COMPUTER-AIDED DESIGN FOR VLSI CIRCUITS

by

A. R. Newton and A. L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M86/16

26 February 1986

COMPUTER-AIDED DESIGN FOR VLSI CIRCUITS

by

A. R. Newton and A. L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M86/16

26 February 1986

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Computer-Aided Design for VLSI Circuits

A. R. Newton and A. L. Sangiovanni-Vincentelli
Department of Electrical Engineering
and Computer Sciences
University of California, Berkeley, 94720

1. INTRODUCTION

Computer aids have been used for both the design and verification of electronic systems for many years prior to the introduction of commercial Integrated Circuits (ICs) in the early 1960s. Such tools have found their way into virtually every aspect of the design of such systems, from IC process technology to the design of complex computer architectures. However, it is the IC and the complex electronic systems the IC has made possible that have made computer aids an indispensable part of the design of an electronic circuit or system. Not only are computer aids necessary for both the design and verification of integrated circuits today but, as the semiconductor processing technologies mature, computer aids will soon also provide key proprietary advantages as semiconductor and system design houses vie for the promising Application-Specific IC (ASIC) market of the next decade. We believe that the pivotal technologies in future IC CAD systems include *tools for IC synthesis*, such as placement and routing, combinational and sequential logic synthesis tools, and architectural design aids, *design system management tools*, including the management of design versions and alternatives in a distributed computing environment, data dependency management, and efficient and flexible interfaces to new tools, *verification tools*, including physical and electrical rules checking, simulation, and formal verification techniques. In many cases, the new verification tools will take advantage of new multiprocessing hardware to improve their performance or use the evolving heuristic programming technologies, such as rule-based expert systems, to improve flexibility and to encourage the evolution of the tool. In the following sections, the state of each of these areas is reviewed and key areas are noted.

In the remainder of this paper, the CAD tools and techniques used to support the most common design styles are reviewed. Nowadays, the field of CAD for IC design is very broad and it is not possible to cover all aspects of IC/CAD in a single paper. For that reason, the paper is focussed on the techniques critical to both custom and ASIC design.

the directions of present research and development for these areas, and future trends. In the following section, some basic cad and design style concepts are introduced and the CAD requirements for ASIC development are explored from a design and marketing point of view. Each of these requirements is then reviewed in the following sections in the context of design management systems, verification tools, and synthesis tools. While the area of testing also involves extensive use of CAD and is also a key technology for IC design, it is dealt with in detail in another paper in this issue and is, therefore, not reviewed here. With the recent rapid increase in compute power per dollar we have seen at the engineer's desk, the next few years promise spectacular progress in all of these areas.

2. DESIGN METHODS AND CAD

2.1. Introduction

The use of a particular class of circuit structures is referred to as a *design method*, or design style, and while the development of new algorithms and techniques for CAD continues, a significant contribution to the design of VLSI circuits will continue to come from the development of new circuit design methods. However, while the implementation of a design method does not *require* the use of computer aids *per se*, the most successful design methods will be those designed to take maximum advantage of the computer in both the circuit design and verification phases. The design method must provide the *structure* necessary to use both human and computer resources effectively. For VLSI, this structure also provides the reduction in design complexity necessary to reduce design time and to ensure that the circuit function can be verified and the resulting circuit can be tested. In describing the variety of computer-aids used for IC design, a distinction is made between those techniques used for *design*, or synthesis, of the IC and those techniques used for its

verification. In both of these categories, a further distinction is made between techniques relating to the *physical*, or topological, aspects of the design process, such as the generation and verification of mask layout data or the placement of components in a circuit, and *functional* considerations, such as logic description, synthesis, simulation, and test-pattern generation.

Computer aids for design, or synthesis, at both the functional and physical levels, are primarily concerned with the use of *optimization* to improve performance and cost. These design tasks may be formulated as combinatorial optimization problems for operations such as cell placement, routing, logic minimization, and logic state assignment, or as parametric optimization problems for operations such as design at the electrical level. These optimization problems are often too complex to solve directly. Therefore, *partitioning* is often used to reduce the problem to a set of simpler sub-problems. The solutions of these subproblems are later combined in a separate step. Both the partitioning task and the solution of each sub-problem generally involves the use of heuristics to reduce the complexity further.

Design methods can be classified in four categories: programmable arrays, standard-cell, macro-cell, and procedural design. A VLSI circuit may consist of one large building block or it may consist of a number of building blocks combined either manually or by a computer program.

A programmable array is a one- or two-dimensional array of repeated cells which can be customized by adding or deleting geometry from specific mask layers. Since a number of processing steps are completed prior to customization, the locations of components on those layers are independent of a particular circuit implementation. Examples of programmable arrays include the Gate-Array[??], Weinberger Array[??], Storage-Logic Array(SLA)[??], Programmable Logic Array (PLA)[??], and Read-Only Memory (ROM).

The *gate-array* (also referred to as master-slice, or uncommitted logic array) is by far the most common programmable array designed by computer. It is also the case that the computer aids for gate-array design are the most advanced and the most mature. In this approach, a two-dimensional array of replicated transistors is fabricated to a point just prior to the interconnection levels. A particular circuit function is then implemented by customizing the connections within each local group of transistors, to define its characteristics as a basic cell, and by customizing the interconnections between cells in the array to define the overall circuit. Generally a two-level interconnection scheme is used for signals and, in some approaches, a third, more coarsely defined layer of interconnections is provided for power and ground connections. The interconnections are implemented on a rectilinear grid in the *channels* between the cells. In many cases, channels are also provided which run over the cells themselves and in some arrays, wider channels are provided in the center of the array to alleviate the congestion often found in that area if particular routing strategies are employed.

Gate-arrays are used in many technologies, in particular bipolar and CMOS, and arrays containing many thousands of gates have been used[56,57]. In the SLA approach, each "gate" consists of a storage element (flip-flop) and a small, uncommitted PLA. This design method has considerable potential for VLSI but effective design-aids for the synthesis of logic functions in SLA form are not yet available.

PLAs may also be used to implement building blocks directly, with storage elements in the feedback path to implement sequential logic in the classical Moore or Mealy style[??]. The PLA consists of a number of transistor arrays which implement logic AND and OR operations. In MOS technology, NOR arrays are used[60]. A conventional PLA consists of two arrays of cells: an input, or look-up, plane followed by an output plane. A *folded* PLA may use additional planes, since rows and/or columns in the structure may be shared by more than one circuit variable, as described later.

The *standard cell* (or polycell) approach refers to a design method where a library of custom-designed cells is used to implement a logic function. These cells are generally of the complexity of simple logic gates or flip-flops and may be restricted to constant height and/or width to aid packing and ease of power distribution. Nowadays, however, state-of-the-art standard cell systems permit cells of different height *and* width to be included in the same design. This results in non-uniform routing channels between adjacent rows and requires a more sophisticated channel routing capability if the silicon area is to be used to its maximum efficiency. Unlike the programmable array approach, standard cell layout involves the customization of all mask layers. This additional freedom permits variable width channels to be used. While most standard cell systems only permit inter-cell wiring in the channels between rows of cells or through cells via predetermined "feed-through" cells, some systems permit over-cell routing if additional levels of interconnect are available. Standard cell systems are also used extensively in a variety of technologies including bipolar and CMOS[??].

It is often relatively inefficient to implement all classes of logic functions in a single design approach. For example, a standard cell approach is inefficient for memory circuits such as RAM and stack. In the *macro-cell*, or building block, method, large circuit blocks, customized to a certain type of logic function, are available in a circuit library. These blocks are of irregular size and shape and may allow functional customization via interconnect, such as a PLA or ROM macro[??], or they can be parameterized with respect to topology as well[??]. With the parameterized cell, the number of inputs and outputs may be parameters of the cell. In some systems macro cells may also be embedded in gate-array or standard-cell designs. The macrocell floorplan style is evolving as the floorplan of choice for large, ASIC designs.

All of the design methods described above may be classified as *data driven*. That is, a description of the required logic function, in the form of equations or an interconnection

list, is used as input to a software system which interprets the data and generates the final design. Techniques have been developed over the last few years which can be classified as *procedure*, or program, driven[??]. These *procedural design* approaches, as well as their advantages and limitations as implemented today, are described below. Most of the 'silicon compiler' companies of today, including Silcon Compilers Inc., Seattle Silicon Technology, and SDL, support macrocell-based floorplans, with procedurally-based *module generators* as described in detail later.

2.2. The Relationship Between CAD and Design

Since the first CAD tools were applied to the IC design process, designers have complained that CAD lags design. There are those who feel that such a situation is inevitable since, once a designer finds a problem for the CAD engineer to tackle, it takes some time for a the appropriate CAD tool to be written, debugged, and documented. By that time, the designer has "moved on" to new designs and, with them, new problems for CAD. In the early days, designers were able to "work around" problems with the CAD tools at their disposal. Today, however, the job just can't be done without CAD.

For an ASIC design environment, a simplified designer's view of the role of CAD is illustrated in Fig. 1. The designer, driven by the marketing need for a circuit that meets a particular cost, performance, and functional specification, works with system, logic, and circuit architectures to create a chip design. In that process, the CAD tools are used to evaluate tradeoffs and alternative designs, to construct specific circuit components, and to assemble and interconnect the components to form the final chip patterns. Once the IC mask patterns have been assembled, CAD tools are used to check the final layout and prepare it for the automated manufacture of masks. As the competition for designs increases, driven by the increasing number of companies in the ASIC business and by the high capital cost of a modern IC processing facility, there is increasing demand for

designers to be able to differentiate their IC product from that of their competitors. Higher performance, lower cost, more features, or a faster time to market are all major factors which differentiate IC products in the ASIC marketplace. In the past, different companies have been able to provide such product differentiation through their IC fabrication technologies: the ability to pack more transistors on a given chip or to provide a higher switching speed per gate drove the designs and their advantage in the marketplace. However, the silicon planar process technology is maturing rapidly — significant gains in performance and density are becoming increasingly expensive and many companies are resorting to "joint ventures," often with former competitors in the United States, Europe, or Asia, to maintain their position in IC process technologies. Because of this decrease in the relative competitive advantage obtained from process technology, semiconductor companies and "silicon foundries" must emphasize other aspects of the design process if they are to compete effectively for the ASIC market. The two avenues available are in architecture — hiring better designers and system architects than their competitors, which is often difficult and is certainly expensive — and in CAD.

Unfortunately, the perspective shown in Fig. 1 is simplistic and incomplete. As illustrated in Fig. 2, the design task involves three major components: CAD programs, support for specific design styles, and support for component libraries. A lack of CAD support in any of these areas may result in a significant reduction in the competitiveness of the designers' final product. On the other hand, a significant proprietary advantage in any or all of these areas will maintain a companies' position as a force in the marketplace.

The CAD tools area may be subdivided further into three areas: tools for circuit design, or synthesis, tools for circuit verification, and tools for design data management and for the managing the flow of the design process. This last category of tools is of particular importance for it provides the foundation on which the CAD system is build. If the design *framework* is inflexible and cannot adapt to new tools, new design styles, and

changes in process technology, then the design system will soon become obsolete. It is also important in maintaining a competitive advantage in the design process since an *open* framework, which support the addition of locally-developed as well as commercial tools, can be used to provide a proprietary difference between one system and a competitor's system. For that reason, this important area is reviewed in detail in the following section.

As mentioned earlier, since design verification has received a great deal of attention in the past, most of the techniques and tools are relatively mature. The major research issues in the verification area concern improving the performance of the tools for large designs without sacrificing reliability of the results. The use of special-purpose hardware and new computer architectures are playing a major role here. In addition, new algorithms are being developed which exploit the properties of large circuits, such as the repetitive use of circuit structures. Many of the new techniques, while novel and requiring large engineering investments to achieve their potential, are relatively easy to duplicate and therefore cannot provide the foundation for a proprietary technology.

On the other hand, with a few notable exceptions, design synthesis systems for ASIC designs are far less mature and large gains in circuit efficiency and design time are still to be had. In addition, many of the state-of-the-art synthesis techniques involve far more "inspiration" than "perspiration" and, as a result, can form the basis of a proprietary and differentiating technology for ASIC design. Techniques for efficient synthesis (system design, register-level design, logic design, placement, routing, and array compilation) will provide a major focus for both University research and Industrial competition over the coming years and, for that reason, they are reviewed in detail in this paper.

The second important area for differentiation is that of CAD support for design styles. In particular, CAD support for floorplan style (gate-array, standard cell, macrocell, etc.) and support for the design of so-called "random" logic — that portion of a design that cannot be cast into a straightforward and efficient regular layout style, such as RAM,

ROM or datapath. Since designers are finding improved circuit design styles and layout styles continuously, it is essential that a CAD system be able to support a variety of design styles and adapt easily to new development in these areas.

Finally, all ASIC systems require a library of primitive components, whether they be individual transistors, logic gates, or entire subsystems. These library cells may be invariant designs, such as the traditional standard cell or gate array building blocks, the may be parameterized cells, such as those in the libraries offered by the "silicon compiler companies", or they may be sophisticated, module-generator-based libraries, where different cell topologies are generated on the fly as a function of the user's input description.

If a designer is to compete in the competitive ASIC marketplace of tomorrow, he must be able to customize his CAD design environment in all three of these areas.

3. THE CAD FRAMEWORK

3.1. Design Data Management

In the 1960's, data-base management was not an issue for IC design — the entire data-base often consisted of a box of punched cards and a hand-drawn roll of mylar that the designer carried with him. In the early and mid 1970's, as circuit complexity increased, proprietary and tool-dependent data formats were developed to represent particular classes of design data, such as mask layout data(e.g. [xymask][stream]) and transistor or gate-level netlist descriptions(e.g. [tegas][spice]). Since most CAD programs were developed independent of one another and had their own input formats, coupling them together to form an integrated system for IC design involved writing translators to and from each program. In the worst case, for N programs, $(N-1)N$ translators would be needed, as illustrated in Figure 3.1(a). However, the CAD tools were evolving and their input formats were changing along with them. As a result, it was often necessary to

keep a family of translators for each program, with each translator corresponding to a different version of the input data format. Maintaining such a family of translators soon became a CAD manager's nightmare! The number of translators can be reduced to a worst-case of $2N$ by choosing a common, central format and translating to and from that format, as shown in Figure 3.1(b). A number of *defacto* standard formats evolved in the late 1970's to meet the need for a common format and different companies standardized internally on one format for each class of data. In the mid and late 1970's, a number of public-domain standard formats were adopted and the most successful examples are the CIF (Caltech Intermediate Form)[m&c] for mask-level layout descriptions and SDL (Stanford Design Language)[sdl] for gate-level netlist descriptions. While such formats provide a consistent way of *storing* the design data, there is no support for *managing* the data — Which copy is the latest version? Has the layout been changed since the schematic diagram was updated? If I change this cell, which cells that use it will be affected? It is the ability to answer such queries that differentiates a true data management system from a simple data repository.

In parallel with this work, a number of companies developed conventional database systems for managing their IC design data. Often these companies were the large computer or system houses who had experience with the use of database management techniques for discrete digital system design. These *record-oriented* database management systems (DBMS) were developed to manage IC *parts* inventories, part location on standard printed circuit (PC) boards, and the connections among IC pins necessary to implement the logic schematic. These lists of connections, used to guide wire-wrap or stitch-weld machines, are generally referred to as *netlists*. While these companies found that the application of conventional relational, network, or hierarchical database management techniques was effective for structured, semi-custom design styles like gate-array and standard-cell, these approaches were not successful for custom design styles or in situations where the underlying process technology and design style was evolving rapidly[4,5]

However, the same rapid increases in complexity that makes the use of conventional database management techniques difficult has made the need for a unified data management system critical, especially for full-custom or structured-custom design styles. No longer is the entire design process the responsibility of a small, tight-knit group but rather teams of system designers, logic designers, circuit designers, and layout technicians must all work together and share the vast amount of data representing an modern IC-based system.

The representations of IC design data, such as mask layout, schematic diagrams, documentation, simulator input and output, are quite diverse and new representations are being developed continuously. This evolution requires a flexible data management system which can adapt readily to new design methods. The use of conventional database management technology in this area has met with limited success[4,5]. The major limitations here are related to problems specific to engineering applications, while todays data base technology has often evolved from the business area. While specific differences between the needs of the business world and the needs of IC designers can be used to illustrate the problems here, simply solving these problems may not be sufficient. Rather, a completely different approach to the problem is necessary. These systems also exhibit low efficiency compared with the special-purpose solutions that have often been developed in-house. The requirements of a data management system for custom design include: access methods for storing and retrieving geometrical data, multiple versions and design alternatives, back-out of nested transactions, support for workstation and network-based transactions, procedural attachment, and near-optimal performance with relatively cheap hardware.

An analogy can be used to explain where conventional database management fails for custom IC design. Many researchers have noted a strong similarity between the custom IC design process and writing computer programs — mask layout is akin to a binary

image, symbolic layout is analogous to assembly code, and gates or modules are compared with lines of code in a high-level language. The sorts or operations the programmer performs on code, the IC designer would like to be able to perform on the IC design data. In fact, it is from such an analogy that the terms *silicon compiler* and *silicon assembler* evolved. Taking the analogy a step further, it is worth noting that programmers do not store their code in conventional DMBS systems. Rather, they have used conventional file systems (nowadays, often organized hierarchically) with tools to aid management of their code, including source-code control systems (e.g. [SCCS]) and dependency management tools (e.g. [make]). In fact, these tools add to the programming environment many of the capabilities provided by modern database management systems. Over the past few years, a number of IC design data management systems have been developed based on this model [squid] and have been used effectively for custom IC design.

In recent years, the notion of procedural circuit design[14-16] and the rule-based expert system technology have emerged as key components in the design process. These techniques, coupled with the ever-moving boundary between entire systems, printed-circuit boards, and chips, have broadened the requirements for an integrated design system. What is needed to support this work is a flexible design and programming environment that allows a variety of approaches to design to coexist and permits system-level, logic-level, and circuit-level designers, as well as CAD algorithm developers to work together in a single, unified environment. The keys to such a system are common levels of abstraction and standard interfaces among them, as well as a powerful set of synthesis and verification tools which form the basis on which further research is carried out. Each *object* in such a system may be described by *data*, such as its mask layout, by a local *procedure*, such as a parameterized cell, by generic synthesis tools, such as a channel router or a placement program, or by a combination of all of these techniques.

As software systems continue to grow in size and complexity, programmers have turned to *object-oriented* approaches to code development and support (e.g.[flavors][smallTalk][loops]). The next generation of workstations, with an order of magnitude increase in performance at the desktop for comparable price to workstations of today, will be a key factor in making such approaches practical and affordable outside the research laboratory. In an analogous way, IC designers are beginning to develop and use *procedural* descriptions of design components, akin to the objects in many of these languages. In addition, the database management community is directing its attention to the management of object-based descriptions of systems. From an IC design point of view, these three technologies will converge in the next generation of data management and programming systems for IC design. The interfaces to these systems will be indistinguishable from that of an object-oriented, message-based programming environment.

Unfortunately, it is unlikely that a standard, object-based data management interface will be developed in the near future. There is still considerable research required to fully understand the issues involved before a suitable standard can be developed. In addition, competitive market pressures will continue to keep such interfaces proprietary. However, there is a need to move data from one design system to another. The design data represents the "life blood" of an IC design company. If a particular design tool does not function correctly under certain conditions, or a workstation or mainframe computer fails, the problem can generally be overcome and work can proceed. However, if the design data were to be lost in the middle of a large design project, the cost could be astronomical. Not only would the investment in design effort be lost to that point but such a situation would also cost valuable time and a market window might be missed. This is one reason why most IC design companies have resisted trusting all of their data management tasks to a single vendor, particularly if it is not possible to archive all of the data in a non-proprietary format. In addition, once a company has committed their data to a particular vendor's system, they are "locked in" to that vendor unless there is a way of migrating

the data to another system.

Another need for data transfer arises in situations where more than one design system or design site is involved. For ASIC design, schematic capture and simulation may take place on a low-cost, customer operated workstation, while the actual silicon implementation of the design occurs at the ASIC vendor's plant.

To meet these and other needs, standard, textual interchange formats are being developed for IC design data. In some cases the definition has focussed on the data necessary to support a particular design style. For example, the EDIF(Electronic Design Interchange Format)[edif], has been developed by a broad base of semiconductor manufacturers, CAD workstation vendors, and system houses to address the needs of ASIC-based system development. In other cases, a broader charter is being attempted, such as the efforts to standardize hardware description languages and behavioral descriptions of systems. The VHDL(VHSIC Hardware Description Language)[vhdl] and HSL-FX[hslfx] activities are the most active in this area today.

In summary, we believe that true *open access* to design data is an essential first step in the development of a data management framework. Such access must occur at two levels — an object-oriented, programming interface to all data in the database, and a non-proprietary, standard textual format which reflects as much of the semantic content of such systems as is practical.

3.2. Representations of the Design

Throughout the IC design process, a variety of different representations or *views* of the design are used. These representations may reflect a particular level of abstraction, such as the functional specification of the circuit or its mask layout, or they may reflect the view required for a certain application, such as the information required for simulation. The choice of appropriate representations for each level of the design process is a key

factor in determining the effectiveness of computer aids since it is via these representations that both the structure of the design as well as specific information relating to a particular design level are expressed. The design process then involves transformations between these representations, both for design and verification. In this section a brief review and classification of the most common representations is presented. This classification is used in the later sections to relate different design aids.

While the particular set of representations used in a design depends on the particular design approach being used, the major categories may be defined as shown in Figure 3.2. These representations fall into three major categories: behavioral, schematic, and physical. At the behavioral or algorithmic level, functional intent of the design is described independent of a particular implementation. In most cases, programming languages such as concurrent Pascal[99] or Modula 2[modula] have been used to represent the design at this level, as well as providing a simulation capability. Languages specifically designed for this task have also been developed[114,116-117].

Once a functional implementation strategy has been determined, a schematic view may be generated. At its most abstract level, this schematic view consists of a *chip plan*, illustrating the loose physical placement of the major components and busses. Depending on the complexity of the system, this description may be classified as a Processor, Memory, Switch(PMS)-level description[pms] which describes a system as an interconnection of processors, memory modules, peripherals, and switching networks, or a Register Transfer Level(RTL) or microinstruction-level description, defining the functional relationships between the major components of the design. A separate representation that is used for programmable systems is an Instruction Level description that describes the instructions of the machine.

As the implementation is refined further, logic gate level and finally transistor level schematics may be generated. While the nature of the information contained at each level

is different, each more detailed view may be considered a different level of "zooming in" on the implementation. With each new level of refinement more information concerned with the detailed physical and functional implementation of the circuit is included in the description. The final transformation consists of the generation of detailed, mask-level geometries from a device-level schematic view.

The transition between functional and schematic descriptions, and between schematic and mask layout, may involve the use of additional views. The two most common approaches to transforming a behavioral description to a structural, or schematic, representation are the extraction of *control-flow* and/or *data-flow* information. The two approaches differ in the way they derive sequencing information from the behavioral description of the system. Over the past few years, a number of control-flow-based languages and data-flow-based languages have been developed or adapted to meet these needs, as described in more detail later. While CAD tools are available to help perform this transformation for restricted underlying hardware architectures, it remains an area where human creativity generally outperforms the CAD tools.

Symbolic layout forms a bridge between a schematic view of the circuit and its mask-level layout. A symbolic layout contains explicit connectivity information as well as the relative placement of circuit components, such as transistors, to form a basic circuit cell, cells to form a building-block, and building-blocks to describe the entire circuit. At the transistor level, one particular form of symbolic layout is called a *stick diagram* since interconnections are represented by their centerlines and hence resemble sticks. In Figure 3.3, a schematic diagram, its symbolic layout, and the corresponding mask-level figures are shown to illustrate the bridging role. One of the key advantages of a symbolic layout is its ability to maintain explicit electrical connectivity information through to the mask level descriptions. Not only can symbolic layout be used to aid the verification of the circuit, but by separating layout-sensitive cells and interconnections, computer programs can

be used to optimize the area utilization of the circuit by modifying only the noncritical interconnections. This process is called *compaction*, or *spacing*, and is described later.

Once an appropriate set of representations for a particular design method has been determined, important that an integrated set of computer aids, coupled with a unified approach to data management, be provided to the IC designer[118].

At each level of the design process described above, these descriptions must permit the structure of the design to be expressed in such a manner that it can be exploited by the design aids. In particular, *regularity* and *hierarchy* must be exploited. For example, regularity in the form of one- or two-dimensional arrays of similar, e.g. RAM, or iterated, e.g. ROM, components can reduce the design time since only a small number of basic circuit types need be designed by hand. The verification time is reduced also since only one example of each possible spatial combination of this small number of cells need be verified to certify the entire array. Hierarchy can aid the verification process in a similar manner. The components of a circuit block, such as the logic gates used to implement an arithmetic-logic unit, need only be checked in detail once. When the composition of these cells is checked, only the relationships *between* the cells need be verified. A detailed check of the internals of the cells is not necessary. If these cells are used a number of times, this process can provide substantial savings in computer time. Circuit structure can also be exploited in other areas, such as simulation, circuit synthesis, and testing, as described in the remaining sections of the paper.

3.3. User Interface

Since early CAD tools were developed in isolation and often for batch, punched-card environments, diverse card-image-oriented data entry and card-image or line-printer-oriented data output formats evolved. For mask artwork entry, a digitizing table and puck were in common use until only a few years ago. These systems have been displaced

by interactive graphics-entry stations [calma][applicon][cv] and significant improvements in layout productivity were reported using these systems. However, since these systems were very expensive costing over $130,000 per station, they were only used by experts trained for fast layout entry and were kept busy around the clock. Noone would think of using one of these stations to write a memo and only occasionally are they used for entering a schematic diagram. Separate, low-cost entry stations were used for these tasks.

Over the past five years, the advent of low-cost, high-resolution bit-mapped graphics terminals and workstations has altered the economics of user interface dramatically. Low-cost artwork entry systems[kic][caesar][magic] and schematic entry systems have evolved to compete with the more expensive systems and it has become cost-effective to have such capability at each engineer's desk. In the more advanced CAD environments, general-purpose computing functions such as computer mail and networked file systems are also integrated with the CAD interfaces. In most cases, the user interface is similar to that developed at Xerox PARC in the early 1970's[parc] and as embodied in the Apple Macintosh and Lisa families of personal computers — separate windows for each application, pop-up or pull-down menu-based command selection, and bit-mapped graphics interfaces to most tools. In the most recently announced systems, the command syntax is also common for all tools from schematic editing and mask layout to testing and report preparation. A typical screen from such a system is shown in Figure 3.4.

However, from a framework point of view, it important that the user interface be exposed to the CAD developer for the addition of new tools. Ideally, in a consistent object-oriented environment, the user-interface (windows, menus, menu selections, etc.) would be treated as objects in the same environment as the data management system. With the exception of some experimental, in-house systems, there are no systems available today that provide such a uniform interface for the tool builder or CAD system developer.

4. VERIFICATION

4.1. Introduction

To shorten the design cycle and to decrease design costs, it is crucial to eliminate as many errors as possible before manufacturing an integrated circuit. Verification compares the design at a certain level of the hierarchy, e.g. circuit, logic or structural level, with a set of specifications identifying possible inconsistencies between specifications and design.

Verification tools have been the first CAD tools to be developed and are probably the most used tools for the production of integrated circuits. There are several different kinds of verification tools. We classify them as: *structural verification* tools, *simulation* tools, *performance verification* tools, and *logic verification* tools.

Structural verification is the task of verifying that the *structure* of a design — the arrangement of mask-layout shapes, the connections among those shapes or among the components of a design — satisfy a particular set of rules. For structural verification, the behavior of the components of the design is not considered, only there spatial relationships and connectivity.

Simulation has replaced bread-boarding for the functional verification of integrated circuits since the late 1960's. Components of the design and their interconnections are represented by mathematical models of different complexity according to the level of accuracy desired and to the representation available for the design at a particular stage of the design process. Then, input patterns are presented to the mathematical models and the corresponding outputs are obtained by solving sets of equations. These equations can be very complex, such as partial differential equations for process simulation, or rather simple, such as Boolean equations for register transfer level simulation. The outputs are then compared to the expected outputs.

While simulation has been used successfully for the verification of large circuits, it cannot guarantee that certain timing specifications are met for all possible input combinations unless all such combinations are tried - an often impractical proposition. Performance verification techniques aim at the determination of the critical delays in a circuit independent of the input patterns.

Logic verification tools are also input pattern independent and verify that two design descriptions at gate or structural level and functional level are formally equivalent. In general, these techniques are much more expensive than simulation but their use increases the level of confidence in the design and are therefore more and more important as the complexity of IC design grows.

4.2. Structural Verification

Structural verification is subdivided into three areas: *layout-rule checking*(LRC), where mask geometries are verified to check if they satisfy a set of spacing, sizing and enclosure rules, *electrical rule checking*(ERC), where the circuit schematic is verified to find electrical errors due to wrong connections of the devices, and *connectivity verification systems*(CVS), where a netlist description extracted from the layout is compared against the netlist description extracted from an alternate description, such as the schematic diagram.

For custom and structured custom design styles, structural verification has made possible a substantial reduction in the design time needed to obtain functionally correct circuits[lattin]. However, structural verification alone cannot provide a guarantee that the design meets necessary performance specifications.

For design styles where the circuit is constructed automatically from pre-characterized and verified cells using computer programs, such as the standard cell and gate-array design styles, or where large areas of the chip are constructed automatically

using module generators, these techniques are used more as a final verification of the output of the CAD synthesis tools than as an active part of the iterative design process itself. For example, a design rule checker may be invoked after an automatic place-and-route system and the following symbolic spacing system have completed a layout, to verify that all the interconnections are properly spaced and that all the components of the design have been placed so that the design rules are satisfied. In our own experience with synthesis tools, we have found that a verification step after the synthesis step often detects obscure bugs that would have been very difficult to find otherwise. In addition, in systems where the user is permitted to modify the output of the synthesis system (for engineering changes, or "green wires", or to improve the quality of the final output), a verification step is not only useful but essential.

4.2.1. Mask-Level Layout Verification

In some design methods, such as full custom or structured custom, the mask geometries corresponding to the devices are entered manually by the designer. *Physical layout rules* or *design rules* specify the the legal or illegal relationships among the polygons used in the IC mask-making and fabrication process to implement the final circuit. These rules account for necessary electrical separation of different components and signals as well as for imperfections in the mask preparation and manufacturing processes. If a designer enters the geometries of the masks such that they satisfy all layout rules, such as minimum spacing minimum size, and minimum enclosure constraints, then the distortions occurring in the translation from the original drawings to the actual geometries in silicon should not reduce the yield of the design significantly.

Industrial rule-sets can be very complex, especially when the shapes of geometries are not restricted. In addition, electrical considerations may add to the complexity of the rules. For example, capacitive coupling between lines requires that long, parallel lines be

spaced more conservatively than short parallel runs. For high-performance circuits, as the sizes of components continues to decrease, the importance of parasitic effects such as capacitance, resistance, and inductance is increasing and adding complexity to the electrical aspects of the rules.

In the early days of ICs, layout rules were verified by hand-checking the Mylar masks or the layout plots obtained after digitizing the design. When the number of circuit components was small, this procedure was feasible, but as the size of IC designs increased the time required for manual checking increased, along with the probability of missing an error. In the mid-1970's, computer programs for automatically checking layout rules began to find widespread use[ROS74][MITC74][YAM74]. It is necessary that all the violations be reported by a Layout-Rule Check (LRC) program; missing even a single a rule may affect significantly the yield and the performance of a circuit. Often, the types of rules required to check a new technology are more advanced than the rule-specification language can handle. For example, if the rule involves specific edges of a figure while the rule-language only permits specification in terms of entire figures, the user must either omit the rule, and run a risk of a missed error, or specify a more conservative rule in terms of figures, in which case the LRC program will probably report many "false errors" — situations the rule specifies as being in error which are actually not errors — as well as any real errors that may occur. In addition, LRC programs must be flexible enough to deal with different technologies (e.g. NMOS, CMOS, bipolar) and with different processes.

Because of the size of IC designs today, a complete layout rule check requires that millions of polygons must be inspected with several rules each. The running time of a batch LRC program is often of the order of days on a super minicomputer or mainframe. While early algorithms used for checking layout rules often showed $O(N^{1.5})$ performance, where N is the number of figures on the layer being checked, over the past decade researchers (e.g.[BAI76][BEN80]) have been able to reduce the expected run time to

almost linear in the number of mask shapes. Once the analysis has been performed, the errors have been reported, and the false errors have been discounted, the true errors must be corrected. Fixing such layout rule violations is an expensive process in a batch checking environment because a layout rule check must be performed every time the layout is modified to verify that no new errors have been introduced.

LRC programs have been developed in house by large merchant and captive semiconductor companies such as INTEL[WAG75], AT&T Bell Laboratories[MITC74], Hitachi[??] and IBM[??], and by vendors such as Calma, NCA, Phoenix Data Systems, Metheus, ECAD and SDA Systems. These programs can verify complex design with complicated layout rules. Some offer the users a language with which new layout rules can be added rather easily[SCH85]. Over the past few years, in conjunction with VLSI design courses[MEA81], several Universities have developed LRC programs which are based on the Mead and Conway simplified *lambda-based* rules. Because the design styles used often require *Manhattan* geometries (layouts comprised of rectangles only, with their edges aligned with the X and Y axes, akin to the organization of streets in down-town Manhattan) only and do not consider conditional rules, these programs are relatively simple and very fast.

A number of new approaches to LRC have emerged from this activity and have found application in industrial products as well. For example, the concept of *hierarchical* [LAN83] and *incremental* [KIC80][CAE80][MCC83][OUS85] LRC have been explored over the past few years and have been implemented in industrial systems[SCH85]. In incremental approaches, a background process checks the layout rules in the vicinity of each figure as it is added to the layout. Since manual layout is a slow process relative to the computing speed of a modern workstation, this is an effective way of using machine cycles that might otherwise have been wasted. It also permits errors to be corrected and re-checked in a tight, local loop so that the number of expensive, batch-mode checks can be reduced or even eliminated completely, interactively. However, incremental LRC

must be applied so that temporary LRC violations consciously introduced by the designer as an intermediate state are not continuously reported, disturbing the editing session. This problem is handled using one of several techniques. One approach is to maintain a file of layout rule violations, constantly updated in the background, that can be graphically displayed at the user's request. Another is to verify geometries only at the user's request, avoiding checks on regions which are not completed by the designer[KIC83][CAE83].

Hierarchical LRC takes advantage of the fact that cells are often used more than once in a large design. Once the insides of a cell have been checked, the cell is marked as "done" and then for each use, or instance, of the cell only the local context of the cell need be checked. In fact, it is not the *hierarchical* property of the design that really accounts for the savings but rather the *repetition* of identical cells or collections of figures. As ICs increase in size, the complexity of the design is often managed by increasing the *regularity*[LAT79] or repetition of cells in the circuit so that tools which exploit this fact often show large performance gains. In a modern, symbolic design system, where cells are often parameterized and many different variations can be created form the same master cell, or where a spacing system may adjust each cell differently to meet external constraints such as cell pitch, the advantage of hierarchical LRC is quickly lost[SHA85].

The techniques used by LRC systems can be classified into three categories: region-operation-based, raster-based, and corner-based. An excellent review of these techniques can be found in [ARN85].

A large number of LRC systems use variations of the region-operation approach. In this approach, layout rules are expressed as a sequence of selection operations, isolating regions to which the rules apply, followed by a check. More complicated rules may require dozens of operations. Boolean operations such as AND, OR, AND-NOT, and sizing operations such as GROW and SHRINK, are implemented to identify the regions and to perform the checks. Most of these LRC systems represent the regions in terms of their

edges and the operations are usually specified in terms of edges. Scan-line algorithms are generally used to process effectively massive number of edges [BEN, LAU]

Raster-based approaches represent the design in terms of a raster grid, where each of the raster pixels is labeled with the mask layers present in that location. The amount of data required by this approach is obviously very large, much larger than in the previous approach. However, some savings can be achieved by using hierarchical storage schemes where contiguous regions containing the same type of pixel are treated as a single data object[??]. The verification can then be carried out with very simple algorithms. Because of the uniform representation of the data that is provided by this approach, the raster representation supports the use of special-purpose hardware in a straight-forward way. Several hardware accelerators for raster-based LRC have been proposed but no one has been used in production at this time. While this approach is certainly interesting, much work remains to make it practical.

The corner-based approach uses pattern-directed rule application [ARN82]. In this respect, it can be considered as an implementation of a rule-based expert system. Patterns at each corner of the geometries determine which rules to apply and which tolerances are to be checked. Present implementations of corner-based LRC, such as Lyra and Leo [ARN85], are limited to Manhattan or 45 degree angles and cannot handle wide-region operations needed to check conditional rules without some extensions. A rule-based approach has also been applied successfully using an edge-based representation in the Magic system[TAY84].

4.2.3. Extraction and Electrical Rule Checking

Once the mask patterns satisfy the physical layout rules, it is necessary to verify that they will actually implement a working circuit. The first step in this process is to recreate a netlist description of the circuit from the mask pattern data. This process is called *extraction*. Since the only information present in the mask layout data is the relative placement of shapes on different mask layers, mask-level operations must be performed to recognize individual components, such as transistors, capacitors, and nets. For example, in an NMOS technology, a transistor may be formed where figures on the layers **poly** and **diffusion** overlap one another. So a rule for recognizing a transistor might be expressed as:

(define transistor (and poly diffusion)))

where a transistor is defined as that region where the logical **and** (intersection) of figures on the **poly** layer and the **diffusion** is not empty. Of course, in real systems the rules are significantly more complicated with many exception conditions. Since the types of operations necessary to recognize components from the layout are very similar to those used for checking the layout rules, it is not surprising that in most cases a LRC program forms the basis of an extraction program. The extraction program also determines parameter values for simulation, such as the sizes of the transistors extracted as well as related parasitic capacitance values. Depending on the design styles supported by the implementors of the extraction tool, the program may only extract gross components such as transistors, lumped parasitic resistance, and parasitic capacitance from interconnect to ground[FIT83], or it may perform a very detailed extraction including interlayer and inter-figure parasitic capacitance calculations [gummel] and even solve two-dimensional field equations where necessary[mitClue].

Like LRC, extraction can take many hours of computer time to perform if and accurate analysis of an entire chip is required. In addition, if coupling capacitances between

parallel lines are extracted and RC-networks are used to represent interconnections, the amount of output data generated can be massive. While many of the industrially-developed extractors are used to obtain detailed parasitic information, those in general use in University design systems tend to favor speed rather than detailed analysis. In the most common University design styles for VLSI circuits, the layout rules and electrical requirements are constrained to reduce the likelihood of parasitic components altering the function of the circuit. Recently, the concepts of hierarchical and incremental extraction has been developed which makes it possible to verify modifications to design interactively without sacrificing the level of verification that is needed in quality IC designs [SCO85].

The extracted netlist provides the basis for a number of additional checks. In particular, connectivity verification, as described in the next section, and electrical rules checking. Electrical rules checking programs evolved from simple implementations that searched the extracted netlist for ridiculously large or small transistors or direct short-circuits among the power supply lines, clock lines, and the ground line to programs which check for more complex relationships. Such rules include searching a local area of a circuit to be sure transistors related to one another in an electrical sense all have the correct sizes to ensure correct circuit operation. In general, however, new rules were added to these programs by adding additional "hard-wired" procedures or data structures. Recently, a new breed of electrical rule checking programs has evolved based on the use of rule-based expert systems[critter][dialog][rubicc][cv]. Here, the rule-based system is used as a convenient way of expressing the required relationships among components and signals. The fast pattern matching facility of such systems is then used to recognize specific arrangements of components and apply the rules to those arrangements. The rule-based approach provides a convenient programming environment for adding additional, and higher-level, checks.

4.2.4. Connectivity Verification

If a transistor-level netlist description of a circuit is available, either from manually-generated simulator input data or from a schematic entry system, the pattern of interconnections among those components and the pattern of interconnections obtained from the extracted netlist can be compared. This process is called *connectivity verification* and connectivity verification systems (CVS) have been used over the past decade to improve dramatically the probability of functionally-correct silicon on the first fabrication run.

Connectivity verification was first used at the board level, for comparing a logic schematic input with a placed and routed board-level implementation of the circuit[apples].

In general, the comparison involves a one-to-one correspondence between the circuit elements, such as transistors, and the nets in the two circuits, not a functional equivalence. Each circuit is represented by a graph, where the nodes in the graph represent either the circuit elements[wombat] or the nets[gemini], and the nets or circuit elements, respectively, are represented by the arcs in the graph. In some cases, both the circuit elements and the nets are represented by nodes in the graph and the arcs simply represent the connections between them. The problem of determining that the two graphs are the same is equivalent to the *graph isomorphism* problem, a well known combinatorial optimization problem. The worst-case complexity of graph isomorphism is not known. No algorithm has been found with running time bounded by a polynomial in the size of the input, i.e., number of nodes and edges of the graph, but it has no been proven that this problem belongs to the class of NP-complete problems. However, good heuristics are available which can quickly detect if two graphs are isomorphic in most of the cases. If they are not isomorphic, the programs can then isolate the subgraphs that differ in the two netlists. This information is then provided th the user who tries to locate and correct

the error.

There are two basic algorithms in use today for comparing two circuits: *signature calculation* (using element signatures as in the Wombat program [14,15] or using node signatures as in the Gemini program [16]). and *path tracing*. Several approaches have been described in the literature and most LRC/ERC vendors and large IC companies have developed a connectivity verifier.

In the signature calculation approach, signatures are calculated for each element or node in both circuits. A signature is a combination of information about the element or node and its neighbors. The signature can be thought of as a hashing function. All unique signatures in each circuit are compared and any elements or nodes with the same signatures between circuits are marked as the same. This process is repeated until all elements are marked or no more unique signatures can be generated. The information gained on each iteration is fed back into the signature calculations. Almost all connectivity verifiers can handle the straight-forward problem very efficiently with fast, heuristic algorithms. However, most of the time in these programs is spent handling the special cases.

There are many special cases that can degrade the performance of the basic algorithms. Two such cases are *terminal permutability* and *parallel paths*. For some elements, the terminals are logically and/or electrically equivalent and are allowed to permute. The inputs to the basic logic gates (NAND, NOR, etc.) and the source and drain of MOSFETS are examples of such situations. In handling terminal permutability, many connectivity verifiers assume that they will be working with MOSFETS and "hard-wire" the fact that sources and drains can permute; others allow the user to specify how terminals on arbitrary elements can permute, but some do this very inefficiently and others do not always work.

Identical or nearly-identical parallel paths (as in bit-slice circuits and RAMs) also present a problem to current connectivity verifiers. If the paths are identical, the algorithms currently used can not distinguish between the paths and may not be able to handle them. In that case, the program may make a random binding of two elements from the possible candidates and proceed. If it discovers later that the choice was erroneous, it must undo the binding and choose another one. Also, if two paths have only small differences (as in ROMs), since only local effects are taken into account, connectivity verifiers also may not be able to distinguish between them.

** CV **

4.3. Simulation

4.3.1. Introduction

For circuits made from discrete components, *bread-boards* (prototype boards with discrete components) were used extensively to check the functional correctness of the design as well as its performance. However, this approach does not work well for integrated designs since the parasitics on the bread-board are quite different from those on the IC and the thermal and electrical characteristics, as well as component matching properties of the discrete components are also quite different from their IC counterparts. For this reason, electrical circuit simulation was one of the first CAD tools to be developed for IC design and has completely replaced bread-boarding for analog and digital cell design. For large digital designs, breadboards are still used for software development.

Many different forms of simulation can be used for the verification of large digital integrated circuit designs at the various stages of the design process. They may be classified as *Behavioral* (also called algorithmic or functional) simulators, *Register Transfer Level (RTL)* simulators, *Gate Level Logic* simulators, *timing* simulators, *circuit* simulators, *device* simulators, and *process* simulators.

Behavioral simulators [51] are used at the initial design phase to verify the *algorithms* of the digital system to be implemented. Not even a general structure of the design implementation is necessary at this stage. Behavioral simulation might be used to verify the communication protocols in a multiprocessor design, for example.

Once the algorithms have been verified, a potential implementation *structure* is chosen. For example, a microprocessor, some memory, and a special-purpose input-output module may be chosen to implement the handshaking protocol mentioned above. An RTL simulator can be used to verify the design at this level. Only crude timing models may be available, since the exact circuit parasitics and other implementation details are not yet known. Useful information relating to congestion and hardware/firmware tradeoffs can be obtained from this level of analysis. A variety of RTL languages and associated simulators have been described in the literature [29].

Depending on the design methodology and certain technology issues, a gate-level design may be undertaken, where each of the RTL modules is further partitioned into low-level logic building blocks, or gates. A logic simulator may then be used to verify the design at this level. Sophisticated delay models may be introduced and testability analyses performed.

From the gate level, transistors and associated interconnections are generated to implement the design as an integrated circuit. Accurate electrical analysis can be performed for small parts of this design using a circuit analysis program [1][54] or larger blocks may be analyzed in less detail using a timing simulator [9-10][13]. Once the integrated circuit layout is complete, accurate circuit parameters, such as parasitic capacitance values and transistor characteristics, may be extracted and used at the electrical level. These analyses may then be used to improve the delay models at both the gate and RTL levels to verify the circuit design using accurate timing data.

Device simulators are used to verify whether the device characteristics corresponding to a particular sequence of processing steps are close to a ideal device characteristics.

Finally, the design of a new process or the tuning of an exiting process is aided by process simulators, where the control parameters of the process, such as furnace temperature and initial impurity doping densities, are the input variables and physical information such as impurity profiles are the outputs of the simulator.

A number of simulators have been developed recently which span a range of these levels in the simulation hierarchy. These simulators are called *mixed level* simulators [40-42] and allow different parts of a circuit to be described at different levels of abstraction. Not only does this approach permit a smooth transition between different levels of simulation (since the same simulator and associated input processor is used) but it allows the designer to take advantage of the time and memory savings available from higher-level descriptions of parts of the circuit.

Excellent tools, either developed in house by IC manufacturer or by tool vendors, are available at all levels of the hierarchy. The parameters often used to judge the quality of a simulation tool are accuracy, speed, and flexibility. By flexibility, we mean the range of analyses supported by the program (time-domain transient, small-signal frequency-domain analysis, timing verification, fault simulation, etc.) and the range and quality of the component models it provides (both N and P channel MOSFETS, bipolar transistors, bi-directional switches, built-in registers, etc.). Because of the increasing size of IC designs, even the fastest simulators are not able to perform simulation as extensively as desired by today's designers. For this reason, several hardware accelerators have been developed for simulation and new algorithms are being explored to exploit multi-processor architectures (see the window in page for a detailed discussion of the use of multi-processors for simulation).

4.3.2. Process and Device Simulation

The electrical characteristics of IC devices depend very strongly on the manufacturing process. This process continues to become more complex and more sophisticated and it is often difficult to relate specific processing steps with the overall device characteristics obtained after manufacture. Process engineers are responsible for the design of an IC manufacturing process. They must define a sequence of processing steps, including masking and pattern exposure, implantation, oxidation, and etching, and they must specify and control parameters for each of these steps, including time, temperatures, and implant dosage. Their goal is to design a process which can produce reliably devices with well-defined electrical properties in a manufacturing environment.

The design of the manufacturing process could be carried out by trial and error, monitoring the electrical characteristics of test devices as the parameters of the process are varied. However, not only is such a process time-consuming and expensive, but with the high cost of today's IC manufacturing lines, it is important to design and test the process before it is actually implemented. In addition, while a process is being implemented, it is important that circuit designers have accurate device models so that they can develop their first circuits in parallel with the process development phase. Process simulation is used as a convenient tool in the design and refinement of processing sequences.

Process simulation has been a very active area of research for the past few years and has become an indispensable tool for industry. A key aspect of process simulation is process modeling. An excellent review of this field can be found in [DUT81].

Two approaches have been followed to model accurately the processing of IC devices: the analytical approach and the numerical approach. The trade-offs involved in the selection of one of the approaches are accuracy and compute time. In particular, analytical solutions can be given under simplifying assumptions or from a functional fit from measured data. Hence, these approaches tend to be valid only over a limited range of process-

ing conditions and useful for tuning an existing process more than for a complete design of a new process. On the other hand, the complexity of computation is small and information such as impurity profiles, can be obtained at the expense of function evaluations which do not involve the solution of complicated nonlinear equations. At the other end of the spectrum, numerical techniques can be used to obtain the necessary information by solving set of nonlinear partial differential equations describing the processing steps in detail. As such they can be very accurate, but these computational techniques are time consuming since they involve a double discretization process: in both space and time. FABRICS[STR], developed at Carnegie-Mellon University, is an example of process simulator using analytical models while SUPREM[DUT], developed at Stanford, is an example of a process simulator using the numerical approach. While the process simulators developed at Stanford focus on the oxidation, implantation, and diffusion steps, SAMPLE[NEU], developed at Berkeley, models the photolithographic and etching aspects of the manufacturing process. Simulators currently used in industry are based on numerical techniques. IBM and AT&T Bell Labs have been technical leaders in this field.

In general, IC fabrication processes are affected by random disturbances, such crystal imperfections and dust, which affect greatly the yield of IC circuits. Over the past few years, techniques for designing processes, devices and circuits to maximize yield have become a necessity to provide economically sound products. Unfortunately, accurate numerical techniques such as the ones used by SUPREM cannot be used to predict yield if a statistical characterization of a process is sought since today's computers are not powerful enough. However, programs like FABRICS can be used to provide a statistical characterizations of processes, given the improved speed with which physical information can be generated from processing step information in this type of program.

The ultimate goal of process design is the production of devices with given electrical characteristics. However, process simulation produces as output impurity profiles. The

necessary next step is to map this physical information onto device parameters that describe the electrical behavior of the devices resulting from the process. This step is accomplished by programs called *device simulators*. Approaches to device simulation are similar, from a mathematical point of view, to the ones used in process simulation. Numerical techniques involving the solution of partial differential equations, such as Poisson's Equation and the Continuity Equation, are used by programs such as SEDAN developed at Stanford[DUT] and MINIMOS developed at the University of Vienna[SEL]. Analytical techniques are used by FABRICSII[NAS84], a combined process and device simulator, to obtain device parameters for a variety of technologies and transistor models which are used in circuit simulators such as SPICE2[NAG75].

Recently, attention has been devoted to the use of special-purpose hardware to reduce the cost of numerical process and device simulation. In the past, only one-dimensional effects were modeled by these tools. Recently, two dimensional process and device simulation has been possible. Three-dimensional effects are now been addressed to represent the processing steps in their full complexity. Japanese companies, in particular, are devoting significant resources to this problem.

For the analytical approach to process and device simulation, more attention is being paid to the development of accurate analytical formulae and of optimization techniques which can be used to design a manufacturing process to reduce the time needed to obtain a satisfactory design.

4.3.2. Circuit Simulation and Modeling

As mentioned earlier, circuit simulation was one of the earliest tools applied to the design and verification of ICs[??], since 'bread-boards' prototypes of these circuits could not adequately represent the parasitic or thermal effects necessary for prediction of circuit performance. When accurate circuit models are available, circuit simulators provide precise electrical information, such as frequency response, time-domain waveforms, and sensitivity information, about the circuit under analysis. The majority of circuit simulators currently in use contain models for a wide range of active devices, including bipolar junction transistors, MOSFETs, JFETs, MESFETs, and diodes, and hence are largely independent of technology. For this reason, these programs must employ general algorithms for the solution of the set of coupled, nonlinear, ordinary differential equations which describe the integrated circuit and hence cannot exploit the special characteristics of a particular technology.

The most used general purpose circuit simulator is SPICE, developed at the University of California, Berkeley. This program has been adapted for use in many IC design companies, e.g. ADVICE used at AT&T Bell Labs, TI-SPICE used at Texas Instruments, SLICE developed at Harris. The program ASTAP has also been widely used. ASTAP is based on different algorithms than SPICE and has additional capabilities such as user-defined models and statistical analysis. In addition, commercial versions of SPICE, SCEPTRE, and the ASPEC program are used in industry.

Without models whose accuracy is well matched to the expected accuracy of a simulation, the results of the simulation may not reflect the performance of the circuit under analysis. Recent work on modeling for MOS circuit simulation[??] has focussed on the development of both analytic[??] and semi-empirical[??] models for MOS transistors which predict the characteristics of the devices accurately without requiring large amounts of computer time. With increasingly small geometries on ICs, signal delays and signal degra-

dation caused by interconnect can dominate circuit operation. For this reason, explicit models for interconnect are necessary for accurate simulation and interconnect modelling has returned as an active area of research[??]. The parameters of such models may be provided by the designer interactively or by design programs directly, as described in Section 4.2.2.

Since circuit simulators have been with us for almost twenty years, and because the problems they attempt to solve are very well understood, the core algorithms used in a modern circuit simulator are generally quite robust. However, as a consequence of their long history, most circuit simulators are batch-oriented programs and the input to the program consists of a textual description of the transistors and their interconnections. Nowadays, using a CAD workstation, an interactive graphics editor is often used to capture the schematic diagram and provide simulator input.

Circuit simulation techniques can provide accurate waveform analysis for circuits of building-block complexity. However, as circuit size increases the time and memory requirements of a circuit simulation become prohibitive. On an IBM 370/168 computer, the average cost of a SPICE[16] analysis is 6ms/device/clock/timepoint. For a 10,000 device circuit, with 3 clocks and for an analysis of 10us at 1ns steps, the computation time would be in excess of 20 computer days! Nevertheless, the success of circuit simulation in design evaluation has been such that designers wish to continue to simulate large circuits at the level of accuracy provided by this type of program.

By applying node tearing techniques[82,83] to the interface between cells in the circuit, inactive cells can be bypassed during the equation solution phase. However, these techniques alone provided less than an order of magnitude speed improvement. This is not sufficient improvement in performance to permit cost effective device-level analysis of VLSI circuits.

If simulation algorithms are tailored to specific technologies or applications substantial speed improvements can be achieved. Many components of digital MOS or I^2L circuits can be considered unilateral in nature. This characteristic, as well as the facts that these families are saturating and hence accumulated voltage errors are lost at the extremes of signal swing, and that large digital circuits are relatively inactive at the gate level, are exploited in *timing simulation*. Timing simulators[84-86] can improve simulation speed by up to two orders of magnitude while maintaining acceptable waveform accuracy. These savings are achieved by using node decoupling techniques in conjunction with simplified table look-up models for nonlinear devices.

Where a library of cells is used during the design, or when a group of transistors is used to implement a common function, such as a cell or logic gate, it is often possible to exploit the known structure of the circuit and use a simplified representation which maintains the essential characteristics of the cell at reduced computational expense. Such a reduced representation is called a *macromodel* [87-89] and macromodels are used in both circuit and timing-level analysis.

While timing simulators are fast, they may be inaccurate for circuits containing tight feedback loops and large floating elements. Designers have often applied these simulators to problems which were not well suited for this type of analysis and have obtained incorrect answers. For this reason, converged relaxation-based circuit simulators[LEL82, NEW83] were developed. These programs evolved from the basic ideas of timing simulators guarantee an analysis as accurate as the one provided by standard circuit simulators such as SPICE.

Two basic algorithms have been used in these simulators:

(1) *Waveform Relaxation*, (WR) where the system of Ordinary Differential Equations (ODEs) representing the circuit is solved by a relaxation process at the differential equation level, i.e., the variables which are relaxed are waveforms. This approach is

used in the RELAX[LEL82,WHI84], TOGGLE[??] and SWAN[DUL85] programs.

(2) *Iterated Timing Analysis* (ITA) where the system of differential equations is first discretized and the resulting nonlinear algebraic system of equations is solved by the SOR-Newton iteration. This approach is used in the used in the SPLICE[KLE82,KLE84] and MOTIS[CHE84] programs.

Both of these techniques exploit the unilateral nature of MOS devices, the inactivity of the circuit (*latency*) and the fact that node voltages and branch currents of the circuit change in time at different, sometimes very different, rates (*multirate* behavior). Savings in running time of up to two orders of magnitude have been obtained over standard circuit simulation programs such as SPICE2. However, for circuits that contain tightly-coupled subcircuits or where many parasitic components are involved, the relaxation-based approaches alone may not perform as well as standard circuit simulators.

Under these conditions, relaxation must be applied with great care to maintain the speed advantage over standard techniques. The key idea here is to solve the tightly coupled subcircuits with standard methods while the loosely coupled connections are dealt with relaxation[NEW79]. Most relaxation-based simulators in use today provide this capability. Automatic partitioning algorithms have been developed to partition large scale circuits into collections of tightly-coupled subcircuits [SAK85, WHI85].

These techniques alone cannot provide the speed that is needed for the detailed electrical analysis of VLSI circuits. Special-purpose hardware and multi-processor systems are now being used to provide dramatic speed improvement for circuit simulation. The direct methods, such as the ones used by standard circuit simulators, can certainly be parallelized but this operation is not straight-forward at either the algorithmic or the implementation levels due to the sparse, irregular nature of the circuit matricies. Relaxation-based simulation algorithms are much easier to parallelize. For this reason, the first results published in this area are related to iterated timing analysis algorithms

[DEU84, DEU85] or to waveform-relaxation [WHI85a, SAN85, WHI85b].

4.3.3. Logic and Switch Simulation

When the complexity of an integrated circuit design reaches the point where electrical analysis is no longer cost effective, logic simulation may be used. Rather than dealing with voltages and currents at signal nodes discrete logic *states* are used. Only simple Boolean operations are required to obtain the output state value of a logic gate and these are generally the most efficient operations available on a digital computer. Rather than modeling the circuit at the individual transistor level, in a logic simulator transistors are grouped into logic *gates* wherever possible and a *gate-level* model is used. As in modern, relaxation-based simulators, asynchronous logic simulators exploit the inactivity in the circuit to improve speed by using *event-driven* and *selective-trace* techniques. The term event-driven refers to the fact that only logic gates whose input values change are processed and the term selective-trace refers to the technique used to find the logic gates whose inputs have changed. Rather than checking every gate at every timepoint to determine if its inputs have changed — an expensive process if most of the gates are not changing — when an output changes, a table containing pointers to the gates to which this output is connected is used to schedule the fanout gates for processing. As a result, the program selectively traces paths of activity through the circuit. With selective trace analysis and the above simplifications, asynchronous logic simulators are typically 10 to 100 times faster than the most efficient forms of electrical analysis.

However, the major objective of simulation is accuracy and simulators must accurately predict the behavior, both normal and abnormal, of the physical circuits they model. It is clear that the transition from the continuous electrical domain to the discrete logic domain will result in the loss of some circuit information. It is important, therefore, that the circuit design methodology allow such an hierarchical simplification or logic simulators cannot be used effectively. In most cases, once a subcircuit of the design has

been verified in detail at the electrical level, a simplified gate-level model can be used for logic simulation. However, it still may be necessary to analyze critical paths in the network at the detailed electrical level.

The tradeoff between the accuracy of logic simulation, and hence the amount of information it can produce about circuit operation, and the computer time required to perform the simulations, is very important. The number of logic states used in the simulator and their meaning, the logic delay models used, even the type of scheduling algorithms employed, are determined by the technology in which the circuits are to be implemented, and its associated circuit characteristics, as well as the particular design methodology being used.

It is this wide variety of factors that has resulted in the development of such a large number of logic simulators, almost every one addressing a different set of tradeoffs. Logic simulators have been in use for the design of digital hardware since the early 1950s [33], and it is impossible to address all aspects of simulator development in this brief review.

Rather than using event-driven techniques, many of the early logic simulators were *compiled simulators* [32], where the logic circuit was described in a programming language which was compiled directly to machine code and executed. Although this approach provides a very efficient mode of simulation, no compiled simulators accurately model asynchronous circuits. Compiled techniques are used, however, for higher-level Register Transfer Level (RTL) simulation and, of course, the bottom-level models in logic simulators are generally compiled. Recently, compiled simulators have been developed for the switch-level simulation of clocked, synchronous MOS circuits where the circuit is analyzed and switches that form distinct, combinational blocks are clustered as subcircuits and a corresponding program fragment is generated. These fragments are compiled and form the scheduled blocks of an event-driven simulator. An equivalent speedup can be achieved by using a gate-extraction program which recognizes specific patterns of switches

and replaces then with an equivalent gate-level representation[hpref].

The earliest use of logic simulation was for the verification of combinational logic. Since the logic was assumed to have zero delay and logic gates were assumed to implement ideal Boolean operations such as AND, OR and COMPLEMENT, only two states were required: a state representing *true* (logic 1) and a state representing *false* (logic 0) [34].

As technologies have become more complex and the logic designer continues to exploit the features of a particular technology, such as tri-state outputs, additional states have been added to logic simulation. Early simulators used "unit-delay" models for gates, where the rise and fall times of a gate were assumed to be the same and the same as for all other gates. While this was a reasonable approximation for TTL SSI circuits, with MSI came the need for assignable delays for different gate types. With MOS design separately-assignable rise and fall delays were added due to the disparity in rise and fall delay present in NMOS circuits. In addition, the use of wired logic where even though two switches may try to assert different logic vales onto a net the "stronger" switch should win, required additional *strength* states to be added. By adding an "unknown" state to the simulator, efficient detection of all single-input circuit hazards can also be accomplished[eich]. Modern MOS-oriented logic simulators provide at least nine static states for describing logic.

Even the nine-state simulator does not adequately model the interaction between transfer gates of different geometry, or the effect of parasitic capacitance variations on the charge sharing across a transfer gate. These effects could be represented by a finer resolution (adding more states) or to accurately model this behavior timing simulation can be used. The bidirectional nature of transfer gates can be approximated by noting all the forcing states at the nodes of an arbitrary tree of transfer gates and tracing their fanouts through the tree, setting all affected nodes to the appropriate soft state unless a conflict is detected.

4.3.4. Register and Behavior

As mentioned in Section 3, there are a number of levels of description above the logic gate level. For each of these levels there are associated simulators. While different description languages are described as RTL, PMS, or Behavioral, it is often the case that the distinction is blurred in actual use. While the purpose of an RTL description is to describe a register-level implementation of a system, RTL descriptions are often used as input to synthesis systems where the structural information they contain may be ignored. On the other hand, the program structure in a behavioral or algorithmic description is often used as an initial hardware structure for implementation.

As the design representation becomes increasingly abstract, it also tends to become increasingly domain-specific. For example, while most designers can agree on what an AND gate is, at the system level a signal processing engineer will refer to a "sample time," the microcomputer designer talks of "4-phase clocks", and the data-driven system designer talks about "self-timed" modules. If a language selects any of these notions as its abstract representation of time, it will often not be used in the other application areas. The only way, therefore, to build a general behavioral language and simulator is to reduce the domain-specific notions to their lowest common denominator and to permit individual users to built libraries of domain-specific objects and operators. As a result, most behavioral-level simulators are implemented in existing concurrent programming languages, such as Simula[simula], Modula-2[mod2], or concurrent Pascal[adlib]. The major exceptions are the behavioral levels of mixed-level simulators, described in the following section. Since in the mixed simulation environment, the behavioral descriptions must coexist with lower-level descriptions, the behavioral language often inherits many of the characteristics of the lower level (e.g. logic gate level) description formats.

Register Transfer Level simulators may be classified by the manner in which they deal with time and the way in which they order the evaluation of blocks. In some simu-

lators, all assignments correspond to clocked, register transfers. All logic between assignments is combinational and thus can be compiled on a sequential machine after a suitable static ordering has been determined. Such an ordering can be found using breadth-first search of the data-flow graph that represents the logic expressions. Since the logic is combinational, the graph must be cycle-free. This approach leads to a very fast implementation but cannot handle hazard detection or timing within a clock cycle and can only represent synchronous systems.

The other approach is to implement the simulator much more like a logic simulator. Use a dynamic ordering, based on the next-event approach[szgenda] (event-driven selective trace), and schedule code modules. The modules may be those specified by the user (code block or procedure boundaries) or may be extracted from the description between explicit delay assignment statements. If the system supports asynchronous design, specific delay constructs must be provided and these are used to determine locations for scheduling

Register-transfer simulators are over an order od magnitude faster than gate-level simulators, for the same circuit, since they deal with fewer blocks and their model of time and signals is typically a lot coarser than in the logic case.

4.3.5. Mixed-Level

For the analysis of most large IC circuits neither electrical nor logic analysis is alone sufficient. The detailed waveform information of an electrical analysis is required for some parts of the circuit but an electrical analysis of the entire network would require an excessive amount of computer resources. A logic-level analysis is often sufficient for parts of the circuit and can be performed much more efficiently than an electrical analysis. Although timing analysis is generally much more efficient than circuit analysis there are circuits which cannot be simulated accurately using timing analysis, such as circuit blocks containing strong bilateral coupling between nodes. For these blocks, a detailed circuit

analysis may be required.

A number of *mixed level* or *mixed mode* simulators have been developed which combine analyses at more than one conceptual level. The SABLE system developed at Stanford[39], the DECSIM system developed at Digital Equipment[], the MICROSIM system developed at Intel[??], the HILO system[??], and the SILOS system[JEN] are among the most successful examples of mixed-level simulators that address the high-end of the design and allow behavioral, RTL, and gate-level descriptions to be combined. Both the Diana program [40] and the SPLICE program [41] allow concurrent circuit, timing and logic analyses of different parts of the circuit. The MOTIS program [9] has also been extended to combine timing, logic, and RTL level analyses [42]. Recently, process, device and circuit simulation have been combined into MEDUSA, a mixed level simulator developed at Aachen[ENG] and process and device simulation have been combined into FABRICS2[STR85].

The most difficult mixed-level simulator is the one that involves electrical and logic levels since the representations of the signals is totally different thus making the combined analysis complicated. For this reason, we focus on this type of mixed-level simulation.

One approach to the design of a mixed-level simulator is to combine existing circuit, logic, and RTL simulators via data *pipes* such that the three forms of analysis can be performed concurrently and may pass values of circuit variables, with the appropriate transformations, between one another via the data pipes which connect them[hughes]. While this approach is useful where the circuit contains large blocks of elements to be simulated at each level, such an approach would result in a very inefficient simulation if the the different levels of modelling and analysis were coupled tightly in the circuit.

In a table-driven simulator[NEW81, KLE84], it is the structure of the data tables that makes for efficient simulation. Hence a common data format has been determined [NEW81] for all types of circuit elements and circuit nodes so that a single event

scheduler can be used to process them all.

Timing analysis and circuit analysis may be coupled directly since they both use voltage and impedance to model the conditions at a node and hence an additional interface is not required. Discrete logic simulation does require an interface to and from the electrical analysis. This interface may be achieved by implicit signal *coercions* [NEW81]. That is, whenever an electrical element is connected to a logic node, and *vice versa*, an automatic signal transformation is implied. Alternately, special circuit elements may be used to perform the transformation[DEM8140]. In either case, *thresholding* may be used to convert voltage and impedance to logic levels while *logic-electrical conversion* may be used for the reverse transformation.

The most successful mixed-level simulators have been the ones addressing various levels of logic simulation, from gate-level all the way up to functional level. Mixed-level simulators involving electrical analysis have been used to a lesser extent by the designer community. One of the still unresolved problems in this type of mixed-level simulation is the partitioning of the circuit into the various levels to achieve the degree of efficiency and accuracy desired by the designer. In most cases, the designer is called upon this difficult decision. Some automatic partitioning approaches have been developed and are still under study to cope with this problem.

4.4. Performance Verification

Digital circuit design depends critically on the delay of the signals. For asynchronous designs, if the timing of the signals is not carefully considered, incorrect results may be obtained from an otherwise functionally correct design. For synchronous designs, the longest delay among the signals from primary inputs, or latches, to latches, or primary outputs, the *critical path*, determines the period of the clock and hence the speed of the circuit.

To optimize the performance of a circuit, critical paths have to be identified and minimized. In addition, paths which are non critical may be unnecessarily fast and consume extra power to no avail. This optimization is becoming a more and more difficult task as circuit size and complexity increase, but more and more necessary because of the competition to build faster and faster circuits. Simulation can be used, and has been used, to identify critical paths and to optimize circuit performance. However, detecting critical paths may involve the simulation of several thousands transistors for large ICs. Furthermore, pathological conditions may not be detected by simulation, unless particular inputs are fed. *Timing verification* is a technique which aims at the determination of critical paths without performing simulation. This technique has played a very important role in the design of digital integrated circuit, in particular for bipolar gate-array technologies. The first timing verifiers were built around 1973 by IBM and other computer companies for the design of large high-performance computers (see [HIT82] for a description of timing analysis techniques, their history and their relevance to computer design).

Other applications of timing verification have recently come to the attention of the designers' community. In particular, Ousterhout at Berkeley [OUS85] and Youppi at Stanford [YOU] developed timing verifiers for nMOS and CMOS technologies where design constructs are not limited to logic gates.

Most of the CAD vendors have developed timing verifiers with various degrees of complications and power. Both gate-level and switch-level timing verification is available.

In general, timing verification programs are partitioned into two parts: a path-analysis section and a delay modeler. Most timing verifiers represent a circuit by a node-signal flow graph. The path-analysis section extracts part of the circuit systematically using the signal flow graph and transfers it to a delay modeler that computes the delay along the path corresponding to the part extracted. The nodes along the path are then labeled with the worst case delay found so far. Note that unlike simulation, timing verification is *value-independent*. This means that, for example, if a changing signal arrives to a NAND gate, its effect is always propagated to the output node, regardless of the signal states at the other input terminals. In simulation, the changing signal propagates to the output of the NAND gate, if and only if the signal states at the other input terminals support the propagation.

There are two basic approaches to timing verification, *path enumeration* and *critical-path analysis*.

The basic difference between the two methods is that in path enumeration Glauert Kirk all possible paths in the circuit are checked, while in critical-path analysis, the search is pruned and only the slowest paths are detected with techniques borrowed from the PERT critical path algorithm Path enumeration is conceptually simple, but it may suffer from rather long CPU-times due to the potentially large number of possible paths Even though the complexity of implementation of critical path methods is higher, most of the programs in use today are based on this technique

Both strength and weakness of timing verification techniques come from value-independence. Since not all input combinations, whose number is exponential in the number of the inputs, are generated, timing verification can be much faster than simulation. However, since timing verification ignores specific signal values, it may report critical paths which can never occur under real operating conditions. These paths are called

false paths. False paths tend to hide the real critical paths in the circuits under test. In this case, a mechanism called *case analysis*, [MCW80] has been used to exclude certain paths that cannot occur by fixing the values of certain inputs. However, it must be used with caution since by fixing too many input values, not only false paths but also *bona fide* critical paths may be eliminated.

A very important part of timing verification is the delay estimation method used by the delay modeler. For bipolar gate-arrays, the delay of a component is a well-characterized quantity that can be assigned to the component independently on load conditions and input waveforms. In the case of MOS circuits, the delay depends critically on input waveform shapes, loading conditions and size and type of transistors. Most MOS timing verifiers represent the MOS transistor with an ideal switch with a resistance in series and represent the capacitance of the transistors as well as the parasitic capacitance along a wire with a capacitor from every node in the circuit to ground. Then the delay is computed using approximate formulae based on the values of the resistors and capacitors. This approach is very efficient in terms of CPU-time, but it has several weaknesses. For example, it assumes that there is only one direct path from a reference node (power supply or ground) to the signal-nodes of the circuit. For example, in the case of an nMOS inverter, if the driver is on, only the driver is considered for delay time estimates, and the load is ignored. While some of the programs based on this "RC" approach (e.g. Crystal), incorporate information about input waveform shape and load condition in order to obtain more accurate delay estimates, they use the ratio approach first suggested by Pilling and Skalnik Pilling Skalnik The ratio approach improves the accuracy of the delay estimates significantly with small amount of work for most circuits, but it may still result in large timing errors.

Recently, a new approach to modeling the path delay, called *Elogic* [KIM84] has been proposed to solve some of the inaccuracy problems of timing verifiers. Essentially, the Elogic approach simulates the path extracted by the path analyser with a model of the com-

ponents of the circuit somewhere in between the models used in logic simulation and circuit simulation. Because of the use of a more accurate model of the components as well as a more accurate representation of the subcircuit responsible for the delay at a node, the delay estimation is much more accurate. In addition, since there are several levels of accuracy in the Elogic models, a trade-off between speed and accuracy is offered to the user. Elogic models have been recently added to Crystal and tested on a number of circuits. The results show that a reduction of an order of magnitude in the delay error can be achieved at the expense of a thirty fold running time increase.

4.5. Formal Verification

In general, the functional verification step is carried out by simulating the design with a set of input patterns which cannot *guarantee* that the design is correct, i.e., that the transformation from one level of the design hierarchy to the next has not changed its functionality. *Formal verification* techniques are input-independent and are designed to guarantee functional equivalence between two representations of the design at different levels.

Most of the formal verification techniques deal with behavioral, structural and logic level representations. These techniques have great potential for producing correct-the-first-time designs. In addition, verification is important for technology remapping, i.e., for the transfer of a logic design from a technology (for example bipolar ECL gate arrays) to another (for example CMOS gate arrays). In this case, the functionality of the new implementation has to be checked against the functionality of the former. Despite the importance of formal verification, only a few of the techniques developed in the past have been applied to industrial designs due to their complexity and computational requirements.

Most of the formal verification techniques proposed recently can be classified into three major categories: *formal software verification* techniques, *semi-exhaustive logic*

simulation and *logic comparison*.

Formal software verification techniques can be used because at the highest levels of the design hierarchy, hardware descriptions are similar to computer programs written in a programming language. In particular, the inductive assertions approach and the symbolic simulation approach have been proposed for design verification. The inductive assertions method requires a set of assertions to be made at the input, the output and each internal loop of the high level description of the design. These assertions specify relationships between variables in the description and represent a formal "definition" of the correctness of the design. In fact, if the design is correct, then all the assertions must be verified, in particular the output assertions. The key problem in this approach is to state the assertions correctly. At this moment, the inductive assertions approach has mostly theoretical interest.

Symbolic simulation replaces the calculation of the set of primary outputs given a set of primary inputs with the calculation of the logic expression computed by the design, i.e., a formula for each of the outputs where the variables are the primary inputs. Once this calculation has been performed, it still remains to verify that the formula is indeed the correct one. A formal relation between primary inputs and primary outputs is sometimes available and, in this case, the two formulae have to be verified one against the other for equivalence. This computation can be carried out by means of rule-based systems such as MAXIMA[]. There are two major difficulties in this approach: the first is in the actual calculation of the formula implemented by a specific collections of modules that can be very expensive in terms of computer time, the other is the equivalence calculation, an NP-complete problem.

Of course, if all the input patterns are exhaustively fed into the two descriptions of the design, then the two representations are equivalent if the outputs corresponding to the same inputs are equal. However, for large circuits this approach is out of the question.

because the set of all possible inputs is 2^N where N is the number of the inputs. Quasi-exhaustive simulation tries to limit the number of input patterns to apply to verify equivalence and has been used successfully in some designs. In particular, the mixed level simulator MIXS developed by NEC [] has a formal verification mode, where the input patterns are derived so that the design is guaranteed to be equivalent to a set of functional specifications. This approach focussed on the reduction of the number of input patterns that have to be generated and simulated to guarantee the correctness of the design. Several heuristics have been proposed, but their use may lead to an incorrect answer.

The most successful formal verification techniques are based on logic comparison. These techniques have been in use at IBM for many years[SMI82]. They are applied to designs that satisfy a set of restrictive assumptions. The main assumption is that a one-to-one correspondence between the memory elements such as flip-flops of the design at the two levels of interest has been established. Then the problem of logic verification is reduced to the one of verifying the equivalence of combinational logic. This assumption is verified when Level-Scan Sensitive Design (LSSD) or scan-path techniques are used in the design to make the task of testing the design easier.

Other approaches involve the automatic translation of the high-level description of the design into a gate-level description, that can be then compareed with the "real" gate-level design. This automatic translation can be done quickly since the quality of the synthesized logic is of no concern. For example, the Fujitsu verification system uses DDL[] to describe both the high-level behavioral specification and the structural level representation. The DDL representation is then mapped into a logic design that is compared with the actual design which may have been generated automatically with an effective synthesis tool or manually.

This problem is very similar to the testing problem, and in fact can be formulated as a redundancy identification problem of a circuit obtained from the two circuits to com-

pare by forcing the primary inputs to be the same and by tying the primary outputs to the input of a comparator (an XOR gate). If the designs implement the same logic function, then the output of the comparator is always zero. The verification problem is equivalent to proving that stuck-at-zero fault at the output of the XOR gate is redundant. This problem can be easily shown to be NP-complete. Experience gained in coping with testing problem is instructive in this case. However, in the testing problem, the more common situation is that the fault under test is not redundant and we want to find a test for it as soon as possible, while in logic verification we expect the fault to be redundant and we wish to come to that conclusion as rapidly as possible. This implies that efficient algorithms for testing may not be efficient for logic verification.

The first logic comparison approach was developed by J.P. Roth of IBM who proposed to use the back-ward justification part of the D-algorithm to prove equivalence of the two designs. The Differential Boolean Analyzer and its variations were used in the verification of the IBM 3081 computer design, a most impressive accomplishment since the hardware modules being verified have approximately 30,000 gates each. This approach uses the iterative application of the Shannon expansion[SHA38] theorem to specify the set of input patterns that have to be considered for a complete verification. The application of the expansion is done so that a minimal set of vectors has to be identified to carry out the verification.

While some of the results can be used to verify formally a design, we believe that more research is still needed to improve the speed and the domain of applicability of the formal verification techniques.

5. SYNTHESIS

5.1. Introduction

As pointed out in the Introduction, *synthesis* is a crucial component of future CAD systems. The competitive edge of IC design will most probably come from the use of effective synthesis tools.

A complete synthesis system should generate layout masks from a high-level *algorithmic*, *behavioral* or *functional* description of a VLSI system, a description of the target technology and a description of the constraints and cost functions. The design should be completed in reasonable time and with the quality a human designer could obtain.

Very few design-aids are available to assist the VLSI designer at the algorithmic level. At this level, the designer describes the system by specifying its operations or functions without necessarily giving implementation details such as the "hardware" components needed to implement the system. Design at this level involves the translation of a required algorithmic-level specification into an *architectural* or *register-transfer-level* implementation. The architectural representation of the design includes components such as registers, memories, processors, which specify the high level implementation of the system.

Once the functional partitioning of the design is completed, estimates of the layout size, power-supply requirements, and speed of the high-level circuit blocks used to implement the various sub-functions are required. A chip-plan must also be constructed to determine the relative placement of these building blocks. This chip plan is then further refined as the design proceeds. These tasks are often performed manually, perhaps with the help of the computer to perform book-keeping tasks such as the storage of interconnection data.

Silicon compilers have been proposed to carry out the entire synthesis process. Since the task is so complex, early silicon compilers assumed that a target technology and a floor-plan were chosen by human designers. In this case, the difficult steps of linking the

high level synthesis task to the layout problem was resolved by eliminating computer intervention. Among the most important contributions of the early research on silicon compilers is the development of *procedural design languages*. These languages are used to write programs which, when executed, generate in a flexible and possibly technology independent way the layout of entire chips and/or of leaf cells, i.e., the basic low level cells such as nand gates, nor gates, inverters, register cells. The work by D. Johannsen at Caltech[JOH79] and the work on DPL[BAT80] at MIT was an example of such important contributions. Procedural design languages can be used effectively to generate the layout of regular structures such as ROMs, RAMs, PLAs and data paths. In particular, the use of these languages eases the construction of parametrized and technology independent *module generators*, i.e., of computer programs that generate the layout of a block such as a ROM, RAM or PLA, given a functional representation, such as a truth table.

Working designs have been produced with silicon compilers, but the quality of the design has always been a problem. While for a restricted class of designs, such as Digital Signal Processors (DSP), the use of a fixed floor-plan has been successful, (e.g. the LAGER silicon compiler developed by R. Brodersen at Berkeley[]), its use for less constrained applications results in inefficient utilization of area and poor performance. In addition, the structure of the control logic is often too rigid and not optimized, thus yielding a slow and large chip. J. Fox[] has illustrated the pitfalls of an existing silicon compiler, MacPitts[], by comparing a design for a telecommunications chip generated by a silicon compiler with a design obtained with the use of a standard-cell place and route system.

The present trend is to break the synthesis process into stages, and to use tools that optimize real estate and/or performance to go from one stage to the next. At first, attention has been paid to the optimal generation of regular arrays such as ROMs, RAMs and PLAs. For example, module generators have been built by VLSI Technologies Inc. and Silicon Compilers Inc. using a procedural design language. Simple routing techniques were

also offered to connect the modules generated by these tools. These two companies were the first to introduce the concept of silicon compilation and procedural design languages in industry. While these concepts have now gained considerable attention in the industrial community and several other companies are offering procedural design languages (e.g. Silicon Design Laboratories and SDA) and module generators (e.g. Silicon Compilers Inc., SDA and Seattle Silicon), a few years ago when these companies were founded traditional designers expressed a great deal of resistance towards these new design techniques.

The SILC silicon compiler under development by J. Fox at GTE and the Yorktown Silicon Compiler (YSC) being developed at the T.J. Watson Research Center by R. Brayton[BRA84], are two examples of systems where layout optimization and efficient logic synthesis are introduced. While SILC addresses the problem of translating an algorithmic description of the system to be designed into an architectural description, YSC starts with an architectural description of the design leaving the task of determining the architecture of the chip to the human designer. The Design Automation Assistant under development at AT&T Bell Laboratories[KOW85] is the most recent entry in the automatic synthesis arena. This system is based on the work done at Carnegie-Mellon University for high-level synthesis and on the work done at Stanford for layout. One of the most interesting aspect of this system is the use of a knowledge-based expert system to carry out the translation of the behavioral level description into a register transfer level description and to generate an optimized floor-plan. Research work on silicon compilation is also carried out at the University of Illinois with the ARSENIC silicon compiler developed by D. Gaijski[].

The procedural design aspects of early silicon compilers has been neglected for some time. Recently Silicon Design Laboratories (SDL), founded by the developers of the PLEX system[] for automatic synthesis of micro-processor-based designs, introduced a procedural design language for IC design which is available commercially[]. We believe that pro-

cedural design systems and knowledge-based expert systems are crucial for the synthesis systems of tomorrow.

In this section we review the three basic components of a synthesis system:

1- Physical synthesis or layout, including floor-planning, partitioning, placement, routing and compaction;

2- Logic synthesis, including combinational logic, sequential logic and algorithmic or behavioral synthesis.

3- Procedural design and module generation.

5.2. Physical Synthesis

The layout of integrated circuits consists of the placement of the devices (cells) composing the design in a two dimensional finite space and of the interconnection of the pins of these devices according to the schematic of the circuit to be implemented. The goal of this process is to complete the placement and interconnection of the design in the smallest possible area satisfying a set of design constraints, such as the ones posed on the position and size of the devices, a set of technological constraints, such as the ones posed by design rules and levels of interconnects, and a set of performance constraints, such as the ones posed by the timing of the logic to be implemented. This optimization problem is very complex, even simplified versions of it are NP-complete,[1] and given the number of modules to be laid out, has to be decomposed into smaller sub-problems to be tractable.

The layout problem is traditionally subdivided into several stages that will be reviewed in the next sections. While these stages are often common to the various design styles such as gate-arrays, standard cells and macro cells, their complexity may differ.

[1] An NP-complete problem belongs to a class of difficult combinatorial optimization problems such as the traveling salesman problem and the coloring problem for which an algorithm whose complexity is bound by a polynomial in the size of the input is not know and is unlikely to be found.

Automatic layout systems were introduced first for gate-arrays and standard-cells, since the layout problems associated with these design styles are in general simpler than the ones associated with the macro-cell design style. Good gate-array systems developed both in house and by vendors companies are now widely available. The Engineering Design System of IBM was among the first complete layout systems to be developed for gate-arrays in the late 1960s, even though the first papers describing the algorithms used in the system were published in the early 1970s. In Japan, the first gate-array place and route system was developed by Oki and reported in 1974. Silicon foundries such as LSI logic offer their customers place and route systems optimized for their gate-array families. Mentor/CADI, Tektronix/VR Systems and Daisy have good place and route systems. Daisy has also a gate-array placement hardware accelerator which implements the simulated annealing algorithm.

The first publications describing the concept of gate-arrays date back to 1964, with the papers by Rex Rice of Fairchild[RIC64] and of E. Sack of Westinghouse[SAC65]. For a while, gate-arrays completely dominated the semi-custom market, in part because of the wide availability of CAD tools for their design. Now standard cells are capturing the attention of a large user community. Interestingly, the concepts of standard cells and of gate-arrays were developed at about the same time. However, automatic placement and routing tools for standard cells were developed earlier than the ones for gate-arrays. Philco Ford was the first company to do work in the area of automatic place and route of standard cells around 1964. This work was picked up and extended by S. Daram and his group at Fairchild in 1965 with the MOSAIC system. The cells were arranged in rows and pins appeared on both sides of the cell. The routing was accomplished with a precursor of the channel routing algorithm which recently became the most popular routing techniques. Fairchild also developed an automatic rubylith cutter for this structure. RCA Camden developed in 1966 the PRF (Placement, Routing and Folding) system for placement and routing. In this system cells were placed back-to-back and hence had pins on

one side only. This system was then taken to the National Security Agency and improved. In the late 1960s RCA extended this approach to PR2D where a two-dimensional placement algorithm was developed. In the 1970-1972 time-frame the MP2D system was developed at RCA. This system is still widely used. Bell Laboratories developed a poly-cell (a synonym for standard-cell) place and route system called LTX in 1973. The successor of this system, LTX2 was extended to gate-arrays and to limited version of macro-cell place and route.

The growing interest in the standard cell design style has prompted a number of companies to develop new place and route systems.

Macro cell placement and routing is particularly difficult. Japanese companies are the clear leader in this area. NTT, NEC, Hitachi, Sharp and Sony have all good working systems. In Europe, Siemens has been among the first companies to develop, on an experimental basis, macro-cell tools. However, they are not in production use. The GAELIC/COMPEDA system was developed by the University of Edinburgh and made available by a commercial company, but it has not found wide use. In the US, some companies such as Hughes aircraft have recently developed experimental systems for the VHSIC (Very High Speed Integrated Circuits) program of the Department of Defense. The CICLOPS system first developed by Preas and VanCleemput at Stanford has been improved at SANDIA labs. However, to the best of our knowledge, no complete system is currently in production use. Because of the importance that macro-cell system are bound to have in the future, we expect more companies to develop such systems.

In the academic world, Japanese and US Universities have the lead in the development of algorithms and programs for placement and routing. Recently, Universities have placed emphasis on the development of complete place and route system for a variety of design styles. The PI system[] for macro cell place and route from MIT, the BBL (Berkeley Building-block Layout) system[CHE83] for macro-cell place and route, the BAGEL sys-

tem[] for gate-array palce and route, and the ThunderBird, standard cell system, all developed at the University of California, Berkeley, and the hierarchical layout system developed at the Osaka University are a few examples. The distinguishing feature of these systems is the use of new and experimental algorithms.

Most of the work done on layout has been concentrated on optimizing area. Recently, performances, in particular speed, have become a major concern. Coupling timing analysis with placement and routing has been proposed to influence the layout process of gate-arrays with speed considerations[BUR84]. We expect this problem will receive more attention in the future.

In the next sections, we will review the various stages of the layout process: floor-planning, partitioning, placement, routing, and compaction. Since the literature on layout methods is huge, we will limit ourselves to the basic techniques, pointing to the relevant papers when appropriate.

5.2.1. Floor-planning

Floor-planning is the first stage in the layout of VLSI circuits. In this stage, the relative positions of the modules to be laid out are determined. Timing, power and area estimations are the factors guiding the relative placement. Floor-planning can be used to verify the feasibility of integrating a design onto a chip without performing the detailed layout and design of all the blocks and functions. This stage is typical of the less constrained design styles, e.g. macro-cell.

At this stage, the aspect ratio of some of the modules may still be unconstrained. For example, if the control logic is implemented with standard cells, then the number of rows used for the modules is not necessarily fixed. Many rows will produce a block that is long and skinny, few rows will produce a block that is short and fat. Different aspect ratios correspond to better packing of the modules in the available area. Figure 5.2.3.1 shows a

particular example of a chip with blocks of fixed and variable size. As other examples, folding and partitioning of a PLA can be used to modify the aspect ratio of the module, or the number of bits used for row and column decoding in a RAM or ROM module can also modify their aspect ratio.

Another degree of freedom is the position of the pins of some of the modules. When standard cells are used for control logic, the signals may leave the block on any point of its periphery. According to their positions, the routing area may be minimized.

Because of the many degrees of freedom and of the uncertainties, the optimization is quite difficult. Very few tools have been developed for this task. Some such as the CAF program[] are interactive and provide estimates for wiring length, channel congestion, area utilization, timing and power dissipation. Others such as the CHAMP program developed by NTT[] and the SPIDER program developed at Honeywell[] offer automatic relative placement and aspect ratios selection. Note that these tools are not generally available; there is no major commercial vendor offering such an interactive system.

Recently, the introduction of simulated annealing algorithms (see window in this page) has made it possible to develop algorithms where the optimization can be carried out with all the degrees of freedom mentioned above. A system at the IBM T.J. Watson Research Center[OTT84] and the TimberWolf package developed at Berkeley[SEC85] use the simulated annealing algorithm to produce a floor-plan that not only gives the relative positions of the modules, but also aspect ratios and pin positions.

Other activities in this area are carried out at Carnegie Mellon University and at GE Research Center in Schenectady. Because of the large number of degrees of freedom in the optimization problem and of the many criteria to be followed, there has been an interest in applying Artificial Intelligence techniques to floor-planning[KOW85]. No working knowledge-based systems for floor-planning are available yet, but they are definitely an interesting and fruitful research area.

5.2.2. Placement and Partitioning

The placement problem involves the assignment of specific locations to building blocks of the layout. This includes the assignment of logic gates within a gate-array[67], the placement of cells in a standard cell layout[68,69] or the placement of macro cells [71,72]. While a considerable amount of theoretical work has been done in this area [72,73], the most successful approaches involve the use of simple heuristics. In these approaches, either total interconnect length, estimated or exact routing requirements are used by the placement algorithms as an indication of the quality of the placement.

As for partitioning, placement algorithms may be subdivided into two basic categories: constructive placement and iterative improvement. Constructive placement algorithms build a placement from initial data such as the size and the type of the cells to be placed and the net-list. Iterative improvement algorithms start with a given initial placement, which can be given by the user, generated randomly or obtained by constructive placement algorithms, and modify the layout to improve its quality. Interestingly, most of the placement algorithms can be used in the basic design styles, gate-array, standard cell and macro cell, with minor variations.

The most well-known heuristics are clustering[], force-directed [], pairwise interchange[], and min-cut []techniques. In all cases the placement problem is represented by modules with a given size and a connectivity matrix $C = [c_{ij}]$ where c_{ij} represents the nets connecting module i to module j with appropriate weights to model the relative importance of the interconnections.

Clustering adjoins incrementally modules to a subset of modules already placed. The modules are adjoined according to their size and connectivity. Critical factors for this algorithm are the selection of the seed, (i.e. of the first module to place), the selection of the next module to be placed and the position of the module with respect the modules already placed. A number of papers have been published describing a variety of schemes for each

of the factors above. Note that once the position of a module has been assigned, it is not changed during the remaining part of the algorithm. The LILAC system[] for macro cells developed at Hitachi used clustering techniques for placement. These methods provide fairly good initial placements but iterative improvement methods should be applied afterwards to obtain satisfactory results. For example, the SHARPS system[] for macro cells and the Oki system[] for gate arrays have followed this strategy. The running time of these methods are in general short.

The force-directed heuristics can be used both for constructive placement and for iterative improvement. They have found wide applications in a variety of gate arrays and macro cell systems such as the PLINT[] system developed by GE for standard and macro cells, the APLS2[] system developed by Hughes for standard cells, the SHARPS system developed by Sharp for macro cells. the BAGEL system developed by the University of California at Berkeley, the MARC system[]developed by NTT , the MASTER and the LAMBDA systems developed by NEC, and the MARS-M3 system developed by Mitsubishi, all for gate-arrays. The basic idea is to represent the interactions between modules with a set of forces

$$F_{ij} = c_{ij}\ d_{ij} \qquad (5.2.3.1)$$

where d_{ij} is the distance between module i and j in general measured as the distance between the centers of the modules. Note that (5.2.3.1) is the expression for the force between two points connected by a spring with constant c_{ij} . The distance can be measured according to different metrics, for example the L_1 metric corresponds to the Manhattan distance, i.e., to the sum of the x and y distances. An initial placement is then constructed by finding the locations of the modules that minimize the overall force exercised on the modules. If no repulsive force is modeled, then all the modules end up in the same location. However, if pads are introduced and their location is maintained fixed, then the force exercised by the connections between the pads and the modules will avoid the complete overlap of all the modules. However, some overlaps may remain. Repulsive forces

have been introduced [] to avoid overlaps. The MARC system used this technique.

When overlaps are allowed, a feasible placement is constructed by modifying as little as possible the placement obtained by the force-directed technique. In the case of standard cells and gate arrays, the force-directed placement is used to determine the relative positions of the cells, which are then placed in rows according to the floor-plan of the chip.

Force-directed heuristics can also be used for iterative improvement algorithms. The SHARPS, MASTER, LAMBDA systems used variations of force-directed relaxation to improve layouts obtained by other methods such as min-cut and partitioning.

Pairwise interchange methods are very simple. Two modules are selected for consideration and interchanged if the cost function, whatever this function may be, is decreased. The key issue here is how to select the pair to be examined. A random selection is often used. An exhaustive examination of all pairs of modules is also possible, although quite expensive.

One of the most successful technique for placement is the min-cut method, proposed by Breuer[] for PCBs, gate-arrays and standard cells and by Lauther for macro cells. The basic procedure is based on the recursive application of the bi-partitioning algorithm by Kernighan and Lin []. At first the area of the chip is subdivided into two parts either with a vertical "cut-line" or with an horizontal "cut-line". Modules are assigned to the two areas so that the interconnections between the modules are minimized and the area of the modules assigned to the two parts is roughly equal. Note that in the macro-cell case, it is very difficult to take into account the aspect ratio of the modules, hence only the area of the modules is used in the partition process. Once the first partition has been applied, the two areas are subdivided again each into two parts. This subdivision can be obtained with either vertical or horizontal cuts. When an area is occupied only by one module, the area is obviously not subdivided any more. When all the areas cannot be subdivided further,

the process terminates.

Note that because of the different aspect ratios of the modules, the placement can be considerably improved by rotating and mirroring the modules. In fact, the algorithm presented by Lauther includes post-processing steps that improve the original placement. In particular, rotations and mirroring are used to improve both area utilization and net length. Finally, modules are shifted to eliminate as much as possible empty areas.

Recently, placement programs have been developed based on simulated annealing [KIR83, VEC84, SEC85]. The results obtained are excellent and often better than the ones obtained manually at the expense of computer time. Accelerators have been developed to speed up the execution of the algorithm [SPI85].

The overall area occupied by the design is obviously dependent on the routing area. The estimation of the wiring area is one of the most difficult parts of a placement algorithm especially for the macro-cell case. Simple estimates based on the number of pins on each side of the modules are often used to enlarge the area of a module to account for routing area. In other approaches, the routing area is not taken into account and only after global routing the routing area is inserted in the floor-plan of the chip. Local rearrangements of the placement are then applied to make room for the additional area.

Partitioning, which is a fundamental component of the min-cut algorithm, is often used to decrease the complexity of placement. This task is certainly useful when thousands of objects have to be placed and the running time of the best algorithms often increases more than linearly (usually quadratically or cubically) with the number of objects. In this case, logic gates or functions are grouped together and assigned to blocks with fixed or variable dimensions. Then, the placement stage determines the actual positions of the components in the blocks and of the blocks on the chip. Note that partitioning can be and has been applied to all design styles. For example, the Engineering Design System of IBM uses partitioning[] as a pre-process to their gate-array placement program. In

the standard cell system of Hitachi[] the cells are first partitioned into rows without assigning them a precise position inside the rows.

Partitioning is described as the assignment of objects with a certain size and connected by weighted nets, to partitions that have a bounded capacity so that the weight of the nets that span partitions is minimized. This problem is NP-complete and hence many heuristic partitioning algorithms have been proposed over the years. These algorithms are similar in their architecture to the algorithms used in placement.

5.2.4. Routing

After the modules have been placed, the interconnections have to be completed in the available space. If routing over the cell is not allowed by the technology or by the design style, the area available for interconnections is the one which is not occupied by the modules. In the gate-array and standard cell case, this area is fairly regular, it consists of a collection of rectangular regions alternating with rows or columns of cells. In the case of macro cell design style, the routing area is much less regular and can be "organized" in many different ways.

In general, it is convenient to decompose the routing region into rectangular regions, called channels. The way in which the routing region is subdivided may make the routing problem easier. This decomposition is called *channel* or *routing region definition* stage.

The routing stage where the interconnections are laid out on the chip follows. This stage is in general broken down into two stages: *global* or *loose routing* and *detailed routing*. The global routing stage, sometimes called channel assignment, determines which channels the interconnections will go through. Finally the detailed routing stage determines the actual physical location of the interconnections inside the routing regions. If the region to be routed contains pins on two sides only, then effective detailed routing tools called *channel routers* can be used. Extensions to the basic tools can provide programs

that can route channels with pins on three sides [BRA] If the routing region has pins on four sides, then a switch box router can be used []. In general, channel routers have the best results in terms of the area used to complete the interconnections. Thus routing regions with fixed pins on more than two sides should be avoided. Channel definition and the ordering with which the channels are routed has a great impact on this issue.

Since global routing and routing region definition depend critically on the routing strategy followed in the detailed routing stage, we review first the work in detailed routing. Then we present channel definition and ordering, and finally global routing.

5.2.4.1. Detailed Routing

Given a region with pins on its sides and, possibly, in the middle, detailed routing is the process of implementing the actual geometries of the interconnections among the pins specified by a net list. In the most general case, the regions may be of irregular shape with internal obstructions. However, the most effective algorithms work on regions of regular shape, in general rectangular or close to rectangular, with no obstructions and with pins on two opposite sides.

The basic algorithms for detailed routing are:

1- The Lee maze router [LEE61];

2- The Hightower line expansion algorithm [HIG69];

3- The Hashimoto and Stevens channel router [HAS71].

The Lee maze router, also called the Lee-Moore algorithm or the grid expansion algorithm, is applied to the interconnection nets, one at a time, on a region where a grid has been superimposed. The grid specifies intermediate locations which can be reached by an interconnection while it is being built. In general, it is assumed that the interconnections have Manhattan geometry, i.e., that they are formed only by vertical and horizontal segments. At each point of the grid, the interconnection may change direction. The grid may

be built so that two interconnection running in parallel in two adjacent grid locations do not violate any design rules.

This router has been applied to gate-array and macro-cell design. Its strengths are its flexibility (it can be applied to irregular regions with pins distributed everywhere and with obstacles, it can generate paths with minimum number of bends) and in the capability of finding a solution, if one exists; its weakness, besides the running time, is the dependence on net ordering. In fact, the first nets to be routed, have a large region basically empty to use, while the last nets to be routed find the region almost full. If the nets are chosen in the wrong order, the last nets may not be routable due to the blockages created by the previously routed nets. Several heuristics are available to speed up this algorithm as well as to choose a good net ordering.

The Hightower algorithm is gridless in principle. It starts from both pins to be connected and generates an horizontal segment and a vertical one of maximum extension from the pins. Once these four lines are generated, the orthogonal lines of maximum expansion are generated next. If more than one orthogonal line can be found of the equal maximum extension, the one which is closer to the opposite pin is selected. This procedure is iterated until two lines expanded from the two pins to be connected intersect. The actual interconnection pattern is then constructed by tracing back the lines at their intersection points.

Note that the algorithm can be quite fast for simple mazes with a small number of barriers and obstructions, while it may be slow for complicated regions, because of the many lines that can be generated before an intersection is found. In addition, it is not guaranteed to produce a solution, if one exists.

The contribution of Hashimoto and Stevens with their channel router is two-fold: i) the abstraction of a routing problem which is simpler to solve than the general problem stated previously; ii) an algorithm to solve the simplified routing problem. Many exten-

5.2.2. Placement and Partitioning

The placement problem involves the assignment of specific locations to building blocks of the layout. This includes the assignment of logic gates within a gate-array[67], the placement of cells in a standard cell layout[68,69] or the placement of macro cells [71,72]. While a considerable amount of theoretical work has been done in this area [72,73], the most successful approaches involve the use of simple heuristics. In these approaches, either total interconnect length, estimated or exact routing requirements are used by the placement algorithms as an indication of the quality of the placement.

As for partitioning, placement algorithms may be subdivided into two basic categories: constructive placement and iterative improvement. Constructive placement algorithms build a placement from initial data such as the size and the type of the cells to be placed and the net-list. Iterative improvement algorithms start with a given initial placement, which can be given by the user, generated randomly or obtained by constructive placement algorithms, and modify the layout to improve its quality. Interestingly, most of the placement algorithms can be used in the basic design styles, gate-array, standard cell and macro cell, with minor variations.

The most well-known heuristics are clustering[], force-directed [], pairwise interchange[], and min-cut []techniques. In all cases the placement problem is represented by modules with a given size and a connectivity matrix $C = [c_{ij}]$ where c_{ij} represents the nets connecting module i to module j with appropriate weights to model the relative importance of the interconnections.

Clustering adjoins incrementally modules to a subset of modules already placed. The modules are adjoined according to their size and connectivity. Critical factors for this algorithm are the selection of the seed, (i.e. of the first module to place), the selection of the next module to be placed and the position of the module with respect the modules already placed. A number of papers have been published describing a variety of schemes for each

of the factors above. Note that once the position of a module has been assigned, it is not changed during the remaining part of the algorithm. The LILAC system[] for macro cells developed at Hitachi used clustering techniques for placement. These methods provide fairly good initial placements but iterative improvement methods should be applied afterwards to obtain satisfactory results. For example, the SHARPS system[] for macro cells and the Oki system[] for gate arrays have followed this strategy. The running time of these methods are in general short.

The force-directed heuristics can be used both for constructive placement and for iterative improvement. They have found wide applications in a variety of gate arrays and macro cell systems such as the PLINT[] system developed by GE for standard and macro cells, the APLS2[] system developed by Hughes for standard cells, the SHARPS system developed by Sharp for macro cells, the BAGEL system developed by the University of California at Berkeley, the MARC system[]developed by NTT , the MASTER and the LAMBDA systems developed by NEC, and the MARS-M3 system developed by Mitsubishi, all for gate-arrays. The basic idea is to represent the interactions between modules with a set of forces

$$F_{ij} = c_{ij}\, d_{ij} \qquad (5.2.3.1)$$

where d_{ij} is the distance between module i and j in general measured as the distance between the centers of the modules. Note that (5.2.3.1) is the expression for the force between two points connected by a spring with constant c_{ij} . The distance can be measured according to different metrics, for example the L_1 metric corresponds to the Manhattan distance, i.e., to the sum of the x and y distances. An initial placement is then constructed by finding the locations of the modules that minimize the overall force exercised on the modules. If no repulsive force is modeled, then all the modules end up in the same location. However, if pads are introduced and their location is maintained fixed, then the force exercised by the connections between the pads and the modules will avoid the complete overlap of all the modules. However, some overlaps may remain. Repulsive forces

the standard cell system of Hitachi[] the cells are first partitioned into rows without assigning them a precise position inside the rows.

Partitioning is described as the assignment of objects with a certain size and connected by weighted nets, to partitions that have a bounded capacity so that the weight of the nets that span partitions is minimized. This problem is NP-complete and hence many heuristic partitioning algorithms have been proposed over the years. These algorithms are similar in their architecture to the algorithms used in placement.

5.2.4. Routing

After the modules have been placed, the interconnections have to be completed in the available space. If routing over the cell is not allowed by the technology or by the design style, the area available for interconnections is the one which is not occupied by the modules. In the gate-array and standard cell case, this area is fairly regular, it consists of a collection of rectangular regions alternating with rows or columns of cells. In the case of macro cell design style, the routing area is much less regular and can be "organized" in many different ways.

In general, it is convenient to decompose the routing region into rectangular regions, called channels. The way in which the routing region is subdivided may make the routing problem easier. This decomposition is called *channel* or *routing region definition* stage.

The routing stage where the interconnections are laid out on the chip follows. This stage is in general broken down into two stages: *global* or *loose routing* and *detailed routing*. The global routing stage, sometimes called channel assignment, determines which channels the interconnections will go through. Finally the detailed routing stage determines the actual physical location of the interconnections inside the routing regions. If the region to be routed contains pins on two sides only, then effective detailed routing tools called *channel routers* can be used. Extensions to the basic tools can provide programs

that can route channels with pins on three sides [BRA] If the routing region has pins on four sides, then a switch box router can be used []. In general, channel routers have the best results in terms of the area used to complete the interconnections. Thus routing regions with fixed pins on more than two sides should be avoided. Channel definition and the ordering with which the channels are routed has a great impact on this issue.

Since global routing and routing region definition depend critically on the routing strategy followed in the detailed routing stage, we review first the work in detailed routing. Then we present channel definition and ordering, and finally global routing.

5.2.4.1. Detailed Routing

Given a region with pins on its sides and, possibly, in the middle, detailed routing is the process of implementing the actual geometries of the interconnections among the pins specified by a net list. In the most general case, the regions may be of irregular shape with internal obstructions. However, the most effective algorithms work on regions of regular shape, in general rectangular or close to rectangular, with no obstructions and with pins on two opposite sides.

The basic algorithms for detailed routing are:

1- The Lee maze router [LEE61];

2- The Hightower line expansion algorithm [HIG69];

3- The Hashimoto and Stevens channel router [HAS71].

The Lee maze router, also called the Lee-Moore algorithm or the grid expansion algorithm, is applied to the interconnection nets, one at a time, on a region where a grid has been superimposed. The grid specifies intermediate locations which can be reached by an interconnection while it is being built. In general, it is assumed that the interconnections have Manhattan geometry, i.e., that they are formed only by vertical and horizontal segments. At each point of the grid, the interconnection may change direction. The grid may

be built so that two interconnection running in parallel in two adjacent grid locations do not violate any design rules.

This router has been applied to gate-array and macro-cell design. Its strengths are its flexibility (it can be applied to irregular regions with pins distributed everywhere and with obstacles, it can generate paths with minimum number of bends) and in the capability of finding a solution, if one exists; its weakness, besides the running time, is the dependence on net ordering. In fact, the first nets to be routed, have a large region basically empty to use, while the last nets to be routed find the region almost full. If the nets are chosen in the wrong order, the last nets may not be routable due to the blockages created by the previously routed nets. Several heuristics are available to speed up this algorithm as well as to choose a good net ordering.

The Hightower algorithm is gridless in principle. It starts from both pins to be connected and generates an horizontal segment and a vertical one of maximum extension from the pins. Once these four lines are generated, the orthogonal lines of maximum expansion are generated next. If more than one orthogonal line can be found of the equal maximum extension, the one which is closer to the opposite pin is selected. This procedure is iterated until two lines expanded from the two pins to be connected intersect. The actual interconnection pattern is then constructed by tracing back the lines at their intersection points.

Note that the algorithm can be quite fast for simple mazes with a small number of barriers and obstructions, while it may be slow for complicated regions, because of the many lines that can be generated before an intersection is found. In addition, it is not guaranteed to produce a solution, if one exists.

The contribution of Hashimoto and Stevens with their channel router is two-fold: i) the abstraction of a routing problem which is simpler to solve than the general problem stated previously; ii) an algorithm to solve the simplified routing problem. Many exten-

sions and improvements have been made to the original algorithms, but the concept of solving the routing problem by "carving" simple routing regions has permeated routing packages for all design styles for many years.

The basic assumptions in the original formulation of the channel routing problem by Hashimoto and Stevens are: (i) the routing region is rectangular with no obstructions and with pins on two opposite sides; (ii) floating pins on the other two sides of the rectangular region are possible. These floating pins indicate the need to extend some of the nets outside the channel; (iii) there are only two layers available for interconnections; all the horizontal segments of the nets are routed on one layer, all the vertical segments on the other (this assumption is called the wiring model); (iv) the pins are placed on a regular grid; (v) the channel is subdivided in rows or tracks whose spacing is such that interconnections placed on these tracks does not violate any design rule.

Some of these restrictions have been removed often paying with the final quality of the solution. The goal of channel routing is to complete all the interconnections in the minimum number of tracks. The allowable configuration of the nets is either (i) a horizontal segment in one layer, which is connected to the top and bottom pins of the net by vertical segments in the other layer, to minimize the number of change of layers or (ii) a set of horizontal segments joined by vertical segments (doglegs).

If we define as density of a channel the maximum number of nets which crosses any one column of the channel, then the best we can do with a channel routing algorithm which satisfies the assumptions above, is to route the channel with a number of tracks equal to the density of the channel. Interestingly, most of the best channel routers available today (Yoshimura and Kuh [YOS], Rivest[RIV], Burstein[BUR] and YACR[REE]) behave well, i.e., in most of the cases they route channels in a number of tracks which is close to density.

When pins are placed on all four sides of the channel, then the routing problem is called a switch-box problem. Some of the algorithms described above can be generalized to route a switch-box, even though this problem is much more difficult.

Note that all the algorithms used for detailed routing use symbolic data, for example, a channel router places horizontal segments with no vertical dimensions and with no information about contact sizes. In general, a post-processor is used to replace the symbolic data with actual geometries. This post-processor may be intelligent, i.e., it can change some of the interconnections to maximize the use of the layer with better electrical characteristics and to minimize the number of contact vias.

5.2.4.2. Routing Region Definition and Ordering

As pointed out in Section 5.2.3.1, channel routers are most effective to generate compact routing. Hence, the basic goal of any routing region definition and ordering scheme is to decompose the routing area and decide in which order the regions should be routed so as to use channel routers as much as possible. Note that in the case of gate-arrays and standard-cells the floor-plan is such that all the routing regions are already defined to be channels with pins on opposite sides. The real problem arises for the macro-cell design style.

There are two basic requirements for a routing region to be considered a channel: (i) all the pins on two opposite sides have to be fixed while the pins on the other two sides have to be floating, (ii) once a channel has been routed, only the distance between the two sides with fixed pins may be changed. This second constraint, also called *rigidity constraint*, is introduced to avoid re-routing a channel which has already been routed. Note that increasing the relative distance of the two opposite sides does not change the difficult part of routing. In fact, the updating of the interconnections after this move amounts simply to extending the vertical segments connecting the horizontal segments to the pins.

Assuming that all the modules are rectilinear (not necessarily rectangular), the routing region can always be decomposed into rectangles. This step can be carried out quite efficiently by using sorting techniques on geometries as commonly done in computational geometry[].

The most rigorous approach to channel definition and ordering is followed by Dai et al. in [DAI85]. This approach can also be used as a framework for the work of others. Here, routing regions are represented by *walls*. Walls can meet orthogonally either with a "T" shape connection or a cross connection. Cross connections can always be represented by "T" connections and hence will not be considered further. If each routing region represented by a wall is to be considered as a channel, the constraints introduced above, induce an ordering relation on the walls. In particular, a wall which is the "vertical" part of a "T" connection, corresponds to a channel that must be routed before the one which corresponds to the "horizontal" part of the junction. In fact, if we route the region corresponding to the "horizontal" wall in the "T" connections first, we have a set of pins whose position is not specified, on one of the sides where fixed pins are located If the "vertical" part is routed first, then the channel router specifies, when it terminates, the exact location of the previously floating pins. The connections then induce a precedence relation among channels. If such a relation is acyclic,i.e., there is no "conflict", then there exists an ordering of the routing regions so that each of the channels considered in the sequence has fixed pins on two opposite sides and floating pins on the other two sides.

Some placement algorithms have the nice property that the relation defined on the walls is always acyclic and they even provide the routing order. One such algorithm is the min-cut algorithm presented in Section 5.2.2. In this case, each cut corresponds to a routing region and the routing regions corresponding to the cuts identified by the leaves of the binary tree constructed by the algorithm, are the first to be routed since they have pins on fixed positions on the two sides facing the blocks separated by the cut. After the

channels corresponding to the leaves of the tree have been routed, the next level cuts identify the channels to be routed next. The routing order is then completely specified by the binary tree.

All the placement algorithms that yield an acyclic relation on the walls are said to generate a *slicing structure* [OTT]. Unfortunately, not all the placement algorithms yield a slicing structure. Indeed, in some cases, an algorithm that generates always a slicing structure may result in wasted area, especially in the case of non rectangular blocks. In this case, other algorithms that do not yield a slicing structure may be used and a cyclic relation among the walls may be generated. Several approaches have been tried to solve this problem: some modify an existing placement to create a slicing structure, others define routing regions which are more complicated than simple channels, switch boxes, L-shaped channels, and route these to break cycles.

5.2.3.3. Global Routing

Channel routing is the most effective way of routing regions. Unfortunately the entire routing region of a chip is not a channel. We have described ways of subdividing the routing regions into channels. Before applying the channel routing algorithms to the problem, nets have to be assigned to channels. Global routing assigns nets to routing regions, taking into consideration net length, the congestion of the routing regions, priority of signals, and electrical characteristics. It does not specify the route followed by the interconnections inside the routing regions. Global routing is very important to obtain a good overall layout. In this section, we review briefly the main approaches. The excellent review by Sadowska and Kuh[] can be consulted for additional information.

Global routing is used in all design style: gate-arrays, standard-cells and macro-cells. The first mention to loose routing can be found in Nan and Feuer[].

There are two basic approaches to global routing: one deals with the interconnections one at a time, in this respect similar to the Lee algorithm, the other deals with the interconnections all at once. Of course, the second approach does not suffer from the ordering problem and has a better "global" view of the chip. However, the running time can sometimes be prohibitive.

The basic problem that a global router has to solve is to distribute the nets in the available channels so that either the density of each of the channel does not exceed a bound (gate-arrays) or the overall size of the chip is minimized. In the latter case, the density of the channels is a variable to be determined by the router. In all the applications of global routing, it is assumed that, because of the quality of channel routers, the detailed routing can be completed within the density of the channels or just above density.

A net-at-a-time approach chooses a path by using shortest path algorithms that penalizes paths for crossing congested channels. This strategy tends to congest fewer areas and improve the performance of the chip. The cost function associated with each "segment" crossing a particular channel is usually expressed as

$$aL + \frac{b}{c^{T+1}}$$

where a, b and c are parameters to be tuned for the particular application, L is the length of the channel and T is the number of available tracks. Of course, T changes for the channels after each net has been considered by the global router.

If the nets have only two pins, then the approach mentioned above is straightforward. However, if the nets have more than two pins, then many possible interconnection topologies are possible. In fact, the net list specifies only that a set of pins have to be connected but it does not specify in which order they should be connected. For example, for the four pin net shown in Figure FIGNO the interconnection can be any one of the patterns displayed. According to the policy followed by the designer, the geometries can

be restricted to be Manhattan. The problem then is to find the best topology for each of the nets to be routed. Following a shortest path algorithm between pair of pins does not give an optimal result in general. The complication arises from the possibility of introducing additional "pins" in the interconnection. This problem is known as the optimal Steiner tree problem. If the set of modules and channels is modeled as an undirected graph, whose nodes represent the modules and whose edges represent the channels, then the problem becomes the Steiner tree problem on graphs. Unfortunately Steiner tree problems are NP-complete and heuristics are commonly used[].

The other important approaches deal with the nets all at once. One approach formulates the global routing problem as a mathematical programming problem: a 0-1 linear programming problem. It is well known that this problem is also NP-complete and heuristics have to be used to solve it in a reasonable time. One of the most interesting approaches is to use a linear programming algorithm to find approximate solutions to the 0-1 problem, and then round the solutions to obtain a feasible solution.

Simulated Annealing has been used by Vecchi and Kirckpatrick[] to solve the mathematical programming problem. In this case, the cost function is equal to the sum of the squares of the congestion for each channel, i.e. the number of nets crossing that channel. In this way, the algorithm penalizes congested areas. The "moves" correspond to switching one net from a path to another.

Burstein [BUR85] considers the global routing problem hierarchically. The chip is first divided into four regions and the nets are routed in these regions. Then, each of the regions is subdivided into four regions and the implications of the first routing phase are propagated to the next level of hierarchy. The procedure terminates when the real routing regions are considered at the appropriate level of hierarchy. Marek-Sadowska[MAR85] considers a similar approach where the hierarchy is traversed bottom-up instead of top-down.

In the gate-array case, if the constraints on density are not met, then a rip-up and reroute phase is added at the end of the main procedure. This process has traditionally been carried out by human designers with the help of a symbolic graphic editor. Rule-based approaches have been used by Marek-Sadowska [MAR] and by Goto et al. [] to identify the nets to be ripped and the ones to be re-routed.

5.2.4. Compaction

In any design style, basic circuit cells have to be designed and verified, either for each design as in the case of full custom, or once for gate-array design style, where the designers responsible for the structure of the gate-array has to design the basic cell, and for standard-cells, where those responsible for the libraries have to design the cells. These cells may be simple logic gates, such as nand, nor gates, and flip-flops. Alternatively, they may be cells used in regular arrays, such as the one-transistor cells used in a Programmable Logic Array (PLA), which do not perform a complete logic function alone. The most commonly used aid for the physical design of a cell is a mask-level digitization and interactive correction program.

Some programs allow direct symbolic layout entry, using either fixed-grid[33-35] or relative-grid[36-39] schemes. With the fixed-grid symbolic approach, the grid is designed to ensure all basic layout rules are satisfied upon data entry. For relative-grid schemes, it is necessary to modify the layout such that all layout rules are satisfied. Programs which carry out this operation are often referred to as *compaction* programs since they also attempt to reduce the area occupied by the circuit.

Once a symbolic layout has been entered into the computer, it may be compacted by adjusting the size of non-critical components, such as interconnections, under the constraints imposed by the physical and electrical layout rules of a given technology. Hence, this approach allows symbolic layouts to be updated more easily than physical ones as

design rules or technology change. The FLOSS program[36], developed at RCA for the compaction of circuit cells, paved the way for the development of transistor-level compaction programs for IC's.

It can be proved that the two-dimensional compaction problem is NP-complete. For this reason, the algorithms used for compaction are heuristic and generally perform x and y axis iterative compaction steps until all layout rules are satisfied and no further area reduction can be achieved. Recently there have been efforts to develop algorithms that could perform x and y axis compaction at the same time [Wong] but the results obtained are not suitable for a practical implementation.

Local modifications to the layout can be performed to allow further compaction. These modifications generally consist of distortions to interconnect, such as the introduction of "jogs", or the rotation of transistors and cells[32,37]. Critical path algorithms and force-directed heuristics are used to determine the best location for the introduction of these layout modifications. To ensure the layout is least sensitive to processing tolerances, non-critical components must then be placed midway between constraints to maximize yield.

The use of an hierarchical description of the circuit can be exploited to reduce the analysis time by compacting the cells independently. The resulting compacted cells may then be combined and compacted to form the circuit. While this may not result in an optimal area utilization, the primary objective of error-free layout is achieved.

Lower-bound constraints on the positions of the elements of the symbolic layout may not be enough to capture the constraints on the cell layout. For example, upper-bound constraints may be necessary to express constraints on timing performance of the circuit [Wong trans on cad]. The inclusion of both upper-bound and lower-bound constraints may lead to over-constrained problems that do not have a solution.

More complex constraints need to be addressed by future compaction programs. In fact, high performance digital circuit layout as well as analog circuit layout may require that two signal paths are of the same length. This implies that compaction algorithms that can accommodate couplings between constraints have to be developed.

5.3. Logic Synthesis

The synthesis of a circuit — deciding how to partition the logic, in what form to implement specific pieces of the logic, and what layout-style to use for implementation — is still a largely manual process. For digital circuits separated into data-path and control circuits, the control logic portion of the chip is often the most time-consuming piece to design. It is generally on the critical path for timing, and, because of limits in design time, is often implemented in a very inefficient way. Automated synthesis of the control logic blocks of a chip, optimized for speed and area, provides one of the major challenges facing CAD today. In this section, the state-of-the-art for the synthesis of combinational and sequential, two-level and multi-level logic synthesis is presented. Areas which provide the most potential for improvement are presented and recent work in this area is described.

5.3.1. PLA-Based Synthesis of Control Logic

Programmable Logic Arrays (PLAs) are perhaps the most popular structures for the implementation of two-level logic functions. Most modern VLSI microprocessors include large PLAs to implement the datapath control, as well as a variety of smaller PLAs for controlling other activities on the chip. Other chips, such as memory management circuits often consist almost solely of PLAs.

Many PLA layout generators have been written based on simple translations of the boolean equations into layout, e.g. [LAN81, GLA80]. However, a straight-forward imple-

mentation of the logic entered by the designer may result in PLAs which are large and, as a result, have poor performance in terms of speed and power.

5.3.1.1. Combinational Logic

It is clear that PLA optimization is necessary to obtain an effective implementation. The optimization steps involved in the transformation of combinational logic into the layout of a PLA are:

(1) *Logic-level* optimization which aims at the reduction of the number of product terms needed to implement the function.

(2) *Topological* optimization which aims at the elimination of unused space inside the core of the PLA. (e.g., folding and simple partitioning).

(3) Layout and circuit optimization, which attempts to perform optimal sizing and placement of drivers, loads, core cells, and additional ground lines.

Over the past few years, a great deal of attention has been paid to logic minimization of two-level logic. When the logic function is implemented using a PLA, logic minimization both reduces the area occupied by the PLA and improves its electrical performance. The algorithmic complexity of complete logic minimization is very high and so approximate logic minimizers are used when medium and large logic functions have to be minimized. MINI[] developed by Hong et al. at the IBM T.J. Watson Research Center was the first efficient heuristic logic minimizer to provide quality minimization. Recent research on approximate logic minimization algorithms has produced an efficient new logic minimizer, Espresso II-C [BRA84,RUD85]. Espresso II-C has been found to be very effective in minimizing complex logic functions while consuming a reasonable amount of computer resources.

Once the logic minimization is complete, topological optimization can be performed to minimize the area of the core occupied only by interconnect which does not contribute directly to the implementation of the logic function. The objective of folding is to

determine a permutation of rows and/or columns of the array which permits a maximal set of column pairs to be implemented in the same column or row of the logic array.

The first optimized PLA synthesis system described in the literature was the PRESTO/BLAM/PLAID[HOF81] system developed at Berkeley in 1980. This system incorporated the two-level logic minimizer PRESTO[BRO81], a folder, BLAM and the layout program PLAID. The PRESTO program was later replaced by an improved optimizer, POP. The system produced NMOS simply-folded PLAs. The interface between the logic designer and the system was EQNTOTT[CME81], a program which accepts an arbitrary combinational logic function expressed in forms of logic equations and produces a truth table for PRESTO. A system developed at the IBM T.J. Watson Research Center, written in APL, included similar optimization steps (folding and minimization) with in addition a partitioning capability. The logic minimization MINI[HON74] was part of this system.

A mathematical formulation of the optimal folding problem was postulated and new folding algorithms were developed[HAC80] that yielded better results than the ones obtained by the algorithms initially included in BLAM. Research at Stanford culminated in the development of a PLA synthesis system whose input is a high level functional language, DDL [KAN81]. PLA minimization was performed by SPAM and the topological optimization was accomplished by PAPA, a program which decomposed large PLAs in smaller ones.

Many folding algorithms were developed following the mathematical model presented in [HAC80], e.g. [SUW81, CHU82, EGA82, GRA82, HU83]. More recently, a new folding technique called multiple folding [DEM83] was developed which can reduce substantially the area used by a PLA and was incorporated in the program Pleasure. The PLA synthesis system PLASCO[BAR85] includes a folding program which can also generate multiply folded PLAs.

5.3.1.2. PLA-based Finite-State Machines

Recently, sequential circuits have been implemented using regular arrays. In particular, PLA-based Finite-State Machines (FSM) have been used in the design of several micro-processors and telecommunication circuits. Such circuits use a PLA to implement the combinational part of the logic and the secondary outputs are fed back to the secondary inputs of the PLA via clocked latches. For a given set of primary (external) inputs and a required set of primary outputs, the objective is to:

(1) choose the number of secondary outputs to be fed back, via the latches, as inputs and

(2) assign values to these outputs (logic-'1' or logic-'0') for each state specified in the FSM description

such that the total area occupied by the combinational logic and/or the critical-path delay through the PLA are minimized. The 'textbook' approach to this problem, originated in the days of discrete SSI circuits, is to choose the number of secondary variables so as to minimize the number of latches used. By doing so, one minimized the number of expensive IC packages needed for latches. To reduce the amount of combinational logic, various heuristic schemes were used. The most common approach was to used a 'distance-one' state encoding for adjacent states. Unfortunately, such a state-encoding strategy does not work well for PLA-based FSM designs.

To solve this problem optimally for PLA-based designs, an efficient approach to state assignment is needed. Many algorithms have been proposed in the past, e.g. [DOL64, HAR61] to perform optimal state-assignment. However the results obtained were not satisfactory because of the complexity of the algorithms suggested or of the poor electrical performance of the PLA used to implement the resulting combinational part of the FSM. A new approach for performing an optimal assignment of binary codes to the inputs of the PLA implementing the combinational part of the FSM has been developed in [DEM84]. The program KISS [DEM84b] was developed to determine a state-assignment based on this

algorithm. The advantage of such an approach over a conventional design method is illustrated in Figure 2. In Figure 2(a), the original FSM is shown with a 5-bit, minimal-length state vector (and three additional outputs). In a 3μ P-well CMOS process, this circuit occupied $0.75mm^2$. After processing the FSM description using KISS, a 9-bit state vector was chosen. This resulted in four additional output columns but reduced the overall area to $0.39mm^2$ by reducing the number of product terms substantially, as shown in Figure 2(b).

The KISS approach is successful, but an extension to this technique is needed to capture the full aspect of the optimal state-assignment problem. The algorithms for folding have also to be modified if the PLA to be folded comes from the implementation of the combinational part of a FSM, because additional constraints are created by the presence of feedback registers. New algorithms are needed to cope with these constraints.

Further work is needed to partition a large FSM into smaller machines where the intermediate output values are encoded. Though this process may result in more stages of logic, in many cases it is expected that the increase in clock speed that can be achieved using the small PLAs will more than compensate for the additional clock cycles caused by more stages. Once again, algorithms have been proposed in the past [HAR66], but the technological constraints and objectives which drive the decomposition have changed drastically so as to make the existing algorithms inappropriate.

Figure 2. FSM Synthesis using KISS: (a) The hand-designed circuit
(b) After KISS redesign.

5.3.2. Multi-level Synthesis

As seen in the previous section, a great deal of work has been done to implement combinational logic in optimal, two-level form using the PLA. However, some control logic has a two-level representation which can have as many as 2^n product terms, where n is the number of primary inputs of the logic, even after minimization. In addition, even if a two-level representation contains a reasonable number of terms, there are cases in which a multi-level representation can be implemented in much less area and generally as a much faster circuit. In fact, a two-level logic representation can be viewed as a special case of general multi-level representations. Hence, a general framework for control logic design should offer multi-level synthesis tools which are able to select a two-level implementation wherever the two-level form is more effective in terms of area and/or speed. To be able to explore the design trade-offs such a system should offer a variety of both electrical design style (e.g. Domino logic, static CMOS) and layout design style (e.g. Weinberger arrays[WEI66], gate matrix [KRA82], standard cells, and gate-arrays) alternatives.

Several systems are being built for the design of control logic using multiple levels of logic. The precursors in multi-level logic synthesis are two systems developed at IBM: the IBM Logic Synthesis System (LSS) [DAR84], has as target technology a variety of gate-arrays and has been extended to standard cells and to CMOS dynamic logic; the Yorktown Silicon Compiler [BRA84b], has Cascode Voltage Switches [ERD84] as its target technology; and the MAMBO system[HOF85] uses Domino Logic. AT&T Bell Laboratories with the FDS system, NTT with the Angel system, NEC and Hitachi, with the POLARIS system were all developed with standard cells and gate-arrays as target technologies. Only recently, electrical consideration have been taken into account during the synthesis

process. MAMBO developed at the University of California, Berkeley is an example.

For multi-level design, there are two basic approaches to the logic optimization step:

(1) Global optimization, where the logic function is re-factored into an optimal multi-level form without considering the form of the original description (e.g. the Yorktown Silicon Compiler[BRA84b], part of Angel[HOS84], and FDS[DUS84]).

(2) 'Peephole' optimization, where local transformations are applied to the user-specified (or globally-optimized) logic function (e.g. a part of Angel, LSS[DAR84], MAMBO[HOF85]).

Some global optimization algorithms were proposed in the past (e.g. [ASH57]) to factorize a Boolean function, but these techniques required an exhaustive search which is prohibitively expensive for the complexity of control logic designers are interested in today. Some other algorithms suffered from the lack of understanding of the technological constraints associated with particular implementation of the logic. New algorithms have been proposed by R. Brayton and co-workers [BRA84] which are effective in partitioning complex logic functions and can take into consideration the technological constraints of a particular implementation.

The Logic Synthesis System of IBM [DAR84] uses a prototype expert system to accomplish the mapping of combinational logic in random form (from an initial "high level" description) into a gate array implementation. The LSS system has been very successful and was used for the design of a number of circuits in the recently-announced IBM3090 computer. The local transformations it uses are quite simple and relatively few in number. The transformations used for NAND gates are summarized in Figure 3. Such local approaches, as used in LSS, tend to be faster than the global schemes but they are somewhat limited in their search for a better design.

Figure 3. Local transformations used by LSS for NAND structures

5.3.3. Synthesis from High-Level Behavioral Description

The translation from a behavioral description to a register transfer level description involves architectural decisions that have great impact on the final quality of the design. The difficulty stems from the large number of constraints, design objectives and design configurations to consider. In addition, it is very difficult to evaluate at this stage the effect of an architectural decision on the speed, power consumption and area of the chip.

A key issue in behavioral level synthesis is the selection of the language used to describe the design. The language must be concise and have high level constructs to express compactly the intent of the designer. High level programming languages such as Modula, PL/1 and concurrent Pascal have been used to describe designs at behavioral level.

Special purpose languages such as the ISPS language developed at CMU[] and the MacPitts language[SOU83] have also been developed. These languages are powerful enough to allow the description of existing computers in only a few pages.

ISPS is the input to the CMU Design Automation System[THO83, THO81] and the MacPitts language is the input to the MacPitts silicon compiler[SOU83]. The two systems differ radically in that the CMU system does not follow the "style" of the input language program for implementation while MacPitts implementations follow closely the style of the input description.

In the CMU system, the ISPS input is translated into an internal, data-flow representation called Value Trace (VT), which is then used for the synthesis process. The VT representation depends on the ISPS "programming" style of the designer, but to optimize the final result, such dependency should be minimized. In fact, in this system it is assumed that the goal of the designer in describing the design at the behavioral level is compactness and clarity more than the optimality of the synthesized system. Thus, the VT representation is manipulated using techniques similar to those used in optimizing compilers[].

The next step, the mapping of the VT representation into an architectural representation, is the core of the synthesis system. Many approaches have been tried by the CMU group, some involving algorithmic approaches such as EMUCS[THO83] and Facet[TSE83], some involving Knowledge-Based Expert Systems such as the Design Automation Assistant (DAA)[KOW85] and the Sugar system under development[DIR85] (see the Expert System Window for more details). The architectural description is still technology independent but is selected with an eye on the technology that may be available for the design. The components of this description are modules, e.g. registers, operators, memories, multiplexers and buses, links and symbolic microcode that describes the control structure of the design. The selection of the architecture is either accomplished with algorithmic techniques, e.g.

graph theoretic algorithms[TSE83], or with a set of rules embedded in an expert system[KOW84].

Once the architecture has been specified, the modules have to be bound to components available in the selected technology and the microcode has to be implemented either with a PLA, a microprogrammed controller or random logic. This step is accomplished by the *module binder* and the *control allocator*.

The module binder selects technology dependent cells stored in a library to implement the modules specified by the synthesis tools. The control allocator determines the control signals that drive the data path. The output of the control allocator is either a PLA-format such as the one used by the Berkeley PLA tools or a micro-programmed-style output for an AM2910 microengine[KOW85].

The MacPitts input description is compiled into a data path and control by replacing the language constructs with hardware. The control part is expressed as a finite-state machine and implemented with a Weiberger array. The data path is synthesized using basic one-bit units called *organelles*. Operations which are specified in the input as mutually exclusive are implemented in parallel, i.e., hardware is generated for all the operations which can be done in parallel, while operations which are not mutually exclusive are implemented using as much as possible hardware already implemented for other operations. It is clear that if the input description specifies more operations as mutually exclusive, the execution of the operations is faster but more silicon is used. Thus, the user has some control over the hardware generated by the silicon compiler. As pointed out before, the fixed floor-plan and design style for the control unit has resulted in silicon implementations that are not as compact and efficient as human designs.

5.3.4. Procedural Design and Module Generation

In recent years, the notion of procedural circuit design[ref] has emerged as a key component in the design process. The use of procedure, rather than just graphics (or data), for describing IC designs was pioneered at Caltech [ref] and applied later at MIT in the DPL project[ref]. These efforts, and others, have inspired a large number of projects at University and Industrial sites for in-house use. The term *silicon compiler* is often associated with procedural design[2] and a number of new companies are advertising *silicon compilers*. Unfortunately, the majority of systems offered so far do *not* offer the general user a procedural design capability. Rather, they can be characterized as cell-based systems where the circuit building-blocks are parameterized cells. These cells can be assembled in a variety of ways, depending on the *design style* the "compiler" is using.

True procedural design, where the IC designer can write programs which, when executed, produce layout is still of key importance to the productivity of the custom IC designer. Silicon Design Labs offers a procedural Design system based on their 'L' language[ref], which is an evolution of the 'I' language, developed at AT&T Bell Laboratories[ref].

Many of the early procedural design systems had limited success for a number of reasons:

- The relationship between graphics and procedure was not exploited sufficiently. Graphics and procedure are generally treated as disjoint descriptions of the design. The graphical mask layout was the *result* of the procedure, rather than an active part of it.
- Verifying the correctness of a procedurally generated design is generally performed at the mask layout level. At that point, no correspondence between the active

[2] The term was first used at Caltech to describe the early procedural design work there.

procedure which created an object and its geometrical layout is maintained. As a result, debugging the design is very difficult.

- Often the designs produced by the early systems did not achieve high density, high performance, or meet power requirements.

Another way of reducing the potential for errors, as well as ensuring the technology-parameterization necessary at the lowest level of design, is to have the procedural design system generate symbolic layout rather than detailed geometry. A spacing program[ref], or constraint-solver[ref], can then be used to guarantee a layout-rule-correct design and to convert the symbolic layout to mask artwork.

6. SUMMARY

As described in this paper, computer aids have been used for both the design and verification of electronic systems for many years prior to the introduction of commercial ICs in the early 1960s. These tools have found their way into virtually every aspect of the design of such systems, from IC process technology to the design of complex computer architectures. Today, it would not be possible to design a complex IC without CAD tools and we believe soon these tools: for data management, verification, and synthesis; will be as significant than the underlying semiconductor technology in differentiating products in the marketplace.

The use of CAD in IC design is now a very broad and very deep subject. While it was not possible to go into detail in this paper, we have indicated the history of CAD for IC design, the state-of-the-art, and the present directions for future work. CAD is now a large industry and is growing rapidly. It has become a relatively sophisticated industry and is staying abreast of developments in computer science as well as computer architecture and IC design. As a result, the IC CAD industry is setting a direction for other CAD industries, including mechanical and board-level CAD.

8. REFERENCES

TO BE PROVIDED LATER

WINDOW: An Historical Perspective of CAD for ICs

Since the advent of the first IC, the evolution of computer aids for IC design has occurred in an *ad hoc* manner. In most cases, computer programs have been written to solve specific problems as they have arisen and very few truly integrated Computer-Aided Design (CAD) systems exist for the design of IC's. Most CAD systems currently in use for the design of complete ICs consist of a loose collection of programs, requiring a large collection of data formats and often requiring manual intervention to move from one program or computer system to another.

The first digital IC's were available commercially in the early 1960's and, in retrospect, it is surprising how little the computer was used in the design of IC's prior to 1980. Early circuits were sufficiently small that mask patterns could be drawn by hand on rubylith, and then photographically reduced to generate the IC masks directly. However, for the verification of the *function* of the circuit, simulators proved quite useful. Hence initial work in the mid-1960's focussed on the development of device analysis[??] and circuit analysis[??] techniques. These circuit simulators were originally developed for the analysis of nonlinear, temperature, and radiation effects in discrete circuits and it was not until the early 1970's that circuit simulators suitable for IC analysis became generally available[??].

As the complexity of the circuits increased, industry turned to the computer to store integrated circuit mask layout data; the arrangements of polygons that would be used to define transistors and interconnect on the final chip; and to produce the masks required for manufacture. Systems for layout digitization, where the layout is first drawn by hand on sheets of Mylar and then entered into the computer using a tablet and a puck, and interactive correction of the layout data, found extensive use by the early 1970's. However, it was not until the mid-1970's that programs for checking the physical layout rules for the

circuit (LRC) began to find widespread use[??]. These programs process the geometric descriptions of the layout and check to make sure that layout rules, such as minimum spacing between adjacent polygons or required enclosure of one polygon by another, are met.

By 1975 it had become clear that computer-aids were a necessity in the design of complex integrated circuits, both for physical and for functional design and verification. Until then, the layout of an IC and its transistor-level schematic diagram had been quite separate. In the late 1970's, computer programs became available for such tasks as *extraction* of transistor-level schematics from IC artwork data[??]: recognizing transistors and interconnect from patterns in the artwork data, connectivity verification[??]: comparing the transistors and their connections expressed in a schematic diagram with the connections extracted from the artwork data, and even extraction of gate-level netlists from the transistor list[??]. These programs were loosely-coupled and were often incompatible with one another. All of the early tools were developed with a "batch" computing environment in mind. None of the tools address the problem of design data management (other than for the data they deal with directly) and tools from different vendors typically use different input and output formats. The task of coordinating the tools and integrating them into a particular design flow fell largely to the IC house; largely to the central, or corporate, CAD group; and this task has traditionally been responsible for a lot of their headaches. In fact, the *only* fully integrated CAD systems that are in general use today for the design of complex ICs are those for some highly-specialized design approaches, such as the standard cell and gate array design styles.

Prior to, and in parallel with, the development of computer-aids for IC design, a great deal of work has been ongoing to aid the digital system designer, particularly as applied to printed circuit board design using standard components. In particular, algorithms and programs for the optimal placement and routing of cells[??], logic simulation

techniques[??], and test grading[??] have resulted in sophisticated design packages.

As the complexity of ICs and IC-based systems increased, these two worlds began to merge in the late 1970s and early 1980s. In addition, the IC industry saw the introduction of the first personal workstation-based interactive mask layout and schematic entry systems. The rapid drop in price/performance provided by these systems has had a dramatic effect on the IC design community, as well as causing a great deal of confusion! The additional advantages of predictable response time, communication among designers, improved user interface to the CAD tools, and the wide range of possible price/performance options has accelerated the acceptance of these systems. But the advent of the workstation had not been without its drawbacks. Early systems were often clumsy to use and did not live up to many of their claims. Over the past few years, however, workstation-based CAD systems have improved dramatically and are now available on machines from a standard personal computer to advanced, color-display workstations with the power of a superminicomputer of just a few years ago. There is no question that workstations for CAD are here to stay. With such rapid improvements in hardware, it has been difficult to keep the CAD tools portable enough to keep up. Workstation-based CAD vendors have either committed their systems to a particular manufacturer or have chosen a portable operating system, such as UNIX, in which to develop their tools. In that way, provided UNIX is available on the new hardware, the job of porting their software has been made considerably easier. Other techniques that have been used to ease portability while minimizing loss in efficiency include the use of portable programming environments such as Mainsail[??] and Lisp[??].

With the advent of low-cost, high-resolution graphics, another portability issue has emerged — that of user interface. Early implementations of window-managed user interfaces have involved customization of the code down to the assembly language level. In some cases, hardware manufacturers have provided efficient interfaces that have a com-

mon program-level interface to a variety of display devices. However, the interface is often proprietary or, if defined in terms of a graphics standard such as CORE or GKS, the interface is often too slow or inadequate for a window-managed environment. Recent University developments, such as MIT Project Athena and the CMU Spice project, may provide the basis for future portability in this area.

WINDOW: Managing CAD Development

For many years, most companies have worked with a central, or corporate, CAD group supporting the entire company, or business unit, CAD needs. Occasionally, the CAD is distributed and a CAD team is responsible for all CAD aspects of a single design project. In either case, a major dilemma is always the "buy versus build" decision. Most companies would buy if they could and only build what they have to but it is a common problem to find a tool that isn't "quite right" or to buy the tool only to find it does not perform as advertised.

Since the late 1960's, all large IC companies and most small companies have relied on the central CAD group to supply CAD support for a large number of design teams. It was often felt that since the computers themselves were expensive, often corporate-level resources, the group which supports design aids on such a facility should also be centralized. The group would support common tools, such as circuit simulators and layout-rule checking programs, for the entire company. They would also develop new and innovative tools to support the particular design styles favored by their designers. Unfortunately, this often lead to a "computer center" style of relationship between the IC design teams and the central CAD group. On the one hand, the designers felt that by having access to the CAD code themselves, with "just a few minor changes" a tool might do the job they wanted. On the other hand, the central CAD group was well aware that if they released the source to each design group, they would be asked — no, they would be *expected* — to support the resulting variant tools, tools they had no part in creating.

There have been many battles between a design team and the central CAD group. In recent years, design groups have often taken matters into their own hands by forming small, CAD support teams for each design under way. These teams are responsible for bringing together the appropriate CAD technology for the design method in use and for

building a CAD *system* to support the design. This generally involves obtaining tools from the central CAD group, buying some tools, and, generally as a last resort, building some tools of their own. While this approach often leads to a satisfactory solution for the design in progress, unless the design *framework* is managed carefully, it often leads to a CAD system of tools which is difficult to support and therefore cannot be used in other, similar design projects.

In many ways, the relationship described above is a no-win situation from both sides. On the other hand, a central CAD group has a very important role in a company which has a number of on-going, state-of-the-art design activities. It is certainly the best place to support common, basic tools such as circuit simulators, layout-rule checkers, and mask pattern generation software. It can also play an important role in the dissemination of information about new tools or a new use of an old tool. When one design team completes a successful design project, the central CAD group can often follow up on the use of CAD tools in that project and make the successes and pitfalls of the design style available to other, new IC design projects. Today, such information is often lost, rediscovered with each new design, or carried to the next project by the senior designers in the team.

Perhaps the most important role a central CAD group can play is the maintenance of the design *framework*, described in detail in Section 2. In its simplest terms, this involves the specification and support of standard interfaces between tools. Even today, these interfaces are often textual or binary *interchange formats*," such as the Calma GDS2 format for geometric data or the Spice2 input format for netlist data. Often there are many, occasionally synonymous, formats in use within a large organization — again resulting in a nightmare for the central CAD group. New formats, such as EDIF[??] and the VHDL[??], promise to reduce the complexity of this problem.

However, a circuit design framework can be far more powerful. It can include both common data management tools which support interactive design styles, as well as a com-

mon user interface. Such design frameworks are just beginning to be developed, as described in Section 3.

In the late 1970s and early 1980s, a number of new companies were formed, often by frustrated designers who saw the opportunity that low-cost computers provided, to address these problems. These companies focussed on data management, integrated user interface, and a selection of supported tools. Often they focussed their efforts on a particular segment of the market. Logic schematic entry and associated logic-level tools proved to be the most successful as they addressed both custom and semi-custom IC designers and board designers alike, therefore providing a much larger market to sustain growth. In contrast to earlier offerings, these systems provided interactive, graphical entry of the design data and they promised an interface to a wide variety of tools, as well as the ability to store and retrieve design data.

These companies have been joined by many others and the systems have broadened in terms of the design data they can deal with. At the 22nd Design Automation Conference in June, 1985, over fifty companies were offering CAD systems which ran on machines from personal computers to high-end mainframes. The systems showed a correspondingly broad range of capabilities. However, while the popular press promises the designer a "personal, integrated workstation," such an environment is still not cost-effective. The more useful workstations today are still relatively expensive — too expensive to provide a worstation per designer. A recent survey of designers and CAD managers indicated that on average an engineering workstation is being used to support 7.5 engineers: two years from now, the anticipated average is 2.5 engineers per workstation. Bringing the cost of these workstsions down is an important challenge to the worstation vendor. In many cases the design systems run on proprietary hardware, where a full range of cost/performance and multi-user systems is not available, or use proprietary networking technology, making it difficult to integrate such systems smoothly into a com-

plete office environment. Most engineers continue to require a computer terminal or PC with which to perform their work. Another interesting aspect of the survey was that while engineers felt that improving the quality of schematic entry is most important to enhancing their productivity and that document preparation tools (for writing reports, electronic mail, preparing presentations) was of least importance, they also agreed that while on average they spent 3 hours/week entering schematics, they spent a full 2 days/week using document preparation tools!

While the central CAD group plays an important role in the design process, care must be taken to avoid the CAD *tool* under development becoming the focus of the group, rather than a successful chip on first silicon being the ultimate goal of the CAD group as well as the design group. Often, a central team is more concerned with squeezing the last microsecond out of a logic simulation, rather than making the designer interface easier to use or offering the analysis options of most use during the design. Management plays the most important role here, by allocating some chip design responsibility to the CAD group. For example, by assigning a specific CAD individual to the task of supporting interactive routing aids for a specific, custom IC designer, a synergistic effect is achieved. The designer is pleased with the support he or she is receiving, while the CAD engineer sees the direct benefit of his or her work. The resulting tools designed in this way are generally far more useful than those designed from an initial "specification."

WINDOW: Simulated Annealing

Simulated annealing is a relatively new approach to combinatorial optimization problems. The results that have been obtained on a number of layout problems, from partitioning, to gate-array placement, from floor-planning to global routing, have been so interesting that a fairly detailed explanation is warranted here.

Heuristic algorithms are used to solve NP-complete problems approximately, i.e. to find "good" solutions which are "close" to the optimum. These algorithms explore a discrete space of admissible configurations, S, in a deterministic fashion. Starting from an initial configuration j_0, a sequence of configurations is selected and compared until a satisfactory one is found. The rules according to which a configuration is generated and the termination criteria, specify the algorithm. Often the search terminates at a local minimum, i.e. with a configuration $\hat{j}$ such that if we denote by $c(j)$ the cost of j and by $S(j)$ the set of configurations that can be generated from j by the algorithm in one step, $c(\hat{j}) \leq c(j), \forall j \in S(\hat{j})$. The local minimum reached can be quite far apart from the global minimum measuring their distance with the difference in cost. This is often due to the fact that heuristic algorithms are "greedy", i.e., only moves which reduce "maximally" the cost are accepted.

To avoid this behavior, randomizing algorithms can be devised which generate the next configuration randomly. The configuration is recorded as a new temporary solution if its cost is lower than the present temporary solution. The algorithm terminates after a certain number of moves. Randomizing algorithms perform well if the number of optimal solutions is fairly high, since the probability of stopping at an optimum is proportional to the ratio between the number of optimal configurations and the number of total configurations. Note that randomizing algorithms can "climb hills", i.e., they allow moves that generate configurations of higher cost than the present one are accepted.

Simulated annealing as proposed by Kirckpatrick et al. [KIR83], allows "hill climbing" moves but these moves are accepted according to a certain criterion which takes the cost into consideration and not blindly as randomizing algorithms. The controlling mechanism is based on the observation that combinatorial optimization problems with a large configuration space exhibit properties similar to physical processes with many degrees of freedom.

In particular, bringing a fluid into a low energy state such as growing a crystal, has been considered in [KIR83] similar to the process of finding an optimum solution of a combinatorial optimization problem. Annealing is a well-known process to grow crystals. It consists in melting the fluid and then lowering the temperature slowly until the crystal is formed. The rate of decrease of temperature has to be very low around the freezing temperature. The Metropolis Monte Carlo method can be used to simulate the annealing process. It has been proposed as an effective method for finding global minima of combinatorial optimization problems. This method when applied to combinatorial optimization generates moves randomly and checks whether the cost of the new configuration satisfies an acceptance criterion based on temperature. If the cost decreases, the move is accepted. If the cost increases, then a random number between zero and one is generated and compared with $f\ (\Delta c_{ij}, T) = \exp(\frac{-\Delta c_{ij}}{T})$ where Δc_{ij} is the change in cost obtained by moving from configuration i to j and T is temperature, the controlling parameter. If the random number is larger than f, the move is accepted, otherwise the move is discarded. Note that the higher the temperature is, the more likely it is that a "hill climbing" move is accepted. Note also that "hill climbing" moves are less and less probable as the temperature is decreased. A certain number of moves are generated and checked before a decrease in temperature is allowed. The initial temperature, the number of moves generated at each temperature and the rate of decrease of temperature are all important parameters that affect the speed of the algorithm and the quality of the final configuration. Experimental

results, e.g. [KIR82, VEC83, SEC84, OTT84], show that Simulated Annealing produces very good results when compared to other techniques for the solution of combinatorial optimization problems such as those arising from the layout of integrated circuits, at the expense of large computer time (a 1,500 standard cell placement problem can take as much as 24 hours of a VAX 11/780 [SEC84]).

A mathematical analysis of the algorithm is very important to understand the essential features which make the algorithm work well and to suggest techniques for controlling its operation. Markov chains can be used as a mathematical model of Simulated Annealing. It has been proved that under certain assumptions on the number of moves generated by the algorithm at each temperature, Simulated Annealing produces asymptotically the optimum solution of combinatorial optimization problems with probability one.

WINDOW: Heuristic Programming

Over the past two decades, many *algorithmic-based* tools have been developed for the analysis of ICs. For example, the simulators, layout-rule checkers, connectivity verifiers described in the text are all part of a broad-based IC design system. However, many of these tools must process a large number of 'special-cases' which are exceptions to the basic algorithm. It is often the processing of the special cases that dominates the run time, such as for connectivity verification described in Section 4. In other situations, a simple algorithm may not be known for a particular problem but a collection of simple heuristics may be used, such as for checking for more complex electrical rule violations in the design or for selecting a structure from a behavioral specification in the synthesis process. These problems are amenable to the relatively new field of heuristic programming.

Over the past few years, a number of both synthesis and verification tools have been developed which rely on the use of such an approach in the form of Expert Systems. In general, it is our conjecture that if a satisfactory algorithmic solution to a problem is known it should be used. When such a solution is not known, or where a problem has many "special cases" which dominate the run time of the algorithm, the problem is a good candidate for an expert-system-based approach. For many IC/CAD problems, a meta-system, involving the use of an expert system controlling the application of powerful, algorithmic tools, will probably provide the best solution.

Typical rule-based systems (e.g. [11]) are composed of three parts: the working memory, the rules, and the rule interpreter. The working memory, for most of the CAD applications described here, is the description of the circuit. The rules are condition-action pairs, where the conditions are patterns to match against the working memory or expressions to evaluation, and the actions are operations to perform (input/output, calculations and changes to the data) if the conditions are satisfied. The rule interpreter determines

which rule to fire based on the rules and the working memory contents; how the rule interpreter determines which rule to fire is known as conflict-resolution. A number of key areas where heuristic programming approaches have been applied successfully to CAD problems to date are outlined below:

- *Connectivity Verification*: As described in Section 4, connectivity verification is the process of comparing two circuit descriptions to make sure that they are the same and if they are not the same then to discover where they differ. Almost all connectivity verifiers can handle the straightforward problem very efficiently with fast, heuristic algorithms. However, most of the time in these programs is spent handling the special cases.

There are many special cases that can degrade the performance of the basic algorithms. Two such cases are *terminal permutability* and *parallel paths*. For some elements, the terminals are logically and/or electrically equivalent and are allowed to permute. The inputs to the basic logic gates (NAND, NOR, etc.) and the source and drain of MOSFETS are examples of such situations. In handling terminal permutability, many connectivity verifiers assume that they will be working with MOSFETS and "hard-wire" the fact that sources and drains can permute; others allow the user to specify how terminals on arbitrary elements can permute, but some do this very inefficiently and others do not always work.

Identical and almost identical parallel paths (as in bit-slice circuits and RAMs) also present a problem to current connectivity verifiers. If they are identical, the algorithms currently used can not distinguish between the paths and may not process them. Also, if two paths have only small differences (as in ROMs), since only local effects are taken into account, connectivity verifiers also may not be able to distinguish between them.

A connectivity verification technique using a rule-based system has been developed that handles the above special cases without the performance penalty seen in existing algorithmic systems. Whereas standard connectivity verifiers process circuits *locally*

(looking at individual elements and nodes, and their immediate neighbors), in the rule-based approach the circuit is processed in a global manner (collections of elements and nodes). The designer's hierarchy from the schematic is used to build patterns that match equivalent groups of elements in the layout. These patterns are rules for a rule-based system, with the circuit description being the data.

In the context of connectivity verification the conditions are patterns that match collections of elements (the subcircuit) and the actions are the removal of the individual elements from the working memory and the addition to the working memory of a subcircuit element.

- *Advanced Electrical Rules Checking:* A peer-group review is one of the major checkpoints of the VLSI design process. During this phase, a designer will have his work reviewed by other members of the project team. This review is usually performed by one or more experienced designers who study the schematics and layouts of a design, their mission being to "flush out bugs" that may have been overlooked by the original engineer and to provide feedback to him on his work. To date, this is an area where no effective algorithmic approach has been found short of complete simulation at the circuit level. Even then, the design critic may find errors that circuit simulators cannot detect.

The items turned up during this review cycle take many forms. In some cases they are very simple problems. Perhaps a new designer may not understand the implications of a design rule or needs some guidance as to the more practical ways of implementing a logic function. In other cases, extremely subtle problems are found which can elude even the most careful worst case simulations. Charge coupling, MOS capacitor inversion time constants, charge sharing on dynamic buses, voltage swings on bootstrapped nodes, and problems due to clock undershoot, overshoot, overlap and skew are some examples of the latter.

A number of *circuit design critic* programs have been developed over the past few years to perform these checks[refs]. CRITTER[4] and the DIALOG[5] systems which perform a design verification and review function; and the SCHEMA System [6] which is intended to act in the capacity of a complete design assistant, not limited to design verification alone.

● *Routing:*

As presented in Section 5.2.3.1, detailed routing is quite complicated if the region to route has pins on four side (two-dimensional routing problem). In addition, most of the available detailed routers are constrained to route vertical connections and horizontal connections on two different layers to simplify the structure of the algorithms. Many objective functions are used to evaluate the quality of the results by designers. For example, the number of vias used to complete the routing can be important to improve the reliability of the chip and net length is important to optimize the speed of the chip. Thus, the detailed routing problem in two-dimensions seems to be a natural application of expert systems. WEAVER [JOO85] is a knowledge-based system for detailed routing developed at CMU. The performance of the system is quite impressive in terms of the quality of the final solution, even though its running time is quite larger than the one required by standard algorithms. We believe that this system represents an important first step in the application of heuristic programming in the placement and routing area, however we believe that global routing, floor-planning and macro-cell placement should be a better test-bed for knowledge-based expert systems, since satisfactory solutions are already available with standard algorithmic techniques in the case of detailed routing.

Another important application of expert systems to routing is the system developed by S. Goto et al. at NEC. This system rips and re-routes nets to provide 100% completion on gate-array problems. This approach is quite interesting, because the algorithms used to route gate-arrays rarely provide 100% interconnection completion and several hours if not days of designer time are needed to complete the interconnections left out by the

algorithms. The NEC expert system uses a set of rules obtained by capturing the designer expertise in performing the rip and re-route operations. In addition, the expert system can call on algorithms to evaluate the application of a particular rule.

- *Multi-Level Logic Synthesis:*

As we pointed out in Section 5.3.2, one of the first systems for automatic optimization of multi-level logic, LSS of IBM, could be considered a prototypical rule-based system. Recently, a full-fledged rule-based expert system, Socrates, has been developed at GE Microelectronics Center to optimize combinational logic for a specific target technology [DEG85]. The system starts with a description of the logic to optimize and performs a series of algorithmic steps to produce a starting point to which a rule-based expert-system applies a series of local transformations. The knowledge-base can be easily enriched by the user through a rule generation module that automatically encodes new rules and inserts them in the knowledge base. A control module directs the application of the rules to the data. In particular, the module determines in which sequence the transformations are to be applied based on evaluations of the effectiveness of the transformation.

- *Behavioral Synthesis:*

A Rule-Based Expert System has been built at CMU by Kowalski to translate effectively a behavioral representation of a design into a structural representation. The rule base has been built by successive approximations asking designers to criticize designs that were obtained with the tool. The first rule set used contained 70 rules[THO83], the latest version of the tool contains more than 500 rules[DIR85].

The conclusion of the CMU researchers has been that while the algorithmic approach can be more effective if the cost function is well specified, the expert system approach offers more flexibility and a better environment to capture the often difficult to express intent of the designer.

WINDOW: Multiprocessors for CAD

As we approach the fundamental performance limits of uniprocessors, it is clear that only new, multiprocessor computer architectures can offer the large performance improvements needed to solve the complex problems of our time. Throughout the world, a great deal of research is in progress on the development of new, often unconventional, computer architectures for both symbolic and numerical processing [38].

However, such machines will not be able to exploit the parallelism available in problems unless new algorithms are developed that are well suited to a multiprocessor environment. In the past, it has often been assumed that advanced compiler technology would be sufficient to translate a conventional computer algorithm for optimal use on a special-purpose machine. In the case of circuit simulation, even the most advanced compiler technology, used in conjunction with a number of computer consultants, has shown poor speedup on pipelined machines (about 12%-15% hardware utilization on the Cray 1 [9]).

There are several compute-intensive problems that face CAD of VLSI circuits today. In some cases, such as logic simulation[] and design-rule checking[], new special-purpose machines have been designed to fully exploit the characteristics of the problem to be solved. In others, such as circuit simulation[], existing general purpose parallel processors have been used to speed up the simulation time[]. A careful analysis of the economic trade-offs has to be made to decide whether to build a special-purpose machine or to use existing multi-processors.

It is clear that the cost of the processors would be much lower for the case of general-purpose multi-processors, while the performance of special-purpose machines could be orders of magnitude better. A "mixed" approach could probably offer the advantages of both strategies. In this case, the interconnection network and the basic software

should be the ones offered by the general-purpose environment, while the particular compute-intensive tasks to be carried out by the single processors could be sped up by the design of special-purpose accelerators board to be used as co-processors for the processors of the general-purpose system.

Several approaches have been followed in the past few years to develop CAD accelerators. An excellent review of the field can be found in []. Here we will mention only a few relevant projects.

Logic simulation was the first application of hardware accelerators to IC CAD. The first working prototype proposed for this task was the Boeing Computer Simulator[]. However, the first machine actually used in the design of digital system was the Logic Simulation Machine (LSM)[] developed jointly by IBM T. J. Watson Research Center and IBM Los Gatos. The Yorktown Simulation Engine (YSE) developed by the IBM T.J. Watson Research Center has similar architecture but better performance. NEC developed a hardware logic simulator, HAL []. A Wire-Routing Machine has been proposed and implemented by a group of researchers at IBM T.J. Watson Research Center. An interesting architecture resembling the MIT connection machine, was designed by NTT to speed up a variety of CAD algorithms[].

In the vendors' arena, Zycad has developed a fast, but expensive, special-purpose machine for logic simulation and is investigating special-purpose machines for circuit simulation. Daisy and Valid Logic Systems have developed a cheaper, but slower, logic simulation engine . The architecture developed for this task by Daisy has also been used to speed up the simulated annealing algorithm applied to the problem of optimally placing gate-arrays. Shiva multi-systems is offering hardware accelerators for circuit simulation by extending the capabilities of a commercial multiprocessor, the Sequent Balance 8000[ref].

In academia, several researchers have proposed new architectures and corresponding CAD algorithms. A special purpose machine for design rule checking was proposed by MIT[]. A Virtual Bit Map Processor was designed at Stanford to provide hardware support for operations involving bit map representations[]. Similar techniques have been used in [] to support physical design.

Systolic machines have been proposed by Kung at Carnegie-Mellon University for a variety of numerical algorithms, from the solution of linear algebraic equations to the computation of Fast Fourier Transforms, and the fast implementation of simulated annealing for printed circuit board placement.

Other architectures have been proposed for the solution of linear system of algebraic equations and in particular for sparse matrices[]. However, no working prototype has been built because of the complexity of these machines. Algorithms for existing multiprocessors are also being studied[]. Other numerical problems have been investigated, for example research has been carried out on parallel solution of partial differential equations at Maryland [rheiboldt].

Parallel algorithms have been investigated for combinatorial optimization problems in a number of computer science departments from the theoretical point of view.[] These projects have a direct impact on CAD because many of the optimization problems involved in the development of effective tools are combinatorial. In addition, the methodology developed in this research to evaluate parallel algorithms can be used as a guideline for the development of new parallel algorithms.

We can classify the new research areas opened by the feasibility of designing special-purpose processors as well as by the availability of general-purpose multiprocessors, into three areas: development of new algorithms for existing multi-processor architectures, for example [] and [], development of new architectures for existing algorithms, for example [] and [], and development of new algorithms and new architectures.

for example[]. It is important to note that when using non-conventional architectures existing algorithms which are considered less efficient when using conventional uniprocessors, may become much faster. For example, in the case of the LSM[] and YSE[], event driven algorithms widely recognized as the most efficient algorithms for logic simulation on a uniprocessor, have not been used to exploit the particular architecture of the special purpose machine.

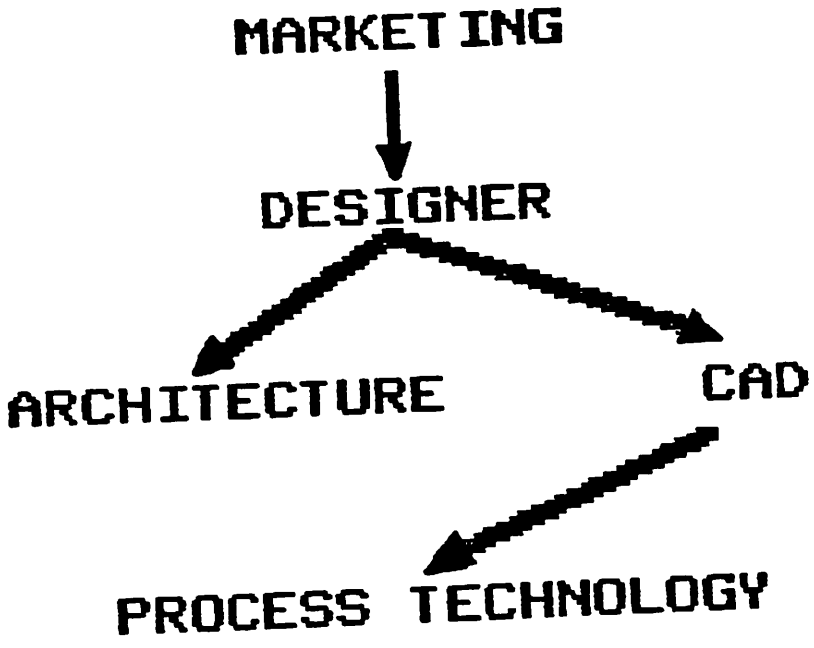


Fig. 1. Simplified View of ASIC Designer's Task.

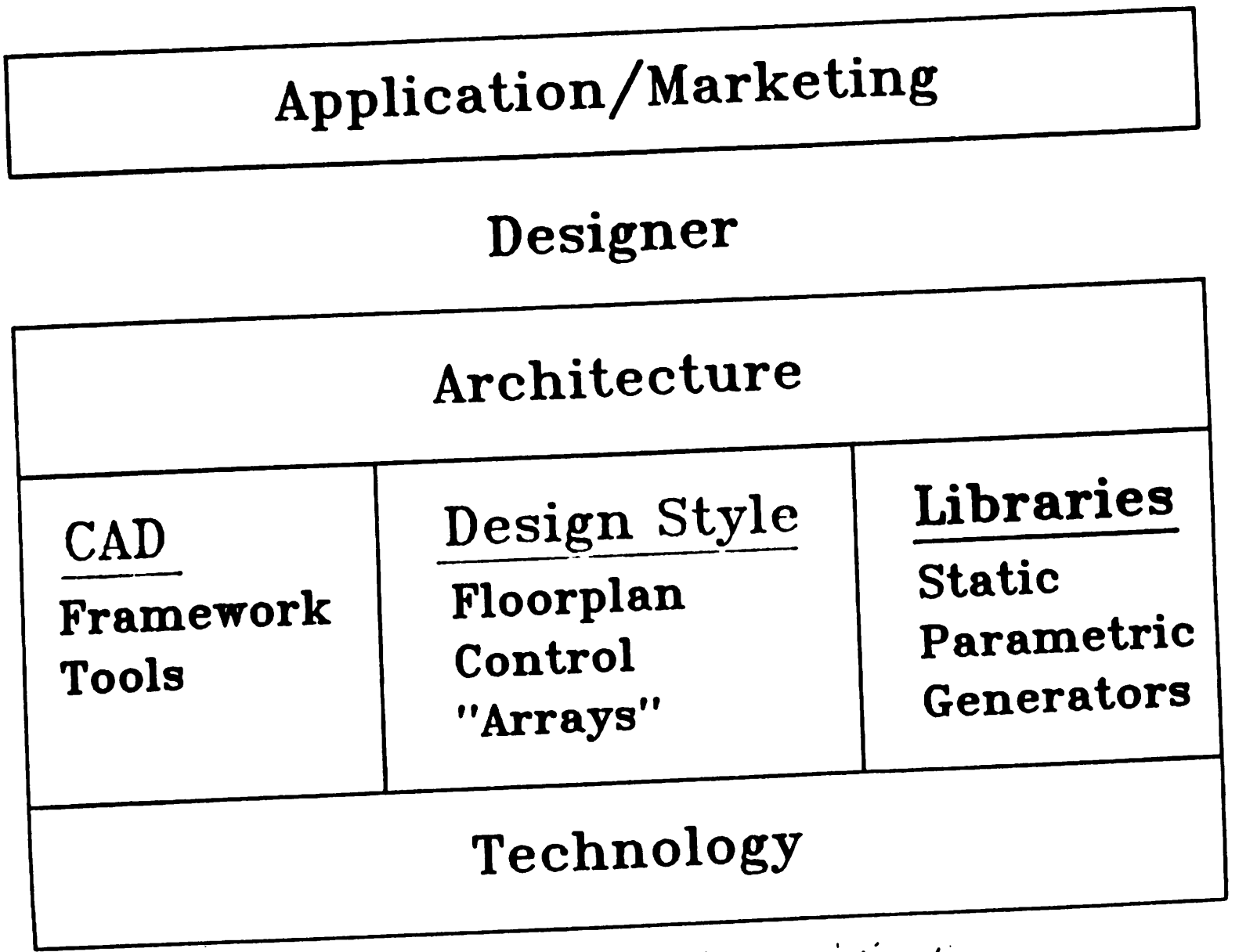


Fig. 2. More Detailed Classification
[illegible] Designers' Tools.

WINDOW: Heuristic Programming

Over the past two decades, many *algorithmic-based* tools have been developed for the analysis of ICs. For example, the simulators, layout-rule checkers, connectivity verifiers described in the text are all part of a broad-based IC design system. However, many of these tools must process a large number of 'special-cases' which are exceptions to the basic algorithm. It is often the processing of the special cases that dominates the run time, such as for connectivity verification described in Section 4. In other situations, a simple algorithm may not be known for a particular problem but a collection of simple heuristics may be used, such as for checking for more complex electrical rule violations in the design or for selecting a structure from a behavioral specification in the synthesis process. These problems are amenable to the relatively new field of heuristic programming.

Over the past few years, a number of both synthesis and verification tools have been developed which rely on the use of such an approach in the form of Expert Systems. In general, it is our conjecture that if a satisfactory algorithmic solution to a problem is known it should be used. When such a solution is not known, or where a problem has many "special cases" which dominate the run time of the algorithm, the problem is a good candidate for an expert-system-based approach. For many IC/CAD problems, a meta-system, involving the use of an expert system controlling the application of powerful, algorithmic tools, will probably provide the best solution.

Typical rule-based systems (e.g. [11]) are composed of three parts: the working memory, the rules, and the rule interpreter. The working memory, for most of the CAD applications described here, is the description of the circuit. The rules are condition-action pairs, where the conditions are patterns to match against the working memory or expressions to evaluation, and the actions are operations to perform (input/output, calculations and changes to the data) if the conditions are satisfied. The rule interpreter determines

which rule to fire based on the rules and the working memory contents; how the rule interpreter determines which rule to fire is known as conflict-resolution. A number of key areas where heuristic programming approaches have been applied successfully to CAD problems to date are outlined below:

- *Connectivity Verification*: As described in Section 4, connectivity verification is the process of comparing two circuit descriptions to make sure that they are the same and if they are not the same then to discover where they differ. Almost all connectivity verifiers can handle the straightforward problem very efficiently with fast, heuristic algorithms. However, most of the time in these programs is spent handling the special cases.

There are many special cases that can degrade the performance of the basic algorithms. Two such cases are *terminal permutability* and *parallel paths*. For some elements, the terminals are logically and/or electrically equivalent and are allowed to permute. The inputs to the basic logic gates (NAND, NOR, etc.) and the source and drain of MOSFETS are examples of such situations. In handling terminal permutability, many connectivity verifiers assume that they will be working with MOSFETS and "hard-wire" the fact that sources and drains can permute; others allow the user to specify how terminals on arbitrary elements can permute, but some do this very inefficiently and others do not always work.

Identical and almost identical parallel paths (as in bit-slice circuits and RAMs) also present a problem to current connectivity verifiers. If they are identical, the algorithms currently used can not distinguish between the paths and may not process them. Also, if two paths have only small differences (as in ROMs), since only local effects are taken into account, connectivity verifiers also may not be able to distinguish between them.

A connectivity verification technique using a rule-based system has been developed that handles the above special cases without the performance penalty seen in existing algorithmic systems. Whereas standard connectivity verifiers process circuits *locally*

(looking at individual elements and nodes, and their immediate neighbors), in the rule-based approach the circuit is processed in a global manner (collections of elements and nodes). The designer's hierarchy from the schematic is used to build patterns that match equivalent groups of elements in the layout. These patterns are rules for a rule-based system, with the circuit description being the data.

In the context of connectivity verification the conditions are patterns that match collections of elements (the subcircuit) and the actions are the removal of the individual elements from the working memory and the addition to the working memory of a subcircuit element.

• *Advanced Electrical Rules Checking:* A peer-group review is one of the major checkpoints of the VLSI design process. During this phase, a designer will have his work reviewed by other members of the project team. This review is usually performed by one or more experienced designers who study the schematics and layouts of a design, their mission being to "flush out bugs" that may have been overlooked by the original engineer and to provide feedback to him on his work. To date, this is an area where no effective algorithmic approach has been found short of complete simulation at the circuit level. Even then, the design critic may find errors that circuit simulators cannot detect.

The items turned up during this review cycle take many forms. In some cases they are very simple problems. Perhaps a new designer may not understand the implications of a design rule or needs some guidance as to the more practical ways of implementing a logic function. In other cases, extremely subtle problems are found which can elude even the most careful worst case simulations. Charge coupling, MOS capacitor inversion time constants, charge sharing on dynamic buses, voltage swings on bootstrapped nodes, and problems due to clock undershoot, overshoot, overlap and skew are some examples of the latter.

A number of *circuit design critic* programs have been developed over the past few years to perform these checks[refs]. CRITTER[4] and the DIALOG[5] systems which perform a design verification and review function; and the SCHEMA System [6] which is intended to act in the capacity of a complete design assistant, not limited to design verification alone.

● *Routing:*

As presented in Section 5.2.3.1, detailed routing is quite complicated if the region to route has pins on four side (two-dimensional routing problem). In addition, most of the available detailed routers are constrained to route vertical connections and horizontal connections on two different layers to simplify the structure of the algorithms. Many objective functions are used to evaluate the quality of the results by designers. For example, the number of vias used to complete the routing can be important to improve the reliability of the chip and net length is important to optimize the speed of the chip. Thus, the detailed routing problem in two-dimensions seems to be a natural application of expert systems. WEAVER [JOO85] is a knowledge-based system for detailed routing developed at CMU. The performance of the system is quite impressive in terms of the quality of the final solution, even though its running time is quite larger than the one required by standard algorithms. We believe that this system represents an important first step in the application of heuristic programming in the placement and routing area, however we believe that global routing, floor-planning and macro-cell placement should be a better test-bed for knowledge-based expert systems, since satisfactory solutions are already available with standard algorithmic techniques in the case of detailed routing.

Another important application of expert systems to routing is the system developed by S. Goto et al. at NEC. This system rips and re-routes nets to provide 100% completion on gate-array problems. This approach is quite interesting, because the algorithms used to route gate-arrays rarely provide 100% interconnection completion and several hours if not days of designer time are needed to complete the interconnections left out by the

algorithms. The NEC expert system uses a set of rules obtained by capturing the designer expertise in performing the rip and re-route operations. In addition, the expert system can call on algorithms to evaluate the application of a particular rule.

● *Multi-Level Logic Synthesis:*

As we pointed out in Section 5.3.2, one of the first systems for automatic optimization of multi-level logic, LSS of IBM, could be considered a prototypical rule-based system. Recently, a full-fledged rule-based expert system, Socrates, has been developed at GE Microelectronics Center to optimize combinational logic for a specific target technology [DEG85]. The system starts with a description of the logic to optimize and performs a series of algorithmic steps to produce a starting point to which a rule-based expert-system applies a series of local transformations. The knowledge-base can be easily enriched by the user through a rule generation module that automatically encodes new rules and inserts them in the knowledge base. A control module directs the application of the rules to the data. In particular, the module determines in which sequence the transformations are to be applied based on evaluations of the effectiveness of the transformation.

● *Behavioral Synthesis*:

A Rule-Based Expert System has been built at CMU by Kowalski to translate effectively a behavioral representation of a design into a structural representation. The rule base has been built by successive approximations asking designers to criticize designs that were obtained with the tool. The first rule set used contained 70 rules[THO83], the latest version of the tool contains more than 500 rules[DIR85].

The conclusion of the CMU researchers has been that while the algorithmic approach can be more effective if the cost function is well specified, the expert system approach offers more flexibility and a better environment to capture the often difficult to express intent of the designer.

WINDOW: Multiprocessors for CAD

As we approach the fundamental performance limits of uniprocessors, it is clear that only new, multiprocessor computer architectures can offer the large performance improvements needed to solve the complex problems of our time. Throughout the world, a great deal of research is in progress on the development of new, often unconventional, computer architectures for both symbolic and numerical processing [38].

However, such machines will not be able to exploit the parallelism available in problems unless new algorithms are developed that are well suited to a multiprocessor environment. In the past, it has often been assumed that advanced compiler technology would be sufficient to translate a conventional computer algorithm for optimal use on a special-purpose machine. In the case of circuit simulation, even the most advanced compiler technology, used in conjunction with a number of computer consultants, has shown poor speedup on pipelined machines (about 12%-15% hardware utilization on the Cray 1 [9]).

There are several compute-intensive problems that face CAD of VLSI circuits today. In some cases, such as logic simulation[] and design-rule checking[], new special-purpose machines have been designed to fully exploit the characteristics of the problem to be solved. In others, such as circuit simulation[], existing general purpose parallel processors have been used to speed up the simulation time[]. A careful analysis of the economic trade-offs has to be made to decide whether to build a special-purpose machine or to use existing multi-processors.

It is clear that the cost of the processors would be much lower for the case of general-purpose multi-processors, while the performance of special-purpose machines could be orders of magnitude better. A "mixed" approach could probably offer the advantages of both strategies. In this case, the interconnection network and the basic software

should be the ones offered by the general-purpose environment, while the particular compute-intensive tasks to be carried out by the single processors could be sped up by the design of special-purpose accelerators board to be used as co-processors for the processors of the general-purpose system.

Several approaches have been followed in the past few years to develop CAD accelerators. An excellent review of the field can be found in []. Here we will mention only a few relevant projects.

Logic simulation was the first application of hardware accelerators to IC CAD. The first working prototype proposed for this task was the Boeing Computer Simulator[]. However, the first machine actually used in the design of digital system was the Logic Simulation Machine (LSM)[] developed jointly by IBM T. J. Watson Research Center and IBM Los Gatos. The Yorktown Simulation Engine (YSE) developed by the IBM T.J. Watson Research Center has similar architecture but better performance. NEC developed a hardware logic simulator, HAL []. A Wire-Routing Machine has been proposed and implemented by a group of researchers at IBM T.J. Watson Research Center. An interesting architecture resembling the MIT connection machine, was designed by NTT to speed up a variety of CAD algorithms[].

In the vendors' arena, Zycad has developed a fast, but expensive, special-purpose machine for logic simulation and is investigating special-purpose machines for circuit simulation. Daisy and Valid Logic Systems have developed a cheaper, but slower, logic simulation engine . The architecture developed for this task by Daisy has also been used to speed up the simulated annealing algorithm applied to the problem of optimally placing gate-arrays. Shiva multi-systems is offering hardware accelerators for circuit simulation by extending the capabilities of a commercial multiprocessor, the Sequent Balance 8000[ref].

In academia, several researchers have proposed new architectures and corresponding CAD algorithms. A special purpose machine for design rule checking was proposed by MIT[]. A Virtual Bit Map Processor was designed at Stanford to provide hardware support for operations involving bit map representations[]. Similar techniques have been used in [] to support physical design.

Systolic machines have been proposed by Kung at Carnegie-Mellon University for a variety of numerical algorithms, from the solution of linear algebraic equations to the computation of Fast Fourier Transforms, and the fast implementation of simulated annealing for printed circuit board placement.

Other architectures have been proposed for the solution of linear system of algebraic equations and in particular for sparse matrices[]. However, no working prototype has been built because of the complexity of these machines. Algorithms for existing multiprocessors are also being studied[]. Other numerical problems have been investigated, for example research has been carried out on parallel solution of partial differential equations at Maryland [rheiboldt].

Parallel algorithms have been investigated for combinatorial optimization problems in a number of computer science departments from the theoretical point of view.[] These projects have a direct impact on CAD because many of the optimization problems involved in the development of effective tools are combinatorial. In addition, the methodology developed in this research to evaluate parallel algorithms can be used as a guideline for the development of new parallel algorithms.

We can classify the new research areas opened by the feasibility of designing special-purpose processors as well as by the availability of general-purpose multiprocessors, into three areas: development of new algorithms for existing multi-processor architectures, for example [] and [], development of new architectures for existing algorithms, for example [] and [], and development of new algorithms and new architectures.

for example[]. It is important to note that when using non-conventional architectures, existing algorithms which are considered less efficient when using conventional uniprocessors, may become much faster. For example, in the case of the LSM[] and YSE[], event driven algorithms widely recognized as the most efficient algorithms for logic simulation on a uniprocessor, have not been used to exploit the particular architecture of the special purpose machine.

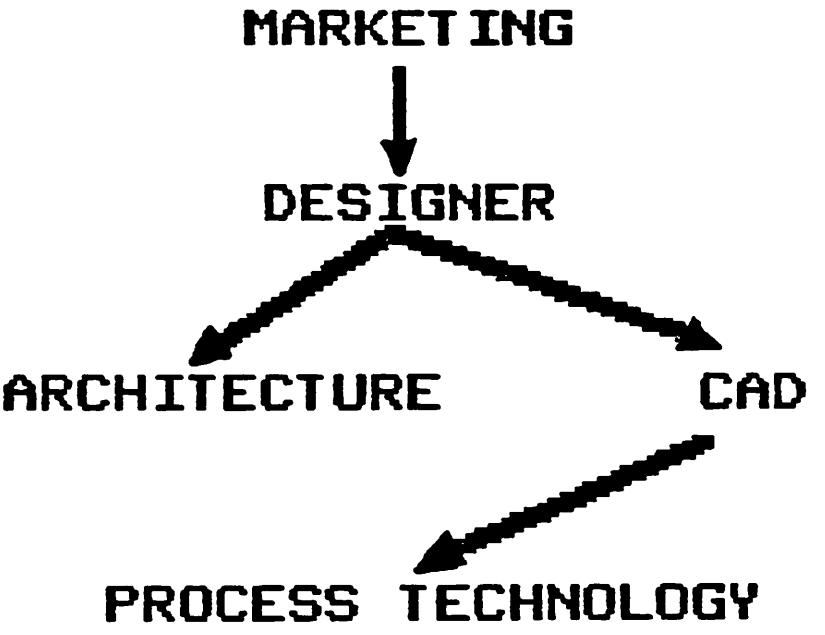


Fig. 1. Simplified View of ASIC Designer's Task.

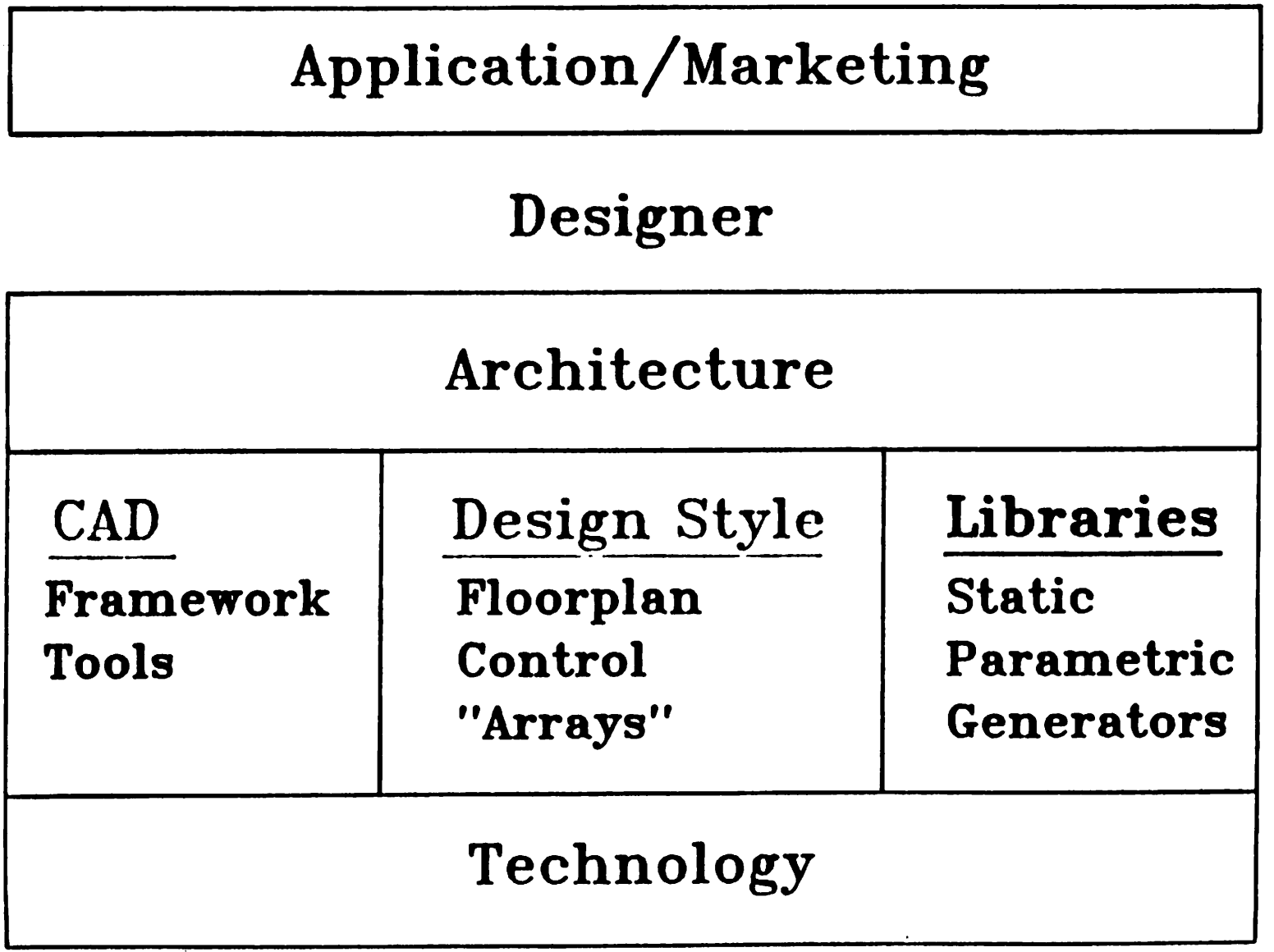


Fig. 2. More Detailed Classification
... Design Tools.

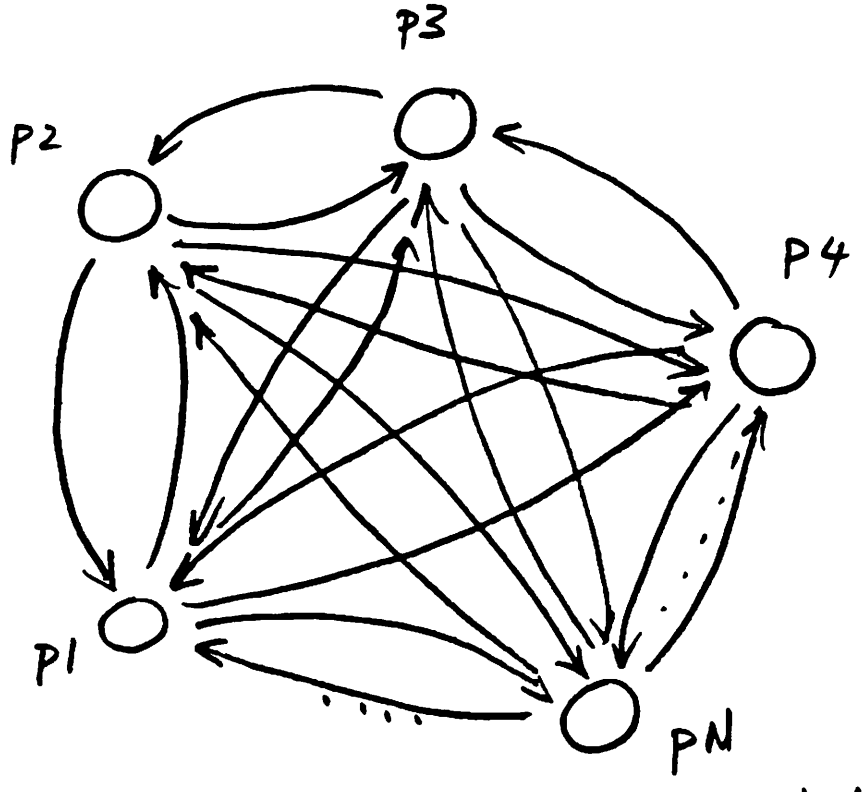


Fig. 3.1(a) Translators among N tools, P_i, $i = 1 \ldots N$.

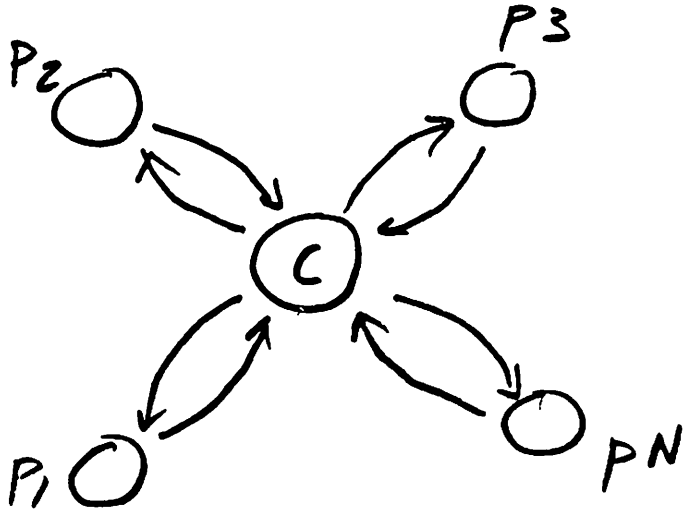


Fig. 3.1(b) Translators among N tools using Common format, C.

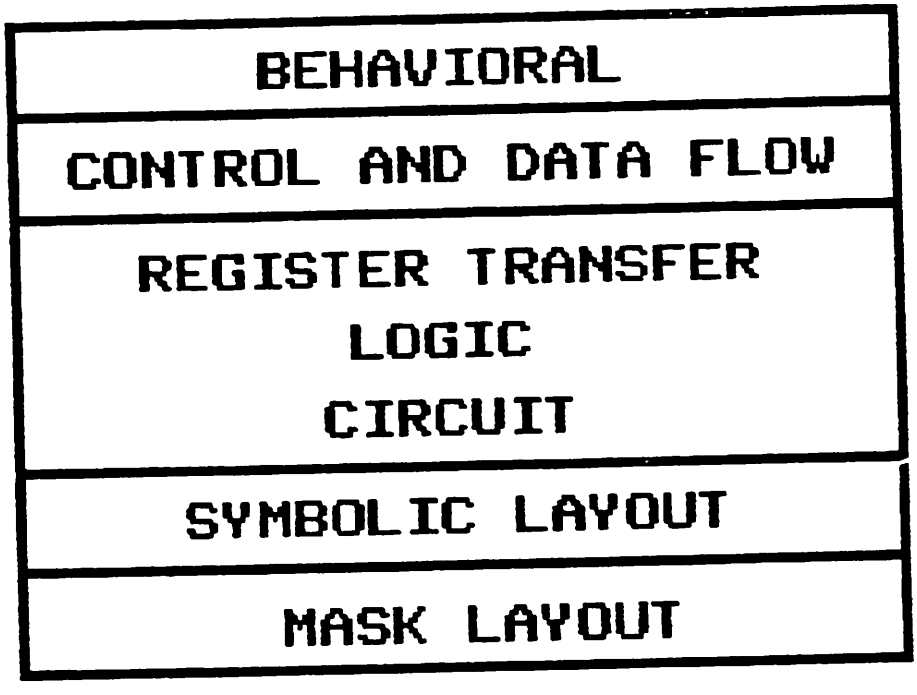
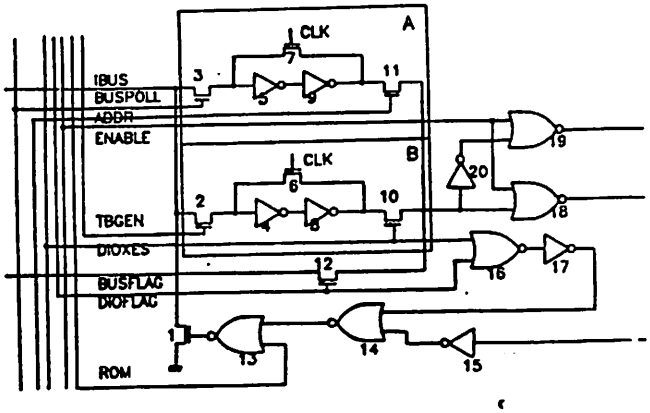


Fig. 3.2 Levels of Design Description



-3.3(A) schematic diagram

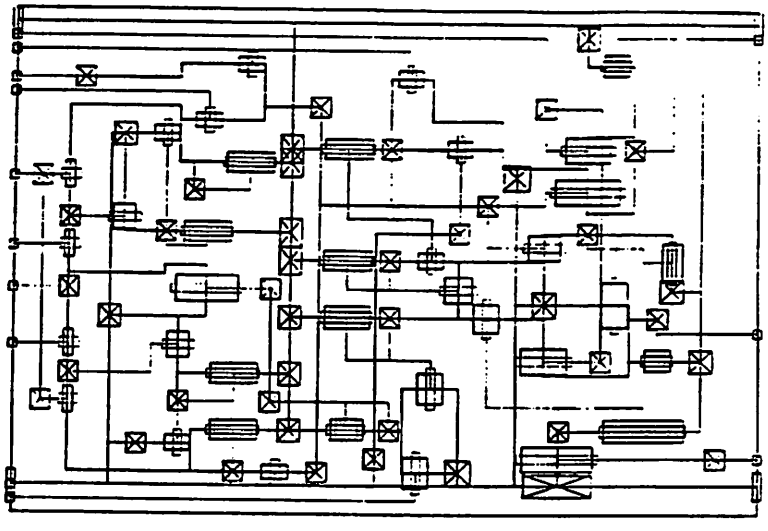


Figure 3.3(b) The symbolic layout plan of the complete latch-driver blo

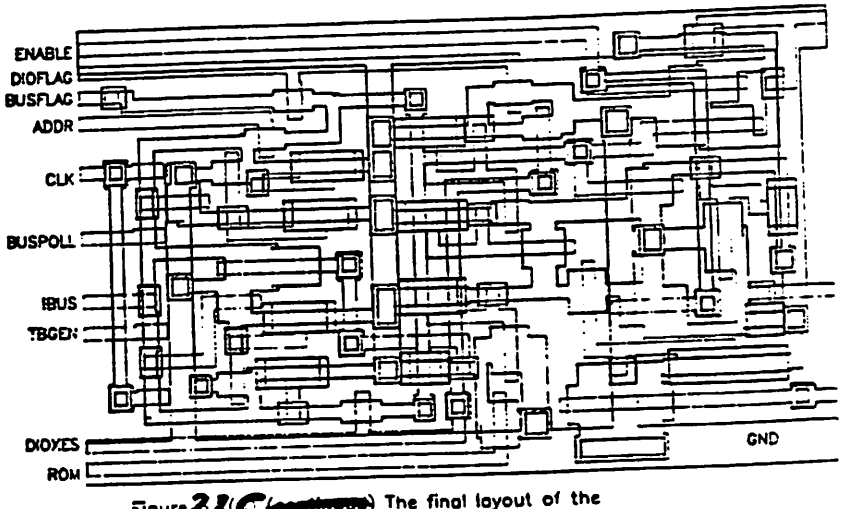
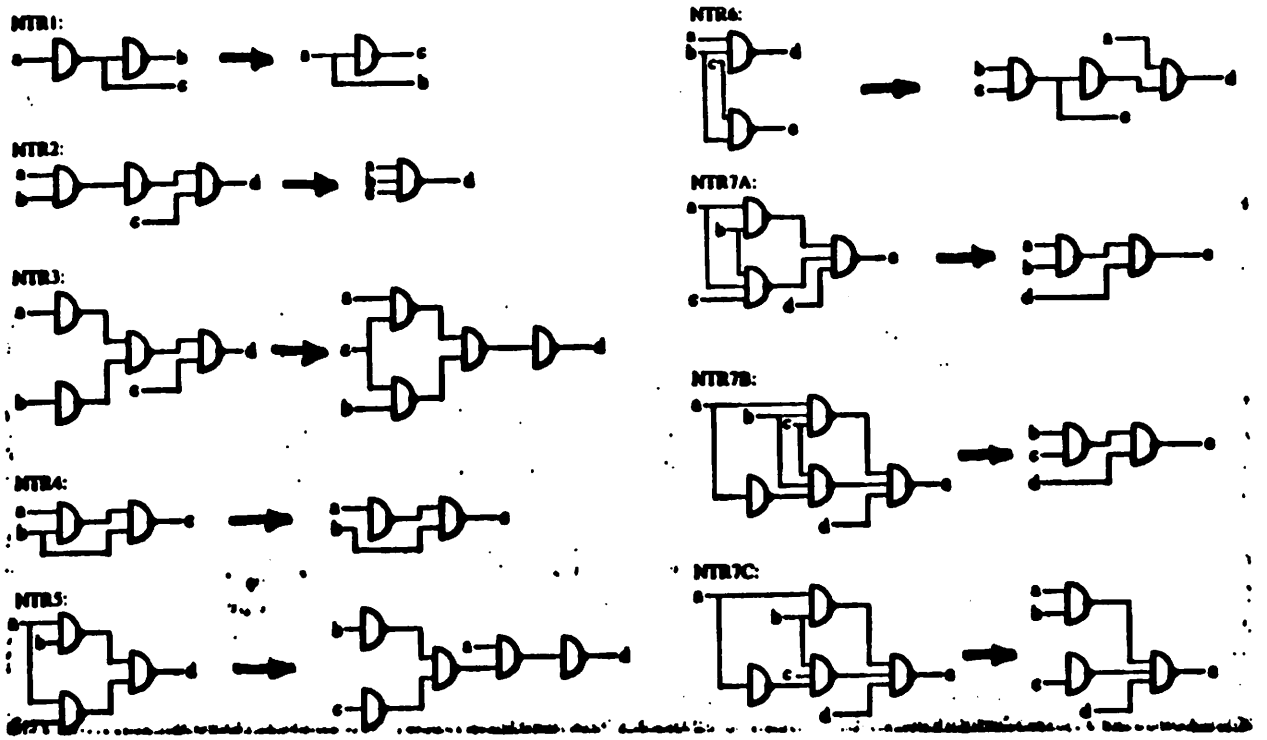


Figure 3.3(c) ~~(continued)~~ The final layout of the latch-driver block generated with CABBAGE.

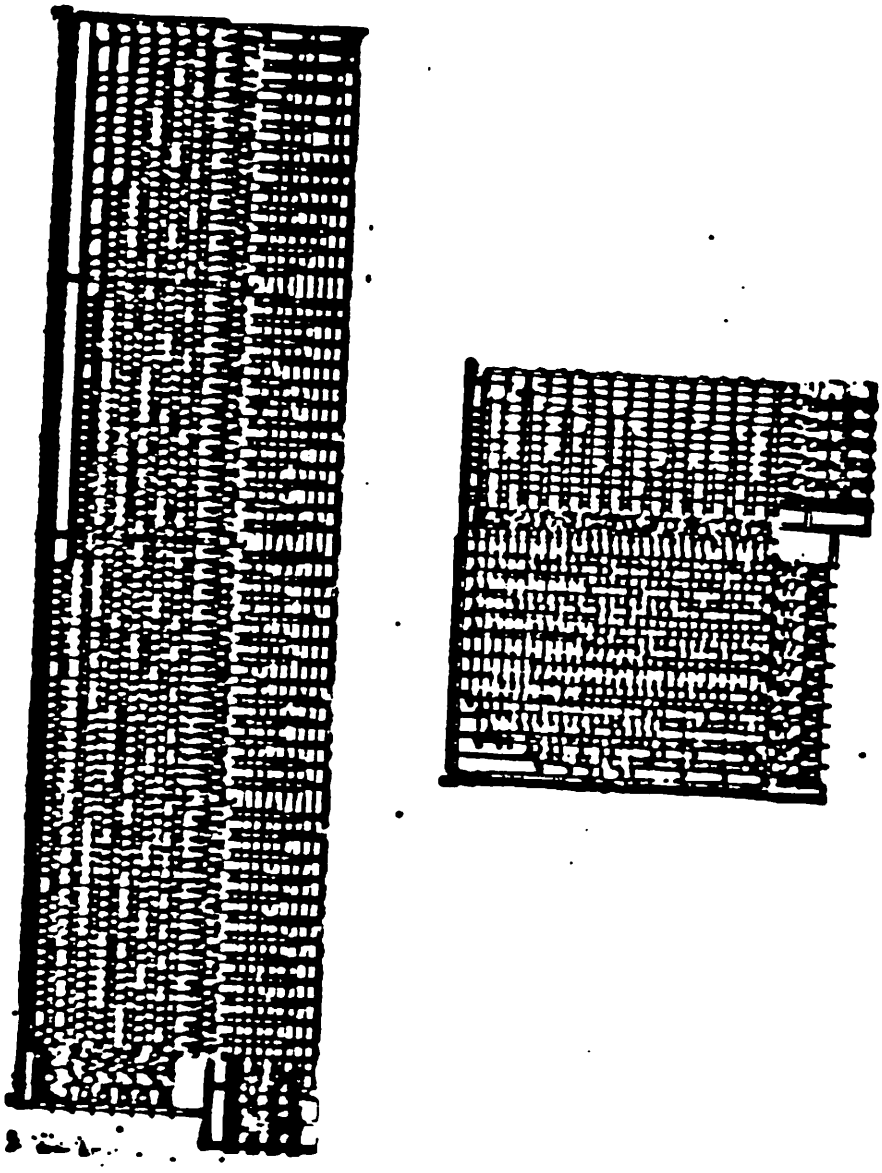
Rule-based Approach

LSS

- Local, "peephole" transformations



5.3.2 Fig. 3



5.3.1.2

Figure 2. FSM Synthesis using KISS: (a) The hand-designed circuit (b) After KISS redesign.

Fig 3.4 will be screen shot (color) of state-of-the-art workstation.