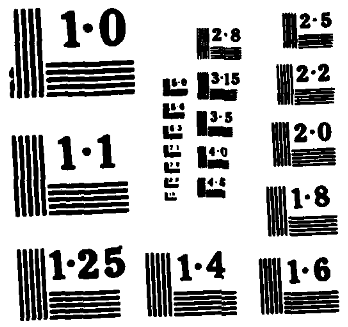
AD-A163 437 SCHEMA: AN ARCHITECTURE FOR KNOWLEDGE BASED CAD(U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE C G CLARK ET AL. OCT 85 N00014-80-C-0622 1/1

UNCLASSIFIED F/G 9/2 NL


1·0
2·8
2·5
3·15
2·2
3·5
2·0
1·1
4·0
4·5
1·8
1·25
1·4
1·6

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

CAMBRIDGE, MASSACHUSETTS 02139

(3)

VLSI Memo No. 85-271 October 1985

Schema: An Architecture for Knowledge Based CAD*

C. G. Clark and R. E. Zippel**

ABSTRACT

Schema provides an integrated environment for all aspects of the synthesis and analysis of electronic designs from PC boards through circuit and mask design of VLSI devices. It simplifies the development of synthesis and analysis tools by using uniform data structures and by making available libraries of standard routines and advanced control structures appropriate for CAD tool development. Because all tools in the Schema environment utilize the same abstract data structures it is easy for tools to interchange data about a design or even pieces of the design itself. Schema also permits much of the design to be done in a technology independent fashion by allowing the designer to delay implementation decisions until the last possible moment. The information associated with a particular component of a design is organized as a module. Modules contain schematics, icons, topologies, layouts, simulation results, and other descriptive information for this component. The descriptions contained in modules are implemented as procedures which utilize other modules in a hierarchical fashion. Schema is under joint development by MIT and Harris Corporation.

*This work was supported in part by the Defense Advanced Research Projects Agency under contract number N00014-80-C-0622 and by the Harris Corporation.

**Zippel: Laboratory for Computer Science, MIT, Room NE43-418, Cambridge, MA 02139, (617) 253-6028; Clark: Harris GSS, P. O. Box 37, Melbourne, FL 32902.

1-3

Schema
An Architecture for Knowledge Based CAD

G. C. Clark†— R. E. Zippel‡

†Harris GSS, P. O. Box 37, Melbourne, FL 32902, USA
‡MIT Laboratory for Computer Science, Cambridge, MA 02139, USA

Abstract

Schema provides an integrated environment for all aspects of the synthesis and analysis of electronic designs from PC boards through circuit and mask design of VLSI devices. It simplifies the development of synthesis and analysis tools by using uniform data structures and by making available libraries of standard routines and advanced control structures appropriate for CAD tool development. Because all tools in the Schema environment utilize the same abstract data structures it is easy for tools to interchange data about a design or even peices of the design itself. Schema also permits much of the design to be done in a technology independent fashion by allowing the designer to delay implementation decisions until the last possible moment. The information associated with a particular component of a design is organized as a module. Modules contain schematics, icons, topologies, layouts, simulation results, and other descriptive information for this component. The descriptions contained in modules are implemented as procedures which utilise other modules in a hierarchical fashion. Schema is under joint development by MIT and Harris Corporation.

SCHEMA is an environment for developing knowledge based. computer aided design tools for electronic systems. The three major goals of its design are:

- Provide an integrated environment for all aspects of the synthesis and analysis of electronic designs from PC boards through circuit and mask design of VLSI devices.
- Simplify the creation of computer aided design tools by encouraging and supporting their construction from libraries of standard routines, by using uniform data structures and by providing libraries of advanced control structures appropriate for CAD development.
- Allow the designer to delay making decisions until necessary; for example, the technology (TTL, ECL, gate array or custom MOS) used in a logic design need not be specified until timing simulations or physical design is begun.

The key to achieving these goals is the development of a totally *integrated design environment* where design tools easily communicate and cooperate. This has been achieved by the innovative software architecture used in the development of SCHEMA.

SCHEMA achieves coherence not by specifying the interchange formats to be used between different CAD programs, but rather by specifying the data structures the programs should use. SCHEMA specifies a set of abstract data types for dealing with electronic designs, and a set policies to be used when dealing with the new data types. This approach provides a common layer on which different CAD tools may built, and it allows the CAD tools to invoke each other and easily cooperate by interchanging pieces of electronic designs.

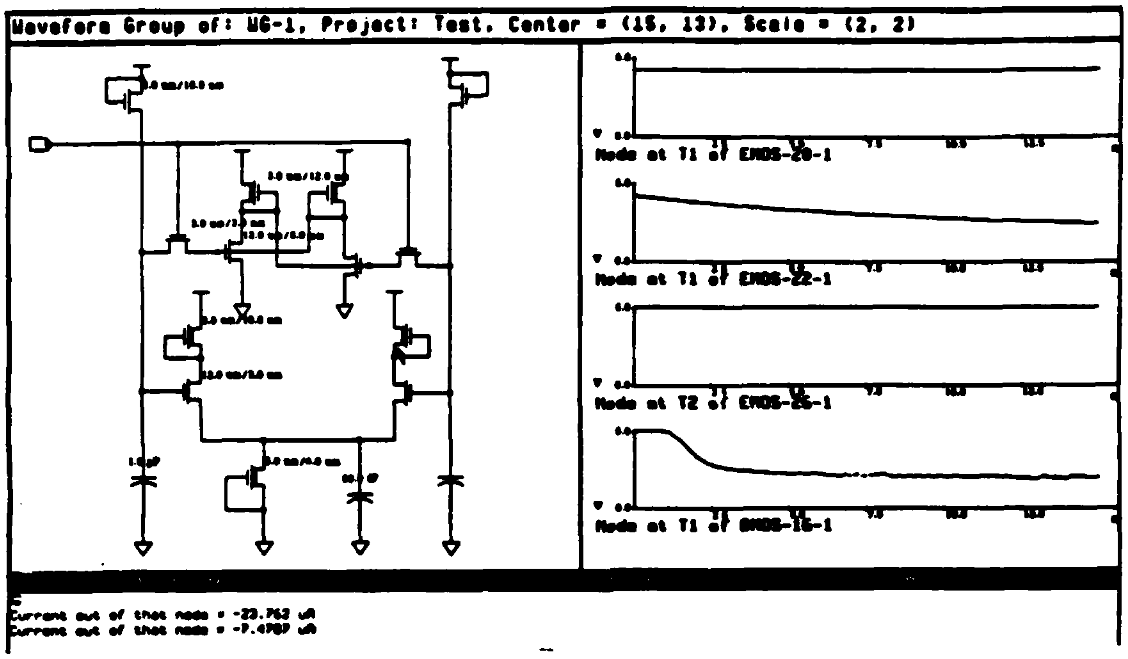
These data types are implemented using an object oriented programming system called Flavors[4]. These structures represent circuit topologies and schematics, mask artwork, floorplans and simulation waveforms (both digital and analog). Circuit topologies represent the connectivity of a circuit; schematics are represenations of the graphic images of a circuit that are drawn on paper. Since these structures are instances of flavors, they also incorporate pieces of code that allow them to directly provide procedural functionality. That is, a transistor contains the information and code required to display itself on the screen, write itself out to a file or participate in a simulation. This raises the semantic level at which the CAD tools deal with objects, simplifying their development. It also allows implementation and operation decisions to be delayed and even changed without modifying the code that makes use of them.

Modules

The basic component of a design in SCHEMA is a *module*. Each module consists of a *topology* and several *descriptions*, e.g. schematics, icons, layouts and simulation results. Examples of modules in a design include: an inverter, a half adder, an arithmetic logic unit, a data path, a cache, instruction fetch unit and a memory system. Each of these module includes not only the schematic (and its corresponding topology), but also the results of various tests that have been performed on the circuit (simulation results), documentation and design notes and physical specifications (VLSI layouts or PC board designs). The modules represent a complete view of a design component.

The designer rarely interacts directly with the topology of a module, but instead deals with the descriptions (schematics). The analysis tools (simulators, timing verifiers and other consistency checkers) work with the topology, and usually use the descriptions only for communicating with the designer. The only major exceptions are the physical design tools, VLSI layout system and wire wrap and PC board systems that, by necessity, must work with the physical descriptions.

The system ensures that the topology remains consistent with the descriptions provided by the designer through the use of timestamps and limited edit trails. It also warns the designer when two descriptions of a design become in-



Waveform Group of: WG-1, Project: Test, Center = (15, 13), Scale = (2, 2)

```
Current out of that node = -23.762 uA
Current out of that node = -7.4787 uA
```

consistent. This division allows the electronic designer to use the most appropriate mechanism for describing the design without worrying about getting formats correct for the CAD tools, and the CAD tool designer deals only with design descriptions that are both appropriate and "preparsed."

The topology and its descriptions are implemented as procedures, though they are usually edited via one of the *description editors*: schematic, layout or waveform. This procedural structure, similar to the approach used in DPL[1], allows a great deal of flexibility parameterizing the different components and provides an excellent point at which to install intelligent synthesis modules. For instance, in an earlier version of SCHEMA this was used to implement an ALU module that chose different carry look-ahead schemes depending on the width of the data word[2].

These hierarchical descriptions also incorporate a *multiple viewpoint* or *Slices*[3] mechanism to allow simulation and analysis modules to annotate the topologies. The multiple viewpoints are used to control the visibility of certain information to the CAD tools. For instance, transient analysis programs like SPICE want to be aware of parasitic capacitances and resistances while a simple logic simulator might not. Rather than generating the two different topologies for the different simulators, the same topology is used for both but the parasitics are only visible when the *transient* viewpoint is made visible by SPICE. This way annotations to the topology made by the two simulators can be examined by their conterparts easily.

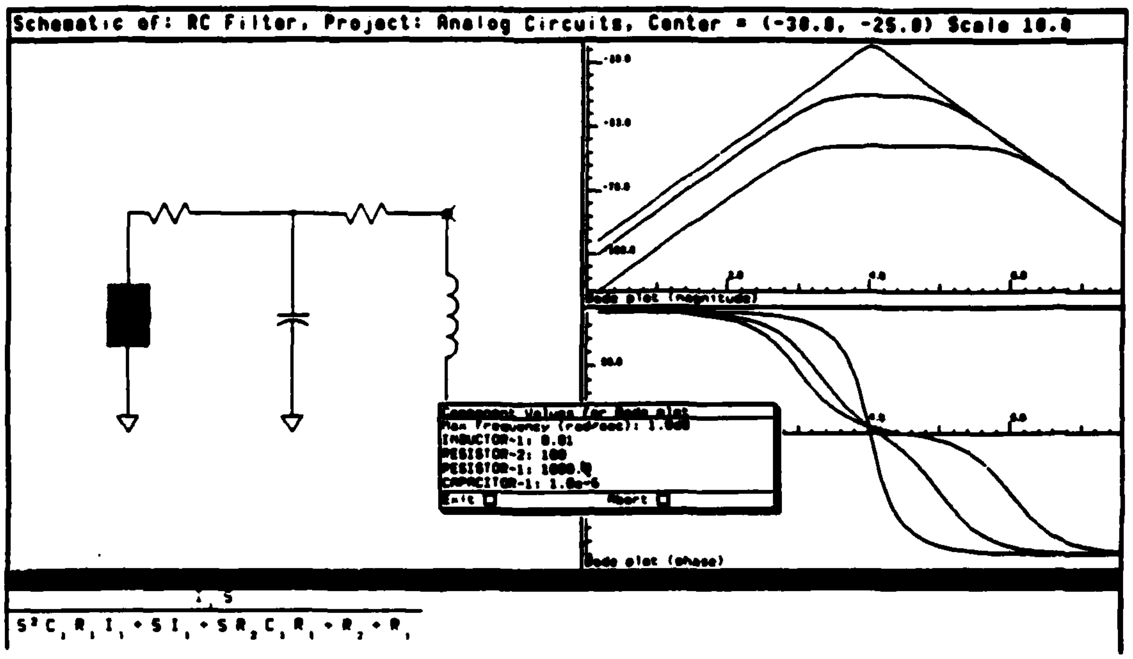
Project/Module Hierarchy

The modules and all other information relating to a design are collected into a *project*, which in turn can be a component of a larger project. For instance, there might be an *L machine* project that is used to hold all the design components of the L machine. Several different versions of the L machine might be designed, so there might be *TTL*, *CMOS* and *ECL* sub-projects of *L machine*. Within the *CMOS* project there might different projects to contain the design of the datapath, control logic, and memory management system. The modules of each of these projects would be combined by the main module contained in *CMOS* to produce the final chip.

Each designer maintains his or her own hierarchy of projects. The root of this hierarchy is called a *portfolio*. By having sub-projects point to the same save file, designers can share projects. This project/module hierarchy is a very useful way of organizing and managing the material related to a design.

Environments

By specifying an *environment* the designer makes precise what types of modules and tools should be available for the design. Each environment consists of a collection of primitive modules that may be used, command dispatch tables for the description editors, design rules, simulation models and so on. The environments themselves are organized as a directed acyclic graph. At any time, the designer can refine the environment being used. For instance, one could begin a design in the *Basic Logic* environment and later when it had been decided to use CMOS, switch to the *Generic CMOS* environment. Finally, when a foundry had been chosen, the designer would select an evironment for the specific process to be used. While the environment was *Basic Logic*, the designer would be able to draw logic schematics and simulation, but would be unable to get any timing information (other than in gate delay units) or do any circuit design. After switching to *Generic CMOS*, transistor level circuits and sticks diagrams could be developed. When the process specific environment has been chosen, detailed masks could be designed and accurate timing information would be available.


Schematic of: RC Filter, Project: Analog Circuits, Center = (-30.0, -25.0) Scale 10.0

Component Values for Bode plot
Max Frequency (rad/sec): 1.000
INDUCTOR-1: 0.01
RESISTOR-2: 100
RESISTOR-1: 1000.0
CAPACITOR-1: 1.0e-6

$$\frac{1\ S}{S^2 C_1 R_1 I_1 + S I_1 + S R_2 C_1 R_1 + R_2 + R_1}$$

Software Tools

Though most of the time the data structures needed by a CAD designer are already in place, SCHEMA also includes a large library of compatible flavors (abstract data types) for constructing new structures. Within this library are mechanisms for dealing with many different types of hierarchy, prototypes, "creation on demand," timestamping, and so on. When creating a new data structure, the designer merely picks the flavors that provide the functionality desired and includes his own customizations. This fine grained modularity has helped maintain a high level of uniformity within the system. The modularity techniques used are based on the Capsule ideas[5].

In addition there is also a growing library of useful CAD oriented procedures that may be drawn upon. Among them are sparse matrix routines, linear and non-linear equation solvers, a moderate size symbolic algebra package, topological traversal routines, two dimensional spatial management packages and so on. The existence of these packages has enabled CAD builders to build on each others work more than in previous systems.

The totality of these tools, mechanisms and policies remove much of the drudgery from CAD tool development and encourages tool developers to proceed in a cooperative, cummulative fashion. For the electronic designer it provides a uniform environment, with uniform access to a wide variety of different synthesis and analysis tools.

A simple transient simulator was built on this base by Chris Terman. An example of its use is shown on the preceding page. The results of the simulation are left as annotations on the topology that the user (or another program) can examine. The top left window shows the circuit being simulated, the top right one shows a few selected waveforms. In the bottom window, the currents into the depletion transitors are given.

The second figure illustrates the use of the linear systems analysis tools. The bottom window gives the exact transfer function of the *RLC* circuit shown in the top left window. At the right, several Bode plots are given, and a pop up menu is shown which gives the parameters of last plot.

Conclusions

We have given a brief summary of the internal architecture of SCHEMA and shown a few of its uses. Its novel architecture and extremely high degree of integration make SCHEMA easir to use both by CAD tool designers and designers than many other systems.

The authors would like to acknowledge the assistance of the other members of the SCHEMA design group, Anuja Kohli, Siu-Ling Ku, Mike McDonald, Margaret St. Pierre and Steve Seda. This work was supported by DARPA contract N00014-80-C-0622 and Harris corp.

References

[1] J. Batali and A. Hartheimer, "The Design Procedure Language Manual." MIT Artificial Inteligence Laboratory Report 598. 1980.

[2] M. Ross, *A Datapath Generator*. B. S. Thesis, Dept. EECS, Massachusetts Institute of Technology. June 1982.

[3] G. J. Sussman, "SLICES: At the Boundary between Analysis and Synthesis." *Proc: Artificial Intelligence and Pattern Recognition in Computer Aided Design*, IFIP WG 5.2, Grenoble, France, 1978.

[4] D. L. Weinreb and D. A. Moon, *Lisp Machine Manual*. MIT Artficial Intelligence Laboratory, Cambridge, MA, 1981.

[5] R. Zippel, "Capsules," *SIGPLAN Bulletin*, vol. 18, no. 6, pp. 166–169, 1983.