The Design of VLSI Design Methods

Lynn Conway

Xerox Palo Alto Research Center
Palo Alto, California 94304, U. S. A.

Abstract

The Mead-Conway VLSI design and implementation methodologies were deliberately generated to be simple and accessible, and yet have wide coverage and efficiency in application. An overview is given of the methods used to "design the design methodology." We sketch the results and the status of these methods, and of the associated infrastructure of university courses, computer network communities, silicon implementation systems, and silicon foundries in the United States.

Building on this context, we present newly evolving knowledge concerning the principled design of design methods. Revisiting the Mead-Conway experiences to illustrate specific concepts, we consider how the properties of particular systems of knowledge, methods, and infrastructure affect the rates and extents of knowledge generation, diffusion, convergence, displacement, and integration.

1. Origins of the work

During the early '70's, Carver Mead began a pioneering series of course in integrated circuit design at Caltech, presenting the basics of industry nMOS design practice at the time. Observing the students' successes in later doing projects using these basics, Mead sensed that it might be possible to create new, much simpler methods of IC design than those then used in industry.

In the mid 70's, a collaborative effort involving my group at Xerox PARC and Mead's group at Caltech was begun to search for improved, simplified methods for VLSI system design. We hoped to create methods that could be very easily learned and applied by system designers, people skilled in the problem domain of digital system architecture and design, but having limited backgrounds in the solution domain of circuit design and device physics.

2. The opportunity

Our research yielded important basic results during '76 and '77. We were able to formulate very simple rules for crafting and composing FET switches to do logic and make registers, so that system designers could easily visualize the mapping of synchronous digital systems into nMOS. We formulated a simple set of concepts for estimating system performance. We then created a number of design examples that applied and illustrated the methods.

The new methods can be visualized as a covering by one simple body of knowledge of the previously separate bodies of knowledge used by the system architect, logic designer, integrated circuit designer, and chip layout designer. Those existing layers of specialized knowledge had incrementally accumulated over many years without reorganization, while tracking a large accumulation of technology change. We had seized the opportunity inherent in the accumulated technology for a major restructuring and redesign of digital system design knowledge.

When using the methods, we as individual designers could conceptually carry a design and make all the decisions from architecture to chip layout. It furthermore seemed possible to explore the design space and optimize designs in ways precluded when using the usual sequence of narrow specialties. We had thus created a hypothetical new design methodology appearing to have great promise. But what could we do with this knowledge? I was very aware of the difficulty of evolving and bringing forth a new *system of knowledge* by just publishing bits and pieces of it in among traditional work.

3. The challenge

When new design methods are introduced in any technology, a large-scale exploratory application of the methods by many designers is necessary in order to evaluate and validate the methods. The more explorers involved, and the better they are able to communicate, the faster this process runs to completion. However, even if design methods have been proven useful by a community of exploratory designers, there remains the challenge of taking methods that are new and perhaps considered *unsound methods*, and turning them into *sound methods*. Here numbers are important again: A lot of usage is necessary to enable sufficient individual viewpoint shifts and social organization shifts to occur to effect the cultural integration of new methods, in a process bearing similarities to those involved in the integration of new paradigms in natural science [1].

New design methods normally evolve via ad hoc, undirected processes of cultural diffusion through loosely connected groups of practitioners [2]. When the underlying technology changes in some important way, various new design methods exploiting the change compete for market share of designer mind-time. Bits and pieces of design lore, design examples, design artifacts, and news of successful applications, diffuse through the interactions of individual designers, and through the trade and professional journals, conferences, and mass media. The integration time can be quite long compared to that needed to initiate an individual into the methods. Once a new method has widely integrated into practice, we finally see texts and university courses introduced on the subject.

We've come to visualize the cognitive and social processes involved in the cultural integration of new knowledge as important factors in the "design" of the knowledge. As our understanding of such processes expands [2, 3, 4], and the relevant scaling effects are quantified, we are challenged to discover predictable, controllable, efficient alternatives to the unpredictable, undirected, natural processes. Back in 1977, we set out to meet this challenge as best we could, in the particular case of our hypothesized design methods. There was little prior work to guide us on our way.

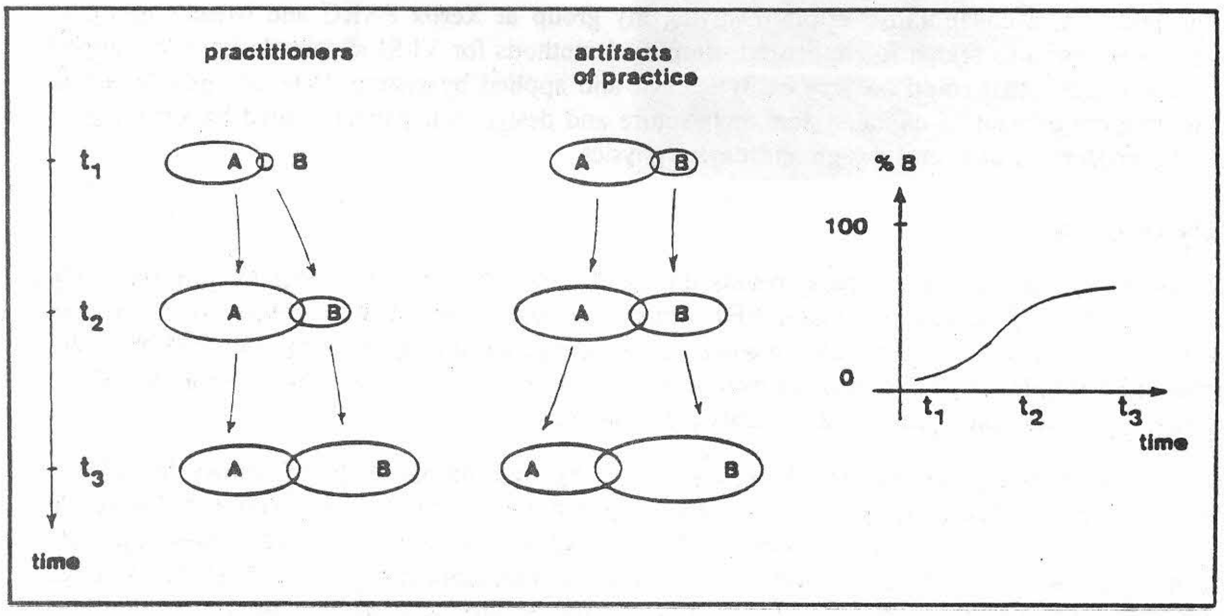


Figure 1. Knowledge diffusion and evolution.
This figure shows two competing methods (e.g., sailing ships and steam ships) labeled A and B. Social historians of technology [2] measure the population of practitioners and their artifacts over time. In this example, method B is gradually displacing method A as indicated by the size of the population diagrams and by the slope of the S-curve on the right. Actual diffusion of methods can follow more complicated patterns as new areas open up, and as groups displace each other or expand to compete in other areas.

4. The receiving community

At the time, the industrial world of system designers and IC designers was fragmented into a vast array of independent, competing clans having very different practices. Design groups specialized in different market application areas, and were further divided by the technology of implementation (nMOS, CMOS, etc). Industrial secrecy had fostered local craft practices, and cultural drift had produced wide gaps between firms. Within each clan, expertise was further split by divisions of labor such as system architecture, logic design, circuit design, and layout design. As a result, most architects were unable to understand layouts, and most layout designers unable to understand the system-level functions of chips, even within their own domains of application and technology.

Under such circumstances, for whom should we design the design knowledge? The selection of a receiving community in which to test our new methods would be a key decision. We decided to bypass the fragmented world of traditional practitioners in industry. Our hypothetical new synthesis of knowledge would appear too simple and non-optimal, and any systematic advantages it had would remain invisible, when viewed from any particular specialized perspective in that world. We chose instead to create a new "integrated system" design community, by propagating the knowledge into a community of students of digital system design in selected key universities. In this way we hoped to experiment with the methods, refine them, and propagate any useful results at least into the practices of the next generation of system designers.

5. Experimental method

In this research we've often applied a basic method of experimental computer science, a method especially useful when creating systems of primitives and composition rules that many other people will then use to generate larger constructs: We test and evolve a new concept by constructing a prototype system embodying that concept, running the system, and observing it in operation. Modifications are made until the concept is proven or disproven in actual use. This simple, iterative procedure is sketched in Figure 2. After experimentation has generated sufficient knowledge, we may move on to some later phase in the concept's evolution. In this way, a workable concept may be experimentally evolved all the way into use as an operational system (see Fig. 3).

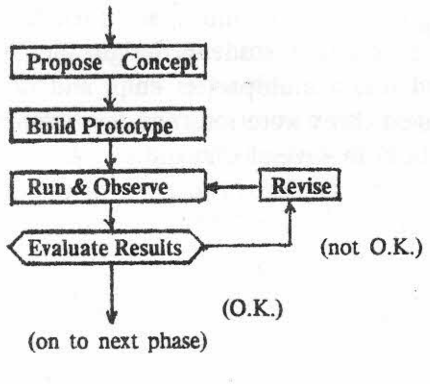


Fig. 2. Experimental Method

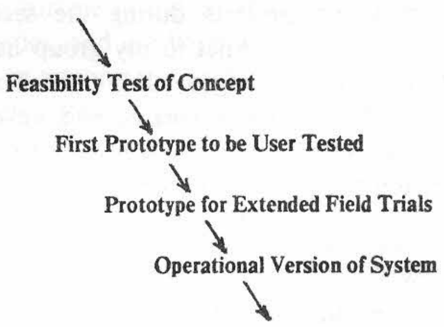

Feasibility Test of Concept
First Prototype to be User Tested
Prototype for Extended Field Trials
Operational Version of System

Fig 3. Phases in a System's Evolution

6. Community infrastructure

The rate of system evolution under this experimental method is greatly affected by the computer and communication infrastructure available to the community of researchers and experimental users. Our work took great advantage of the advanced infrastructure provided by the ARPAnet. Although we have yet to develop a quantitative understanding of the many social-evolutionary scaling-effects of such infrastructure, many profound qualitative effects are directly observable [5].

Such computer-communication environments enable rapid diffusion of knowledge through a large community, most likely as a result of their high social branching-ratios, short time-constants, and flexibility of social-structurings when compared to traditional alternatives. Under proper leadership, such networks enable large, geographically dispersed groups of people to function as a tightly-knit research and development community. It becomes possible to modify a system interactively while it is under test, by broadcasting messages to the user community in response to user feedback identifying system bugs, etc. It becomes relatively easy to achieve convergence on standards, especially if such standards enable access to interesting servers and services. The network also enables rapid accumulation of sharable knowledge, since much of what occurs is in machine representable form and thus easily stored and propagated.

7. The MPC adventures

I'll now sketch how experimental method and computer-communications infrastructure were applied to the directed evolution and cultural integration of the Mead-Conway design methods. For more detailed information about these experiments, see reference [6].

In 1977, while speculating about how to evaluate our hypothesized new design methods, I got the idea of evolving a book as a mechanism for organizing the experimental evolution and propagation of the methods. And so in August 1977, Mead and I began work on the Mead-Conway text [7]. We hoped to document a complete but simple system of design knowledge in the text, along with detailed design examples. Caltech students began work in parallel on the important "OM2" design example, using the new design methods as they were being documented.

We quickly drafted the first three chapters, and collaborators began testing the material in university MOS design courses that fall. The OM2 example was incorporated into the next draft of five chapters of the text, and collaborators tested that material in university courses in the spring of '78. These early drafts were rapidly debugged and improved in response to immediate feedback from the courses, feedback obtained by using the ARPAnet to interact with university collaborators. Our methods appeared to be very easily learned and applied by students, the work gained momentum, and by the summer of '78 we had completed a draft of the entire textbook.

During the summer of 1978, I prepared to visit M.I.T. to introduce the first VLSI design course there. This was the first major test of our new methods and of a new intensive, project-oriented form of course. I spent the first half of the course presenting the design methods, and then had the students do design projects during the second half. The resulting student design files were transmitted via the ARPAnet to my group at PARC, merged into a multiproject chip, and rapidly conveyed through prearranged mask and fab services. Packaged chips were returned to students six weeks later. Many projects worked, and we discovered the bugs in several that didn't. As a result of this experience, we found bugs in the design methods, the text, etc. Thus project implementation not only tested the projects; it also tested the design methods, the text, and the course.

While reflecting on this experience, and on our collaborations with colleagues via the ARPAnet, I realized how the course and the knowledge might be transported to many new environments: If we could somehow support rapid implementation of many projects produced by many universities, we might trigger the running of courses at many schools, and thus conduct a very large test. If successful and of sufficient scale, such a test might effect the cultural integration of the methods. I pondered ways we might expand our infrastructure to conduct such an experiment.

During the spring of '79 the final manuscript of the text was prepared for publication, and other materials [8, 9] were prepared to help transport the knowledge into other environments. That summer we offered intensive courses for instructors preparing design courses, and by the fall quite a few universities were prepared to offer courses. We at PARC gathered up our nerve and announced to these schools: "If you offer a VLSI design course, we will take any resulting projects that you transmit over the ARPAnet on a specified date, implement those projects, and send back packaged chips shortly after the end of your course!" This multi-university, multiproject chip implementation effort came to be known as "MPC79," and took on the characteristics of a great "network adventure," as about a dozen universities became involved in the experiment.

During the fall Alan Bell pioneered the creation at PARC of a "VLSI Implementation System", a software system enabling remote shared access to mask and fab services. The system contains a user message handler and design file processing subsystem, a die-layout planning and merging subsystem, and a CIF to MEBES format-conversion subsystem to prepare the data for hand off to the foundry.

The MPC79 events were coordinated by broadcasts of detailed "informational messages" over the APRAnet to our university collaborators, and by electronic messages from users to the system at PARC. At the conclusion of the courses, our implementation system generated MEBES mask specifications containing some 82 projects from 124 participating designers, merged into 12 die-types that were distributed over two mask sets. Just 29 days after the design deadline time at the end of the courses, packaged chips were shipped back to all MPC79 designers [10].

Many of the projects worked as planned, and the overall activity was a great success. Several projects springing from our efforts established important new architectural paradigms, later reported in the literature [11, 12, 13]. Because of the huge scale of the these MPC experiments and demonstrations, enabled by the ARPAnet computer-communications network and our VLSI implementation system, the possible merits of the design methods, the courses, and the implementation systems were brought to the attention of many research and development leaders.

8. Results

Table 1 tabulates the courses, design-aid environments, and project experiences (as of the summer of 1980) at the group of 12 universities that collaborated with us during the MPC efforts [14]. Interesting patterns of knowledge diffusion and convergence can be derived from this table and from the data on which it is based. It may help you visualize how rapidly the new system of knowledge swept through this university community, most of whom are on the ARPAnet.

The design methodology has become well integrated into the university computer science culture and curriculum, and thus into the next generation of digital system designers. Courses are being offered in the present school year at more than 100 universities. During the early MPC work, new analysis aids appropriate for our design methods began to surface in the universities [15, 16]. Tools of that type are now in routine use at many other universities, and are beginning to find widespread use in industry. A VLSI implementation system is now in routine use at Xerox PARC to support exploratory VLSI system design within Xerox. A similar system is operated by USC/ISI for the VLSI research community supported by Defense Advance Research Projects Agency (DARPA). That community is located in certain large U. S. research universities (M.I.T., CMU, Stanford, U.C. Berkeley, Caltech, etc.), and within a number of Department of Defense research contractors.

Many industrial organizations now offer internal courses on the design methodology, and/or have design projects underway using the methods. In addition, a number of firms have been established to seize the many new business opportunities associated with this work, opportunities in VLSI chip designs, design aids, implementation systems, foundry service brokerage, and foundry services.

An important practical effect of the work has been the isolation of highly transportable forms of the knowledge, and of methods for rapidly inserting that knowledge into other environments having adequate supporting infrastructure. Perhaps the classic example of a rapid startup of a new design community under these methods is the recent Australian "AUSMPC" effort led by Craig Mudge. Mudge created a design-aid environment in Australia in the fall of '81, provided an instructor's course in February '82, supported the offering of multiple university courses in Australia in the spring of '82, and supported project implementation following those courses [17]. These activities rapidly and efficiently integrated the design culture into the Australian engineering community.

TABLE 1.

Computing and Design Environments for 1980-81 VLSI Design Courses at Universities that participated in MPC79/MPC580.

[Reprinted with permission of *LAMBDA, The Magazine of VLSI Design*]

UNIVERSITY:	MIT	Caltech	Stanford	CMU	U.C.B.	U. of Col. (C.S.)	U. of Illinois	U. of Wash.	U. of Rochester	UCLA	Wash. U. (St. L.)	U.S.C.
COURSE INFORMATION												
Instructor(s)	J. Allen, L. Glasser	C. Mead, C. Sietz	J. Newkirk, R. Mathews	R. Sproull	R. Newton C. Sequin	J. Murray	J. Abraham E. Davidson	T. Kehl	E. Kinnen G. Kedem	V. Tyree	F. Rosenberger	J. Nelson
Course #	6.371	CS181, CS182	EE271	15-846	CS248	EE594	EE325	CS590D	492	M258A, B, C	EE463	EE599
Sem. or Qtr.	F, Sp	F-W-Sp	F, Sp	Sp	F	F	F, Sp	F, W, Sp	F	F-W-Sp	Sp	F
#stud./Class	35	40	60	30	50	20	20	15	25	20	25	30
COMPUTING ENVIRONMENT												
CPU	DEC-20	DEC-20, VAX	DEC-VAX	DEC-VAX	DEC-VAX,	DEC-20	HP1000	DEC-20, VAX	ALTO, VAX	DEC-VAX	DEC-20	DEC-KL10
Op. Sys.	TOPS-20	TOPS-20, UNIX	UNIX	UNIX	UNIX	TOPS-20	RTE IV	TOPS-20, VMS	ALTO, UNIX	UNIX	TOPS-20	TOPS-10
Prog. Lang.	LISP, APL, CLU	Simula, C	C	C	C	Simula, Pascal	Pascal	FORTRAN	C, Pascal	C, Pascal	Simula, FORTRAN	Pascal
DESIGN AID ENVIRONMENT												
Synthesis aids	PLAG, MI	MG, MI	PLAG	PLAG	PLAG, SGC	MG, MI	- - -	PLAG, MI	- - -	- - -	PLAG	- - -
Description aids	SLL	SLL	SLL	SLL,IGL	SLL,IGS	SLL	IGL	SLL	IGL	SLL	SLL	SLL
Analysis aids	CX,SS, DRC,CS	CX,SS,CS	CX,DRC,SS	CX,SS,DRC	CX,SS,CS	CS	CS	DRC	CX,SS,DRC	CS	CS	CS
Viewing aids	CPP, BRP	CPP,CRP, CD	BRP	CPP,BRP, CD,BD	BRP,BD	CPP	CPP,BD	CPP	CPP,BRP, CD,BD	CPP	CPP,CD	CPP
Testing aids	MTE	MTE	MTE	- - -	- - -	- - -	MTE	- - -	MTE	MTE	- - -	- - -
PROJECT EXPERIENCE												
(# projects, # designers)												
MPC79	15, 27	24, 28	19, 35	5, 5	4, 4	1, 1	5, 8	1, 3	5, 9	- - -	- - -	- - -
MPC580	11, 13	21, 22	32, 59	12, 17	8, 12	12, 21	8, 13	15, 15	3, 3	9, 9	9, 11	10, 15

SUMMARY OF DESIGN-AID CODES:
BD: B/W Display; *BRP:* B/W Raster Plotter; *CD:* Color Display; *CPP:* Color Pen Plotter; *CRP:* Color Raster Plotter; *CS:* Circuit Simulator; *CX:* Circuit eXtractor; *DRC:* layout Design Rule Checker; *IGL:* Interactive Graphic Layout; *IGS:* Interactive Graphic Sticks; *MG:* Module Generator; *MI:* Module Interconnector; *MTE:* Minimal Test Environment; *PLAG:* PLA generator; *SGC:* Sticks-to-layout Generator/Compressor; *SLL:* Symbolic Layout Language; *SS:* Switch Simulators; *SSL:* Symbolic Sticks Language

Beyond these important direct results, we've gained much insight into research methods that can be reapplied in the future. For example, visualize the developing design methods as a multilevel cluster of systems undergoing joint evolution, as in Fig. 4. Each system was experimentally conveyed through the various phases of its own evolution. Perhaps you can see the importance of rapid implementation of projects in (i) stimulating designs, design courses, and the creation of design environments, and (ii) rapidly closing the experimental loops on all systems in the hierarchy. Such techniques could be reused to evolve design methods in other technological arenas.

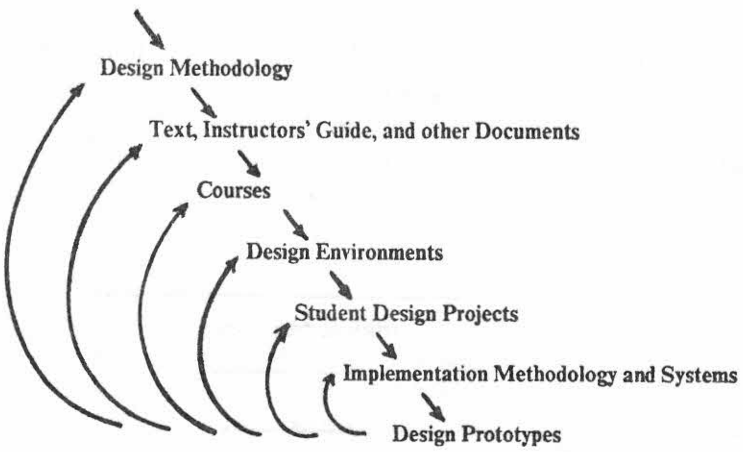


Figure 4. The Joint Evolution of the Multi-Level Cluster of Systems

Note that such enterprises are organized at the meta-level of research methodology and heuristics, rather than being planned in fully-instantiated detail. There's a strong element of chance in such explorations. The unfolding of events depends upon what is discovered, and upon how the discoveries are seized upon and exploited by the overall community of explorers.

9. Ongoing work in knowledge engineering {references [18, 19] provide background material on this new field}

In the preceding sections, the ideas for creating and refining methods *per se* were described anecdotally in terms of a generate-test-revise cycle, with novel infrastructure applied to support the substantial exploratory usage required to debug, refine, and diffuse the methods. While resuable qualitative methods and heuristics can be derived from these experiences, we still seek deeper, quantitative insights into the cognitive and social processes underlying the resulting successes.

As part of a recent knowledge engineering effort, in a collaboration between my group at Xerox and the Heuristic Programming Project at Stanford, we have begun to obtain interesting insights into properties of the Mead-Conway knowledge itself [20, 21, 22]. We hypothesize that the knowledge embedded in the methods gives practitioners a *cognitive advantage*, characterized as a simpler cognitive mapping from architectural concepts down through layouts in silicon. We have begun to further engineer that knowledge under the guidance of certain new principles. Our aim is to explore the application of AI methods towards the creation of improved infrastructure, improved and formally represented knowledge, and expert assistants for the VLSI design community.

By suggesting that *knowledge* is subject to design [22], we place knowledge engineering among the sciences of the artificial [23]. Designed objects are *artificial* in that they are man-made and shaped to suit a designer's purposes for use in some environment. As an engineering practice develops, engineering principles emerge that account for the constraints imposed by designer goals and an environment. Although no substantial body of knowledge engineering principles has yet been articulated, a partial picture of some of its elements is beginning to appear. Refer to [21, 22] for examples of this work, examples suggesting that a reusable body of practice may eventually emerge.

10. Abstraction frameworks

When our knowledge engineering work began, the Mead-Conway community had not produced a formal body of design knowledge, from the knowledge engineering point of view. The community's methods were relatively simple, and a descriptive textbook existed. But most of the embedded knowledge was informal, and was communicated in the traditional manner: by way of examples. It was clear from the examples that the designers worked within multiple levels ranging from abstract system descriptions to chip layouts. During efforts to formalize these abstraction levels, we gained insight into how certain of the levels were different from those traditionally used in IC design.

Traditional digital IC design has four levels of specialization: System architects perform the highest level of design, specifying the function blocks, registers, and data paths of a design. Logic designers then work out the details of the logic implementing the functional blocks. Circuit designers then specify the circuit devices and interconnections to be used to implement the logic designs. Finally, layout designers specify geometric patterns in the various layers of the integrated circuit chips to implement the devices and interconnections. Implicit in this division of labor is a set of informally shared abstraction levels, one per specialty.

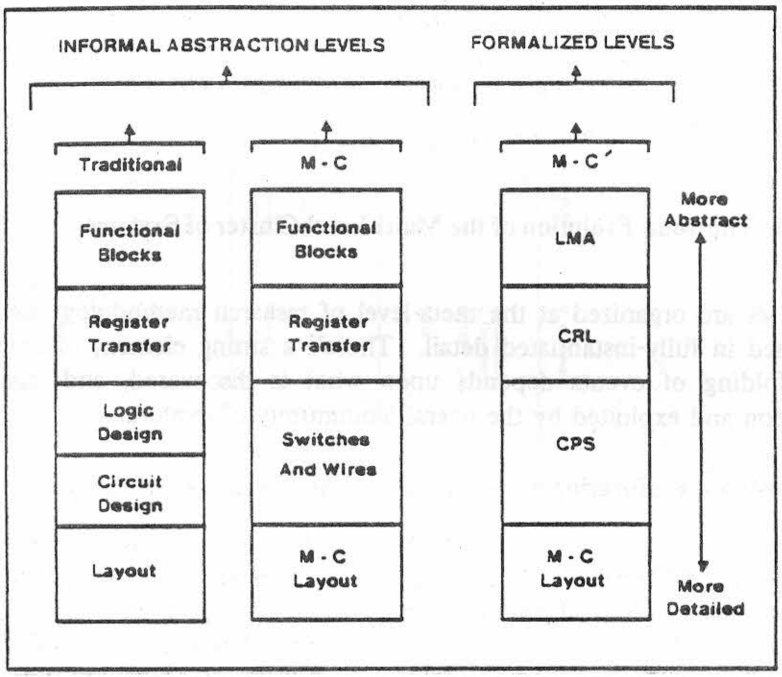


Figure 5. Comparison of relative placements of abstraction levels.
The sequences of levels are shown for traditional IC design methods, informal Mead-Conway methods, and also for re-engineered, formalized Mead-Conway methods disussed in Section 11. [22]

In contrast with traditional practice, the Mead-Conway methods bypass the requirement for Boolean logic gate representation as an intermediate step in design. They thus eliminate an unneeded step in the design process, a step that often introduces unwanted complications while precluding important design constructs. The methods also advocate the consistent use of simple charge-storage methods instead cross-coupled gates for saving state between register transfer stages. A simple "primitive switches" design step, which can generate not only logic gates when needed, but also steering logic and charge-storing registers, replaced *both* the logic-gate step and the detailed electrical circuit-design step of previous methods. Included are simplified electrical models, timing models and layout rules for guiding design under the resulting methods. The methods are sufficiently simple to learn and to use so that an individual designer can now rapidly carry a design from architecture to layout in silicon, where previously a team of specialists would have been required.

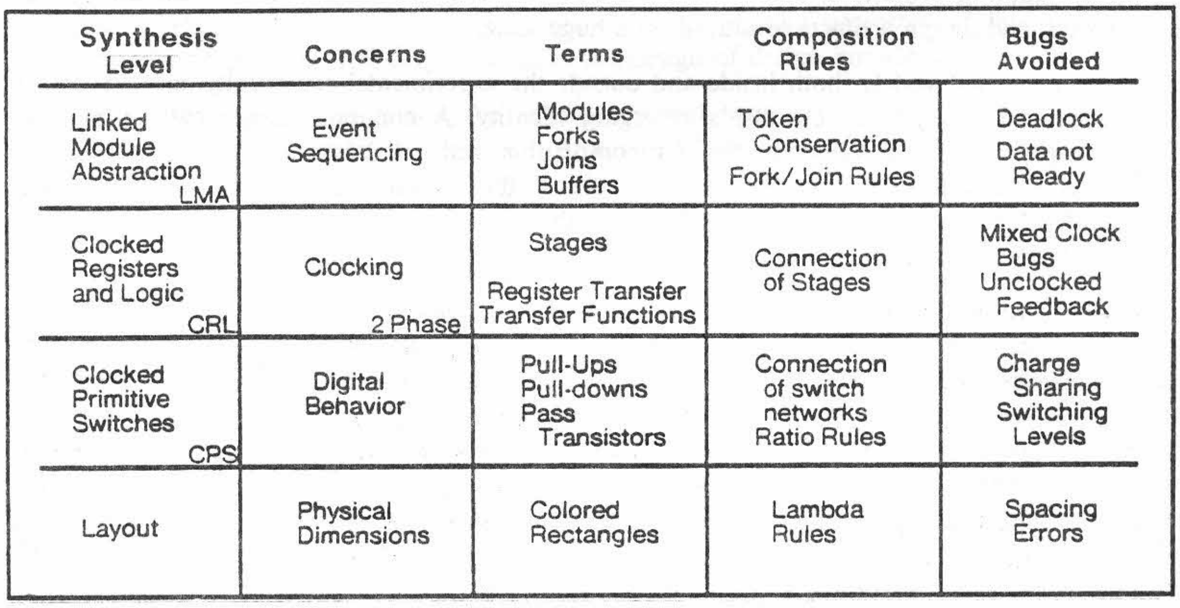
11. Cognitive factors

The Mead-Conway work simplified VLSI design practice, apparently by reducing the amount of knowledge required and by restructuring the form of the knowledge. We hypothesize that further analysis of the system of abstraction levels embedded in the Mead-Conway methods, as contrasted with the traditional levels, will reveal the sources of some of the advantages of the methods. In order to conduct such analyses, and to embed the methods in an expert system, we have proceeded to formalize these abstraction levels. We are also studying the *general* properties of sets of abstractions, hoping to find bases for comparing and understanding the relative utility of different sets, and to perhaps even find principles for *designing* sets of abstractions.

The importance of effective problem decomposition for taming large search problems has been recognized in AI for many years. This idea was quantified by Minsky [20], who explained the combinatorial advantage of introducing *planning islands* for reducing search by what he called a "fractional exponent." Planning islands decompose a search into a set of subproblems. Although such search reduction can be dramatic, it depends heavily on the placement of the islands.

A language that enables synthesis of suitable abstractions can guide the decomposition of problems into subproblems. This idea can be iterated to yield an ordered set of languages providing intermediate abstractions. When an ordering of a language set is found that on average provides low backtracking in problem solving, the languages can effectively guide problem decomposition in the domain, and we say that the languages exhibit the *planning island effect.* The influence on a problem solver is akin to that proposed in the Whorfian hypothesis: language shapes the patterns of habitual thought [24]. A designer who systematically carries a design through synthesis in several languages is guided by an "invisible hand" that determines the kinds of decisions made at each step.

Using certain new principles [21, 22], we have created the set of *synthesis languages* characterized in Figure 6. The languages are a re-engineered formalization of the Mead-Conway abstraction levels, with the inclusion of a new type of top abstraction level (see Figure 5). Each language provides a vocabulary of terms, and composition rules defining legal combinations of the terms. The concerns of each language are characterized by specific classes of *bugs* avoided when the composition rules are followed. Collectively, the synthesis languages factor the concerns of a digital designer.



Synthesis Level	Concerns	Terms	Composition Rules	Bugs Avoided
Linked Module Abstraction LMA	Event Sequencing	Modules Forks Joins Buffers	Token Conservation Fork/Join Rules	Deadlock Data not Ready
Clocked Registers and Logic CRL	Clocking 2 Phase	Stages Register Transfer Transfer Functions	Connection of Stages	Mixed Clock Bugs Unclocked Feedback
Clocked Primitive Switches CPS	Digital Behavior	Pull-Ups Pull-downs Pass Transistors	Connection of switch networks Ratio Rules	Charge Sharing Switching Levels
Layout	Physical Dimensions	Colored Rectangles	Lambda Rules	Spacing Errors

Figure 6. Synthesis languages of an expert system for aiding VLSI design.
Each language has a set of *terms* that can be composed to form systems and a set of *composition rules* that define legal combinations of the terms. The concerns of each language are characterized by specific classes of *bugs* avoided when the composition rules are followed. [21]

While working to quantify the utility of sets of synthesis languages for creating well-spaced planning islands, we have found it useful to define several measurable branching factors between levels [22]: The *choice factor*, a measure of the alternatives in decision making, is defined as the average number of possible alternative implementations for a primitive term at the next lower level. The *expansion factor*, a measure of the expansion of detail, is defined as the average increase in the amount of information for specification of a primitive term at the next lower level. Quantifications of these factors for our synthesis languages are still a ways off, and will depend on a careful information-theoretic analysis. However, as our experience with our languages expands, we will attempt to apply such measures and further refine our abstraction levels in response to the results.

12. Social factors, infrastructure, and scaling effects

The generation, selection, and diffusion of knowledge depend on a variety of social, ecological, and economic factors, in addition to the properties of the knowledge itself. Social structures often mediate the generation process through complex membership and career feedback processes [3], Social networks of knowledge carriers, sometimes invisible to outsiders, can provide a means for rapid diffusion of new knowledge [4]. Technological diffusion and displacement are increasingly being scrutinized under quantitative methods, resulting in useful new insights and models of the underlying cultural and economic processes [2]. As we better understand these natural processes, we can hypothesize and test how they might be modified by improved methods and infrastructure.

During our work we uncovered useful generation and propagation methods that we've since refined and successfully reapplied. I'll briefly sketch several of the ideas. Practical knowledge is usually acquired in an initiation process based on observation and imitation of existing practice. Thus knowledge diffusion is scaled by the proximity of models and frequency of opportunities for observation and imitation. We used the network infrastructure to increase the rate of production of examples that could be copied. By extending the community for observation of design examples, we scaled up the opportunities for imitating, competing with, and evolving the example set.

Having many sites of action caused many competing design languages to be developed. Interchange among these sites could be an n^2 problem; by developing a standard design interchange format we linearized the format conversion problem, and enabled widespread network access to services for implementation. We promoted the integration of that standard into the community by providing free implementation of projects in that format. As a result, design files were easily transported from group to group, and design artifacts produced on a huge scale.

Those artifacts diffused widely, both inside and outside the experimental community, and served as attractants and tokens of the new clan's emerging identity. A common design culture rapidly emerged, presenting growing opportunities for competition and collaboration. We found that we could "design" the competitive-collaborative structure of the community, simultaneously initiating activities and injecting particular material in one subset of universities, and reflecting the later results towards another subset.

Looking ahead, we now believe that the merging of knowledge engineering into the existing cultural infrastructure can enable large increases in the rates and extents of knowledge generation and diffusion processes, as suggested in Figure 7 [22]. A common literacy regarding the representation and mechanization of practical knowledge would encourage placement of more effort into the design, rather than the routine application, of knowledge. Knowledge engineered for good cognitive matching to receiving cultures will diffuse more rapidly. Knowledge engineered for more efficient cognitive and computational processing will provide economic advantages. The mechanization of knowledge in expert systems will enable huge scale-ups of its problem-solving applications. Of course, for these results to occur, the practice of knowledge engineering must itself successfully integrate into our culture under the operation of displacement and diffusion processes!

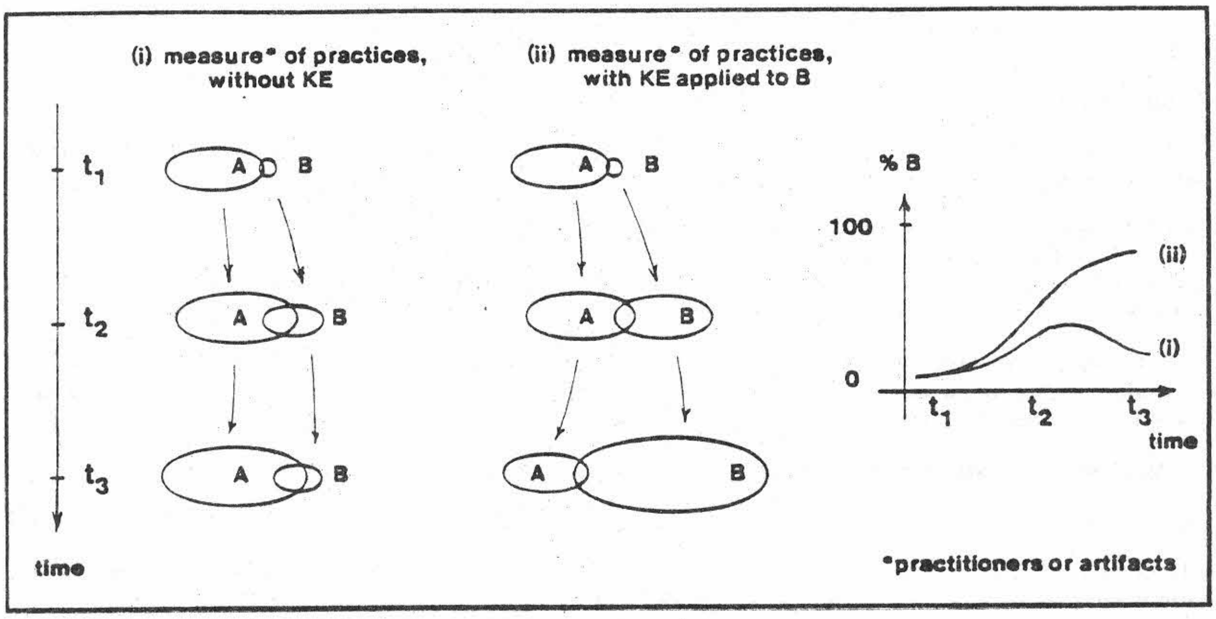

(i) measure* of practices, without KE
(ii) measure* of practices, with KE applied to B
t1
t2
t3
time
A
B
% B
100
0
(ii)
(i)
t1
t2
t3
time
*practitioners or artifacts

Figure 7. Knowledge engineering mediating the transformation of knowledge.
The processes that underlie the diffusion of technology and knowledge depend on a variety of factors including properties of the knowledge itself. Does it provide economic advantages? Is it too complex to apply? Can it propagate through a particular culture? Knowledge engineering can potentially augment the infrastructure in which these natural transformation, displacement, and diffusion processes operate.

Conclusions

As a result of the properties of the underlying knowledge, the research methodology used, and the very large scale of the interactions with the university community, our design methods evolved unusually rapidly, going from concept to integration within industry within just a few years. The story of the work yields many lessons. We have especially learned the utility of including and dealing with relevant cognitive and social phenomena during the processes of theory formation, experimentation, and theory revision involved in the design of design knowledge.

Through this work we have gained the confidence and insight to look more deeply into the properties of knowledge itself, and into the processes of its evolution. We are thereby finding opportunities for the practical application of results from computer science and artificial intelligence to the development and application of new principles for the engineering of knowledge.

Acknowledgements

I am ever grateful to my many university collaborators, the folks who have played such key roles in bringing this work forward. The synergistic combination of their contributions, merging ideas from many viewpoints, has given the research a sense of excitement and common purpose. Thanks go to to the Defense Advanced Research Projects Agency for consistently strong support of computer science research, and I note that the Stanford University component of the knowledge-based VLSI design project is funded by DARPA. Thanks also go to the Xerox Corporation for providing the intellectual and computational environment in which this work could be done. Finally, I owe very special thanks to my compatriots Mark Stefik, Alan Bell, and Danny Bobrow, for their intellectual stimulation and creative contributions to the ongoing work.

References

1. T. Kuhn, *The Structure of Scientific Revolutions*, 2nd Ed., Univ. of Chicago Press, Chicago, 1970.

2. D. Sahal, *Patterns of Technological Innovation*, Addison-Wesley, Reading, MA, 1981.

3. B. Latour and S. Woolgar, *Laboratory Life: The Social Construction of Scientific Facts*, Vol. 80, Sage Library of Social Research, Sage Publications, Beverly Hills, 1979.

4. D. Crane, *Invisible Colleges: Diffusion of Knowledge in Scientific Communities*, Univ. of Chicago Press, Chicago, 1972.

5. J. Lederberg, "Digital Communications and the Conduct of Science: The New Literacy," *Proceedings of the IEEE*, Vol., 66, No. 11, November, 1978.

6. L. Conway, "The MPC Adventures: Experiences with the Generation of VLSI Design and Implementation Methodologies." *Proc. of the 2nd Caltech Conference on VLSI*, Jan. 19-21, 1981. Reprinted as Xerox PARC Technical Report VLSI-81-2, Jan. 1981. Accepted for publication in *Microprocessing and Microprogramming: The Euromicro Journal.*

7. C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA, 1980.

8. R. Hon and C. Sequin, *A Guide to LSI Implementation, 2nd Ed.*, Xerox PARC Technical Report SSL-79-7, Jan. 1980.

9. L. Conway, *The MIT '78 VLSI System Design Course: A Guidebook for the Instructor of VLSI System Design*, Limited Printing, Xerox PARC, Palo Alto, CA, August 1979.

10. L. Conway, A. Bell, M. Newell, "MPC79: A Large-Scale Demonstration of a New Way to Create Systems in Silicon," *LAMBDA, the Magazine of VLSI Design*, Second Quarter, 1980.

11. J. Clark, "A VLSI Geometry Processor for Graphics," *Computer*, Vol. 13, No. 7, July, 1980.

12. J. Holloway, G. Steele, Jr., G. Sussman, A. Bell, *The Scheme-79 Chip*, AI Memo No. 559, Artificial Intelligence Laboratory, M.I.T., January 1980.

13. R. Rivest, "A Description of a Single-Chip Implementation of the RSA Cipher," *LAMBDA, the Magazine of VLSI Design*, Fourth Quarter, 1980.

14. L. Conway, "University Scene," *LAMBDA, the Magazine of VLSI Design*, Fourth Qtr., 1980.

15. C. Baker and C. Terman, "Tools for Verifying Integrated Circuit Designs," *LAMBDA, the Magazine of VLSI Design*, Fourth Quarter, 1980.

16. R. Bryant, "An Algorithm for MOS Logic Simulation," *LAMBDA, the Magazine of VLSI Design*, Fourth Qtr., 1980.

17. J. Mudge and R. Clarke, "Australia's First Multi-Project Chip Implementation System," *Proceedings of the National Conference on Microelectronics*, Adelaide, Austrialia, 12-14 May 1982.

18. E. Feigenbaum, "The Art of Artificial Intelligence - Themes and Case Studies of Knowledge Engineering," *Proc. of the 1978 National Computer Conference*, AFIPS Press, Montvale, N.J., 1978.

19. M. Stefik, J. Aikins, R. Balzer, J. Benoit, L. Birnbaum, F. Hayes-Roth, and E. Sacerdoti, "The Organization of Expert Systems: A Prescriptive Tutorial." *Artificial Intelligence*, Vol. 18, No. 2, March 1982. Reprinted as Xerox PARC Technical Report VLSI-82-1, Jan. 1982.

20. M. Minsky, "Steps toward Artificial Intelligence." In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and Thought*, McGraw-Hill, New York, 1961.

21. M. Stefik, D. Bobrow, A. Bell, H. Brown, L. Conway, and C. Tong, "The Partitioning of Concerns in Digital System Design." *Proceedings, Conference on Advanced Research in VLSI*, Jan. 25-27, 1982. Artech House, Cambridge, MA.

22. M. Stefik and L. Conway, "Towards the Principled Engineering of Knowledge," *AI Magazine*, American Association for Artificial Intelligence, Menlo Park, CA, (in press).

23. H. Simon, *The Sciences of the Artificial*, 2nd Ed., M.I.T. Press, Cambridge, MA, 1981.

24. B. Whorf, "The Relation of Habitual Thought and Behavior to Language," in *Language, Thought, and Reality*, Technology Press, Cambridge, MA, 1956, pp. 134-159.