Simulation Standard

Connecting TCAD To Tapeout

A Journal for CAD/CAE Engineers

Expert Expanded with Client-Server Project/Library Management

The ***Expert*** layout processor v.2.0 introduces a client-server system for managing large projects in multiuser design environments. This system is a significant improvement and enhances productivity by providing concurrent access to the same design data by several designers while maintaining data integrity and ensuring security for intellectual property.

This article describes basic guidelines for the design and implementation of client-server subsystems in ***Expert.***

Figure 1. An example of client-server Library Management in ***Expert***

General Requirements for Multiuser Data Access System

Management Capabilities

A concurrent development environment requires a complex administrative system that provides integration of different user and data management capabilities. Among these capabilities are:

- Version Control - the management of incremental design changes by different users
- Modification Control - preventing unauthorized / unintentional modifications
- Data Security - all of the above plus the prevention of data losses
- Design Configuration Management - the ability to create a new design by reusing existing components
- Distributed Working Environment - linking the activities of several designers into a team. The individual must be able to work independently and easily share his results with the team
- Workspace Configuration Management - the ability to provide compatible workspace configurations for different users
- Accessibility - smooth flow of information flow between several copies of the system running at different workplaces

Architecture

A client-server architecture is adopted in the ***Expert*** layout editor to provide access to design data in a multi-user environment.

Under a client-server architecture an application is split into two parts: the client part and the server part. The front end for a user is the client part, which processes and displays data. The server part handles the kernel application database and performs functions that support concurrent access to application data. Client and server parts may be executed on separate computers in the network.

Continued on page 2....

INSIDE

SILVACO INTERNATIONAL

A network interface provides communication between server and user (client) processes. It allows programs running on remote sites to access, modify, and store data on the server site. It performs the formatting of database queries, data conversion and transfer.

The core of the user interface comprises communication software running various communication protocols, such as TCP/IP. Network drivers perform data transfer across the network.

The basic components of a client-server application are data structures and processes. All application data structures are assumed to reside in the core computer memory. A client-server application has two basic types of processes: user processes and kernel processes. Kernel processes run on the server computer. User processes run on client workplaces.

The client parts of an application request data from the server. The server verifies user privileges, assigns a buffer for the user, fetches the requested data, and transfers it across the network.

A database management system must be able to handle huge amounts of data in a multiuser environment with simultaneous access to the same data. It must have a means of protection against unsanctioned and incorrect access, i.e., it must provide a robust mechanism for security, restriction, and monitoring of database access.

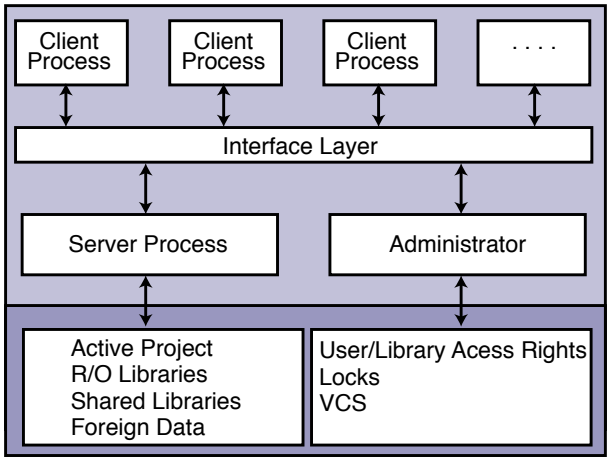

Figure 2. Interaction of processes and data in the client-server framework of ***Expert.***

One of the most important requirements for a multiuser system is controlled concurrent access to data that maintains data integrity. Without proper control of concurrent access data modification cannot be guaranteed to be performed properly. All user requests must be processed as independently as possible. A client-server system addresses these issues using mechanisms such as locking and multi-versioned data.

Locking mechanisms ensure the following:

* users that perform data fetching do not have to wait until the users that modify the same data complete their activity
* users that modify data do not have to wait until other users retrieving the same data complete their activity
* users that modify data have to wait only for the completion of activity of those users modifying the same data

Locking mechanisms use rollback structures (with previous versions of data) created during data modification, and until modification is completed, other users read data as it was before modification.

A client-server system must provide restricted access to data based on assignment of privileges. These restrictions consist of system security and data security.

System security verifies user identity and privileges. It allows access to data only for those users who have specific access rights assigned by the system administrator.

Data security assures that users with data access rights do not interfere with each other operations.

A multiuser system must provide the means for user management of simultaneous work. The system must have a list of users. When data is accessed, the system must verify user identity to prevent unauthorized access. Every user may create and process objects for which he has access rights.

Accounting Subsystem

A client-server system must provide the means for proper identification of both users and data.

User Identification

The User Table is a list of User Account records.

Field 1:	Login Name
Field 2:	Full Name
Field 3:	Access rights
Field 4:	Password

Table 1. User Account Record.

After the initial installation, the system has a single record in the user table for the 'Administrator' user, with a known password. Administrators can create and modify any user account records.

Data Identification

Data identification is necessary for data protection in a multiuser work environment. Data identification is provided for each indivisible, retrievable data item.

Field 1:	Data Item ID
Field 2:	Parent Item ID
Field 3:	Data Item Owner
Field 4:	Access Attribute
Field 5:	Status

Table 2. Data Item Account Record.

Data item ID is the field that uniquely identifies the data item within its dataset. Notice that the uniqueness property does not provide fast access to the element. Mechanisms for fast data fetching are not considered in this article.

Parent item is the data item that contains the given data item in the dataset hierarchy, if the latter is present in the dataset.

Owner is a user that has modification rights for the data item. For example, a user who checks out a data item becomes its owner.

Access Attribute field allows the administrator to override the access rights to particular data items in the dataset for all users.

Functional Subsystem

The functional subsystem supports the following actions:

* Managing user accounts and access rights;
* Backing up and restoring data;
* Creation of new datasets;
* Creation of new versions of datasets;
* Freezing and unfreezing datasets;
* Checking in/out of data items;
* Data locking under concurrent access.

Administrative Subsystem

The administrative subsystem is a tool enabling the Administrator to set up and modify the design environment. This tool is used, e.g., during the following actions:

* Starting a new project
* Updating existing projects
* Monitoring and resetting locks
* Monitoring and updating access attributes for datasets
* Monitoring and updating user accounts
* Backing up and restoring projects

This subsystem is available only for users with administrative rights.

Basic Operation

Here 'Administrator' means any user with administrative rights for user and data access rights management. After installation, the system has a single Administrator user account record, with known login name and password. This account provides access to all administrative features, and therefore after logging in for the first time, the default password for this login must be changed to prevent unauthorized access. Afterwards this Administrator can add any number of users with administrative and ordinary rights. The Administrator can add new datasets into the project, assign access attribute to them and access rights for different users to these datasets.

A user logs into a system using his login name and password and selects for work any of projects that are made available for him by the Administrator. When new data items are created, the system verifies if a user has rights for object creation.

Figure 3. Login panel in ***Expert***

To modify a data item from a common database, the user must check it out. Only one user at a time may check a data item out. While this item is checked out, the remaining users which have read access rights for this item see the version of this item which was in the database before it was checked out.

After the modifications are finished, the item is checked in, and all users receive a notification about changes. They may either update their local projects that use this particular data item, or they may prefer to "freeze" its previous version.

Before the final release of the project, data versions must be synchronized.

At some stages of development, the Administrator may "freeze" the whole project. This means that no user may modify any data from the project. Data may be accessed only in read-only mode. This freezing may be performed, e.g., during global verification of the project.

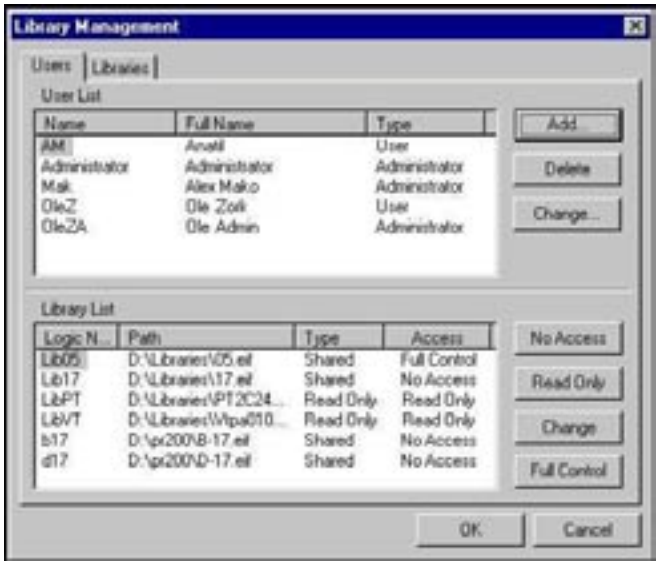

Figure 4. Admin Panel for user and library management in ***Expert***.

Moreover, designs (libraries) may be delivered as intellectual property in "frozen" form, to maintain the integrity of the library.

Implementation in Expert Layout Processor

Database Organization

The database in ***Expert,*** version 2.00, has been designed to meet the following goals:

* Preserve fast editing / viewing operations with geometric data available in previous versions of ***Expert***
* Provide controlled access to data items
* Support a library approach during layout design in order to ensure efficient and synchronized reuse of intellectual property.

These goals are achieved by a suitable combination of a general-purpose database with geometric data structures. The details of the design will be described in a subsequent issue of the *"Simulation Standard"*. Here we will note only that the main idea is to store indivisible data units of ***Expert's*** geometric engine as records in the general-purpose database. Some data units, such as cells or layers, are available for direct access (e.g., for checking cells out). Others are accessed from the disk database indirectly during the operation of the geometric engine. Currently ***Expert*** uses Microsoft's DAO for the general-purpose database engine.

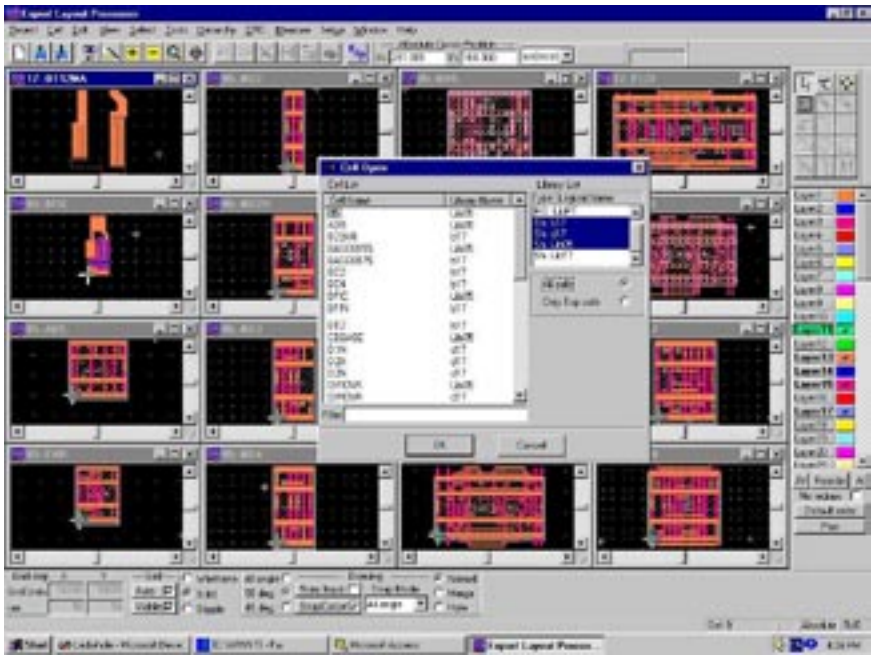

Figure 5. Access to library cells in ***Expert***

Library Approach in Expert

To provide reuse of intellectual property and concurrent access to the master design, the following library approach is adopted. Files with layouts in ***Expert*** may be of three types:

* A working project is a layout opened in ***Expert*** for exclusive modification. Any other user may access it only in read-only mode
* Cells from a read-only library may be used in other projects, but they cannot be modified
* Cells in a shared library may be used in other projects, and users with corresponding access rights may modify them via the check-out / check-in procedure

(to be continued in a future issue)

LISA

Powerful Script language for Expert

This article introduces the ***LISA*** (Language for Interfacing Silvaco Applications) command language and gives an overview of ***LISA*** features.

Expert interface scripts (xi-scripts) based on ***LISA,*** deliver powerful means for creation of any complicated shapes, parametrized cell constructions, recovery log maintenance, and execution of editing commands from the command line. The possibility to define custom commands delivers ultimate control over editing capabilities. ***LISA*** is a command language based on object-oriented concepts. Underlying ***LISA*** is a system of library routines called ***LISA-lib*** that establishes a common interactive system or environment through which multiple tools can define commands and exchange data. ***LISA*** is an integral part of this library.

LISA provides:

* A common environment for multiple tools
* Predefined commands, operators, and data types
* The ability to define new commands and types

LISA Applications

There are two general groups of users for ***LISA:*** application developers and application users.

Application developers use LISA as the command language for their tools. They do this by linking their application code with ***LISA-lib*** and setting up a command interface from which users of the application can enter ***LISA*** commands. They then create their own ***LISA*** commands and procedures that call their application routines.

Application users may not even know the application they are using is built from ***LISA-lib/LISA***. While users have available to them all of the same ***LISA*** features that are available to developers, users may be more interested in the application-specific commands and procedures. The fact that these application commands and procedures are built from ***LISA*** is irrelevant to the user. When application users work with a ***LISA***-based tool, they enter the application's commands using ***LISA*** syntax. Regardless of the application, the predefined ***LISA*** commands are always available. (Figure 1.)

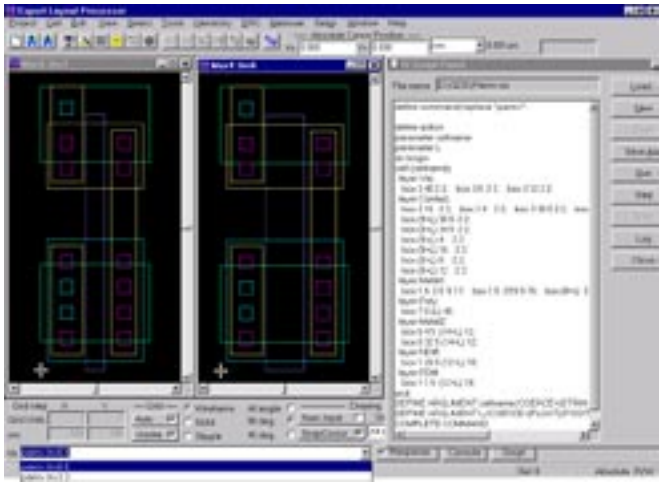

Figure 1. Automated creation of parametric cells using an XI script.

Ultimately users can group ***LISA*** statements into scripts, creating powerful programs to automate repetitive functions.

Multi-Platform Environment

LISA frees the user from having to write command interfaces in the native code of user's application. Even if the application is designed to run on different platforms and operating systems, the user needs to only write and maintain one common source for ***LISA*** commands. ***LISA-lib/LISA*** is currently supported on UNIX and Windows NT.

Libraries

LISA provides libraries of general purpose user-created ***LISA*** commands and procedures. This standard libraries can be used as a supplement to user created ***LISA*** commands and procedures. They can also be used as generic templates for creating new commands and procedures.

Syntax of Statements

Syntax rules apply to all applications that use ***LISA.*** There are, however, some differences in statement syntax between interactive and script input.

A statement is an element that performs some action in ***LISA.*** There are two classes of ***LISA*** statements: Commands and Expressions. This section deals with General ***LISA*** syntax issues, which are relevant to both commands and expressions. ***LISA*** interprets input differently depending on whether a line is a command or an expression. In particular ***LISA*** syntax dictates the use of parentheses to distinguish commands from expressions in certain situations. Commands can be used within expressions and expressions can be used within commands if you observe the proper syntax rules.

Using LISA Interactively vs. Scripts

There are two modes of entering ***LISA*** statements: interactive and script.

Interactive input involves a user entering data, usually from a keyboard, in response to prompts from an application. The method of input can vary from entering

statements at a command line or selecting menu options, depending on the application. A script is a group of statements in a file that perform some function, or set of related functions, that are executed sequentially when loaded into a ***LISA*** application. You create a script with any text editor. The file is read and executed in ***LISA*** by using the predefined LOAD command or by accessing a package. The script file can be either an ASCII ***LISA*** file or a binary file created by compiling ***LISA*** scripts with the ***LISA*** compiler. Compiled files load much faster than ***LISA*** scripts. Interactive and script input differ in the way that they handle line continuation, abbreviations, and statement termination.

In interactive mode, a command is executed or an expression is evaluated as soon as you enter a carriage return (by pressing the <Enter> key). When the statement is completed, command-line terminal displays its prompt (e.g., > or LISA>) to indicate that it is ready for the next statement.

Multiple statements can be entered on a single line by placing a semicolon (;) between each statement. The statements are executed separately, in the order entered, after the carriage return is depressed. Optionally, a semicolon may be entered before the carriage return, but it is not required for interactive input.

Each statement in a script must be terminated with a semicolon (;). Carriage returns have no significance in scripts. Other constructs can be used as terminating commands in particular situations. For example, placing a single statement in parentheses (forcing it to be an expression) terminates the statement. Using a semicolon in this case is not valid syntax. Also, the END keyword terminates the immediately preceding statement, so a semicolon preceding an END is not required (but is allowed).

For interactive input, a statement can be continued to another line using the ***LISA*** continuation character, the hyphen (-). To enter a statement that continues to the next line, enter a hyphen as the last nonblank character on the line, but before any comments.

The following example demonstrates statement continuation:

```
LISA> x = 1 -     ! Continuing a statement ...
_ lisa> + 2       !    ...  to another line
LISA>
```

Note that the continuation character and the minus sign are both represented by the hyphen (-). ***LISA*** interprets the hyphen as a continuation character only if it is the last nonblank character on the line.

In scripts, lines automatically continue until they are terminated by a semicolon. The hyphen cannot be used as a continuation character scripts.

The one exception to automatic line continuation is for quoted strings, which cannot extend across multiple lines. The same result can be achieved by using the concatenation operator (&) to join multiple smaller strings.

To enter a long string, break it into smaller strings, one to a line, then use the concatenation operator to join them:

The exclamation point (!) marks the beginning of a comment. A comment is text that provides information for a user(s) but is ignored by ***LISA.*** Typically, comments are used to document a script.

Comments are valid in both interactive and script modes; the exclamation point must follow a complete statement or a continuation character. However, there is typically no need to use comments while entering statements interactively.

In scripts, the comment must be the last element on a line. The statement on that line need not be complete, however.

Blank lines in the file are ignored and can be used as a formatting aide. Blank lines do not need to be proceeded by an exclamation point. The following example demonstrates the use of comments and blank lines in a script:

```
! This script defines two variable, x and y.

x = 1 + 2;  ! x is assigned the value  3
y = x +     ! This statement is not complete ...
5;          !   ... but now it is.  y is 8.
! End of script
```

General Rules for Names

LISA is not case sensitive. Any combination of upper and lower characters can be used for all syntatic elements. Case is maintained in all quoted strings, however. All names in ***LISA*** -- including names of commands, arguments, procedures, and variables -- must follow these rules:

* Names must start with an alphabetic character, a dollar sign ($), or an underscore (_). This initial character can be followed by any number of alphanumeric characters, dollar signs, and underscores
* Some special characters (! @ # % & * - + < > [ ] / \ . : ?) are reserved and cannot be used in names
* Predefined ***LISA*** constants and reserved keywords for naming can not be used
* Command and argument names can be abbreviated. All other elements in ***LISA,*** including procedures and variable names, operators, and other keywords, cannot be abbreviated

Parentheses ( ) serve several purposes in ***LISA:***

* They allow commands to be used within expressions
* They allow expressions to be specified as values for command arguments
* They make complex expressions more readable, or to override the precedence of the operations within an expression

BEGIN..END keywords can be used anywhere with parentheses. BEGIN..END keywords delimit a block of

statements that may be used anywhere where expression is valid. A block is a set of statements (commands or expressions) that are executed sequentially as a group. You can use a block anywhere where a single expression is expected. A block starts with the keyword BEGIN and ends with the keyword END. The value of a block is the last calculated value within that block, and it is that value that is supplied to the expression of which the block is a part. You must terminate all statement in a block, except the last, with a semicolon. The END keyword terminates the last line in the block, so a semicolon is optional .

The following shows an example block:

```
BEGIN        ! No semicolon for BEGIN and END

keywords

 command1;

 expression1;

 command2;

 expression2 ! Last expression, semicolon is optional.

END          ! No semicolon, unless the block is the final

             ! element of a statement
```

The value of this block is the value of expression2. BEGIN and END are not commands, so they are not terminated by semicolons. However, a BEGIN..END block is often the last component of a statement, so a semicolon follows the END keyword as a terminator for the complete statement. Placing statements in a block forces evaluation of the statements to be delayed. The statements following the BEGIN keyword are placed dynamically in a body and are not executed until the END keyword is encountered.

Expressions

Expressions are statements that ***LISA*** evaluates to a single value. Some ***LISA*** expressions are similar to the familiar algebraic form found in most programming languages and many command languages. Others are unique to ***LISA .***

Expressions are used for the following:

* To calculate values
* To assign values to variables
* To produce side effects during evaluation (when commands and procedures are invoked)

An expression written in the ***LISA*** language is composed of various elements:

* Constant
* Variables (and constant variables)
* Operators (including indexing)
* Procedure calls.

If a command name does not start a statement, ***LISA*** interprets the line in the expression mode. Expressions and commands can be mixed using parentheses to differentiate between the types of statements.

In addition, you can specify a block of statements, delimited by BEGIN..END keywords, anywhere an expression is valid.

LISA recognizes four types of constants: Integers, Floats, Strings, Enumerated types

LISA automatically determines the type of constants that you enter in expressions. Numeric characters are interpreted as either integer constants or float constants depending on their format. Numeric characters without a decimal point or exponent indicator are taken as integers, otherwise they are floats.

String constants are any group of characters that are enclosed in quotation marks. For example:

```
"This is string."
```

The following shows some type of enumerated types (the #( ) is ***LISA's*** label for enumerated types):

```
# (white, blue, black)
```

Enumerated Type constants are equivalent if all of their elements match; case is not important.

Variables are created with the DEFINE VARIABLE command or by specifying a variable name as the target of the assignment operator (=):

```
DEFINE VARIABLE a_y  30;
          X = 4.1;
```

When you assign a value to a variable with the assignment operator (=), ***LISA*** expects an expression to the right of the equals sign; therefore parentheses are not needed. Since DEFINE VARIABLE is a command, any expression entered as an argument value must be enclosed in parentheses to indicate that it is an expression .

It is possible to assign a value of any type to a variable. After a variable is initially defined, you can also reassign values of different types to the variable. If you want to restrict reassignment of a variable, create a constant variable. Constant variables are assigned a value when they are created and cannot be redefined. Any variables that is defined outside of delayed code has global scope. It can be referenced by any expression in a ***LISA*** program.

LISA operators are symbols that can be used in expressions to execute basic operations:

Arithmetic, Comparison, Logical, String, Selection (BY, DOWNTO, FOR, TO), Indexing ([]), Assignment (=).

The basic operations determine the actions to be performed when an expression of a certain type is evaluated by a given operator.

Calling Procedures

Procedures are groups of ***LISA*** code that accept input values (parameters) and perform some function, usually calculating and returning a value. A procedure does not have to take parameters, however, nor does it have to return a value.

LISA provides three ways to call procedures:

* Directly
* Using CALL PROCEDURE command
* As the action procedure to a command

Only direct procedure calls are expressions. ***LISA*** allows a users definition of own procedures. When procedure is invoked, values for all of the parameters defined in the procedure definition must be supplied. Parameters are variables that pass information into the procedure. Procedures may or may not return a value. If there is a return value, you may store the value in a variable or ignore it. A direct procedure call can appear anywhere a value is needed in an expression. A procedure that is called as part of an expression must return a value. The value returned is used to further evaluate the expression.

The procedure parameters must follow the procedure name, enclosed in parentheses and separated by commas. If there are no parameters, empty parentheses are still required. The parameters are expressions. The expression value is calculated and passed to the procedure.

The following are examples of procedure calls:

```
add_int (numb_1, numb_2);        ! two pa-
rameters, return value ignored

current_time = get_time ( );      ! no pa-
rameters, return value stored

IF (value_is_odd(x)) THEN         ! accepts
one parameter, returns a          Boolean
```

If you are defining a recursive procedure, or need to forward reference a procedure (all of a procedure that is not yet defined), you must use CALL PROCEDURE instead of directly calling the procedure. CALL PROCEDURE has the following format:

```
CALL PROCEDURE procedure_name [ [{]param-
eter_value, ... [}} } ;
```

The parameter values must be separated with commas. You can optionally surround the parameter values with curly braces({}). CALL PROCEDURE internally coerces the comma-separated values to a sequence however, so the braces are not necessary. If there are no parameters, omit the parameter argument.

CALL PROCEDURE must be used to implement recursive procedures or to forward reference a procedure. These situations may occur when you are defining your own procedures.

Commands

A ***LISA*** command is an envelope around a procedure that enables preprocessing of input data, the displaying of prompts, and other features not available with procedures alone.

LISA commands usually take input arguments and perform some action using those arguments. Commands can also return a value to the caller. A command name is a verb phrase made up of one or more keywords, separated by one or more spaces. The following are examples of possible command names:

```
READ
SHOW DEFAULTS
```

No prefix of a compound command name, which is a name that is more than one word, can be a complete name by itself. For example, if SHOW DEFAULTS is a complete command name, then SHOW by itself cannot also be a command name, though DEFAULTS can be a command name.

Most ***LISA*** commands have arguments. When a command has arguments, they must follow the complete command name. ***LISA*** supports two types of command arguments: positional and named.

Arguments used in ***LISA*** commands set the values of the parameters used in the command's action procedure. A command's action procedure performs the action of the command.

LISA commands can perform type conversion or set default values before passing argument values from the command line into the procedure for processing.

If a command is terminated without specifying a value for an argument, ***LISA*** will do one of the following:

* Use a default value (if defined as part of the command)
* Prompt for missing values (if the interface allows prompting)
* Signal an exception (if the interface does not allow prompting) Positional Arguments

Positional arguments must be specified in the order defined by the command definition, and they must be separated from each other by one or more spaces, unless the argument value is enclosed in parentheses, in which case whitespace is optional. Specify values on the command line for all positional arguments unless all positional arguments are omitted to the end of the line. For omitted trailing arguments, the default values are used if the command has default values, or you are prompted to enter values (interactive mode only).

Positional-argument values can be expressions or literals Expressions must be enclosed in parentheses.

The following examples show some uses of positional arguments.

```
READ (filename)
COPY  STRUCTURE  (structure_name)
(destination)
ADDINT 17 3
```

Named arguments begin with a slash (/) followed by the argument name. Named arguments can appear anywhere on the command line, and in any order, as long as they come after the complete command name. Named arguments can be defined to take or not to take a value, following an equals sign (=), on the command line. Named-argument values can be expressions or literals , depending on how the argument is defined.

The following examples show some named arguments:

```
TRANSLATE GATE  /ALL
SET DEFAULT /FORMAT=EIF  /NAME=FULL
```

A command definition can include multiple arguments that set a particular parameter in the ***LISA*** command procedure. A group of related arguments that set the same parameter is called a cluster. Only one argument from a cluster can appear in any specific command invocation.

For each argument cluster, you must either specify one of the arguments on the command line, or the cluster must have a default argument as part of the command definition. Attempting to specify multiple arguments from the same cluster causes an exception.

Both positional and named arguments can be in the same cluster. However, there can be at most, one positional argument in a cluster.

For example, the definition of a command DELETE NODES might include two named arguments, /VERIFY and /NOVERIFY, that form a cluster. Assume that these arguments set alternate values for a Boolean parameter in the command's procedure. Only one of these named arguments can appear on any particular DELETE NODES command. The command definition would also define one of these as the default, so if either was specified on the command line, one of them would be passed to the command's procedure.

Argument values can be expressions, which must be surrounded by parentheses, or they can be literals if the argument definitions have defined coercion of the values. Argument values are either entered explicitly, specified by default as part of the argument definition or hard-coded in the argument definition.

If the argument definition specifies a default you can take the default values, if you omit all other positional arguments that would follow it on the command line.

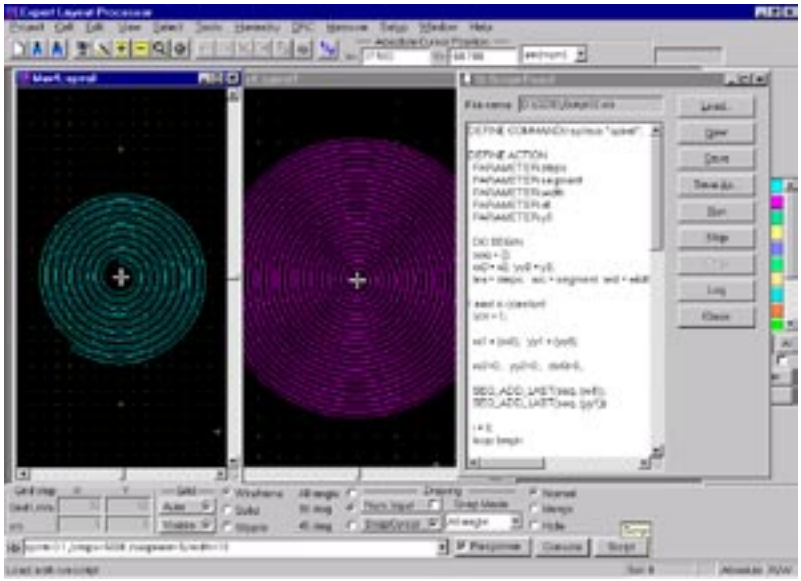

Figure 2: An example of complex shapes created in ***Expert*** using an xi-script demonstrate the power of ***LISA.***

For example, a command, SET POINT, could take two positional arguments, X and Y. Assume that the argument definitions specify a default of zero or both X and Y.

For named arguments, values can be explicit, optional, or implied.

You enter an explicit value for a named argument after an equals sign (=), which follows the argument name. The value may be an expression or literal. For example: /FORMAT=text.

If the argument definition specifies a default and an explicit value was not specified, the default is used. For example, a named argument /FORMAT, could either take an explicit value (following an equals sign) or /FORMAT could be specified alone, in which case, a default value is used.

The argument definition can specify a value to use for a named argument that does not take an explicit value. For example, /TEXT could be a switch that implies a predefined value in the command definition.

Conclusion

The XI script based on the ***LISA*** language, essentially extends the efficiency of IC layout design allowing to customize and extend design tools and environment. Utilizing some routine programming work (e.g. handling of exceptions) inside predefined procedures, it allows the user to concentrate on basic design problems.

XI/LISA brings to the command line a functional interface to the underlying subsystems, allows easily customizable related applications (e.g. verification suite) and offers a great help in the development of new CAD products. Abstracting from data base structure makes possible the integration of important TCAD/CAD applications into the ***Expert*** layout processor environment utilizing the power of its geometry engine.

Calendar of Events

December

1	
2	
3	
4	
5	
6	
7	IEDM - San Francisco
8	IEDM - San Francisco
9	IEDM - San Francisco
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	Merry Christmas
26	
27	
28	
29	
30	
31	

January

1	Happy New Year
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	

Bulletin Board

Good News for East Coast Customers

The city of Chelmsford, MA has issued all necessary permits to construct a large office building for Silvaco's Eastern Region Applications Center. The approved building architecture can be viewed at www.silvaco.com.

New Release of Expert and Savage

New release of ***Expert*** 2.0 and ***Savage*** 2.0 were benchmarked by a number of customers and found to have crossed the threshold of capability of leading UNIX CAD providers of simulator tools. These well documented packages are now expected to gain worldwide rapid acceptance.

Silvaco Has Hometown Advantage for IEDM '98

See what the future holds for you in TCAD and Parasitic Interconnect Simulation, as the San Francisco Hilton and Towers will host both IEDM '98 and Silvaco International on December 6th through the 8th. At the Powell Room on the sixth floor, a large number of Silvaco application engineers will demonstrate the latest in TCAD simulation, parasitic interconnect extraction and circuit simulation.

For more information on any of our workshops, please check our web site at ***http://www.silvaco.com***

Hints, Tips and Solutions

Mikalai Karneyenka, Applications and Support Engineer

Q: I took the example file mux4.gds from *Expert's* installation and tried the Edit-in-place operation. I entered into an instance of mux1, modified a box in it and exited from EIP. There are options to change the edited cell and to save instance under another cell name. I selected the latter option, but all four instances of mux1 in mux4 have been changed, rather than the instance I modified in place. (See Figure 1.)

A: This behavior is not a bug. As you may have noticed, cell mux4 contains a 1x4 array of cell mux1. When you edit in place an element of an array, you are editing all elements of the array. If you change a single instance from this array, the whole array will be destroyed. You must split the array into separate instances or rows or columns first. See Figure 2.

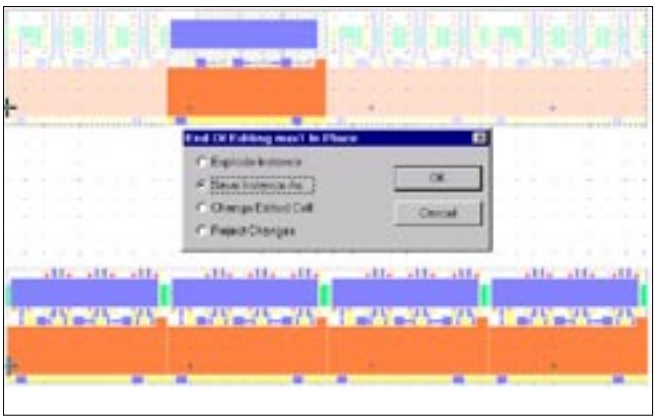

Figure 1.Mux4.gds during and after editing in place.

Q: What is the difference between "flat" and "hierarchical" modes in the DRC script panel of *Expert*?

A: Flat vs. hierarchical are modes for error reports only. ***Expert*** runs DRC checks in flat mode, but it groups errors into cells where they actually occur. For example, if you have 15000 instances of a via cell with contact of the wrong size, flat mode will report 15000 errors, while hierarchical mode will report only one. Clearly, this is a great convenience when you need to analyze DRC error reports. The actual checks are performed in exactly the same way for both modes. A truly hierarchical DRC checker, ***Dragon*** is expected soon from Silvaco.

Figure 2: Instance/array modification dialog

Q: When drawing boxes *Expert* provides a prompt in the message line about box parameters. We have noticed that sometimes that information goes away, for example, when a box is moved. Also, if a box gets converted to a polygon, the information is no longer available.

A: Information in the message line is "volatile": It is always updated for the current operation. If you want to see info about an object again, select it or pick it for modification.

Q: How can I view a set of coords with either the window view or pan view?...Basically I'm hunting drc errors down by coordinates and haven't found the easiest way to do this.

A: There are "Cursor Position" fields in ***Expert*** just above the layout window. There is also a field to select the measurement units. You can enter the required coordinates in the Cursor "Position" fields and then press the <Enter> key. The cursor will go to the required point. If this point was off screen, ***Expert*** will pan to this point. If the cursor is in the layout window, then you may press <Ctrl-P> to start the input of cursor position.

Call for Questions

If you have hints, tips, solutions or questions to contribute, please contact our Applications and Support Department

Phone: (408) 567-1000 Fax: (408) 496-6080

e-mail: *support@silvaco.com*

Hints, Tips and Solutions Archive

Check our our Web Page to see more details of this example plus an archive of previous Hints, Tips, and Solutions

www.silvaco.com

Join the Winning Team!

Silvaco Wants You!

- Process and Device Application Engineers
- SPICE Application Engineers
- CAD Applications Engineers
- Software Developers

fax your resume to:
408-496-6080, or
e-mail to:
jobs@*silvaco.com*

Opportunities worldwide for apps engineers: Santa Clara, Phoenix, Austin, Boston, Tokyo, Guildford, Munich, Grenoble, Seoul, Hsinchu. Opportunities for developers at our California headquarters.

SILVACO
INTERNATIONAL

USA Headquarters:

Silvaco International
4701 Patrick Henry Drive, Bldg. 2
Santa Clara, CA 95054 USA

Phone: 408-567-1000
Fax: 408-496-6080

sales@silvaco.com
www.silvaco.com

Contacts:

Silvaco Japan
jpsales@silvaco.com

Silvaco Korea
krsales@silvaco.com

Silvaco Taiwan
twsales@silvaco.com

Silvaco Singapore
sgsales@silvaco.com

Silvaco UK
uksales@silvaco.com

Silvaco France
frsales@silvaco.com

Silvaco Germany
desales@silvaco.com

e*ECAD PARTNER

*Products Licensed through Silvaco or e*ECAD*