P. W. Case
M. Correia
W. Gianopulos
W. R. Heller
H. Ofek
T. C. Raymond
R. L. Simek
C. B. Stieglitz

Design Automation in IBM

Within the context of the changing design requirements of digital systems spanning the semiconductor era, this paper describes the significant steps in the development of Design Automation technology in IBM. We cover the design tools which support the design of the electronic portion of such systems. The paper emphasizes the systems approaches taken and the topics of design verification, test generation, and physical design. Descriptions of the technical contributions and interactions which have led to the unique characteristics of IBM's Design Automation systems are included.

Introduction

Design Automation (DA) has become a term to describe the use of computers by engineers and other specialists to assist in the design, development, and production of complex systems. In IBM, the object designs are digital electronic computer systems, and the DA programs are organized into coherent, interdependent sets, hence the term DA systems. The technological evolution of electronic computer systems over the past thirty years has been remarkable in its scope and rate of change. This growth has been made possible by the use of computers themselves to assist in the design of new generations of computers. DA has been both a leading application of computer technology and a significant part of it.

This paper traces the development of DA technology in IBM from its inception, highlighting some important technical steps and their genesis, without attempting to be comprehensive in its coverage. The objective is to put in context a description of the development of this new technology and its part in the growth of computer systems themselves. This paper highlights primarily IBM achievements but acknowledges and provides some reference to the considerable work done outside IBM in this area.

Early IBM electronic products, such as the 604 Electronic Calculator, used design and documentation practices substantially inherited from earlier electromechanical technology products. Despite the self-imposed discipline which limited logic designs to the use of a small standard set of predefined circuits, the inherent flexibility of electronic logic led to designs which were more complex (and, of course, functionally richer) than those using earlier technologies. This led to more variants of the initial design, both for engineering changes (ECs) and for features, and to logistic problems with the design documentation. As designs became more complex with the 701, 702, 704, 705 series of computers, accurate product documentation was difficult to maintain through hand-annotation of printed, draftsman-originated diagrams and manual recording of ECs installed. This problem led P. Case and R. Simek to develop, in 1958, the first system application of computers in IBM development, called Engineering Change Control, which maintained a data base of the engineering level of each product and automated the initiation of field installation of ECs. This effort demonstrated the utility of the previous generation of computers in assisting with current designs, and led to

Table 1 Technology evolution and DA support systems.

Technology era	*SMS*	*SLT/MST*	*LSI*
Logic circuits per unit			
Chip	NA	1-31	100-700
Chip carrier (module)	NA	1-31	100-45 000
Card	4-50	6-480	
Board	400-2000	600-5000	180 000-230 000
Logic circuits per typical large CPU	9000 7094 II	90 000 3033	460 000 3081
Host computer systems used	IBM 704/705	IBM 7094 IBM System/360	IBM System/370
Data base	Tape sequential	Tape sequential	Disk-indexed sequential

investigations of ways to improve the handling of the detailed design data. In parallel, an IBM Research effort, led by S. Dunwell and J. Logue, initiated work on mechanized logic diagrams. The movement of the Stretch project into development, together with the introduction of IBM's first comprehensive transistorized technology (Standard Modular System, SMS), precipitated the formation of a DA development group, led by P. Case, with the objective of designing and implementing DA systems.

Design automation system evolution

Forming the conceptual foundation of the initial DA efforts was a systems approach that envisioned a data base which grew apace with the product under development until the completed detail was ready for release to manufacturing. This data accumulation process paralleled the two major steps of the existing design process: logic design followed by physical circuit and wiring design. It provided a single accurate definition of the state of the design at any point in time. Subsequent evolution of DA svstems can be viewed as the interaction of three changing technological forces: 1) circuit and packaging technology, 2) product design methodology, and 3) design automation technology itself.

- *Circuit and packaging technology*

The dramatic rate of change in the circuits and packages used as building blocks by digital circuit designers has been the predominant influence on the evolution of DA systems. Table 1 shows three major stages of technology and identifies the DA systems associated with each one. Design complexity has also grown with the increase in the number of package levels which the product designer personalized. In early SMS technology, the circuit cards consisted primarily of a predesigned set, and the specification of the panel and cable wiring was the essential physical design variable. Later the cards became more nearly unique, and in SLT/MST (Solid Logic Technology/ Monolithic Systems Technology), design of the circuit cards became an additional physical package design variable. LSI brought a third design level into the picture, with its requirement for designing chips.

Alternative design approaches sometimes created dichotomies for the DA system designer. Nowhere was this more apparent than in the support requirements for custom *versus* masterslice chips. The latter's regular structure of circuit cells and wiring channels presents physical design, layout, and wiring problems quite different from those of the more free-form custom chip with its arbitrarily sized macro islands, which are often formed from unlike logic components. The logic design portions of the DA system, however, have more often proved flexible enough to support custom chip design needs.

Memory-space and execution-time constraints caused the program structures of first-generation DA systems to be strongly dependent on the hardware packaging schemes they were designed to support. Package parameters were often hard-coded into the application programs, and fixed-format data bases represented the state of the art. The pace of circuit and hardware evolution put severe pressure on DA developers because of the time required to write new programs and to create and verify the circuit and package rules data base. Most importantly, new classes of problems were often introduced which had a critical dependency on untried DA algorithms, as with the introduction of panels using etched circuits instead of discrete wire connections. Schedule pressures often resulted in DA programmers having to predict the effectiveness of heuristics prior to writing their programs. Such predictions were often regarded as pessimistic by engineers who judged new package capabilities based on manual wiring trials. Test-generation programmers were

presented with cards of unexpected complexity for which tests had to be generated. As DA systems evolved, the DA technologists increased efforts to generalize and parameterize the systems and applications, and succeeded in making major areas of the system less sensitive to changes in hardware technology. For this capability modern DA systems sometimes pay a price in program size and complexity and execution time.

- *Design methodology*

DA systems had revolutionary effects on IBM's design methodology. Not all changes took place painlessly. Early users experienced a decrease in flexibility, as program constraints restricted even seemingly trivial factors such as the position of logic blocks on a page. Computers proved unforgiving and autocratic in their relentless insistence on detail, as in matching alphanumeric line names, where previously humans had been able to readily match entries despite minor differences. Even the traditional sequence of the design process was altered, as verification and correction of details could no longer be postponed to a late stage of development. Gradually, DA systems have evolved to embody, to codify, and to enforce the requirements attendant to the design and, especially, the release of IBM's products. DA systems provide explicit definitions of documents, nomenclature, conventions, standards, and auditing limits, both through the programs and through the library data sets of circuits and physical units, across a multilaboratory environment. Manufacturing and field organizations use these systems as a vehicle for ensuring commonality, compliance with agreed standards, and information accuracy.

Design documentation requirements provide a particularly illuminating example of the influence of design methodology on DA systems. While it may seem surprising now, prior to the existence of DA systems the design documents *were the design*. In order to make the digitized data base *the design*, a scheme had to be devised that ensured an accurate relationship between the documents and the data base, while still permitting the engineer to design in his customary way—by creating new, or by altering existing, logic diagrams. Thus the basic structure of the data base was necessarily reflective of the primary documentation format, and the content had to be capable of being printed or displayed in document form. This was the origin of the unique form of logic block diagram used in IBM, called an ALD, for Automated Logic Diagram (see [1, 2]). An alternative would have been to make severe changes in engineering, manufacturing, and servicing practices by substituting a listing form of documentation. Although listing formats have since gained favor for some uses, the value of block diagrams has proven so powerful an aid to human understanding that the systems effort to utilize this graphic form is regarded as well justified.

While the initial DA approach to assisting engineers in reaching accuracy goals was to compare the detailed design with sets of predefined rules, attention was later given to helping with the analysis of a design's functionality. This process is called design verification, and DA applications with this goal are still evolving. Without such aids, engineers are forced to build physical hardware prototypes to validate their designs. Early LSI technology users often constructed a prototype in a pre-LSI technology. DA systems actually anticipated the need for logic simulation tools to aid in design verification and alleviate the need for hardware modeling. Gradually, driven by the difficulty of prototyping dense LSI technologies, software modeling has become a widely accepted practice.

Overall, current DA systems support a methodology for handling, organizing, and processing design data that makes possible the complex designs indicated in the LSI column of Table 1. Other than the continual evolution of applications to meet the needs of new technologies, perhaps the most significant systems improvement affecting design methodology now taking place is the evolution to the use of interactive alphanumeric and graphic terminals as the predominant mode of access to the systems.

- *Design automation technology*

DA systems and applications have been at the forefront of programming technology since the earliest DA work. Programmers continually sought ways to compress data so that arrays could be contained in available memory space, devised swapping schemes prior to the availability of virtual addressing, and invented operating systems which contained generalized input/output routines, access methods, and automatic program linkage mechanisms prior to their availability as part of general-purpose operating systems. In IBM, these efforts formed a part of a larger view of design automation as a comprehensive system. Included in the system structure were functions to support the access, control, and maintenance of a common data base, auxiliary libraries of parameters (rules), and interfaces for the exchange of data between the data base and the application programs which constituted the specific design assistance tools.

Design automation has grown into an established scientific discipline, with many facets and subtopics. Breuer of the University of Southern California [3] and Van Cleemput of Stanford University [4] have prepared valuable bibliographic lists which indicate the breadth and scope of

the technical work being done. Much of the emphasis and interest in DA today is due to the gating influence of DA on the ability to utilize LSI technology. In some instances in LSI, and certainly in VLSI, the progress in DA algorithms determines the extent to which designers are able to exploit the inherent semiconductor density and performance potential. The balance of this paper will unfold, both for systems and applications, many of the DA technology areas in which IBM has participated. The emphasis is on the work of IBM's organized, centralized DA development groups. There have been many innovative DA programs, contributed by individuals at several IBM laboratories, which could not be included. Examples include wiring programs by J. Cooper, and logic and simulation programs by D. Rozenberg and A. McBride.

IBM design automation systems

Each of the technologies shown in Table 1 required the development of a new-generation DA system. This section discusses the highlights of each one.

• *The SMS DA system*

IBM's first DA system [1] was named for and designed to be used with SMS technology. An initial characteristic of the technology which influenced the DA system design was the use of a set of predefined transistorized circuit cards configured as "unit logic" portions which plugged into a wire-wrapped panel. Unit logic design presumed that a relatively small set of multiple-usage cards could be designed from which the product designer would select, place, and interconnect the subset required. The DA system was therefore configured with an AND/OR-level logic-block-oriented data base (then called a "logic master file") together with a standard rules data base describing the card/circuit library. Subsets of the design, such as a panel of logic, could be selected from the data base, audited against the rules, checked for implicit inconsistencies, and processed by other applications. Two applications of note were the logic page drawing and panel wiring programs.

The logic block diagram programs were programmed to compute the routing of the interconnecting lines. The first successful heuristics were programmed by S. Sobel and R. Carpenter. Solutions to this topological problem proved more difficult than expected, and required several years of evolution before programs by C. Warburton and R. Christopher prepared logic diagrams that began to be as esthetically pleasing as had the previous manually drafted drawings. Nevertheless, the improvement in accuracy and the reduction in data entry workload provided a strong incentive, and in fact made the use of automated block diagrams feasible.

The panel wiring programs provided the first step in automatic design, since they extracted directly from the logic diagram file the implied set of pins to be interconnected to form each logical net, and computed a set of actual wires to accomplish the needed physical connections. These programs also maintained a secondary data base containing earlier engineering levels of wiring and calculated the minimal set of add-deletes to upgrade an existing design. The resultant improvement in accuracy of wiring data became a hallmark of the DA process.

An important system principle, originated in SMS DA, was the general policy of accepting manually specified input to override algorithmically supplied data. Thus pins, specific wires, etc. could be predefined and left intact if desired. This compromise furnished a workable bridge between algorithmic efficiency and practical engineering requirements.

• *The SLT/MST DA system*

IBM's second-generation DA system [2] included functions previously implemented and retained a structure similar to that of the first DA system. Its primary purpose was to support a new generation of technology, characterized in SLT by etched circuit boards interconnecting pluggable unit logic cards, and later, in MST, by functional cards containing many levels of logic. Important systems concepts introduced by W. Murley were 1) the detection of certain classes of errors at the time the data base file was updated rather than during subsequent checking runs, 2) the establishment of a conceptual basis for consistently labeling logical blocks and their associated wiring nets, eliminating arbitrary net numbers and the resultant human and program tasks of correlating them with the logic, and 3) the organization of the logic file in a nested delta arrangement which supported multiple versions of a basic design automatically.

SLT unit logic card design was supported by a design subsystem called Small Card Design Automation, SCDA, developed by J. Barnes and his coworkers. MST subsequently superseded SLT, bringing with it the requirement to support complex functional cards unique to each design. Fortunately, primarily due to the foresight of H. Graff, the DA system was adaptable to this major shift in engineering practice.

Other innovations of this generation of DA systems included work by C. Haspel and others on partitioning programs which assigned logic to cards, placement programs which positioned cards to aid board wiring, and logic simulation programs to support design verification [2]. The SLT/MST data base concepts grew with extensions to support the automatic generation of diagnostic

tests [5, 6]. Subsystems which supported the design of microprograms introduced concepts of behavioral-level description of digital logic [7] which eventually led to multilevel design verification support.

- *The EIS DA system*

An effort to implement a next-generation DA system to satisfy LSI and its associated requirements was launched in 1968. The system was called EIS (Engineering Information System) and in 1973 was renamed the Engineering Design System. Several new concepts were introduced by the principal architects, who included C. Haspel, F. Worthmann, J. Boyle, B. Dzubak, T. Spence, and R. Taylor.

The primary data base was organized for random access processing [8]. It allowed all types of design data (logical, physical, control, etc.) to be stored for all levels of packaging. This structure avoided many of the problems inherent with serial tape-oriented systems. Additionally, the data structure was parameterized so that it could be easily adapted to varying packaging nomenclatures and requirements. This has allowed the heart of the DA system to accommodate designs which include many varieties of chips, modules, cards, planars, and boards. Methods of segmenting the data base were eventually developed to accomplish hierarchical processing. This allowed the chips to be designed independently of, but in parallel with, the module or card. When the chip design was complete, the necessary data were available for completing the processing of the next level of package. Since the necessary data were a small subset of the entire chip design data, very dense second- and third-level packages could be processed. Another major thrust has been to provide interactive processing with both graphic and alphanumeric terminals. The emphasis has been on physical design applications for both custom [9] and regular [10] designs. Concurrently, batch mode test pattern generation and design verification capabilities were provided.

Programs were also implemented to audit the design process. This allowed the DA system to ensure that the right level of technology rules was used, that the design, checking, and test-generation programs all ran successfully, that no severe design errors were detected, and, most importantly, that if the design data were changed in the middle of the process, the necessary programs had been rerun to ensure total design data integrity prior to manufacturing. This feature has promoted a design discipline which has made masterslice (gate array) chip design in particular a very reliable process.

The following sections single out three DA application areas covering some of the most challenging technical problems. These are design verification, test generation, and physical design, the last emphasizing chip wiring. We trace their technological evolution and show their status and importance in the LSI area.

Design verification

Design verification is a term used to denote a host of tasks which must be performed by a logic design engineer in order to ascertain the correctness of his design. We examine the major design verification functions of simulation, Boolean verification, and timing analysis in some detail, while mentioning other functions only briefly. There are several reasons for the engineer to verify his design. The initial specification of the design is basically behavioral, is often expressed informally in prose, and contains inherent ambiguities. Designers need a way to ensure that their design is a correct implementation of the specification. Also, the complexity of the design usually forces designers to think in terms of multiple conceptual levels. In particular, the detailed implementation of some portions of the design may affect the general design of other portions. Thus, implementation constraints and the interactions between portions of the design often prevent a pure "top-down" design process. The designer needs a tool capable of verifying the correctness of a mixed software model of the design, with each portion described on an appropriate conceptual level. Another factor is that design iterations are commonplace, as conflicting partial solutions are resolved. Again, verification tools are needed to check self-consistency of the total design. Lastly, human errors and oversights are unavoidable because of the size and complexity of the designs being considered. Thus, tools are needed to check for such problems whenever manual intervention occurs during the design process.

Initially, IBM DA developers concentrated only on computerized structural checking tools to aid in verifying the correctness of the design. Logical and physical checks as well as design rules checking were introduced to detect inconsistencies in the design data, to identify sourceless and sinkless nets, to ascertain that parameters were within prescribed limits (*e.g.*, checks for maximum fan-in and maximum fan-out), to check against a library for the family of circuits in use, etc. During the mid-1950s, such checking algorithms were developed in various IBM laboratories. One set of early functional checks to be programmed dealt with computation of signal delays through a logic network. These checks, along with delay simulation, are needed in addition to functional verification. Functional verification, in which programs perform Boolean evaluations or propagate signals through the logical model, started in the 1960s. The class of Boolean evaluators includes heuristic and definitive tools. The

former include the variety of logic numerical simulators, and the latter consist of Boolean analyzers. The remainder of the discussion on design verification focuses on the key aspects of functional verification.

• *Functional verification*

Simulation has been the workhorse of functional design verification over a period of almost twenty years. The SLT/MST DA System was the first one to offer logic simulation to the design engineer in IBM. The SLT/MST DA simulator [2], developed by H. Graff and D. Hoffman and their associates, operated on a model consisting of a network of interconnected logic blocks, selected from the system logic file. It used parameters stored in a circuit library to compute the Boolean output of each given block as a function of its input signal values. The output of the simulator included a sequence chart and a timing chart, both of which served as debugging tools in the hands of the engineer. The simulator was a nominal delay simulator, and it utilized the concept of significant event simulation. The significant event technique is important since it limits the amount of examination and calculations done by the simulator. By considering only those design elements which could possibly change logical state as a result of the latest change of state, running time is kept to a low level. One problem encountered was that of settling due to oscillations, which was solved by introducing artificial delays to create a wait long enough for small spikes to disappear. The simulator was a two-value simulator, and thus did not have the accuracy of modeling needed to perform transition analysis. While the software model of the design to be simulated was obtained via a direct reading of the Logic Master File, test pattern sequences to simulate the logic had to be manually specified by the engineer in the form of input bit patterns. A significant innovation was introduced by F. Hackl and E. Carlstrom, who linked the simulator to the system used to develop microcode for the System/360 models. This allowed the microcode to be used as stimulus for hardware design verification, a concept which is used to this day.

A significant development was three-valued simulation. The invention of a three-valued algebra by Eichelberger [11], which solved a test-generation problem, also helped logic simulators detect and handle logic circuit hazard and race conditions. The third value, *X*, was added to the stable Boolean values "0" and "1" to allow analysis of networks in which some signals are not in stable states. One such simulator is described in [12].

The next major thrust was to combine the valuable features of the variety of logic simulators developed locally within each laboratory into a functionally richer and faster logic simulator. The Variable Mesh Simulator (VMS) was designed by R. Forbes, M. Kelly, and J. Teets [13], first as a standalone program, and subsequently as part of the EIS. VMS was first used in 1971 and, due to its generality, it is still in use today. One significant feature was its four-value modeling capability which allowed more accurate results than those obtained by three-value simulation. In addition to "0," "1," and *X*, the *U* value was added, denoting an uninitialized value. At start of simulation, all nets are assigned the value *U*. In the course of simulation, these *U*s are replaced by other values. If any remain, however, at the conclusion of the simulation, it means that specific nets were not affected by the stimulus applied, indicating either a design error or incomplete coverage by that stimulus. VMS also included varying degrees of delay and timing accuracies, namely, zero delay, unit delay, nominal delay, and extreme delay. In VMS, any combination of value range and delay type appropriate to a phase of product development may be chosen. VMS is named for its ability to mix these cases in a single run, the variable-mesh concept. In the mixed simulation mode, the design can be represented by a number of interconnected subsets, each of which may be described in a different form or on a different level. Thus, it is possible to handle each part of the design in a way appropriate to its state. In 1973, a new mode, the behavioral simulation mode, was added to VMS. Behavioral modeling enables parts of the design, which are of interest only insofar as their input/output behavior is concerned, to be modeled at a high level, thus speeding up the simulation, since only the portion of logic being debugged has to be treated as a detailed logic network. VMS also included a high-level programming language designed to assist the engineer in specifying stimuli. P. Agnew's formulation of engineering problems and their solution through simulation ensured that VMS indeed solved the real problems in a manner acceptable to design engineers.

In 1973, the need for a functional cycle simulation capability resulted in the formulation of a register transfer language (RTL) [14] and its translation to an executable VMS model. This capability was subsequently replaced, in 1978, by a new table-driven functional simulator which featured efficient model generation with a fast incremental update capability. A variation of functional simulation compares an RTL description of a design specification to its logic network implementation. This is done by generating VMS models for the two descriptions and simulating them with VMS, using the same stimulus and comparing the results. Another significant technique, which was developed by H. Kriese and D. Baglan, allows the designer to simulate his detailed design at a greater speed by translating the primitive-level logic networks into executable behavioral models. Simulating these behavior-

al models by VMS is more efficient than simulating the original networks because it allows significantly lower storage requirements and reduced event scheduling.

• *Delay and timing analysis*

The analysis of timing and the computation of signal delays through a logic network constitute an important part of design verification. An early functional check to be programmed involves the counting of logic circuits along a path in order to get an approximation of the signal delays through the logic. This check considers only the time element and not the Boolean function element. Subsequently, more advanced delay and timing analysis programs were developed, incorporating varying degrees of accuracy and a variety of computational concepts. These tools complement delay simulation, which in itself did not provide a flexible or powerful enough tool for engineers seeking maximum performance in their products. A requirement for more accurate timing and delay analysis led to the development in the late 1960s of a delay path analysis program. Using parameters defining both circuit delays and wire path delays, this program calculates delays for all paths for a given logic network. It was little used, since its exhaustive analysis consumed considerable execution time and produced voluminous output. With the advent of LSI, the delay calculation program mentioned above was replaced by a variety of technology-dependent programs. A general timing analysis algorithm developed by R. Hitchcock and his coworkers is now in use. It is based on block- rather than path-oriented processing, and as a result it requires much less execution time than the path-oriented algorithms.

• *Boolean verification*

Successful incorporation of LSI required development of a novel design verification methodology that included a hardware flowchart specification of the design, an automated translation of the flowchart to pseudologic design, a functional cycle simulation of the flowchart, a detailed description of the implemented logic, a timing analysis of the logic, and Boolean comparison between the logic and the pseudologic. The overall approach is important because it eliminates direct low-level hardware simulation. Also, timing verification is separated from logic functional verification.

A key element is the Boolean comparison which determines, by analysis, whether or not two combinational logic networks are functionally equivalent. In case they are not equivalent, input states are determined that illustrate the unlike behavior of the two networks. One of the networks is assumed to be "good" or correct, while the other represents a hardware design that is to be verified by comparison with the "good" network. The "good" network description is obtained by the translation of the flowchart description of a machine or part of a machine into a logic network description consisting of interconnected primitive blocks such as AND, OR, NAND, etc. This is the pseudologic which is used as the standard and against which the manually implemented design is compared. The flowchart description of the machine is a graphic form of the RTL hardware description language described in [14]. Its translation into equivalent pseudologic is accomplished by using synthesis techniques without optimization to a particular technology. Programs to translate an RTL-type description into logic could be found in the Logic Automation programs, and the flowchart translator which is used today is based on an algorithm developed by J. Roth [15] and H. Halliwell.

Test-generation programs are particularly suitable for Boolean comparison, since one can view the networks being compared as a pair, a good machine and a "faulty" version of the good machine. If no test exists to distinguish the "faulty" network from the original, the networks are equivalent. A difference between test generation and Boolean comparison is that in test generation one usually expects to find an input state (pattern) that distinguishes between the original network and the "faulty" network. In Boolean comparison, however, it is usually expected that there will be no input state to distinguish between the networks being compared.

One specific approach to the solution of the Boolean analysis problem, using test-generation techniques, is the VERIFY algorithm, developed by Roth [16] as a derivative of his well-known D-ALG test-generation algorithm. VERIFY, which may be viewed as a special case of the consistency subroutine of the D-algorithm, was implemented by H. Halliwell, who introduced important efficiency enhancements to the original algorithm.

Another Boolean analysis approach was introduced by A. Brown and his coworkers. It included the Differential Boolean Analyzer (DBA), which was developed by R. Bahnsen as an application of the expansion theorem of Boolean algebra. A significant factor in the success of DBA was the segmentation of the logic into a set of overlapping segments. These segments were processed independently by the Boolean analyzer, thus decreasing its running time. The use of structure processing ahead of Boolean analysis contained the data explosion problem which is common to Boolean analysis algorithms. Additional research on Boolean comparison algorithms involved the use of simulation as an aid to the Boolean verification process. Numeric random simula-

tion was used to improve the efficiency of Boolean analysis [17], while symbolic simulation was used to generalize the process [18] to include a higher-level model comparison.

The Boolean verification methodology is gaining in importance because of the growing acceptance of structured design methodologies and the use of LSSD design (discussed in the next section). G. Smith has made significant contributions to Boolean verification by introducing innovations in both usability and efficiency areas, thus helping make this tool useful for the engineer. More recently, early user experience was combined with improved algorithms and methodology to add a Boolean analysis function to the Engineering Design System.

Test generation

The derivation of tests for logical functionality of digital equipment is a process known as test generation. This process is normally performed through an analysis of the logical structure of the product.

What is the technical problem making test generation difficult? Why not just verify operation as it was intended? Most concisely, the problem is one of combinatorics due to restricted access. A logic network with n inputs is "intended" to operate correctly for at least 2^n different input combinations. If the network contains storage elements it is "intended" to operate correctly for each of the allowable states of the storage elements. For m independent storage elements, the number of combinations of "intended" operation becomes at least 2^{n+m}. The number of combinations, and hence the testing time, becomes impractical for even modest values of n and m. The practical problem of test generation is thus one of finding small subsets of allowable combinations which when applied to actual equipment can identify the presence of faults. A second problem involves the identification of what is wrong when the test fails.

In this section we present snapshots taken at three points in time illustrating important problems and their evolutionary solutions: first, the early experimental work in 1958–1960, second, the first large-scale card test system of about 1965, and third, the recent past and the impact of the introduction of LSI.

- *The SEA project*

Early experiments

The Systems Error Analysis (SEA) Project [19], started at IBM Endicott in 1958 by R. Forbes, was motivated by the large effort required to write system diagnostic programs. The system concept was to automate the test-generation process, using the design automation file as source data and an analyzer program to generate tests, followed by a fault simulator to generate symptom characteristics for diagnosis. An important part of this work was the introduction of test objectives (faults, or commonly, stuck-faults). This concept reduced test size and served as a basis for diagnosis. Three key technical problems recognized at that time were addressed simultaneously: First, sequential networks were more difficult than combinational networks and required special solutions; second, simulation was potentially time-consuming, and third, there were no techniques for applying tests to actual products or for guiding the repair process.

Combinational networks

The first formal approach to test generation was developed by Roth [15], who had been working on Boolean minimization using algebraic topological methods. One of the operators developed, called Π^* (Pi-star), generated the on and off arrays for a Boolean network. If one viewed a fault as transforming the network into a second network, one could then operate on each with Π^* to generate a pair of on and a pair of off arrays. If we call these [ON], [OFF], [ON′], and [OFF′], then ([ON] ∩ [OFF′]) ∪ ([OFF]) ∩ [ON′]) would be the totality of tests for the given fault. If one did this for each fault, the problem of generating a test set for all faults was reduced to a covering problem. Though theoretically sound, this approach proved to be impractical. Even modest networks (100 gates) resulted in excessive computer running time.

A second approach developed by C. Stieglitz of the SEA Project proved more successful. Basically, it was a tracing approach, which started at the point of a fault and traced first forward to an output and then backward toward the inputs. In tracing forward, a sensitized path was created which would have one of two values depending on whether or not the fault was present. In tracing backward, inputs were generated which would sustain the conditions necessary for the sensitized path. This tracing process could produce many false starts; that is, paths could be chosen which proved to be unsensitizable or requirements on net values could contradict each other. Processing for these false starts could potentially become overly time-consuming. The key solution to this problem was an operation called IMPLY. Each time a choice in tracing was made, IMPLY was used to project its consequences. In this way, many mutually exclusive choices were eliminated, streamlining the test-generation process. Essentially the same algorithm minus the IMPLY operator was formalized by Roth [16] using his D calculus in what has since become a classic paper in the field.

Sequential networks

Test generation for sequential networks proved much more difficult. The first difficulty was the lack of an

algorithm for analyzing sequential networks. Secondly, sequential networks contained races and hazards which could cause unexpected operation. One approach to this problem was pursued by S. Seshu of Syracuse University, acting as a consultant to the SEA Project. Seshu's test generator produced input sequences which changed one input at a time to minimize potential race conditions. If the network had n inputs, n potential successors to each test existed. Each of these was evaluated using a fault simulator and the "best" one in terms of information gain was chosen [20]. This approach to test generation had sporadic success. Its biggest value was in providing a model and a vocabulary for describing the testing process.

Fault simulation

It was recognized fairly early that fault simulation running time could potentially increase as the cube of the network size. As a result, the efficiency of the program was considered very important. The simulators used in the SEA Project were based on a concept from L. Tung called parallel compiled simulation. The instructions to be executed were compiled from the network model to avoid any conditional branching instructions. In addition, each bit in a computer word was used to represent a different fault so that as many faults as there were bits in a word could be simulated at once [21].

Applications

The early programs resulted in some successful applications. These all involved the programs for combinational networks. The algorithm for sequential networks proved too costly and erratic. The first commercial application was on the IBM 1418 Optical Character Recognition machine, where they were used to generate tests for the recognition logic, a very large combinational circuit. The programs generated the tests on the basis of the logic description in the design automation master file. Perhaps a more important application in terms of future directions was in the testing of SMS twin cards, initiated by M. Correia. Until that point, cards were tested exhaustively by applying all possible combinations of inputs. Test symptoms were generated by cutting components and running the tests. With the increasing density (greater than 10 to 20 circuits), this was no longer feasible. The SEA programs were demonstrated to be an effective alternative greatly reducing costs and elapsed time. Since the internal card logic did not then exist on any DA system, the logic was entered manually into the system.

Another application area was in system test. K. Maling, M. Evans, and others, as reported by Preiss in [22], adopted the applied concept and combined it with a technique of deductive fault simulation in a system which supported logic designs with tens of thousands of logic gates. These programs were incorporated in the Fault Locating Test (FLT) System [6] and successfully used on the IBM 9020 and System/360 products to automate the diagnostic process. A principal contribution of the FLT system was the idea that the computer system should incorporate a reasonable ($\approx 5\%$) increment of special hardware to give the sequential machine logic the appearance of combinational logic during testing.

Test generation for sequential networks

In the early 1960s, the introduction of SLT with its anticipated high volumes and many part numbers mandated the automatic generation of card tests. Further, these cards contained sequential networks, requiring extensions to the existing systems. One of the problems in working with sequential networks is to have a good model on which to operate. The most popular model had been one generated by "cutting" the feedback lines. Feedback lines were chosen somewhat arbitrarily, but enough were cut to eliminate all loops. The network was then converted to a Huffman model by inserting delay elements in the cut lines. In this model races were detected when more than one feedback changed state at the same time. The races were resolved by considering all possible sequences for the changes. The problem with this model is the way it treats delays. In a real network, delays are distributed throughout the circuits and their interconnections. In the Huffman model, as well as in the other switching theory models of the time, the delays were concentrated at a small number of points.

A system called SALT (Sequential circuit Automated Logic Test) was developed for the purpose of generating tests for SLT cards. Because of the experience with the Huffman model, a modification was chosen to capitalize on the independent behavior of the storage elements within the network. Storage elements were given a topological definition as a set of loops having a common circuit. Each subset of circuits comprising a storage element was then analyzed using a Huffman model of its own. The results of the analysis were saved in state tables and other specialized tables used in the test-generation process. The new system worked reasonably well but a serious problem remained. In spite of the network model changes, hazard conditions still produced occasional anomalous behavior. Since one of the purposes of an automated system is integrity of the data, these anomalies were very troublesome.

This problem led Eichelberger to propose a three-valued algebra [11] (see the preceding section on design verification). The SALT simulator had been implemented as a three-valued simulator for an entirely different reason, that being to allow partial specification of test

pattern inputs. As a result, the technique was quickly implemented, with two very good results. Simulation time was greatly reduced and simulation results became accurate. The time reduction came because in using the Huffman model, oscillations caused repeated simulations. Only after 2^n (n = number of feedback cuts) passes without reaching stability could it be determined there was an oscillation. Using the three-value approach, there were at most $2n$ passes, n passes each changing one of the feedback signals to an X, and another n passes each changing a feedback signal to a known value. In no case would a feedback signal change more than twice. Accuracy was increased because the model would now produce an X if there was any delay distribution that could produce ambiguous results. As a result the system was fail-safe or pessimistic. Real hardware never behaved differently than predicted and the test coverage was always at least as claimed. The only disadvantage was that in the case of controlled races (because the hardware delays were bounded) the results would be overly pessimistic.

Applications

The SALT system marked the integration of card test generation with DA in IBM. SLT cards had their own Design Automation System and corresponding data base, which served as the source of data for test generation. In addition to the logic tests described, SALT also consisted of programs to generate nonfunctional tests involving pin-to-pin impedance. The tests were kept in an on-line manufacturing data system. The final test system was a highly automated computer-controlled system that managed a continuous stream of mixed part numbers through the various test stations. A printout containing repair actions was attached to each card not passing.

• *Test generation for LSI*

Technology evolution has been characterized by constantly increasing density. By 1969, it was having profound effects on card test generation due to a proliferation of very complex cards. The original SALT concepts, as embodied in a Technology Independent Test Engineering System (TITES) [23], were not achieving test objectives. The TITES test-generation programs were modified in at least two important ways. Because the increased number of storage elements caused the tracing algorithms to become less effective, the programs were modified to become more heuristic and less predictable. Often the tests had to be augmented by manually generated tests. Second, the three-valued simulation approach became too pessimistic. Many of the circuits used, such as latches, contained managed races which the simulator flagged as being unable to operate. Gradually, simulation algorithms to selectively reduce hazard detection were introduced, compromising test data. This required further manual intervention and increased test-generation time.

IBM's decision to enter LSI with an open-part-number set meant that system designers would now be designing chips as well as cards. Because of the situation in card test generation, it was decided to shift test-generation responsibility to the system designer. In addition to chips, the designer also had to generate tests for multichip modules and cards, each adding a significant level of complexity. An automated test-generation methodology became necessary. Starting with a system that was just barely working, LSI added a level of complexity and time constraints that could not be met.

LSSD (level sensitive scan design)

A solution was proposed by E. Eichelberger. Since test-generation programs were much more effective on combinational than on sequential networks, the problem was to find a way to physically transform sequential networks into combinational ones without excessive overhead. This was the same problem which FLT had dealt with in System/360, but now the difficulty was to find a hardware solution appropriate to the testing of LSI chips, with circuits accessible only through the chip pads. Eichelberger suggested that circuits be designed with an independent way to load and unload the storage elements. Each storage element could then look like an input or an output to the test generator, which could then generate tests for combinational networks interconnecting the storage elements. Considering chip pins to be a valuable asset, Eichelberger proposed that this be done by connecting the storage elements into a shift register with each end connected to a pin [24]. A further requirement was that there be no race conditions during testing. This could be accomplished if the design used polarity-hold latches for storage elements, a two-or-more phase clock, and a structure such that no direct path from the output of a latch fed the input of another latch gated with the same phase. Collectively, this approach came to be known as LSSD. The important difference between test generation for LSSD designs and previous test-generation schemes was that the designer was given a set of design rules or constraints which made automatic test generation easier.

In the early 1970s IBM started to put together a Design Automation test-generation system for LSSD. The requirements of the system were that it be applicable to chips, multichip modules, and cards. The network sizes to be covered ranged from several hundred to many thousands of circuits. The tests were to be generated in the product design laboratory and sent via digital interface to manufacturing locations for direct application on the testers. This called for high confidence in the integrity of the data.

Because of the reliance on design rules, it was important to provide the designer with tools to check that the rules were being followed. To do this, Godoy [25] devised a novel way to use the design verification simulation system. This was accomplished by replacing the subroutines that normally emulate the logic functions of circuits with special subroutines that propagate the check signals having special functional meaning. Large networks were handled by programs that separated the network into smaller subnetworks bounded by latches, primary inputs, and primary outputs. The test generators operated on the subnetworks to generate tests. Other programs reassembled the tests to be consistent with the original large network [26]. The LSSD system is the current preferred test methodology within IBM. It has been applied successfully to generate tests for chips and cards on the IBM System 38 and on System/370 Models 3081, 4331, and 4341.

Automated physical design using regular package structures

In the earlier parts of this paper, brief historical references were made to the evolution of physical design techniques as packaging technology has become more sophisticated. Although both the density and the details of structure of all levels of these packages have advanced rapidly, a simplifying hierarchy of regular images has continued to be the key feature of physical design. On each level, objects with input and output connectors are placed into "sockets" on a fixed grid, and wiring is done in X and Y directions in channels allotted for the purpose on distinct planes with "via holes" for plane-to-plane communication. Chips obeying these constraints, in general, *share* a planar space for power bussing, devices, and wires, whereas chip carriers and higher-level packages often *divide* their vertical structure of planes among pad-to-grid redistribution wires, power, and signal wiring.

- *Pre-LSI physical design*

In IBM's early transistorized designs (the 1400 and 7000 series), engineers handled directly the problems of module (chip-carrier) and card placement as well as connector pin assignment on all package levels. The back panels which held SMS cards were wired using discrete connectors made by numerically controlled machinery [27]. On each of these panels, which had 40 sockets of 16 pins each, approximately 800 wires were connected by automated wire-wrap techniques. A principal contribution of the original wiring program was to produce digitized instructions for the wire-wrap machinery. The extra space afforded by the third dimension permitted the wiring algorithm to avoid parallelism and cross-talk by spreading out the wiring.

The advent of SLT, with its etched card and board wiring, focused attention on the problem of wireability and on other physical design aids which affected wireability, such as placement of cards in sockets on the boards and pin assignment on cards and boards. In addition to layout of etched wiring, programs specified discrete wires, both for overflow and for engineering changes to the boards. SLT, like all subsequent regular packages, permitted only a fixed number of wiring tracks in each wiring channel, all of course running parallel on a given wiring plane. These boards had two wiring planes, one of which was used for principally horizontal and the other for principally vertical routing, with predrilled via holes on a regular grid to permit passage from one plane to another. SLT layout first had to face the joint problems of estimating the required *track* and via *capacities* of packages, in relation to the statistical distribution of wiring *demand*. Analytic formulation of this complex problem was lacking, and the approach taken by U. Kodres and H. Lippman was to use a heuristic algorithm completing longest nets first, one by one, followed by a "maze-running" algorithm to embed as many nets as possible. This last technique was first suggested by E. Moore and was given detailed definition by C. Lee of Bell Laboratories [28]. Depending on the available board capacity and on the wire-length distribution determined by the placement and pin assignments, this separation into phases permitted completing 50–90% of the wires in the first phase, followed by completion of 80–100% of nets in the maze-running. A feature in the design schedule and economics was that a hopefully small remaining number of "overflow" wires could be embedded manually into the image, or actually attached as discrete "yellow" wires to one surface of the package.

MST made effective use of the programs developed for SLT by applying them to internal wiring of the pluggable cards themselves. Automatic placement was refined by T. Lavery and included the effects of electrical constraints. An optimum combination of the wiring algorithms and human intervention was worked out for expeditious completion of the card designs. During the period, Hitchcock proposed and developed his cellular approach to wiring images, which simplified and speeded up the use of the maze-runner by permitting the application of hierarchical subdivision of the wiring image in a simplified representation [29]. A summary of subsequent development and use of wiring algorithms has been given by Hightower [30], then at Bell Labs, who originated a fast-running variation of maze-running known as the "line-probe" technique. Up to this time, the high proportion of manufacturing cost relative to development cost and duration led to the design of packages capable of holding more subpackages than could be efficiently

wired. We next discuss the influence of LSI in changing this approach.

• *Physical design in the LSI era*

Prior to the use of LSI, the design of chips themselves had not been a part of the product design process. The use of chips and chip design presented three significant differences from earlier work with pluggable packages: 1) Chips are not "reworkable"; *i.e.*, they cannot be repaired or changed economically once manufactured. Each engineering change may be viewed as a new layout, although design of the next package level usually makes it necessary to retain existing assignments of chip I/O (input/output) connectors. 2) Chips cannot be internally "probed" in production testing operations, which must be done through the I/O connectors. 3) Placement and wiring of circuits on chips, whether automatically programmed or done partly manually, is a more difficult design task, since discrete wires may not be added. The additional package complexity of LSI made analytic assistance necessary to the solution of the wiring problem.

It was established IBM practice that a regular package hierarchy made both design and manufacturing simpler. Work in many companies with such hierarchies led to an important simplifying observation which was found to relate logic package connector count to the number of "modules" or subpackages carried and wired together by the package itself. In IBM this relation is known as Rent's rule, after E. Rent; it had been shown both theoretically by Donath [31] and experimentally by R. Russo and B. Landman to be a consequence of typical logic partitioning by designers aiming at an efficient compromise between performance (reducing delay through the logic) and cost (minimizing module count, connector usage, and package size for a given circuit count). This compromise historically has occurred at a subfunctional package level (*i.e.*, at a level in the package hierarchy where the total circuit count is only a fraction of architecturally recognizable functions such as CPU, storage control, etc.).

The rule gives a fractional power dependence of the package I/O connector count as a function of packaged circuit count. The Rent relation is $T = AC^p$, $1/2 < p < 3/4$, where A is the average used subpackage connector count, C = number of subpackages contained in the package, and T = used I/O connector count for the package itself. This relation is of enormous importance, since it permits prior definition of a range of acceptable package sizes and connector counts at all subfunctional package levels. It was shown by Donath [32] that the rule implies a corresponding but slower growth of average wire length with package circuit count. This in turn implies that the wiring load *per circuit* grows heavier with increasing levels of integration. To this information, and the subpackage count itself, one can add the typical number of wires per subpackage. Together, they reflect the degree of success of partitioning, placement, and pin assignment techniques in subdividing and locating the logic in the package hierarchy. W. Vilkelis, W. Thompson, and L. Poch had made early estimates of the wiring capacity packages must contain. This work was followed by the probabilistic model of Heller, Mikhail, and Donath [33], who were able to give "wireability" formulas to determine package wiring capacities. An agreed means of analytically evaluating proposed package designs thereby became available, and these formulas are now widely used in IBM.

The use of sharp estimates of chip size and wiring capacity is crucial to a successful LSI product. Our understanding of physical design algorithms has gradually developed. With the wireability estimation tool, we can match the *presently asymptotic approach to optimality* of the entire suite of these algorithms to the wiring capacities of the packages constituting a large logic function. One can then evaluate the tradeoff in final manual editing and net completion *versus* the asymptotically more difficult improvement of automatic wiring programs.

• *Assignment and placement of circuits in a standardized image*

Assignment of package pins to particular signal inputs and outputs was recognized early as amenable to computer handling. At first this was done after placement of subpackages on the package. The general problem of linear assignment had already been solved in another context. A matrix is formed assigning "costs," *e.g.*, a weighted combination of wire lengths and local congestion associated with the assignment of a given wire to a given pin (*i.e.*, a given matrix row). Preassignment of some pins can be accommodated, and the least cost can be found or approximated in polynomial time as a function of the number of pins.

Placement of subpackages on a carrier is a more difficult problem, and in some of the work discussed herein, pin assignment and placement are often combined into one program. A simple model of the problem was early studied in operations research as the "quadratic assignment problem." Since the problem is n-p complete, one must use a heuristic, which can be aimed at more realistic constraints. The simplest effective approach uses some weighting of wire lengths and local congestion. Most successful work (a summary reference is [34]) has recognized that the problem has to be attacked *hierarchically* and *globally*. That is, one must use the "divide and

conquer'' approach to break the overall problem into successively manageable subunits, starting from an *overall* view. Khokhani and Patel [35] were the first to incorporate specifically, in a global placement algorithm, some measure of the global wiring implied by the placement of circuits on a plane. This, weighted together with a minimum length criterion, led to successful and relatively rapid placement of circuits on a masterslice (gate array) chip and is now widely used in IBM chip designs. The growth of the computer time associated with this algorithm is empirically found to be given by $n^{1+\delta}$, where $\delta < 0.2$. The algorithm has two parts: a constructive phase based on relative connectivity, followed by an iterative phase which exchanges blocks.

• *Global and local wiring algorithms*

Early wiring programs were handicapped by a lack of understanding of the degree of difficulty of the problem they were attempting to solve, or knowing whether a solution was possible. Inadequate approaches to optimality of wiring capacity, of connector assignment, and of placement all dump their burdens on the wiring program. The more closely all these factors work together, therefore, the more effective can be the wiring routines. The crucial insight into successful wiring algorithms for large problems is the built-in capability to look at the global picture of the unit to be wired. Work for LSI automatic wiring of chips at IBM by Chen, Nan, Feuer, Khokhani, and Schmidt [36], integrated with the evaluation and algorithmic capabilities earlier mentioned [35], has made use of hierarchical global wiring. In this way, final channel selection is deferred as long as possible and each connection is given equal treatment. The method consists of subdividing the wiring space into blocks in a hierarchical fashion and assigning connections to the block boundaries they cross by successive perturbation of a global trial solution, which converges to an overall lowering of demand *versus* capacity in each block. Then the solution is mapped to a finer grid and, eventually, to wire segments (*e.g.*, Steiner tree representations) in the actual wiring channels. Often the connections are allocated to channels in the vertical and horizontal directions by ''line-packing'' techniques, which date back to the work of A. Hashimoto and J. Stevens at the University of Illinois and of B. Kernighan, D. Schweikert, and G. Persky of Bell Laboratories (see Reference [30]).

An essential novelty in the global wiring, introduced by Chen [36], is first to define the wiring for each net independently on an empty global block image. Then the nets are superposed, and perturbations of appropriate nets are carried out by moving wires to reduce any excess of demand over capacity at individual block boundaries. This way, all nets are treated equally, any convenient scheme for routing individual nets can be followed, and nets can be varied in location at the possible expense of increasing lengths. At the end of these procedures, an efficient maze runner can be used [30], if desired, to clean up some or all of the remaining nets.

Habra [37] and also Skinner [10] have developed valuable, time-saving alternative interactive editing schemes to put in and check the final few unwired nets. Thousands of chips, like those described in [35] and [36] and, more recently, chips containing hundreds of circuits on similar images, have been completed using the programs. Better than 90% of the chips achieve 96% completion of nets. The CPU times for placement and wiring programs lie in the range of one to two hours, including the set-up time, using IBM System/370 Model 168-3 class systems. A parallel development to this work was later carried through by H. Koch and P. Backer for use on higher-level packages.

Checking programs and custom chip design

The foregoing discussion of masterslice chip design has assumed the availability and use of a library of predesigned ''books'' which are placed into the slots of regular arrays. These ''books'' are themselves irregular multilevel circuits. Their design, along with the initial layout of the masterslice itself, has also stimulated the development of design assistance tools. Of particular interest are the problems posed by the manipulation and analysis of irregularly shaped entities in the various package levels. A graphic design assistance tool which has been developed for this purpose is the Interactive Graphic System (IGS) [9]. The USC (Universal Shapes Checker), a collection of programs performing analytic and shapes checking [38], provides a means of detecting whether such designs conform to physical and electrical constraints for the circuit family. While the use of these tools has been extended to very complex structures with hundreds of thousands of shapes, the design of entire custom chips which have an irregular layout has evolved separately.

The emergence of FET technology, with its higher densities, resulted in the design of layout-assistance tools which included IBM 1620 and 1130 computers attached to various graphic workstations [39–45] and programs for postprocessing, like those developed and used by W. Donath and others. MST-era efforts led to two memory chip design tools, Memory Graphics Program and Memory Graphics Programming System, as well as graphic tools for laying out logic chips.

Another application of FET technology was to random logic chip design. A columnar arrangement proposed by A. Weinberger in 1967 [46], called a master-image, at-

tempted to bridge the gap between the pure custom and the regular grid approaches, and led to the development of a design system supporting FET logic design called Chip Design System, CDS. CDS permitted definition of a library of logic "books" which could be checked electrically and placed on a regular cell array. As interest grew in FET logic, automatic programs were developed for placement by B. Dunham and J. North and for pin assignment and wiring by N. Nan and M. Feuer. FET design efforts were merged by W. Rosenbluth and his colleagues into the FADS system (FET Adaptive Design System) which supported small system users.

Several other important efforts which strongly influenced the eventual design technology and tools had been proceeding in parallel. Early FET work by J. Logue was directed toward avoiding the limitations in circuit density which were a consequence of the master-image approach. Parallel work was done in Los Gatos by A. McBride, L. Warren and others. For products having very high volumes of production (like microprocessors), arbitrarily limiting oneself to lower levels of integration and more costly production in order to reduce design time and attain low-risk manufacturing proved to be noncompetitive. Eventually, the master-image approach, which was suitable for products where development costs dominated, was supplemented with custom design techniques. Among the first products to use custom design techniques was the UC-0, a 1973 design of a two-chip microprocessor which achieved an integration level four times that of its master-image equivalent in the same technology. This effort demonstrated the leverage of custom circuit design, achieving 2000 equivalent circuits on a convection-cooled chip.

The most significant design-related by-product of the UC-0 work was to demonstrate the formidable difficulties associated with a design approach which used individually tuned logic cells at high levels of integration. Each UC-0 cell had to be capacitively "padded" to preserve the logic level presented to the logic blocks in its fan-out. Moreover, this padding was a function of placement and wiring as well as fan-out, causing an iterative design process throughout the physical design of the chip. The demands for shorter development time were inconsistent with such a time-consuming design process. Subsequent special-function logic designs have built and expanded on custom techniques, which exploit FET unique circuitry and reduce design logistics through functional partitioning of the logic into islands of "macros." These macros permitted almost unlimited design innovations internal to the macro, while providing a more rigid interface at the macro-to-macro boundaries. Thus, the iterative process described earlier is improved while still achieving highly customized circuitry. Another approach, which eliminated the iteration and retained most of the masterslice advantages, was developed by P. Satre and his coworkers. Called RMS (Rochester Master Slice), this method further sacrificed density for improved speed of design, although improvements have been made along these lines by workers at the IBM Rochester and Essonnes laboratories.

A second important advance in IBM's custom design concepts came from early work by H. Fleisher and his collaborators, who suggested partitioning schemes for trading off decoder, AND-array, and OR-array configurations in a logic chip to achieve yield or performance advantages [47]. This work led to macros composed of programmed logic arrays, now a widespread application area. In the years following 1973, increasing FET densities and the success of custom macro concepts led to their application throughout the semiconductor industry. Outside IBM, double-polysilicon-layer FET chips made possible more versatile designs with greater wiring complexity. Within IBM, the first available 64K-bit FET memory chip was designed using metal gate technology.

In the future, in order for increasing density to mean increasing complexity of function, custom design will be required. Improved techniques will be needed if design times for VLSI logic are to be held within economical levels. Emphasis on the logical and physical design of macros and their interconnections can be expected.

Summary

In a period of slightly more than twenty years, design automation has become a recognized discipline encompassing the realms of data processing applications, methodology of design of digital systems, and the exploitation of semiconductor technology. We have outlined the growth of DA technology in IBM, emphasizing the systems basis of the work as well as the key DA applications of test generation, design verification, and physical design, to illustrate the increasing interdependence of semiconductor product designs and DA capability.

While we have related many technical contributions to the evolution of DA technology, the authors recognize and regret their inability to comprehensively credit many important individual efforts.

The authors hope that readers of this paper will gain insight into the potential future use and value of DA technology, and, more generally, into the problems and opportunities associated with the emergence of wholly new technical disciplines.

Acknowledgments

Space does not permit comprehensive acknowledgment of all the individuals who have contributed to the success of IBM's Design Automation projects. The following individuals deserve special mention for their personal efforts, and serve as examples of others whose contributions have not been referenced herein. Among the development and liaison engineers who substantively influenced the evolution of DA are S. G. Tucker, W. T. Burke, and A. E. Fitch. Within the DA development groups F. J. Worthmann, R. E. Forbes, R. Bertolino, J. Sanborn, C. Von Krogh, D. Cooper, and I. S. Saba made important technical contributions.

References

1. P. W. Case, H. H. Graff, and M. Kloomok, "The Recording Checking and Printing of Logic Diagrams," *Proceedings of the Eastern Joint Computer Conference*, Philadelphia, PA, 1958, pp. 108–118.
2. P. W. Case, H. H. Graff, L. E. Griffith, A. R. Leclercq, W. B. Murley, and T. M. Spence, "Solid Logic Design Automation," *IBM J. Res. Develop.* **8**, 127–140 (1964).
3. M. A. Breuer, "Recent Developments in Automated Design and Analysis of Digital Systems," *Proc. IEEE* **60**, 12–27 (1972).
4. W. M. Van Cleemput, *Computer-Aided Design of Digital Systems: A Bibliography*, Computer Science Press, Woodland Hills, CA, Vols. 1-4, 1976–1979.
5. R. E. Forbes, C. B. Stieglitz, and D. Muller, "Automated Fault Diagnosis of Switching Failures," unpublished report presented at the 1961 AIEE Conference on Diagnosis of Failures in Switching Circuits, University of Michigan, May 1961. (C. B. Stieglitz and D. Muller are located at the IBM System Products Division laboratory, Endicott, NY. R. E. Forbes is located at the IBM System Products Division laboratory, Charlotte, NC.)
6. R. J. Preiss, "The Use of Fault Location Tests in Prototype Bring-Up," *Proceedings of the IFIP Congress*, New York, 1965, pp. 511–517.
7. B. R. S. Buckingham, W. C. Carter, W. R. Crawford, and G. A. Nowell, "The Controls Automation System," *Proceedings of the Sixth Annual Symposium on Switching*, University of Michigan, 1965, pp. 279–288.
8. T. Beretvas, "A General Purpose Multi-Indexed Data Management System with History Capabilities," *Technical Report TR00.2078*, IBM Data Systems Division laboratory, Poughkeepsie, NY, 1970.
9. P. Carmody, A. Barone, J. Morrell, A. Weiner, and J. Hennesy, "An Interactive Graphics System for Custom Design," *Proceedings of the 17th Design Automation Conference*, Minneapolis, MN, 1980, pp. 430–489.
10. F. Skinner, "Interactive Wiring System—IWS 370," *Proceedings of the 17th Design Automation Conference*, Minneapolis, MN, June 1980, pp. 296–308.
11. E. B. Eichelberger, "Hazard Detection in Combinational and Sequential Switching Circuits," *IBM J. Res. Develop.* **9**, 90–99 (1965).
12. J. S. Jephson, R. P. McQuarrie, and R. E. Vogelsberg, "A Three-Value Computer Design Verification System," *IBM Syst. J.* **8**, 178–188 (1969).
13. P. N. Agnew and M. Kelly, "The VMS Algorithm," *Technical Report TR01.1338*, IBM System Products Division laboratory, Endicott, NY, 1970.
14. G. J. Parasch and R. L. Price, "Development and Application of a Designer Oriented Cyclic Simulator," *Proceedings of the 13th Annual Design Automation Conference*, Palo Alto, CA, 1976.
15. J. P. Roth, *Computer Logic, Testing, and Verification*, Computer Science Press, Potomac, MD.
16. J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," *IBM J. Res. Develop.* **10**, 278–291 (1966).
17. W. E. Donath and H. Ofek, "Automatic Identification of Equivalence Points for Boolean Logic Verification," *IBM Tech. Disclosure Bull.* **18**, No. 8, 2700–2703 (1976).
18. H. Ofek *et al.*, "Structured Design Verification of Sequential Machines," *Research Report RC 7037*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1978.
19. R. E. Forbes and C. B. Stieglitz, "Status Report on Systems Error Analysis for Diagnostics," *Technical Report TR01.11.083.578*, IBM System Products Division laboratory, Endicott, NY, 1959.
20. S. Seshu and D. N. Freeman, "The Diagnosis of Asynchronous Sequential Switching Systems," *IRE Trans. Electron. Computers* **EC-11**, 459–465 (1962).
21. R. N. Ascher, D. N. Freeman, J. S. Jephson, and L. H. Tung, "Some Problems in Automation of Diagnostic Procedures," presented at the Conference on Diagnosis of Failures in Switching Circuits, Michigan State University, May 15–16, 1961.
22. M. A. Breuer, *Design Automation of Digital Systems*, Vol. 1, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1972, Ch. 7, pp. 352–358.
23. P. S. Bottorff and R. A. Rasmussen, "A View of a User-Oriented Production Test Generation System," *Proceedings of the 7th Annual Design Automation Workshop*, San Francisco, CA, June 1970, pp. 90–94.
24. E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," *Proceedings of the 14th Design Automation Conference*, New Orleans, LA, 1977, pp. 462–468.
25. H. C. Godoy, G. B. Franklin, and P. S. Bottorff, "Automatic Checking of Logic Design Structures for Compliance with Testability Ground Rules," *Proceedings of the 14th Design Automation Conference*, New Orleans, LA, 1977, pp. 469–478.
26. P. S. Bottorff, R. E. France, N. H. Gorges, and E. J. Orosz, "Test Generation for Large Logic Networks," *Proceedings of the 14th Design Automation Conference*, New Orleans, LA, 1977, pp. 479–485.
27. G. W. Altman, L. A. DeCampo, and C. R. Warburton, "Automation of Computer Panel Wiring," *Trans. AIEE* **79**, Part 1, 118–125 (1960).
28. C. Y. Lee, "An Algorithm for Path Connection and its Applications," *IRE Trans. Electron. Computers* **EC-10**, 346–365 (1961).
29. R. B. Hitchcock, "Cellular Wiring and the Cellular Modelling Technique," *Proceedings of the 6th Annual Design Automation Workshop*, Miami Beach, FL, 1969, pp. 25–42.
30. D. W. Hightower, "The Interconnection Problem, A Tutorial," *Proceedings of the 10th Annual Design Automation Workshop*, Portland, OR, 1973, pp. 1–21.
31. W. E. Donath, "Equivalence of Memory to 'Random Logic,'" *IBM J. Res. Develop.* **18**, 401–407 (1974).
32. W. E. Donath, "Placement and Average Interconnection Lengths of Computer Logic," *IEEE Trans. Circuits Syst.* **CAS-26**, 272–277 (1979).
33. W. R. Heller, W. F. Mikhail, and W. E. Donath, "Prediction of Wiring Space Requirements for LSI," *Proceedings of the 14th Annual Design Automation Conference*, New Orleans, LA, 1977, pp. 32–42.
34. M. Hanan and J. Kurtzberg, "Placement Techniques," in *Design Automation of Digital Systems: Theory and Techniques*, Vol. 1, M. Breuer, Ed., Prentice-Hall, Inc., Englewood Cliffs, NJ, 1972, Ch. 5, pp. 213–282.
35. K. Khokhani and A. M. Patel, "The Chip Layout Problem: A Placement Procedure for LSI," *Proceedings of the 14th Annual Design Automation Conference*, New Orleans, LA, 1977, pp. 291–297.

36. K. A. Chen, M. Feuer, K. Khokhani, N. Nan, and S. Schmidt, "The Chip Layout Problem: An Automatic Wiring Procedure," *Proceedings of the 14th Annual Design Automation Conference*, New Orleans, LA, 1977, pp. 298–302.
37. R. R. Habra, "Interactive Graphics for Wiring," *Proceedings of the International Conference on Interactive Techniques in Computer Aided Design* (ACM), Bologna, Italy, 1978, pp. 317–320.
38. C. McCaw, "Unified Shapes Checker—A Checking Tool for LSI," *Proceedings of the 16th Annual Design Automation Conference*, San Diego, CA, 1979, pp. 81–87.
39. J. S. Koford, G. A. Sporzynski, and P. R. Strickland, "Using a Graphic Data Processing System to Design Artwork for Manufacturing Hybrid Integrated Circuits," *Proceedings of the Fall Joint Computer Conference*, San Francisco, CA, 1966, pp. 229–246.
40. A. M. Barone, M. E. Harris, W. T. James, and D. M. Sheppard, "A Computer-Aided Method for Checking and Making Integrated Circuit Masks," *Technical Papers*, Western Electronic Show and Convention, Los Angeles, CA, 1966, Session 1/4.
41. F. E. Grace, "Planning for Automated Artwork," *Proceedings of the 2nd National Conference of the Association for Precision Graphics*, Los Angeles, 1968, Section VI.
42. A. D. Levit, "ADL, An Automated Drafting Language," *Proceedings of the 2nd National Conference of the Association for Precision Graphics*, Los Angeles, 1968, Section V.
43. W. E. Donath and J. Lesser, "LAGER, A Language for the Digital Transcription of Design Patterns," *Research Report RC 1730*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1966.
44. General Information Manual, "Memory Graphics Processing System," Technical and Business Publications, Dept. 183, East Fishkill, NY, D60C-100-5-70, DCS2-6500.
45. Program Reference Manual, "GSP/Custom Chip Design System User's Guide," RM2-8032, Dept. 123, IBM East Fishkill, NY, Sept. 1969.
46. A. Weinberger, "Large Scale Integration of MOS Complex Logic, A Layout Method," *IEEE J. Solid-State Circuits* **SC-2**, 182–190 (1967).
47. H. Fleisher, A. Weinberger, and V. Winkler, "The Writeable Personalized Chip," *Computer Design* **5**, 55 (1970).

Received August 20, 1980; revised February 20, 1981

P. W. Case is located at the IBM System Communications Division laboratory, Neighborhood Road, Kingston, New York 12401. M. Correia is located at the IBM General Technology Division laboratory, East Fishkill Facility, Hopewell Junction, New York 12533. W. Gianopulos is located at the IBM Data Processing Product Group headquarters, 1000 Westchester Avenue, White Plains, New York 10604. W. R. Heller and R. L. Simek are located at the IBM Data Systems Division laboratory, and H. Ofek and T. C. Raymond at the IBM General Technology Division laboratory, both at Poughkeepsie, New York 12602. C. B. Stieglitz is located at the IBM System Products Division laboratory, P.O. Box 6, Endicott, New York 13760.