The EVE VLSI Management Environment[1]

Technical Report No. CENG 89-06

by

Hamideh Afsarmanesh[2]
Esther Brotoatmodjo
Kwang June Byeon
Alice Parker

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-0781

Address All Communication to:
Dr. Alice Parker
University of Southern California
Department of EE Systems
SAL 300 Mail Code 0781
Los Angeles, CA 90089-0781
(213)743-5560
FAX: (213)745-7284

[1] This research was supported by the Department of Air Force, Department of Army, Department of Navy contract no. N00039-87-C0194 and the National Science Foundation under Computer Engineering Grant no. DMC-8310774

[2] Also on faculty at California State University, Dominguez Hills

The EVE VLSI Management Environment[1]
Technical Report No. CENG 89-06

by

Hamideh Afsarmanesh[2]
Esther Brotoatmodjo
Kwang June Byeon
Alice Parker

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-0781

Address All Communication to:
Dr. Alice Parker
University of Southern California
Department of EE Systems
SAL 300 Mail Code 0781
Los Angeles, CA 90089-0781
(213)743-5560
FAX: (213)745-7284

[1] This research was supported by the Department of Air Force, Department of Army, Department of Navy contract no. N00039-87-C0194 and the National Science Foundation under Computer Engineering Grant no. DMC-8310774

[2] Also on faculty at California State University, Dominguez Hills

submitted to IEEE International
Conference on Computer-Aided Design (ICCAD-89)
November 6 - 9, 1989; Santa Clara, CA

The EVE VLSI Information Management Environment

Abstract

This paper describes the implementation of a complex object-oriented database system constructed using a physical database which is record-oriented. The data model includes behavioral and timing information, and is comprehensive, covering the entire design process. The use of a commercial record-oriented database allowed rapid implementation of the system, supported schema changes, and provided other database control support functions. Using databases to support high-level synthesis activities is novel. A description of such a comprehensive database has not appeared previously in the literature.

1 Introduction

The management of VLSI design data is difficult. Designers need to manage large quantities of data and descriptive information about the data (called meta-data). The data is dynamic with frequent accessing and modification [7,15,18]. High-level design systems have unique representational requirements that further complicate the management of design information. Some of these complexities lie in representing behavioral information, relationships between behavioral and structural information, and how timing is handled. Also, the data must represent both the behaviors of specified designs and implemented designs which may be different.

The ADAM Advanced Design AutoMation synthesis system under construction at the University of Southern California (USC) has all these information management problems. This paper describes a solution to the problems, beginning with abstract models for design representation, and ending with implementation in a commercial database.

At USC we have developed a novel Experimental VLSI/CAD information management Environment (EVE) which addresses these problems. EVE has a number of attributes which make it unique. It is the first integrated database which contains sufficient behavioral information to support high-level data path, control and interface design. It allows designs to be described with a number of models, each structured hierarchically, and supports incremental design and backtracking. EVE gracefully supports schema design and change, and high-level object-oriented operations to access and manipulate data, and also provides data sharing and concurrency control.

The physical design and implementation of EVE began in the fall of 1986 with the aim of constructing an object-oriented database framework which could effectively handle VLSI design data for the ADAM synthesis system [14]. However, a decision was made to implement EVE with SunUNIFY, a relational database, because of SunUNIFY's proven reliability, efficiency, maintainability, and portability, and because of the potential for a rapid implementation. A major task in this implementation was to overcome the limitations of using a record-oriented database system to implement an object-oriented framework.

The rest of the paper is organized as follows. The remainder of Section 1 describes related research. Section 2 includes an overview of the ADAM system and also describes the abstract model of representation used in ADAM, called the Design Data Structure (DDS). Section 3 describes the 3 Dimensional Information Space (3DIS) modeling technique which was used to produce a formal database specification schema for the DDS. Section 4 describes the design of EVE using a commercial database, SunUNIFY.

1.1 Related Research

A number of systems have been proposed and constructed for management of VLSI design data. Most of the past literature on digital design databases reported on the use of file systems to store raw design data. File-based database systems are costly to modify and maintain since the description of the stored data is completely hidden in the application programs and the users' minds.

Record-oriented database models, such as the relational data model, have been applied to several VLSI/CAD design environments (e.g. [19]). However, these database models do not support the representation of unformatted information, e.g. text or behavioral data, and are not easily used by VLSI designers and programmers who need to build, use and maintain their own databases. Furthermore, since the modeling constructs of record-oriented database models do not directly correspond to identifiable entities or concepts in application environments, the information that represents a design entity is typically spread among many constructs.

The suitability of object-oriented database models as tools to help in the construction and use of design databases is now being examined [2,5,8,10,12,16]. Object-oriented database models are based on more-generalized, higher-level views of application environments that allow data to be related and interpreted in various ways. Also, the modeling constructs of these models are defined so as to naturally correspond to the concepts and activities of application environments. Therefore the modeling and specification of complex information such as VLSI design data can be achieved more successfully by object-oriented database models.

2 The Design Automation Environment

2.1 The ADAM System Overview

USC's ADAM synthesis system [11] provides a high-level, unified system for VLSI design that allows automatic design of register-transfer level data paths. A separate design-for-testability system automatically modifies register-transfer designs so that they become testable.

Concurrently with the development of EVE, a testability database for ADAM, called Cbase [6], was designed on top of VBASE, an object-oriented database system [4]. The parallel efforts allowed us to study both approaches. Plans are now underway to merge the two databases.

2.2 Design Data Structure

The conceptual schema used in the ADAM system is based on the Design Data Structure (DDS) [9,13,14,15]. The DDS is a unified representation for design information. The DDS representation is interpretable for simulation purposes, supports description of families of designs, allows the design effort to be partitioned in a well-defined manner, corresponds directly to formal models of the synthesis process, supports easy tool interfacing, and supports incremental design, user interaction, and backtracking. In addition, it provides the necessary semantics to represent synchronous and asynchronous events, timeouts, delays, pipelining, and process priorities.

2.2.1 The Component

The fundamental structure of the DDS conceptual schema is the **component**. A component can represent either a specification, a design in progress, or a member of the design library, with identical representational formalisms. This facilitates the task of design verification and validation (e.g., testing equivalence).

In the initial stages of design, the target component contains primarily dataflow and timing information; in the later stages it contains more structural and physical information. The original specification is preserved for documentation and verification purposes.

2.2.2 Models of a Component

The Design Data Structure uses separate *Models* for the dataflow behavior, timing and control behavior, logical structure, and physical structure. This allows both control and timing and dataflow behavior to be modeled accurately.

The dataflow model describes the data transformation operations performed by the component. Its primitive elements are **operations** and **values**. Operations represent data transformations; values represent data passed between operations. The timing and control model describes time-domain and branching behavior of the component. Its primitives are **ranges** and **points**. A range represents a time interval during which an operation can take place; points represent infinitesimal 'events', which are partially ordered. The structural model describes the schematic diagram of the component. Its primitives are **modules** and **carriers**. A module represents a schematic block, gate, transistor etc.; a carrier represents a schematic wire. The physical model describes the layout, position, size, packaging and power dissipation of the component. Its primitives are **blocks** and **nets**, which represent layout cells and interconnect respectively.

For example, the OEM component '74181', which is a 4-bit TTL ALU slice, has a dataflow model with add, subtract, AND, and OR operations, which represent its data transformations. This component has a timing and control model that describes its propagation delays for various combinations of control inputs. It has a schematic diagram that either consists of a box with connection points or a gate-level diagram. It also has a physical description that describes its being packaged in a 14-pin DIP.

Syntactically, the four models are virtually identical. Each model may be hierarchically decomposed, but the hierarchies may not be isomorphic. This is illustrated by a processor example where Fetch and Execute can both be decomposed into more detailed behavior. The CPU and memory can be decomposed likewise. However, both Fetch and Execute make use of CPU and memory. Thus, a family of designs can be represented with identical dataflow behavior, variations in control flow and timing, and completely different structures. By separating different aspects of design information into models, instead of lumping behavior, structure and timing into a single representation at each 'level' of design, consistency checking is simpler, and redundancy is reduced.

2.2.3 Relationships between Models

Items in one model are explicitly related to those of other models by bindings. There are two basic types of bindings: **operation** bindings, which relate dataflow elements to structural elements and time ranges, and **realization** bindings, which relate structural elements to physical elements. For example, a binding can occur between an addition in the dataflow, the ALU in the structural model which is used to implement it, and the time range during which the addition is performed.

Bindings correspond to the 0-1 variables in a formal model of the synthesis process, supporting verification as well as synthesis. The bindings also support cleaner partitioning of the design effort, since correspondences between design models are explicit. Incremental design is possible with the simple addition of bindings, and backtracking is supported by deletion of bindings. Users can interact by viewing, creating, or deleting bindings.

3 An Overview of the 3DIS

A formal schema for the DDS was produced using the 3 Dimensional Information Space (3DIS) [1,3], an extensible, object-oriented framework for information management. It is specifically oriented to support the database requirements for data-intensive information system applications in which: (1) information objects of various levels of abstraction and modalities must be accommodated, (2) descriptive and structural information (meta-data) is rich and dynamic, and (3) users must be able to design, manipulate, and evolve their own databases. In the 3DIS, data and the descriptive information about data are handled uniformly in an extensible framework. The 3DIS provides the database user with a geometric representation of data, which forms a basis for understanding, defining and manipulating databases. Several prototype implementations based upon the 3DIS have been designed and implemented, and are in experimental use [1,3].

3DIS databases contain collections of interrelated objects where an object represents any identifiable information fact in an application environment. For example, a component **Adder11**, a component's attribute **Designer-Names**, a string of characters **Two-Bit-Adder**, a component's type (meta-data) **In-House-Component**, and a procedure **Insert-a-Component** are all modeled uniformly as objects in a homogeneous framework. What distinguishes different kinds of objects in a 3DIS database is the set of structural (meta-data) and non-structural (data) relationships defined on them.

The 3DIS model supports *atomic*, *composite* and *type objects. Atomic* objects represent the symbolic constants in databases. These objects cannot be decomposed into other objects. Strings of characters, numbers, Booleans, text, messages, audio, and video objects, as well as behavioral (procedural) objects are examples of atomic objects. *Composite* objects describe (non-atomic) entities and concepts of application environments. The information content of these objects can be interpreted by the 3DIS system through their decomposition into other atomic objects. Mapping objects are a special kind of composite object that describes relationships between other objects. Mapping objects may be decomposed into their domain and range type objects and their inverse mapping objects. *Type* objects contain the descriptive and classification information of a database. Every type object is a structural specification of a group of atomic or composite objects.

Basic relationships among objects are defined through the three fundamental abstraction mechanisms of *generalization, classification*, and *aggregation.* Generalization represents type/supertype relationships by relating a type object, e.g. **In-House-Component**, to a more general type object, e.g. **Component**. Classification represents instance/type relationships by relating an atomic or composite object, e.g. **Adder11**, to its generic type object(s), e.g. **In-House-Component**. Aggregation represents mapping/type relationships by relating a mapping object, e.g. **Has-Designer-Names** or **Has-Realization-Bindings**, to a type object, e.g. **In-House-Component**.

The 3DIS model has also been extended to accommodate other kinds of abstractions that are useful in VLSI design applications like the recursive definition of VLSI components. This allows us to define components as being composed of other components.

The 3DIS geometric representation (information cube) organizes database objects (both structural and non-structural) and their interrelationships graphically. The geometric representation is discrete and orthogonal, and is assumed to be located in the positive octant of a 3-D space. The three axes in the space represent the domain (**D**), the mapping (**M**), and the range (**R**) axes. All database objects appear on both D and R axes, but the M-axis holds mapping objects (a subset of all objects) only. Relationships among objects are represented by specific, explicit points in this geometric space, defined via their three coordinates (domain-object, mapping-object, range-object). Figure 1 illustrates a perspective view of the geometric representation of a 3DIS database example. In this figure, **Full-Adder-1** and **Full-Adder-2** are instances of the type object **Single-Node**, while they are also the model constituents of **Adder11-Dataflow**.

Geometric components of the 3DIS representation framework such as points, lines, and planes, are units of access and play meaningful roles in encapsulating database information. For instance, in Figure 1, the vertical line emanating from the object **Adder11-Dataflow** contains points which correspond to all mappings defined on that object. Similarly, the orthogonal plane passing through the same object contains the information about

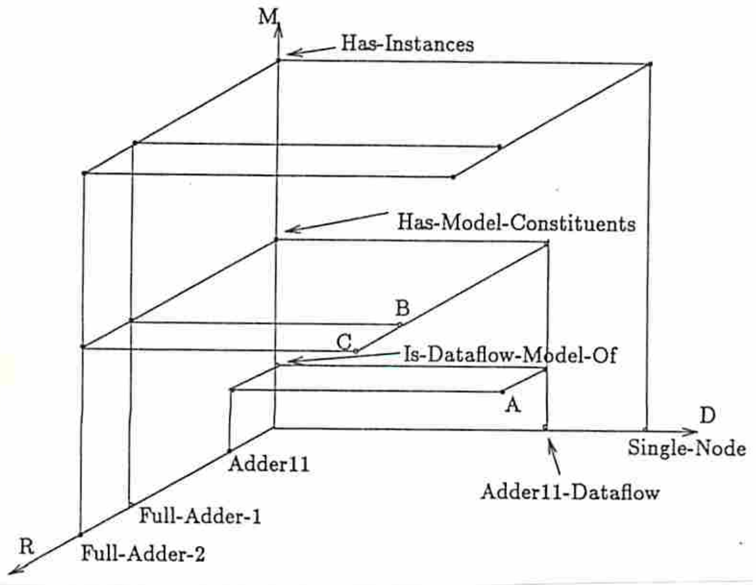


Figure 1: Perspective view of information in a 3DIS database

all objects directly related to **Adder11-Dataflow** through mappings. For example, this plane contains the points **A**, **B**, and **C** that are represented by the triples (**Adder11-Dataflow, Is-Dataflow-Model-of, Adder11**), (**Adder11-Dataflow, Has-Model-Constituents, Full-Adder-1**), and (**Adder11-Dataflow, Has-Model-Constituents, Full-Adder-2**), respectively.

4 The Eve Implementation Strategy

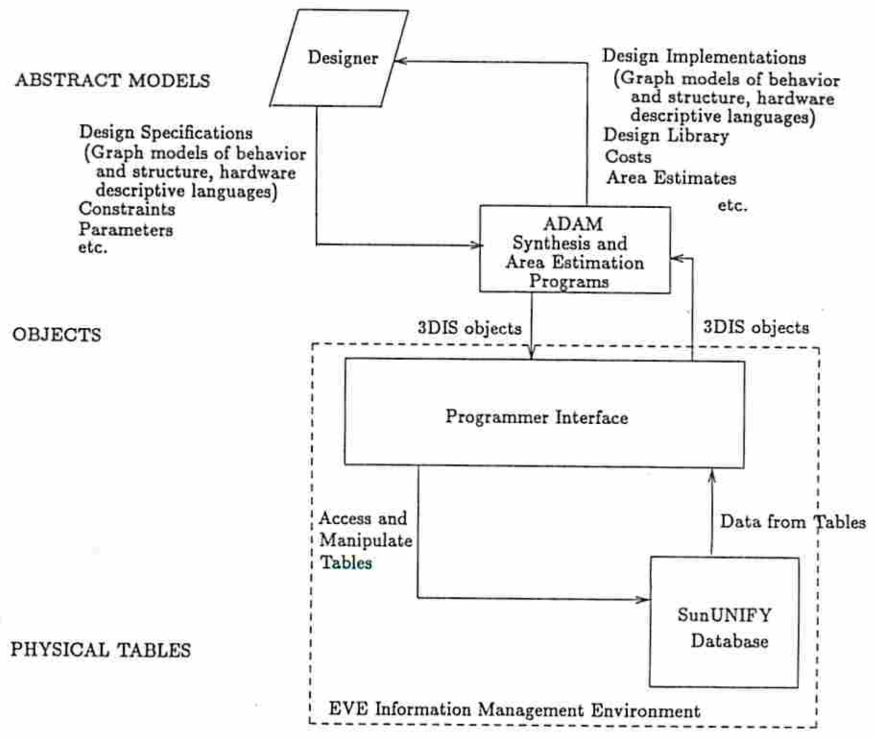
4.1 Overview

The 3DIS framework was implemented using SunUNIFY, and then customized to represent the DDS conceptual schema. The main goal of EVE implementation was to produce an operational database and a user interface that (1) appeared to application programmers as a 3DIS object-oriented environment, (2) could be implemented in a short period of time, and (3) could meet the requirements on semantic expressiveness, storage space, and speed. Initially it was thought that application programmers would use SQL and query the SunUNIFY relational database directly, an approach that proved tedious, slow, complicated and confusing. Therefore, a much more powerful and simple-to-use mechanism was developed. An object-oriented programmer interface was introduced which converted 3DIS objects directly to and from the tables (relations) in SunUNIFY. Figures 2 and 3 show the data transformation process and representation of data in the ADAM system. All application programs deal with abstract models of design information. For instance, a program might deal conceptually with a data flow graph as a list of operations and a list of arcs. However the programmer interfacing the synthesis program to the EVE's programmer interface must be concerned with operation objects, value objects, and objects which connect operations to values. The programmer interface takes the application programs' requests to create, delete, and modify objects, and translates these requests into accessing and updating operations on database tables.

4.2 The EVE Database Implementation

The development of the EVE database has been based on two modeling techniques, DDS and 3DIS. The conceptual schema of EVE is the DDS. The formal database specification for DDS was constructed using the 3DIS framework. Therefore, the database contains a collection of interrelated objects, where an object represents any identifiable information fact in the VLSI design environment representable using the DDS.

In the formal database specification, complex descriptions of VLSI design data (meta-data) are captured by types. In the physical implementation every type is implemented as a table. Instances of a type are recorded in that type's table entries. Mappings defined on a type are represented by attributes in the table representing that type.



ABSTRACT MODELS
Designer
Design Implementations (Graph models of behavior and structure, hardware descriptive languages)
Design Library
Costs
Area Estimates
etc.
Design Specifications (Graph models of behavior and structure, hardware descriptive languages)
Constraints
Parameters
etc.
ADAM Synthesis and Area Estimation Programs
OBJECTS
3DIS objects
3DIS objects
Programmer Interface
Access and Manipulate Tables
Data from Tables
SunUNIFY Database
PHYSICAL TABLES
EVE Information Management Environment

Figure 2: Data Transformation Process in ADAM

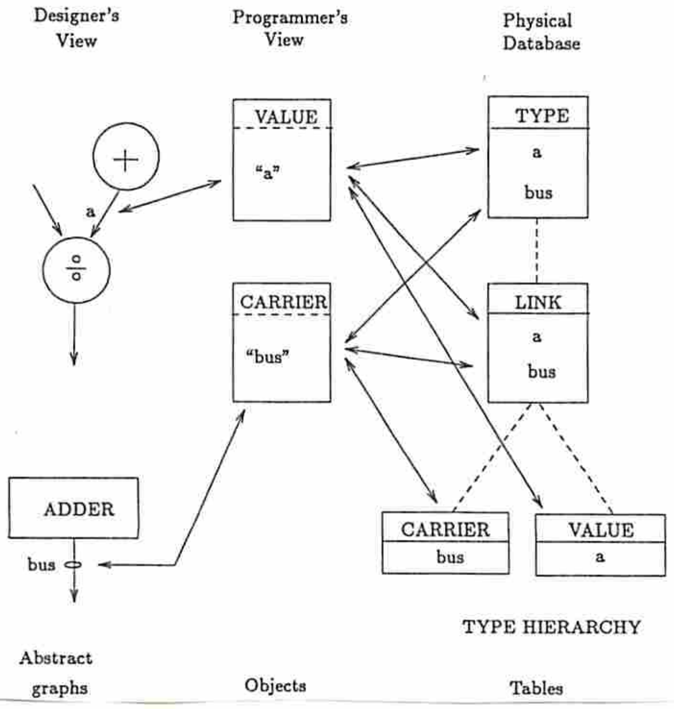


Figure 3: Representation of Data in ADAM

Several capabilities offered in the 3DIS framework were not supported by the record-oriented SunUNIFY system and needed to be incorporated into the EVE environment. Some of those capabilities are described by four special tables in EVE called TYPE-HIERARCHY, OBJECT-CLASSIFICATION, MAPPINGS, and INVERSES. The first table, TYPE-HIERARCHY, represents the *generalization* mechanism by specifying the supertypes for each type. This table captures the complex relationships among types in EVE database. The second table, OBJECT-CLASSIFICATION, describes the *classification* mechanism by specifying the types to which each object belongs. This table captures the fact that every instance of a type is also an instance of all its supertypes, and is also useful for finding instance information. Using the vertical partitioning technique[17], the information describing each object is stored in several tables representing the object type and all its supertypes. Each type object may reference other type objects via some mappings. The third table, MAPPINGS, represents the *aggregation* mechanism by specifying the single-valued and multi-valued mapping objects associated with each type. Thus the tables TYPE-HIERARCHY, OBJECT-CLASSIFICATION, and MAPPINGS support access of instance and mapping information across the tables. The fourth table, INVERSES, contains inverse mapping information. Each mapping in 3DIS has an inverse (i.e., A refers to B and B refers to A). Since SunUNIFY does not support inverse attributes, this table had to be constructed. One common use of the table INVERSES occurs when the designer needs to find out which objects reference a given object.

EVE also provides specific capabilities for handling multi-valued mappings, (multiple) inheritance of mappings, and recursions. Types with multi-valued mappings have been implemented in one of two ways. One way is to use a distinct attribute for each value (e.g., **DesignerName0**, **DesignerName1**) in the table representing that type. The other way is to refer to a relationship table which is a more explicit and efficient form and saves storage space. The values of the multi-valued mapping are represented by entries in the relationship table. In the EVE database implementation a relationship table is used for each multi-valued mapping with five or more values.

Each type inherits all mappings from its supertypes except in special cases where the mappings are redefined for the type itself. Maintaining this inheritance of all mappings is important for database integrity and efficiency and saves a significant amount of storage space. The mappings inherited by a type can be found in the EVE database by looking for the supertypes of that type in the table TYPE-HIERARCHY and finding the mappings defined on those supertypes in the table MAPPINGS. Since each type may have more than one supertype in EVE, a multiple inheritance problem may happen when more than one supertype has mappings with the same names. This problem is handled in the EVE implementation by arbitrarily choosing one supertype and inheriting those same-named mappings from that supertype. This approach is suitable for the EVE database, whose current schema is relatively fixed (therefore the future occurrence of this problem is rare), even though it is not a general solution for the multiple inheritance problem.

The recursion abstraction defined in 3DIS has also been incorporated in the database schema of EVE. This abstraction is useful to handle recursively (hierarchically) defined VLSI design information [3]. For example, VLSI components are recursively defined by their subcomponents.

4.3 The Interface Implementation

A small set of simple but functionally powerful object-oriented primitives has been developed in EVE to support the basic data definition and manipulation operations defined for the 3DIS. These primitive operations allow designers or application programs to add new objects to a database, delete existing objects, create and remove relationships among objects, retrieve both objects and relationships among objects, and access meta-data information. In addition, several high-level operations which invoke the primitive operations have been provided to designers, and are much easier to use than the primitive operations. They are presented in a menu format. When an operation is selected from the menu, EVE provides a template for users to fill. After the user completes entering data into the template, EVE processes the operation. The high-level operations allow users to create a composite object and all relationships that it can participate in, remove an object and its relationships with other objects, and generate and destroy relationships among objects, as required.

Two object-oriented interfaces using ISL (Information Sub-Language), the user interface language designed for the 3DIS[1] were implemented in EVE. One interface was implemented using the C language and SunUNIFY's SQL; the other using the C and PROLOG languages, with a PROLOG-to-SunUnify interface supplied by Quintus PROLOG. The latter has turned out to be much more efficient, since it directly accesses tables instead of going through SunUNIFY's SQL query processor. For example, the interface using the C and PROLOG languages takes about 0.1 second to access 80 objects, and 4 seconds for 600 objects[1]. This speed is more than four times faster than that of the other interface. A final interface is being constructed which combines the performance advantages of C with the efficiency of direct database access.

The EVE database currently contains 78 tables with a capability of storing a moderately-

[1]600 objects denotes 600 constituents of a VLSI component, such as operations, links, pins, nets, and so on.

sized design. Its size is approximately 4.0 M bytes including both interfaces (0.8 M bytes).

5 Conclusions

The EVE information management environment provides an object-oriented database and a user interface for the ADAM VLSI design application, using relational database technology. Several limitations of record-oriented database environments were tackled and overcome. Many object-oriented modeling constructs have been incorporated into EVE including generalization, classification, and aggregation mechanisms, inverse mappings, multi-valued mappings, multiple inheritance of mappings, and recursion abstraction. A number of SunUNIFY's software features, such as the easy construction, manipulation, and updating of the database schema, have been invaluable in the development of EVE. Additional capabilities such as improving the speed of accessing the information have been achieved by directly accessing the tables in SunUNIFY. The use of high-level languages such as PROLOG supports the construction of inference engines to access and make decisions based on database information.

While many object-oriented features are contained in the EVE environment, the implementation of several other features remains an open issue. Future work on EVE involves adding database operations (methods) to types, design versions, and design constraint rules. Also, a future goal is the development of a better object-oriented interface to serve the specific needs of engineers/designers and application programs. One such need that is not yet addressed by EVE is the graphical display of the design's structure.

New implementations of commercial object-oriented databases may achieve better performance to compete with the power of existing relational systems. If so, it is possible and straightforward to replace SunUNIFY with a more powerful physical object-oriented database.

References

[1] H. Afsarmanesh. *The 3DIS: An Extensible Object-Oriented Framework for Information Management.* Technical report CRI-85-21, Department of Computer Science, University of Southern California, Los Angeles, CA 90089-0782, October 1985.

[2] H. Afsarmanesh, D. Knapp, D. McLeod, and A. Parker. An Extensible, Object-Oriented Approach to Databases for VLSI/CAD. In *Proceedings of the International Conference on Very Large Databases*, VLDB Endowment, August 1985.

[3] H. Afsarmanesh and D. McLeod. The 3DIS: An Extensible Object-Oriented Information Management Environment. *To appear in ACM Transactions on Office Information Systems*, 1989.

[4] T. Andrews and C. Harris. Combining Language and Database Advances in an Object-Oriented Development Environment. In *Proceedings of the Conference on Object-Oriented Programming Systems, Langusges, and Applications*, pages 430–440, ACM, 1987.

[5] D. Batory and W. Kim. Modelling Concepts for VLSI CAD Objects. *ACM Transactions on Database Systems*, 10(3):322–346, September 1985.

[6] M. Breuer, W. Cheng, R. Gupta, I. Hardonag, E. Horowitz, and S. Lin. Cbase 1.0: A CAD Database for VLSI Circuits Using Object Oriented Technology. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 392–395, ACM-IEEE, November 1988.

[7] R. Davis, H. Shrobe, W. Hamscher, K. Wieckert, M. Shirley, and S. Polit. Diagnosis Based on Description of Structure and Function. In *Proceedings of the National Conference on AI*, pages 137–142, AAAI, 1982.

[8] K. Dittrich, A. Kotz, and J. Mulle. *An Event/Trigger Mechanism to Enforce Complex Consistency Constraints in Design Databases.* Technical Report, Institut fuer Informatik II, Universitaet Karlsruhe, West Germany, 1985.

[9] J. Granacki. *Unverstanding Digital System Specifications Written in Natural Language.* PhD thesis, University of Southern California, December 1987.

[10] D. S. Harrison, P. Moore, R. L. Spickelmier, and A. R. Newton. Data Management and Graphics Editing in the Berkeley Design Environment. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 24–27, ACM-IEEE, 1986.

[11] R. Jain, K. Küçükçakar, M Mlinar, and A. Parker. Experience with the ADAM Synthesis System. In *Proceedings of the 26th Design Automation Conference*, ACM-IEEE, June 1989.

[12] R. Katz. A Database Approach for Managing VLSI Design Data. In *Proceedings of the 19th Design Automation Conference*, ACM-IEEE, 1982.

[13] D. Knapp. *A Planning Model of the Design Process.* PhD thesis, University of Southern California, December 1986.

[14] D. Knapp, J. Granacki, and A. Parker. An Expert Synthesis System. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 419–424, ACM-IEEE, September 1983.

[15] D. Knapp and A. Parker. A Unified Representation for Design Information. In *Proceedings of the Conference on Hardware Description Languages*, IFIP, 1985.

[16] D. McLeod, K. Bapa Rao, and K. Narayanaswamy. Information Modelling for CAD/VLSI. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Jose, California, May 1983.

[17] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou. Vertical Partitioning Algorithms for Database Design. *ACM Transactions on Database Systems*, 9(4):680–710, December 1984.

[18] J. Nestor and D. Thomas. Defining and Implementing a Multilevel Design Representation With Simulation Applications. In *Proceedings of the 19th Design Automation Conference*, pages 740–746, ACM-IEEE, 1982.

[19] S. Wong and W. Bristol. A CAD Database. In *Proceedings of the 16th Design Automation Conference*, ACM-IEEE, 1979.