Simulation Standard

TCAD Driven CAD

A Journal for CAD/CAE Engineers

Scholar: An Enhanced Multi-Platform Schematic Capture

Introduction

One of the main features of modern complex CAD systems consists in the ability to support the schematic capture function that provides a user with convenient design entry. In our opinion, a perfect schematic capture subsystem (SC) possesses the following features that differ it from other graphic intensive CAD tools such as layout editors:

- The main attention in schematic capture is given to effective user service and convenience support while the problem of large amount graphical information processing is not so acute. It is solved by dividing a cell schematic into pages.
- Owing to schematic capture software supports a design entry and provides the user with a graphical interface; it should be considered as a design representation hub. Therefore, access from SC user interface to other design aspects (layout, netlist, simulation results, etc.), cross-probing these aspects, schematic driven IC design, and other schematic related functionality should be supported.

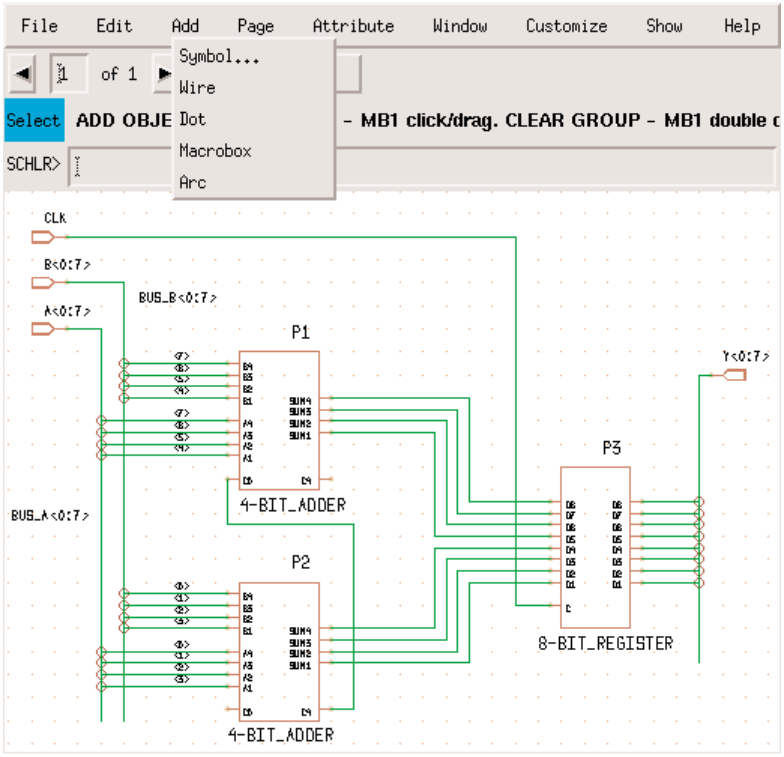

Figure 1. An accumulator schematic.

Silvaco's new enhanced schematic capture, ***Scholar,*** covers both of these features. On the one hand, it gives the user the great variety of possibilities to create and check large complicated hierarchical schematics. On the other hand, ***Scholar*** can automatically generate a netlist for Silvaco's layout synthesis tools and ***SmartSpice*** input deck. It supports cross-probing between schematic and other design aspects such as layout and ***SmartSpice*** simulation results.

Another important feature of ***Scholar*** is that it is a multi-platform product and can be run on both Unix and Windows NT platforms. So, it provides both Unix-based and PC-based for Silvaco's products with design entry.

Continued on page 2....

INSIDE

SILVACO INTERNATIONAL

Main Concepts

The main concepts on which ***Scholar*** is based accumulate the best-in-class schematic capture experience of leading CAD companies. The overview of these concepts is given bellow.

Basic notions. The basic notions which ***Scholar*** operates with are: symbol, wire, macrobox, bristle, bubble group, and page. They allow the user to create a hierarchical multi-page schematicof of unlimited size.

Creating a hierarchical multi-page schematic. The user may organize a schematic as a set of hierarchically related drawings. Each drawing may be divided into pages that are connected to each other.

Using symbols and macroboxes. Two methods are supported in ***Scholar*** to work with schematic hierarchy: symbols and macroboxes. Symbols that are icons representing pieces of schematic are created and used to organize a pure hierarchical design. Macroboxes are convenient to repeat a piece of logic in a flat design.

Creating symbol versions. This possibility allows the user to create different symbol representations such as De Morgan equivalents.

Using bubble groups. If the user creates a symbol ***Scholar*** can support the situation when the certain bristles on that symbol are related to each other.

Using attributes. Attribute mechanism that is widely used in ***Scholar*** makes the schematic more expressive. It allows to pass needed descriptive information about schematic to other tools such as simulators and layout synthesis tools.

Checking a schematic. When schematic is ready errors can be automatically found and displayed. That can be done both before or in process of netlist generation.

Netlist generation. This possibility allows to create netlist that may be used as input for simulation and layout synthesis.

Spice simulation. ***Scholar*** can generate a ***SmartSpice*** input deck using an extensive library of elements. The simulation process may be run from ***Scholar's*** user interface.

Scholar **-** ***Expert*** **- Spice cross-probing.** A client-server cross-probing mechanism allows Silvaco's ***Scholar*** schematic capture, ***Expert*** layout editor, and ***SmartSpice*** simulator to tie together. The user can, for example, identify a schematic wire to highlight its layout representation or corresponding ***SmartSpice*** diagram.

Customizing ***Scholar.*** This possibility allows the user to adjust the schematic visualization characteristics, customize some operations, define accelerator keys, function keys, and so forth. An user can make the ***Scholar's*** Schematic representation similar to representations used in popular schematic captures such as Cadence's Composer.

In what follows the ***Scholar's*** functionality that embodies these concepts is discussed.

Creating a Hierarchical Schematic

Scholar effectively allows to draw hierarchical schematic based either on bottom-up or top-down strategies to be drawn. The symbol is the fundamental notion on which hierarchical schematic is based.

Creation of a hierarchical design involves four basic steps:

- Create a schematic
- Test and verify the design
- Create the symbol
- Create another schematic and instantiate the symbol

This order of steps assumes a bottom-up strategy. It is also possible to create a symbol and instantiate it before creating the corresponding schematic.

Figure 1 demonstrates an example of a hierarchical schematic created in the ***Scholar*** environment. It is the schematic of accumulator that consists of two 4-bit adders, one 8-bit register, three input bristles and one

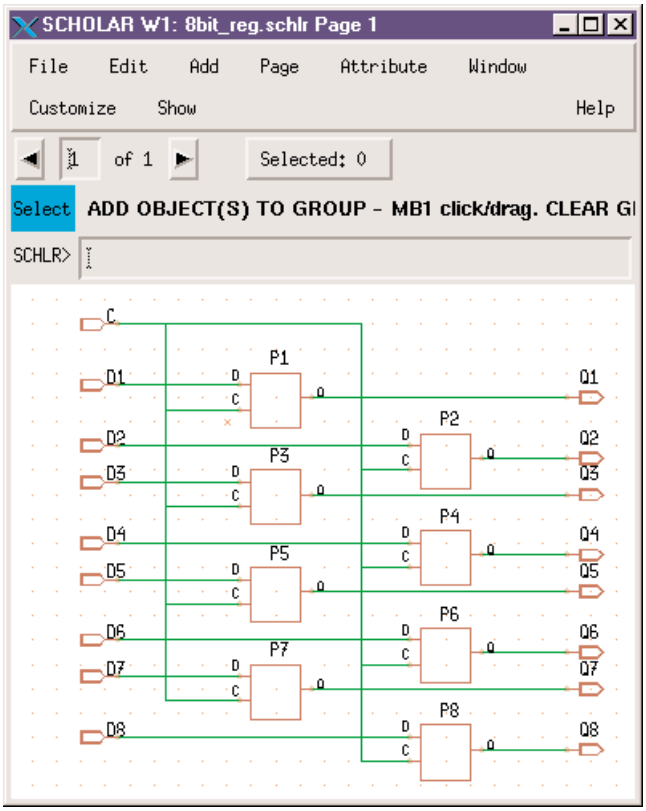

Figure 2. The 8-bit register schematic

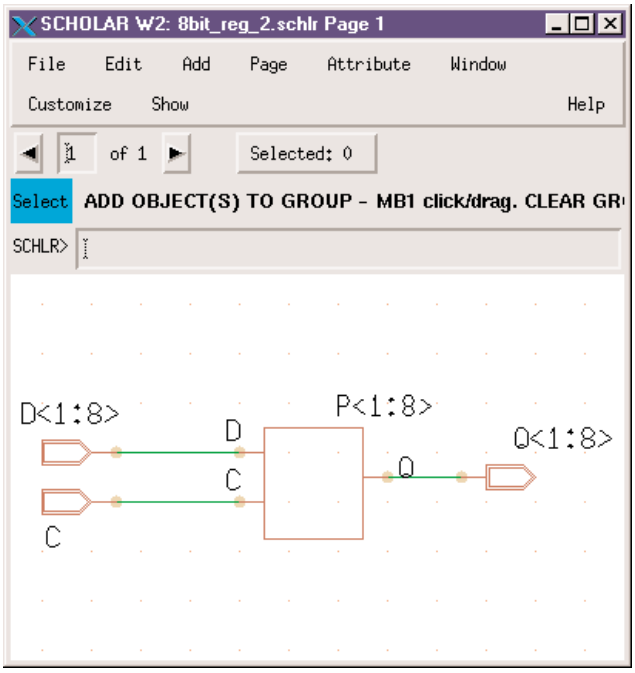

Figure 3. Arrayed representation of the 8-bit register

output bristle, and three 8-bit buses. All of these components are represented as symbols. And, at the same time, user defined symbols, such as 4-bit adder, have a corresponding schematic.

One reason for this, ***Scholar*** operates with two types of files: *drawing file* and *symbol file*, which correspond with each other.

Scholar has the library of symbols that provides the user with standard set of transistor levels, logical levels and special symbols. At the same time, with a complicated design the user can define his own symbols to the needed hierarchy and, therefore, simplify the drawing process. It makes sense to note that one of the strong sides of the ***Scholar*** is that its library is co-ordinated with ***SmartSpice*** and ***Expert*** layout editor libraries. That is why seamless interface between these programs was a success.

Scholar supports a simple and clear way to navigate the hierarchy. The user can click any user defined symbol in order to open it in the separate window. The 8-BIT_REGISTER schematic that is opened in such a way is shown in Figure 2.

This schematic is a regular structure which consists of eight instances of the one-bit register. They are represented as user defined symbols. ***Scholar*** may use this regularity to do a schematic more clear. It creates instance arrays with input buses that vary in width according to the size of the array. The arrayed representation of the 8-bit register is shown in Figure 3.

Attributes

Scholar powerfully embodies a very popular idea to associate any needed information with selected objects in a drawing using attributes. Names of objects, signal names and other additional information are represented in ***Scholar*** in the form of attributes.

Every symbol in the library has one or more attributes attached to it. When instantiating a library or any other symbol on a schematic, its attributes become the default attributes of the instance.

Some attributes are interpreted and modified before being placed in the netlist. Other attributes are placed in the netlist without interpretation or modification, and are used by downstream tools.

Scholar allows needed attributes to be attached to objects or to modify current attributes. During the modification process the user can:

- Find and display all the attributes
- Find specific attributes
- Find attributes of selected objects
- Remove attributes from the dialog box
- Select attributes from the drawing
- Modify several attributes
- Change the attribute visibility
- Change the text size of attribute
- Sort attributes by ancestor, location, or as selected

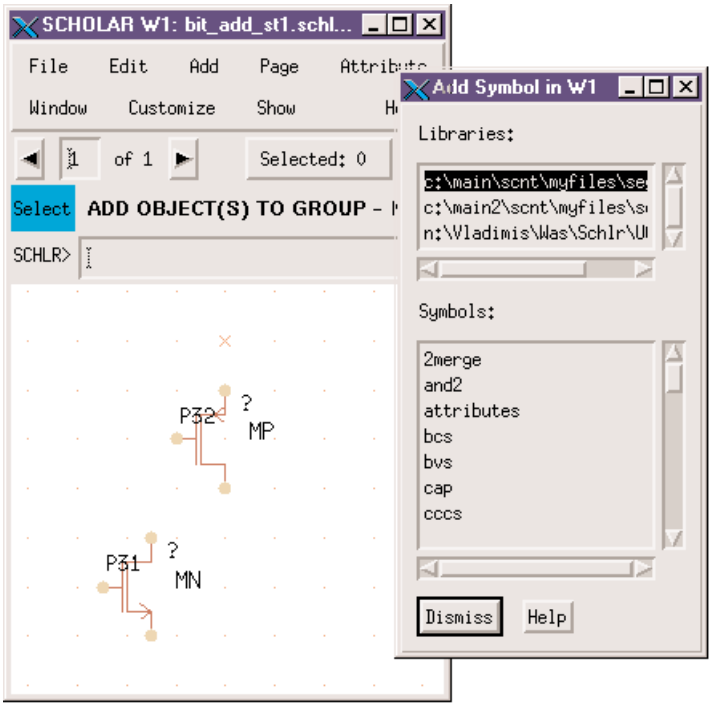

Figure 4a. Library elements placement.

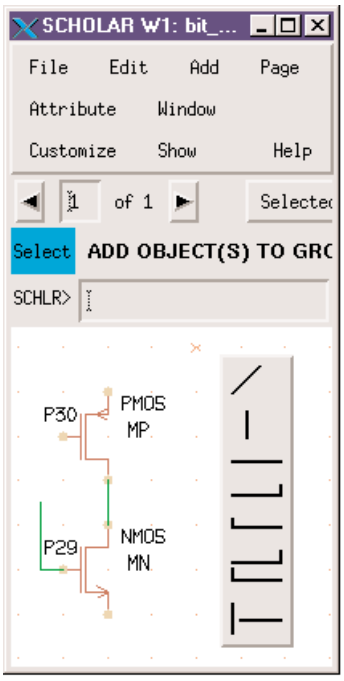

Figure 4b. Nodes placement.

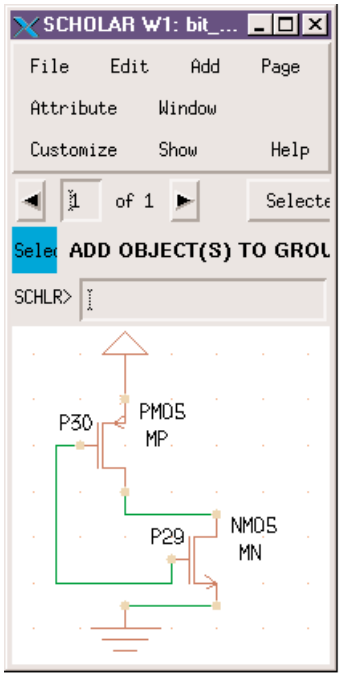

Figure 4c. Moving the transistor.

Using attributes is a fundamental, powerful part of ***Scholar's*** mechanisms. On a level with defining names of objects, it can be effectively applied for very special purposes useful in VLSI designing, such as adjustment of gates on transistor sizes.

Sometimes it may be necessary to use the same piece of logic again in a drawing but transistor sizes may need to be changed, depending on where the logic will be reused. ***Scholar*** gives the possibility to avoid the creating a different symbol for each differently sized piece of logic. You may create one symbol to which to pass different sizes as parameters.

Drawing a Schematic

The wide spectre of interactive and graphical features (operating with windows and menus, zooming In and Out, using the grid, moving, panning, fitting, etc.) allows ***Scholar*** to be considered as one of the most perfect modern schematic captures. It possesses with advanced functionality intended for schematic drawing. The schematic drawing process is illustrated in Figures 4-5.

Suppose, the user decided to create a 1-bit adder schematic on the transistor level. In this case, he should use library elements such as PMOS and NMOS transistors. Figure 4a demonstrates the library elements selection and placement. ***Scholar*** allows the complete set of graphical operations for element placement to be used, (i.e. Move, Rotate, Mirror, Orient, Copy, Delete, and others).

Figure 4b illustrates the process of wire creation using the set of wire styles that can be accessed through the pop-up menu. ***Scholar*** considers a node to be set of any connected wires and bristles to be at the same electrical potential. Any name may be selected for any node on the schematic except for power, ground or other global node's. Naming a node is really a matter of attaching a attribute called SIG_NAME to the node.

One of the most powerful features of the ***Scholar*** lies in its possibility to move a connected object with stretching wires, as shown in Figure 4c. The NMOS transistor is being selected and moved here.

Other schematic fragments can be created the same way this one was. You may use Select, Copy, and Paste commands to place them.

In ***Scholar***, bristles on schematic provide external connections for nodes on a drawing and are represented by flag symbols. Any node (except power, ground, or other global nodes) that continue from or onto another schematic must be attached to a bristle. An unique name must be assigned to each bristle on the schematic. After fragments and bristle placements are accomplished, the user can create nodes as was discussed above.

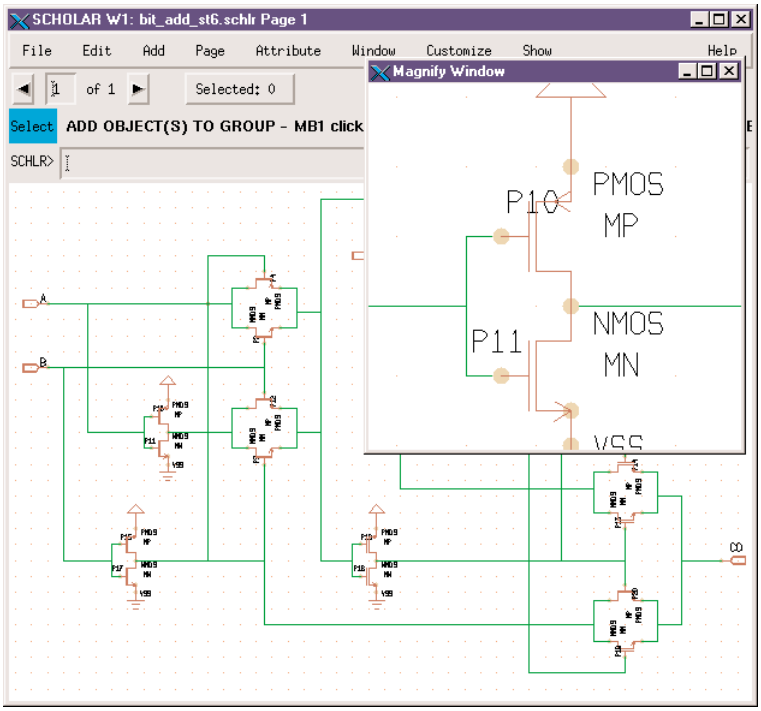

Figure 5. 1-bit adder schematic.

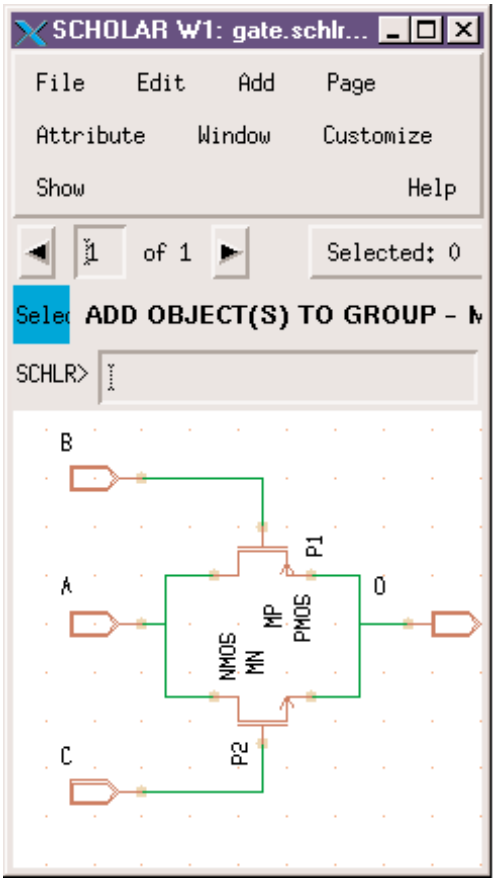

Figure 6. The gate schematic

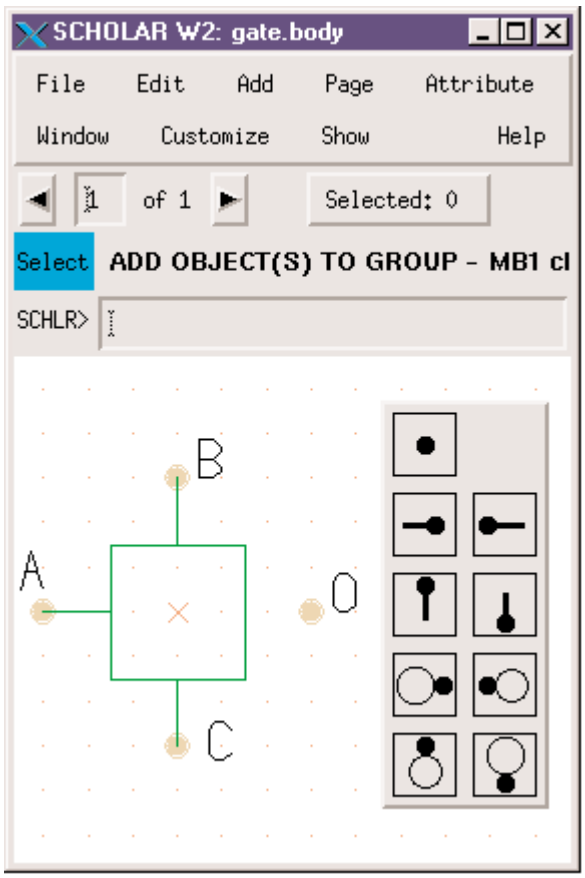

Figure 7. The gate symbol creation

The accomplished variant of the transistor-level 1-bit adder schematic is shown in Figure 5. This figure also illustrates another one of ***Scholar's*** feature which consists in the use of the Magnify Window. It is a special pop-up window that provides the user with a quick way to zoom in on a part of the drawing. ***Scholar*** performs any drawing or editing function (or bring up any pop-up menus) while working in the magnify pop-up window.

Creating and Using Symbols

While the process of a transistor level schematic creation is quite convenient ***Scholar*** allows it to be done much more effectively. For this purpose, it proposes mechanisms based on the use of symbols and macroboxes.

Evidently the 1-bit adder schematic has some repeated pieces of circuitry. In the example discussed above they were placed by Copy and Move operations. Another way is to build the same schematic based on the use of both library and user-defined symbols.

While the library symbol may be used for the inverter, the user-defined symbol should be created for another repeated transistor group (gate) in this schematic. In order to create a symbol, a user has to create the corresponding schematic, first. The gate schematic is shown in Figure 6.

After that, the symbol to which this schematic is attached is created. This process is illustrated in Figure 7. First of all, the shape of the symbol is drawn. Then, bristles are added. ***Scholar*** places different sorts of bristles: nonbubbleable and bubbleable, that, in their turn, may be unbubbled or bubbled. The user can select the needed bristle type using the pop-up menu.

On top of that, ***Scholar*** supports a unique mechanism of bubble groups creation. An user can specify that certain bristles on the symbol are related to one another. In other words, if one bristle is (un)bubbled, the other bristle must also be (un)bubbled. Creating a bubble group instructs the ***Scholar*** to (un)bubble an appropriate bristle when a related bristle is (un)bubbled.

Another effective possibility of ***Scholar*** is the creation of different versions of the symbol. All versions of a symbol refer to the same schematic.

Using Macroboxes

A unique and powerful feature of ***Scholar*** is its possibility to use macroboxes. This mechanism provides another way to do the drawing process much more efficiently. In some cases this may be more preferable from the user's point of view than the use of symbols.

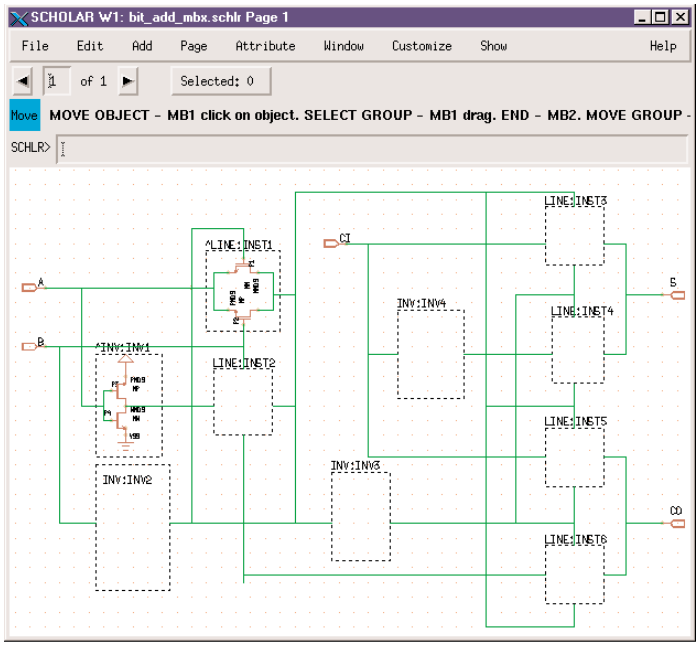

Figure 8. Macrobox based 1-bit adder schematic.

Just like a symbol, a macrobox can be considered a piece of logic that can be instantiated many times. But there is a distinct difference. While a symbol is defined in a separate symbol file and can be instantiated on any page in any schematic, a macrobox is defined in the drawing where it is used. So, during a cell schematic creation, the user can pick out or create a repeatable part of it in the drawing, define this part as a macrobox definition, and then instantiate a macrobox as many times as is needed.

At first sight, the macrobox use may look like schematic piece Copying and Pasting only. However, a macrobox mechanism is in fact, much more powerful. While placing a macrobox, the user does not see anything inside it. He only draws boxes (they may be in different sizes), gives them appropriate instance names, and attaches nodes to appropriate box sides.

Just like symbols, macroboxes are useful whether the design is following a bottom-up strategy or a top-down strategy.

Macroboxes are interpreted as one of the following:

- Definition
- Instance
- Definition/instance, i.e the definition that at the same time plays the role of an instance.

The use of macroboxes is illustrated in Figure 8 where inverter and gate schematics are represented as macroboxes.

In this example, ^INV;INV1 and ^LINE;INST1 are definitions/instances of corresponding fragments because both of them play the role of a macrobox definition and macrobox instance simultaneously. Other building blocks of this schematic such as INV;INV2 or LINE;INST6 are only macrobox instances.

The following rule must be adhered: the number and width of bristles on each side of the macrobox instance must match the number and width of bristles on each side of the macrobox definition.

Scholar proposes flexible possibilities to work with macroboxes. For example, it allows the array of macroboxes, as shown in Figure 9, to be built. In this example, the macrobox definition (a cell of memory) is created. It is after that instantiated as the array of cells.

Another way to create the array of macroboxes is that in the name of a macrobox the user can specify the number of times the macrobox is to be arrayed. In this case, when netlisting is performed, the macrobox will be arrayed a specified number of times. Another way is by using the special Array command that creates automatically-numbered macrobox instances.

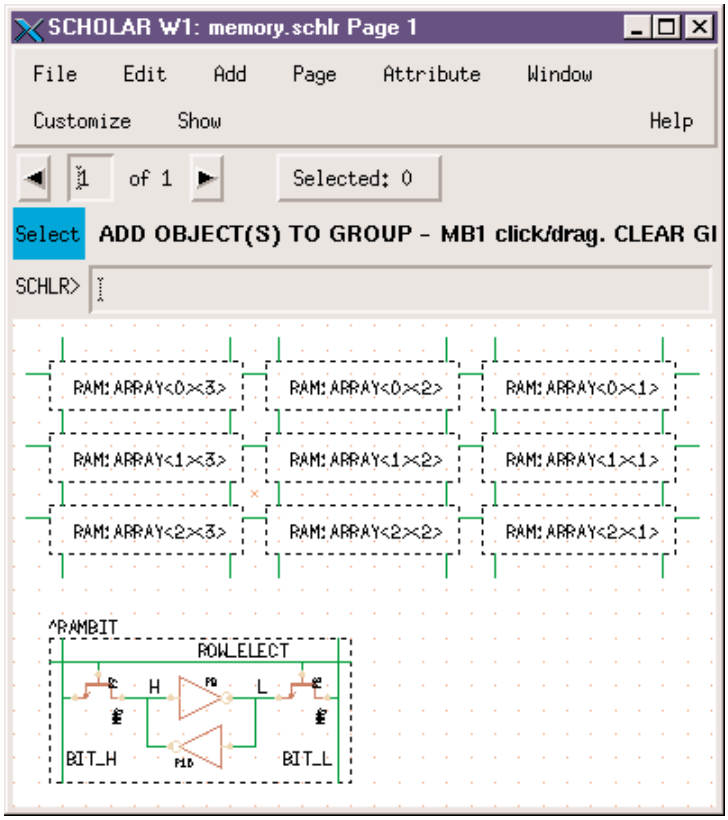

Figure 9. A memory schematic fragment.

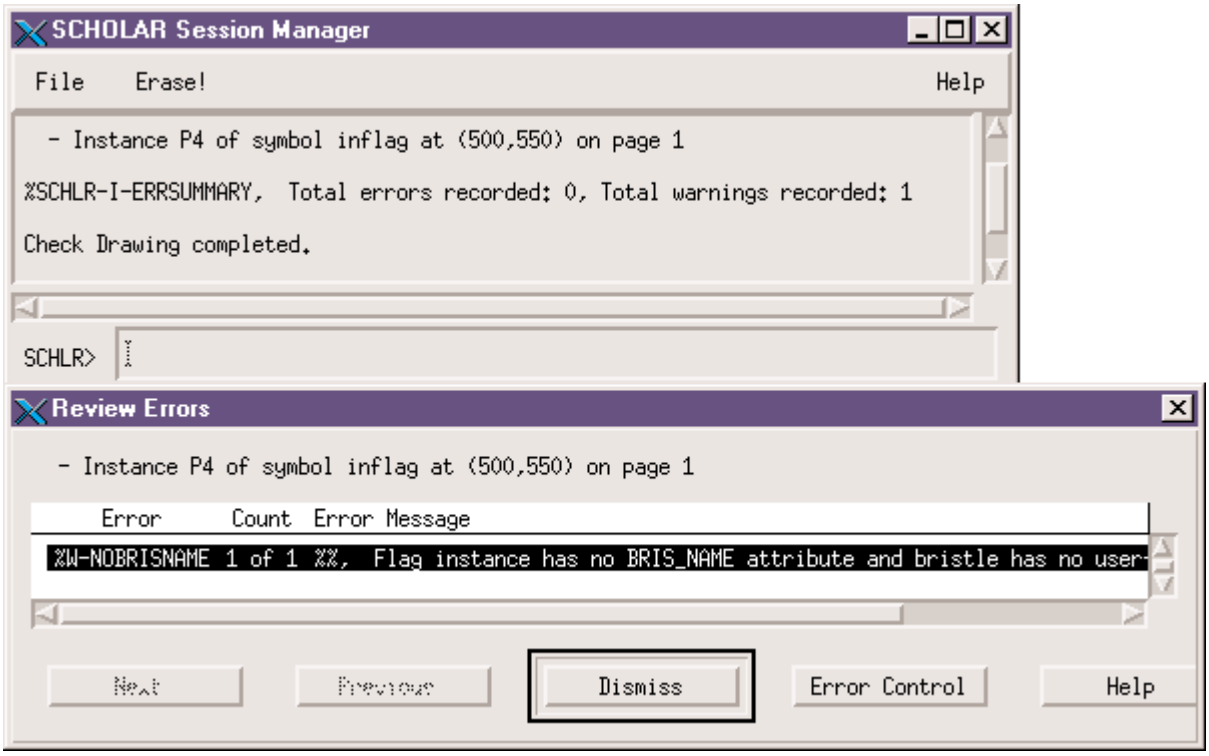

Figure 10. Error checking.

Checking Drawing

Once the schematic is drawn and appropriate attributes and parameters are attached, a netlist can be generated. A netlist is hierarchical. It contains only the information for this schematic, on this current level of the hierarchy. ***Scholar*** supplies the user with utilities that allows it to manipulate a netlist. In particular, it may be flattened or the ***SmartSpice*** input deck may be generated.

If there were errors in the schematic during netlisting, a list of errors is displayed in the Session Manager window as shown in Figure 10. The drawing can also be checked for errors before netlisting by selecting the special Check Drawing function. Another Check Drawing related function, Review Errors, allows it to open the Review Errors dialog box. It displays how many errors occurred and list of error messages. If an error message is selected (highlighted), the appropriate error in the drawing is highlighted.

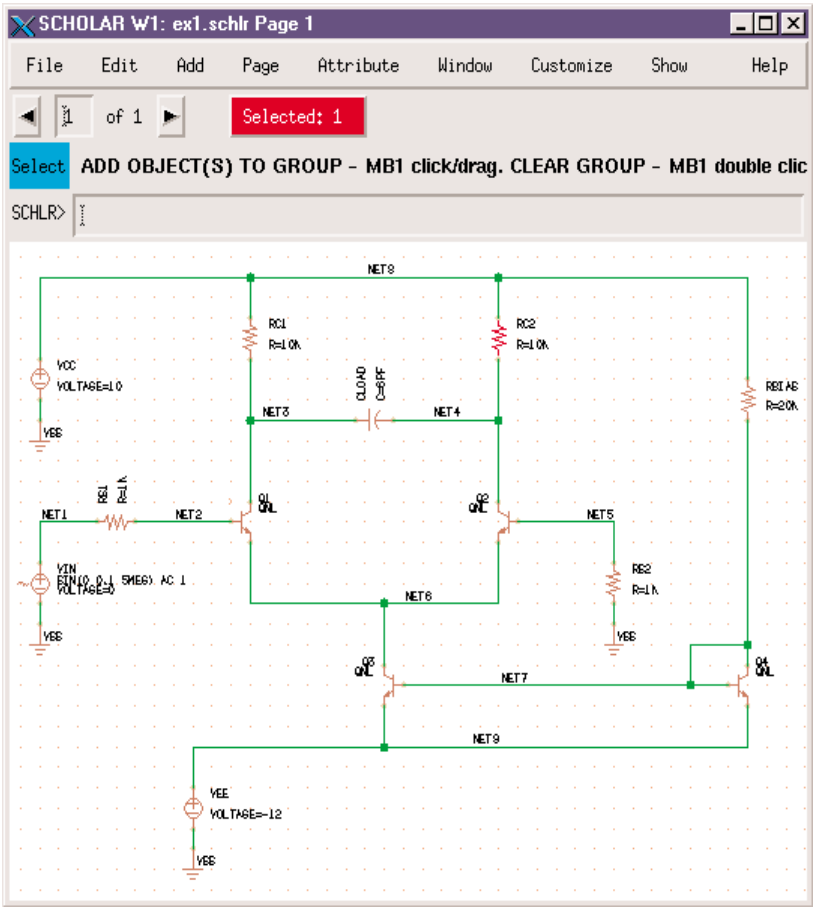

Figure 11. A schematic prepared for ***SmartSpice*** simulator.

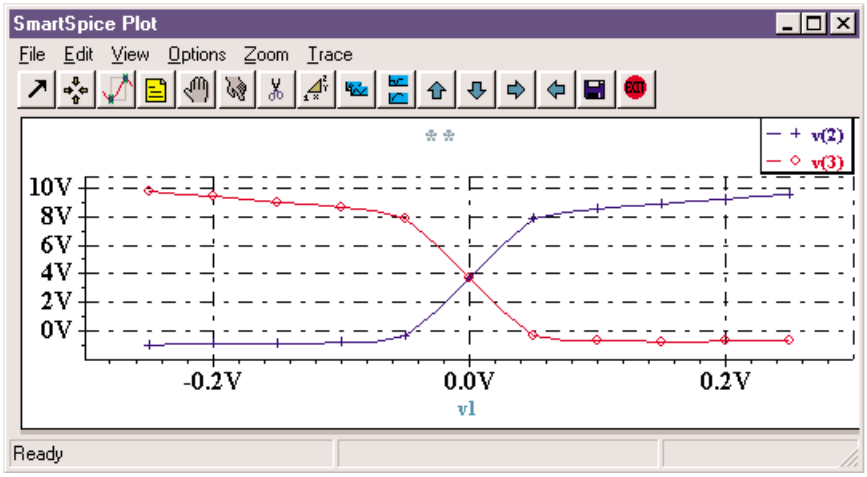

Figure 12. Results of DC analysis.

Connection with *SmartSpice*

In ***Scholar***, special attention is paid to its efficient connection with other CAD tools. That is why Scholar may really be considered as a central tool of a CAD system.

Scholar has, for example, two basic possibilities for passing output information to downstream tools such as the ***SmartSpice*** simulator:

- To generate a netlist in the special text (readable) format named GWL format
- To generate Spice input deck and simultaneously to run ***SmartSpice***

In order to create an input deck, ***Scholar*** uses a special elements library that co-ordinated with the ***SmartSpice*** library, i.e. elements contained in the ***Scholar's*** library having all the attributes needed for ***SmartSpice*** simulation.

Besides this, the following files are also should be prepared:

- *Model file.* It contains statements that define parameters of models used in a current project.
- *Control file.* It contains control statements that will be included into the input deck as a control section.

Figure 11 shows the schematic that has been created to be simulated by ***SmartSpice***. The user can now generate the input deck and run ***SmartSpice***. Some results of simulation (DC analysis, NET3 and NET4) are shown in Figure 12.

Besides flat schematics like just was discussed ***Scholar*** supports the transformation of hierarchical schematics into ***SmartSpice*** input deck.

Customizing *Scholar*

As any other graphics intensive tools, ***Scholar*** is required to be customized within great limitations. So, ***Scholar*** corresponds to modern requirements concerning customization. The user has the following possibilities to customize ***Scholar:***

- Using the Customize menu
- Changing things like window geometry, screen and menu color, or accelerators keys

Using the Customize menu changes ***Scholar*** features for that particular drawing session only. This menu changes the drawing grid, how text appears, and how objects are moved around the window. For example, if it is necessary to customize the drawing grid, the user clicks the Grid item from the Customize menu. In effect, the Customize Grid dialog box shown in Figure 13 appears.

Another possibility to customize the Scholar consists of changing the ***Scholar's*** customization file. When ***Scholar*** is invoked, it finds and executes this file that sets up colors, screen geometry, menu items, and so on. A default initialization file is supplied.

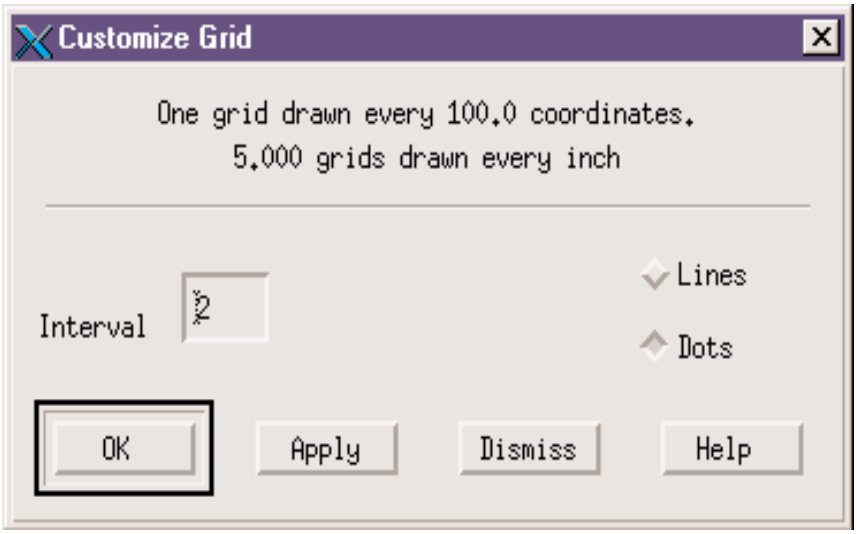

Figure 13. Customize Grid dialog box.