JUNE 1982 VOL. 5 NO. 2

a quarterly bulletin of the IEEE computer society technical committee on

Database Engineering

Contents

Database Engineering Bulletin is a quarterly publication of the IEEE Computer Society Technical Committee on Database Engineering. Its scope of interest includes: data structures and models, access strategies, access control techniques, database architecture, database machines, intelligent front ends, mass storage for very large databases, distributed database systems and techniques, database software design and implementation, database utilities, database security and related areas.

Contribution to the Bulletin is hereby solicited. News items, letters, technical papers, book reviews, meeting previews, summaries, case studies, etc., should be sent to the Editor. All letters to the Editor will be considered for publication unless accompanied by a request to the contrary. Technical papers are unrefereed.

Membership in Database Engineering Technical Committee is open to IEEE Computer Society members, student members, and associate members. (Application form in this issue.)

Letter from the Guest Editor

This issue of the Database Engineering Newsletter is devoted to the topic of Design Data Management, i.e., how to apply traditional database system techniques to the new applications area of the design environment. We have reports of current research from five groups in Academia and Industry: Boeing Computer Services, I.B.M. Research at San Jose, Stanford University, U.C. Berkeley, and the University of Wisconsin-Madison.

The papers fall into three categories. The group from Boeing describe their design environment and its data management requirements, from the viewpoint of the CAD applications builder. The papers from I.B.M. and Wisconsin describe systems to support design data being built by database people. The remaining two papers, from Stanford and Berkeley, address the question of whether conventional database systems can provide adequate performance for CAD applications. Interestingly enough, the two groups come to very different conclusions!

In reading these reports, one is immediately struck by two common themes: support for hierarchically constructed design objects (e.g., "composite objects", "complex objects", "ragged relations", "design hierarchy"), and the pervasive need to keep accessing costs "reasonable."

All of these papers are snapshots of works-in-progress. We are just beginning to understand the problems and how they map into database system structures. We are far from solving all of them. Design data management is becoming an important area of database research as we enter the mid-1980's.

Yours truly,

Randy H. Katz

Randy H. Katz
Madison, Wisconsin

Engineering Data Management Activities Within the IPAD Project

H. R. Johnson
D. L. Bernhardt

Boeing Computer Services
Seattle, Washington

ABSTRACT

In this paper we summarize the research and development in engineering data base management systems by the IPAD project at the Boeing Company.[1]

1.0 Introduction

In 1976 NASA awarded the Boeing Company a contract to develop IPAD (Integrated Programs for Aerospace-Vehicle Design). The specific goal of IPAD was to increase productivity in the United States aerospace industry through the application of computers to manage engineering data. The contract included a requirement for Boeing to form an Industrial Technical Advisory Board (ITAB) to guide the development of IPAD [1] . Members of ITAB represent major manufacturing (aerospace and other) and computer companies. NASA, Boeing, and ITAB have worked together since that time to analyze engineering design methodologies; to establish requirements for integrated, computer based systems for managing engineering data, and develop software to demonstrate these concepts. Results have included development of the RIM and IPIP data base management systems and a network facility for multi-host intertask communication. IPAD documentation and software is in the public domain.[2] See [2] for a comprehensive discussion of IPAD objectives and products.

2.0 Requirements for Engineering Data Management

Early in the IPAD contract, the engineering design process was analyzed and a number of requirements for engineering data management were identified. This work was documented in [3 , 4]. The following briefly summarizes these requirements.

Some of the more obvious requirements included: a FORTRAN interface; support for scientific data types for both scalar and non-scalar (large matricies) attributes; selection and projection of data with respect to specific elements in matricies; support for defining and manipulating geometry data and interfacing this with design drafting and graphic display systems.

Multiple levels and styles of data description are required to provide for differing data requirements and for data independence to support a variety of users in a dynamic environment. Specifically, the network and relational data models are required.

[1]This work is funded under NASA contract NAS1-14700.

[2]Correspondence may be addressed to the IPAD Program Management Office, The Boeing Company, P. O. Box 24346, Seattle, WA 98124, M/S 73-03

Support is required for dividing a data base into logical partitions, each containing data associated with an engineering task. A logical partition, referred to in the following as a "data set", may include tuples/records from one or more relations/record types. A relation/record type may span data sets. Facilities must be provided to restrict access to data sets.

The data manager must also support user description of data sets. This meta-data may include information such as sources, uses, time of creation, and the quality of data in the data set. The data manager must support user access to this meta-data.

Versioning of data sets is required to support the iterative nature of the design process. The data manager must provide for efficient access to versioned data sets while minimizing physical redundancy across versions. Comparative analysis of versions of a data set should be supported, along with facilities for tracking the history of change to a particular design.

A mechanism must be supported to release (approve or sign off) a data set and to insure that it cannot be changed once released. New versions may be created and modified to record design changes.

Long term archival of data sets must be supported by the data manager. Archived data sets must be available upon user demand. Data describing archived data sets must be available for online perusal.

The data manager must support interactive retrieval and update to support interactive design and ad hoc queries.

The data manager must support distributed processing in a loosely coupled network of heterogeneous machines so that data may be communicated between geographically dispersed divisions of the same firm as well as firms which have received subcontracts for portions of the design work. The data set is one natural unit of transfer within the network.

A uniform interface to system facilities should be provided to facilitate learning and use of the system and communication with other users.

To support the total CAD/CAM environment, the data manager must also provide support to manufacturing in the traditional sense in terms of providing support for shop scheduling, material requirements planning, inventory control of finished goods, and the accounting function. It must also provide for the generation of machine tapes to aid in the transfer of the design which exists in the data base to the tools which will machine the parts of the designed product. See [5] for a more complete description of these requirements.

Finally, the data manager must provide capabilities to support project management. This should include triggering on user meta-data so that the creation of data sets may be scheduled and progress monitored by the system.

3.0 The RIM Data Base Management System

The RIM (Relational Information Managment) data base management system was developed on the IPAD project to explore relational data base concepts prior to the development of IPIP. RIM has been enhanced by the University of Washington and The

Boeing Company in cooperation with ITAB and the IPAD Project.

A RIM data base may be accessed in read mode by multiple users. Access to the data base is restricted to single user, when it is opened for update.

RIM supports a single level of data definition. Relations are organized into schemas. Schema definition may be entered and modified interactively through a menu interface. Rules on data relationships within and between relations may be declared. Rules can be used as constraints, although the user may turn rule checking off and on.

RIM provides several scientific support capabilities with its matrix, vector, and real data types. It also supports a tolerance capability which supports qualification by user-specified approximation of equality.

RIM offers both algebra-level (including join) and calculus-level (excluding join) data manipulation commands through an interactive interface, along with facilities for formatting retrieved data. RIM supports the calculus-level data manipulation commands for FORTRAN programs via subroutine calls.

RIM is written primarily in FORTRAN 66, but will compile in FORTRAN 77. It is available on several hosts: CDC, UNIVAC, VAX, and PRIME. More than 130 copies of RIM have been supplied to universities and corporations. RIM is used on a daily basis in many of these organizations.

4.0 The IPIP Data Base Management System

The IPIP (IPAD Information Processor) data base management system is intended to manage engineering data over time through CAD and CAM environments. To this end, IPIP supports multiple data models, multiple levels of schemas, and concurrent, multiple user access through multiple application interfaces in a distributed environment.

Scientific data types and arrays are supported. Composite objects called structures, which may consist of multiple tuples from multiple relations, may be declared and manipulated as entities to manage geometry and other scientific data. Data may be partitioned logically into data sets to support concurrent access, versioning, releasing, and archival procedures. See [6]for a general description of IPIP.

IPAD has developed a general purpose network facility for intertask communication in a hetergeneous distributed processing environment. The IPAD network has been implemented in accordance with ISO specifications of layered protocol. Access to IPIP and IPIP-managed data is via this network. See [7] for a general description of the network facility. IPIP and the network are written primarily in Pascal.

IPIP is in early stages of release and consequently is in an evolutionary state. For example, the latest version of IPIP supports some of the locking aspects of data sets, but not versioning, releasing, or archival; and initial facilities for the declaration and manipulation of structures are in integration testing. Application interfaces are limited to FORTAN programs. IPIP proper, its schema compilers, and CDC and DEC FORTRAN precompilers execute on CDC CYBER series machines operating under the NOS operating system. DEC VAX 11 FORTRAN programs against the CYBER-resident data base may be submitted via the IPAD network from a VAX 11/780 (operating under the VMS operating system) and then executed on the VAX, DML commands being forwarded to the CYBER for execution and results being returned via the network to the VAX.

Application programs submit and receive data in native machine representation. Currently, IPIP is executing only in a test environment at the Boeing IPAD site.

4.1 Compliance With Standards

One major objective of IPIP has been compatibility with specifications for data definition and manipulation languages which seem to be leading to a standard. It was felt that this approach would take advantage of previous work and tested constructs, would facilitate migration between standard compliant systems and IPIP, and should facilitate incorporation of IPIP extensions into standards.
Given the work of CODASYL and ANSI committees and the relationships between these committees, and given the engineering requirement for a FORTRAN interface, the IPIP Logical Schema Language (LSL) was based upon a subset of the 1978 CODASYL DDL [8] and the IPIP Data Manipulation Language (DML) was based on a subset of the 1978 CODASYL FORTRAN DML [9] . Extensions and departures from these specifications were made in some instances. Some of these are discussed in the following sections.

4.2 Integration of User Interfaces and Capabilities

Integration has been a major objective of IPIP: integration of user interfaces and integration of capabilities. A single Logical Schema Language (LSL) and Data Manipulation Language (DML) supports both the relational and network data models and applications, be they interactive or written in one of several programming languages. The LSL is used at both the conceptual and external levels of data definition [10] . The Internal Schema Language (ISL) resembles the LSL as nearly as possible. Scientific features such as the structure-defined composite objects for geometry are integrated with other data mangement capabilities.

Most aspects of data definition and data manipulation are invariant or are very similar across data models, application environments, and logical and physical descriptions. The DATA MODEL clause in the LSL determines which data model specific constructs (e.g., CODASYL set or relational foreign key) may be used in a particular schema. Relational model specific constructs resemble corresponding network constructs. The HOST LANGUAGE clause in the LSL and ISL specifies which host language rules must be used in a particular schema to name relations and attributes and to specify data types. (Alternate, host-independent names may be specified for readibility.) An application treats both a simple object and a structure-defined composite object in the same way (i.e., as a relation/record).

The advantages of this integration are several: less language to learn for users of multiple data definition and/or data manipulation environments; uniformity of semantics across environments; access to common data at any level of logical schema through any IPIP-supported data model or application environment; support for the shop where just one data model and/or application environment is desired; availability of capabilities traditionally ascribed to one environment in other environments; commonality in implementation supporting multiple environments.

4.3 Data Architecture

By data architecture we mean the framework for configuring data definition and data manipulation. Data definition for IPIP is organized into schemas, which may be tied together to provide data independence and various views of data. Data manipulation commands appear in application programs and interactive sessions.

There are three types of IPIP schemas: internal, logical, and mapping. The IPIP internal schema corresponds to the internal schema of the ANSI DBSG [10] and the storage schema of the CODASYL DDLC [8] . IPIP logical schemas correspond to ANSI conceptual and external schemas and also to CODASYL schemas and subschemas. The IPIP mapping schema is used for mapping between schemas of the other two types.

An IPIP data base is described by a single level of internal schemas and one or more levels of logical schemas mapped to underlying logical and/or internal schemas. An application program or session may be formulated against logical schemas at any level.

Logical schemas can be configured in the ANSI and CODASYL tree structure arrangement to provide for centralized definition of a data base through a comprehensive, base-level logical (conceptual) schema.

Multiple tree configurations of logical schemas can be coupled by mapping one logical schema to multiple logical schemas or by invoking multiple logical schemas in a single application program or session. This provides for decentralized definition of a data base or of a federation of data bases[11]. This coupling capability may be used for a variety of purposes ranging from integration of existing data bases with minimal change to data definition, to decentralization of data administration over multiple clusters of shared and/or private data.

The ability to vary the depth of logical schemas supports control of data independence over a data cluster. The ability to couple logical schemas horizontally supports control of independence of data and data administration across data clusters -additional dimensions of independence. Data independence is enhanced in both cases by the use of mapping schemas, since change often involves only interschema mappings.

IPIP provides for pre-runtime binding of programs and logical schemas. Programs may be bound at runtime regardless of whether underlying schemas have been bound.

Together, the richness of the IPIP data architecture along with IPIP binding options permits tradeoffs between degrees of data independence and the overhead of creating, maintaining, and referencing data description.

4.4 Support for Multiple Data Models

The IPIP LSL and DML support both the relational and network data models. These languages were based on subsets of the 1978 CODASYL DDL and FORTRAN DML specifications. Included were those CODASYL constructs supporting sets for which relationships are determined by states (value or null) of corresponding attributes in owner and members. Constructs specific to other set selection criteria were excluded. Inclusion of some constructs (e.g., for set ordering and for multiple member types) was deferred for one reason or another.

The CODASYL INSERTION and RETENTION clauses which specify constraints on association/disassociation of members with/from owners were retained and extended with respect to the handling of null attributes and dovetailed with an IPIP extension (MEMBERSHIP clause), which provides for member records to be put into ownerless 'potential' set occurrences and to be associated automatically by IPIP with an owner when it is created as well as providing for the CODASYL option where owner must exist whenever member does (referential integrity in relational terminology). IPIP extensions include explicit clauses governing IPIP propagation (both from owner to member and member to owner) of record deletion. The CODASYL SOURCE clause which provides for system propagation of item state from owner to member was extended to provide for bidirectional propagation.

For more direct support of the relational data model, a FOREIGN KEY clause was included in the LSL using syntax which retained the relational flavor of the concept, but paralleled set syntax. Clauses were included to specify insertion, retention, and membership options in terms of item states. Propagation of record (tuple) deletion and item state may be specified relative to foreign keys as well as to sets.

The IPIP DML provides for operations relative to foreign keys (by name) paralleling CODASYL operations relative to sets. A WHERE phrase was incorporated into the IPIP FIND, FETCH, MODIFY, and DELETE commands to support specification of more general conditions for calculus-level operations. Relation name in a DML command may be qualified by a cursor name to support program definition of multiple relations, concurrently, over a single schema-defined relation.

A DATA MODEL clause was included in the LSL to govern which data model dependent constructs (i.e., set, foreign key, or both) may be used in a particular schema. 'RELATION' and 'RECORD' are treated as synonyms in IPIP languages, and may be used interchangeably regardless of data model specified. A DOMAIN clause, which was included in the LSL and ISL to provide for user-declaration of data types, may be used with either data model. And it is intended that in future releases of IPIP that regardless of data model specified, DML commands across relations (records) may be expressed either in the CODASYL style of referencing schema-declared relationships (e.g., FETCH items of A, items of B VIA (name of) schema-declared set or foreign key relating A and B; where $a_1=b_1$, ... $a_n=b_n$ is the criteria defining that relationship) or in the relational style of in-line specification of relationship criteria (e.g., FETCH items of A, items of B WHERE $a_1=b_1$, ... $a_n=b_n$). The full relational join is not available at this time.

The unified approach to support for the network and relational data models is described more fully in [6 , 12].

4.5 Other Scientific Capabilities

Currently, attributes of IPIP relations may be integer, real, character, or boolean scalars or arrays. Selection and projection on individual elements of an array is not supported at this time.

Initial capabilities for data set partitions of a data base are supported. A data set is declared implicitly on first user access. Data set intersections with relations are the units of locking data for read/update. Access by data set is supported by IPIP indexing (B-tree). IPIP indexing and address conversion structures have been designed to support access to versions of data sets while minimizing physical redundancy of data across versions. Data sets may be used in specifying data to be processed by a prototype facility for transferring data between IPIP and RIM data bases.

Composite objects called structures are supported to manage geometry and other scientific data. A structure is defined in a logical schema to consist of tuples from a tree or network of relations as related by foreign keys. A structure may also be defined in terms of records and sets. A relation/record in one logical schema may be mapped to a structure in another schema. Such a relation/record is said to be structure-defined. A structure is manipulated (retrieval and update) as an entity through operations on a structure-defined relation; the same commands used on non-structure defined records.

A user accesses structure-defined data at the entity (e.g., surface, curve, segment) level. A single user command may result in IPIP processing of multiple underlying tuples as specified by schema-declared constraints and propagated actions for relations and foreign keys within the structure. On store, IPIP will generate values for unique keys when the user does not. IPIP will sequence retrieved data according to schema specification for inclusion in the structures-defined record. User productivity is enhanced through support of entities which are natural to his application. Data integrity is enhanced through definition of structure processing in the schema, as opposed to complicated command sequences being embedded in numerous applications.

5.0 Distributed Processing

Some initial capabilities for distributed processing are in place. This includes the IPAD network and the VAX interface to CYBER data bases (see Section 4.5). The ability to loosely couple IPIP data bases (see Section 4.3) has extensive potential for distributed data management.

A prototype facility is available for transferring data between IPIP and RIM data bases on a copy and replace basis. One area for further investigation here is the use of IPIP and RIM in tandem to manage global and local data bases; data sets being copied to RIM for local use. Questions arise as to the implications for versioning of data sets and/or locks on data sets which span program executions.

REFERENCES

[1] W. E. Swanson. "Industry Involvement in IPAD Through the Industry Technical Advisory Board", Proceedings of IPAD National Symposium, NASA Conference Publication 2143, September 1980, pages 21-26.

[2] R. E. Miller. "IPAD Products and Implications for the Future", Proceedings of IPAD National Symposium, NASA Conference Publication 2143, September 1980, pages 219-234.

[3] D. D. Meyer. "Reference Design Process", IPAD Document D6-IPAD-70010-D, March 1977.

[4] J. W. Southall. "Integrated Information Processing Requirements", IPAD Document D6-IPAD-70012-D, June 1977.

[5] T. R. Fisher, E. G. McKenna, D. D. Meyer, and J. E. Schweitzer. "Manufacturing Data Management Requirements", IPAD Document D6-IPAD-70038-D, December 1981.

[6] D. L. Comfort, H. R. Johnson, and D. D. Shull. "An Engineering Data Management System", Proceedings of IPAD National Symposium, NASA Conference Publication 2143, pages 145-178, September 1980.

[7] F. M. Ives, D. M. Kirkwood, and J. G. Tanner. "Executive and Communication Service to Support the IPAD Environment", Proceedings of IPAD National Symposium, NASA Conference Publication 2143, September 1980, pages 95-144.

[8] CODASYL Data Description Language Committee. "Journal of Development", January 1978.

[9] CODASYL COBOL Committee. "Journal of Development", October 1978.

[10] A. Klug and D. Tsichritzis, Editors. "The ANSI/X3/SPARC DBMS Framework", Report of the Study Group on Data Base Management Systems, AFIPS Press, 1977.

[11] D. Heimbigner and D. McLeod. "A Federated Architecture for Data Base Systems", Proceedings of the National Computer Conference, 1980, pages 283-289.

[12] H. R. Johnson, J. A. Larson, and J. D. Lawrence. "Network and Relational Data Modeling in a Common Data Base Architecture Environment", Sperry Univac Research Report TMA00720, Roseville MN., March 1979.

[13] R. P. Dube, G. J. Herron, J. E. Schweitzer, and E. R. Warkentine. "An Approach for Management of Geometry Data", Proceedings of IPAD National Symposium, NASA Conference Publication 2143, September 1980, pages 179-202.

Using a Relational Database System for Circuit Design

Roger Haskin
Raymond Lorie

IBM Research
5600 Cottle Road
San Jose CA 95193

The revolution in circuit fabrication technology poses problems that threaten to make many conventional design automation systems obsolete. The size and complexity of modern computer systems increases both the amount of data to be managed and the number of designers accessing the data. Also, the long turnaround time necessary to correct chip design errors requires extensive simulation to be done prior to circuit fabrication. Data necessary to the simulation must be correlated to the network specification (i.e. parts and wires), and this correlation must remain consistent as the design evolves.

The difficulty of adapting the usually ad-hoc data management facilities of existing design systems to handle larger designs has created interest in employing modern database technology on their behalf. However, while many features of modern database systems make them attractive for use in a design environment, the fact that they were intended to manage business data compromise their ability to be integrated into a design system.

At IBM San Jose Research, we are in the process of modifying System R ([1]) to improve its ability to handle design data. Some of this work is described in [2]. briefly, the extensions we are making include:

Complex Objects are a facility that allows the user to declare and to specify structural relationships among semantically related data. Fig. 1 shows a complex object for a simplified 'module' (e.g. logic block). Complex objects are implemented by defining three new column types. The MODULES.MID, PARTS.PID, FUNCTIONS.FID, and SIGNALS.SID columns are all of the new type IDENTIFIER. The PARTS.MID, SIGNALS.MID, FUNCTIONS.PID, AND PINS.FID are all of type COMP_OF, denoting that the tuples in these relations are components of (i.e. hierarchical descendents of) higher-level objects (e.g. a pin belongs to a function residing on a part of a module). PINS.SID is of type REF, which is used to express non-hierarchical references. The complex object mechanism allows partitioning the relations containing object components (e.g. PARTS, FUNCTIONS, etc.), and expressing the connectivity among elements of the partitions (e.g. saying which gates are wired to which). Fig. 2 shows a small example fragment of a logic diagram expressed in this schema. Entries in the MID, PID, FID, and SID columns are shown as integers, but are really system generated unique values (machine id, timestamp). Complex objects retain the use of value matching to define the partitions and connections, and also retains the relational query language to perform data manipulation (although see the comment below). However, by defining the object structure to the system, it is possible to provide enhanced performance and integrity checking.

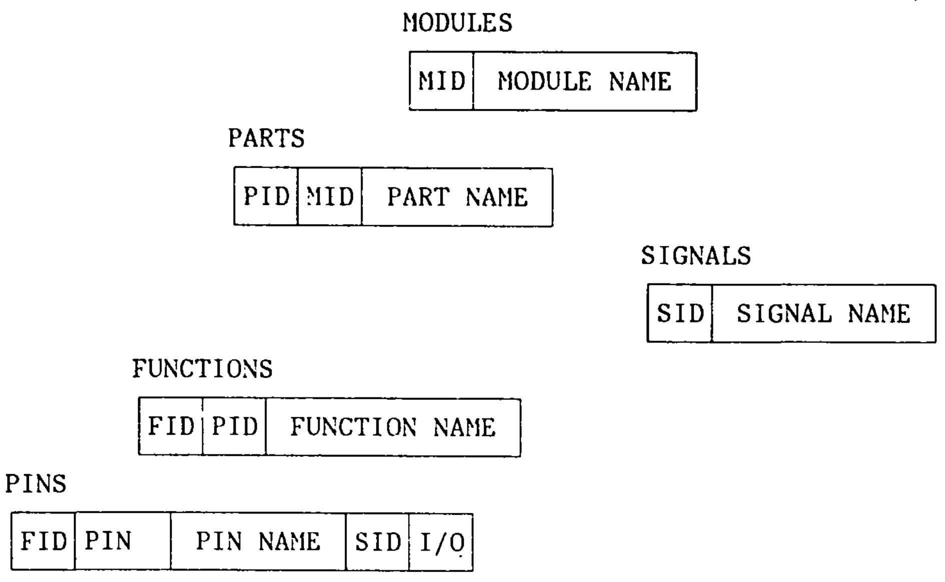

Fig. 1 - Complex Object Schema for Gedanken Logic Designs

Conversational Transactions. Use of the design database is likely to be highly interactive. This poses problems for the transaction management and concurrency control facilities of System R (as it would for most other systems). A transaction is defined to span the period during which the database is inconsistent. System R was designed for transactions that access only a few records, complete quickly (i.e. fractions of a second), and have inputs that are known in advance. Conversational transactions address two problems of standard transaction management:

a. Changes made by a transaction can be backed out (e.g. due to system crashes). Backout is intolerable when the transaction spans an inconsistency due to a long series of interactively entered changes.
b. It is standard practice to suspend (block) a transaction attempting to access a locked object until the lock is acquired (i.e. rather than informing the transaction and letting it decide what do). Interactive users would undoubtedly find such suspensions bothersome.

In most interactive applications, the program's data access patterns map well onto complex objects. In these situations, complex objects are a more appropriate lock granularity than relations or tuples. The heart of the conversational transaction mechanism is a facility for acquiring locks on entire complex objects, thus including them in the lock graph ([2,3]). Conversational locks persist past end of transaction until explicitly released by the user. An attempt to acquire a conversational lock on an already locked object returns an error rather than suspending the transaction.

Database-Data Structure Interface. To achieve adequate performance, many functions (e.g. display management, simulation) will probably be done using memory-resident data structures. Perhaps the overriding database performance issue is how fast these structures can be built from data

PID	MID	PART NAME
23	4	SN7400
71	4	SN7474

SID	SIGNAL NAME
21	STROBE

FID	PID	FUNCTION NAME
117	23	2inp. NAND
81	23	2inp. NAND
59	71	D FF w/ P,Clr

FID	PIN	PIN NAME	SID	I/O
117	3	Y1	21	O
81	4	A2	21	I
59	2	CK	21	I

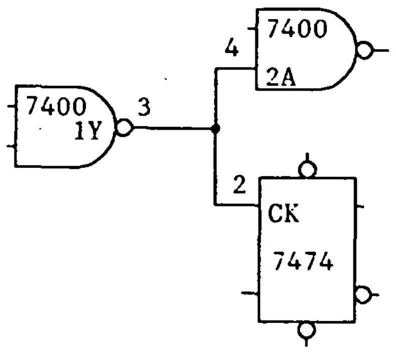

Fig. 2 - Complex Object Fragment for a Signal Path

in the database. The overhead of the record-at-a-time SQL interface (shared by most other systems) is too high to allow adequate performance. We are therefore providing an object-oriented interface to allow a data structure to be built from the data in one complex object or a set of objects. In addition to creating a data structure from complex objects, this interface will provide language constructs to initialize elements in the structures that do not contain data from the database, and will automatically translate structural interrelationships defined in the complex object into pointers in the data structure.

Storing Non-Coded Data. In common with most business-oriented data management systems, System R has only rudimentary facilities for handling the non-coded data necessary to many phases of the design process (e.g. simulation checkpoint data, circuit mask patterns). System R's facilities for managing non-coded data are being modified to greatly improve performance on updates, to support data items of essentially unlimited length (2+ Gb), and to manage data stored outside of System R (e.g. in VSAM datasets).

Preliminary Results

A detailed description of these extensions is available in [2], and will not be repeated here. However, experience with the design gained during the implementation effort have led us to several interesting observations.

Our original intent was to implement complex objects by giving each tuple in the object a unique identifier, and handling structural navigation (e.g. retrieving all tuples in a relation belonging to a given object) using joins. This is, of course, the way a user would implement such objects on top of a relational system. This proved unsatisfactory for three reasons: the space overhead of storing identifiers in tuples of large objects, the time overhead of performing a join (or at least consulting an index) to follow a reference, and the fact that duplicating an object would have required a network copy, generating new identifiers and resolving references to them. we therefore implemented intra-object access paths using a special table, called the 'object map'. The object map contains an entry for each tuple in the object (the root tuple occupying the first entry), each entry holding the tuple's address (TID). The user thinks that COMP_OF and REF columns contain identifiers, but in fact they contain an index into the object map. This speeds following logical links (to at most two I/O's) and enables copying an object by merely duplicating the tuples and rebuilding the map as the new tuples are created.

Another interesting question involves the suitability of the relational query language for performing queries on complex objects. Consider the object shown in Fig. 1. Further, suppose we wish to select all modules where an output pin is connected to more than ten inputs. This would require the following SQL query:

```
SELECT UNIQUE MODULE_NAME
FROM   MODULES,PARTS,FUNCTIONS,PINS P1
WHERE  MODULES.MID = PARTS.MID AND
       PARTS.PID = FUNCTIONS.PID AND
       FUNCTIONS.FID = PINS.FID AND
       PINS.IO = 'O' AND
       10 < (SELECT COUNT (UNIQUE PNID)
             FROM   PINS P2
             WHERE  P2.IO = 'I' AND
                    P2.SID = P1.SID);
```

Not only is this query clumsy, requiring explicit coding of the joins necessary for navigating from the pins to the modules containing them, but they do not even map well into the access path that will be used since, as mentioned above, the root tuple is known to be the first entry in the object map. It therefore appeared desirable to provide improved facilities for maneuvering in complex objects, for example:

```
SELECT UNIQUE MODULE_NAME
FROM   MODULES,PINS P1
WHERE  MODULE.MID = ROOT_OF(PINID) AND
       PINS.IO = 'O' AND
       10 < (SELECT .....
```

Although the example schema and queries are rather contrived, we are in the process of investigating using the extended System R to manage data for a large internal design system. Preliminary estimates show that in addition to the added integrity offered by complex objects, the performance will be dramatically better than it would be for a comparable schema using INTEGER columns and joins rather than IDENTIFIER, etc. and the object map.

Summary

We believe the System R extensions described here overcome many of the problems conventional database systems have in managing design data. In particular,

1. By defining the structure of objects to the system, improvements in performance, structural integrity, and ease of applications programming can be achieved.

2. Complex objects are a more appropriate unit of locking in many cases than are relations or tuples. Object locks were used to implement conversational transactions, which greatly improve the ability of the system to be used interactively.

3. Providing the ability to build program data structures directly from large objects or sets of objects, and allowing non-coded data to be stored and retrieved efficiently and in synchronization with the transaction and recovery mechanism enhance the performance and utility of the database system for many common design operations.

References

1. Astrahan, M. M., et. al., 'System R: Relational Approach to Database Management,' ACM Transactions on Database Systems, Vol. 1 No. 2, June 1976, pp 97-137.

2. Haskin, R. L., Lorie, R. A., 'On Extending the Functions of a Relational Database System,' Proc. 1982 ACM-SIGMOD Intl. Conf. on Management of Data, June 1982.

3. Gray, J. N., Lorie, R. A., Putzolu, G. R., and Traiger, I. L., 'Granularity of Locks and Degrees of Consistency in a Shared Data Base,' in Modelling in Data Base Management Systems (G. M. Nijssen, ed.), North Holland, 1976.

PERFORMANCE OF DATABASE MANAGEMENT SYSTEMS IN VLSI DESIGN

Anne Beetem
Jack Milton
Gio Wiederhold

I. Introduction.

VLSI custom design involves the manipulation of large volumes of diverse interrelated data. The need to invest 4 to 30 man years for microprocessor design makes single-designer oriented methodologies infeasible [Sch79]. The effort to communicate concepts and data among multiple design specialists probably contributes significantly to the design cost, and the volume of data is certain to grow as refinements in fabrication methods allow increases in component count and heterogeneity. Because database technology has dealt with both the management of large volumes of data and problems caused by concurrency of data access ([Ful80], [Mal80]), we investigate the effective handling of VLSI design data through the use of database management systems with their attendant advantages of flexibility, ease of use, provisions for protection, increased data independence, and concurrency control, for example.

There are several key issues which must be addressed if databases are to become effective tools in VLSI design. There seems to be little disagreement that design systems should exploit hierarchy [Kat82]. Additionally, it is of critical importance that the system allow for a variety of design methodologies so that 1) changes at any level can be reflected both up and down the design hierarchy, and 2) different representations of the design (e.g. sticks, layout geometry, circuits, gate logic) are allowed. Ultimately, the system should maintain equivalence between these representations more or less automatically ([Be82],[Kat82]). Explicit storage for all attributes of all elements at lower levels of a design must be avoided in order to permit effective management of change emanating at higher levels, because excessive replication of lower level elements vitiates the benefits of hierarchical design methods, and because explicit storage at the gate instance level creates excessive storage demands. To avoid these demands the database system must be able to fetch actually instantiated data, compute potential, non-instantiated data using stored algorithms, or "infer" certain parameters of the design from relationships between elements (See [Ko81]). Moreover, in order to make the move away from specialized design files to database systems acceptable to the design engineering professionals, the performance of the system has to be such that the degradation of performance relative to that of specialized files is proportional to the benefits gained.

II. Current Work.

The main objective of our work is to determine whether commercial database systems, used knowledgeably, can perform adequately.

The initial set of experiments has revolved around DEC's DBMS-20, a system based on the published CODASYL database definition [DBT71]. We spent a major initial effort to design the database schema. In order to obtain correct and high performance operation from current commercial systems it is essential that the semantics of the application be modelled carefully before attempting to map them into a database schema [WEM79]. Due to availability of data and programs, we are using data from conventional circuit board design to model the VLSI design process. Since many of the problems are similar, we believe that the results are 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.

We first demonstrated the replacement of design files by an equivalent database representation. Specialized design files of circuit information were used as the control. SDL (Structural Design Language [vC79]) is a straightforward hierarchical description language (e.g. registers are described in terms of flip-flops, flip-flops are described in terms of gates, gates are described in terms of transistors). The SDL compiler produces a binary file from a design written in SDL. The binary file contains the logical description of each component described in the design which is then loaded into the SPRINT database [vC77], the specialized control subject, via a "hardwired" schema. For test purposes, we chose the PDP-11 processor, as described in SDL [SL79]. There are eight different levels in the hierarchical description: cpu (central processing unit), reg (registers), rf1 (priority arbiter and timer), rf2 (register files), ff (flip flops), gate (gates), trans (transistors) and bottom (primitives).

The CODASYL schema was modelled closely to the SPRINT schema except for revisions to take advantage of the network structure. The binary file produced by the SDL compiler was loaded into the DBMS-20 database. The initial loading of the PDP-11 example took SPRINT about one minute and required about four minutes for DBMS-20, which we find acceptable.

Reading from the Database

The next experiment tested the time it takes to both read from and write into the database. A macroexpander program [Pay80] was used which first reads the logical description of a high level component from a database. The user then specifies the levels to be expanded. Thus, if the user requests that the program expand all levels down to the transistor, the resulting output will be the logical description of the original component described totally in terms of transistors. This form of description could be the input to 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 very many 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 PDP-11 and its ALU:

		SPRINT	DBMS-20	Records read	Words read
ALU	IO time	21s	33s	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

To test writing into the database we do not have a control, as the SPRINT design system does not have this capability. We proceeded with the DBMS-20 tests hoping that the times would be acceptable both in an absolute sense and relative to the retrieval times. In order to simulate a real application of VLSI design, we considered how the database would handle instantiations.

To exercise the representation of a given device within the database, the macroexpander program was modified so that it could write back into the database. In this mode, after the program expands a component, it stores the new representation as another internal description. In the design atmosphere, if the user makes some changes to the original description at a given level, and stores this version in the database, the program can choose the appropriate internal description to use when a higher level component is expanded. This scenario was tested on one of the registers, the ALU.

Two of the parts in the upper level description of the ALU are the 40181 and the 40182. When these pieces were expanded, the 40181 gave a total read time of 17.4 seconds and a total write time of 41.3 seconds. The corresponding times for the 40182 were 8.78 and 8.82 seconds. Two other pieces at this level (the MUX and the XOR) were also expanded, and the ALU was expanded again with the expanded versions of these lower level pieces in the database. The total read time was 48.0 seconds, and the total write time was 203.0 seconds. We consider that the DBMS-20 implementation showed an acceptable performance.

Signalling Changes in the Database

At this stage, we explored implementation of data management facilities not provided by specialized design files. A strong motivation for using a database is that it provides the data as a resource to all design participants and so becomes the communication medium in a future VLSI design environment. To carry this function effectively the database system has to be augmented with communication paths. These paths may ultimately be oriented in any combination of directions -- up or down between different hierarchical levels, and sideways between different representations of a given design.

The downward direction is supported by the hierarchical design process, but in practice design changes also flow upwards, as instances at a lower level are modified to satisfy layout, timing, fanout and other performance oriented needs. We hence have developed communication in the upward direction, which relates detail to more abstract specifications. The creation or modification of lower level instances has to be bound to the appropriate higher level elements. We implement this task by a signalling scheme. Multiple structures at higher hierarchical levels 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 of the change creates an exception flag at the higher structures. At a later time, when the level to which the signal was directed is accessed by the specialist responsible at that design level, the system provides a warning that a change has been made. An appropriate action could then be taken; for example, verification of continued correctness at that level, the introduction of a new version of a component, or a new parameterization of the library description.

One approach to the signalling problem would be to have each component keep track of the upper level parts that use that component. This is a costly and rigid approach. Alternatively, we developed a method to find the owners indirectly and implemented a "height first" algorithm to signal by flagging upper level components. Without going into further detail (See [Wie82]), we indicate some sample flagging times:

Piece	Level	Time
ALU	reg	0.13s
MUX,XOR	gate	0.9
INV	trans	7.1
TN	bottom	21.0

III. Dealing with Partially Instantiated Levels.

We have written procedures which will obtain all dependent instances at a lower level of some element. This requires that first all actually instantiated elements are retrieved, and then all other elements are obtained by expansion using the macroexpander. Actual instantiations are required to be stored whenever an element contains data not derivable from the expansion. This is typically true for elements at the extremes of an otherwise regular structure. For such elements the driving transistors, the connection configuration, or some layout detail may have to be specified.

IV. Conclusions and Future Work.

The initial approach continues to seem promising: DBMS-20 allowed us to implement both initial designs and modifications relatively quickly and easily. Coupled with the fact that performance has not been badly affected, this has allowed us to experiment productively. We expect to use these experiments to obtain the knowledge needed for eventually designing versatile and complete database systems for VLSI design. Our immediate goals and activities are the following:

1.) The database description needs to be expanded to deal with the signalling procedures. We are developing the schema further [Be81] to incorporate multiple representations and other desirable expansions using the same reloaded data and timing tests to reaffirm our theory that the overhead involved in using a commercial database is not prohibitive.

2.) We plan to investigate the passage of information using procedures.

3.) With the development of systems for maintaining equivalence between representations (for instance the logical and layout aspects of our design), we plan to test the "sideways" passage of information in order to handle different representations of the same design in a unified manner. The new schema is designed to facilitate this investigation.

4.) We are in the process of acquiring relational database management systems and wish to do similar experiments with those systems.

5.) We would like to develop further ways to manage queries that access partially instantiated, computable and/or retrievable elements.

References

[Be81] Beetem, Anne: "A Database Approach to Communication in VLSI Design: Part II - The New Schema", Stanford University, Department of Computer Science, Internal Report, 1981.

[Be82] Beetem, John: "Structured Design of Electronic Systems Using Isomorphic Multiple Representations", Stanford University, Electrical Engineering Department, PhD Thesis, January 1982.

[DBT71] *DBTG(71): Report of the CODASYL Database Task Group*, available from ACM, NY or IAG, Amsterdam, April 1971.

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

[Kat82] Katz, Randy H.: "A Database Approach for Managing VLSI Design Data", *The Nineteenth Design Automation Conference Proceedings*, June 1982.

[Ko81] Koenig, Wolfgang and Raymond A. Lorie: "Storage of VLSI Physical Designs in a Relational Database", IBM Research Laboratory, unpublished report, 1981.

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

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

[Sch79] Scheffer, Lou: "Database Considerations for VLSI Design", *Design Automation at Stanford*, ed. W. vanCleemput, Stanford CSL Technical Report No. 178, 1979.

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

[vC77] vanCleemput, W., T.C. Bennett, J.A. Hupp, and K.R. Stevens: "SPRINT: An Interactive System for Printed Circuit Board Design", *Proc. IEEE Computer Society Int. Computer Software and Applications Conf.*, pp. 113-119, November 1977.

[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 Ramez El-Masri: "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.

[Wie82] Wiederhold, Gio, Anne Beetem and Garrett Short: "A Database Approach to Communication in VLSI Design", *IEEE Transactions on Computer Aided Design of Integrated Circuits*, April 1982.

USING A RELATIONAL DATABASE MANAGEMENT SYSTEM FOR COMPUTER AIDED DESIGN DATA

Antonin Guttman
Michael Stonebraker

University of California
Berkeley

1. Introduction

There has been considerable interest in using relational database systems to manage data for computer aided design (CAD) systems. However, concerns have been expressed that database systems generally have been designed for other uses and might not be well suited to CAD. In addition, there is concern that relational DBMS's are too slow. The purpose of this paper is to report our experience with implementing a CAD application in a relational DBMS and comparing its performance with a non-database CAD package.

In Section 2 we discuss the database schema that was used and describe the structure of the special purpose CAD package. Then in Section 3 we indicate the experiments performed and give the results. Section 4 contains a discussion of the results. Lastly, Section 5 indicates areas where a relational DBMS was found to be inadequate as a support tool for CAD applications.

2. The Database Schema.

The application package benchmarked was KIC, a graphics editor for integrated circuit designs developed at Berkeley [2]. A KIC database consists of a collection of circuit "cells". Each cell can contain mask geometry and subcell references. A complete circuit design expands into a tree, with a single cell at the root and other cells, used as subcells, for the non-root nodes. Mask geometry can be associated with each node in the tree. During editing KIC stores a circuit in virtual memory on a VAX 11/780 computer.

Our INGRES schema reflects the above structure and has five main relations:

cell master (name, author, master id, defined)

cell ref (parent, child, cell ref id, t11-t32)

box (owner, use, x1, x2, y1, y2)

wire (owner, use, wire id, width, x1, y1, x2, y2)

polygon (owner, use, polygon id, vertnum, x, y)

In the cell master relation, name is the textual name given to the cell and author is the name of the person who designed it. Master id is a unique identification number assigned to each cell. It is used for unambiguous references to the cell within the database.

The cell ref relation describes subcell references. For example, if the cell "cpu" contains "register" as a part, then the cell ref relation contains a tuple in which parent is the identifier of "cpu" and child is the identifier of "register". T11 through t32 are a 3 X 2 matrix specifying the location, orientation and scale of the subcell with respect to the parent. This representation of a spatial transform is the one generally used in computer graphics [3].

The box relation describes mask rectangles. Owner is the identifier of the cell of which the box is a part. Use specifies the mask layer; e.g. metal or diffusion. X1 and x2 are the x-coordinates of the left and right sides of the box while y1 and y2 are the y-coordinates of the top and bottom.

A "wire" is a connected path of lines. Each tuple in the wire relation describes one line segment, giving the coordinates of its centerline (x1, y1, x2, y2) and its width. Wire id is a unique identifier for a particular wire. Owner and use mean the same as for the box relation.

A "polygon" is a solid shape with any number of vertices. One vertex is stored in each tuple of the polygon relation. X and y are the coordinates of the vertex, and vertnum orders the vertices (tuples) within one polygon.

3. The Experiment

For our test data we used circuit cells from two design projects at Berkeley. GORDA is a prototype layout for a switched capacitor filter [5], and dec0-1 is a part of the RISC VLSI computer [6,7]. The design descriptions were translated from KIC files and loaded into the INGRES database.

We programmed three representative CAD retrieval operations and measured the speed of our test programs compared to KIC performing the same operations. The test programs were written in C and made calls to the INGRES DBMS [4].

Top-level geometry retrieval. The first program retrieved the geometry data associated with a given circuit cell, not including geometry belonging to subcells. This is the first step in most CAD editing sessions.

Geometry retrieval with tree expansion. The second program retrieved geometry for all cells referenced in a fully expanded design tree. Geometry for lower level cells was transformed to its proper position relative to the root cell. This operation produces a plot of an entire design.

Retrieval by location. The third program retrieved top-level geometry that fell within a small area in the middle of a cell. This operation would be used to "window" a circuit.

The tables below show the results of the tests. _Geometry Tuples_ refers to the number of tuples representing geometry retrieved from the INGRES database during each operation. _CPU Time_ and _Elapsed Time_ were reported by the operating system. The _msec/Tuple_ figures are time divided by the number of INGRES tuples. The _Relative Time_ rows give the slowdown factor of our test programs relative to KIC.

Circuit	GORDA	
Geometry Tuples	592	
	KIC	INGRES
CPU Seconds	7.5	24.0
CPU msec/Tuple	13	41
Relative CPU Time	1	3.2
Elapsed Seconds	7.5	41
Elapsed msec/Tuple	13	69
Relative Elapsed Time	1	5.5

Test 1: Top Level Retrieval

Circuit	GORDA	
Geometry Tuples	12,779	
	KIC	INGRES
CPU Seconds	128.3	443.5
CPU msec/Tuple	10	35
Relative CPU Time	1	3.5
Elapsed Seconds	128.3	649
Elapsed msec/Tuple	10	51
Relative Elapsed Time	1	5.1

Test 2: Retrieval with Tree Expansion

Circuit	dec0-1	
Geometry Tuples	448	
	KIC	INGRES
Geometries in Window	87	85
CPU Seconds	.75	14.5
CPU msec/Tuple	9	171
Relative CPU Time	1	20
Elapsed Seconds	.75	33
Elapsed msec/Tuple	9	388
Relative Elapsed Time	1	45

Test 3: Retrieval by Location

4. Discussion of Results

In Tests 1 and 2, the factor of three increase in CPU time for INGRES can be attributed primarily to the fact that INGRES is a general-purpose DBMS while KIC includes only functions that it requires. Any DBMS used in place of special purpose code will show some decrease in performance. This cost must be balanced against the advantages of using a DBMS, e.g. simplification of application software, data independence, etc.

The elapsed time measurements show a larger performance difference, about a factor of five. This is mainly because INGRES geometry data is disk resident. KIC geometry data resides in virtual memory, and since the tests were run on a dedicated machine with ample main memory, KIC's data was actually in real memory.

KIC has a spatial bin structure that allows it to quickly isolate geometry in the window for Test 3. INGRES has no such access method and must perform a sequential search.

The new features suggested in the next section may narrow the performance gap. In addition, changing INGRES to manage a virtual memory database would dramatically improve performance.

5. New Features

During our experimentation we identified four features that should be added to a relational DBMS to facilitate CAD applications. We discuss them in turn.

5.1. Ragged relations

It would have been convenient for a field in a relation to repeat a variable number of times. In our database, for example, this would have allowed us to store data for a varying number of "boxes" in the _cell master_ relation, instead of in a separate _box_ relation. The _cell master_ relation could have been defined by

cell master (_master id_, _name_, _author_, _defined_, &(_use_, _x1_, _x2_, _y1_, _y2_)).

where the notation "&(...)" means that the group of five fields describing a box is repeated, once for each box. The _wire_, _polygon_ and _cell reference_ relations could be coalesced with the _cell master_ relation in a similar way.

This revised schema might be more natural for a user to understand. In addition, it would speed access to the geometry for a particular cell, since the geometry would be in one tuple rather than distributed across three relations.

We propose this extension under the name "ragged relations". One way to support ragged relations would be to first provide ordered relations, which has been suggested by Stonebraker and others [8,11], and then allow relations to nest, so that a field of a relation could be an ordered relation. We are investigating this idea.

A database with ragged relations or nested relations is not in first normal form, and reflects the hierarchical nature of the data. A language that provides just the standard relational data manipulation operations must be augmented before it can be used with ragged relations or nested relations. We are investigating such an augmentation.

5.2. Transitive Closure

Our second test program was required to expand subcells which could nest a variable number of times. This is an operation similar to a transitive closure and does not correspond to a single INGRES query. We have extended the syntax of QUEL with a * operator to facilitate such tasks. For example:

```
range of r is cell ref
range of t is tree
retrieve * into tree (cell = r.child)
        where r.parent = ROOT
        or r.parent = t.cell
```

Here the tuple variable _t_ ranges over the _tree_ relation, which is the result of the query. The _retrieve_ adds tuples to _tree_, and is repeated (with _t_ ranging over the new tuples) until there are no new additions.

Some database operations involving transitive closure are possible in Query-By-Example [12] and in the ORACLE system [13].

5.3. Access by spatial location

Many CAD programs, like our third test program, need to retrieve design data according to its spatial location. This test would have run much faster in INGRES if data could be classified according to a system of spatial "bins", i.e. by approximate location.

We are considering adding a spatial bin mechanism to INGRES as an extension of the secondary index facility. A two-dimensional array of bins is defined by a grid. A bin index is a file containing, for each bin, pointers to all the objects that might overlap that bin. Geometries in a particular area can be found quickly by looking in the index under the appropriate bins. For example, a bin index could be defined by:

```
index on box is boxbins
        (min(x1,x2) through max(x1,x2) by 10,
         min(y1,y2) through max(y1,y2) by 10).
```

This bin index is based on a grid of 10 X 10 squares. The min-max expressions specify the size of each box and define the set of bins it could overlap.

5.4. Unique identifiers

Codd and others [9,10] have suggested the usefulness of unique identifiers for database objects. They should be generated automatically by the database system and handled internally as a special type. In our application we were forced to manage our own unique identifiers for cells, wires, etc.

6. Conclusions

INGRES performs typical computer aided design retrieval operations considerably slower than special purpose programs. We have suggested a number of enhancements that could be made to a relational database system that would improve performance and make the system easier to use for CAD applications. We are continuing to look for additional mechanisms along the same lines.

7. Acknowledgements

This work was sponsored by NSF grant ECS-8007683-Wong-2/82 and by AFOSR grant 78-3596-Stonebraker/Wong-6/82.

8. References

[1] Held, G.D., M.R. Stonebraker and E.Wong "INGRES: A Relational Data Base System," _Proc. AFIPS 1975 NCC_, Vol. 44, AFIPS Press, Montvale, N.J.

[2] Keller, K. _Kic, A Graphics Editor for Integrated Circuits_, Masters Thesis, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, CA. June 1981.

[3] Newman, W.M. and R.F. Sproull "Principles of Interactive Computer Graphics," McGraw-Hill, N.Y. 1979.

[4] Woodfill, J. et al. "INGRES Version 6.2 Reference Manual," Memo No. UCB/ERL M79/43, Electronics Research Laboratory, University of California, Berkeley, CA. July 1979.

[5] The circuit cell GORDA is a layout prototype of a switched capacitor filter made by Jesus Guinea at Electronics Research Labs, University of California, Berkeley, CA.

[6] Patterson, D.A. and C.H. Sequin "RISC I: A Reduced Instruction Set VLSI Computer," Proc. Eighth International Symposium on Computer Architecture, May 1981, pp. 443-457.

[7] Fitzpatrick, D.T. et al. "A RISCy Approach to VLSI," VLSI Design, Fourth Quarter (October 1981), pp. 14-20. (Also appears in Computer Architecture News, March 1982.)

[8] Stonebraker, M. and J. Kalash "TIMBER: A Sophisticated Relation Brouser," Memo No. UCB/ERL M81/94, Electronics Research Laboratory, University of California, Berkeley, CA. December 1981.

[9] Codd, E.F. "Extending the Database Relational Model to Capture More Meaning," ACM Transactions on Database Systems, Vol. 4, No. 4, December 1979, pp. 397-434.

[10] Lorie, R.A. "Issues in Database for Design Applications," IBM Research Report RJ3176 (38928) 7/10/81.

[11] Lorie, R.A., R. Casajuana, and J.L. Becerril "GSYSR: A Relational Database Interface for Graphics," IBM Research Report RJ2511 (32941) 4/24/79.

[12] Zloof, M. "Query-By-Example: Operations on the Transitive Closure," IBM Research Report RC 5526 (Revised) (#24020) 10/29/76

[13] "ORACLE SQL Language - Reference Guide," Copyright by Relational Software Incorporated, October 1980.

DAVID: Design Aids for VLSI using Integrated Databases

Randy H. Katz
Computer Sciences Department
University of Wisconsin-Madison
Madison, WI 53706

Abstract: We briefly describe on-going research at the University of Wisconsin into ways of applying database technology for the management of computer-aided design data.

1. Introduction

Modern database systems have evolved to satisfy the requirements of traditional EDP applications. These have had either a transaction (short duration, touch little data) or report (read only, touch most of database) orientation. Database research has recently been directed towards the needs of the design automation community. Computer-aided design systems have a tremendous need for data management facilities unlike those required by the canonical debit-credit transaction [GRAY78].

Some of these are described here. A design data management system must support the hierarchical construction of a design object, to mirror the wide spread use of hierarchical design methodologies. For example, a microprocessor consists of a control part and a data path part; the latter consists of an ALU and a register file; and so forth. Further, a design exists in multiple representations, and the equivalence across representations should be enforced by the system. For example, a VLSI circuit design can simultaneously exist as a block diagram, a functional description, a transistor network, a stick diagram, a logic diagram, and a mask layout [MEAD80]. The system must manage the evolution of a design, as it progresses from release to release. Finally, since design objects are so complex, the system must support multiple simultaneous designers.

2. The Library Paradigm

A major complaint of the design automation community is that state-of-the-art relational database systems are too slow for real-time design applications (see [GUTT82]). We feel that it is perhaps unrealistic to expect a database system to interactively perform updates to a design. A more realistic approach is to extract data from the system, update it in memory over a long period of time, and return it when done. We call this model of usage the Library Paradigm.

A design database resembles a hierarchically structured "electronic filing cabinet," and serves as a centralized repository of a design information. A subpart of design (e.g., an ALU slice of the ALU of the Data Path of a Microprocessor) is checked-out for design activity in such a way that no concurrent designer is working on any of its subparts, nor on any parts that contain it. The mechanism is based on hierarchical locks with intentions. Designers can work in parallel as long as they are in parallel subtrees of the design hierarchy [KATZ82a]. When a design part is successfully checked out, the data associated with it is loaded into virtual memory data structures, which are manipulated directly by the applications program. The data structures are periodically checkpointed back into the database. When through, the design subpart is returned to the database (i.e., the locks are released). The system must now enforce integrity constraints up the hierarchy and across equivalent representations (e.g., has a cell now outgrown its "bounding box" of geometries?, has a representation been invalidated by a design change?).

By not updating the database directly, we avoid much of the overhead of entering and leaving a database system. Applications programs have more knowledge about the design data and how it is to be manipulated, and thus provide special purpose data structures for its representation. The database system still provides the sharable, centralized data repository, as well as a standard interface for data access.

3. Schema for Design Data Management

We choose to represent a design with three types of tables. The cell table describes each primitive and composite design cell. Associated with a unique cell-id is the cell's name, designer, and other control information. The composition table encodes the design hierarchy by describing how more primitive cells are placed and oriented within higher-level composite cells. Finally, the representation tables describe how primitive objects, such as transistors or geometries, are placed within cells. The details of the representation, as well as the encoding of orientations and placements, are determined by the programs that use the representation. For example, mask geometries are represented as boxes assigned to specific layers and placed on a Cartesian grid.

4. Work at Wisconsin

We have just begun to undertake a research program in design data management. Currently underway is an effort to implement a low-level database system based on System-R's RSS. The system, called WiSS (Wisconsin Storage System), is being designed and implemented in conjunction with Professor David DeWitt and a group of students led by Tobin Lehman and Richard Simkin. It provides facilities for sequential files, B-tree indexes, binary links (also implemented wih B-tree structures), long records, and file level versions. We are using the system as a research vehicle for a number of experimental projects in database management.

The features of WiSS most relevent to design data management are the latter two. Long records are those which are much longer than a single page. They are meant to handle non-traditional data, such as text and rasterized images, that frequently arise in the design environment. Our model of access for long records is the same as random access files. It is possible to seek to any location within the record and to read/write any length of data at that location.

We strongly believe that the system should support versions, alternatives, and releases. In the design environment, it is useful to keep old versions of data on-line for reference. Further, because of legal requirements, old released versions cannot be destroyed. Since it is not feasible to keep all versions on-line, the system must provide facilities for archive and restore, and mechanisms for differentially encoding old on-line versions. Alternatives are hypothetical designs, and are implemented as hypothetical databases [STON80, STON81]. System generated surrogates are used to identify records across versions. An implementation of differential files based on B-tree indexes over surrogates is described in [KATZ82b]. In addition, we have been investigating how optical digital disk technology can be used for version management (similar work appears in [SVOB81]).

5. Future Work

WiSS is essentially an access method with some features useful for the design environment. We intend to build DAVID, a design management system, on top of it. DAVID will keep consistent a design´s multiple representations and design hierarchy. It has been structured to keep small that part of the system that is specific to VLSI design. It should be possible to use DAVID for other design domains, as long as the objects are designed hierarchically.

We are also examining the role of a centralized design database "server" within a local network of designer workstations. Again, a paradigm related to a library appears appropriate. Design parts are checked out and later returned. They can be put "on-reserve" (readible, but not updatable). They can also be "held" for a designer if already checked out to someone else.

6. Conclusions

The design automation community has real needs for design management facilities, many of which are not currently supported by state-of-the-art database systems. A fruitful area for database research will be to study how database systems can be extended to fit into this new environment, providing the needed functionality with a reasonable (and hopefully small) additional overhead.

7. References

[GRAY78] Gray, J., "Notes on Database Operating Systems," in Operating Systems -- An Advanced Course, R. Bayer, R. M. Graham, G. Seegmuller, eds., Springer-Verlag, N.Y., 1978.

[GUTT82] Guttman, A., M. R. Stonebraker, "Using a Relational Database Management System for Computer Aided Design Data," Database Engineering Newsletter, this issue.

[KATZ82a] Katz, R. H., "A Database Approach for Managing VLSI Design Data," 19th Design Automation Conference, Las Vegas, NV, (June 1982).

[KATZ82b] Katz, R. H., T. Lehman, "Storage Structures for Versions and Alternatives," Working Paper in progress, (Mar. 1982).

[MEAD90] Mead, C., L. Conway, Introduction to VLSI Systems, Addison-Wesley, Reading, MA., 1980.

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

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

[SVOB81] Svobodova, L., "A Reliable Object-Oriented Repository for a Distributed Computer System," Proc. 8th Symposium on Operating Systems Principles, Pacific Grove, CA, (Dec. 1981).

ANNOUNCING

The 3rd International Conference on

DISTRIBUTED COMPUTING SYSTEMS

Miami/Ft. Lauderdale, Florida • October 18-22, 1982

SPONSORED BY
IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC

in Cooperation with
Information Processing Society of Japan (IPSJ)
Institut National de Recherche en Informatique et en Automatique (INRIA)

SCOPE

The scope of this conference encompasses the technical aspects of specifying, designing, implementing, and evaluating distributed computing systems. In such systems, there is a multiplicity of interconnected processing resources able to cooperate under system-wide control on a single problem, often with minimal reliance on centralized procedures, data, or hardware. The location of computing resources may span the spectrum from physical adjacency to geographical dispersion. The topics of interest include the following aspects of distributed computing systems:

- ☐ **SYSTEM AND HARDWARE ARCHITECTURE, INCLUDING SIMD**
- ☐ **DECENTRALIZED CONTROL, EXECUTIVES, AND OPERATING SYSTEMS**
- ☐ **DISTRIBUTED DATABASES**
- ☐ **LOGICAL AND PHYSICAL INTERCONNECTION NETWORKS**
- ☐ **SOFTWARE ENGINEERING AND PROGRAMMING LANGUAGES**
- ☐ **SURVIVABILITY, RELIABILITY, AND FAULT TOLERANCE**
- ☐ **SPECIFICATION, VERIFICATION, AND VALIDATION**
- ☐ **DESIGN METHODOLOGIES**
- ☐ **VLSI-BASED SYSTEMS**
- ☐ **ANALYSIS, MODELING, AND MEASUREMENT**
- ☐ **COMPUTER COMMUNICATION**
- ☐ **APPLICATIONS, INCLUDING SIMULATION**

The conference will include technical presentations, panel discussions, tutorials & exhibits.

General Chairman
H. J. Siegel
Purdue Univ.
School of EE
West Lafayette, IN 47907
Program Chairman
Carl G. Davis
Ballistic Missile
Defense Advanced
Technology Center
Tutorial Chairman
K. H. Kim
Univ. of South Florida
Exhibits Chairman
Edith W. Martin
Deputy Undersec. of Def. (Res. & Adv. Tech.), Pentagon, Rm 3E 114
Washington, D.C. 20301
Standing Committee Chairman
Charles R. Vick
Auburn Univ.
Awards Chairman
T. Y. Feng
Ohio State Univ.
Treasurer
Duncan H. Lawrie
Univ. Illinois - Urbana
Local Arrangements
H. Troy Nagle, Jr.
Auburn Univ.
Publications Chairman
Ben Wah
Purdue Univ.
Professional Societies Liaison
S. Diane Smith
Univ. Wisc. - Madison
Publicity Chairman
Bill Buckles
Univ. Texas - Arlington

TUTORIALS

Complementing the Conference there will be two full days set aside for tutorials. The following have been tentatively selected:
"Pragmatic View of Distributed Processing," ***by Ken Thurber***
"Microcomputer Networks," ***by Harvey Freeman***
"Fault-Tolerant Computing," ***by Vic Nelson and Bill Carroll***
"Decentralized Control," ***by Bob Larson, Paul McEntire and John O'Reiley***

PROGRAM COMMITTEE

Bob Arnold (Honeywell)
Geneva Belford (U. Ill.)
Bill Carroll (U. Texas-Arlington)
Glenn Cox (General Research Corp.)
Barry Gilbert (Mayo Clinic)
J. C. Huang (U. of Houston)
Robert Keller (U. Utah)
Annette Krygiel (Defense Mapping Agency)
Bill McDonald (Systems Development Corp.)
Vic Nelson (Auburn U.)
Peter Ng (U. Missouri)
Dan Siewiorek (Carnegie-Mellon U.)
Harold Stone (U. Mass.)
Ken Thurber (Architecture Tech. Corp.)
Larry Wittie (SUNY/Buff.)
Ken Batcher (Goodyear)
Bharat Bhargava (U. Pittsburgh)
Doug DeGroot (IBM)
Clarence Giese (Dept. of Army AIRMICS)
Bob Heath (U. Kentucky)
Lana Kartashev (U. Nebraska)
Jack Lipovski (U. Texas-Austin)
Mike Liu (Ohio State U.)
John Musa (Bell Labs)
Chong Nam (Systems Control, Inc.)
C. V. Ramamoorthy (UC/Berkeley)
Azriel Rosenfeld (U. Maryland)
Steve Smoliar (Schlumberger-Doll Research
Joe Urban (U. Southwestern La.)

International Associate Chairpeople

Helmut Kerner, Austria
C. M. Woodside, Canada
Paul Pearson, England
Gerard Le Lann, France
Herbert Weber, Germany
Mariagiovanna Sami, Italy
Hideo Aiso, Japan
J. Wilmink, Netherlands
R. C. T. Lee, Taiwan
Leah J. Siegel, U.S.A.

CONFERENCE LOCATION

Diplomat Hotel in Hollywood, Florida near the international airport at Miami. Beachfront resort with tennis, golf, swimming, shopping, and boating.

Additional Program Committee Members will be chosen by the International Associate Chairpeople.

If you wish to receive a copy of the Advance Program for the Third International Conference on Distributed Computing Systems, clip and mail this coupon to: Harry Hayman, IEEE Computer Society, 1109 Spring Street, Suite 201, Silver Spring, MD 20910.

NAME ______________________ EMPLOYER ______________________

STREET ____

CITY ____________ STATE ____________ ZIP ____________ COUNTRY ____________