A DATABASE MANAGEMENT SYSTEM FOR A VLSI DESIGN SYSTEM

Gwo-Dong Chen and Tai-Ming Parng

Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan 10764, R.O.C.

ABSTRACT

A special purpose database management system for VLSI design environment is presented. Besides supporting design data management and tools integration, the system provides lots of facilities for supporting fast development of efficient and powerful VLSI CAD tools. This system could simplify the task and reduce efforts of implementing an integrated VLSI design system.

1.INTRODUCTION

To construct a VLSI design system, a design database management system is needed to integrate tools and manage design data [9,10,11,12]. With the rapid advances in VLSI and software technologies, the design tools, design styles, and design methods of VLSI circuits keep on evolving. Being able to accommodate new tools quickly and provide a comprehensive set of high-performance tools is crucial to the life of a VLSI design system. Therefore, the design database management system should provide not only an environment for integration of design tools and management of design data but also a tool development environment so that new tools can be easily developed and gracefully integrated into the design system.

The conventional database management system is not applicable because of their inadequate performance and difficulty of use [12]. Many systems and researches on VLSI database systems have been reported [3,7,8,13]. Most of them focused only on providing environment for design data management or tool integration. Recently, object oriented database management systems have been proposed. It is regarded that an object oriented database system is more appropriate for VLSI design environment than a record oriented database system [6,18]. However, the need of a database management system to serve as a good VLSI tool development environment has not been addressed yet.

To attack the problem of developing VLSI design systems, we have implemented a new special purpose database management system. In addition to supporting tool integration and design data management, the system supports a tool development environment which eases the task of VLSI design tool construction, integration and maintenance.

The tool development environment offers convenient programming facilities for tasks of efficient manipulation of in-memory design data, fast access of 2D objects, in-memory transaction control, display and graphic output control, and user interface management. By the aid of these in-memory data management facilities, tasks of tool development can be greatly simplified and the tool developing time can be shortened, while the performance of tools built on the system is still comparable to that of tools built on a file system.

For easy tool integration and effective on-disc design data management, the design data storage system of our system is based on the relational model [4] and equipped with database reorganization and multiple representations supports.This feature makes design systems built on our system can be flexible, extensible and tailorable.

2. SYSTEM OVERVIEW

The design database management system (DDMS) is implemented on Pyramid 90x and VAX 8200 computers as well as several SUN workstations. These machines form a distributed design environment via an Ethernet local area network. The system implemented with the C language is built on the network file system (NFS) of UNIX.

The DDMS includes two sub-systems, the in-memory data manager (IMDM), and the design data storage system (DDSS). The DDSS is an object oriented storage system based on the relation data model. It supports hierarchical design data storage, mul-

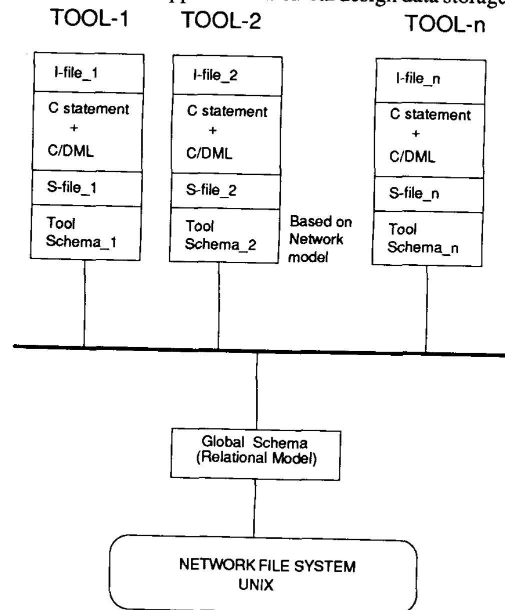

Figure-1 Tool Programmer's view of the system

This work was supported by the National Science Council R.O.C. under Grant NSC 76-0201-E002-03.

CH2540-3/88/0000/0257$01.00 © 1988 IEEE

tiple representations, versions and backups. Furthermore, it supports dynamic reorganization of design data. Only object oriented operations are supported by it. The data manipulation can only be done in IMDM. Note that the term "object" used throughout this paper means "design object". The object oriented programming environment such as Small-talk is not provided by this system.

With the advent of low cost and high performance workstations equipped with millions bytes of main memory, it is possible to keep all active design data of a design transaction in the main memory. The IMDM provides network model [4] database facilities for in-memory data. The facilities include a C like Data Manipulation Language (C/DML), 2-D access paths support, in-memory transaction control, graphic display manager, and user interface generator.

The tool programmer's view of DDMS is shown in Figure-1. To develop a tool on DDMS, the tool designer should do the following tasks. First, define a tool schema based on the global schema of the design data storage system. The schema describes the data and data structure required. Second, make an S-file according to the tool schema. This file describes how the design object is to be displayed in this tool. Third, use the C language embedded with C/DML to implement this tool. Fourth, specify the user interface requirements of this tool in an I- file. The precompiler of DDMS will generate a tool program which includes all the database faccilities according to these files. All the database facilities would be included in this tool program. The generation process of a tool program is shown in Figure-2.

3. DESIGN DATA STORAGE SYSTEM

The design data storage system is a hierarchical object storage system with version and backup control mechanism. It provides functions of object creating, navigating, loading, saving and deleting. To develop tools on it, the tool designer needs only define a tool schema. A design object manager will be generated according to the tool schema and included into the tool. Users of tools directly interact with the design object manager to handle design object in DDSS. The data structure conversion between disc (DDSS) and memory (tool) is done by the design object manager.

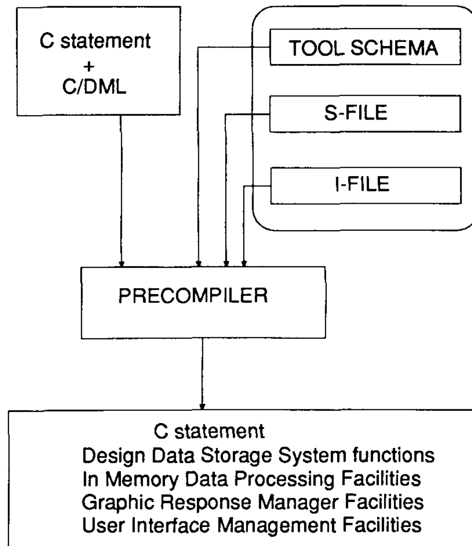

Figure-2 tool generation in ddms

Basically, the design data storage system is a relational database system [4]. However, data of an object are treated as a unit. A design object is described by a set of record types (relation). A relation of an object is stored in a file. The multiple representations and multiple data view of tools are supported by the view mechanism of the relational model. Because the tools are developed on the same schema, the design data are shared by them. Therefore, the design data and design changes can be propagated through the database.

3.1 Support for multiple representations

A design object in DDSS is described by the global schema which contains descriptions for the abstract behavior (abstract view), the register transfer level behavior (RTL view) and the net-list of all objects in the database. The net-list has a logic view, a transistor view and a layout view. A view is a subset of the global schema. These views share the same interface. The conversion and consistence maintenance between views are done by tools.

The global schema is designed according to the requirements of all tools built on the system. Figure-3 shows part of the global schema. The record types (relations) for *net, net_impl, net_imp2,*

```
/* description of composite data type used in RECORD description; the
syntax is similar to C language */
STRUCT BOX { int x1; int y1; int x2; int y2;}
STRUCT LINE { long x1, y1, x2, y2;}
STRUCT LOC { long x,y;}

RECORD net
/* net description shared by schematic, layout and cell editors*/
      { long net_id;
      char name[DB_NET_NAME_LEN];
      int pin_no; /*no. of pin involved */
      int bus_id; /* id of bus it belongs */}

RECORD net_impl   /* for SCHEMATIC EDITOR */
            /* net description used only by schematic editor */
      { LOC name_loc;
            /* LOC is STRUCT LOC declared above; */
            /* name_loc is variable */
            /* coordinate of point to display the net name */
            /* in schematic sheet */
      int net_mem;}

RECORD net_imp2   /* for LAYOUT EDITOR    */
      { char flag;} /* has this net been rounted   */
                /* in layout sheet */

RECORD net_imp3   /* for CELL EDITOR      */
            { LOC name_loc; }

RECORD wire1      /* for SCHEMATIC EDITOR */
            /* wire description for schematic editor */
            /* a net is represented by a list of lines in schemetic diagram*/
      { long net_id;
      int layer; /* horizontal wire:0, vertical:1 */
      LINE length;
      unsigned wire_id; }

RECORD wire2      /* for LAYOUT EDITOR    */
                /* a net is implemented by a list of boxes */
      {  long net_id;
      int layer; /* poly or diffusion or metal layer */
      BOX size;}

RECORD wire3      /* for CELL EDITOR      */
      { long net_id;
      INST_ID inst_id;
      int layer;
      BOX wire;}
```

Figure-3 Part of the GLOBAL schema

net_imp3 are used to describe nets. The record type *net* is used to propagate data among schematic, layout and cell editors. For example, a chip designer uses a schematic editor to design a circuit. The schematic editor generates data of record types *net, net_imp1* and *wire1*. Then, he can use a layout editor to layout this circuit. The layout editor reads in the data of the *net* record type and generates data of record types *net_imp2* and *wire2*. If he develops two layout diagrams for this schematic sheet, only the *net_imp2* and *wire2* have two copies of data.

An entity, such as a net, may be described by a set of record types. It is uniquely identified by a system generated id (SRID). This SRID is included in all record instances of the record types describing this entity. A SRID is a chronological sequence number, which is assigned when the entity instance is generated. Record types which identify the same entity share the same chronological count. For example, the record types *net, net_imp1, net_imp2* and *net_imp3* share the same chronological count; their SRID uniquely identify an entity such as a net; therefore, related records of the same SRID can be combined. Besides, the use of SRID makes data reorganization easy. For example, when a new tool which needs new description data for the nets is introduced, a new record type *net_imp4* can be added to the global schema without impacting the existing tools.

3.2 Support for multiple versions and backups

A version plane is shown in Figure-4 as appeared in [13]. Each version has a separate set of data in DDSS. Each version has a list of backups. The backup facility allows a designer to "roll back" the older design data during iterative activities of design progression. To get such capability, a negative differential file concept [10] is adopted. When a new backup is created, the negative differential data items are stored into history file. The data of the newest backup is stored. However, the record type files which have not been modified remain unchanged.

Unlike the version server of [13], the equivalence plane is merged with the version plane. The version plane is divided into layers. The sibling nodes are alternatives. The child node is the derivative of the parent node. Nodes of lower layer inherit data from its ancestor nodes. When an object is derived across layers, the hierarchy should be reconstructed for the next layer. This scheme is explained in Figure-4.

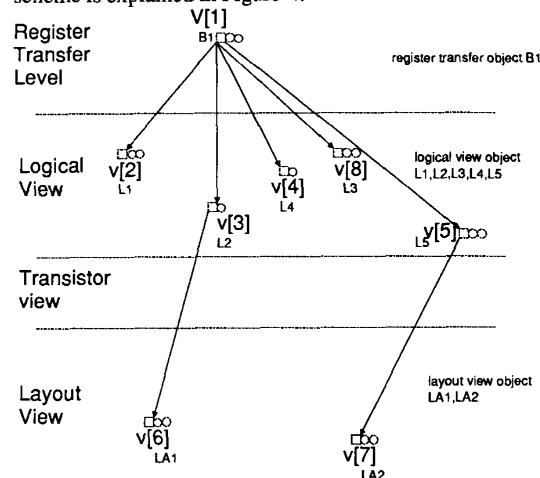

Figure-4 An example of version plane
Objects L1,L2,L3,L4,L5 are derived from B1
Object LA1 is derived from L2
Object LA2 is derived from L5
Each object has a list of backup
V[6] represents a composite object composed by LA1,L2,B1

```
$ TOOL_NAME : layout_editor;/* the tool schema is defined */
                /*based on global schema in figure-3 */
    $ACCESS_PATH HOR_TILE
    { $TYPE htile;
    $RECORD WIRE;}
            /* A horizontal corner stitch is built for record type*/
            /* WIRE. The "htile" is the name of access path */
            /* type. The "HOR_TILE" is the name of this*/
            /* access path in this tool. In this tool, the */
            /* HOR_TILE is used to identify this path */
$RECORD NET
/* This NET record type is composed of record*/
/* types net and net_imp2 in global schema */
    { $RECORD net: $READ; /* Only read operation is allowed */
                /*for this record type */
    $RECORD net_imp2: $WRITE; } /*Read and write operation*/
                /* is allowed for this record type */
    $KEY = net_id; /* The key of this NET record type is */
                /* net_id. This field is in the net record */
                /* type in global schema. The NET record*/
                /* is unique identify by the key value */
$RECORD WIRE
    { $RECORD wire2: $WRITE;
    int wire_mark;} /* this field is in memory temporary data */
    $KEY = $ALL; /* All fields combined used as key */

$SET NET_WIRE $LINK=$P2O+$ASCLINK+$DESLINK;
                /*LINK describe the type of data structure to implement
                this set. P2O is pointer to owner set type. */
                /* ASCLINK is ring structure set type. Members of
                this set is ordered by the value of net_id field */
                /* All of P2O, ASCLINK and DESLINK is provided
                for this set */
    {$OWNER NET;
    $MEMBER WIRE;
    $COND ( $OWN.net_id == $MEM.net_id );
                /* All the WIRE record ($MEM MEMBER) and
                NET record ($OWN OWNER) with the smee
                net_id construct an instance of set
                NET_WIRE */
    }
```

FIGURE-5 Part of the tool schema of a layout editor

By this organization, validation process can be simplified. After two views have been verified to be equivalent, the equivalence flag can be propagated along the derivation path. For example, the equivalence flag of L2 and B1 in V[3] can be propagated to V[6]. Thereafter, validation process can focus on a single node. Besides, chip designers can find all the layout versions which have same function from the version plane. For example, if a function specification B1 is given, we can find two layout objects LA1 and LA2 from the version plane.

3.3 Reorganization support

When a record type in the global schema is added, deleted or modified, all the tools which use this record type are recompiled to match the newest global schema. At the same time, a program for reorganizing the database is generated. After that, these recompiled tools will automatically call this program when the design data of the old global schema is loaded. This reorganization operation is done at the time when the object is being loaded. Therefore, old design data and design tools can be still available after the global schema had been modified.

3.4 Support for in-memory database

To modify a design object, data must be loaded into in-memory database first. The data manipulation is done on in-memory database. After a design transaction has been done, the modified object is then saved back to DDSS. The DDSS supports a list of object oriented functions for object loading, saving, creating and deleting

Tool schema definition

Each tool has its tool schema. The tool schema is the in-memory data structure of the tool. Tool programmers use set structure of the network data model [4] to define his data structure. A tool schema is a subset of the global schema extended with some in-memory temporary schema. An example tool schema is shown in Figure-5. A record type in the tool schema may be an in-memory temporary record or a set of record types in DDSS with some in-memory temporary field. The tool schema plays the corresponding role as the subschema in the network data model. A set is defined by a join predicate of relational query languages. If both owner and member record types are in the global schema, the set structure would be constructed automatically at the object load time. For example, the net_wire set in Figure-5 would be constructed automatically after an object had been loaded. 2D access path can also be declared in the tool schema and be constructed at the object loading time. As in Figure-5, a horizontal tile corner stitch [14] is adopted.

Memory consideration

The memory management of this system is handled by the UNIX operating system. The in-memory data are actually stored in the virtual memory space. However, retrieving data from the virtual memory space is still faster than from disc files. DDSS provides DB_unload and DB_savetemp functions to release memory space for latter use. If an object can not be loaded into the main memory,the tool designer can selectively load some of required record types. The tools of this system can not continue in case of loading failure.

4. IN-MEMORY DATA MANAGER

The in-memory data manager (IMDM) can be viewed as an in-memory network database management system. Before the chip designer can actually manipulate an object, the object must be fetched into the main memory. Therefore, the overhead of slow disk access and buffer management of traditional disk-based database system can be avoided.

The in-memory data manager consists of a set of program generators for programs of design data manipulation, graphic display management and user interface management. The tool programmer uses C/DML to manipulate design data, an interface description language to specify the user interface, and a sheet description language to describe the format of graphic display. The in-memory data manager will generate programs according to these description and include in them the facilities of graphic display control and in-memory transaction control.

```
/* TOOL NAME: layout_editor  */
/*  This tool is developed based on the tool schema in Figure-5*/

## RECORD NET     *net_rec; /* declare a pointer to record net*/
## RECORD WIRE    *wire_rec;
....
##find_in_2D(HOR_TILE, point_find, x1, y1, wire_tile);
     /* x y1 --- a point input from mouse  */
     /*  use corner stitch to locate a wire_tile in corner stitch */
     /* structure by the given point */
##assign(HOR-TILE, wire_tile, wire_rec) ;
     /* the wire_tile is data in corner stitch */
     /* the assign function find the corresponding wire_rec */
     /*  in database  */
##find_in_set_member(NET_WIRE,,wire_rec,,net_rec);
     /* find the net which contains the wire via set NET_WIRE*/

/* delete wires belonged to this net */
##find_in_set_member(NET_WIRE, WIRE, FIRST,,,net_rec,
           ,  wire_rec)
while (db_found)  /* a C statement */
     { ## disconnect (NET_WIRE,,wire_rec);
     ## delete (WIRE,,wire_rec);
     ## find_in_set_member(NET_WIRE,WIRE,NEXT,,
            ,net_rec,,wire_rec);
```

Figure-6 Part of the sourceprogram of a layout editor

4.1 Data manipulation language (C/DML)

The C/DML commands, whose syntax is similar to that of C function calls, are similar to those of a data manipulation language of the network model [4] . The currency of a record is used as a C-language pointer. An example of C/DML program is shown in Figure-6. This program is based on the tool schema of Figure-5.

4.2 Recovery facilities

Whenever a change is made by issuing a modification commands to the in-memory data, a record containing the old and new values of the changed item is written to a special data set called log. Thus, if a failure occurs, the design database can be restored to a correct state by using the log to undo all unwanted changes.

4.3 2D access path support

There are three two dimensional access (2D) paths supported in C/DML: corner stitch [14], quad tree [2] and multi-dimensional binary tree (KDTREE) [16].

Different 2D access paths are quite different in their data structure, algorithms, and even supported functions. To integrate the various access paths, two constraints are enforced to bind these different access methods to form a unified interface for 2D access paths.

First, all the access paths should support INIT, CREATE, DELETE, and CLEAR functions for access path construction. INIT initializes an access path; CREATE creates an access path element; DELETE deletes an access path element; and CLEAR frees all the access path elements.

Second, POINT_FIND, AREA_SEARCH, AREA_ENUMERATE, READ_BOUNDARY are the standard

```
TOOL: layout_editor
SHEET : LAYOUT
            /* define a layout sheet for tool schema of figure 5 */

$RECORD WIRE %redraw_path:(HOR_TILE);
            /* redraw_path specify the access path which is used to
            locate the data of WIRE record type when this sheet
            is redrawed. */

     { SHAPE size;
            /* size is the field name of WIRE record     */
            /* SHAPE describes the size field is
            to be displayed by the following description */
       %color : COLOR_RED,
            /* color of this shape */
       %condition:"($.layer == HOR_VW),
            /* $ is  the currency of record type WIRE   /
            /* the condition to display this shape */
            /* is wire.layer == HOR_VW        */
            /* HOR_VW is a C variable of tool program */
       %lstyle:0,
            /* define line style of bounding box    */
       %fstyle:0,
            /* define fill type */
       %mode:0,
            /* display mode: replace, xor, not .... */
       %box:("$.size.x1","$.size.y1",
       "$.size.x2","$.size.y2);
            /* type of this SHAPE is box */
            /*coordinate of box is wire.size.x1 ..wire.size.y1*/
            /*     wire.size.x2, wire.size.y2 */
     }
ENDSHEET
```

Figure-7 An example sheet description for the automatic graphic response system

functions of C/DML for 2D data searching. POINT_FIND locates a design object by its location; AREA_SEARCH checks if there is any design object intersecting with a rectangular window; AREA_ENUMERATE reports all the object intersecting with a rectangular window; and READ_BOUNDARY returns the object space in the 2D plane.

With the find_in_2D command of C/DML, tool programmers can make use of these 2D access paths easily. Declaring a 2D access path is as easy as declaring a C-language variable. Thus, it is easy to catch up with evolutions of 2D access paths.

4.4 Graphic response manager

The graphic response manager is designed based on a scope/sheet model. A sheet is a VLSI diagram or a subset of a VLSI diagram which satisfies some qualification. For example, a schematic diagram can be defined as a sheet. The metal layer of the layout diagram can be defined as a sheet too. A scope, which is called a window in other systems, is a subrange of a sheet. A VLSI diagram can be defined as a set of sheets such as the metal layer layout sheet, the diffusion layer layout sheet, etc. These sheets can be overlapped if sheets are declared as transparent and placed in the same scope. An example is shown in Figure-8. The screen has 3 scopes, scope-1, scope-2 and scope-3. A schematic sheet is displayed on scope-1 and a layout sheet is displayed on scope-3. The layout sheet on scope-3 has a box which defines the scope displayed on scope-2.

Sheets are defined based on the tool schema and stored in an S- file. A sheet description example is shown in Figure-7. The basic graphic types are POINT, BOX, BOX45, GTEXT, SHAPE, LINE, CIRCLE, ARC and POLYGON. Tool designers should define (1) what graphic type that a record is to be displayed, (2) the graphic attribute of this graphic type, and (3) how to get the coordinates of this graphic type from the database.

In Figure-7, the wire record is defined to be displayed as a box. The $ means currency of the wire record. The values of x1, y1, x2 and y2 fields are the coordinates of the box. The %color, %lstyle, %fstyle and %mode are graphic attributes for a box to be drawn. The %condition is the condition to draw this shape.

The graphic response manager is organized as three levels, scope/sheet operations, draw sheet functions, and scope/sheet primitives.

The scope/sheet primitives are procedures to display a line, a box, a circle, or other graphic primitives on the desired sheets in appropriate scopes. The management for multiple windows and multiple world coordinates are handled by these primitives. These primitives include functions which tell the tool designer information of the mouse. The information includes (1) the coordinates in the screen, (2) what sheet it locates, and (3) coordinates in the sheet. Therefore, tool designer needs not care about the multiple windows and multiple coordinates for both graphic input and output.

The draw sheet functions are generated from the S-file, which includes data for describing the sheets. These functions call the scope/sheet primitives to draw the sheets. By the aid of sheet/scope primitive, only the coordinates of the sheet needs to be tackled.

The scope/sheet operations include place_sheet, define_scope, and move_scope operations. As in Figure-8, a schematic sheet is put into scope_1 by "place_sheet" operation and a layout sheet is put into scope-3. Then, a scope is defined by "define scope" operation on scope-3 for scope-2. It defines a region of the sheet on scope-3 to be displayed on scope-2 of the screen. User can use "move scope" operation in scope-3 to change the content of scope_2. These scope/sheet functions are automatically included in tool programs by the DDMS precompiler.

With the above facilities, the tool designers need only focus on using C/DML to manipulate in-memory data. The task of displaying the graphic is done by the procedures generated according to data in the S-file. These procedures are embedded into the tool program during precompiling. When an insert, delete or modify operation is done on in-memory data, it will trigger appropriate procedures to modify the graphic on the screen.

4.5 User interface generator

To design the user interface of a tool on DDMS, the tool designer only has to describe the user interface specification in an I-file. The user interface management program can be generated from the I-file.

The tasks needed to be done for user interface management are screen management and command handling, which are described in the following.

Screen Management

A screen is divided into areas for scopes, panels and ttys. The scopes are used to display design diagrams, sheets. The sheet/scope is managed by the graphic manager. The area of panels are used for menu driven command handling. The ttys are used as emulator for tty. A tty is used to provide a way to do I/O for the C language standard I/O.

The commands for screen management are listed as follows:
-SCOPE(lbx, lby, dx, dy, show_on/off)
-PANEL(type, lbx, lby, fore_color, back_color, [,dx, dy])
-TTY(lbx, lbx, dx, dy)

The lbx, lby, dx, dy are used to define a box on the screen. A screen division example is shown in Figure-8.

Command handling

The commands' hierarchy and associated operations can be described as a set of state transition hierarchy. A hierarchical pop-up menu called "walking menu" is generated in the panel region to ease command selection for chip designers. A command handling program is generated by the interface generator. An example of a command hierarchy is described in the following.

```
STATE: OBJECT
SUBSTATE:
    STATE: CREATE  @{ . ..        @}
    STATE: LOAD @{ load(object); @}
    STATE: SAVE @{ save(object); @}
    STATE: TRAVEL  @{ ....... @}
    SUBSTATE:
        STATE: SET @( set(object); @}
        STATE: GO @{ go(object); @}
        SUBSTATE
            STATE: HOME_OBJ @{ ..... @}
            STATE: PARENT  @{ parent(object) @}
        E NDSUB
        STATE: SHOW_OBJ .
    ENDSUB
ENDSUB
```

A walking menu generated from the above description is shown in Figure-8. In this example, the user selects the TRTAVEL fuction and GO to the PARENT object.

CONCLUSION

We have described a database management system designed for developing VLSI design systems. The features of this system are summarized as follows:

*The DDMS is composed of two subsystems. The design data storage system (DDSS) is designed for easy tool integration and effective data management. The in-memory data manager (IMDM) is designed for fast tool development.

*By providing an in-memory data manager, which is based on the network model and supports 2D access paths, tools developed on our system are efficient.

*The graphic response manager together with the user interface generator provides a graphic manipulation environment. Tool designers are relieved of most of the programming efforts for graphic display output.

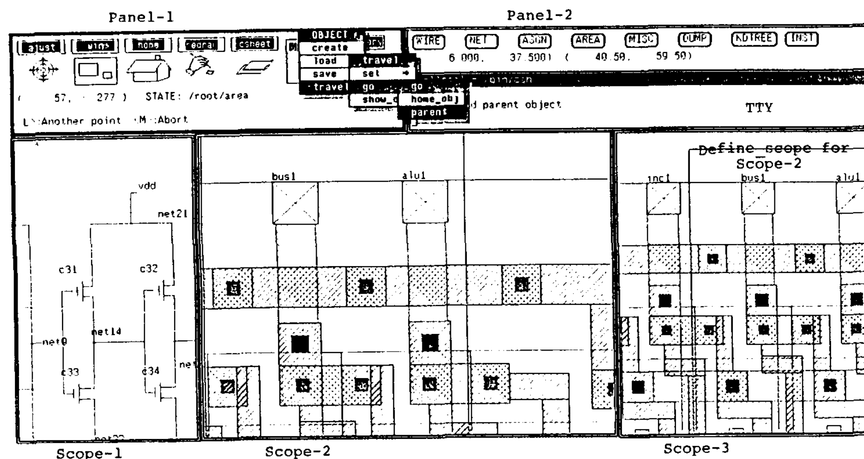

Figure-8 A screen dump of the cell editor built on the system

*The tasks of design data manipulation, graphic display and display control, in-memory transaction control, and user interface generation are handled by the system; therefore, the task of tool development can be greatly simplified.

*The design data storage system supports hierarchical construct, multiple representations, multiple versions and backups features of VLSI design objects. This makes DDMS a good environment for design data management.

*With the use of the relational model and support of reorganization in the design data storage system, tools developed on our system are independent of underlying storage structure. New tools can be gracefully integrated.

The first version of this system had been in operation since April, 1987. Many tools such as logic simulation tool, cell editor, layout editor, schematic editor have been implemented on or ported to it. These tools form a VLSI design system without programs for data extraction and data translation. The average manpower to develop or port a tool is four man-months. Except for the time required for object loading, these tools are friendly and efficient. Further performance improvement, analysis and measurement are being done.

ACKNOWLEDGEMENT

The authors would like to thank the graduate students, especially Ming-Huei Hong, Shih-pin Hsu, and Bing-Yie Hsieh, for their significant contrtibution to the implementaion of this system.

REFERENCE

[1] J. Banerjee and W. Kim: "Supporting VLSI Geometry Operations in a Database System", *Proc. of IEEE Int. Conf. on Computer Design* (ICCD'86), 1986, pp.409-414.

[2] R. L. Brown: "Multiple Storage Quad Tree: A Simpler Faster Alternative to Bisector List Quad Trees" *IEEE Trans. on Computer-Aided Design*, Vol. CAD-5, No. 3, 1986, pp.413-419.

[3] K. C. Chu, et. al.: "VDD -A VLSI Design Database System" *Proc. of ACM SIGMOD conference on Engineering Design Applications*, 1983

[4] C. J. Date, *"An Introduction to Database System,"* 3rd ed., Addison-Wesley, 1981.

[5] C. M. Eastman, "Database Facilities for Engineering Design," Proceedings of IEEE, Vol. 17, No.1, Jan. 1983, pp. 1249-1263.

[6] S. Heiler, U. Dayal, J. Orenstein and S. R. Sproull, "An Object Oriented Approach to Data Management: Why Design Databases Need It" *Proc. ACM/IEEE 24th Design Automation Conference*, 1987, pp. 335-340

[7] L. Hollar, B. Nelson, T. Carter, R. A. Lorie: "The Structure and Operation of a Relational Database System in a Cell Oriented Integrated Circuit Design System" *Proc. of 21st Design Automation Conference*, 1984

[8] C. Jullien, A. Leblond and J. Lecourvoisier: "A Database Interface for an Integrated CAD System" *Proc. ACM/IEEE 23rd Design Automation Conference*, 1986, pp.760-767

[9] R. H. Katz, "Managing the Chip Design Database," *IEEE Computer*, Dec. 1983, pp. 26-36.

[10] R. H. Katz, T. J. Lehman, "Database Support for Versions and Alternatives of Large Design Files," *IEEE transaction on software engineering*, March 1984, pp. 191-200.

[11] R. H. Katz, "Computer-Aided Design Databases," *IEEE Design & Test*, Feb. 1985, pp 70-75.

[12] R. H. Katz, *"Information Management for Engineering Design,"* Spring_Verlag, Berlin, 1985.

[13] R. H. Katz, M. Anwaruddin and, E. Chang, "A Version Server for Computer-Aided Design Data," *Proc. ACM/IEEE 23rd Design Automation Conference*, 1986, pp.27-33.

[14] J. K. Ousterhout, "Corner-Stitching: A Data Structuring Technique for VLSI Layout Tools," *IEEE Trans. Computer-Aided Design, Vol. CAD-3*, No. 1, Jan. 1984, pp.87-100.

[15] T. M. Parng, G. D. Chen, M. H. Hong, Y. C. Chen, B. Y. Shiech, S. P. Hsu, "IDAF - A Framework for Developing VLSI Design Automation Systems," *Proc. of Int. Symp. on VLSI Technology, Systems, and Applications*, Taipei, Taiwan, ROC, May 1987, pp.138- 142.

[16] J. B. Rosenberg: "Geographical Data Structures Compared: A Study of Data Structures Supporting Region Queries", *IEEE Trans. Computer-Aided Design*, Vol. CAD-4, No.1, Jan. 1985, pp. 53-67.

[17] D. G. Severance and G. M. Lohman, "Differential files: Their Application to The Maintenance of Large Database," *ACM Trans. on Database System*. Vol. 1, No. 3, Sep. 1976.

[18] S. Weiss et al "DOSS : A storage system for design data," *Proc. ACM/IEEE 23rd Design Automation Conference*, 1986, pp.41-47.