Stanford Department of Computer Science
Report No. STAN-CS-80-826

October 1980

A DATABASE APPROACH TO COMMUNICATION IN VLSI DESIGN

by

Gio Wiederhold
Anne Beetem
Garrett Short

Research sponsored by

Defense Advanced Research Projects Agency

AO 3889
N00039-80-C-0132
(NESC)

DEPARTMENT OF COMPUTER SCIENCE
Stanford University

N00039-80-C-0132
ARPA Order-3889

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER STAN-CS-80-826	2. GOVT ACCESSION NO. AD-A096 347	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A DATABASE APPROACH TO COMMUNICATION IN VLSI DESIGN		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Gio Wiederhold, Anne Beetem, Garrett Short		8. CONTRACT OR GRANT NUMBER(s) N3980C0132
9. PERFORMING ORGANIZATION NAME AND ADDRESS Stanford University Department of Computer Science		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS AO 3889
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd., Arlington, VA 22209		12. REPORT DATE Oct 1980 13. NUMBER OF PAGES 14
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) NESC		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE. DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Database
Communication
Hierarchial Design
DBMS-20
Engineering Change Orders
Commercial databases
Macro Expansion

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(U) This paper describes recent and planned work at Stanford in applying database technology to the problems of VLSI design. In particular, it addresses the issue of communication within a design's different representations and hierarchical levels in a multiple designer environment. We demonstrate the heretofore questioned utility of using commercial database systems, at least while developing a versatile, flexible, and generally efficient model and its associated communication paths. Completed work and results from initial work using DEC DBMS-20 is presented, including macro expansion within the database, and signalling of changes to higher structural levels. Considerable discussion regarding overall philosophy for continued work is also included.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Ø94120

A Database Approach to Communication in VLSI Design

Gio Wiederhold, Anne Beetem, and Garrett Short
Stanford University
Dept. of Computer Science
October, 1980

Abstract

This paper describes recent and planned work at Stanford in applying database technology to the problems of VLSI design. In particular, it addresses the issue of communication within a design's different representations and hierarchical levels in a multiple designer environment. We demonstrate the heretofore questioned utility of using commercial database systems, at least while developing a versatile, flexible, and generally efficient model and its associated communication paths. Completed work and results from initial work using DEC DBMS-20 is presented, including macro expansion within the database, and signalling of changes to higher structural levels. Considerable discussion regarding overall philosophy for continued work is also included.

Keywords and Phrases: database, communication, hierarchical design, DBMS-20, engineering change orders, commercial databases, macro expansion

A Database Approach to Communication in VLSI Design

Gio Wiederhold, Anne Beetem, and Garrett Short
Stanford University
Dept. of Computer Science
October, 1980

I. Introduction

The design of a VLSI device involves the manipulation of a large volume of diverse interrelated data. For a given design task, such as simulation or layout, some data may be generated through either computation or through recording of interactively made human designer decisions. In current design methodologies, these design tasks operate on different representational levels of the same device, ranging from overall functional or logical descriptions [Hill80] down to the physics of individual switching cells. A single task may carry out verification within one level, may generate lower level units by expansion of more abstract definitions, or otherwise necessitate movement between different representational levels. In addition, if complex designs are to be produced in a tolerable time frame, then several design specialists must be allowed to work concurrently on distinct sections or at distinct levels. All this causes the management of the design data to become increasingly difficult since not only will data requirements overlap, but one design task will often be dependent on the data being manipulated in another area. This data communication effort probably contributes significantly to the increase in effort from 4 to 30 man-years for microprocessor designs, and makes single-designer oriented methodologies infeasible [Sch79]. The exponential growth of design cost while production costs diminish - as stated by Faggin and Moore in [Rob80] - is changing the outlook of the microprocessor industry, pointing to the need for more effective handling of design data.

Database technology has dealt with both the management of large volumes of data and the problems caused by concurrency of data access. Many existing database systems support inter-area communication functions; instances are found of systems dealing with inventories, where production and sales issues interface, or patient management, where individual care and global health care concerns come together. Databases are used as well to resolve the multidisciplinary design issues in aircraft design [Ful80] and to manage the problems of engineering changes in computing systems [Sie80]. Given this background, we foresee that databases may provide tools to serve the VLSI designers; however, the tool is complex and will not be effective unless well understood and adapted to the demands of this application.

We have identified several issues that have to be addressed if databases are to become effective tools in VLSI design. (The interested reader may wish to compare these to [Ley79] which has a somewhat different approach.)

1.) The database system must support a variety of design methodologies. The support of a top-down approach is most easily achieved, but processes at a lower level may contribute information to higher levels. Also, a designer may need to make changes at any level, which have to be reflected throughout the design levels. An example of the first exception to a pure top-down method is the handling of timing data than can be produced only after a detailed layout is available, while gates have been dimensioned using simple assumptions of intergate transit times. A second case arises when a particular gate of a series is changed to satisfy special interface demands or terminating conditions.

2.) Explicit storage for all attributes of all elements at lower levels must be avoided in order to permit effective management of change emanating at a higher level. Excessive replication of lower level elements vitiates the benefits of hierarchical design methods, and creates excessive storage demands. This implies that the database system must be able to fetch actually instantiated data or compute potential, non-instantiated data using stored algorithms.

3.) The database must be able to provide a convenient interface to a wide and changing variety of programs. This interface should not change as the database is developed and extended.

4.) The performance of the database system has to be such that the degradation of performance relative to an isolated design file operation is proportional to the benefits gained.

II. Approach

The approach we are using in our work includes the use of commercial database systems to assess the general suitability of database approaches to VLSI design data. One objective is to determine whether commercial database systems, used knowledgeably , can perform adequately, or if they do not, where the bottlenecks are. The current set of experiments uses a network system, DEC's DBMS-20. This is a system based on the published CODASYL database definition, which has a strong orientation to well-understood business applications, such as inventory management. The database designer can, through schema specification, select which of the logically appropriate linkages should be implemented [WEM80]. As a successor experiment in this area we plan to investigate use of a relational database system, RIMS, with strong automatic query optimization capabilities [Sim80].

After having demonstrated replacement of a custom design by an equivalent database representation, we explored implementation of novel data management facilities not provided by specialized design files. Specifically, as is necessary in order to operate in the mode for the VLSI design environment, the database system has to be augmented with communication paths between levels. These paths may ultimately be oriented in any combination of directions. We are investigating the constituent directions, viz. down hierarchical level, up to higher level units, and sideways.

In the downward direction, a method must exist to create the effect of lower level instantiation using procedures and higher level descriptions. It is desirable that the query interfaces which access the lower levels do not have to distinguish between actual or computed data elements. Along these same lines, we intend to look at the issues and possibilities of when and how we wish to store redundant or computed data.

Communication in the upward direction relates detail to more abstract specifications. The creation or modification of lower level instances has to be bound to the appropriate higher level elements. An initial approach we consider promising and have implemented is signalling such changes to the next levels up in the hierarchy. Multiple structures may become involved because an element may be defined by an association of several higher level entities [Wie77]: a simple example is an element that is defined from the expansion of a functional component and a library description. The signal creates an exception flag at the higher structures. At a later time, when the level to which the signal was directed is acc. ~d by its owner, the system can provide a warning. An appropriate action could then be taken; for example, verification of continued correctness of the design at that level, or the introduction of a new version of : component, or a new parameterization of the library descriptions. With experience, selected types of changes could trigger automatic updates.

While the need and techniques for up and down passing of information are relatively clear, travel in the orthogonal direction is not nearly so, yet may be essential in the future design system. To explain the need consider the following scenario. Suppose we are designing a microcomputer chip with an ALU composed primarily of registers , dense RAM and ROM, a finite state PLA controller, some random logic, etc. Designing this chip with only one methodology would be a terrific waste of time and energy as, although there are design tools which can fairly efficiently model, for example, random logic, it would very inefficient to design a regular structure like a PLA using random logic techniques. It would be far more effective to allow a sub-module to be designed or simulated in its most appropriate manner. Allowing for generalized sideways communication could then greatly increase the overall flexibility and efficiency of a design system. Current working design systems give no consideration to this sideways communication issue, but the research has begun (see [Be80] for example) . We hope to learn from it and incorporate these new ideas into our database system.

III. Current Work and Results

Before we to discuss what actual work we have done, it must be noted that due to availability of data, programs, and otherwise existing material at hand, we are using data from conventional circuit board design to model the VLSI design process. Even so, many of the problems are similar and as such, we believe that the results are validly applicable to the VLSI design process as well. More importantly, by using this data, we were able to make benchmark comparisons with existing specialized design files and programs on the same data. This is essential for measuring the relative performance of commercial database systems. We feel that we have gained insight into the problems involved in a database oriented design system, and in particular, have demonstrated the viability and efficiency of using commercial database systems at least until the complete model of the design process and its communication requirements is well understood. So, without further ado let us describe exactly what we have done so far.

Reading from a Database

First, an evalution was made of DEC's DBMS-20 performance on data retrieval times. Specialized design files of circuit information were used as the control to the experiment. The user begins with his design written in SDL (Structural Design Language [vC79]), a straightforward hierarchical description language. The SDL description is input to the SDL compiler, which then produces a dump file containing the logical description of each component described in the design. This design methodology is strictly hierarchical: the logical description of one component is described in terms of smaller lower level components. For example, gates are described in terms of transistors, flip-flops are described in term of gates, registers are described in terms of flip-flops, etc. Each description level is given an unique name in the design file. Figure 1 shows the hierarchical structure of the PDP-11 processor, as described in SDL [SL79]. The dump file from the compiler is loaded into the SPRINT database [Ste79], the specialized control subject, via a 'hardwired' schema.

Level	*Components*
cpu	PDP-11
reg	BLEG, PROCOUNT, BUSCOUNT, RAM16X16, ALU, STATUS, AMUX
rf1	PRIOARB, TIMER
rf2	TCFF2, RAM16, 40181, 40182
ff	TCFF, LFF16, DFF4, BUSD16, MUX16, RAM
gate	DFF, LFF, XOR, BUSD, MUX
trans	TRANSP, INV, NAND, 3NAND, 4NAND, NOR, 3NOR, 4NOR
bottom	TN, TP, PLA-SMALL, PLA-MED, PLA-LARGE

Figure 1

Initially the CODASYL schema was modeled very closely to the SPRINT schema, and due to SPRINT's hierarchical structure, CODASYL network capabilities were not utilized in the first iteration. Later the schema was modified to take advantage of network structures: this effort provided linked access to the library file of components. A loader program was then written to load the same dump file produced by the SDL compiler into the DBMS-20 database. Comparison of loading times showed the SPRINT database to be quicker, as expected, but DBMS times were quite acceptable. The initial loading of the PDP-11 example took SPRINT about one minute and required about four minutes for DBMS-20.

For the actual test of the retrieval times, a macroexpander program [Pay80] was used. This program reads the logical description of a high level component from the database. The user then specifies the levels to be expanded. Thus, if the user specified the program to expand all levels down to the transistor, then the resulting output will be the logical description of the original component described totally in terms of transistors. This form of description may not be very useful to the designer, but could be the input to, for example, a simulator program. Also, in the VLSI environment, this could be the first step in producing the layout diagram.

The macroexpander program needs to do a tremendous amount of random read operations, especially if the component that is being expanded is large and described at a high level. The following are the results of expanding the ALU and the PDP-11:

		SPRINT	*DBMS-20*	*records read*	*words read*
ALU	IO time	21 s	33 s	1514	7754
	CPU time	30	45		
PDP-11	IO time	66	115	5925	28501
	CPU time	120	190		

These results show less than a factor of two degradation in performance of the DBMS-20. This is an acceptable trade-off for the increased flexibility and generality of CODASYL, and disputes the original theories that there would be at least an order of magnitude difference in performance.

Writing into the Database

We next evaluated DBMS-20 performance in writing back into the database. In order to simulate a real application of VLSI design, we considered how the database would handle instantiations. Seeing how this might be done requires a closer look at the schema. Figure 2 shows a subset of the network diagram of the schema. This is a version of the SPRINT schema, with a few changes to more closely model VLSI design. For each component described in the database there is a Logical Description record containing its *name*. This field is used as the key to find the record directly via a hashing function. Records are connected to related records through three distinct rings of pointers, forming a complex network. One of the rings off of the Logical Description record describes each external pin, with a unique name and its function (i.e. input, output, tristate). The equivalent-group ring describes the equivalence between pins and sets of pins. The remaining ring off of the logical description record supplies the internal description of the part. For each Internal Description record, a General Information record is kept describing general characteristics. The attribute fields, *creator*, a *time* stamp, *level*, *purpose*, and *version* number are meant to fully

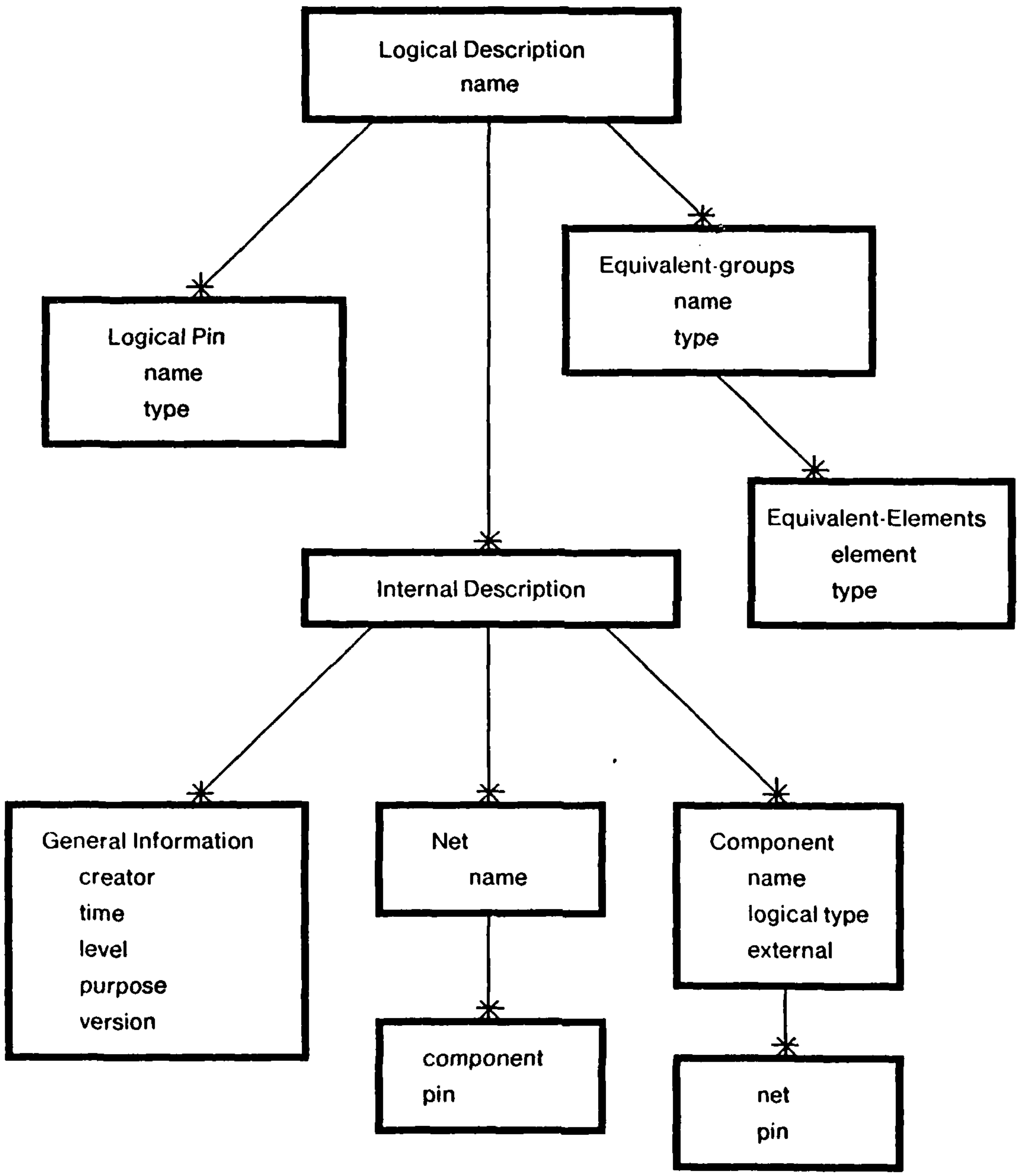

Logical Description
name
Logical Pin
name
type
Equivalent-groups
name
type
Equivalent-Elements
element
type
Internal Description
General Information
creator
time
level
purpose
version
Net
name
component
pin
Component
name
logical type
external
net
pin

Figure 2

describe the corresponding internal description.

In order to exercise the representation, the macroexpander program was modified so that it could write back into the database. When working in this mode, after the program expands a component, it stores the new representation as another internal description along with a general information record with the appropriate data. In the design atmosphere, suppose one is working on a component at another level or perhaps at the same level. The user makes some changes from the original description and stores his version into the database. This may be done to several or all the components. Now when a higher level component is expanded, the program can selectively choose the appropriate internal description to use. The above scenario was tested on the ALU with following results:

The upper level description of the ALU contains the following pieces and levels:

40181	rf2
40181	rf2
MUX	gate
XOR	gate
INV	trans
NAND	trans
4NAND	trans
NOR	trans
4NOR	trans

The following is some of the data on expanding the pieces of the ALU:

40181: Total read time = 17.4 seconds.

record name	quantity
net	152
component	127
comp-pin	564
Total read	843 (4432 words)

Total write time = 41.3 seconds.

record name	quantity
net	154
net-pin	898
component	293
comp-pin	898
Total written	2243 (10136 words)

40182: Total read time = 8.78 seconds

record name	quantity
net	79
component	67
comp-pin	268
Total read	414 (2240 words)

Total write time = 8.82 seconds

record name	quantity
net	38
net-pin	186
component	59
comp-pin	186
Total written	469 (2128 words)

The mux and the XOR pieces were also expanded. Then the ALU was expanded again with the expanded version of the lower level pieces in the database.

Total read time = 48.0 seconds.

record name	quantity
net	341
component	441
comp-pin	1575
Total read	2357 (12745 words)

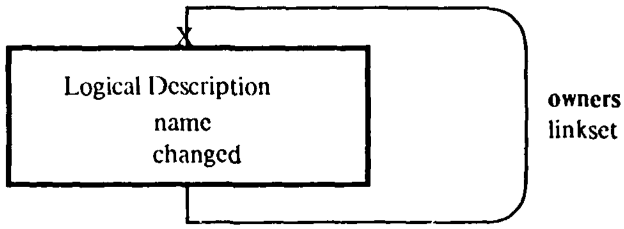
Total write time = 203 seconds.

record name	quantity
net	721
net-pin	4384
component	1441
comp-pin	4384
Total written	10930 (49364 words)

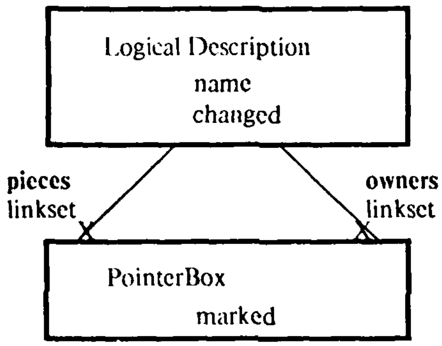
The current state of the SPRINT design system does not allow writing instantiations back into the database, so no parallel experiments were done, but again the DBMS-20 implementation showed an acceptable performance.

Signalling Changes in the Database

The next and most recent test on DEC DBMS-20 was to flag the neccessary upper level pieces regarding changes made to lower level components. A straightforward approach to this problem would be to have each component keep track of the upper level parts that use that component. To implement this in CODASYL, we would like to have each Logical Description record own other Logical Description records in a owners set, as such:



Besides causing cyclic errors, this structure is not allowed in the CODASYL definition. The technique used was to add another record off the logical description to indirectly find the owners, illustrated below.



The following example illustrates how upper level pieces are warned of lower level changes. Suppose the multiplexer (MUX) has been changed and we wish to flag all the components that use the multiplexer. In the database there are several parts that use the multiplexer, and for each one of these there is a PointerBox record in the multiplexer's **owners** linkset. Some of these include the ALU, status register, and a 16-input multiplexer (MUX16), see figure 1. These same PointerBox records are in the **pieces** linkset of the parts that use the multiplexer. The multiplexer itself is made up of inverters and transistor pairs (TRANSP). Thus there are two PointerBox records in the **pieces** linkset of the multiplexer. The flagging process begins by marking the first record of the **owners** linkset, in this case say it is the ALU. Here the field *marked* of the PointerBox is incremented. Next, the owner record of the **pieces** linkset of the current PointerBox is found. Each PointerBox record has a pointer directly to the owner record of both the **pieces** linkset and the **owners** linkset. At this time, the logical description record of the ALU is located and the field *change* is incremented. Flagging continues from the ALU in a recursive manner. When a logical description record with a empty **owners** linkset is found, the upper most level has been reached. The flagging must then move back to the previous level by finding the next PointerBox record of the **owners** linkset.

This method of flagging upper level components is best characterized as "height-first". This method requires more record accesses, since many of the same records are retrieved many times, than a possible level or breath-first flagging, but the height-first method allows more information to be stored.

Consider the following example: Suppose the multiplexer and the inverter are changed and their upper level pieces are notified. As shown in figure 1, the multiplexer and the inverter are described on different levels of the hierarchy, and a change to the inverter will also affect the multiplexer. If you now query the PDP-11 data record as to whether any changes have been flagged, only the inverter will show up as an altered component. If the upper level pieces of the inverter are unflagged (decrement the *change* and *marked* fields), and the PDP-11 data record is queried again, the multiplexer will be identified as the modified component since correcting for the inverter does not resolve the othere multiplexer modification. Now, suppose the multiplexer and the NAND gate, which is described on the same level as the inverter, are changed. A subsequent query from the PDP-11 record will reveal that both the NAND gate and the multiplexer have been changed. The multiplexer is caught in this situation because none of its lower level pieces were changed. Thus this flagging algorithm allows easy detection of all the lowest level independent modifications. Also, every component can be unflagged uniquely by reversing the flagging procedure. The time to carry out this flagging algorithm depends on the level at which the process is started. Here are some sample flagging times:

Piece	Level	Time (seconds)
ALU	reg	.13
MUX, XOR	gate	.9
INV	trans	7.1
TN	bottom	21.

IV. Conclusion

With the successful implementation of flagging, we have taken time out to write this paper and so we shall summarize our findings and predict our next moves. Looking back over our work, we believe our initially stated approach to be most promising: the commercial database system allowed us to implement both initial designs and modifications relatively quickly and easily. Coupled with the discovery that performance is not badly affected, this has allowed us to experiment productively. This ability to experiment is important to obtain the knowledge needed for designing useful database systems for VLSI design. As we, and designers themselves, gain more insight into the design process, we hope to develop a database supporting all the types of communication paths for many different kinds of interlocking design tasks. The final long range goal is, of course, a usable, flexible, expandable, and efficient database oriented VLSI design system.

Further experiments are in progress to complete the tasks outlined in Section II. The immediate task is to manage queries that access partially instantiated and partially computable elements. Work by others is in progress on the relational database approach and a comparison will provide further insights and directions for future work.

References

[Be80] Beetem, John: "Structured Design Using Isomorphic and Non-Isomorphic Multiple Representations", Stanford University, Electrical Engineering Department, CIS (Center for Integrated Systems), August, 1980.

[Ful80] Fulton, Robert E., "National Meeting to Review IPAD Status and Goals", *Astronautics and Aeronautics,* July/August 1980.

[Hill80] Hill, Dwight: *ADLIB User's Manual,* Stanford CSL Technical Report No. 177, August 1979.

[Ley79] Leyking, L.W.: "Database Considerations for VLSI", *Proceedings of the Caltech Conference on Very Large Scale Integration,* pp. 275-301, January 1979.

[Pay80] Payne, Thomas: Pascal macroexpander program, Stanford University, Electrical Engineering Department, CSL (Computer Systems Laboratory), CIS, 1980.

[Rob80] Robinson, Arthur L.: "Giant Corporations from Tiny Chips Grow", *Science,*vol. 208, no. 4443, pp. 480-484, May 2, 1980.

[Sch79] Scheffer, Lou: "Database Considerations for VLSI Design", *Design Automation at Stanford, ed. W. vanCleemput,* Stanford University, CSL, CIS, 1979.

[Mal80] Mallmann, Felix P., "The Management of Engineering Changes Using the Primus System", *The Seventeenth Design Automation Conference Proceedings,* pp. 348-366, June 1980.

[Sim80] Simpson, J. et al: *User's Manual for the Rims/FPDB System*; Science Applications Inc. Palo Alto, CA., March 1980.

[SL79] Slutz, Eric: SDL description of DEC PDP-11, Stanford University, CSL, CIS, 1979.

[Ste79] Stevens, K., vanCleemput, W.: *Design File Organization in the SPRINT System,* Stanford CSL Technical Report No. 133, 1979.

[vC79] vanCleemput, W.: *SDL: A Structural Design Language for Computer Aided Design of Digital Systems,* Stanford CSL Technical Report No. 136, 1979.

[WEM79] Wiederhold, Gio and El-Masri, Ramez: "The Structural Model for Database Design"; *Proceedings of the International Conference on Entity-Relationships Approach to Systems Analysis and Design,* pp. 247-267, December 1979.

[Wie77] Wiederhold, Gio : *Database Design,* McGraw-Hill, 1977.