A DATABASE APPROACH FOR MANAGING VLSI DESIGN DATA

by

Randy H. Katz

Computer Sciences Technical Report #457

November 1981

A Database Approach for Managing VLSI Design Data

Randy H. Katz
Computer Sciences Department
1210 West Dayton St.
University of Wisconsin-Madison
Madison, WI 53706
(608) 262-0664

ABSTRACT: We describe an approach to managing information about VLSI designs, founded upon database system methods. A database component provides a low-level relational interface to stored data. Built on top is a design data management system, supporting the hierarchical construction of a design from primitive cells, and organizing data about alternative design representations and versions. Programs to provide a tailored interface to design data are also provided. The system simplifies the rapid construction of new design tools by taking responsibility for design data management.

1. Introduction

The simplified design method of Mead and Conway [MEAD80] has spawned a new culture of custom chip design. Researchers have successfully designed a variety of large systems in silicon, (e.g., [PATT81, STEE80, CLAR80]). There appear to be many creative uses for the "million transistors on a chip" that should be available in a few years [PATT80].

However, many experts agree there will be a limit of growth in VLSI circuit complexity because of the lack of tools to adequately support the design process. The design automation community has responded with sophisticated tools to aid in circuit layout (e.g., [OUST81, KELL81]) and verification (e.g.,[BAKE80, BRYA81, TERM81]). The important issue not adequately addressed by these tools is the management of the VLSI design data itself.

Computer-aided design tools for VLSI must efficiently manage large volumes of information that describe a design. This includes the descriptions of the different representations of a design (e.g., electrical, logical, behavioral, functional, and topological), and the various versions of a design as it evolves. The data management component is crucial to the success of a CAD system, yet few systems provide sophisticated data management facilities [LOSL80].

The current state-of-the-art is to store design data in a traditional file system. These typically do not provide a wide range of data management facilities. Database systems, on the other hand, support such features critical to effective design data management as: dynamically changeable file structures, a variety of data access mechanisms, control mechanisms for secure and concurrent access, and recoverable storage of data.

In this paper, we outline an alternative approach for managing circuit design data. Our system is founded on database management facilities, not those directly supported by file systems. While the idea of employing database techniques for computer-aided design is not new [CIAMP76a, CIAMP76b, FOST75, HAYN81, HOSK79, KORE75, KAWA78, LORI81, MITS80, NIEN79, ROBE81, SUCH79, VALL75, WILM79, WONG76a, WORK74, ZINT81], these systems are tailored to specific design domains, with "wired" database structures that make it difficult for them to support a wide range of design tools and methodologies. The system described here explicitly supports hierarchical design without a prespecified selection of design representations, is founded on current database techniques, and concentrates on providing an interface to design data upon which new design tools can easily be constructed.

This paper is organized as follows. In the next section, we give a summary of the features desired in a system for managing design data. In Section 3, we argue that existing database systems, while supporting many of these features, fall short in supporting several critical ones. In Section 4, we briefly review the previous efforts to apply database techniques to design data management, and contrast them with our approach. Section 5 describes the architecture of our system for design data management, comprised of a database component, a design manager, and a collection of design interpreters. We indicate how many of the techniques developed for database management can be applied to the new problems of design management. We close the paper with conclusions and the current status of our system.

2. Feature Summary for Design Data Management*

Hierarchical "levels of abstraction" are important for tackling any complex problem, especially so for describing a VLSI design. [MEAD80] describes a hierarchical design method for VLSI systems design, wherein a complete system is first

*This list has been compiled from a variety of sources, primarily [EAST80, EAST81, LORI81, ROSE80]. We only include those features we deem to be truly desirable for design data management.

conceptualized, and then decomposed recursively into constituent, more primitive parts. The leaves of the hierarchy represent the detailed specifications of the primitive cells that constitute the design. Internal nodes describe how the more primitive components are composed and interconnected. In a series of papers about CAD tools for VLSI, [ALLE81, DIRE81, DUTT81, NEWT81, TRIM81], a common theme emerges that the design tools must reflect this hierarchical structure. A design data management system is no exception.

The hierarchical approach is generally agreed upon; the details of design represention are not. A design may be described by many alternative representations, including: layout geometries, stick diagrams, block diagrams, logic diagrams, transistor networks, functional specifications, or behavioral specifications. Rather than impose an a priori assumption on the kinds of design representations supported, we believe that a design data management system should provide support for organizing multiple representations of a design, without it needing to understand their detailed structure. Higher level software is responsible for choosing the form of the representation and for its interpretation.

A system for managing designs does not itself interpret the representations of a design, but must be responsible for keeping track of the multiple representations of a design cell. It should tag the alternative representations that require update after a design change to a particular representation. Note that the design system itself cannot actually propagate design changes to alternative representations without an understanding of the semantics of the representation.

A design data management system keeps track of the multiple versions of a design as well. Facilities for declaring a design as "released", i.e., no longer subject to update, must be provided. Also needed is the ability to archive old versions and later restore them.

A design data management system not only propagates the "need to update" to alternative representations of a modified design, but also provides mechanisms to aid in propagating changes up the design hierarchy. A component of a design may require update because of a change in one of its subcomponents. The method by which this update is performed should be left to higher level software.

A completely instantiated VLSI design is likely to stress the capabilities of most modern storage systems. The design system must provide facilities for procedurally deriving parts of the design from stored data. A design is actually a combination of stored data and procedurally defined data.

Finally, the system must serve as a convenient base upon which to develop new design tools for VLSI. Therefore, the interface to these tools should be flexible and easily tailored for the particular design data representations manipulated by these tools. Existing tools should interface to the system without great difficulty, by making the system appear identical to a file system.

3. Suitability of Existing Database Systems for Design Data Management

Relational database technology, as developed in such prototype systems as INGRES [STON76a] and System-R [ASTR76], provides many desirable features for the management of design data. Further, relational data handling techniques have not been available to builders of design systems until recently. However, even relational database systems are not perfectly matched to the problem of managing design data (e.g., see [HAYN81, LORI81]). Extensions are necessary, and we describe them here.

The conceptual contribution of the relational approach is the clean separation of data's physical organization from the flat file view presented to the user. While the user sees his data organized as a collection of tables, in actuality, sophisticated techniques for organizing data on secondary storage are employed. For example, extendible hashing [FAGI79] and B-trees [COME79] can be used to provide associative access to dynamically changing files. These are new access mechanisms that are not to be found in many older database systems.

Relational systems provide mechanisms for the reliable storage of data, even in the face of system crashes [GRAY78]. Mechanisms to support data sharing, while isolating a user from the concurrent updates of other users, are also provided.

Database systems support the notion of a transaction: a collection of database operations that are either executed completely or not at all [GRAY78]. Finally, sophisticated access control [GRIF76, STON76b] and integrity control [ESWA75, STON75] mechanisms are supported by these systems.

However, current systems do not conveniently support all the facilities needed for design data management. First, database systems have been designed for the management of regular, formatted data. It is not easy to represent unstructured data, such as text or graphical images, which are often used to describe VLSI designs. Second, relational systems do not provide operations to manipulate hierarchically structured data. Note, however, that systems that support the hierarchical data model do not provide the desirable features of relational systems, such as independence from the physical data organization. Third, few systems provide physical storage structures that exploit the hierarchical nature of design data. For example, in INGRES, it is awkward to cluster (i.e., group together) the child records of a particular parent because there is no storage structure that spans relations. On the other hand, System-R supports the notion of a "clustering link" to connect interrelated records from different tables. Fourth, database systems do not explicitly support multiple representations or versions. Finally, the kinds of transactions encountered in the design environment differ considerably from those currently supported by database systems. The latter are short in duration and typically involve little data. In a design environment, a designer "checks-out" a piece of a design, modifies it extensively over a long period of time, and finally returns it to the database. These kinds of transactions require large volumes of data and persist for long periods of time ("conversational transactions" [LORI81]).

4. Previous Approaches to Design Data Management

Design automation encompasses many diverse fields of design activity. What seems to separate VLSI from the others is: 1) the large volume of complex data involved, and 2) the need for supporting an ever changing collection of representations for a design. Tools developed for previous technologies may not be applicable to VLSI design.

A major weakness of earlier systems is their inflexible choice of the set of supported representations. While this may be acceptable for well understood design domains, the choice of what constitutes the correct set of representations for VLSI has not yet been determined.

For example, [TAKA80] chooses to represent a design as a network of devices. [ZINT81] describes a design in terms of a logic diagram in addition to the network. [SUCH79] describes a design as an interconnection of parts from a standard catalog of discrete transistors, SSI, MSI, and LSI parts. [CIAM76a, CIAM76b] represent a design as devices, pins, interconnections, and signals. [KORE75] describes a design in terms of parts, nets, wiring, and layout artwork. Logical, physical, and electrical representations are part of the database described in [WORK74]. None of these systems make it easy to create new design representations.

In addition, none of the systems mentioned above are based on state-of-the-art database technology. All have been built on network database systems. These provide primitive access and concurrency control features, and do not adequately isolate the design management software from the physical data organization.

Some systems have been based on relational techniques [WILM79, VALL75]. However, these systems do not address the issues of organizing a design hierarchy or supporting multiple design representations.

[ROBE81] describes a system that most closely resembles the one we describe here. It supports multiple design representations constructed from a set of basic data organizations (symbol tables, relational tables, structured data tables, unstructured data). Interfaces are provided for a variety of tools. However, it fails to support a design hierarchy, and provides no mechanisms for maintaining consistency across representations. Further, it is built on top of a CODASYL database system, implying that its database schema is static, and thus new design representations cannot be added without a major reorganization of the database.

The GLIDE System [EAST80] is another start-of-the-art design data management system. Its features include dynamic schemas, automatic invocation of procedures for integrity enforcement, and record types and operations for geometric modelling. Further, the system is accessed through its own sophisticated programming language, rather than via a procedure call interface. However, because the system is oriented towards general purpose geometric modelling, it does not support special features important for VLSI design. In particular, the design hierarchy is not explicitly supported, and thus cannot be used to aid in design change propagation or to control concurrent access. Neither is a flexible choice of design representations, although there is support for design alternatives. Finally, the choice of interface, i.e., through a special programming language, may make it difficult to interface existing tools to the design system.

The utility of database techniques for design data management is not universally accepted by the design automation community. [SIDL80] claims that commercial database systems are hard to use and suffer from poor performance. While true for older systems, this is not as much of an issue for relational systems. [BAND75] claims that certain useful operations, such as tree traversal, are difficult to preform in relational systems. This is more an artifact of high-level relational queries languages than the data organization. A system with a low-level interface could support the operation. [TRIM81] claims that extensions to a database are disruptive and costly, and that interfacing to a database requires too much work. The first claim is not as true for relational systems, which support schemas and structures that can change dynamically. The latter issue is addressed by the remainder of this paper: the system described below is designed to make it easy to interface design tools to a database.

No previous system supports the complete range of facilities needed to support VLSI design. What sets ours apart is its explicit representation of a design hierarchy (crucial for conquering the complexity of VLSI design), its support for a flexible choice of design representations, and its multilevel architecture, which combines the advantages of a relational interface to stored data, with a customized, domain specific interface to design data.

5. A Multilevel Architecture for Design Data Management

5.1. System Overview

The deficiencies of current systems for managing design data has led us to develop a multilevel architecture for design data management (see Figure 5.1). The system's core is the database component. Its responsibility is to provide a reliable, efficient interface to stored data, and to isolate the higher layers from considerations of reliability, physical storage structure, data sharing, access control, and (low-level) integrity control. It supports unstructured data and record clustering, while providing a simple and flexible interface upon which the rest of the system is built.

A design management component is built on top of the database component. It is responsible for managing the design

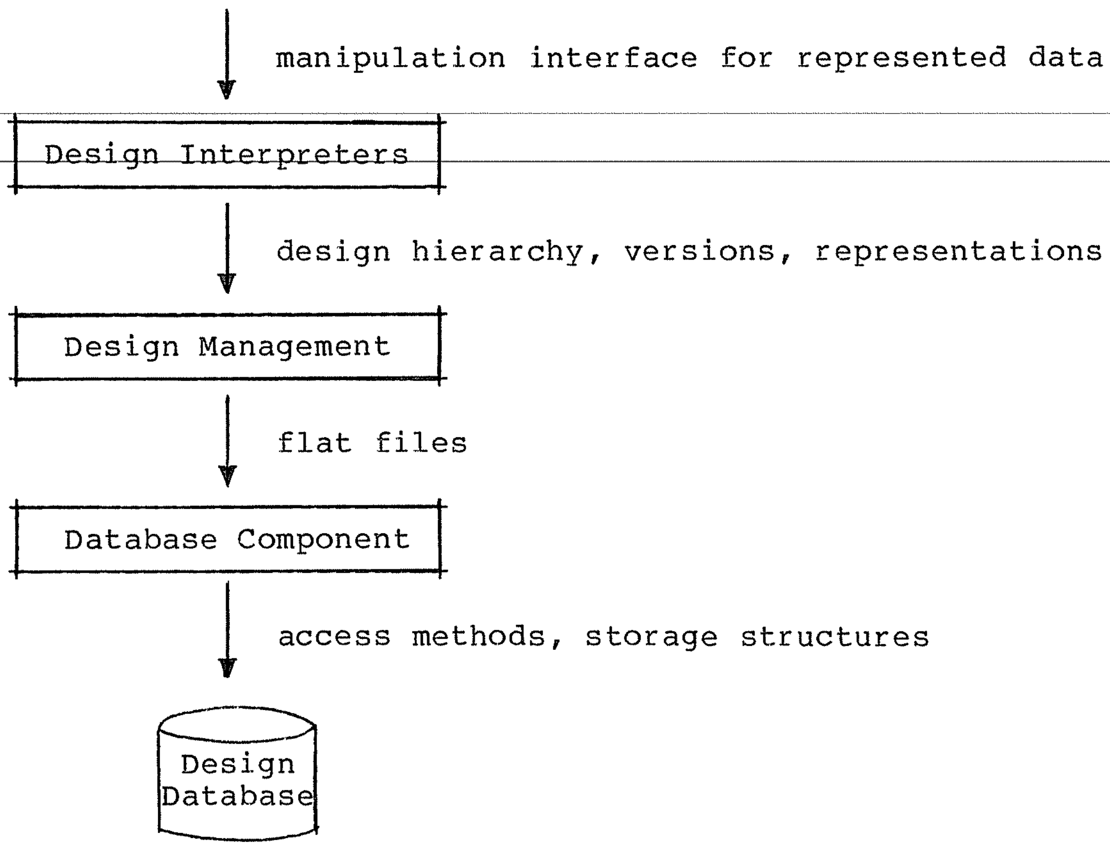
manipulation interface for represented data
Design Interpreters
design hierarchy, versions, representations
Design Management
flat files
Database Component
access methods, storage structures
Design Database

Figure 5.1 - Three Level Architecture

hierarchy, and organizing the alternative design representations and multiple versions within a database supported by the database component. It provides an interface for extracting and replacing subcomponents of a design from the design hierarchy.

The final layer consists of programs that interpret the data stored about a design. Design interpreters provide an interface to design tools that is appropriate for manipulating the data at hand, e.g., layout geometries, transistors, etc.

In the remainder of this section, we describe each of the levels in more detail.

5.2. Database Component

The database component is a low-level data management system that supports a relational style data interface. Although the data appears to be organized as simple tables, it actually has a complicated physical organization to facilitate efficient data access. The details of the physical organization are hidden from higher levels of the system.

Methods for implementing a database component are now well understood. The RSS (Relational Storage System) of System-R is illustrative of the general architecture [ASTR76]. While the RSS is not a full function relational system, it does provide operators for data recovery, transaction management, and data definition. The interface supports record-at-a-time access via sequential scan, indexed scan, or interrelational links.

The database component is composed of the subsystems of a database management system responsible for supporting access to physical storage structures, including buffer management and physical I/O (access methods), atomic actions and isolation from concurrent users (transaction manager), and reliable storage and recovery from crashes (recovery manager). [GRAY78, GRAY81a, GRAY81b] provide extensive discussions of the implementation aspects of these subsystems.

A database component for design data management must be easy to extend with new access methods. Indexed and hashed associative storage structures are supported, but new methods for handling unstructured data, such as text and other data peculiar to the design environment, are also needed. For example, we are adapting techniques proposed for handling textual databases in the office automation environment [ELLI80, TSIC80].

5.3. Design Manager

The database component supports databases organized as a collection of tables (or files), and provides a set of record-at-a-time operations for their manipulation. The design manager is responsible for mapping the design hierarchy, including alternative representations and multiple design versions, onto these structures.

In [MEAD80], a design hierarchy consists of primitive leaf cells and composition cells (internal nodes). A sample design hierarchy is shown in Figure 5.2. We store the hierarchy in a

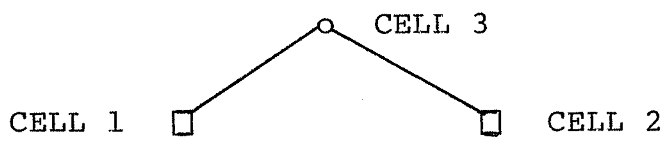

Figure 5.2 -- Simple Design Hierarchy
□ leaf cells, o composition cells

database as follows. Each cell is uniquely identified by a system assigned cell-id. Information about cells, such as its designer's name and the last time it was updated, are stored in a cell table. Associated with each primitive cell is a collection of representations for the cell, e.g., layout geometries, transistor networks, etc. Information about a particular kind of cell representation is stored in a representation table. There is one representation table for each representation type. A row is an instance of a represented object in a named cell. All rows describing the same cell should be grouped together (e.g., clustered on cell-id). A simple extension allows a collection of tables (e.g., boxes, wires, polygons) to collectively represent a particular type (e.g., geometries). This is similar to the database concept of generalization [SMIT77], wherein a geometry "is-a" box, wire, or polygon.

The design hierarchy is stored in the composition table. This table describes a composition cell as placements and orientations of more primitive composition cells and leaf cells. For simplicity, we assume a common hierarchy for all the alternative representations of a design. It is a simple extension to support separate hierarchies for each type of representation. In addition, it is possible to represent a design hierarchy as a rooted directed acyclic graph instead of a tree. Figure 5.3 shows the database representation of the design hierarchy of Figure 5.2.

The scheme described above supports a flexible representation of the design. The design manager knows about three kinds of

Cell Table

Cell-id	Cell-name	Designer	Date
1	CELL 1	R. Katz	10/6
2	CELL 2	R. Katz	10/7
3	CELL 3	R. Katz	10/9

Composition Table

Composition-cell	Primitive-cell	Placement
3	1	---
3	2	---

Representation Table: Boxes

Cell-id	layer	minx	miny	maxx	maxy
1	metal	0	0	0	40
---	---	----	----	----	----
1	poly	10	10	20	20
2	diff	5	15	20	35
---	---	----	----	----	----
2	implant	-10	-10	5	25
REQUIRED	—Determined by Design Interpreters—				

Figure 5.3 -- Database Representation of Design Hierarchy

tables, but the detailed structure of these is determined by higher levels. Certain columns are required by the design manager, but additional columns can be appended for use by the design interpreters and other software.

A version is a consistent state of a design that can no longer be updated. The concept is similar to that of a database snapshot [ADIB80]. In this case, the snapshot consists of the entire database, including the database catalogs that describe its structure. The latter information is needed whenever it is necessary to restore a design to a previous version, even after the database schema has changed (e.g., the representation details for specific types have changed). Methods for implementing database snapshots are described in [ADIB80]. Some of the implementation techniques are similar to those for differential files [SEVE76].

The database manager provides two basic operations for manipulating a design hierarchy: an extract operator for "checking-out" a subdesign from the design repository (identified by the root of a subtree of the design hierarchy) and a replace operator to modify the contents of a subdesign. The design manager is a librarian, keeping track of which parts of the design are currently being editted by designers. In effect, we treat the design database as a hierarchically organized filing cabinet.

The size and complexity of a VLSI design make it important to support multiple designers simultaneously accessing design data. We exploit the hierarchical structure of a design to control concurrent access to design parts. Parallel subtrees indicate independence among the design elements within them. Multiple designers can simultaneously access subdesigns as long as each is in an independent subtree, i.e., is not contained within any checked-out subdesign, and does not contain a subdesign checked-out to another designer.

Hierarchical locking protocols [GRAY78] can be used to support concurrent access to parallel subtrees. A designer holds an "intentions lock" on a subtree if he wishes to lock one of its subtrees for design activity. Many designers can simultaneously hold intentions locks on the same composition cell, but only one can have "exclusive" access for update. Intentions locks are requested for all the nodes on the path from the root of the design hierarchy to the root of the subdesign. An exclusive lock is requested on the latter node. It will not be granted if the node is already held with an exclusive lock (a designer already has it checked-out) or an intentions lock (a subdesign of this one has been checked-out). Although the above discussion has focused on a tree-structured design hierarchy, the protocol can be used with design hierarchies that form a directed acyclic graph.

Extracting a subdesign consists of setting the locks as discussed. The database component provides the hierarchical locking mechanisms, and tuple-level locking. Extraction does not imply that data has been read from the database. It simply insures that the data to be updated by one designer cannot be simultaneously accessed by another.

The replace operation causes a new subtree to replace one that has already been extracted. The act of modifying a subtree may invalidate higher levels of the design hierarchy. Rather than impose some a priori policy, we provide a mechanism for implementing a variety of policies. For every node along the path back to the root of the design, a consistency checking program, provided by the design interpreters or other higher level software, is automatically invoked whenever a subtree is replaced. A sophisticated checker can be written with full knowledge of the structure of the representation. For example, if there is a bounding box associated with each cell, the checker can insure that the new subtree does not exceed its allocated area. If it does, then the update can be aborted, or more likely, the checker can warn the user (e.g., a graphical circuit editor or some other design tool) of the potential problem. The design manager, without detailed knowledge of the representations of the design, can at best indicate those cells that may require update (i.e., the composition cells on the path from the root of the subtree to the root of the design hierarchy).

The above observation applies to the propagation of the "need to update" to other representations of an updated primitive cell. The formulation of a consistency checker across representations is more difficult because the checker must understand the equivalence across representations. The problem of mapping between different database representations of the same semantic objects has already been studied (e.g., [WONG79]). These techniques may be applicable here, but require complete identification of equivalent objects across representations. At least, the design manager can indicate those rows of representation tables that might require updating.

An alternative is possible if the design method supports a hierarchy of representations, with one representation algorithmically derived from another. For example, a sticks representation can be compiled into a two dimensional layout. The design manager automatically invokes the compiler whenever a change has been in the more abstract representation. This is similar to the Make facility of UNIX [FELD79].

The design data management system also provides operations for version control. A design is created by describing its initial structure to the design manager, who instructs the database component to create the necessary relations. A version is an unupdatable snapshot of a design. Version creation must be synchronized with update activity in the design hierarchy. Versions can be archived, and later restored.

The above discussions assume that the process of design is an orderly progression from version to version. Alternative designs are not supported. These could arise from the simultaneous update of a subdesign by more than one designer, each leading to a new alternative. Design alternatives can be represented as changes to a read-only version of a design. An alternative becomes the next version by merging its accumulated change file into the original. Design alternatives are similar to the notion of hypothetical databases [STON80, STON81], and can be implemented by differential files [SEVE76] or views [STON75].

5.4. Design Interpreters

The design interpreters choose a design representation (structure of the representation tables), and provide an interface to design tools for manipulating objects of the representation. In addition, they provide a consistency checker for use by the design manager.

To illustrate the structure of a design interpreter, we describe an interpreter for simple layout data. A layout is a collection of boxes assigned to particular layers. A "boxes table" represents boxes by the cell to which they are assigned, the layer, and the minimum and maximum x and y coordinates for the placement of the box within a cell. The latter can be translated by the placement and orientation information found in the composition table. In addition, each cell description contains its bounding box, determined from the minimum and maximum x and y coordinates of any feature within the cell. This will be used by the consistency checker.

The manipulation interface tailored for a hypothetical graphics editor for integrated circuit layout is the following. The basic required operations include the ability to retrieve boxes for display, to select all boxes that contain a particular coordinate, to delete or modify accessed boxes, and to create new leaf cells and generally manipulate the design hierarchy. To edit a particular cell requires that it be extracted from the design hierarchy (i.e., to begin a conversational transaction). The cumulative effects of cell compositions can be determined for each leaf cell. Retrieve, select, delete, and modify operations can be mapped into conventional database operations applied to the boxes representation table. The creation of a new cell requires updates to the cell, composition, and boxes representation tables. These are straightforward database operations issued by the boxes interpreter. When editing is done, the subtree is replaced and the consistency checker is invoked to insure that no cell has outgrown its boundary (i.e., to end a conversational transaction). The checker can be written so that invalid cells can be identified, and passed to the editor for highlighted display on a graphics terminal.

All software dealing with the design hierarchy and database access is located in the boxes interpreter. The writer of the integrated circuits editor can concentrate on the issues of man/machine dialogue and data presentation. Once an interpreter for the boxes data representation has been written, new tools that manipulate boxes can be developed more quickly.

6. Conclusions and Status

We have described an approach for applying database techniques to managing VLSI circuit design data. A prototype implementation is currently underway. We are building our system on top of a locally written storage system modelled along the lines of System-R´s RSS, and are constructing design interpreters for a limited number of representational types. In particular, we are designing an interpreter for geometric layout data. We have acquired several design tools from other universities, and intend to interface these to our interpreter.

Many important research issues require further investigation. First, we are developing new methods to store unstructured data in a database system. Second, we are investigating how the concept of design version differs from database snapshots, and whether new techniques are needed for their support. Third, we are exploring how to maximize parallel design activity by exploiting the hierarchical structure of the design. Are new methods of concurrency control required to support "conversational transactions", or can existing mechanisms, such as hierarchical locking protocols, be adapted? Fourth, we are investigating how to maintain design consistency by automatically propagating design changes to design components, either across design representations or through the design hierarchy. Finally, we are investigatng how to support alternative designs by adapting the techniques for hypothetical databases.

Eventually, we plan to adapt our architecture to a network environment, in which the design manager acts as a server for a network of design workstations. The issue is how to partition the layers of the design system between the workstations and the centralized server machines.

To demonstrate that new tools can be developed more quickly on top of a design data management system, we also intend to develop a set of tools for direct use with our system. These will include a graphical editor for integrated circuit layout, design and electrical rules checkers, and circuit simulators.

7. References

[ADIB80] Adiba, M. E., B. G. Lindsay, "Database Snapshots," Proc. Intl. Conference on Very Large Databases, (Oct. 1980).

[ALLE81] Allen, J., P. Penfield, "VLSI Design Automation Activities at MIT," IEEE Trans. on Circuits and Systems, V CAS-28, N 7, (July 1981).

[ASTR76] Astrahan, M. M., et. al., "System R: Relational Approach to Database Management," ACM Trans. on Database Systems, V 1, N 2, (June 1976).

[BAKE80] Baker, C. M., "Artwork Analysis Tools for VLSI Circuits," M.I.T. Technical Report, MIT/LCS/TR-239, 1980.

[BAND75] Bandurski, A. E., D. K. Jefferson, "Data Description for Computer-Aided Design," Proc. ACM SIGMOD Conference, (May 1975).

[BRYA81] Bryant, R. E., "Simulation of MOS LSI," Ph.D. Thesis, M.I.T. Department of E.E.C.S., 1981.

[CIAM76a] Ciampi, P. L., et. al., "Control and Integration of a CAD Database," 13th Design Automation Conference, 1976.

[CIAM76b] Ciampi, P. L., J. D. Nash, "Concepts in CAD Database Structures," 13th Design Automation Conference, 1976.

[CLAR80] Clark, J. H., R. Hannah, "Distributed Processing in a High-Performance Smart Image Memory," Lambda, V 1, N 3, (4th Quarter, 1980).

[COME79] Comer, D., "The Ubiquitous B-tree," ACM Computing Surveys, V 11, N 2, (June 1979).

[DIRE81] Director, S. W., et. al., "A Design Methodology and Computer Aids for Digital VLSI Systems," IEEE Trans. on Circuits and Systems, V CAS-28, N 7, (July 1981).

[DUTT81] Dutton, R. W., "Stanford Overview in VLSI Research," IEEE Trans. on Circuits and Systems, V CAS-28, N 7, (July

1981).

[EAST80] Eastman, C. M., "System Facilities for CAD Databases," 17th Design Automation Conference, 1980.

[EAST81] Eastman, C. M., "Recent Developments in Representation in the Science of Design," 18th Design Automation Conference, 1981.

[ELLI80] Ellis, C. A., G. Nutt, "Office Information Systems and Computer Science," ACM Computing Surveys, V 12, N 1, (Mar 1980).

[ESWA75] Eswaren, K, P., D. D. Chamberlain, "Functional Specifications of a Subsystem for Database Integrity," Proc. Intl. Conf. on Very Large Databases, (Sep. 1975).

[FAGI79] Fagin, R., et. al., "Extendible Hashing -- A Fast Access Method for Dynamic Files," ACM Trans. on Database Systems, V 4, N 3, (Sep. 1979).

[FELD79] Feldman, S. J., "Make -- A Program for Maintaining Computer Programs," UNIX Time-Sharing System UNIX Programmer´s Manual, Seventh Edition, Volume 2A, (Jan. 1979).

[FOST75] Foster, J. C., "The Evolution of an Integrated Database," 12th Design AUtomation Conference, 1975.

[GRAY78] Gray, J., "Notes on Database Operating Systems," IBM Research Report RJ2188(30001), 2/23/78.

[GRAY81a] Gray, J., et. al., "The Recovery Manager of the System-R Database Manager," ACM Computing Surveys, V 13, N 2, (June 1981).

[GRAY81b] Gray, J., "The Transaction Concept: Virtues and Limitations," Proc. Intl. Conference on Very Large Databases, (Sep. 1981).

[GRIF76] Griffiths, P. P., B. W. Wade, "An Authorization Mechanism For a Relational Database System," ACM Trans. on Database Systems, V 1, N 3, (Sep. 1976).

[HAYN81] Haynie, M. N., "The Relational/Network Hybrid Data Model for Design Automation Databases," 18th Design Automation Conference, 1981.

[HOSK79] Hoskins, E. M., "Descriptive Databases in Some Design/Manufacturing Environments," 16th Design Automation Conference, 1979.

[KAWA78] Kawano, et. al., "The Design of a Database Organization for an Electronic Equipment Design Automation System," 15th

Design Automation Conference, 1978.

[KELL81] Keller, K., "KIC: A Graphics Editor for Integrated Circuits," U. C. Berkeley Report, 1981.

[KORE75] Korenjak, A. J., A. H. Tiger, "An Integrated CAD Database System," 12th Design Automation Conference, 1975.

~~[LORI81] Lorie, R. A., "A Project on Design Systems," Database~~ Engineering Newsletter, V 4, N 1, (Sep. 1981).

[LOSL80] Losleben, P., "Computer Aided Design for VLSI," in Very Large Scale Integration VLSI: Fundamentals and Applications, D. F. Barbe, ed., Springer Series in Electrophysics 5, Springer Verlag, Berlin, 1980.

[MEAD80] Mead, C., L. Conway, Introduction to VLSI Systems, Addison-Wesley, Reading, MA, 1980.

[MITS80] Mitsuhasi, T., et. al., "An Integrated Mask Artwork and Analysis System," 17th Design Automation Conference, 1980.

[NEWT81] Newton, A. R., et. al., "Design Aids for VLSI: The Berkeley Perspective," IEEE Trans. on Circuits and Systems, V CAS-28, N 7, (July 1981).

[NIEN79] Nieng, K-Y, D. A. Beckly, "Component Library for an Integrated Design Automation System," 16th Design Automation Conference, 1979.

[OUST81] Ousterhout, J., "Caesar: An Interactive Editor for VLSI Circuits," U.C. Berkeley Report, 1981.

[PATT80] Patterson, D. A., C. H. Sequin, "Design Considerations for Single-Chip Computers of the Future", IEEE Trans. on Computers, V C-29, N 2, (Feb 1980).

[PATT81] Patterson, D. A., C. H. Sequin, "RISC I: A Reduced Instruction Set VLSI Computer," 8th Annual Symp. on Computer Architecture, 1981.

[ROBE81] Roberts, et. al., "A Vertically Organized Computer-Aided Design Data Base, 18th Design Automaton Conference, 1981.

[ROSE80] Rosenberg, L. M., "The Evolution of Design Automation to Meet the Challenges of VLSI," 17th Design Automation Conference, 1980.

[SEVE76] Severence, D. G., G. M. Lohman, "Differential Files: Their Application to the Maintenance of Large Databases," ACM Trans. on Database Systems, V 1, N 3, (Sep. 1976).

[SIDL80] Sidle, T. W., "Weakness of Commercial Database

Management Systems in Engineering Applications," 17th Design Automation Conference, (June 1980).

[SMIT77] Smith, J., D. Smith, "Database Abstractions: Aggregations and Generalizations," ACM Trans. on Database Systems, V 2, N 3, (Sep. 1977).

~~[STEE80] Steele, G. L., G. J. Sussman, "Design of a LISP-Based Microprocessor," Comm. ACM, V 23, N 11, (Nov 1980).~~

[STON75] Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification," Proc. ACM SIGMOD Conf., (May 1975).

[STON76a] Stonebraker, M. R., et. al., "The Design and Implementation of INGRES," ACM Trans. on Database Systems, V 1, N 3, (Sep. 1976).

[STON76b] Stonebraker, M. R., P. Rubenstein, "The INGRES Protection System," Proc. 1976 ACM National Conference, (Oct. 1976).

[STON80] Stonebraker, M. R., K. Keller, "Embedding Expert Knowledge and Hypothetical Data Bases into a Data Base System," Proc. ACM SIGMOD Conf., (May 1980).

[STON81] Stonebraker, M. R., "Hypothetical Databases as Views," Proc. ACM SIGMOD Conf., (May 1981).

[SUCH79] Sucher, D. J., D. F. Wann, "A Design Aids Database for Physical Components," 16th Design Automation Conference, 1979.

[TERM81] Terman, C. J., "Simulation Tools for VLSI Design," Ph.D. Thesis, M.I.T. Department of E.E.C.S., 1981.

[TRIM81] Trimberger, S., et. al., "A Structured Design Methodology and Associated Software Tools," IEEE Trans. on Circuits and Systems, V CAS-28, N 7, (July 1981).

[TSIC80] Tsichritzis, D. "OFS: An Integrated Form Management System," Univ. of Toronto Computer Systems Research Group Tech. Rep. CSRG-111, (Apr. 1980).

[VALL75] Valle, G. "Relational Data Handling Techniques in IC Mask Layout Procedures," 12th Design Automation Conference, 1975.

[WILM79] Wilmore, J. A., "The Design of an Efficient Data Base to Support an Interactive LSI Layout System," 16th Design Automation Conference, 1979.

[WONG79a] Wong, S., W. Bristol, "A Computer Aided Design

Database," 16th Design Automation Conference, 1979.

[WONG79b] Wong, E., R. H. Katz, "Database Design and Schema Conversion for Relational and DBTG Databases," 1st Intl. Conference on Entities and Relationships, (Dec. 1979).

[WORK74] Works, K., et. al., "Engineering Data Management Systems (EDMS) for Computer Aided Design of Digital Systems," 11th Design Automation Conference, 1974.

[ZINT81] Zintl, G., "A CODASYL CAD Database System," 18th Design Automation Conference, 1981.