THE EVOLUTION OF DESIGN AUTOMATION TO MEET THE CHALLANGES OF VLSI

Lawrence M. Rosenberg

RCA Laboratories
Solid State Technology Center
Somerville, NJ 08876

Abstract

This paper presents the author's opinion of the major problems confronting Design Automation for VLSI and how Design Automation may evolve to meet these challenges. The paper first takes a historical look at the driving forces for Design Automation development by analyzing the evolution of Design Automation at RCA. It looks at both some successful and unsuccessful development efforts and attempts to isolate some of the criteria necessary for success. It review RCA's current LSI Design Automation capabilities and compares them to the challenge of VLSI. The major challenges -- layout, design verification and testability -- are discussed along with possible achievable solutions.

Introduction

The issue challenging Design Automation for VLSI is learning how to cope with the continuing explosion of complexity of IC's. Although the challenges facing technology development are substantial -- lithography, both imaging and registration, etching fine lines, implantation, multilevel fine-line interconnect, and so on -- there is no evidence that there is any fundamental physical barrier preventing IC complexity from continuing to grow exponentially in time -- doubling in device count per chip every two years or so for the next decade or more until design rules of order one-quarter to one-half micron are achieved [1]. It is the view of many observers including myself that the rate limiting factor constraining the growth of IC product complexity may well be the Design Automation tools required for design (system, logic, circuit, and process) and its verification, for physical implementation (layout) and its verification, for test generation and its verification (fault coverage) and for test data analysis. Each of these present substantial challenges to the Design Automation tool builders and their users because of the large and growing amount of data involved.

In order to gain some perspective of where we are going and how we may get there, it may be useful to first look at where we have been, where we thought we were going and where we have actually arrived, i.e., at today's capabilities. It would also be valuable to ask what were the driving forces that got us here. I strongly suspect that history, i.e., Design Automation's track record, does provide good basis for extrapolating into the future. Our past experience should temper, not dampen, our view of the future, and our enthusiasum for it.

As I am certainly not a Design Automation historian, my views will be heavily influenced by my ten years of experience at RCA, my limited knowledge of the literature and my extensive discussions with fellow Design Automation professionals. In fact, my discussions will center around the Design Automation experience at RCA; I suspect that although details will surely differ, the general experience and insights are quite representative and applicable to others.

In this discussion, I will contrast two idealized driving forces for Design Automation development: the DREAM and the NECESSITY; these terms will be defined in context. It of course is realized that in reality a combination of the two always exist. However, I believe that usually one or the other is dominant, and which one, determines to a large extent, the ultimate success of the Design Automation development effort. This too should become clearer from the context of the discussion.

For the purposes of this paper, please allow me to operationally define VLSI as the IC complexity of the (near) future, i.e., one-to-two orders of magnitude more complex than our Design Automation systems can comfortably handle today.

In addition, because this paper addresses the issue of Design Automation for VLSI, it is from the perspective of a commercial semiconductor house, not a system house, custom vendor, research laboratory, or university.

Some History

Where to begin? I start in 1970 -- the year of the Seventh Annual DAC (then called a workshop), so clearly not at the beginning. Extensive work has gone on in areas such as interactive graphics, automatic layout, and circuit and logic simulation. However, in my view, at this point in time, most of the enthusiasum for Design Automation evolved from the Design Automation tool builders at locations such as large corporate research centers, not from the design community. During this time, the driving force behind Design Automation was the DREAM, not the NECESSITY.

I suspect the most useful Design Automation tools were built by the circuit designers themselves at custom facilities such as system houses. This is because the low volume and quick turnaround required of custom designers forced them to find computerized solutions which gave them design leverage. That the techniques developed, such as the standard cell automated layout approach (described in more detail later), substantially increased chip size (and hence incremental cost) was not an issue because of the low volume of production required.

However, it was just this penalty which, justifiably, discouraged commercial semiconductor houses -- the main focus of this paper -- from pursuing this approach. In fact, to them IC complexity was modest enough (perhaps 100-500 devices) so that layout aids, logic simulation and circuit simulation were not viewed as crucial or even necessary. In fact, I believe these tools were viewed as too inaccurate and expensive to be cost-effective.

In actuality, for the commercial semiconductor house, rubylith cutting for photomask generation was simple, adequately fast and inexpensive. Although work was going on in Interactive Graphics, what could one do with the resulting digitized mask-artwork polygons except use computerized ruby cutters. (Photoplotters, capable of making 80X foils, were in use at various government installations, such as RCA's Government Systems Division, as far back as 1967 or so. It was fed by the Design Automation system which automatically laid out standard cell chips. However, I know of no semiconductor house using them at that time.) Digitizing these polygonal shapes was tedious and time consuming. Worse, editing of these computer representations was tedious if done on a source language format or expensive: interactive graphics required very expensive hardware because memory and other essential hardware were expensive in those dark ages; only CAD researchers could afford them. I suspect that it was the advent of and necessity for the pattern generator that really brought Design Automation into the world of the commercial semiconductor manufacturer. It was the first true NECESSITY driving force because of the need to cope with MSI complexity from a manufacturing point of view. Let me explain.

What is the limiting practical complexity for rubylith? If the largest practical ruby sheet is of order 10 feet on a side, if the smallest practical ruby grid step is of order .05 inch, if the required silicon grid step is of order .05 mil (1.25 micron to support 7-10 micron design rules) then these imply a ruby scale factor of 1000X and a maximum chip size of about 100 mil on a side. As MOS complexity pushed past these limits, pattern generation of a 10X reticle become a necessity.

But, how do we feed the pattern generator beast? How do we get the artwork into a computer? At RCA Labs, an English-like language called PLOTS [2] was developed. Although easy to learn and use, especially for regular structures such as memories, it soon outgrew its ability to conveniently support random logic designs as complexity pushed to the MSI level. Design Automation system builders had to come up with a low-cost system to capture hand drawn designs into the computer. We, therefore, developed a simple, inexpensive Interactive Graphics System (based on a PDP-8 with 4K memory, a Calcomp digitizer plotter -- hence, the name Digitizer Plotter System, DPS, and a hand held joystick) [3]. By today's standards, its "interactive" editing capabilities were as primitive as its hardware configuration. But mask artwork could now be quickly and accurately captured by a computer. One element of human error was eliminated.

The state-of-the-art Design Automation problem now was how to develop pattern-generation (PG) software which could translate a general (all angle) polygonal shape into a minimal covering set of rectangles (with the further constraint of minimal overlap at the edges of the polygon). This problem required three generations of production software at RCA to finally achieve an optimal solution. We did not choose to limit ourselves to orthogonal or 45 degree geometics (nonacute) because certain technologies, such as bipolar with lateral complementary transistors, require an all-angle polygon capability.

This latter constraint -- support of many technologies -- has had other effects on our Design Automation strategy at RCA. To emphasize this point, at RCA Design Automation deals with metal and silicon-gate MOS, CMOS/SOS, "Linear" Bipolar, I^2L, High Speed Bipolar, and various power transistor technologies.

Because of this diversity, we have endeavored to keep our software as technology independent as possible. We will return to this point later.

Concurrently with this work, the feasibility of a computerized design rule checking (DRC) facility was investigated (by the DREAMERS). Initial investigation found it to be too difficult in the general case; there were too many special cases. Additional data from the designers was essential. However, as complexity continued to increase, the number of digitized design rule violations was becoming a major problem. Furthermore, continuing complexity growth was rapidly moving us past that awesome level of integration now called MSI. NECESSITY being the mother of invention, DRC had to be developed -- even if it only solved a subset of the problem -- and it was [4].

This was the first truly new capability that storing the artwork in a computer brought us; after all, the pattern generator results in the same functional reticle as from rubylith. However, now the computer could do something truly new: automated design rule verification. We also discovered we could do more verification with the data at hand than we originally guessed. NECESSITY was a very strong driving force indeed.

At this point, it was dramatically clear (at least to the DREAMERS) that Design Automation possibilities were truly unlimited! New capabilities such as connectivity verification, device extraction from the artwork layout itself, automated artwork geometry modification, and a common data base linking all these to circuit and logic simulation were conceived rapidly and enthusiastically. Allow me to focus on this.

Common Data Base

It was quite clear to many of us at this point that computer-aided design and verification systems truly had an important role to play in the future of IC design. This was for two reasons. First, automated DRC proved that it was possible and achievable. Second, we began to internalize the fact that IC progress was relentless; that next year would bring even more complexity. However, we all felt that this year's Design Automation tools were barely adequate for today, let alone tomorrow. Hence, the Design Automation systems we were now envisioning had to cope with far more complex chips than existed today. A whole new (systems) approach had to be taken; we had to solve the growing complexity issue and we had to unify the worlds of circuit, logic and physical design. In addition, learning from the systems people as we gazed into the "complex" future, we had to cope with the problem of data integrity (sometimes called configuration control).

We decided that a centralized, "strongly-coupled" design data base, hierarchical in structure, was required to solve all these problems. We called this CADDB (Computer-Aided Design Data Base) [5]. This strategy "solved" many problems. Data integrity was insured because there was but one copy of the design (centralization); "strong coupling" between the physical and logical design representations guaranteed they remained in sync. The "common" data base also guaranteed that only one conversion program had to be written for each existing or new CAD tool (e.g., logic simulation, circuit simulation, system simulation, automated layout and so on: the tool's input data structure need only communicate with the data base, not with N other programs which implies N^2 conversion programs, in worst case). The hierarchical structure would solve the complexity issue. Finally, the overall design process was characterized and a new unified,

disciplined methodology was postulated. It was recognized that this would require a substantial effort and would lead to a quantum leap in capability. The CAD project team set their sights on success in a time frame of 3-5 years. In contrast, most of the CAD effort continued to focus on incremental capabilities, based on specific needs, with these new tools going into production in a time frame of 1-3 years.

The CADDB System was developed in the time frame expected. Hopes were high for its successful introduction into the production system. Unfortunately, the system was not enthusiastically accepted. In fact, to this day it still has not been accepted. Many of its concepts and some of its software components have been, but not the overall system.

What went wrong? In essence, the Design Automation tool builders had postulated a design methodology: logic simulation, followed by structured layout and DRC, followed by connectivity extraction, followed by comparison of extracted vs. simulated logic networks, followed by parasitic extraction for timing simulation, and so on. And all of this in a well defined environment. However, the designers were not as enthusiastic about all this structure as we were. After a number of attempts to use this methodology -- all heavily encouraged by management -- the general feeling was that the cost-benefit ratio was too high.

Perhaps we had imposed too rigid a structure on the lives of innovative designers who could use their ingenuity to solve design problems. Put another way, creating a total design structure assumes we not only understand all the normal steps in the design process but also what flexibility must be included if changes are to occur. Design systems (like all human institutions and ecological systems), must be flexible enough to permit creative change and be easily verified along the way; they must be adaptive or they can not survive.

Meanwhile, the CAD development effort --which defined needed, incremental capabilities -- continued to expand our overall set of design tools into the highly successful system it is today. (Some of this system will be described in the section on "Current Capabilities".) Perhaps the most credible way to highlight its success is to summarize some of the data from our Software Usage Monitoring System: in 1979, over 30 of these CAD programs were accessed over 100,000 times by over three dozen engineering organizations throughout the RCA corporation.

Based on these and many other experiences in Design Automation software development -- some quite successful, some not -- can we discover any general principles to help us maximize success in the future? This is not an easy challenge. The following are a few of my own observations.

Criteria For Success

First of all, I define a successful Design Automation tool to be one which is actively used by many different design teams on a variety of designs; not simply one which has been tested successfully on a single, benchmark test vehicle. I believe that a number of criteria are essential for a Design Automation tool to be successful.

"The design community must perceive that use of the Design Automation tool will cost-effectively result in a tangible benefit."

This criteria implies many things. Let me explain. In the early days the first and perhaps only tangible benefit perceived was a PG tape: one could make a reticle from it. It was real. Most of the other Design Automation capabilities were viewed as promises. When IC complexity advanced to the MSI/LSI level, the output from a DRC Program was suddenly perceived to be quite tangible; it was impossible for IC designers to find all design rule violations by eye.

Hence, the word "perceive" is an important one. It is not adequate that the Design Automation group define and solve a problem. The designers themselves must agree the problem exists and the solution is real and cost-effective; the latter must be true not only in dollars, but in terms of the designer's time and effort. This implies that the Design Automation solution must occur at or near the same point of the design cycle where the problem occurs and that either it reduce the overall effort at that point or it is clearly impossible to proceed without it (PG tape example). Finally, all of this implies that the solution be somewhat localized, i.e., the designer sees that the effort expended to utilize the Design Automation tool will solve a real problem right there and that endless side efforts beyond his control will not occur. This latter rule precludes systems which redefine or restructure an entire design process.

What seems to work is to institute an incremental Design Automation capability somewhere in the design process and then allow the system to adapt to it -- and adapt it to the system -- before introducing the next incremental capability. For this to occur, the Design Automation tools must be designed to be quite robust; the requirement of supporting many technologies seems to aid this.

This allows both the Design Automation System and the design process to adaptively evolve with time. Most importantly, it allows the creative designer to use the system in ways that were not even envisioned by the Design Automation tool builders -- who are generally not designers. Also, it encourages the designer's management to support and encouage its usage. Afterall, engineering managers are responsible for product development, not Design Automation development.

I believe a second criterion for success is that "The approach must be technology independent." This is clearly a generalization. It is based on my belief that if the Design Automation capability is technology dependent, technology evolution will obsolete the software. In those rare cases where success is obtained, the approach will not be readily transportable to other technologies. We have found that the utilization of Design Automation software by many design activities has dramatically increased the ROI (Return On Investment) of the original Design Automation developmental effort. It also helps keep the maintenance of all this software a tractable problem, as each piece of software has a large user base.

A third and perhaps obvious criterion for success is "There must be close communication between the Design Automation tool builders and the designers." Although this point is close to motherhood, it is too often neglected in practice.

This brings me to my last criterion of success. I believe that "90% of a Design Automation tool's 'paper' capabilities can be developed with approximately 50% of the total effort." This implies that it takes "forever"

to finally get the "last few" bugs out and to add the last few "bells and whistles" required for user acceptance. It also implies that the creative Design Automation developers who got it to the 90% point are probably bored with the whole project now and are anxious to move on to the next challenge, and to let the "second team" finish the project.

However, I believe that the "last 10%" is "make-or-break" for many Design Automation tools and that the second team may not have the perspective or experience to make the tweeks and compromises required for success. In fact, often some of the most challenging problems -- involving designer acceptance -- occur during the introductory phase of the development -- the "last 10%". Needless to say this is a true challenge to management, unless the Design Automation team has truly accepted that it is their challenge to make the tool a success; and this means getting it used by the design community.

I have symbolically summarized some of these thoughts in Figure 1. The dotted arrows imply that both the design community and the Design Automation team are encouraged to higher expectations based on a track record of accomplishments. The rest is obvious: success feeds itself. Unfortunately, the converse is also true.

Current Capabilities

At this point, I would like to snapshot some of the LSI Design Automation capabilities of RCA. I will then equate this system to the complex designs and point out where I feel our greatest challenges will be on the road to VLSI and beyond. I will then attempt to use some of the historical perspectives gained to project the future evolution of Design Automation to meet the challenges ahead.

Artwork

At the heart of RCA's artwork system, shown in Figure 2, is an artwork representation called DFL - Design File Language [3]. It is designed to be an efficient, compact, program-readable language. Its primitive components include general shaped (all-angle) polygons, orthogonal polygons, center lines with width, general purpose "comments" (to permit new extensions to work with old versions of software), and general "definition" (building block) function with an instance-specification capability (which includes scaling, rotation and mirroring). DFL allows hierarchical artwork specification (with nesting up to 12 levels deep). All rather forward looking for a language developed over ten years ago. This language has remained upward and downward compatible over the years, although new features have been added (using the "comment" extention capability), i.e., recently developed Design Automation programs can read a ten-year-old file while a ten-year-old program can read a recently created file. We believe long range compatibility of both software and data is an essential issue in a real production environment.

Although DFL is friendly to Design Automation software, it is not really friendly to humans. Therefore, the PLOTS language [2], already referred to, was created as a free-format, textual language for designers, which has a one-to-one function equivalence to DFL. One may view PLOTS as a source (e.g., assembly) language while viewing DFL as its machine language counterpart.

This language is useful for designing simple structures or ones that are highly repetitive (such as memories). The arrows between DFL and PLOTS in both directions in Figure (2) indicate that both a PLOTS compiler and decompiler were developed.

In order to design more complex structures, we designed the Digitizer-Plotter System which directly works with DFL during the late 1960's, as already discussed. By the mid 1970's, the DPS Systems were reaching their limitation due to growing IC complexity. We studied upgrading them and also evaluated turnkey systems. Based on our analysis, it became clear that it was more economical at this point to purchase than upgrade, and we purchased our first vendor-supplied interactive-graphics system. In Figure 2 the arrows in both directions between the interactive graphics systems and the DFL boxes indicate that we developed conversion software to translate from their internal artwork representations to DFL and visa versa. Hence, these systems became an integral part of our overall Design Automation System; i.e., a design generated on these systems could utilize any CAD capability such as PG and DRC while designs generated on DPS or by any other manner could be edited on these systems.

Another method to create DFL is via an automatic layout program. RCA has developed a family of these programs generically called APAR (Automatic Placement And Routing) [7] which refers to the automatic placement of predefined standard cells into regular rows of cells, which are then automatically routed, i.e., interconnected. The first generation of this software was called PRF (Placement Routing and Folding) while the second was PR2D (Placement and Routing in 2 Dimensions). The third generation called MP2D, (Multi-Port 2 Dimensional) now is being used. The MP2D Program has been highly successful in automatically laying out well over one hundred chips. (In fact over the last ten years RCA's APAR Programs have generated over 1000 custom LSI devices.)

Typically, a layout can be performed in weeks (from logic diagrams to PG tape). If the logic is simulated by our MIMIC logic simulation program, its connectivity description can be automatically translated to MP2D format. Automated layout is a highly useful capability for prototype chips (to quickly develop a custom IC for insertion into a system breadboard) and for low volume custom designs. It trades off increased chip size to gain shorter design time and lower front-end cost.

After creating the computerized artwork, the next step is to verify that it is correct. This can be done in a variety of ways. The most basic is to generate a checkplot of all the features for visual inspection. Software exists to communicate to a variety of penplotters, to graphic terminals or to high-speed electrostatic plotters.

More powerful than the visual inspection provided by a checkplot is the CRITIC (Computer Recognition of Illegal Technology in Integrated Circuits) design rule checking program [4] already mentioned. The designer may describe to CRITIC the geometrical rules of his technology in an English-like language. This is usually done once for a new technology and this description, contained in a user-named "CRITIC control file", is simply invoked when CRITIC is executed. CRITIC generates a listing which describes for each violation: what topological condition (e.g., disjoint, contain, inside, overlap, abutt) or tolerance (width, notch, clearance, or enclosure tolerance value) was violated; a PLOTS listing of the offending features and what definition they came from; and also a DFL "error" file containing these offending features and tolerance "tic marks" for generating an error check plot.

The ARTCON (ARTwork-to-CONnectivity Extraction Program) [6] may be used to extract logical connectivity

directly from the physical artwork for a subset of design styles. These include ones utilizing predefined standard cells or other more general artwork building blocks. This program is used for all RCA's GUA [11] (Gate Universal Array) designs (including metal gate, SOS and C^2L technologies) to automatically extract the connectivity for a variety of logic simulators. The circuit is again simulated to verify functionality and look for possible logic hazards. The truth table output is then automatically translated by the AFTER (Automatic Functional Test Encoding Routine) Program to a variety of automatic tester languages to provide a functional test program. This is highly useful, especially for Quick Turnaround Designs such as GUA and APAR.

Another important function of the Artwork System is automated modification of the Artwork. The SMART Program (Selective Modification of Artwork Regions and Topologies) based on the Baird Algorithms [8] provides many essential capabilities. Its primitive capabilities include the Artwork Boolean Operators (AND, OR, NOT), over-and-under-sizing of features and translation.

SMART is used to dimensionally "bias" (undersize) features to compensate for variations that occur during the mask making and semiconductor processing operations. It has also been used to partition large chips which are too large to be run in a single section on our MEBES Electron Beam Exposure System (whose limit is 16mm by 32mm, until a special reticle mode is developed) into abutting subchips.

It has been used to automatically notch down the channel region of polysilicon gates (yet not narrow the interconnect portion) to increase performance of critical parts. It can be used to automatically generate new mask levels from existing mask levels under a number of conditions and perform other automatic modification tasks.

At the working end, the MAP (Mask Artwork Program), translates the general polygonal shapes described in the DFL language into a format capable of driving a mask making machine such as a D. W. Mann or Electromask Pattern Generator (to make a 5-or-10X reticle), a Gerber photoplotter (to make an 80X foil) or a MEBES Electron Beam Exposure System (to make a fully step-and-repeated mask). The SANDRA Program (Step-AND-Repeat Array) generates the required chip location information including dropouts and test inserts, and also labeling information.

A capability that we have found to be highly useful over the years is the DEFORMAT Program. This decompiler-like program translates the data contained on a mask-making control tape back into DFL. This permits us to generate a shaded electrostratic checkplot of this data before the mask is made (for designer sign off). This verification procedure is invaluable as it tends to detect gross errors and otherwise obvious ones (e.g., improperly generated borders, improperly positioned or choosen library elements, missing alignment keys, an opaque "hole", and so on) before expensive masks are made.

The MASK System is another valuable facility at RCA. This software greatly simplifies the task of specifying MEBES or optical masters. It interactively requests all the information required to make a master: it gets chip information such as step-and-repeat distance and chip corners to automatically generate borders; it requests wafer size, chip dropout locations (or a standard code) and test insert patterns to create the chip array; and lots of other required information. As much of this information is common to a technology, it may be specified in a dataset called the Technology File which may be invoked for future circuit types of the same technology family.

After assimilating all this information, and performing syntax and semantics checks, MASK generates an "exec" file to automatically run MAP, which translates a DFL file to a PG tape, to run SANDRA, which generates the step-and-repeat array, to run DEFORMAT, which converts the PG tape back to DFL, and finally to run the checkplot software, which produces control tapes to generate the electrostatic checkplots.

We have found this type of system to be extremely valuable for automating standard procedures because it reduces effort, time and most importantly, the chance of procedural errors.

A similiar system to MASK is AUTOROM. It automatically generates a PG control tape and an automated tester tape from an input file consisting of the ONEs and ZEROs desired for custom mask programmable ROM. All of the permanent data required for a specified ROM type, (e.g., step-and-repeat distance) is stored in a permanent file, and so need not be respecified to generate a new custom variant of that type. In addition, AUTOROM does over a dozen obvious validity checks on the customer supplied bit-pattern-and-option file before proceeding to generate the PG and tester tapes. Over a half-dozen CAD programs are automatically invoked by AUTOROM.

Simulation

We presently have sophisticated circuit and logic simulation software tools. The purpose of these tools is to verify the performance of an IC before manufacture.

Our recently developed MIMIC logic simulator [12] has state-of-the-art capabilities. It may be used in interactive or batch mode. It is a four-level simulator (1, 0, X, Z). It permits a hierarchical building-block-specification capability. It uses the CADL language developed for the CADDB Common Data Base System already described (but not the Data Base itself). It has built-in models for primitive elements such as NANDs, NORs, ANDs, ORs, and a variety of flip-flops, both clocked and not. It properly models bilateral transmission gates including an arbitrary network of these. It supports inertial delay models for all gates: separate rise-and-fall delays may be automatically derived from tables of delay vs. either fanout or capacitive loading. A flexible hazard detection facility exists. A large variety of interactive facilities are supported; different time intervals may be simulated, breakpoints may be inserted, desired outputs may be respecified, NET states may be displayed or reset. The network may start in the unknown (X) or an initialized state. The designer may request detailed timing information (showing all the intermediate activity in response to an input change) or just stable-state results (the final state resulting from a new input state).

Our TESTGEN logic simulator [9], which can simulate up to 512 "faulty" networks in parallel, is used to determine the "stuck-at" fault coverage of a set of test vectors.

Another significant capability allows the designer to automatically translate from MIMIC input description format to APAR format (and the other way). This is not only fast and convenient, but eliminates translation errors. To further aid this process, a library of MIMIC subnetworks is being developed to functionally model each of the standard cells in APAR's library.

Furthermore, MIMIC output format is directly compatible with the AFTER Program which allows automatic translation of test vectors (network inputs and simulated outputs to automatic tester format. This greatly simplifies functional test generation.

Our R-CAP circuit simulation program [10] is similar to well-known programs such as SPICE in its capabilities. It employs a highly efficient DC algorithm, has standard transient response and small-signal-AC capabilities and has sensitivity-to-passive-components and noise-modeling capabilities.

Its short channel MOS transistor model is accurate to 3μ and is being extended further. It employs an Extended Ebers-Moll bipolar-transistor model and also, optionally a Gummel-Poon model which truly models base charge (unlike many other implementations). The user may choose an economical background batch mode or a highly interactive foreground mode. Plotting of all state variables, such as node-or-branch voltages, component current including device-pin current and component power is possible. Significantly, R-CAP has a correlated-component capability which allows the designer to specify any parameter (such as a resistance, capacitance or any model parameter such as threshold voltage or beta) in terms of a general algebraic expression. This supports verifying manufacturability, i.e., meeting electrical specifications in the face of processing variations.

We have developed the very sophisiticated TDAS Test Data Analysis System [13] which is also cost-effective and easy to use. The latter is true because the system provides basic primitive capabilities, such as histograms, wafer maps, trend diagrams and so on, which may be custom assembled for a specific application. This system is used on a daily basis to support six different processing lines at three different RCA manufacturing locations. Its reports are automatically distributed to process, type and design engineers.

Under active development is a general parameter correlation capability. This will provide graphical tools, such as scatter diagrams, and analytic tools, such as regression analysis. This will further aid the goal of Design for Manufacturability.

The Challenges of VLSI

A major question before us is: What are the compelling challenges to Design Automation as we move toward VLSI? Where must we expend most of our future efforts to enable VLSI to become a reality? I see vast challenges ahead, especially in layout, design verification and testing. I can envision solutions to these problems just as many of my co-workers in this field can.

However, as I think back to my early days in Design Automation, I have this strong feeling of deja vu. In the early 1970's, we clearly saw some of the problems of complexity and outlined elegant and achievable solutions. Only many of the more elegant and far looking solutions never took hold in our design communities. However, whenever we added a new (but incremental) capability which solved a real problem, and which cleanly fit into the existing structure, it was accepted. The addition of incremental capabilities has evolved our Design Automation System to where it is today. Therefore, as I try to project the evolution of Design Automation to serve the needs of VLSI I feel the need to constrain my optimism to growth modes that I have seen work first hand in the past.

Without a doubt, I believe the major problems to be solved on the way to VLSI are in layout, testing and design verification, probably in that order of importance. Moore [14] predicted that the design problem -- product definition, design and layout -- will be the rate-limiting process constraining VLSI utilizaiton: first because this cost -- which I call "front-end cost" -- is growing exponentially with time -- thereby tracking complexity -- such that the front-end cost per function is remaining roughly constant. Second, because as we move toward the system level of complexity on a chip, product uniqueness increases, thereby decreasing potential volume. This increases front-end cost of a function per unit IC. Meanwhile, technology advances are dramatically decreasing the per unit manufacturing cost of each function. Clearly, front-end cost will soon dominate the total cost of a VLSI IC.

Physical Design

Clearly, a major challenge to Design Automation is to develop design and layout techniques which dramatically improve productivity in the front-end areas. I do not believe this is an impossible problem. In fact, I am confident that we are evolving, slowly, to a solution of the productivity and cost-per-function issues.

The fundamental reason why front-end cost per function is not declining is because we are still mostly designing at or near the device or gate level.

I believe that the way to reduce front-end cost in IC design is to design and layout VLSI chips using higher-level functional building blocks which may be of SSI, MSI or even LSI complexity. For each product group, these blocks will include universal functions (such as clocked flip-flops, shift registers, ALU, RAMs, ROMs, encoders, decoders, expandable PLAs, comparators, A/Ds, D/As) and specialized proprietary application-oriented (yet often reusable) elements.

In addition, we must develop higher level, i.e., more productive, techniques to create these functional blocks, such as symbolic layout [15]. Furthermore, these blocks must be assembled within a structured, hierarchical design environment to allow design verification to remain tractable; a nontrivial problem!

Is this apple-pie technique achievable both technically and by the crucial criterion of designer acceptance?

I am particularly encouraged in the practicality of this approach by our large success in utilizing the APAR techniques (already described) in our semiconductor commercial product areas. (As indicated, it has been highly successful in the systems area for over 15 years.) At this time, the acceptance of this technique -- for a subset of commercial designs -- is growing rapidly, precisely because of the necessity to reduce front-end cost and design time for system-type components for which handpacked layout techniques are uneconomical.

The use of any automatic layout approach will not result in minimal area. For standard cell approaches, I believe the major cause of the area penalty is due to the use of fixed height standard cells and their algorithmic placement. Fixed height cells are rarely optimal in size -- simple cells tend to be too narrow (hence, "skinny") and complex ones too wide (hence, squat). Also, rigid pinout requirements tend to reduce cell density. In addition, algorithmic placement is never optimal because computers lack global perspective -- local optimums are usually found (global optimization is an NP-complete problem!). This leads to larger than required routing surfaces which impacts overall chip size and sometimes even performance (due to long interconnects with too many tunnels -- these add excess series resistance and capacitance to ground). Fortunately, CMOS technology is quite tolerant of these problems.

I believe the advantages of APAR can be extended to dense VLSI while minimizing its limitations by adopting a semiautomatic technique which cleverly elicits designer guidance. We are developing one such approach called FLOSS (Finished LayOut Starting from Sketch) [16]. This approach allows the use of arbitrary sized cells which can be SSI, MSI or LSI in complexity. The key insight of FLOSS is that it uses the global perspective of the human to get a rough initial layout (the sketch within which the designer optimizes placement and wireability) and then uses the computer to perform the tedious compaction process. The rough layout is essentially equivalent to the chip plan required before handcrafted layout is begun.

This process can be truly optimized by allowing extensive but rapid iteration to occur. The loose initial sketch is modified by insights gained from subsequent compaction. Another benefit of this approach is that future design modifications (which occur more often than we would like to admit) can be done to the loosely packed sketch, not the densely packed final layout. We plan to further optimize the iteration capability of this approach by developing a distributive computer system tying together an interactive graphics system with a fast processor.

Using this building block technique both the logic and layout designers have vastly improved productivity by designing at a much higher level of complexity. The challenge for this design technique is to develop a set of "macrocells" which have reasonably high complexity (which is the source of productivity gain) yet which are not too functionally unique (which would require designing a large library of macrocells; each, of course, must be accomplished at the device level). Other challenges for this approach are developing a design style which flexibly addresses aspect ratio, pinout-location and power-bussing issues.

A technique which is highly promising for the design of macrocells -- but not VLSI chips -- is symbolic layout such as a STICKS approach [15]. The challenge here is that the "compiling" of a STICKS diagram into silicon mask features is highly technology dependent. Optimal density of the macrocells may be difficult to achieve as technology evolves. Of course, parameterization of design rules may be a huge advantage in the face of rapid process evolution. One hope is that most of the parametrized designs will be readily scalable to tighter design rules -- this, of course, assumes that design rules will only change quantitatively not qualitatively, i.e., topologically. Or, if there are a few qualitative changes, the software compiler must be readily modifiable.

An advantage of the FLOSS approach is that it is not vulnerable to the above risks but can readily capitalize on whatever benefits symbolic layout can offer. Whenever it is judicious, a macrocell can be designed at the device level.

Design Verification

An other crucial issue is that of design verification — both logic and circuit verification and the crucial issue of physical design verification.

As described, we and other companies have developed excellent circuit and logic simulation tools. Our MIMIC logic simulator is hierarchical in nature. This permits developing a library of logical models for the predefined physical building blocks (macrocells) described in the previous section. This permits the logic designer to achieve the same type of productivity leverage as the layout designer, i.e., working with high-level primitives.

However, this approach will not be efficient as complexity increases. Although it will be simple for the designer to specify and analyze rather complex systems by designing them in a top-down, hierarchical structur, the simulator itself must perform a complete macro expansion of the system, as it simulates on the gate level.

Clearly, a general, transparent high level, macromodel capability must be developed to improve simulator efficiency; this is crucial for fast, economical interaction with the designer. This will be relatively straightforward for simple functional blocks such as ROMs, RAMs, PLAs, counters and shift registers. However, many subtle problems do exist, e.g., detailed timing accuracy, hazard detection and intelligent, i.e., minimal, propagation of the unknown (X) logic state. It will be even more challenging for more general functional blocks such as ALUs and decoders.

On the positive side, it is obvious that we can incrementally solve this problem, i.e., we can keep adding additional functional models (macromodels), one at a time, focusing on those that most heavily impact simulator cost.

In general, the ability to simulate VLSI-complexity circuits, i.e., subsystems, will require even higher levels of analysis, such as RTL (Register Transfer Level) and behavioral-level simulation. Although these types of simulators exist, I believe the challenge will be to structure a hierarchical simulation capability involving multilevels of simulation that communicate with one another, efficiently. This will permit system level simulation to verify lower-level-block design as top-down design proceeds. Impressive work [17] has occurred in this area but I believe, the efficiency challenge remains to be solved.

There are many challenges in the physical design verification area. I believe many of these can be tackled by developing structured design techniques. These techniques make extensive use of hierarchical layout implementation, i.e., nested building blocks with clearly defined design rules governing their proximity and interconnection. The latter is crucial if currently available design rule and connectivity verification techniques are to be extended to VLSI. I believe this will be the largest challenge facing physical design verification; it will require strong coordination between CAD tool developers and the design community.

Testability

Another huge challenge to the successful utilization of VLSI capability is the ability to test the devices we design. This is already a significant problem for many designs of LSI complexity. A number of techniques for "Design For Testability" (DFT) have been proposed including scan techniques [18], on-board test circuitry including utilization of signature analysis, behavioral-level test development [19] and designs using high visibility of all internal sequential logic to either probeable pins or the bus structures. In addition, I believe CAD testability aids such as controllability/observability measures [20] will be helpful to the designer incorporating DFT.

However, there is the obvious trade-off between testability and unit cost. The whole industry has traditionally emphasized silicon real estate above DFT for unit cost reasons. Our studies have shown that DFT using currently understood techniques can be quite expensive; more so than their proponents acknowledge. In addition, conventional testability measures, such as "stuck-at" fault-coverage monitors (used by all known parallel and concurrent fault simulation programs) do not address the issue of pattern-sensitive faults. Nor do the scan techniques test for them. These may become more significant as design rules shrink and inter-electrode couplings become more significant. This may also be crucial in the realiability area where migration of metal interconnections after stress testing may accentuate pattern-sensitivity (or lead to failure). As this will be a function of process evolution, built-in DFT techniques may be essential for even mature VLSI devices.

I believe the development of cost-effective design-for-testability approaches and their supporting Design Automation tools remain a large challenge.

Summary

The historical growth of Design Automation capabilities has been reviewed. The most successful ones appear to be those which focused on a specific problem and solved it. This incremental evolution has lead to the highly successful system we now have at RCA. Some of the challenges of VLSI have been highlighted. These include the necessity to improve approaches to logical and physical design -- in order to reduce front-end IC design costs -- and to develop cost-effective Design for Testability approaches. Promising approaches -- based on our historical perspective -- have been discussed and critiqued.

Acknowledgements

I wish to thank Henry Baird, Larry French and Fred Teger who thoughtfully reviewed an early version of this paper thereby contributing valuable insights and encouragement. I would also like to thank my staff, Rich Auerbach, Chris Davis, and Hans Schnitzler for their many and significant contributions to our Design Automation efforts over the years. I am grateful to Larry French who guided the direction and development of Design Automation for many years. I am totally indebted to my many co-workers at RCA who created the capabilities discussed in this paper.

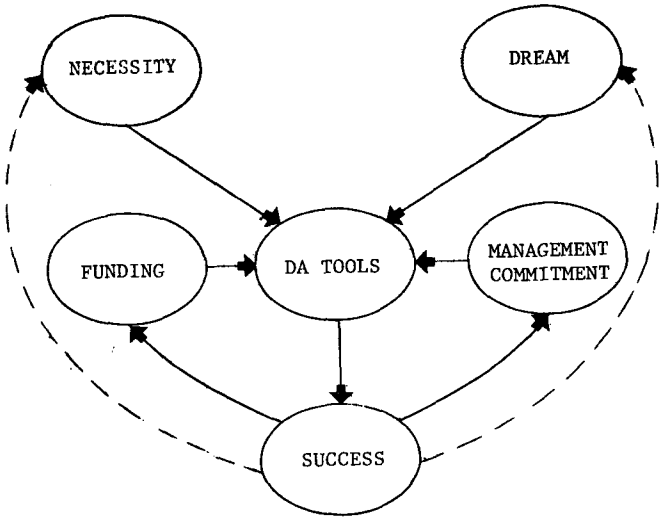

Figure 1. The Driving Forces of a Production Design Automation System.

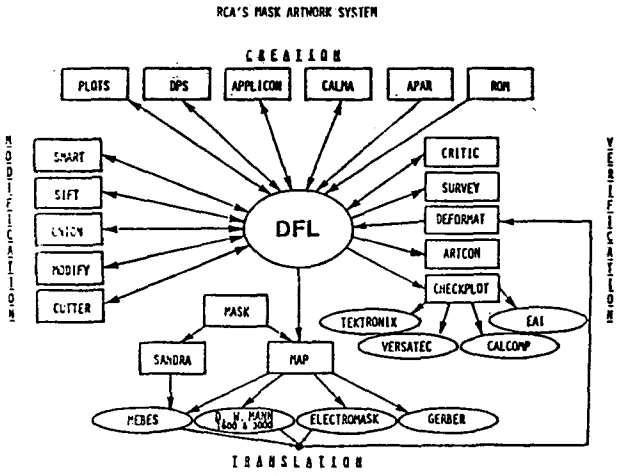

Figure 2

REFERENCES

[1] Mead, C. and Conway, L., Introduction to VLSI Systems. Addison-Wesley, 1980, pp. 137-143.

[2] Korenjak, B. J., "PLOTS: A User-oriented Language for CAD Artwork". RCA Engineer, Volume 20, Number 4, (December 1974), pp. 20.

[3] Ressler, D. G., "Simple Computer-aided Artwork System That Works". Proc. 11th Design Automation Workshop, June 1974, Denver, Colorado.

[4a] Rosenberg, L. and Benbassat, C., "CRITIC: An Integrated Circuit Design Rule Checking Program". Proc. 11th Design Automation Workshop, June, 1974, Denver, Colorado, pp. 14-18.

[4b] Auerbach, R. A. and Deutsch, S., "CRITIC: Improved Capabilities and A User-oriented Language". Internal Report, 1977.

[5] Korenjak, A. J. and Teger, A. H., "An Integrated CAD Data Base System". Proc. 12th Design Automation Workshop, June, 1975, Boston, Massachusetts.

[6] Baird, H. S. and Cho, Y. E., "An Artwork Design Verification System". Proc. 12th Design Automation Workshop, June, 1975, Boston, Massachusetts pp. 414-420.

[7] Feller, A. and Noto, R., "A Speed-oriented, Fully-Automatic Layout Program for Random Logic VLSI Devices". National Computer Conference Proceedings, 1978, pp. 303-311.

[8] Baird, H., "Fast Algorithms for LSI Artwork Analysis". Proc. 14th Design Automation Conference, June, 1977, New Orleans, Louisiana, pp. 303-311.

[9] Hellman, H. I., "TESTGEN: An Interactive Test Generation and Logic Simulation Program". RCA Engineer, Volume 21, Number 1,(June/July 1975).

[10a] Davis, C. B., Miller, J. C. and Rosenberg, L. M., "Digital Simulation as a Design Tool". RCA Engineer, Volume 17, Number 4, (December 1971), pp.34-39.

[10b] Davis, C. B. and Payne, M I., "R-CAP: An Integrated Circuit Simulator". RCA Engineer, Volume 21, Number 1, (June/July 1975).

[11] Bergman, R., Aguilera, M. and Skorup, G. E., "MOS Array Design: Universal Array, APAR or Custom". Ibid.

[12] Ashkinazy, A. and Hellman, H. I., "MIMIC User Guide". Internal Publication, August, 1979.

[13] Gianfagna, M., "A Unified Approach to Test Data Analysis". Proc. 15th Design Automation Conference, June, 1978, Las Vegas, Nevada.

[14] Moore, G., "VLSI: Some Fundamental Challenges" IEEE Spectrum, April, 1979, pp. 30-37.

[15a] Williams, J. D., "STICKS - A Graphical Compiler for High Level LSI Design". National Computer Conference, 1978, pp. 289-295.

[15b] Hsueh, M., "Symbolic Layout and Compaction of Integrated Circuits". Memorandum Number UCB/ERL M79/80, Electronics Research Lab, U. C. Berkeley, December, 1979.

[16] Cho, Y. E., Korenjak, A. and Stockton, D. E., "FLOSS: An Approach to Automated Layout for High-Volume Designs". Proc. 14th Design Automation Conference, June, 1977, New Orleans, Louisiana, pp. 138-141.

[17] Hill, D. and vanClemmpat, W., "SABLE: A Tool for Generating Structured, Multi-level Simulations". Proc. 16th Design Automation Conference, June, 1979 San Diego, California, pp. 272-279.

[18a] Eichelberger, E. B. and Williams, T. W., "A Logic Design Structure for LSI Testability". Proc. 14th Design Automation Conference, June, 1977, pp. 462-468.

[18b] Funatsu, S., Wakatsuki, N. and Yamada, A., "DESIGN". "Designing Digital Circuits With Easily Testable Consideration". 1978 Semiconductor Test Conference, pp. 98-102.

[18c] Mulder, C., Niessen, C. and Wijnhoven, R. M. G., "Layout and Test Design of Synchronous LSI Circuits". 1979 International Solid-State Circuits, pp. 248-249.

[19] Johnson, W. A., "Behavioral-Level Test Development". Proc. 16th Design Automation Conference, June, 1979, San Diego, California, pp. 171-179.

[20] Goldstein, L. H. "Controllability/Observability Analysis of Digital Circuits". IEEE Trans. Circ. and Sys., Volume CAS-25, Number 9, (September 1979), pp. 685-693.