(cel_pt[i]->box.llx), MOD(cel_pt[i]
      ->box.lly), MOD(cel_pt[i]->box.urx), MOD(cel_pt[i]->box.lly));
    fprintf(fp1,"%d %d %d %d 1h", MOD(cel_pt[i]->box.urx), MOD(cel_pt[i]
      ->box.lly), MOD(cel_pt[i]->box.urx), MOD(cel_pt[i]->box.ury));
    fprintf(fp1,"%d %d %d %d 2h", MOD(cel_pt[i]->box.urx), MOD(cel_pt[i]
      ->box.lly), MOD(cel_pt[i]->box.urx), MOD(cel_pt[i]->box.ury));
    fprintf(fp1,"%d %d %d %d 1h", MOD(cel_pt[i]->box.urx), MOD(cel_pt[i]
      ->box.ury), MOD(cel_pt[i]->box.llx), MOD(cel_pt[i]->box.ury));
    fprintf(fp1,"%d %d %d %d 2h", MOD(cel_pt[i]->box.urx), MOD(cel_pt[i]
      ->box.ury), MOD(cel_pt[i]->box.llx), MOD(cel_pt[i]->box.ury));
    fprintf(fp1,"%d %d %d %d 1h", MOD(cel_pt[i]->box.llx), MOD(cel_pt[i]
      ->box.ury), MOD(cel_pt[i]->box.llx), MOD(cel_pt[i]->box.lly));
    fprintf(fp1,"%d %d %d %d 2h", MOD(cel_pt[i]->box.llx), MOD(cel_pt[i]
      ->box.ury), MOD(cel_pt[i]->box.llx), MOD(cel_pt[i]->box.lly));
  }
/*****************************************************************/
  fclose(fp1);

/*****************************************************************/
  system("/home/pacific/cad/cadbin/mighty_wj mighty_in mighty_out");
```

```
  mighty_does_work=0;
/*********************************************************************/
  /* read in the routing results of mighty from "mighty_out" */
  if ((fp1 = fopen("mighty_out","r")) == NULL){ /* if cannot open */
      printf("0rror: cannot open mighty_out");
        exit(1);
    }
/*********************************************************************/
  while ( fscanf(fp1, "%s", input_string) != EOF) {
    if( strcmp(input_string, "vias") == 0 ) {
      mighty_does_work=1;
      fscanf(fp1, "%d", &vias_num);
      for(i=0;i<vias_num;i++) {
      fscanf(fp1, "%d", &dummy);
      fscanf(fp1, "%d", &vias[i].x);
      fscanf(fp1, "%d", &vias[i].y);
      fscanf(fp1, "%d", &dummy);
      fscanf(fp1, "%d", &dummy);
      }
    }
    if(strcmp(input_string, "wires") == 0 ) {
      fscanf(fp1, "%d", &wires_num);
      for(i=0;i<wires_num;i++) {
      fscanf(fp1, "%d", &dummy);
      fscanf(fp1, "%d", &wires[i].p1.x);
      fscanf(fp1, "%d", &wires[i].p1.y);
      fscanf(fp1, "%d", &wires[i].p2.x);
      fscanf(fp1, "%d", &wires[i].p2.y);
      fscanf(fp1, "%d", &wires[i].material);
      }
    }
    if(strcmp(input_string, ".end") == 0 ) break;
  }

  if(mighty_does_work ==0){
    printf("MIGHTY cannot get a reasonable results!h");
    printf("Please try again.h");
  }

/*********************************************************************/
  if((fp1=fopen(fname,"w")) == NULL){
```

```
    printf("0an't open file name = %sh",fname);
    exit(1);
  }

timestamp = time((time_t *) 0);
bk.timestamp=timestamp;
fprintf(fp1,"magich");
fprintf(fp1,"tech scmosh");
fprintf(fp1,"timestamp %dh",timestamp);

/*********************************************************************/
for(i=0;i<cel_num;i++){
  for(j=0;j<10;j++){
    if(cel_name[i][j] != '.') m_name[i][j]=cel_name[i][j];
    else break;
  }
  fprintf(fp1,"use %s %s_0h",m_name[i],m_name[i]);
  fprintf(fp1,"timestamp %dh",cel[i].timestamp);
  fprintf(fp1,"transform %d %d %d %d %d %dh",transform[i].t1,transform[i].t2,
      transform[i].t3,transform[i].t4,transform[i].t5,transform[i].t6);
  fprintf(fp1,"box %d %d %d %dh",box[i].llx,box[i].lly,box[i].urx,box[i].ury);
}
/*********************************************************************/
printf("vias_num=%d",vias_num);
  for(i=0;i<vias_num;i++){
    fprintf(fp1,"<< m2contact >>h");
    fprintf(fp1,"rect %d %d %d %dh",vias[i].x*8-halfmc,vias[i].y*8-halfmc,
      vias[i].x*8+halfmc,vias[i].y*8+halfmc);
}
/*********************************************************************/
printf("wires_num=%d",wires_num);
  for(i=0;i<wires_num;i++){
    if(wires[i].material == 1){
      fprintf(fp1,"<< metal1 >>h");
      fprintf(fp1,"rect %d %d %d %dh",wires[i].p1.x*8-halfmc,wires[i].p1.y*8
      -halfmc,wires[i].p2.x*8+halfmc,wires[i].p2.y*8+halfmc);
    }
    if(wires[i].material == 2){
      fprintf(fp1,"<< metal2 >>h");
      fprintf(fp1,"rect %d %d %d %dh",wires[i].p1.x*8-halfmc,wires[i].p1.y*8
      -halfmc,wires[i].p2.x*8+halfmc,wires[i].p2.y*8+halfmc);
```

```
    }
  }
/*************************************************************************/
fprintf(fp1,"<< metal2 >>\n");
fprintf(fp1,"rect %d %d %d %d\n",borderllx,borderlly-2*mc,borderlrx,borderlry);
      /* Vss line */
fprintf(fp1,"rect %d %d %d %d\n",borderulx,borderuly,borderurx,borderury+2*mc);
      /* Vdd line */

/*************************************************************************/
fprintf(fp1,"<< end >>\n");
/*************************************************************************/
  fclose(fp1);
  return(bk);
}
/*************************************************************************/
```

```
/*************************************************************************/
/* place.c is to place a cell w.r.t specified reference cell   */
/* return a struct POINT                                        */
/*                                                              */
/*************************************************************************/


struct POINT place(cell,refer,dir,align,offset)
struct CELL *cell, *refer;
char        dir,align;
int         offset;
{
struct POINT shift;


offset = offset * (mc + mc2mc);
/*************************************************************************/
switch (dir) {
```

```
case 'e' :  /* east of the reference cell */
case 'E' :  /* east of the reference cell */
  if (align == 'l'){
    shift.x = refer->box.urx + offset;
    shift.y = refer->box.lly;
    transit(cell,shift.x,shift.y);
  }
  if ((align == 'r')(align == 'u')){
    shift.x = refer->box.urx + offset;
    shift.y = refer->box.ury - (cell->box.ury - cell->box.lly);
    transit(cell,shift.x,shift.y);
  }
    break;
case 'w' :  /* west of the reference cell */
case 'W' :  /* west of the reference cell */
  if (align == 'l'){
    shift.x = refer->box.llx - (cell->box.urx - cell->box.llx) - offset;
    shift.y = refer->box.lly;
    transit(cell,shift.x,shift.y);
  }
  if ((align == 'r')(align == 'u')){
    shift.x = refer->box.llx - (cell->box.urx - cell->box.llx) - offset;
    shift.y = refer->box.ury - (cell->box.ury - cell->box.lly);
    transit(cell,shift.x,shift.y);
  }
    break;
case 'n' :  /* north of the reference cell */
case 'N' :  /* north of the reference cell */
  if (align == 'l'){
    shift.x = refer->box.llx;
    shift.y = refer->box.ury + offset;
    transit(cell,shift.x,shift.y);
  }
  if ((align == 'r')(align == 'u')){
    shift.x = refer->box.urx - (cell->box.urx - cell->box.llx);
    shift.y = refer->box.ury + offset;
    transit(cell,shift.x,shift.y);
  }
    break;
case 's' :  /* south of the reference cell */
case 'S' :  /* south of the reference cell */
```

```
    if (align == 'l'){
      shift.x = refer->box.llx;
      shift.y = refer->box.lly - offset - (cell->box.ury - cell->box.lly);
      transit(cell,shift.x,shift.y);
    }
    if ((align == 'r')(align == 'u')){
      shift.x = refer->box.urx - (cell->box.urx - cell->box.llx);
      shift.y = refer->box.lly - offset - (cell->box.ury - cell->box.lly);
      transit(cell,shift.x,shift.y);
    }
      break;
  }
/*****************************************************************/
    return(shift);
}



/*****************************************************************/
/* rotate.c is used to rotate a cell                             */
/* result : return a struct of FORM                              */
/*          pin location has been changed                        */
/*****************************************************************/


struct FORM rotate(cell,degree)
struct CELL *cell;
int         degree;
{
struct FORM transform;
int         i,dx,dy,sx,sy,gx,gy,tempx,tempy;
/*****************************************************************/

  switch (degree){
  case 0 :  /* rotate clockwise 0 degree */
    transform.t1 = 1;
    transform.t2 = 0;
    transform.t4 = 0;
    transform.t5 = 1;
    transform.t3 = 0;
    transform.t6 = 0;
```

```
  break;
case 90 : /* rotate clockwise 90 degree */
  transform.t1 = 0;
  transform.t2 = 1;
  transform.t4 = -1;
  transform.t5 = 0;
  transform.t3 = cell->box.llx-cell->box.lly;
  transform.t6 = cell->box.lly+cell->box.urx;
  for(i=0;i<cell->txnum;i++){
    dx = transform.t3 + cell->drn[i].y;
    dy = transform.t6 - cell->drn[i].x;
    cell->drn[i].x = dx;
    cell->drn[i].y = dy;
    sx = transform.t3 + cell->src[i].y;
    sy = transform.t6 - cell->src[i].x;
    cell->src[i].x = sx;
    cell->src[i].y = sy;
    gx = transform.t3 + cell->gat[i].y;
    gy = transform.t6 - cell->gat[i].x;
    cell->gat[i].x = gx;
    cell->gat[i].y = gy;
  }
  tempx = cell->box.urx;
  tempy = cell->box.ury;
  cell->box.urx = cell->box.llx + tempy - cell->box.lly;
  cell->box.ury = cell->box.lly + tempx - cell->box.llx;

  break;
case 180 :/* rotate clockwise 180 degree */
  transform.t1 = -1;
  transform.t2 = 0;
  transform.t4 = 0;
  transform.t5 = -1;
  transform.t3 = cell->box.llx+cell->box.urx;
  transform.t6 = cell->box.lly+cell->box.ury;
  for(i=0;i<cell->txnum;i++){
    dx = transform.t3-cell->drn[i].x;
    dy = transform.t6-cell->drn[i].y;
    cell->drn[i].x = dx;
    cell->drn[i].y = dy;
    sx = transform.t3-cell->src[i].x;
```

```
    sy = transform.t6-cell->src[i].y;
    cell->src[i].x = sx;
    cell->src[i].y = sy;
    gx = transform.t3-cell->gat[i].x;
    gy = transform.t6-cell->gat[i].y;
    cell->gat[i].x = gx;
    cell->gat[i].y = gy;
   }

   break;
 case 270 :/* rotate clockwise 270 degree */
   transform.t1 = 0;
   transform.t2 = -1;
   transform.t4 = 1;
   transform.t5 = 0;
   transform.t3 = cell->box.llx+cell->box.ury;
   transform.t6 = cell->box.lly-cell->box.llx;
   for(i=0;i<cell->txnum;i++){
    dx = transform.t3-cell->drn[i].y;
    dy = transform.t6+cell->drn[i].x;
    cell->drn[i].x = dx;
    cell->drn[i].y = dy;
    sx = transform.t3-cell->src[i].y;
    sy = transform.t6+cell->src[i].x;
    cell->src[i].x = sx;
    cell->src[i].y = sy;
    gx = transform.t3-cell->gat[i].y;
    gy = transform.t6+cell->gat[i].x;
    cell->gat[i].x = gx;
    cell->gat[i].y = gy;
   }
   tempx = cell->box.urx;
   tempy = cell->box.ury;
   cell->box.urx = cell->box.llx + tempy - cell->box.lly;
   cell->box.ury = cell->box.lly + tempx - cell->box.llx;

   break;
 }

/*****************************************************************/
 return(transform);
```

```
}


/*****************************************************************/
/* transit.c is used to make cell transit                        */
/* result : return VOID                                          */
/*          pin location has been changed                        */
/*****************************************************************/


transit(cell,x,y)
struct CELL *cell;
int          x,y;
{

int          i,dx,dy,sx,sy,gx,gy;
/*****************************************************************/

    for(i=0;i < cell->txnum;i++){
      dx = x - cell->box.llx + cell->drn[i].x;
      dy = y - cell->box.lly + cell->drn[i].y;
      cell->drn[i].x = dx;
      cell->drn[i].y = dy;
      sx = x - cell->box.llx + cell->src[i].x;
      sy = y - cell->box.lly + cell->src[i].y;
      cell->src[i].x = sx;
      cell->src[i].y = sy;
      gx = x - cell->box.llx + cell->gat[i].x;
      gy = y - cell->box.lly + cell->gat[i].y;
      cell->gat[i].x = gx;
      cell->gat[i].y = gy;
    }

    cell->box.urx = x + cell->box.urx - cell->box.llx;
    cell->box.ury = y + cell->box.ury - cell->box.lly;
    cell->box.llx = x;
    cell->box.lly = y;

/*****************************************************************/
```

```
}


/*****************************************************************/
/* upsidedown.c is used to make cell upsidedown                  */
/* result : return a struct of FORM                              */
/*          pin location has been changed                        */
/*****************************************************************/


struct FORM upsidedown(cell)
struct CELL *cell;
{
struct FORM transform;
int         i,dx,dy,sx,sy,gx,gy;
/*****************************************************************/

    transform.t1 = 1;
    transform.t2 = 0;
    transform.t4 = 0;
    transform.t5 = -1;
    transform.t3 = 0;
    transform.t6 = cell->box.lly+cell->box.ury;
    for(i=0;i<cell->txnum;i++){
      dx = transform.t3+cell->drn[i].x;
      dy = transform.t6-cell->drn[i].y;
      cell->drn[i].x = dx;
      cell->drn[i].y = dy;
      sx = transform.t3+cell->src[i].x;
      sy = transform.t6-cell->src[i].y;
      cell->src[i].x = sx;
      cell->src[i].y = sy;
      gx = transform.t3+cell->gat[i].x;
      gy = transform.t6-cell->gat[i].y;
      cell->gat[i].x = gx;
      cell->gat[i].y = gy;
    }

/*****************************************************************/
  return(transform);
```

```
}


/*****************************************************************/
/* gen_cap.c is used to generate capacitor                       */
/* return a struct CELL                                          */
/* a magic file will be generated.                               */
/*****************************************************************/


struct CELL gen_cap(fname)
char  *fname;

{
  FILE          *fp1,*fopen();
  int           i,j,area,width,length;
  int           x0,x1,x2,x3,x4;
  int           y0,y1,y2,y3,y4;
  char          cap_name[10];
  double        cap,unit;
  int           timestamp;
  extern time_t time();
  struct CELL   cap_cell;
/*****************************************************************/
  if((fp1=fopen(fname,"w")) == NULL){
    printf("0an't open file name = %sn",fname);
    exit(1);
  }
  for(j=0;j<10;j++){
    if(fname[j] != '.') cap_name[j]=fname[j];
    else {
      cap_name[j]=' ';
      break;
    }
  }
    cap_cell.name=cap_name;
/*****************************************************************/
  printf("Please enter the capacitance value[PF] in floating number formatn");
  scanf("%lf",&cap);
  unit=0.0003457*(1000.0/tox); /* tox=1000A SiO2 capacitance=0.3457 fF/um**2 */
```

```
  area=(int)(cap/unit);
  area=area-4*4*5;  /* subtract the area used for connection */
  printf("The capacitance area is %d UM**2\n",area);
  printf("Please enter the width and the length of a rectangular capacitor:\n");
  scanf(" %d,%d",&width,&length);
  printf("width=%d length=%d\n",width,length);
/* Landmark assignment for cap */
    x0 = 0;             /* x0=lower left corner of nwell area */
    x1 = x0+mc;          /* x1=lower left corner of pdiffusion area */
    x2 = x1+mc+1;        /* x2=lower left corner of poly area */
    x3 = x2+length;     /* x3=upper right corner of poly area */
    x4 = x2+length/2;   /* x4=poly plate connection */
/*********************************************************************/
    y0 = 0;             /* y0=lower left corner of nwell area */
    y1 = y0+mc;          /* y1=lower left corner of pdiffusion area */
    y2 = y1+mc+1;        /* y2=lower left corner of poly area */
    y3 = y2+width;      /* y3=upper right corner of poly area */
    y4 = y2+width/2;    /* y4=poly plate connection */
/*********************************************************************/
  cap_cell.box.llx=x0-mc;
  cap_cell.box.lly=y0-mc;
  cap_cell.box.urx=x3+1+3*mc;
  cap_cell.box.ury=y3+1+3*mc;
  cap_cell.pinnum=2;
  cap_cell.txnum=4;

/*********************************************************************/
timestamp = time((time_t *) 0);
cap_cell.timestamp=timestamp;
fprintf(fp1,"magic\n");
fprintf(fp1,"tech scmos\n");
fprintf(fp1,"timestamp %d\n",timestamp);
/*********************************************************************/
fprintf(fp1,"<< nwell >>\n");
fprintf(fp1,"rect %d %d %d %d\n",x1,y1,x3+1+mc,y3+1+mc);
/*********************************************************************/
fprintf(fp1,"<< nsubstratendiff >>\n");
fprintf(fp1,"rect %d %d %d %d\n",x0,y0,x4-halfmc-mc,y1);
fprintf(fp1,"rect %d %d %d %d\n",x4+halfmc+mc,y0,x3+1+2*mc,y1);
fprintf(fp1,"rect %d %d %d %d\n",x0,y0,x1,y4-halfmc-mc);
fprintf(fp1,"rect %d %d %d %d\n",x0,y4+halfmc+mc,x1,y3+1+2*mc);
```

```
fprintf(fp1,"rect %d %d %d %dh",x0,y3+1+mc,x4-halfmc-mc,y3+1+2*mc);
fprintf(fp1,"rect %d %d %d %dh",x4+halfmc+mc,y3+1+mc,x3+1+2*mc,y3+1+2*mc);
fprintf(fp1,"rect %d %d %d %dh",x3+1+mc,y0,x3+1+2*mc,y4-halfmc-mc);
fprintf(fp1,"rect %d %d %d %dh",x3+1+mc,y4+halfmc+mc,x3+1+2*mc,y3+1+2*mc);
/*****************************************************************/
fprintf(fp1,"<< pdiffusion >>\n");
fprintf(fp1,"rect %d %d %d %dh",x1,y1,x3+1+mc,y3+1+mc);

/*****************************************************************/
fprintf(fp1,"<< pdcontact >>\n");
fprintf(fp1,"rect %d %d %d %dh",x1,y4-halfmc-2*mc,x1+mc,y4-halfmc-mc);
fprintf(fp1,"rect %d %d %d %dh",x1,y4+halfmc+mc,x1+mc,y4+halfmc+2*mc);
fprintf(fp1,"rect %d %d %d %dh",x3+1,y4-halfmc-2*mc,x3+1+mc,y4-halfmc-mc);
fprintf(fp1,"rect %d %d %d %dh",x3+1,y4+halfmc+mc,x3+1+mc,y4+halfmc+2*mc);
fprintf(fp1,"rect %d %d %d %dh",x4-halfmc-2*mc,y1,x4-halfmc-mc,y1+mc);
fprintf(fp1,"rect %d %d %d %dh",x4+halfmc+mc,y1,x4+halfmc+2*mc,y1+mc);
fprintf(fp1,"rect %d %d %d %dh",x4-halfmc-2*mc,y3+1,x4-halfmc-mc,y3+1+mc);
fprintf(fp1,"rect %d %d %d %dh",x4+halfmc+mc,y3+1,x4+halfmc+2*mc,y3+1+mc);

/*****************************************************************/
fprintf(fp1,"<< polysilicon >>\n");
fprintf(fp1,"rect %d %d %d %dh",x2,y2,x3,y3);
fprintf(fp1,"rect %d %d %d %dh",x0,y4-halfmc,x2,y4+halfmc);
fprintf(fp1,"rect %d %d %d %dh",x4-halfmc,y0,x4+halfmc,y4);
fprintf(fp1,"rect %d %d %d %dh",x3,y4-halfmc,x3+1+2*mc,y4+halfmc);
fprintf(fp1,"rect %d %d %d %dh",x4-halfmc,y3,x4+halfmc,y3+1+2*mc);
/*****************************************************************/
fprintf(fp1,"<< nsubstratencontact >>\n");
fprintf(fp1,"rect %d %d %d %dh",x0,y4-halfmc-2*mc,x1,y4-halfmc-mc);
fprintf(fp1,"rect %d %d %d %dh",x0,y4+halfmc+mc,x1,y4+halfmc+2*mc);
fprintf(fp1,"rect %d %d %d %dh",x3+1+mc,y4-halfmc-2*mc,x3+1+2*mc,y4-halfmc-mc);
fprintf(fp1,"rect %d %d %d %dh",x3+1+mc,y4+halfmc+mc,x3+1+2*mc,y4+halfmc+2*mc);
fprintf(fp1,"rect %d %d %d %dh",x4-halfmc-2*mc,y0,x4-halfmc-mc,y1);
fprintf(fp1,"rect %d %d %d %dh",x4+halfmc+mc,y0,x4+halfmc+2*mc,y1);
fprintf(fp1,"rect %d %d %d %dh",x4-halfmc-2*mc,y3+1+mc,x4-halfmc-mc,y3+1+2*mc);
fprintf(fp1,"rect %d %d %d %dh",x4+halfmc+mc,y3+1+mc,x4+halfmc+2*mc,y3+1+2*mc);
/*****************************************************************/
fprintf(fp1,"<< polycontact >>\n");
fprintf(fp1,"rect %d %d %d %dh",x0-mc,y4-halfmc,x0,y4+halfmc);
fprintf(fp1,"rect %d %d %d %dh",x4-halfmc,y0-mc,x4+halfmc,y0);
fprintf(fp1,"rect %d %d %d %dh",x3+1+2*mc,y4-halfmc,x3+1+3*mc,y4+halfmc);
```

```
fprintf(fp1,"rect %d %d %d %dh",x4-halfmc,y3+1+2*mc,x4+halfmc,y3+1+3*mc);
/*****************************************************************/
fprintf(fp1,"<< metal1 >>h");
fprintf(fp1,"rect %d %d %d %dh",x1,y1,x3+1+mc,y2-1);
fprintf(fp1,"rect %d %d %d %dh",x1,y1,x2-1,y3+1+mc);
fprintf(fp1,"rect %d %d %d %dh",x1,y3+1,x3+1+mc,y3+1+mc);
fprintf(fp1,"rect %d %d %d %dh",x3+1,y1,x3+1+mc,y3+1+mc);
fprintf(fp1,"rect %d %d %d %dh",x4-halfmc-2*mc,y0-mc,x4-halfmc-mc,y0);
fprintf(fp1,"rect %d %d %d %dh",x4+halfmc+mc,y0-mc,x4+halfmc+2*mc,y0);
fprintf(fp1,"rect %d %d %d %dh",x0-mc,y4-halfmc-2*mc,x0,y4-halfmc-mc);
fprintf(fp1,"rect %d %d %d %dh",x0-mc,y4+halfmc+mc,x0,y4+halfmc+2*mc);
fprintf(fp1,"rect %d %d %d %dh",x4-halfmc-2*mc,y3+1+2*mc,x4-halfmc-mc,y3+1+3*mc);
fprintf(fp1,"rect %d %d %d %dh",x4+halfmc+mc,y3+1+2*mc,x4+halfmc+2*mc,y3+1+3*mc);
fprintf(fp1,"rect %d %d %d %dh",x3+1+2*mc,y4-halfmc-2*mc,x3+1+3*mc,y4-halfmc-mc);
fprintf(fp1,"rect %d %d %d %dh",x3+1+2*mc,y4+halfmc+mc,x3+1+3*mc,y4+halfmc+2*mc);
/*****************************************************************/
fprintf(fp1,"<< end >>h");
/*****************************************************************/
  cap_cell.drn[1].x=x4+halfmc+mc+halfmc; /* south side */
  cap_cell.drn[1].y=y0-mc;
  cap_cell.src[1].x=x4-halfmc-mc-halfmc;
  cap_cell.src[1].y=y0-mc;
  cap_cell.gat[1].x=x4;
  cap_cell.gat[1].y=y0-mc;
  cap_cell.src[3].x=x4+halfmc+mc+halfmc; /* north side */
  cap_cell.src[3].y=y3+1+3*mc;
  cap_cell.drn[3].x=x4-halfmc-mc-halfmc;
  cap_cell.drn[3].y=y3+1+3*mc;
  cap_cell.gat[3].x=x4;
  cap_cell.gat[3].y=y3+1+3*mc;
  cap_cell.src[2].x=x0-mc;              /* west side */
  cap_cell.src[2].y=y4+halfmc+mc+halfmc;
  cap_cell.drn[2].x=x0-mc;
  cap_cell.drn[2].y=y4-halfmc-mc-halfmc;
  cap_cell.gat[2].x=x0-mc;
  cap_cell.gat[2].y=y4;
  cap_cell.drn[0].x=x3+1+3*mc;          /* east side */
  cap_cell.drn[0].y=y4+halfmc+mc+halfmc;
  cap_cell.src[0].x=x3+1+3*mc;
  cap_cell.src[0].y=y4-halfmc-mc-halfmc;
  cap_cell.gat[0].x=x3+1+3*mc;
```

```
  cap_cell.gat[0].y=y4;
/*************************************************************************/
  fclose(fp1);
  return(cap_cell);
}
/*************************************************************************/

/*************************************************************************/
/* gen_cm.c is used to generate current mirror cell                      */
/* return the box of the cell                                            */
/* a magic file will be generated.                                       */
/*************************************************************************/


struct CELL gen_cm(fname)
char  *fname;

{
  FILE            *fp1,*fopen();
  char            type;
  int             diode,bulk,width,length,num;
  int             i,j,tx_num,xcor,ycor,shift[10];
  int             timestamp;
  extern time_t   time();
  char            tx_name[10][10],cm_name[10],m_name[10][10];
  struct CELL     cm;
  struct TRAN     gen_tx(),trans[10];
/*************************************************************************/
  if((fp1=fopen(fname,"w")) == NULL){
    printf("0an't open file name = %sn",fname);
    exit(1);
  }
  for(j=0;j<10;j++){
    if(fname[j] != '.') cm_name[j]=fname[j];
    else {
      cm_name[j]=' ';
      break;
    }
  }
  cm.name=cm_name;
  cm.pinnum=0;
```

```
  printf("How many transistors are included in this cell (%s):h",cm.name);
  scanf("%d",&tx_num);
  printf("Please enter the transistor type [p/n]:h");
  scanf(" %c",&type);
  for(i=0;i<tx_num;++i){
    printf("Please enter a file name for #%d transistor:",i);
    scanf("%s",tx_name[i]);
    printf("Enter its [diode(1/0)],[bulk(1/0)],[width],[length],[gate number]:h");
    scanf("%d,%d,%d,%d,%d",&diode,&bulk,&width,&length,&num);
    trans[i]=gen_tx(type,diode,bulk,width,length,num,tx_name[i]);
    cm.pinnum=cm.pinnum+trans[i].pinnum;
    cm.drn[i]=trans[i].drn;
    cm.src[i]=trans[i].src;
    cm.gat[i]=trans[0].gat; /* common gate */
  }

  cm.pinnum=cm.pinnum-(tx_num-1);/* common gate */
  cm.txnum=tx_num;
/*********************************************************************/
  cm.box.llx=trans[0].box.llx;
  cm.box.lly=trans[0].box.lly;
  cm.box.urx=0;
for(i=0;i<tx_num;++i)  cm.box.urx=cm.box.urx+(trans[i].box.urx-trans[i].box.llx);
  cm.box.urx=cm.box.urx+(tx_num-1)*dc2dc+trans[0].box.llx;
  cm.box.ury=trans[0].box.ury;
for(i=0;i<tx_num;++i)  cm.box.ury=MAX(cm.box.ury,trans[i].box.ury);
shift[0]=0;
for(i=1;i<tx_num;++i){
  shift[i]=shift[i-1]+(trans[i-1].box.urx-trans[i-1].box.llx)+dc2dc;
  cm.drn[i].x=cm.drn[i].x+shift[i];
  cm.src[i].x=cm.src[i].x+shift[i];
}
for(i=0;i<tx_num;++i){
  cm.src[i].y=cm.box.ury;
}
/*********************************************************************/

timestamp = time((time_t *) 0);
cm.timestamp=timestamp;
fprintf(fp1,"magich");
fprintf(fp1,"tech scmosh");
```

```
fprintf(fp1,"timestamp %d\n",timestamp);

/*********************************************************************/
fprintf(fp1,"<< polysilicon >>\n");
fprintf(fp1,"rect %d %d %d %d\n",cm.box.llx,cm.box.lly,cm.box.urx,cm.box.lly+mc);
/*********************************************************************/
fprintf(fp1,"<< metal1 >>\n");
for(i=0;i<tx_num;i++){
fprintf(fp1,"rect %d %d %d %d\n",cm.src[i].x-halfmc,trans[i].src.y-mc,cm.src[i]
      .x+halfmc,cm.box.ury);
}
/*********************************************************************/
  for(j=0;j<10;j++){
    if(tx_name[0][j] != '.') m_name[0][j]=tx_name[0][j];
    else {
      m_name[0][j]=' ';
      break;
    }
  }
  fprintf(fp1,"use %s %s_0\n",m_name[0],m_name[0]);
  fprintf(fp1,"timestamp %d\n",trans[0].timestamp);
  fprintf(fp1,"transform 1 0 0 0 1 0\n");
  fprintf(fp1,"box %d %d %d %d\n",trans[0].box.llx,trans[0].box.lly,trans[0]
      .box.urx,trans[0].box.ury);
/*********************************************************************/
  xcor=0;
  ycor=0;
for(i=1;i<tx_num;++i){
  for(j=0;j<10;j++){
    if(tx_name[i][j] != '.') m_name[i][j]=tx_name[i][j];
    else {
      m_name[i][j]=' ';
      break;
    }
  }
  fprintf(fp1,"use %s %s_0\n",m_name[i],m_name[i]);
  xcor=xcor+trans[i-1].box.urx+dc2dc;
  fprintf(fp1,"timestamp %d\n",trans[i].timestamp);
  fprintf(fp1,"transform 1 0 %d 0 1 0\n",xcor-trans[0].box.llx);
  fprintf(fp1,"box %d %d %d %d\n",trans[i].box.llx,trans[i].box.lly,trans[i]
      .box.urx,trans[i].box.ury);
```

```
}
/*********************************************************************/
fprintf(fp1,"<< end >>\n");
/*********************************************************************/
  fclose(fp1);
  return(cm);
}
/*****************************************************************************/

/*********************************************************************/
/* gen_difin.c is used to generate a common source input pair  */
/* return a struct of the CELL                                 */
/* a magic file will be generated.                             */
/*********************************************************************/



struct CELL gen_difin(fname)
char *fname;
{
  FILE              *fp1,*fopen();
  int               x0,x1,x2,x3,x4,x5,x6;
  int               y0,y1,y2,y3,y4,y5,y6;
  int               j,transtep,xvar,yvar;
  int               width,length,num;
  char              type,difin_name[10],anti_type;
  struct CELL       difin;
  int               timestamp;
  extern time_t     time();
/*********************************************************************/
  if((fp1=fopen(fname,"w")) == NULL){
    printf("0an't open file name = %s\n",fname);
    exit(1);
  }
  for(j=0;j<10;j++){
    if(fname[j] != '.') difin_name[j]=fname[j];
    else {
      difin_name[j]=' ';
      break;
    }
  }
```

```
  difin.name=difin_name;
/*****************************************************************/
  printf("Please enter the [type],[width],[length],and [gate num] of
      input tx:n");
  scanf(" %c,%d,%d,%d",&type,&width,&length,&num);
  if(type=='p') anti_type='n';
  if(type=='n') anti_type='p';
  width=width/num;
/*****************************************************************/
/* Landmark assignment for fet */
  transtep =mc+ 2*dc2py+length;
  x0 = 0;                  /* x0=the left bound of diffusion area */
  x1 = x0+halfmc;           /* x1=the beginning of the drainA */
  x2 = x1+transtep;         /* x2=the beginning of the common source */
  switch (num % 2){
  case 0 : /* even number of tx's requested */
    x3 = x1+4*transtep*(num/2); /* x3=the end of drainA */
    x4 = x3-2*transtep;          /* x4=the end of drainB */
    x5 = x2+2*transtep*(num-1); /* x5=the end of common source */
    break;
  case 1 : /* odd number of tx's requested */
    x3 = x1+4*transtep*((num+1)/2-1); /* x3=the end of drainA */
    x4 = x3+2*transtep;                /* x4=the end of drainB */
    x5 = x2+2*transtep*(num-1);         /* x5=the end of common source */
    break;
  }
  x6 = MAX(x3,x4);
/*****************************************************************/
  y0 = 0;                 /* y0=the lower bound of diffusion area */
  y1 = y0+width;           /* y1=the upper bound of diffusion area */
  y2 = y1+wc2dc+halfmc; /* y2=center of common source rail
                           wc2dc=well contact to diff contact
                           mc=min metal contact */
  y3 = y2+mc2mc+mc;       /* y3=center of polyA rail
                  mc2mc=metal contact to metal contact */
  y4 = y3+mc2mc+mc;       /* y4=center of polyB rail */
  y5 = y0-mc2mc-1-halfmc; /* y5=center of drainA rail */
  y6 = y5-mc2mc-mc;       /* y6=center of drainB rail */
/*****************************************************************/
  difin.pinnum=5;
  difin.txnum=2;
```

```
  difin.box.llx=x0;
  difin.box.lly=y6-halfmc;
  difin.box.urx=x6+halfmc;
  difin.box.ury=y4+halfmc;
if(x6==x3){
  difin.drn[0].x=x1;
  difin.drn[0].y=y6-halfmc;
  difin.drn[1].x=x6-2*transtep;
  difin.drn[1].y=y6-halfmc;
  difin.src[1].x=x6+halfmc;
  difin.src[1].y=y2;
  difin.gat[0].x=x1+halfmc+dc2py+halfmc;
  difin.gat[0].y=y4+halfmc;
  difin.gat[1].x=x6-2*transtep+halfmc+dc2py+halfmc;
  difin.gat[1].y=y4+halfmc;
  difin.src[0].x=x1-halfmc;
  difin.src[0].y=y2;
}
if(x6==x4){
  difin.drn[0].x=x1;
  difin.drn[0].y=y6-halfmc;
  difin.drn[1].x=x6;
  difin.drn[1].y=y6-halfmc;
  difin.src[0].x=x6+halfmc;
  difin.src[0].y=y2;
  difin.gat[0].x=x1+halfmc+dc2py+halfmc;
  difin.gat[0].y=y4+halfmc;
  difin.gat[1].x=x6-transtep+halfmc+dc2py+halfmc;
  difin.gat[1].y=y4+halfmc;
  difin.src[1].x=x1-halfmc;
  difin.src[1].y=y2;
}
/*********************************************************************/
timestamp = time((time_t *) 0);
difin.timestamp=timestamp;
fprintf(fp1,"magic\n");
fprintf(fp1,"tech scmos\n");
fprintf(fp1,"timestamp %d\n",timestamp);

/*********************************************************************/
fprintf(fp1,"<< %cdiffusion >>\n",type);
```

```
fprintf(fp1,"rect %d %d %d %dn",x0,y0,x6+halfmc,y1);
/*********************************************************************/
fprintf(fp1,"<< %csubstrate%ccontact >>n",anti_type,anti_type);
for(xvar = x1;xvar <= x6;xvar = xvar+transtep){
fprintf(fp1,"rect %d %d %d %dn",xvar-halfmc,y2-halfmc,xvar+halfmc,y2+halfmc);
}
/*********************************************************************/
fprintf(fp1,"<< polysilicon >>n");
for(xvar = x1+halfmc+dc2py;xvar <= x6-halfmc-dc2py-length;xvar = xvar+transtep){
fprintf(fp1,"rect %d %d %d %dn",xvar,y0-pyw,xvar+length,y2+halfmc);
}
if(x6==x3){  /* x6=drainA */
  for(xvar = x1+halfmc+dc2py;xvar < x6;xvar = xvar+4*transtep){
    fprintf(fp1,"rect %d %d %d %dn",xvar,y2+halfmc,xvar+length,y3-halfmc);
    fprintf(fp1,"rect %d %d %d %dn",xvar+3*transtep,y2+halfmc,xvar+3*transtep
      +length,y3-halfmc);
  }
  for(xvar = x2+halfmc+dc2py;xvar < x6;xvar = xvar+4*transtep){
    fprintf(fp1,"rect %d %d %d %dn",xvar,y2+halfmc,xvar+length,y4-halfmc);
    fprintf(fp1,"rect %d %d %d %dn",xvar+transtep,y2+halfmc,xvar+transtep
      +length,y4-halfmc);
  }
}
if(x6==x4){  /* x6=drainB */
  for(xvar = x2+halfmc+dc2py;xvar < x6;xvar = xvar+4*transtep)
    fprintf(fp1,"rect %d %d %d %dn",xvar,y2+halfmc,xvar+length,y4-halfmc);
  for(xvar = x2+transtep+halfmc+dc2py;xvar < x6;xvar = xvar+4*transtep)
    fprintf(fp1,"rect %d %d %d %dn",xvar,y2+halfmc,xvar+length,y4-halfmc);
  for(xvar = x1+halfmc+dc2py;xvar < x6;xvar = xvar+4*transtep)
    fprintf(fp1,"rect %d %d %d %dn",xvar,y2+halfmc,xvar+length,y3-halfmc);
  for(xvar = x1+3*transtep+halfmc+dc2py;xvar < x6;xvar = xvar+4*transtep)
    fprintf(fp1,"rect %d %d %d %dn",xvar,y2+halfmc,xvar+length,y3-halfmc);
}

/*********************************************************************/
fprintf(fp1,"<< polycontact >>n");
if(x6==x3){  /* x6=drainA */
  for(xvar = x1+halfmc+dc2py;xvar < x6;xvar = xvar+4*transtep){
    fprintf(fp1,"rect %d %d %d %dn",xvar,y3-halfmc,xvar+MAX(length,mc),y3+halfmc);
    fprintf(fp1,"rect %d %d %d %dn",xvar+3*transtep,y3-halfmc,xvar+3*transtep
      +MAX(length,mc),y3+halfmc);
```

```
  }
  for(xvar = x2+halfmc+dc2py;xvar < x6;xvar = xvar+4*transtep){
    fprintf(fp1,"rect %d %d %d %d\n",xvar,y4-halfmc,xvar+MAX(length,mc),y4+halfmc);
    fprintf(fp1,"rect %d %d %d %d\n",xvar+transtep,y4-halfmc,xvar+transtep
      +MAX(length,mc),y4+halfmc);
  }
}
if(x6==x4){  /* x6=drainB */
  for(xvar = x2+halfmc+dc2py;xvar < x6;xvar = xvar+4*transtep){
    fprintf(fp1,"rect %d %d %d %d\n",xvar,y4-halfmc,xvar+MAX(length,mc),y4+halfmc);
     if((xvar+transtep)<x6){
       fprintf(fp1,"rect %d %d %d %d\n",xvar+transtep,y4-halfmc,xvar+transtep
      +MAX(length,mc),y4+halfmc);
     }
  }
  for(xvar = x1+halfmc+dc2py;xvar < x6;xvar = xvar+4*transtep){
    fprintf(fp1,"rect %d %d %d %d\n",xvar,y3-halfmc,xvar+MAX(length,mc),y3+halfmc);
     if((xvar+3*transtep)<x6){
       fprintf(fp1,"rect %d %d %d %d\n",xvar+3*transtep,y3-halfmc,xvar+3*transtep
      +MAX(length,mc),y3+halfmc);
     }
  }
}

/*****************************************************************/
fprintf(fp1,"<< polycontact >>\n"); /* can be replaced by m2contact */
if(x6==x3){  /* x6=drainA */
  for(xvar = x1;xvar <= x6;xvar = xvar+4*transtep){
    fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y5-halfmc,xvar+halfmc,y5+halfmc);
    if(xvar+2*transtep < x6)
    fprintf(fp1,"rect %d %d %d %d\n",xvar+2*transtep-halfmc,y6-halfmc,xvar
      +2*transtep+halfmc,y6+halfmc);
  }
}
if(x6==x4){  /* x6=drainB */
  for(xvar = x2+transtep;xvar <= x6;xvar = xvar+4*transtep)
    fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y6-halfmc,xvar+halfmc,y6+halfmc);
  for(xvar = x1;xvar < x6;xvar = xvar+4*transtep)
    fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y5-halfmc,xvar+halfmc,y5+halfmc);
}
```

```
/*********************************************************************/
fprintf(fp1,"<< %cdcontact >>\n",type);
for(xvar = x1;xvar <= x6;xvar = xvar+transtep){
fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y0,xvar+halfmc,y1);
}

/*********************************************************************/
fprintf(fp1,"<< metal1 >>\n");
if(x6==x3){  /* x6=drainA */
  for(xvar = x1;xvar <= x6;xvar = xvar+4*transtep){
    fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y5+halfmc,xvar+halfmc,y0);
    if(xvar+2*transtep < x6)
    fprintf(fp1,"rect %d %d %d %d\n",xvar+2*transtep-halfmc,y6+halfmc,xvar
      +2*transtep+halfmc,y0);
  }
}
if(x6==x4){  /* x6=drainB */
  for(xvar = x2+transtep;xvar <= x6;xvar = xvar+4*transtep)
    fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y6+halfmc,xvar+halfmc,y0);
  for(xvar = x1;xvar < x6;xvar = xvar+4*transtep)
    fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y5+halfmc,xvar+halfmc,y0);
}
for(xvar = x2;xvar <= x6;xvar = xvar+2*transtep){
fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y1,xvar+halfmc,y2-halfmc);
}
fprintf(fp1,"rect %d %d %d %d\n",x1,y2-halfmc,x6,y2+halfmc);
if(x6==x3){  /* x6=drainA */
fprintf(fp1,"rect %d %d %d %d\n",x1+halfmc+dc2py,y3-halfmc,x6-halfmc-dc2py,
      y3+halfmc);
fprintf(fp1,"rect %d %d %d %d\n",x2+halfmc+dc2py,y4-halfmc,x6-transtep
      -halfmc-dc2py,y4+halfmc);
}
if(x6==x4){  /* x6=drainB */
fprintf(fp1,"rect %d %d %d %d\n",x1+halfmc+dc2py,y3-halfmc,x6-transtep
      -halfmc-dc2py,y3+halfmc);
fprintf(fp1,"rect %d %d %d %d\n",x2+halfmc+dc2py,y4-halfmc,x6-halfmc-dc2py,
      y4+halfmc);
}
fprintf(fp1,"rect %d %d %d %d\n",x1-halfmc,y6-halfmc,x1+halfmc,
      y5-halfmc);       /* for gate A port */
fprintf(fp1,"rect %d %d %d %d\n",x1+halfmc+dc2py,y3-halfmc,x1+halfmc+dc2py
```

```
      +mc,y4+halfmc);  /* for drain A port */
/*************************************************************************/
fprintf(fp1,"<< polysilicon >>\n");
if(x6==x3){  /* x6=drainA */
    fprintf(fp1,"rect %d %d %d %d\n",x1-halfmc,y5-halfmc,x6+halfmc,y5+halfmc);
    fprintf(fp1,"rect %d %d %d %d\n",x2+transtep-halfmc,y6-halfmc,
      x6-2*transtep+halfmc,y6+halfmc);
  }
if(x6==x4){  /* x6=drainB */
    fprintf(fp1,"rect %d %d %d %d\n",x2+transtep-halfmc,y6-halfmc,
      x6+halfmc,y6+halfmc);
    fprintf(fp1,"rect %d %d %d %d\n",x1-halfmc,y5-halfmc,x6-2*transtep
      +halfmc,y5+halfmc);
  }
/*************************************************************************/
fprintf(fp1,"<< end >>\n");
/*************************************************************************/
  fclose(fp1);
  return(difin);
}
/*************************************************************************/

/*****************************************************************/
/* gen_tx.c is used to generate individual transistor          */
/* result : return a struct of TRAN                             */
/*          a magic file will be generated.                     */
/*****************************************************************/


struct TRAN gen_tx(type,diode,bulk,width,length,num,fname)
char  fname[10];
int   diode,bulk,width,length,num;
char  type;
{
  FILE   *fp1,*fopen();
  int      x0,x1,x2,x3,x4,x5;
  int      y0,y1,y2,y3,y4,y5;
  int      transtep,xvar,yvar;
  int      j,timestamp;
  extern  time_t time();
  struct TRAN tran;
```

```
  char    anti_type;
  char    m_name[10];

  if(type=='p') anti_type='n';
  if(type=='n') anti_type='p';
  width=width/num;
/*****************************************************************/
  if((fp1=fopen(fname,"w")) == NULL){
    printf("0an't open file name = %sn",fname);
    exit(1);
  }
  for(j=0;j<10;j++){
    if(fname[j] != '.') m_name[j]=fname[j];
    else {
      m_name[j]=' ';
      break;
    }
  }
  tran.name=m_name;
  tran.diode=diode;
  if(diode==1) tran.pinnum=2;
  if(diode==0) tran.pinnum=3;
/* Landmark assignment for fet */
  transtep =mc+ 2*dc2py+length;
  x0 = 0;              /* x0=left bound of diffusion area */
  x1 = x0+halfmc;     /* x1=the beginning of source */
  x2 = x1+transtep;  /* x2=the beginning of drain */
  switch (num % 2){
  case 0 : /* even number of gate of tx's requested */
    x3 = x2+transtep*(num-2); /* x3=the end of drain */
    x4 = x3+transtep;          /* x4=the end of source */
    break;
  case 1 : /* odd number of tx's requested */
    x3 = x2+transtep*(num-1); /* x3=the end of drain */
    x4 = x3-transtep;          /* x4=the end of source */
    break;
  }
  x5 = MAX(x3,x4);   /* x5= the max of x3 and x5 */
/*****************************************************************/
  y0 = 0;          /* y0=lower bound of diffusion area */
  y1 = y0+width; /* y1=upper bound of diffusion area */
```

```
  y2 = y1+wc2dc+halfmc; /* y2=center of common source rail
                           wc2dc=well contact to diff contact
                           mc=min metal contact */
  y3 = y0-mc2mc-halfmc; /* y3=center of poly rail or common drain rail
                   mc2mc=metal contact to metal contact */

/*********************************************************************/
timestamp = time((time_t *) 0);
tran.timestamp=timestamp;
fprintf(fp1,"magic\n");
fprintf(fp1,"tech scmos\n");
fprintf(fp1,"timestamp %d\n",timestamp);

/*********************************************************************/
fprintf(fp1,"<< %cdiffusion >>\n",type);
fprintf(fp1,"rect %d %d %d %d\n",x0,y0,x5+halfmc,y1);
/*********************************************************************/
if (bulk==1){
fprintf(fp1,"<< %csubstrate%ccontact >>\n",anti_type,anti_type);
for(xvar = x1;xvar <= x5;xvar = xvar+transtep)
fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y2-halfmc,xvar+halfmc,y2+halfmc);
}
/*********************************************************************/
fprintf(fp1,"<< polysilicon >>\n");
for(xvar = x1+halfmc+dc2py;xvar <= x5-halfmc-dc2py-length;xvar = xvar+transtep)
  fprintf(fp1,"rect %d %d %d %d\n",xvar,y3+halfmc,xvar+length,y1+pyw);
fprintf(fp1,"rect %d %d %d %d\n",x0,y3-halfmc,x5+halfmc,y3+halfmc);

/*********************************************************************/
if(diode==1){
fprintf(fp1,"<< polycontact >>\n");
  if(x5==x3){  /* x5=drain */
    for(xvar = x2;xvar <= x5;xvar = xvar+2*transtep)
    fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y3-halfmc,xvar+halfmc,y3+halfmc);
  }
  if(x5==x4){  /* x5=source */
    for(xvar = x2;xvar < x5;xvar = xvar+2*transtep)
    fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y3-halfmc,xvar+halfmc,y3+halfmc);
  }
}
if(diode==0){
```

```
fprintf(fp1,"<< polycontact >>\n");
fprintf(fp1,"rect %d %d %d %d\n",x1-halfmc,y3-halfmc,x1+halfmc,y3+halfmc);
}

/*************************************************************************/
fprintf(fp1,"<< %cdcontact >>\n",type);
for(xvar = x1;xvar <= x5;xvar = xvar+transtep){
fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y0,xvar+halfmc,y1);
}

/*************************************************************************/
fprintf(fp1,"<< metal1 >>\n");
if(x5==x3){  /* x5=drain */
 for(xvar = x1;xvar < x5;xvar = xvar+2*transtep)
  fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y1,xvar+halfmc,y2+halfmc);
 for(xvar = x2;xvar <= x5;xvar = xvar+2*transtep)
  fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y3-halfmc,xvar+halfmc,y0);
}
if(x5==x4){  /* x5=source */
 for(xvar = x1;xvar <= x5;xvar = xvar+2*transtep)
  fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y1,xvar+halfmc,y2+halfmc);
 for(xvar = x2;xvar < x5;xvar = xvar+2*transtep)
  fprintf(fp1,"rect %d %d %d %d\n",xvar-halfmc,y3-halfmc,xvar+halfmc,y0);
}

fprintf(fp1,"rect %d %d %d %d\n",x1-halfmc,y2-halfmc,x5+halfmc,y2+halfmc);

if(diode==0){
  if(x5==x3)  /* x5=drain */
    fprintf(fp1,"rect %d %d %d %d\n",x2-halfmc,y3-halfmc,x5+halfmc,y3+halfmc);
  if(x5==x4)  /* x5=source */
    fprintf(fp1,"rect %d %d %d %d\n",x2-halfmc,y3-halfmc,x5-transtep+halfmc,
      y3+halfmc);
}

/*************************************************************************/
fprintf(fp1,"<< end >>\n");
/*************************************************************************/
  tran.box.llx=0;
  tran.box.lly=y3-halfmc;
  tran.box.urx=x5+halfmc;
```

```
  tran.box.ury=y2+halfmc;
if(diode==0){
  tran.drn.x=x3;
  tran.drn.y=y3-halfmc;
  tran.src.x=x1;
  tran.src.y=y2+halfmc;
  tran.gat.x=x1;
  tran.gat.y=y3-halfmc;
}
if(diode==1){
  tran.drn.x=x2;
  tran.drn.y=y3-halfmc;
  tran.src.x=x1;
  tran.src.y=y2+halfmc;
  tran.gat.x=x2;
  tran.gat.y=y3-halfmc;
}
/*****************************************************************/
  fclose(fp1);
  return(tran);
}
/*****************************************************************************/
```

Appendix D

Symbols

A_v	Voltage gain
A_{vo}	Op-Amp open-loop voltage gain
B	Bias node
C	Capacitance
C_c	Compensation capacitor
C_L	Load capacitor
C_p	Capacitor plate
CCM	Cascode current mirror primitive
CF	Correction factor
CG	Common-gate devices primitive
CM	Simple current mirror primitive
CM-OPIL	Op-Amp input-stage load primitive
C_{MRR}	Common mode rejection ratio
D	Transistor drain terminal
DC	Diode-connected node

E_r	Error factor
f_o	Op-Amp unity-gain bandwidth
$f(\cdot)$	Transfer function
g_m	Transistor transconductance
g_o	Transistor output conductance
g_{on}	N-channel transistor output conductance
g_{op}	P-channel transistor output conductance
G	Transistor gate terminal
G_m	Amplifier circuit transconductance
h	Cell height
H	Horizontal cut
HI	High-impedance node
I	Input node
I_{bias}	Bias current
I_{ss}	Bias current of differential amplifier
IS	Insensitive net
k_p'	Transistor transconductance coefficient
L	Transistor channel length

LS	Less-sensitive net
M	Transistor
N	Neutral net
N_p	Parameterizable neuron
NY	Noisy net
O	Output node
P	Power source node
P_d	Power dissipation
PSM	Parameterizable synapse matrix
P_{SRR}	Power supply rejection ratio
r_o	Transistor output resistance
R	Recognition rule
R	Resistance
R	Shape constraint relation
R_{eq}	Neuron input resistance
R_i	Synapse connection resistance
R_o	Amplifier output resistance
S	Transistor source terminal

SC	Source-coupled node
SC-IN	Input differential pair primitive
S_r	Slew rate
t	Transistor type
t_{set}	Settling time
T	Single transistor primitive
T_i	Synapse weight
TG	Transmission gate
T-OPOD	Op-Amp output driver primitive
U	Transistor substrate terminal
V_{DD}	Positive supply voltage
V	Vertical cut
V_i	Neuron input voltage
V_{in}	Input voltage
V_{in-}	Minus input voltage
V_{in+}	Positive input voltage
V_{ip}	Input voltage range
V_{noise}	Input-referred noise

V_o	Neuron output voltage
V_{op}	Output voltage swing
V_{os}	Input offset voltage
V_{out}	Output voltage
V_R	Reference voltage
V_S	Analog input voltage
V_θ	Neuron threshold voltage
V_{SS}	Negative supply voltage
w	Cell width
W	Transistor channel width
ϕ_M	Phase margin
θ	Threshold constant

Appendix E

Glossary

ADC: Analog-to-digital converter. See A/D Converter.

A/D Converter: Analog-to-digital converter. A circuit which converts an analog (continuous) voltage or current into an output digital word.

AFE: Analog front end, the analog interface part of a digital signal processing system. It typically contains a digital-to-analog converter, an analog-to-digital converter, and some pre- and post-filtering.

AGC: Automatic gain controller. An amplifier circuit with a digitally controlled gain for use in communications and data acquisition systems.

AI: See Artificial Intelligence.

Artificial Intelligence: The subfield of computer science concerned with developing intelligent computer programs. This includes programs that can solve problems, learn from experience, understand language, interpret visual scenes, and, in general, behave in a way that would be considered intelligent if observed in a human.

Axon: An output of a biological neuron.

ALU: Arithmetic logic unit. A circuit which provides arithmetic and logic operations on data in a digital processor.

Analog Multiplier: A key analog signal processing circuit which gives as its output, the product of two variables (voltage or current).

Artificial Neural Network: See Neural Network.

BICMOS: Bipolar CMOS, a new technology which supports both bipolar and CMOS devices.

Bottom-Up Layout Generation: A layout style that is used to generate mask

geometries of a circuit starting from the transistor (bottom) level of the circuit hierarchy. This expression is derived from the relative positions of design operations in the flowchart.

C: A low-level, efficient, general-purpose programming language associated with the UNIX operating system. C is normally used for system programming.

CAD: Computer-aided design, the use of computer technology to assist in the design process, e.g., the design of integrated circuits.

CAMP: An experimental expert system developed at USC to assist the design of operational amplifiers.

Cascode: A common analog circuit technique used to increase the small signal output resistance of an amplifier circuit.

Cascode Current Mirror: An improved current mirror with the use of cascode techniques. The output resistance of the cascode current mirror is significantly greater than the simple current mirror.

Cascode Current Source: An improved current source with the use of cascode techniques.

Channel Router: A router which can handle regular regions with fixed pins on two sides of the regions and floating pins on the other two sides.

Chopper-Stabilized Op-Amp: An operational amplifier which employs a special chopper (modulator-demodulator) circuit to reduce the low-frequency noise of input transistors.

Circuit Recognition: A rule-based method for identifying key logic functions or circuit elements using a computer program.

Class-A Op-Amp: An Op-Amp with the class-A amplifier configuration. The class-A amplifier is a circuit whose power consumption and output current availability are fixed independently of the value of the signal applied to it.

Class-AB Op-Amp: An Op-Amp with an input to output class-AB configuration. The class-AB amplifier is a circuit whose power dissipation and output current availability are a function of the applied signal with peak values that can be many times larger than the stand-by ones.

CMFB: Common-mode feedback circuit, typically used in fully-differential operational amplifiers to control the common-mode voltage of the differential outputs.

CMOS: Complementary metal oxide semiconductor, a MOS technology that has both n-channel and p-channel devices available. It becomes a dominant technology for both analog and digital VLSI applications.

CMRR: Common-mode rejection ratio, is the ratio of differential voltage gain to common-mode voltage gain of an amplifier, generally expressed in dB.

Comb Filter: A special class of digital FIR filter. A comb filter of length D is an FIR filter with all D coefficients equal to one.

Common-Mode Input Voltage Range: The voltage range over which the input common-mode signal can vary. Typically, this range is several volts less than the higher power supply voltage and several volts greater than the lower power supply voltage.

Constraint-Based Layout Generation: A layout generation approach in which the circuit layout constraints are internally generated and subsequently incorporated in the final layout generation using constraint-driven floorplanning and routing techniques.

Constraint-Driven Floorplanning and Routing: A heuristic floorplanning and routing procedure which is mainly driven by the internally generated layout constraints.

Critical Net: A critical circuit node which requires special layout care.

CTF: Continuous-time filter, an implementation of the classic active filter in the continuous-time frequency domain.

Current Mirror: Two or more transistors connected so that current in one node is duplicated in another node. Two MOS transistors, for example, having sources tied together and gates connected by one of the drains, would duplicate the current in that drain in the second drain.

Current Source: A circuit which provides a constant current source or sink to the external circuitry.

DAC: Digital-to-analog converter. See D/A Converter.

D/A Converter: Digital-to-analog converter. A circuit which converts a digital code word into an output analog (continuous) voltage or current.

Dangling Component: The transistor element whose source terminal is not directly tied to a power supply.

Dendrite: An input of a biological neuron.

Design Synthesis: Generation of sized device-level schematic diagrams from performance specifications and process specifications.

Differential Pair: Two transistors whose sources (or emitters) are tied together to a current source. The voltage difference between the two gates steers the current between the two drains.

Digital Decimation Filter: A digital filter which reduces the sampling rate of the signals from a high value Nf_c to a lower value f_c where N is an interger number greater than 1.

Diode-Connected Node: A circuit junction node that connects the gate terminal of a transistor to its drain terminal (a diode configuration).

Domain Knowledge: Knowledge about the problem domain.

DRAM: Dynamic random access memory. A semiconductor memory which uses charge storage on a capacitor to represent binary data values.

DSP: Digital signal processor. A programmable, computational processor that allows effective implementations of typical digital signal processing tasks.

Dynamic Range: The ratio of the maximum input amplitude of an Op-Amp without causing saturation or an excessive nonlinear distortion noise referred to the input. Dynamic range is often termed as maximum signal to noise ratio.

Expert: A person who, through years of training and experience, has become extremely proficient at problem solving in a particular domain.

Expert System: A computer program that relies on a body of knowledge to perform a somewhat difficult task usually performed only by a human expert. The

principal power of an expert system is derived from the "knowledge" the system contains rather than exclusively from search algorithms and specific reasoning methods. An expert system successfully deals with problems for which clear algorithmic solutions do not exist.

Facsimile Modem: One type of modem that allows facsimile machines to transfer graphics-oriented documents.

Fact: An elementary piece of knowledge. Facts are fixed data in the knowledge base.

FIR: Finite impulse response. A filter impulse response that has a finite duration. The type of digital filter with a finite impulse response, called FIR filter, is always stable.

Floorplanning: The task of determining the relative positions of the modules on the plane so that performance objectives such as area, timing and power consumption of the chip are optimized.

Folded Cascode Op-Amp: An Op-Amp which employs a folded cascode configuration to eliminate the internal high-impedance node. This Op-Amp is useful in achieving a wide common-mode input voltage range.

Four-Quadrant Multiplier: An analog multiplier which can operate the multiplication of two variables in four quadrants.

Full-Custom Design: A chip design approach in which all the mask geometries are designed *ad hoc* by human designers with little help from CAD tools.

Full-Duplex Operation: The operation that allows a modem to transmit and receive data simultaneously on a two-wire line at full speed. It requires the ability to separate a receive signal from the reflection of the transmitted signal.

Fully-Differential Op-Amp: An Op-Amp which features fully differential (balanced) signal paths from input to output. Compared to single-ended Op-Amps, the fully-differential Op-Amps provide better power-supply noise immunity and dynamic range performance.

Fuzzy Logic: An approach to approximate reasoning in which truth values and quantifiers are defined as possibility distributions that carry linguistic labels, such as *true, very ture, not very true, many, not very many, few,* and *several.* The

rules of inference are approximate, rather than exact, in order to better manipulate information that is incomplete, imprecise, or unreliable.

Gilbert Multiplier: A four-quadrant multiplier which is the basis for most integrated circuit analog multipliers.

Half-Duplex Operation: The modem operation that is capable of passing signals in either direction, but not in both simultaneously.

Heuristic: A rule of thumb or simplification that limits the search for solutions in domains that are difficult and poorly understood.

High-Impedance Node: A circuit junction node that connects drain terminals of multiple transistors.

Hopfield Neural Network: A neural network architecture proposed by Tank and Hopfield [72] for solving certain engineering optimization problems. This network is a single-layer neural network with feedback.

Hysteresis: The quality of a comparator in which the input threshold changes as a function of the input (or output) level. A comparator with hysteresis is needed in a noisy environment.

IC: See Integrated Circuit.

Inference Engine: The part of a knowledge-based system or expert system that contains the general problem-solving knowledge. The inference engine processes the domain knowledge (located in the knowledge base) to reach new conclusions.

Integrated Circuit: A combination of interconnected circuit elements inseparably associated on or within a continuous substrate.

Intelligent Sensor: A sensor that can be integrated with appropriate circuits.

Junction Node: A circuit node that connects two or more transistor terminals together.

Knowledge: The information a computer program must have to behave intelligently.

Knowledge Base: The portion of a knowledge-based system or expert system

that contains the domain knowledge.

Knowledge-Based System: See Expert System.

Layer: The group of neurons which are arranged into a disjointed structure.

Logic-Based Method: A programming method that uses predicate logic to structure the program and guide execution.

Layout Constraints: A set of constraints on area, aspect ratio, and performance that needs to be satisfied for the given layout optimization problem.

Layout Synthesis: Generation of mask geometries form sized circuit schematics.

Macro-Cell Design Style: A design method in which large circuit blocks, customized to a certain type of logic function, are available in a circuit library.

Mighty: A switch-box router developed at the University of California, Berkeley.

Min-Cut Algorithm: A floorplanning algorithm proposed by Breuer [63] for slicing structures. This method is based on the recursive application of a bi-partitioning procedure.

Mixed-Signal IC: A chip that contains mixed analog and digital circuits on the same substrate.

Modem: A contraction of MOdulator-DEModulator, a system that connects data terminal equipment to a communication line.

Module: A major building block for circuit subsystems. Typical module examples include Op-Amps and comparators.

Module Generation: A custom module design method based on a set of parameterizable cell generators.

MOS: Metal oxide semiconductor. A process technology that uses a metal oxide semiconductor field effect transistor, or MOSFET, as the basic active device.

MOSIS: MOS implementation system. An MOS fabrication program sponsored by the U.S. Department of Defense Advanced Research Projects Agency

(DARPA).

Multiple-Fixed-Architecture Approach: A design synthesis approach in which a number of fixed circuits are stored in the design knowledge base.

Multi-Layer Neural Network: One type of neural network which is a feed-forward network with hidden layers located in the middle stages.

Netlist: A file that contains the information that can be used to reconstruct a circuit schematic diagram.

Net Sensitivity: A measure of net criticality based on sensitivity classification for analog circuit nets.

Neural Network: A parallel processing system which can be implemented by hundreds of artificial neurons and thousands of synapses on a single chip. With the immense computational power and self-learning capability, neural networks is emerging as a new promising technology for solving complex problems.

Neuron: The basic cell in the nervous system. An artificial neuron is typically implemented as an amplifier in the VLSI neural networks. See Appendix B.

Neuron Block: A circuit block that contains a group of neurons.

Neuton-Raphson's Algorithm: An algorithm for solving nonlinear equations. According to this algorithm, a solution for a nonlinear equation is quadratically reached through tangential extrapolations of intermediate results.

NMOS: N-channel MOS. An MOS technology that uses only n-channel MOSFETs as the active devices.

Nyquist Rate: The minimum sampling rate, which is twice the highest frequency of the input signal, required in a sampled-data system.

Op-Amp: Operational amplifier. A high-gain differential-input amplifier. It has become one of the most versatile and important building blocks in analog circuit design.

Oversampling A/D: A new type of A/D conversion technique in which the input signal is sampled at a frequency much higher than the Nyquist rate.

Passive Components: These components include the capacitor and the resistor.

Phase Margin: A measure of Op-Amp stability; specifically, phase of output voltage at unity-gain frequency relative to 180 degrees of phase lag.

PLL: Phase-Locked Loop is a circuit that locks an input analog signal onto a particular frequency and phase. It is typically used in applications that require the tuning or selecting of communication channels.

Primitive: The lowest-level circuit element in the given circuit hierarchy.

Primitive Recognition: See Circuit Recognition.

Predicate: A fact or rule in Prolog.

Predicate Logic: A formal language of classic logic that uses functions and predicates to describe relations between individual entities.

PMOS: P-channel MOS. An MOS technology that uses only p-channel MOSFETs as the active devices.

PSRR: Power supply rejection ratio. The ratio of the open loop gain of the Op-Amp to the change in the output voltage of the Op-Amp caused by the change in the power supply.

RAM: Random-access memory. A volatile semiconductor memory whose content can be read and written.

ROM: Read-only memory. A semiconductor memory using the presence or absence of a single MOS transistor as the storage mechanism.

Rule: A formal way of specifying a recommendation, directive, or strategy, expressed as IF *condition* THEN *action.*

Sample-and-Hold: A circuit which accurately acquires and stores an analog voltage on a capacitor for a specified period of time.

Scanline-Based Router: An incremental channel router that scans the channel along its spine and evaluates all possible paths between a source and multiple target points.

SCF: Switched-capacitor filter. An implementation of active filters in discrete-time frequency domain (z-domain) using MOS technology. Each resistor is realized with a switched capacitor.

Self-Configuration Technique: A design synthesis technique that allows the circuit topology to be modified to meet the performance specifications during the design process. This method uses circuit primitives such as current sources and active load as "replacement parts" in a design.

Semi-Custom Design: A design style which uses the standard cell or gate array approach.

Sense Amplifier: A circuit that is used to sense the state of DRAM memory cells.

Sensitivity-Based Floorplanning: A heuristic floorplanning procedure in which the relative positions of slicing structures are determined by a zone-sensitivity partitioning algorithm.

Settling Time: The time elapsed from the application of a full scale step input to a circuit to the time when the output has entered and remained within a specified error band around its final value.

Shape Constraint Relation: The set of y-x pairs so that a rectangle module with width equal to x and height equal to y contains at least a shape/orientation realization of the module.

Sigma-Delta A/D: One type of oversampling A/D which uses a 1-bit DAC as the feedback element to achieve the noise shaping function.

Sigma-Delta Modulator: A circuit which implements the noise shaping function in a modulator loop. It converts the input analog signal to a digital bit-stream.

Signal-to-Noise Ratio: The ratio of the RMS voltage of the fundamental harmonic to the RMS noise in a specified bandwidth about the signal frequency of interest.

Simulated Annealing: An approach proposed by Kirkpatrick et al. [64] as an effective method for finding good solutions of combinatorial optimization problems such as floorplanning. By simulating an annealing process, this method generates moves randomly and checks whether the cost of the new configuration

satisfies an acceptable criterion based on temperature.

Single-Ended Op-Amp: The conventional type of Op-Amp which has a single-ended output.

Single-Layer Neural Network: One type of neural network which contains a neuron input stage, a synapse connection matrix, and a neuron output stage.

Sinx/x-Compensated Sample-ans-Hold: A circuit which can perform the sample-and-hold operation without introducing the usual sinx/x distortion.

Slew Rate: The maximum rate, in volts per microsecond, at which the output voltage of an Op-Amp changes when a square-wave or step input is applied.

Slicing Tree: A tree representation of the dissection of the slicing structures.

Soma: The cell body of a biological neuron.

Source Component: The transistor element whose source terminal is directly tied to a power supply.

Source-Coupled Node: A circuit junction node that connects source terminals of multiple transistors.

SPICE: A widely accepted computer program for semiconductor circuit simulation. SPICE was originally written by a group of researchers at the University of California at Berkeley.

Standard-Cell Approach: A design approach in which a library of fixed pre-designed cells is used to implement a circuit function.

Standard-Cell Library: A library of standard cells which contains the most common circuit elements and blocks. These cells typically have fixed heights with power supply rails running along the top and bottom.

Stockmeyer's Algorithm: An algorithm introduced by Stockmeyer [65] to determine the optimal shape and orientation of the modules given their relative positions as a slicing structure.

Switch-box Router: A 2-dimensional router which can route regions with fixed pins on all sides of the region.

Synapse: An input contact to a dendrite that in turn drives a neuron.

Synapse Matrix: A circuit block which contains a matrix of synapse modules.

Synapse Weight: A value which specifies the strength of the coupling between an input of a neuron and an output of another neuron.

Thermal Matching: An effect which can be achieved by properly matching the layout of sensitive circuit primitives with respect to the high-current output devices.

Tools: Computer programs that are used to help solving specific tasks.

Top-Down Design Synthesis: A design style that is used to generate sized device-level schematic diagrams starting from the module (top) level of the circuit hierarchy. This expression is derived from the relative positions of design operations in the flowchart.

Transconductance Amplifier: An unbuffered Op-Amp which has a high output resistance.

Turbo Prolog: One version of Prolog, which is one of the leading languages for artificial intelligence research and applications.

Unity-Gain Bandwidth: The frequency at which the gain of an Op-Amp equals one.

UNIX: An operating system used in Sun workstations. Most UNIX software are written in C.

VCO: Voltage-controlled oscillator. An oscillator whose frequency is controlled by an external voltage.

VLSI: Very large scale integrated circuit. VLSI is used as a general term to indicate today's increasingly complex ICs with a very large number of components.

Voltage Comparator: A circuit that has a binary output whose value is based upon a comparison of two analog voltage inputs.

Zone-Sensitivity Partitioning: An algorithm that partitions a plane into multiple layout zones based on sensitivity levels of circuit primitives.

Appendix F

Publications

Conference Papers:

[1] A. H. Fung, D. J. Chen, Y.-N. Lai, and B. J. Sheu, "Knowledge-Based Analog Circuit Synthesis with Flexible Architecture," *Proc. IEEE Int. Conf. Computer Design*, pp. 48-51, Cambridge, MA, Oct. 1988.

[2] D. J. Chen, J.-C. Lee, and B. J. Sheu, "SLAM : A Smart Analog Module Layout Generator for Mixed Analog-Digital VLSI Design," *Proc. IEEE Int. Conf. Computer Design*, pp. 24-27, Cambridge, MA, Oct. 1989.

[3] D. J. Chen and B. J. Sheu, "Automatic Layout Generation for Mixed Analog-Digital VLSI Neural Chips," *Proc. IEEE Int. Conf. Computer Design*, pp. 29-32, Cambridge, MA, Oct. 1990.

[4] D. J. Chen and B. J. Sheu, "Automatic Custom Layout of Analog ICs Using Constraint-Based Module Generation," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 5.5.1-5.5.4, San Diego, CA, May 1991.

[5] D. J. Chen, W. Ngai, S. Taylor, D. Shum, F. In't veld, M. Uratani, H. Ogawa, S. Hattori, G. Kinoshita, and T. Kojima, "A Single-Chip Mixed Analog/Digital Signal Processor for Voiceband Applications," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 24.4.1-24.4.4, San Diego, CA, May 1991 (from Sharp Corp.).

Journal Paper:

[1] D. J. Chen and B. J. Sheu, "Automatic Layout Synthesis of Analog ICs Using Circuit Recognition and Constraint Analysis Techniques," *Int. Jour. of Analog Integrated Circuits and Signal Processing*, vol. 1, no. 1, Kluwer Academic Publishers, March 1991.

References

[1] P. R. Gray, B. Wooley, and R. Brodersen (ed.), *Analog MOS Integrated Circuits, II*, IEEE Press: New York, NY, 1989.

[2] P. R. Gray and R. Meyer, *Analysis and Design of Analog Integrated Circuits, 2nd Ed.*, Wiley: New York, NY, 1984.

[3] A. B. Grebene, *Bipolar and MOS Analog Integrated Circuit Design*, Wiley: New York, NY, 1984.

[4] Y. Tsividis and P. Antognetti (ed.), *Design of MOS VLSI Circuits for Telecommunications*, Prentice-Hall: Englewood Cliffs, NJ, 1985.

[5] R. Gregorian and G. Temes, *Anolog MOS Integrated Circuits for Signal Processing*, Wiley: New York, NY, 1986.

[6] P. Allen and D. Holberg, *CMOS Analog Circuit Design*, Holt, Rinehart & Winston: New York, NY, 1987.

[7] R. Unbehauen and A. Cichocki, *MOS Switched-Capacitor and Continuous-Time Integrated Circuits and Systems*, Springer-Verlag: Berlin, Germany, 1989.

[8] R. Geiger, P. Allen and N. Strader, *VLSI Design Techniques for Analog and Digital Circuits*, McGraw-Hill: New York, NY, 1990.

[9] M. Haskard and I. May, *Analog VLSI Design -- nMOS and CMOS*, Prentice-Hall: Englewood Cliffs, NJ, 1988.

[10] R. Batruni, W. Hee, P. Lemaitre et al., "Mixed Digital/Analog Signal Processing for A Single-Chip 2B1Q U-Interface Transceiver," *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 26-27, San Francisco, CA, Feb. 1990.

[11] F. Cepl, A. Deierling, O. Duffner, H. Hauer, and D. Seitzer, "Integration of A CMOS Mixed-Analog-Digital Eight Channel Speech Transmission Circuit," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 12.3.1-12.3.4, Boston, MA, May 1990.

[12] D. J. Chen, W. Ngai, S. Taylor, D. Shum, F. In'tveld, M. Uratani, H. Ogawa, S. Hattori, G. Kinoshita, and T. Kojima, "A Single-Chip Mixed Analog/Digital Signal Processor for Voiceband Applications," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 24.4.1-24.4.4, San Diego, CA, May 1991.

[13] T. Schmerbeck, R. Richetta, and L. Smith, "A 27MHz Mixed Analog/Digital Magnetic Recording Channel DSP Using Partial Response Signalling with Maximum Likelihood Detection," *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 136-137, San Francisco, CA, Feb. 1991.

[14] Y. Okada, T. Matsuura, T. Shinmi et al., "A Mixed Analog/Digital Video Signal Processing LSI with On-Chip AD and DA Converters," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 24.1.1-24.1.4, San Diego, CA, May 1989.

[15] M. Ohta, S. Kobatake, K. Kohiyama et al., "A Single-Chip CMOS Analog/Digital Mixed NTSC Decoder," *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 118-119, San Francisco, CA, Feb. 1990.

[16] J.-C. Lee and B. J. Sheu, "Parallel Digital Image Restoration Using Adaptive VLSI Neural Chips," *Proc. IEEE Int. Conf. Computer Design*, Cambridge, MA, Oct. 1990.

[17] C. A. Mead, *Analog VLSI and Neural Systems*, Addison-Wesley: Reading, MA, 1989.

[18] C. A. Mead and M. Ismail, *Analog VLSI Implementation of Neural Systems*, Kluwer Academic Publishers: Boston, MA, 1989.

[19] B. W. Lee and B. J. Sheu, *Hardware Annealing in Analog VLSI Neurocomputing*, Kluwer Academic Publishers: Boston, MA, 1991.

[20] M. Ismail and J. Franca, *Introduction to Analog VLSI Design Automation*, Kluwer Academic Publishers: Boston, MA, 1990.

[21] P. E. Allen and H. Nevarez-Lazano, "Automated Design of MOS Opamps," *Proc. IEEE Int. Symp. Circuits and Systems*, pp. 1286-1289, May 1983.

[22] R. J. Bowman and D. L. Lane, "A Knowledge-Based System for Analog Integrated Circuit Design," *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 210-212, Santa Clara, CA, Nov. 1985.

[23] R. Harjani, R. A. Rutenbar, and L. R. Carley, "A Prototype Framework for Knowledge-Based Analog Circuit Synthesis," *Proc. IEEE Design Automation Conf.*, pp. 42-49, June 1987.

[24] H. Y. Koh, C. H. Sequin, and P. R. Gray, "Automatic Synthesis of Operational Amplifiers Based on Analytic Circuit Models," *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 502-505, Santa Clara, CA, Nov. 1987.

[25] M. Degrauwe, "An Analog Expert Design System," *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 212-213, New York, NY, Feb. 1987.

[26] E. Berkcan and M. d'Abreu, "Physical Assembly for Analog Compilation of High Voltage ICs," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 14.3.1-14.3.7, Rochester, NY, May. 1988.

[27] B. J. Sheu, A. H. Fung, and Y.-N. Lai, "A Knowledge-Based Approach to Analog Integrated Circuit Design," *IEEE Trans. on Circuits and Systems*, vol. 35, no. 2, pp. 256-258, Feb. 1988.

[28] R. Harjani, R. A. Rutenbar, and L. R. Carley, "Analog Circuit Synthesis and Exploration in OASYS," *Proc. IEEE Int. Conf. Computer Design*, pp. 44-47, Cambridge, MA, Oct. 1988.

[29] A. H. Fung, D. J. Chen, Y.-N. Lai, and B. J. Sheu, "Knowledge-Based Analog Circuit Synthesis with Flexible Architecture," *Proc. IEEE Int. Conf. Computer Design*, pp. 48-51, Cambridge, MA, Oct. 1988.

[30] A. H. Fung, B. W. Lee, and B. J. Sheu, "Self-Reconstructing Technique for Expert System-Based Analog IC Designs," *IEEE Trans. on Circuits and Systems*, vol. CAS-36, pp. 318-321, Feb. 1989.

[31] D. J. Chen and B. J. Sheu, "A Flexible Architecture Approach to Analog Circuit Synthesis," *Proc. IEEE Workshop on Analog Circuit Engineering*, Princeton, NJ, April 1989.

[32] D. C. Stone, J. E. Schroeder et al., "Analog CMOS Building Blocks for Custom and Semicustom Applications," *IEEE Jour. of Solid-State Circuits*, vol. SC-19, no. 1, pp. 55-61, Feb. 1984.

[33] T. Pletersek, J. Trontelj, and L. Trontelj, "Analog LSI Design with CMOS Standard Cells," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 479-483, 1985.

[34] C. D. Kimble, A. Dunlop, G. Gross et al., "Autorouted Analog VLSI," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 72-78, Portland, OR, May 1985.

[35] J. Kuhn, "Analog Module Generators for Silicon Compilation," *VLSI System Design*, pp. 74-80, May 4, 1987.

[36] E. Berkcan and M. d'Abreu, "Physical Assembly for Analog Compilation of High-Voltage ICs," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 14.3.1-14.3.4, Rochester, NY, May 1988.

[37] M. Kayal, S. Piquet, M. Declercq, and B. Hochet, "SALIM: A Layout Generation Tool for Analog ICs," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 7.5.1-7.5.4, Rochester, NY, May 1988.

[38] J. Rijmenants, T. Schwarz, J. Litsios, and R. Zinszner, "ILAC: An Automated Layout Tool for Analog CMOS Circuits," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 7.6.1-7.6.4, Rochester, NY, May 1988.

[39] H. Y. Koh, C. H. Sequin, and P. R. Gray, "Automatic Layout Generation for CMOS Operational Amplifier," *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 548-551, Santa Clara, CA, Nov. 1988.

[40] D. Garrod, R. Rutenbar, and L. Carley, "Automatic Layout of Custom Analog Cells in ANAGRAM," *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 544-547, Santa Clara, CA, Nov. 1988.

[41] H. Onodera, H. Kanbara, and K. Tamaru, "Operational-Amplifier Compilation with Performance Optimization," *IEEE Jour. Solid-State Circuits*, vol. SC-25, no. 2, pp. 466-473, Apr. 1990.

[42] D. J. Chen, J.-C. Lee, and B. J. Sheu, "SLAM : A Smart Analog Module Layout Generator for Mixed Analog-Digital VLSI Design," *Proc. IEEE Int. Conf. Computer Design*, pp. 24-27, Cambridge, MA, Oct. 1989.

[43] D. J. Chen and B. J. Sheu, "Automatic Layout Generation for Mixed Analog-Digital VLSI Neural Chips," *Proc. IEEE Int. Conf. Computer Design*, pp. 29-32, Cambridge, MA, Oct. 1990.

[44] D. J. Chen and B. J. Sheu, "Automatic Custom Layout of Analog ICs Using Constraint-Based Module Generation," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 5.5.1-5.5.4, San Diego, CA, May 1991.

[45] D. J. Chen and B. J. Sheu, "Automatic Layout Synthesis of Analog ICs Using Circuit Recognition and Constraint Analysis Techniques," *Int. Jour. of Analog Integrated Circuits and Signal Processing*, vol. 1, no. 1, Kluwer Academic Publishers, March 1991.

[46] L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," *Electron. Res. Lab. Memo UCB/ERL M520*, University of California, Berkeley, May 1975.

[47] B. Johnson, T. Quarles, A. R. Newton, D. O. Pederson, and A. Sangiovanni-Vincentelli, SPICE3 Version 3E.1 User's Guide, *Department of Electrical Engineering and Computer Science*, University of California, Berkeley, Apr. 1991.

[48] HSPICE User's Manual H9001, Meta-Software Inc., Campbell, CA, 1991.

[49] L. O. Chua and P.-M. Lin, *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques*, New York: Prentice-Hall,

p. 215, 1975.

[50] P. R. Gray and R. Meyer, "MOS Operational Amplifier Design -- A Tutorial Overview," *IEEE Jour. Solid-State Circuits*, vol. SC-17, no. 6, pp. 969-982, Dec. 1982.

[51] C. Townsend, *Mastering Expert Systems with Turbo Prolog*, Chap. 2, Howard W. Sams & Co.: Indianapolis, IN, 1986.

[52] D. Shafer, *Turbo Prolog Primer*, Chap. 1, Howard W. Sams & Co.: Indianapolis, IN, 1986.

[53] J. McCreary, "Matching Properties, and Voltage and Temperature Dependence of MOS Capacitors," *IEEE Jour. Solid-State Circuits*, vol. SC-16, no. 6, pp. 608-616, Dec. 1981.

[54] D. Allstot and W. C. Black, Jr., "Technological Design Considerations for Monolithic MOS Switched-Capacitor Filter Systems," *Proceedings of the IEEE*, vol. 71, no. 8, pp. 967-986, Aug. 1983.

[55] R. Otten, "Automatic Floorplan Design," *Proc. 19th Design Automation Conf.*, pp. 261-267, June 1982.

[56] D. F. Wong and C. L. Liu, "A New Algorithm for Floorplan Design," *Proc. 23rd Design Automation Conf.*, pp. 101-107, June 1986.

[57] C. E. Wu, L. M. Ni, and A. S. Wojcik, "Function Recognition of static CMOS Circuits," *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 306-309, Santa Clara, CA, Nov. 1987.

[58] W. C. Black, Jr., D. J. Allstot, and R. A. Reed, "A High Performance Low Power CMOS Channel Filter," *IEEE Jour. Solid-State Circuits*, vol. SC-15, no. 6, pp. 929-938, Dec. 1980.

[59] S. Qin and R. Geiger, "A +5-V CMOS Analog Multiplier," *IEEE Jour. Solid-State Circuits*, vol. SC-22, no. 6, pp. 1143-1146, Dec. 1987.

[60] S. Fujii, S. Saito, Y. Okada et al., "A 50-μA Standby 1Mx1/256Kx4 CMOS DRAM with High-Speed Sense Amplifier," *IEEE Jour. Solid-State Circuits*, vol. SC-21, no. 5, pp. 643-647, Oct. 1986.

[61] N. Lu and H. Chao, "Half-V_{DD} Bit-Line Sensing Scheme in CMOS DRAM's," *IEEE Jour. Solid-State Circuits*, vol. SC-19, no. 4, pp. 451-454, Aug. 1984.

[62] R. Castello and P. R. Gray, "A High-Performance Micropower Switched-Capacitor Filters," *IEEE Jour. Solid-State Circuits*, vol. SC-20, no. 6, pp. 1122-1132, Dec. 1985.

[63] M. A. Breuer, "Min-Cut Placement," *Jour. Design Fault-Tolerant Computing*, vol. 1, no. 4, pp. 343-362, Oct. 1977.

[64] S. Kirkpatrick, C. Gelatt Jr., and M. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, May 1983.

[65] L. Stockmeyer, "Optimal Orientations of Cells in Slicing Floorplan Designs," *Information and Control*, vol. 59, pp. 91-101, 1983.

[66] H. Shin and A. Sangiovanni-V., "MIGHTY: A 'Rip-Up and Reroute' Detailed Router," *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 2-5, Santa Clara, CA, Nov. 1986.

[67] J. K. Ousterhout, G. Hamachi, R. Mayo, W. Scott, and G. Taylor, "MAGIC : A VLSI Layout System," *Proc. 21st Design Automation Conf.*, pp. 152-159, June 1984.

[68] C. Tomorich, "MOSIS-A Gateway to Silicon," *IEEE Circuits and Devices Magazine*, vol. 4, pp. 22-23, 1988.

[69] H. Yaguthiel, A. Sangiovanni-Vincentelli, and P. R. Gray, "A Methodology for Automated Layout of Switched-Capacitor Filters," *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 444-447, Santa Clara, CA, Nov. 1986.

[70] D. Lucas, "Analog Silicon Compiler for Switched Capacitor Filters," *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 506-509, Santa Clara, CA,

Nov. 1987.

[71] R. P. Sigg, A. Kaelin, A. Muralt, W. C. Black Jr., and C. Moschytz, "An SC Filter Compiler: Fully Automated Filter Synthesizer and Mask Generator for A CMOS Gate-Array-Type Filter Chip," *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 510-513, Santa Clara, CA, Nov. 1987.

[72] D. W. Tank and J. J. Hopfield, "Simple 'Neural' Optimization Networks: An A/D converter, Signal Decision Circuit, and A Linear Programming Circuit," *IEEE Trans. on Circuits and Systems*, vol. CAS-33, no. 5, pp. 533-541, May 1986.

[73] B. W. Lee and B. J. Sheu, "A Compact and General-Purpose Neural Chip with Electrically Programmable Synapses," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 26.6.1-26.6.4, San Diego, CA, May 1990.

[74] T. Morishita, Y. Tamura, and T. Otsuki, "A BICMOS Analog Neural Network with Dynamically Updated Weights," *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 142-143, San Francisco, CA, Feb. 1990.

[75] H. Graf and D. Henderson, "A Reconfigurable CMOS Neural Network," *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 144-145, San Francisco, CA, Feb. 1990.

[76] Y. Arima, K. Mashiko, K. Okada et al., "A 336-Neuron 28k-Synapse Self-Learning Neural Network Chip with Branch-Neuron-Unit Architecture," *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 182-183, San Francisco, CA, Feb. 1991.

[77] B. Boser and E. Sackinger, "An Analog Neural Network Processor with Programmable Network Topology," *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 184-185, San Francisco, CA, Feb. 1991.

[78] A. Oppenheim and R. Schafer, *Digital Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1975.

[79] L. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall: Englewood Cliffs, NJ, 1975.

[80] K. Yamamoto, H. Ohtake, and J. Maruyama, "An Analog Front End for 2400b/s Split-Band Full-Duplex Modems," *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 172-173, Anaheim, CA, Feb. 1986.

[81] R. Castello, L. Tomasini, S. Pernici et al., "Analog Front-End of an ECBM Transceiver for ISDN," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 16.4.1-16.4.4, San Diego, CA, May 1989.

[82] J. Roesgen and G. Warren, "An Analog Front End Chip for V.32 Modems," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 16.1.1-16.1.5, San Diego, CA, May 1989.

[83] J. C. Candy, "A Use of Double Integration in Sigma Delta Modulation," *IEEE Trans. Comm.*, vol. Com-33, pp. 249-258, March 1985.

[84] J. L. McCreary and P. R. Gray, "All-MOS Charge Redistribution Analog-to-Digital Conversion Techniques - Part I," *IEEE J. Solid-State Circuits*, vol. SC-10, no. 6, pp. 371-379, Dec. 1975.

[85] Y. Tsividis, "Analog MOS Integrated Circuits -- Certain New Ideas, Trends, and Obstacles," *IEEE Jour. Solid-State Circuits*, vol. SC-22, no. 3, pp. 317-321, June 1987.

[86] H. P. Graf and P. deVegvar, "A CMOS Implementation of A Neural Network Model," *Proc. Stanford Conf.*, pp. 351-362, MIT Press: Cambridge, MA, 1987.

[87] B. W. Lee, "VLSI Design of Adaptive Neural Systems," *Ph.D. Thesis*, Department of Electrical Engineering, University of Southern California, Los Angeles, May 1990.

[88] Professor C. R. Crowell, University of Southern California, *Private communication*, 1991.