Preprint from Proceedings of the Summer School on VLSI Tools and Applications,
Switzerland, July 1986, Kluwer Acadmic Publishers
W. Fichtner and M. Morf editors.

# V L S I DESIGN STRATEGIES

*Carlo H. Séquin*

Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

*ABSTRACT*

The growing complexity of VLSI chips creates a need for better CAD tools and data management techniques. The rapidly changing nature of the field requires a modular toolbox approach — rather than a fixed monolithic design system — and the involvement of the designer in the tool-building process. A short overview over the Berkeley design environment and our recent Synthesis Project is also given.

## 1. INTRODUCTION

Very large scale integration (VLSI) has made it economically viable to place several hundred thousand devices on a single chip, and the technological evolution will continue to increase this number by more than an order of magnitude within a decade. While the limits on chip growth imposed by technology and materials are still another three orders of magnitude away,[1] the design of the present-day chips already causes tremendous problems. G. Moore coined the term "complexity barrier".[2] This is the major hurdle faced today in the construction of ever larger integrated systems. In Section 2 the nature of the VLSI complexity problem will be discussed.

In order to deal with this complexity and to exploit fully the technological potential of VLSI, some structure has to be introduced into the design process; the resulting design styles are reviewed in Section 3, and the general nature of the VLSI design process is discussed. The size of the task is such that it cannot be done without tools; new tools and new ways of managing the information associated with the design of a VLSI chip must be developed (Section 4). This changes the role of the designer (Section 5).

Section 6 illustrates with the example of the Berkeley Synthesis Project how we think the art of VLSI design is going to evolve.

## 2. VLSI COMPLEXITY

In the early 1980's. VLSI complexity became a hot topic for concern and discussion.[3] This may appear surprising if one compares the complexity of the chips of that period with other technological structures that mankind has built in the

past. Certainly the number of components on a VLSI chip does not exceed the number of parts in a telephone switching station or in the space shuttle, and mainframe computers with an even larger number of transistors have been built for at least a decade before they were integrated onto a chip. System complexity should not differ markedly whether a circuit is contained within a cabinet, on a printed circuit board, or on a single silicon chip.

It is the "large", potentially unstructured space of the VLSI chip that causes the concern. Nobody would dare to insert a million discrete devices into a large chassis using discrete point-to-point wiring. Large systems built from discrete devices are broken down into sub-chassis, mother-boards, and module-boards carrying the actual components. This physical partitioning encourages careful consideration of the logical partitioning and of the interfaces between the modules at all levels of the hierarchy. Since such systems are typically designed by large teams, early top-down decisions concerning the partitioning and the interfaces must be made and enforced rather rigidly — for better or for worse. This keeps the total complexity in the scope of each individual designer limited in magnitude, and thus manageable.

VLSI permits the whole system to be concentrated in a basically unstructured domain of a single silicon chip which does not *a priori* force any partitioning or compartmentalization. On the positive side, this freedom may be exploited for significant performance advantages. On the negative side, it may result in a dangerous situation where the complexity within a large, unstructured domain simply overwhelms the designer.

A similar crisis was faced by software engineers when unstructured programs started to grow to lengths in excess of 10,000 lines of code. The crisis was alleviated by the development of suitable design methodologies, structuring techniques, and documentation styles. Many of the lessons learned in the software domain are also applicable to the design of VLSI systems.[3]

Furthermore, the field of VLSI is rather interdisciplinary in nature. To achieve optimal results, we need a tight interaction of algorithms, architecture, circuit design, IC technology, etc. However, designers who are experts in all these fields are rarely found. How can ordinary mortals attempt to do a reasonable VLSI design? Here again, suitable abstractions have to be found, so that the details of processing are hidden from the layout designer, and the details of the circuit implementation are hidden from the microarchitect. Models that are accurate enough to permit sound decisions based on them need to be created, and clean interfaces between the various domains of responsibility need to be defined. For instance, the semantic meaning of the geometry specified in a layout has to be defined carefully: Is this the geometry of the fabrication masks? Is this the desired pattern on the silicon chip? Or is this a symbolic representation of some of the desired device parameters? These questions still lead to much discussion and often to bad chips. The emergence of silicon brokerage services such as MOSIS[4] has forced clarification of many of these issues.

## 3. THE DESIGN SPECTRUM

To make the task of filling the void on a VLSI chip manageable, some widely accepted abstractions have emerged that lead to a hierarchy (or rather a multidimensional space) of *views* of a particular design. The different representations generally address different concerns. A typical list of design levels and of the concerns they address is shown in Table 1.

| Design Level | Concerns Addressed |
|---|---|
| Behavior | Functionality |
| Functional blocks, Linked module abstraction[5] | Resource allocation sequencing, causality |
| Register-transfer level Clocked register and logic[5] | Testability timing, synchronization |
| Gate Level, Clocked primitive switches[5] | Implementation with proper digital behavior |
| Circuit Level | Performance, noise margins |
| Sticks Level | Layout topology |
| Mask Geometry | Implementation, yield |

**Table 1.** *Levels of abstraction in chip design.*

The other saving notion is that of prefabricated parts. The same functions at various levels of the design space are needed again and again. Successful designs of frequently used parts can thus be saved in libraries for the reuse by many customers. The nature of these parts and the level to which they are predefined or even prefabricated leads to a variety of different design styles.

### 3.1. VLSI Design Styles

A large spectrum of possibilities for the design of a VLSI chip has evolved, offering wide ranges of expected turn-around time, resulting performance, and required design effort. Table 2 gives a strongly simplified view of the spectrum of possibilities.

On one end of the spectrum are the Gate Array and Standard Cell technologies. Predesigned logic cells at the SSI and MSI level permit the engineers to use functional blocks that they are already familiar with from TTL breadboard designs. The abstraction and prefabrication of these cells lead to minimal design effort and faster turn-around time but at the price of less functionality per chip and less performance for a given technology.

| Method | Complexity | Effort | Main Strength | Automation |
|---|---|---|---|---|
| Gate Array | 20,000 | 4-8 weeks | Fast changes | Yes |
| Standard Cell | 40,000 | 4-8 weeks | Resuse of logic | Yes |
| Macro Cell | 100,000 | 1-2 years | Good area use | Almost |
| Flexible Modules | 200,000 | 1-2 years | High density | Almost |
| Standard Functions | 200,000 | 2-8 years | Testability | Not yet |
| Optimized Layout | 400,000 | $\geq$8 years | High performance | Not so soon |

**Table 2.** *Styles of IC chip design.*

At the other end of the spectrum is the full custom chip in which all modules have been hand-designed with the utmost care for performance and density and have been integrated and packed onto the chip in a tailor-made fashion. This design style can lead to spectacular results in terms of functionality and performance of an individual chip, but it comes at the price of an exorbitant design effort.

Somewhere in the middle between these two extremes are mixed approaches in which the crucial cells have been hand-designed with great care — particularly the cells that are in the critical path determining performance and the cells that are used in large arrays, as they will make the dominant contribution to the size of the chip. Uncritical "glue" logic that is used only once may be generated by a program either in the form of a PLA or as a string of standard cells. These macro cells of varying sizes and shapes are then placed and wired by hand or by emerging CAD programs.[6] This approach leads to higher densities than standard cells, since the degree of integration in the macro cells is typically higher and since a smaller amount of area is wasted in partly filled wiring channels. If the macro cells are procedurally generated and suitably parameterized so that they can be adjusted to the available space, even higher densities can be achieved. When properly used, these intermediate approaches can compete with a full custom design in terms of performance but typically result in a somewhat larger chip size.

Concerns of modularity and testability may outweigh aims for density and performance; functional modules designed for testability with clean interfaces are then used. This is in analogy to the use of properly abstracted and encapsulated software modules. This approach has started to gain acceptance also in VLSI. A good VLSI design environment will permit the designer to mix these various design styles in appropriate ways.

### 3.2. The Design Process

Even with an agreed-upon set of hierarchical levels, an extensive library of predefined parts, and a chosen design style, the design process can still be rather

involved. It is rarely a single forward pass through all the transformation steps that takes a high-level behavioral description through register-transfer and logic level descriptions into a symbolic representation capturing the topology and finally into a dense layout suitable for implementation with a particular technology (Table 1). The overall problem may be structured in a top down manner into simpler subtasks with clearly defined functions. But in parallel, designers intimately familiar with the implementation technology will explore good solutions for generic functions in the given technology in a bottom-up fashion. This effort will result in an understanding of what functions can best be implemented in this environment and produce a set of efficient building blocks.

Hopefully, the top-down decomposition and the bottom-up provision of solutions will meet in the middle and permit completion of the design. However, for a new technology, it is unlikely that this will happen on the first try. The natural building blocks must first be discovered; only then can the architectures be modified and partitioned appropriately. Thus there is an iteration of top-down and bottom-up moves in a Yo-Yo like fashion until the optimal path linking architecture to technology has been found.

It should also be pointed out that the design process is often a mixture of solid established procedures and of free associations and 'trial-and-error'. The guessing part plays a role in finding good partitioning schemes as well as in the definition of generic functions that might constitute worthwhile building block in the given technology. Proven checking methods are then used to evaluate objectively whether the guesses made are indeed usable: Is the decomposition functionally correct? Is it appropriate — or does it cut through some inner loop, causing unnecessary communications penalties? Are the building blocks of general use? How many algorithms, tasks, or architectures can actually make use of them? Is their performance reasonable?

In the next section we explore to what extent this design process can be supported by the computer, and for which part the human intelligence might be hard to replace.

## 4. THE ROLE OF CAD TOOLS

Good tools help man to achieve more, to obtain better results, or to reach given goals more effortlessly. VLSI design is no exception. I like to split the CAD tools useful in the design of ICs into five classes:

1) *Checking and Verification Tools* typically answer questions such as: Are there any errors? Are the connections between blocks consistent? Does this function behave as specified?

2) *Analysis Tools* tell the designer: How well does a particular approach work? How much power does this circuit consume? What is the worst case settling time?

3) *Optimization Tools* can help the designer to vary component values to achieve a specific performance goal, or they can find "optimal" module placements within given constraints.

4) *Synthesis Tools* combine construction procedures and optimization algorithms. They may decompose a logic function into a minimum number of gates, or they may find a good floor plan from a connectivity diagram.

5) *High-level Decision Tools* support the designer in the "guessing phase" of the design process. These tools try to suggest particular solutions, i.e., partitioning schemes, micro architectures, or network topologies.

### 4.1. The CAD Wave

Building tools in the above classes 1) through 5) gets progressively more difficult. Typically, checking and verification tools are the first to become viable, helping to eliminate well-defined mistakes. Next, analysis tools permit the designer to find out how good a solution he has chosen and whether the design meets specifications. Based on the analysis algorithms, optimization tools emerge, assisting the designer in fine-tuning a design and in optimizing particular aspects of it. Gradually, these tools evolve into self-reliant synthesis tools; these may use heuristic methods or simulated annealing techniques to find solutions that are becoming competitive in quality with the work of human designers. Finally the tools will invade the areas where it is most difficult to replace the human mind — the high-level decision making process. Here tools from all the previously mentioned classes need to be employed in an iterative way; often techniques from the field of *Artificial Intelligence* are used.

Sweeping through the various levels of the design hierarchy in a bottom-up manner, tools will start to take over the function of the human designer. This general trend has started many years ago. Historically, the first tools to be developed for IC designers were circuit analysis tools such as SPICE.[7] There was a real need for such tools, since calculating the performance even of small integrated circuits would have been too tedious, and including actual fabrication of the chip in every design iteration would have been too slow and costly. At that time, the circuits were small enough so that most checking tasks could be performed without computer assistance. Optimization was done by hand with the help of the available analysis tools. Design decisions were largely based on the intuition or experience of the designer.

In the meantime, tools have matured at the layout level. Circuit extractors and design rule checkers are relied upon by every designer of large ICs. Without timing verification and circuit simulation, it would be impossible to obtain chips that meet performance specifications. Circuit optimization, however, is still largely done by the designer, using analysis tools in the "feedback loop", and synthesis tools are being investigated in the research laboratories.

At the higher levels of the design hierarchy tools have not claimed as much ground yet. Functional simulators are used to verify the correctness of the functional behavior and to obtain some crude idea of the expected performance. Optimization and synthesis tools are the subject of active research. High-level decision tools are being contemplated.

Tools are important at all the levels of the design hierarchy introduced in Section 3. The development of CAD tools started at the circuit level, because there the need was most urgent. This was the level of abstraction that could not easily be breadboarded and evaluated by measurement. As larger and larger systems get integrated onto a single chip, we will need better tools also at the higher levels in the design hierarchy.

### 4.2. Design Representation

Traditionally, many design systems for custom circuitry have used the geometrical layout information to "glue" everything together. From this low-level description that other representations are derived, and many of the analysis start from this level, e.g. circuit extraction and design rule checking. This is an unsatisfactory approach. Too much of the designer's intent has been lost in that low-level representation and has to be rediscovered by the analysis tools.

If there is to be a "core" description from which other representations are derived, it has to be at a higher level. The trend is to move to a symbolic description[8-10] that is still close enough to the actual geometry, so that ambiguities in the layout specification can be avoided. Yet at the same time, this description must have provisions to specify symbolically the electrical connections and functional models of subcircuits.[11]

In the long run, there is no way that a proper, integrated data management system can be avoided. Such a system can capture the design at various levels of the design hierarchy and, with the help of various tools, ensure consistency between the various representations. An integrated tool system will have to support the mentioned Yo-Yo design process in order to be effective.

### 4.3. Tool Integration

The art of VLSI design is not yet fully understood, and new methodologies are still evolving. It is thus too early to specify a rigid design system that performs the complete design task; quite likely, such a system would be obsolete by the time it becomes available to the user. It is more desirable to create a framework that permits the usage of many common tools in different approaches and that supports a variety of different design styles and methods. In short, the environment should provide *mechanisms* and *primitives* rather than *policies* and *solutions*.

Intricate interaction between the various tools must be avoided; every tool should do one task well and with reasonable efficiency.[12] The tools are coupled

through compatible data formats or a joint data base to which they all interface in a procedural manner. The former solution causes less overhead in the early development phase of a new tool and makes it easy for workers in different locations to share data and test examples since ASCII text files can easily be transmitted over electronic networks.

The data base approach leads to a more tightly coupled system. It has the advantage (or disadvantage) that all data is in one central location. Interfacing a tool to this data base is normally more involved and costly than to simply read and write ASCII files. Unless the data management system is properly constructed and supported, the access to the data base can also get painfully slow. A practical solution is to use a combination of both: a data base that also has proper ASCII representations for each view of the design.

At Berkeley such a collection of tools[13] has been under development since the late 1970s. All tools are embedded in the UNIX[14] operating system. UNIX already provides many of the facilities needed in such an environment: a suitable hierarchical *file structure*, a powerful monitor program in the form of the UNIX *shell*,[15] and convenient mechanisms for *piping* the output of one program directly into the input of a successor program. An example of a newer, object oriented data base[16] will be discussed briefly in Section 6. The corresponding ASCII representation and interchange format is EDIF.[17]

Regardless of the exact structure of the data base, the various different representations of a design should be at the fingertips of the designer, so that he can readily choose the one representation that best captures the problem formulation with which he is grappling at the moment.

## 5. THE ROLE OF THE DESIGNER

The wave of emerging CAD tools at all levels of the design hierarchy is changing the role of the designer.

### 5.1. The CAD System Virtuoso

Designers of solid-state systems will spend an ever smaller fraction of their time designing at the solid-state level. More and more technical tasks, particularly at the lower levels of the design hierarchy, can be left to computer-based tools. Systems designers will rely increasingly on tools and on prototype modules generated by expert designers. They will thus change from being technical designers to being players of a sophisticated and rich CAD system.

It will take effort to learn the new skills. The essential experience no longer consists in knowing how to best lay out a Schottky-clamped bipolar gate, but rather in choosing the right tool, setting the right parameters and constraints, using a reasonable number of iterations, or knowing what to look for in a simulation producing a wealth of raw data.

The results obtainable with any CAD system depend to a large extent on the skill with which the designer moves through the maze of options. Furthermore, many of our design tools are still in the state corresponding to the early days of the automobile, where the driver also had to be a mechanic and be prepared to take care of frequent breakdowns.

### 5.2. The Designer as a Tool Builder

Good tools cannot be constructed in an isolated CAD department. They must be built in close relationship with the user. Who is better qualified than the actual user to understand the needs for a tool and to test whether a new tool really meets expectations? Further, a good CAD tool cannot be built in a single try. Only after the designers have a prototype to play with, they can decide what they really need and provide more accurate specifications for the new tool. The emergence of a tool often changes the nature of the job enough to shift the emphasis to a different bottle neck, thus altering the requirements for the tool. This in turn may necessitate a revision of the user interface or the performance targets. This iterative process to arrive at the proper specifications leads to the tool development spiral shown in Figure 1.

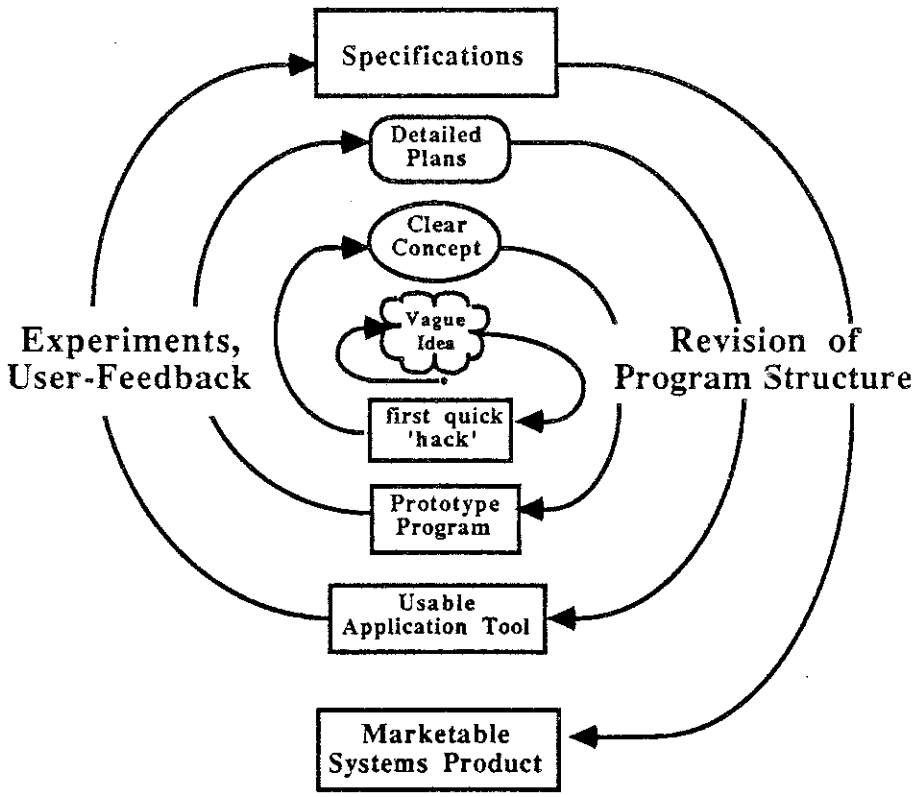


**Figure 1.** *The spiral of tool development.*

Each implementation serves as the basis for clearer specifications for the next round. The more rounds one can make around this spiral, the better the tool will get. In going from one round to the next, one should not be afraid to start completely from scratch, to throw out the old code, and to keep nothing but the experience and plans for an improved approach. The temptation to just patch up the old code can be reduced if the implementation language is switched. Many tool developers have found it productive to create early prototypes in LISP, SmallTalk, or Prolog, and to code later versions in an efficient procedural language providing some control over the machine resources.

The first one or two turns on the development spiral are crucial. This is where the general directions of a new tool are determined. On later turns it is much harder to make significant conceptual changes. Thus, on the first turn(s) it is particularly important that the development is done in close contact with the designers actually using the tool. How much closer can you get than having the designer himself do the first "quick hack"? Nowaday, more and more engineers receive a good education in programming, and it is thus easier to find persons with the right combination of skills.

Once the framework of the tool is well established and the user interface defined, a formal CAD group could take over to recode the tool, modularize the program, look at efficiency issues, and provide decent documentation. In the production of a good manual, the designers must again be strongly involved, as they understand the needs of the users.

The most leverage out of human ingenuity can be obtained if the latter is used to build new and better tools, which then can help many other designers to do the job better or faster. Using the designers as tool-builders, the impact of the work of individual engineers can be compounded.

### 5.3. The Design Manager

A good manager, will not only focus on getting the job done on time, but will also concentrate on creating an environment in which the job can get done most efficiently. The improvement of the environment must not be neglected under the pressures of immediate deadlines; it is a necessary investment for the future.

This is also true for the individual design engineer, as he too is a manager of his task, his time, and his environment. This requires a change of attitude on the part of the typical engineer. He may have to spend a larger amount of time, learning about available tools, acquiring new tools, or building tools himself, than working on 'the job'. But experience has shown that, amortized over two or three jobs, this investment into the environment pays off.

## 6. THE SYNTHESIS PROJECT AT BERKELEY

In Spring 1986, an ambitious project concerning the automatic synthesis of integrated circuits was undertaken in the Department of EECS at Berkeley. The "official" goal of the project was to integrate and enhance our various CAD tools to create a suit of tools that could synthesize a complex microcomputer from a behavioral-level description to the mask-level output with as little manual intervention as possible. As a "fringe benefit" we expected to gain a thorough understanding of the major issues in IC synthesis and to find out where our CAD tool design efforts need to be focussed.

### 6.1. Project Organization

The Synthesis Project was led by Professors Newton, Sangiovanni-Vincentelli and Séquin together with seven visiting Industrial Fellows. Following a tradition in our department, the project was tightly integrated with our graduate instruction. During the Spring term of 1986, the project was carried by two graduate courses, a design-oriented class (CS 292H) and a CAD tool-oriented class (EECS 290H), both of which had to be taken by all 35 participating students.

The tool development was tied to the SPUR (Symbolic Processing Using RISCs) project[18] which had been in progress for about a year. The main focus of the SPUR project was the development of a set of three chips: the central RISC processor (CPU), a cache controller (CCU), and a floating point coprocessor (FPU), for use in a multiprocessor workstation. We planned to use the architects and original implementors of these chips as consultants and hoped to obtain large parts of chip descriptions in machine-readable form.

A matrix organization was adopted for the graduate-student design teams. Each student, as a participant in the *design class*, was involved in the design of one of the three chips and was responsible for the generation of at least one specific module. As a participant in the *CAD class*, each student was a member of one of several tool development groups (e.g., logic synthesis, place and route, module generation) and was working towards developing a tool suite that would be usable for all three chips.

### 6.2. Resources and Infrastructure

Resources available to the Synthesis Project included a dozen DEC VAXstationII workstations and seven color VAXstationII/GPX machines. The backbone was a VAX8650 CPU with 500 Mbytes of disc storage dedicated to the course. This machine acted as the central database and as the repository for all the existing and emerging CAD tools. All these machines, as well as all the other computing resources in the department, were coupled through an ethernet, creating a tightly coupled, highly interactive computing and communications environment.

Important software support was provided by the Digital Equipment Corporation in the form of the DECSIM mixed-level simulator and its associated behavioral design language, BDS. This software package was chosen primarily because of its availability and because DEC personnel were on site to provide support for its application. DECSIM also offered the possibility of using mixed-level simulation at the behavior, register-transfer, gate-logic, or switch level — even though during the course we did not get far enough to use all these options.

For the integration of our tools we chose to use a single object-oriented data management system, OCT,[16] the development of which had started some time ago. OCT has as its basic unit the 'cell' which can have many 'views' — physical, logical, symbolic, geometrical. A cell is a portion of a chip that a designer wishes to abstract; it can vary in size from a simple transistor to the entire floorplan of a CPU. The system is hierarchical, i.e., cells can contain instances of other cells. Moreover, cells can have different abstract representations depending on the intended application, and these are represented in OCT by 'facets', which are the accessible units that can be edited. OCT provides powerful constructs for complex data structures but manages this complexity unseen by the user.

A graphical CAD shell, VEM, was developed that permits the user to inspect and alter the contents of the various cells in the data base in a natural manner. OCT also provides project management support in the form of change-lists, time stamps, and search paths. All evolving synthesis tools were provided with interfaces to the OCT data manager.

### 6.3. Module Generation Tools

One major effort during the Synthesis Project concerned the creation of a module generator that transforms logic equations at the behavioral level into a final mask layout. The important representation levels and the tools that perform the transformations between them are shown in Table 3.

Module generation starts from a DEC BDS behavioral description which is converted with the help of a language translator into BDSYN, a subset of BDS, developed to represent logic partitioned into combinational blocks and latches. From there, another translator maps the BDSYN description into BLIF, the Berkeley Logic Intermediate Format, by expanding high-level constructs into Boolean equations.

MIS, a multilevel interactive logic synthesis program, then restructures the equations to minimize area and to attempt to satisfy timing constraints. MIS first implements global optimization steps that involve the factoring of Boolean equations and multiple-level minimization. Local optimization is then performed to transform locally each function into a set of implementable gates. Finally, MIS includes a timing-optimization phase that includes delay approximation based on technology data and critical-path analysis.[19]

| Design Function | Representation Level | Program Name |
|---|---|---|
| | Logic / Behavior | |
| Logic Synthesis | | BDSYN<br>MIS |
| | Logic / Gate | |
| Topology Optimization | | TOPOGEN / EDISTIX<br>GENIE / MKARRAY |
| | Symbolic / Graphic | |
| Layout Generation | | SPARCS<br>ZORRO |
| | Layout / Geometry | |

**Table 3.** *Transformations in the module generation process.*

Once the logic equations have been optimized the module generators are responsible for optimal packing of the logic into regular or irregular array-based structures.[20] Some of these tools also consider slack times for critical paths.

TOPOGEN generates a standard-cell-like layout at the symbolic level from a description of a Boolean function in the form of nested AND, OR, INVERT expressions. A complex static CMOS gate is produced in which first the transistors and then the gates have been arranged so as to minimize the module area. The output from TOPOGEN can be inspected and modified with EDISTIX, a graphic editor using a symbolic description on a virtual grid.[10] The symbolic layout can then be sent to one of the compactors mentioned below.

A more sophisticated module generator is the combination of GENIE and MKARRAY. GENIE is a fairly general software package using simulated annealing to optimize the topology of a wide range of array design styles, including PLAs, SLAs, Gate Matrix, and Weinberger arrays. It handles nonuniform transistor dimensions, allows a variety of pin-position constraints, approximates desired aspect ratios by controlling the degree of column folding, and performs delay optimization. Its output is sent to the array composition tool, MKARRAY, which takes specifications of arrays of cells at the topological level. It then places the cells and aligns and interconnects all the terminals.

The modules at the symbolic level have to be spaced or compacted to a dense layout obeying a particular set of design rules.[21] SPARCS is a new constraint-based IC compaction tool that provides an efficient graph-based solution to the spacing problem. It can deal with upper bounds, user constraints, even symmetry requirements. It detects of over-constrained elements, and permits adjustable positioning of noncritical path elements

Another compactor under development, ZORRO,[22] works in two dimensions and is derived from the concept of *zone refining* used in the purification of crystal ingots. ZORRO passes an *open zone* across a precompacted layout. Circuit elements are taken from one side of this zone and are then reassembled at the other side in a denser layout.[10] This compactor gives denser layouts at the cost of longer run times.

### 6.4. Chip Assembly tools

All the tools described above are employed in the automatic synthesis of modules that are to be used in the design of an entire chip. Various tools have been developed to perform module placement, channel definition and ordering, global routing, and finally detailed routing.[6] These tools handle routing on multiple layers as well as over-the-cell wiring. Table 4 shows the sequence of transformations carried out on the representations in the OCT database from the original tentative floor plan to the final placement of all the modules and of the wiring in-between.

| Layout Function | | OCT Symbolic View |
|---|---|---|
| Floorplanning & Placement | ---> | Placed |
| Channel Definition and Ordering | ---> | Channel Defined |
| Global Router | ---> | Routed |
| Detailed Router | ---> | Unspaced |
| Spacing-Compaction | ---> | Spaced |

**Table 4.** *Functions of the chip composition tools.*

The TIMBERWOLF-MC[23] package performs the placement function using simulated annealing techniques. This program handles cells of arbitrary rectilinear shape; it accommodates fixed or variable shapes with optional bounds on aspect ratio, and accepts fixed, constrained, or freely variable pin locations.

CHAMELEON[24] is a new multi-level channel router that allows the specification of layer-dependent pitch and wire widths. It has as its primary objective the minimization of channel area and as its secondary objective the minimization of the number of vias and the length of each net. On two-layer problems it performs as well or better than traditional channel routers.

MIGHTY[25] is a 'rip-up and reroute' two-layer detailed switch-box router that can handle any rectagon-shaped routing region with obstructions and pins positioned on the boundary as well as inside the routing region. It outperforms all the known switch-box routers and even performs well as a channel router on problems with a simple rectangular routing region.

### 6.5. Results

Fifteen weeks is not enough time to build a complete synthesis system — thus we could not "press the button" on the last day of class and watch the layouts for the three SPUR chips pop out of the computer.

After the fifteen-week course period, all three chip designs had been converted from their original descriptions in 'N.2' or SLANG formats to BDS and inserted into our data management system. In the last few weeks of the course, these descriptions were then used to exercise the pipeline of tools that had been created in parallel. Major parts of these designs have run through various tool groups and produced results of widely varying quality. Improvements were quite visible as the tools were debugged and improved.

The major benefit of this course is a very good understanding of the bottlenecks and missing links in our system and concrete plans to overcome these deficiencies. Over all, the Synthesis Project of Spring 1986 must have been a positive experience; the students polled at the end of the term voted strongly in favor of continuing the Synthesis Project in the Fall term.

## 7. CONCLUSIONS

There is a broad spectrum of design styles that have proven successful for the construction of VLSI circuits and systems. For all these styles and for all the levels in the design hierarchy, good computer aided tools and data management techniques are indispensable. The emerging wave of CAD tools shows a trend to start at the lower hierarchical levels and to move upwards and to sweep the verification and analysis tools before the synthesis and high-level decision making tools. There is no doubt that eventually the whole design spectrum will be covered.

To make the emerging tools truly useful, the new tools should be developed in close cooperation with the user, or even by the user himself. Several iterations are normally needed to produce a good tool. The development of tools should be planned with this in mind.

Due to the changing nature of VLSI design, a design system will never be "finished". In order to keep up with the needs of the chip designers, the environment and the data representations must be kept flexible and extensible. A modular set of tools coupled to an object-oriented, integrated data base is a good solution.

Finally, we believe that the most effective tool development takes place under the forcing function of actual designs. In a recent push to integrate and complete our synthesis tools at Berkeley, we have used the chip set of an emerging VLSI-based multiprocessor workstation. This effort has given us a clear understanding of the tools that we are still missing. It has charted out enough work to keep us busy for several more years.

## ACKNOWLEDGEMENTS

The development of our CAD tools is strongly supported by the Semiconductor Research Corporation.

### References

1. J.D. Meindl, "Limits on ULSI," *Keynote Speech at VLSI'85, Tokyo*, Aug. 26, 1985.
2. G.E. Moore, *Quote at the First Caltech Conf. VLSI*, Pasadena, CA, Jan. 1979.
3. C.H. Séquin, "Managing VLSI Complexity: An Outlook," *Proc. IEEE*, vol. 71, no. 1, 1983.
4. MOSIS, "MOS Implementation Service," DARPA-sponsored 'Silicon Brokerage Service' at the University of Southern California's Information Sciences Institute , since 1981.
5. M. Stefik, D.G. Bobrow, A. Bell, H. Brown, L. Conway, and C. Tong, "The Partitioning of Concerns in Digital Systems Design," *Proc. Conf. on Adv. Research in VLSI*, pp. 43-52, M.I.T., Cambridge, MA, Jan. 1982.
6. J. Burns, A. Casotto, G. Cheng, W. Dai, M. Igusa, M. Kubota, U. Lauther, F. Marron, F. Romeo, C. Sechen, H. Shin, G. Srinath, H. Yaghutiel, A.R. Newton, A.L. Sangiovanni-Vincentelli, and C.H. Séquin, "MOSAICO: An Integrated Macrocell Layout System," *submitted to ICCAD-86*, Santa Clara, CA, Nov. 1986.
7. L.W. Nagel and D.O. Pederson, "Simulation Program with Integrated Circuit Emphasis," *Proc. 16th Midwest Symp. Circ. Theory*, Waterloo, Canada, April 1973.
8. J.D. Williams, "STICKS - A Graphical Compiler for High Level LSI Design," *AFIPS Conf. Proc., NCC*, vol. 47, pp. 289-295, 1978.
9. N. Weste and B. Ackland, "A Pragmatic Approach to Topological Symbolic IC Design," *Proc. VLSI 81*, pp. 117-129, Academic Press, Edinburgh, Aug. 18-21, 1981.
10. C.H. Séquin, "Design and Layout Generation at the Symbolic Level," in *Proceedings of the Summer School on VLSI Tools and Applications*, ed. W. Fichtner and M. Morf, Kluwer Acadmic Publishers, 1986.
11. S.A. Ellis, K.H. Keller, A.R. Newton, D.O. Pederson, A.L. Sangiovanni-Vincentelli, and C.H. Séquin, "A Symbolic Layout Design System," *Int. Symp. on Circuits and Systems*, Rome, Italy, May 1982.
12. S. Gutz, A.I. Wasserman, and M.J. Spier, "Personal Development Systems for the Professional Programmer," *Computer*, vol. 14, no. 4, pp. 45-53, April 1981.
13. A.R. Newton, D.O. Pederson, A.L. Sangiovanni-Vincentelli, and C.H. Séquin, "Design Aids for VLSI: The Berkeley Perspective," *IEEE Trans. on Circuits and Systems*, vol. CAS-28, no. 7, pp. 666-680, July 1981.
14. B.W. Kernighan and J.R. Mashey, "The Unix Programming Environment," *Computer*, vol. 14, no. 4, pp. 12-22, April 1981.
15. S.R. Bourne, "UNIX Time-sharing System: The UNIX Shell," *Bell Syst. Tech. J.*, vol. 57, no. 6, pp. 1971-1990, Jul.-Aug. 1978.

16. D. Harrison, P. Moore, A.R. Newton, A.L. Sangiovanni-Vincentelli, and C.H. Séquin, "Data Management in the Berkeley Design Environment," *submitted to ICCAD-86*, Santa Clara, CA, Nov. 1986.

17. The EDIF User's Group, "EDIF, Electronic Design Interchange Format, Version 1.0," Technical Report, Design Automation Department, Texas instruments, Dallas, TX, 1985.

18. M.D. Hill, S.J. Eggers, J.R. Larus, G.S. Taylor, G. Adams, B.K. Bose, G.A. Gibson, P.M. Hansen, J. Keller, S.I. Kong, C.G. Lee, D. Lee, J.M. Pendleton, S.A. Ritchie, D.A. Wood, B.G. Zorn, P.N. Hilfinger, D.A. Hodges, R.H. Katz, J.K. Ousterhout, and D.A. Patterson, "SPUR: A VLSI Multiprocessor Workstation," CS Division Report No. UCB/CSD 86/273, University of California, Berkeley, CA, 1986.

19. R. Brayton, A. Cagnola, E. Detjens, K. Eberhard, S. Krishna , P. McGeer, L.F. Pei, N. Phillips, R. Rudell, R. Segal, A. Wang, R. Yung, T. Villa, A.R. Newton, A.L. Sangiovanni-Vincentelli, and C.H. Séquin, "Multiple-Level Logic Optimization System," *submitted to ICCAD-86*, Santa Clara, CA, Nov.1986.

20. G. Adams, S. Devadas, K. Eberhard, C. Kring, F. Obermeier, P.S. Tzeng, A.R. Newton, A.L. Sangiovanni-Vincentelli, and C.H. Séquin, "Module Generation Systems," *submitted to ICCAD-86*, Santa Clara, CA, Nov. 1986.

21. J.L. Burns, T. Laidig, B. Lin, H. Shin, P.S. Tzeng, A.R. Newton, A.L. Sangiovanni-Vincentelli, and C.H. Séquin, "Symbolic Design Using the Berkeley Design Environment," *submitted to ICCAD-86*, Santa Clara, CA, Nov. 1986.

22. H. Shin and C.H. Séquin, "Two-Dimensional Compaction by Zone Refining," *Proc. Design Autom. Conf., Paper 7.3*, Las Vegas, July 1986.

23. C. Sechen and A. Sangiovanni-Vincentelli, "TIMBERWOLF 3.2: A New Standard Cell Placement and Global Routing Package," *Proc. Design Autom. Conf., Paper 26.1*, Las Vegas, July 1986.

24. A. Sangiovanni-Vincentelli, D. Braun, J. Burns, S. Devadas, H.K. Ma, K. Mayaram, and F. Romeo, "CHAMELEON: A New Multi-Layer Channel Router," *Proc. Design Autom. Conf., Paper 28.4*, Las Vegas, July 1986.

25. H. Shin and A. Sangiovanni-Vincentelli, "MIGHTY: A 'Rip-up and Reroute' Detailed Router," *submitted to ICCAD-86*, Santa Clara, CA, Nov. 1986.