AN ABSTRACT OF THE THESIS OF

Daniel Lim for the degree of Master of Science in Electrical and Computer Engineering presented on May 3, 1985.

Title: VLSI Design Methodologies and Computer Tools

Abstract approved: Redacted for Privacy

Dr. Megha Shyam

The rapid development of semiconductor technology and the increasing complexity of VLSI chips have prompted both the industry and the academic community alike to take an indepth look at the VLSI design problem. Many design methodologies have been proposed and associated computer-aided tools developed. This thesis is a study of current design methodologies including some of the computer tools developed to support these methodologies. The concept of a VLSI design space was presented and some design concepts were discussed. The design methodologies looked at included Computer-Aided Design (CAD) systems, Expert Systems and Design Automation systems.

VLSI DESIGN METHODOLOGIES AND COMPUTER TOOLS

by

Daniel Lim

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed May 3, 1985

Commencement June 1985

APPROVED:

Redacted for Privacy

Professor of Electrical and Computer Engineering in charge of major

Redacted for Privacy

Head of department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented May 3, 1985

ACKNOWLEDGEMENTS

My sincere gratitude is expressed to Dr. Megha Shyam for his encouragement, guidance and assistance throughout the course of this study. He has been a major source of motivation in my graduate program at Oregon State University.

Special thanks are extended to Dr. Ragu Raghavan of Mentor Graphics Corporation for his time and effort in showing me their workstations and VLSI design tools.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The VLSI era has finally arrived. VLSI technology is more a statement of the circuit complexity attainable on a chip than the actual transistor count. Since the time of its inception, integrated circuit complexity has grown at an exponential rate. The number of transistors in the circuit gives a measure of circuit complexity. According to G.E. Moore [1], the number of transistors that could be placed on a chip have been doubling approximately every year as depicted in Figure 1.1. However, since the beginning of this decade (1980's), this growth rate has slowed down to doubling approximately every two years.

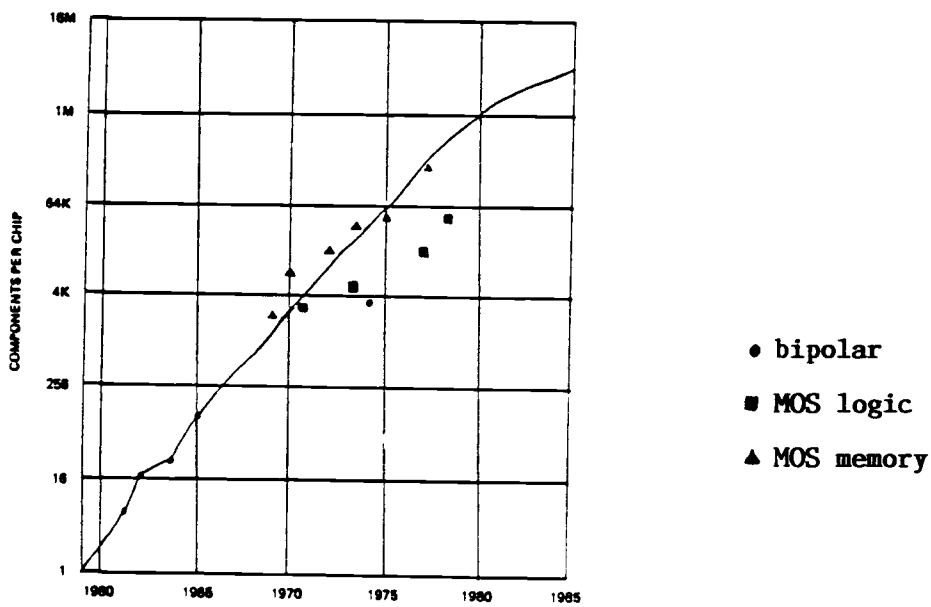


Fig. 1.1: The growth of IC technology

A number of factors have contributed to this exponential growth in circuit density [2]. The major contributor has been technological advances, two of which are worthy of mention. A serious effort has been made at reducing the feature size of present day circuits. Refinements in photolithographic procedures have enabled the minimum feature dimension to be reduced significantly. Submicron feature dimensions have been demonstrated in the laboratory through the use of X-ray lithography and electron exposure systems [2]. A second area is the advanced developments in process technology which have managed to reduce the defect density considerably. Thus making it possible to increase chip size without suffering a significant loss in overall yield.

Traditionally, the factors limiting IC design advancements have been the physical and technological limits. The physical limits pertain to device physics of the materials used. And the technological limits relate to the fabrication of VLSI devices repeatably and economically. However, with this explosive advancement in VLSI technology, it now has become feasible to design whole digital systems on a chip with high levels of complexity. The mind-boggling complexity that designers face is now becoming a major limiting factor to contend with. The physical and technological limitations may impose absolute limits on the growth of IC complexity, but not for at least another decade [3]. Ultimately, it will be the complexity barrier that will constrain the growth of IC complexity.

The continual desire to shrink device dimensions to the sub-micron level has led to a rather disturbing increase in the design cycle time. It will take longer to design and develop these complex chips and according to [4], with present design methodologies, a 100,000 transistor MOS chip will take approximately 60 man-years to lay out and another 60 to debug. This is unacceptable to most manufacturers because the competitive nature of the VLSI market does not permit it. The solution does not lie with simply putting more people on the job - this only lends itself

to increased difficulties in communicating between design groups and in itself promotes inefficiency due to the design communication barrier.

The need to overcome this VLSI design problem has caught the attention of both the industry and the academic community alike; and this has led to numerous proposals in design methodologies and design tools being put forth and developed in the last couple of years.

1.1 PURPOSE

The sudden introduction of all these design methodologies and computer design tools has definitely been overwhelming. Four years ago, companies specializing in VLSI computer-aided engineering work stations did not exist [Fortune, June 11, 1984], but last year (1984) the CAE business was expected to rack up sales of more than $200 million and is expected to grow at a blistering rate of 65% annually to top more than $1 billion by 1988. At the rate at which all these new design methodologies and computer design tools are being thrust at the VLSI community, there appears to be a need to take an indepth look at all that has transpired in the last couple of years and evaluate the strengths and weaknesses of such design methodologies and design tools. This research is geared towards a study of existing VLSI design methodologies and provides a cross-sectional look at the CAD tools presently available to VLSI system designers.

1.2 OVERVIEW

In such a fast developing field as VLSI system design, it is inherently difficult to provide an absolute coverage of all existing design methodologies and design systems. The design methodologies that

are in use in industry today span such a broad spectrum that studying all of them would prove to be beyond the scope of this research. Generally, they may be classified under the following headings:

1. Computer-Aided Design (CAD) Systems
2. Expert Systems
3. Design Automation (DA) Systems

At the lower extreme are the computer-aided design systems such as graphic editors and design verification tools. These CAD tools are often integrated through a database. CAD methodologies support the belief that design decisions should be made solely by the experienced human designer. Then there are the expert systems that are evolving rapidly and these can be classified as systems that have a certain degree of artificial intelligence in them. At the top most extreme of the design spectrum are the design automation systems such as automatic placement and routing tools; PLA and RAM generators; and silicon compilers. These systems approach the design problem from an algorithmic manner and the logic and final layout of the design is generated automatically by such compilers.

Chapter Two of this thesis first looks at the conventional VLSI design approach. Through a study of its strengths and weaknesses, new design concepts are recognized that better meet the needs of the VLSI environment.

In Chapter Three, the VLSI design space is introduced and some VLSI design concepts are discussed. The three main categories of the design spectrum mentioned above and representative systems that are in use presently will then be studied. There are of course systems that span the categories and these will be viewed accordingly.

Chapter Four looks at the trend of things to come in the VLSI system design scene.

CHAPTER 2

A CONVENTIONAL DESIGN METHODOLOGY

In the pioneering days of integrated circuit design, designers worked with unlimited freedom to achieve the best possible results because of severe technological limitations imposed on them [10]. The potentials and the unknowns of the new technology demanded unlimited freedom. There were no strict design guidelines to hinder their creative freedom. This trend of design methodology, which emerged from the SSI and MSI period, has carried over to present-day design methodologies. It is this unlimited freedom in the design environment that has led to difficulties and an increase in design time in dealing with the highly complex nature of today's VLSI circuits.

Although there did not exist one fixed design methodology in the early days, there was an obvious mainstream design flow that characterized the design methods of that time [5],[9]. It is this design flow that is referred to as the conventional design methodology. In this chapter, the design stages outlined in Fig.2.1 are discussed and a critique on the strengths and weaknesses of this conventional design method will be presented.

2.1 SYSTEM DEFINITION

The first step in the design process involves defining the system (chip) to be designed. Viewing the design process from a hierarchical approach, this level is the highest level and may involve high-level architectural design. For example, in the design of a peripheral support chip in a digital system, this may involve specifying communications protocol with other chips in the digital system; as well as defining performance, timing and funtional characteristics of the chip. Although

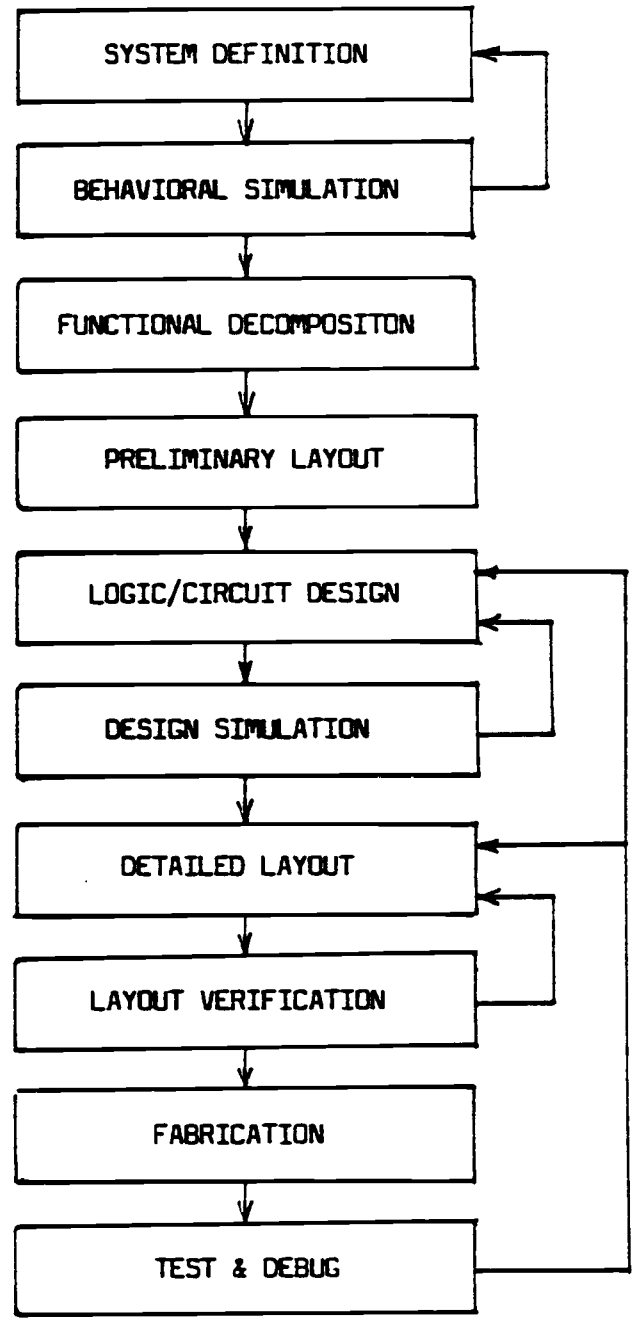


Fig. 2.1: A Conventional Design Methodology

this step involves a high level of abstraction, it is crucial that the design definition be as accurate as possible; since the performance characteristics of the final product depend heavily on this first design step. In the mid-70's, this step may not have involved much specification by virtue of the chip complexity attainable at the time. However, in this present decade, at the level of complexity achievable, designers are faced with system level definitions because they are no longer merely designing single function chips but rather they are now designing whole digital systems on a single chip.

2.2 BEHAVIORAL SIMULATION

In the early beginnings of IC design, there did not exist much behavioral simulation in the design process. This again was due to the relatively simple nature of the chips designed at that time. However, in recent years, as the degree of integration and complexity increased, there appeared a need to simulate the workings of the chip at this level before any actual circuit transistors were specified. This is often done in a high-level simulation language and provides the designer with a means of evaluating the overall functional aspects of the chip. Examples of high-level general-purpose simulation languages are GPSS [7] and SIMSCRIPT [8]. These deal mainly with a timing analysis of the system modeled in terms of subsystem components; such as ALU's, memory modules and peripherals [6].

2.3 FUNCTIONAL DECOMPOSITION

This stage involves partitioning the high-level design into functionally distinct elements. Partitions are defined such that the amount of information transferred bewteen functional elements is minimized. By

partitioning the high-level design into functionally distinct elements, the designers are able to deal with each functional element as a separate subproblem that can be individually verified for design correctness. The functional modules may be simulated at the register-transfer level to verify design correctness.

2.4 PRELIMINARY LAYOUT

This step deals with allocating preliminary chip area and shape to each functional block. The preliminary layout provides a means of identifying potentially critical data paths and gives an estimate of the aspect ratios of the individual modules.

2.5 LOGIC / CIRCUIT DESIGN

Following the preliminary layout, each functional module goes through a detailed logic design based on earlier functional descriptions established at the partitioning level. Some form of logic design verification may be carried out using a prototype breadboard or on a logic simulator such as LOSIS.

Once the technology (TTL, ECL, NMOS, CMOS, etc.) to be used has been determined at the design definition stage, the circuit design follows rather rigidly from the logic design. The circuit design includes specifying transistor sizes and in the early days of IC design, each transistor in the circuit was designed individually. This was referred to as full-custom design. Simulation on a circuit simulator such as SPICE may take place simultaneously during this design stage. Often during this stage, some form of layout would take place because each transistor geometry is represented using rectangles and in full-

custom design each rectangle is placed individually on the floor plan.

2.6 DESIGN SIMULATION

In addition to the simulation at the logic and circuit levels for each of the functional modules as stated above; it is often necessary to verify the entire chip design when the logic/circuit design for all the functional modules are complete. However, this can only be done within reasonable limits as it may not be computationally practical or possible for that matter, to simulate an entire chip at the circuit level using SPICE. In the mid-70's, this verification was frequently done using a breadboard prototype. However, in the 1980's, with the advent of simulation engines with awesome computing power, it is now possible to verify the design to a higher degree without the use of prototypes.

2.7 DETAILED LAYOUT

The next step involves laying out and interconnecting all the partitioned modules. This entails specifying transistor geometries as rectangles on the actual chip design area and as indicated above, some form of early chip layout may already have taken place during the circuit design stage. In the mid-70's, the layout was often done on a scaled-up model using mylar as the working medium. Each separate mask level would have to be laid out on mylar using colored pencils and then digitized to produce the artwork database to generate masks. In later years, the computer replaced the pencil and paper approach to detailed layout by introducing interactive graphic editors.

2.8 LAYOUT VERIFICATION

After completing the detailed layout, the task of checking the entire layout for potential errors begins. The checking is usually performed on the layout for each mask level. The type of errors looked for include design rule violations and layout conformity to the actual circuit design specification. Before the introduction of computer-aided layout verification, most of the checking was done manually through visual observation. This was often a tedious and time consuming task with much room for human error. In the late 1970's, computers were employed in this task which otherwise would have taken their human counterparts several weeks to complete. However most of the early computer-aided layout verification systems could only check for design rule violations without checking for conformity to the actual circuit design. Even with computers, as the circuits grew in complexity and size, the layout checking often took many hours of computer time.

2.9 FABRICATION

After completing all the above checks, the next step involves mask making and fabrication. This is dependent on the technology used and the scope of fabrication is sufficient for a thesis study on its own. For this reason, the finer aspects of fabrication will not be covered here.

2.10 TEST AND DEBUG

In the conventional design methodology, after the chip is fabricated, it is put through a series of tests to determine its worth. Basically, there are two sets of tests the fabricated circuit will go through. The first is a set of A.C. tests that involve testing the

functional characteristics of the chip. The second set are the D.C. tests or parametric tests. These determine the electrical specifications of the chip, such as leakage currents, stand-by power and temperature effects. Both these sets of tests are carried out at different levels.

The first level involves testing at the wafer stage. Depending on the type of circuit, this is often done by using a probe card which makes contact with all the pads on the chip. A test vector is then run on the probe station and the chip response is then monitored to determine the condition of each circuit. The bad die are stained with ink and are removed after dicing.

There are a number of test techniques that are employed at this level for different types of circuits. For SSI and MSI circuits, a truth-table approach may be feasible. The inputs are stepped through the entire truth table using a form of gray code and then the outputs are monitored for the expected values. This is only possible for a small set of inputs and outputs. For LSI and VLSI circuits, a form of signature testing may be more practical. This involves testing the circuit with a predefined set of inputs that will generate a unique set of outputs. There is another approach to testing RAMs which involves using different test patterns (diagonal, checker board, parity, etc).

After the bad circuits have been identified, the good ones are separated and packaged. There may be further testing done at the package level to check if the circuit characteristics have been altered by the packaging process; for example the line capacitances may be increased.

The final level of testing is done at the system level and there are a number of industrial testers available for this. Some of the testers are rather complex and they support between 60 to 120 pins. The testing and debugging will continue until all errors are identified and the functional modules that are faulty will have to go through the logic/circuit design stage all over again. This iterative process is

repeated until all errors have been identified and corrected.

2.11 STRENGTHS

The apparent strength of this conventional design methodology lies in its hierarchical approach to the design problem. The top down approach is evidently necessary for the complex circuits designed in this present decade because of the different levels of abstraction it presents in approaching the design problem [9],[10]. However, there appears more to be desired about this particular hierarchical approach, especially in relation to the need for a greater degree of integration between design levels.

Another apparent advantage of this design approach is the level of freedom the designer has in the design space. The full-custom approach permits total creativity and utilizes the innovativeness of the circuit designer. The circuits designed using this approach are somewhat more efficient than those designed using some newer design methodology such as gate arrays or standard cells. However, this apparent advantage may not apply in the VLSI era because the complexity of the circuits may make it totally inefficient to design and layout each transistor individually. It is being widely recognized that some trade-off in circuit efficiency is necessary to achieve a reduction in total design time and cost [10],[11].

2.12 WEAKNESSES

Having its inception at the time of SSI and MSI circuits, this conventional design methodology is unable to face up to the demands of the VLSI age. There are many weaknesses that are evident when viewed in light of the VLSI environment.

First amongst the weaknesses is the lack of a formal description language for the behavioral definition of the design [10]. There are description languages for digital/analog circuits and layout but not for the initial stage of the design process. In many cases, the system definitions reside only in the minds of the designers or at best, attempts are made at describing it in the natural language. This lack of proper documentation early in the conception stage of the design hinders communication between logic design groups working on partitioned modules and promotes misunderstanding. Without a formal definition language, most of the early designers tended to skip the behavioral simulation stage of the design process. This could also be attributed to a lack of proper computer simulation tools at that level.

One of the chief impediments of this design process is the lack of continuity and integration between design stages. In the mid-70's, most of these design tasks were done by separate groups of engineers; there were the system designers, the logic designers, the circuit designers and the layout specialists. The lack of continuity in the design flow made it difficult to convey design concepts through the different levels. The layout specialists could have interpreted the logic designers' design concepts in a different manner from which they were intended. Although computer-aided tools were used in logic and circuit design verification and to assist in layout, none of these tools shared a common database and they hardly had any integration with one another. Hence, a change in one level of the design often reflected a change throughout most of the lower levels. Often this involved a time consuming task of re-designing at each level.

With different engineering teams working on logic, circuit and layout, there inevitably would be certain design teams waiting on the others. This is typical in such a pipeline setup where the layout people may be waiting for the logic and circuit designers to complete their

part. This is inefficient use of manpower and is a basic flaw of the design methodology.

Another major drawback of the conventional design mentality is the failure to integrate testing and debugging considerations early in the design process. Most of the early designs did not go through testing and debugging considerations until after the first chips were fabricated. This did not provide the test engineers with much room to work. As the circuits grew more complex, it became exceedingly difficult to test and debug the chips. Design for testablility was unheard of in the pioneering days.

CHAPTER 3

PRESENT DAY DESIGN METHODOLOGIES

Aside from the hefty capital investments, total design time is a major factor contributing to IC development costs. Material costs do not account for more than an insignificant fraction of the total development costs. Silicon is relatively cheap. In recent years, the complexity barrier faced by VLSI designers has accounted for a staggering increase in design time. It has been reported that the M68000 microprocessor required 52 manyears of effort to design and the layout for the Intel 8086 required 13 manyears of effort [18].

The primary goal of present day design methodologies is to cut down on total development costs by reducing the design time. The most viable solution to achieve this goal is to look to computers in aiding the designer. Hence, most new design methodologies delegate some part of the human designer's role to computers which are good at performing tasks at a great speed. The methodologies differ in the amount of designer role delegated to computers.

In the last four to five years, many new design methodologies have evolved to better deal with the demands of the VLSI design environment. Coupled with these new methodologies are a set of evolutionary computer tools. This chapter studies some of the present day design methodologies and some of the VLSI design tools that shape the methodology. The field of VLSI Design is relatively new and there appears the lack of a commonly agreed upon set of terminology. This makes it exceedingly difficult to study and rather confusing at times. As far as possible, an attempt has been made to include synonymous terms.

In order to have a better perspective of these different design methodologies, it is necessary to first of all define the design space. Unlike most other design situations, the VLSI design may be represented

in three different forms as illustrated in Figure 3.1. It is crucial to understand this design space well because the following discussion on design methodologies will refer closely to this tripartite representation of the design [12].

3.1 THE DESIGN SPACE

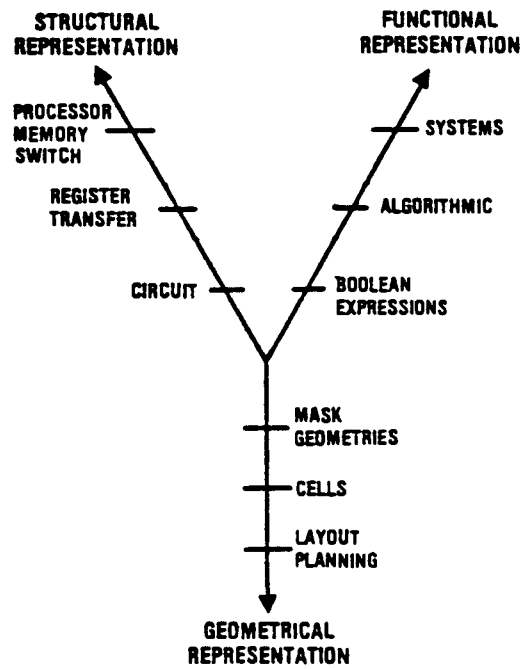


Figure 3.1: The VLSI Design Space

This concept of a design space is an abstract aid to help visualize the various design methodologies in terms of their design entry points and trajectories. This should not be confused with the flow diagram of design activity as discussed in Chapter Two. As was mentioned, most design methodologies attempt to cut down on the design time by transferring as much as possible of a human designer's role to the computer

without sacrificing too much in design quality. Hence, each methodology has a design entry point which is the highest level in the design space where the computer first assumes a part of the design role. The design trajectory is the path the particular methodology follows in translating the design from the design entry point to the end point. The end point is often the data tape containing the masks geometries.

The design may be represented functionally, structurally and geometrically. Often, not all three representations are necessary, especially when the design entry point is at the circuit schematic level which implicitly contains a functional representation. Along each arm of the design space, design refinement is conducted in stages. Each level closer to the center contains a higher degree of refinement and detail. Likewise, proceeding outward along an arm, the degree of abstraction increases and the level of detail decreases. The center of the design space represents the final stage in the design process which is the finished design. From a hierarchical point of view, the final stage represents the lowest level of the design because of the high degree of detail in the finished product. Depending on the design methodology, the design entry point may be along any arm of the design space. In most cases, the design begins from a high-level description of the functional (behavioral) properties and proceeds down to specifying the geometries that go into the masks. The design entry point would then be along the Functional Representation arm. The design trajectory which takes a design from the design entry point to the end point is also dependent on the methodology.

Functional Representation. This is the highest level at which a design can be represented. The functional representation of a design concerns the behavioral aspects and does not include details concerning implementation or circuit structure. There are basically three levels of functional representation - Systems, Algorithmic

and Boolean Expression.

Structural Representation. This level of representation concerns the means of implementing the desired functional aspects of the design. As mentioned above, sometimes the structural representation may serve as a functional representation because it implicitly contains a functional description. The structural representation provides a means of translating the functional representation into the geometrical representation. Refinement of the structural representation commonly proceeds along the Processor Memory Switch (PMS), the Register Transfer and the Circuit levels.

Geometrical Representation. This is the lowest level which represents the design. It concerns mainly the means of implementing the structural representation on silicon. The levels of refinement are often Layout Planning, Cell and Physical Mask Geometries.

There is some overlap between the functional and structural representations. In a well-rounded design methodology, this overlap is often used to examine the design from both viewpoints. This provides a means of verifying the design through cross-checking and comparing the structural progress with the expected functional results.

Design refinement of the functional and structural representations proceeds in a top-down manner. Beginning at the system level, each complex function is divided into less complex subfunctions and each complex module is divided into less complex submodules. The submodules may in turn be divided into even simpler submodules. The process continues until a sufficiently simple function is defined which permits implementation.

Implementation of these submodules corresponds to the geometrical representation arm of the design space. A hierarchical approach may be adopted here as well. By connecting the defined primitives together, more complex structures are formed. These in turn are connected to form yet more complex submodules. This process is continued until the desired submodule is implemented. This is known as bottom-up implementation.

3.2 CONCEPTS IN VLSI DESIGN METHODOLOGIES

Reflecting on the needs of the VLSI design environment, a number of important concepts have been developed which characterize some of the present day design methodologies. A look at some of these general concepts is first presented.

3.2.1 STRUCTURED APPROACH

The realm of VLSI design is in a way analogous to software design [11]. In the 1950's software development was often done by a small group of highly qualified people, each with their own 'style' in programming. As the programs grew larger and more complex in nature, these ad hoc methods could not handle the amount of complexity involved. More formal and structured methods were required to deal with these larger programs - hence, the introduction of structured programming concepts. Likewise, it has been recognized that the structured approach should be adopted in handling the complexity present in the VLSI circuits of today [9],[10]. A structured approach implies two things: hierarchy and regularity/repetition [4].

3.2.1.1 HIERARCHICAL CONCEPT

As was noted in Chapter Two, one of the strengths of the traditional design methodology was the hierarchical nature of the design process. However, this was basically an informal hierarchy as the designer was free to design without any restrictions in the design space at the physical layout level [10]. The design space corresponding to this physical layout level is along the geometrical representation arm. The designer was only faced with technological limitations but was otherwise free to choose and specify each

circuit element to achieve the best possible performance and utilization of silicon area.

The hierachical nature of the conventional design approach is evident through the top-down manner of handling each design stage. Beginning at the design specification stage, each lower level incorporates a lesser degree of abstraction. However, at the physical layout stage, there is usually a bottom-up approach of synthesizing the design. The term "synthesis" means "the combination of simple elements to arrive at a more complex whole". Each circuit is built-up using a mess of rectangles arranged in silicon. In the informal hierarchy of the conventional approach, there is a lack of rigid abstraction at this level.

The new methodologies boast of a formal hierarchy [10], also referred to as a constrained hierarchy [11] and a separated hierarchy [4]. Basically, they all refer to the same thing using different terminology; namely, they imply a structured approach with a predefined set of rules governing inter-level communications which applies to the physical layout level as well. The design space in a formal hierarchy is redefined in terms of the allowed primitives.

In the conventional full-custom approach, the primitives are individual transistors and circuit elements. However, these new structured methodologies define a different set of building blocks as their primitives. Informal hierarchy differs from formal hierarchy in that there is no predefined set of communication principles between primitives in an informal hierarchy [10]. This is especially true at the physical layout level. Simply stated, in an informal hierarchy, there are no rules governing modules from being designed independently of each other. An example of a primitive used in the formal hierarchy may be individual logic gates such as NAND, NOR gates.

The hierarchical approach presents a top down flow of design activity and supports the use of abstraction in handling the complexity factor. This presents the designer with a set of building blocks with a slightly higher level of abstraction than those used in the conventional approach. In the structured approach, the design space at the higher levels is not cluttered with details at the transistor level. This use of abstraction in the hierarchical approach frees the designer from worrying about low-level circuit interactions and permits a clearer focus on important factors pertaining to that design level of the hierarchy. Often abstraction may take the form of user-defined macros or cell calls. More about this later.

3.2.1.2 REGULARITY / REPETITION

The other component in a structured approach is the use of repetition. Often in a hierarchical approach, the presence of regularity in the system permits certain subunits to be replicated. This effectively reduces the complexity of the actual problem by allowing the designer to concentrate on smaller modules; and then replicating and interconnecting these to arrive at the desired design. The regularity factor, derived by dividing the total device count by the number of devices actually drawn by the designer, gives a measure of the use of repetition in the design. Present day microprocessor chips have regularity factors ranging from approximately 5 for commercial products to about 20 for experimental devices [11]. Certain types of VLSI circuits such as RAM's and ROM's will obviously have a much higher regularity factor because of their intrinsic regular structure.

3.2.2 CORRECT BY CONSTRUCTION

Going along the lines of a structured approach, there seems to be a division as to how much rigidity there should be in the structuring. With a formal structuring, the designer looses some freedom in the design space. As a result, the circuit implementations may be somewhat lacking in performance compared to designs done using the full-custom approach. However, there is a definite gain in going with the formal structuring approach.

It has been said that "testing may detect the presence of errors but it will never be able to show their absence" [4]. Presently, approximately one third to one half of the design time is devoted to debugging and testing for design errors. A new design philosophy concentrates on tools which generate designs to be correct in some sense, rather than on tools which check the design for errors after the design has been completed [4]. This conceptual approach is the "correct by construction" methodology. Basically, this concept calls for a formal structuring using a limited set of constructs that have been proven to work previously. Following this methodology, the designer has less control over the design steps and as such, there is a smaller possibility of errors being introduced. Silicon compilers adopt this correct by construction philosophy. The human designer is prone to introducing errors while translating one level of the design to the next. The silicon compiler generates the layout automatically from a high level description of the design, hence reducing the possibility of translation errors. The role of the human designer is somewhat displaced in this approach.

3.2.3 USE OF PAST EXPERIENCE

In IC design, often certain parts of a chip may be similar to parts that have been designed previously in another chip. Hence, it would be a waste to design each chip starting from scratch, when it would be much

easier to use modules that have been designed previously. There are definite advantages in adopting this concept. Firstly, the time it takes to design the chip will be significantly reduced if certain modules can be copied directly from previous designs. Secondly, in using these tested modules from previous designs, the risk of errors is reduced. However, it is often difficult to define standard modules that meet performance requirements in all design situations. Hence, designs which adopt this design approach are often inefficient to a certain degree. This use of past experience is the basis of the Standard Cell methodology.

3.3 SPECTRUM OF VLSI DESIGN METHODOLOGIES

Although an attempt is made at grouping these new methodologies under specific categories, in no way is the taxonomy a rigid and precise one. Most methodologies can be classified under more than one category. However, there are enough distinct characteristics to warrant the following groupings. Most of these new design methodologies can be classified under one of three main approaches as shown in Figure 3.2.

CHARACTERISTICS	COMPUTER-AIDED DESIGN	EXPERT SYSTEMS	DESIGN AUTOMATION
1. Degree of human designer interaction	High	Average	Low
2. Susceptibility to errors	High	Low	Low
3. Time needed to complete design	Long	Short	Shorter
4. Circuit performance	Good	Good	Average

Fig. 3.2: Spectrum of VLSI Design Methodologies

Basically, the first approach evolved from the conventional design methodology described in Chapter Two. This approach believes that all design decisions should be made solely by the experienced human designer who is best able to optimize the design. The designer is presented with a comprehensive set of computer design tools such as graphic editors,

logic and circuit simulators, and design verification tools which are often well integrated through an efficient database. This approach has come to be known as Computer-Aided Design (CAD).

The second school of thought follows along the lines of artificial intelligence. The proponents of this approach believe that human knowledge can be captured in the form of design rules and stored in a knowledge base. This Expert System approach is still a relatively new approach and the IC industry is just beginning to tap the possibilities it has to offer. Most existing expert systems are efficient only in analyzing or critiquing a design [12].

Advocates of the third approach believe that in order to reduce the design cycle time for VLSI circuits, a form of algorithmic approach must be adopted to represent design knowledge. This approach utilizes translators and compilers to generate the design at the circuit/layout level from a high-level description of the design problem. This approach is known as Design Automation (DA) and is on the opposite end of the design spectrum from CAD methodologies. Unlike the other two approaches which are aimed at assisting the human designer; this Design Automation approach displaces part of the job of the human designer. The designer has little or no control over the lower-level design stages as this is generated automatically from the high-level design specifications. Examples of DA methodologies are silicon compilers, PLA generators and ROM/RAM generators. Design Automation systems are just being introduced in the IC industry and cater to just a small class of design situations involving designs with predominantly regular structures.

3.3.1 COMPUTER AIDED DESIGN

The methodologies that come under this classification can be better understood when viewed in relation to the design space concept presented

on page 17. Most of the present day CAD methodologies concentrate on refining the LOGIC/CIRCUIT DESIGN and PHYSICAL LAYOUT stages of the design process. This corresponds to a design entry point along the Structural Representation arm of the design space and the trajectory of most CAD methodologies proceed in refining the design down to the geometrical representation arm as depicted in Figure 3.3. Methodologies which attempt to refine the higher levels of the design process (such as the DESIGN SPECIFICATION and PARTITIONING stages) through delegating some of the designer's responsibilities to computers, are geared more towards design automation. Some of the methodologies that fall under this category of Computer Aided Design are the Full-Custom approach, the Standard Cell methodology and Gate Arrays. Methodologies that are classified under the CAD approach are characterized by a high degree of human designer interaction in the design process. The human designer makes most of the critical design decisions and these tools are used only to assist in the deign.

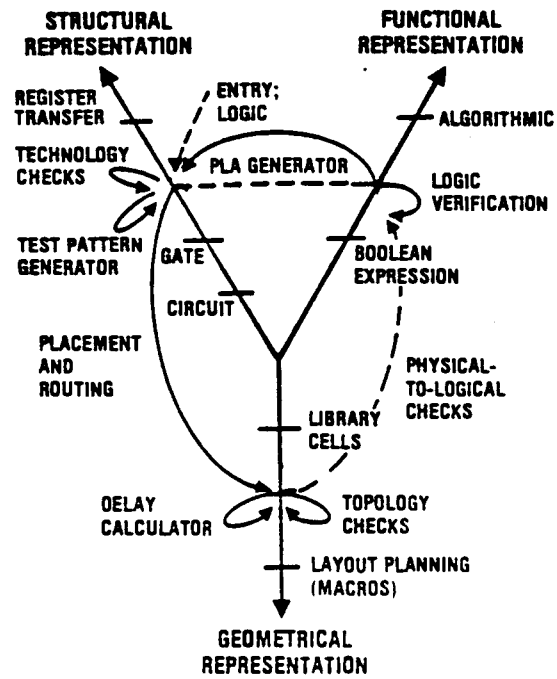


Fig. 3.3: Design Space of CAD Methodologies

Traditionally, in the design of SSI and MSI circuits, the use of computer aided design tools has been limited to circuit level simulators such as SPICE and perhaps some form of graphics editor for schematic capture. As the circuit complexity grew, LSI circuit designers began using logic simulators to verify their designs. These early logic simulators often were only capable of simulating logic behavior without concern for timing considerations. The circuit simulators needed a circuit description at the transistor level using a form of node connection to represent the circuit. This form of circuit representation was different from the circuit representations needed for the logic simulators. Also, there would be another form of circuit representation for the graphics editor. Hence, each computer tool was practically independent of the other and there was hardly any form of integration between these design tools. There was no means of sharing data between these design tools because these tools did not speak the same language.

With VLSI circuits, there is even a greater need to use computer aided tools at each stage of the design. Figure 3.4 illustrates some of the CAD tools used at each stage of the design. Often, these tools are integrated through a common database using a common descriptive language.

3.3.1.1 FULL-CUSTOM APPROACH

The Full-Custom approach is similar to the conventional methodology discussed in Chapter Two, in that each circuit element is individually designed. However, the present day full-custom design methodology is accompanied with a comprehensive set of CAD tools, often integrated through an efficient database. Circuit capture is often done at the layout level using an efficient interactive graphics terminal. Each transistor is represented by manually drawing rectangles on the respective mask levels. The primitives

SYSTEM DEFINITION	———	SILICON COMPILERS
BEHAVIORAL SIMULATION	———	FUNCTIONAL/BEHAVIORAL SIMULATORS
FUNCTIONAL DECOMPOSITION	———	(human designer)
PRELIMINARY LAYOUT	———	(human Designer)
LOGIC/CIRCUIT DESIGN	———	LOGIC SYNTHESIS TOOLS: PLA GENERATORS ROM/RAM GENERATORS SCHEMATIC EDITORS
DESIGN SIMULATION	———	LOGIC SIMULATORS WITHOUT TIMING LOGIC SIMULATORS WITH TIMING CIRCUIT SIMULATORS
DETAILED LAYOUT	———	INTERACTIVE LAYOUT GRAPHICS EDITOR SYMBOLIC LAYOUT TOOLS AUTOMATIC PLACEMENT TOOLS AUTOMATIC ROUTING TOOLS
LAYOUT VERIFICATION	———	DESIGN RULE CHECK PROGRAMS CIRCUIT EXTRACTION PROGRAMS SCHEMATIC-ARTWORK COMPARE PROGRAMS
TEST & DEBUG	———	AUTOMATIC TEST PATTERN GENERATORS

Fig. 3.4: CAD tools at each design stage

in this design approach are the individual circuit elements. There are no fixed rules governing the use of these primitives, other than the design rules which dictate the minimum feature size allowable. This approach constitutes an informal hierarchy at the physical layout level.

STRENGTHS: The full-custom approach is used only when critical performance requirements exist because each transistor is individually designed and this enables the designer to overcome any critical signal paths.
This approach usually leads to a better optimization of silicon area compared to the Standard Cell or Gate array methods.

WEAKNESSES: This approach requires a longer time to complete the design and as such the design costs will be higher.
The full-custom approach does not help resolve the complexity factor because it still basically adopts the informal hierarchy in the layout stage. This complexity factor also leads to a greater possibility of errors being introduced in the design.

3.3.1.2 STANDARD CELLS

In VLSI circuit design, the factors to optimize are somewhat different from that in the design of SSI and MSI circuits. Previously, when the minimum feature size was still relatively large, it was crucial to optimize the silicon area used in order to accomodate the whole circuit on a chip. However, in this decade, with the advances in photolithography technology, there has been a significant reduction in device dimensions and coupled with that,

the developments in process technology has led to a larger die size being used. This reduction in device dimensions and the larger die sizes imply that more circuit devices can be placed on a single chip. Hence, it no longer becomes that crucial to optimize silicon area.

The increase in the number of devices that can be put on a chip has led to an alarming increase in the complexity of the circuits being built. As was mentioned in Chapter One, it takes much longer to complete the design and because of this increase in complexity, there is a greater possibility of errors in the designs generated. Hence, the stress is now on minimizing the design time and also to reduce the amount of work needed in verifying the final design.

The standard cell approach proposes to achieve this optimization through the use of a library of fixed circuit constructs called standard cells. This is basically an application of the "Use of Past Experience" concept discussed earlier. This approach is analogous to designing on a printed circuit board (PCB) using standard off-the-shelf DIP components. In this case, the PCB will be the active chip area and the catalog of DIP chips will be the library of standard cells. As in the case of Dual-in-line Packaging, standard cells are constructed with each cell having the same height to facilitate their interconnection. The cells are arranged in rows on the chip with areas between the rows for interconnections. Most standard cell systems offer the designer the added feature of automatic placement and routing of these cells. However, this feature comes under the grouping of Design Automation and will be discussed there. The standard cell approach is one of the most widely used CAD methodologies in the industry today. A number of variations on the standard cell approach exist:

Standard Cell Library. Software libraries of these standard cells are commercially available through vendors such as Texas Instruments and National Semiconductor. The TI standard cell library contains over 200 logic blocks using a 3-um CMOS process, which include RAM's, ALU's, decoders, adders and analog cells like op amps [17].

Custom Cell Library. The custom cell approach allows the user to define his/her own cells and store these in a library to be used in very much the same way as the standard cell library. Modules that are commonly used in the design can be designed just once and stored in the custom cell library and then replicated throughout the design. Often the custom cell library is used to supplement the standard cell library.

Semi-custom Cell Library. This is basically an extension of the two approaches mentioned above. However, instead of constructing each custom cell from scratch, the user has the option of modifying the existing standard cells from the library into semi-custom cells which may better meet the design specifications.

STRENGTHS: The use of a standard set of building blocks supports the hierarchical concepts discussed on page 17 and presents the levels of abstraction needed to handle the complexity factor.

With this set of well characterized building blocks, the possibility of errors in the layout stage is reduced. Errors can only be introduced in the interconnection (interfacing) of these cells because the

internal cell structures have been checked thoroughly before being stored in the library.

Also, the design time is reduced significantly because it would not be necessary to design each circuit down to the transistor level, but rather, it would suffice to design to the standard cell level. The design time is also shortened through use of regularity and repetition concepts.

Changes can also be carried out easily using this approach. If there should be an error in a certain cell, it would only be necessary to effect the change on the particular cell structure in the library. It would not be necessary to correct each occurence of the cell in the whole design because each occurence of the cell is nothing more than a macro call. The final cell structure is then called from the library when the final design is done.

WEAKNESSES: This approach does not offer a good compaction of cells in the chip. The design performed using the standard cell approach tends to occupy more silicon area than the same design done using the full-custom approach. However, as mentioned above, utilization of silicon area is no longer the prime factor to optimize in the design of VLSI circuits.

3.3.1.3 GATE ARRAYS

The Gate Array is also referred to as the Master Slice, Universal Array and Standard Array [18]. This approach attempts to cut down on the design time by prefabricating a standard array of logic

components on the wafer up to the first layer of metalization and then stockpiling these for future customization. The desired logic functions are then implemented on these gate arrays by connecting the required gates in the metal layer. This allows for a fast turn-around time. Automatic routing of these interconnects is also possible and again, this comes under the grouping of Design Automation and will be discussed later. Currently, gate arrays with up to 10,000 gates are commercially available.

STRENGTHS: The gate array approach provides a fast turn-around time and relatively low cost in developing an application-specific IC. This is suited for system manufacturers without their own in-house IC development facilities. Companies such as LSI Logic (Milpitas, California) and Gould-AMI (Santa Clara, California) provide such gate array development facilities to system manufacturers.

The likelihood of errors in the design is reduced somewhat because the gates have already been fabricated and tested. Errors can only be introduced in interconnecting these gates.

WEAKNESSES: Gate arrays are not suited for high-performance oriented applications because the rigidly defined gates and long interconnect lines make it almost impossible to achieve fast switching capabilities.

Circuit density is even poorer than in the standard cell approach. This is partly because not all of the gates in the gate array are used in any given application.

3.3.2 EXPERT SYSTEMS

The VLSI era has brought about many changes other than those affecting circuit size and density. With the capabilities of VLSI technology, a brand new market for application-specific IC's is opening up. Digital system designers are now looking to application-specific IC's to replace whole digital circuit boards built using off-the-shelf components. The Apple MacIntosh is a good example of this trend where the designers managed to use slightly over 50 chips in the whole computer as opposed to approximately three times that number in the IBM PC [21]. This significant reduction in chip count was possible through the use of application-specific IC's. This new demand for application-specific IC's has spurred VLSI designers to achieve even greater productivity in their efforts to meet the shorter turn around time imposed by this new market. VLSI design methodologies are turning to design automation systems to meet the needs of this market segment, which is characterized by short turn around times and relatively low volume demand. The application-specific IC is also referred to as a custom IC. However, this should not be confused with fully-custom designed ICs. The fully-custom designed IC refers to the design methodology adopted in the design of the IC (i.e. the full-custom approach as mentioned earlier). Currently, the research and development of these automation systems can be grouped under two main approaches. There are the Expert Systems and the Design Automation Systems.

Expert systems are also known as knowledge-based systems. Both these terms are often associated with artificial intelligence (AI); which is the field of Computer Science devoted to making computers 'think and reason' like human beings. Knowledge-based expert systems employ human knowledge to solve problems that ordinarily require human intelligence [20]. Designing an IC requires a vast amount of expert knowledge. Expert systems attempt to capture this knowledge in a data-

base and apply the concepts of AI to create a suitable IC design environment.

What are some of the characteristics of a suitable IC design environment? Before proceeding to answer that question, the term 'suitable' should be better defined. In this context, the term refers to a design environment suited to increasing design productivity. Here then are some of the characteristics of a suitable IC design environment [22]:

* The design environment should support some sort of automatic design synthesis tool. This will enable more design alternatives to be checked out. Designers can then specify parts of a design and have the synthesis program fill in details quickly.

* There should be good design verification tools available. This will permit each stage of the design to be simulated and potential errors identified early and corrected.

* The computer tools should be fully integrated to ensure smooth translation between design stages. A well integrated set of design tools also allows changes to be effected easily. A change in one stage of the design will reflect throughout all the other stages in a well integrated design environment.

* The design environment should also support multilevel representations. Design representations should present multiple levels of abstraction, maintaining a correlation between an abstract specification and the detailed design. Such representations support mixed mode simulators and timing verifiers.

Presently, expert systems are still in their experimental and developmental stages. It would not be possible to include a complete discussion on the mechanics of a knowledge-based system because it clearly extends into the study of Artificial Intelligence and is beyond the scope of this thesis.

However, there are basically two types of expert systems being developed for IC design applications. There are the expert systems used in critiquing or analyzing a design and then there are the systems for design synthesis. Most of the expert systems being developed are of the first type - for analyzing a completed design. An example of such an expert system may be one that will pick out possible critical signal paths or possible race conditions from a circuit layout and highlight them to the designer. Such tools are not totally new to the industry and are nothing more than a new generation of well integrated simulation tools. The second type of expert system being developed, for design synthesis, has not fully been realized. Such expert systems border on the grouping of silicon compilers, which will be covered in the next section on Design Automation systems. As mentioned earlier, most of these systems can be grouped under more than one design methodology.

The line separating Expert systems from Design Automation systems is somewhat vague. This is because methodologies under these groupings all have a certain level of automation in them. The main characteristic that determines whether an automated design methodology is considered an expert system or a Design Automation system is in the way the system is implemented.

An expert system in its simplest form has two levels of action: control-level and base-level actions. Base-level actions modify the representation of the problem and its solution. Control-level actions determine which base-level actions to execute. Control-level actions are usually represented as rules of the form:

If **SITUATION-PREDICATE** then **ACTION**

where the predicate tests for the existence of a particular situation before performing its action. Expert systems are made up of these rules which are stored in a database - hence the term rule-based expert system. When the rules change, or new rules are added to the system, changes are simple and fast - the rule base has only to be updated without affecting much of the overall system.

Design Automation systems on the other hand are implemented somewhat differently. They follow an algorithmic manner of implementation - such as compilers and translators. Hence, when the rules need to be changed or updated, often major changes in the system software is necessary.

To illustrate an expert system, a representative design methodology supporting the expert system approach is presented. Palladio is a circuit design environment for experimenting with methodologies and knowledge-based expert system design aids [19].

3.3.2.1 PALLADIO

The Palladio circuit design environment was developed at Stanford University. This methodology stresses the need for creating an integrated design environment. As can be seen from the design space (see Fig. 3.5), the Palladio methodology supports a multilevel representation of the design. The designer is able to define models of circuit structure or behavior. These circuit models, called perspectives, are similar to circuit design levels; the designer can use them interactively to create and refine circuit design specifications [19]. The dotted horizontal lines in the design space show the correlation between the behavioral and structural representations. "Within Palladio, one or more behavioral perspec-

tives can be associated with a structural perspective and vice versa. This permits a component to be specified from a very abstract behavioral perspective at an early stage in the design process; then from a more concrete, behavioral perspective later in the process." [19].

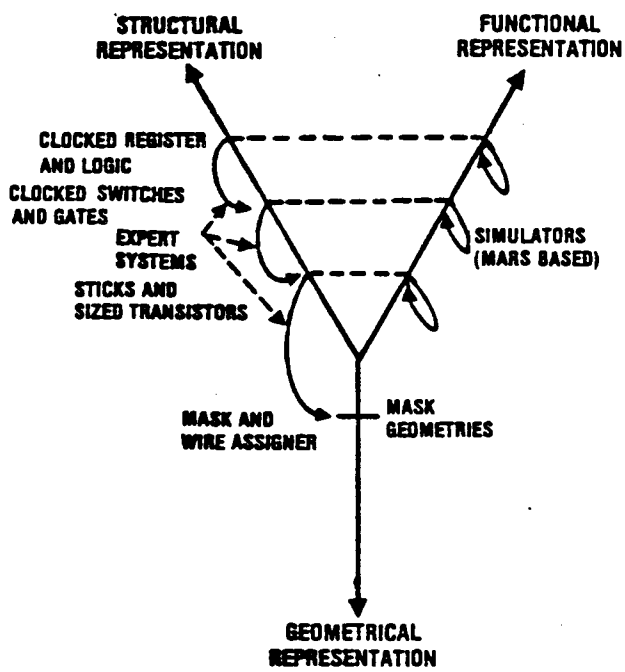


Figure 3.5: A Design Methodology in Palladio

The behavior of a circuit is expressed within Palladio as perspective-specific rules that are invoked by changes in a component's state; which in turn, changes the state of the circuit. For example, the behavior of a unidirectional pass transistor can be represented as follows. The pass transistor has three ports: IN, OUT, and CTL. Based on a three-valued logic, the behavioral rule for the pass transistor is:

```
IF   Signal (Port CTL) = HIGH at time t
THEN Signal (Port OUT) = Signal (Port IN) at
     time t+1.
```

A different perspective of the pass transistor might use a 3X3-valued, level-strength logic [19]. The three following behavioral rules specify the pass transistor's behavior from this perspective:

```
IF   Signal (Port CTL) Level = 3, Strength = s  at time t
THEN Signal (Port OUT) = Signal (Port IN) at
     time t+1
IF   Signal (Port CTL) Level = 1, Strength = s1 at time t
AND  Signal (Port OUT) Level = 1, Strength = s2 at time t
THEN Signal (Port OUT) Level = 1, Strength = 1 at
     time t+1
IF   Signal (Port CTL) Level = 2, Strength = s  at time t
THEN Raise error flag.
```

Structural perspectives are defined by the types of components allowed when partitioning a component into constituent components with respect to that perspective. Basically, there are two types of components within any given structural perspective. There are the primitive components which are components that cannot be further partitioned within the perspective. Then there are the composite components, which are components that can be further decomposed within that perspective. Hence, the terms primitive and composite are really defined relative to a certain perspective. A circuit is fully partitioned in relation to a perspective if all components at the lowest level of the circuit's component hierarchy are of primitive types. The definition of a structural perspective can also include composition rules that limit the ways in which components can be interconnected. An example of a structural perspective in Palladio is now presented.

The CSG Perspective [19]. The Clocked Switches and Gates perspective was implemented in Palladio and is used for designing a broad variety of NMOS circuits. The circuit is viewed as a network of steering logic, clocking logic and restoring logic gates. The CSG perspective is based on concepts similar to those adopted in

switch level simulators. The primitive component type for steering logic is the pass transistor. The composite component of type steering logic may be the 2 X 2 barrel shifter shown in Fig. 3.6(a). The primitive component type for clocking logic is the pass transistor again, but with the CTL port replaced by a qualified clock control. There are two primitive component types for (restoring) logic; the pull-up transistor and the pull-down transistor. The simplest composite component of type logic gate is the inverter, which consists of a pull-up and a pull-down swicth connected as shown in Fig. 3.6(b). Composite components of type logic gate performing more complex functions can be constructed from the

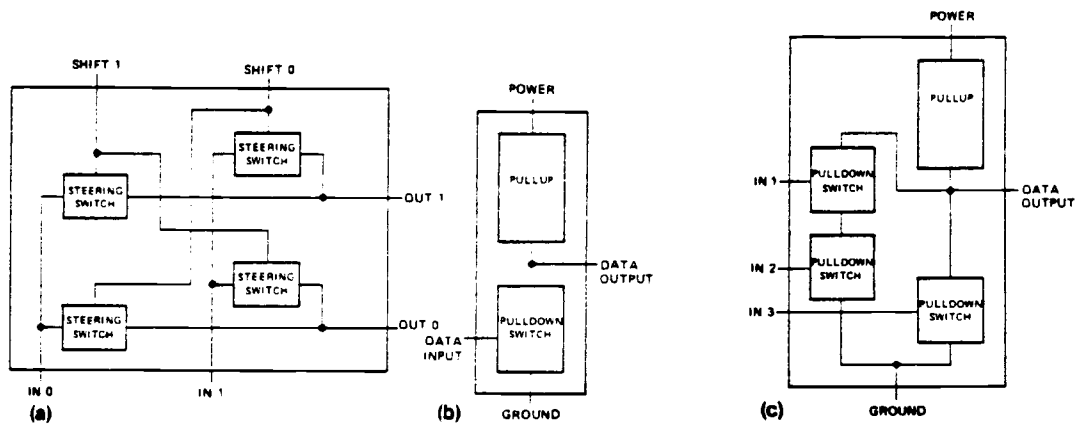


Figure 3.6: Examples of CSG composite components:
(a) 2 X 2 barrel shifter
(b) NMOS Inverter
(c) Complex Logic gate

primitive components as shown in Fig. 3.6(c). The function can be represented as:

$$OUT = NOT((IN1 \text{ AND } IN2) \text{ OR } IN3)$$

There are composition rules governing the manner in which components are connected. For example, in the CSG perspective, there are rules specifying how primitive and composite CSG components can be connected. Three of the CSG composition rules are [19]:

1. A control input of a steering switch or a steering net can be connected only to an output of a restoring logic gate. This composition rule forbids the use of a pass transistor's output for driving the gate of another pass transistor. It reflects a shallow model property, namely that the level of a signal is reduced when it passes through a pass transistor. This reduction cascades when the output of a pass transistor controls another pass transistor. The cascading effect could produce an indeterminate output signal.

2. A control input of a clocking switch can be connected only to a basic clock or to an output of a clock qualifier component. This rule enforces the CSG methodology's distinction between pass transistors that clock data and those that steer data.

3. An output of a clocking switch can be connected only to an input of a restoring logic gate. This rule implies that steering logic cannot be interspersed between clocking switches and logic gates. The rule helps prevent errors in charge sharing that take place when a charge leaks to or from a logic gate storage point (i.e., the input capacitance of the logic gate). Such leaks can result in an indeterminate logic level at the storage point.

There are other composition rules that apply at different structural perspectives. For example, at the layout perspective, there are rules governing the geometric composition of rectangles representing the metal, polysilicon and diffusion regions. These are the design rules that govern size and spacing of the regions.

The Palladio environment adopts a design methodology similar in certain aspects to the Standard Cell CAD approach described earlier. Instead of referring to it as a standard cell library, they call it a prototype library. The translation between levels of

the design is still primarily the responsibility of the human designer. However, the knowledge-based system is used to assist in the translation as well as to refine the design within a level. The Palladio system provides a mask-level assigner to assist in assigning mask levels to interconnect (wires and contacts) the various components. This layout tool is implemented using the knowledge-based approach.

The Palladio Expert System was implemented as a research effort in investigating circuit design environments. In its current implementation, Palladio is not exactly a user-friendly system and the machine-user interface needs improvement. The system is also utterly slow and requires approximately eight hours on a XEROX 1100 computer to run a 1000-event simulation of a multiprocessor circuit. Part of this inefficiency results from attempting to set up a general design environment - basically, in the design of such a general design environment, there is a trade-off between efficiency versus flexibility. Even with a modified version opting for greater efficiency at the expense of flexibility; a flexible, fully integrated design environment for custom, VLSI circuits will require computers more powerful than those currently available. However, this research effort has highlighted certain concepts that are valuable in the development of VLSI design environments.

STRENGTHS: The knowledge-based expert system provides an integrated design environment which supports the structured approach. The integration of design tools is achieved through a common database.

The rule-based system supports the construction of new specification languages and also supports augmentation of the system's expert knowledge to reflect current

design constraints and goals. After all, the expert system is only as good as the set of individual rules that make it up.

Expert systems are good at optimizing large design spaces and dealing with problems of little regularity. This is unlike the algorithmic approach which usually is restricted to problems with predominantly regular structures, such as PLA's.

WEAKNESSES: In order to have the flexibility to deal with a large design space, most expert system implementations demand large computing facilities. This is similar to the computing requirements imposed by most Artificial Intelligence systems.

3.3.3 DESIGN AUTOMATION

The term 'Design Automation' has been used rather loosely in the Computer-Aided Engineering industry. There is some disagreement within the IC design industry as to the design systems and methodologies that come under this grouping. Some consider expert systems as design automation systems as well because of the level of automation in these systems. However, there are sufficient differentiating characteristics in their implementation to warrant a separate category. They may be thought of as a sub-set that branched into a distinct category on its own.

VLSI design activity can be grouped into basically two parts - the functional design and the physical design. Similarly, the design automation tools that have evolved over the years address these two areas of the design process. Hence, the DA tools may be grouped into functional design tools and physical design tools [18].

Design automation systems that deal with the physical design aspects include tools such as automatic placement and automatic routing programs. DA systems that address the higher level of functional design will be systems such as PLA generators and ROM generators. Finally, a complete design automation system that addresses the whole VLSI design cycle from behavioral specification to layout will be the silicon compiler.

3.3.3.1 AUTOMATIC PLACEMENT

In most present-day design automation methodologies, the design entry is done at the schematic level using a schematic editor. The schematic editor is an interactive graphics system that permits the designer to enter the schematic blocks into the system either through a mouse or graphics tablet. At this level, the designer is not concerned about the lower level layout details. The schematic

data is represented in a form of description language which contains no information about layout coordinates and is quite unlike a layout description language such as CIF (Caltech Intermediate Format) [23].

Many design automation systems offer the option of automatic placement. This provides a means of transforming the design from the schematic level to the layout level. A chip is generally made up of many smaller circuit modules. These modules may be standard cells or macros; each of which must be assigned a location on the chip layout. The automatic placement program provides a way of placing these circuit modules automatically; as the name implies. In most systems, the macros are placed manually by the designer because they are generally much larger than the standard cells and are fewer within a given chip. Hence, it would not be much of a problem to hand place these macros.

The placement program often works hand in hand with the automatic routing program [26],[27]. This is because the placement program must ensure that the placement of the circuit modules will enable their wirability. That is, there must be sufficient channel space left in between modules to allow the automatic routing program to lay the interconnect tracks. As will be mentioned below, there are a number of different routing methods which are basically technology dependent - single layer metal, two layer metal and air-bridge crossings. These will affect the optimal solution of the placement program.

Generally, most placement programs adopt a hierarchical solution to the placement problem. The chip area is covered with an underlying square grid. Circuit modules with common pins to be interconnected are grouped into large clusters and are placed in the underlying rectangles. Each cluster and rectangle is then successively broken down until individual cells are placed, as

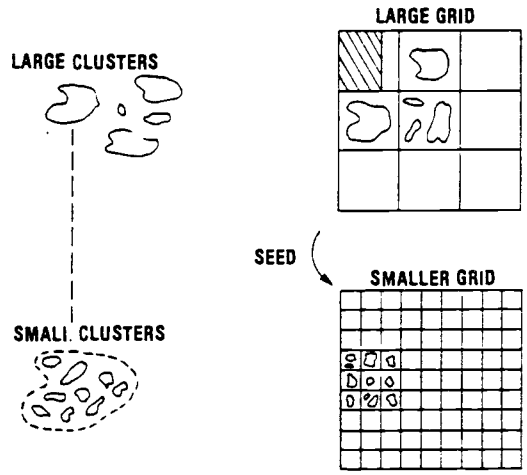


Fig. 3.7: Automatic Placement Program

depicted in Fig. 3.7 . The above placement program is typical of IBM's Engineering Design System (EDS) [25].

However, there are some placement programs that do not follow the above hierarchical approach of global placement. Instead, the placement program is used to shrink the inter-cell spacing in both the X and Y axes until it has the minimum spacing allowed by the design rules. The STICKS program [24] at Hewlett-Packard provides this feature which is more a form of spacing synthesis than automatic placement.

3.3.3.2 AUTOMATIC ROUTING

Programs for integrated circuit layout primarily involve placement and routing. The automatic routing program serves to interconnect pins on the circuit cells. The designer has only to specify the pins to be connected with the same signal name and the automatic routing program will do the rest to ensure their connec-

tion. The automatic routing program depends heavily on the automatic placement program in the sense that the quality of the routing is strongly dependent on the quality of the placement. On the other hand, the automatic placement program needs to have some indication as to the quality of routing it can expect without actually having to route the wires. Algorithms have been developed to perform this estimate of channel area needed for the routings without actually laying them out [28].

There are a number of different routing algorithms being developed and used. The most basic of which is perhaps the River Routing algorithm [26] which involves interconnecting two sets of terminals in sequence and this is all done in one plane. Each set of terminals is restricted to be along a straight line and is called a sequence. This algorithm is limited by the fact that the inteconnection topology is all in one plane. In most technologies, there is more than one level of interconnects permitted. R. Y. Pinter [28] describes an algorithm for optimal routing in rectilinear channels.

Generally, there are two levels of refinement within the automatic routing program. First of all, a gloabl wiring of the chip is performed. This involves setting up a coarse grid of rows and columns. Wires are then routed in these rows and columns without being assigned specific wiring tracks. This global wiring is aimed at reducing congestion at certain locations where the wires being routed outnumber the available tracks. The routing program counts the number of wires crossing each horizontal and vertical boundary and then reroutes the wires if the number of crossings exceed the number of available channels. Fig. 3.8 illustrates this procedure using a two layer interconnect technology.

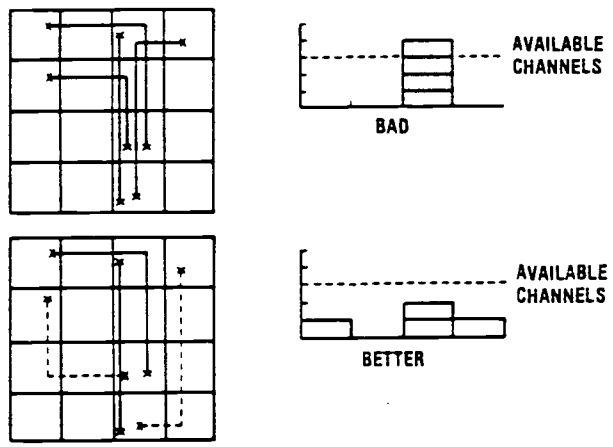


Fig. 3.8: A global wiring scheme

This global wiring procedure does not necessarily guarantee a solution as it depends on the placement of the cells and the degree of doglegging permitted [29],[30]. Doglegging is the process of adding 90^{o}-bends to the wire. Fig. 3.9 illustrates the effect of restricted doglegging.

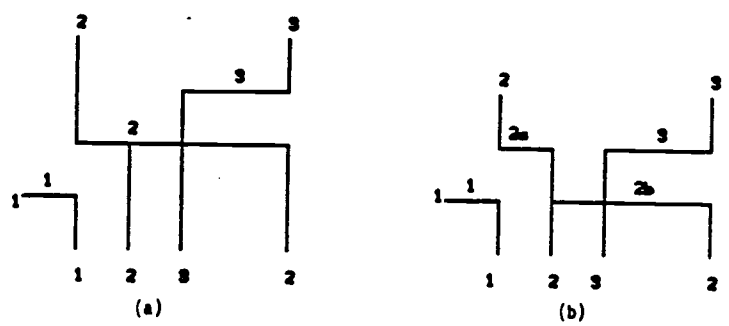


Fig. 3.9: (a) Routing without doglegging;
(b) Routing with restricted doglegging.

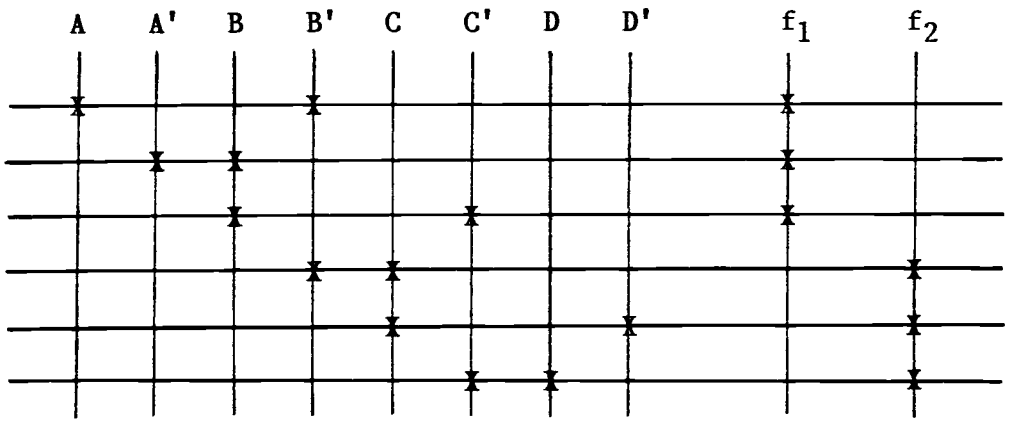
After the global wiring is performed, the second level of refinement involves the detailed wiring [25]. This part of the

automatic routing program is deterministic because the global wiring procedure ensures the available wiring tracks. If a congestion should arise that cannot be resolved, it would have been noted in the global wiring section. The detailed wiring involves assigning specific tracks to the wires.

3.3.3.3 PLA GENERATOR

The Programmable Logic Array (PLA) is commonly used for logic implementation in VLSI applications. It is particularly useful for implementing the combinational logic in finite state machines. This is because the combinational logic used in the feedback paths of finite state machines is often highly complex and of an irregular nature; making a random logic implementation highly irregular and inherently difficult to effect changes without major redesigning involved. Conceptually, the PLA consists of two arrays that implement the AND and OR logic functions for the sum-of-products realization of the desired Boolean functions. However, in practice, either NAND or NOR logic is used to implement the PLA. The two arrays are also referred to as the AND and OR-planes [23]. These arrays contain a set of columns and a shared set of rows in a grid-like manner as shown in Fig. 3.10. The columns running through the AND-plane represent the literals and their complement and the columns in the OR-plane are the realized Boolean functions. The rows represent each of the product terms in the minimized Boolean function(s).

The PLA has the advantage of being programmed or personalized at a later stage after the basic layout of the columns and rows has been completed. However, the number of literals (or inputs to the PLA) must be known to determine the number of columns in the AND-plane. The number of Boolean functions to be generated from the PLA



AND-plane OR-plane

$f_1 = AB' + A'B + BC'$ $f_2 = B'C + CD' + C'D$

Fig. 3.10: AND-plane and OR-plane of a PLA

will determine the number of columns in the OR-plane. The number of product terms in the minimal SOP expression will give an indication of the number of rows needed. If there is more than one output from the PLA (which is usually the case), then these output functions must be jointly minimized before the layout pattern can be generated.

Programming the PLA is a rather simple task which basically involves placing a diffusion region at a row-column intersection if the column literal appears in the product term represented by that particular row. The diffusion region is not exactly placed at the row-column intersection, but rather in a manner such that a transistor is formed with the column literal as the input to the gate of the transistor and the source of the transistor is connected to ground. Because of the inherently regular structure of the PLA, it appears to be a prime area for design automation to take over this task.

The PLA generator is a design tool that permits the user to specify the Boolean function(s) to be generated in either a Boolean expression format or in certain cases as a truth table entry; and have the generator complete the entire design of the PLA. This involves a number of separate steps. The PLA generator will first attempt to minimize the Boolean functions. Often this is a whole design tool on its own. The minimization program is basically a heuristic routine that checks the SOP Boolean expression for cover; redundant terms; targets for further minimization such as minterms with unit distance; and other minimization criteria involving multiple output functions [31].

Once the minimized expressions are determined, the PLA generator will automatically generate the necessary diffusion regions at the grid intersections to personalize the PLA. The complete layout is generated automatically. The area of the PLA will be proportional to the number of rows and columns needed. However, based on the above mentioned scheme of laying out the rows and columns, a large percentage of the row-column intersections are not used in personalizing the PLA. Typically about 90% of the row-column intersections are not used based on this layout scheme [32]. Hence there is an inefficient use of chip area when using a PLA. This less than optimal use of chip real estate may out weigh the benefits gained through the PLA approach.

There are several techniques proposed for minimizing the layout area of the PLA. The most commonly used method is that of PLA folding studied by Hachtel et al.[34],[35], Suwa [36], and Wood [37]. Greer [38] and Paillotin [39] have proposed methods similar to folding. Basically, there are two forms of PLA folding - that involving column splitting and the other involves row-splitting [33]. These folding techniques are possible because the relative

order of the rows or columns in the grid does not affect the logic functions realized.

Folding the columns of a PLA involves splitting the column into two segments such that two inputs or outputs can share the same column space as shown in Fig. 3.11. To ensure each segment is directly accessible to the incoming input lines or the outgoing output lines, each column can only be split into two segments. Folding the columns of a PLA can lead to as much as a 50% reduction in layout area [32].

Row-splitting is similar to column-splitting. However, the consequences are somewhat different. Column-splitting does not

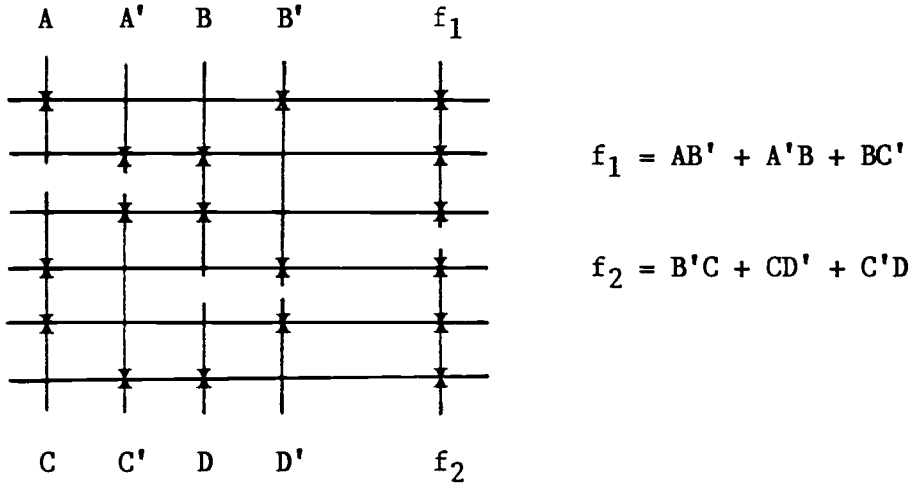


Fig. 3.11: Column-folded PLA

alter the PLA planes significantly; there is still the AND-plane and the OR-plane. However, with row-splitting, it becomes necessary to implement the row-folded PLA with either an OR-AND-OR or an AND-OR-AND configuration. This is because folding the row results in fitting two product terms into a row space and these must then be passed through the OR-plane to form the desired output function. Hence the need for two OR-planes, one on each side of the AND-plane

to tap the product terms off a row as shown in Fig 3.12. Row-splitting can result in as much as a 50% savings in layout area as well. With both columns and rows folded, up to a 75% savings can be expected [32].

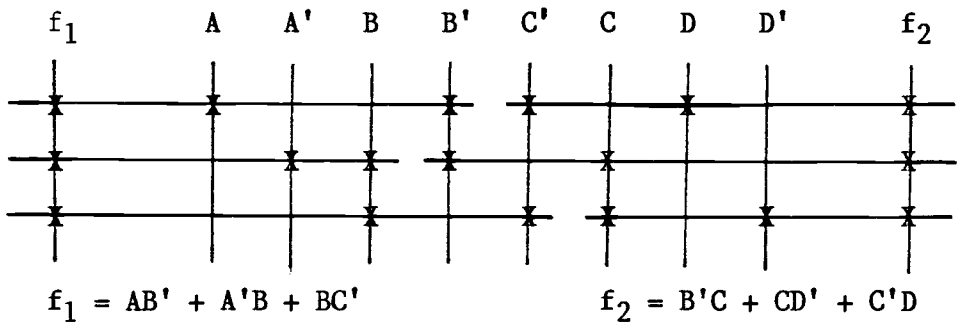


Fig. 3.12: Row-folded PLA with OR-AND-OR configuration.

The PLA generator provides the designer with a means of specifying a design at the logic level and have the layout generated automatically.The logic level representation of the design is process independent and hence the same design data at this level can be used for different chip fabrication technologies. This is the intention of design automation systems such as PLA and ROM/RAM generators. In such systems, there is usually a process file attached that contains parameters relating to the current design rules. However, in reality, such tools are still process dependent to a certain degree because the electrical properties of a transistor do not scale in the same ratio as the physical dimensions [40]. For example, using a process with 1.5 um gate length, the input buffers to the PLA may be able to drive a certain line capacitance, but for a 3 um gate length process, the drive capacity will be reduced and hence the buffer may need a few extra stages of inver-

ters to meet the drive requirements.

3.3.3.4 RAM GENERATOR

Often in most VLSI applications, there is some memory incorporated into the chip. The RAM generator is a design tool the VLSI designer can use to generate the RAM needed by specifying certain parameters such as bit capacity, data width and the row-to-column ratio of the RAM. The RAM generated often resembles off-the-shelf RAM chips and can be inserted into the chip as a macro cell. The RAM generator is used more for this purpose than to generate a whole RAM chip. Besides, most RAM generators can only generate static RAMS with a maximum bit capacity of 16K. The on-chip memory in a VLSI chip is often small and typically has values of between 256 bits to 16K bits, depending on application. The RAM generator will automatically generate the layout of the specified RAM including the necessary decoders, sense amplifiers and line buffers.

3.3.3.5 SILICON COMPILERS

Up to now, most of the design methodologies looked at had the design entry point either along the structural or geometrical representation arms of the design space. An example of a methodology with an entry point along the structural representation arm would be the standard cell approach. This approach supports symbolic layout at the schematic level. The design is then translated into the geometrical representation through the use of the automatic placement and routing tools. An example of a methodology that has an entry point along the geometrical representation arm of the design space would be the full custom approach. This approach enters the design at the artwork layout level using a form of automated drafting system.

A silicon compiler is a design automation system that permits the designer to specify a design in a high-level behavioral language and then automatically generate the geometrical shapes on silicon to achieve the desired behavior of the specified system. This involves a front end that handles logic synthesis from the behavioral specification and then a back end that serves as a silicon assembler. Such a system in actuality unites a host of computer tools that individually tackle some particular level of design activity.

Silicon compilers have a design entry point along the functional respresentation arm of the design space. The silicon compiler then translates the functional representation of the design into the lower level geometrical representation (see Fig. 3.13). This translation includes logic synthesis which is the step involved in bringing the design from the functional representation to the structural representation. All the other design methodologies looked at before this did not include the logic synthesis stage -

the logic synthesis was done entirely by the human designer. However, in silicon compilers, the computer does this for the human designer.

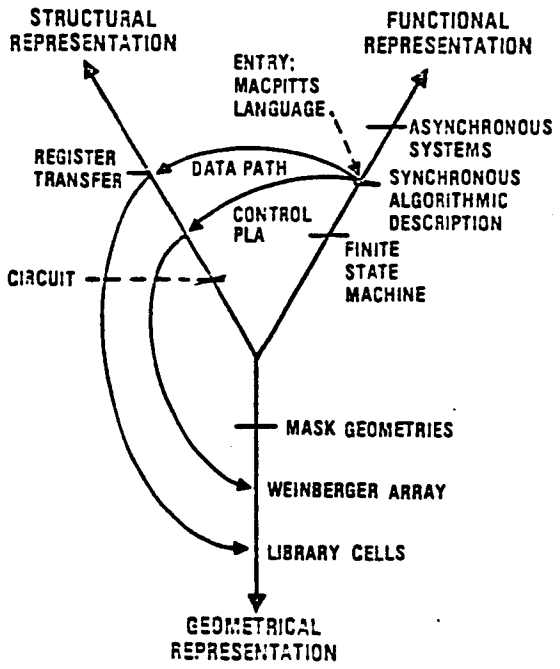


Fig. 3.13: Methodology of the MacPitts Silicon Compiler

Within the IC industry, there is some disagreement over the term 'silicon compiler'. The logic synthesis stage involves a number of different synthesis tools which some refer to as silicon compilers; while others reserve that term to include the entire compilation process - from design capture through mask creation. The disagreement arises from the difference between functional and structural representations. Most of the design automation tools transform a design from either the functional or structural representation of the design space to the geometrical representation. An ideal silicon compiler has the design entry in the form of a functional/behavioral representation. Fig. 3.14 illustrates some of the

computer tools that address the different design activities in the VLSI design process.

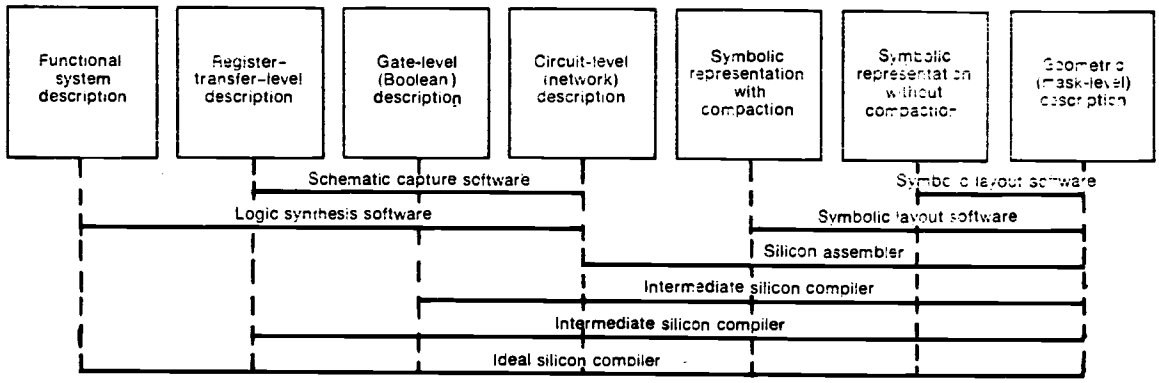


Fig. 3.14: Design Automation tools

The development of VLSI design methodologies very closely resembles the developement of software engineering. The structured and hierarchical concepts are shared by both disciplines. In fact, the silicon compiler was developed from the software compiler concept - to translate a high-level language into a lower level of information representation. In the case of the silicon compiler, the high-level language is in the form of an architectural description language such as ISP (Instruction Set Processor) [41]. There are some so-called silicon compilers that will accept a lower-level description (eg. circuit level description) of the design and translate it into the artwork layout; these systems are referred to as silicon assemblers (see Fig. 3.14).

Silicon compilers are fairly new tools to the IC design industry and most are still in the development stage. However, CAE companies are beginning to incorporate these compilers into their existing design workstations. Valid Logic Systems Inc.(San Jose, California) and Seattle Silicon Technology Inc.(Bellevue, Wash-

ington) have jointly developed seven silicon compilers for different analog and digital circuits based on a 3-um CMOS process. The compilers will run on the existing SCALD systems offered by Valid Logic [17]. Daisy Systems Corp.(Sunnyvale, California) is expected to introduce a silicon compiler that will run on its Chipmaster workstation. Mentor Graphics (Beaverton, Oregon) is working on a silicon compiler for their IDEA series as well. Silicon Compilers Inc. (Los Gatos, California) has a silicon compiler that is a descendant of the Bristle Blocks compiler developed by Dave Johannsen at the California Institute of Technology (Pasadena, California). Metalogic Inc.(Cambridge, Massachussets) has introduced the MetaSyn compiler which is a spinoff from the MacPitts compiler developed at M.I.T.'s Lincoln Laboratory. Lattice Logic Ltd. (Edinburgh, Scotland) has the Chipsmith silicon compiler that works on a high-level description language.

Two representative silicon compilers will now be presented to illustrate the mechanics involved in such systems. The first system is the Bristle Blocks silicon compiler developed at Caltech; which is the birth place of the silicon compiler concept. The Bristle Blocks system represents an academic effort at developing these tools. The second system is the commercially available MetaSyn silicon compiler from MetaLogic Corp.

3.3.3.5.1 BRISTLE BLOCKS

Bristle Blocks is a cell-based compiler system that will generate the artwork for an IC from a high-level description langauage. The user input to the compiler consists of three sections. The first states the microcode instruction width and format; the second relates to the data word width and the buses that run through the core of the chip; and the third

section specifies the elements that make up the chips core. The Bristle Blocks compiler is tailored to produce chips for a particular architecture [42] and adopts the Mead and Conway [23] methodology of structured design.

Bristle Blocks is a three pass compiler. The first pass is the core pass which constructs the core of the chip based on the user inputs and the low-level cell definitions. The second pass is the control pass which adds the instruction decoder circuitry. The final pass is the pad pass which adds the pads to the perimeter of the chip and completes the global connection of pads and the corresponding connections in the core and decoder.

The low-level cells in the library are left to the designer to layout in a standard-cell design language. This was not assigned to the compiler to perform because it is believed that the human designer can do a better job than computers; in the design of low-level cells, the designer is not overwhelmed by tons of detail and can apply human ingenuity to arrive at the best designs.

The cells in Bristle Blocks are procedural cells and differ from cells used in the standard cell approach. The standard cells are represented by a geometrical description language which presents a static picture of the circuits they describe in the form of rectangles on the mask levels. Porcedural cells are quite different in that they are little programs that do several things; including draw themselves, compute their stretch points for optimum layout [24] ; compute their power requirements; and also simulate themselves.

The compiler is able to generate the chip layout from a high-level description in a fairly short period of time - the

author of the system indicated times in the neighborhood of 10 - 15 minutes for fairly large chips [42]. The layout generated by Bristle Blocks requires approximately 10% more area than the hand drawn versions.

3.3.3.5.2 METASYN SILICON COMPILER

The MetaSyn silicon compiler can be defined as an ideal silicon compiler because it addresses all levels of the design process, beginning at the behavioral specification stage. It accepts a purely high-level behavioral description language and is able to generate the layout of the chip in CIF (Caltech Intermediate Format). The specification language supports parallelism and pipelining implementations in the hardware - the compiler will automatically generate the necessary hardware to implement the designer's specifications.

There are two kinds of behavioral simulation that MetaSyn supports before synthesizing the IC layout. The first is an interactive input simulator using a set of windows that monitor the major high-level blocks of the design: such as the registers, I/O ports, and labeled instruction states [43]. These are the major elements of the compiler's behavioral specification and the designer is able to observe and change these values through a mouse.

The second form of simulation is performed at the system level. In order to simulate a chip such as a processor design, it must be simulated in the context of a complete system - that is at the system level. The MetaSyn system-level simulator creates several LISP functions that can be used to simulate the other system components and use these to drive and sense the processor's activities.

The activity performed by the compiler is illustrated in Fig. 3.15. The input source description is subjected to a prepass stage that checks for syntax errors, expands macros and if the simulator option is selected, it generates the necessary LISP functions.

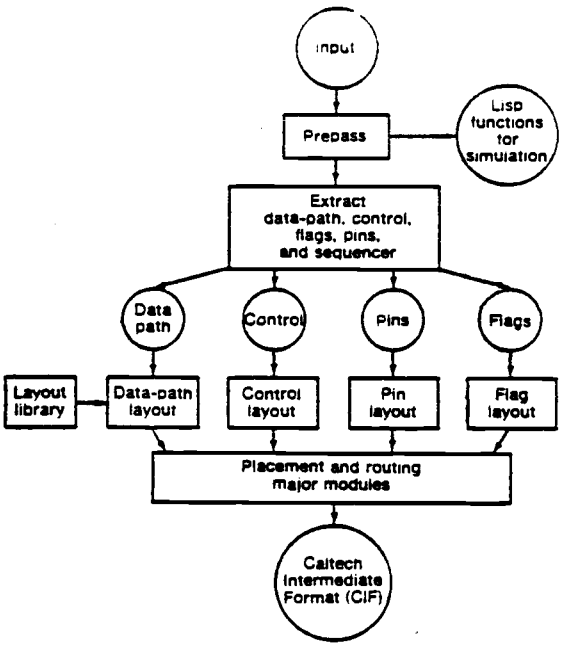


Fig. 3.15: MetaSyn Silicon Compiler activity.

The compiler then extracts the hardware components from the source specification. These components are grouped into the following major modules: Data-path, Control logic, Pad and flag modules. Then the groups are laid out using the Data-path, Control logic, Pad and Flag generators. The individual modules are then passed through an automatic placement and routing program to create the final layout in CIF.

Several chips have been designed using this silicon compiler, ranging from simple counters and shifters to signal-processing chips, computer peripheral controllers and micro-

processors such as Intel's 8080 and a PDP-8. Most of the layouts do not require more than 10% additional area than a full-custom layout of the same chip. The compilation time has ranged from a few minutes for small designs to 4 hours for the compilation of a RISC (Rreduced Instruction Set Computer) type chip. These times apply to the August 1984 version of the MetaSyn compiler running on a Symbolics 3600 LISP machine with 474 M bytes of disk storage and 1 M word of RAM [43].

CHAPTER 4:
THE FUTURE

The present-day design methodologies presented have mostly evolved from the SSI and MSI design process; with the CAE tools developed to assist in the design stages. These evolved methodologies are characterized by different computer tools for each stage of the design with each tool often running on a different system and having slightly different data representations. Often, the user would be faced with some form of patchworking to translate the design from one computer tool to another - not exactly user-friendly systems.

Silicon compilers and expert systems capable of chip synthesis are a positive step towards designing design methodologies and not simply following the path of methodology evolution. These areas of research hold the promise for future VLSI design methodologies. Turn-key systems that permit a system designer without much working knowledge of IC design to design a VLSI chip by simply specifying the design in a high-level language are realizable within the next 5 years. Two key areas of development are:

* Establishing an industry standard for design description languages. Currently at layout level, there is CIF, STIF, EDIF and a host of other layout description languages that are not even documented. Having a standard interchange language will permit designs to be portable among methodologies and allow design knowledge to be shared.

* Making the design methodologies as technology independent as possible. This will permit process refinements to be effected without much impact on existing chip designs. Using a silicon compiler, this may involve re-compiling the source code.

The area of VLSI design encompasses far more than was covered by this thesis. The topics presented were selected to give an overview of current design methodologies and CAE tools. It was not possible to present an indepth study of each topic in a thesis of this nature - the scope and material was overwhelming. Certain important and somewhat relevent topics were omitted because they could by themselves be areas of another thesis study - topics such as VLSI design complexity theory; design database management and design testability.

BIBLIOGRAPHY

[1] G.E. Moore, **"Progress in Digital Integrated Electronics"**, Tech. Digest, International Electron Devices Meeting, Washington D.C., 1975, pp 11-13.

[2] C.H. Sequin, **"Trends in Very Large Scale Integration and their impact on the designer"**, Design Methodologies for VLSI circuits (NATO Advanced Study Institute Series), Sijthoff and Noordhoff 1982.

[3] C.V. Ramamoorthy and Y.W. Ma, **"Impact of VLSI on Computer Architectures"**, VLSI Electronics Vol. 3, Academic Press 1982.

[4] S. Trimberger, J.A. Rowson, C.R. Lang and J.P. Gray, **"A Structured Design methodology and Associated Software Tools"**, IEEE Trans. on Circuits and Systems, Vol. CAS-28, No.7, July 1981.

[5] W.A. Dees, K.M. Parmar, A. Goyal, R.Y. Tsui, B.D. Rathi and R.J. Smith II, **"A Computer-aided VLSI layout system"**, National Computer Conference, 1981.

[6] D. Lewin, **"Computer-Aided Design of Digital Systems"**, Crane, Russak & Co., New York 1977, pp 117-124.

[7] **"General Purpose Digital Simulation and examples of its applications"**, IBM Systems J.3, 1964, pp 22-34.

[8] H.M. Markowitz, B. Hausner and H.N. Kerr, **"SIMSCRIPT - A Simulation Programming Language"**, Prentice Hall, Englewood Cliffs, New Jersey, 1963.

[9] M.E. Daniel and C.W. Gwyn, **"Hierarchical VLSI Circuit Design"**, IEEE Conference on Circuits and Computer, Vol.1, pp 92-97, 1980.

[10] C. Niessen, **"Hierarchical Design Methodologies and Tools for VLSI Chips"**, IEEE Proceedings, Vol.71, No.1, January 1983.

[11] C.H. Sequin, **"Managing VLSI Complexity: An Outlook"**, IEEE Proceedings, Vol.71, No.1, January 1983.

[12] D.D. Gajski and R.H. Kuhn, **"New VLSI Tools"**, Computer, December 1983, pp 11-14.

[13] J.G. Peterson, **"Keys to Successful VLSI System Design"**, VLSI Systems and Computations, Editors: H.T. Kung, B. Sproull and G. Steele, Computer Science Press, Maryland, 1981, pp 21-28.

[14] S.W. Director, A.C. Parker, D.P. Siewiorek and D.E. Thomas, Jr., **"A Design Methodology and Computer Aids for Digital VLSI Systems"**, IEEE Trans. on Circuits and Systems, Vol. CAS-28, No.7, July 1981.

[15] J.M. Siskind, J.R. Southard and K.W. Crouch, **"Generating Custom High Performance VLSI Designs from Succint Algorithmic Descriptions"**, Conference on Advanced Research in VLSI, M.I.T. 1982.

[16] M. Feuer, **"VLSI Design Automation: An Introduction"**, IEEE Proceedings, Vol.71, No.1, January 1983.

[17] M. Schindler, **"Advances in Software let System Engineers take charge of IC Design"**, Electronic Design, November 15, 1984, pp 128-140.

[18] D.F. Barbe, **"Very Large Scale Integration: Fundamentals and Applications"**, Springer-Verlag, New York 1980.

[19] H. Brown, C. Tong and G. Foyster, **"Palladio: An Exploratory Environment for Circuit Design"**, Computer, December 1983, pp 41-56.

[20] F. Hayes-Roth, **"The Knowledge-Based Expert System: A Tutorial"**, Computer, September 1984, pp 11-28.

[21] G. Williams, **"The Apple Macintosh Computer"**, BYTE Journal, February 1984, pp 30-54.

[22] D.E. Thomas, C.Y. Hitchcock III, T.J. Kowalski, J.V. Rajan and R.A. Walker, **"Automatic Data Path Synthesis"**, Computer, December 1983, pp 59-69.

[23] C. Mead and L. Conway, **"Introduction to VLSI Systems"**, Addison-Wesley, Reading Massachusetts, 1982.

[24] M. Marshall and L. Waller, **"VLSI pushes super-CAD techniques"**, Electronics, July 31, 1980, pp 73-80.

[25] R.L. Donze and G. Sporzynski, **"Masterimage Approach to VLSI Design"**, Computer, December 1983, pp 18 - 25.

[26] R.Y. Pinter, **"River Routing Methodology and Analysis"**, Third Caltech Conference on VLSI, Computer Science Press, 1983, pp 141-163.

[27] C.E. Leiserson and R.Y. Pinter, **"Optimal Placement for River Routing"**, VLSI Systems and Computations, Computer Science Press, 1981, pp 126-142.

[28] R.Y. Pinter, **"Optimal Routing in Rectilinear Channels"**, VLSI Systems and Computations, Computer Science Press, 1981, pp 160-177.

[29] W.S. Chan, **"A New Channel Routing Algorithm"**, Third Caltech Conference on VLSI, Computer Science Press, 1983, pp 117-139.

[30] D.N. Deutsch, **"A Dogleg Channel Router"**, Proceedings of the 13th Design Automation Conference, 1976, pp 425-433.

[31] Z. Kohavi, **"Switching and Finite Automata Theory"**, McGraw Hill, New York, 1978.

[32] J.R. Egan and C.L. Liu, **"Bipartite Folding and Partitioning of a PLA"**, IEEE Trans. on CAD of Integrated Circuits and Systems, Vol.CAD-3, No.3, July 1984, pp 191 -199.

[33] E.J. McCluskey, **"Programmable Arrays"**, Combinational Circuit Design, (text in preparation for publication), 1984.

[34] G.D. Hachtel, A.R. Newton and A.L. Sangiovanni-Vincentelli, **"Some results in Optimal PLA folding"**, Proc. Int. Conf. on Cicruits and Computers, pp 1023-1027, 1980.

[35] G.D. Hachtel, A.R. Newton and A.L. Sangiovanni-Vincentelli, **"An Algorithm for Optimal PLA folding"**, IEEE Trans. on CAD of Integrated Circuits and Systems, Vol.CAD-1, pp 63-76, 1982.

[36] I. Suwa and W.J. Kubitz, **"A Computer-Aided design system for segment-folded PLA macrocells"**, Proceedings of the 18th Design Automation Conf., pp 398-405, 1981.

[37] R.A. Wood, **"A high density programmable logic array chip"**, IEEE Trans. on Computers, Vol.C-28, pp 602-608, 1979.

[38] D.L. Greer, **"An asociative logic matrix"**, IEEE Journal on Solid-State Circuits, Vol.SC-11, pp 679-691, 1976.

[39] J.F. Paillotin, **"Optimization of the PLA area"**, Proceedings of the 18th Design Automation Conference, pp 406-410, 1981.

[40] Soo-Young Oh, **"MOS Device and Process Design Using Computer Simulations"**, Hewlett-Packard Journal, October 1982, pp 28-32.

[41] D.P. Siewiorek, C.G. Bell and A. Newell, **"Computer Structures: Principles and examples"**, McGraw Hill, New York, 1982

[42] D. Johannsen, **"Bristle Blocks: A Silicon Compiler"**, Proc. of the 16th Design Automation Conf., pp 310-313.

[43] J.R. Southard, **"Silicon Compiler demands no hardware expertise to fashion custom chips"**, Electronic Design, Hayden Publication, Nov.15, 1984, pp 187-200.